पाइथन

हिंदी में

Hindilearn द्वारा प्रकाशित

पाइथन

पाइथन ट्यूटोरियल के बारे में

पाइथन यह एक छोटे और बड़े एप्लीकेशन्स को डेवलप करने के लिए उपयोग में लायी जाती है | इसका उपयोग डाटा साइंस के साथ वेब डेवलपमेंट के लिए किया जाता है | इस ट्यूटोरियल में हम कॉम्प्लेक्स डाटा स्ट्रक्चर जैसे कि Lists, Sets, Dictionary और Tuples के साथ Modules, Exception Handling, File Handling, Regular Expression और आदि पाठ को पढ़ाएंगे |

ट्यूटोरियल किसे पढना चाहिए

डाटा साइंस के लिए सी और सी प्लस प्लस के साथ पाइथन का भी ज्ञान सम्पादित करना अनिवार्य हो चूका है | जिसे डाटा साइंस का ज्ञान अपूर्ण हो उसे पाइथन के इस ट्यूटोरियल का उपयोग करना चाहिए |

पाइथन सिखने के लिए अपेक्षित

जो पाइथन सीखना चाहता है तो उसके पास Computer (Desktop, CPU, Keyboard) या laptop और पाइथन के कौनसे भी IDE (Integrated Development Environment) की जरुरत होती है | इस PDF File को Run करने के लिए किसी भी Browser या PDF Reader का उपयोग करे |

अपने अधिकार

जो भी सामग्री आप इस PDF के माध्यम से उपयोग में ला रहे है वो Hindilearn.in की संपत्ति है जैसे कि छायाचित्र और पाठ | अगर आप इसकी व्यावसायिक रूप में नक़ल करने का प्रयास करेंगे तो आप पर राईट लॉज़ के अनुसार कार्रवाई भी हो सकती है |

संपर्क

अगर आपको कोई गलत जानकारी या अशुद्धलेखन इस ट्यूटोरियल के माध्यम से प्राप्त होता है तो आप हमें help@hindilearn.in इस पते पर संपर्क कर सकते है |

पाइथन
हिन्दी में

विषय सूची

PY INTRODUCTION 14

Introduction and History 14
Python Features 14
Python Applications 15
Hello World 16
Python Keywords 18

PY VARIABLES 34

What is Variable 34
Types of Variable 38

PY DATA TYPES 41

Number 41
Strings 44
List 45
Tuple 45
Dictionary 46
Set 46

PY INDENTING 48

Indenting Code 48

विषय सूची

PY OPERATORS 49

Arithmetic Operators 49
Relational Operators 52
Logical Operators 54
Assignment Operators 56
Bitwise Operators 61
Identity Operators 64
Membership Operators 65

PY INPUT AND OUTPUT 66

Introduction 66
Type Conversion 67
print Function 68

PY LOOPS 70

While Loop 70
infinite Loop 71
For In Loop 72

PY CONTROL STATEMENTS 75

If Statement 75
If Else Statement 76
If Elif Else Statement 77
pass Statement 78
break Statement 78
Continue Statement 79

पाइथन
हिन्दी में

विषय सूची

PY FUNCTIONS **80**

What is Function *80*

Types of Function *80*

Anonymous or Lambda Function *86*

PY LISTS **88**

Creating List *88*

Positive and Negative Indexing *88*

Accessing List *89*

List Functions *93*

len	93	min	94
max	94	list	95

List Methods *1*

append	96	insert	99
clear	97	pop	100
copy	97	remove	100
count	98	reverse	101
extend	98	sort	101
index	99		

विषय सूची

PY TUPLES 103

Introduction 103

Tuple Functions in Python 107

all	108	min	111
any	109	sorted	112
enumerate	110	sum	114
len	110	tuple	116
max	111		

PY DICTIONARY 117

Introduction 117

Dictionary Functions in Python 120

all	120	len	123
any	121	str	123
dict	122		

Dictionary Methods in Python 1

clear	124	keys	128
copy	125	pop	129
fromkeys	125	popitem	130
get	126	setdefault	131
has_key	127	update	132
items	128	values	133

विषय सूची

PY SETS 134

Introduction 134

Set Functions in Python 137

all	138	*min*	141
any	139	*set*	142
enumerate	140	*sorted*	143
len	140	*sum*	145
max	141		

Set Methods in Python 146

add	147	*issubset*	153
clear	147	*issuperset*	154
copy	148	*pop*	155
difference	149	*remove*	156
difference_update	150	*symmetric_difference*	156
discard	151	*symmetric_difference_update*	157
intersection	151		
intersection_update	152	*union*	158
isdisjoint	153	*update*	159

PY STRINGS 160

Introduction 160

Python Escape Characters 162

Old Way String Formatting(C-Style) 164

Format Specifier with Placeholder 169

Flags 170

New Way String Formatting(Python-Style) 172

Format Specifiers 174

विषय सूची

String Functions in Python 178

len	178	min	179
max	179		

String Methods in Python 180

capitalize	182	ljust	198
center	183	rjust	199
casefold	183	lower	200
count	184	upper	200
endswith	185	swapcase	201
expandtabs	186	lstrip	201
find	186	rstrip	202
format	187	strip	203
index	189	partition	203
isalnum	190	rpartition	205
isalpha	191	replace	206
isdecimal	192	rfind	207
isdigit	192	rindex	208
isidentifier	193	split	208
islower	194	startswith	209
isnumeric	194	title	210
ispritable	195	zfill	210
isspace	196	len	211
istitle	196	max	211
isupper	197	min	212
join	198		

विषय सूची

PY DATE AND TIME **213**

Introduction 213

Time Module 214

Time Module Attributes 216

altzone	216	timezone	217
daylight	216	tzname	217

Time Module Functions 218

asctime	219	sleep	225
ctime	220	strftime	225
gmtime	221	strptime	228
localtime	223	time	229
mktime	224		

PY MATHEMATICS **230**

Math Constants in Python 230

e	230	pi	231
inf	231	tau	232
nan	231		

Math Functions in Python 233

acos	235	cosh	241
acosh	235	degrees	241
asin	236	exp	242
asinh	236	expm1	243
atan	237	fabs	243
atan2	238	factorial	244
atanh	238	floor	244
ceil	239	fmod	245
copysign	240	frexp	246
cos	240	fsum	247

विषय सूची

gcd	247	modf	254
hypot	248	pow	255
isclose	249	radians	256
isfinite	250	sin	256
isinf	250	sinh	257
ldexp	251	sqrt	257
log	252	tan	258
log10	252	tanh	258
log1p	253	trunc	259
log2	254		

PY RANDOM MODULE AND FUNCTIONS **260**

Functions *260*

choice	260	sample	263
randint	261	seed	263
random	262	shuffle	264
randrange	262	uniform	265

PY MODULES **266**

PY EXCEPTIONS AND HANDLING **272**

Exceptions *272*

Exception Handling *274*

विषय सूची

FILE HANDLING INTRODUCTION **280**

Opening a File 280
Closing a File 282
Reading a File 282
Writing a File 283
Renaming a File 284
Removing a File 285
File Methods 285

REGULAR EXPRESSION **286**

Introduction 286
Character Classes 289

[..]	289	*[^A-Z]*	298
[^..]	292	*[a-z]*	299
[0-9]	293	*[^a-z]*	301
[^0-9]	295	*[mf]ail*	302
[A-Z]	296	*Grac[ey]*	303

विषय सूची

Quantifiers 305

n+ 305
n* 307
n? 308
n{X} 309
n{X,Y} 310
n{X,} 312
n$ 313
^n 314
p|q 316
python+ 317
python* 317
python? 318

Metacharacters 320

dot(.) 320
b 321
B 323
d 324
D 325
f 327
n 328
r 329
s 330
S 332
t 333
v 334
w 335
W 336

Modifiers or Flags 337

re.S or re.DOTALL 337
re.I or re.IGNORECASE 338
re.L or re.LOCALE 339
re.M or re.MULTILINE 339
re.U or re.UNICODE 340
re.X or re.VERBOSE 341

Functions 342

compile 342
findall 344
finditer 345
match 347
search 348
split 350
sub 352
subn 354

Methods 356

Match Object Methods 356

group 357
groups 358
groupdict 359
start 360
end 361

विषय सूची

CLASSES AND OBJECTS 362

INHERITANCE 367

Introduction 367
Multilevel Inheritance 369
Multiple Inheritance 371

CLASS ATTRIBUTES AND FUNCTIONS 373

Class Attributes 373
Functions for Class Attributes 375

delattr 375
hasattr 376
getattr 376
setattr 377

CONSTRUCTOR AND DESTRUCTOR 378

Constructor 378
Destructor 379

METHOD OVERRIDING 380

OPERATOR OVERLOADING 383

Introduction

Introduction and History

Introduction for Python

- Python ये सभी Programming Languages में से सबसे आसान Language है |
- Python ये एक High-Level Object-Oriented Programming Language है |
- Python में जैसे English Language होती है लगभग वैसे ही code python language में होता है |
- Python ये Language open source Language है | ये किसी OS Platform पर free में उपलब्ध है |
- अगर Python Language को सीखना हो तो किसी भी Basic Programming Language की जरुरत नहीं है |

Python History

- Python Language का अविष्कार 'Guido Van Rossum' इस dutch programmer ने किया |
- Python की शुरुआत 1980 में की और करीब एक दशक साल बाद में python को 1991 में launch किया गया |

Why Named Python ?

- C++ Programming Language का इस्तेमाल Computer Software बनाने के लिए किया जाता है |
- Computer Software के साथ-साथ Drivers, Computer Hardware, servers के लिए भी इस्तेमाल किया जाता है |

Python Features

Python Programming Features

- Easy to Learn
- Easy to Understand
- Easy to Read
- Portable
- Large Number of Libraries
- Open-Source and Free
- Object-Oriented
- Embeddable
- Extensible

- **Easy to Learn** : Python ये language C, C++ और Java Language जैसी है | इस Language में ज्यादा keywords न होने की वजह से काफी आसानी से सिखी जा सकती है |

- **Easy to Understand** : Plain English के जैसे syntax होने की वजह से काफी अच्छे से समझ आती है |Easy to Read :Python का Program ज्यादा पेचीदा न होने से पढ़ने में आसानी प्रदान करता है |

- **Simple** : Python Language पढ़ने और समझने में बहुत ही आसान है | ये Language पढ़ने में ज्यदातर plain English जैसी होती है |

- **Portable** : Python का program एक platform(OS) से दुसरे platform पर port करके execute किया जाता है |Large Number of Libraries :Python में बहुत सारी libraries होती है जिससे किसी specific code को अलग से नहीं लिखना पड़ता है |

- **Open-Source and Free** : Python का software मुफ्त में वितरित किया जाता है | Python का code पढ़ा जा सकता है या उसमे कुछ फेरबदल भी किये जा सकते है | इसको व्यवसायिक रूप में भी इस्तेमाल किया जाता है |

- **Object-Oriented** : C++ और Java के साथ ये Language भी एक Object-Oriented Language है | Object-Oriented होने के कारण Program को समझने में आसानी प्रदान करता है |

- **Embeddable** : Python को C, C++ या आदि Languages के साथ embed किया जा सकता है |

- **Extensible** : अगर किसी दूसरे language का code अच्छे से काम नहीं कर रहा है या धीमी गति से काम कर रहा है तो उसके बदले में Python को इस्तेमाल किया जा सकता है |

Python Applications

1. Scientific and Computational Applications
2. Web Applications
3. Gaming Applications
4. ERP Applications
5. Graphical Applications

1.Python Programming Features

Scientific और Computational Operation के लिए Python में कुछ libraries आती है जैसे कि, scientific computing के लिए SciPy और Numeric computing के लिए NumPy library का इस्तेमाल किया जाता है |

2.Web Applications

Python की मदद से Web Applications बनाने के लिए Content Management System(CMS) और Frameworks का इस्तेमाल किया जाता है जैसे कि, Django, Bottle, Flask और CherryPy और भी कई Frameworks का इस्तेमाल किया जाता है |

3.Gaming Applications

Game Development के लिए Python में कुछ modules, libraries का इस्तेमाल किया जाता है जैसे कि, Soya3D और Panda3D ये framework high-level 3D games को develop करने के लिए इस्तेमाल किया जाता है |

4.ERP Applications

Enterprise और Business Applications के लिए ERP5, ERPNext और OpenERP का इस्तेमाल किया जाता है | Youtube, Yahoo और reddit इनमें Python का इस्तेमाल किया गया है |

5.Graphical Applications

Graphics Design के लिए Python में Python-Orge, PyQt, PyGtk और आदि frameworks का इस्तेमाल किया जाता है |

Hello World

Install Python on Local Computer

- **Step 1 (Download Python IDLE)** : Python को मुफ्त में download किया जाता है | Python को download करने के लिए http://www.python.org/downloadsइस link पर जाकर latest(3.6.1) version को download करें, चाहे तो अपने जरुरत के हिसाब से कोई भी versions download किये जा सकते है |
- **Step 2 (Installing Python)** : Download करने के बाद download हुई file को double click करके Install करे |
- **Step 3 (Open Python IDLE)** : जहा पर उसे install किया गया है उस path पर जाकर उसे open करे |
- **Step 4 (Python Program in Shell)** : Python के साधारण से Program को Shell में लिखिए |
- **Step 5 (Create .py file)** : Shell के बावजूद python program को अपने computer पर save करने के लिए Shell में file menu पर जाकर new(Ctrl+N) पर click करके अपना program लिखे |Save करने के लिए Ctrl+S या file में save button पर click करे | उसके बाद उसे नाम देकर .py के साथ वो default set किया जायेगा | अगर extension दिया भी जाता तब भी कोई दिक्कत नहीं आती है |For eg. new.py
- **Step 6 (Run .py file)** : Python file को save करने के बाद उसे Run करने के लिए Run में Run Module या F5 का इस्तेमाल करे |

'Hello World' Program in Python

In Shell

```
>>> print("Hello World")
Hello World #Output
```

Save new.py

```
print("Hello World")
```

Output :

```
Hello World
```

Difference Between Version 2.7.x and Version 3.x.x

In 2.7.x

Python 2.7.x में print ये function नहीं होता है |

```
print "Hello World"
a = 1 / 2
print a
```

Output :

```
Hello World
0
```

In 3.x.x

Python 3.c.x में print() ये function होता है |

```
print "Hello World"
a = 1 / 2
print a
```

Output :

```
Hello World
0.5
```

Python Keywords

Print All Python Keywords

```
import keyword
print(keyword.kwlist)
```

Output :

```
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else',
'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or',
'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

and	as	assert	break
class	continue	def	del
elif	else	except	nonlocal
False	finally	for	from
global	if	import	in
is	lambda	None	not
or	pass	print	raise
return	True	try	while
with	yield		

'and'(Logical AND)

'and' keyword का इस्तेमाल जब दिए गए दो या दो से ज्यादा operands true होते है तब true return किया जाता है |

0 = True
1 = False

Truth Table of 'and'

x	**y**	**x and y**
0	0	0
0	1	1
1	0	1
1	1	1

Source Code :

```
print(5!=6 and 5==6)  #False
print(5!=6 and 5==5)  #True
print(5!=6 and 5==6 and 4<6) #False
```

Output :

```
False
True
False
```

'as' Keyword

'as' keyword का इस्तेमाल module को कोई नया या उपनाम(alias) नाम देकर import किया जाता है |

Source Code :

```
import keyword as pyKey
print(pyKey.kwlist)
```

Output :

```
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else',
'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or',
'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

'assert' Keyword

Python Program में debugging के लिए 'assert' keyword का इस्तेमाल किया जाता है |

'assert' keyword के साथ जब condition true होता है तब कुछ नहीं होता और जब condition false होता है तब 'AssertionError' occur होता है |

Source Code :

```
assert 4==4 #No Exception
assert 4==5 #AssertionError Exception Occured
```

Output :

```
    assert 4==5 #AssertionError Exception Occured
AssertionError
```

'break' Keyword

Python में break Statement Loop को किसी expression पर बंद कर देता है |

Source Code :

```
nums = [1, 2, 3, 4, 5]
for n in nums :
  print(n)
  if(n == 4):
    break
```

Output :

```
1
2
3
4
```

'class' Keyword

User द्वारा classes को define करने के लिए 'class' keyword का इस्तेमाल किया जाता है |
classes में कुछ attributes और कुछ methods होते है |
Classes 'OOP' में सबसे महत्वपूर्ण हिस्सा है |

Source Code :

```
class myClass:
    def func():
        print("Hello")

myClass.func()
```

Output :

```
Hello
```

'continue' Keyword

'continue' keyword का इस्तेमाल iterate हो रहे कुछ statements को skip करता है |
'continue' का उपयोग iterate न हो रहे Loop को iterate करने के लिए किया जाता है |

Source Code :

```
a = 0
while(a < 10):
    if(a == 5):
        print("skipped value of a is ", a)
        a = a + 1
        continue
    print("value of a is ", a)
    a = a + 1
```

Output :

```
value of a is  0
value of a is  1
value of a is  2
value of a is  3
value of a is  4
skipped value of a is  5
value of a is  6
value of a is  7
value of a is  8
value of a is  9
```

'def' Keyword

'def' keyword का इस्तेमाल user-defined function बनाने के लिए किया जाता है |

Source Code :

```
def func(a):
    print("Value of a is", a)

func(5)
```

Output :

```
Value of a is 5
```

'del' Keyword

Python 'del' keyword का मतलब delete होता है |
'del' keyword का इस्तेमाल list, tuple, dictionary या किसी और collection से element को delete करने के लिए इस्तेमाल किया जाता है |

Source Code :

```
list = [1, 4, 3, 5, 7]
del list[2]
print(list)
```

Output :

```
[1, 4, 5, 7]
```

'del' keyword के बजाय remove() function का भी इस्तेमाल किया जाता है |

Source Code :

```
list = [1, 4, 3, 5, 7]
list.remove(3)
print(list)
```

Output :

```
[1, 4, 5, 7]
```

'elif' Keyword

'elif' keyword का इस्तेमाल if और else keyword के साथ किया जाता है | एक से ज्यादा conditions को check करने के लिए 'elif' का इस्तेमाल किया जाता है |

Source Code :

```
a = 11

if(a < 10):
    print("a is less than 10")
elif(a > 10):
    print("a is greater than 10")
else:
    print("a is equal to 10")
```

Output :

```
a is greater than 10
```

'else' Keyword

'else' keyword का इस्तेमाल if, elif या सिर्फ if के साथ किया जाता है |अगर if या if और elif की condition false होती है तो else का statement execute होता है |

Source Code :

```
a = 11

if(a%2==0):
    print("Number is even.")
else:
    print("Number is odd.")
```

Output :

```
Number is odd.
```

'except' Keyword

'except' keyword का इस्तेमाल exception/error handling के लिए किया जाता है | try clause के साथ except को इस्तेमाल किया जाता है |
Example में try block में जो भी exception occur होगा उसका तुरंत Block execute हो जाएगा |

Example1

Source Code :

```
try:
    a
except ZeroDivisionError:
    print("Divided by zero Error")
except NameError:
    print("a is not found")
```

Output :

```
a is not found
```

Example2

Source Code :

```
try:
    4/0
except ZeroDivisionError:
    print("Divided by zero Error")
except NameError:
    print("a is not found")
```

Output :

```
Divided by zero Error
```

'nonlocal' Keyword

'nonlocal' keyword का इस्तेमाल nested funtion के लिए किया जाता है | 'nonlocal' variables ये global भी नहीं होते और global भी नहीं होते है | अगर inner function में उनको लिया जाता है तो outer functions में उनकी values change नहीं होती है |

Without nonlocal

Source Code :

```
var = 2
def mainFunc():
    var = 5
    def subFunc():
        var = 10
        print("subFunc : ", var)

    subFunc()
    print("mainFunc : ", var)

mainFunc()
print("global : ", var)
```

Output :

```
subFunc :  10
mainFunc :  5
global :  2
```

With nonlocal

Source Code :

```
var = 2
def mainFunc():
    var = 5
    def subFunc():
        nonlocal var
        var = 10
        print("subFunc : ", var)

    subFunc()
    print("mainFunc : ", var)

mainFunc()
print("global : ", var)
```

Output :

```
subFunc :  10
mainFunc :  10
global :  2
```

'False' Keyword

'False' keyword boolean false का वर्णन करता है |अगर दी गयी condition या statement गलत होता है तो False; return किया जाता है | False '0' के बराबर होता है |

Source Code :

```
print(False == 0)
print(4 == 5)
```

Output :

```
True
False
```

'finally' Keyword

'finally' keyword का इस्तेमाल 'try-except' या 'try' block के साथ किया जाता है |
exception/error हो या ना हो finally block execute होता है |

Source Code :

```
try:
    a
except NameError:
    print("NameError Exception occured")
finally:
    print("Always executed")
```

Output :

```
NameError Exception occured
Always executed
```

'for' Keyword

'for' keyword का इस्तेमाल looping के लिए किया जाता है | Collection के elements को iterate करने के लिए 'for' Loop का इस्तेमाल किया जाता है |

Source Code :

```
list = ["Rakesh", "Ramesh", "Suresh"]
for n in list:
    print(n)
```

Output :

```
Rakesh
Ramesh
Suresh
```

'from' Keyword

'from' keyword का इस्तेमाल 'import' keyword के साथ किया जाता है | कुछ विशिष्ट function या attribute को import करने के लिए 'from' का इस्तेमाल किया जाता है |
अगर सिर्फ 'Module' को import किया जाता है और उसमे से func1() को लेना पड़ता है तो 'Module.func1()' लिखना पड़ता है |

Module.py

Source Code :

```
def func1():
    print("I am in func1")
def func2():
    print("I am in func2")
```

Output :

```
```

sample.py

Source Code :

```
from Module import func1
func1()
```

Output :

```
I am in func1
```

'global' Keyword

'global' keyword का इस्तेमाल function के variable को module में global बनाने के लिए किया जाता है |

Difference between Local and Global Variable in function

test1.py

Source Code :

```
def func():
    string = "Local Variable"
    print(string)
func()
print(string)
```

Output :

```
Local Variable
    print(string)
NameError: name 'string' is not defined
```

test2.py

Source Code :

```
def func():
    global string
    string = "Local Variable"
    print(string)
func()
print(string)
```

Output :

```
Local Variable
Local Variable
```

'if' Keyword

'if' keyword का इस्तेमाल if statement, if_else statement और if_elif_else statement में किया जाता है |

Source Code :

```
var = 0
if(var == 0):
    print("var is equal to 0")
```

Output :

```
var is equal to 0
```

'import' Keyword

'import' keyword का इस्तेमाल module को current program पर import करने के लिए किया जाता है | Example पर math module को import किया गया है |

Source Code :

```
import math
print(math.sqrt(16))
```

Output :

```
4.0
```

'in' Keyword

'in' keyword का इस्तेमाल collections(list, tuple, string, dictionary) में element है या नहीं ये check करके boolean value return करता है | for Loop में भी 'in' keyword का होना अनिवार्य होता है |

Source Code :

```
list = [6, 8, 4, 7, 8]
print(7 in list) #True
print(10 in list) #False
```

Output :

```
True
False
```

'is' Keyword

'is' keyword का इस्तेमाल object equality के लिए किया जाता है | ये keyword '==' operator के थोडा-बहुत समान होता है | अगर दो object equal होते है तो 'True' return होता है और equal नहीं होते है तो 'False' return होता है |

Source Code :

```
print("one" is "one") #True
print("one" is 1) #False
print("" is "") #True
print("" == "") #True
print([] is []) #False
print([] == []) #True
```

Output :

```
True
False
True
True
False
True
```

'lambda' Keyword

'lambda' keyword से anonymous(without name) function को create किया जाता है | name वाले function में 'def' keyword का इस्तेमाल किया जाता है और anonymous function में 'def' के बजाय 'lambda' keyword का इस्तेमाल किया जाता है | lambda function के लिए return statement नहीं होता है | function के arguments list को parenthesis(()) में close नहीं किया जाता है |anonymous function होने के कारण इनको किसी variable पर assign किया जाता है और उस variable के जरिये call किया जाता है |

निचे दिए गए दोनों Example का मतलब एक ही है |

Normal Function

Source Code :

```
def func(a, b):
    return a * b
print(func(5, 5))
```

Output :

```
25
```

Anonymous Function

Source Code :

```
var = lambda a, b: a * b
print(var(5,5))
```

Output :

```
25
```

'None' Keyword

Python में 'None' keyword का इस्तेमाल किया जाता है लेकिन दूसरे Programming languages में 'null' keyword का इस्तेमाल किया जाता है | 'None' का मतलब कोई value नहीं होती | अगर function कोई value return नहीं करता है तो 'None' return करता है |

Source Code :

```
print(None == '')
print(None == 0)
print(None == None)
print(type(None))
```

Output :

```
False
False
True
<class 'NoneType'>
```

'not'(Logical NOT) Keyword

'not' keyword का इस्तेमाल जब दिए गए Operand को true से false या false से true में convert करता है |

0 = True
1 = False

Truth Table of 'and'

x	not x
0	1
1	0

Source Code :

```
print(not(5!=6))  #False
print(not(5!=5))  #True
```

Output :

```
False
True
```

'or'(Logical OR) Keyword

'or' keyword का इस्तेमाल जब दिए गए operands में से एक भी operand true होता है तब true return किया जाता है |

0 = True
1 = False

Truth Table of 'or'

x	y	x or y
0	0	0
0	1	0
1	0	0
1	1	1

Source Code :

```
print(5!=6 or 5==6)  #True
print(5!=6 or 5==5)  #True
print(5!=6 or 5==6 or 4<6) #True
print(5==6 or 5>6 or 6<6) #False
```

Output :

```
True
True
True
False
```

'pass' Keyword

'pass' keyword का इस्तेमाल Loops, functions और classes को रिक्त रखा है | ये एक null statement/block होता है | 'pass' का इस्तेमाल जब किया जाता है तब ये कुछ नहीं करता है |
pass statement का इस्तेमाल एक ही line पर या seperate line पर भी किया जाता है |

Source Code :

```
def func(): pass
while 4 < 5:
    pass
```

Output :

```
```

'raise' Keyword

'raise' keyword का इस्तेमाल Exception Handling के लिए किया जाता है | raise keyword से custom exception/error का निर्माण किया जाता है | ज्यादातर programming langauges में 'throw' keyword का इस्तेमाल किया जाता है लेकिन Python में 'raise' का इस्तेमाल किया जाता है |

Source Code :

```
a = 12
b = 10
if a > b:
    raise ValueError("a should be smaller than b")
```

Output :

```
Traceback (most recent call last):

    raise ValueError('a should be smaller than b')
ValueError: a should be smaller than b
```

'return' Keyword

'return' keyword का इस्तेमाल function के आखिरी में किया जाता है | function को अगर कोई value return नहीं की जाती है तो null/None return किया जाता है |

Source Code :

```
def func(a):
    b = a * a
    return b
print(func(5))
```

Output :

```
25
```

'True' Keyword

'True' keyword ये एक Boolean value है | Logical या Comparison operaions में boolean value(True, False) return किया जाता है | True की value '1' और False की value '0' होती है |

Source Code :

```
print(True == 1)
print(False == 0)
print(1 == 1)
print(True + True)
```

Output :

```
True
True
True
2
```

'try' Keyword

'try' keyword का इस्तेमाल exception handling में किया जाता है | जिस code exception/error check करना है उस code को try Block पर लिखा जाता है |

Source Code :

```
try:
    a
except NameError:
    print("Value of a is not defined")
```

Output :

```
Value of a is not defined
```

'while' Keyword

'while' keyword का इस्तेमाल looping के लिए किया जाता है | जब तक while condition true होती है तब तक statement iterate होता रहता है और अगर condition false या loop break किया जाता है तब loop का iteration बंद हो जाता है |

Source Code :

```
a = 0
while a < 10:
    print("Value of a is", a)
    a = a + 1
```

Output :

```
Value of a is 0
Value of a is 1
Value of a is 2
Value of a is 3
Value of a is 4
Value of a is 5
Value of a is 6
Value of a is 7
Value of a is 8
Value of a is 9
```

'with' Keyword

'with' keyword का इस्तेमाल file handling के लिए किया जाता है | जब file के लिए 'with_as' statement का इस्तेमाल किया जाता है तब file को close करने की जरुरत नहीं पड़ती है |

निचे दिए गए दोनों examples का मतलब एक ही है |

Without 'with'

Source Code :

```
file = open("textfile.txt", "w")
file.write("Hello World")
file.close()
```

textfile.txt

```
Hello World
```

With 'with'

Source Code :

```
with open("textfile.txt", "w") as file :
    file.write("Hello World")
```

textfile.txt

```
Hello World
```

'yield' Keyword

'yield' keyword का इस्तेमाल function के लिए return statement जैसे किया जाता है | 'yield' keyword; generator को return करता है | return statement; function के आखिर में function terminate करता है लेकिन 'yield' statement; function को suspend करके वहा से आगे वापस शुरू होता है | Normal function जहा पर ख़त्म होता है वहा पर वापस नहीं आता है |

Source Code :

```
def func_name():
    yield 5
    yield 10
    yield 15
for y in func_name():
    print(y)
```

Output :

```
5
10
15
```

next() function का इस्तेमाल 'yield' के लिए किया जाता है | next() function generator object को लेकर अगली value return करता है |

Source Code :

```
def func_name():
    yield 5
    yield 10
    yield 15
y = func_name()
print(next(y))
print(next(y))
print(next(y))
```

Output :

```
5
10
15
```

What is Variable

- Variable जब create किया जाता है तब interpreter द्वारा value को store करने के लिए memory location आरक्षित की जाती है |
- Variable पर कोई भी data type की value store की जा सकती है | जैसे कि, Number, string, list, tuple, dictionary

Assigning Value to Variable

- Python में declaration की जरुरत नहीं होती है | जब variable पर value assign होती है तब automatically declaration होता है |
- declaration न होने के कारण Python में variable की default value नहीं होती है |

Source Code :

```
a = 5      #Number
b = "Hello" #string
c = [2, 5, 9] #list

print(a, b, c)
```

Output :

```
5 Hello [2, 5, 9]
```

Changing Variable's Value

- Python में variable की value change या re-assign की जा सकती है |

Source Code :

```
a = 5
print(a)
a = "Hello"
print(a)
a = [4, 5, 8]
print(a)
```

Output :

```
5
Hello
[4, 5, 8]
```

Assigning Single Value to Multiple Variables

- Python में एक ही value एक से ज्यादा variables पर assign की जा सकती है |

Source Code :

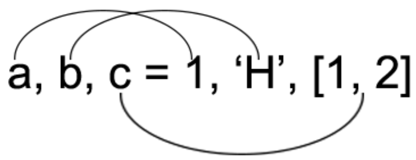
```
a = b = c = d = "Hello"
print(a)
print(b)
print(c)
print(d)
```

Output :

```
Hello
Hello
Hello
Hello
```

Assigning Value to Variable according to order

- Python में क्रमनुसार variable पर value store की जाती है |
- Example पर एक ही memory location multiple variables और उनकी values assign की गयी है |



Source Code :

```
a, b, c = 1, 'H', [1, 2]
print(a)
print(b)
print(c)
```

Output :

```
1
H
[1, 2]
```

Variables Concatenation

- Python में एक ही data types के variables concatenate किय जा सकते है |
- Example पर str() function का इस्तेमाल object को integer से string में convert करने के लिए किया गया है |

Source Code :

```
a = 1
b = 2
print(a + b)
print(str(a) + str(b))
c = "Hello"
print(str(a) + c)
```

Output :

```
3
12
1Hello
```

Types of Variable

Python में variable के दो प्रकार है |

1. Local Variables
2. Global Variables

1.Local Variables

Local Variables; functions के अन्दर होते है | उनकी visibility सिर्फ function के अन्दर होती है, जब वो function के बाहर आते है तब destroy हो जाते है |

Source Code :

```
def func():
    a = 5 #local variable
    print(a)
func()
print(a)
```

Output :

```
5
Traceback (most recent call last):
    print(a)
NameError: name 'a' is not defined
```

2.Global Variables

Global Variables; function के बाहर होते है | उनकी visibility function के अन्दर और बाहर होती है | उनका scope पूरे program पर होता है |

Source Code :

```
a = 10 #global variable
def func():
    print(a)
func()
print(a)
```

Output :

```
10
10
```

Example पर local और global ये दोनों variables declared किये गए है | function के बाहर का variable lobal है और अन्दर का variable local है | global variable का scope function के अन्दर और बाहर होता है लेकिन function के अन्दर अलग से variable declaration होने के कारण func() call करते ही variable की value change हो जाती है |

Source Code :

```
a = 10 #global variable
def func():
    a = 5 #local variable
    print(a)
func() #print local
print(a) #print global
```

Output :

```
5
10
```

With 'global' and Without 'global' Keyword

function में variable के लिए 'global' keyword का भी इस्तेमाल किया जाता है |

Without global

Example पर 'a' variable के declaration से पहले ही 'a' variable को print किया गया है, इसके कारण 'UnboundLocalError' ये exception occur हुआ है |

Source Code :

```
def func():
    print(a)
    a = "local"
    print(a)

a = "global"
func()
print (a)
```

Output :

```
in func
    print(a)
UnboundLocalError: local variable 'a'
referenced before assignment
```

With global

Example पर global keyword का इस्तेमाल शुरुआत में ही किया गया है और उसके बाद variable a को print किया गया है | program में जहा पर भी 'a' नामक global variable होगा वो func() call करते ही पहले print हो जायेगा |

Source Code :

```
def func():
    global a
    print(a) #print global
    a = "local"
    print(a) #print local

a = "global"
func()
print(a) #print local
```

Output :

```
global
local
local
```

No-Local or No-Global Variable

'nonlocal' variable का इस्तेमाल nested funtion के लिए किया जाता है | 'nonlocal' variables ये global भी नहीं होते और global भी नहीं होते है | अगर inner function में उनको लिया जाता है तो outer functions में उनकी values change नहीं होती है |

Without nonloal

Source Code :

```
var = 2
def outer():
    var = 5
    def inner():
        var = 10
        print("inner : ", var)
    inner()
    print("outer : ", var)
outer()
print("global : ", var)
```

Output :

```
inner :  10
outer :  5
global :  2
```

With nonlocal

Source Code :

```
var = 2
def outer():
    var = 5
    def inner():
        nonlocal var
        var = 10
        print("inner : ", var)
    inner()
    print("outer : ", var)
outer()
print("global : ", var)
```

Output :

```
inner :  10
outer :  10
global :  2
```

Number

Numbers Data Types में numeric values; store की जाती है | Python में Integer(int), Floating-Point(float) और Complex(complex) ये तीन प्रकार के Numbers data types होते है | Number Objects; immutable होते है, मतलब जब Number Object को create किया जाता है तब उसकी value बदली नहीं जा सकती है |

Number Data Types के तीन प्रकार होते है |

- Integer Number Data Type
- Floating-point Number Data Type
- Complex Number Data Type

Note : type() function का इस्तेमाल data type check करने के लिए किया जाता है |

Integer Number Data Type

Integer Data Type में Positive और Negative Numeric Values होती है लेकिन उनका अपूर्णांकित हिस्सा नहीं होता है | इस data type के range की कोई limit नहीं होती है |

Source Code :

```
a = 5
print(a, type(a))
b = -5
print(b, type(b))
```

Output :

```
5 <class 'int'>
-5 <class 'int'>
```

Floating-point Number Data Type

Floating-Point Number Data Type में Positive और Negative Values होती है लेकिन उनका अपूर्णांकित हिस्सा होता है | अपूर्णांकित हिस्से की limit 15 तक होती है |

Source Code :

```
a = 5.12345678910121755858545 6
print(a, type(a))
b = -5.45
print(b, type(b))
c = 5.45
print(c, type(c))
```

Output :

```
5.123456789101217 <class 'float'>
-5.45 <class 'float'>
5.45 <class 'float'>
```

Complex Number Data Type

Complex Numbers में real और imaginary हिस्सा होता है जैसे कि, (a + bj) यहाँ पर a ये real Number और b ये imaginary हिस्सा होता है |

Source Code :

```
a = 5 + 2j
print(a, type(a))
```

Output :

```
(5+2j) <class 'complex'>
```

Number Type Conversion

Python में अलग-अलग data type को एक ही data type में convert किया जाता है | Python में कुछ functions ऐसे होते है कि जो data type convert करने में मजबूर करते है |

कुछ Type Converter Functions :

- int(a)
- float(a)
- complex(a, b)

int() Type Conversion

Syntax

```
int(a)
```

Example पर float को integer में convert किया गया है | String में अगर characters होते है तो उसे Integer में convert नहीं किया जा सकता लेकिन String में अगर integer Numbers होते है तो उसे Integer में convert किया जा सकता है |

Source Code :

```
a = "123"
print(int(a))
b = 2.658
print(int(b))
```

Output :

```
123
2
```

float() Type Conversion

Syntax

```
float(a)
```

Example पर integer को float में convert किया गया है | String में अगर characters होते है तो उसे Floating-Point Number में convert नहीं किया जा सकता लेकिन String में अगर integer या Floating-Point Numbers होते है तो उसे Floating-Point में convert किया जा सकता है |

Source Code :

```
a = "123"
print(float(a))
b = "123.564"
print(float(b))
c = 2
print(float(c))
```

Output :

```
123.0
123.564
2.0
```

complex() Type Conversion

Syntax

```
complex(a, b)
```

Parameters :

a : यहाँ पर 'a' ये real Number होता है |
b : Optional. यहाँ पर 'b' ये imaginary हिस्सा होता है | अगर दिया नहीं जाता तो default value '0' होती है |

Source Code :

```
a = "123"
print(complex(a))
b = "123.564"
print(complex(b))
c = 2
print(complex(c), end="\n\n")
d = 123
e = 58
print(complex(d, e))
f = "123"
g = 58
print(complex(f, g))
```

Output :

```
(123+0j)
(123.564+0j)
(2+0j)
(123+58j)
    print(complex(f, g))
TypeError: complex() can't take second arg if first is a string
```

Note : सिर्फ real number दिया जाता है तो वो String number हो सकता है, लेकिन अगर real number के साथ imaginary हिस्सा लिया जाता है तो वो दोनों Integer या Floating-Point Numbers होने चाहिए |

Strings

String ये characters का set होता है | characters; letters, numbers या special symbols हो सकते है | Python में single(' ') या double(" ") quotes के अन्दर लिखे जाते है | String ये immutable data type है |

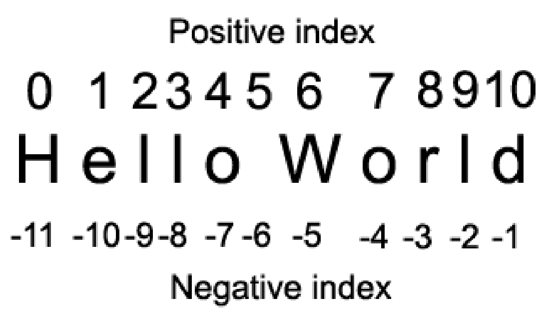
Source Code :

```
str1 = "Hello World"
str2 = "Hello Friends"
print(str1)
print(str2)
```

Output :

```
Hello World
Hello Friends
```

ज्यादातर Programming languages में string को index में print किया जाता है उसी प्रकार से Python में भी string को indexes से print किया जाता है | string के सबसे पहले character को index '0' से शुरू होती है और string के सबसे आखिरी index '-1' होती है |



Source Code :

```
str = "Hello World"
print("First letter in string :", str[0])
print("Last letter in string :", str[-1])
print("Second last letter in string :", str[-2])
```

Output :

```
First letter in string : H
Last letter in string : d
Second last letter in string : l
```

List

Python के list data type में एक से ज्यादा items होते है | हर एक item को comma(,) से seperate किया जाता है | list के सभी items को square bracket([]) के अन्दर close किये जाता है |
List ये एक compound data type है जिसमे किसी भी data types के items लिए जा सकते है | list ये एक mutable data type है | इन data type के items की values change की जा सकती है |

Source Code :

```
list = [1, "Hello", 5.6, (1, "Hii")]
for i in list:
    print(i)
```

Output :

```
1
Hello
5.6
(1, 'Hii')
```

Tuple

Python के list data type में एक से ज्यादा items होते है | ये list data type के जैसे ही होता है | हर एक item को comma(,) से seperate किया जाता है | Tuple के सभी items को parenthesis(()) के अन्दर close किये जाता है |
Tuple ये एक compound data type है जिसमे किसी भी data types के items लिए जा सकते है | list ये एक immutable data type है | इन data type के items की values change नहीं की जा सकती है |

Source Code :

```
tuple = (1, "Hello", 5.6, (1, "Hii"))
for i in tuple:
    print(i)
tuple[0] = 3  #trying to changing 0th index
print(tuple[0])
```

Output :

```
1
Hello
5.6
(1, 'Hii')
Traceback (most recent call last):
    tuple[0] = 3  #trying to changing 0th index
TypeError: 'tuple' object does not support item assignment
```

Dictionary

Dictionary Data Type में keys और values की pairs होती है | हर key value के pair को comma(,) से और key और value को colon(:) से seperate किया जाता है | Dictionary के सभी keys और values को curly braces({}) में लिखा जाता है | ये एक immutable data type है |

Source Code :

```
dict = {1:"H", 5:"e", 7:"l", 8:"l", 9:"o"}
print(dict[5])
print(dict[9])
```

Output :

```
e
o
```

Set

Set Data Type ये items का unordered collection होता है | set में दिए हुआ हर एक item नया होता है | अगर duplicate item मिल जाता है तो उसे remove किया जाता है | set data type के items को curly braces({}) के अन्दर लिखा जाता है |

Source Code :

```
set1 = {"Ramesh", "Suresh", "Kamlesh"}
for i in set1:
    print(i)

set2 = {3, 5, 8, 7, 1, 3}
for j in set2:
    print(j)
```

Output :

```
Suresh
Kamlesh
Ramesh
1
3
5
7
8
```

Indenting Code

Python में Code indentation को काफी महत्व दिया गया है | Python में Code Indentation का इस्तेमाल functions, classes, Loops और control statements के लिए किया जाता है |
Python में curly braces({}) की जगह code indentation का इस्तेमाल किया जाता है |
Python में जब code indentation में colon(:) या delimiter दिया जाता है तब automatically अगले line पर interpreter द्वारा tab( ) दिया जाता है |

For 'Function'

Source Code :

```
def func():
    a = 5
    print(a)
func()
```

Output :

```
5
```

For 'For Loop'

Source Code :

```
list = [1, 5, 8, 9]
for i in list:
    print(i)
```

Output :

```
1
5
8
9
```

For 'Class'

Source Code :

```
class myClass:
    def func():
        a = 5
        print(a)
myClass.func()
```

Output :

```
5
```

Arithmetic Operators

Python में Code indentation को काफी महत्व दिया गया है | Python में Code Indentation का इस्तेमाल functions, classes, Loops और control statements के लिए किया जाता है |
Python में curly braces({}) की जगह code indentation का इस्तेमाल किया जाता है |
Python में जब code indentation में colon(:) या delimiter दिया जाता है तब automatically अगले line पर interpreter द्वारा tab( ) दिया जाता है |

Operators	**Sign**	**Description**
Addition	+	left और right operand को add करता है |
Subtraction	-	left operand से right operand को subtract करता है |
Multiplication	*	left और right operand को multiply करता है |
Division	/	left operand से right operand को divide करता है |
Modulus	%	left operand से right operand को divide करके remainder return करता है |
Floor Division	//	left operand से right operand को divide करके floor value return करता है |
Exponent	**	left operand का power right operand होता है |

Addition Arithmetic Operator

Source Code :

```
print(5 + 6)
print(5.6 + 7.9)
```

Output :

```
11
13.5
```

Subtraction Arithmetic Operator

Source Code :

```
print(5 - 6)
print(5.6 - 7.9)
print(7.9 - 5.6)
```

Output :

```
-1
-2.3000000000000007
2.3000000000000007
```

Multiplication Arithmetic Operator

Source Code :

```
print(5 * 6)
print(5.6 * 7.9)
print(7.9 * 5.6)
```

Output :

```
30
44.24
44.24
```

Division Arithmetic Operator

Source Code :

```
print(5/2)
print(4/2)
print(4.6/2.6)
print(2/4)
print(0/4)
print(4/0)
```

Output :

```
2.5
2.0
1.769230769230769
0.5
0.0
    print(4/0)
ZeroDivisionError: division by zero
```

Modulus Arithmetic Operator

Source Code :

```
print(4%2)
print(4%5)
print(4%3)
print(4.5%3.5)
```

Output :

```
0
4
1
1.0
```

Floor Division Arithmetic Operator

Source Code :

```
import math

print(4/3)
print(4//3)
print(math.floor(4/3))
print(3/4)
print(3//4)
print(math.floor(3/4))
```

Output :

```
1.3333333333333333
1
1
0.75
0
0
```

Exponent Arithmetic Operator

Source Code :

```
import math

print(4**2)
print(math.pow(4, 2))
print(4**10)
print(math.pow(4, 10))
```

Output :

```
16
16.0
1048576
1048576.0
```

Relational Operators

Operators	**Sign**	**Description**
Equal to	==	दो operands equal होते है तो ये True return करता है |
Not Equal to	!=	दो operands equal नहीं होते है तो ये True return करता है |
Less than	<	left operand जब right operand से छोटा होता है तो True return करता है |
Greater than	>	left operand जब right operand से बड़ा होता है तो True return करता है |
Less than or Equal to	<=	left operand जब right operand से छोटा होता है या equal होता है तो True return करता है |
Greater than or Equal to	>=	left operand जब right operand से बड़ा होता है या equal होता है तो True return करता है |

Equal to(==) Relational Operator

Source Code :

```
a = 5
b = 6
print(a == b)
print(a == 5)
print((a<b) == (b>a))
```

Output :

```
False
True
True
```

Not Equal to(!=) Relational Operator

Source Code :

```
a = 5
b = 6
print(a != b)
print(a != 5)
print((a<b) != (b<a))
```

Output :

```
True
False
True
```

Less than(<) Relational Operator

Source Code :

```
a = 5
b = 6
print(a < b)
print(a > b)
```

Output :

```
True
False
```

Greater than(>) Relational Operator

Source Code :

```
a = 5
b = 6
print(a > b)
print(a < b)
```

Output :

```
False
True
```

Less than or Equal to(<=) Relational Operator

Source Code :

```
a = 5
b = 6
c = 5
print(a <= b)
print(a <= c)
```

Output :

```
True
True
```

Greater than or Equal to(>=) Relational Operator

Source Code :

```
a = 5
b = 6
c = 5
print(a >= b)
print(a >= c)
```

Output :

```
False
True
```

Logical Operators

Operators	**Sign**	**Description**
Logical AND	and	दो conditions अगर True होती है तब ये True return करता है	
Logical OR	or	दो conditions में से एक भी condition अगर True होती है तब ये True return करता है	
Logical NOT	not	ये Operator false को true या true को false कर देता है	

Logical AND(and) Logical Operator

Source Code :

```
a = 5
b = 6
print((a<b) and (b>a))
```

Output :

```
True
```

Logical OR(or) Logical Operator

Source Code :

```
a = 5
b = 6
print((a<b) or (b>a))
print((a<b) or (b<a))
```

Output :

```
True
True
```

Logical NOT(not) Logical Operator

Source Code :

```
a = 5
b = 6
print(not(a>b))
print(not(a<b))
```

Output :

```
True
False
```

Assignment Operators

Operators	Sign	Example	Same as
Assignment	=	c = a + b	
Add Assignment	+=	c += a	c = c + a
Subtract Assignment	-=	c -= a	c = c - a
Multiply Assignment	*=	c *= a	c = c * a
Divide Assignment	/=	c /= a	c = c / a
Mudulus Assignment	%=	c %= a	c = c % a
Exponent Assignment	**=	c **= a	c = c ** a
Floor Divide Assignment	//=	c //= a	c = c // a
Bitwise AND Assignment	&=	c &= a	c = c & a
Bitwise OR Assignment	|=	c |= a	c = c | a
Bitwise XOR Assignment	^=	c ^= a	c = c ^ a
Bitwise Left Shift Assignment	<<=	c <<= a	c = c << a
Bitwise Right Shift Assignment	>>=	c >>= a	c = c >> a

Assignment Operator

Source Code :

```
a = 3
b = 4
c = a + b
print(c)
```

Output :

```
7
```

Add Assignment Operator

Source Code :

```
a = 3
c = 6
c += a  #c = c + a
print(c)
```

Output :

```
9
```

Subtract Assignment Operator

Source Code :

```
a = 3
c = 6
c -= a   #c = c - a
print(c)
```

Output :

```
3
```

Multiply Assignment Operator

Source Code :

```
a = 3
c = 6
c *= a   #c = c * a
print(c)
```

Output :

```
18
```

Divide Assignment Operator

Source Code :

```
a = 3
c = 6
c /= a   #c = c / a
print(c)
```

Output :

```
2.0
```

Modulus Assignment Operator

यहाँ पर 'c' variable को 'a' variable से divide करके उनका remainder return करके 'c' variable पर store किया जाता है |

Source Code :

```
a = 3
c = 6
c %= a   #c = c % a
print(c)
```

Output :

```
0
```

Exponent Assignment Operator

यहाँ पर 'c' variable का power 'a' variable है और उनकी value 'c' variable में store की गयी है |

Source Code :

```
a = 3
c = 6
c **= a   #c = c ** a
print(c)
```

Output :

```
216
```

Floor Divide Assignment Operator

यहाँ पर 'c' variable को 'a' variable से divide करके उनकी floor value 'c' variable पर store की गयी है |

Source Code :

```
a = 3
c = 6
c //= a   #c = c // a
print(c)
#same as
import math
a = 3
c = 6
c = math.floor(c/a)
print(c)
```

Output :

```
2
2
```

Bitwise AND(&) Assignment Operator

Bitwise AND(&) के बारे में पढने के लिए Bitwise Operators को पढ़े |

Source Code :

```
a = 3
c = 6
c &= a   #c = c & a
print(c)
```

Output :

```
2
```

Bitwise OR(|) Assignment Operator

Bitwise OR(|) के बारे में पढने के लिए Bitwise Operators को पढ़े |

Source Code :

```
a = 3
c = 6
c |= a   #c = c | a
print(c)
```

Output :

```
7
```

Bitwise XOR(^) Assignment Operator

Bitwise XOR(^) के बारे में पढने के लिए Bitwise Operators को पढ़े |

Source Code :

```
a = 3
c = 6
c ^= a   #c = c ^ a
print(c)
```

Output :

```
5
```

Bitwise Left Shift(<<) Assignment Operator

Bitwise Left Shift(<<) के बारे में पढने के लिए Bitwise Operators को पढ़े |

Source Code :

```
a = 3
c = 6
c <<= a   #c = c << a
```

Output :

```
48
```

Bitwise Right Shift(>>) Assignment Operator

Bitwise Right Shift(>>) के बारे में पढने के लिए Bitwise Operators को पढ़े |

Source Code :

```
a = 3
c = 6
c >>= a   #c = c >> a
print(c)
```

Output :

```
0
```

Bitwise Operators

Operators	**Sign**
Bitwise AND	&
Bitwise OR	|
Bitwise XOR	^
Bitwise Complement	~
Bitwise Left Shift	<<
Bitwise Right Shift	>>

Truth Table for &, |, ^

a	**b**	**a & b**	**a | b**	**a ^ b**
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Operation on AND (a&b)

अगर a = 20, b = 12 हो तो,

Decimal Value	**Binary Value**
20	0 0 0 1 0 1 0 0
12	0 0 0 0 1 1 0 0
4	0 0 0 0 0 1 0 0

Operation on AND (a&b)

अगर a = 20, b = 12 हो तो,

Decimal Value	**Binary Value**
20	0 0 0 1 0 1 0 0
12	0 0 0 0 1 1 0 0
28	0 0 0 1 1 1 0 0

Operation on XOR(a^b)	
अगर a = 20, b = 12 हो तो,	
Decimal Value	**Binary Value**
20	0 0 0 1 0 1 0 0
12	0 0 0 0 1 1 0 0
24	0 0 0 1 1 0 0 0

Binary Left Shift(<<) and Right Shift(>>)

Left Shift(<<) for e.g. a=20; /* 0001 0100 */ a << 2 में numeric value के binary value में हर binary number को 2 binary numbers left side से shift करता है | for e.g.a=20; /* 0001 0100 */ तो इसका 0101 0000 मतलब 80 हो जायेगा |

Right Shift(>>) for e.g. a=20; /* 0001 0100 */ ये Left shift से बिलकुल उलट है | Right Shift a>> 2 में numeric value के binary value में हर binary number को 2 binary numbers right side से shift करता है | for e.g.a=20; /* 0001 0100 */ तो इसका 0000 0101 मतलब 5 हो जायेगा |

Complement Operator (~)

Operation on Complement(~)	
Decimal Value	**Binary Value**
~20	0 0 0 0 1 1 0 0
243	1 1 1 1 0 0 1 1

यहाँ पर Output -13 आने के बजाय 243 आया ऐसा क्यों ?

