Python – Session 2

Ayyappan Murugesan

Phone: 7010774329

Whatsapp: 8870483557

Topics

- Operator Precedance
- Number
- List
- Set
- Tuples
- Dictionary
- String
- Conditional & Loops
- Functions
- Exceptions & Handling

Operator Precedence

Operators	Usage
{ }	Parentheses (grouping)
f(args…)	Function call
x[index:index]	Slicing
x[index]	Subscription
x.attribute	Attribute reference
**	Exponent
~x	Bitwise not
+x, -x	Positive, negative
*, /, %	Product, division, remainder
+, –	Addition, subtraction
<<, >>	Shifts left/right
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR

Operator Precedence

in, not in, is, is not, <, <=, >, >=, <>, !=, ==	Comparisons, membership, identity
not x	Boolean NOT
and	Boolean AND
or	Boolean OR

Python Numbers

```
>>> x = 9
>>> type(x)
<type 'int'>
```

```
>>> x = 9999999999
>>> type(x) # In Py
irrespective of the
<type 'long'>
```

```
>>> x = 9.999
>>> type(x)
<type 'float'>
```

```
>>> x = 3 + 4j
>>> type(x)
<class 'complex'>
>>> x.real
3.0
>>> x.imag
4.0
```

```
>>> x = 0b101
>>> print(x)
5
>>> type(x)
<type 'int'>
>>> print(0b101 + 5)
10
>>> print(0o123)
83
>>> type(0x10)
<type 'int'>
```

isinstance

- If you want to test the class type of a number in Python, then you should use the isinstance() function.

```
isinstance(object, class)
```

```
>>> isinstance(2.2, float)
True
```

Decimal & Arithmetic operators

```
import decimal

print(0.28)

print(decimal.Decimal(0.28))

print(decimal.Decimal('5.30'))
```

```
>>> divmod(7, 2)
(3, 1)
>>> 7 % 2
1
>>> 7 / 2
3.5
>>> 7 // 2
3
```

Mathematics

import math

Function	Description
abs(x)	The absolute value of x: the (positive) distance between x and zero.
ceil(x)	The ceiling of x: the smallest integer not less than x
cmp(a, b)	-1 if a < b, 0 if a == b, or 1 if a > b
exp(x)	The exponential of x: ex
floor(x)	The floor of x: the largest integer not greater than x
log(x)	The natural logarithm of x, for x> 0
log10(x)	The base-10 logarithm of x for x> 0.
max(x1, x2,…)	The largest of its arguments: the value closest to positive infinity
min(x1, x2,…)	The smallest of its arguments: the value closest to negative infinity
modf(x)	The fractional and integer parts of x in a two-item tuple. Both parts share the same sign as x. The integer part coerces into a float.
pow(x, y)	The value of x**y
round(x [,n])	x rounded to n digits from the decimal point.
sqrt(x)	The square root of x for x > 0
pi	The mathematical constant pi.
e	The mathematical constant e.

Create A List In Python

```
# blank list
L1 = []

# list of integers
L2 = [10, 20, 30]

# List of heterogenous data types
L3 = [1, "Hello", 3.4]
```

```
>>> theList = list() #empty list
>>> len(theList)
0
```

List Comprehension

```
#Syntax - How to use List Comprehension

theList = [expression(iter) for iter in oldList if filter(iter)]
```

```
theList = [iter for iter in range(5)]
print(theList)
```

```
listofCountries = ["India","America","England","Germany","Brazil","Vietnam"]
firstLetters = [ country[0] for country in listofCountries ]
print(firstLetters)
```

```
print ([x+y for x in 'get' for y in 'set'])
```

```
print ([x+y for x in 'get' for y in 'set' if x != 't' and y != 'e' ])
```

```
>>> months = ['jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct',
'nov', 'dec']
>>> oddMonths = [iter for index, iter in enumerate(months) if (index%2 == 0)]
>>> oddMonths
```

```
>>> mylist=[10,20,30]
>>> myenu=enumerate(mylist)
>>> list(myenu)
[(0, 10), (1, 20), (2, 30)]
```

Multi-Dimensional List

```
>>> init_list = [0]*3
>>> print(init_list)
[0, 0, 0]
```

```
>>> two_dim_list = [ [0]*3 ] *3
>>> print(two_dim_list)
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
>>> two_dim_list[0][2] = 1
>>> print(two_dim_list)
[[0, 0, 1], [0, 0, 1], [0, 0, 1]]
```

