Python programming for beginners

Stefan Zhauryd
Instructor

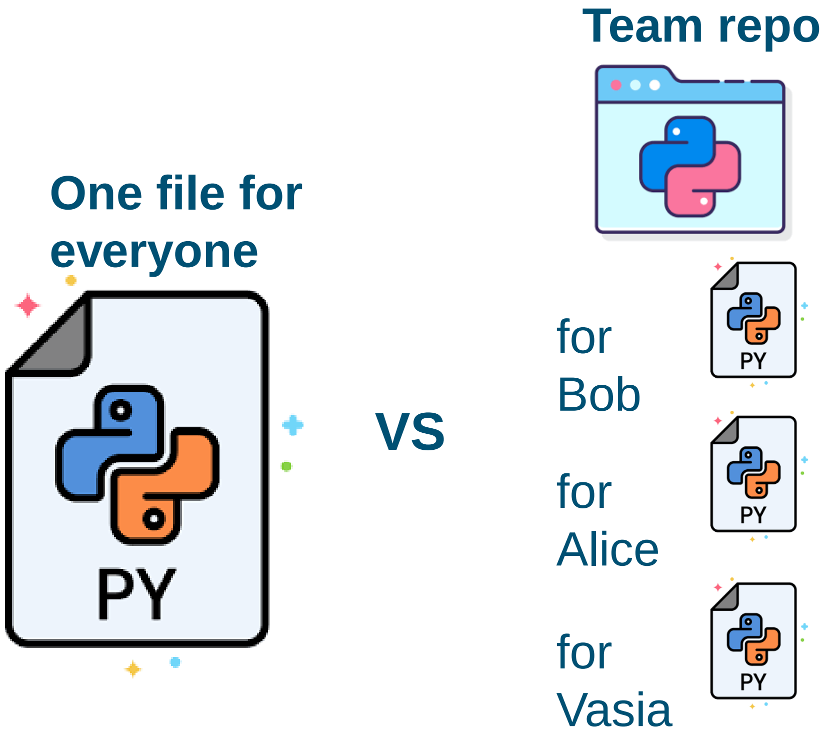
Module 5

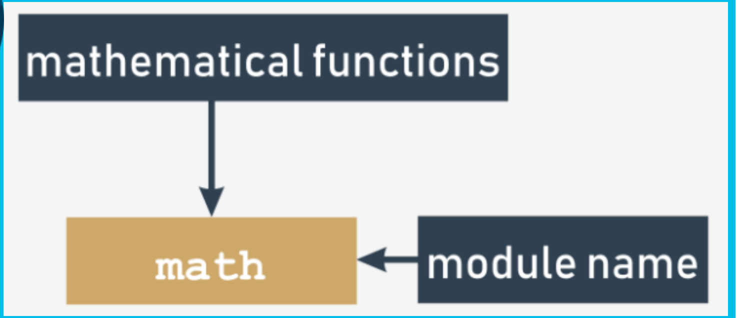
Modules, Packages and PIP

In this module, you will learn about:

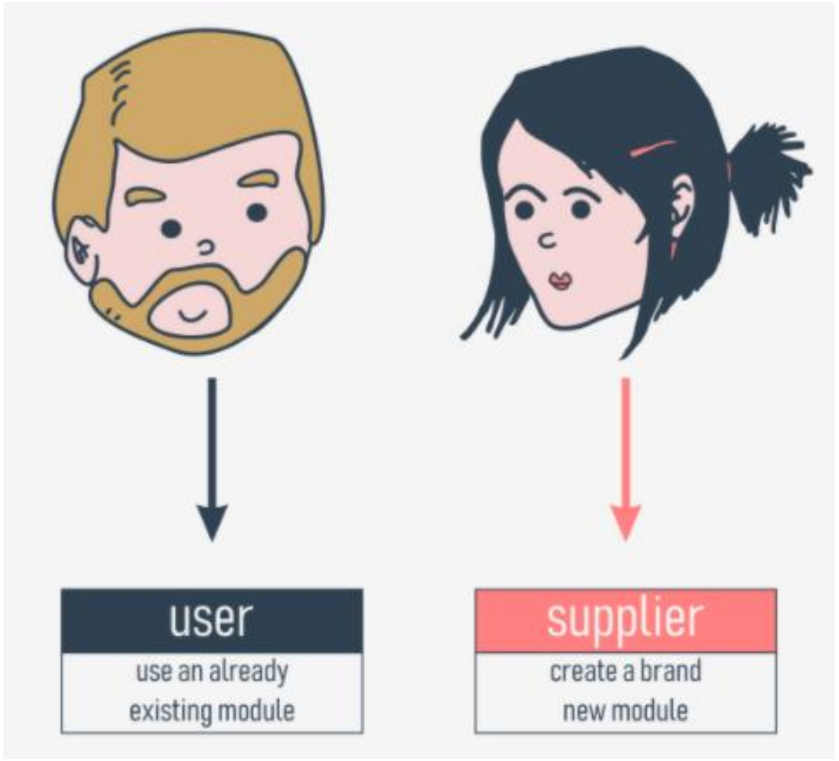
- **importing and using Python modules;**
- **using some of the most useful Python standard library modules;**
- **constructing and using Python packages;**
- PIP (Python Installation Package) and how to use it to install and uninstall ready-to-use packages from PyPI.

