

## PROGRAMMING IN PYTHON

Gavrilut Dragos Course 7

Any Python code (python script) can be used as a module.

Python 3.x File: MyModule.py	Python 3.x File: test.py	
<pre>def Sum(x, y):</pre>	import MyModule	Output
return x+y	<pre>print (MyModule.Sum(10,20))</pre>	30

Both files test.py and MyModule.py are located in the same folder.

After the execution of test.py a folder with the name  $\_\_pycache\_\_$  that contains a file called MyModule.cpython-37.pyc will appear in the same folder (for Python 3.7)  $\rightarrow$  the version will be different for different versions of Python 3 (pyc = python compiled files)

Any Python code (python script) can be used as a module.

Python 3.x File: MyModule.py	Python 3.x File: test.py
<pre>def Sum(x,y):     return x+y</pre>	import MyModule
<pre>print ("MyModule loaded")</pre>	<pre>print (MyModule.Sum(10,20)) import MyModule</pre>

Loading a module will automatically execute any code (main code) that resides in that module.

The main code of a module (code that is written directly and not within a function or a class) will only me executed once (the first time a module is loaded).

#### Output

MyModule loaded 30

Any Python code (python script) can be used as a module.

Python 3.x File: MyModule.py	Python 3.x File: test.py
<pre>def Sum(x,y):     return x+y</pre>	<pre>import MyModule</pre>
<pre>print ("MyModule loaded")</pre>	<pre>print (MyModule.Sum(10,20)) import MyModule</pre>

What if MyModule is not located in the same folder as test.py file?

```
Output

Traceback (most recent call last):
File "test.py", line 1, in <module>
import sys,MyModule
ImportError: No module named 'MyModule'
```

Any Python code (python script) can be used as a module.

Python 3.x File: MyModule.py	Python 3.x File: test.py
<pre>def Sum(x,y):     return x+y</pre>	import sys
print ("MyModule loaded")	sys.path += [" <folder>"]</folder>
	import MyModule
	<pre>print (MyModule.Sum(10,20)) import MyModule</pre>

In the above piece of code "<folder>" represents a path to the folder where the file MyModule.py resides.

Output

MyModule loaded
30

Any Python code (python script) can be used as a module.

```
Python 3.x
File: MyModule.py

def Sum(x,y):
    return x+y
print ("MyModule loaded")

Python 3.x
File: test.py

import MyModule
print (dir (MyModule))
```

```
Output
['Sum', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__']
```

Any Python code (python script) can be used as a module.

Python 3.x File: MyModule.py	Python 3.x File: test.py
<pre>def Sum(x,y):     return x+y</pre>	import MyModule
<pre>print ("MyModule loaded")</pre>	<pre>print (MyModulefile) print (MyModulename) print (MyModulepackage)</pre>

#### Attributes:

- o \_\_file\_\_ → full path of the file that corresponds to the module (it could be a pyc file as well)
- o \_\_name\_\_ → name of the module (in this example : MyModule)
- \_\_package\_\_ → name of the package

Any Python code (python script) can be used as a module.

Python 3.x File: MyModule.py	Python 3.x File: test.py
<pre>def Sum(x,y):</pre>	<pre>import MyModule as m</pre>
return x+y	
doc = "Computes the sum of two numbers"	print (mdoc)

#### Output

Computes the sum of two numbers

#### Alternatively, the keyword "help" can be used as well

Python 3.x File: MyModule.py	Python 3.x File: test.py
<pre>def Sum(x,y):     return x+y</pre>	<pre>import MyModule as m</pre>
doc = "Computes the sum of two numbers"	help (m)

# Output Help on module MyModule: NAME MyModule - Computes the sum of two numbers FUNCTIONS Sum(x, y) FILE ...\facultate\python\_modules\mymodule.py

#### Alternatively, the keyword "help" can be used as well

Python 3.x File: MyModule.py	Python 3.x File: test.py
<pre>def Sum(x,y):     """returns the sum of x plus y"""</pre>	<pre>import MyModule as m</pre>
<pre>return x+y doc = "Computes the sum of two numbers"</pre>	help (m)