List Extension

```
>>> L1 = ['a', 'b']
>>> L2 = [1, 2]
>>> L3 = ['Learn', 'Python']
>>> L1 + L2 + L3
['a', 'b', 1, 2, 'Learn', 'Python']
```

```
>>> L1 = ['a', 'b']
>>> L2 = ['c', 'd']
>>> L1.extend(L2)
>>> print(L1)
['a', 'b', 'c', 'd']
```

```
>>> L1 = ['x', 'y']
>>> L1.append(['a', 'b'])
>>> L1
['x', 'y', ['a', 'b']]
```

Slicing A Python List

```
#The Python slicing operator syntax

[start(optional):stop(optional):step(optional)]

Start -> The starting index (By default included in the slice output)

Stop  -> The closing index (Excluded from the slice output)

Step  -> Tells how many values to exclude from the end. The default is 1.
```

```
>>> mylist=[10,20,30,40,50]
>>>
>>> len(mylist)
5
>>>
>>> mylist[0]
10
>>>
>>> mylist[1]
20
>>>
>>> mylist[-1]
50
>>>
>>> mylist[-2]
40
```

```
>>> mylist[2:4]
[30, 40]
>>>
>>>
>>> mylist[1:4:2]
[20, 40]
```

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

```
>>> theList[2:5]
```

```
>>> theList[2:5:2]
```

```
>>> theList[2:-1]
```

```
>>> theList[:2]
```

```
>>> theList[2:]
```

```
>>> theList[::-1]
```

```
>>> theList[::-2]
```

```
>>> theList[::2]
```

Iterate A List In Python

```
for element in theList:
 print(element)
```

```
for index, element in enumerate(theList):
 print(index, element)
```

```
for index in range(len(theList)):
 print(index)
```

```
it = iter(theList)
element = it.next() # fetch first value
element = it.next() # fetch second value
```

Python List Methods

List Methods	Description
append()	It adds a new element to end of the list.
extend()	It extends a list by adding elements from another list.
insert()	It injects a new element at the desired index.
remove()	It deletes the desired element from the list.
pop()	It removes as well as returns an item from the given position.
clear()	It flushes out all elements of a list.
index()	It returns the index of an element that matches first.
count()	It returns the total no. of elements passed as an argument.
sort()	It orders the elements of a list in an ascending manner.
reverse()	It inverts the order of the elements in a list.
copy()	It performs a shallow copy of the list and returns.

List Built-In Functions

Function	Description
all()	It returns True if the list has elements with a True value or is blank.
any()	If any of the members has a True value, then it also returns True.
enumerate()	It returns a tuple with an index and value of all the list elements.
len()	The return value is the size of the list.
list()	It converts all iterable objects and returns as a list.
max()	The member which has the maximum value.
min()	The member which has the minimum value.
sorted()	It returns the sorted copy of the list.
sum()	The return value is the aggregate of all elements of a list.

Set In Python

```
# create a set of numbers
py_set_num = {3, 7, 11, 15}
print(py_set_num)

# create a set of mixed data types
py_set_mix = {11, 1.1, "11", (1, 2)}
print(py_set_mix)
```

```
# set can't store duplicate elements
py_set_num = {3, 7, 11, 15, 3, 7}
# it'll automatically filter the duplicates
print(py_set_num)

# create a set using the set() method
# creating set with a fixed set of elements
py_set_mix = set([11, 1.1, "11", (1, 2)])
print(py_set_mix)

# creating set with dynamic elements
py_list = [11, 1.1, "11", (1, 2)]
py_list.append(12)
print(py_list)
py_set_mix = set(py_list)
print(py_set_mix)
```

Empty Set

```
# Let's try to create an empty Python set
py_set_num = {}
print("The value of py_set_num:", py_set_num)
print("The type of py_set_num:", type(py_set_num))

py_set_num = set()
print("The value of py_set_num:", py_set_num)
print("The type of py_set_num:", type(py_set_num))
```

```
# output
The value of py_set_num: {}
The type of py_set_num: <class 'dict'>
The value of py_set_num: set()
The type of py_set_num: <class 'set'>
```

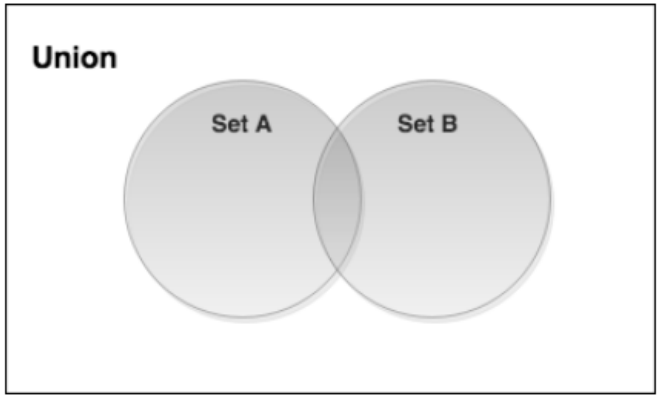
```python
# Let's try to change a Python set
py_set_num = {77, 88}

try:
    print(py_set_num[0])
except Exception as ex:
    print("Error in py_set_num[0]:", ex)

print("The value of py_set_num:", py_set_num)

# Let's add an element to the set
py_set_num.add(99)
print("The value of py_set_num:", py_set_num)

# Let's add multiple elements to the set
py_set_num.update([44, 55, 66])
print("The value of py_set_num:", py_set_num)

# Let's add a list and a set as elements
py_set_num.update([4.4, 5.5, 6.6], {2.2, 4.4, 6.6})
print("The value of py_set_num:", py_set_num)
```