2's Complement of 243 -(reverse of 243 in binary + 1)

Operation on 2's Complement (~)		
Decimal Value	Binary Value	2's Complement
243	1111 0011	-(0000 1100+1) = -(0000 1101) = -13(output)

Source Code :

```
a = 20 # 0001 0100
b = 12 # 0000 1100
c = a & b
print("value of c is ",c) # 4 = 0000 0100
c = a | b
print("value of c is ",c) # 28 = 0001 1100
c = a ^ b
print("value of c is ",c) # 24 = 0001 1000
c = a << 2
print("value of c is ",c) # 80 = 0101 0000
c=a >> 2
print("value of c is ",c) # 5 = 0000 0101
print("value of b is ",~b) # -13 = 1111 0011
```

Output :

```
value of c is 4
value of c is 28
value of c is 24
value of c is 80
value of c is 5
value of b is -13
```

Identity Operators

Operators	Description
is	जब एक जैसे Object का वर्णन किया जाता है तो True return करता है	
is not	जब अलग-अलग Object का वर्णन किया जाता है तो True return करता है	

'is' Identity Operator

Source Code :

```
a = 4
b = 5
c = 4
print(a is c)
print((a<b) is (b>a))
```

Output :

```
True
True
```

'is not' Identity Operator

Source Code :

```
a = 4
b = 5
print(a is not b)
```

Output :

```
True
```

Membership Operators

Operators	Description
in	अगर collection(list, tuple, dictionary, string, set) में element या value मिली तो True return करता है	
not in	अगर collection(list, tuple, dictionary, string, set) में element या value नहीं मिली तो True return करता है	

'in' Membership Operator

Source Code :

```
a = "Hello World"
print("ll" in a)
list = [1, "o", 8, 5]
print(1 in list)
```

Output :

```
True
True
```

'not in' Membership Operator

Source Code :

```
a = "Hello World"
print("ll" in a)
list = [1, "o", 8, 5]
print(1 in list)
```

Output :

```
True
True
```

Input and Output

Introduction

Python में input/output के लिए कुछ functions का इस्तेमाल किया जाता है |
User के keyboard से input लेने के लिए 'input()' इस function का इस्तेमाल किया जाता है और User के screen पर कुछ print करने के लिए 'print()' इस function का इस्तेमाल किया जाता है |

input() Function for taking User Input

Python में जब User से कुछ letters या numbers लेने की बारी आती है तब 'input()' function का इस्तेमाल किया जाता है |

Syntax

```
input("SomeText")
```

Parameter

SomeText : Optional.User को क्या input देना है उसके बारे में यहाँ पर कुछ text दिया जाता है |

Source Code :

```
a = input("Enter your name : ")
print("Your name is", a)
```

Output :

```
Enter your name : Rakesh
Your name is Rakesh
```

जब User; numeric, letters या special symbols; input लेते है तो उसका data type 'string' होता है |

Source Code :

```
a = input("Enter Number : ")
print("Entered input's type is", type(a))
```

Output :

```
Enter Number : 4
Entered input's type is
```

Type Conversion

जब उस String data type को Number(int, float, complex number) में convert करना हो तो Type Conversion functions का इस्तेमाल करना पड़ता है |

Source Code :

```
a = input("Enter Number : ")
print("Entered input's type is", a)
b = int(a)
print("Entered input's type is", b)
print("Entered input's type is", type(b))
c = float(a)
print("Entered input's type is", c)
print("Entered input's type is", type(c))
d = complex(a)
print("Entered input's type is", d)
print("Entered input's type is", type(d))
```

Output :

```
Enter Number : 4
Entered input's type is 4
Entered input's type is 4
Entered input's type is
Entered input's type is 4.0
Entered input's type is
Entered input's type is (4+0j)
Entered input's type is
```

print Function

अभीतक User के screen पर output दिखाने के लिए 'print()' function का इस्तेमाल किया गया था, लेकिन print() function के लिए एक से ज्यादा parameters होते है | जिनका इस्तेमाल एक से ज्यादा newline देना या file handling के लिए किया जाता है |

Source Code :

```
a = 5
print("Value of a is", a) #Output : Value of a is 5
```

Syntax

```
print(value1,value2,...,valueN, sep="", end="\n", file=sys.stdout, flush=False)
```

Parameters

value1,value2,...,valueN : यहाँ पर एक या एक से ज्यादा values दी जाती है | हर value को comma(,) से seperate किया जाता है |
sep=" " : Optional. जब एक से ज्यादा values दी जाती है तो default 'sep' parameter द्वारा space( ) दिया जाता है | User चाहे तो 'sep' parameter पर कोई भी value दे सकता है |
end="\n" : Optional. 'end' parameter पर '\n'(newline) default दिया जाता है | User चाहे तो 'end' parameter पर कोई भी value दे सकता है |
file=sys.stdout : Optional. 'file' parameter पर 'sys.stdout'(print entered text on screen) default दिया जाता है | User चाहे तो 'file' parameter पर किसी भी file को open कर सकता है |
flush=False : Optional. 'flush' parameter पर 'False' ये value default होती है | जब False दिया जाता है तब stream को flush नहीं किया जाता है या जब 'True' दिया जाता है तब stream को जबरन flush किया जाता है |

print() function null/None return करता है |

Example for print() Function with 'sep' Parameter

Example पर '!!!!!' इस string से seperate किया गया है |

Source Code :

```
print("Hello World", "Hello Friends", "Hello Ramesh", sep="!!!!!")
```

Output :

```
Hello World!!!!!Hello Friends!!!!!Hello Ramesh
```

Example for print() Function with 'end' Parameter

Example में end पर '\n\n\n\n\n' इस string दिया गया है |

Source Code :

```
print("Hello World", end="\n\n\n\n\n")
print("Hello Friends")
```

Output :

```
Hello World



Hello Friends
```

Example for print() Function with 'file' Parameter

Example पर print() function के 'file' parameter द्वारा 'textfile.txt' इस file पर 'Hello World' ये string write किया गया है |

Source Code :

```
f = open("textfile.txt", "w")
print("Hello World", file = f)
f.close()
```

textfile.txt

```
Hello World
```

Note : print() function में binary files पर operation नहीं किया जाता है | इसके बदले में write() function का इस्तेमाल किया जाता है |

While Loop

Loop एक ही statement को बार-बार execute करता है |
Looping में indentation का काफी महत्व होता है |
while Loop का इस्तेमाल जब तक condition true होती है तब तक statement execute होता रहता है और जब condition false हो जाती है तब loop का iteration; stop हो जाता है |

Syntax

```
while (expression):
    while_statement(s)
```

Example

```
a = 0
while (a < 10) :
   print("Value a is ", a)
   a = a + 1
```

Output :

```
Value a is  0
Value a is  1
Value a is  2
Value a is  3
Value a is  4
Value a is  5
Value a is  6
Value a is  7
Value a is  8
Value a is  9
```

while Loop with else Statement

while Loop के else का सम्बन्ध बनाया गया है | जब तक condition true होती है तब तक while loop का statement iterate होता रहता है और condition false होती है तब control pass होके else पर जाकर else का statement execute होता है |

Source Code :

```
a = 0
while(a < 5):
    print("Value of a is ", a);
    a = a + 1
else:
    print("Out of Loop");
```

Output :

```
Value of a is  0
Value of a is  1
Value of a is  2
Value of a is  3
Value of a is  4
Out of Loop
```

infinite Loop

जब तक while का expression 'True' होता है तब तक वो अपना statement execute करता रहता है |
Example पर condition false ही नहीं हो रही है इसीलिए infinite times; statement को execute किया जा रहा है |

Source Code :

```
while(True) :  #expression is always true
   print("Hello")
```

Output :

```
Hello
Hello
Note : "Hello" String print infinite times because while expression is always true
```

For In Loop

Loop का इस्तेमाल sequence के elements को iterate करने के लिए किया जाता है |for Loop का इस्तेमाल कोई भी sequence के सभी elements को display करने के लिए किया जाता है |sequence ये list, tuple, dictionary और string होता है |

Syntax

```
for variable in sequence
    for_statement(s)
```

जब iteration शुरू होता है तब variable पर sequence का element assign किया जाता है और दिए गए statement को execute किया जाता है | जितने elements की संख्या होती है उस संख्या तक iteration होता रहता है और उसके बाद loop का control loop के बाहर आ जाता है |

Example for Iterating list sequence

```
a = [5, 6, 9, 8, 5, 7]
for n in a :
        print(n)
```

Output :

```
5
6
9
8
5
7
```

Example for Iterating tuple sequence

```
tuple = (1, 2, [3, 4, 5], 6, 7, 8)
for a in tuple:
        print(a)
```

Output :

```
1
2
[3, 4, 5]
6
7
8
```

Example for Iterating dictionary sequence

```
dict = {1:6, 2:5, 3:3, 4:8, 5:4, 6:2, 7:5, 8:4, 9:11, 12:45 }
for a in dict:
        print(a, dict[a])
```

Output :

```
1 6
2 5
3 3
4 8
5 4
6 2
7 5
8 4
9 11
12 45
```

Example for Iterating string sequence

```
str = "Hello World"
for a in str:
        print(a)
```

Output :

```
H
e
l
l
o
 
W
o
r
l
d
```

else with for Loop

for loop में जब sequence के elements का iteration ख़त्म हो जाएगा तब else का statement; execute होगा |

```
name = ['Rakesh', 'Ramesh', 'Suresh']
for i in name :
    print(i)
else:
        print("Outside of for Loop")
```

Output :

```
Rakesh
Ramesh
Suresh
Outside of for Loop
```

Control Statements

Python में if, if_else और if_elif_else ये तीन प्रकार के Control Statements होते है |
इन control statements से जब तक दिया हुआ expression true नहीं होता तब तक ये अपना statement; execute नहीं होता है |

If Statement

if Statement में जब expression true होता है तब वो अपने statement को execute होता है और अगर expression false होता है तब वो execute नहीं होता है |

Syntax

```
if expression :
    if_statement(s)
```

Example for if Statement

Source Code :

```
a = 2
b = 3
if(a < b) :
    print("a is less than b")
```

Output :

```
a is less than b
```

If Else Statement

if Statement में जब expression true होता है तब वो अपने statement को execute होता है और अगर expression false होता है तब else का statement; execute होता है |

Syntax

```
if expression :
if_statement(s)
else :
    else_statement(s)
```

Example for if Else Statement

Source Code :

```
a = 2
b = 2
if(a < b) :
    print("a is less than b")
else :
    print("a is greater than or equal to b")
```

Output :

```
a is greater than or equal to b
```

If Elif Else Statement

if Statement में जब expression true होता है तब वो अपने statement को execute होता है और अगर expression false होता है तब flow elif पर जाता है अगर elif का expression; true होता है तो उसका statement execute होता है और elif का expression; false होता है तब else का statement execute होता है |

Syntax

```
if expression :
if_statement(s)
elif expression :
elif_statement(s)
else :
    else_statement(s)
```

Example for if Elif Else Statement

Source Code :

```
a = 2
b = 2
if(a < b) :
   print("a is less than b")
elif(a > b) :
   print("a is greater than b")
else :
   print("a is equal to b")
```

Output :

```
a is equal to b
```

pass Statement

pass Statement ये null Statement होता है | जब pass Statement execute होता है तब कुछ नहीं होता है | Program में जब functions, loops, classes या control statement लिखा जाता है उनकी body उसके निचे ही लिखी जाती है लेकिन अगर उनकी body बाद में लिखनी हो तो 'pass' Statement का इस्तेमाल किया जाता है |

Syntax

```
pass
```

Example for pass Statement

Source Code :

```
class A: pass
def func(param):
    pass
func(10)
print("Hello World")
```

Output :

```
Hello World
```

break Statement

Example में अगर 'n' को 4 मिल जाता है तब for Loop का iteration बंद हो जाता है |

Syntax

```
break
```

Example for break Statement

Source Code :

```
nums = [1, 2, 3, 4, 5]
for n in nums :
   print(n)
   if(n == 4):
      break
```

Output :

```
1
2
3
4
```

Continue Statement

continue Statement से Loop में iterate हो रहे कुछ statement(s) को skip करके अगले statements को execute करता है |

Syntax

```
continue
```

Example for continue Statement

Source Code :

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for n in nums :
    if(n == 5) :
        print("Skipped Element :", n)
        continue
    print(n)
```

Output :

```
1
2
3
4
Skipped Element: 5
6
7
8
9
10
```

Functions

What is Function

Python Function ये statements का एक समूह होता है | हर एक Program में एक function तो होता ही है | जहाँ पर Function की statements की जरुरत होती है वहाँ पर Function को call किया जाता है |

Python Functions Advantages

Function में लिखा हुआ code बार-बार लिखना नहीं पड़ता |
बड़े Program को छोटे-छोटे function में विभाजित किया जा सकता है |
Function Programmer का समय, Program की space और memory बचाता है |
अगर Program में कहा पर error आ जाए तो उसे आसानी से निकाला जा सकता है |
जहाँ पर जरुरत हो वहाँ पर function को बार-बार call किया जा सकता है |

Types of Function

Python में दो प्रकार के functions होते है |

1. In-Built/Predefined Function
2. User-Defined Function

1. In-Built/Predefined Function

Python में अभीतक print() नामक function बहुत ही बार इस्तेमाल किया है | print() function ये Python में in-built function है |

2. User-Defined Function

Python में user-defined function में दो प्रकार पड़ते है |

2.1 Function Definition
2.2 Function Call

2.1 Function Definition

Syntax

```
def function_name(parameter(s)):
    "function_docstring"
    function_body
    return statement
```

def : Python में function को create करना हो तो शुरुआत में 'def' keyword का इस्तेमाल किया जाता है |
function_name : def keyword के बाद function का नाम दिया जाता है | हर function का नाम उसके statement से related हो तो अच्छा रहता है |
(parameter(s)) : Optional. parameters optional होते है parenthesis(()) नहीं होते है | function के name के बाद parenthesis(()) दिया जाता है | उन parenthesis में एक या एक से ज्यादा parameters दिए जाते है | parameters; optional होते है |
: : parenthesis के बाद colon(:) देना अनिवार्य होता है | colon देने के बाद Python interpreter द्वारा automatic code indent होता है |
"function_docstring" : Optional. यहाँ पर documentation string दिया जाता है |
function_body : Optional. ये function की body होती है | जिसमे कुछ local variables और statements हो सकते है | body में दिए हुए variables को global भी बनाया जा सकता है |
return statement : Optional. ये return statement होता है | return statement के लिए 'return' keyword का इस्तेमाल किया जाता है | ये statement से function end होता है | अगर return statement दिया नहीं जाता है तो default 'None' return होता है |

Example for Function Definition

```
#Function definition
def func(param):          #function name, parameter and colon
    "This is a docstring" #docstring
    print(func.__doc__)   #function body
    return param          #return statement
```

Example पर 'func' ये function का नाम है |
उसके बाद parenthesis(()) में 'param' नाम का एक parameter और parenthesis के बाहर colon(:) दिया गया है |
function के indent code में पहले docstring दी गयी है |
उसके बाद function की body में class attribute की मदद से docstring को print किया गया है
और आखिर में return statement में 'param' इस parameter को print किया गया है |

2.2 Function Call

Function के call में सिर्फ function का नाम और अगर function को parameter हो तो parameter की value दी जाती है | जबतक function को call नहीं किया जाता तबतक function का code execute नहीं होता है |

Example

```
#Function definition
def func(param):          #function name, parameter and colon
    "This is a docstring" #docstring
    print(func.__doc__)   #function body
    return param          #return statement
print("Call 1")
print(func(5))            #Function call
print("Call 2")
print(func(10))           #Function call
```

Output :

```
Call 1
This is a docstring
5
Call 2
This is a docstring
10
```

Python में जब function को call करके जिस data type की value as a argument दी जाती है तब उसका data type decide हो जाता है |

Example

```
def func(num, str):
    print(num)
    print(str)
print("Function Call1 :")
func("Hello", 2)
print("Function Call2 :")
func(2, "Hello")
```

Output :

```
Function Call1 :
Hello
2
Function Call2 :
2
Hello
```

Function Argument in Python

Python में चार प्रकार के function parameter होते है |

1. Default Argument
2. Required Argument
3. Keyword Argument
4. Variable Number of Argument

1. Default Argument

Default argument में definition पर argument के लिए default value '='(assignment operator) से set की जाती है |
Function Call पर जब definition पर assign किया हुआ argument नहीं दिया जाता तो default value pass की जाती है |

Source Code :

```
def DefArg(num=5, str="Hello"):
    print(num)
    print(str)
print("Call1")
DefArg()
print("Call2")
DefArg(8)
print("Call3")
DefArg(8, "Hii")
```

Output :

```
Call1
5
Hello
Call2
8
Hello
Call3
8
Hii
```

2. Required Argument

Required Argument में function call पर argument देना अनिवार्य होता है |
जब function definition पर argument दिया जाता है तब उसे function call पर उसकी value देना required होता है |

Source Code :

```
def ReqArg(num, str):
    print(num)
    print(str)
print("Call1")
ReqArg(5, "Hello World")
print("Call2")
ReqArg("Hello World", 5)
```

Output :

```
Call1
5
Hello World
Call2
Hello World
5
```

अगर function call पर argument दिया नहीं जाता है तो , TypeError का exception आ जाता है |

Source Code :

```
def ReqArg(num, str):
    print(num)
    print(str)
ReqArg()
```

Output :

```
    ReqArg()
TypeError: ReqArg() missing 2 required positional arguments: 'num' and 'str'
```

अगर function पर दो arguments है | अगर call करते वक्त एक ही argument value दी जाती है तो तब भी 'TypeError' exception आ जाता है |

Source Code :

```
def ReqArg(num, str):
    print(num)
    print(str)
ReqArg(5)
```

Output :

```
    ReqArg(5)
TypeError: ReqArg() missing 1 required positional argument: 'str'
```

3. Keyword Argument

अबतक function पर positional argument का इस्तेमाल किया गया है | Positional argument ये normal parameters होते है |
positional argument में जब values दी जाती है तब उसकी position की हिसाब से call पर value assign की जाती है |
Keyword Argument में function call पर दिया जाता है | Keyword Argument की मदद से arguments की positions को बदला जाता है |

Source Code :

```
def KeyArg(int, float, str):
    print("int :",int)
    print("float :",float)
    print("String :",str)
print("Call1")
KeyArg(int=10, float=1.5, str="H")
print("Call2")
KeyArg(float=2.8, int=20, str="G")
print("Call3")
KeyArg(str="R", float=8.6, int=5)
```

Output :

```
Call1
int : 10
float : 1.5
String : H
Call2
int : 20
float : 2.8
String : G
Call3
int : 5
float : 8.6
String : R
```

4. Variable Number of Arguments

किसी वक्त पर पता नहीं होता कि function पर कितने arguments pass करने है | ऐसे वक्त पर 'Number of Arguments' का महत्व काफी बढ़ जाता है |
जब number of argument का इस्तेमाल करना हो तो Function के definition पर argument से पहले '*'(asterisk) का इस्तेमाल किया जाता है |
'Function definition पर दिया हुआ argument 'tuple' जैसा होता है |

Source Code :

```
def MultiArg(*argTuple):
    print(argTuple)
    print(argTuple[0])
    print(argTuple[1])
    print(argTuple[2])
    for i in argTuple:
        print(i)
MultiArg(5, 10, "Hello")
```

Output :

```
(5, 10, 'Hello')
5
10
Hello
5
10
Hello
```

Anonymous or Lambda Function

जब Normal Function create किया जाता है तब function को विशिष्ट नाम दिया जाता है | लेकिन Anonymous या lambda Function का कोई नाम नहीं होता है |
Normal Function के लिए 'def' keyword का इस्तेमाल किया जाता है लेकिन Anonymous/Lambda Function के लिए 'lambda' keyword का इस्तेमाल किया जाता है | ये function काफी छोटा होता है |

Syntax

```
lambda arg1,arg2,..,argN : expression
```

arg1,arg2,..,argN : lambda function में एक या एक से ज्यादा arguments हो सकते है |
expression : lambda function में एक ही expression होता है | expression; return भी होता है |

Example पर 'anony' नाम के function object पर anonymous function को assign किया गया है | उसके बाद function object का इस्तेमाल function के रूप में call करके और argument पर values pass की गयी है |

Source Code :

```
anony = lambda a, b : a < b
print(anony(5, 4))
print(anony(4, 5))
```

Output :

```
False
True
```

Another Example for Anonymous/Lambda Function

Source Code :

```
anony = lambda a, b : print(a < b)
anony(5, 4)
anony(4, 5)
```

Output :

```
False
True
```

Another Example for Anonymous/Lambda Function

Source Code :

```
anony = lambda a, b : print(a < b) Aanonymous Function
#same as
def anony(a, b):     #Named/Normal Function
    return a < b
```

Lists

Python में 'List' ये data structure होता है | List ये Elements का sequence होता है और mutable(changeable) होता है |
Python के list में जो elements होते है उसे 'items' कहते है |
List के हर item को comma(,) से separate किया जाता है और पूरे items को square brackets([]) के अन्दर close किया जाता है | List में mixed data types भी इस्तेमाल किये जा सकते है |

Creating List in Python

List के आखिरी में semi-colon(;) दे या ना दे इससे कोई फर्क नहीं पड़ता है |

Source Code :

```
list1 = [1, 2, 3, 4, 5] #Integer List
list2 = [1.5, 2.4, 3.8, 4.4, 5.7]; #Float List
list3 = [1, 2, "Hello", 4.5]; #Mixed Data Types List
list4 = [1, 2, [1, 2], 4.5] #Nested List
```

Positive and Negative Indexing

List Positive Indexing

हर Programming Language में index की शुरुआत '0' से होती है उसी तरह से Python में List के पहले item का index '0' होता है |

Source Code :

```
strList = ["H", "e", "l", "l", "o"]
```

strList

ItemsNumber	item1	item2	item3	item4	item5
Index	0	1	2	3	4
Items	"H"	"e"	""l"	"l"	""o"

List Negative Indexing

Python में List के आखिरी index '-1' से शुरू होता है |

Source Code :

```
strList = ["H", "e", "l", "l", "o"]
```

strList

ItemsNumber	item1	item2	item3	item4	item5
Index	-5	-4	-3	-2	-1
Items	"H"	"e"	""l"	"l"	""o"

Accessing List

index से List के items को access किया जा सकता है |

Syntax

```
list[index]
```

Source Code :

```
list = [1, 2, "H", 2.8]
print(list[0])
print(list[1])
print(list[2])
print(list[3])
```

Output :

```
1
2
H
2.8
```

Invalid Indexing

अगर invalid index दिया जाता है तो 'indexError' का exception आ जाता है |

Source Code :

```
list = [1, 2, "H", 2.8]
print(list[10])
```

Output :

```
print(list[10])
IndexError: list index out of range
```

Accessing Nested List

Source Code :

```
list = [1, 2, "Hello", ["R", "U"]]
print(list[3][0]) #Output : R
print(list[3][1]) #Output : U
```

Convert List to Sub-List or List Slicing

colon(:) के left में sub-List कहा से start करना है और colon(:) के right में कहा पर end करना है वो दिया जाता है | Slicing के लिए colon को 'slicing operator' कहा जाता है |

Syntax

```
list[start:end]
```

Source Code :

```
list = [1, 2, "H", 2.8]
print(list[0:1]) # [1]
print(list[0:2]) # [1, 2]
print(list[1:3]) # [2, 'H', 2.8]
print(list[3:1]) #[]
```

Output :

```
[1]
[1, 2]
[2, 'H', 2.8]
[]
```

अगर colon(:) के left side का index invalid होता है तो blank list([]) return होती है और colon(:) के right side का index invalid होता है तो List के left index से आखिरी तक index; return होता है | अगर दोनों ही invalid index होता है तो blank list([]) return होता है |

For Example,

```
list = [1, 2, "H", 2.8]
print(list[0:1]) # [1]
print(list[0:2]) # [1, 2]
print(list[1:3]) # [2, 'H']
print(list[1:15]) # [2, 'H', 2.8]
print(list[10:4]) # []
```

Check Length of List's Items

Source Code :

```
list = [1, 2, "H", 2.8]
print (len(list))

#Output : 4
```

List Concatenation

Source Code :

```
list = [1, 2, "H", 2.8] + ["Hello", [2, 8]]
print (list)

#Output : [1, 2, 'H', 2.8, 'Hello', [2, 8]]
```

List Repetition

Source Code :

```
list = [1, 2, "H", 2.8] * 3
print (list)

#Output : [1, 2, 'H', 2.8, 1, 2, 'H', 2.8, 1, 2, 'H', 2.8]
```

Iterating List Items using For Loop

Source Code :

```
list = [1, 2, "H", 2.8]
for i in list:
    print(i)

#Output :
1
2
H
2.8
```

Add Item(s) in List

List में item(s) को add करने के लिए functions की जरुरत पड़ती है | List में एक item add करने के लिए 'append()' function का इस्तेमाल किया जाता है और एक से ज्यादा items add करने के लिए 'extend()' function का इस्तेमाल किया जाता है |

Source Code :

```
list = [1, 2, "H", 2.8]
list.append("E")
print(list)

#Output : [1, 2, 'H', 2.8, 'E']
```

Source Code :

```
list = ["P", "y", "t"]
list.extend(["h", "o", "n"])
print(list)

#Output : ['P', 'y', 't', 'h', 'o', 'n']
```

Change Items in List

Assignment Operator और index की मदद से List की values को change किया जा सकता है |

Source Code :

```
list = ['P', 'y', 't', 'h', 'o', 'n']
list[2] = "p"
list[0] = "C"
print(list)

#Output : ['C', 'y', 'p', 'h', 'o', 'n']
```

Deleting Items in List

List के item को delete करने के लिए 'del' operator का इस्तेमाल किया जाता है |

Source Code :

```
list = ['P', 'y', 't', 'h', 'o', 'n']
del list[2]
del list[0]
print(list)

#Output : ['y', 'h', 'o', 'n']
```

List Functions in Python

List Functions	Description
len()	दिए गए list की length; number में return करता है	
max()	दिए गए list से max value को return करता है	
min()	दिए गए list से min value को return करता है	
list()	sequence को list में convert करता है	

len() List Function

len() function का इस्तेमाल list की length को return करने के लिए किया जाता है |

Syntax

```
len()
```

Parameter :

list : जिस list की length check करनी है वो list यहाँ पर दी जाती है |

Returning Value

len() function list की length को number में return करता है |

Source Code :

```
list = ["H", "e", "l", "l", "o"]
print(len(list))
```

Output :

```
5
```

max() List Function

max() function का इस्तेमाल list से max value को return करने के लिए किया जाता है |

Syntax

```
max()
```

Parameter :

max() function के लिए कोई parameter नहीं होता है |

Returning Value

max() function list से max value return करता है |
max() और min() function ascii value के हिसाब से item को return करता है |

Source Code :

```
list = ["H", "e", "l", "l", "o", " ", "W", "o", "r", "l", "d"]
print(max(list))
```

Output :

```
r
```

min() List Function

min() function का इस्तेमाल list से min value को return करने के लिए किया जाता है |

Syntax

```
min()
```

Parameter :

min() function के लिए कोई parameter नहीं होता है |

Returning Value

min() function list से min value return करता है |
max() और min() function ascii value के हिसाब से item को return करता है |

Source Code :

```
list = ["H", "e", "l", "l", "o", " ", "W", "o", "r", "l", "d"]
print(min(list))
```

Output :

```

```

list() List Function

list() function; sequence और collection को list में convert करता है |

Syntax

```
list(sequence)
```

Parameter :

sequence : यहाँ पर list में convert करने के लिए sequence दिया जाता है |

Returning Value

list() function; sequence और collection को list में convert करके list को return करता है |

Source Code :

```
tuple = ("H", "e", "l", "l", "o")
set = {"H", "e", "l", "l", "o"}
str = "Hello World"
print(list(tuple))
print(list(set))
print(list(str))
```

Output :

```
['H', 'e', 'l', 'l', 'o']
['H', 'o', 'e', 'l']
['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd']
```

List Methods in Python

List Methods	**Description**
append()	list पर एक ही item जोड़ने के लिए किया जाता है	
clear()	list को clear करता है	
copy()	list की copy बनाता है	
count()	दिए हुए item के list में आये occurrences return करता है	
extend()	दिए गए list पर sequence को जोड़ता है	
index()	दिए गए item की index return करता है	
insert()	दिए गए valid index पर item को insert करता है	
pop()	दिए गए index का item return किया जाता है	
remove()	दिए गए item को remove करता है	
reverse()	दिए गए list को reverse करता है	
sort()	दिए गए sequence को sort करके return करता है	

append() List Method

append() method का इस्तेमाल List पर एक ही item जोड़ने के लिए किया जाता है |

Syntax

```
list.append(item)
```

Parameter :

item : list पर कौनसे item को जोड़ना है वो item यहाँ पर दिया जाता है |

Returning Value

append() method 'None' return करता है |

Source Code :

```
list = ["H", "e", "l", "l"]
list.append("o")
print(list)
```

Output :

```
['H', 'e', 'l', 'l', 'o']
```

clear() List Method

clear() method का इस्तेमाल List को clear करने के लिए किया जाता है |

Syntax

```
list.clear()
```

Parameter :

clear() method के लिए कोई parameter नहीं होता है |

Returning Value

clear() method 'None' return करता है |

Source Code :

```
list = ["H", "e", "l", "l"]
list.clear()
print(list)
```

Output :

```
[]
```

copy() List Method

copy() method का इस्तेमाल List को copy करने के लिए किया जाता है |

Syntax

```
list.copy()
```

Parameter :

copy() method के लिए कोई parameter नहीं होता है |

Returning Value

copy() method copy हुए list को return करता है |

Source Code :

```
list1 = ["H", "e", "l", "l", "o"]
list2 = list1.copy()
print(list1)
print(list2)
```

Output :

```
['H', 'e', 'l', 'l', 'o']
['H', 'e', 'l', 'l', 'o']
```

count() List Method

count() method का इस्तेमाल दिए हुए item के list में आये occurrences return करने के लिए किया जाता है |

Syntax

```
list.count(item)
```

Parameter :

item : जिस item के occurrences को number में return करना है वो item यहाँ पर दिया जाता है |

Returning Value

count() method दिए हुए item के list में आये occurrences; number में return करता है |

Source Code :

```
list = ["H", "e", "l", "l", "o"]

print(list.count("l"))
```

Output :

```
2
```

extend() List Method

extend() method का इस्तेमाल list को extend करने के लिए किया जाता है |

Syntax

```
list.extend(sequence)
```

Parameter :

sequence : जिस sequence और collection को मुख्य list के साथ extend करना है वो sequence यहाँ पर दी जाती है |

Returning Value

extend() method 'None' return करता है |

Source Code :

```
list = ["H", "e", "l", "l", "o"]
list.extend(["W", "o", "r", "l", "d"])
print(list)
```

Output :

```
['H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd']
```

index() List Method

index() method में दिए गए item की index return करता है |

Syntax

```
list.index(item)
```

Parameter :

item : जिस item की index return करनी है वो item यहाँ पर दिया जाता है |

Returning Value

index() method में जो item दिया जाता है उसकी index return की जाती है |

Source Code :

```
list = ["H", "e", "l", "l", "o"]
print(list.index("e"))
print(list.index("l"))
print(list.index("l"))
```

Output :

```
1
2
2
```

insert() List Method

insert() method का इस्तेमाल दिए गए list पर valid index पर नया item जोड़ने के लिए किया जाता है |

Syntax

```
list.insert(index, item)
```

Parameter :

index : जिस index पर item को जोड़ना है वो index यहाँ पर दिया जाता है |

item : जो item; index पर जोड़ना है वो item यहाँ पर दिया जाता है |

Returning Value

insert() method 'None' return करता है |

Source Code :

```
list = [1, 2, 3, 4, 5]
print("Original List : ",list)
list.insert(2, "H")
print("After insert() : ",list)
```

Output :

```
Original List :  [1, 2, 3, 4, 5]
After insert() :  [1, 2, 'H', 3, 4, 5]
```

pop() List Method

pop() method में दिए गए index का item return किया जाता है |
pop() method का इस्तेमाल दिए गए index को remove करके बचे हुए list को print करने के लिए किया जाता है |

Syntax

```
list.pop(index)
```

Parameter :

index : Optional. जिस item को return करना है उसकी index दी जाती है | अगर दिया नहीं जाता है तो default value 'len(list)-1' होता है |

Returning Value

pop() method में दिए गए index का item return किया जाता है |

Source Code :

```
list = ["H", "e", "l", "l", "o"]
print(list.pop())
print(list.pop(2))
print(list.pop(1))
```

Output :

```
o
l
e
```

remove() List Method

remove() method का इस्तेमाल list से item remove करने के लिए किया जाता है |

Syntax

```
list.remove(item)
```

Parameter :

item : जिस item को remove करना है है वो item यहाँ पर दिया जाता है |

Returning Value

remove() method 'None' return करता है |

Source Code :

```
list = ["H", "e", "l", "l", "o"]
list.remove("l")
print(list)
list.remove("l")
print(list)
list.remove("o")
print(list)
```

Output :

```
['H', 'e', 'l', 'o']
['H', 'e', 'o']
['H', 'e']
```

reverse() List Method

reverse() method का इस्तेमाल दिए गए list को reverse करने के लिए किया जाता है |

Syntax

```
list.reverse()
```

Parameter :

reverse() method के लिए कोई parameter नहीं होता है |

Returning Value

reverse() method 'None' return करता है |

Source Code :

```
list = ["H", "e", "l", "l", "o"]
list.reverse()
print(list)
```

Output :

```
['o', 'l', 'l', 'e', 'H']
```

sort() List Method

sort() method का दिए गए sequence को sort करके return करता है |

Syntax

```
list.sort(key, reverse)
```

Parameter :

key : Optional. किस तरह से sort करना है यहाँ पर वो function आता है |

reverse : Optional. अगर reverse True दिया जाता है तो ascending order का sequence; descending order में reverse होता है |

Returning Value

sort() method 'None' return करता है |

sorted() function और sort() method एक जैसे ही होते है लेकिन sorted() function; sort किया हुआ sequence return करता है और sort() method 'None' return करता है |

Source Code :

```
list =  [10, 5, 9, 7, 15]
list.sort()
print(list)
```

Output :

```
[5, 7, 9, 10, 15]
```

Using 'key' Parameter in sort() method in Python

Source Code :

```
list = [[5,10,4], [25,8,10], [10,9,1], [6,7,65]]
def FirstAsc(i):
    return i[0]
list.sort(key=FirstAsc)
print(list)
def SecondAsc(i):
    return i[1]
list.sort(key=SecondAsc)
print(list)
def ThirdAsc(i):
    return i[2]
list.sort(key=ThirdAsc)
print(list)
```

Output :

```
[[5, 10, 4], [6, 7, 65], [10, 9, 1], [25, 8, 10]]
[[6, 7, 65], [25, 8, 10], [10, 9, 1], [5, 10, 4]]
[[10, 9, 1], [5, 10, 4], [25, 8, 10], [6, 7, 65]]
```

Using 'reverse' Parameter in sorted() Function

अगर True दिया जाता है तो sequence को descending order में sort किया जाता है |

Source Code :

```
list =  [1, 2, 3, 4, 5]
list.sort()
print("Ascending Order :", list)
list.sort(reverse=True)
print("Descending Order :", list)
```

Output :

```
Ascending Order : [1, 2, 3, 4, 5]
Descending Order : [5, 4, 3, 2, 1]
```

Tuple

Introduction

Python में 'Tuple' ये data structure होता है | Tuple ये Elements का sequence होता है और immutable(unchangeable) होता है |
Python में List और Tuple एक जैसे ही होते है लेकिन List के items को change या उसके items delete किये जा सकते है और Tuple के items को change या उसके items delete नहीं किये जा सकते है | Tuples सिर्फ read किये जा सकते है |
Python के tuple में जो elements होते है उसे 'items' कहते है |
Tuple के हर item को comma(,) से separate किया जाता है और पूरे items को parenthesis(()) के अन्दर close किया जाता है | Tuple में mixed data types भी इस्तेमाल किये जा सकते है |

Creating Tuple in Python

Tuple के आखिरी में semi-colon(;) दे या ना दे इससे कोई फर्क नहीं पड़ता है |

Source Code :

```
tuple1 = (1, 2, 3, 4, 5) #Integer Tuple
tuple2 = (1.5, 2.4, 3.8, 4.4, 5.7); #Float Tuple
tuple3 = (1, 2, "Hello", 4.5); #Mixed Data Types Tuple
tuple4 = (1, 2, (1, 2), 4.5) #Nested Tuple
```

Tuple Without Parenthesis(())

Tuples बिना parenthesis के भी हो सकते है |

Source Code :

```
tuple = "H", "e", "l", "l", "o"
print(type(tuple))

#Output : <class 'tuple'>
```

Tuple Positive Indexing

हर Programming Language में index की शुरुआत '0' से होती है उसी तरह से Python में Tuple के पहले item का index '0' होता है |

Source Code :

```
strTuple = ("H", "e", "l", "l", "o")
```

strTuple

ItemsNumber	item1	item2	item3	item4	item5
Index	0	1	2	3	4
Items	"H"	"e"	"l"	"l"	"o"

Tuple Negative Indexing

Python में Tuple के आखिरी index '-1' से शुरू होता है |

Source Code :

```
strTuple = ("H", "e", "l", "l", "o")
```

strTuple

ItemsNumber	item1	item2	item3	item4	item5
Index	-5	-4	-3	-2	-1
Items	"H"	"e"	"l"	"l"	"o"

Accesing Tuple

index से Tuple के items को access किया जा सकता है |

Syntax

```
tuple(index)
```

Source Code :

```
tuple = (1, 2, "H", 2.8)
print(tuple[0])
print(tuple[1])
print(tuple[2])
print(tuple[3])
```

Output :

```
1
2
H
2.8
```

Invalid Indexing

अगर invalid index दिया जाता है तो 'indexError' का exception आ जाता है |

Source Code :

```
tuple = (1, 2, "H", 2.8)
print(tuple[10])
```

Output :

```
    print(tuple(10))
IndexError: tuple index out of range
```

Accessing Nested Tuple

Source Code :

```
tuple = (1, 2, "Hello", ("R", "U"))

print(tuple[3][0]) #Output : R
print(tuple[3][1]) #Output : U
```

Tuples Cannot Change But Reassign

tuple को change नहीं कर सकते लेकिन उसे re-assign किया जा सकता है |

Source Code :

```
tuple = (1, 2, 3, 4, 5)
print(tuple)
#Output : (1, 2, 3, 4, 5)

tuple = (6, 7, 8, 9, 10)
print(tuple)
#Output : (6, 7, 8, 9, 10)
```

Deleting Tuple

'del' Operator से पूरा tuple delete किया जा सकता है लेकिन tuple के एक-एक item को delete नहीं किया जा सकता |

Source Code :

```
tuple = (1, 2, 3, 4, 5)
del tuple[0]
#Output :
#    del tuple[0]
#TypeError: 'tuple' object doesn't support item deletion
```

```
tuple = (1, 2, 3, 4, 5)
del tuple
print(tuple)
#Output : <class 'tuple'>
```

Convert Tuple to Sub-Tuple or Tuple Slicing

colon(:) के left में sub-Tuple कहा से start करना है और colon(:) के right में कहा पर end करना है वो दिया जाता है | Slicing के लिए colon को 'slicing operator' कहा जाता है |

Syntax

```
tuple(start:end)
```

Source Code :

```
tuple = (1, 2, "H", 2.8)
print(tuple[0:1]) # (1)
print(tuple[0:2]) # (1, 2)
print(tuple[1:3]) # (2, 'H', 2.8)
print(tuple[3:1]) #()
```

Output :

```
(1)
(1, 2)
(2, 'H', 2.8)
()
```

अगर colon(:) के left side का index invalid होता है तो blank tuple(()) return होती है और colon(:) के right side का index invalid होता है तो Tuple के left index से आखिरी तक index; return होता है | अगर दोनों ही invalid index होता है तो blank tuple(()) return होता है |

For Example

```
tuple = (1, 2, "H", 2.8)
print(tuple[0:1]) # (1)
print(tuple[0:2]) # (1, 2)
print(tuple[1:3]) # (2, 'H')
print(tuple[1:15]) # (2, 'H', 2.8)
print(tuple[10:4]) # ()
```

Check Length of Tuple's Items

```
tuple = (1, 2, "H", 2.8)
print (len(tuple))

#Output : 4
```

Tuple Concatenation

```
tuple = (1, 2, "H", 2.8) + ("Hello", (2, 8))
print (tuple)

#Output : (1, 2, 'H', 2.8, 'Hello', (2, 8))
```

Tuple Repetition

```
tuple = (1, 2, "H", 2.8) * 3
print (tuple)

