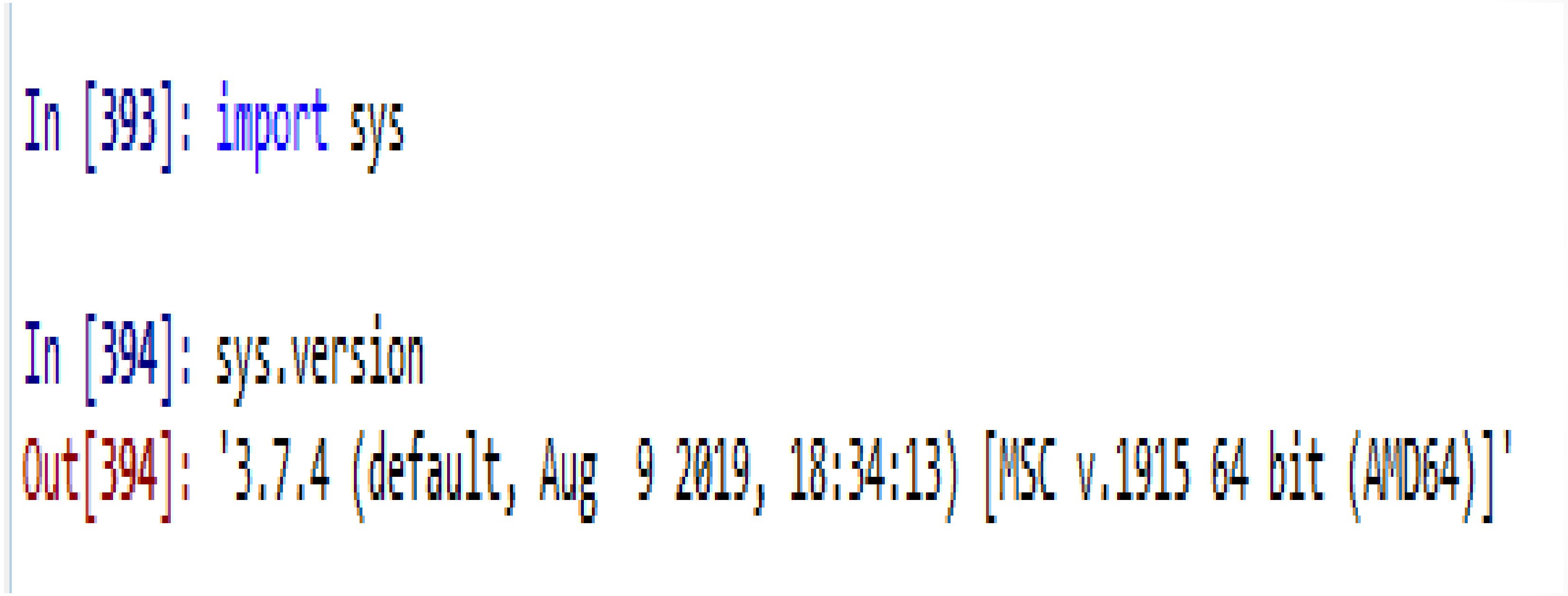
Unit-VII
System Level Interaction with Python

By:
Prof. Mani Butwall,
Asst. Prof. (CSE)

Contents

Sys Module

- The **sys module** in Python provides various functions and variables that are used to manipulate different parts of the Python runtime environment.
- It allows operating on the interpreter as it provides access to the variables and functions that interact strongly with the interpreter. Let's consider the below example.



```
In [393]: import sys


In [394]: sys.version
Out[394]: '3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)]'
```

- In the above example, **sys.version** is used which returns a string containing the version of Python Interpreter with some additional information.
- This shows how the sys module interacts with the interpreter.

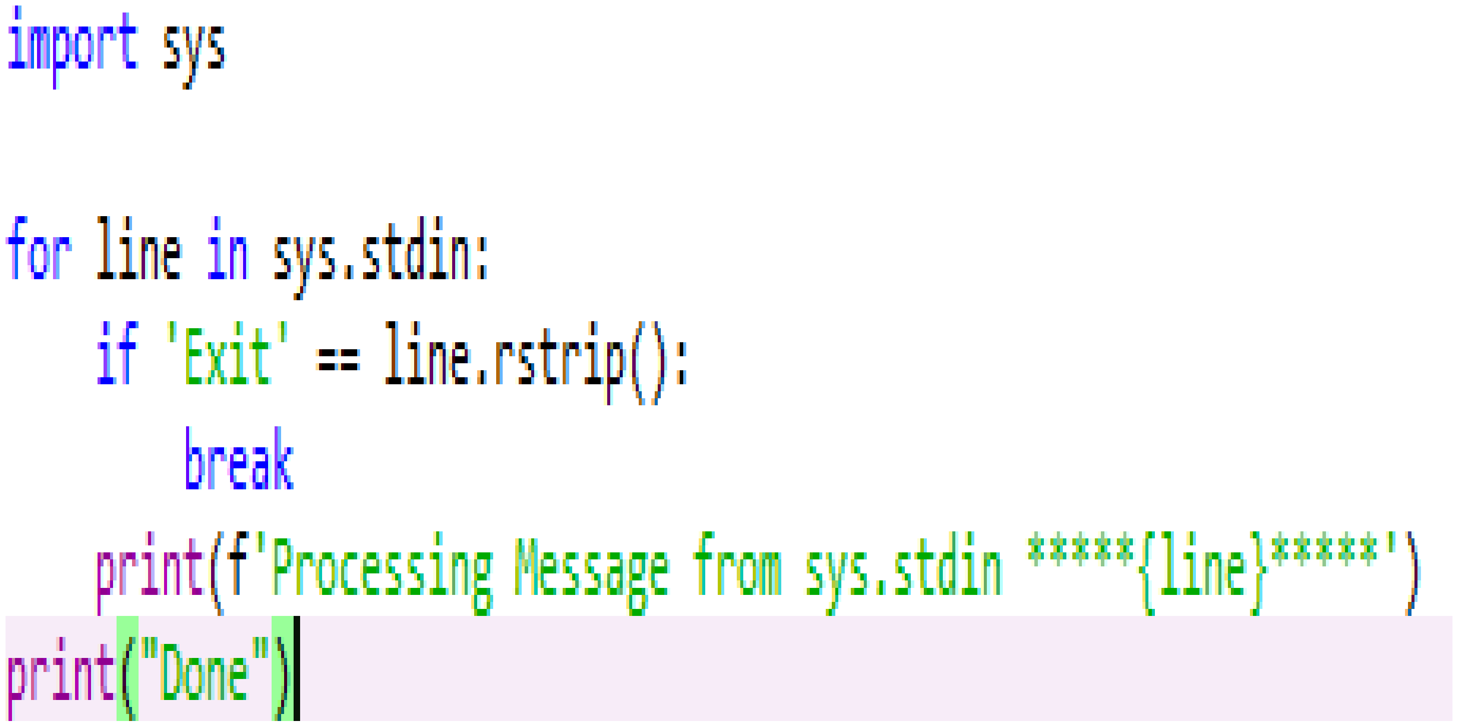
Input and Output using sys

- The sys modules provide variables for better control over input or output. We can even redirect the input and output to other devices. This can be done using three variables –