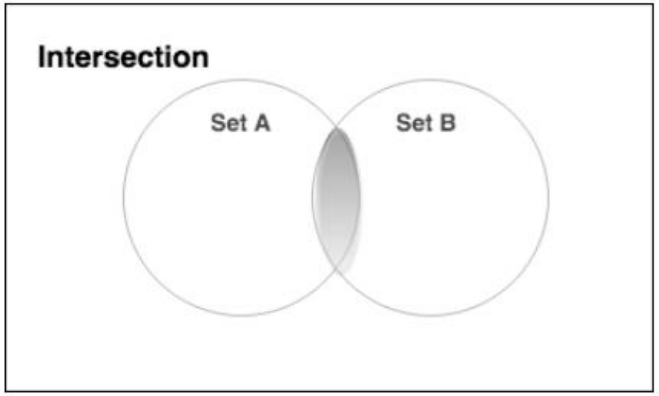
```python
# Let's try to use a Python set
py_set_num = {22, 33, 55, 77, 99}

# discard an element from the set
py_set_num.discard(99)
print("py_set_num.discard(99):", py_set_num)

# remove an element from the set
py_set_num.remove(77)
print("py_set_num.remove(77):", py_set_num)

# discard an element not present in the set
py_set_num.discard(44)
print("py_set_num.discard(44):", py_set_num)

# remove an element not present in the set
try:
    py_set_num.remove(44)
except Exception as ex:
    print("py_set_num.remove(44) => KeyError:", ex)
```

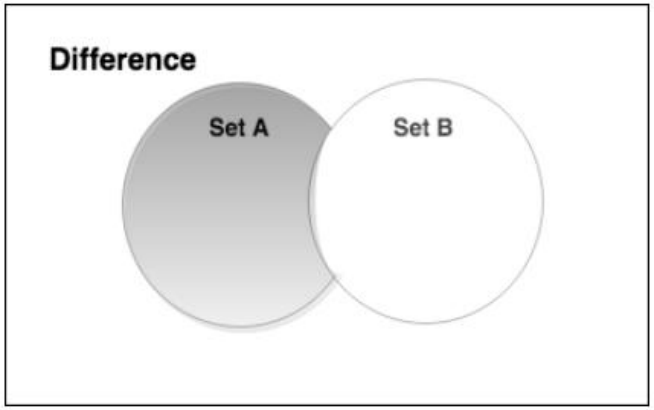
Pop & Clear

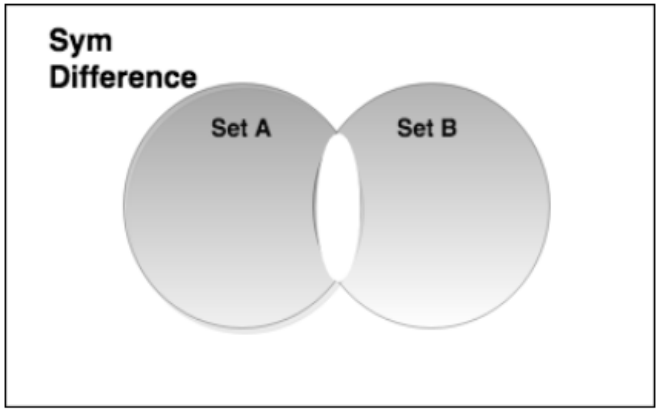
```
# Let's use the following Python set
py_set_num = {22, 33, 55, 77, 99}
print("py_set_num:", py_set_num)

# pop an element from the set
py_set_num.pop()
print("py_set_num.pop():", py_set_num)

# pop one more element from the set
py_set_num.pop()
print("py_set_num.pop():", py_set_num)

# clear all elements from the set
py_set_num.clear()
print("py_set_num.clear():", py_set_num)
```