#Output : (1, 2, 'H', 2.8, 1, 2, 'H', 2.8, 1, 2, 'H', 2.8)
```

Iterating Tuple Items using For Loop

```
tuple = (1, 2, "H", 2.8)
for i in tuple:
    print(i)

#Output :
#1
#2
#H
#2.8
```

Tuple Functions in Python

Tuple Functions	**Description**
all()	sequence के सभी items True होते है तो ये तो ये True return करता है	
any()	sequence का एक या सभी items True होते है तो ये तो ये True return करता है	
enumerate()	दिए गए start से index और उसकी value की pair return करता है	
len()	दिए गए tuple की length; number में return करता है	
max()	दिए गए tuple से max value को return करता है	
min()	दिए गए tuple से min value को return करता है	
sorted()	दिए गए sequence को sort करके return करता है	
sum()	दिए गए sequence के items को add करके उनका sum return करता है	
tuple()	sequence को tuple में convert करता है	

all() Tuple Function

all() function में sequence के सभी items True होते है तो ये तो ये True return करता है |

Syntax

```
all(seq)
```

Parameter :

seq : यहाँ पर sequence दिया जाता है |

अगर sequence के सभी items True होते है तो ये True return करता है |
अगर sequence का एक ही item True और बाकी items False होते है तो ये False return करता है |
अगर sequence का एक ही item False और बाकी items True होते है तो ये False return करता है |
अगर sequence के सभी items False होते है तो ये False return करता है |
अगर sequence empty होता है तो ये True return करता है |

Returning Value

all() function tuple के true और False items के हिसाब से boolean value return करता है |

Source Code :

```
tuple1 = (1, 2, 3, 4, 5)
print(all(tuple1)) #True
tuple2 = (1, 3, tuple1[0]==tuple1[1])
print(all(tuple2)) #False
tuple3 = (False, True, False)
print(all(tuple3)) #False
empTup = ()
print(all(empTup)) #True
```

Output :

```
True
False
False
True
```

any() Tuple Function

any() function में sequence का एक या सभी items True होते है तो ये तो ये True return करता है |

Syntax

```
any(seq)
```

Parameter :

seq : यहाँ पर sequence दिया जाता है |

अगर sequence के सभी items True होते है तो ये True return करता है |
अगर sequence का एक ही item True और बाकी items False होते है तो ये True return करता है |
अगर sequence का एक ही item False और बाकी items True होते है तो ये True return करता है |
अगर sequence के सभी items False होते है तो ये False return करता है |
अगर sequence empty होता है तो ये False return करता है |

Returning Value

any() function tuple के true और False items के हिसाब से boolean value return करता है |

Source Code :

```
tuple1 = (1, 2, 3, 4, 5)
print(any(tuple1)) #True
tuple2 = (1, 3, tuple1[0]==tuple1[1])
print(any(tuple2)) #True
tuple3 = (False, True, False)
print(any(tuple3)) #True
tuple4 = (False, False, False)
print(any(tuple4)) #False
empTup = ()
print(any(empTup)) #False
```

Output :

```
True
True
True
False
False
```

enumerate() Tuple Function

enumerate() function; दिए गए start से index और उसकी value की pair return करता है |

Syntax

```
enumerate(seq, start)
```

Parameter :

seq : यहाँ पर sequence दिया जाता है |

start : Optional. जिस index से start करना है वो index यहाँ पर दी जाती है | For eg., अगर 2 ये index दी जाती है तो index 2 होगा और sequence के पहली value की pair होगी | अगर दिया नहीं जाता तो default '0' होता है |

Returning Value

enumerate() function दिए गए start से index और उसकी value की pair enumerate object में return करता है |

Source Code :

```
set = (1, 2, 3, 4, 5)
print(tuple(enumerate(set)))
set = ("H", "e", "l", "l", "o")
print(tuple(enumerate(set)))
print(tuple(enumerate(set, 2)))
```

Output :

```
((0, 1), (1, 2), (2, 3), (3, 4), (4, 5))
((0, 'H'), (1, 'e'), (2, 'l'), (3, 'l'), (4, 'o'))
((2, 'H'), (3, 'e'), (4, 'l'), (5, 'l'), (6, 'o'))
```

len() Tuple Function

len() function का इस्तेमाल tuple की length को return करने के लिए किया जाता है |

Syntax

```
len(tuple)
```

Parameter :

tuple : जिस tuple की length check करनी है वो tuple यहाँ पर दी जाती है |

Returning Value

len() function tuple की length को number में return करता है |

Source Code :

```
tuple = ("H", "e", "l", "l", "o")
print(len(tuple))
```

Output :

```
5
```

max() Tuple Function

max() function का इस्तेमाल tuple से max value को return करने के लिए किया जाता है |

Syntax

```
max()
```

Parameter :

max() function के लिए कोई parameter नहीं होता है |

Returning Value

max() function tuple से max value return करता है |
max() और min() function ascii value के हिसाब से item को return करता है |

Source Code :

```
tuple = ["H", "e", "l", "l", "o", " ", "W", "o", "r", "l", "d"]
print(max(tuple))
```

Output :

```
r
```

min() Tuple Function

min() function का इस्तेमाल tuple से min value को return करने के लिए किया जाता है |

Syntax

```
min()
```

Parameter :

min() function के लिए कोई parameter नहीं होता है |

Returning Value

min() function tuple से min value return करता है |
max() और min() function ascii value के हिसाब से item को return करता है |

Source Code :

```
tuple = ("H", "e", "l", "l", "o", " ", "W", "o", "r", "l", "d")
print(min(tuple))
```

Output :

```
```

sorted() Tuple Function

sorted() function; में दिए गए sequence को sort करके return करता है |

Syntax

```
sorted(seq, key, rev)
```

Parameter :

seq : यहाँ पर sequence(list, tuple, string) या collection(dictionary, set) दिया जाता है |

key : Optional. किस तरह से sort करना है यहाँ पर वो function आता है |

reverse : Optional. अगर reverse True दिया जाता है तो ascending order का sequence; descending order में reverse होता है |

Returning Value

sorted() function में दिए गए sequence को sort करके list में return किया जाता है |

Source Code :

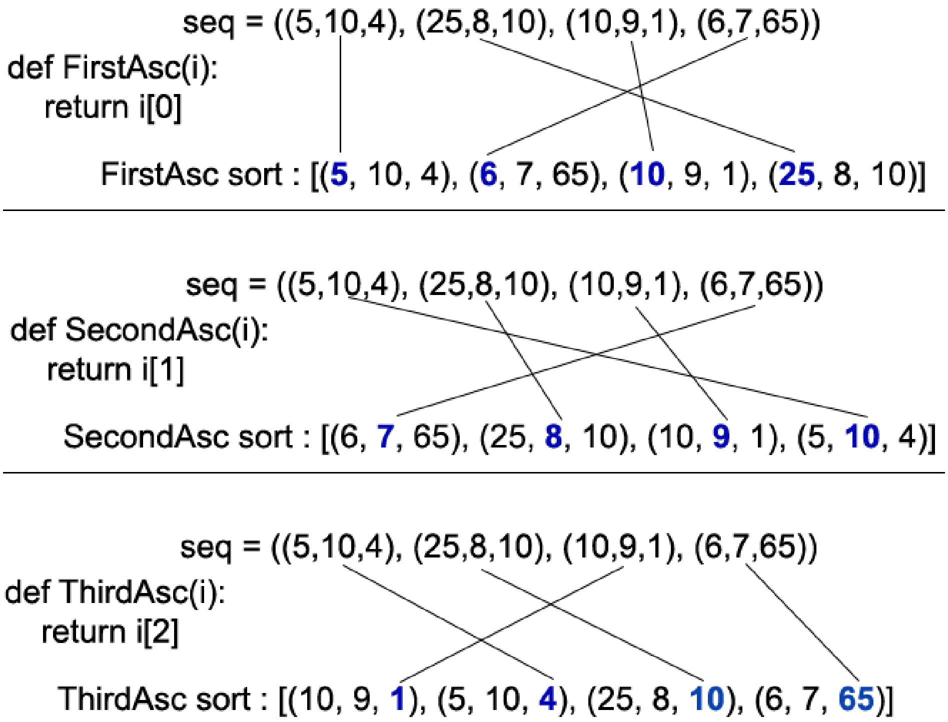
sorted() function; ascending order में default sort करता है |

```
tuple = (5, 8, 9, 7, 5)
print(sorted(tuple))
tuple = ("P", "y", "t", "h", "o", "n") #'P' in Uppercase
print(sorted(tuple))
tuple = ("p", "y", "t", "h", "o", "n") #'p' in lowercase
print(sorted(tuple))
```

Output :

```
[5, 5, 7, 8, 9]
['P', 'h', 'n', 'o', 't', 'y']
['h', 'n', 'o', 'p', 't', 'y']
```

Using 'key' Parameter in sorted() Function



Source Code :

sorted() function; ascending order में default sort करता है |

```
seq = ((5,10,4), (25,8,10), (10,9,1), (6,7,65))
def FirstAsc(i):
    return i[0]
print("FirstAsc sort :",sorted(seq, key=FirstAsc))
def SecondAsc(i):
    return i[1]
print("SecondAsc sort :",sorted(seq, key=SecondAsc))
def ThirdAsc(i):
    return i[2]
print("ThirdAsc sort :",sorted(seq, key=ThirdAsc))
```

Output :

```
FirstAsc sort : [(5, 10, 4), (6, 7, 65), (10, 9, 1), (25, 8, 10)]
SecondAsc sort : [(6, 7, 65), (25, 8, 10), (10, 9, 1), (5, 10, 4)]
ThirdAsc sort : [(10, 9, 1), (5, 10, 4), (25, 8, 10), (6, 7, 65)]
```

Using 'reverse' Parameter in sorted() Function

अगर True दिया जाता है तो sequence को descending order में sort किया जाता है |

Source Code :

```
seq =  (1, 2, 3, 4, 5)
print("Ascending Order :", sorted(seq))
print("Descending Order :", sorted(seq, reverse=True))
```

Output :

```
Ascending Order : [1, 2, 3, 4, 5]
Descending Order : [5, 4, 3, 2, 1]
```

sum() Tuple Function

sum() function; में दिए गए sequence या collection के items को add करके उनका sum return करता है |

Syntax

```
sum(seq, startAdd)
```

Parameter :

seq : यहाँ पर sequence या collection दिया जाता है |

startAdd : Optional. यहाँ पर जो संख्या दी जाती है वो sequence या collection का पहला item होता है | अगर दिया नहीं जाता है तो default '0' होता है |

Returning Value

sum() function में दिए गए sequence या collection के items का sum return किया जाता है |

Source Code :

```
seq = [-5, 6, 1]
print(sum(seq))
print(sum(seq, 5))
```

Output :

```
2
7
```

Using 'key' Parameter in sum() Function

Source Code :

```
seq = ((5,10,4), (25,8,10), (10,9,1), (6,7,65))
def FirstAsc(i):
    return i[0]
print("FirstAsc sort :",sum(seq, key=FirstAsc))
def SecondAsc(i):
    return i[1]
print("SecondAsc sort :",sum(seq, key=SecondAsc))
def ThirdAsc(i):
    return i[2]
print("ThirdAsc sort :",sum(seq, key=ThirdAsc))
```

Output :

```
FirstAsc sort : [(5, 10, 4), (6, 7, 65), (10, 9, 1), (25, 8, 10)]
SecondAsc sort : [(6, 7, 65), (25, 8, 10), (10, 9, 1), (5, 10, 4)]
ThirdAsc sort : [(10, 9, 1), (5, 10, 4), (25, 8, 10), (6, 7, 65)]
```

Using 'reverse' Parameter in sum() Function

अगर True दिया जाता है तो sequence को descending order में sort किया जाता है |

Source Code :

```
seq =  (1, 2, 3, 4, 5)
print("Ascending Order :", sum(seq))
print("Descending Order :", sum(seq, reverse=True))
```

Output :

```
Ascending Order : [1, 2, 3, 4, 5]
Descending Order : [5, 4, 3, 2, 1]
```

tuple() Tuple Function

tuple() function; sequence(list, dictionary, set, string) को tuple में convert करता है |

Syntax

```
tuple(sequence)
```

Parameter :

sequence : यहाँ पर tuple में convert करने के लिए sequence दिया जाता है |

Returning Value

tuple() function; sequence को tuple में convert करके tuple को return करता है |

Source Code :

```
list = ["H", "e", "l", "l", "o"]
set = {"H", "e", "l", "l", "o"}
str = "Hello World"
print(tuple(list))
print(tuple(set))
print(tuple(str))
```

Output :

```
('H', 'e', 'l', 'l', 'o')
('H', 'o', 'e', 'l')
('H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd')
```

Dictionary

Introduction

जैसे PHP और Javascript में Asscociative Array का इस्तेमाल किया जाता है वैसे ही Python में Dictionary इस अलग नाम से इस्तेमाल किया जाता है |
Dictionary; hashes की तरह होती है | Dictionary में key और value की pairs होती है |
Dictionary में key और value को colon(:) से separate किया जाता है और key और value की pair को comma से separate किया जाता है |
Dictionary; mutable(changeable) होता है लेकिन Dictionary में values change हो सकती है और method(pop()) के जरिये keys भी change की जा सकती है लेकिन duplicate key नहीं दी जा सकती |

Creating Dictionary

Dictionary की keys और values को curly braces({}) के अन्दर लिखा जाता है |

Source Code :

```
dict1 = {"name1":"Rakesh", "name2":"Ramesh", "name3":"Kamalesh"}

dict2 = {}
```

Accessing Value in Dictionary in Python

List और Tuple में value को access करने के लिए उनकी 'index' का इस्तेमाल किया जाता है | वैसे ही dictionary की values को access करना हो तो उनकी square brackets([]) में 'keys' का इस्तेमाल किया जाता है |

Source Code :

```
dict = {"name1":"Rakesh", "name2":"Ramesh", "name3":"Kamalesh"}
print(dict["name1"])
print(dict["name2"])
print(dict["name3"])
```

Output :

```
Rakesh
Ramesh
Kamalesh
```

अगर invalid key इस्तेमाल की जाती है तो, 'keyError' exception आ जाता है |

Source Code :

```
dict = {"name1":"Rakesh", "name2":"Ramesh", "name3":"Kamalesh"}
print(dict["name4"])
```

Output :

```
    print(dict["name4"])
KeyError: 'name4'
```

Change Dictionary Values in Python

Assignment Operator और square brackets([]) में keys की मदद से dictionary की values को change किया जा सकता है |

Source Code :

```
dict = {"name1":"Rakesh", "name2":"Ramesh", "name3":"Kamalesh"}
dict["name1"] = "Nagesh"
print(dict["name1"])
print(dict)
```

Output :

```
Nagesh
{'name1': 'Nagesh', 'name2': 'Ramesh', 'name3': 'Kamalesh'}
```

Change Dictionary Keys in Python

pop() method से dictionary की keys को change किया जा सकता है |
pop() method से 'name1' key को remove करके दूसरी जगह पर 'name4' इस key को add किया गया है |

Source Code :

```
dict = {"name1":"Rakesh", "name2":"Ramesh", "name3":"Kamalesh"}
print("Before Changing Key :")
print(dict)
dict["name4"] = dict.pop("name1")
print("After Changing Key :")
print(dict)
```

Output :

```
Before Changing Key :
{'name1': 'Rakesh', 'name2': 'Ramesh', 'name3': 'Kamalesh'}
After Changing Key :
{'name2': 'Ramesh', 'name3': 'Kamalesh', 'name4': 'Rakesh'}
```

Add Dictionary Element in Python

Dictionary में element add करने के लिए square bracket([]) में unique key name और उसकी value दी जाती है |

Source Code :

```
dict = {"name1":"Rakesh", "name2":"Ramesh", "name3":"Kamlesh"}
print("Before Adding Element :")
print(dict)
dict["name4"] = "Rajesh"
print("After Adding Element :")
print(dict)
```

Output :

```
Before Adding Element :
{'name1': 'Rakesh', 'name2': 'Ramesh', 'name3': 'Kamlesh'}
After Adding Element :
{'name1': 'Rakesh', 'name2': 'Ramesh', 'name3': 'Kamlesh', 'name4': 'Rajesh'}
```

Delete Dictionary Element in Python

Dictionary में element को delete करने के लिए 'del' Operator का इस्तेमाल किया जाता है |

Source Code :

```
dict = {"name1":"Rakesh", "name2":"Ramesh", "name3":"Kamlesh"}
print("Before Deleting Element :")
print(dict)
del dict["name2"]
print("After Deleting Element :")
print(dict)
```

Output :

```
Before Deleting Element :
{'name1': 'Rakesh', 'name2': 'Ramesh', 'name3': 'Kamlesh'}
After Deleting Element :
{'name1': 'Rakesh', 'name3': 'Kamlesh'}
```

Dictionary Functions in Python

Functions	Description
all()	sequence के सभी elements True होते है तो ये तो ये True return करता है |
any()	sequence का एक या सभी elements True होते है तो ये तो ये True return करता है |
dict()	इसका इस्तेम्मल नए ढंग से dictionary को create करने के लिए किया जाता है |
len()	dictionary की length को return करता है |
sorted()	दिए गए sequence को sort करके return करता है |
str()	dictionary को string में convert करके return करता है |

all() Dictionary Function

all() function में sequence के सभी elements True होते है तो ये तो ये True return करता है |

Syntax

```
all(seq)
```

Parameter :

seq : यहाँ पर sequence दिया जाता है |

अगर sequence के सभी elements True होते है तो ये True return करता है |
अगर sequence का एक ही element True और बाकी elements False होते है तो ये False return करता है |
अगर sequence का एक ही element False और बाकी elements True होते है तो ये False return करता है |
अगर sequence के सभी elements False होते है तो ये False return करता है |
अगर sequence empty होता है तो ये True return करता है |

Returning Value

all() function dictionary के true और False elements के हिसाब से boolean value return करता है |

Source Code :

```
dict1 = {"one":1, "two":2, "three":3}
print(all(dict1)) #True
dict2 = {"one":1, "two":2, False:1}
print(all(dict2)) #False
dict3 = {False:1, True:2}
print(all(dict3)) #False
empDict = {}
print(all(empDict)) #True
```

Output :

```
True
False
False
True
```

any() Dictionary Function

any() function में sequence का एक या सभी elements True होते है तो ये तो ये True return करता है |

Syntax

```
any(seq)
```

Parameter :

seq : यहाँ पर sequence दिया जाता है |

अगर sequence के सभी elements True होते है तो ये True return करता है |
अगर sequence का एक ही element True और बाकी elements False होते है तो ये True return करता है |
अगर sequence का एक ही element False और बाकी elements True होते है तो ये True return करता है |
अगर sequence के सभी elements False होते है तो ये False return करता है |
अगर sequence empty होता है तो ये False return करता है |

Returning Value

any() function dictionary के true और False elements के हिसाब से boolean value return करता है |

Source Code :

```
dict1 = {"one":1, "two":2, "three":3}
print(any(dict1)) #True
dict2 = {"one":1, "two":2, False:1}
print(any(dict2)) #True
dict3 = {False:1, False:2}
print(any(dict3)) #False
dict4 = {False:1, True:2}
print(any(dict4)) #True
empDict = {}
print(all(empDict)) #True
```

Output :

```
True
True
False
True
True
```

dict() Dictionary Function

dict() function का इस्तेमाल dictionary को create करने के लिए किया जाता है |

Syntax1

```
dict({key1:value1, key2:value2,.., keyN:valueN})
```

Parameter :

{key1:value1, key2:value2,.., keyN:valueN} : यहाँ पर key और value को colon(:) से seperate किया जाता है | उनकी pair को comma(,) से seperate किया जाता है और पूरे dictionary को curly braces({}) के अन्दर लिखा जाता है |

Syntax2

```
dict([(key1,value1), (key2,value2),.., (keyN,valueN)])
```

Parameter :

[(key1,value1), (key2,value2),.., (keyN,valueN)] : यहाँ पर key और value को comma(,) से seperate किया जाता है | उनकी pair parenthesis(()) के अन्दर और comma(,) से seperate किया जाता है और पूरे dictionary को square bracket([]) के अन्दर लिखा जाता है |

Returning Value

dict() function dictionary को return करता है |

Source Code :

```
dict1 = dict({"one":1, "two":2, "three":3})
print(dict1)
#Output : {'one': 1, 'two': 2, 'three': 3}

dict2 = dict([("one",1), ("two",2), ("three",3)])
print(dict2)
#Output : {'one': 1, 'two': 2, 'three': 3}
```

Output :

```
{'one': 1, 'two': 2, 'three': 3}
{'one': 1, 'two': 2, 'three': 3}
```

len() Dictionary Function

len() function का इस्तेमाल dictionary की length को return करने के लिए किया जाता है |

Syntax

```
len(dict)
```

Parameter :

dict : जिस dictionary की length check करनी है वो dictionary यहाँ पर दी जाती है |

Returning Value

len() function dictionary की length को number में return करता है |

Source Code :

```
dict = {"one":1, "two":2, "three":3}
print(len(dict))
```

Output :

```
3
```

str() Dictionary Function

str() function का इस्तेमाल dictionary को string में convert करने के लिए किया जाता है |

Syntax

```
str(dict)
```

Parameter :

dict : जिस dictionary को string में convert करना है वो dictionary यहाँ पर दी जाती है |

Returning Value

str() function string में convert हुए dictionary को return किया जाता है |

Source Code :

```
dict = {"one":1, "two":2, "three":3}
print(str(dict))
```

Output :

```
{'one': 1, 'two': 2, 'three': 3}
```

Dictionary Methods in Python

Methods	Description
clear()	dictionary को clear करता है |
copy()	Dictionary को copy करके return करता है |
fromkeys()	दिए हुए sequence के item को dictionary के key के रूप में return करता है |
get()	दिए गए key की value को return करता है |
has_key()	दी हुई key; dictionary पर है या नहीं ये boolean value में return करता है |
items()	(key, value) इस नए प्रकार से dictionary को return करता है |
keys()	dictionary के सिर्फ keys को return करता है |
pop()	दिए गए key को remove करके उसकी value return की जाती है | अगर दी हुई key नहीं मिल जाती तो set की हुई default value return की जाती है |
popitem()	दिए गए dictionary के last item को remove करके (key, value) return करता है |
setdefault()	दी हुई key exist होती है तो उसकी value return की जाती है | अगर key नहीं होती है तो set की हुई default value को return किया जाता है |
update()	old dictionary को update करके update हुई dictionary को return करता है |
values()	dictionary के सिर्फ keys की values को return करता है |

clear() Dictionary Method

clear() method का इस्तेमाल dictionary को clear करने के लिए किया जाता है |

Syntax

```
dict.clear()
```

Parameter :

clear() method के लिए कोई parameter नहीं होता है |

Returning Value

clear() method 'None' return करता है |

Source Code :

```
dict = {"one":1, "two":2, "three":3}
dict.clear()
print(dict)

#Outout : {}
```

copy() Dictionary Method

copy() method का इस्तेमाल Dictionary को copy करने के लिए किया जाता है |

Syntax

```
dict.copy()
```

Parameter :

copy() method के लिए कोई parameter नहीं होता है |

Returning Value

copy() method copy हुए dictionary को return करता है |

Source Code :

```
dict1 = {"one":1, "two":2, "three":3}
dict2 = dict1.copy()
print(dict1)
print(dict2)
```

Output :

```
{'one': 1, 'two': 2, 'three': 3}
{'one': 1, 'two': 2, 'three': 3}
```

fromkeys() Dictionary Method

fromkeys() method दिए हुए sequence के item को dictionary के key के रूप में return करता है |

Syntax

```
dict.fromkeys(seq, val)
```

Parameter :

seq : यहाँ पर sequence(list, tuple, string) दिया जाता है | return होने वाले नए dictionary के लिए ये 'keys' होती है |

val : Optional. ये keys की values होती है | अगर दी नहीं जाती है तो 'None' return होता है |

Returning Value

fromkeys() method; नयी dictionary return करता है |

Example for fromkeys() Dictionary Method with 'sequence'

```
list = ["H", "e", "l", "l", "o"]
print(dict.fromkeys(list))
```

Output :

```
{'H': None, 'e': None, 'l': None, 'o': None}
```

Example for fromkeys() Dictionary Method with 'value'

```
list = ["H", "e", "l", "l", "o"]
print(dict.fromkeys(list, 2))
```

Output :

```
{'H': 2, 'e': 2, 'l': 2, 'o': 2}
```

get() Dictionary Method

get() method का दिए गए key की value को return करता है |

Syntax

```
dict.get(key, defaultVal)
```

Parameter :

key : यहाँ पर जिस key की value return करनी है वो key दी जाती है |

defaultVal : Optional. अगर 'key' parameter पर दी हुई key dictionary पर exist नहीं होती है तो यहाँ पर default value set की जाती है | अगर दिया नहीं जाता है तो Default 'None' होता है |

Returning Value

get() method में अगर पहले parameter में दी हुई key नहीं मिलती है तो दूसरे parameter पर set की हुई default value set की जाती है |
अगर get() method में दी हुई key नहीं मिलती है और default value set नहीं की जाती है तो 'None' return होता है |

Example for get() Dictionary Method using 'key' Parameter

```
dict = {"one":1, "two":2, "three":3}
print(dict.get("two"))
```

Output :

```
2
```

Example for get() Dictionary Method using 'default' Parameter

यहाँ पर 'four' नाम की key; dictionary में मौजूद नहीं है और default में '4' ये value set की गयी है |
अगर दी हुई key; get() method को नहीं मिलती है तो parameter पर set की हुई value return होती है |

```
dict = {"one":1, "two":2, "three":3}
print(dict.get("four", 4))
```

Output :

```
4
```

Example for get() Dictionary Method using 'default' Parameter

अगर दी हुई key; get() method को नहीं मिलती है और default value set नहीं की जाती है तो 'None' return होता है |

```
dict = {"one":1, "two":2, "three":3}
print(dict.get("four"))
```

Output :

```
None
```

has_key() Dictionary Method

has_key() method का इस्तेमाल दी हुई key; dictionary पर है या नहीं ये boolean value में return किया जाता है |

Syntax

```
dict.has_key(key)
```

Parameter :

key : यहाँ पर key दी जाती है |

Returning Value

has_key() method में दी हुई key; dictionary में होती है तो 'True' return होता है और अगर नहीं होती है तो 'False' return होता है |

has_key() method; Python 3.x versions पर support नहीं करता है |

Source Code :

```
dict = {"one":1, "two":2, "three":3}
print(dict.has_key("two"))
```

Output :

```
True
```

items() Dictionary Method

items() method में (key, value) इस नए प्रकार से dictionary को return करता है |

Syntax

```
dict.items()
```

Parameter :

items() method के लिए कोई parameter नहीं होता है |

Returning Value

items() method (key, value) इस नए format में dictionary को return करता है |

Source Code :

```
dict = {"one":1, "two":2, "three":3}
print(dict.items())
```

Output :

```
dict_items([('one', 1), ('two', 2), ('three', 3)])
```

keys() Dictionary Method

keys() method में dictionary के सिर्फ keys को return करता है |

Syntax

```
dict.keys()
```

Parameter :

keys() method के लिए कोई parameter नहीं होता है |

Returning Value

keys() method; dictionary के सिर्फ keys को return करता है |

Source Code :

```
dict = {"one":1, "two":2, "three":3}
print(dict.keys())
```

Output :

```
dict_keys(['one', 'two', 'three'])
```

pop() Dictionary Method

pop() method में दिए गए key को remove करके उसकी value return की जाती है | अगर दी हुई key नहीं मिल जाती तो set की हुई default value return की जाती है |

Syntax

```
dict.pop(key, default)
```

Parameter :

key : यहाँ पर dictionary से remove होनेवाली key दी जाती है |

default : अगर key नहीं मिलती तो यहाँ पर दी हुई default value return होती है |

Returning Value

pop() method दिए गए key को remove करके उसकी value return की जाती है | अगर दी हुई key नहीं मिल जाती तो set की हुई default value return की जाती है |
अगर key नहीं मिलती और default value भी नहीं set नहीं की जाती है तो 'keyError' exception आ जाता है |

Example for pop() Dictionary Method using 'key'

```
dict = {"one":1, "two":2, "three":3}
print("Before Removing item :")
print(dict)
dict.pop("one")
print("After Removing item :")
print(dict)
```

Output :

```
Before Removing item :
{'one': 1, 'two': 2, 'three': 3}
After Removing item :
{'two': 2, 'three': 3}
```

Example for pop() Dictionary Method using 'default'

```
dict = {"one":1, "two":2, "three":3}
print(dict)
print(dict.pop("four", 4))
```

Output :

```
{'one': 1, 'two': 2, 'three': 3}
4
```

Example for pop() Dictionary Method using 'Invalid key and without default'

अगर key नहीं मिलती और default value भी नहीं set नहीं की जाती है तो 'keyError' exception आ जाता है |

```
dict = {"one":1, "two":2, "three":3}
print(dict)
print(dict.pop("four"))
```

Output :

```
    print(dict.pop("four"))
KeyError: 'four'
```

Rename key name using pop() Method

```
dict = {"one":1, "two":2, "three":3}
print("Before Changing Key :")
print(dict)
dict["four"] = dict.pop("three")
print("After Changing Key :")
print(dict)
```

Output :

```
Before Changing Key :
{'one': 1, 'two': 2, 'three': 3}
After Changing Key :
{'one': 1, 'two': 2, 'four': 3}
```

popitem() Dictionary Method

popitem() method में दिए गए dictionary के last item को remove करके (key, value) return करता है |

Syntax

```
dict.popitem()
```

Parameter :

popitem() method के लिए कोई parameter नहीं होता है |

Returning Value

popitem() method दिए गए dictionary के last item को remove करके (key, value) return करता है |

Source Code :

```
dict = {"one":1, "two":2, "three":3}
print("Before Removing last item : ")
print(dict)
print("Removed item :",dict.popitem())
print("After Removing last item : ")
print(dict)
```

Output :

```
Before Removing item :
{'one': 1, 'two': 2, 'three': 3}
Removed item : ('three', 3)
After Removing item :
{'one': 1, 'two': 2}
```

setdefault() Dictionary Method

setdefault() method में दी हुई key exist होती है तो उसकी value return की जाती है | अगर key नहीं होती है तो set की हुई default value को return किया जाता है |

Syntax

```
dict.setdefault(key, default)
```

Parameter :

key : यहाँ पर value return करने के लिए key दी जाती है |

default : Optional. अगर key exist नहीं होती है तो यहाँ पर दी हुई value retuurn होती है |

Returning Value

setdefault() method दी हुई key exist होती है तो उसकी value return की जाती है | अगर key नहीं होती है तो set की हुई default value को return किया जाता है |
अगर दी हुई key exist नहीं होती है और default value set नहीं की जाती है तो 'None' return होता है |

Source Code :

अगर key dictionary पर होती है तो उसकी value return होती है |

```
dict = {"one":1, "two":2, "three":3}
print(dict.setdefault("one"))
```

Output :

```
1
```

If key is not exist

अगर key dictionary पर नहीं होती है और default parameter का इस्तेमाल नहीं किया जाता है तो 'None' return होता है |

Source Code :

```
dict = {"one":1, "two":2, "three":3}
print(dict.setdefault("four"))
```

Output :

```
None
```

Add new key and value in Dictionary

अगर key dictionary पर नहीं होती है और default parameter का इस्तेमाल किया जाता है तो dictionary के आखिर में उनकी pair add की जाती है |

Source Code :

```
dict = {"one":1, "two":2, "three":3}
print("Before Adding item :")
print(dict)
dict.setdefault("four", 4)
print("After Adding item :")
print(dict)
```

Output :

```
Before Adding item :
{'one': 1, 'two': 2, 'three': 3}
After Adding item :
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
```

update() Dictionary Method

update() method का इस्तेमाल old dictionary को update करने के लिए किया जाता है |

Syntax

```
oldDict.update(newDict)
```

Parameter :

newDict : यहाँ पर old dictionary को update करने के लिए new dictionary दी जाती है |

Returning Value

update() method में update हुए dictionary को return करता है |
update() method में keys की values को update और नए items add किये जाते है |

Source Code :

```
dict1 = {"one":1, "two":2, "three":3}
print("Before Updating or Adding Keys and Values :")
print(dict1)
dict2 = {"one":6, "four":4, "five":5}
print("After Updating or Adding Keys and Values:")
dict1.update(dict2)
print(dict1)
```

Output :

```
Before Updating or Adding Key :
{'one': 1, 'two': 2, 'three': 3}
After Updating or Adding Key :
{'one': 6, 'two': 2, 'three': 3, 'four': 4, 'five': 5}
```

values() Dictionary Method

values() method में dictionary के सिर्फ keys की values को return करता है |

Syntax

```
dict.values()
```

Parameter :

values() method के लिए कोई parameter नहीं होता है |

Returning Value

values() method; dictionary के सिर्फ keys की values को return करता है |

Source Code :

```
dict = {"one":1, "two":2, "three":3}
print(dict.values())
```

Output :

```
dict_values([1, 2, 3])
```

Set

Introduction

Python में set data type में elements का set होता है | Python में set data type ये सबसे अलग hashable object है |
Python के set में element के रूप में Numbers, letters या word हो सकते है |
Python के set में हर एक element नया होता है और इन नए elements को change नहीं किया जा सकता है | Set में दिए हुए elements immutable होते है | लेकिन किसी method के साथ set के elements को add या remove किया जा सकता है |
Set ये intersection, union, difference और symmetric difference जैसे mathematical operations करता है |

Types of Set

Set के दो प्रकार होते है |

1. Set
2. Frozenset

1. Set

Creating Set

set के elements को curly braces({}) के अन्दर लिखा जाता है और हर एक element को comma(,) से seperate किया जाता है |

Source Code :

```
set1 = {"H", "e", "l", "l", "o"}
#elements in curly braces
set2 = set("Hello")
#passing string to set function
set3 = set(["H", "e", "l", "l", "o"])
#passing list to set function
set4 = set(("H", "e", "l", "l", "o"))
#passing tuple to set function
set5 = set()
#empty set
```

Mixed Data Type Set

set ये mutable items को support नहीं करता है | set में [1, 2] ये mutable item है |

Source Code :

```
set = {1, "H", 2.6, [1,2]}

#Output
#    set = {1, "H", 2.6, [1,2]}
#TypeError: unhashable type: 'list'
```

set में tuple का इस्तेमाल किया जा सकता है | क्योंकि set और tuple ये immutable होता है |

Source Code :

```
set = {1, "H", 2.6, (1,2)}
```

Empty Set

Set के elements को curly braces({}) में लिखा जाता है लेकिन अगर empty curly braces({}) का इस्तेमाल किया जाता है तो वो 'dictionary' type हो जाता है |
Empty Set के लिए empty set() function को देना पड़ता है |

Source Code :

```
set1 = {}
print(type(set1))
#Output : <class 'dict'>

set2 = set()
print(type(set2))
#Output : <class 'set'>
```

Removing Duplicate Element

Set duplicate element को remove कर देता है |

Source Code :

```
set = {"H", "e", "l", "l", "o"}
print(set)
#Output : {'o', 'l', 'H', 'e'}
```

Try to Changing Set's element

Set में element की values index से change नहीं की जा सकती है | Set ये indexing को support नहीं करता है |

Source Code :

```
set = {1, 2, 5, 3, 4}
set[1] = 6
#Output :
#    set[1] = 6
#TypeError: 'set' object does not support item assignment
```

Adding Element using add() method to Set

अगर Set पर एक element को add करना हो तो 'add()' method का इस्तेमाल किया जाता है |

Source Code :

```
set = {1, 2, 5, 3, 4}
set.add(6)
print(set)
```

Output :

```
{1, 2, 3, 4, 5, 6}
```

Removing Element using remove() method from Set

अगर Set से एक element remove करना हो तो 'remove()' method का इस्तेमाल किया जाता है |

Source Code :

```
set = {1, 2, 5, 3, 4}
set.remove(5)
print(set)
```

Output :

```
{1, 2, 3, 4}
```

Removing All Elements using 'clear()' method from Set

Set में से सभी elements remove करने के लिए 'clear()' method का इस्तेमाल किया जाता है |

Source Code :

```
set = {1, 2, 5, 3, 4}
set.clear()
print(set)
```

Output :

```
set()
```

Iterating Over a Set

हर element को iterate through print करने के लिए 'for_in' loop का इस्तेमाल किया जाता है |

Source Code :

```
set = {1, 2, 5, 3, 4}
for i in set:
    print(i)
```

Output :

```
1
2
3
4
5
```

Set Functions in Python

Set Functions	Description
all()	sequence के सभी elements True होते है तो ये तो ये True return करता है |
any()	sequence का एक या सभी elements True होते है तो ये तो ये True return करता है |
enumerate()	दिए गए start से index और उसकी value की pair return करता है |
len()	set की length को return करता है |
max()	set से max value को return करता है |
min()	set से min value को return करता है |
set()	अलग ढंग से set को create या sequence को set में convert करता है |
sorted()	दिए गए sequence को sort करके return करता है |
sum()	दिए गए sequence या collection के elements को add करके उनका sum return करता है |

all() Set Function

all() function में sequence के सभी elements True होते है तो ये तो ये True return करता है |

Syntax

```
all(seq)
```

Parameter :

seq : यहाँ पर sequence दिया जाता है |

अगर sequence के सभी elements True होते है तो ये True return करता है |
अगर sequence का एक ही element True और बाकी elements False होते है तो ये False return करता है |
अगर sequence का एक ही element False और बाकी elements True होते है तो ये False return करता है |
अगर sequence के सभी elements False होते है तो ये False return करता है |
अगर sequence empty होता है तो ये True return करता है |

Returning Value

all() function set के true और False elements के हिसाब से boolean value return करता है |

Source Code :

```
set1 = {1, 2, 3, 4, 5}
print(all(set1)) #True
set2 = {1, 3, False}
print(all(set2)) #False
set3 = {False, True, False}
print(all(set3)) #False
empSet = {}
print(all(empSet)) #True
```

Output :

```
True
False
False
True
```

any() Set Function

any() function में sequence का एक या सभी elements True होते है तो ये तो ये True return करता है |

Syntax

```
any(seq)
```

Parameter :

seq : यहाँ पर sequence दिया जाता है |

अगर sequence के सभी elements True होते है तो ये True return करता है |
अगर sequence का एक ही element True और बाकी elements False होते है तो ये True return करता है |
अगर sequence का एक ही element False और बाकी elements True होते है तो ये True return करता है |
अगर sequence के सभी elements False होते है तो ये False return करता है |
अगर sequence empty होता है तो ये False return करता है |

Returning Value

any() function set के true और False elements के हिसाब से boolean value return करता है |

Source Code :

```
set1 = {1, 2, 3, 4, 5}
print(any(set1)) #True
set2 = {1, 3, False}
print(any(set2)) #True
set3 = {False, True, False}
print(any(set3)) #True
set4 = {False, False}
print(any(set4)) #False
empSet = {}
print(any(empSet)) #False
```

Output :

```
True
True
True
False
False
```

enumerate() Set Function

enumerate() function; दिए गए start से index और उसकी value की pair return करता है |

Syntax

```
enumerate(seq, start)
```

Parameter :

seq : यहाँ पर sequence दिया जाता है |

start : Optional. जिस index से start करना है वो index यहाँ पर दी जाती है | For eg., अगर 2 ये index दी जाती है तो index 2 होगा और sequence के पहली value की pair होगी | अगर दिया नहीं जाता तो default '0' होता है |

Returning Value

enumerate() function दिए गए start से index और उसकी value की pair enumerate object में return करता है |

Source Code :

```
set1 = {1, 2, 3, 4, 5}
print(set(enumerate(set1)))
set2 = {"H", "e", "l", "l", "o"}
print(set(enumerate(set2)))
print(set(enumerate(set2, 2)))
```

Output :

```
{(0, 1), (1, 2), (4, 5), (2, 3), (3, 4)}
{(0, 'o'), (2, 'e'), (1, 'l'), (3, 'H')}
{(4, 'e'), (2, 'o'), (3, 'l'), (5, 'H')}
```

len() Set Function

len() function का इस्तेमाल set की length को return करने के लिए किया जाता है |

Syntax

```
len(set)
```

Parameter :

set : जिस set की length check करनी है वो set यहाँ पर दी जाती है |

Returning Value

len() function set की length को number में return करता है |

Example for len() Set Function in Python

Set में 5 ही elements है लेकिन set की length 4 ही return हुई है | Set; duplicate element को remove करता है |

Source Code :

```
set = {"H", "e", "l", "l", "o"}
print(len(set))
```

Output :

```
4
```

max() Set Function

max() function का इस्तेमाल set से max value को return करने के लिए किया जाता है |

Syntax

```
max()
```

Parameter :

max() function के लिए कोई parameter नहीं होता है |

Returning Value

max() function set से max value return करता है |
max() और min() function ascii value के हिसाब से item को return करता है |

Source Code :

```
set = {"H", "e", "l", "l", "o", " ", "W", "o", "r", "l", "d"}
print(max(set))
```

Output :

```
r
```

min() Set Function

min() function का इस्तेमाल set से min value को return करने के लिए किया जाता है |

Syntax

```
min()
```

Parameter :

min() function के लिए कोई parameter नहीं होता है |

Returning Value

min() function set से min value return करता है |
max() और min() function ascii value के हिसाब से item को return करता है |

Source Code :

Example पर space( ) को return किया गया है |

```
set = {"H", "e", "l", "l", "o", " ", "W", "o", "r", "l", "d"}
print(min(set))
```

Output :

```

```

set() Set Function

set() function; का इस्तेमाल अलग ढंग से set को create या sequence को set में convert करने के लिए किया जाता है |

Syntax

```
set(seq)
```

Parameter :

seq : यहाँ पर set में convert करने के लिए sequence(tuple,string) या collection(dictionary,set) दिया जाता है |

Returning Value

set() function; sequence को set में convert करके set को return करता है |

Source Code :

```
set1 = set((2, 3, 5, 6, 9))
print(set1)
#converts tuple to set

set2 = set({"H","e","l","l","o"})
print(set2)
#converts set to set

set3 = set({"H":1,"e":2})
print(set3)
#converts Dictionary to set

set4 = set("Hello")
print(set4)
#converts string to set
```

Output :

```
{2, 3, 5, 6, 9}
{'o', 'l', 'H', 'e'}
{'H', 'e'}
{'o', 'l', 'H', 'e'}
```

sorted() Set Function

sorted() function; में दिए गए sequence को sort करके return करता है |

Syntax

```
sorted(seq, key, rev)
```

Parameter :

seq : यहाँ पर sequence(list, tuple, string) या collection(dictionary, set) दिया जाता है |

key : Optional. किस तरह से sort करना है यहाँ पर वो function आता है |

reverse : Optional. अगर reverse True दिया जाता है तो ascending order का sequence; descending order में reverse होता है |

Returning Value

sorted() function में दिए गए sequence को sort करके list में return किया जाता है |

Source Code :

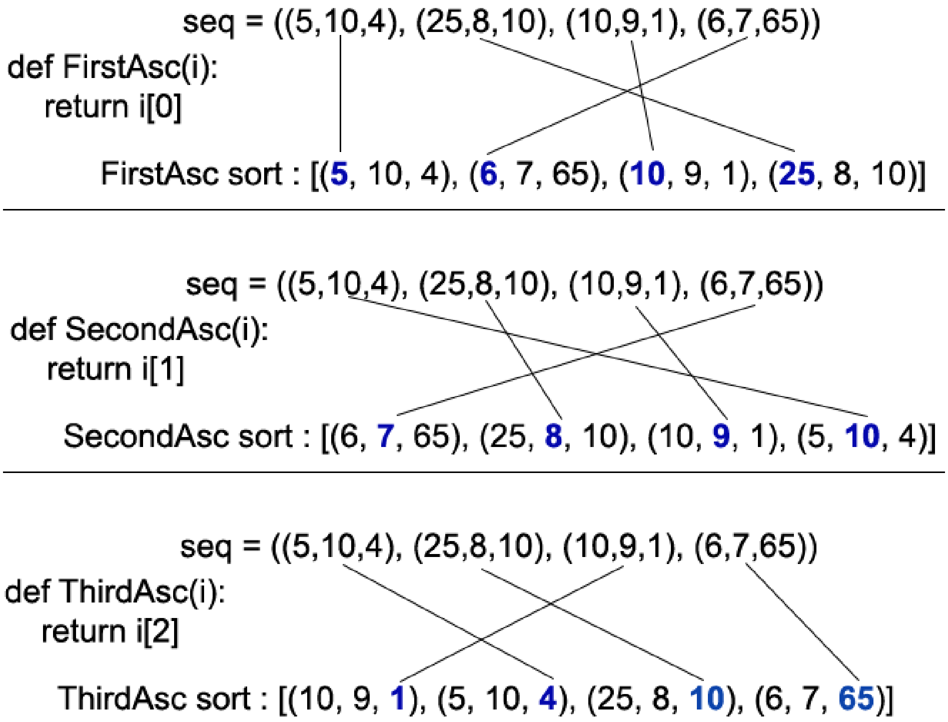
sorted() function; ascending order में default sort करता है |

```
set1 = {5, 8, 9, 7, 5}
print(sorted(set1))
set2 = {"P", "y", "t", "h", "o", "n"} #'P' in Uppercase
print(sorted(set2))
set3 = {"p", "y", "t", "h", "o", "n"} #'p' in lowercase
print(sorted(set3))
```

Output :

```
[5, 7, 8, 9]
['P', 'h', 'n', 'o', 't', 'y']
['h', 'n', 'o', 'p', 't', 'y']
```

Using 'key' Parameter in sorted() Function



Source Code :

sorted() function; ascending order में default sort करता है |

```
seq = {(5,10,4), (25,8,10), (10,9,1), (6,7,65)}
def FirstAsc(i):
    return i[0]
print("FirstAsc sort :",sorted(seq, key=FirstAsc))

def SecondAsc(i):
    return i[1]
print("SecondAsc sort :",sorted(seq, key=SecondAsc))

def ThirdAsc(i):
    return i[2]
print("ThirdAsc sort :",sorted(seq, key=ThirdAsc))
```

Output :

```
FirstAsc sort : [(5, 10, 4), (6, 7, 65), (10, 9, 1), (25, 8, 10)]
SecondAsc sort : [(6, 7, 65), (25, 8, 10), (10, 9, 1), (5, 10, 4)]
ThirdAsc sort : [(10, 9, 1), (5, 10, 4), (25, 8, 10), (6, 7, 65)]
```

Using 'reverse' Parameter in sorted() Function

Source Code :

अगर True दिया जाता है तो sequence को descending order में sort किया जाता है |

```
seq =  {1, 2, 3, 4, 5}
print("Ascending Order :", sorted(seq))
print("Descending Order :", sorted(seq, reverse=True))
```

Output :

```
Ascending Order : [1, 2, 3, 4, 5]
Descending Order : [5, 4, 3, 2, 1]
```

sum() Set Function

sum() function; में दिए गए sequence या collection के elements को add करके उनका sum return करता है |

Syntax

