Notes : **Python Programming Language**

Course Name : **Data Analytics with Python**

Content Provider : **Data Science Lovers**

CORE PYTHON COMMANDS

Python is a high-level programming language that uses instructions to teach the computer how to perform a task.
Python is an easy to learn, powerful programming language.
A language which is closer to the human language (like English) is known as a high-level language.
Python provides an easy approach to object-oriented programming.
Object-oriented is approach used to write programs.
Python is a free and open source language i.e., we can read, modify and distribute the source code of Python scripts.
It was developed by Guido van Rossum and was released in 1991.
Python finds its application in various domains. Python is used to create web applications, used in game development, to create desktop applications, is used in Machine Learning and Data Science.

How Python Works ?

We write instructions in Python language.
Python is an interpreted language, so there is no need to compiling it.
Python programs runs (executed) directly through source code.
The source code is converted into Intermediate Bytecode and then Bytecode is converted into the native language of computer (i.e., machine language) internally by Python Interpreter.
The code is executed and the output is presented.
Python Source Code > Intermediate Bytecode > Machine Language > Code Executed

A Program is a set of instructions that tells the computer to perform a specific task. A programming language is the language used to create programs.

Eg. When we click on Play button on media player, then there is a program working behind the scene which tells the computer to turn on the music.

A built-in function is a function which is predefined and can be used directly. Eg. print()

Comments are the pieces of code which are ignored by the python interpreter. Comments are used to make source code easier to understand by other people. Python supports single line comments mean they can cover only one line.

VARIABLES

a = 2 , b = 1.2 , c = ‘Ram’, d = lambda (‘any function’)

Variables are used to store values.
The stored values in the variables can be used later in the programs.
We can retrieve them by referring to the variable names.

DATATYPES IN PYTHON

Integer (int), Float , String (str) , List , Tuple , Set , Dictionary

String – String is a series of characters, surrounded by single or double quotes. Eg. “Hello”, ‘Hello999’, ‘999’.

1. LIST

[int /float / str] → A = [1 , 2 , 3.4 , 3.4, ‘a’ , ‘bcd’]

→ Collection of data-types, Mutable : Values can be changed , Ordered : Values order will be as it is , Changeable , Allows duplicate values.

2. TUPLE

(int / float / str) → B = (1 , 2 , 3.4 , 3.4 , ‘a’ , ‘bcd’)

→Immutable : Values can’t be changed , Ordered : Values order will be as it is , Unchangeable, Heterogeneous Data, Allows duplicate values.

3. SET

{ int / float / str } → C = { 1 , 2 , 3.4 , 5.6 , ‘a’ , ‘bcd’ }

→Values can’t be changed but new values can be added , Unordered : Values order may change , Arrange the items in ascending order, Doesn’t allow duplicate values, Un-indexed.

4. DICTIONARY

{ Key : Value } → D = { K1 : 1 , K2 : 2 , K3 : 3.4 , K4 : 5.6 , K5 : ‘ab’ , K6 : ‘bcd’ }

→ Mutable , Unordered , Doesn’t allows duplicate keys , Indexed, Keys must be unique & immutable.

CONCATENATION – Combining Strings

first = ‘Data’
last = “Science”
new = first + ‘ ’ + last + ‘ is the combined string’

“\n” – For next new line

print("My Name is", **"\n"** , "My city is ", **"\n"** ,"My country is")
print(‘Delhi’) , print(‘’) , print(‘Noida’) # To create a gap of one line between two strings.

LIST FUNCTIONS

< Press ‘Tab’ button from the keyboard after typing the list name (A here) to show the available functions >

A.append(55) - To add a new value at the end of the list.

A.clear( ) – To clear/delete/blank a list.

B = A.copy( ) – To create a copy of the list.

A.count(5) – To count how many times a value occurs.

A.extend(c) – To add a new list in the existing list.

A.index(7) – To show the index of a value. # A.index(value, start_index, stop_index)

A.insert(3,66) – To insert a new value at a given position.

A.pop(3) – To delete a value with the help of index. # A.pop( )

A.remove(55) – To delete a value from the list.

A.reverse( ) – To reverse the list.

A.sort( ) – To sort the list. # A.sort(reverse=True)

del A[1 : 4] – To delete some items from the list.

type(A) – To see the type.

List Concatenation - A = [1,2,3,4] , B = [5,6,7,8] ; C = A+B = [1,2,3,4,5,6,7,8]

TUPLE FUNCTIONS

T.count(5) – To count how many times a value occurs.

T.index(7) – To show the index of a value.

SET FUNCTIONS

S.add(5) – To add a new value 5 in the set.

S.clear() – To clear all the elements of the set.