Native Operations










```
# We'll use the setA and setB for our illustration
setA = {'a', 'e', 'i', 'o', 'u', 'g', 'h'}
setB = {'a', 'e', 'z', 'b', 't', 'o', 'u'}
```

```python
# Python set example to access elements from a set
basket = set(["apple", "mango", "banana", "grapes",
"orange"])

for fruit in basket:
    print(fruit)
```

```python
# Python set example to test elements in a set
basket = set(["apple", "mango", "banana", "grapes",
"orange"])

# confirm if 'apple' is in the basket
print("Is 'apple' in the basket?", 'apple' in basket)

# confirm if 'grapes' is in the basket
print("Is 'watermelon' in the basket?", 'watermelon' in
basket)
```

Frozen Sets

- It is a unique type of set which is immutable and doesn’t allow changing its elements after assignment.
- It supports all methods and operators applicable to a set but those that don’t alter its content.

```python
# Python Sample - Standard vs. Frozen Set

# A standard set
std_set = set(["apple", "mango","orange"])

# Adding an element to normal set is fine
std_set.add("banana")

print("Standard Set:", std_set)

# A frozen set
frozen_set = frozenset(["apple", "mango","orange"])

print("Frozen Set:", frozen_set)

# Below code will raise an error as we are modifying a
frozen set
try:
    frozen_set.add("banana")
except Exception as ex:
    print("Error:", ex)
```

Tuples

```
# create an empty tuple
py_tuple = ()
print("A blank tuple:", py_tuple)

# create a tuple without using round brackets
py_tuple = 33, 55, 77
print("A tuple set without parenthesis:", py_tuple, "type:", type(py_tuple))

# create a tuple of numbers
py_tuple = (33, 55, 77)
print("A tuple of numbers:", py_tuple)

# create a tuple of mixed numbers
# such as integer, float, imaginary
py_tuple = (33, 3.3, 3+3j)
print("A tuple of mixed numbers:", py_tuple)

# create a tuple of mixed data types
# such as numbers, strings, lists
py_tuple = (33, "33", [3, 3])
print("A tuple of mixed data types:", py_tuple)

# create a tuple of tuples
# i.e. a nested tuple
py_tuple = (('x', 'y', 'z'), ('X', 'Y', 'Z'))
print("A tuple of tuples:", py_tuple)
```

Using tuple

```
# creating a tuple from a set
>>> py_tuple = tuple({33, 55 , 77})
>>> type(py_tuple)
<class 'tuple'>
>>> py_tuple
(33, 77, 55)
# creating a tuple from a list
>>> py_tuple = tuple([33, 55 , 77])
>>> type(py_tuple)
<class 'tuple'>
>>> py_tuple
(33, 55, 77)
```

```
# A single element surrounded by parenthesis will create a string instead of a tuple
>>> py_tuple = ('single')
>>> type(py_tuple)
<class 'str'>
# You need to place a comma after the first element to create a tuple of size "one"
>>> py_tuple = ('single',)
>>> type(py_tuple)
<class 'tuple'>
# You can use a list of one element and convert it to a tuple
>>> py_tuple = tuple(['single'])
>>> type(py_tuple)
<class 'tuple'>
# You can use a set of one element and convert it to a tuple
>>> py_tuple = tuple({'single'})
>>> type(py_tuple)
<class 'tuple'>
```

Dictionary

```
# Define a blank dictionary with no elements
blank_dict = {}

# Define a dictionary with numeric keys
num_dict = {1: 'soccer', 2: 'baseball'}

# Define a dictionary with keys having different types
misc_dict = {'class': 'senior secondary', 1: [1, 2, 3,4,5]}

# Create a dictionary with dict() method
get_dict_from_func = dict({1:'veg', 2:'non-veg'})

# Create a dictionary from a sequence of tuples
make_dict_from_seq = dict([(1,'jake'), (2,'john')])
```

Methods

```
dict = {'Student Name': 'Berry', 'Roll No.': 12, 'Subject': 'English'}
print(dict.get('Student Name'))
print(dict.get('Roll No.'))
print(dict.get('Subject'))
```

```
# Create a Python dictionary
sixMonths = {1:31, 2:28, 3:31, 4:30, 5:31, 6:30}

# Delete a specific element
print(sixMonths.pop(6))
print(sixMonths)

# Delete an random element
print(sixMonths.popitem())
print(sixMonths)

# Remove a specific element
del sixMonths[5]
print(sixMonths)

# Delete all elements from the dictionary
sixMonths.clear()
print(sixMonths)

# Finally, eliminate the dictionary object
del sixMonths
print(sixMonths)
```

```
>>> {w : i for i, w in enumerate(weekdays)}
{'fri': 5, 'tue': 2, 'wed': 3, 'sat': 6, 'thu': 4, 'mon': 1, 'sun': 0}
```

```
weekdays = {'fri': 5, 'tue': 2, 'wed': 3, 'sat': 6, 'thu': 4, 'mon': 1, 'sun': 0}
# Output: True
print('fri' in weekdays)
# Output: False
print('thu' not in weekdays)
# Output: True
print('mon' in weekdays)
```

