Comments

- Comments in Python are the lines in the code that are ignored by the interpreter during the execution of the program.

There are three types of comments in Python –

- Single line Comments
- Multiline Comments
- Docstring Comments

Single-Line Comments

Python single-line comment starts with the hashtag symbol (#) with no white spaces and lasts till the end of the line.

```
# Print “SkillSpace.ai !” to console
print("SkillSpace.ai")
```

```
SkillSpace.ai
```

Multi-Line Comments

- Using Multiple Hashtags (#)

```
# Python program to demonstrate the various ways through which we can
implement multiline comments
print("Multiline comments")
```

```
Multiline comments
```

- Using String Literals

```
"""Twitter Erupts In Joy As Virat Kohli
Scores His Maiden T20I Century And
Finally Gets His 71st International Century"""
print("Multiline comments")
```

```
Multiline comments
```

Operators in Python

Arithmetic Operators

```
# Examples of Arithmetic Operator
x = 15
y = 7

# Addition of x and y
add = x + y
```

```
# Subtraction of y from x
sub = x - y

# Multiplication of x and y
mul = x * y

# Division(float) of number
div_float = x / y

# Division(floor) of number
div_floor = x // y

# Modulo of two numbers
mod = x % y

# Power
pow = x ** y

# print results
print(add)
print(sub)
print(mul)
print(div_float)
print(div_floor)
print(mod)
print(pow)
```

```
22
8
105
2.142857142857143
2
1
170859375
```

Comparison Operators

Comparison of Relational operators compares the values. It either returns True or False according to the condition.

```
# Examples of Relational Operators
a = 15
b = 37

# a > b is False
print(a > b)

# a < b is True
print(a < b)
```

```
# a == b is False
print(a == b)

# a != b is True
print(a != b)

# a >= b is False
print(a >= b)

# a <= b is True
print(a <= b)
```

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

- Logical operators perform Logical AND, Logical OR, and Logical NOT operations. It is used to combine conditional statements.

```
# Examples of Logical Operator
a = True
b = False

# Print a and b is False
print(a and b)

# Print a or b is True
print(a or b)

# Print not a is False
print(not a)
```

```
False
True
False
```

Python Variables are containers which store values.

```
variable = "SkillSpace.ai"
print(variable)
```

```
SkillSpace.ai
```

```
# assignment of an integer
age = 45

# assignment of a floating point number
```

```
net_worth = 199268.8

# assignment of a string
name = "Jonathan"

print(age)
print(net_worth)
print(name)
```

```
45
199268.8
Jonathan
```

- Assigning a single value to multiple variables

```
a = b = c = 10

print(a)
print(b)
print(c)
```

```
10
10
10
```

How to Take Input from User in Python?

Integer input

```
x = int(input())
```

```
3
```

String input

```
x = input()
```

```
DPhi
```

Take Multiple Inputs in Python

```
p, q, r = map(int, input("Taking multiple numbers as input from the
user: ").split())
print("The Numbers are as follows : ",end = " ")
print(p,q,r)
```

```
Taking multiple numbers as input : 1 9 5
The Numbers are :  1 9 5
```

- Data structure is a data organization, management, and storage format that enables efficient access and modification.
- Text Type: str
- Numeric Types: int, float, complex
- Sequence Types: list, tuple, range
- Mapping Type: dict
- Set Types: set, frozenset
- Boolean Type: bool
- Binary Types: bytes, bytearray, memoryview
- None Type: NoneType

```
from re import A
a = "Data Science"

print(a, type(a))

a = 40

print(a,type(a))

a = 80.7

print(a,type(a))

a = 5j

print(a,type(a))

a = ["maruti", "toyota", "mahindra","tata"]

print(a,type(a))

a = ("maruti", "toyota", "mahindra", "tata")

print(a,type(a))

a = range(8)

print(a,type(a))

a = {"name" : "Lennon", "age" : 45}

print(a,type(a))

a = {"maruti", "toyota", "mahindra", "tata"}

print(a,type(a))

a = True
```

```
print(a,type(a))
```

```
Data Science <class 'str'>
40 <class 'int'>
80.7 <class 'float'>
5j <class 'complex'>
['maruti', 'toyota', 'mahindra', 'tata'] <class 'list'>
('maruti', 'toyota', 'mahindra', 'tata') <class 'tuple'>
range(0, 8) <class 'range'>
{'name': 'Lennon', 'age': 45} <class 'dict'>
{'maruti', 'toyota', 'mahindra', 'tata'} <class 'set'>
True <class 'bool'>
```