```
sum(seq, startAdd)
```

Parameter :

seq : यहाँ पर sequence या collection दिया जाता है |

startAdd : Optional. यहाँ पर जो संख्या दी जाती है वो sequence या collection का पहला element होता है | अगर दिया नहीं जाता है तो default '0' होता है |

Returning Value

sum() function में दिए गए sequence या collection के elements का sum return किया जाता है |

Source Code :

```
seq = {-5, 6, 1}
print(sum(seq))
print(sum(seq, 5))
```

Output :

```
2
7
```

Set Methods in Python

Set Methods	Description
add()	इस method इस्तेमाल set पर element add करने के लिए किया जाता है I
clear()	दिए हुए set को clear करता है I
copy()	दिए हुए set को copy करता है I
difference()	दिए हुए दो sets का difference update करके return करता है I
difference_update()	दिए हुए दो sets का difference update करता है I
discard()	दिए गए set में से दिए हुए element को remove करता है I
intersection()	दिए हुए sets से एक जैसे element(s) को set में return करता है I
intersection_update()	इस method का इस्तेमाल दिए हुए sets से एक जैसे element(s) को ढूंढने के लिए किया जाता है I
isdisjoint()	अगर दिए हुए दो set पूरे अलग होते है तो True return किया जाता है I
issubset()	अगर एक set का subset दूसरा set होता है तो True return होता है I
issuperset()	अगर दिया हुआ मुख्य set; दिए हुए set का superset होता है तो 'True' return होता है I
pop()	random element को remove करके return करता है I
remove()	इस method का इस्तेमाल set से एक element remove करने के लिए किया जाता है I
symmetric_difference()	दिए हुए दो sets से भिन्न elements को set में return किया जाता है I
symmetric_difference_update()	दिए हुए दो sets से भिन्न elements को ढूंढा जाता है I
union()	दिए गए sets को इकठ्ठा करके set को return किया जाता है I
update()	इस method का इस्तेमाल set को update करने के लिए किया जाता है I

add() Set Method

add() method का इस्तेमाल set पर element add करने के लिए किया जाता है |

Syntax

```
set.add(element)
```

Parameter :

element : set पर add करने के लिए यहाँ पर एक element दिया जाता है |

Returning Value

add() method 'None' return करता है |

Source Code :

```
set = {2, 3, 5}
print("Before Adding Element")
print(set)

set.add(6)
print("After Adding Element")
print(set)
```

Output :

```
Before Adding Element
{2, 3, 5}
After Adding Element
{2, 3, 5, 6}
```

clear() Set Method

clear() method का इस्तेमाल set को clear करने के लिए किया जाता है |

Syntax

```
set.clear()
```

Parameter :

clear() method के लिए कोई parameter नहीं होता है |

Returning Value

clear() method 'None' return करता है |

Source Code :

```
set = {2, 3, 5}
print("Before Removing Elements")
print(set)

set.clear()
print("After Removing Elements")
print(set)
```

Output :

```
Before Removing Elements
{2, 3, 5}
After Removing Elements
set()
```

copy() Set Method

copy() method का इस्तेमाल set को copy करने के लिए किया जाता है |

Syntax

```
set.copy()
```

Parameter :

copy() method के लिए कोई parameter नहीं होता है |

Returning Value

copy() method copy हुए set को return करता है |

Source Code :

```
set = {2, 3, 5}
print("Copied Set")
set.copy()
print(set)
```

Output :

```
Copied Set
{2, 3, 5}
```

difference() Set Method

difference() method दिए हुए दो sets का difference update करके return करता है |

Syntax

```
A.difference(B) | B.difference(A)
```

Returning Value

difference() method update हुए set को return करता है |

If A.difference(B),
अगर B के elements; A में हो तो A से remove करके set के difference को return किया जाता है |

Source Code :

```
A = {4, 5, 7, 9}
B = {4, 5, 8}
print(A.difference(B))
#same as A-B
```

Output :

```
{9, 7}
```

If B.difference(A),
अगर A के elements; B में हो तो B से remove करके set के difference को return किया जाता है |

Source Code :

```
A = {4, 5, 7, 9}
B = {4, 5, 8}
print(B.difference(A))
#same as B-A
```

Output :

```
{8}
```

difference_update() Set Method

difference_update() method दिए हुए दो sets का difference update करता है |

Syntax

```
A.difference_update(B) | B.difference_update(A)
```

Returning Value

difference_update() method 'None' return करता है |

If A.difference(B),
अगर B के elements; A में हो तो A से remove/update किया जाता है |

Source Code :

```
A = {4, 5, 7, 9}
B = {4, 5, 8}
A.difference_update(B)
print("set A :",A)
print("set B :",B)
```

Output :

```
set A : {7, 9}
set B : {8, 4, 5}
```

If B.difference(A),
अगर A के elements; B में हो तो B से remove/update किया जाता है |

Source Code :

```
A = {4, 5, 7, 9}
B = {4, 5, 8}
B.difference_update(A)
print("set A :",A)
print("set B :",B)
```

Output :

```
set A : {9, 4, 5, 7}
set B : {8}
```

discard() Set Method

discard() method में दिए गए set में से दिए हुए element को remove करता है |

Syntax

```
set.discard(element)
```

Parameter

element : set से जो element remove करना है वो element यहाँ पर दिया जाता है |

Returning Value

discard() method 'None' return करता है |
discard() method का इस्तेमाल set से एक element remove करने के लिए किया जाता है |

Source Code :

```
A = {4, 5, 7, 9}
A.discard(5)
print(A)
```

Output :

```
{9, 4, 7}
```

intersection() Set Method

intersection() method दिए हुए sets से एक जैसे element(s) को set में return करता है |

Syntax

```
A.intersection(MultiSets)
```

MultiSets में एक या एक से ज्यादा sets हो सकते है | हर set को comma(,) से seperate किया जाता है |

Returning Value

intersection() method में sets से एक जैसे element(s) का set बनाकर return करता है |

Source Code :

```
A = {4, 5, 7, 9}
B = {8, 4, 5}
C = {5, 4, 5, 4, 5}
D = {4, 3, 2, 1}
print(A.intersection(B))
print(A.intersection(B))
print(A.intersection(B, C))
print(A.intersection(B, C, D))
print(A.intersection(C, D))
```

Output :

```
{4, 5}
{4, 5}
{4, 5}
{4}
{4}
```

intersection_update() Set Method

intersection_update() method का इस्तेमाल दिए हुए sets से एक जैसे element(s) को ढूंढने के लिए किया जाता है |

Syntax

```
A.intersection_update(MultiSets)
```

MultiSets में एक या एक से ज्यादा sets हो सकते है | हर set को comma(,) से seperate किया जाता है |

Returning Value

intersection_update() method 'None' return करता है |

Source Code :

```
A = {4, 5, 7, 9}
B = {8, 4, 5, 7}
C = {5, 4, 5, 4, 5}
D = {4, 3, 8, 2, 1}
A.intersection_update(B)
print(A)
B.intersection_update(A)
print(B)
C.intersection_update(B, C, D)
print(C)
D.intersection_update(C)
print(D)
C.intersection_update(D)
print(C)
```

Output :

```
{4, 5, 7}
{4, 5, 7}
{4}
{4}
{4}
```

iisdisjoint() Set Method

isdisjoint() method में अगर दिए हुए दो set पूरे अलग होते है तो True return किया जाता है |

Syntax

```
A.isdisjoint(B)
```

Returning Value

isdisjoint() method में दिए गए दो sets disjoint होते है तो True return करता है या disjoint नहीं होते है False return होता है |

Source Code :

```
A = {4, 5, 7, 9}
B = {1, 2, 3, 10}
C = {5, 10, 15, 20}

print(A.isdisjoint(B))
print(A.isdisjoint(C))
```

Output :

```
True
False
```

issubset() Set Method

issubset() method में एक set का subset दूसरा set होता है तो True return होता है |

Syntax

```
A.issubset(B) | B.issubset(A)
```

Returning Value

issubset() method में अगर A ये B का subset होता है तो 'True' return होता है | अगर subset नहीं होता है तो 'False' return होता है |

If A.issubset(B),
अगर A ये B का subset होता है तो True return करता है |

Source Code :

```
A = {9, 7}
B = {4, 5, 7, 9}

print(A.issubset(B))
```

Output :

```
True
```

If B.issubset(A),
Source Code :

```
A = {9, 7}
B = {4, 5, 7, 9}

print(B.issubset(A))
```

Output :

```
False
```

issuperset() Set Method

issuperset() method में अगर दिया हुआ मुख्य set; दिए हुए set का superset होता है तो 'True' return होता है |

Syntax

```
A.issuperset(B) | B.issuperset(A)
```

Returning Value
issuperset() method में अगर A ये B का superset होता है तो 'True' return होता है | अगर नहीं होता है तो 'False' return होता है |

If A.issuperset(B),
अगर A ये B का superset होता है तो True return करता है |

Source Code :

```
A = {4, 5, 7, 9}
B = {7}

print(A.issuperset(B))
```

Output :

```
True
```

If B.issuperset(A),
Source Code :

```
A = {4, 5, 7, 9}
B = {7}

print(B.issuperset(A))
```

Output :

```
False
```

pop() Set Method

pop() method; set में से random element को remove करके return करता है |

Syntax

```
set.pop()
```

Parameter :

pop() method के लिए कोई parameter नहीं होता है |

Returning Value

pop() method; removed हुए element को return करता है |

Source Code :

```
set = {4, 5, 7, 9}
print(set)
print(set.pop())
print(set)
```

Output :

```
{9, 4, 5, 7}
9
{4, 5, 7}
```

remove() Set Method

remove() method का इस्तेमाल set से एक element remove करने के लिए किया जाता है |

Syntax

```
set.remove(element)
```

Returning Value

remove() method; 'None' return करता है |
अगर दिया हुआ element; set पर नहीं मिलता है तो 'keyError' exception आ जाता है |

Source Code :

```
set = {4, 5, 7, 9}
set.remove(4)

print(set)
```

Output :

```
{9, 5, 7}
```

symmetric_difference() Set Method

symmetric_difference() method में दिए हुए दो sets से भिन्न elements को set में return किया जाता है |

Syntax

```
A.symmetric_difference(B)
```

Returning Value

symmetric_difference() method में दो sets से भिन्न elements को set में return करता है |

Source Code :

```
A = {4, 5, 7, 9}
B = {8, 4, 5}
C = {5, 4, 5, 4, 5}
D = {4, 3, 2, 1}

print(A.symmetric_difference(B))
#same as A^B

print(B.symmetric_difference(C))
#same as B^C

print(C.symmetric_difference(D))
#same as C^D

print(D.symmetric_difference(D))
#same as D^D
```

Output :

```
{7, 8, 9}
{7, 8, 9}
{8}
{1, 2, 3, 5}
set()
```

symmetric_difference_update() Set Method

symmetric_difference_update() method में दिए हुए दो sets से भिन्न elements को ढूंढा जाता है |

Syntax

```
A.symmetric_difference_update(B)
```

Returning Value

symmetric_difference_update() method 'None' return करता है |

Source Code :

```
A = {4, 5, 7, 9}
B = {8, 4, 5}
C = {5, 4, 5, 4, 5}
D = {4, 3, 2, 1}

A.symmetric_difference_update(B)
print(A)

B.symmetric_difference_update(C)
print(B)

C.symmetric_difference_update(D)
print(C)

D.symmetric_difference_update(D)
print(D)
```

Output :

```
{7, 8, 9}
{8}
{1, 2, 3, 5}
set()
```

union() Set Method

union() method में दिए गए sets को इकठ्ठा करके set को return किया जाता है |

Syntax

```
A.union(MultiSets)
```

MultiSets में एक या एक से ज्यादा sets हो सकते है | हर set को comma(,) से seperate किया जाता है |

Returning Value

union() method में दिए गए sets को इकठ्ठा करके set को return करता है |

Source Code :

```
A = {4, 5, 7, 9}
B = {8, 4, 5}
C = {5, 4, 5, 4, 5}
D = {4, 3, 2, 1}

print(A.union(B))
print(B.union(A))

print(C.union(D))
print(D.union(C))
```

Output :

```
{4, 5, 7, 8, 9}
{4, 5, 7, 8, 9}
{1, 2, 3, 4, 5}
{1, 2, 3, 4, 5}
```

update() Set Method

update() method का इस्तेमाल set को update करने के लिए किया जाता है |

Syntax

```
set1.update(set2)
```

Parameter

set2 : यहाँ पर दिए हुए set से set1 को update किया जायेगा |

Returning Value

update() method; 'None' return करता है |

Source Code :

```
set1 = {4, 5, 7, 9}
set2 = {7, 8, 9, 10}
set1.update(set2)
print("set1 :",set1)
set2.update(set1)
print("set2 :",set2)
```

Output :

```
set1 : {4, 5, 7, 8, 9, 10}
set2 : {4, 5, 7, 8, 9, 10}
```

String

String ये Common Data Type है | ये data type सामान्यत: सभी Computer Languages में पाया जाता है |
String ये एक से ज्यादा charcaters का sequence होता है |

Example

```
Hello Programmer ! I am a String
```

Python में String को single quotes(' ') या double quotes(" ") में लिखा जाता है |

Example

```
'Enclosing String in single quotes'
"Enclosing String in double quotes"
```

Example for String

Source Code :

```
var str = "Hello World"
print(str)
```

Output :

```
Hello World
```

Python String Index

String की index '0' से शुरू होता है और आखिरी index '-1' होता है |

Source Code :

```
str = "Hello World"
print(str[0])
print(str[-1])
```

Output :

```
H
d
```

Creating Substring from String

Python में string से substring को create करना हो तो square bracket([]) में colon(:) का इस्तेमाल किया जाता है |

Syntax for Creating Substring

सिर्फ starting index दिया जाता है तो startIndex से पूरा string display किया जाता है |

```
str[start_Index : end_Index(Optional default '-1')]
```

सिर्फ ending index दिया जाता है तो 0th index से endIndex तक string display किया जाता है |

```
str[start_Index(Optional default '0') : end_Index]
```

Source Code :

```
str = "Hello World"
print(str[3:])
print(str[:8])
print(str[3:8])
```

Output :

```
lo World
Hello Wo
lo Wo
```

Python Escape Characters

Escape Characters	**Escape Characters**	**Example/Output**
\a	alert OR Bell	print("\a") /
\b	Backspace	print("Hello\bWorld") /HellWorld
\f	formfeed	print("Hello\fWorld") / Hello World
\n	Newline/LineFeed	print("Hello\nWorld") / Hello World
\r	Carriage return	print("Hello\rWorld") / HelloWorld
\s	Space	print("Hello\sWorld") / Hello World
\t	Tab	print("Hello\sWorld") / Hello World
\v	Vertical Tab	
\\	Backslash	print("\\") / \
\'	single Quote	print("\'") / '
\"	Double Quote	print("\"") / "
\\uxxxx	16-bit Hexadecimal Unicode	print("\u003D") / =
\Uxxxxxxxx	32-bit Hexadecimal Unicode	print("\U0000003D") / =
\\xhh	character based hexadecimal	print("\x30") / 0
\ooo	character based octal	print("\060") / 0

Use Triple single quotes(''' ''') OR double quotes(""" """)

triple single या double quote का इस्तेमाल सिर्फ multiline string और docstring के लिए किया जाता है |

Source Code :

```
str1 = '''Hello
       World'''
print(str1)

str2 = """Hello
       World"""
print(str2)
```

Output :

```
Hello
       World
Hello
       World
```

Working with Mixed Data Type String

सिर्फ string को ही concatenate किया जा सकता है |

Source Code :

```
print("1"+"2")
print(1+2)
print('1'+2)
```

Output :

```
12
3
    print('1'+2)
TypeError: must be str, not int
```

Old Way String Formatting(C-Style)

Python String Formatting with Format Specifiers(Modulus Operator)

सामान्यतः विशेष रूप से C Programming और आदि में Formatting का इस्तेमाल किया जाता है | वहा पर printf() function का इस्तेमाल किया जाता है |
Python में भी print function में Format Specifiers का इस्तेमाल किया जाता है |

Format Specifier	**Description**
%c	character
%d	signed Integer
%e	lowercase exponential notation
%E	uppercase exponential notation
%f	floating point number
%g	%e and %f shorter
%G	%E and %f shorter
%i	signed Integer
%o	octal Integer
%s	String
%u	unsigned Integer
%x	lowercase hexadecimal Integer
%X	uppercase hexadecimal Integer

How to Use Format Specifiers(%c, %d, %s etc)

Format Specifier का इस्तेमाल print() function में किया जाता है | print function में left hand side में format string को और right hand side में tuple का इस्तेमाल किया जाता है |

Example

```
print("String : %s" % ("Hello World"))
#Output : String : Hello World
```

In print()

- **"String : %s"** : Format String
- **%s** : Format Specifier
- **%** : Modulus Operator
- **("Hello World")** : tuple have only one element

%c(character)

Single Character के लिए '%c' इस format specifier का इस्तेमाल किया जाता है |

Example

```
a = "H"
b = "I"
print("%c %c" % (a, b)) #Output : H I
print("%c %c" % (b, a)) #Output : I H
```

%d(signed Integer)

यहाँ numeric value होती है लेकिन अपूर्णांकित हिस्सा नहीं होता है | अगर floating-point number होता है तो उसे Integer में convert किया जाता है |

Example

```
a = 4.4
print("Float to Integer : %d" % (a))
#Output : Float to Integer : 4
b = 4
print("Integer : %d" % (b))
#Output : Integer : 4
```

%e(lowercase exponential notation)

Integer या Floating-point Number को exponential notation में convert किया जाता है |

Example

```
a = 4.45878
print("%e" % (a))
#Output : 4.458780e+00

b = 45
print("%e" % (b))
#Output : 4.500000e+01

c = -45
print("%e" % (c))
#Output : -4.500000e+01
```

%E(uppercase exponential notation)

Integer या Floating-point Number को exponential notation में convert किया जाता है |

Example

```
a = 4.45878
print("%E" % (a))
#Output : 4.458780E+00

b = 45
print("%E" % (b))
#Output : 4.500000E+01

c = -45
print("%E" % (c))
#Output : -4.500000E+01
```

%f(floating-point number)

Floating-point Number के लिए '%f' format specifier का इस्तेमाल किया जाता है | अगर integer number होता है तो उसे floating-point number में convert किया जाता है |

Example

```
a = 4.45878
print("%f" % (a))
#Output : 4.45878

b = 45
print("%f" % (b))
#Output : 45.000000

c = -45
print("%f" % (c))
#Output : -45.000000
```

%g(%e and %f shorter)

'%g' का इस्तेमाल number को काफी छोटे हिस्से में convert करने के लिए किया जाता है | जरुरत पड़ने पर ये number को exponential number में भी convert करता है |

Example

```
a = 544878562656
print("%e \n%f \n%g\n" % (a, a, a))
#Output : 5.448786e+11
#544878562656.000000
#5.44879e+11

b = 2.55458566
print("%e \n%f \n%g" % (b, b, b))
#Output : 2.554586e+00
#2.554586
#2.55459
```

%G(%E and %f shorter)

'%G' का इस्तेमाल number को काफी छोटे हिस्से में convert करने के लिए किया जाता है | जरुरत पड़ने पर ये number को exponential number में भी convert करता है |

Example

```
a = 544878562656
print("%e \n%f \n%G\n" % (a, a, a))
#Output : 5.448786e+11
#544878562656.000000
#5.44879E+11

b = 2.55458566
print("%e \n%f \n%G" % (b, b, b))
#Output : 2.554586e+00
#2.554586
#2.55459
```

%i(signed Integer)

यहाँ numeric value होती है लेकिन अपूर्णांकित हिस्सा नहीं होता है | अगर floating-point number होता है तो उसे Integer में convert किया जाता है |

Example

```
a = 4.4
print("Float to Integer : %i" % (a))
#Output : Float to Integer : 4

b = 4
print("Integer : %i" % (b))
#Output : Integer : 4
```

%o(octal Integer)

Integer Number में decimal value को octal Integer में convert किया जाता है |

Example

```
a = -10
print("Decimal to Octal : %o" % (a))
#Output : Decimal to Octal : -12

b = 10
print("Decimal to Octal : %o" % (b))
#Output : Decimal to Octal : 12
```

%s(String)

String के लिए '%s' का इस्तेमाल किया जाता है | अगर numeric value दी जाती है तो उसे String में convert किया जाता है |

Example

```
a = -10
print("String : %s" % (a))
#Output : String : -10

b = "Hello World"
print("String : %s" % (b))
#Output : String : Hello World
```

%x(lowercase hexadecimal Integer)

Hexadecimal number के लिए '%x' का इस्तेमाल किया जाता है | Integer Number को Hexadecimal Number में convert किया जाता है |

Example

```
a = 15
print("Integer to Hexadecimal : %x" % (a))
#Output : Integer to Hexadecimal : f
```

%X(uppercase hexadecimal Integer)

Hexadecimal number के लिए '%X' का इस्तेमाल किया जाता है | Integer Number को Hexadecimal Number में convert किया जाता है |

Example

```
a = 15
print("Integer to Hexadecimal : %X" % (a))
#Output : Integer to Hexadecimal : F
```

Format Specifier with Placeholder

जरुरत के हिसाब से Integer या Floating-point Number को display करना हो तो placdholder का इस्तेमाल किया जाता है |

Syntax

```
%[flag][length].[precision][type]
```

Parts of Placeholder

% : format specifier में % का होना अनिवार्य होता है |
flag : Optional. #, +, -, 0 और space( ) ये flags दिए जाते है |
length : Optional. यहाँ पर Number की length दी जाती है |
. : Optional. अगर floating-point number होता है तो decimal point को दिया जा सकता है |
precision : Optional. decimal point के बाद कितने numbers चाहिए उन numbers की सख्या यहाँ पर दी जाती है |
type : type को देना अनिवार्य होता है | for Example d, i, f, e, E etc.

Basic Example for Placeholder

Example के पहले Statement में 45.5 ये value दी गयी है और '%5d' ये format specifier दिया गया है | पहले value को float से integer में convert किया जायेगा और बाद में उस integer की length '5' की जायेगी | यहाँ पर integer 2 digit का ही है | उस length को '5' करने के लिए पहले 3 space precede किये जायेंगे |
Example के दूसरे statement में 45.5955 ये value दी गयी है और '%5.2f' ये format specifier दिया गया है | लेकिन यहाँ पर length से ज्यादा महत्व precision को दिया गया है |

Source Code :

```
a = 45.5
print("%5d" % (a))
#Output :    45

b = 45.5955
print("%5.2f" % (b))
#Output : 45.60
```

Flag

String Format Specifier Flags

Flags	**Description**
#	%o, %x और %X के बीच में '#' flag का इस्तेमाल किया जाता है	
+	ये एक sign character है	अगर space precede होता है तो उसे '+' sign से replace किया जाता है	
-	ये left justification है	अगर space precede होता है तो उसे remove किया जाता है	
0	Number के left side से 0 के साथ pad किया जाता है	

Example for '#' Flag

Example में '#' flag का इस्तेमाल किया गया है | o(octal) के साथ '#' को दिया जाता है तो '0o' precede लगाया जाता है |
x(lowercase hexadecimal) के साथ '#' को दिया जाता है तो '0x' precede लगाया जाता है |
X(uppercase hexadecimal) के साथ '#' को दिया जाता है तो '0X' precede लगाया जाता है |

Source code :

```
a = 45
print("%#o" % (a))
print("%#x" % (a))
print("%#X" % (a))
```

Output :

```
0o55
0x2d
0X2D
```

Example for '+' Flag

Source code :

```
a = 45
print("%+5d" % (a))
print("%+d" % (a))
```

Output :

```
  +45
+45
```

Example for '-' Flag

Source code :

```
a = 45
print("%-5d" % (a))
print("%5d" % (a))
```

Output :

```
45
   45
```

Example for '0' Flag

Source code :

```
a = 45
print("%05d" % (a))
print("%5d" % (a))
```

Output :

```
00045
   45
```

New Way String Formatting(Python-Style)

String Formatting with format() function

format() function से किसी भी प्रकार से String Formatting किया जाता है |
Python मे format() function; string formatting के लिए काफी उपयुक्त function है |

Syntax

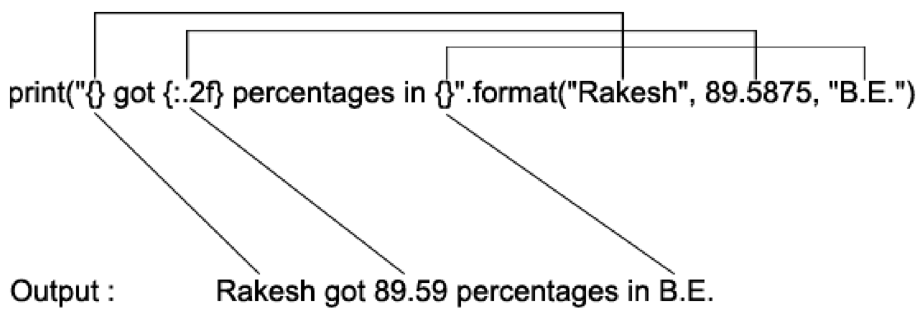
```
template.format(p0, p1, ..., pN, k0=v0, k1=v1, ..., kN=vN)
```

Parts of format() function

template : ये एक format string होता है, जिसमे एक से ज्यादा format codes({}) होते है | Format codes ये output में replace होनेवाली जगह होती है |
p0, p1, ..., pN : यहाँ पर ये positional parameters है | print() function में दिए गए placeholder({index}) की जगह format() पे दिए positional parameter से replace किया जाता है |
k0=v0, k1=v1, ..., kN=vN : यहाँ पर ये keyword parameters है | keyword parameter में key और value(key=value) इन दोनों की pairs होती है |print() function में दिए गए placeholder({key}) की जगह format() पे दिए keyword parameter से replace किया जाता है |
format() function ये formatted string को return करता है |

More About Positional Parameters

format() function को Example और उससे related example को समझे |



Source code :

```
print("{} got {:.2f} percentages in {}".format("Rakesh", 89.5875, "B.E."))
```

Output :

```
Rakesh got 89.59 percentages in B.E.
```

format() function के parameters से ज्यादा placeholders({}) नहीं लिए जा सकते है | अगर लिए जाते है तो 'indexError' exception आ जाता है |

For Example,

```
print("{} got {:.2f} percentages in {}".format("Rakesh", 89.5875))
```

Output :

```
    print("{} got {:.2f} percentages in {}".format("Rakesh", 89.5875))
IndexError: tuple index out of range
```

format() function के parameters Placeholders({}) से ज्यादा लिए जा सकते है |

For Example,

```
print("{} got {:.2f} percentages".format("Rakesh", 89.5875, "B.E."))
```

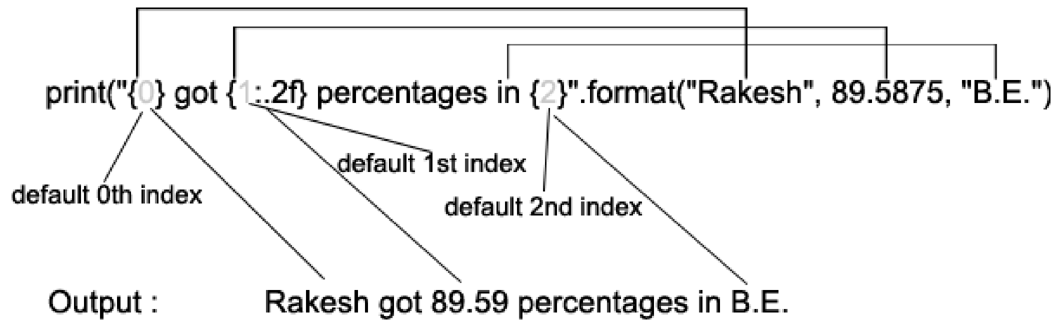
Output

```
Rakesh got 89.59 percentages
```

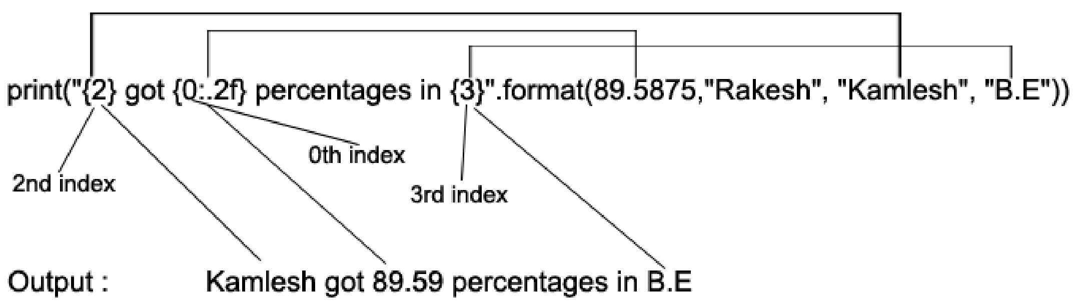
Know more About Placeholders and format() Function(Positional Parameters)

format() function को Example और उससे related example को समझे |

Placeholder में जब index दिया नहीं जाता है तो by default 0, 1, 2, 3 इस प्रकार से index interpreter द्वारा लिए जाते है | निचे image में देखके समझ आएगा |



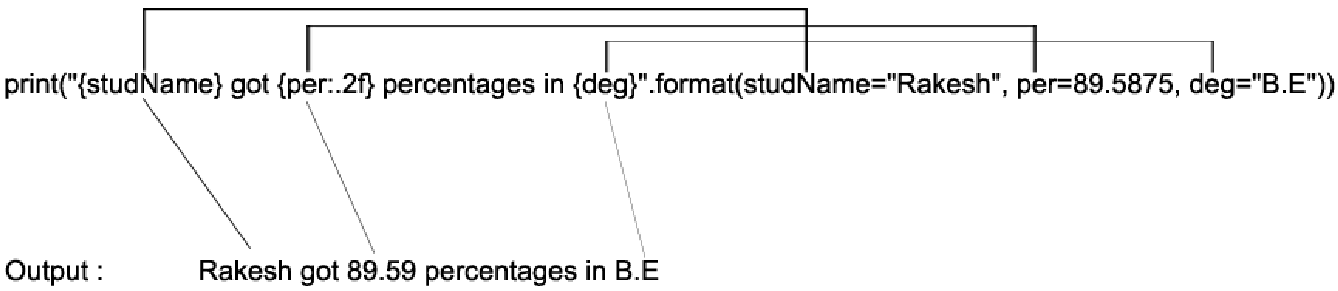
Programmer चाहे तो index को बदल भी सकता है |




```
print("{2} got {0:.2f} percentages in {3}".format(89.5875,"Rakesh", "Kamlesh", "B.E"))
#Output : Kamlesh got 89.59 percentages in B.E
```

More About Keyword Parameters

Keyword Parameters में key और value(key=value) की pairs होती है | Placeholders में key(index) के जरिये उनकी value को access किया जाता है |



Source Code :

```
print("{studName} got {per:.2f} percentages in {deg}".format(studName="Rakesh",
per=89.5875, deg="B.E"))
```

Output

```
Rakesh got 89.59 percentages in B.E
```

Using Format Specifier in Placeholders with format() Function

Format Specifier	**Description**
b	Binary
d	Integer
e	Exponential Notation(Lowercase)
E	Exponential Notation(Uppercase)
f	Floating-point (Lowercase inf, nan)
F	floating-point (Uppercase INF, NAN)
g	अगर number exponent(e) होने की बारी आती है तो decimal point के बाद सिर्फ 4 ही digit लिए जाते है | (like Lowercase 'e')
G	अगर number exponent(e) होने की बारी आती है तो decimal point के बाद सिर्फ 4 ही digit लिए जाते है | (Like Uppercase 'e')
o	Octal
s	String
x	hexadecimal(Lowercase)
X	Hexadecimal(Uppercase)

b(Binary)

Decimal को Binary में convert करने के लिए उपयोगी होता है |

Source code :

```
print("{:b}".format(15))
#Output : 1111
```

d(Integer)

Decimal को Binary में convert करने के लिए उपयोगी होता है |

Source code :

```
print("{:d}".format(15))
#Output : 15
```

e(Lowercase Exponential Notation)

दिए गए Integer या Floating-point Number को lowercase exponential notation में convert करता है |

Source code :

```
print("{:e}".format(15))
print("{:e}".format(15.58564))
#Output :
1.500000e+01
1.558564e+01
```

E(Uppercase Exponential Notation)

दिए गए Integer या Floating-point Number को Uppercase exponential notation में convert करता है |

Source code :

```
print("{:E}".format(15))
print("{:E}".format(15.58564))
#Output :
1.500000E+01
1.558564E+01
```

f(floating-point Number)

Integer को Floating-point Number में convert कर सकता है | decimal point के बाद सिर्फ '6' digit ही लेता है |

Source code :

```
import math
print("{:f}".format(45.55854455654564))
print("{:f}".format(4555854455654564))
print("{:f}".format(math.inf))
print("{:f}".format(math.nan))
#Output :
45.558545
4555854455654564.000000
inf
nan
```

F(Floating-point Number)

Integer को Floating-point Number में convert कर सकता है | decimal point के बाद सिर्फ '6' digit ही लेता है |

Source code :

```
import math
print("{:F}".format(45.55854455654564))
print("{:F}".format(4555854455654564))
print("{:F}".format(math.inf))
print("{:F}".format(math.nan))
#Output :
45.558545
4555854455654564.000000
INF
NAN
```

g(Like lowercase 'e')

Source code :

```
print("{:g}".format(15))
print("{:e}".format(4555854455654564))
print("{:g}".format(4555854455654564))
#Output :
15
4.555854e+15
4.55585e+15
```

G(Like Uppercase 'E')

Source code :

```
print("{:G}".format(15))
print("{:E}".format(4555854455654564))
print("{:G}".format(4555854455654564))
#Output :
15
4.555854E+15
4.55585E+15
```

o(Octal)

Decimal को octal में convert करने के लिए इस्तेमाल किया जाता है |

Source code :

```
print("{:o}".format(15))
print("{:o}".format(7))
print("{:o}".format(8))
#Output :
17
7
10
```

s(String)

String के लिए इस्तेमाल किया जाता है |

Source code :

```
print("{:o}".format(15))
print("{:o}".format(7))
print("{:o}".format(8))
#Output :
17
7
10
```

s(String)

Decimal को Hexadecimal(lowercase) number में convert किया जाता है |

Source code :

```
print("{:x}".format(15))
print("{:x}".format(10))
#Output :
f
a
```

X(Uppercase Hexadecimal)

Decimal को Hexadecimal(Uppercase) number में convert किया जाता है |

Source code :

```
print("{:X}".format(15))
print("{:X}".format(10))
#Output :
F
A
```

String Functions in Python

String Functions	Description
len()	string की length को return किया जाता है |
max()	String में से max character को return किया जाता है |
min()	String में से min character को return किया जाता है |

len() String Function

len() function में string की length return की जाती है |

Syntax

```
len(str)
```

Parameter

str : यहाँ पर जिस string की length return करनी है वो string दिया जाता है |

Returning Value

यहाँ पर string की length number में return की जाती है |

Source Code :

```
str = "Hello World"
print(len(str))
```

Output :

```
11
```

max() String Function

max() function में string में से max character को return किया जाता है |

Syntax

```
max(str)
```

Parameter

str : जिस string में से max character को return करना है वो string यहाँ पर दिया जाता है |

Returning Value

यहाँ पर max character को return किया जाता है |
min() और max() function में ascii value के हिसाब से character को return किया जाता है |

Source Code :

```
str = "Hello World"
print(max(str))
```

Output :

```
r
```

min() String Function

min() function में string में से max character को return किया जाता है |

Syntax

```
min(str)
```

Parameter

str : जिस string में से min character को return करना है वो string यहाँ पर दिया जाता है |

Returning Value

यहाँ पर min character को return किया जाता है |
min() और max() function में ascii value के हिसाब से character को return किया जाता है |

Source Code :

```
str = "Hello World"
print(min(str))
```

Output :

```
 
```

String Methods in Python

String Methods	Description
capitalize()	String के पहले word के पहले character को uppercase में convert किया जाता है	
center()	किसी विशिष्ट character से padded किये गए string को return करता है	
casefold()	normal string को casefold string में convert करता है	
count()	मुख्य string में से substring के occurrences; number में return किये जाते है	
endswith()	दिए गए suffix को string के end पर check करके boolean value return करता है	
expandtabs()	String में tab(s) की size को expand करके string की copy return की जाती है	
find()	substring को मुख्य string में ढूंढकर उसका पहला index return किया जाता है	
format()	इसका इस्तेमाल string formatting के लिए किया जाता है	
index()	दिए गए substring को मुख्य string में ढूंढकर उसका पहला index return किया जाता है	
isalnum()	अगर character या string alphanumeric या alphabetic या numeric होता है तो true return करता है अगर नहीं होते है तो false return होता है	
isalpha()	अगर character या string alphabetic होता है तो true return करता है अगर नहीं होते है तो false return होता है	
isdecimal()	अगर character या string decimal होता है तो true return करता है अगर नहीं होते है तो false return होता है	
isdigit()	अगर character या string digit होता है तो true return करता है अगर नहीं होते है तो false return होता है	
isidentifier()	दिए गए string या character एक valid identifier है या नहीं ये boolean value में return किया जाता है	
islower()	दिए गए string या character lowercase में है या नहीं ये check करके boolean value में return किया जाता है	
isnumeric()	दिए गए string या character numeric में है या नहीं ये check करके boolean value में return किया जाता है	
isprintable()	दिए गए string या character printable है या नहीं ये check करके boolean value में return किया जाता है	
isspace()	दिए गए string या character सिर्फ space है या नहीं ये check करके boolean value में return किया जाता है	
istitle()	दिए गया string; title string है या नहीं ये check करके boolean value में return किया जाता है	

isupper()	दिए गए string या character uppercase में है या नहीं ये check करके boolean value में return किया जाता है |
join()	दिए गए elements के sequence को किसी seperator से join करके string को return किया जाता है |
ljust()	string को left में justfy करके और width के हिसाब से दिए गए character से fill करके string को return किया जाता है |
rjust()	string को right में justfy करके और width के हिसाब से दिए गए character से fill करके string को return किया जाता है |
lower()	Uppercase के character या string को lowercase में convert किया जाता है |
upper()	lowercase के character या string को Uppercase में convert किया जाता है |
swapcase()	अगर string के characters; uppercase हो तो उसे lowercase में return करता है और lowercase हो तो उसे uppercase में return करता है |
lstrip()	left side(leading character) से दिए गए character को strip करके string की copy return की जाती है |
rstrip()	right side(trailing characters) से दिए गए character को strip करके string की copy return की जाती है |
strip()	left side(leading characters) और right side(trailing character) से दिए गए character को strip करके string की copy return की जाती है |
partition()	दिए गए String के लिए seperator का first occurrence को tuple के बीच में रख के tuple को return किया जाता है |
rpartition()	दिए गए String के लिए seperator का last occurrence को tuple के बीच में रख के tuple को return किया जाता है |
replace()	String के लिए दिए गए old substring के सभी occurrence नए substring से replace करके string की copy return की जाती है |
rfind()	दिए गए substring को मुख्य string में ढूंढकर उसका आखिरी index return किया जाता है
rindex()	दिए गए substring को मुख्य string में ढूंढकर उसका आखिरी index return किया जाता है
split()	दिए गए seperator से string को split करके list(sequence) में return किया जाता है |
rspilt()	दिए गए seperator से string को right side से split करके list(sequence) में return किया जाता है |
startswith()	दिए गए prefix को string के end पर check करके boolean value return करता है |
title()	Normal string को title string में convert करके return करता है |
zfill()	अगर string से ज्यादा length(width) दी जाती है तो left side से अतिरिक्त जगह पर '0' दिया जाता है और string की copy return की जाती है |

len()	String की length को return किया जाता है |
max()	String में से max character को return किया जाता है |
min()	String में से min character को return किया जाता है |

capitalize() String Method

capitalize() function का इस्तेमाल String के पहले word के पहले character को uppercase में convert किया जाता है |

Syntax

```
str.capitalize()
```

Parameter

capitalize() function के लिए कोई parameter नहीं होता है |

Returning Value

String के पहले word के पहले character को uppercase में convert करके पूरा string return करता है | अगर String के word का पहला character uppercase या Numeric value या special symbol हो तो original string return करता है |
Note : capitalize() function सिर्फ पहले word का पहला character uppercase में convert करता है और बाकी के characters; lowercase में convert कर देता है |

Source Code :

```
str = "hello world"
print(str.capitalize())
str = "Hello WORLD"
print(str.capitalize())
str = "123 Hello"
print(str.capitalize())
str = "@hello World"
print(str.capitalize())
```

Output :

```
Hello world
Hello world
123 hello
@hello world
```

center() String Method

center() function किसी विशिष्ट character से padded किये गए string को return करता है |

Syntax

```
str.center()
```

Parameter

width : padded हुए string के साथ string की length दी जाती है |
fillChar : Optional. यहाँ पर padded character होता है | अगर character दिया नहीं जाता तो space( ) default होता है |

Returning Value

Padded किये गए string को return किया जाता है |

Source Code :

```
str = "Hello World"
print(str.center(20))
print(str.center(20, "$"))
```

Output :

```
    Hello World
$$$$Hello World$$$$$
```

casefold() String Method

casefold() function; normal string को casefold string में convert करता है |
casefolded string का इस्तेमाल caseless matching के लिए किया जा सकता है |
casefold() function का मकसद normal string में से case distinction को निकलना होता है |

Syntax

```
str.casefold()
```

Parameter

casefold() function के लिए कोई parameter नहीं होता है |

casefold function और lower() function एक जैसे होते है |

Returning Value

ये function casefolded string को return करता है |

Source Code :

```
str1 = "Hello World"
case = str1.casefold()
str2 = "Hello World"
str3 = "hello world"

print(str1 == str2) #True
print(str1 == case) #False
print(str3 == case) #True
```

Output :

```
True
False
True
```

count() String Method

count() function मुख्य string में से substring के occurrences; number में return किये जाते है |

Syntax

```
str.count(substr, start, end)
```

Parameter

substr : ये substring है | जिसे मुख्य string में search किया जायेगा |

start : Optional. किस character से search करना है वहा पर वो number दिया जाता है | अगर दिया नहीं जाता तो default string की शुरुआत होती है |

end : Optional. किस character तक search करना है वहा पर वो number दिया जाता है | अगर दिया नहीं जाता तो default string की length(len(str)) होती है |

Returning Value

start से लेकर end तक मुख्य string में कितने substring occurrences हुए है उनका number return किया जाता है |

start और end ये parameters दिए नहीं जाते है तो पूरे string में से substring के occurrences number में return किये जाते है |

Source Code :

```
str = "Hello World"
substr = "l"
print(str.count("l"))
```

Output :

```
3
```

Another Example for count() String Function

Source Code :

```
str = "Hello World"
print(str.count("l", 3))
print(str.count("l", 3, 4))
```

Output :

```
2
1
```

endswith() String Method

endswith() function में दिए गए suffix को string के end पर check करके boolean value return करता है |

Syntax

```
str.endswith(suffix, start, end)
```

Parameter

suffix : यहाँ पर suffix या tuple में suffix दिया जाता है |

start : Optional. ये शुरुआती position होती है | जहा से string में suffix को check किया जाता है |

end : Optional. ये अंतिम position होती है | जहा तक string में suffix को check किया जाता है |

endswith() function; case sensitive होता है |

Returning Value

अगर string के end पर suffix मिलता है तो 'True' return होता है और अगर नहीं मिलता है तो 'False' return होता है |

Source Code :

```
str = "Hello World"
sufTup = ("Hello", "World")
print(str.endswith(sufTup)) #True
suf = "World"
print(str.endswith(suf)) #True
suf = "world"
print(str.endswith(suf)) #False endswith() is case-sensitive
suf = "o"
print(str.endswith(suf, 1, 5)) #True
print(str.endswith(suf, 1, 6)) #False
```

Output :

```
True
True
False
True
False
```

expandtabs() String Method

expandtabs() function में String में tab(s) की size को expand करके string की copy return की जाती है |

Syntax

```
str.expandtabs(tabSize)
```

Parameter

tabSize : Optional. यहाँ पर tab(\t) की size की संख्या दी जाती है | अगर दिया नहीं जाता तो '8' ये default tabSize होती है |

Returning Value

Original String में tab के size के साथ उसकी copy return की जाती है |

Source Code :

```
str = "Hello\tWorld\tHello\tFriends"
print(str.expandtabs()) #Default tabsize is 8
print(str.expandtabs(2))
print(str.expandtabs(4))
print(str.expandtabs(6))
```

Output :

```
Hello   World   Hello   Friends
Hello World Hello Friends
Hello   World   Hello   Friends
Hello World Hello Friends
```

find() String Method

find() function में दिए गए substring को मुख्य string में ढूंढकर उसका पहला index return किया जाता है |

Syntax

```
str.find(subStr, start, end)
```

Parameter

subStr : यहाँ पर substring होता है | जिसे मुख्य string में ढूंढा जायेगा |

start : यहाँ पर starting index होती है | अगर दिया नहीं जाता '0' default value होती है |

end : यहाँ पर ending index होती है | अगर दिया नहीं जाता string की length(len(str)) default होती है |

Returning Value

यहाँ पर ढूंढा गया substring का पहला index return किया जाता है अगर substring ढूंढा नहीं जाता है तो '-1' return होता है |

find() function; rfind() function के विरुद्ध होता है |

String में find() function और index() function एक जैसे होते है | find() substring found नहीं होता है तो '-1' return करता है और index() substring found नहीं होता है तो valueError exception दे देता है |

Source Code :

```
str = "Hello World"
print(str.find("o"))
print(str.find("o", 6))
print(str.find("o", 2, 5))
```

Output :

```
4
7
4
```

format() String Method

format() function का इस्तेमाल string formatting के लिए किया जाता है |
Python मे format() function; string formatting के लिए काफी उपयुक्त function है |

Syntax

```
template.format(p0, p1, ..., pN, k0=v0, k1=v1, ..., kN=vN)
```

Parameter

template : ये एक format string होता है, जिसमे एक से ज्यादा format codes({}) होते है | Format codes ये output में replace होनेवाली जगह होती है |

p0, p1, ..., pN : यहाँ पर ये positional parameters है | print() function में दिए गए placeholder({index}) की जगह format() पे दिए positional parameter से replace किया जाता है |

k0=v0, k1=v1, ..., kN=vN : यहाँ पर ये keyword parameters है | keyword parameter में key और value(key=value) इन दोनों की pairs होती है |print() function में दिए गए placeholder({key}) की जगह format() पे दिए keyword parameter से replace किया जाता है |

format() function ये formatted string को return करता है |

Returning Value

Original String में tab के size के साथ उसकी copy return की जाती है |

More About Positional Parameters

format() function को Example और उससे related example को समझे |

Source Code :