Scope

- **Local**
- **Global**
- **Built-in**

```
a_var = 5
b_var = 7

def outer_foo():
    global a_var
    a_var = 3
    b_var = 9
    def inner_foo():
        global a_var
        a_var = 4
        b_var = 8
        print('a_var inside inner_foo :', a_var)
        print('b_var inside inner_foo :', b_var)
    inner_foo()
    print('a_var inside outer_foo :', a_var)
    print('b_var inside outer_foo :', b_var)

outer_foo()
print('a_var outside all functions :', a_var)
print('b_var outside all functions :', b_var)
```

Import

```
from <module name> import *
```

```
print("namespace_1: ", dir())

from math import *
print("namespace_2: ", dir())
print(sqrt(144.2))

from cmath import *
print("namespace_3: ", dir())
print(sqrt(144.2))
```

```
from <module name> import <foo_1>, <foo_2>
```

```
print("namespace_1: ", dir())

from math import sqrt, pow
print("namespace_2: ", dir())
print(sqrt(144.2))
```

```
import <module name>
```

```
print("namespace_1: ", dir())

import math
print("namespace_2: ", dir())
print(math.sqrt(144.2))
```

Python Strings

```
# Python string examples - all assignments are identical.
String_var = 'Python'
String_var = "Python"
String_var = """Python"""

# with Triple quotes Strings can extend to multiple lines
String_var = """ This document will help you to
explore all the concepts
of Python Strings!!! """

# Replace "document" with "tutorial" and store in another variable
substr_var = String_var.replace("document", "tutorial")
print (substr_var)
```

Accessing characters

```
sample_str = 'Python String'

print (sample_str[0])          # return 1st character
# output: P

print (sample_str[-1])         # return last character
# output: g

print (sample_str[-2])         # return last second character
# output: n
```

```
sample_str = 'Python String'
print (sample_str[3:5])      #return a range of character
# ho
print (sample_str[7:])        # return all characters from index 7
# String
print (sample_str[:6])        # return all characters before index 6
# Python
print (sample_str[7:-4])
# St
```

Python String Operators

Operator	Operation	Description	Example Code
+	Concatenation	Combining two Strings into one.	var1 = 'Python' var2 = 'String' print (var1+var2) # *PythonString*
*	Repetition	Creates new String by repeating the String given number of times.	var1 = 'Python' print (var1*3) # *PythonPythonPython*
[ ]	Slicing	Prints the character at given index.	var1 = 'Python' print (var1[2]) # *t*
[:]	Range Slicing	Prints the characters present at the given range .	var1 = 'Python' print (var1[2:5]) # *tho*
in	Membership	Returns 'True' value if character is present in the given String.	var1 = 'Python' print ('n' in var1) # *True*
not in	Membership	Returns 'True' value if character is not present in given String.	var1 = 'Python' print ('N' not in var1) # *True*

for	Iterating	Using for we can iterate through all the characters of the String.	`var1 = 'Python'`
			`for var in var1:`
			`  print (var)`
			`# P`
			`# y`
			`# t`
			`# h`
			`# o`
			`# n`

r/R	Raw String	Used to ignore the actual meaning of Escape characters inside a string. For this we add 'r' or 'R' in front of the String.	print (r'\n') # *\n* print (R'\n') # *\n*

Escape Characters

Escape Character	Used To Print
\\	Backslash (\)
\"	Double-quote (")
\a	ASCII bell (BEL)
\b	ASCII backspace (BS)
\cx or \Cx	Control-x
\f	ASCII Form feed (FF)
\n	ASCII linefeed (LF)
\N{name}	Character named name in the Unicode database (Unicode only)
\r	Carriage Return (CR)
\t	Horizontal Tab (TAB)
\uxxxx	Character with 16-bit hex value xxxx (Unicode only)
\Uxxxxxxxx	Character with 32-bit hex value xxxxxxxx (Unicode only)
\v	ASCII vertical tab (VT)
\ooo	Character with octal value ooo
\xnn	Character with hex value nn where n can be anything from the range 0-9, a-f or A-F.

