Input and Output in Python

Input

Python provides us with two inbuilt functions to read the input from the keyboard.

- input(prompt)
- raw_input(prompt)

input()
This function first takes the input from the user and converts it into a string. The type of the returned object always will be <class 'str'>. It does not evaluate the expression, it just returns the complete statement as String.

How the input function works in Python :

When the input() function executes, program flow will be stopped until the user has given input. The text or message displayed on the output screen to ask a user to enter an input value is optional i.e. the prompt, which will be printed on the screen is optional.
Whatever you enter as input, the input function converts it into a string. if you enter an integer value, the input() function converts it into a string. You need to explicitly convert it into an integer in your code using typecasting .

Example:

```
num = input ("Enter number :")
print(num)
name = input("Enter name : ")
print(name)


print ("type of number", type(num))
print ("type of name", type(name))
```

Output:

```
Enter number :20
20
Enter name : kabin
kabin
```

```
type of number <class 'str'>
type of name <class 'str'>
```

Since input() always returns a string, if we need a different data type (like an integer or float), we must explicitly convert it using functions like int(), float(), etc.

For example:

```
name = input("Enter name: ")
roll = int(input("Enter roll number: "))
ispass = bool(input("Enter True for pass and False for fail: "))
print("Student Information:")
print("Name: ", name)
print("Roll Number: ", roll)
print("Result: ", ispass)
```

Output:

```
Enter name: Kabin
Enter marks: 44
Enter True for pass and False for fail: True
Student Information:
Name:  Kabin
Roll Number:  44
Result:  True
```

In the above example the problem is that instead of writing True if we write any other string then bool() will always return True, so we have solution for that which is **eval().**

The problem is:

```
ispass = bool(input("Enter True for pass and False for fail: "))
print(ispass)
```

Output:

```
Enter True for pass and False for fail: hello
True
```

So we use eval() function , lets see by example:

```
name = input("Enter name: ")
roll = int(input("Enter roll number: "))
ispass = eval(input("Enter True for pass and False for fail: "))
print("Student Information:")
print("Name: ", name)
print("Roll Number: ", roll)
print("Result: ", ispass)
```

Output Case1:

```
Enter name: Kabin
Enter roll number: 44
Enter True for pass and False for fail: True
Student Information:
Name:  Kabin
Roll Number:  44
Result:  True
```

Case2:

```
Enter name: Shyam
Enter roll number: 101
Enter True for pass and False for fail: False
Student Information:
Name:  Shyam
Roll Number:  101
Result:  False
```

argv - argument vector

In Python, argv (short for "argument vector") is a list that stores command-line arguments passed to a Python script. It comes from the sys module, and is useful when you want to provide inputs to a program directly from the command line, rather than interactively via

input().
Example:

```
from sys import argv

print("The number of command line arguemtn", len(argv))
print("The command line arguments are: ", argv)
print(type(argv))

for x in argv:
    print(x)
```

Here, we input arguments from command line as follows:

```
PS G:\INTERN\05-Day5> &
C:/Users/girik/AppData/Local/Programs/Python/Python312/python.exe
g:/INTERN/05-Day5/input/argv.py 10 20 30 40 50
The number of command line arguemtn 6
The command line arguments are:
['g:/INTERN/05-Day5/input/argv.py', '10', '20', '30', '40', '50']
<class 'list'>
g:/INTERN/05-Day5/input/argv.py
10
20
30
40
50
```

Taking multiple inputs at same time in python:

In Python, we can take multiple values from the user at the same time by using the input() function along with the split() method. This allows us to get multiple inputs in a single line and split them into separate variables.

Example 1:

```
# Input multiple values separated by spaces
name, age, city = input("Enter your name, age, and city (separated
by spaces): ").split()
```

```
print("Name:", name)
print("Age: ", age)
print("City: ", city)
```

Output:

```
Enter your name, age, and city (separated by spaces): ABC XYZ PQR
Name: ABC
Age:  XYZ
City:  PQR
```

Example 2: Using List Comprehension

```
numbers = [int(n) for n in input("Enter numbers separated by
spaces: ").split()]
print("Numbers:", numbers)
```

Output:

```
Enter numbers separated by spaces: 10 20 30
Numbers: [10, 20, 30]
```

Example 3: Using map

```
x, y, z = map(int, input("Enter three numbers separated by
spaces:").split())
print(f"x: {x}, y: {y}, z: {z}")
```

Output:

```
Enter three numbers separated by spaces: 100 200 900
x: 100, y: 200, z: 900
```

Output in Python

Output using print() function

Python print() function prints the message to the screen or any other standard output device.

Syntax:

```
print(value(s), sep= ' ', end = '\n', file=file, flush=flush)
```

Parameters:

- value(s): Any value, and as many as you like. Will be converted to a string before printed
- sep=’separator’ : (Optional) Specify how to separate the objects, if there is more than one.Default :’ ‘
- end=’end’: (Optional) Specify what to print at the end.Default : ‘\n’
- file : (Optional) An object with a write method. Default :sys.stdout
- flush : (Optional) A Boolean, specifying if the output is flushed (True) or buffered (False). Default: False

Return Type: It returns output to the screen.

Example:

```
print("Hello")
print("Nepal")
print(123)
print(3.145)
print(False)
```

Output:

```
Hello
Nepal
123
3.145
False
```

So we can see as specified earlier the print() function automatically moves to next line.

Using formatted string literal provides a way to embed expressions inside string literals, using curly braces {}. This makes formatting strings simpler and more readable.

Example:

```
name = "Hari"
age = 21

print(f"My name is {name} and I am {age} years old.")
```

Output:

```
My name is Hari and I am 21 years old.
```

Operations in String

```
first_name = "Rajesh"
last_name = "Hamal"

print(first_name + " " + last_name)

dialogue = "Hey!"
print(dialogue * 5)
```

Output:

```
Rajesh Hamal
Hey!Hey!Hey!Hey!Hey!
```

In Python, the % operator can be used for string formatting, allowing you to insert variables into a string. The %d and %f format specifiers are used for formatting integers and floating-point numbers, respectively.

Example:

```
age = 22
print("I am %d years old" %age)
```

Output:

```
I am 22 years old
```

Using %f Controlling Decimal Places

We can control the number of decimal places displayed with %f by specifying a precision. For example, %.2f formats the number to two decimal places:

```
PI = 3.14159265358979323846264338327950288419 7

print("The value of PI is %.2f" %PI)
```

The value of PI is 3.14