#### Output

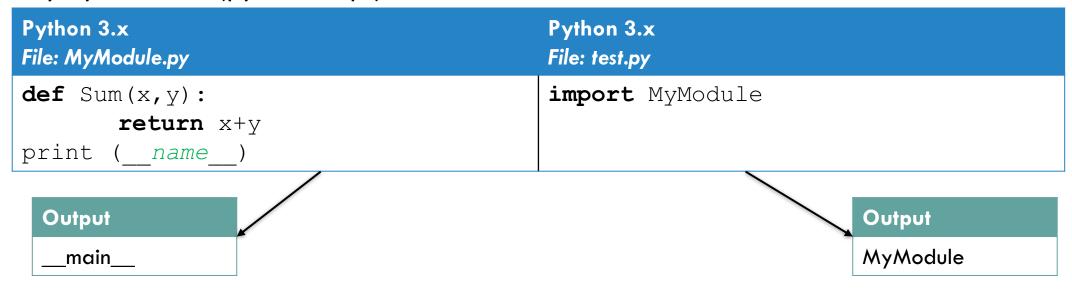
```
Help on module MyModule:

NAME
    MyModule - Computes the sum of two numbers

FUNCTIONS
    Sum(x, y)
    returns the sum of x plus y

FILE
    e:\documente\facultate\python\2020-2021\mymodule.py
```

Any Python code (python script) can be used as a module.



If a python script is executed directly, the value of <u>\_\_name\_\_</u> parameter will be <u>\_\_main\_\_</u>.

If it is executed using import, the value of <u>\_\_name\_\_</u> parameter will be the name of the module.

Any Python code (python script) can be used as a module.

```
Python 3.x
File: MyModule.py

def Sum(x,y):
    return x+y
if __name__ == "__main__":
    print("Main code")
    print("Testing sum(10,20) = ",Sum(10,20))
else:
    print("Module loaded")
```