Python Format Characters

Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPER-case letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPER-case 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

String Conversion Functions

Function Name	Description	Example Code
capitalize()	Returns the String with first character capitalized and rest of the characters in lower case.	var = 'PYTHON' print (var.capitalize()) # *Python*
lower()	Converts all the characters of the String to lowercase.	var = 'TechBeamers' print (var.lower()) # *techbeamers*
upper()	Converts all the characters of the String to uppercase.	var = 'TechBeamers' print (var.upper()) # *TECHBEAMERS*
swapcase()	Swaps the case of every character in the String means that lowercase characters are changed to uppercase and vice-versa.	var = 'TechBeamers' print (var.swapcase()) # *tECHbEAMERS*
title()	Returns the 'titlecased' version of String which means that all words start with uppercase and rest of the characters in the words are in lowercase.	var = 'welcome to Python programming' print (var.title()) # *Welcome To Python Programming*

count(str[,beg [,end]])	Returns the number of times substring 'str' occurs in range [beg, end] if beg and end index are given. If it is not given then substring is searched in whole String. Search is case-sensitive.	var='TechBeamers' str='e' print (var.count(str)) # *3* var1='Eagle Eyes' print (var1.count('e')) # *2* var2='Eagle Eyes' print (var2.count('E',0,5)) # *1*

Strings Comparison Functions

islower()	Returns 'True' if all the characters in the String are in lowercase. If any one character is in uppercase it will return 'False'.	var='Python' print (var.islower()) # *False* var='python' print (var.islower()) # *True*
isupper()	Returns 'True' if all the characters in the String are in uppercase. If any one character is in lowercase it will return 'False'.	var='Python' print (var.isupper()) # *False* var='PYTHON' print (var.isupper()) # *True*
isdecimal()	Returns 'True' if all the characters in String are decimal. If anyone character in the String is of other data-type, it will return 'False'. Decimal characters are those from Unicode category 'Nd'. Complete list of 'Nd' is present at following link: http://www.fileformat.info/info/unicode/category/Nd/list.htm	num=u'2016' print (num.isdecimal()) # *True*

isdigit()	Returns ‘True’ for any character for which isdecimal() would return ‘True and some characters in ‘No’ category. If there are any characters other than these, it will return ‘False’. Precisely, digits are the characters for which Unicode property includes: Numeric_Type=Digit or Numeric_Type=Decimal. For example, superscripts are digits but fractions not. Complete list of ‘No’ is present at following link: http://www.fileformat.info/info/unicode/category/No/list.htm	print (‘2’.isdigit()) # *True* print (‘²’.isdigit()) # *True*

isnumeric()	Returns 'True' if all the characters of the Unicode String lie in any one of the category 'Nd','No' and 'Nl'. If there are any characters other than these, it will return 'False'. Precisely, Numeric characters are those for which Unicode property includes Numeric_Type=Digit, Numeric_Type=Decimal or Numeric_Type=Numeric. Complete list of 'Nl' is present at following link: http://www.fileformat.info/info/unicode/category/Nl/list.htm	num=u'2016' print (num.isnumeric()) # *True* num=u'year2016' print (num.isnumeric()) # *False*
isalpha()	Returns 'True' if String contains at least one character (non-empty String) and all the characters are alphabetic, 'False' otherwise.	print ('python'.isalpha()) # *True* print ('python3'.isalpha()) # *False*
isalnum()	Returns 'True' if String contains at least one character (non-empty String) and all the characters are either alphabetic or decimal digits, 'False' otherwise.	print ('python'.isalnum()) # *True* print ('python3'.isalnum()) # *True*

String Padding Functions

rjust(width[,fillchar])	Returns a padded version of String with the original String right-justified to a total of width columns. By default, Padding is done by using space. Otherwise 'fillchar' specifies the filler character.	var='Python' print (var.rjust(10)) # *Python* print (var.rjust(10,'-')) # *Python—-*
ljust(width[,fillchar])	Returns a padded version of String with the original String left-justified to a total of width columns. By default, Padding is done by using space. Otherwise 'fillchar' specifies the filler character.	var='Python' print (var.ljust(10)) # *Python* print (var.ljust(10,'-')) # *Python—-*
center(width[,fillchar])	Returns a padded version of String with the original String moved to center to a total of width columns. By default, Padding is done by using space. Otherwise 'fillchar' specifies the filler character.	var='Python' print (var.center(20)) # *Python* print (var.center(20,'*')) # *******Python*******
zfill(width)	Returns a padded version of String with the original String padded on the left with zeros so that total length of String becomes equal to width. If there is a leading sign (+/-) present in the String, then with this function padding is done after the sign, not before it.	var='Python' print (var.zfill(10)) # *0000Python* var='+Python' print (var.zfill(10)) # *+000Python*

