

303105257 - Programming in Python with Full Stack Development

Introduction to python programming
Computer Science & Engineering

Shaikh Mohd Maaz (Lecturer. PIET-CSE)



Introduction to python

- Python is an open source, interpreted, high-level, general-purpose programming language
- Python is a dynamically typed and garbage-collected language.
- Python was Created by Guido van Rossum and first released in 1991.
- Python 2.0, released in 2000,
- Python 3.0 was released in 2008 The latest version of python is 3.13.0, which was released on October 7, 2024.

Advantages:

- Easy to learn
- Less code
- Syntax is easier to read
- Open source
- Huge amount of additional open-source libraries

Variable in python:

- A Python variable is a reserved memory location to store values.
- Unlike other programming languages, Python has no command for declaring a variable.
 - A variable is created the moment you first assign a value to it.
 - Python uses Dynamic Typing so,
 - Rules for variable name
 - Name can not start with digit
 - Space not allowed
 - Can not contain special character
 - Python keywords not allowed
 - Should be in lower cases.

Cont....

```
1 x = 10
2 print(x)
3 print(type(x))
4
5 y = 123.456
6 print(y)
7
8 x = "milan"
9 print(x)
10 print(type(x))
```

Output

1	10
2	int
3	123.456
4	milan
5	str

Data Types In Python:

In Python, data types are categorized into several groups based on the kind of data they represent. Below is an overview of Python's data types:

1. Numeric Types

- **int**: Represents integers, e.g., 10, -5.
- **float**: Represents floating-point numbers (decimals), e.g., 3.14, -2.7.
- **complex**: Represents complex numbers with a real and imaginary part, e.g., 3 + 4j.

```
# int
x = 10
y = -5
print(type(x)) # Output: <class 'int'>
```

```
# float
pi = 3.14
negative_float = -2.7
print(type(pi)) # Output: <class 'float'>
```

```
# complex
z = 3 + 4j
print(type(z)) # Output: <class 'complex'>
```

Cont....

2. Sequence Types

- str: Represents a sequence of characters (text), e.g., "Hello, World!".
- list: An ordered, mutable collection, e.g., [1, 2, 3, "Python"].
- tuple: An ordered, immutable collection, e.g., (1, 2, 3, "Python").

```
# str
name = "Alice"
print(type(name)) # Output: <class 'str'>
```

```
# tuple
coordinates = (10.0, 20.0, 30.0)
print(type(coordinates)) # Output: <class 'tuple'>
```

```
# list
fruits = ["apple", "banana", "cherry"]
print(type(fruits)) # Output: <class 'list'>
```

Cont....

3. Mapping Types

- dict: A collection of key-value pairs, e.g., {"name": "Alice", "age": 25}.

```
# dict
person = {"name": "Alice", "age": 25, "city": "New York"}
print(type(person)) # Output: <class 'dict'>
```

4. Set Types

- set: An unordered, mutable collection of unique items, e.g., {1, 2, 3}.
- frozenset: An unordered, immutable collection of unique items, e.g., frozenset({1, 2, 3}).

```
# frozenset
immutable_set = frozenset({1, 2, 3, 4})
print(type(immutable_set)) # Output: <class 'frozenset'>
```

```
# set
unique_numbers = {1, 2, 3, 4}
print(type(unique_numbers)) # Output: <class 'set'>
```

Cont....

5. Boolean Type

- bool: Represents True or False.

```
# bool
is_python_fun = True
is_sky_green = False
print(type(is_python_fun)) # Output: <class 'bool'>
```

Control Statement & Loops

If statement.

- if statement is written using the if keyword followed by condition and colon(:)

```
1 if some_condition :  
2     # Code to execute when condition is true
```

if statement ends with :

```
1 x = 10  
2  
3 if x > 5 :  
4     print("X is greater than 5")
```

Output

X is greater than 5

Cont.

If else statement.

```
1 if some_condition :  
2     # Code to execute when condition is true  
3 else :  
4     # Code to execute when condition is false
```

```
1 x = 3  
2  
3 if x > 5 :  
4     print("X is greater than 5  
5 ")  
6 else :  
7     print("X is less than 5")
```

Output

```
1 X is less than 5
```

Cont....

