*All the IPython Notebooks in **Python Introduction** lecture series by **[Dr. Milaan Parmar](https://www.linkedin.com/in/milaanparmar/)** are available @ **[GitHub](https://github.com/milaan9/01_Python_Introduction)** </small>

# Python Input, Output and Import

This class focuses on two built-in functions `print()` and `input()` to perform I/O task in Python. Also, you will learn to import modules and use them in your program.

Python provides numerous **built-in functions (https://github.com/milaan9/04_Python_Functions/tree/main/002_Python_Functions_Built_in)** that are readily available to us at the Python prompt.

Some of the functions like `print()` and `input()` are widely used for standard input and output operations respectively. Let us see the output section first.

## 1. Python Output Using `print()` function

We use the `print()` function to output data to the standard output device (screen). We can also **output data to a file (https://github.com/milaan9/05_Python_Files/blob/main/001_Python_File_Input_Output.ipynb)**, but this will be discussed later.

An example of its use is given below.

In [1]:
```
# Example 1:

print('This sentence is output to the screen')
```

```
This sentence is output to the screen
```

In [2]:
```
# Example 2:

a = 9
print('The value of a is', a)
```

```
The value of a is 9
```

In the second `print()` statement, we can notice that space was added between the **string (https://github.com/milaan9/02_Python_Datatypes/blob/main/002_Python_String.ipynb)** and the value of variable `a`. This is by default, but we can change it.

**Syntax** of the `print()` function is:

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

Here, objects is the value(s) to be printed.

The `sep` separator is used between the values. It defaults into a space character.

After all values are printed, `end` is printed. It defaults into a new line.

The `file` is the object where the values are printed and its default value is `sys.stdout` (screen). Here is an example to illustrate this.

In [3]:
```
print(1, 2, 3, 4)
print(1, 2, 3, 4, sep='#')   # It will separate your elements with '#'
print(1, 2, 3, 4, sep='*', end='&')  # It will separate your elements with '*' and end with
'&'
```

```
1 2 3 4
1#2#3#4
1*2*3*4&
```

## Output formatting

Sometimes we would like to format our output to make it look attractive. This can be done by using the **`str.format()`** method. This method is visible to any string object.

In [4]:
```
x = 6; y = 12
print('The value of x is {} and y is {}'.format(x,y))
```

```
The value of x is 6 and y is 12
```

Here, the curly braces **`{}`** are used as placeholders. We can specify the order in which they are printed by using numbers (tuple index).

In [5]:
```
print('I love {0} and {1}'.format('Mango','Banana'))
print('I love {1} and {0}'.format('Mango','Banana'))
```

```
I love Mango and Banana
I love Banana and Mango
```

We can even use keyword arguments to format the string.

In [6]:
```
print('Hello {name}, {greeting}!'.format(greeting = 'Good morning', name = 'Mark'))
```

```
Hello Mark, Good morning!
```

We can also format strings like the old **`sprintf()`** style used in C programming language. We use the **`%`** operator to accomplish this.

In [7]:
```
x = 12.34567890
print('The value of x is %0.2f' %x)  # "%0.2" means only 2 number after decimal

print('The value of x is %0.3f' %x)  # "%0.3" means only 3 number after decimal
```

```
The value of x is 12.35
The value of x is 12.346
```

## 2. Python Input Using `input()` function

Up until now, our programs were static. The value of variables was defined or hard coded into the source code.

To allow flexibility, we might want to take the input from the user. In Python, we have a built-in function **`input()`** to accept user input.

**Syntax**:

```
input([prompt])
```

where **`prompt`** is the string we wish to display on the screen. It is optional.

In [8]:
```
num = input('Enter a number: ')
num
```

```
Enter a number: 90
```

Out[8]:
```
'90'
```

Here, we can see that the entered value **`90`** is a string, not a number. To convert this into a number we can use **`int()`** or **`float()`** functions.