Find Functions

find(str [,i [,j]])	Searches for 'str' in complete String (if i and j not defined) or in a sub-string of String (if i and j are defined).This function returns the index if 'str' is found else returns '-1'. where, i=search starts from this index j=search ends at this index.	var="Tech Beamers" str="Beam" print (var.find(str)) # *5* var="Tech Beamers" str="Beam" print (var.find(str,4)) # *5* var="Tech Beamers" str="Beam" print (var.find(str,7)) # *-1*
index(str[,i [,j]])	This is same as 'find' method. The only difference is that it raises 'ValueError' exception if 'str' is not found.	var='Tech Beamers' str='Beam' print (var.index(str)) # *5* var='Tech Beamers' str='Beam' print (var.index(str,4)) # *5* var='Tech Beamers' str='Beam' print (var.index(str,7)) # *ValueError: substring not found*

rfind(str[,i [,j]])	This is same as find() just that this function returns the last index where 'str' is found. If 'str' is not found it returns '-1'.	var='This is a good example' str='is' print (var.rfind(str,0,10)) # *5* print (var.rfind(str,10)) # *-1*
count(str[,i [,j]])	Returns the number of occurrences of substring 'str' in the String. Searches for 'str' in complete String (if i and j not defined) or in a sub-string of String (if i and j are defined). where, i=search starts from this index j=search ends at this index.	var='This is a good example' str='is' print (var.count(str)) # *2* print (var.count(str,4,10)) # *1*

Replace Function

replace(old,new[,count])	Replaces all the occurrences of substring 'old' with 'new' in the String. If 'count' is defined then only 'count' number of occurrences of 'old' will be replaced with 'new'. where, old =substring to be replaced new =substring that will replace the old count =number of occurrences of old that will be replaced with new.	var='This is a good example' str='was' print (var.replace('is',str)) # *Thwas was a good exampleprint (var.replace('is',str,1))* # *Thwas is a good example*
split([sep[,maxsplit]])	Returns a list of substring obtained after splitting the String with 'sep' as delimiter. where, sep= delimiter, default is space maxsplit= number of splits to be done	var = "This is a good example" print (var.split()) # *['This', 'is', 'a', 'good', 'example']print (var.split(' ', 3))* # *['This', 'is', 'a', 'good example']*

splitlines(num)	Splits the String at line breaks and returns the list after removing the line breaks. where, num = if this is positive value. It indicates that line breaks to be included in the returned list.	var='Print new line\nNextline\n\nMove again to new line' print (var.splitlines()) # *['Print new line', 'Nextline', '', 'Move again to new line']print (var.splitlines(1))* # *['Print new line\n', 'Nextline\n', '\n', 'Move again to new line']*
join(seq)	Returns a String obtained after concatenating the sequence 'seq' with a delimiter string. where, seq= sequence of elements to be joined	seq=('ab','bc','cd') str='=' print (str.join(seq)) # *ab=bc=cd*

Misc String Handling Functions

lstrip([chars])	Returns a String after removing the characters from the beginning of the String. where, Chars=this is the character to be trimmed from the String. Default is whitespace character.	var=' This is a good example ' print (var.lstrip()) # *This is a good example* var='*****This is a good example*****' print (var.lstrip('*')) # *This is a good example*********** var=' This is a good example ' print (var.rstrip()) # *This is a good example* var='*****This is a good example*****' print (var.lstrip('*')) # *****This is a good example*
rstrip()	Returns a String after removing the characters from the End of the String. where, Chars=this is the character to be trimmed from the String. Default is whitespace character.	

rindex(str[,i [,j]])	Searches for 'str' in complete String (if i and j not defined) or in a sub-string of String (if i and j are defined).This function returns the last index where 'str' is found. If 'str' is not found it raises 'ValueError' exception.where, i=search starts from this index j=search ends at this index.	var='This is a good example' str='is' print (var.rindex(str,0,10)) # *5* print (var.rindex(str,10)) # *ValueError: substring not found*
len(string)	Returns the length of given String	var='This is a good example' print (len(var)) # 22

Conditional Statement

```
if BOOLEAN EXPRESSION:
    STATEMENTS
```

```
if BOOLEAN EXPRESSION:
    STATEMENTS_1
else:
    STATEMENTS_2
```

```
if x < y:
    STATEMENTS_A
elif x > y:
    STATEMENTS_B
else:
    STATEMENTS_C
```

Loop Statements

```
for LOOP_VARIABLE in SEQUENCE:
    STATEMENTS
```

```
while BOOLEAN_EXPRESSION:
    STATEMENTS
```

```
for i in [12, 16, 17, 24, 29]:
    if i % 2 == 1:  # if the number is odd
        break        # immediately exit the loop
    print(i)
print("done")
```

```
for i in [12, 16, 17, 24, 29, 30]:
    if i % 2 == 1:      # if the number is odd
        continue        # don't process it
    print(i)
print("done")
```

Function in Python

```
def NAME( LIST OF PARAMETERS ):
    STATEMENTS
```

```
>>> def f():
...     print("Hello from function f!")
...
>>> type(f)
<type 'function'>
>>> f()
Hello, from function f!
>>>
```

```
>>> do_stuff = [f, g, h]
>>> for func in do_stuff:
...     func(10)
```