Strings are a sequence of characters that allow us to store multiple values in an organized and efficient way.

```
print("Democratizing Data Science Learning")
```

```
Democratizing Data Science Learning
```

Iterate String

To iterate through a string in Python, “for…in” notation is used.

```
str = "DPhi"
for c in str:
  print(c)
```

```
D
P
h
i
```

Built-in Function len()

In Python, the built-in len() function can be used to determine the length of an object. It can be used to compute the length of strings, lists, sets, and other countable objects.

```
length = len("SkillSpace")
print(length)

brand = ['raymond', 'peter_england', 'allen_solly']
print(len(brand))
```

```
10
3
```

String Concatenation

To combine the content of two strings into a single string, Python provides the + operator. This process of joining strings is called concatenation.

```
x = 'Democratizing Data, '
y = 'Science Learning.'

z = x + y

print(z)
```

```
Democratizing Data, Science Learning.
```

String Method .lower()

The string method .lower() returns a string with all uppercase characters converted into lowercase.

```
greeting = "Hearty Welcome to Absolute Barbeque"

print(greeting.lower())
```

```
hearty welcome to absolute barbeque
```

Accessing String Elements

```
String1 = "SkillSpace.ai"

print("Initial String: ")
print(String1)

# Printing First character
print("\nFirst character of String is: ")
print(String1[0])

# Printing Last character
print("\nLast character of String is: ")
print(String1[-1])
```

```
Initial String:
SkillSpace.ai

First character of String is:
S

Last character of String is:
i
```

Indexing and Slicing Strings

Python strings can be indexed using the same notation as lists, since strings are lists of characters. A single character can be accessed with bracket notation ([index]), or a substring can be accessed using slicing ([start:end]).

Indexing with negative numbers counts from the end of the string.

```
str = 'yellow'
str[1]     # => 'e'
str[-1]    # => 'w'
str[4:6]   # => 'ow'
str[:4]    # => 'yell'
str[-3:]   # => 'low'

{"type":"string"}
```

String Method .strip()

The string method .strip() can be used to remove characters from the beginning and end of a string.

A string argument can be passed to the method, specifying the set of characters to be stripped. With no arguments to the method, whitespace is removed.

```
text1 = '   apples and oranges   '
text1.strip()       # => 'apples and oranges'

text2 = '...+...lemons and limes...-...'

# Here we strip just the "." characters
text2.strip('.')    # => '+...lemons and limes...-'

# Here we strip both "." and "+" characters
text2.strip('.+')   # => 'lemons and limes...-'

# Here we strip ".", "+", and "-" characters
text2.strip('.+-')  # => 'lemons and limes'

{"type":"string"}
```

String Method .split()

The string method .split() splits a string into a list of items:

If no argument is passed, the default behavior is to split on whitespace. If an argument is passed to the method, that value is used as the delimiter on which to split the string.

```
text = "Bengaluru IT Hub"

print(text.split())

print(text.split('i'))
```

```
['Bengaluru', 'IT', 'Hub']
['Bengaluru IT Hub']
```

Python string method .find()

The Python string method .find() returns the index of the first occurrence of the string passed as the argument. It returns -1 if no occurrence is found.

```
highest_peak = "Mount Everest"
print(highest_peak.find("o"))

1
```

String .replace()

The .replace() method is used to replace the occurence of the first argument with the second argument within the string.

The first argument is the old substring to be replaced, and the second argument is the new substring that will replace every occurence of the first one within the string.

```
fruit = "Raspberry Pi"
print(fruit.replace('r', 'R'))

# StRawbeRRy

RaspbeRRy Pi
```

What are Lists in Python?