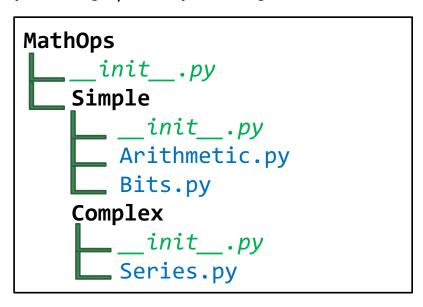
#### Output

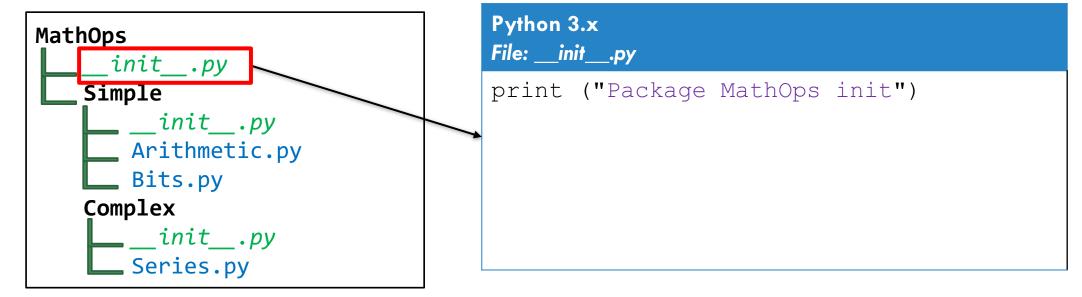
Main code

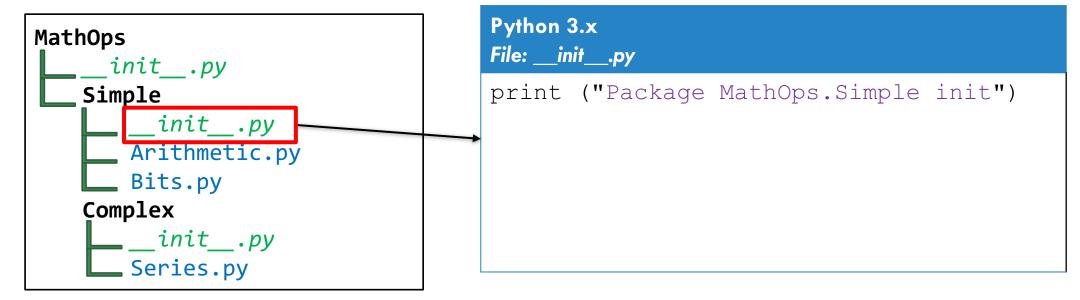
Testing sum(10,20) = 30

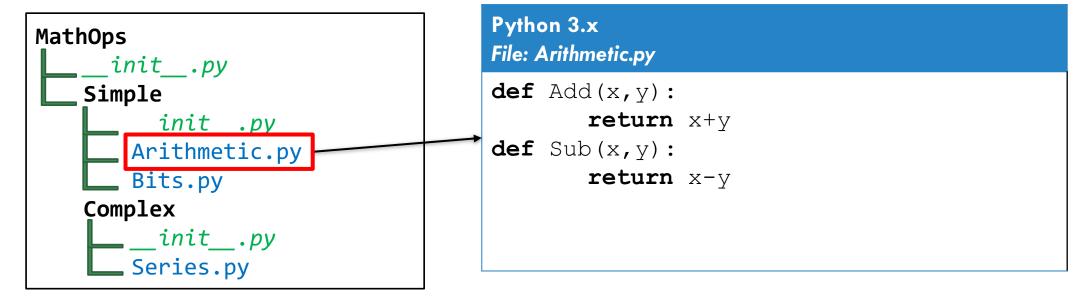
Output

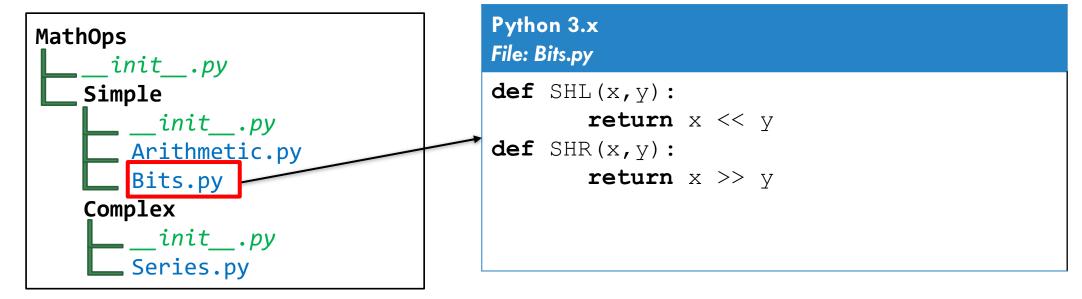
Module loaded

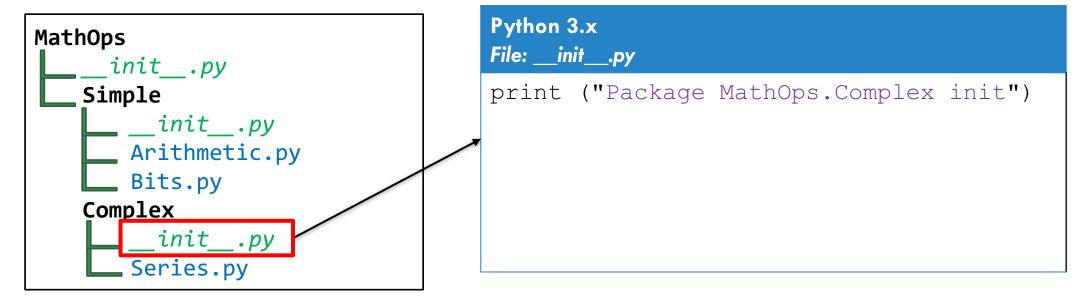


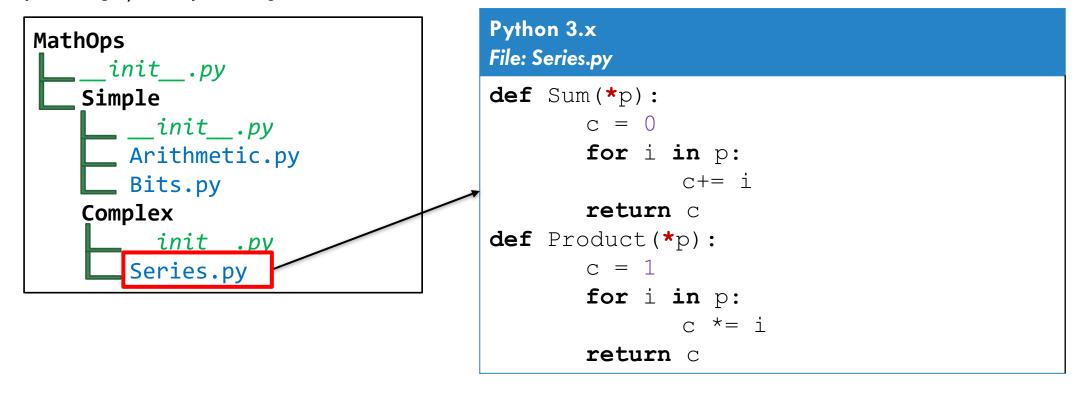












#### Usage:

```
import MathOps.Simple.Arithmetic

print (MathOps.Simple.Arithmetic.Add(2,3))

from MathOps.Simple import Arithmetic as a

print (a.Add(2,3))
```

#### Output

```
Package MathOps init
Package MathOps.Simple init
5
```

#### Usage:

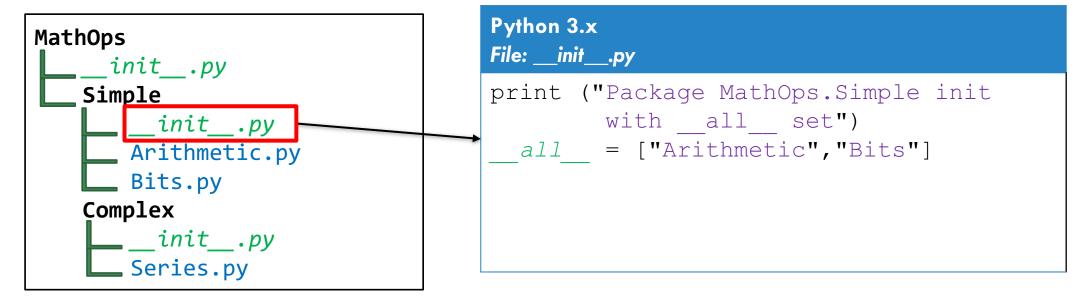
```
Python 3.x
from MathOps.Simple import *

print (Arithmetic.Add(2,3))
print (Bits.SHL(2,3))
```

#### Output

Package MathOps init
Package MathOps.Simple init
Traceback (most recent call last):
File "test.py", line 3, in <module>
print (Arithmetic.Add(2,3))
NameError: name 'Arithmetic' is not defined