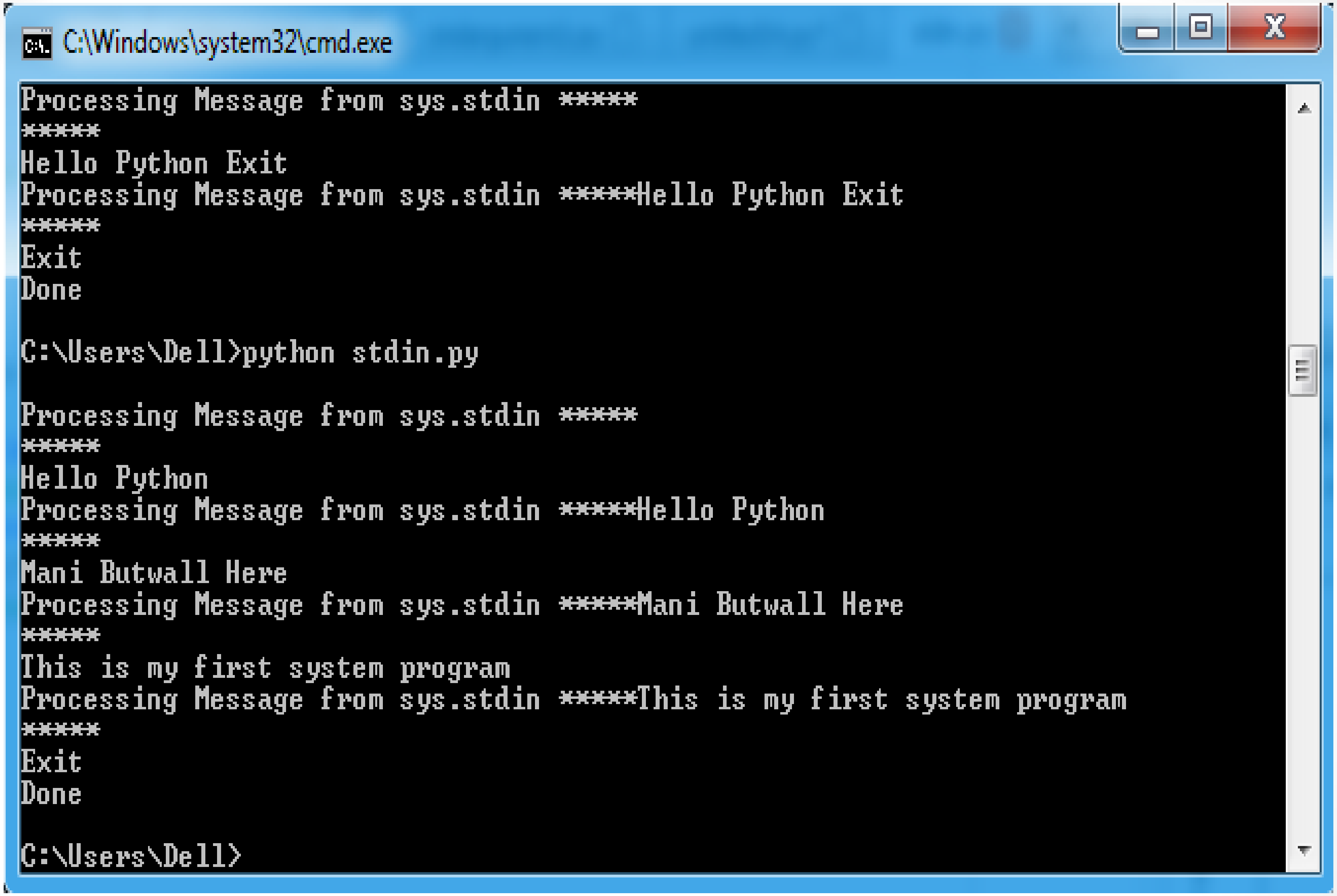
1. **stdin**
2. **stdout**
3. **stderr**

stdin

- **stdin:** It can be used to get input from the command line directly. It used is for standard input.
- It internally calls the input() method. It, also, automatically adds ‘\n’ after each sentence.



```python
import sys

for line in sys.stdin:
    if 'Exit' == line.rstrip():
        break
    print(f'Processing Message from sys.stdin *****{line}*****')
print("Done")
```

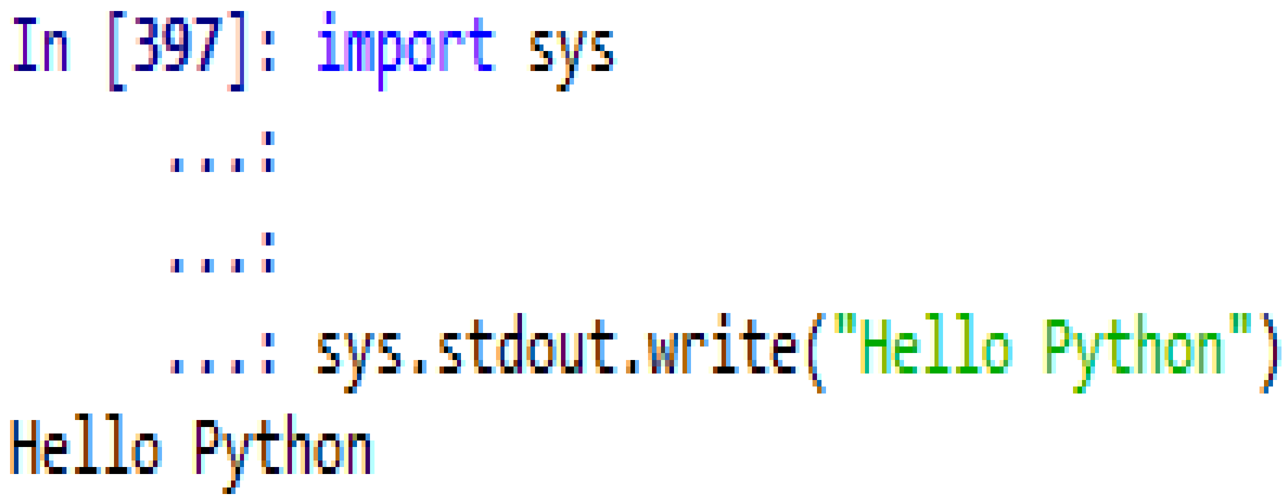


C:\Windows\system32\cmd.exe

```
Processing Message from sys.stdin *****
*****
Hello Python Exit
Processing Message from sys.stdin *****Hello Python Exit
*****
Exit
Done

C:\Users\Dell>python stdin.py

Processing Message from sys.stdin *****
*****
Hello Python
Processing Message from sys.stdin *****Hello Python
*****
Mani Butwall Here
Processing Message from sys.stdin *****Mani Butwall Here
*****
This is my first system program
Processing Message from sys.stdin *****This is my first system program
*****
Exit
Done

C:\Users\Dell>
```