Python Lists are just like dynamically sized arrays, declared in other languages (vector in C++ and ArrayList in Java). In simple language, a list is a collection of things, enclosed in [ ] and separated by commas.

```
Var = ["Democratizing", "Data-Science", "Learning"]
print(Var)

['Democratizing', 'Data-Science', 'Learning']
```

An overview of the various list methods

Accessing elements from the List

```
List = ["Democratizing", "Data-Science", "Learning"]

# we need to access an element from the list using it's index number
print("Accessing a element from the list")
```

```
print(List[0])
print(List[2])
```

```
Accessing a element from the list
Democratizing
Learning
```

Accessing elements from a multi-dimensional list

```
List = [['Skill', 'Space'], ['AI']]

# accessing an element from the Multi-Dimensional List using index
number
print("Accessing a element from a Multi-Dimensional list")
print(List[0][1])
print(List[1][0])
```

```
Accessing a element from a Multi-Dimensional list
Space
AI
```

Append in Python Lists

```
my_list = ['Democratizing', 'Data-Science']
my_list.append('Education')
print(my_list)
```

```
['Democratizing', 'Data-Science', 'Education']
```

insert() method

- append() method only works for the addition of elements at the end of the List, for the addition of elements at the desired position, insert() method is used.

```
List = [1,9,3,2]
print("Initial List: ")
print(List)

# Addition of Element at specific Position (using Insert Method)
List.insert(1, 42)
List.insert(0, 'DPhi')
print("\nList after performing Insert Operation: ")
print(List)
```

```
Initial List:
[1, 9, 3, 2]

List after performing Insert Operation:
['DPhi', 1, 42, 9, 3, 2]
```

Reversing a List

A list can be reversed by using the reverse() method in Python.

```
# Reversing a list
mylist = [9,7,5,6,48, 'Data', 'Science']
mylist.reverse()
print(mylist)
```

```
['Science', 'Data', 48, 6, 5, 7, 9]
```

Using remove() method

- Elements can be removed from the List by using the built-in remove() function but it throws an Error arises if the element is not present in the list.
- Remove() method only removes one element at a time.The remove() method removes the specified item.

```
List = [1, 2, 3, 4, 5, 6,  7,"DPhi", 8, 9, 10, 11, 12]
print("\nList before deletion: ")
print(List)

List.remove("DPhi")
print("\nList after Removing a range of elements: ")
print(List)
```

```
List before deletion:
[1, 2, 3, 4, 5, 6, 7, 'DPhi', 8, 9, 10, 11, 12]

List after Removing a range of elements:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

Slicing in Python

```
List = ['M','A','C','H','I','N','E','L','E','A','R','N','I','N','G']
print("Initial List : ")
print(List)

# Print elements of a range using Slice operation
Sliced_List = List[3:8]
print("\nSlicing elements in a range 3-8: ")
print(Sliced_List)

# Print elements from a pre-defined point to end
Sliced_List = List[5:]
print("\nElements sliced from 5th "
      "element till the end: ")
print(Sliced_List)

# Printing elements from beginning till end
```

```
Sliced_List = List[:]
print("\nPrinting all elements using slice operation: ")
print(Sliced_List)

# Print elements from beginning to a pre-defined point using Slice
Sliced_List = List[:-6]
print("\nElements sliced till 6th element from last: ")
print(Sliced_List)

# Print elements of a range using negative index List slicing
Sliced_List = List[-6:-1]
print("\nElements sliced from index -6 to -1")
print(Sliced_List)

# Printing elements in reverse using Slice operation
Sliced_List = List[::-1]
print("\nPrinting List in reverse: ")
print(Sliced_List)
```

```
Initial List :
['M', 'A', 'C', 'H', 'I', 'N', 'E', 'L', 'E', 'A', 'R', 'N', 'I', 'N',
'G']

Slicing elements in a range 3-8:
['H', 'I', 'N', 'E', 'L']

Elements sliced from 5th element till the end:
['N', 'E', 'L', 'E', 'A', 'R', 'N', 'I', 'N', 'G']

Printing all elements using slice operation:
['M', 'A', 'C', 'H', 'I', 'N', 'E', 'L', 'E', 'A', 'R', 'N', 'I', 'N',
'G']

Elements sliced till 6th element from last:
['M', 'A', 'C', 'H', 'I', 'N', 'E', 'L', 'E']

Elements sliced from index -6 to -1
['A', 'R', 'N', 'I', 'N']

Printing List in reverse:
['G', 'N', 'I', 'N', 'R', 'A', 'E', 'L', 'E', 'N', 'I', 'H', 'C', 'A',
'M']
```

count() method

- Python List count() method returns the count of how many times a given object occurs in a List.