```
print("{} got {:.2f} percentages in {}".format("Rakesh", 89.5875, "B.E."))
```

Output :

```
Rakesh got 89.59 percentages in B.E.
```

format() function के parameters से ज्यादा placeholders({}) नहीं लिए जा सकते है | अगर लिए जाते है तो 'indexError' exception आ जाता है |

For Example

```
print("{} got {:.2f} percentages in {}".format("Rakesh", 89.5875))
```

Output :

```
    print("{} got {:.2f} percentages in {}".format("Rakesh", 89.5875))
IndexError: tuple index out of range
```

format() function के parameters Placeholders({}) से ज्यादा लिए जा सकते है |

For Example

```
print("{} got {:.2f} percentages".format("Rakesh", 89.5875, "B.E."))
```

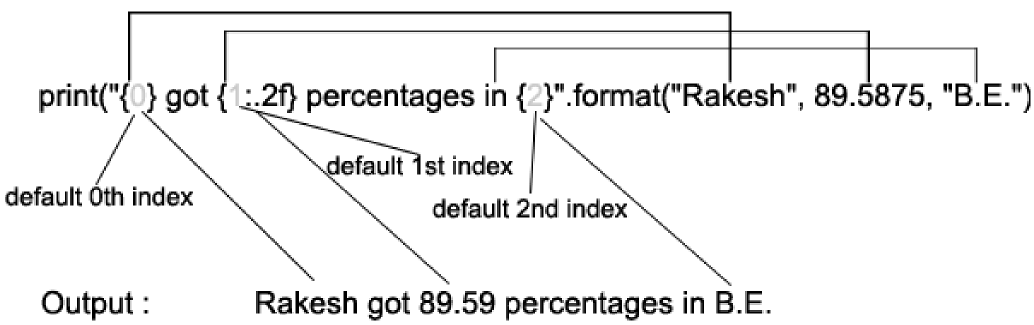
Output :

```
Rakesh got 89.59 percentages
```

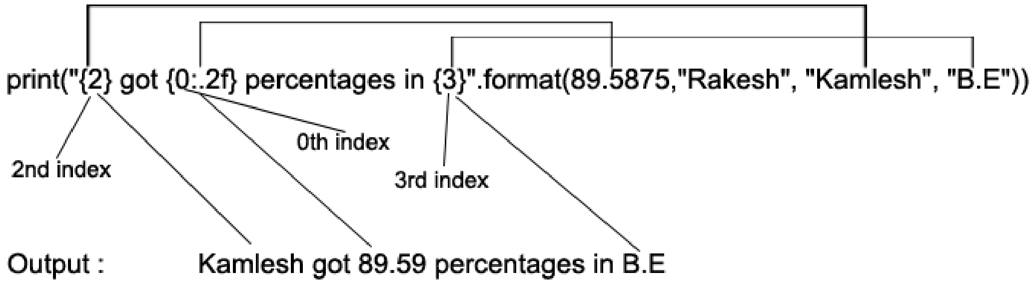
Know more About Placeholders and format() Function(Positional Parameters)

format() function को Example और उससे related example को समझे |

Placeholder में जब index दिया नहीं जाता है तो by default 0, 1, 2, 3 इस प्रकार से index interpreter द्वारा लिए जाते है | निचे image में देखके समझ आएगा |



Programmer चाहे तो index को बदल भी सकता है |




```
print("{2} got {0:.2f} percentages in {3}".format(89.5875,"Rakesh", "Kamlesh", "B.E"))
#Output : Kamlesh got 89.59 percentages in B.E
```

More About Keyword Parameters

Keyword Parameters में key और value(key=value) की pairs होती है | Placeholders में key(index) के जरिये उनकी value को access किया जाता है |

Source Code :

```
print("{studName} got {per:.2f} percentages in {deg}".format(studName="Rakesh",
per=89.5875, deg="B.E"))
```

Output :

```
Rakesh got 89.59 percentages in B.E
```

index() String Method

index() function में दिए गए substring को मुख्य string में ढूंढकर उसका पहला index return किया जाता है |

Syntax

```
str.index(subStr, start, end)
```

Parameter

subStr : यहाँ पर substring होता है | जिसे मुख्य string में ढूंढा जायेगा |

start : यहाँ पर starting index होती है | अगर दिया नहीं जाता '0' default value होती है |

end : यहाँ पर ending index होती है | अगर दिया नहीं जाता string की length(len(str)) default होती है |

Returning Value

यहाँ पर ढूंढा गया substring का पहला index return किया जाता है अगर substring ढूंढा नहीं जाता है तो valueError exception आ जाता है |
String में find() function और index() function एक जैसे होते है |
String में find() function और index() function एक जैसे होते है | find() substring found नहीं होता है तो '-1' return करता है और index() substring found नहीं होता है तो valueError exception दे देता है |

Source Code :

```
str = "Hello World"
print(str.index("o"))
print(str.index("o", 6))
print(str.index("o", 2, 5))
```

Output :

```
4
7
4
```

isalnum() String Method

isalnum() function में अगर character या string alphanumeric या alphabetic या numeric होता है तो true return करता है अगर नहीं होते है तो false return होता है |

Syntax

```
str.isalnum()
```

Parameter

isalnum() function के लिए parameter नहीं होता है |

Returning Value

अगर character या string alphanumeric या alphabetic या numeric होता है तो true return करता है अगर नहीं होते है तो false return होता है |

Source Code :

```
print("1o".isalnum()) #True
print("o1".isalnum()) #True
print("Hello123".isalnum()) #True
print("123Hello".isalnum()) #True
print("Hello".isalnum()) #True
print("123".isalnum()) #True
print("1".isalnum()) #True
print("@".isalnum()) #False
print("@Hello".isalnum()) #False
print(" ".isalnum()) #False
```

Output :

```
True
True
True
True
True
```

```
True
True
False
False
False
```

isalpha() String Method

isalpha() function में अगर character या string alphabetic होता है तो true return करता है अगर नहीं होते है तो false return होता है |

Syntax

```
str.isalpha()
```

Parameter

isalpha() function के लिए parameter नहीं होता है |

Returning Value

अगर character या string alphabetic होता है तो true return करता है अगर नहीं होते है तो false return होता है |

Source Code :

```
print("1o".isalpha()) #False
print("o1".isalpha()) #False
print("Hello123".isalpha()) #False
print("123Hello".isalpha()) #False
print("Hello".isalpha()) #True
print("123".isalpha()) #False
print("1".isalpha()) #False
print("@".isalpha()) #False
print("@Hello".isalpha()) #False
print(" ".isalpha()) #False
```

Output :

```
False
False
False
False
True
False
False
False
False
False
```

isdecimal() String Method

isdecimal() function में अगर character या string decimal होता है तो true return करता है अगर नहीं होते है तो false return होता है |

Syntax

```
str.isdecimal()
```

Parameter

isdecimal() function के लिए parameter नहीं होता है |

Returning Value

अगर character या string decimal होता है तो true return करता है अगर नहीं होते है तो false return होता है |

Source Code :

```
print("1".isdecimal()) #True
print("10".isdecimal()) #True
print("0".isdecimal()) #True
print("123Hello".isdecimal()) #False
```

Output :

```
True
True
True
False
```

isdigit() String Method

isdigit() function में अगर character या string digit होता है तो true return करता है अगर नहीं होते है तो false return होता है |

Syntax

```
str.isdigit()
```

Parameter

isdigit() function के लिए parameter नहीं होता है |

Returning Value

अगर character या string digit होता है तो true return करता है अगर नहीं होते है तो false return होता है |

isdecimal() function और isdigit() function एक जैसे होते है |

Source Code :

```
print("1".isdigit()) #True
print("10".isdigit()) #True
print("0".isdigit()) #True
print("123Hello".isdigit()) #False
```

Output :

```
True
True
True
False
```

isidentifier() String Method

isidentifier() function में दिए गए string या character एक valid identifier है या नहीं ये boolean value में return किया जाता है |

Syntax

```
str.isidentifier()
```

Parameter

isidentifier() function के लिए parameter नहीं होता है |

Returning Value

अगर valid identifier होता है तो true return करता है अगर नहीं होते है तो false return होता है |

identifier का मतलब एक valid variable name होता है |

Source Code :

```
print("l".isidentifier()) #True
print("Hello123".isidentifier()) #True
print("123Hello".isidentifier()) #False
print("123".isidentifier()) #False
print("_".isidentifier()) #True
print("_Hello".isidentifier()) #True
```

Output :

```
True
True
False
False
True
True
```

islower() String Method

islower() function में दिए गए string या character lowercase में है या नहीं ये check करके boolean value में return किया जाता है |

Syntax

```
str.islower()
```

Parameter

islower() function के लिए parameter नहीं होता है |

Returning Value

अगर दिया गया string या character lowercase में होता है तो true return करता है अगर नहीं होते है तो false return होता है |

Source Code :

```
print("Hello".islower()) #False
print("hello".islower()) #True
print("hello123".islower()) #True
print("123hello".islower()) #True
print("@hello".islower()) #True
```

Output :

```
False
True
True
True
True
```

isnumeric() String Method

isnumeric() function में दिए गए string या character numeric में है या नहीं ये check करके boolean value में return किया जाता है |

Syntax

```
str.isnumeric()
```

Parameter

isnumeric() function के लिए parameter नहीं होता है |

Returning Value

अगर दिया गया string या character numeric में होता है तो true return करता है अगर नहीं होते है तो false return होता है |

Source Code :

```
print("123".isnumeric()) #True
print("0".isnumeric()) #True
print("Hello".isnumeric()) #False
print("hello".isnumeric()) #False
print("hello123".isnumeric()) #False
```

Output :

```
True
True
False
False
False
```

isprintable() String Method

isprintable() function में दिए गए string या character printable है या नहीं ये check करके boolean value में return किया जाता है |Syntax for isprintable() String Function

Syntax

```
str.isprintable()
```

Parameter

isprintable() function के लिए parameter नहीं होता है |

Returning Value

अगर दिया गया string या character printable होता है तो true return करता है अगर नहीं होता है तो false return होता है |

Source Code :

```
print("123".isprintable()) #True
print("0".isprintable()) #True
print("Hello".isprintable()) #True
print("hello".isprintable()) #True
print("hello123".isprintable()) #True
print("\n".isprintable()) #False
```

Output :

```
True
True
True
True
True
False
```

isspace() String Method

isspace() function में दिए गए string या character सिर्फ space है या नहीं ये check करके boolean value में return किया जाता है |

Syntax

```
str.isspace()
```

Parameter

isspace() function के लिए parameter नहीं होता है |

Returning Value

अगर दिया गया string या character में सिर्फ space होता है तो true return करता है अगर नहीं होता है तो false return होता है |

Source Code :

```
print("Hello World".isspace()) #False
print(" Hello123".isspace()) #False
print("@Hello ".isspace()) #False
print("\t".isspace()) #True
print(" ".isspace()) #True
print("       ".isspace()) #True
```

Output :

```
False
False
False
True
True
True
```

istitle() String Method

istitle() function में दिए गया string; title string है या नहीं ये check करके boolean value में return किया जाता है |

Syntax

```
str.istitle()
```

Parameter

istitle() function के लिए parameter नहीं होता है |

Returning Value

अगर दिया गया string title string होता है तो true return करता है अगर नहीं होता है तो false return होता है |

Source Code :

```
print("Hello World".istitle()) #True
print("Hello world".istitle()) #False
print("hello world".istitle()) #False
print("hello 123".istitle()) #False
print("hello123".istitle()) #False
```

Output :

```
True
False
False
False
False
```

isupper() String Method

isupper() function में दिए गए string या character uppercase में है या नहीं ये check करके boolean value में return किया जाता है |

Syntax

```
str.isupper()
```

Parameter

isupper() function के लिए parameter नहीं होता है |

Returning Value

अगर दिया गया string या character uppercase में होता है तो true return करता है अगर नहीं होते है तो false return होता है |

Source Code :

```
print("Hello".isupper()) #False
print("hello".isupper()) #False
print("HELLO123".isupper()) #True
print("123HELLO".isupper()) #True
print("@HELLO".isupper()) #True
```

Output :

```
False
False
True
True
True
```

join() String Method

join() function में दिए गए elements के sequence को किसी seperator से join करके string को return किया जाता है |

Syntax

```
str.join()
```

Parameter

join() function के लिए parameter नहीं होता है |

Returning Value

sequence के हर एक element को किसी string से seperate करके और उसे cocatenate करके string को return किया जाता है |

Source Code :

```
sep = "$"
str = "Hello"
print(sep.join(str))
list = ["H", "e", "l", "l", "o"]
print(sep.join(list))
tuple = ("H", "e", "l", "l", "o")
print(sep.join(tuple))
```

Output :

```
H$e$l$l$o
H$e$l$l$o
H$e$l$l$o
```

ljust() String Method

ljust() function में string को left में justfy करके और width के हिसाब से दिए गए character से fill करके string को return किया जाता है |

Syntax

```
str.ljust(width, fillChar)
```

Parameter

width : यहाँ पर return किये जानेवाले string की length होती है | अगर original string से कम length दी जाती है तो original string return किया जाता है |

fillChar : Optional. original string से ज्यादा width दी जाती है तो बाकी space पे इस character से fill किया जाता है | अगर दिया नहीं जाता तो default space( ) होता है |

Returning Value

string को left में justfy करके और width के हिसाब से दिए गए character से fill करके string को return करता है |

Source Code :

```
str = "Hello World"
print(str.ljust(20, "!"))
print(str.ljust(20))
```

Output :

```
Hello World!!!!!!!!!
Hello World
```

rjust() String Method

rjust() function में string को right में justfy करके और width के हिसाब से दिए गए character से fill करके string को return किया जाता है |

Syntax

```
str.rjust(width, fillChar)
```

Parameter

width : यहाँ पर return किये जानेवाले string की length होती है | अगर original string से कम length दी जाती है तो original string return किया जाता है |

fillChar : Optional. original string से ज्यादा width दी जाती है तो बाकी space पे इस character से fill किया जाता है | अगर दिया नहीं जाता तो default space( ) होता है |

Returning Value

string को right में justfy करके और width के हिसाब से दिए गए character से fill करके string को return करता है |

Source Code :

```
str = "Hello World"
print(str.rjust(20, "!"))
print(str.rjust(20))
```

Output :

```
!!!!!!!!!Hello World
         Hello World
```

lower() String Method

lower() function में Uppercase के character या string को lowercase में convert किया जाता है |

Syntax

```
str.lower()
```

Parameter

lower() function के लिए कोई parameter नहीं होता है |

Returning Value

Lowercase में convert किये गए string को return किया जाता है |

Source Code :

```
print("Hello World".lower())
print("HELLO WORLD".lower())
print("HELLO123".lower())
print("123HELLO".lower())
```

Output :

```
hello world
hello world
hello123
123hello
```

upper() String Method

upper() function में lowercase के character या string को Uppercase में convert किया जाता है |

Syntax

```
str.upper()
```

Parameter

upper() function के लिए कोई parameter नहीं होता है |

Returning Value

Uppercase में convert किये गए string को return किया जाता है |

Source Code :

```
print("Hello World".upper())
print("Hello world".upper())
print("hello123".upper())
print("123hello".upper())
```

Output :

```
HELLO WORLD
HELLO WORLD
HELLO123
123HELLO
```

swapcase() String Method

swapcase() function में अगर string के characters; uppercase हो तो उसे lowercase में return करता है और lowercase हो तो उसे uppercase में return करता है |

Syntax

```
str.swapcase()
```

Parameter

swapcase() function के लिए कोई parameter नहीं होता है |

Returning Value

ये swapcased string को return करता है |

Source Code :

```
print("Hello World".swapcase())
print("Hello world".swapcase())
print("hello123".swapcase())
print("123hello".swapcase())
```

Output :

```
hELLO wORLD
hELLO WORLD
HELLO123
123HELLO
```

lstrip() String Method

lstrip() function में left side(leading character) से दिए गए character को strip करके string की copy return की जाती है |

Syntax

```
str.lstrip(char)
```

Parameter

char : Optional. String में से जिस character(s) को strip करना है वो character(s) यहाँ पर दिए जाते है |

Returning Value

String के left side(leading character) से दिए गए character को strip करके strip की copy return की जाती है | अगर strip नहीं होता है तो 'original string' return किया जाता है |

Source Code :

```
str = "Hello World"
print("       Hello World".lstrip())
print(str.lstrip("Hel"))
print(str.lstrip("e"))
```

Output :

```
Hello World
o World
Hello World
```

rstrip() String Method

rstrip() function में right side(trailing characters) से दिए गए character को strip करके string की copy return की जाती है |

Syntax

```
str.rstrip(char)
```

Parameter

char : String में से जिस character(s) को strip करना है वो character(s) यहाँ पर दिए जाते है |

Returning Value

String के right side(trailing characters) से दिए गए character को strip करके strip की copy return की जाती है | अगर strip नहीं होता है तो 'original string' return किया जाता है |

Source Code :

```
str = "Hello World"
print("Hello World       ".rstrip())
print(str.rstrip("ld"))
print(str.rstrip("l"))
```

Output :

```
Hello World
Hello Wor
Hello World
```

strip() String Method

strip() function में left side(leading characters) और right side(trailing character) से दिए गए character को strip करके string की copy return की जाती है |

Syntax

```
str.strip(char)
```

Parameter

char : Optional. String में से जिस character(s) को strip करना है वो character(s) यहाँ पर दिए जाते है |

Returning Value

String के left side(leading characters) और right side(trailing character) से दिए गए character को strip करके string की copy return की जाती है |

Source Code :

```
str1 = "   Hello World   "
print(str1.strip())
print(str1.strip("He"))

str2 = "Hii Rakesh Hii"
print(str2.strip("Hii"))

print(str2.strip("ii"))

str3 = "Hello World"
print(str3.strip("re"))
```

Output :

```
Hello World
   Hello World
 Rakesh
Hii Rakesh H
Hello World
```

partition() String Method

partition() function में दिए गए String के लिए seperator का first occurrence को tuple के बीच में रख के tuple को return किया जाता है |

Syntax

```
str.partition(sep)
```

Parameter

sep : यहाँ पर ये seperator होता है | इस seperator की मदद से string के तीन हिस्से tuple में return किये जाते है |

Returning Value

Return हुए Tuple में String के 3 elements return होते है |

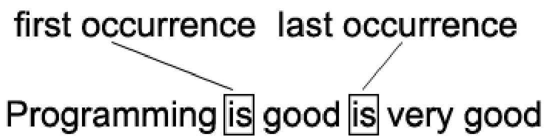
पहला हिस्सा : अगर seperator found नहीं होता है तो पूरा string यहाँ पर आता है और बाकी के दो blank ही रह जाते है |
अगर String के पहले ही seperator found होता है तो ये blank होता है | नीचेवाला example देखे |

दूसरा हिस्सा : अगर seperator found नहीं होता है तो blank रह जाता है |
अगर String के पहले ही seperator found होता है तो seperator वाला हिस्सा यहाँ पर आता है | नीचेवाला example देखे |

तीसरा हिस्सा : अगर separator found नहीं होता है तो blank रह जाता है |
अगर String के पहले ही separator होता है तो separator को छोडके बचा हुआ हिस्सा यहाँ पर आता है | नीचेवाला example देखे |

partition() function case-sensitive होता है |

अगर String में separator found होता है तो वो 3 elements वाले tuple के बीच के element पर होता है |



Source Code :

```
string = "Programming is good"

print(string.partition('computer'))
#computer is not found
#Output : ('Programming is good', ", ")

print(string.partition('pro'))
#pro is found but partition() is case-sensitive
#Output : ('Programming is good', ", ")

print(string.partition('Pro'))
#Pro is found
#Output : (", 'Pro', 'gramming is good')

print(string.partition('is'))
#'is' is found
#Output : ('Programming ', 'is', ' good')

print(string.partition('good'))
#'good' is found
#Output : ('Programming is ', 'good', ")
```

Output :

```
('Programming is good', ", ")
('Programming is good', ", ")
(", 'Pro', 'gramming is good')
('Programming ', 'is', ' good')
('Programming is ', 'good', ")
```

rpartition() String Method

rpartition() function में दिए गए String के लिए seperator का last occurrence को tuple के बीच में रख के tuple को return किया जाता है |

Syntax

```
str.rpartition(sep)
```

Parameter

sep : यहाँ पर ये seperator होता है | इस seperator की मदद से string के तीन हिस्से tuple में return किये जाते है |

Returning Value

Return हुए Tuple में String के 3 elements return होते है |

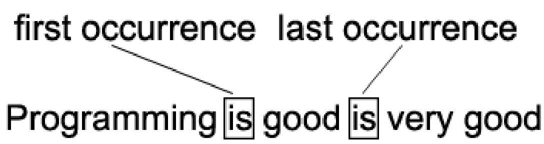
पहला हिस्सा : अगर separator found नहीं होता है तो ये blank रह जाता है | अगर separator शुरुआत में found होता है तो, separator को छोडके बचा हुआ हिस्सा यहाँ पर आ जाता है |

दूसरा हिस्सा : अगर separator found नहीं होता है तो ये blank रह जाता है | अगर separator शुरुआत में found होता है तो, seperator वाला हिस्सा यहाँ पर आ जाता है |

तीसरा हिस्सा : अगर separator found नहीं होता है तो यहाँ पर पूरा string आ जाता है | अगर seperator शुरुआत में found होता है तो, separator को छोडके बचा हुआ हिस्सा यहाँ पर आ जाता है |

rpartition() function case-sensitive होता है |

अगर String में separator found होता है तो वो 3 elements वाले tuple के बीच के element पर होता है |



Source Code :

```
string = "Programming is good"

print(string.rpartition('computer'))
#computer is not found
#Output : ('', '', 'Programming is good')

print(string.rpartition('pro'))
#pro is found but rpartition() is case-sensitive
#Output : ('', '', 'Programming is good')

print(string.rpartition('Pro'))
#Pro is found
#Output : ('', 'Pro', 'gramming is good')

print(string.rpartition('is'))
#'is' is found
#Output : ('Programming ', 'is', ' good')

print(string.rpartition('good'))
#'good' is found
#Output : ('Programming is ', 'good', '')
```

Output :

```
('', '', 'Programming is good')
('', '', 'Programming is good')
('', 'Pro', 'gramming is good')
('Programming ', 'is', ' good')
('Programming is ', 'good', '')
```

replace() String Method

replace() function में String के लिए दिए गए old substring के सभी occurrence नए substring से replace करके string की copy return की जाती है |

Syntax

```
str.replace(oldsubStr, newsubStr, max)
```

Parameter

oldsubStr : String में से जिस substring को replace करना है है वो substring यहाँ पर दिया जाता है |

newsubStr : Replace किये गए substring की जगह कौनसा substring दिया जाये, वो string यहाँ पर आता है |

max : Optional. ज्यादा से ज्यादा string के शुरुआत से कितने replace करने है वो number यहाँ पर दिया जाता है |

Returning Value

old substring के सभी occurrence नए substring से replace करके string की copy return की जाती है |

Source Code :

```
str = "Hello World Hello Ramesh Hello Friends"

print(str.replace("ello", "ii"))
#Output : Hii World Hii Ramesh Hii Friends

print(str.replace("ello", "ii", 2))
#Output : Hii World Hii Ramesh Hello Friends
```

Output :

```
Hii World Hii Ramesh Hii Friends
Hii World Hii Ramesh Hello Friends
```

rfind() String Method

rfind() function में दिए गए substring को मुख्य string में ढूंढकर उसका आखिरी index return किया जाता है |

Syntax

```
str.rfind(subStr, start, end)
```

Parameter

subStr : यहाँ पर substring होता है | जिसे मुख्य string में ढूंढा जायेगा |

start : यहाँ पर starting index होती है | अगर दिया नहीं जाता '0' default value होती है |

end : यहाँ पर ending index होती है | अगर दिया नहीं जाता string की length(len(str)) default होती है |

Returning Value

यहाँ पर ढूंढा गया substring का आखिरी index return किया जाता है अगर substring ढूंढा नहीं जाता है तो '-1' return होता है |

rfind() function; find() function के विरुद्ध होता है |

String में rfind() function और rindex() function एक जैसे होते है | rfind() substring found नहीं होता है तो '-1' return करता है और rindex() substring found नहीं होता है तो valueError exception दे देता है |

Source Code :

```
str = "Hello World"

print(str.rfind("l"))
print(str.rfind("l", 6))
print(str.rfind("l", 2, -3))
```

Output :

```
9
9
3
```

rindex() String Method

rindex() function में दिए गए substring को मुख्य string में ढूंढकर उसका आखिरी index return किया जाता है |

Syntax

```
str.rindex(subStr, start, end)
```

Parameter

subStr : यहाँ पर substring होता है | जिसे मुख्य string में ढूंढा जायेगा |

start : यहाँ पर starting index होती है | अगर दिया नहीं जाता '0' default value होती है |

end : यहाँ पर ending index होती है | अगर दिया नहीं जाता string की length(len(str)) default होती है |

Returning Value

यहाँ पर ढूंढा गया substring का आखिरी index return किया जाता है अगर substring ढूंढा नहीं जाता है तो valueError exception आ जाता है |

String में rfind() function और rindex() function एक जैसे होते है | rfind() substring found नहीं होता है तो '-1' return करता है और rindex() substring found नहीं होता है तो valueError exception दे देता है |

Source Code :

```
str = "Hello World"

print(str.rindex("l"))
print(str.rindex("l", 6))
print(str.rindex("l", 2, -3))
```

Output :

```
9
9
3
```

split() String Method

split() function में दिए गए seperator से string को split करके list(sequence) में return किया जाता है |

Syntax

```
str.split(sep, max)
```

Parameter

sep : Optional. यहाँ पर string; split करने के लिए seperator दिया जाता है | यहाँ पर default value 'space'( ) होता है |

max : Optional. यहाँ पर जितने split चाहिए उससे +1 splits list में return किये जाते है | यहाँ पर default value '-1' होता है |

Source Code :

```
str = "Hello World Hello Ramesh";
print (str.split(" ", 2))
str = "Hello, World, Hello, Ramesh";
print (str.split(",", 2))
str = "Hello, World, Hello, Ramesh";
print (str.split("e", 3))
str = "Hello, World, Hello, Ramesh";
print (str.split("l", 3))
```

Output :

```
['Hello', 'World', 'Hello Ramesh']
['Hello', ' World', ' Hello, Ramesh']
['H', 'llo, World, H', 'llo, Ram', 'sh']
['He', '', 'o, Wor', 'd, Hello, Ramesh']
```

startswith() String Method

startswith() function में दिए गए prefix को string के end पर check करके boolean value return करता है |

Syntax

```
str.startswith(prefix, start, end)
```

Parameter

prefix : यहाँ पर prefix या tuple में prefix दिया जाता है |

start : Optional. ये शुरुआती position होती है | जहा से string में prefix को check किया जाता है |

end : Optional. ये अंतिम position होती है | जहा तक string में prefix को check किया जाता है |

startswith() function; case sensitive होता है |

Returning Value

अगर string के start पर prefix मिलता है तो 'True' return होता है और अगर नहीं मिलता है तो 'False' return होता है |

Source Code :

```
str = "Hello World"
preTup = ("Hello", "World")
print(str.startswith(preTup)) #True
pre = "He"
print(str.startswith(pre)) #True
pre = "hello"
print(str.startswith(pre)) #False startswith() is case-sensitive
pre = "e"
print(str.startswith(pre, 1, 5)) #True
print(str.startswith(pre, 2, 6)) #False
```

Output :

```
True
True
False
True
False
```

title() String Method

title() function में Normal string को title string में convert करके return करता है |

Syntax

```
str.title()
```

Parameter

title() function के लिए कोई parameter नहीं होता है |

Returning Value

ये function string के हर word का पहला character uppercase में convert करके return करता है |

Source Code :

```
str1 = "hello world"
print(str1.title())
str2 = "123hello"
print(str2.title())
str3 = "@hello"
print(str3.title())
```

Output :

```
Hello World
123Hello
@Hello
```

zfill() String Method

zfill() function में अगर string से ज्यादा length(width) दी जाती है तो left side से अतिरिक्त जगह पर '0' दिया जाता है और string की copy return की जाती है |

Syntax

```
str.zfill(width)
```

Parameter

width : यहाँ पर return की जानेवाली string की length दी जाती है | अगर original string से कम width दी जाती है तो original string return किया जाता है |

Returning Value

अतिरिक्त width पर left side से '0' को padded करके string की copy return की जाती है |

Source Code :

```
str = "Hello World"
print(str.zfill(20))
print(str.zfill(30))
print(str.zfill(5))
```

Output :

```
000000000Hello World
0000000000000000000Hello World
Hello World
```

len() String Method

len() function में string की length return की जाती है |

Syntax

```
len(str)
```

Parameter

str : यहाँ पर जिस string की length return करनी है वो string दिया जाता है |

Returning Value

यहाँ पर string की length number में return की जाती है |

Source Code :

```
str = "Hello World"
print(len(str))
```

Output :

```
11
```

max() String Method

max() function में string में से max character को return किया जाता है |

Syntax

```
max(str)
```

Parameter

str : जिस string में से max character को return करना है वो string यहाँ पर दिया जाता है |

Returning Value

यहाँ पर max character को return किया जाता है |

min() और max() function में ascii value के हिसाब से character को return किया जाता है |

Source Code :

```
str = "Hello World"
print(max(str))
```

Output :

```
r
```

min() String Method

min() function में string में से max character को return किया जाता है |

Syntax

```
min(str)
```

Parameter

str : जिस string में से min character को return करना है वो string यहाँ पर दिया जाता है |

Returning Value

यहाँ पर min character को return किया जाता है |

min() और max() function में ascii value के हिसाब से character को return किया जाता है |

Source Code :

```
str = "Hello World"
print(min(str))
```

Output :

```

```

Date and Time Module Introduction

हर एक Programming Language में Date और Time का concept होता है |
Python में भी Date और Time का concept होता है |
Date और time के लिए कुछ modules को import करना पड़ता है | वैसे ही कुछ Modules नीचे दिए गए है |

- time Module
- datetime Module
- calendar Module

time : time module पे बहुत सारे date और time से related functions, classes और constants होते है |
datetime : datetime module में भी constants, classes; date और time से related होते है | ये module date और time के साथ काम करने के लिए Object-Oriented Programming में बनाया गया है |
calendar : Calendar module में calendar से related कुछ functios और classes होते है |

Ticks/Timestamp in Python

12.00am January 1, 1970 से अबतक के बीते हुए seconds को 'Ticks' कहते है |
Python में 12.00am January 1, 1970 से अबतक के बीते हुए seconds को return करने के लिए time module में 'time()' ये function होता है | ये time() function बीते हुए seconds कोfloating-point number में return करता है |

For Example

```
import time
print("Timestamp/ticks :",time.time())

#Output :
#Timestamp : 1497628217.2675047
```

What is epoch in Python?

epoch ये हमेशा 12.00am January 1, 1970 होता है | Python में time module के 'gmtime()' इस function से epoch को return किया जाता है |

Source code :

```
import time
print("epoch :",time.gmtime(0))
```

Output :

```
epoch :
time.struct_time(
tm_year=1970,
tm_mon=1,
tm_mday=1,
tm_hour=0,
tm_min=0,
tm_sec=0,
tm_wday=3,
tm_yday=1,
tm_isdst=0)
```

Time Module

time module में date और time से related कुछ functions और constants होते है | time module में कुछ classes C या C++ में बने है | निचे दिए हुए example में gmtime() function से C या C++ में बने हुए 'struct_time' इस object को return किया गया है |

Source code :

```
import time
print(time.localtime())
```

Output :

```
time.struct_time(
tm_year=2017,
tm_mon=6,
tm_mday=16,
tm_hour=17,
tm_min=4,
tm_sec=32,
tm_wday=4,
tm_yday=167,
tm_isdst=0)
```

struct_time structure में कुछ attributes होते है |

Index	Attribute	Values
0	tm_year	Year in 4 Digit(for eg, 2017)
1	tm_mon	1 to 12(Months)
2	tm_mday	1 to 28,29,30,31(Days)
3	tm_hour	0 to 23(Hours)
4	tm_min	0 to 59(Minutes)
5	tm_sec	0 to 61 (60 or 61 are leap-seconds)
6	tm_wday	0 to 6(Weekday)
7	tm_yday	1 to 366(Yearday)
8	tm_isdst	DST(Daylight Saving Time) 0, if not DST 1, if DST, -1, if not known about DST

How to Get Current Date and Time using time Module ?

Example पर localtime() द्वारा current time को display किया गया है लेकिन इस return हुए time को आसानी से पढ़ा नहीं जा सकता है |

Source code :

```
import time
print("Current Time :",time.localtime())
```

Output :

```
Current Time : time.struct_time(tm_year=2017, tm_mon=6, tm_mday=17,
tm_hour=23, tm_min=16, tm_sec=7, tm_wday=5, tm_yday=168, tm_isdst=0)
```

Getting Current Date and Time in readable Format using time Module

Readable Format में current time को return करना हो तो 'asctime()' function का इस्तेमाल किया जाता है |

Source code :

```
import time
print(time.asctime())
```

Output :

```
Sat Jun 17 23:38:53 2017
```

Important Attributes of 'time' Module

Attributes	**Description**
altzone	ये local DST timezone का offset; seconds में return करता है |
daylight	ये local DST timezone का offset; seconds में return करता है |
timezone	ये local DST timezone का offset; seconds में return करता है |
tzname	local(non-DST) timezone और local DST timezone के साथ या उसके बिना tuple में return होते है |

altzone time Module Attribute

altzone इस time module attribute से local DST timezone का offset; seconds में return करता है | अगर daylight nonzero होता है तो इसका इस्तेमाल किया जाता है |

Syntax

```
time.altzone
```

Returning Value

altzone ये attribute seconds में local DST timezone का offset return करता है |

Source Code :

```
import time
print(time.altzone)
```

Output :

```
-23400
```

daylight time Module Attribute

daylight इस time module attribute से अगर DST(Daylight Saving Time) timezone defined होता है तो nonzero value return होती है |

Syntax

```
time.daylight
```

Returning Value

daylight ये attribute में अगर DST timezone defined होता है तो nonzero value return होती है |

Source Code :

```
import time
print(time.daylight)
```

Output :

```
0
```

timezone time Module Attribute

timezone इस time module attribute ये local(non-DST) timezone का offset seconds में return करता है |

Syntax

```
time.timezone
```

Returning Value

timezone ये attribute ये seconds में local(non-DST) timezone का offset return करता है |

Source Code :

```
import time
print(time.timezone)
```

Output :

```
-19800
```

tzname time Module Attribute

tzname ये time module attribute; local(non-DST) timezone और local DST timezone के साथ या उसके बिना tuple में return होते है |

Syntax

```
time.tzname
```

Returning Value

tzname ये attribute ये local(non-DST) timezone और local DST timezone के साथ या उसके बिना tuple में return होते है |
Tuple के पहले item में local(non-DST) timezone का string होता है |
Tuple के दूसरे item में local DST timezone का string होता है या नहीं होता है |

Source Code :

```
import time
print(time.tzname)
print(time.tzname[0])
print(time.tzname[1])
```

Output :

```
('India Standard Time', 'India Daylight Time')
India Standard Time
India Daylight Time
```

Important Functions of 'time' Module

Attributes	Description
asctime()	दिए हुए time tuple को या gmtime() या localtime() से return होनेवाले struct_time को string में convert करके return करता है ।
ctime()	epoch local time(Jan 1 05:30:00 1970) और दिए गए seconds उसमे add करके उस time को string में return किया जाता है ।
gmtime()	epoch GMT time(Jan 1 00:00:00 1970) और दिए गए seconds उसमे add करके उस time को struct_time में return किया जाता है ।
localtime()	epoch local time(Jan 1 05:30:00 1970) और दिए गए seconds उसमे add करके उस time को struct_time में return किया जाता है ।
mktime()	दिए हुए time-tuple या struct_time को floating-point value को seconds में return करता है ।
sleep()	इस function का इस्तेमाल calling thread को suspend करने के लिए किया जाता है ।
strftime()	time-tuple या 'gmtime() या localtime()' से return होनेवाले 'struct_time' और दिए हुए format से 'time' को return किया जाता है ।
strptime()	दिए हुए format के अनुसार string को parse करके 'struct_time' में return किया जाता है ।
time()	epoch GMT time(Jan 1, 1970, 00:00:00) से लेकर बीते हुए time को seconds(floating-point number) मे return करता है ।

asctime() time Module Function

asctime() इस time module function दिए हुए time tuple को या gmtime() या localtime() से return होनेवाले struct_time को string में convert करके return करता है |

Syntax

```
time.asctime(tuple/struct_time)
```

Parameter

tuple/struct_time : Optional. यहाँ पर time-tuple या 'gmtime() या localtime() function' से return होनेवाले stuct_time को दिया जाता है |

Returning Value

asctime() ये function दिए गए time-tuple या 'gmtime() या localtime() function; द्वारा return होनेवाले struct_time को string में return करता है |

Example for asctime() time Module Function without parameter

asctime() function पर अगर parameter दिया नहीं जाता है तो current time 'Sun Jun 18 19:20:28 2017' इस format में string में return होता है |

Source Code :

```
import time
print(time.asctime())
```

Output :

```
Sun Jun 18 19:20:28 2017
```

Example for asctime() time Module Function with 'time-tuple' parameter

time-tuple में 9 parameter या tuple के items होते है | निचे tuple के items क्रमनुसार दिया गया है |

(Year, month, day, hour, minute, second, wday, yday, isdst)

asctime() function पर दिए गए time-tuple पर दिये हुए time को string में convert करके return करता है |

Source Code :

```
import time

tupleTime = (1993, 12, 31 , 23 , 59, 59, 6, 365, 0)
#(Year, month, day, hour, minute, second, wday, yday, isdst)

print(time.asctime(tupleTime))
```

Output :

```
Sun Dec 31 23:59:59 1993
```

Example for asctime() time Module Function with 'gmtime()' parameter

gmtime() ये function current GMT Time को struct_time में return करता है |

Example में asctime() function में localtime() इस function को parameter के रूप में लिया गया है |

Source Code :

```
import time
print(time.asctime(time.localtime()))
```

Output :

```
Sun Jun 18 19:56:18 2017
```

ctime() time Module Function

ctime() इस time module function; epoch local time(Jan 1 05:30:00 1970) और दिए गए seconds उसमे add करके उस time को string में return किया जाता है |

Syntax

```
time.ctime(sec)
```

Parameter

sec : यहाँ पर seconds दिए जाते है | जिसे epoch time पर add किया जाता है | यहाँ पर negative value नहीं दी जा सकती |

ctime() और localtime() ये दोनों एक जैसे functions है लेकिन ctime() time को readable string में return करता है और localtime() time को 'struct_time' में return करता है |

Returning Value

ctime() ये function epoch local time(Jan 1 05:30:00 1970) और दिए गए seconds उसमे add करके उस time को string में return किया जाता है | अगर seconds दिए नहीं जाते तो current local time; string में return होता है |

Example for ctime() time Module Function without parameter

अगर parameter दिया नहीं जाता है तो current time return होता है |

Source Code :

```
import time
print (time.ctime())
```

Output :

```
Sun Jun 18 21:17:18 2017
```

Example for ctime() time Module Function with 'seconds' parameter

अगर 'sec' parameter दिया जाता है तो दिए हुए seconds और epoch local time पर add करके उस time को string में return किया जाता है |

Source Code :

```
import time
print (time.ctime(46))
```

Output :

```
Thu Jan  1 05:30:46 1970
```

gmtime() time Module Function

gmtime() इस time module function; epoch GMT time(Jan 1 00:00:00 1970) और दिए गए seconds उसमे add करके उस time को struct_time में return किया जाता है |

Syntax

```
time.gmtime(sec)
```

Parameter

sec : यहाँ पर seconds दिए जाते है | जिसे epoch GMT time पर add किया जाता है | यहाँ पर negative value नहीं दी जा सकती |

Returning Value

gmtime() ये function epoch GMT time(Jan 1 00:00:00 1970) और दिए गए seconds उसमे add करके उस time को struct_time में return किया जाता है | अगर seconds दिए नहीं जाते तो current GMT time; struct_time में return होता है |

Example for gmtime() time Module Function without parameter

अगर parameter दिया नहीं जाता है तो current GMT time; struct_time में return होता है |

Source Code :

```
import time
print (time.gmtime())
```

Output :

```
time.struct_time(tm_year=2017, tm_mon=6, tm_mday=18, tm_hour=16, tm_min=4,
tm_sec=4, tm_wday=6, tm_yday=169, tm_isdst=0)
```

Example for gmtime() time Module Function with 'seconds' parameter

अगर 'sec' parameter दिया जाता है तो दिए हुए seconds और epoch GMT time पर add करके उस time को struct_time में return किया जाता है |

Source Code :

```
import time
print (time.gmtime(46))
```

Output :

```
time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=0, tm_min=0,
tm_sec=46, tm_wday=3, tm_yday=1, tm_isdst=0)
```

Example for Convert struct_time to string

अगर struct_time को string(readable format) में convert करना हो तो 'asctime()' function का इस्तेमाल किया जाता है |

Source Code :

```
import time
print (time.asctime(time.gmtime(46)))
```

Output :

```
Thu Jan  1 00:00:46 1970
```

localtime() time Module Function

localtime() इस time module function; epoch local time(Jan 1 05:30:00 1970) और दिए गए seconds उसमे add करके उस time को struct_time में return किया जाता है |

Syntax

```
time.localtime(sec)
```

Parameter

sec : यहाँ पर seconds दिए जाते है | जिसे epoch time पर add किया जाता है | यहाँ पर negative value नहीं दी जा सकती |

ctime() और localtime() ये दोनों एक जैसे functions है लेकिन ctime() time को readable string में return करता है और localtime() time को 'struct_time' में return करता है |

Returning Value

localtime() ये function epoch local time(Jan 1 05:30:00 1970) और दिए गए seconds उसमे add करके उस time को struct_time में return किया जाता है | अगर seconds दिए नहीं जाते तो current local time; struct_time में return होता है |

Example for localtime() time Module Function without parameter

अगर parameter दिया नहीं जाता है तो current time 'struct_time' में return होता है |

Source Code :

```
import time
print (time.localtime())
```

Output :

```
time.struct_time(tm_year=2017, tm_mon=6, tm_mday=18, tm_hour=22,
tm_min=36, tm_sec=7, tm_wday=6, tm_yday=169, tm_isdst=0)
```

Example for localtime() time Module Function with 'seconds' parameter

अगर 'sec' parameter दिया जाता है तो दिए हुए seconds और epoch local time पर add करके उस time को struct_time में return किया जाता है |

Source Code :

```
import time
print (time.localtime(46))
```

Output :

```
time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=5, tm_min=30,
tm_sec=46, tm_wday=3, tm_yday=1, tm_isdst=0)
```

Example for Convert struct_time to string

अगर struct_time को string(readable format) में convert करना हो तो 'asctime()' function का इस्तेमाल किया जाता है |

Source Code :

```
import time
print (time.asctime(time.localtime(46)))
```

Output :

```
Thu Jan  1 05:30:46 1970
```

mktime() time Module Function

mktime() इस time module function; दिए हुए time-tuple या struct_time को floating-point value को seconds में return करता है |

Syntax

```
time.mktime(time-tuple/struct_time)
```

Parameter

time-tuple/struct_time : यहा पर time-tuple या struct_time दिया जाता है |

mktime() function; localtime() के बिलकुल उल्टा होता है |

Returning Value

mktime() ये function दिए हुए time-tuple या struct_time को floating-point value को seconds में return करता है | अगर दिए हुआ time valid नहीं होता है तो 'overflowError' या 'valueError' exception आ जाता है |

Source Code :

```
import time
timeTuple = (2015, 4, 24, 15, 3, 38, 1, 48, 0)

print (time.mktime(timeTuple))
#convert time-tuple/struct_time to seconds

print (time.localtime(time.mktime(timeTuple)))
#convert seconds to struct_time
```

Output :

```
1429868018.0
time.struct_time(tm_year=2015, tm_mon=4, tm_mday=24, tm_hour=15, tm_min=3,
tm_sec=38, tm_wday=4, tm_yday=114, tm_isdst=0)
```

sleep() time Module Function

sleep() इस time module function का इस्तेमाल calling thread को suspend करने के लिए किया जाता है |

Syntax

```
time.sleep(sec)
```

Parameter

sec : यहा पर calling thread को suspend करने के लिए कुछ seconds दिए जाते है |

Returning Value

sleep() ये function कुछ भी return नहीं करता है |

Source Code :

```
import time
print (time.ctime())
time.sleep(1)
print (time.ctime())
time.sleep(1)
print (time.ctime())
time.sleep(1.50)
print (time.ctime())
```

Output :

```
Sun Jun 18 23:42:48 2017
Sun Jun 18 23:42:49 2017
Sun Jun 18 23:42:50 2017
Sun Jun 18 23:42:52 2017
```

strftime() time Module Function

strftime() ये time module function; पर time-tuple या 'gmtime() या localtime()' से return होनेवाले 'struct_time' और दिए हुए format से 'time' को return किया जाता है |

Syntax

```
time.strftime(format, time-tuple/struct_time)
```

Parameter

format : ये format string होता है | यहाँ पर एक या एक से ज्यादा 'directives' दिए जाते है |

time-tuple/struct_time : Optional. यहाँ पर time-tuple या 'gmtime() या localtime()' से return होनेवाले 'struct_time' को दिया जाता है | अगर दिया नहीं जाता तो current time को दिए हुए format से display किया जाता है |

Returning Value

strftime() ये function time-tuple या 'gmtime() या localtime()' से return होनेवाले 'struct_time' और दिए हुए format से 'time' को return किया जाता है |

Source Code :

```
import time
print (time.ctime())
time.sleep(1)
print (time.ctime())
time.sleep(1)
print (time.ctime())
time.sleep(1.50)
print (time.ctime())
```

Output :

```
Sun Jun 18 23:42:48 2017
Sun Jun 18 23:42:49 2017
Sun Jun 18 23:42:50 2017
Sun Jun 18 23:42:52 2017
```

Directives for Format String

Directive	**Description**	**Example**
%a	abbreviated weekday name	Tue
%A	full weekday name	Tuesday
%b	abbreviated month name	Apr
%B	full month name	April
%c	date and time representation	Tue Apr 24 15:03:38 2015
%d	day of the month(1 to 31)	24
%e	day of th month(1 to 31) same as %d	24
%g	two digit Year	15
%G	four digit year	2015
%h	abbreviated month name. same as %b	Apr

%H	hour(24-hour clock)(00 to 23)	15
%I	hour(12-hour clock)(01 to 12)	03
%j	day of the year(001 to 366)	048
%m	month in number(01 to 12)	04
%M	minute in number(00 to 59)	03
%p	AM or PM according to time	PM
%R	Only time without second(24-hour clock), format HH:MM	15:03
%S	second in number(00 to 61)	38
%U	week number of the year(00 to 53)(sunday is the first day of the week)	07
%w	weekday in number(0 to 6)(sunday is the first day of the week)	2
%W	week number of the year(00 to 53)(monday is the first day of the week)	07
%x	Only date, format(mm/dd/yy)	04/24/15
%X	Only time with second(24-hour clock), format HH:MM:SS	15:03:38
%y	two digit year	15
%Y	four digit year	2015
%z	timezone offset, format -HH:MM to +HH:MM	+0530
%Z	timezone name	India Standard Time
%%	for '%' character	%

Source Code :