stdout

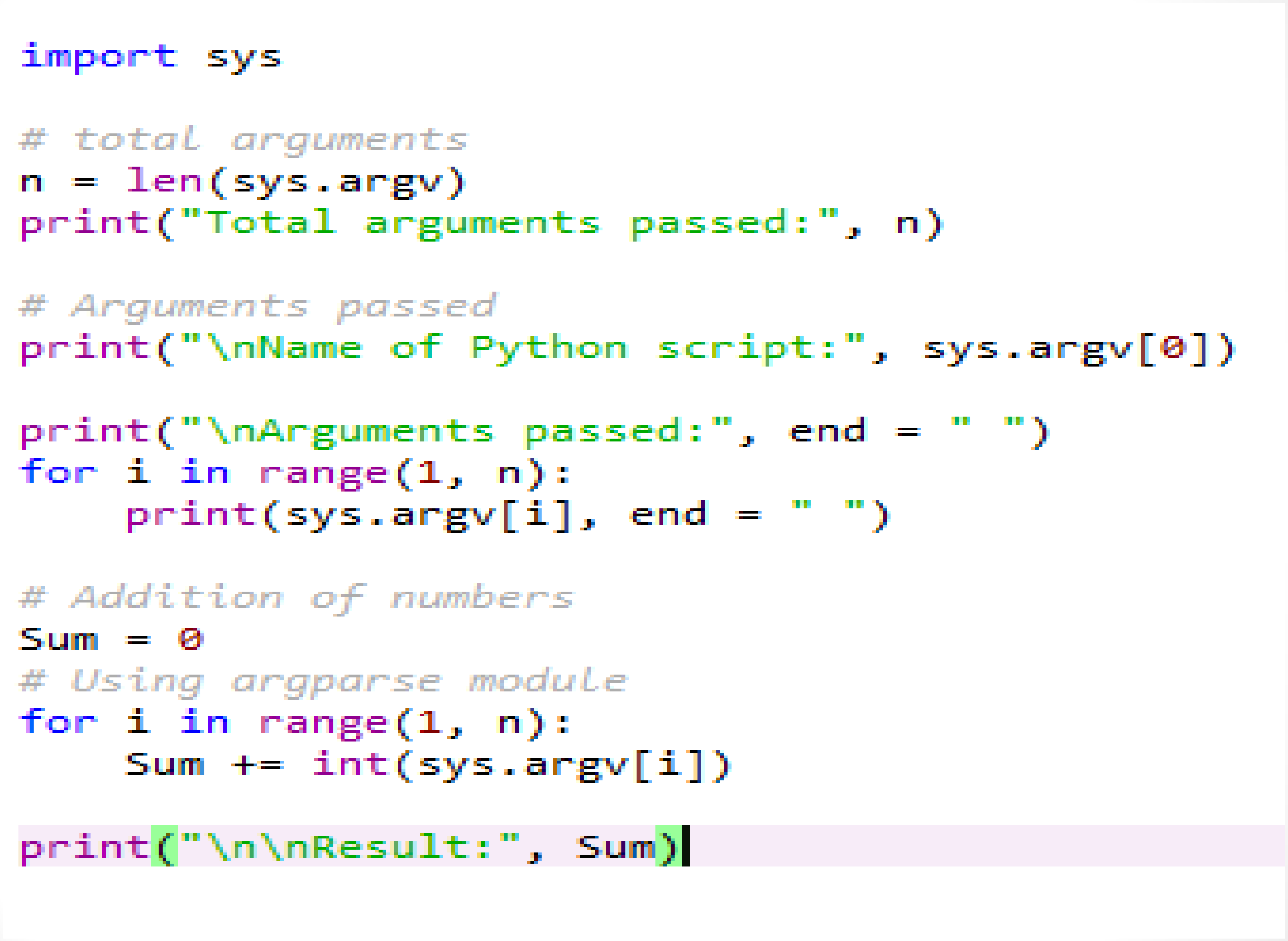
- **stdout:** A built-in file object that is analogous to the interpreter's standard output stream in Python.
- stdout is used to display output directly to the screen console. Output can be of any form, it can be output from a print statement, an expression statement, and even a prompt direct for input.
- By default, streams are in text mode. In fact, wherever a print function is called within the code, it is first written to sys.stdout and then finally on to the screen.