```
list2 = ['a', 'a', 'a', 'b', 'b', 'a', 'c', 'b']
print(list2.count('b'))
```

```
3
```

sort()

- This function sorts the list in increasing order.

```
lis = [2, 1, 3, 5, 3, 8]

lis.sort()
print("\nPrinting the list in ascending order")
print(lis)

lis.sort(reverse=True)
print("\nPrinting the list in descending order")
print(lis)
```

```
Printing the list in ascending order
[1, 2, 3, 3, 5, 8]

Printing the list in descending order
[8, 5, 3, 3, 2, 1]
```

Tuples in Python

- It is a collection of objects separated by commas. In some ways, a tuple is similar to a list in terms of indexing, nested objects, and repetition but a tuple is immutable, unlike lists which are mutable.

```
var = ("Democratizing", "Data-Science", "Learning")
print(var)
```

```
('Democratizing', 'Data-Science', 'Learning')
```

Accessing Values in Tuples in Python

- Method 1: Using Positive Index Using square brackets we can get the values from tuples in Python.

```
var = ("Democratizing", "Data-Science", "Learning")

print("Value in Var[0] = ", var[0])
print("Value in Var[1] = ", var[1])
print("Value in Var[2] = ", var[2])
```

```
Value in Var[0] =  Democratizing
Value in Var[1] =  Data-Science
Value in Var[2] =  Learning
```

Method 2: Using Negative Index. In the above methods, we use the positive index to access the value in Python, and here we will use -ve index within [].

```
var = ("Democratizing", "Data-Science", "Learning")

print("Value in Var[-3] = ", var[-3])
print("Value in Var[-2] = ", var[-2])
print("Value in Var[-1] = ", var[-1])
```

```
Value in Var[-3] =  Democratizing
Value in Var[-2] =  Data-Science
Value in Var[-1] =  Learning
```

Concatenation of Tuples in Python

- To concatenate the Python tuple we will use plus operators(+).

```
# Code for concatenating 2 tuples

tuple1 = (1, 9, 8, 4)
tuple2 = ('Data', 'Science')

# Concatenating above two
print(tuple1 + tuple2)

#Nesting of Tuples in Python Code for creating nested tuples

tuple1 = (2, 3, 6, 4)
tuple2 = ('Data', 'Science')
tuple3 = (tuple1, tuple2)
print(tuple3)
```

```
(1, 9, 8, 4, 'Data', 'Science')
((2, 3, 6, 4), ('Data', 'Science'))
```

Slicing Tuples in Python

```
# code to test slicing

tuple_slice = (9 ,3, 8, 5)

print(tuple_slice[2:4])
print(tuple_slice[::-1])
print(tuple_slice[1:])
```

```
(8, 5)
(5, 8, 3, 9)
(3, 8, 5)
```

Finding Length of a Tuple

```
# Code for printing the length of a tuple

tuple_len = ('Data', 'Science')
print(len(tuple_len))
```

```
2
```

Converting list to a Tuple

```
# Code for converting a list and a string into a tuple

dphi = [0, 1, 2]
print(tuple(dphi))
print(tuple('SQL')) # string 'python'
```

```
(0, 1, 2)
('S', 'Q', 'L')
```

Dictionary in Python

- It is a collection of keys values, used to store data values like a map, which, unlike other data types which hold only a single value as an element.
- A dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'. Dictionary holds pairs of values, one being the Key and the other corresponding pair element being its Key:value. Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be immutable.