To be able to use a syntax similar to "<u>from <module> import \*</u>" a module variable "<u>all</u>" must be defined. That variable will hold a list of all modules that belongs to that package.



#### Usage:

```
Python 3.x
from MathOps.Simple import *

print (Arithmetic.Add(2,3))
print (Bits.SHL(2,3))
```

#### Output

```
Package MathOps init
Package MathOps.Simple init with __all__ set
5
16
```

# MODULES/PACKAGES

If you want a module and/or package to be available to all the scripts that are executed on that system just copy the module or the entire package folder on the Python search path and you will be able to access it directly. These paths are:

- Windows: <PythonFolder>\Lib
   Exemple: C:\Python27\Lib or C:\Python37\Lib
- Linux: /usr/lib/<PythonVersion>
   Example: /usr/lib/python2.7 or /usr/lib/python3.7)

# MODULES/PACKAGES

Python also has a special library (importlib) that can be used to dynamically import a module.

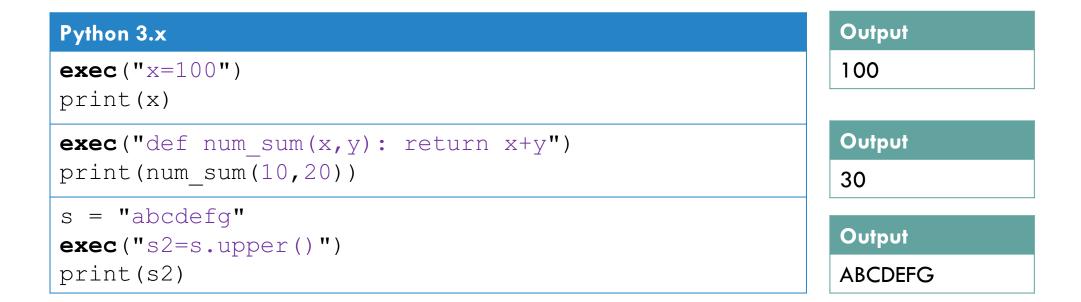
- o *importlib.*import\_module (moduleName,package=None) → to import a module
- o importlib.reload (module) 🗩 to reload a module that was already loaded