If, elif and else statement.

```
1 if some_condition_1 :  
2     # Code to execute when condition 1 is true  
3 elif some_condition_2 :  
4     # Code to execute when condition 2 is true  
5 else :  
6     # Code to execute when both conditions are false
```

```
1 x = 10  
2 if x > 12 :  
3     print("X is greater than 1  
4 2")  
5 elif x > 5 :  
6     print("X is greater than 5  
7 ")  
8 else :  
9     print("X is less than 5")
```

Output

```
1 X is greater than 5
```

Cont.....

For loop in python.

- Many objects in python are iterable, meaning we can iterate over every element in the object.
- such as every elements from the List, every characters from the string etc..
- We can use for loop to execute block of code for each element of iterable object.

Syntax

```
1 for temp_item in iterable_object :  
2     # Code to execute for each object in iterable
```

For loop ends with :

Indentation (tab/whitespace) at the beginning

Cont....

While loop.

- While loop will continue to execute block of code until some condition remains True.
- For example,
 - while felling hungry, keep eating
 - while have internet pack available, keep watching videos

Syntax

```
1 while some_condition :  
2     # Code to execute in loop
```

while loop ends with :

Indentation (tab/whitespace) at the beginning

Cont....

break, continue & pass keywords.

- **break** : Breaks out of the current closest enclosing loop.
- **continue** : Goes to the top of the current closest enclosing loop.
- **Pass** : Does nothing at all, will be used as a placeholder in conditions where you don't want to write anything

breakdemo.py

```
1 for temp in range(5) :  
2     if temp == 2 :  
3         break  
4  
5     print(temp)
```

Output :
0
1

continuedemo.py

```
1 for temp in range(5) :  
2     if temp == 2 :  
3         continue  
4  
5     print(temp)
```

Output :
0
1
3
4

passdemo.py

```
1 for temp in range(5) :  
2     pass
```

Output : (nothing)

Heterogenous Data Types.

List.

- List is a mutable ordered sequence of objects, duplicate values are allowed inside list.
- List will be represented by square brackets []
- We can use slicing similar to string in order to get the sub list from the list.

Output : ['institute', 'rajkot']
Note : end index not included

```
1 my_list = ['darshan', 'institute', 'rajkot', 'gujarat', 'INDIA']  
2 print(my_list[1:3])
```

List Methods.

- `append()` method will add element at the end of the list.

```
1 my_list = ['milan', 'institute', 'rajkot']  
2 my_list.append('gujarat')  
3 print(my_list)
```

Output : ['milan', 'institute', 'rajkot', 'gujarat']

- `insert()` method will add element at the specified index in the list.

```
1 my_list = ['milan', 'institute', 'rajkot']  
2 my_list.insert(2, 'of')  
3 my_list.insert(3, 'engineering')  
4 print(my_list)
```

Output : ['milan', 'institute', 'of', 'engineering', 'rajkot']

- `extend()` method will add one data structure (List or any) to current List.

```
1 my_list1 = ['milan', 'institute']  
2 my_list2 = ['rajkot', 'gujarat']  
3 my_list1.extend(my_list2)  
4 print(my_list1)
```

Output : ['milan', 'institute', 'rajkot', 'gujarat']

Cont....

- pop() method will remove the last element from the list and return it.

```
1 my_list = ['milan', 'institute', 'rajkot']  
2 temp = my_list.pop()  
3 print(my_list)
```

Output : ['milan', 'institute']

- pop() method will remove the last element from the list and return it.

```
1 my_list = ['milan', 'institute', 'darshan', 'rajkot']  
2 my_list.remove('milan')  
3 print(my_list)
```

Output : ['institute', 'darshan', 'rajkot']

- pop() method will remove the last element from the list and return it.

```
1 my_list = ['milan', 'institute', 'milan', 'rajkot']  
2 my_list.clear()  
3 print(my_list)
```

Output : []

Tuple.