What is a module?





How to make use of a module?
import time
math



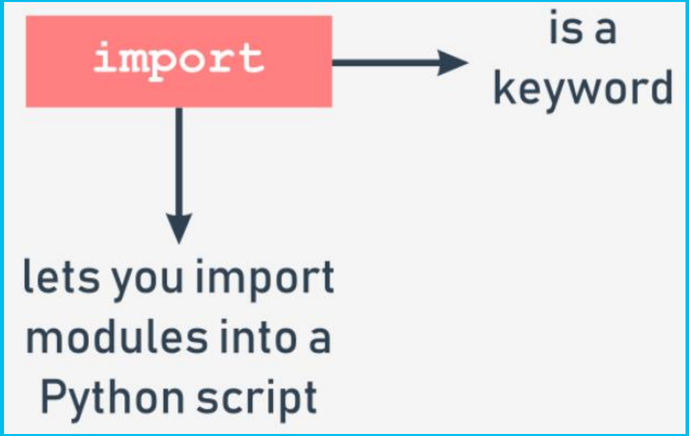
Tap to link:

https://docs.python.org/3/tutorial/modules.html

https://docs.python.org/3/library/index.html

Importing a module

print(math.pi)
pi = 3.14
print(pi)



```
import math
```

```
import math
import sys
```

```
import math, sys
```

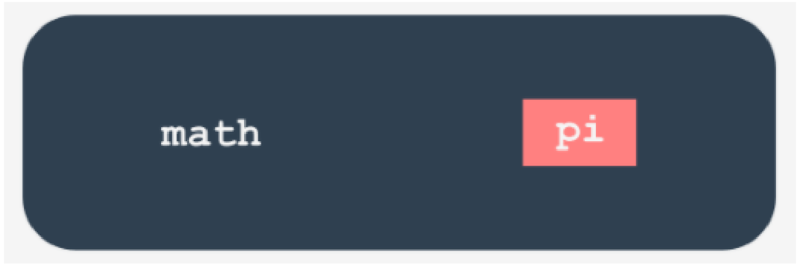
https://digitology.tech/posts/prostranstva-imion-v-python/

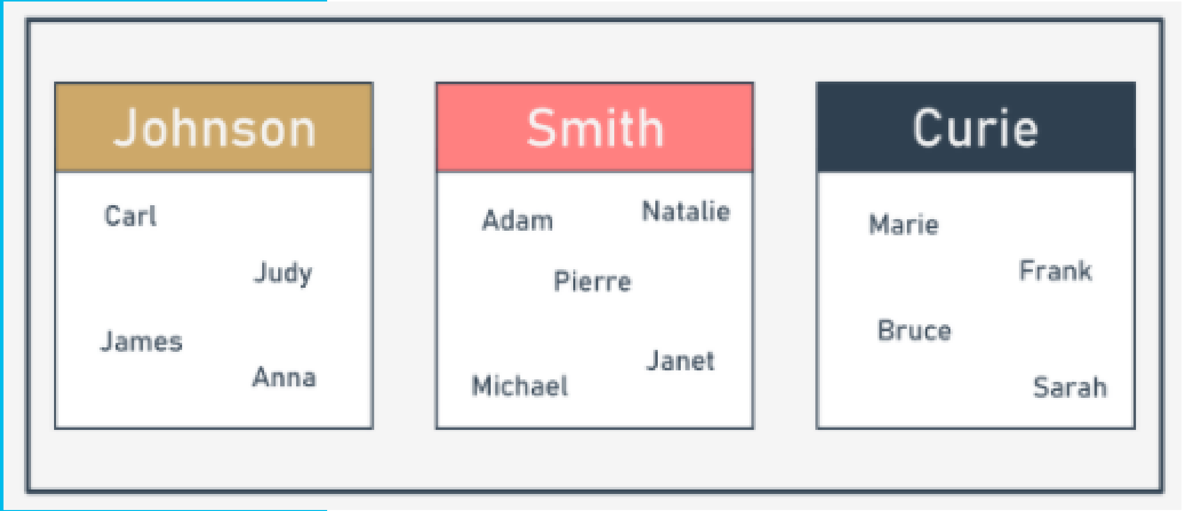
https://pyneng.readthedocs.io/ru/latest/book/09_functions/namespace.html