```
import time

timeTuple = (2015, 4, 24, 15, 3, 38, 1, 48, 0)
print (time.strftime("%H:%M:%S %Y %b %d"))
print (time.strftime("%H:%M:%S %Y %b %d", timeTuple))
```

Output :

```
09:42:55 2017 Jun 19
15:03:38 2015 Apr 24
```

strptime() time Module Function

strptime() ये time module function; दिए हुए format के अनुसार string को parse करके 'struct_time' में return किया जाता है |

Syntax

```
time.strptime(timeStr, format)
```

Parameter

timeStr : यहाँ पर time string दी जाती है | जिसे दिए हुए format के अनुसार parse किया जाता है |

format : यहाँ पर एक या एक से अधिक directives दिये जाते है |

अगर format के अनुसार string parse नहीं होता है या format से अधिक string parse की जाती है तो 'valueError' exception आ जाता है |

Returning Value

strptime() ये function दिए हुए format के अनुसार string को parse करके 'struct_time' में return किया जाता है |

Source Code :

```
import time
print (time.strptime("24 Jun 1995", "%d %b %Y"))
```

Output :

```
time.struct_time(tm_year=1995, tm_mon=6, tm_mday=24, tm_hour=0, tm_min=0,
tm_sec=0, tm_wday=5, tm_yday=175, tm_isdst=-1)
```

time() time Module Function

time() ये time module function; epoch GMT time(Jan 1, 1970, 00:00:00) से लेकर बीते हुए time को seconds(floating-point number) मे return करता है |

Syntax

```
time.time()
```

Parameter

time() ये function के लिए कोई parameter नहीं होता है |

Returning Value

time() ये function में epoch GMT time(Jan 1, 1970, 00:00:00) से लेकर बीते हुए time को seconds(floating-point number) मे return करता है |

Source Code :

```
import time
print (time.time())
print (time.gmtime(time.time()))
print (time.asctime(time.gmtime(time.time())))
```

Output :

```
1497851308.0017974
time.struct_time(tm_year=2017, tm_mon=6, tm_mday=19, tm_hour=5, tm_min=48,
tm_sec=28, tm_wday=0, tm_yday=170, tm_isdst=0)
Mon Jun 19 05:48:28 2017
```

Python में numbers पर mathematical operations करने के लिए math functions का इस्तेमाल किया जाता है | Python के program में अगर math functions का इस्तेमाल करना हो तो 'math' module का इस्तेमाल किया जाता है |

Math Constants in Python

Important 'math' Constants

Math Constant	Description
e	Math की 'E' Property Euler का number return करता है |
inf	infinity को return करता है |
nan	not a number(nan) का वर्णन करता है |
pi	pi की value को return करता है |
tau	tau(τ) की value को return करता है |

e math constant

e ये math constant; Math की 'E' Property Euler का number return करता है | ये Number John Napier के Natural Logarithm(loge) का base होता है |

e लगभग 2.718 होता है |

Syntax

```
math.e
```

Returning Value

e math constant e की value return की जाती है |

Source Code :

```
import math
print(math.e)
```

Output :

```
2.718281828459045
```

inf math constant

inf ये math constant; infinity को return करता है |

Syntax

```
math.inf
```

Returning Value

inf math constant; infinity को return करता है |

Source Code :

```
import math
print(math.inf)
print(-math.inf)
```

Output :

```
inf
-inf
```

nan math constant

nan ये math constant; not a number(nan) का वर्णन करता है |

Syntax

```
math.nan
```

Returning Value

nan math constant; not a number(nan) return करता है |

Source Code :

```
import math
print(math.nan)
```

Output :

```
nan
```

pi math constant

pi ये math constant; pi की value को return करता है |

Syntax

```
math.pi
```

Returning Value

pi math constant; pi की value को return करता है |

Source Code :

```
import math
print(math.pi)
```

Output :

```
3.141592653589793
```

nan math constant

tau ये math constant; tau(τ) की value को return करता है |

tau ये circle का 2π होता है |

Syntax

```
math.tau
```

Returning Value

tau math constant; tau की value को return करता है |

Source Code :

```
import math
print(math.tau)
print(2*math.pi)
```

Output :

```
6.283185307179586
6.283185307179586
```

Math Functions in Python

Python में numbers पर mathematical operations करने के लिए math functions का इस्तेमाल किया जाता है |

Python के program में अगर math functions का इस्तेमाल करना हो तो 'math' module का इस्तेमाल किया जाता है |

Math Functions in Python

Math Function	Description
acos()	दिए हुए number का arc cosine; radians में return करता है |
acosh()	दिए हुए number का inverse hyperbolic cosine; को return करता है |
asin()	दिए हुए number का arc sine; radians में return करता है |
asinh()	दिए हुए number का inverse hyperbolic sine; को return करता है |
atan()	दिए हुए number का arc tangent; radians में return करता है |
atan2()	atan(y/x) को radians में return करता है |
atanh()	दिए हुए number का inverse hyperbolic tangent; को return करता है |
ceil()	दिए हुए number की ceiling value को return करता है |
copysign()	दिए हुए x की value पर y के sign को copy करके x को return करता है |
cos()	दिए हुए number का cosine; radians में return करता है |
cosh()	दिए हुए number का hyperbolic cosine; return करता है |
degrees()	दिए हुए number के angle को radians से degrees में return करता है |
exp()	दिए हुए number का exponential return करता है |
expm1()	दिए हुए number का exponential से '-1' करके return करता है |
fabs()	दिए हुए number की absolute value को return करता है |
factorial()	दिए हुए number का factorial को return करता है |
floor()	दिए हुए number की floor value को return करता है |
fmod()	दिए हुए x को y से divide करके उनका remainder return करता है |
frexp()	दिए हुए number से manissa और exponent को (m, e) इस tuple के रूप में return करता है |
fsum()	दिए हुए sequence या collection के items का sum return करता है |

gcd()	दिए हुए x और y का gcd(Greatest Common Divisor) return करता है |
hypot()	दिए हुए x side और y side की मदद से hypotenuse(कर्ण) floating-point number में return करता है |
isclose()	दिए हुए x और y ये close है या नहीं ये boolean value में return करता है |
isfinite()	दिए हुआ number मर्यादित है या नहीं ये Boolean value में return करता है |
isinf()	दिए हुआ number अमर्यादित है या नहीं ये Boolean value में return करता है |
ldexp()	x * (2**i) floating-point number में return करता है |
log()	दिए हुए number का natural logarithm return करता है |
log10()	दिए हुए Number का base-10 logarithm return करता है |
log1p()	दिए हुए Number पर '+1' करके उसका natural logarithm return करता है |
log2()	दिए हुए Number का base-2 logarithm return करता है |
modf()	दिए हुए floating-point या integer number का fractional part और integer part को tuple में return करता है |
pow()	x**y return करता है |
radians()	दिए हुए number के angle को degrees से radians में return करता है |
sin()	दिए हुए number का sine; radians में return करता है |
sinh()	दिए हुए number का hyperbolic sine; return करता है |
sqrt()	दिए हुए number का square root return किया जाता है |
tan()	दिए हुए number का tangent; radians में return करता है |
tanh()	दिए हुए number का hyperbolic tangent; return करता है |
trunc()	दिए हुए number का अगर fraction part होता है तो उसे remove करके integer value को return किया जाता है |

acos() Math Function

acos() ये Math Function दिए हुए number का arc cosine; radians में return करता है |

Syntax

```
math.acos(x)
```

Parameter

x : यहाँ पर number दिया जाता है | ये number -1 से +1 के बीच का होता है |

अगर -1 से +1 के बाहर की value दी जाती है तो 'valueError' exception आता है |

Returning Value

acos() Math function; number का arc cosine; radians में return करता है |

Source Code :

```
import math
print(math.acos(0.80))
print(math.acos(1))
print(math.acos(-1))
```

Output :

```
0.6435011087932843
0.0
3.141592653589793
```

acosh() Math Function

acosh() ये Math Function दिए हुए number का inverse hyperbolic cosine; को return करता है |

Syntax

```
math.acosh(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

acosh() Math function; number का inverse hyperbolic cosine; को return करता है |

Source Code :

```
import math
print(math.acosh(45))
```

Output :

```
4.499686190671499
```

asin() Math Function

asin() ये Math Function दिए हुए number का arc sine; radians में return करता है |

Syntax

```
math.asin(x)
```

Parameter

x : यहाँ पर number दिया जाता है | ये number -1 से +1 के बीच का होता है |

अगर -1 से +1 के बाहर की value दी जाती है तो 'valueError' exception आता है |

Returning Value

acos() Math function; number का arc cosine; radians में return करता है |

Source Code :

```
import math
print(math.asin(0.90))
print(math.asin(1))
print(math.asin(-1))
```

Output :

```
1.1197695149986342
1.5707963267948966
-1.5707963267948966
```

asinh() Math Function

asinh() ये Math Function दिए हुए number का inverse hyperbolic sine; को return करता है |

Syntax

```
math.asinh(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

asinh() Math function; number का inverse hyperbolic sine; को return करता है |

Source Code :

```
import math
print(math.asinh(45))
```

Output :

```
4.49993310426429
```

atan() Math Function

atan() ये Math Function दिए हुए number का arc tangent; radians में return करता है |

Syntax

```
math.atan(x)
```

Parameter

x : यहाँ पर number दिया जाता है | ये number -1 से +1 के बीच का होता है |

अगर -1 से +1 के बाहर की value दी जाती है तो 'valueError' exception आता है |

Returning Value

atan() Math function; number का arc tangent; radians में return करता है |

Source Code :

```
import math
print(math.atan(0.90))
print(math.atan(1))
print(math.atan(-1))
print(math.atan(50))
```

Output :

```
0.7328151017865066
0.7853981633974483
-0.7853981633974483
1.550798992821746
```

atan2() Math Function

atan2() ये Math Function; atan(y/x) को radians में return करता है |

Syntax

```
math.atan2(y, x)
```

Parameter

y : यहाँ पर number दिया जाता है |

x : यहाँ पर number दिया जाता है |

Returning Value

atan2() Math function; atan(y/x) को radians में return करता है | return हुई value -pi और +pi के बीच की होती है |

Source Code :

```
import math
print(math.atan(1))
print(math.atan2(1, 1))
print(math.atan2(2, 2))
print(math.atan(2/2))
```

Output :

```
0.7853981633974483
0.7853981633974483
0.7853981633974483
0.7853981633974483
```

atanh() Math Function

atanh() ये Math Function दिए हुए number का inverse hyperbolic tangent; को return करता है |

Syntax

```
math.atanh(x)
```

Parameter

x : यहाँ पर number दिया जाता है | यहाँ पर -0.99 और +0.99 के बीच की value दी जाती है |

Returning Value

atanh() Math function; number का inverse hyperbolic tangent; को return करता है |

Source Code :

```
import math
print(math.atanh(-0.99))
print(math.atanh(0.99))
```

Output :

```
-2.6466524123622457
2.6466524123622457
```

ceil() Math Function

ceil() ये Math Function दिए हुए number की ceiling value को return करता है |

Syntax

```
math.ceil(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

floating-point number दिया जाता है तो उससे बड़ी integer value return की जाती है |

Returning Value

ceil() Math function; number की ceiling value को return करता है |

Source Code :

```
import math
print(math.ceil(4.49))
print(math.ceil(4.50))
print(math.ceil(4.51))
print(math.ceil(4.01))
print(math.ceil(-4.49))
print(math.ceil(-4.50))
print(math.ceil(-4.51))
```

Output :

```
5
5
5
5
-4
-4
-4
```

copysign() Math Function

copysign() ये Math Function दिए हुए x की value पर y के sign को copy करके x को return करता है |

Syntax

```
math.copysign(x, y)
```

Parameter

x : यहाँ पर number दिया जाता है |

y : यहाँ पर number दिया जाता है |

अगर copysign एक ही parameter का इस्तेमाल करता है तो ceil() और copysign() functions एक जैसे होते है |

अगर x पर y के sign की copy नहीं होती है तो x की absolute value return होती है | absolute value; positive होती है |

Returning Value

copysign() Math function; number की ceiling value को return करता है |

Source Code :

```
import math
print(math.copysign(4.49, -5.8))
print(math.copysign(-4.49, 5.8))
```

Output :

```
-4.49
4.49
```

cos() Math Function

cos() ये Math Function; दिए हुए number का cosine; radians में return करता है |

Syntax

```
math.cos(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

cos() Math function; number का cosine; radians में return करता है | return होने वाली value -1 और +1 के बीच की होती है |

Source Code :

```
import math
print(math.cos(45))
print(math.cos(-45))
```

Output :

```
0.5253219888177297
0.5253219888177297
```

cosh() Math Function

cosh() ये Math Function; दिए हुए number का hyperbolic cosine; return करता है |

Syntax

```
math.cosh(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

cosh() Math function; number का hyperbolic cosine; return करता है |

Source Code :

```
import math
print(math.cosh(30))
print(math.cosh(-30))
print(math.cosh(0.30))
```

Output :

```
5343237290762.231
-5343237290762.231
0.3045202934471426
```

degrees() Math Function

degrees() ये Math Function; दिए हुए number के angle को radians से degrees में return करता है |

Syntax

```
math.degrees(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

degrees() Math function; number के angle को radians से degrees में return करता है |

Source Code :

```
import math
pi = math.pi
print(math.degrees(pi/2))
print(math.degrees(pi/3))
print(math.degrees(pi/4))
print(math.degrees(pi))
print(math.degrees(pi*2))
```

Output :

```
90.0
59.99999999999999
45.0
180.0
360.0
```

exp() Math Function

exp() ये Math Function; दिए हुए number का exponential return करता है |

दिया हुआ number 'e' का घातांक होता है |

e की value '2.71828183' होती है |

$$e^x$$

Syntax

```
math.exp(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

exp() Math function; number का exponential return करता है |

Source Code :

```
import math
r = math.exp(10);
print("e^10 = ",r);
```

Output :

```
e^10 =  22026.465794806718
```

expm1() Math Function

expm1() ये Math Function; दिए हुए number का exponential से '-1' करके return करता है |

(exp(x))-1 और expm1(x) ये दोनों एक जैसे है |

e की value '2.71828183' होती है |

$$(e^{x}) - 1$$

Syntax

```
math.expm1(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

expm1() Math function; number का exponential इ 1 को घटाकर value को return करता है |

Source Code :

```
import math
e1 = math.exp(10)-1;
print("exp(10)-1 = ",e1);
e2 = math.expm1(10);
print("expm1(10) = ",e2);
```

Output :

```
exp(10)-1 =  22025.465794806718
expm1(10) =  22025.465794806718
```

fabs() Math Function

fabs() ये Math Function; दिए हुए number की absolute value को return करता है |

Syntax

```
math.fabs(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

fabs() Math function; number की absolute value को return करता है |

Source Code :

```
import math
print(math.fabs(-4.9))
print(math.fabs(4.9))
```

Output :

```
4.9
4.9
```

factorial() Math Function

factorial() ये Math Function; दिए हुए number का factorial को return करता है |

Syntax

```
math.factorial(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

factorial() Math function; number का factorial को return करता है | अगर दिया हुआ number floating-point number या negative number होता है तो 'valueError' exception आता है |

Source Code :

```
import math
print(math.factorial(5)) #5*4*3*2*1
print(math.factorial(6)) #6*5*4*3*2*1
```

Output :

```
120
720
```

floor() Math Function

floor() ये Math Function; दिए हुए number की floor value को return करता है |

Syntax

```
math.floor(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

floor() function में अगर floating-point value दी जाती है तो उससे छोटी value return की जाती है |

Returning Value

floor() Math function; number की floor value को return करता है |

Source Code :

```
import math
print("(0.49) :",math.floor(0.49))
print("(0.50) :",math.floor(0.50))
print("(0.51) :",math.floor(0.51))
print("(0.99) :",math.floor(0.99))
print("(-0.49) :",math.floor(-0.49))
print("(-0.50) :",math.floor(-0.50))
print("(-0.51) :",math.floor(-0.51))
print("(-0.99) :",math.floor(-0.99))
```

Output :

```
(0.49) : 0
(0.50) : 0
(0.51) : 0
(0.99) : 0
(-0.49) : -1
(-0.50) : -1
(-0.51) : -1
(-0.99) : -1
```

fmod() Math Function

fmod() ये Math Function; दिए हुए x को y से divide करके उनका remainder return करता है |

सामान्यतः fmod(x, y) और (x%y) दोनों एक जैसे होते है |

Syntax

```
math.fmod(x, y)
```

Parameter

x : यहाँ पर number दिया जाता है |

y : यहाँ पर number दिया जाता है |

Returning Value

fmod() Math function; दिए हुए x को y से divide करके उनका remainder floating-point number में return करता है |

अगर x और y की value 0 दी जाती है तो 'valueError' exception आता है |

अगर y की value 0 दी जाती है तो 'valueError' exception आता है |

Source Code :

```
import math
print(math.fmod(7, 4))
print(7%4)
print(math.fmod(0, 1))
```

Output :

```
3.0
3
1.0
```

frexp() Math Function

frexp() ये Math Function; दिए हुए number से manissa और exponent को (m, e) इस tuple के रूप में return करता है |

Syntax

```
math.frexp(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

frexp() Math function; number से manissa और exponent को (m, e) इस tuple के रूप में return करता है | tuple का पहला item floating-point number में mantissa होता है और दूसरा item integer number में exponent होता है |

Source Code :

```
import math
print(math.frexp(4.9))
print(math.frexp(0))
print(math.frexp(1))
```

Output :

```
(0.6125, 3)
(0.0, 0)
(0.5, 1)
```

fsum() Math Function

fsum() ये Math Function; दिए हुए sequence या collection के items का sum return करता है |

Syntax

```
math.fsum(seq)
```

Parameter

seq : यहाँ पर sequence(list, tuple) या collection(dictionary, set) होता है |

Returning Value

fsum() Math function; दिए हुए sequence या collection के items का sum return करता है |

Source Code :

```
import math
list = [5, -9.5, 4, 3]
print(math.fsum(list))
tuple = (4, -8, 5, 7)
print(math.fsum(tuple))
set = {5, -9, 7, 0.8}
print(math.fsum(set))
dict = {1:"H", 3:"e", 8:"l", 7.6:"l", -0.8:"o"}
print(math.fsum(dict))
```

Output :

```
2.5
8.0
3.8
18.8
```

gcd() Math Function

gcd() ये Math Function; दिए हुए x और y का gcd(Greatest Common Divisor) return करता है |

Syntax

```
math.gcd(x, y)
```

x और y पर float-point value दी नहीं जा सकती |

Parameter

x : यहाँ पर number दिया जाता है |

y : यहाँ पर number दिया जाता है |

Returning Value

gcd() Math function; दिए हुए x और y का gcd(Greatest Common Divisor) return करता है |

Source Code :

```
import math
print(math.gcd(3, 4))
print(math.gcd(8, 4))
print(math.gcd(16, 48))
```

Output :

```
1
4
16
```

hypot() Math Function

hypot() ये Math Function; दिए हुए x side और y side की मदद से hypotenuse(कर्ण) floating-point number में return करता है |

hypot() ये function 'Pythagoras Theorem' है |

Equation of Pythagoras Theorem

$$z^2 = x^2 + y^2$$

$$z^2 = 3^2 + 4^2$$

Syntax

```
math.hypot(x, y)
```

Parameter

x : यहाँ पर number दिया जाता है |

y : यहाँ पर number दिया जाता है |

Returning Value

hypot() Math function; x side और y side की मदद से hypotenuse(कर्ण) return करता है |

Source Code :

```
import math
print(math.hypot(3, 4))
print(math.hypot(4, 3))
print(math.hypot(4, 4))
```

Output :

```
5.0
5.0
5.656854249492381
```

isclose() Math Function

isclose() ये Math Function; दिए हुए x और y ये close है या नहीं ये boolean value में return करता है |

Syntax

```
math.isclose(x, y)
```

Parameter

x : यहाँ पर number दिया जाता है |

y : यहाँ पर number दिया जाता है |

Returning Value

isclose() Math function में अगर x और y close होते है तो 'True' return होता है और अगर x और y close नहीं होते है तो 'False' return होता है |

Source Code :

```
import math
print(math.isclose(3, 3.9))
print(math.isclose(3, 6/3))
print(math.isclose(3, 3.0))
print(math.isclose(3, 3))
```

Output :

```
False
False
True
True
```

isfinite() Math Function

isfinite() ये Math Function; दिए हुआ number मर्यादित है या नहीं ये Boolean value में return करता है |

Syntax

```
math.isfinite(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

isfinite() Math function में अगर number मर्यादित होता है तो 'True' return होता है और अगर number मर्यादित नहीं होता है तो 'False' return होता है |

Source Code :

```
import math
print(math.isfinite(0))
print(math.isfinite(math.inf))
print(math.isfinite(2))
```

Output :

```
True
False
True
```

isinf() Math Function

isinf() ये Math Function; दिए हुआ number अमर्यादित है या नहीं ये Boolean value में return करता है |

Syntax

```
math.isinf(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

isinf() Math function में अगर number अमर्यादित होता है तो 'True' return होता है और अगर number अमर्यादित नहीं होता है तो 'False' return होता है |

Source Code :

```
import math
print(math.isinf(0))
print(math.isinf(math.inf))
print(math.isinf(2))
```

Output :

```
False
True
False
```

ldexp() Math Function

ldexp() ये Math Function; x * (2**i) floating-point number में return करता है |

अगर x की value 9 है और i की value 8 है तो,

9 * (2 ** 8)

9 * 256

2304

Syntax

```
math.ldexp(x, i)
```

Parameter

x : यहाँ पर number दिया जाता है |

i : यहाँ पर number दिया जाता है |

Returning Value

ldexp() Math function x * (2**i) floating-point number return करता है |

Source Code :

```
import math
print(math.ldexp(9, 8))
#same as
print(9 * (2 ** 8))
```

Output :

```
2304.0
2304
```

log() Math Function

log() ये Math Function; दिए हुए number का natural logarithm return करता है |

Syntax

```
math.log(x, base)
```

Parameter

x : यहाँ पर number दिया जाता है | दिया जानेवाला number negative नहीं होता है |

base : Optional. यहाँ पर logarithm का base होता है |

$$\log_{base}x$$

Returning Value

log() Math function; number का natural logarithm return करता है |

Source Code :

```
import math
print(math.log(3.5, 3))
print(math.log(4, 3))
print(math.log(3.5))
print(math.log(4))
```

Output :

```
1.1403139955899648
1.2618595071429148
1.252762968495368
1.3862943611198906
```

log10() Math Function

log10() ये Math Function; दिए हुए Number का base-10 logarithm return करता है |

Syntax

```
math.log10(x)
```

Parameter

x : यहाँ पर number दिया जाता है | दिया जानेवाला number negative नहीं होता है |

$$\log_{10}x$$

Returning Value

log10() Math function; Number का base-10 logarithm return करता है |

Source Code :

```
import math
print(math.log10(3.5))
```

Output :

```
0.5440680443502757
```

log1p() Math Function

log1p() ये Math Function; दिए हुए Number पर '+1' करके उसका natural logarithm return करता है |

Syntax

```
math.log1p(x)
```

Parameter

x : यहाँ पर number दिया जाता है | दिया जानेवाला number negative नहीं होता है |

$$\log x + 1$$

Returning Value

log1p() Math function; Number पर '+1' करके उसका natural logarithm return करता है |

Source Code :

```
import math
print(math.log1p(3.5))
#same as
print(math.log(3.5+1))
```

Output :

```
1.5040773967762742
1.5040773967762742
```

log2() Math Function

log2() ये Math Function; दिए हुए Number का base-2 logarithm return करता है |

Syntax

```
math.log2(x)
```

Parameter

x : यहाँ पर number दिया जाता है | दिया जानेवाला number negative नहीं होता है |

$$\log_2 x$$

Returning Value

log2() Math function; Number का base-2 logarithm return करता है |

Source Code :

```
import math
print(math.log2(3.5))
```

Output :

```
1.8073549220576042
```

modf() Math Function

modf() ये Math Function; दिए हुए floating-point या integer number का fractional part और integer part को tuple में return करता है |

Syntax

```
math.modf(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

modf() Math function; दिए हुए floating-point या integer number का fractional part और integer part को tuple में return करता है |

Tuple का पहला item fractional part होता है और दूसरा item integer part होता है |

Source Code :

```
import math
print(math.modf(5.9))
print(math.modf(5))
```

Output :

```
(0.9000000000000004, 5.0)
(0.0, 5.0)
```

pow() Math Function

pow() ये Math Function; x**y return करता है |

y की value ये x की value का घातांक होता है |

x ** y

Syntax

```
math.pow(x, y)
```

Parameter

x : यहाँ पर number दिया जाता है |

y : यहाँ पर number दिया जाता है |

Returning Value

pow() Math function; x को y ये घातांक देकर floating-point value को return किया जाता है |

Source Code :

```
import math
print(math.pow(3, 2))
print(math.pow(5, 3))
```

Output :

```
9.0
125.0
```

radians() Math Function

radians() ये Math Function; दिए हुए number के angle को degrees से radians में return करता है |

Syntax

```
math.radians(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

radians() Math function; number के angle को degrees से radians में return करता है |

Source Code :

```
import math
pi = math.pi
print(math.radians(90))
print(math.radians(60))
print(math.radians(45))
print(math.radians(180))
print(math.radians(360))
```

Output :

```
1.5707963267948966
1.0471975511965976
0.7853981633974483
3.141592653589793
6.283185307179586
```

sin() Math Function

sin() ये Math Function; दिए हुए number का sine; radians में return करता है |

Syntax

```
math.sin(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

sin() Math function; number का sine; radians में return करता है | return होने वाली value -1 और +1 के बीच की होती है |

Source Code :

```
import math
print(math.sin(45))
print(math.sin(-45))
```

Output :

```
0.8509035245341184
-0.8509035245341184
```

sinh() Math Function

sinh() ये Math Function; दिए हुए number का hyperbolic sine; return करता है |

Syntax

```
math.sinh(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

sinh() Math function; number का hyperbolic sine; return करता है |

Source Code :

```
import math
print(math.sinh(30))
print(math.sinh(-30))
print(math.sinh(0.30))
```

Output :

```
5343237290762.231
-5343237290762.231
0.3045202934471426
```

sqrt() Math Function

sqrt() ये Math Function; दिए हुए number का square root return किया जाता है |

$\sqrt{16}$

Syntax

```
math.sqrt(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

sqrt() Math function; number का square root return किया जाता है |

Source Code :

```
import math
print(math.sqrt(4))
print(math.sqrt(5.6))
```

Output :

```
2.0
2.3664319132398464
```

tan() Math Function

tan() ये Math Function; दिए हुए number का tangent; radians में return करता है |

Syntax

```
math.tan(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

tan() Math function; number का tangent; radians में return करता है | return होने वाली value -1 और +1 के बीच की होती है |

Source Code :

```
import math
print(math.tan(45))
print(math.tan(-45))
```

Output :

```
1.6197751905438615
-1.6197751905438615
```

tanh() Math Function

tanh() ये Math Function; दिए हुए number का hyperbolic tangent; return करता है |

Syntax

```
math.tanh(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

tanh() Math function; number का hyperbolic tangent; return करता है |

Source Code :

```
import math
print(math.tanh(45))
print(math.tanh(-45))
```

Output :

```
1.0
-1.0
```

trunc() Math Function

trunc() ये Math Function; दिए हुए number का अगर fraction part होता है तो उसे remove करके integer value को return किया जाता है |

Syntax

```
math.trunc(x)
```

Parameter

x : यहाँ पर number दिया जाता है |

Returning Value

trunc() Math function; अगर fraction part होता है तो उसे remove करके integer value को return करता है |

Source Code :

```
import math
print(math.trunc(5.6))
print(math.trunc(9.5))
print(math.trunc(-9))
```

Output :

```
5
9
-9
```

Random Module and Functions

Python द्वारा random number(s) generate करने के लिए 'random' module का इस्तेमाल किया जाता है |
Random Number को generate करने के लिए कुछ functions होते है | वो निचे दिए गए है |

Important random Functions

Functions	**Description**
choice()	दिए हुए sequence(list, tuple) में से random number return करता है |
randint()	दिए हुए beg(inclusive) और end(inclusive) के बीच का random number return करता है |
random()	0 और 1 के बीच का random number; floating-point value में return करता है |
randrange()	दिए हुए range(beg to end) के बीच का random number return करता है |
sample()	दिए हुए sequence और length से random items को list में return किया जाता है |
seed()	इसका इस्तेमाल random number generator को initialize कने के लिए किया जाता है |
shuffle()	इसका इस्तेमाल sequence(list,tuple) के items की positions को randomly change करने के लिए किया जाता है |
uniform()	दिए हुए दो numbers के बीच से floating-point random value return करता है |

choice() Random Module Function

choice() random function दिए हुए sequence(list, tuple) में से random number return करता है |

Syntax

```
random.choice(seq)
```

Parameter

seq : यहाँ पर sequene दिया जाता है |

Returning Value

choice() random function दिए हुए sequence(list, tuple) में से random number return करता है | अगर sequence empty होता है तो 'indexError' raise होता है |

Source Code :

```
import random
print(random.choice((2,10,45,10,75)))
```

Output :

```
75
```

randint() Random Module Function

randint() random function; दिए हुए beg(inclusive) और end(inclusive) के बीच का random number return करता है |

Syntax

```
random.randint(beg, end)
```

Parameter

beg : Optional. यहाँ पर beginning value दी जाती है | जिसका समावेश random value में होता है |

end : यहाँ पर ending value दी जाती है | जिसका समावेश random value में होता है |

Returning Value

randint() random function; दिए हुए beg और end के बीच का integer random number return करता है | यहाँ पर beg के value का समावेश होता है और end के value का भी समावेश होता है |

Source Code :

```
import random
print(random.randint(1, 10)) #return 1|2|3|4|5|6|7|8|9|10
print(random.randint(8, 10)) #return 8|9|10
```

Output :

```
1
10
```

random() Random Module Function

random() random function; 0 और 1 के बीच का random number; floating-point value में return करता है |

Syntax

```
random.random()
```

Parameter

random() function के लिए कोई parameter नहीं होता है |

Returning Value

random() random function; 0 और 1 के बीच का random number; floating-point value में return करता है |

Source Code :

```
import random
print(random.random())
```

Output :

```
0.9898701210393435
```

randrange() Random Module Function

randrange() random function; दिए हुए range(beg to end) के बीच का random number return करता है |

Syntax

```
random.randrange(beg, end, skip)
```

Parameter

beg : Optional. यहाँ पर beginning value दी जाती है | जिसका समावेश random value में होता है |

end : यहाँ पर ending value दी जाती है | जिसका समावेश random value में नहीं होता है |

skip : Optional. जिस value को skip करना है वो value यहाँ पर दी जाती है | beginning पर दी हुई value को skip नहीं किया जा सकता |

Returning Value

randrange() random function; दिए हुए range(beg to end) के बीच का integer random number return करता है |

यहाँ पर beg के value का समावेश होता है और end के value का समावेश नहीं होता है |

Source Code :

```
import random
print(random.randrange(1, 10)) #return 1|2|3|4|5|6|7|8|9
print(random.randrange(8, 10, 8)) #return 8|9
print(random.randrange(8, 10, 9)) #return 8
```

Output :

```
5
8
9
```

sample() Random Module Function

sample() random function; दिए हुए sequence और length से random items को list में return किया जाता है |

Syntax

```
random.sample(seq, k)
```

Parameter

seq : यहाँ पर sequence(list, tuple) या collection(set) दिया जाता है |

k : यहाँ पर sequence या collection की length दी जाती है | यहाँ पर floating-point value नहीं दी जा सकती है |

Returning Value

sample() random function; दिए हुए sequence और length से items को list में return किया जाता है |

Source Code :

```
import random
list = {5,8,7,2,7,4}
print(random.sample(list, k=2))
```

Output :

```
[4, 7]
```

seed() Random Module Function

seed() random function; का इस्तेमाल random number generator को initialize कने के लिए किया जाता है |

Syntax

```
random.seed(x, ver)
```

Parameter

x : Optional. ये random number के लिए seed होता है | अगर दिया नहीं जाता है तो default value 'None' होती है |

ver : Optional. यहाँ पर version number होता है | अगर दिया नहीं जाता है तो default 2 होता है |

Returning Value

seed() random function; कोई value return नहीं करता है |

Source Code :

```
import random

random.seed()
print(random.random()) #return 0.7653484223501371

random.seed()
print(random.random()) #return 0.9984665569104412

random.seed(5)
print(random.random()) #return 0.6229016948897019

random.seed(5)
print(random.random()) #return same as above
```

Output :

```
0.7653484223501371
0.9984665569104412
0.6229016948897019
0.6229016948897019
```

shuffle() Random Module Function

shuffle() random function; का इस्तेमाल sequence(list,tuple) के items की positions को randomly change करने के लिए किया जाता है |

Syntax

```
random.shuffle(seq)
```

Parameter

seq : यहाँ पर sequence दिया जाता है |

Returning Value

shuffle() random function; 'None' return करता है |

shuffle() function का इस्तेमाल sequence(list,tuple) को shuffle करने के लिए किया जाता है |

Source Code :

```
import random
list = [1,8,9,5,7,1,7]
random.shuffle(list)
print(list)
```

Output :

```
[8, 1, 7, 7, 1, 5, 9]
```

uniform() Random Module Function

uniform() random function; दिए हुए दो numbers के बीच से floating-point random value return करता है |

Syntax

```
random.uniform(x, y)
```

Parameter

x : ये lower limit होता है | random number में इसका समावेश किया जाता है |

y : ये upper limit होता है | random number में इसका समावेश नहीं किया जाता है |

Returning Value

uniform() random function; दिए हुए दो numbers के बीच से floating-point random value return करता है |

Source Code :

```
import random
print(random.uniform(4, 9))
print(random.uniform(4.99, 9))
```

Output :

```
7.160481801336008
7.101289127835615
```

Modules

Python में कुछ inbuilt Modules होते है जैसे कि , math, random | हर module की अलग-अलग पहचान होती है | math module में mathematics से related कुछ functions होते है जैसे कि, sin(), cos() और sqrt() इत्यादी. Python में Modules ये एक तरह का program ही होता है | modules में function और classes होते है |

Python में दो प्रकार के Modules होते है |

1. In-built Modules
2. User-Defined Modules

1. In-built Modules

Python में 350+ Modules होते है | ये modules अलग-अलग काम के लिए बने होते है |
Python के सभी in-built modules देखने के लिए ,

```
help("modules")
```

2. User-Defined Modules

Creating Module in Python

निचे एक program दिया गया है | उसके file का नाम 'arithmetic.py' है | जो नाम file का होता है वही नाम 'module' का होता है | 'arithmetic' module में add, sub, mul और div नाम के functions दिए गए है |

File Name : arithmetic.py

```
def add(a, b):
    return a + b

def sub(c, d):
    return c - d

def mul(e, f):
    return e * f

def div(g, h):
    return g / h
```

Importing Module in Python

Python में current program पर module को import करना हो तो 'import' keyword करना पड़ता है | जिस module को import किया जाता है उसके सारे functions और classes; current program पर इस्तेमाल किये जा सकते है |

Syntax for import Statement

```
import module_name1, module_name2, ...,module_nameN
```

Example पर arithmetic ये user-defined module को import किया गया है और उसके सभी functions(add,sub,mul,div) का इस्तेमाल किया गया है |

अगर किसी भी module के function या class को current program पर access करना हो तो module_name के बाद dot(.) operator और उसके बाद function का या class का नाम देना पड़ता है |

Source Code :

```
import arithmetic

print(arithmetic.add(4, 6))
print(arithmetic.sub(4, 6))
print(arithmetic.mul(4, 6))
print(arithmetic.div(4, 6))
```

Output :

```
10
-2
24
0.6666666666666666
```

import_as Statement for Changing Module Name

Syntax

```
import module_name as alias_name
```

Module के नाम को change करना हो तो import_as statement का इस्तेमाल किया जाता है |

Example पर arithmetic module का नाम change करके myModule रख दिया गया है |

Source Code :

```
import arithmetic as myModule

print(myModule.add(4, 6))
print(myModule.sub(4, 6))
print(myModule.mul(4, 6))
print(myModule.div(4, 6))
```

Output :

```
10
-2
24
0.6666666666666666
```

from_import Statement

Syntax

```
from module_name import name1, name2,.., name2
```

अगर current program पर बिना module name से function को या class को access करना हो तो 'from_import' statement का इस्तेमाल किया जाता है |

Example पर arithmetic module के एक ही functions का इस्तेमाल किया गया है |

Source Code :

```
from arithmetic import add

print(add(4, 6))
```

Output :

```
10
```

from_import Statement with *(asterisk)

अगर current program पर बिना module के सभी functions या classes को access करना हो तो 'from_import' के साथ *(asterisk) का इस्तेमाल किया जाता है |

Syntax

```
from module_name import *
```

Source Code :

```
from arithmetic import *

print(add(4, 6))
print(sub(4, 6))
print(mul(4, 6))
print(div(4, 6))
```

Output :

```
10
-2
24
0.6666666666666666
```

Deleting imported Module

Module को current program से remove करना हो तो 'del' operator का इस्तेमाल किया जाता है |

Syntax

```
del module_name
```

Example पर arithmetic module को remove किया गया है |

Source Code :

```
import arithmetic

print(arithmetic.add(4, 6))
del arithmetic
print(arithmetic.mul(4, 6))
```

Output :

```
10
    print(arithmetic.mul(4, 6))
NameError: name 'arithmetic' is not defined
```

dir() Function

dir() function ये module में कौन-कौन से names defined किये गए है वो सभी inbuilt names ascending order में list के रूप में return करता है | Example पर add, div, mul और sub ये functions भी अलग से ascending order में return किये गए है |

Source Code :

```
import arithmetic
print(dir(arithmetic))
```

Output :

```
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'add', 'div', 'mul', 'sub']
```

Accessing Module's names(attributes)

जैसे module की मदद से function को access किया जाता है वैसे ही modules के names को access किया जाता है | names को access करने के लिए module_name.names(attributes) इस तरह से access किया जाता है |

Source Code :

```
import arithmetic
print(arithmetic.__file__)
print(arithmetic.__name__)
```

Output :

```
C:\Users\UD\AppData\Local\Programs\Python\Python36-32\arithmetic.py
arithmetic
```

reload() Function

Python में Interpreter द्वारा जब module के functions या class को execute किया जाता है तब उसके value को जबतक session शुरू है तबतक change नहीं किया जा सकता |

जैसे कि,

myModule.py

```
def func():
    print("Hello")
```

Output In Shell

```
>>> import myModule
>>> myModule.func()
Hello
```

After Changing value of Module's func() function

Python में session के दौरान सिर्फ एक बार ही module को import किया जा सकता है |
Example में func() function की value को change किया गया है लेकिन shell में जो पहला output था वही output display किया गया है |

myModule.py

```
def func():
    print("Hiiii")
```

Output In Shell

```
>>> import myModule
>>> myModule.func()
Hello
```

Reloading a Module using reload() function

reload() का इस्तेमाल module को refresh या reload करने के लिए किया जाता है | ये function module के session को restart कर देता है |

Example पर reload() function से 'myModule' इस module को reload किया गया है |

myModule.py

```
def func():
    print("Hiiii")
```

Output In Shell

```
>>> import myModule,importlib
>>> importlib.reload(myModule)

>>> myModule.hello()
Hiiii
```

Exceptions and Handling

Exceptions

Python में जब error occured होता है तब वहा पर कोई ना कोई exception raise होता है |

Exceptions ये error का ही एक प्रकार है | Python में कुछ built-in Exceptions होते है | एक execeptions के जरिये Python के errors को handle किया जाता है |

Normal Example for Exception

Example पर 'a' ये variable defined नहीं किया गया है | इसीलिए 'NameError' ये exception raise हुआ है |

Source Code :

```
print(a)
```

Output :

```
    print(a)
NameError: name 'a' is not defined
```

निचे कुछ महत्वपूर्ण Exceptions दिए गए है |

Important Exceptions in Python

Exceptions	**Description**
ArithmeticError	math या numeric calculations के सम्बंधित error का ये base class होता है |
AssertionError	assert statment जब नाकाम होता है तब ये exception raise होता है |
AttributeError	अगर attribute reference या assignment fail होता है तो ये exception raise होता है |
EOFError	जब 'input()' ये built-in function बिना data read किये जब end-of-file(EOF) इस condition पर पहुंचता है तब ये exception raise होता है |
EnvironmentError	जब Python Environment के बाहर से कुछ पाया जाता है तो ये उन सभी exceptions का base class होता है |
Exception	सभी exceptions का ये base class होता है |
FloatingPointError	floating-point calculation जब नाकाम होता तो ये exception raise होता है |
GeneratorExit	जब generator का close() method call किया जाता है तब ये exception raise होता है |

IOError	जब input or output operations नाकाम होते है तब ये exception raise होता है |
ImportError	जब import किया हुआ module नहीं मिलता तब ये exception raise होता है |
IndentationError	जब indentation गलत होता है तो तब ये exception raise होता है | ये SyntaxError का subclass होता है |
IndexError	जब index out of range दिया जाता है तब ये exception raise होता है |
KeyError	dictionary में जब key found नहीं होती है तब ये exception raise होता है |
KeyboardInterrupt	जब program execution के वक्त कुछ बाधा आती है तब ये exception raise होता है | ख़ास करके जब execution के वक्त Ctrl+c को दबाया जाता है |
LookupError	ये सभी lookup errors का base class होता है |
MemoryError	जब operation out of memory हो जाता है तब ये exception raise होता है |
NameError	जब local या global scope पर variable found नहीं होता है तब ये exception raise होता है |
NotImplementedError	ये exception abstract methods द्वारा raise होता है |
OSError	ये operating-system से सम्बंधित exception होता है |
OverflowError	जब numeric calculations हद से ज्यादा बड़े होते है तब ये exception raise होता है |
RuntimeError	जब error किसी भी category में नहीं होता है तो ये exception raise होता है |
StandardError	StopIteration और Systemexit के सिवाय ये सभी exceptions का base class होता है |
StopIteration	जब next() function के iterator किसी भी object का वर्णन नहीं करता है तब ये exception raise होता है |
SyntaxError	जब Python के syntax में error होता है तब ये exception raise होता है |
SystemError	जब interpreter द्वारा internal problem found होता है तब ये exception raise होता है |
SystemExit	जब sys.exit() द्वारा interpreter को बंद किया जाता है तब ये exception raise होता है |
TabError	जब indentation पर अतिरिक्त tabs और spaces दिए जाते है तब ये exception raise होता है |
TypeError	जब जरुरत के हिसाब से invalid data type की value दी जाती है तब ये exception raise होता है |
UnboundLocalError	अगर function के local variable या method को access किया जाता है और उनकी value वहापर assign नहीं होती है तो ये exception raise होता है |

UnicodeEncodeError	encoding के वक्त जब unicode से सम्बंधित error आता है तब ये exception raise होता है |
UnicodeError	unicode से सम्बंधित जब encoding या decoding error आता है तब ये exception raise होता है |
UnicodeTranslateError	translating के वक्त जब unicode से सम्बंधित error आता है तब ये exception raise होता है |
ValueError	जब in-built function पर valid data type देना जरुरी होता है लेकिन वहा पर valid data type की value नहीं दी जाती है तो ये exception raise होता है |
ZeroDivisonError	जब division का दूसरा operand '0' होता है तब ये exception raise होता है |

Exception Handling

What is Exception ?

