Python

Aditya Kumar 01/08/2025

1 Basic

- Python was created by Guido van Rossum in the early 90s. It is now one of the most popular languages in existence. I fell in love with Python for its syntactic clarity. It's basically executable pseudocode.
- Single line comments start with # (numbered symbol)

1.1 Comments

```python
# this is your single line comment.
```

- Multiline strings can be written using three "s, and are often used as documentation.

```python
""" this
    is
    your
    multi-line
    string
"""
# it can be used in printing ascii arts
# for example :
art="""
   _____                .__.__                 __
/  _  \   ______ ____ |__|__| _____ ________/  |_
/  /_\  \ /  ___// ___\|  |  | \__  \\_  __ \   __\
/    |    \\___ \\   \___|  |  |  / __ \|  | \/|  |
\____|__  /____  >\___  >__|__| (____  /__|   |__|
\/     \/     \/             \/
"""
# printing art
print(art)
```

1.2 helloWorld

- We can print any string with print("String")

- Here print() is a function.
- Anything between "" is String.

```
print("hello world")
```

- This will give output *hello World* on the console

1.3 Escape Sequence

- In python the escape sequence represent new line character
- We can print multiple line in shell using only one line of code.
- we use *\n* where we want to break the line

```
print("This is first line.\nThis is second line.\nThis is ↩
   third line.\n and goes on.........")
```

- Previous command will give us the following output

```
This is first line.
This is second line.
This is third line.
 and goes on.........
```

1.4 Concatenate

- Concatenate means joining two or more things together in sequence.
- It usually refers to joining strings, arrays, or lists end-to-end.
- The result of concatenation is a single combined object of the same type (string + string = string, list + list = list).
- Concatenation is order-sensitive: "A" + "B" is different from "B" + "A".
- Only compatible types can be concatenated (string + string works, string + number does not unless converted).
- Concatenation does not alter the original objects unless you explicitly reassign the result.

```
# String + String -> String
a = "Hello"
```

```
b = "World"
result = a + b
print(result)
print(type(result))

# Output
HelloWorld
<class 'str'>



# List + List -> List
list1 = [1, 2, 3]
list2 = [4, 5, 6]
result = list1 + list2
print(result)
print(type(result))


# Output
[1, 2, 3, 4, 5, 6]
<class 'list'>
```

1.5 type()

- *type()* is a built-in Python function used to find the type (class) of an object.
- It tells what kind of data (string, list, int, float, etc.) the object belongs to. [we will know about these in next section]
- Useful for debugging, validation, and understanding how Python treats different values.
- The return value of type() itself is a type object (like <class 'str'>, <class 'list'>).
- Objects of the same type can usually be combined, manipulated, or iterated in similar ways.
- For String ;

```
# for String

text = "Python"
print("text = " , text , type(text))


# output should be like
```

```
text =  Python <class 'str'>
```

- For Int ;

```
# for int
num = 42
print("num = " , num , type(num))


# output
num =  42 <class 'int'>
```

- For list ;

```
# for list
numbers = [1, 2, 3]
print("list = " , numbers , type(numbers))


# output
list =  [1, 2, 3] <class 'list'>
```

1.6 input()

- input() is a built-in Python function used to take user input from the keyboard.
- It always returns the entered data as a string, no matter what the user types.
- If you need another type (like int or float), you must convert the result using functions like int() or float().
- It pauses program execution until the user presses Enter.
- Useful for interactive programs where user input is required.
- String ;

```
# string
name = input("Enter your name: ")
print("Hello,", name)
print(type(name))

# output
Enter your name: roxx
```

```
Hello, roxx
<class 'str'>
```

- Converting Inputing to Integers ;

```
age = int(input("Enter your age: "))
print("You are", age, "years old")
print(type(age))


# output
Enter your age: 25
You are 25 years old
<class 'int'>
```

- Converting input to float ;

```
pi_val = float(input("Enter value of pi: "))
print("Pi is approx:", pi_val)
print(type(pi_val))


# output
Enter value of pi: 3.147
Pi is approx: 3.147
<class 'float'>
```

1.7 Typecasting

- *Typecasting* means converting one data type into another in Python.
- It is done using constructor functions like int(), float(), str(), list(), tuple(), etc.
- Typecasting is required when you want to perform operations that need specific types (e.g., arithmetic on numbers instead of strings).
- Some conversions are safe and natural (e.g., int("10") → 10), while others may raise errors (e.g., int("abc") → error).
- Typecasting always creates a new object of the target type; the original object remains unchanged.
- String -> Integers

```python
# String -> Integers
s = "123"
num = int(s)
print(num, type(num))


# output
123 <class 'int'>
```

- String -> float

```python
# String -> float
s = "3.14"
pi = float(s)
print(pi, type(pi))

# output
3.14 <class 'float'>
```