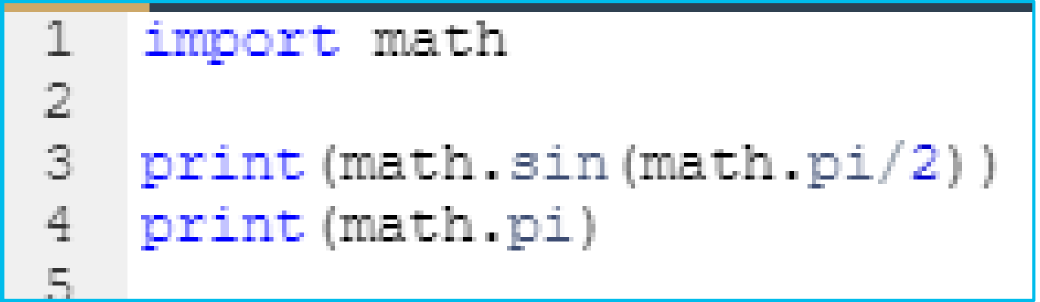
Importing a module: continued

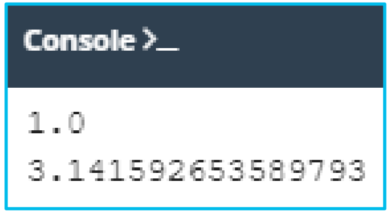
import Smith, math

Smith.Adam
Curie.Adam
math.pi

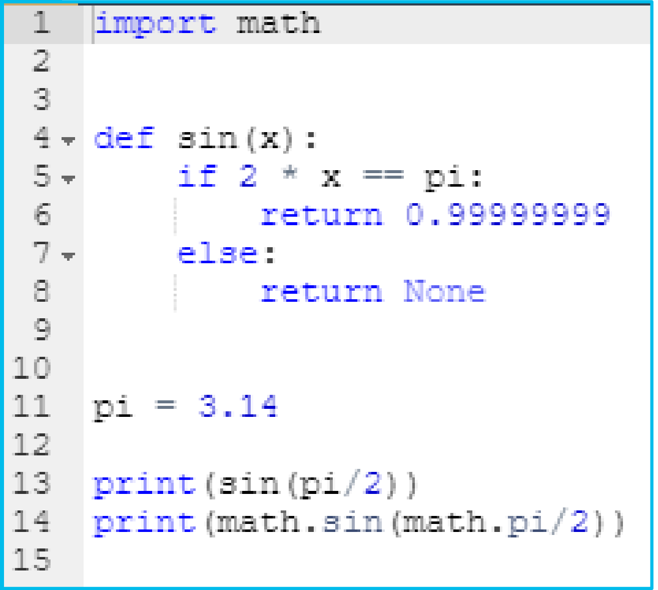


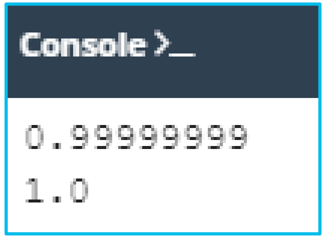



```python
import math

print(math.sin(math.pi/2))
print(math.pi)
```

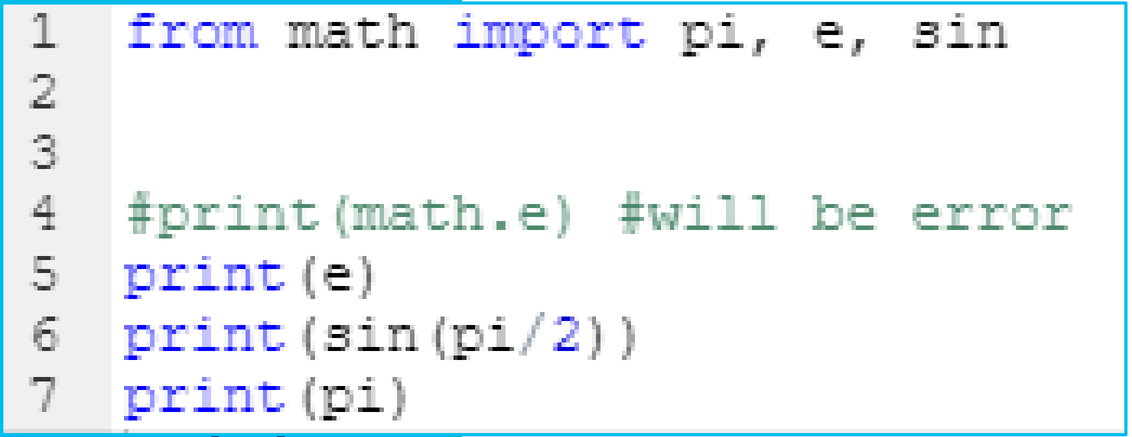

```
Console >_
1.0
3.141592653589793
```