Exceptions ये Python में Errors के प्रकार होते है | Exception के अलग-अलग प्रकार पड़ते है | हर अलग-अलग error के लिए अलग-अलग exception होता है |
जब Python के program पर error occurred होता है तब कोई ना कोई exception raise होता है |
जब program पर किसी statement के लिए exception raise होता है तो उस statement के आगे की script exception की वजह से execute नहीं होती है |
इन scripts को executes करने के लिए error/exception को handle करना पड़ता है |

For Example,

Example पर तीन variables को defined करके print किया गया है | लेकिन जहा पर print statements है वहा पर अतिरिक्त undefined 'd' variable को print करने की अनुमति दी गयी है |
'd' variable defined न होने की वजह से exception raise होता है | 'd' के print statement से पहले का print statement execute होता है लेकिन उसके बाद के statements print नहीं हो पाते है |
उन अगले statements को print करने के लिए exception को handle करना पड़ता है |

Source Code :

```
a = 10
b = 20
c = 30
print(a)
print(d) #exception raised
print(b)
print(c)
```

Output :

```
10
    print(d)
NameError: name 'd' is not defined
```

Example for Exception Handling

Example पर Exception को handle किया गया है |

Source Code :

```
a = 10
b = 20
c = 30

print(a)
try:
   print(d)
except(Exception):
   print("Variable is not defined")

print(b)
print(c)
```

Output :

```
10
Variable is not defined
20
30
```

try_except Statement(Exception Handling) in Python

Python में error को handle करना हो तो 'try_except' statement का इस्तेमाल करना पड़ता है | Python में जिस code पर error occurred होने की संभावना होती है वो code 'try' block पर आता है और except block पर exception/error को handle करने के लिए कुछ code दिया जाता है |

Syntax for 'try_except' Statement

```
try:
      try_block_code
except Exception1:
      exception handling code 1
except Exception2:
      exception handling code 2
```

Example for 'try_except' Statement

Example में try block पर जो code दिया है वो 'ZeroDivisionError' exception raise करता है इसीलिए ZeroDivisionError का ही except clause execute होगा |

Source Code :

```
try:
    a = 10
    a/0
except NameError:
    print("'a' is not defined")
except ZeroDivisionError:
          print("Divided By Zero Error")
```

Output :

```
Divided By Zero Error
```

'try_except' with 'else' Statement(Exception Handling) in Python

Syntax for 'try_except' Statement

```
try:
      try_block_code
except Exception1:
      exception handling code 1
except Exception2,Exception3:
      exception handling code 2
else:
      else_block_code
```

Syntax में देखे तो एक try block के लिए एक या एक से अधिक except clauses लिए जा सकते है |

जब try block code में दिया हुआ code कोई भी exception raise नहीं करता है तो else block code execute होता है |

Example में try block code कोई exception raised नहीं करता है इसीलिए else block code execute होगा |

Source Code :

```
try:
    a = 10
    a/4
except NameError:
    print("'a' is not defined")
except ZeroDivisionError:
    print("Divided By Zero Error")
else:
        print("No Exception raised")
```

Output :

```
No Exception raised
```

except clause with More than one Exception

Syntax

```
try:
     try_block_code
except Exception1:
     exception handling code 1
except (Exception2, Exception3):
     exception handling code 2
```

Syntax में देखे तो एक try block के लिए एक या एक से अधिक except clauses लिए जा सकते है और हर except clause में एक या एक से अधिक exceptions दिए जा सकते है | लेकिन उन multiple exceptions को parenthesis(()) में close करना पड़ता है |

For Example

```
try:
    b = a/4
    print("Value of b is",b)
except ZeroDivisionError:
    print("'a' is not defined")
except (TypeError,NameError):
    print("Something is wrong")
else:
        print("No Exception raised")
```

Output :

```
Something is wrong
```

try_except_finally Block(Exception Handling) in Python

Syntax

```
try:
    try_block_code
except Exception(s):
    exception_handling_code
finally:
    always_executed_code
```

try block का code जब कोई exception raised करता है और उसे handle नहीं किया जाता है तो finally block का code पहले execute होता है |
try block का code जब कोई exception raised करता है और उसे handle किया जाता है तो except block का code पहले execute होता है और बाद में finally block का code execute होता है |
अगर exception raised होता है या नहीं होता है finally block का code हमेशा execute होता है |

Source Code :

```
try:
   "5" + 4
except TypeError:
   print("Value must be a Number")
finally:
       print("Alway executed")
```

Output :

```
Value must be a Number
Alway executed
```

Raising Exceptions

Exception user द्वारा raise करने 'raise statement' का इस्तेमाल किया जाता है |
raise Statement program में कोई भी exception raised करने के लिए मजबूर कर देता है |

Syntax

```
raise exception('message') #argument is optional
```

Source Code :

```
raise TypeError("HI")
```

Output :

```
    raise TypeError("HI")
TypeError: HI
```

Another Example for Raising Exception

Source Code :

```
try:
    raise ValueError('ValueError raised forcely')
except Exception as e:
    print ('Error :',e)
```

Output :

```
Error : ValueError raised forcely
```

File Handling

File Handling Introduction

जब Computer पर कोई data लिखा जाता है तब वो पहले अपने RAM(Random Access Memory) पर store होता है और जब Computer को turn off किया जाता है तब वो सारा data lose हो जाता है |
और जब उस data को save किया जाता है तब वो Computer के Secondary Storage Device(Hard Drive) पर permanently store किया जाता है |

Python में File को handle करने के लिए कुछ महत्वपूर्ण operations निचे दिए गए है |

- Opening File
- Closing File
- Reading File
- Writing File
- Renaming File
- Deleting File

Opening a File

File के data को जब manipulate करना हो तो पहले file को open किया जाता है |
Python में file को open करने के लिए python के in-built 'open()' method का इस्तेमाल किया जाता है |

Syntax

```
fileObj = open(fileName, mode)
```

Parameter

open() method के लिए 8 parameters होते है लेकिन यहाँ पर सिर्फ 2 महत्वपूर्ण parameters दिए गए है |

fileName : यहाँ पर file name के साथ उसका extension भी दिया जाता है | अगर same location पर file हो तो उसे path देने की जरुरत नहीं होती है |

mode : यहाँ पर file को open करने के लिए mode दिया जाता है | अगर mode दिया नहीं जाता है तो default 'r'(reading) mode होता है |

Example

```
file = open('textfile.txt') #default mode is 'r'
#or
file = open('textfile.txt', 'r')
```

Modes for Opening a File

Modes	Description
a(append)	file को appending के लिए open किया जाता है | अगर file पहले से ही exist होती है तो बिना data remove किये वो end पर पहुँच जाता है और अगर file exist नहीं होती है तो new file create की जाती है |
ab(append in binary)	file को binary format में appending के लिए open किया जाता है | अगर file पहले से ही exist होती है तो बिना data remove किये वो end पर पहुँच जाता है और अगर file exist नहीं होती है तो new file create की जाती है |
a+(append and read)	file को appending और reading के लिए open किया जाता है | अगर file पहले से ही exist होती है तो बिना data remove किये वो end पर पहुँच जाता है और अगर file exist नहीं होती है तो new file create की जाती है |
ab+(append and read in binary)	file को binary format में appending और reading के लिए open किया जाता है | अगर file पहले से ही exist होती है तो बिना data remove किये वो end पर पहुँच जाता है और अगर file exist नहीं होती है तो new file create की जाती है |
r(read)	ये default होता है | file को reading के लिए open किया जाता है | ये file के beginning पर होता है |
rb(read in binary)	file को binary format में reading के लिए open किया जाता है | ये file के beginning पर होता है |
r+(read and write)	file को reading और writing के लिए open किया जाता है | ये file के beginning पर होता है |
rb+(read and write in binary)	file को binary format में reading और writing के लिए open किया जाता है | ये file के beginning पर होता है |
w(write)	file को writing के लिए open किया जाता है | अगर file exist होती है तो उसे overwrite किया जाता है और अगर file exist नहीं होती है तो new file create की जाती है |
wb(write in binary)	file को binary format में writing के लिए open किया जाता है | अगर file exist होती है तो उसे overwrite किया जाता है और अगर file exist नहीं होती है तो new file create की जाती है |
w+(write in binary)	file को reading और writing के लिए open किया जाता है | अगर file exist होती है तो उसे overwrite किया जाता है और अगर file exist नहीं होती है तो new file create की जाती है |
wb+(write and read in binary)	file को binary format में reading और writing के लिए open किया जाता है | अगर file exist होती है तो उसे overwrite किया जाता है और अगर file exist नहीं होती है तो new file create की जाती है |

Closing a File

जब file को open() method से open किया जाता है तब file पर कुछ operations करके close भी किया जाता है | File को close करने के लिए 'close()' method का इस्तेमाल किया जाता है |
close() method को जब file object की मदद से call किया जाता है तब file पर write हुए hidden data को flush किया जाता है |
जहा पर file को close किया जाता है उसके बाद बिना file open किये उस file पर कोई भी operation नहीं किया जा सकता है |

Syntax

```
fileObj.close()
```

Parameter

close() method के लिए कोई parameter नहीं होता है |

Source Code :

```
file = open("textfile.txt",'a') #Opening File

file.close() #Closing File
```

Reading a File

File का data read करने के लिए read() method का इस्तेमाल किया जाता है |

Syntax

```
fileObj.read(size)
```

Parameter

size : Optional. जितना data लेना है उसकी size यहाँ पर दी जाती है | अगर दिया नहीं जाता है तो end of file तक data को read किया जाता है |

अगर size पर invalid value दी जाती है तो end of file तक data को read किया जाता है |

textfile.txt

```
Hello World
```

Example पर file को read mode पर open किया गया है और open किये हुए file का data read करने के लिए read() method का इस्तेमाल किया गया है और बाद में file को close किया गया है |

Source Code :

```
file = open("textfile.txt",'r')
print(file.read())
file.close()
```

Output :

```
Hello World
```

Example for read() Method with Parameter

Example पर read() method पर parameter में size सिर्फ 2 दी हुई है इसीलिए file से सिर्फ पहले दो character ही return हुए है |

Source Code :

```
file = open("textfile.txt",'r')
print(file.read(2))
file.close()
```

Output :

```
He
```

Writing a File

File पर data को write करने के लिए write() method का इस्तेमाल किया जाता है |

Syntax

```
fileObj.write(str)
```

Parameter

str : जो string file पर write करना है वो string यहाँ पर दिया जाता है |

write() method file को overwrite करके शुरुआत से data को write किया जाता है |

Returning Value

write() method ये file के character की size return करता है |

textfile.txt

```
Hello World
```

Example पर write mode पर file को open किया गया है उसके बाद file पर data को write() method की मदद से write करके file को close किया गया है |

Source Code :

```
file = open("textfile.txt",'w+')
file.write("Hello")
file.close()
```

textfile.txt:

```
Hello
```

Renaming a File

File को rename करने के लिए 'os' module को import करके उसके rename() method का इस्तेमाल किया जाता है |

Syntax

```
os.rename(fileName, newFileName)
```

Parameter

fileName : जिस file का name rename करना है उस file का name यहाँ पर दिया जाता है |

newFileName : जो नाम file को देना है वो file का नाम यहाँ पर दिया जाता है |

अगर पहले parameter में file exist नहीं होती है तो 'FileNotFoundError' ये exeception raise होता है |

अगर दुसरे parameter में file exist होती है तो 'FileExistsError' ये exeception raise होता है |

Source Code :

```
import os
os.rename("textfile.txt", "myfile.txt")
#rename textfile.txt to myfile.txt
```

Removing a File

File को remove करने के लिए 'os' module को import करके उसके remove() method का इस्तेमाल किया जाता है |

Syntax

```
os.remove(fileName)
```

Parameter

fileName : जिस file का name remove करना है उस file का name यहाँ पर दिया जाता है |

अगर parameter पर दी हुई file exist नहीं होती है 'FileNotFoundError' ये exception raise होता है |

Source Code :

```
import os
os.remove("myfile.txt")
#removed file 'myfile.txt'
```

File Methods

File को handle करने के लिए और भी method है | वो सभी methods निचे दिए गए है |

File Methods	**Description**
close()	open किये गए file को close किया जाता है |
flush()	file stream से writing buffer को flush करने के लिए इस्तेमाल किया जाता है |
fileno()	file descriptor integer में return करता है |
isatty()	file; terminal device से connect हुआ है या नहीं ये boolean value में return करता है |
read()	File का data read करने के लिए इस्तेमाल किया जाता है |
readline()	file के line से दिए हुए size तक character/byte को read करके return किया जाता है |
readlines()	file के line से दिए हुए दिखाऊ size तक character/byte को read करके lines को list में return किया जाता है |
seek()	इसका इस्तेमाल offset पर file की current position set करने के लिए किया जाता है |
tell()	file पर data read या write करने के file की current position return की जाती है |
writable()	file पर data write किया गया है या नहीं ये boolean value में return किया जाता है |
write()	File पर string को write करके उस string की length को return करता है |
writelines()	इसका इस्तेमाल दिए हुए sequence को file पर write किया जाता है |

Regular Expression Introduction

Python में regular expression ये string पर operation करता है |
String पर match या किसी string को find/search करने के लिए regular expression काफी मददगार साबित होता है |
Python में अगर regular expression का इस्तेमाल करना हो तो 're' इस module का इस्तेमाल करना पड़ता है |
किसी भी regular expression के हिस्से का इस्तेमाल करना हो तो program पर 're' इस module को import करना पड़ता है |
जैसे कि.

Source Code :

```
import re
```

Raw String Notation for Regular Expression Patterns

Python में जब regular expression लिखना होता है तब Normal string के बजाय raw string का इस्तेमाल किया जाता है |
Raw String की शुरुआत 'r' इस prefix से की जाती है | अगर string पर 'r'(raw string) prefix दिया जाता है तो backslashes(\\) को किसी भी प्रकार से handle नहीं किया जा सकता है |

Difference Between Normal String and Raw String

Normal String

Normal String में '\n'(escape sequence) ये एक ही character होता है |

Source Code :

```
print(len("\n"))
#return 1

print("Hello World\nHello World")
#return
#Hello World
#Hello World
```

Raw String

Raw String में '\' और 'n' ये दोनों अलग-अलग characters होते है |

Source Code :

```
print(len(r"\n"))
#return 2

print("Hello World\nHello World")
#return Hello World\nHello World
```

In Regular Expression

Regular Expression में raw string का इस्तेमाल pattern के रूप में काफी बार किया जाता है |

Source Code :

```
"\\d+\\w" #Normal String

r"\d+\w" #Raw String
```

Matching a String

String से match करने के लिए re module के 'match()' function का इस्तेमाल किया जाता है |

Syntax for match() Regular Expression Function

```
re.match(pattern, raw/string, flags)
```

Parameter

pattern : यहाँ पर pattern दिया जाता है |

raw/string : यहाँ पर pattern से match करने के लिए string या raw string दी जाती है |

flags : Optional. यहाँ पर एक या एक से ज्यादा flags दिए जाते है | Bitwise OR(|) से एक से ज्यादा flags का इस्तेमाल किया जाता है |

Example for match() Function in Python

Example में regular expression के match() function का इस्तेमाल किया गया है |

pattern : r"(\w+)\ (\d+)\ (\w+)

ये raw string दी गयी है | पहले group में characters(\w+) को match किया गया है बाद में एक space(\ ) दिया गया है उसके बाद digits(\d+) को match किया गया है उसके बाद फिर एक space(\ ) और आखिर में फिर characters को match किया गया है |

str : Hello 123 Hello

इस normal string पर operation किया गया है |

Source Code :

```
import re

str = "Hello 123 Hello"
a = re.match(r"(\w+)\ (\d+)\ (\w+)", str)

print("group() :",a.group()) #return entire matches ((\w+)\ (\d+)\ (\w+))
print("group(0) :",a.group(0)) #return entire matches ((\w+)\ (\d+)\ (\w+))
print("group(1) :",a.group(1)) #return first subgroup match (\w+)
print("group(2) :",a.group(2)) #return seconnd subgroup match (\d+)
print("group(3) :",a.group(3)) #return third subgroup match (\w+)
```

Output :

```
group() : Hello 123 Hello
group(0) : Hello 123 Hello
group(1) : Hello
group(2) : 123
group(3) : Hello
```

Python में Regular Expression में String पर operation करने के लिए कुछ हिस्से बनाये गए है |

- Character Classes
- Quantifiers
- Metacharacter
- Modifiers/Flags
- Regular Expression Functions
- Regular Expression Object Methods
- Regular Expression Match Object Methods

Character Classes - Regular Expression

Expressions	Description
[..]	कोई भी character या digits ढूंढने के लिए दिया जाता है	
[^..]	कोई भी character या digits दिया जाता है , उसे ढूंढा नहीं जाता	
[0-9]	0 से 9 तक के digits को ढूंढा जाता है	
[^0-9]	0 से 9 तक के digits को ढूंढा नहीं जाता है	
[A-Z]	Uppercase A से लेकर uppercase Z तक characters को ढूंढा जाता है	
[^A-Z]	Uppercase A से लेकर uppercase Z तक characters को ढूंढा नहीं जाता है	
[a-z]	lowercase a से लेकर lowercase z तक characters को ढूंढा जाता है	
[^a-z]	lowercase a से लेकर lowercase z तक characters को ढूंढा नहीं जाता है	
[mf]ail	यहाँ पर mail और fail को string पर ढूंढा जाता है	
Grac[ey]	यहाँ पर Grace और Gracy को string पर ढूंढा जाता है	

[..] - Character Class

[..] इस Class में Bracket के अन्दर जो भी character या digit दिया जाता है उसे string में से ढूंढा जाता है |

Syntax

```
r"[..]"
or
"[..]"
```

Example पर सिर्फ 'o'(case-sensitive) को function के हिसाब से ढूंढा गया है |

Source Code :

```
import re
str = "123 HellO World 482 Hiii"
pattern = r"[o]+" # try to removing '+' quantifier from pattern

a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")

a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")

a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : o
match() : No Match
findall() : ['o']
```

Another Example for [..] Character Class in Python

Example पर सिर्फ 'o'(case-insensitive) को function के हिसाब से ढूंढा गया है |

Source Code :

```
import re

str = "123 HellO World 482 Hiii"
pattern = r"[o]+" # try to removing '+' quantifier from pattern

a = re.search(pattern, str, re.I) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")

a = re.match(pattern, str, re.I) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str, re.I) #findall function
print("findall() :",a)
```

Output :

```
search() : O
match() : No Match
findall() : ['O', 'o']
```

[^..] - Character Class

[^..] इस Class में Bracket के अन्दर जो भी character या digit दिया जाता है उसे string में से ढूंढा नहीं जाता है |

Syntax

```
r"[^..]"
or
"[^..]"
```

Example for [^..] Character Class in Python

Example पर सिर्फ 'o'(case-sensitive) को function के हिसाब से ढूंढा नहीं गया है |

Source Code :

```
import re

str = "123 HellO World 482 Hiii"
pattern = r"[^o]+" # try to removing '+' quantifier from pattern

a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")

a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : 123 HellO W
match() : 123 HellO W
findall() : ['123 HellO W', 'rld 482 Hiii']
```

Another Example for [^..] Character Class in Python

Example पर सिर्फ 'o'(case-insensitive) को function के हिसाब से ढूंढा नहीं गया है |

Source Code :

```
import re

str = "123 HellO World 482 Hiii"
pattern = r"[^o]+" # try to removing '+' quantifier from pattern

a = re.search(pattern, str, re.I) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
```

```
a = re.match(pattern, str, re.I) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")

a = re.findall(pattern,str, re.I) #findall function
print("findall() :",a)
```

Output :

```
search() : 123 Hell
match() : 123 Hell
findall() : ['123 Hell', ' W', 'rld 482 Hiii']
```

[0-9] - Character Class

[0-9] इस Class में Bracket के अन्दर जो भी digit दिया जाता है उसे string में ढूंढा जाता है |

Syntax

```
r"[0-9]"
or
"[0-9]"
```

Note : 0 और 9 इन दोनों का समावेश होता है |

Example for [0-9] Character Class in Python

Example पर 0 से 9 तक function के हिसाब से ढूंढा गया है |

Source Code :

```
import re
str = "123 Hello World 482 Hiii"
pattern = r"[0-9]+" # try to removing '+' quantifier from pattern
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : 123
match() : 123
findall() : ['123', '456']
```

Example for [4-6] Character Class in Python

Example पर 4 से 6 तक function के हिसाब से ढूंढा गया है |

Source Code :

```
import re
str = "123 Hello World 482 Hiii"
pattern = r"[0-9]+" # try to removing '+' quantifier from pattern
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : 123
match() : 123
findall() : ['123', '456']
```

script पर [46] या [64] ऐसा expression दिया जा सकता है इससे सिर्फ 3 और 5 को ढूंढा जा सकता है |

Note : यहाँ पर [6-4] ऐसा expression नहीं दिया जा सकता |

[^0-9] - Character Class

[^0-9] इस Class में Bracket के अन्दर जो भी digit दिया जाता है उसे string में ढूंढा नहीं जाता है |

Syntax

```
r"[^0-9]"
or
"[^0-9]"
```

Note : 0 और 9 इन दोनों का समावेश होता है |

Example for [^0-9] Character Class in Python

Example पर 0 से 9 तक function के हिसाब से ढूंढा नहीं गया है |

Source Code :

```
import re
str = "123 Hello World 482 Hiii"
pattern = r"[^0-9]+" # try to removing '+' quantifier from pattern
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() :  Hello World 
match() : No Match
findall() : [' Hello World ', ' Hiii']
```

Example for [^4-6] Character Class in Python

Example पर 4 से 6 तक function के हिसाब से ढूंढा नहीं गया है |

Source Code :

```
import re
str = "123 Hello World 482 Hiii"
pattern = r"[^4-6]+" # try to removing '+' quantifier from pattern
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : 123 Hello World
match() : 123 Hello World
findall() : ['123 Hello World ', '82 Hiii']
```

script पर [^46] या [^64] ऐसा expression दिया जा सकता है इससे सिर्फ 3 और 5 को ढूंढा नहीं जा सकता है |

Note : यहाँ पर [^6-4] ऐसा expression नहीं दिया जा सकता |

[A-Z] - Character Class

[A-Z] इस Class में Bracket के अन्दर जो भी uppercase alphabet दिया जाता है उसे string में ढूंढा जाता है |

Syntax

```
r"[A-Z]"
or
"[A-Z]"
```

Note : A और Z इन दोनों का समावेश होता है

Example for [A-Z] Character Class in Python

Example पर uppercase A से uppercase Z तक function के हिसाब से ढूंढा गया है |

Source Code :

```
import re
str = "123 Hello World 482 Hiii"
pattern = r"[A-Z]+" # try to removing '+' quantifier from pattern
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : H
match() : No Match
findall() : ['H', 'W', 'H']
```

Example for [U-X] Character Class in Python

Example पर uppercase U से uppercase X तक function के हिसाब से ढूंढा गया है |

Source Code :

```
import re
str = "123 Hello World 482 Hiii"
pattern = r"[U-X]+" # try to removing '+' quantifier from pattern
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : W
match() : No Match
findall() : ['W']
```

script पर [UX] या [XU] ऐसा expression दिया जा सकता है इससे सिर्फ U और X को ढूंढा जा सकता है |

Note : यहाँ पर [Z-A] ऐसा expression नहीं दिया जा सकता |

[^A-Z] - Character Class

[^A-Z] इस Class में Bracket के अन्दर जो भी uppercase alphabet दिया जाता है उसे string में ढूंढा नहीं जाता है |

Syntax

```
r"[^A-Z]"
or
"[^A-Z]"
```

Note : A और Z इन दोनों का समावेश होता है

Example for [^A-Z] Character Class in Python

Example पर uppercase A से uppercase Z तक function के हिसाब से ढूंढा नहीं गया है |

Source Code :

```
import re
str = "123 Hello World 482 Hiii"
pattern = r"[^A-Z]+" # try to removing '+' quantifier from pattern
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : 123
match() : 123
findall() : ['123 ', 'ello ', 'orld 482 ', 'iii']
```

Example for [U-X] Character Class in Python

Example पर uppercase U से uppercase X तक function के हिसाब से ढूंढा नहीं गया है |

Source Code :

```
import re
str = "123 Hello World 482 Hiii"
pattern = r"[U-X]+" # try to removing '+' quantifier from pattern
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : 123 Hello
match() : 123 Hello
findall() : ['123 Hello ', 'orld 482 Hiii']
```

script पर [^UX] या [^XU] ऐसा expression दिया जा सकता है इससे सिर्फ U और X को ढूंढा नहीं जा सकता है |

Note : यहाँ पर [^Z-A] ऐसा expression नहीं दिया जा सकता |

[a-z] - Character Class

[a-z] इस Class में Bracket के अन्दर जो भी lowercase alphabet दिया जाता है उसे string में ढूंढा जाता है |

Syntax

```
r"[a-z]"
or
"[a-z]"
```

Note : a और z इन दोनों का समावेश होता है

Example for [a-z] Character Class in Python

Example पर lowercase a से lowercase z तक function के हिसाब से ढूंढा गया है |

Source Code :

```
import re
str = "123 Hello World 482 Hiii"
pattern = r"[a-z]+" # try to removing '+' quantifier from pattern
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : ello
match() : No Match
findall() : ['ello', 'orld', 'iii']
```

Example for [u-x] Character Class in Python

Example पर lowercase u से lowercase x तक function के हिसाब से ढूंढा गया है |

Source Code :

```
import re
str = "123 Hello World 482 Hiii"
pattern = r"[u-x]+" # try to removing '+' quantifier from pattern
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : No Match
match() : No Match
findall() : []
```

script पर [ux] या [xu] ऐसा expression दिया जा सकता है इससे सिर्फ u और x को ढूंढा जा सकता है |

Note : यहाँ पर [z-a] ऐसा expression नहीं दिया जा सकता |

[^a-z] - Character Class

[^a-z] इस Class में Bracket के अन्दर जो भी lowercase alphabet दिया जाता है उसे string में ढूंढा नहीं जाता है |

Syntax

```
r"[^a-z]"
or
"[^a-z]"
```

Note : a और z इन दोनों का समावेश होता है

Example for [^a-z] Character Class in Python

Example पर lowercase a से lowercase z तक function के हिसाब से ढूंढा नहीं गया है |

Source Code :

```
import re
str = "123 Hello World 482 Hiii"
pattern = r"[^a-z]+" # try to removing '+' quantifier from pattern
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : 123 H
match() : 123 H
findall() : ['123 H', ' W', ' 482 H']
```

Example for [^u-x] Character Class in Python

Example पर lowercase u से lowercase x तक function के हिसाब से ढूंढा नहीं गया है |

Source Code :

```
import re
str = "123 Hello World 482 Hiii"
pattern = r"[^u-x]+" # try to removing '+' quantifier from pattern
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : 123 Hello World 482 Hiii
match() : 123 Hello World 482 Hiii
findall() : ['123 Hello World 482 Hiii']
```

script पर [^ux] या [^xu] ऐसा expression दिया जा सकता है इससे सिर्फ u और x को ढूंढा नहीं जा सकता है |

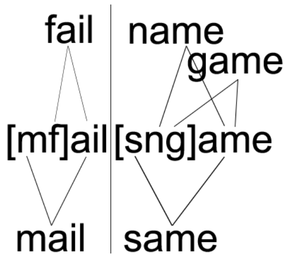
Note : यहाँ पर [^z-a] ऐसा expression नहीं दिया जा सकता |

[mf]ail - Single Character Beginning - Character Class

[mf]ail इस Class में Bracket के अन्दर एक या एक से ज्यादा rhyming words के पहले characters और उसके बाद का हिस्सा bracket close होने के बाद दिया जाता है और बाद में तैयार होने वाले word को ढूंढा जाता है |
जैसे कि,
[mf]ail = mail, trail. [sng]ame = same, name, game

Syntax

```
r"[mf]ail"
or
"[mf]ail"
```



Example for [mf]ail Character Class in Python

Source Code :

```
import re
str = "mail sent has failed"
pattern = r"[mf]ail"
a = re.search(pattern, str, re.I) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str, re.I) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str, re.I) #findall function
print("findall() :",a)
```

Output :

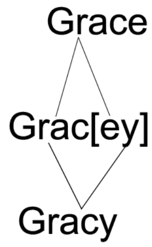
```
search() : mail
match() : mail
findall() : ['mail', 'fail']
```

Grac[ey] - Single Character End - Character Class

Grac[ey] इस Class में Bracket के अन्दर word के आखिर के characters दिए जाते है और bracket से पहले उन words के एक जैसे characters दिए जाते है और तैयार हुए words को string में ढूंढा जाता है |
जैसे कि,
Grac[ey] = Grace, Gracey.

Syntax

```
r"Grac[ey]"
or
"Grac[ey]"
```



Example for Grac[ey] Character Class in Python

Source Code :

```
import re
str = "Grace and Gracy are good friends"
pattern = r"Grac[ey]" # try to removing '+' quantifier from pattern
a = re.search(pattern, str, re.I) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str, re.I) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str, re.I) #findall function
print("findall() :",a)
```

Output :

```
search() : Grace
match() : Grace
findall() : ['Grace', 'Gracy']
```

Quantifiers - Regular Expression

Expressions	Description
n+	String में कम से कम एक या उससे ज्यादा 'n' occurrences को match किया जाता है	
n*	String में zero या उससे ज्यादा 'n' occurrences से match किया जाता है	
n?	String में zero या एक occurrence 'n' से match किया जाता है	
n{X}	String में से n को sequences of X number तक match किया जाता है	
n{X,Y}	String में से n को sequences of X number से Y number तक match किया जाता है	
n{X,}	String में से n को sequences of कम से कम X number तक match किया जाता है	
n$	String में से n को end पर match किया जाता है	अगर multiline(re.M) mode होता है तो हर newline के end पर match किया जाता है	
^n	String में से n को start पर match किया जाता है	अगर multiline(re.M) mode होता है तो हर newline के start पर match किया जाता है	
p|q	String में से p या q को match किया जाता है	
python+	String में से python या python में 'n' एक या एक से ज्यादा match किया जाता है	
python*	String में से python या python में 'n' शून्य या उससे ज्यादा match किया जाता है	
python?	String में से python या python में 'n' शून्य या एक match किया जाता है	

n+ - Quantifier

n+ इस Quantifier में String में कम से कम एक या उससे ज्यादा 'n' occurrences को match किया जाता है |

Syntax

```
r"n+"
or
"n+"
```

Example for n+ Quantifier in Python

Example पर 'o' character के एक या उससे ज्यादा occurrences को match किया गया है |

Source Code :

```
import re
str = "Python is very good Language"
pattern = r"o+"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : o
match() : No Match
findall() : ['o', 'oo']
```

Example for Without '+' Quantifier in Python

Example पर 'o' character के एक occurrence को match किया गया है |

Source Code :

```
import re
str = "Python is very good Language"
pattern = r"o"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : o
match() : No Match
findall() : ['o', 'o', 'o']
```

n* - Quantifier

n* इस Quantifier में String में zero या उससे ज्यादा 'n' occurrences से match किया जाता है |

Syntax

```
r"n*"
or
"n*"
```

Example for n* Quantifier in Python

Example पर 'o' character के 0 या उससे ज्यादा occurrences को match किया गया है |

Source Code :

```
import re
str = "Python is very good Language"
pattern = r"o*"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() :
match() :
findall() : ['', '', '', '', 'o', '', '', '', '', '', '', '', '', '', '', '', 'oo', '', '', '', '', '', '', '', '', '', '']
```

n? - Quantifier

n? इस Quantifier में String में zero या एक 'n' occurrence से match किया जाता है |

Syntax

```
r"n?"
or
"n?"
```

Example for n? Quantifier in Python

Example पर 'o' character के 0 या एक occurrence को match किया गया है |

Source Code :

```
import re
str = "Python is very good Language"
pattern = r"o?"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() :
match() :
findall() : ['', '', '', '', 'o', '', '', '', '', '', '', '', '', '', '', '', '', 'o', 'o', '', '', '', '', '', '', '', '', '', '', '']
```

n{X} - Quantifier

n{X} इस Quantifier में String में से n को sequences of X number तक match किया जाता है |

Syntax

```
r"n{X}"
or
"n{X}"
```

Example for n{X} Quantifier in Python

Example पर 'o' के 2 sequence तक match किया गया है |

Source Code :

```
import re
str = "Pythoon is very good Language"
pattern = r"o{2}" #try to '1' on X
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : oo
match() : No Match
findall() : ['oo', 'oo']
```

Another Example for n{X} Quantifier in Python

Example पर 'o' के 1 sequence तक match किया गया है |

Source Code :

```
import re
str = "Pythoon is very good Language"
pattern = r"o{1}" #try to '1' on X
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : o
match() : No Match
findall() : ['o', 'o', 'o', 'o']
```

n{X,Y} - Quantifier

n{X,Y} इस Quantifier में String में से n को sequences of X number से Y number तक match किया जाता है |

Syntax

```
r"n{X,Y}"
or
"n{X,Y}"
```

Example for n{X,Y} Quantifier in Python

Example पर 'o' के 1 से 4 sequences तक match किया गया है |

Source Code :

```
import re
str = "Pythooon is very gooooood Language"
pattern = r"o{1,4}"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : ooo
match() : No Match
findall() : ['ooo', 'oooo', 'oo']
```

Another Example for n{X,Y} Quantifier in Python

Example पर 'o' के 3 से 4 sequences तक match किया गया है |

Source Code :

```
import re
str = "Pythooon is very gooooood Language"
pattern = r"o{3,4}"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : ooo
match() : No Match
findall() : ['ooo', 'oooo']
```

n{X,} - Quantifier

n{X,} इस Quantifier में String में से n को sequences of कम से कम X number तक match किया जाता है |

Syntax

```
r"n{X,}"
or
"n{X,}"
```

Example for n{X,} Quantifier in Python

Example पर 'o' के कम से कम 3 sequences तक match किया गया है |

Source Code :

```
import re
str = "Pythooon is very gooooood Language"
pattern = r"o{3,}"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : ooo
match() : No Match
findall() : ['ooo', 'oooooo']
```

n$ - Quantifier

n$ इस Quantifier में String में से n को end पर match किया जाता है | अगर multiline(re.M) mode होता है तो हर newline के end पर match किया जाता है |

Syntax

```
r"n$"
or
"n$"
```

Example for n$ Quantifier in Python

Example पर 'ge' को end पर match किया गया है |

Source Code :

```
import re
str = "Pythooon is very gooooood Language"
pattern = r"ge$"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : ge
match() : No Match
findall() : ['ge']
```

Example for n$ Quantifier in Python

Example पर 'ge' को multiple lines के end पर match किया गया है |

Source Code :

```
import re
str = """Pythooon is very gooooood Language
Pythooon is Language
Pythooon is very gooooood Language"""
pattern = r"ge$"
a = re.search(pattern, str, re.M) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str, re.M) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str, re.M) #findall function
print("findall() :",a)
```

Output :

```
search() : ge
match() : No Match
findall() : ['ge', 'ge', 'ge']
```

^n - Quantifier

^n इस Quantifier में String में से n को start पर match किया जाता है | अगर multiline(re.M) mode होता है तो हर newline के start पर match किया जाता है |

Syntax

```
r"^n"
or
"^n"
```

Example for ^n Quantifier in Python

Example पर 'Py' को start पर match किया गया है |

Source Code :

```
import re
str = "Pythooon is very gooooood Language"
pattern = r"^Py"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : Py
match() : Py
findall() : ['Py']
```

Another Example for ^n Quantifier in Python

Example पर 'Py' को multiple lines के start पर match किया गया है |

Source Code :

```
import re
str = """Pythooon is very gooooood Language
Pythooon is Language
Pythooon is very gooooood Language"""
pattern = r"^Py"
a = re.search(pattern, str, re.M) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str, re.M) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str, re.M) #findall function
print("findall() :",a)
```

Output :

```
search() : Py
match() : Py
findall() : ['Py', 'Py', 'Py']
```

p|q - Quantifier

p|q इस Quantifier में String में या तो 'p' या तो 'q' को match किया जाता है |

Syntax

```
r"p|q"
or
"p|q"
```

Example for p|q Quantifier in Python

Example पर 'o' या 'n' को match किया गया है |

Source Code :

```
import re
str = "Pythooon is very gooooood Language"
pattern = r"o|n"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : o
match() : No Match
findall() : ['o', 'o', 'o', 'n', 'o', 'o', 'o', 'o', 'o', 'o', 'n']
```

python+ - Quantifier

python+ इस Quantifier में String में से python या python में 'n' एक या एक से ज्यादा match किया जाता है |

Syntax

```
r"python+"
or
"python+"
```

Example for python+ Quantifier in Python

Example पर 'o' या 'n' को match किया गया है |

Source Code :

```
import re
str = "python pythonnnnnnnn pytho"
pattern = r"python+"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : python
match() : python
findall() : ['python', 'pythonnnnnnnn']
```

python* - Quantifier

python* इस Quantifier में String में से python या python में 'n' शून्य या उससे ज्यादा match किया जाता है |

Syntax

```
r"python*"
or
"python*"
```

Example for python* Quantifier in Python

Example पर 'o' या 'n' को match किया गया है |

Source Code :

```
import re
str = "python pythonnnnnnnnn pytho"
pattern = r"python*"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : python
match() : python
findall() : ['python', 'pythonnnnnnnnn', 'pytho']
```

python? - Quantifier

python? इस Quantifier में String में से python या python में 'n' शून्य या एक match किया जाता है |

Syntax

```
r"python?"
or
"python?"
```

Example for python? Quantifier in Python

Source Code :

```
import re
str = "python pythonnnnnnnnn pytho"
pattern = r"python?"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : python
match() : python
findall() : ['python', 'python', 'pytho']
```

Metacharacters- Regular Expression

Expressions	Description
dot(.)	single dot से single character ढूंढा जाता है |
b	String में से word के शुरुआत या end के matches ढूंढे जाते है |
B	String में से word के शुरुआत या end के matches ढूंढे नहीं जाते है |
d	String में से 0 से 9 numbers तक ढूंढा जाता है |
D	String में से non-numbers character ढूंढा जाता है |
f	String में से form-feed character को ढूंढा जाता है |
n	String में से newline character को ढूंढा जाता है |
r	String में से carriage return character को ढूंढा जाता है |
s	String में से whitespace character को ढूंढा जाता है |
S	String में से non-whitespace character को ढूंढा जाता है |
t	String में से tab character को ढूंढा जाता है |
v	String में से vertical tab character को ढूंढा जाता है |
w	String में से word character को ढूंढा जाता है |
W	String में से non-word character को ढूंढा जाता है |
0	String में से null character को ढूंढा जाता है |

dot(.) - Metacharacter

dot(.) ये Expression में String में से single dot से single character ढूंढा जाता है |

Syntax

```
r"python?"
or
"python?"
```

Example for dot(.) Metacharacter in Python

Source Code :

```
import re
str = "Hello World worLd Word"
pattern = r"."
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : H
match() : H
findall() : ['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', ' ', 'w', 'o', 'r', 'L', 'd', ' ', 'W', 'o', 'r', 'd']
```

b - Metacharacter

lowercaseb(\b) इस Metacharacter से String में से word के शुरुआत या end के matches ढूंढे जाते है |

Syntax

```
r"\bexp" #beginning
r"exp\b" #end
or
"\\bexp" #beginning
"exp\\b" #end
```

Example for lowercaseb(\bexp) Metacharacter in Python

Example पर String के word के 'Wo' इस character को beginning पर match किया गया है |

Source Code :

```
import re
str = "Hello World worLd Word"
pattern = r"\bWo"
a = re.search(pattern, str) #search function
if a:
   print("search() :",a.group())
else:
   print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
   print("match() :",a.group())
else:
   print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : Wo
match() : No Match
findall() : ['Wo', 'Wo']
```

Another Example for lowercaseb(exp\b) Metacharacter in Python

Example पर String के word के 'ld' इस character को end पर match किया गया है |

Source Code :

```
import re
str = "Hello World worLd Word"
pattern = r"ld\b"
a = re.search(pattern, str) #search function
if a:
   print("search() :",a.group())
else:
   print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
   print("match() :",a.group())
else:
   print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : ld
match() : No Match
findall() : ['ld']
```

B - Metacharacter

UppercaseB(\B) इस Metacharacter से String में से word के ढूंढे हुए matches शुरुआत या end पर नहीं होते है |

Syntax

```
r"\Bexp" #beginning
r"exp\B" #end
or
"\\Bexp" #beginning
"exp\\B" #end
```

Example for UppercaseB(\Bexp) Metacharacter in Python

Example पर 'ld' ये String के word के शुरुआत पर नहीं है |

Source Code :

```
import re
str = "Hello World worLd Word"
pattern = r"\Bld"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : ld
match() : No Match
findall() : ['ld']
```

Another Example for UppercaseB(exp\B) Metacharacter in Python

Example पर 'Wo' ये String के word के end पर नहीं है |

Source Code :

```
import re
str = "Hello World worLd Word"
pattern = r"Wo\B"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : Wo
match() : No Match
findall() : ['Wo', 'Wo']
```

d - Metacharacter

lowercased(\d) इस Metacharacter से String में से 0 से 9 numbers तक ढूंढा जाता है |

Syntax

```
r"\d"
or
"\\d"
```

Example for lowercased(\d) Metacharacter in Python

Source Code :

```
import re
str = "Hello 123 World 456"
pattern = r"\d"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : 123
match() : No Match
findall() : ['123', '456']
```

D - Metacharacter

UppercaseD(\D) इस Metacharacter से String में से 0 से 9 numbers तक ढूंढा नहीं जाता है |

Syntax

```
r"\D"
or
"\\D"
```

Example for UppercaseD(\D) Metacharacter in Python

Source Code :

```
import re
str = "Hello 123 World 456"
pattern = r"\D"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : H
match() : H
findall() : ['H', 'e', 'l', 'l', 'o', ' ', ' ', 'W', 'o', 'r', 'l', 'd', ' ']
```

f - Form feed - Metacharacter

Form-Feed(\f) इस Metacharacter से String में से form-feed character को ढूंढा जाता है |

Syntax

```
r"\f"
or
"\\f"
```

Example for Form-Feed(\f) Metacharacter in Python

Source Code :

```
import re
str = "Hello World \f Hello World"
pattern = r"\f"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() :
match() : No Match
findall() : ['\x0c']
```

n - newline - Metacharacter

Newline(\n) इस Metacharacter से String में से newline character को ढूंढा जाता है |

Syntax

```
r"\n"
or
"\\n"
```

Example for Newline(\n) Metacharacter in Python

Source Code :

```
import re
str = "Hello World \n Hello World"
pattern = r"\n"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() :
match() : No Match
findall() : ['\n']
```

r - carriage return- Metacharacter

carriage return(\r) इस Metacharacter से String में से carriage return character को ढूंढा जाता है |

Syntax

```
r"\r"
or
"\\r"
```

Example for carriage return(\r) Metacharacter in Python

Source Code :

```
import re
str = "Hello World \r Hello World"
pattern = r"\r"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() :
match() : No Match
findall() : ['\r']
```

s - whitespace- Metacharacter

whitespace(\s) इस Metacharacter से String में से whitespace character को ढूंढा जाता है |

Syntax

```
r"\s"
or
"\\s"
```

Example for whitespace(\s) Metacharacter in Python

Source Code :

```
import re
str = "Hello World Hello World"
pattern = r"\s"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() :
match() : No Match
findall() : [' ', ' ', ' ']
```

s - non-whitespace- Metacharacter

non-whitespace(\S) इस Metacharacter से String में से non-whitespace character को ढूंढा जाता है |

Syntax

```
r"\S"
or
"\\S"
```

Example for non-whitespace(\S) Metacharacter in Python

Source Code :

```
import re
str = "Hello World Hello World"
pattern = r"\S"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : H
match() : H
findall() : ['H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd', 'H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd']
```

S - non-whitespace- Metacharacter

non-whitespace(\S) इस Metacharacter से String में से non-whitespace character को ढूंढा जाता है |

Syntax

```
r"\S"
or
"\\S"
```

Example for non-whitespace(\S) Metacharacter in Python

Source Code :

```
import re
str = "Hello World Hello World"
pattern = r"\S"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : H
match() : H
findall() : ['H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd', 'H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd']
```

t - tab- Metacharacter

tab(\t) इस Metacharacter से String में से tab character को ढूंढा जाता है |

Syntax

```
r"\t"
or
"\\t"
```

Example for tab(\t) Metacharacter in Python

Source Code :

```
import re
str = "Hello World\tHello World"
pattern = r"\t"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() :
match() : No Match
findall() : ['\t']
```

v - vertical tab- Metacharacter

vertical tab(\v) इस Metacharacter से String में से vertical tab character को ढूंढा जाता है |

Syntax

```
r"\v"
or
"\\v"
```

Example for vertical tab(\t) Metacharacter in Python

Source Code :

```
import re
str = "Hello World\vHello World"
pattern = r"\v"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : 
match() : No Match
findall() : ['\x0b']
```

w - word- Metacharacter

word(\w) इस Metacharacter से String में से word character को ढूंढा जाता है |

Syntax

```
r"\w"
or
"\\w"
```

Example for word(\w) Metacharacter in Python

Source Code :

```
import re
str = "Hello @123 World 456"
pattern = r"\w"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : H
match() : H
findall() : ['H', 'e', 'l', 'l', 'o', '1', '2', '3', 'W', 'o', 'r', 'l', 'd', '4', '5', '6']
```

W - non-word- Metacharacter

non-word(\W) इस Metacharacter से String में से non-word character को ढूंढा जाता है |

Syntax

```
r"\W"
or
"\\W"
```

Example for non-word(\W) Metacharacter in Python

Source Code :

```
import re
str = "Hello @123 World 456"
pattern = r"\W"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() :
match() : No Match
findall() : [' ', '@', ' ', ' ']
```

Modifiers or Flags - Regular Expression

Function या Method पर एक से ज्यादा Modifiers/Flags इस्तेमाल करने हो तो बीच में OR(|) Operator इस्तेमाल किया जाता है |

Expressions	Description
re.S or re.DOTALL	'.' character हर character को newline के साथ match किया जाता है	
re.I or re.IGNORECASE	Regular Expression को case-insensitive किया जाता है	
re.L or re.LOCALE	\w, \W, \b, \B ये current locale का पालन करता है	
re.M or re.MULTILINE	String की हर line की beginning(^) और end($) match किया जाता है	
re.U or re.UNICODE	\w,\W,\b,\B,\d,\D ये unicode character rules का पालन करता है	
re.X or re.VERBOSE	pattern पर # के साथ comment को add करने की अनुमति देता है	

DOTALL - Modifiers or Flags

re.DOTALL | re.S इस Modifier में '.' character हर character को newline के साथ match किया जाता है | अगर DOTALL Modifier नहीं दिया जाता है तो newline के बिना हर character को match किया जाता है |

Example for re.DOTALL | re.S Modifier in Python

Source Code :

```
import re
str = """Hello World
Hello Friends"""
pattern = "."
a = re.search(pattern, str, re.DOTALL) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str, re.DOTALL) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str, re.DOTALL) #findall function
print("findall() :",a)
```

Output :

```
search() : H
match() : H
findall() : ['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '\n', 'H', 'e', 'l', 'l', 'o', ' ', 'F', 'r', 'i', 'e', 'n', 'd', 's']
```

IGNORECASE - Modifiers or Flags

re.IGNORECASE | re.I इस Modifier case-insensitive match करता है | re.IGNORECASE क इस्तेमाल नहीं किया जाता है तो case-sensitive match किया जाता है |

Example for Without re.IGNORECASE | re.I Modifier in Python

Example पर 'l'(case-sensitive) match किया गया है |

Source Code :

```
import re
str = "HelLo WoRLD heLLo woRld"
pattern = "l"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str) #findall function
print("findall() :",a)
```

Output :

```
search() : l
match() : No Match
findall() : ['l', 'l']
```

Example for With re.IGNORECASE | re.I Modifier in Python

Example पर 'l'(case-insensitive) match किया गया है |

Source Code :

```
import re
str = "HelLo WoRLD heLLo woRld"
pattern = "l"
a = re.search(pattern, str, re.IGNORECASE) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str, re.IGNORECASE) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern,str, re.IGNORECASE) #findall function
print("findall() :",a)
```

Output :

```
search() : l
match() : No Match
findall() : ['l', 'L', 'L', 'L', 'L', 'l']
```

MULTILINE - Modifiers or Flags

re.MULTILINE | re.M इस Modifier से String की हर line की beginning(^) और end($) match किया जाता है |

Example for Without re.MULTILINE | re.M Modifier in Python

Example पर String के पहले line के पहले word को match किया गया है |

Source Code :

```
import re
str = """Hello World
Python is good language
Hello Friends"""
pattern = "^\w+"
a = re.search(pattern, str) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
```

```
a = re.match(pattern, str) #match function
if a:
print("match() :",a.group())
else:
print("match() : No Match")
a = re.findall(pattern, str) #findall function
print("findall() :",a)
```

Output :

```
search() : Hello
match() : Hello
findall() : ['Hello']
```

Example for With re.MULTILINE | re.M Modifier in Python

Example पर re.MULTILINE के साथ String के हर line के शुरुआत के word को match किया गया है |

Source Code :

```
import re
str = """Hello World
Python is good language
Hello Friends"""
pattern = "^\w+"
a = re.search(pattern, str, re.MULTILINE) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str, re.MULTILINE) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern, str, re.MULTILINE) #findall function
print("findall() :",a)
```

Output :

```
search() : Hello
match() : Hello
findall() : ['Hello', 'Python', 'Hello']
```

VERBOSE - Modifiers or Flags

re.VERBOSE | re.X इस Modifier से pattern पर comment के लिए अनुमति ली जाती है | अगर pattern में whitespace दिया जाता है तो उसे ignore किया जाता है | लेकिन जरुरत पर whitespace का इस्तेमाल किया जा सकता है |