S.copy() – To copy a set.

S1.difference(S2) – S1-S2 - It shows the elements of set S1 only.

S1.difference_update(S2) – It removes all common elements from the set1.

S.discard(x) – It will remove an element(x) from the set. If x is not in set, it will not show error.

S.remove(x) – It will remove an element(x) from the set. If x is not in set, it will show an error.

S.pop() – It deletes the first/random element of the set.

S1.Union(S2) – **Set1 | Set2** – It shows all elements of set1 and set 2.

S1.Intersection(S2) – **Set1 & Set2** – It shows common elements of set1 and set2.

S1.Intersection_update(S2) – Now set S1 will contain only common elements.

S1.isdisjoint(S2) – It returns True, if S1 & S2 don't have any common values, otherwise False.

S1.issubset(S2) – It returns True, if all elements of S1 are in set S2.

S2.issuperset(S1) – It returns True, if all elements of S1 are in set S2, otherwise False.

len(S) – It shows the no. of unique elements in the set.

S1.symmetric_difference(S2) – S1^S2 – To show the non-common elements from S1 and S2.

S1.symmetric_difference_update(S2) - Now set S1 will contain only non-common elements.

S1.update([4,5,6]) – To add multiple items, in list/tuple/set form.

DICTIONAY FUNCTIONS

D.clear( ) – To delete the dictionary.

E = D.copy( ) – To copy a dictionary.

D.get('K1') – To get the value against a key in the dictionary. If the key is not in dictionary, it will show None, without showing any error.

D.items( ) – To show all the items of a dictionary.

D.keys( ) – To show all the keys of a dictionary.

D.values( ) – To show all the values of a dictionary.

D.pop('K1') – To delete the key alongwith its index.

D.popitem( ) – To delete the last key with value.

D.setdefault('K3') , D.setdefault('K4', value), D['K4'] = value - To add a key at the end of the dictionary.

D.update('E') – To add a new dictionary in the existing dictionary.

D.fromkeys(A) – To create a dictionary, using list items as keys. And adding a value to all keys is optional.

"Key" in D – To check the presence of any element(key) in the dictionary.

DATATYPE CASTING

Converting a datatype into another.

int (1) =>1 - Converting int into int

int (3.2) => 3 – Converting float into int

int ('5') => 5 – Converting a numerical string into int

int ('a') => error – Can't convert an alphabetical string into int

float (3.2) => 3.2 – Converting float into float

float (6) => 6.0 – Converting int into float

float ("10") => 10.0 – Converting a numerical string into float

float (‘b’) => error – Can’t convert an alphabetical string into float

Str (‘a’) => ‘a’ – Converting a string into string

str (1) => ‘1’ – Converting an int into string

str (3.2) => ‘3.2’ – Converting a float into string

RANGE

It creates a sequential list of numbers.

range(start value, stop value, step value) , range(0,50,1) , range(1, 50) , range(50)

FUNCTION

A function is a block of code, which is defined to perform some task. We have call a function to run it whenever required.

Parameter : Given at the time of defining function . Ex : def func(a,b)
Arguments : Given at the time of calling the function . Ex : func(2,3)
def fun_name (args / parameters) : multiple line statement ,
def fun_name (var1, var2) : multiple line statement
def new (2 , 3) : c = a + b , return c

If the number of arguments to be passed is not fixed…then we use the Arbitrary Arguments (with *args)

Ex : def func(*values) : for i in values print(i) # It can take any number of arguments.

Keyword Arguments : We can also send the args with key=value syntax.

Ex : def new(b,a,c): print("The winner is " , a)

new(a= ‘Ram’, b= ‘Sham’, c= ‘Shiva’) ….. O/p will be : The winner is Ram

LAMBDA FUNCTION

It is a single line function.

fun_name = lambda parameters : single line statement

Ex : sum = lambda a , b : a + b

INPUT FUNCTION

It takes an input and can save it to a variable.

Ex 1 : a = input ('Enter your name') ,

Ex 2 : print ('Enter your name')
x = input ( )

INDEXING

list.index(item) , list [index value] , list [start : stop : step]

A.index(25) , A[1] , A [1 : 20 : 2] , A [: 4] , A[2 :] , A [:]

Negative Indexing – A[-1] , A [8 : 0 : -1] , A [: : -1]

String Indexing – A.index('r') , A[: 16]

Nested List - List in a list

Ex : A = [[1,2,3] , 4 , 5 , 6 , [7,8,9]]

FOR LOOP

for val in sequence : body of for loop,

Ex 1 : for x in [1,2,3,4,5] : print (x) ,

Ex 2 : for i in 'banana' : print (i)