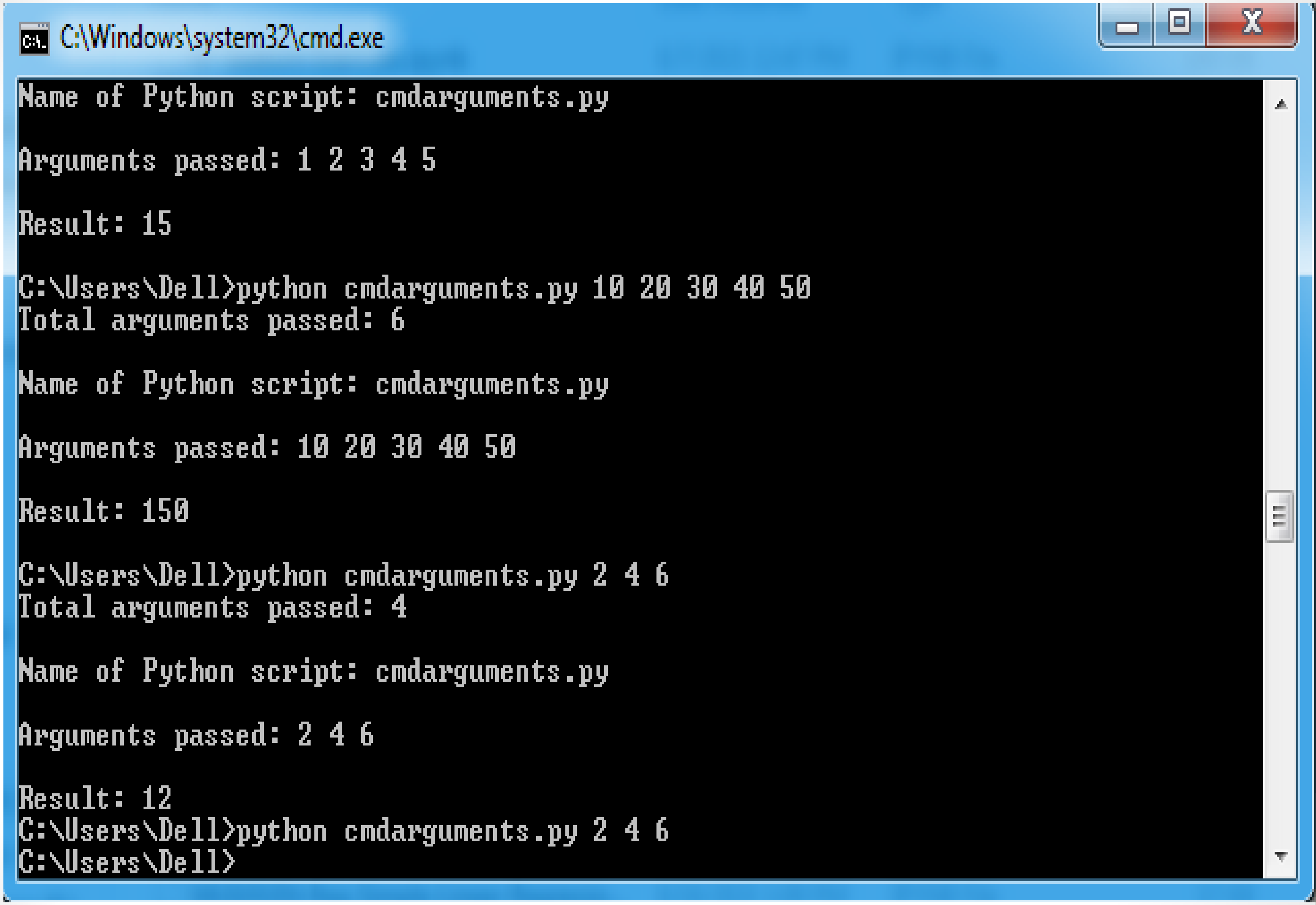
```
In [397]: import sys
     ...:
     ...:
     ...: sys.stdout.write("Hello Python")
Hello Python
```

Command Line Arguments

- **Command-line arguments** are those which are passed during the calling of the program along with the calling statement. To achieve this using the sys module, the sys module provides a variable called **sys.argv.** It’s main purpose are:
- It is a list of command-line arguments.
- len(sys.argv) provides the number of command-line arguments.
- sys.argv[0] is the name of the current Python script.
- **Example:** Consider a program for adding numbers and the numbers are passed along with the calling statement.



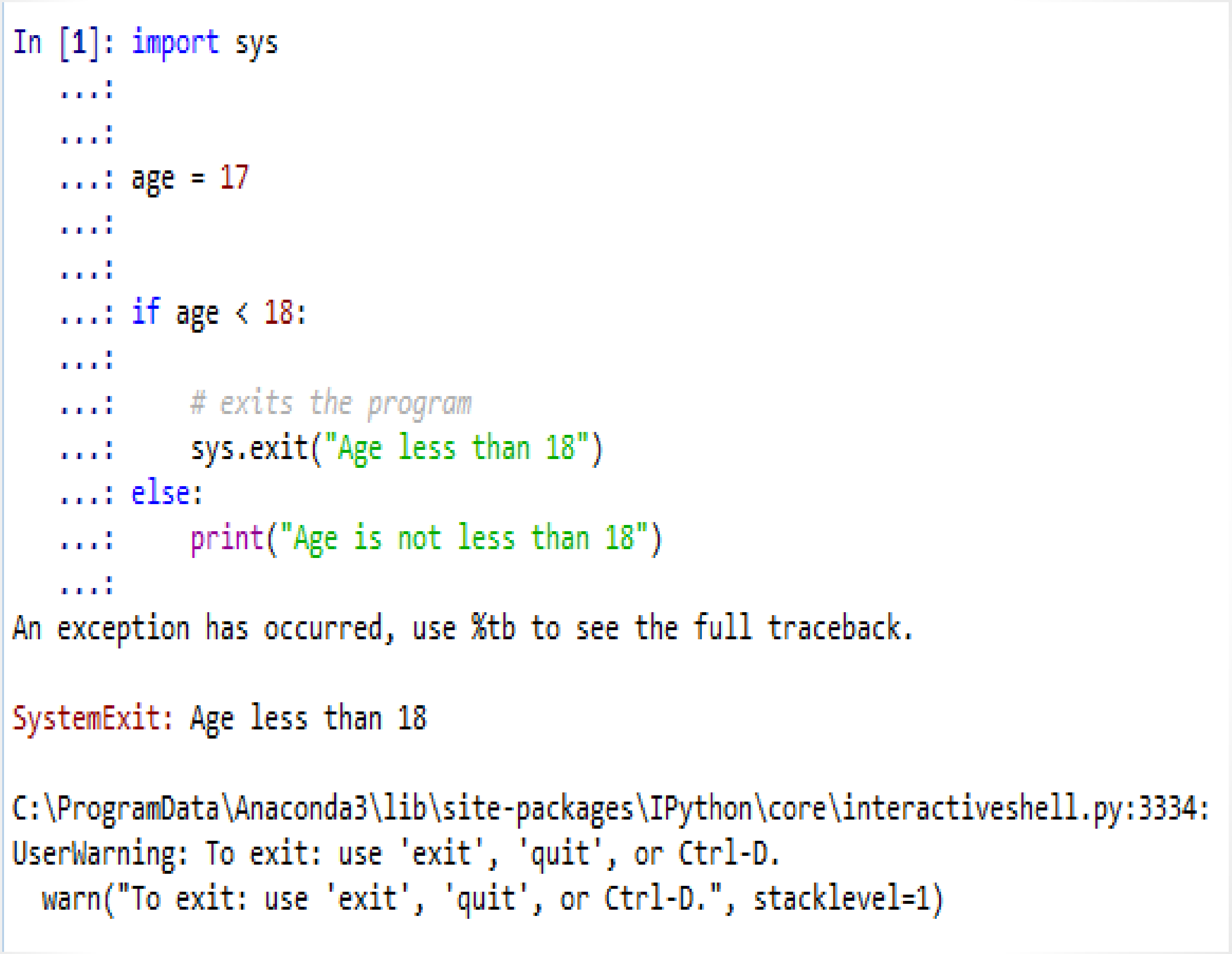
```python
import sys

# total arguments
n = len(sys.argv)
print("Total arguments passed:", n)

# Arguments passed
print("\nName of Python script:", sys.argv[0])

print("\nArguments passed:", end = " ")
for i in range(1, n):
    print(sys.argv[i], end = " ")

# Addition of numbers
Sum = 0
# Using argparse module
for i in range(1, n):
    Sum += int(sys.argv[i])

print("\n\nResult:", Sum)
```


