Python

Comments:

Either the explanation of the code or for more clear understanding we use comments. Whatever information represented under these comments will not be executed, they are just excluded.

For single line comments we use # symbol

For multiple line comments we use \*\*\* \*\*\* symbol

Indentation:

Python uses [whitespace](https://en.wikipedia.org/wiki/Whitespace_character) indentation, rather than [curly brackets](https://en.wikipedia.org/wiki/Curly_bracket_programming_language) or keywords, to delimit [blocks](https://en.wikipedia.org/wiki/Block_(programming)). An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block.

These are nothing but the 4spaces before starting a code. These are represented with “:” symbol. This indentation plays key role in python. The IDE will automatically hit indentation after giving enter.

Data Types:

There are 5 types.

1. None Type – an object that does not contain any value.
2. Numeric data types – int, float, complex.
3. Sequence data types – strings, bytes, byte array, lists, tuple, range.
4. Sets data types – it will not allow any duplicates.
5. Mapping data types – used to map and reduce.

Identifiers:

The names we give to the variables and the functions within our python programming are called identifiers.

Python is case sensitive

Strings:

Sequence of characters which are enclosed within single or double quotes.

Ex: s = ”Hello Supriya!”

Print(s)

To print each character, we can call it using index number

Print(s[1])

To print the string multiple times

Print(s\*2)

To print the string length

Print(len(s))

String Slicing:

Obtaining sub-string of a string. We can create a sub-string by slicing it using its index numbers. The characters before the end index value will be printed.

Ex: s = “You are awesome.”

Print(s[0:5]) # o/p: You a

Print(s[0:]) # o/p: You are awesome

Print(s[:8]) # o/p: You are 🡪 space is also counted.

Print(s[-3:-1]) # o/p: om 🡪 negative number indicates that we need to print the string from back direction.

Print(s[0:9:2]) # o/p: om 🡪 By default the step value is 1 it means alternate character will be printed. If we give 2, alternate numbers with 2 characters gap will be printed.

Print(s[::-1]) # o/p: om 🡪 The string will be printed in back ward direction.

Strip the spaces:

If there are spaces before and after the string we can remove it using this strip() function.

Ex: s = “ You are awesome ”

Print(s.strip()) # o/p: You are awesome

Print(s.lstrip()) # o/p: The leading spaces on the left hand side are removed and on right side remains same.

Print(s.rstrip()) # o/p: The leading spaces on the right hand side are removed and on left side remains same.

Print(s.find(“awe”, 0, len(s))) # o/p: The index number where the character we are searching will be the displayed and we can also mention from where we must start the search and end the search.

Lists:

Used to store multiple items in a single variable. It is used to store different variables.

lst = [10, 20, ‘Hello’, -10, 30.5]

Print(lst)

Print(lst[3]) # indexing o/p: -10

Print(lst[3:5]) # slicing o/p: -10, 30.5

Print(lst\*2) # repetition o/p: 10, 20, ‘Hello’, -10, 30.5 10, 20, ‘Hello’, -10, 30.5

Print(len(lst)) # length o/p: 5

lst.append(40) # append is used to add elements to the list.

Print(lst) # o/p: 10, 20, ‘Hello’, -10, 30.5, 40

lst.remove(10) # remove is used to remove elements to the list.

Print(lst) # o/p: 20, ‘Hello’, -10, 30.5, 40

Del(lst[10]) # remove is used to remove elements to the list.

Print(lst) # o/p: 20, ‘Hello’, -10, 30.5, 40

Lst.clear() # clears elements to the list.

Print(lst) # o/p: 0

maxl(lst) # finds the maximum element to the list.

Print(lst) # o/p: 40

minl(lst) # finds the minimum element to the list.

Print(lst) # o/p: -10

lst.insert(3, 99) # used to insert/add elements to the list.

Print(lst) # o/p: 10, 20, ‘Hello’, 99, -10, 30.5, 40

lst.sort() # arranges the list in ascending order

Print(lst) # o/p: -10, 10, 20, 30.5, 40, 99, ‘Hello’

lst.sort(reverse=true) # arranges the list in descending order

Print(lst) # o/p: -10, 10, 20, 30.5, 40, 99, ‘Hello’

Tuple:

It is nothing but a list which cannot be modified. It is an immutable list. We use this only when the requirement is read only.