In [9]:
```
int('90')  # converting string '90' to integer
```

Out[9]:
```
90
```

In [10]:
```
float('90')
```

Out[10]:
```
90.0
```

In [11]:
```
a = "6"  # Is this a STRING character
b = "P"  # IS this a STRING chracter
c = a + b
print(c)
```

```
6P
```

In [12]:
```
int('6+3') # ERROR! cannot add numbers as string data type
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-12-b4c816180ad9> in <module>
----> 1 int('6+3') # ERROR! cannot add numbers as string data type

ValueError: invalid literal for int() with base 10: '6+3'
```

This same operation can be performed using the **`eval()`** function. But **`eval`** takes it further. It can evaluate even expressions, provided the input is a string.

In [13]:
```
eval('6+3') # Eval function can add numbers in string data type
```

Out[13]:
```
9
```

In [14]:
```
eval('3*6')
```

Out[14]:
```
18
```

## Accepting User Inputs (as both integer and string)

**`input(prompt)`** prompts for and returns input as a string. Hence, if the user inputs a integer, the code should convert the string to an integer and then proceed.

In [15]:
```
a = input("Hello, \nHow are you?") # \n means new line
```

```
Hello,
How are you?I'm fine, thanks and you?
```

In [16]:
```
abc = input("Type something here and it will be stored in variable abc \t")
```

```
Type something here and it will be stored in variable abc       999999
```

In [17]:
```
type(abc)
```

Out[17]:
```
str
```

**Note**: **`type( )`** returns the format or the type of a variable or a number

In [18]:
```
name = input("Enter Student Name: ")
major = input("Enter Major: ")
university = input("Enter University: ")

print("\n")
print("Printing Student Details")
print("Name", "Major", "University")
print(name, major, university)
```

```
Enter Student Name: Arthur
Enter Major: IT
Enter University: Oxford


Printing Student Details
Name Major University
Arthur IT Oxford
```

In [19]:
```
number = input("Enter number: ")
name = input("Enter name: ")

print("\n")
print("Printing type of a input value")
print("type of number", type(number))
print("type of name", type(name))
```

```
Enter number: 007
Enter name: Bond


Printing type of a input value
type of number <class 'str'>
type of name <class 'str'>
```

## Accepting User Inputs (only as integer)

A useful function to use in conjunction with this is **`eval()`** which takes a string and evaluates it as a python expression.

*Note:* In notebooks it is often easier just to modify the code than to prompt for input.

In [20]:
```
xyz =  input("xyz = ")
print(xyz)
print(type(xyz))

xyzValue=eval(xyz)  # change input to integer data type from string data type
print(xyz,'=',xyzValue)
```

```
xyz = 66
66
<class 'str'>
66 = 66
```

In [21]:
```
int("99")
```

Out[21]:
```
99
```

In [22]:
```
# program to calculate addition of two input numbers

first_number = int(input("Enter first number: "))  # converting input value to integer
second_number = int(input("Enter second number: "))  # converting input value to integer

print("\n")
print("First Number: ", first_number)
print("Second Number: ", second_number)
sum1 = first_number + second_number
print("Addition of two number is: ", sum1)
```

```
Enter first number: 3
Enter second number: 6


First Number:  3
Second Number:  6
Addition of two number is:  9
```

## Practice Problem

Accept one integer and one float number from the user and calculate the addition of both the numbers.

In [23]:
```
num1 = int(input("Enter first number: "))
num2 = float(input("Enter second number: "))

result=(num1+num2)
print("Final result is: ", result)
```

```
Enter first number: 6
Enter second number: 3.9
Final result is:  9.9
```

In [24]:
```
# Write code to get three numbers and add first 2 number and multiply with third number

num1 = int(input("Enter first number: "))   # converting input value to integer
num2 = int(input("Enter second number: "))  # converting input value to integer
num3 = int(input("Enter third number: "))   # converting input value to integer

print("\n")
print("First Number: ", num1)
print("Second Number: ", num2)
print("Third Number: ", num3)

result=(num1+num2)*num3
print("Final result is: ", result)
```

```
Enter first number: 1
Enter second number: 2
Enter third number: 3


First Number:  1
Second Number:  2
Third Number:  3
Final result is:  9
```