- Integer -> string

```python
#Integer -> String
n = 42
s = str(n)
print(s, type(s))


# output
# 42 <class 'str'>
```

- list -> tuple

```python
# List -> tuple

lst = [1, 2, 3]
t = tuple(lst)
print(t, type(t))

# output
# (1, 2, 3) <class 'tuple'>
```

- tuple -> list

```python
# tuple -> list

t = (4, 5, 6)
lst = list(t)
print(lst, type(lst))


# output
# [4, 5, 6] <class 'list'>
```

1.8 len() function

- len() is a built-in Python function that returns the number of items in an object.
- It works on sequences (strings, lists, tuples) and collections (sets, dictionaries).
- For strings, len() counts the number of characters.
- For lists, tuples, and sets, len() counts the number of elements.
- For dictionaries, len() counts the number of key-value pairs.
- It does not work on integers or floats directly (unsupported types).
- String ;

```python
a = " : "
# String
text = "Python"
print(text , len(text))

# output
# Python 6
```

- list ;

```python
# list
nums = [10, 20, 30, 40]
print(nums ,a, len(nums))

# output
# [10, 20, 30, 40]  :  4
```

- tuple ;

```python
        # tuple ;
        t = (1, 2, 3, 4, 5)
        print(t,a,len(t))


        # output
        # (1, 2, 3, 4, 5)  :  5
```

- set ;

```python
        # set ;
        s = {1, 2, 3, 4}
        print(s , a, len(s))

        # output
        # {1, 2, 3, 4}  :  4
```

1.9 Variables

- Variables in Python are names that store data values in memory.
- They act as references or labels pointing to objects (numbers, strings, lists, etc.).
- Python variables do not need explicit type declaration — the type is inferred from the assigned value.
- A variable's type can change if you assign a new value of a different type.
- Variable names are case-sensitive (Name and name are different).
- They must begin with a letter or underscore, and cannot start with a number.
- Assignment is done using the equals sign =.
- Multiple variables can be assigned in one line.
- Basic Variable Examples ;

```python
        x = 10
        y = "Python"
        print(x, y)

        # output
        # x and y :  10 Python
```

- Checking variable type ;

```
        a = 10
        print("a : " , type(a))
        # output
        # a :  <class 'int'>
```

- Dynamic typing (Types Change automatically);

```
        b = 100
        b = "Now I am a string"
        print("b : ", b , type(b))

        # output
        # b :  Now I am a string <class 'str'>
```

- Multiple assignments ;

```
        p, q, r = 1, 2, 3
        print("p,q and r : ", p, q, r)

        # output
        # p,q and r :  1 2 3
```

- Value to multiple variables ;

```
        # Same value to multiple variables
        j = k = l = "Data"
        print("value of j,k and l : ",j, k, l)

        # output
        # value of j,k and l :  Data Data Data
```

- Reassigment ;

```
        # Reassignment
        n = 5
        n = n + 2
        print("n : " ,n)

        # output
        # n :  7
```

- With different Variables ;

```python
# with different types

name = "Alice"
age = 25
height = 5.4
is_student = True
print(name, age, height, is_student)
print("name : " ,type(name) , "age : ", type(age) , "height : " , type(height) , "is_student : " ,type(is_student))

# output
# Alice 25 5.4 True name :  <class 'str'> age :  <class 'int'> height :  <class 'float'> is_student :  <class 'bool'>
```

- Swapping two numbers ;

```python
a =input("a : ")
b =input("b : ")

# temp Value

temp = a
a = b
b = temp

# printing

print("value of a after swapping : ",a)
print("value of b after swapping : ",b)


# output
'''
a : 123
b : 121233
value of a after swapping :  121233
value of b after swapping :  123

'''
```

- Band Name generator ;

```
city = input("What is the name of the city you grew up in ?? ↩
    : ")
pet =input("What is the name of you pet ?? : ")

print("Your band name could be " + city + " "+ pet)


# output

'''
$ python 10-BandNameGenerator.py

What is the name of the city you grew up in ?? : patna
What is the name of you pet ?? : cat
Your band name could be patna cat
'''
```

1.10 Primitive Data Types

- Primitive data types in Python are the most basic kinds of data used to represent simple values.
- They are immutable — once created, their values cannot be changed directly.
- Python's main primitive types are:
  1. int → integers (whole numbers)
  2. float → decimal or real numbers
  3. bool → boolean values (True or False)
  4. str → strings (text)
  5. complex → complex numbers (with real and imaginary parts)
- Integer(int) ;

```
print("\n\n\nFor Int ;")
x = 10
y = -25
print("x and y : ",x, y)
print("type of x is : " ,type(x))
print("adding x and y : " , (x+y))

# output
'''
x and y :  10 -25
```

```
        type of x is :  <class 'int'>
        adding x and y :  -15

        '''
```