Importing a module: continued

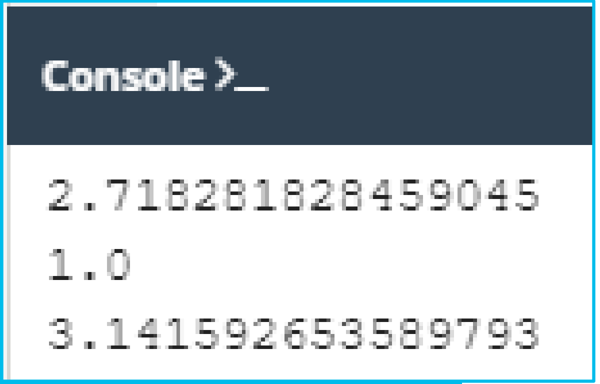

```python
import math


def sin(x):
    if 2 * x == pi:
        return 0.99999999
    else:
        return None


pi = 3.14

print(sin(pi/2))
print(math.sin(math.pi/2))
```


```
Console >_
0.99999999
1.0
```

Importing a module: continued

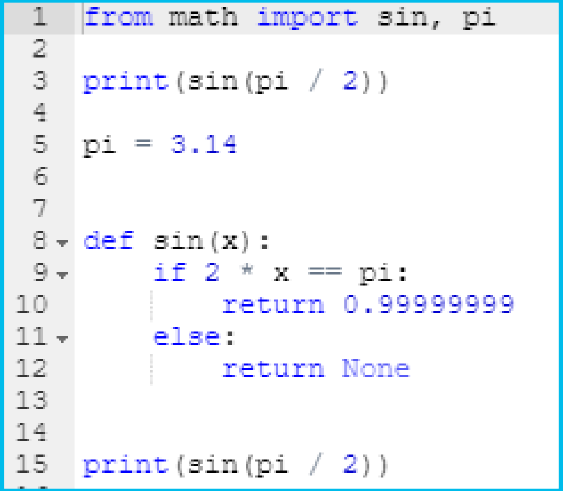


```python
from math import pi, e, sin


#print(math.e) #will be error
print(e)
print(sin(pi/2))
print(pi)
```

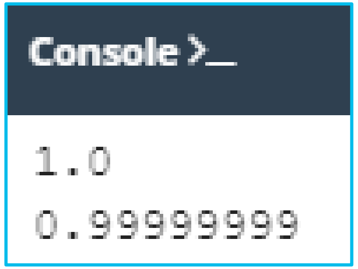


```
2.718281828459045
1.0
3.141592653589793
```

Importing a module: continued

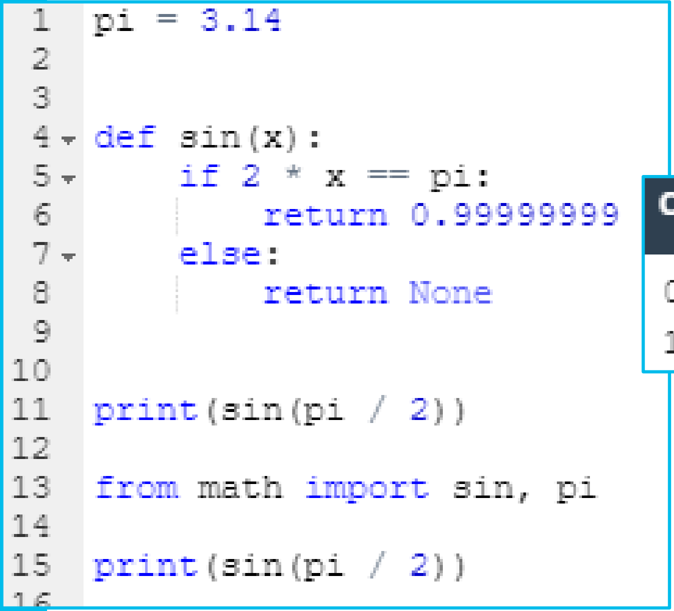


```python
from math import sin, pi

print(sin(pi / 2))

pi = 3.14


def sin(x):
    if 2 * x == pi:
        return 0.99999999
    else:
        return None


print(sin(pi / 2))
```

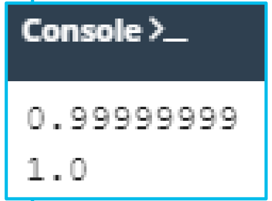


Console >_

```
1.0
0.99999999
```



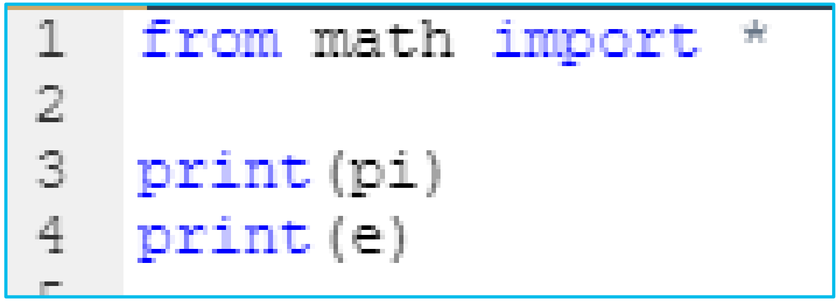
```python
pi = 3.14


def sin(x):
    if 2 * x == pi:
        return 0.99999999
    else:
        return None


print(sin(pi / 2))

from math import sin, pi

print(sin(pi / 2))
```