In [25]:
```
# Write a code to get four numbers:
#Step 1. Multiply first and fourth number
#Step 2. Divide second and third number
#Step3. Add Step 1 and Step 2 outputs.

num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
num3 = int(input("Enter third number: "))
num4 = int(input("Enter fourth number: "))

print("\n")
print("First Number: ", num1)
print("Second Number: ", num2)
print("Third Number: ", num3)
print("Fourth number: ", num4)

result=(num1*num4)+(num2/num3)
print("Final result is: ", result)
```

```
Enter first number: 5
Enter second number: 6
Enter third number: 7
Enter fourth number: 8


First Number:  5
Second Number:  6
Third Number:  7
Fourth number:  8
Final result is:  40.857142857142854
```

In [26]:
```
# Write a code to get cube of the number

def cube(x):
    return x*x*x

a=int(input("Enter number: "))
print("cube of",a,"is",cube(a))
```

```
Enter number: 3
cube of 3 is 27
```

## Accept float input from User

Let's see how to accept float value from a user in Python. You need to convert user input to the float number using the **`float()`** function as we did for the integer value.

In [27]:
```
float_number = float(input("Enter float number: "))  # converting input value to float
print("\n")
print("input float number is: ", float_number)
print("type is:", type(float_number))
```

```
Enter float number: 9


input float number is:  9.0
type is: <class 'float'>
```

## Get multiple input values from a user in one line

In Python, It is possible to get multiple values from the user in one line. i.e., In Python, we can accept two or three values from the user in one **`input()`** call.

For example, in a single execution of the **`input()`** function, we can ask the user his/her name, age, and phone number and store it in three different variables. Lets see how to do this.

In [28]:
```
name, age, phone = input("Enter your Name, Age, Phone_number separated by space: ").split()
print("\n")
print("User Details: ", name, age, phone)
```

```
Enter your Name, Age, Phone_number separated by space: Clark 36 13756537093


User Details:  Clark 36 13756537093
```

## Accept multiline input from a user

As you know, the input() function does not allow the user to enter put lines separated by a newline. If the user tries to enter multiline input, it prints back only the first line. Let's see how to gets multiple line input.

As you know, the input() function does not allow the user to enter put lines separated by a newline. If the user tries to enter multiline input, it prints back only the first line. Let's see how to gets multiple line input.

We can use for loop. In each iteration, we can get input string from the user and then .join() them using \n, or you can also concatenate each input string using + operator separated by \n.

In [29]:
```
MultiLine = []
print("Tell me about yourself")
while True:
    line = input()
    if line:
        MultiLine.append(line)
    else:
        break
finalText = '\n'.join(MultiLine)
print("\n")
print("Final text input")
print(finalText)
```

```
Tell me about yourself
Hello! I am happy to learn Python and I love travelling.
Also, I love cooking and singing.



Final text input
Hello! I am happy to learn Python and I love travelling.
Also, I love cooking and singing.
```

## Format output strings by its positions

In [30]:
```
firstName = input("Enter First Name: ")
lastName = input("Enter Last Name: ")
organization = input("Enter Organization Name: ")

print("\n")
print('{0}, {1} works at {2}'.format(firstName, lastName, organization))
print('{1}, {0} works at {2}'.format(firstName, lastName, organization))
print('FirstName {0}, LastName {1} works at {2}'.format(firstName, lastName, organization))
print('{0}, {1} {0}, {1} works at {2}'.format(firstName, lastName, organization))
```

```
Enter First Name: Ethan
Enter Last Name: Hunt
Enter Organization Name: Mission Impossible


Ethan, Hunt works at Mission Impossible
Hunt, Ethan works at Mission Impossible
FirstName Ethan, LastName Hunt works at Mission Impossible
Ethan, Hunt Ethan, Hunt works at Mission Impossible
```

## Accessing output string arguments by name

In [31]:
```
name = input("Enter Name: ")
marks = input("Enter marks: ")

print("\n")
print('Student: Name:  {firstName}, Marks: {percentage}%'.format(firstName=name, percentage=m
arks))
```

```
Enter Name: David
Enter marks: 85


Student: Name:  David, Marks: 85%
```

## Output text alignment specifying a width

In [32]:
```
text = input("Enter text: ")

print("\n")
# left aligned
print('{:<25}'.format(text)) # Right aligned print('{:>25}'.format(text))
# centered
print('{:^25}'.format(text))
# right aligned
print('{:>25}'.format(text))
```

```
Enter text: Python


Python
          Python
                    Python
```