BREAK STATEMENT (For Loop) –

To stop the loop at a given condition

1) for val in sequence : body of for loop if val == 'seq_value' , break

Ex : for x in [1,2,3,4,5,6,7] :
print (x)
if x == 5
break

2) for val in sequence : if val == 'seq_value' break , print(val)

Ex : for x in [1,2,3,4,5,6,7] :
if x == 5

```
        break
        print(x)
```

CONTINUE STATEMENT (For Loop) –

To skip over an iteration

```
1) for x in [1,2,3,4,5] :
        if x == 4
                continue
        print(x)

2) for x in [1,2,3,4,5] :
        print (x)
        if x == 4
                continue
```

BREAK & CONTINUE STATEMENT (For Loop) –

```
Ex : for x in [1,2,3,4,5,6,7]:
        if x == 5 :
                continue
        if x == 6:
                break
        print(x)
```


RANGE FUNCTION –

```
for x in range (6):
        print (x)
```

ELSE IN FOR LOOP –

```
1) for x in range(6):
        print (x)
else :
        print ('loop is finished')

2) for x in range(0,6):
        print (x)
        if x == 4 :
        break
else :
        print('loop is finished')
```

PASS STATEMENT –

To pass over to the next commands

```
1) for x in [1,2,3,4,5,6,7]:
        Pass

2) for x in [1,2,3,4,5,6,7]:
        if x == 3:
                pass
        print (x)
```

WHILE LOOP

A while loop repeats a block of code as long as a certain condition is true.

```
1) i = 0
while i < 6 :
        print (i)
        i = i +1

2) i = 0
while i < 6 :
        i = i +1
        print (i)
```


BREAK STATEMENT (While Loop) –

```
1) i = 0
while i < 6 :
        print (i)
        if i == 4 :
                break
        i = i +1

2) i = 0
while i < 6 :
        if i == 4 :
                break
        print (i)
        i = i + 1
```

CONTINUE STATEMENT (While Loop) –

```
1) i = 0
while i < 6 :
        i = i +1
        if i == 3 :
                continue
        print (i)

2) i = 0
while i < 6 :
        if i == 3 :
                continue
        print (i)
        i = i +1

3)i = 0
while i < 6 :
        if i == 3:
                continue
        i = i + 1
print (i)
```

ELSE IN WHILE LOOP –

```
1) i = 0
while i < 6 :
        print (i)
        i = i+1
else:
        print ('condition ends')
```

BREAK & CONTINUE STATEMENT (While Loop) –

```
i = 0
while i < 10 :
        i = i + 1
        if i = = 3:
                continue
        if i = = 9 :
                break
        print (i)
```

SPLIT FUNCTION

It splits a string into a list.

Syntax : string.split (separator , maxsplit)

MAP FUNCTION

It takes all items of a list and apply a function to it.

Syntax : map(function, iterables) or map(condition, values)

Ex : list (map (lambda x : x+1 , [1,2,3,4,5]))

FILTER FUNCTION

It takes all items of a list and apply a function to it & returns a new filtered list.

Syntax : filter(function, sequence)

Ex : list (filter (lambda x : x%2 != 0 , [1,2,3,4,5,6]))

ENUMERATE FUNCTION

It is used to display output with index. We can enumerate as list, tuple, set, dictionary.

Syntax : enumerate(list)

Ex : list (enumerate (‘apple’ , ‘mango’ , ‘orange’))

OPERATOR.ITEMGETTER FUNCTION

It can be used to assign variables from the list.

Syntax : from operator import itemgetter

itemgetter(index1, index2, index3) (list_name)

Ex : A = [1,2,3,4,5,6,7,8,9]

from operator import itemgetter
itemgetter(1,3,5)(A)

ITERTOOLS.COMPRESS FUNCTION

It can be used to assign variables from the list.

Syntax : from itertools import compress

compress(list_name , (0,1,1,0,1,1)

Ex : A = [1,2,3,4,5,6,7,8,9]
from itertools import compress
compress(A, (0,1,1,0,1,1,0,0,1))

ZIP FUNCTION

It is used to zip different iterators(lists) in one.

Syntax : z = zip(list1, list2, list3)

z = list(z) , print(z)

Example : A = [1,2,3] , B = [‘Ram’ , ‘Sham’ , ‘Shiva’] , C = [‘Delhi’, ‘Noida’, ‘Agra’]

z = zip(A, B, C) , z = list(z) , print(z)

UNZIP FUNCTION

Syntax : list1, list2, list3 = zip(*z)

Ex : A, B, C = zip(*z)

To know about your current working directory : **import os …. os.getcwd()**

To change the current working directory : **os.chdir(‘D:\New Folder’)**

---

by – ROHIT GREWAL