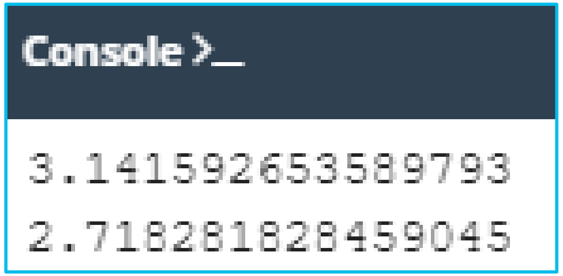
Console >_

```
0.99999999
1.0
```

Importing a module: *



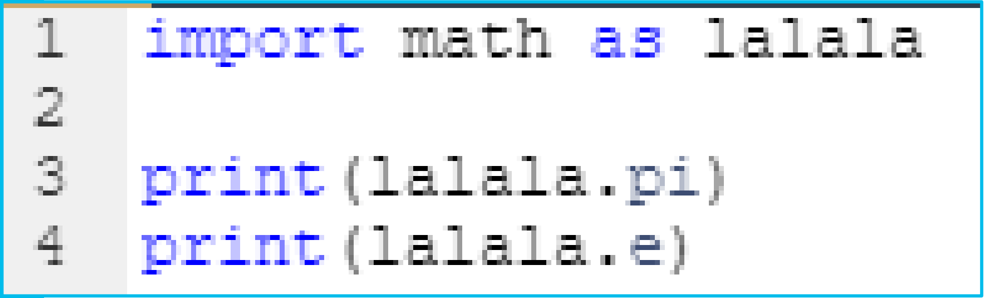
```python
from math import *

print(pi)
print(e)
```



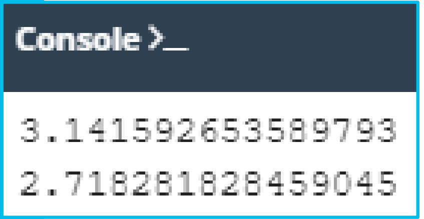
```
3.141592653589793
2.718281828459045
```

Importing a module: the **as** keyword

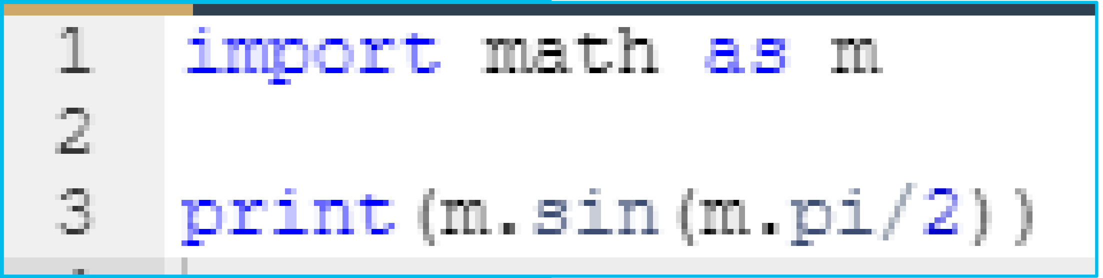
Note: **as** is a keyword.



```python
import math as lalala

print(lalala.pi)
print(lalala.e)
```

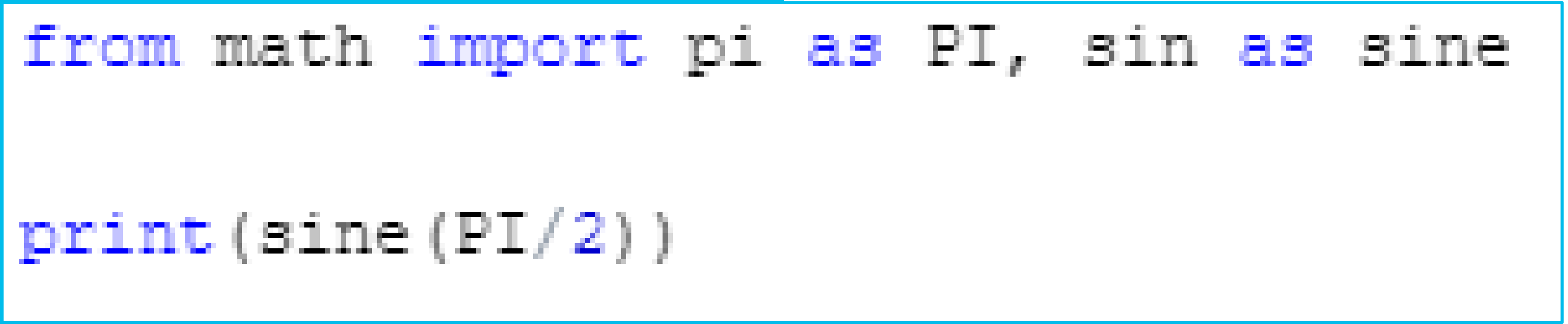


```
3.141592653589793
2.718281828459045
```