Tuple follows insertion order means which ever order we give it follows the same.

Tuple allows duplicates and different types of elements.

T1 = (1, 2, 3) / t1 = 1, 2, 3 / t1 = 1,

tpl = (10, 20, ‘Hello’, 10, 30.5)

Print(tpl)

Print(tpl[3]) # indexing o/p: 10

Print(tpl\*2) # repetition o/p: 10, 20, ‘Hello’, 10, 30.5 10, 20, ‘Hello’, 10, 30.5

Print(tpl.count(10)) # 2

Print(tpl.index(20)) # 1

Set:

Collection of data types that cannot be ordered, cannot perform indexing or slicing or reputation.

Range:

r = range(5)

for i in r:

Print(i) # o/p: 0, 1, 2, 3, 4

r = range(1,6)

for i in r:

Print(i) # o/p: 1, 2, 3, 4, 5

r = range(1, 15, 3)

for i in r:

Print(i) # o/p: 1, 4, 7, 10, 13

Bytes and Bytearray:

lst = [10, 20, 30, 234] # The maximum value of byte is 255.

Print(type(lst)) # <class list>

b = bytes(lst)

Print(type(b)) # <class bytes>

We cannot perform or add elements to bytes. Slicing and Repetition is not allowed.

b1 = bytearray(lst)

Print(type(b1)) # <class bytearray>

We can perform and modify elements to bytearray. Slicing and Repetition is not allowed.

Dictionary:

They are represented as key and value pairs.

dict = {1:”John”, 2:”Bob”, 3:”Bill”}

print(dict) # 1:”John”, 2:”Bob”, 3:”Bill”

print(dict.items()) dict.items(1:”John”, 2:”Bob”, 3:”Bill”)

k = dict.keys()

for i in k: print(i) # 1, 2, 3

v = dict.values() # John, Bob, Bill

for i in v: print(i)

print(dict[3]) # Bill

del dict[2]

print(dict) # 1:”John”, 3:”Bill”)

Immutability:

It is the concept of allocating memory locations to the values stores in variables.

If 2 variables are having same value, they will be allocated to same memory location.

If 2 variables are having different value, they will be allocated to different memory location.

Constants:

In Python there is any no particular keyword for declaring a constant. We will be declaring a constant by writing them in capital letters.

Escape Characters:

\n – for new line

\t – tab space will be added

To print a string within single quotes or double quotes:

Single quotes: Print(“You are ‘awesome’”) # You are ‘awesome’

Double quotes: Print(“You are \“awesome\””) # You are “awesome”

Backward slash: Print(“You are \\awesome\\”) # You are \awesome\

Operators:

Example:

A=10, B=5

Print(Addition: , a+b) # addition of a & b is 15

Print(Subtraction: , a-b) # Subtraction of a & b is 5

Print(Mul: , a\*b) # Multiplication of a & b is 50

Print(Div: , a/b) # Division of a & b is 2.0

Print(Mod: , a%b) # Modulus of a & b is 0

Print(Pow: , a\*\*b) # Exponent of a & b is 100,000

Print(Floor div: , a//b) # Floor Division of a & b is 2

Assignment operator:

= is an assignment operator and it assigns value on the right-hand side to the variable on the left hand side.

A = 10; A = x+y;

Compound athematic operators: A+=X --> A = A+X ; A-=X; A\*=X; A/=X; A//=X; A\*\*=X; A%=X

A=B=C=10

Print(A,B,C) # A=10, B=10, C=10

X,Y=10,5

X+=Y

Print(X) # o/p: X=15

Comparison operators:

X = 10, Y = 5

Print(X==Y) # False

Print(X!=Y) # True

Print(X>=Y) # True

Print(X<=Y) # False

Print(X>Y) # True

Print(X<Y) # False

Logical operators:

X = 10, Y = 5

Print(X==10 and Y==5) # True

Print(X==10 or Y==15) # True

Print(Not(X==20 or Y==25)) # True