## Specifying a sign while displaying output numbers

In [33]:
```
positive_number = float(input("Enter Positive Number: "))
negative_number = float(input("Enter Negative Number: "))

print("\n")
# sign '+' is for both positive and negative number
print('{:+f}; {:+f}'.format(positive_number, negative_number))

# sign '-' is only for negative number
print('{:f}; {:-f}'.format(positive_number, negative_number))
```

```
Enter Positive Number: 9
Enter Negative Number: -4


+9.000000; -4.000000
9.000000; -4.000000
```

## Display output Number in various type format

In [34]:
```
number = int(input("Enter number: "))

print("\n")
# 'd' is for integer number formatting
print("The number is:{:d}".format(number))

# 'o' is for octal number formatting, binary and hexadecimal format
print('Output number in octal format : {0:o}'.format(number))

# 'b' is for binary number formatting
print('Output number in binary format: {0:b}'.format(number))

# 'x' is for hexadecimal format
print('Output number in hexadecimal format: {0:x}'.format(number))
```

```
Enter number: 63


The number is:63
Output number in octal format : 77
Output number in binary format: 111111
Output number in hexadecimal format: 3f
```

### Display numbers in floating-point format

In [35]:
```
number = float(input("Enter float Number: "))

print("\n")
# 'f' is for float number arguments
print("Output Number in The float type :{:f}".format(number))

# padding for float numbers
print('padding for output float number{:5.2f}'.format(number))

# 'e' is for Exponent notation
print('Output Exponent notation{:e}'.format(number))

# 'E' is for Exponent notation in UPPER CASE
print('Output Exponent notation{:E}'.format(number))
```

```
Enter float Number: 33


Output Number in The float type :33.000000
padding for output float number33.00
Output Exponent notation3.300000e+01
Output Exponent notation3.300000E+01
```

### Output String justification

Let's see how to use **`str.rjust()`**, **`str.ljust()`** and **`str.center()`** to justify text output on screen and console.

In [36]:
```
text = input("Enter String: ")

print("\n")
print("Left justification", text.ljust(60, "*"))
print("Right justification", text.rjust(60, "*"))
print("Center justification", text.center(60, "*"))
```

```
Enter String: Jupyter


Left justification Jupyter**************************************************
Right justification **************************************************Jupyter
Center justification ************************Jupyter************************
```

## 3. Python `Import` function

When our program grows bigger, it is a good idea to break it into different modules.

A module is a file containing Python definitions and statements. [Python modules (https://github.com/milaan9/04_Python_Functions/blob/main/007_Python_Function_Module.ipynb)](https://github.com/milaan9/04_Python_Functions/blob/main/007_Python_Function_Module.ipynb) have a filename and end with the extension **`.py`**.

Definitions inside a module can be imported to another module or the interactive interpreter in Python. We use the **`import`** keyword to do this.

For example, we can import the **`math`** module by typing the following line:

```
>import math
```

We can use the module in the following ways:

In [37]:
```
import math
print(math.pi) # do not have to make vairable for pi
```

```
3.141592653589793
```

Now all the definitions inside **`math`** module are available in our scope. We can also import some specific attributes and functions only, using the **`from`** keyword. For example:

In [38]:
```
from math import pi
pi
```

Out[38]:
```
3.141592653589793
```

While importing a module, Python looks at several places defined in **`sys.path`** . It is a list of directory locations.

In [39]:
```
import sys
sys.path
```

Out[39]:
```
['C:\\Users\\Deepak\\01_Learn_Python4Data\\01_Python_Introduction',
 'C:\\ProgramData\\Anaconda3\\python38.zip',
 'C:\\ProgramData\\Anaconda3\\DLLs',
 'C:\\ProgramData\\Anaconda3\\lib',
 'C:\\ProgramData\\Anaconda3',
 '',
 'C:\\Users\\Deepak\\AppData\\Roaming\\Python\\Python38\\site-packages',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\locket-0.2.1-py3.8.egg',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\win32',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\win32\\lib',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\Pythonwin',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\IPython\\extensions',
 'C:\\Users\\Deepak\\.ipython']
```

We can also add our own location to this list.

In [ ]:
```

```