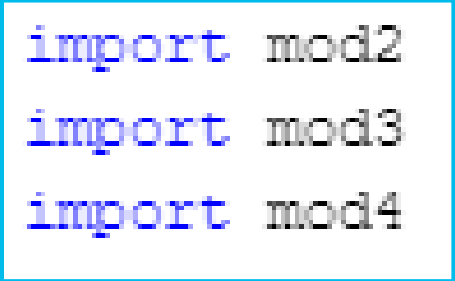
```
import math as m

print(m.sin(m.pi/2))
```

Importing a module: continued

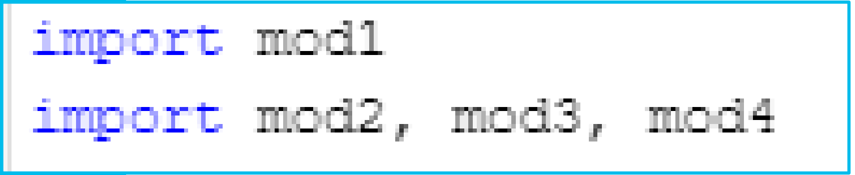


```
from math import pi as PI, sin as sine


print(sine(PI/2))
```

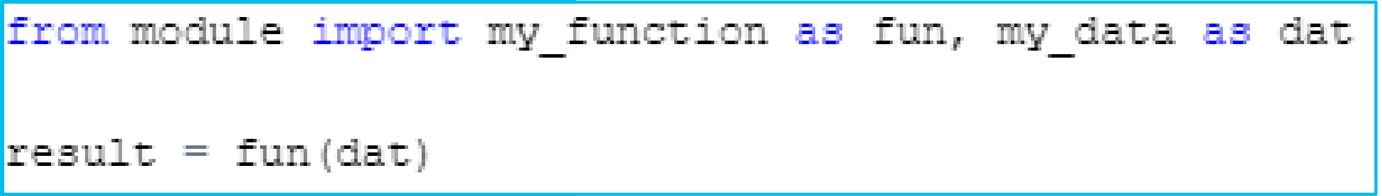
Key takeaways



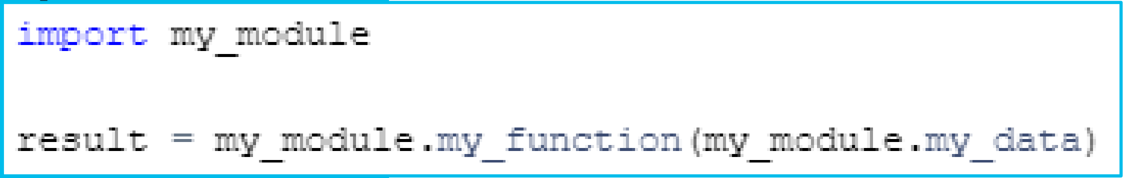
```python
import mod2
import mod3
import mod4
```



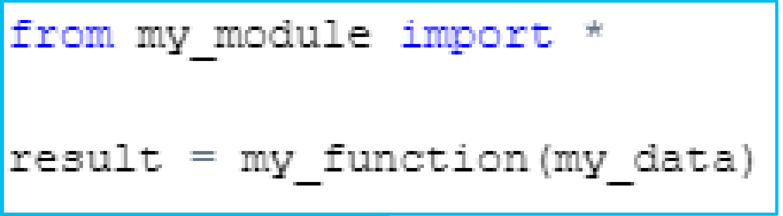
```python
import mod1
import mod2, mod3, mod4
```



```python
from module import my_function as fun, my_data as dat

result = fun(dat)
```



```python
import my_module

result = my_module.my_function(my_module.my_data)
```



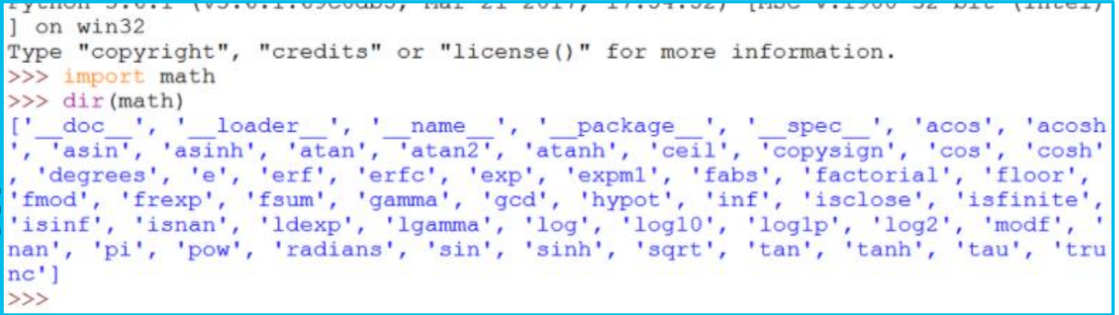
```python
from my_module import *

result = my_function(my_data)
```

```
dir(module)
```