```
Dict = {1: 'Data', 2: 'is', 3: 'the', 4: 'new', 5:'oil'}
print(Dict)
```

```
{1: 'Data', 2: 'is', 3: 'the', 4: 'new', 5: 'oil'}
```

Dictionary containing keys of Integer and Mixed Types

```
# Creating a Dictionary with Integer Keys
Dict = {1: 'Democratizing', 2: 'Data-Science', 3: 'Learning'}
print("\nDictionary with the use of Integer Keys: ")
print(Dict)

# Creating a Dictionary with Mixed keys
Dict = {'Name': 'DPhi', 1: [1, 2, 3, 4]}
print("\nDictionary with the use of Mixed Keys: ")
print(Dict)
```

```
Dictionary with the use of Integer Keys:
{1: 'Democratizing', 2: 'Data-Science', 3: 'Learning'}
```

```
Dictionary with the use of Mixed Keys:
{'Name': 'APhi', 1: [1, 2, 3, 4]}
```

Adding elements to a Dictionary

- Addition of elements can be done in multiple ways. One value at a time can be added to a Dictionary by defining value along with the key e.g. Dict[Key] = 'Value'. Updating an existing value in a Dictionary can be done by using the built-in update() method.

Note- While adding a value, if the key-value already exists, the value gets updated otherwise a new Key with the value is added to the Dictionary.

```
# Creating an empty Dictionary
Dict = {}
print("Empty Dictionary: ")
print(Dict)

# Adding elements one at a time
Dict[0] = 'Skill'
Dict[2] = 'Space'
Dict[3] = 1
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Adding set of values to a single Key
Dict['Value_set'] = 2, 3, 4
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Updating existing Key's Value
Dict[2] = 'Grand Welcome'
print("\nUpdated key value: ")
print(Dict)

# Adding Nested Key value to Dictionary
Dict[5] = {'Nested': {'1': 'Life', '2': 'Geeks'}}
print("\nAdding a Nested Key: ")
print(Dict)
```

```
Empty Dictionary:
{}

Dictionary after adding 3 elements:
{0: 'Skill', 2: 'Space', 3: 1}

Dictionary after adding 3 elements:
{0: 'Skill', 2: 'Space', 3: 1, 'Value_set': (2, 3, 4)}

Updated key value:
```

```
{0: 'Skill', 2: 'Grand Welcome', 3: 1, 'Value_set': (2, 3, 4)}

Adding a Nested Key:
{0: 'Skill', 2: 'Grand Welcome', 3: 1, 'Value_set': (2, 3, 4), 5:
{'Nested': {'1': 'Life', '2': 'Geeks'}}}
```

Accessing elements of a Dictionary In order to access the items of a dictionary refer to its key name. Key can be used inside square brackets.

```
# Creating a Dictionary
Dict = {1: 'DPhi', 'Data': 'Science', 3: 'Learning'}

# accessing a element using key
print("Accessing a element using key:")
print(Dict['Data'])

# accessing a element using key
print("Accessing a element using key:")
print(Dict[1])
```

```
Accessing a element using key:
Science
Accessing a element using key:
DPhi
```

A collection of popular Dictionary methods that we use in our daily lives

```
# demo for all dictionary methods
dictionary = {1: "Python", 2: "C++", 3: "Ruby", 4: "R"}

# copy() method
dict2 = dictionary.copy()
print(dict2)

# clear() method
dictionary.clear()
print(dictionary)

# get() method
print(dict2.get(3))

# items() method
print(dict2.items())

# keys() method
print(dict2.keys())

# pop() method
dict2.pop(2)
```

```
print(dict2)

# popitem() method
dict2.popitem()
print(dict2)

# update() method
dict2.update({4: "Kotlin"})
print(dict2)

# values() method
print(dict2.values())
```

```
{1: 'Python', 2: 'C++', 3: 'Ruby', 4: 'R'}
{}
Ruby
dict_items([(1, 'Python'), (2, 'C++'), (3, 'Ruby'), (4, 'R')])
dict_keys([1, 2, 3, 4])
{1: 'Python', 3: 'Ruby', 4: 'R'}
{1: 'Python', 3: 'Ruby'}
{1: 'Python', 3: 'Ruby', 4: 'Kotlin'}
dict_values(['Python', 'Ruby', 'Kotlin'])
```