C:\Windows\system32\cmd.exe

```
Name of Python script: cmdarguments.py

Arguments passed: 1 2 3 4 5

Result: 15

C:\Users\Dell>python cmdarguments.py 10 20 30 40 50
Total arguments passed: 6

Name of Python script: cmdarguments.py

Arguments passed: 10 20 30 40 50

Result: 150

C:\Users\Dell>python cmdarguments.py 2 4 6
Total arguments passed: 4

Name of Python script: cmdarguments.py

Arguments passed: 2 4 6

Result: 12
C:\Users\Dell>python cmdarguments.py 2 4 6
C:\Users\Dell>
```

Exit

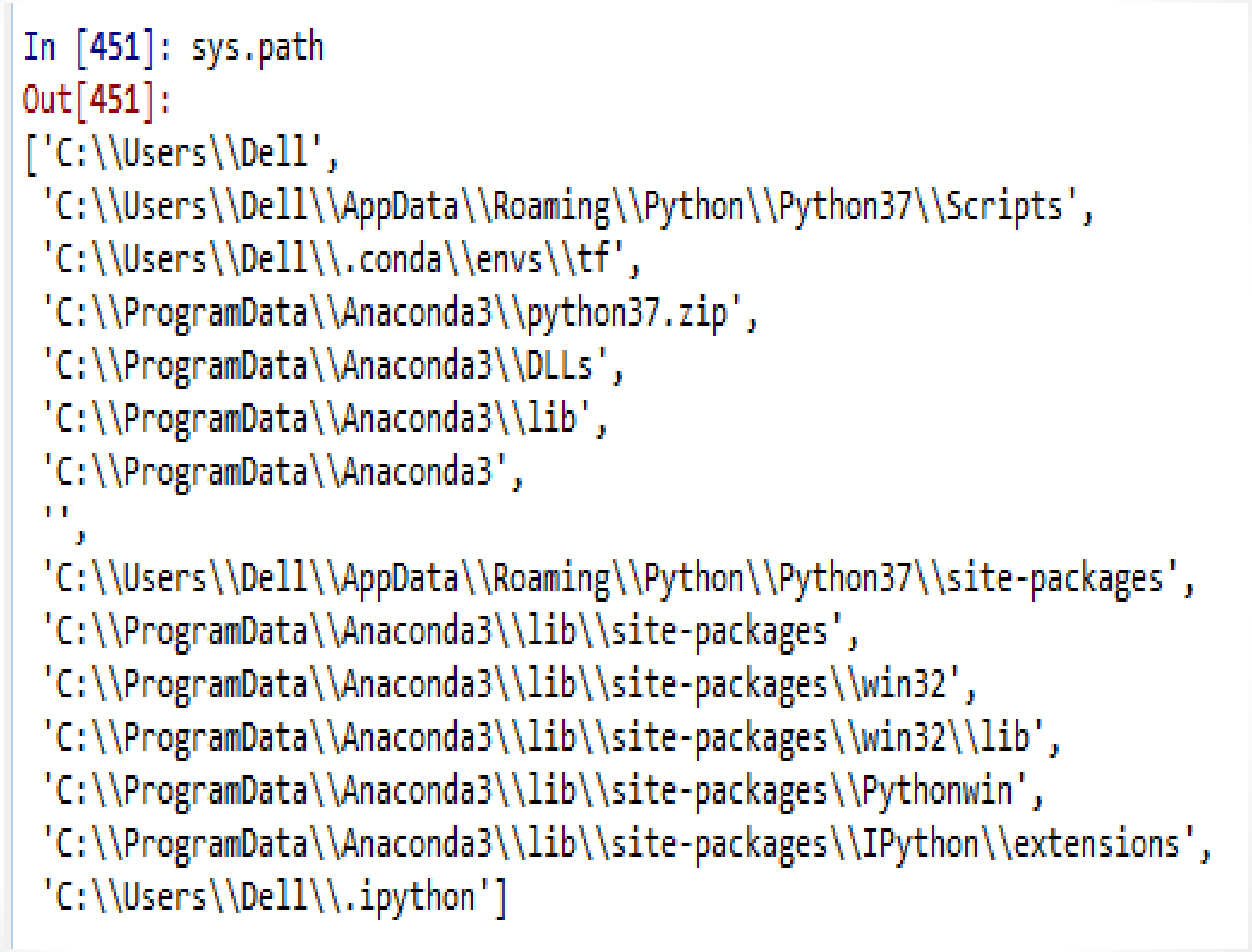
- **Exiting the Program**
- **sys.exit([arg])** can be used to exit the program. The optional argument arg can be an integer giving the exit or another type of object. If it is an integer, **zero is considered "successful termination".**
-



```
In [1]: import sys
   ...:
   ...:
   ...: age = 17
   ...:
   ...:
   ...: if age < 18:
   ...:
   ...:     # exits the program
   ...:     sys.exit("Age less than 18")
   ...: else:
   ...:     print("Age is not less than 18")
   ...:
An exception has occurred, use %tb to see the full traceback.

SystemExit: Age less than 18

C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3334:
UserWarning: To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

Working with Modules

- **sys.path** is a built-in variable within the sys module that returns the list of directories that the interpreter will search for the required module.
- When a module is imported within a Python file, the interpreter first searches for the specified module among its built-in modules. If not found it looks through the list of directories defined by **sys.path**.
- **Note:** sys.path is an ordinary list and can be manipulated.
- **Example 1:** Listing out all the paths