The function returns an alphabetically sorted list containing all entities' names available in the module identified by a name passed to the function as an argument:

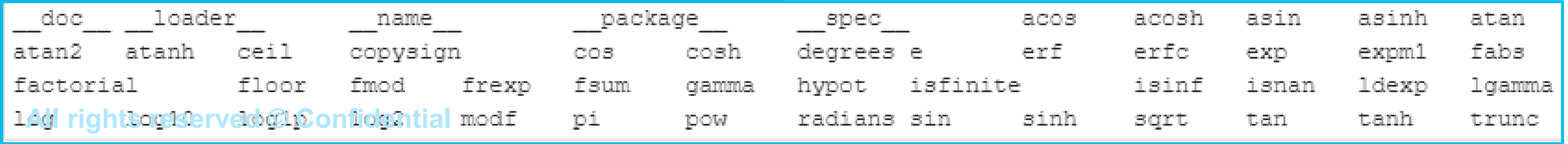
```
import math
dir(math)
```

Working with standard modules



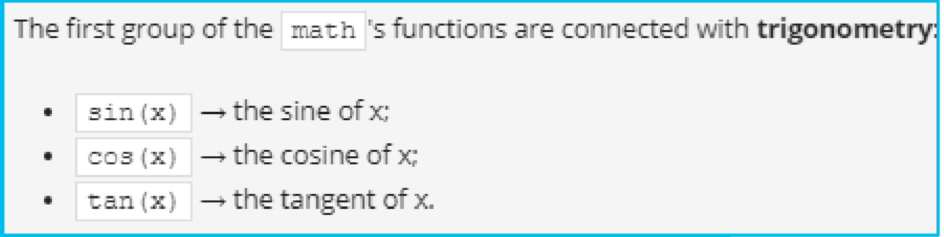
```
] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh
', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh'
, 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor',
'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite',
'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', '
nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'tru
nc']
>>>
```

```
import math


for name in dir(math):
    print(name, end="\t")
```

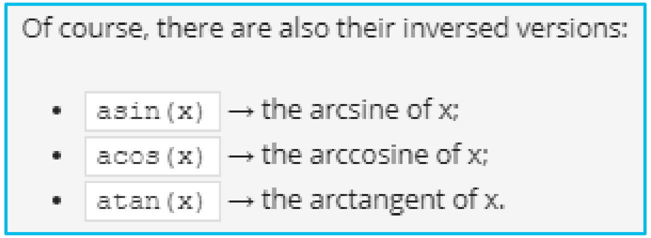


```
__doc__  __loader__       __name__         __package__      __spec__         acos     acosh    asin     asinh    atan
atan2    atanh    ceil    copysign         cos      cosh    degrees e        erf      erfc     exp      expm1    fabs
factorial         floor   fmod     frexp   fsum     gamma   hypot   isfinite          isinf    isnan    ldexp    lgamma
log      log10    log1p   log2     modf    pi       pow     radians sin      sinh     sqrt     tan      tanh     trunc
```

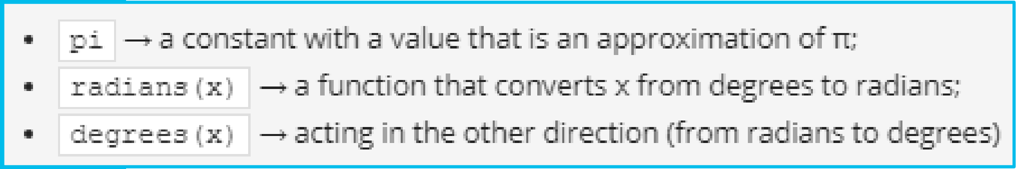
Selected functions from the math module

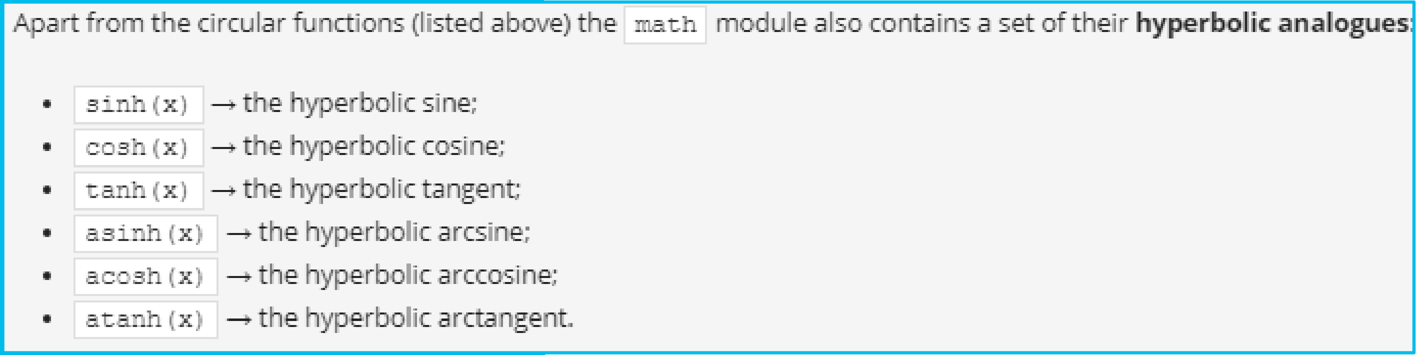

The first group of the `math`'s functions are connected with **trigonometry**:

- `sin(x)` → the sine of x;
- `cos(x)` → the cosine of x;
- `tan(x)` → the tangent of x.