Example for re.VERBOSE | re.X Modifier in Python

Example पर String के पहले line के पहले word को match किया गया है |

Source Code :

```
import re
str = "Hello 21257"
pattern = r"""\w+\ #matches word characters and whitespace
\d+#matches digits"""
a = re.search(pattern, str, re.VERBOSE) #search function
if a:
    print("search() :",a.group())
else:
    print("search() : No Match")
a = re.match(pattern, str, re.VERBOSE) #match function
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
a = re.findall(pattern, str, re.VERBOSE) #findall function
print("findall() :",a)
```

Output :

```
search() : Hello
match() : Hello
findall() : ['Hello']
```

Functions- Regular Expression

Functions	**Description**
compile()	regular expression object पर regular expression pattern को compile किया जाता है	
findall()	सभी non-overlapping matches list में return करता है	
finditer()	सभी non-overlapping matches; callable object में return करता है	
match()	pattern के हिसाब से string के beginning पर ही match करके match object return करता है	
search()	pattern के हिसाब से string के पहले occurrence को match करके match object return करता है	
split()	pattern द्वारा string को split करके अलग-अलग substrings को list में return किया जाता है	
sub()	pattern के हिसाब से string के substring(s) को replace करके नया string return करता है	
subn()	pattern के हिसाब से string के substring(s) को replace करके replaced string और replaced हुए pattern occurrence(s) को tuple में return करता है	

compile() - Regular Expression Function

compile() Function में regular expression object पर regular expression pattern को compile किया जाता है |

match करने के लिए search() और match() में इस function का इस्तेमाल किया जाता है |

Syntax

```
re.compile(pattern, flag(s))
```

Parameter

pattern : यहाँ पर pattern दिया जाता है |

flag(s) : Optional. यहाँ पर modifier दिया जाता है | एक से ज्यादा modifiers भी दिए जाते है |

Example for compile() Function in Python

Example पर String में से a से z तक case-sensitive matches किये गए है |

Source Code :

```
import re
pattern = re.compile(r"[a-z]+")
str = "Hello 123 World 456"
a = re.findall(pattern, str)
if a:
    print(a)
else:
    print("No Match")
```

Output :

```
['ello', 'orld']
```

Example for compile() Function with Flag in Python

Example पर String में से a से z तक case-insensitive matches किये गए है |

Source Code :

```
import re
pattern = re.compile(r"[a-z]+", re.IGNORECASE)
str = "Hello 123 World 456"
a = re.findall(pattern, str)
if a:
    print(a)
else:
    print("No Match")
```

Output :

```
['Hello', 'World']
```

findall() - Regular Expression Function

findall() Function ये सभी non-overlapping matches list में return करता है |
String के start से end तक match किया जाता है |

Syntax

```
re.findall(pattern, string, flag(s))
```

Parameter

pattern : यहाँ पर pattern दिया जाता है |

string : यहाँ पर string दिया जाता है |

flag(s) : Optional. यहाँ पर modifier दिया जाता है | एक से ज्यादा modifiers भी दिए जाते है |

Returning Value

findall() Function में सभी non-overlapping matches list में return करता है |

findall() और finditer() इन function में कोई फर्क नहीं होता है | लेकिन findall() function list को return करता है और finditer() function callable object को return करता है |

Example for findall() Function in Python

Example में a से z तक case-sensitive सभी matches list में return किये गए है |

Source Code :

```
import re
pattern = r"([a-z])"
str = "Hello 123 World 456"
a = re.findall(pattern, str)
if a:
    print(a)
else:
    print("No Match")
```

Output :

```
['e', 'l', 'l', 'o', 'o', 'r', 'l', 'd']
```

Example for findall() Function with Flag in Python

Example में a से z तक case-insensitive सभी matches list में return किये गए है |

Source Code :

```
import re
pattern = r"([a-z])"
str = "Hello 123 World 456"
a = re.findall(pattern, str, re.IGNORECASE)
if a:
    print(a)
else:
    print("No Match")
```

Output :

```
['H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd']
```

finditer() - Regular Expression Function

finditer() Function ये सभी non-overlapping matches; callable object में return करता है |
String के start से end तक match किया जाता है |

Syntax

```
re.finditer(pattern, string, flag(s))
```

Parameter

pattern : यहाँ पर pattern दिया जाता है |

string : यहाँ पर string दिया जाता है |

flag(s) : Optional. यहाँ पर modifier दिया जाता है | एक से ज्यादा modifiers भी दिए जाते है |

Returning Value

finditer() Function ये सभी non-overlapping matches; callable object में return करता है |

findall() और finditer() इन function में कोई फर्क नहीं होता है | लेकिन findall() function list को return करता है और finditer() function callable object को return करता है |

Example for finditer() Function in Python

Example में a से z तक case-sensitive सभी matches callable Object में return किये गए है |

Source Code :

```
import re
pattern = r"([a-z])"
str = "Hello 123 World 456"
a = re.finditer(pattern, str)
print(a)
for i in a:
    print(i.group(), end="")
```

Output :

```
<callable_iterator object at 0x027D3EF0>
elloorld
```

Example for finditer() Function with Flag in Python

Example में a से z तक case-insensitive सभी matches callable Object में return किये गए है |

Source Code :

```
import re
pattern = r"([a-z])"
str = "Hello 123 World 456"
a = re.finditer(pattern, str, re.IGNORECASE)
print(a)
for i in a:
    print(i.group(), end="")
```

Output :

```
<callable_iterator object at 0x03190C90>
HelloWorld
```

match() - Regular Expression Function

match() Function ये pattern के हिसाब से string के beginning पर ही match करके match object return करता है |

Syntax

```
re.match(pattern, string, flag(s))
```

Parameter

pattern : यहाँ पर pattern दिया जाता है |

string : यहाँ पर string दिया जाता है |

flag(s) : Optional. यहाँ पर modifier दिया जाता है | एक से ज्यादा modifiers भी दिए जाते है |

Note : अगर MULTILINE modifier दिया जाता है तो match() function सिर्फ string के beginning पर ही match करता है | किसी भी line के beginning पर match नहीं करता है |

Returning Value

match() Function; string के beginning पर ही match करके match object को return करता है | अगर match नहीं होता है तो 'None' return होता है |

Example for match() Function in Python

Example पर case-insensitive beginning के a और z के बीच का character match किया गया है |

Source Code :

```
import re
pattern = r"([a-z])"
str = "Hello 123 World 456"
a = re.match(pattern, str, re.IGNORECASE)
print(a)
print(a.group())
```

Output :

```
<_sre.SRE_Match object; span=(0, 1), match='H'>
H
```

Difference Between match() and search() Function

match() function string के beginning पर ही match करता है | match नहीं होता है तो 'None' return करता है |
search() function string के पहले occurrence को match करता है | match नहीं होता है तो 'None' return करता है |

Source Code :

```
import re
str = "Hello 123 World 456"
pattern = r"d"  #try '^d'(beginning) Quantifier
a = re.match(pattern, str)
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
b = re.search(pattern, str)
if b:
    print("search() :",b.group())
else:
    print("search() : No Match")
```

Output :

```
match() : No Match
search() : d
```

search() - Regular Expression Function

search() Function ये pattern के हिसाब से string के पहले occurrence को match करके match object return करता है |

Syntax

```
re.search(pattern, string, flag(s))
```

Parameter

pattern : यहाँ पर pattern दिया जाता है |

string : यहाँ पर string दिया जाता है |

flag(s) : Optional. यहाँ पर modifier दिया जाता है | एक से ज्यादा modifiers भी दिए जाते है |

Returning Value

search() Function; string के beginning पर ही match करके match object को return करता है | अगर match नहीं होता है तो 'None' return होता है |

Example for search() Function in Python

Example पर case-insensitive 'o' का पहला occurrence match किया गया है |

Source Code :

```
import re
str = "HellO 123 World 456"
pattern = r"o"
a = re.search(pattern, str, re.IGNORECASE)
print(a)
print(a.group())
```

Output :

```
<_sre.SRE_Match object; span=(4, 5), match='o'>
o
```

Difference Between search() and match() Function

match() function string के beginning पर ही match करता है | match नहीं होता है तो 'None' return करता है | search() function string के पहले occurrence को match करता है | match नहीं होता है तो 'None' return करता है |

Source Code :

```
import re
str = "Hello 123 World 456"
pattern = r"d"  #try '^d'(beginning) Quantifier
a = re.match(pattern, str)
if a:
    print("match() :",a.group())
else:
    print("match() : No Match")
b = re.search(pattern, str)
if b:
    print("search() :",b.group())
else:
    print("search() : No Match")
```

Output :

```
match() : No Match
search() : d
```

split() - Regular Expression Function

split() Function ये दिए हुए pattern द्वारा string को split करके अलग-अलग substrings को list में return किया जाता है |

Syntax

```
re.split(pattern, string, maxsplit, flag(s))
```

re module के split() function में pattern का इस्तेमाल delimiter के रूप में किया जाता है | OR(|) Operator के साथ एक से ज्यादा delimiters का इस्तेमाल pattern में किया जा सकता है |

Parameter

pattern : यहाँ पर pattern दिया जाता है |

string : यहाँ पर string दिया जाता है |

maxsplit : Optional. यहाँ पर कितने split करने है उसके हिसाब से pattern occurrences की संख्या दी जाती है | अगर pattern occurrences की संख्या invalid होती है तो पूरा string split किया जाता है |

flag(s) : Optional. यहाँ पर modifier दिया जाता है | एक से ज्यादा modifiers भी दिए जाते है |

Returning Value

split() Function; pattern द्वारा string को split करके अलग-अलग substrings को list में return किया जाता है |

Example for split() Function in Python

Example पर whitespace( ) से string को split किया गया है |

Source Code :

```
import re
str = "Python is good programming language"
pattern = r" "
a = re.split(pattern, str)
print(a)
```

Output :

```
['Python', 'is', 'good', 'programming', 'language']
```

Example for split() Function With maxsplit parameter in Python

Example पर whitespace( ) के पहले दो occurrences से string को split किया गया है |

Source Code :

```
import re
str = "Python is good programming language"
pattern = r" "
a = re.split(pattern, str, 2)
print(a)
```

Output :

```
['Python', 'is', 'good programming language']
```

Example for split() Function With maxsplit and Flag parameter in Python

Example पर p(case-insensitive) इस delimiter के पहले एक occurrence से string को split किया गया है |

Source Code :

```
import re
str = "Python is good prgramming language"
pattern = r"p"
a = re.split(pattern, str, 1, re.IGNORECASE)
print(a)
```

Output :

```
['', 'ython is good prgramming language']
```

sub() - Regular Expression Function

sub() Function ये दिए हुए pattern के हिसाब से string के substring(s) को replace करके नया string return करता है |

Syntax

```
re.sub(pattern, replace, string, maxreplace, flag(s))
```

Parameter

pattern : यहाँ पर pattern दिया जाता है |

replace : यहाँ पर pattern parameter पर दिए pattern के हिसाब से replace करने के लिए यहाँ पर string दिया जाता है |

string : यहाँ पर string दिया जाता है |

maxreplace : Optional. ज्यादा से ज्यादा कितने replace करने है वो number यहाँ पर दिया जाता है | अगर दिया नहीं जाता है तो default '0' होता है |

flag(s) : Optional. यहाँ पर modifier दिया जाता है | एक से ज्यादा modifiers भी दिए जाते है |

Returning Value

sub() Function; pattern के हिसाब से string के substring(s) को replace करके नया string return करता है |

Example for sub() Function in Python

Example पर a से z(case-sensitive) तक 'l' इस character से सभी pattern occurrences को replace करके replaced string को return किया गया है |

Source Code :

```
import re
str = "Python Is Good Programming Language"
pattern = r"[a-z]"
a = re.sub(pattern, 'l', str)
print(a)
```

Output :

```
Plllll Il Glll Plllllllll Llllllll
```

Example for sub() Function with maxreplace parameter in Python

Example पर a से z(case-sensitive) तक 'l' इस character से सिर्फ दो ही pattern occurrences replace करके replaced string को return किया गया है |

Source Code :

```
import re
str = "Python Is Good Programming Language"
pattern = r"[a-z]"
a = re.sub(pattern, 'l', str, 2)
print(a)
```

Output :

```
Pllhon Is Good Programming Language
```

Example for sub() Function with flag parameter in Python

Example पर a से z(case-insensitive) तक 'l' character से सभी pattern occurrences replace करके replaced string को return किया गया है |

Source Code :

```
import re
str = "Python Is Good Programming Language"
pattern = r"[a-z]"
a = re.sub(pattern, 'l', str, 0, re.IGNORECASE)
print(a)
```

Output :

```
llllll ll llll lllllllllll llllllll
```

subn() - Regular Expression Function

subn() Function ये दिए हुए pattern के हिसाब से string के substring(s) को replace करके replaced string और replaced हुए pattern occurrence(s) को tuple में return करता है |

Syntax

```
re.subn(pattern, replace, string, maxreplace, flag(s))
```

Parameter

pattern : यहाँ पर pattern दिया जाता है |

replace : यहाँ पर pattern parameter पर दिए pattern के हिसाब से replace करने के लिए यहाँ पर string दिया जाता है |

string : यहाँ पर string दिया जाता है |

maxreplace : Optional. ज्यादा से ज्यादा कितने replace करने है वो number यहाँ पर दिया जाता है | अगर दिया नहीं जाता है तो default '0' होता है |

flag(s) : Optional. यहाँ पर modifier दिया जाता है | एक से ज्यादा modifiers भी दिए जाते है |

Returning Value

subn() Function; pattern के हिसाब से string के substring(s) को replace करके replaced string और replaced हुए pattern occurrence(s) को tuple में return करता है |

tuple के पहले item में replaced string होता है और दुसरे item में pattern occurrence(s) number में होता है |

Example for subn() Function in Python

Example पर a से z(case-sensitive) तक 'l' इस character से सभी pattern occurrences को replace किया गया है |

Source Code :

```
import re
str = "Python Is Good Programming Language"
pattern = r"[a-z]"
a = re.subn(pattern, 'l', str)
print(a)
```

Output :

```
('Plllll Il Glll Pllllllllll Llllllll', 26)
```

Example for subn() Function with maxreplace parameter in Python

Example पर a से z(case-sensitive) तक 'l' इस character से सिर्फ दो ही pattern occurrences replace किया गया है |

Source Code :

```
import re
str = "Python Is Good Programming Language"
pattern = r"[a-z]"
a = re.subn(pattern, 'l', str, 2)
print(a)
```

Output :

```
('Pllhon Is Good Programming Language', 2)
```

Example for subn() Function with flag parameter in Python

Example पर a से z(case-insensitive) तक 'l' character से सभी pattern occurrences replace किया गया है |

Source Code :

```
import re
str = "Python Is Good Programming Language"
pattern = r"[a-z]"
a = re.subn(pattern, 'l', str, 0, re.IGNORECASE)
print(a)
```

Output :

```
('llllll ll llll lllllllllll llllllll', 31)
```

Methods- Regular Expression

Functions	Description
findall()	दिए हुए pattern के हिसाब से और starting से ending position तक match करके सभी non-overlapping matches को list में return करता है |
finditer()	दिए हुए pattern के हिसाब से और starting से ending position तक match करके सभी non-overlapping matches को callable object में return करता है |
match()	दिए हुए pattern के हिसाब से और starting से ending position तक के beginning के match को match object में return करता है |
search()	दिए हुए pattern के हिसाब से और starting से ending position तक के पहले occurrence को match करके match object में return करता है |
split()	दिए हुए pattern के हिसाब से string को split करके list में return करता है |
sub()	pattern के हिसाब से string के substring(s) को replace करके नया string return करता है |
subn()	pattern के हिसाब से string के substring(s) को replace करके replaced string और replaced हुए pattern occurrence(s) को tuple में return करता है |

Match Object Methods - Regular Expression

Functions	Description
group()	match के एक या एक से ज्यादा subgroups को return करता है |
groups()	match के एक या एक से ज्यादा subgroups को tuple में return करता है |
groupdict()	key और उसके matches के साथ dictionary को return करता है |
start()	दिए हुए group String पर जहा से match होता है वहा से उसकी index return करता है |
end()	दिए हुए group String पर जहा तक match होता है वहा से उसकी index return करता है |

group() - Regular Expression Match Object Method

group() Match Object Method; match के एक या एक से ज्यादा subgroups को return करता है |
pattern में हर group को parenthesis(()) के अन्दर close और अलग किया जाता है |

Syntax

```
match.group(group1,group2,..,groupN)
```

Parameter

group1,group2,..,groupN : Optional. यहाँ पर एक या एक से ज्यादा groups दिए जा सकते है |

Returning Value

group() Match Object Method; match के एक या एक से ज्यादा subgroups को return करता है |
अगर group() method पर एक से ज्यादा subgroups दिए जाते है तो वो सभी को tuple में return करता है |

अगर invalid या negative group number दिया जाता है तो 'IndexError' exception raise होता है |
group() method का इस्तेमाल match() और search() के लिए किया जाता है |

Source Code :

```
import re
str = "Python 123 Is Good Programming Language"
pattern = r"([a-z]+)\W( [0-9]+)\W([a-z]+)"
a = re.match(pattern, str, re.I)
if a:
    print(a.group(0))
    print(a.group(1))
    print(a.group(2))
    print(a.group(3))
    print(a.group(1, 2, 3))
else:
    print("No Match")
```

Output :

```
Python 123 Is
Python
123
Is
('Python', '123', 'Is')
```

groups() - Regular Expression Match Object Method

groups() Match Object Method; match के एक या एक से ज्यादा subgroups को tuple में return करता है | pattern में हर group को parenthesis(()) के अन्दर close और अलग किया जाता है |

Syntax

```
match.groups()
```

Parameter

groups() method के लिए कोई parameter नहीं होता है |

Returning Value

groups() Match Object Method; match के एक या एक से ज्यादा subgroups को tuple में return करता है |

groups() method का इस्तेमाल match() और search() के लिए किया जाता है |

Source Code :

```
import re
str = "Python 123 Is Good Programming Language"
pattern = r"([a-z]+)\W([0-9]+)\W([a-z]+)"
a = re.match(pattern, str, re.I)
if a:
    print(a.groups())
else:
    print("No Match")
```

Output :

```
('Python', '123', 'Is')
```

groupdict() - Regular Expression Match Object Method

groupdict() Match Object Method; key और उसके matches के साथ dictionary को return करता है |
pattern में हर group को parenthesis(()) के अन्दर close और अलग किया जाता है |

Syntax

```
match.groupdict()
```

Parameter

groupdict() method के लिए कोई parameter नहीं होता है |

Returning Value

groupdict() Match Object Method; key और उसके matches के साथ dictionary को return करता है |

groupdict() method का इस्तेमाल match() और search() के लिए किया जाता है |

Source Code :

```
import re
str = "Python 123 Is Good Programming Language"
pattern = r"(?P<key1>[a-z]+)\W(?P<key2>[0-9]+)\W(?P<key3>[a-z]+)"
a = re.match(pattern, str, re.I)
if a:
    print(a.groupdict())
else:
    print("No Match")
```

Output :

```
{'key1': 'Python', 'key2': '123', 'key3': 'Is'}
```

start() - Regular Expression Match Object Method

start() Match Object Method; दिए हुए group String पर जहा से match होता है वहा से उसकी index return करता है |

pattern में हर group को parenthesis(()) के अन्दर close और अलग किया जाता है |

Syntax

```
match.start(group)
```

Parameter

group : Optional. यहाँ पर group; number में दिया जाता है | अगर दिया नहीं जाता है तो '0' default होता है |

Returning Value

start() Match Object Method में दिए हुए group String पर जहा से match होता है वहा से उसकी index return की जाती है |

start() method का इस्तेमाल match() और search() के लिए किया जाता है |

Source Code :

```
import re
str = "Python 123 Is Good Programming Language"
pattern = r"([a-z]+)\W([0-9]+)\W([a-z]+)"
a = re.match(pattern, str, re.I)
if a:
    print(a.start())
    print(a.start(0))
    print(a.start(1))
    print(a.start(2))
    print(a.start(3))
else:
    print("No Match")
```

Output :

```
0
0
0
7
11
```

end() - Regular Expression Match Object Method

end() Match Object Method; दिए हुए group String पर जहा तक match होता है वहा से उसकी index return करता है |
pattern में हर group को parenthesis(()) के अन्दर close और अलग किया जाता है |

Syntax

```
match.end(group)
```

Parameter

group : Optional. यहाँ पर group; number में दिया जाता है | अगर दिया नहीं जाता है तो '0' default होता है |

Returning Value

end() Match Object Method में दिए हुए group String पर जहा तक match होता है वहा से उसकी index return की जाती है |

end() method का इस्तेमाल match() और search() के लिए किया जाता है |

Source Code :

```
import re
str = "Python 123 Is Good Programming Language"
pattern = r"([a-z]+)\W([0-9]+)\W([a-z]+)"
a = re.match(pattern, str, re.I)
if a:
    print(a.end())
    print(a.end(0))
    print(a.end(1))
    print(a.end(2))
    print(a.end(3))
else:
    print("No Match")
```

Output :

```
13
13
6
10
13
```

Classes and Objects

Python ये Object-Oriented Programming(Objects) Language है | इसके साथ-साथ ये Procedural-Oriented Programming(Functions) Language भी है |
ज्यादातर Programming Languages में OOP का concept होता है | Python के Object में variables, functions या methods होते है |
जैसे कि, अगर कोई प्राणी है तो उस प्राणी का behavior और properties जैसे कि, उसका भौकना, चलना, देखना , उसके शरीर की रचना को variables और functions या methods के जरिये लिखा जाता है |हर एक से अधिक प्राणी को अलग-अलग नाम से उनके objects भी बनाये जाते है |

What is a Class ?

Class के अन्दर कुछ functions या methods होते है और उससे निगडित कुछ variables दिए जाते है | उन functions और variables को एक ही class पर इकट्ठा किया जाता है और उन data को access करने के लिए उस class का object बनाया जाता है |
Class अपने data को hold करने का काम करता है |
Class ये Object की blueprint या layout होता है |
एक Class के एक या एक से ज्यादा Object create किये जा सकते है |

Defining Class in Python

class को create करने के लिए पहले 'class' keyword का इस्तेमाल किया जाता है |

Syntax

```
class MyClass:
    "I am a Docstring"
    #class_body
```

Syntax पर 'myClass' ये class का नाम है |
उसके बाद एक docstring लिया गया है | ये docstring class में optional होता है | docstring ये class के बारे में कुछ जानकारी देने के लिए लिया जाता है |
उसके बाद class_body में कुछ statements के रूप में कुछ variables, functions या methods हो सकते है |

Example for Creating a Class

Example पर Class के नाम से ही Object का इस्तेमाल किया गया है |इस class object; का इस्तेमाल class के अलग-अलग attributes access करने के लिए किया जाता है | निचे docstring को access करने के लिए '__doc__' इस class attribute का इस्तेमाल किया गया है और उसके बाद func ये attribute function object को return करता है |

Source Code :

```
class MyClass:          # Class Name
    "I am a Docstring"
    def func(self):
        print("Hello World")
print(MyClass.__doc__)   # docstring attribute
print(MyClass.func)      # func attribute
```

Output :

```
I am a Docstring
<function MyClass.func at 0x02423270>
```

Creating an Object in Python

Object को variable के रूप में ही create किया जाता है | उस variable पर जैसे function को call किया जाता है वैसे ही उस class को function के रूप में variable पर store किया जाता है |

Syntax

```
Object_Name = Class_Name()
```

Example पर MyClass इस Class का Object 'obj' बनाया गया है |

Source Code

```
class MyClass:
    "I am a Docstring"
var = 1
    def func(self):
        print("Hello World")
obj = MyClass()  #Class Object
```

Accessing Class Variable and Function

Source Code :

```
class MyClass:
    "I am a Docstring"
    var = 1
    def func(self):
        print("Hello World")
obj = MyClass() #Object
print("MyClass.var :",MyClass.var)  #1
#same as
print("obj.var :",obj.var)         #2
print("MyClass.func :",MyClass.func)#3
print("obj.func :",obj.func)        #4
MyClass.func(obj)                  #5
#same as
obj.func()                         #6
```

Output :

```
MyClass.var : 1
obj.var : 1
MyClass.func : <function MyClass.func at 0x02993270>
obj.func : <bound method MyClass.func of <__main__.MyClass object at
0x0278AD50>>
Hello World
Hello World
```

Explaination of Above Example

Comment#1 : यहाँ पर Class name वाले Object(MyClass) से class variable को access किया गया है |

Comment#2 : यहाँ पर Class के बनाये हुए Object(obj) से class variable को access किया गया है |

Comment#3 : यहाँ पर Class name वाले Object(MyClass) से func इस class function को access किया गया है लेकिन ये 'function object' को return करता है |

Comment#4 : यहाँ पर Class के बनाये हुए Object(obj) से func इस class function को access किया गया है लेकिन ये 'method object' को return करता है |

Comment#5 : यहाँ पर MyClass इस class के function पर जो definition पर 'self' इस parameter को pass किया गया है वैसे ही 'obj' इस class के object को argument पर pass किया गया है |

Comment#6 : definition पर 'self' ये parameter pass किया गया है लेकिन यहाँ पर कोई भी argument को pass नहीं किया गया है इसका मतलब है कि जब बनाये हुए object द्वारा function को access किया जाता है तब खुद object ही argument पर pass हो जाता है | उसे अलग से देने की जरुरत होती है |

Class Attribute/Variable and Instance Attribute/Variable

Example पर class variable 'classVar' और instance variable 'inVar' लिया गया है |
class Variable अपने className या उसके object के जरिये access किये जा सकते है लेकिन instance variable को access करने के लिये class के object की जरुरत पड़ती है |
Instance Variable को constructor के अन्दर create किया जाता है |

Source Code :

```
class MyClass:
    classVar = 1 #classVar is a Class Variable

    def __init__(self, inVar):
        self.inVar = inVar   #inVar is a Instance Variable
obj1 = MyClass("Ramesh")
obj2 = MyClass("Rahul")

print("Access Class Variable using Class Name :", MyClass.classVar)
print("Access Class Variable using Class Object :", obj1.classVar)
print(obj1.classVar, obj1.inVar) #Value of inVar is Ramesh
print(obj2.classVar, obj2.inVar) #Value of inVar is Rahul
```

Output :

```
Access Class Variable using Class Name : 1
Access Class Variable using Class Object : 1
1 Ramesh
1 Rahul
```

Changing Class Variable's and Instance Variable's Values

Class के variable की value change करने के लिए ClassName या ClassObject को इस्तेमाल किया जा सकता है लेकिन instance variable की value को change करना हो तो ClassObject का ही इस्तेमाल किया जाता है |

Source Code :

```
class MyClass:
    classVar = 1 #classVar is a Class Variable

    def __init__(self, inVar):
        self.inVar = inVar   #inVar is a Instance Variable
obj1 = MyClass("Ramesh")
obj2 = MyClass("Rahul")
print("Before Changing :")
print(obj1.classVar, obj1.inVar)

MyClass.classVar = 2 #Changing Class Variable's Value by ClassName
obj1.inVar = "Rakesh" #Changing Instance Variable's Value
print("After Changing :")
print(obj1.classVar, obj1.inVar)
```

Output :

```
Before Changing :
1 Ramesh
After Changing :
2 Rakesh
```

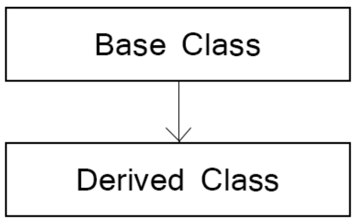
Inheritance

Inheritance Introduction

Inheritance OOP(Object-Oriented Programming) का एक हिस्सा है | Inheritance में एक से ज्या classes होते है |

Inheritance में दो प्रकार के मुख्य class देना अनिवार्य होता है |

- **Base Class(Parent)** : Base class को super या parent class भी कहा जाता है |
- **Derived Class(Child)** : Derived class को sub या child class भी कहा जाता है | Derived class ये Base class की properties या attributes को inherit करता है |



Syntax for Inheritance/Single Inheritance

```
class Base:
    "This is a Docstring(Optional)"
    Base_Class_Body
class Derived(Base):
    "This is a Docstring(Optional)"
    Derived_Class_Body
```

Example For Inheritance in Python

Example पर Fitness Class के obj इस object से Employee Class की properties को inherit किया गया है |

Source Code :

```
class Employee:
"Class Employee"
    def set1(self,empid,name,salary):
        self.empid = empid
        self.name = name
        self.salary = salary
class Fitness(Employee):
    "Class Fitness"
def set2(self,height,weight):
        self.height = height
        self.weight = weight
    def display(self):
        print("id is",self.empid)
        print("name is",self.name)
        print("salary is",self.salary,"Rs")
        print("height is",self.height,"cm")
        print("weight is",self.weight,"Kg")
obj = Fitness()
obj.set1(1,"Rakesh",27000)
obj.set2(176,60)
     obj.display()
```

Output :

```
id is 1
name is Rakesh
salary is 27000 Rs
height is 176 cm
weight is 60 Kg
```

issubclass() and isinstance() functions in Python

```
issubclass(subclass, superclass)
     isinstance(object, class)
```

issubclass() ये function दिए हुए superclass का दिया हुआ subclass है या नही ये boolean value में return करता है |

isinstance() ये function दिए हुए class का दिया हुआ object है या नही ये boolean value में return करता है |

Source Code :

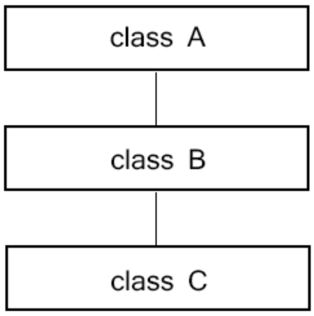
```
class Employee:
    pass
class Fitness(Employee):
    pass
obj1 = Fitness()
obj2 = Employee()
print(issubclass(Fitness, Employee))
print(isinstance(obj1, Fitness))
     print(isinstance(obj2, Fitness))
```

Output :

```
True
True
False
```

Multilevel Inheritance

Multilevel Inheritance में एक base class और दो derived class होते है |
Multilevel Inheritance में derived class का एक base class होता है और derived class का भी एक derived class होता है |



Syntax for Multilevel Inheritance

```
class Base:
     Base_Class_Body
class Derived1(Base):
     Derived1_Class_Body
class Derived2(Derived1):
     Derived2_Class_Body
```

Example for Multilevel Inheritance

Example पर Employee class के Fitness और Company ये दो derived class है |
उसके बाद Fitness इस class का एक base class(Employee) और एक derived class(Company) है |
उसके बाद Company इस class के Fitness और Employee ये दो base class है | Company class ये Fitness और Employee की properties को inherit करता है |
सिर्फ Company class का object Fitness और Employee इन दोनों की properties को access करता है |

Source Code :

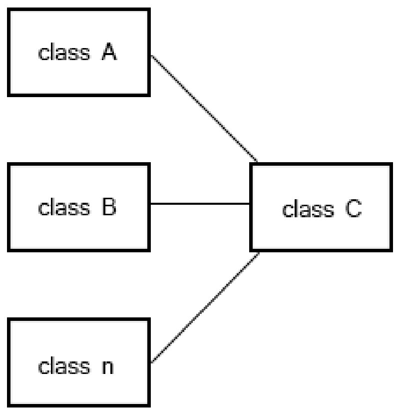
```
#Base class of Fitness and Company
class Employee:
    def set1(self,empid,name,salary):
        self.empid = empid
        self.name = name
        self.salary = salary
#Base class of Company and Derived class of Employee
class Fitness(Employee):
    def set2(self,height,weight):
        self.height = height
        self.weight = weight
#Derived class of Fitness and Employee
class Company(Fitness):
    def set3(self,company,dep):
        self.company = company
        self.dep = dep
    def display(self):
        print("id :",self.empid)
        print("name :",self.name)
        print("salary :",self.salary,"Rs")
        print("height :",self.height,"cm")
        print("weight :",self.weight,"kg")
        print("Company :",self.company)
        print("Department :",self.dep)
obj = Company()
obj.set1(1,"Rakesh",27000)
obj.set2(176,60)
obj.set3("Infosys", "IT")
     obj.display()
```

Output :

```
id : 1
name : Rakesh
salary : 27000 Rs
height : 176 cm
weight : 60 kg
Company : Infosys
Department : IT
```

Multiple Inheritance

Multiple Inheritance में एक से अधिक base classes हो सकते है और एक derived class होता है |
Derived class को दो या दो से ज्यादा base classes को inherit किया जाता है, उसे Multiple Inheritance कहते है |



Multiple Inheritance में एक Child Class और उसके एक से ज्यादा Parent Classes को inherit करता है |

Syntax for Multiple Inheritance

```
class Base1:
    Base1_Class_Body
class Base2:
    Base2_Class_Body
class Derived(Base2,Base1):
    Derived_Class_Body
```

Example for Multiple Inheritance

Example पर Employee और Fitness ये दो Base class है और उनका एक ही derived class है जो दोनों base class की properties को inherit करता है |

Source Code :

```
#Base class of Company
class Employee:
    def set1(self,empid,name,salary):
        self.empid = empid
        self.name = name
        self.salary = salary
#Base class of Company
class Fitness:
    def set2(self,height,weight):
        self.height = height
        self.weight = weight
#Derived class of Fitness and Employee
class Company(Fitness,Employee):
    def set3(self,company,dep):
        self.company = company
        self.dep = dep
    def display(self):
        print("id :",self.empid)
        print("name :",self.name)
        print("salary :",self.salary,"Rs")
        print("height :",self.height,"cm")
        print("weight :",self.weight,"kg")
        print("Company :",self.company)
        print("Department :",self.dep)
obj = Company()
obj.set1(1,"Rakesh",27000)
obj.set2(176,60)
obj.set3("Infosys", "IT")
      obj.display()
```

Output :

```
id : 1
name : Rakesh
salary : 27000 Rs
height : 176 cm
weight : 60 kg
Company : Infosys
Department : IT
```

Class Attributes and Functions

Class Attributes

Python में हर class के साथ In-Built Class Attributes होते है |

- **__doc__** : ये दिए गए class की docstring को return करता है अगर docstring नहीं होती है तो 'None' return होता है |
- **__name__** : ये दिए गए class का नाम return करता है |
- **__module__** : जहा पर class होता वो module name return किया जाता है अगर class current program पर होता है तो '__main__' return होता है |
- **__bases__** : ये दिए गए class का base class(es) tuple में return करता है | अगर कोई base class नहीं होता है तो empty tuple return करता है |
- **__dict__** : ये दिए गए class के attributes को dictionary में return करता है |

Source Code :

```
class Employee:
    "I am in class Employee"
    pass
class Fitness(Employee):
    "I am in class Fitness"
    pass
class Company(Fitness):
    "I am in class Company"
    pass

print("Docstring of class Company :",Company.__doc__)
print("Class Name :",Company.__name__)
print("Module Name :",Company.__module__)
print("Base Class of Company :",Company.__bases__)
     print("Company's class attributes :",Company.__dict__)
```

Output :

```
Docstring of class Company : I am in class Company
Class Name : Company
Module Name : __main__
Base Class of Company : (,)
Company's class attributes : {'__module__': '__main__', '__doc__': 'I am in class
Company'}
```

Example for All Class Attributes in Python

Source Code :

```
class Employee:
    "I am in class Employee"
    pass
class Fitness(Employee):
    "I am in class Fitness"
    pass
class Company(Fitness):
    "I am in class Company"
    pass

obj = Employee()
for attr in dir(Employee):
   if attr.startswith('__'):
          print(attr)
```

Output :

```
__class__
__delattr__
__dict__
__dir__
__doc__
__eq__
__format__
__ge__
__getattribute__
__gt__
__hash__
__init__
__init_subclass__
__le__
__lt__
__module__
__ne__
__new__
__reduce__
__reduce_ex__
__repr__
__setattr__
__sizeof__
__str__
__subclasshook__
__weakref__
```

Functions for Class Attributes

Functions	Description
delattr()	object के attribute को delete करने के लिए इस्तेमाल किया जाता है |
getattr()	object के attribute की value को return करता है |
hasattr()	object का attribute है या नहीं ये check boolean value में return करता है |
setattr()	object के attribute के set किया जाता है अगर पहले ही set होता है तो उसे overwrite किया जाता है लेकिन ये अलग-अलग memory share करता है | ये None return करता है |

delattr() Function

Syntax

```
delattr(object, attribute)
```

Source Code :

```
class MyClass:
    a = 5
obj = MyClass()
print(obj.a)
delattr(obj, 'a') #same as   del obj.a

#Output :
#     delattr(obj, 'a')
#AttributeError: a
```

getattr() Function

Syntax

```
getattr(object, attribute)
```

Source Code :

```
class MyClass:
    a = 5
obj = MyClass()
print(getattr(obj, 'a'))
#same as
print(obj.a)

#Output
#5
#5
```

hasattr() Function

Syntax

```
hasattr(object, attribute)
```

Source Code :

```
class MyClass:
    a = 5
obj = MyClass()

print(hasattr(obj, 'a'))
#same as
print(bool(obj.a))
#Output
#True
#True

print(hasattr(obj, 'b')) #return False if attribute is not set
#not same as
print(bool(obj.b)) #raise AttributeError if attribute is not set
#Output
#False
#AttributeError
```

setattr() Function

Syntax

```
setattr(object, attribute, attr_value)
```

Source Code :

```
class MyClass:
    a = 5

obj = MyClass()

setattr(obj, 'a', 8)
print(getattr(obj, 'a'))
#Output
#8

setattr(obj, 'b', 10)
print(getattr(obj, 'b'))
#Output
#10
```

Constructor

Python के अलावा C++ और Java में भी Constructor होता है लेकिन उन constructor में खुद class के नाम से ही constructor को बनाया जाता है |
लेकिन Python में Constructor को create करने के लिए __init__() function का इस्तेमाल किया जाता है |
जब class में __init__() function define किया जाता है और उस class का object बनाया जाता है तब ये function automatically call हो जाता है | उसे अलग से call करने की जरुरत नहीं पड़ती है |

Syntax

```
class className:
    class_body(Optional)
    def __init__(parameter(s)):   #Constructor
        constructor_body
    class_body(Optional)
```

Example for Constructor in Python

Example में MyClass में Constructor को define किया गया है और उसके बाद class के 'obj' call किया गया है |
object create होते ही constructor call होता है |
Constructor का इस्तेमाल सामान्यतः variables को initialize करने के लिए किया जाता है |

Source Code :

```
class MyClass:
    def __init__(self, a, b):
        self.a = a
        self.b = b
        print(self.a,self.b)
        print("Constructor invoked")

obj = MyClass(4, "Hello")
```

Output :

```
4 Hello
Constructor invoked
```

Destructor

C++ और Java में destructor को ~(tilde) sign के साथ class के नाम की जरुरत पड़ती और वो object बनते ही अपने आप call होता है |
लेकिन Python में Destructor के लिए '____del___()' function का इस्तेमाल किया जाता है और जब class का object बनाया जाता है तब वो automatically call नहीं होता है |
Destructor को destroy करने के लिए 'del' operator से object को delete करना पड़ता है |

Example for Destructor in Python

Source Code :

```
class MyClass:
    def __init__(self, a, b):
        self.a = a
        self.b = b
        print(self.a,self.b)
        print("Constructor invoked")
    def __del__(self):
        print("Destructor invoked")

if __name__ == "__main__":
    obj = MyClass(4, "Hello")
    del obj
```

Output :

```
4 Hello
Constructor invoked
Destructor invoked
```

Method Overriding

Method Overriding ये OOP का एक हिस्सा है | Method Overriding का इस्तेमाल inheritance में किया जाता है | इसमे एक ही नाम के function को अलग-अलग class में अलग-अलग definition दी जाती है |
Method Overriding में function को override किया जाता है |

Syntax for Method Overriding

```
class Base:
    def func(self):
    func_body
class Derived(Base):
    def func(self):
    func_body
```

Example for Method Overriding

Example पर func() इस function को override किया गया है |

Source Code :

```
class MyClass1:
    def func(self):
        a = 10
        print("Value of a :",a)

class MyClass2(MyClass1):
    def func(self):
        b = 5
        print("Value of b :",b)

obj = MyClass2()
obj.func()
```

Output :

```
Value of b : 5
```

Try to Different Type of Method Overriding in Python

अगर आप C++ या Java का Method Overriding पढ़े होंगे तो निचे दिया हुआ Method Overriding का example बिलकुल support करता है |
लेकिन इस प्रकार का example; python में support नहीं करता है |
C++ या Java में अगर overriding में एक ही function का type अलग-अलग हो तो उस type का function; class पर ढूंढ के execute करता है |
लेकिन Python में एक ही function का अलग-अलग type हो तो, जो आखिरी derived class पर function का type होता है वही type execute किया जाता है |

Source Code :

```
class MyClass1:
    def func(self):
        a = 10
        print("Value of a :",a)
class MyClass2(MyClass1):
    def func(self, b):
        self.b = b
        print("Value of a :",self.b)

obj = MyClass2()
obj.func(20)
obj.func()
```

Output :

```
Value of a : 20
    obj.func()
TypeError: func() missing 1 required positional argument: 'b'
```

Try to Different Type of Method Overriding in Python

ऊपर दिए हुए example में और यह example में फर्क यह है कि,
सिर्फ MyClass1 और MyClass2 की positions को change किया गया है | MyClass1 ये derived class है और MyClass2 ये base class है |

Source Code :

```
class MyClass2:
    def func(self, b):
        self.b = b
        print("Value of a :",self.b)
class MyClass1(MyClass2):
    def func(self):
        a = 10
        print("Value of a :",a)

obj = MyClass1()
obj.func()
obj.func(20)
```

Output :

```
Value of a : 10
    obj.func(20)
TypeError: func() takes 1 positional argument but 2 were given
```

Operator Overloading

Python में किसी विशिष्ट Operator का मतलब या उसके बर्ताव को बदलने के लिए Operator Overloading किया जाता है |
जैसे कि, अभीतक देखा है कि '+' Operator से दो Numbers का addition, दो string का concatenate और दो sequences(list,tuple) का addition/merge देखा है , लेकिन
क्या दो class objects को '+' operator से add किया जा सकता है ? Example देखिये |

Trying to add Two Objects Without Overloading

Example पर देखे तो Operator Overloading के सिवाय दो class objects को add करने का प्रयास किया गया है | लेकिन A class के objects unsupported operands type होने के कारण 'TypeError' का exception raise होता है |

Source Code :

```
class A:
    def __init__(self, a):
        self.a = a
        print(self.a)

obj1 = A(5)
obj2 = A(2)
print(obj1 + obj2)
```

Output :

```
5
2
    print(obj1 + obj2)
TypeError: unsupported operand type(s) for +: 'A' and 'A'
```

अगर दो objects का addition करना हो तो Operator Overloading का इस्तेमाल करना पड़ता है |

Example for '+' Operator Overloading in Python

Operator Overloading में हर Operator के लिए अलग-अलग function का इस्तेमाल किया जाता है |
जैसे कि,निचे दिए गए example में '+' operator से दो objects को add करने के लिए '__add__()' function का इस्तेमाल किया गया है |

Source Code :

```
class A:
    def __init__(self, a):
        self.a = a
    def disp(self):
        return self.a

    def __add__(self, param):
        return A(self.a + param.a)

obj1 = A(5)
obj2 = A(2)
c = obj1 + obj2
print(c.disp())
```

Output :

```
7
```

Functions for Operator Overloading

Operators	**Functions**	**Functions**
+	__add__()	Addition
-	__sub__()	Subtraction
*	__mul__()	Multiplication
/	__truediv__()	Division
%	__mod__()	Modulus
//	__floordiv__()	Floor Division
**	__pow__()	Exponent
<	__lt__()	Less than
<=	__le__()	Less than or Equal to
>	__gt__()	Greater than

>=	__ge__()	Greater than or Equal to
==	__eq__()	Equal to
!=	__ne__()	Not Equal to
<<	__lshift__()	Left Shift
>>	__rshift__()	Right Shift
&	__and__()	Bitwise AND
|	__or__()	Bitwise OR
^	__xor__()	Bitwise XOR
~	__invert__()	Bitwise NOT
index	__getitem__(self, index)	Index
str	__str__()	String
len	__len__()	Length

Other Example for Operator Overloading

Source Code :

```
class A:
    def __init__(self, a):
        self.a = a
    def disp(self):
        return self.a

    def __add__(self, param):
        return A(self.a + param.a)
    def __sub__(self, param):
        return A(self.a - param.a)
    def __mul__(self, param):
        return A(self.a * param.a)
    def __truediv__(self, param):
        return A(self.a / param.a)
    def __mod__(self, param):
        return A(self.a % param.a)
    def __floordiv__(self, param):
        return A(self.a // param.a)
    def __pow__(self, param):
        return A(self.a ** param.a)
    def __lt__(self, param):
        return A(self.a < param.a)
    def __le__(self, param):
        return A(self.a <= param.a)
    def __gt__(self, param):
        return A(self.a > param.a)
    def __ge__(self, param):
        return A(self.a >= param.a)
    def __eq__(self, param):
        return A(self.a == param.a)
    def __ne__(self, param):
        return A(self.a != param.a)
```

```
obj1 = A(5)
obj2 = A(2)
c = obj1 + obj2                #or obj1.__add__(obj2)
print("+ Operator = ",c.disp())
c = obj1 - obj2                #or obj1.__sub__(obj2)
print("- Operator = ",c.disp())
c = obj1 * obj2                #or obj1.__mul__(obj2)
print("* Operator = ",c.disp())
c = obj1 / obj2                #or obj1.__truediv__(obj2)
print("/ Operator = ",c.disp())
c = obj1 % obj2                #or obj1.__mod__(obj2)
print("% Operator = ",c.disp())
c = obj1 // obj2               #or obj1.__floordiv__(obj2)
print("// Operator = ",c.disp())
c = obj1 ** obj2               #or obj1.__pow__(obj2)
print("** Operator = ",c.disp())
c = obj1 < obj2                #or obj1.__lt__(obj2)
print("< Operator = ",c.disp())
c = obj1 <= obj2               #or obj1.__le__(obj2)
print("<= Operator = ",c.disp())
c = obj1 > obj2                #or obj1.__gt__(obj2)
print("> Operator = ",c.disp())
c = obj1 >= obj2               #or obj1.__ge__(obj2)
print(">= Operator = ",c.disp())
c = obj1 == obj2               #or obj1.__eq__(obj2)
print("== Operator = ",c.disp())
c = obj1 != obj2               #or obj1.__ne__(obj2)
print("!= Operator = ",c.disp())
```

Output :

```
+ Operator =  7
- Operator =  3
* Operator =  10
/ Operator =  2.5
% Operator =  1
// Operator =  2
** Operator =  25
< Operator =  False
<= Operator =  False
> Operator =  True
>= Operator =  True
== Operator =  False
!= Operator =  True
```