```
In [451]: sys.path
Out[451]:
['C:\\Users\\Dell',
 'C:\\Users\\Dell\\AppData\\Roaming\\Python\\Python37\\Scripts',
 'C:\\Users\\Dell\\.conda\\envs\\tf',
 'C:\\ProgramData\\Anaconda3\\python37.zip',
 'C:\\ProgramData\\Anaconda3\\DLLs',
 'C:\\ProgramData\\Anaconda3\\lib',
 'C:\\ProgramData\\Anaconda3',
 '',
 'C:\\Users\\Dell\\AppData\\Roaming\\Python\\Python37\\site-packages',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages',
 '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\\Dell\\.ipython']
```

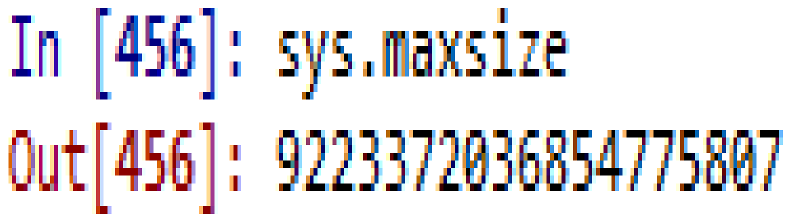
Reference Count

- **sys.getrefcount()** method is used to get the reference count for any given object. This value is used by Python as when this value becomes 0, the memory for that particular value is deleted.

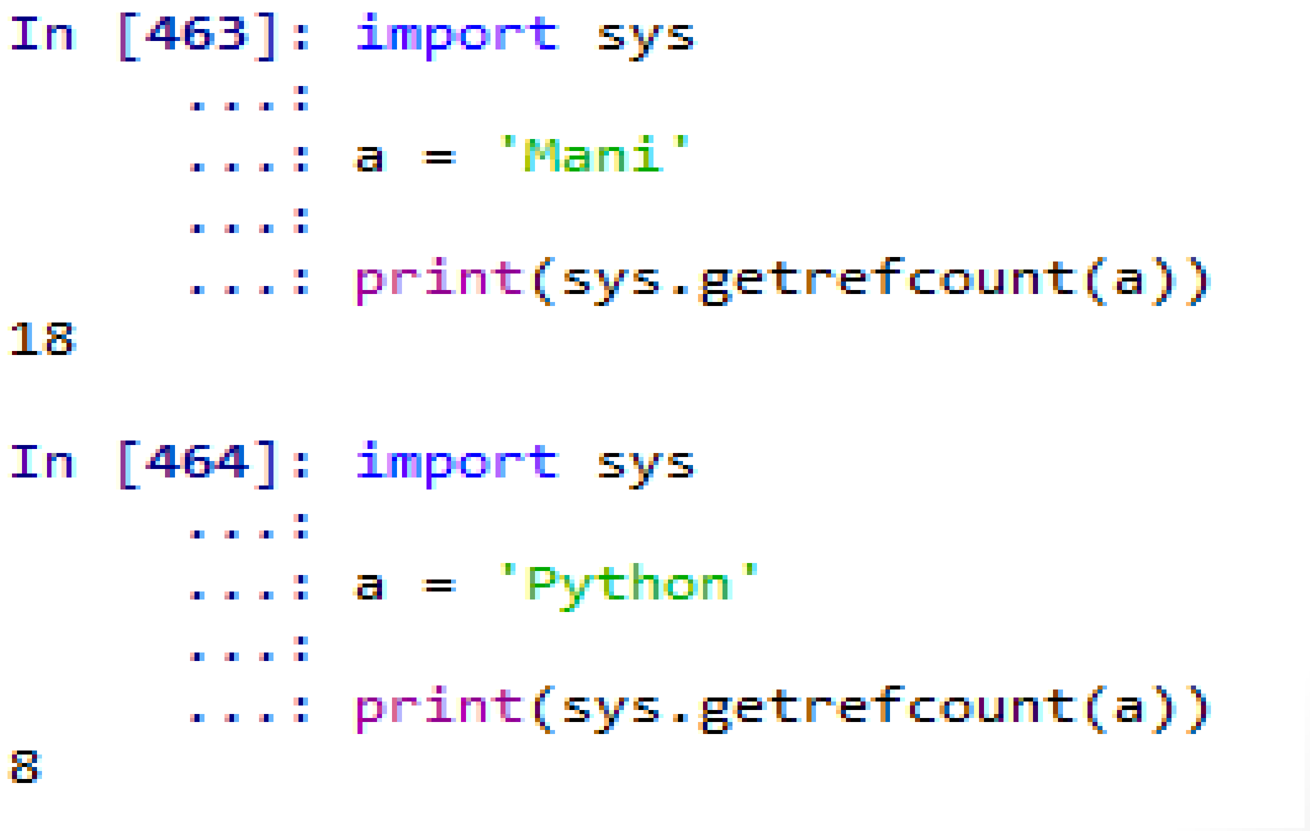
Function	Description
sys.setrecursionlimit()	sys.setrecursionlimit() method is used to set the maximum depth of the Python interpreter stack to the required limit.
sys.getrecursionlimit() method	sys.getrecursionlimit() method is used to find the current recursion limit of the interpreter or to find the maximum depth of the Python interpreter stack.
sys.settrace()	It is used for implementing debuggers, profilers and coverage tools. This is thread-specific and must register the trace using threading.settrace(). On a higher level, sys.settrace() registers the traceback to the Python interpreter
sys.setswitchinterval() method	sys.setswitchinterval() method is used to set the interpreter's thread switch interval (in seconds).
sys.maxsize()	It fetches the largest value a variable of data type Py_ssize_t can store.
sys.maxint	maxint/INT_MAX denotes the highest value that can be represented by an integer.
sys.getdefaultencoding() method	sys.getdefaultencoding() method is used to get the current default string encoding used by the Unicode implementation.

Function	Description
platform	Name of the platform on which Python is running, e.g. "linux2" for Linux and "win32" for Windows>>> sys.platform 'linux' >>>
version	Version number of the Python interpreter>>> sys.version '3.8.5 (default, Sep 4 2020, 07:30:14) \n[GCC 7.3.0]' >>>
version_info	Similar information than sys.version, but the output is a tuple containing the five components of the version number: major, minor, micro, release-level, and serial. The values of this tuple are integers except the value for the release level, which is one of the following: 'alpha', 'beta', 'candidate', or 'final'.>>> sys.version_info sys.version_info(major=3, minor=8, micro=5, releaselevel='final', serial=0) >>>

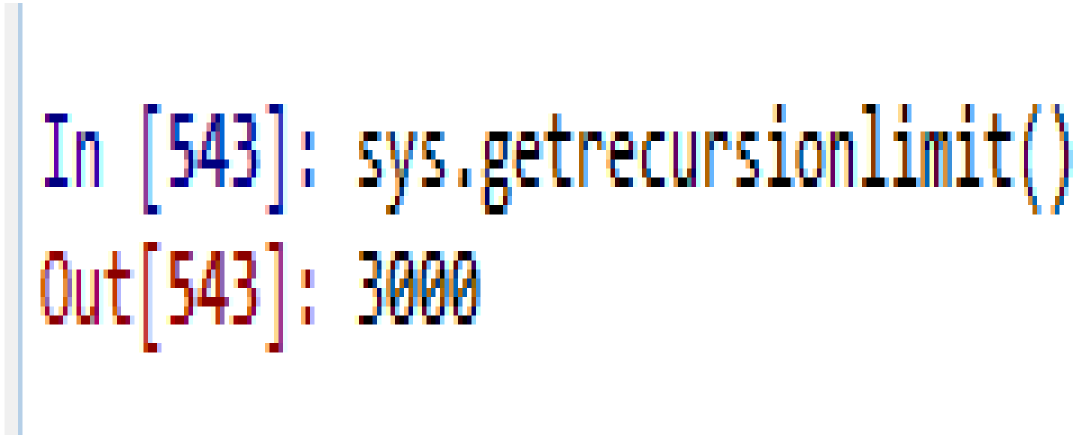
Function	Description
getsizeof	Return the size of object in bytes.>>> x = 5 >>> sys.getsizeof(x) 28 >>> s = "hello" >>> sys.getsizeof(s) 54
__stdin__ __stdout__ __stderr__	These attributes contain the original values of stdin, stderr and stdout at the start of the program. They can be useful to print to the actual standard stream no matter if the sys.std* object has been redirected. They can also be used to restore the actual files to known working file objects in case they have been overwritten with a broken object. But instead of using these values, the original stream should always be explicitly saved in a variable (as shown in our previous examples) before replacing it, and the original state should be restored by using the saved object.



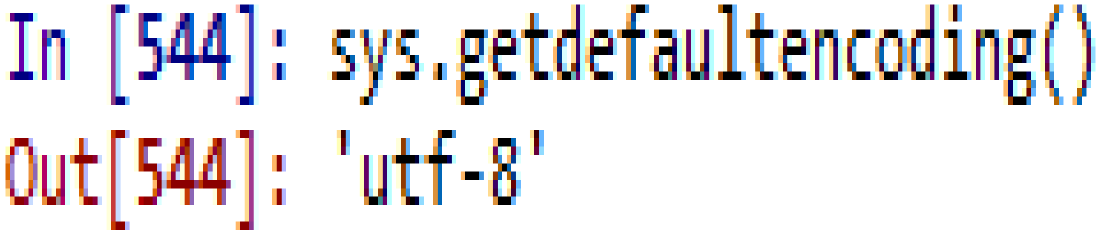
```
In [456]: sys.maxsize
Out[456]: 9223372036854775807
```



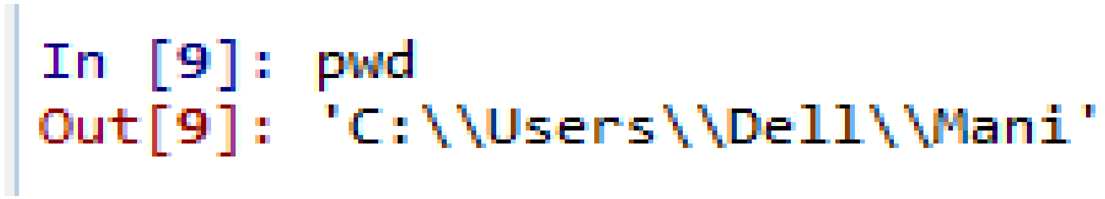
```
In [463]: import sys
     ...:
     ...: a = 'Mani'
     ...:
     ...: print(sys.getrefcount(a))
18

