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

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

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

```
""" 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.

```
1         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

```
1         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

```
1         This is first line.
2         This is second line.
3         This is third line.
4          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.

```
1         # String + String -> String
2         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
```

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

- For Int ;

```
1          # for int
2          num = 42
3          print("num = " , num , type(num))
4
5
6          # output
7          num =  42 <class 'int'>
```

- For list ;

```
1          # for list
2          numbers = [1, 2, 3]
3          print("list = " , numbers , type(numbers))
4
5
6          # output
7          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 ;

```
1          # string
2          name = input("Enter your name: ")
3          print("Hello,", name)
4          print(type(name))
5
6          # output
7          Enter your name: roxx
```

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

- Converting Inputing to Integers ;

```
1         age = int(input("Enter your age: "))
2         print("You are", age, "years old")
3         print(type(age))
4
5
6         # output
7         Enter your age: 25
8         You are 25 years old
9         <class 'int'>
```

- Converting input to float ;

```
1         pi_val = float(input("Enter value of pi: "))
2         print("Pi is approx:", pi_val)
3         print(type(pi_val))
4
5
6         # output
7         Enter value of pi: 3.147
8         Pi is approx: 3.147
9         <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

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


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

- String -> float

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

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

- Integer -> string

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


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

- list -> tuple

```
# List -> tuple

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

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

- tuple -> list

```
        # 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 ;

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

        # output
        # Python 6
```

- list ;

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

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

- tuple ;

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


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

- set ;

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

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