Python 3.x File: C:\Python3\Lib\MyModule.py	Python 3.x File: test.py
<pre>def Sum(x,y):     return x+y</pre>	<pre>import importlib</pre>
<pre>print ("MyModule loaded")</pre>	<pre>m = importlib.import_module("MyModule") print (m.Sum(10,20))</pre>

## MyModule loaded 30

Python has a keyword (exec) that can be used to dynamically compile and execute python code.

The format is exec (code, [global],[local]) where [global] and [local] represents a list of global and local definition that should be used when executing the code.



Because of this keyword, python code can obfuscate or modify itself during runtime.

Because of this keyword, python code can obfuscate or modify itself during runtime.

```
Python 3.x

data = [0x65, 0x66, 0x67, 0x21, 0x54, 0x76, 0x6E, 0x62, 0x29, 0x79, 0x2D, 0x7A, 0x2D, 0x7B, 0x2A, 0x3B, 0x0E, 0x0B, 0x0A, 0x73, 0x66, 0x75, 0x76, 0x73, 0x6F, 0x21, 0x79, 0x2C, 0x7A, 0x2C, 0x7B]

s = ""

for i in data:

    def Suma(x,y,z):
        return x+y+z

exec(s)

print (Suma(1,2,3))
```

print(s(10,20))

Multiple layers of encryption are also possible:

Multiple layers of encryption are also possible:

```
Python 3.x
buf =
[0x74,0x21,0x3E,0x21,0x23,0x67,0x68,0x69,0x5D,0x23,0x76,0x2B,0x64,0x2F,0x65,
0x2C, 0x3D, 0x5D, 0x23, 0x75, 0x68, 0x77, 0x78, 0x75, 0x71, 0x5D, 0x23, 0x64, 0x2E, 0x65, 0
x23,0x0E,0x0B,0x74,0x33,0x21,0x3E,0x21,0x23,0x23,0x0E,0x0B,0x67,0x70,0x73,0x
                                     0 \times 2D = 0 \times QE, 0x0B, 0x0A, 0x74, 0x33, 0x21, 0x2C, 0x3
21,0x64,0x
          s = "fgh\"u*c.d+<\"tgvwtp\"c-d"
                                               0x29,0x64,0x2A,0x2E,0x33,0x2A,0x0E
E, 0x21, 0x
,0x0B,0x0
                                               x2A]
           for c in s:
for i in
             s2 += chr(ord(c)-2)
           exec(s2)
                                                                      Output
exec(s)
                                                                      30
print(s(10,20))
```

Multiple layers of encryption are also possible:

```
Python 3.x
buf =
 [0x74,0x21,0x3E,0x21,0x23,0x67,0x68,0x69,0x5D,0x23,0x76,0x2B,0x64,0x2F,0x65,
0x2C, 0x3D, 0x5D, 0x23, 0x75, 0x68, 0x77, 0x78, 0x75, 0x71, 0x5D, 0x23, 0x64, 0x2E, 0x65, 0
x23,0x0E,0x0B,0x74,0x33,0x21,0x3E,0x21,0x23,0x23,0x0E,0x0B,0x67,0x70,0x73,0x
                                                                                                                       0.74 0.20 0.00E, 0.00B, 0.00A, 0.00
21,0x64,0x21
E, 0 \times 21, 0 \times s = \text{"fgh}\u^*c.d+<\\text{"tgvwtp}\c-d"
                                                                                                                                                                                  0x29,0x64,0x2A,0x2E,0x33,0x2A,0x0E
 1x2A1
                                          for c in s:
for i in/
                                                        s2 += chr(ord(c)-2)
                                         exec s2
                                                                                                                                                                                                                                                                        Output
exed(s)
                                                                                  def s(a,b): return a+b
                                                                                                                                                                                                                                                                       30
print(s(10,20))
```