In [464]: import sys
     ...:
     ...: a = 'Python'
     ...:
     ...: print(sys.getrefcount(a))
8
```



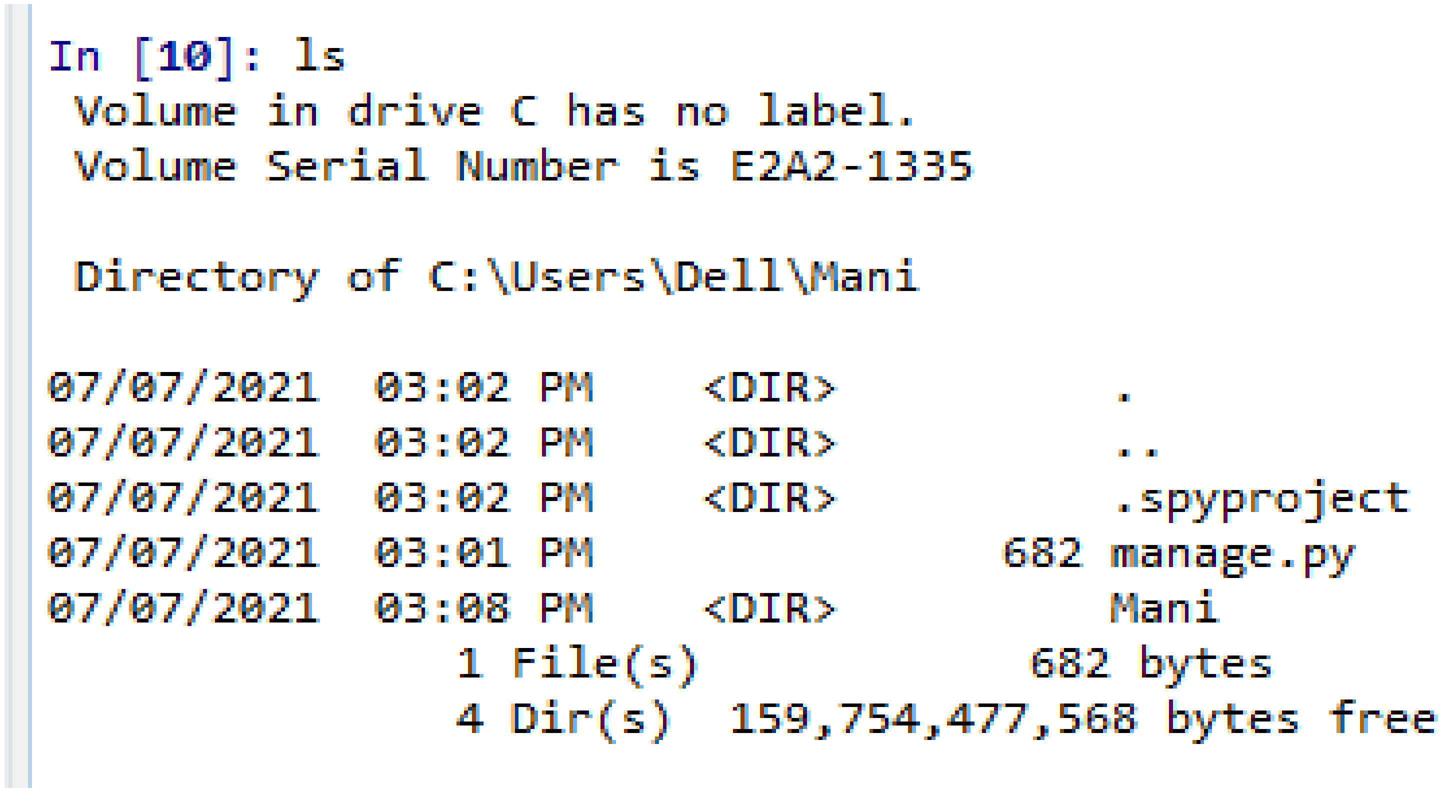
```
In [543]: sys.getrecursionlimit()
Out[543]: 3000
```



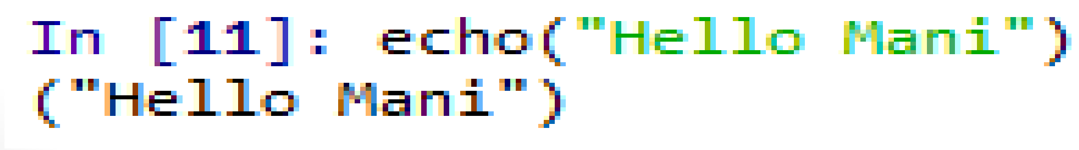
```
In [544]: sys.getdefaultencoding()
Out[544]: 'utf-8'
```



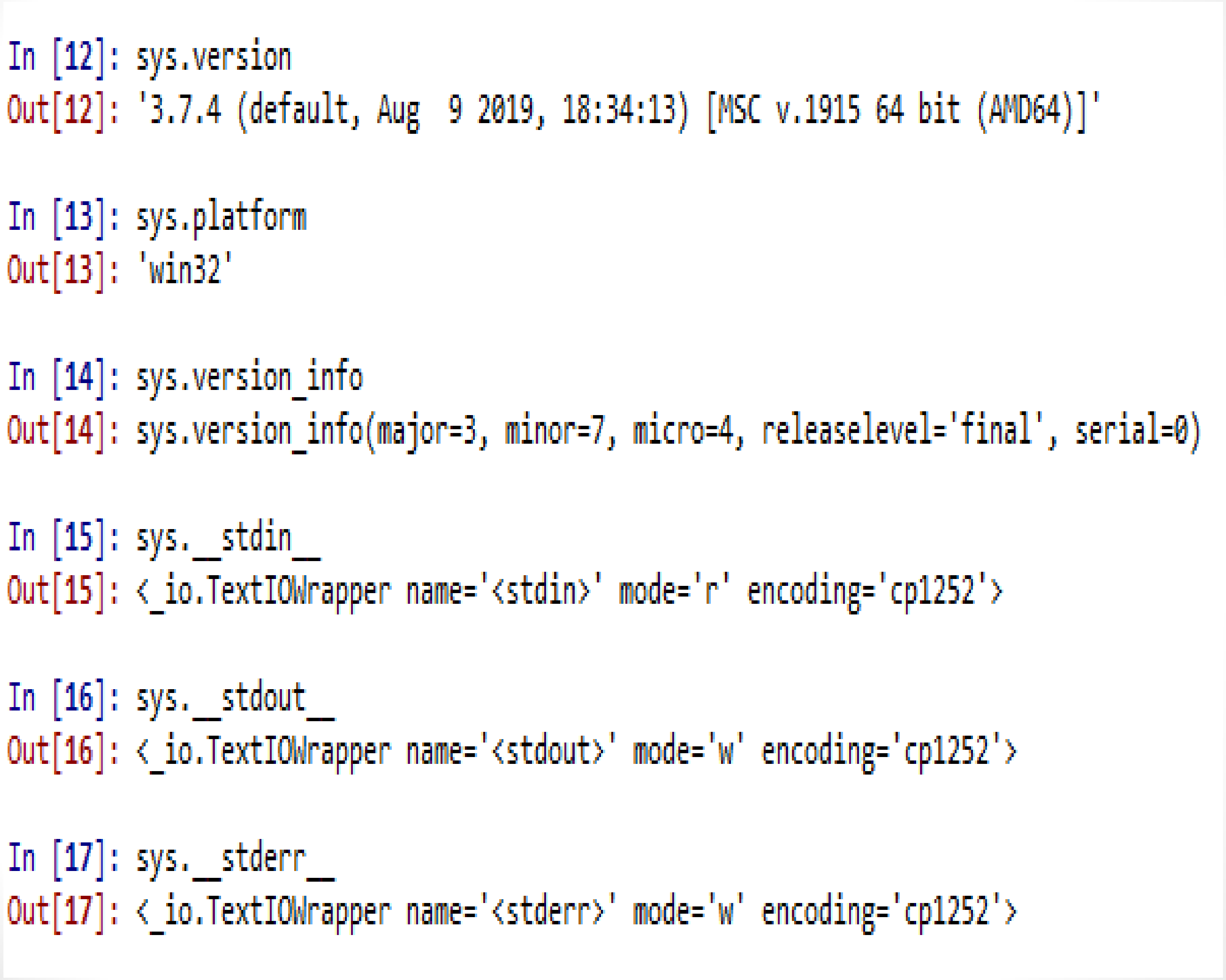
```
In [9]: pwd
Out[9]: 'C:\\Users\\Dell\\Mani'
```