How to import function from file?

```
def double_stuff_v1(a_list):
    index = 0
    for value in a_list:
        a_list[index] = 2 * value
        index += 1
```

pure_v_modify.py

```
>>> from pure_v_modify import double_stuff_v1
>>> things = [2, 5, 'Spam', 9.5]
>>> double_stuff_v1(things)
>>> things
[4, 10, 'SpamSpam', 19.0]
```

Polymorphism

```
>>> def double(thing):
...     return 2 * thing
...
>>> double(5)
10
>>> double('Spam')
'SpamSpam'
>>> double([1, 2])
[1, 2, 1, 2]
>>> double(3.5)
7.0
>>> double(('a', 'b'))
('a', 'b', 'a', 'b')
```

Recursion

```
def recursive_sum(nested_num_list):
    the_sum = 0
    for element in nested_num_list:
        if type(element) == list:
            the_sum = the_sum + recursive_sum(element)
        else:
            the_sum = the_sum + element
    return the_sum
```

N number of Arguments

```
def manyArgs(*arg):
  print "I was called with", len(arg), "arguments:", arg

>>> manyArgs(1)
I was called with 1 arguments: (1,)
>>> manyArgs(1, 2,3)
I was called with 3 arguments: (1, 2, 3)
```

Optional Parameters

```
def info(object, spacing=10, collapse=1):
```

```
info(odbchelper)                          ❶
info(odbchelper, 12)                      ❷
info(odbchelper, collapse=0)              ❸
info(spacing=15, object=odbchelper)       ❹
```

Exception Handling

```
try:
        You do your operations here;
        ......................
except ExceptionI:
        If there is ExceptionI, then execute this block.
except ExceptionII:
        If there is ExceptionII, then execute this block.
        ......................
else:
        If there is no exception then execute this block.
```

```
try:
    You do your operations here;
    ......................
except:
    If there is any exception, then execute this block.
    ......................
else:
    If there is no exception then execute this block.
```

List Of Python Built-In Exceptions.

AirthmeticError	For errors in numeric calculation.
AssertionError	If the assert statement fails.
AttributeError	When an attribute assignment or the reference fails.
EOFError	If there is no input or the file pointer is at EOF.
Exception	It is the base class for all exceptions.
EnvironmentError	For errors that occur outside the Python environment.
FloatingPointError	When floating point operation fails.
GeneratorExit	If a generator's <close()> method gets called.
ImportError	When the imported module is not available.
IOError	If an input/output operation fails.
IndexError	When the index of a sequence is out of range.
KeyError	If the specified key is not available in the dictionary.

KeyboardInterrupt	When the user hits an interrupt key (Ctrl+c or delete).
MemoryError	If an operation runs out of memory.
NameError	When a variable is not available in local or global scope.
NotImplementedError	If an abstract method isn't available.
OSError	When a system operation fails.
OverflowError	If the result of an arithmetic operation exceeds the range.
ReferenceError	When a weak reference proxy accesses a garbage collected reference.
RuntimeError	If the generated error doesn't fall under any category.
StandardError	It is a base class for all built-in exceptions except <StopIteration> and <SystemExit>.

StopIteration	The <next()> function has no further item to be returned.
SyntaxError	For errors in Python syntax.
IndentationError	When indentation is not proper.
TabError	For inconsistent tabs and spaces.
SystemError	When interpreter detects an internal error.
SystemExit	The <sys.exit()> function raises it.
TypeError	When a function is using an object of the incorrect type.
UnboundLocalError	If the code using an unassigned reference gets executed.
UnicodeError	For a Unicode encoding or decoding error.
ValueError	When a function receives invalid values.
ZeroDivisionError	If the second operand of division or modulo operation is zero.

finally

```
>>> def divide(x, y):
...     try:
...         result = x / y
...     except ZeroDivisionError:
...         print "division by zero!"
...     else:
...         print "result is", result
...     finally:
...         print "executing finally clause"
...
>>> divide(2, 1)
result is 2
executing finally clause
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

Raise An Exception

```
raise [Exception [, args [, traceback]]]
```

```
>>> raise MemoryError
Traceback (most recent call last):
...
MemoryError

>>> raise MemoryError("This is an argument")
Traceback (most recent call last):
...
MemoryError: This is an argument


>>> try:
        a = int(input("Enter a positive integer value: "))
      if a <= 0:
                raise ValueError("This is not a positive number!!")
     except ValueError as ve:
        print(ve)
```

Any Query ?

Ayyappan Murugesan

Phone: 7010774329

WhatsApp: 8870483557