- Tuple is a immutable ordered sequence of objects, duplicate values are allowed inside list.
- Tuple will be represented by round brackets ().
- Tuple is similar to List but List is mutable whereas Tuple is immutable

Output : ("milan", 'institute', 'of', 'engineering', 'of', 'rajkot')

Output : 3 (index of 'engineering')

Output : 2

Output : rajkot

```
1 my_tuple = ('milan', 'institute', 'of', 'engineering', 'of', 'rajkot')
2 print(my_tuple)
3 print(my_tuple.index('engineering'))
4 print(my_tuple.count('of'))
5 print(my_tuple[-1])
```

Dictionary.

- Dictionary is a unordered collection of key value pairs.
- Dictionary will be represented by curly brackets { }.
- Dictionary is mutable.

```
my_dict = { 'key1':'value1', 'key2':'value2' }
```

Key value is seperated by :

Key value pairs is seperated by ,

```
1 my_dict = {'college':'milan', 'city':'rajkot','type':'engineering'}  
2 print(my_dict['college'])  
3 print(my_dict.get('city'))
```

values can be accessed using key inside square brackets as well as using get() method
Output : *milan*
rajkot

Dictionary Methods.

`keys()` method will return list of all the keys associated with the Dictionary.

`values()` method will return list of all the values associated with the Dictionary.

`items()` method will return list of tuples for each key value pair associated with the Dictionary.

Set.

- Set is a unordered collection of unique objects.
- Set will be represented by curly brackets { }.
- Set has many in-built methods such as add(), clear(), copy(), pop(), remove() etc.. which are similar to methods we have previously seen.
- Only difference between Set and List is that Set will have only unique elements and List can have duplicate elements.

Output : {1, 2, 3, 5, 9}

```
1 my_set = {1,1,1,2,2,5,3,9}
2 print(my_set)
```

Working with String.

String in python

- String is an Ordered Sequence of characters such as “milk”, ‘college’, “cat” etc.
- String can be represented as single, double, or triple quotes.
- String in Python is immutable.
- Square brackets can be used to access elements of the string,

String functions in python:

- Python has lots of built-in methods that you can use on strings, we are going to cover some frequently used methods for strings like

- len()
- count()
- capitalize(), lower(), upper()
- istitle(), islower(), isupper()
- find(), rfind(), replace()
- index(), rindex() etc. 1 2

Note: len() is not the method of the string but can be used to get the length of the string

```
1 x = "milan"  
2 print(len(x))
```

Output : 5 (length of "milan")

String Methods.

count() method will return the number of times a specified value occurs in a string.

```
1 x = "milan"  
2 ca = x.count('a')  
3 print(ca)
```

Output : 1 (occurrence of 'a' in "milan")

title(), **lower()**, **upper()** will returns capitalized, lower case and upper case string respectively

```
1 x = " milan,Institute, rajkot"  
2 c = x.title()  
3 l = x.lower()  
4 u = x.upper()  
5 print(c)  
6 print(l)  
7 print(u)
```

Output : Milan, Institute, Rajkot

Output : milan, institute, rajkot

Output : MILAN INSTITUTE, RAJKOT

Cont.

istitle(), **islower()**, **isupper()** will returns True if the given string is capitalized, lower case and upper case respectively.

```
1 x = '    milan    '  
2 f = x.strip()  
3 print(f)
```

Output : mialn(without space)

strip() method will remove whitespaces from both sides of the string and return the string.

```
1 x = 'milan, institute, rajkot'  
2 c = x.istitle()  
3 l = x.islower()  
4 u = x.isupper()  
5 print(c)  
6 print(l)  
7 print(u)
```

Output : *False*

Output : *True*

Output : *False*

rstrip() and **lstrip()** will remove whitespaces from right and left side respectively.

find() method will search the string and returns the index at which they find the specified value

```
1 x = 'milan institute, rajkot, india'  
2 f = x.find('in')  
3 print(f)
```

Output : 6 (occurrence of 'in' in x)

rfind() will search the string and returns the last index at which they find the specified value

```
1 x = 'milan institute, rajkot, india'  
2 r = x.rfind('in')  
3 print(r)
```

Output : 24 last occurrence of 'in' in x)

x DIGITAL LEARNING CONTENT

0



Parul[®] University



www.paruluniversity