```
In [10]: ls
 Volume in drive C has no label.
 Volume Serial Number is E2A2-1335

 Directory of C:\Users\Dell\Mani

07/07/2021  03:02 PM    <DIR>          .
07/07/2021  03:02 PM    <DIR>          ..
07/07/2021  03:02 PM    <DIR>          .spyproject
07/07/2021  03:01 PM               682 manage.py
07/07/2021  03:08 PM    <DIR>          Mani
               1 File(s)            682 bytes
               4 Dir(s)  159,754,477,568 bytes free
```



```
In [11]: echo("Hello Mani")
("Hello Mani")
```



```
In [12]: sys.version
Out[12]: '3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)]'

In [13]: sys.platform
Out[13]: 'win32'

In [14]: sys.version_info
Out[14]: sys.version_info(major=3, minor=7, micro=4, releaselevel='final', serial=0)

In [15]: sys.__stdin__
Out[15]: <_io.TextIOWrapper name='<stdin>' mode='r' encoding='cp1252'>

In [16]: sys.__stdout__
Out[16]: <_io.TextIOWrapper name='<stdout>' mode='w' encoding='cp1252'>

In [17]: sys.__stderr__
Out[17]: <_io.TextIOWrapper name='<stderr>' mode='w' encoding='cp1252'>
```

Thank
You