Python Input, Output and Import

programiz.com/python-programming/input-output-import

Join our newsletter for the latest updates.

This tutorial 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.

Video: Python Take User Input

Python provides numerous built-in functions that are readily available to us at the Python prompt.

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

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, but this will be discussed later.

An example of its use is given below.

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

Output

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

Another example is given below:

```
a = 5
print('The value of a is', a)
```

Output

```
The value of a is 5
```

In the second `print()` statement, we can notice that space was added between the string and the value of variable *a*. This is by default, but we can change it.

The actual 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.

```
print(1, 2, 3, 4)
print(1, 2, 3, 4, sep='*')
print(1, 2, 3, 4, sep='#', end='&')
```

Output

```
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.

```
>>> x = 5; y = 10
>>> print('The value of x is {} and y is {}'.format(x,y))
The value of x is 5 and y is 10
```

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

```
print('I love {0} and {1}'.format('bread','butter'))
print('I love {1} and {0}'.format('bread','butter'))
```

Output

```
I love bread and butter
I love butter and bread
```

We can even use keyword arguments to format the string.

```
>>> print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning', name =
'John'))
Hello John, Goodmorning
```

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

```
>>> x = 12.3456789
>>> print('The value of x is %3.2f' %x)
The value of x is 12.35
>>> print('The value of x is %3.4f' %x)
The value of x is 12.3457
```

Python Input

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 the `input()` function to allow this. The syntax for `input()` is:

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

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

```
>>> num = input('Enter a number: ')
Enter a number: 10
>>> num
'10'
```

Here, we can see that the entered value `10` is a string, not a number. To convert this into a number we can use `int()` or `float()` functions.

```
>>> int('10')
10
>>> float('10')
10.0
```

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

```
>>> int('2+3')
Traceback (most recent call last):
  File "<string>", line 301, in runcode
  File "<interactive input>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '2+3'
>>> eval('2+3')
5
```

Python Import

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 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:

```
import math
print(math.pi)
```

Output

```
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:

```
>>> from math import pi
>>> pi
3.141592653589793
```

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

```
>>> import sys
>>> sys.path
['',
 'C:\\Python33\\Lib\\idlelib',
 'C:\\Windows\\system32\\python33.zip',
 'C:\\Python33\\DLLs',
 'C:\\Python33\\lib',
 'C:\\Python33',
 'C:\\Python33\\lib\\site-packages']
```

We can also add our own location to this list.