Of course, there are also their inversed versions:

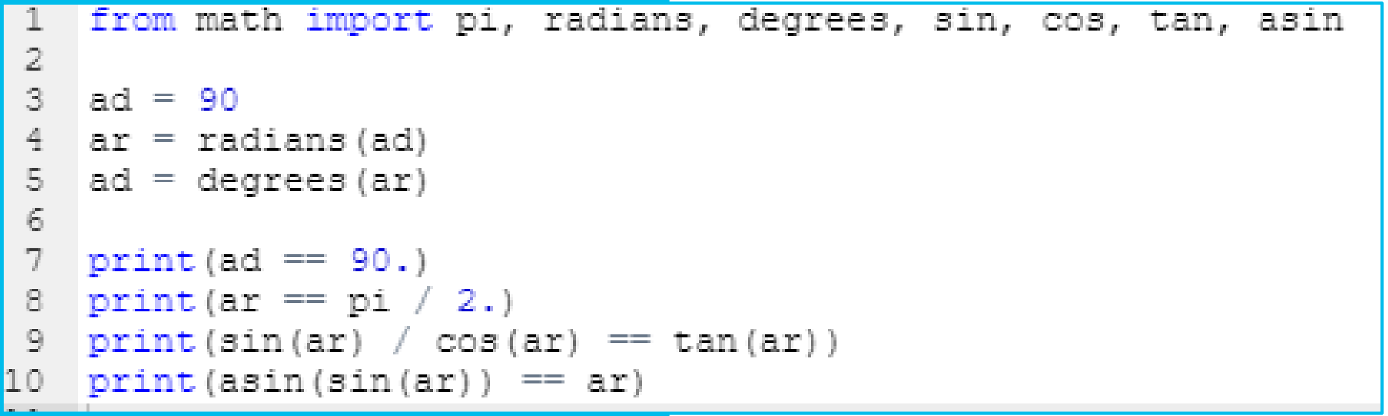
- `asin(x)` → the arcsine of x;
- `acos(x)` → the arccosine of x;
- `atan(x)` → the arctangent of x.


- `pi` → a constant with a value that is an approximation of π;
- `radians(x)` → a function that converts x from degrees to radians;
- `degrees(x)` → acting in the other direction (from radians to degrees)


Apart from the circular functions (listed above) the `math` module also contains a set of their **hyperbolic analogues**:

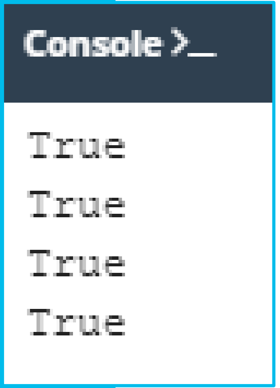
- `sinh(x)` → the hyperbolic sine;
- `cosh(x)` → the hyperbolic cosine;
- `tanh(x)` → the hyperbolic tangent;
- `asinh(x)` → the hyperbolic arcsine;
- `acosh(x)` → the hyperbolic arccosine;
- `atanh(x)` → the hyperbolic arctangent.

https://www.w3schools.com/python/module_math.asp



```python
from math import pi, radians, degrees, sin, cos, tan, asin

ad = 90
ar = radians(ad)
ad = degrees(ar)

print(ad == 90.)
print(ar == pi / 2.)
print(sin(ar) / cos(ar) == tan(ar))
print(asin(sin(ar)) == ar)
```

Selected functions from the math module

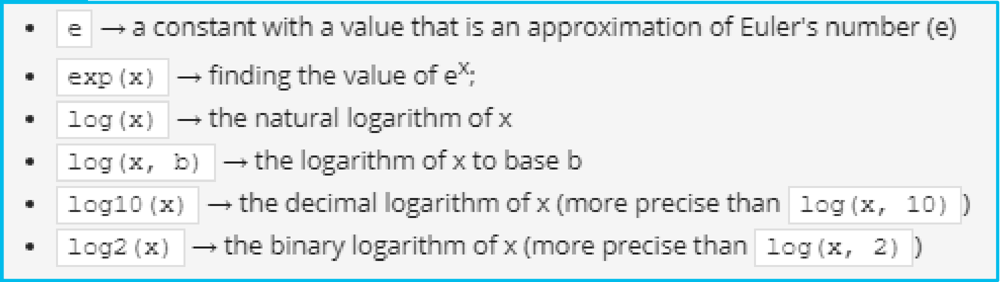


Console >_

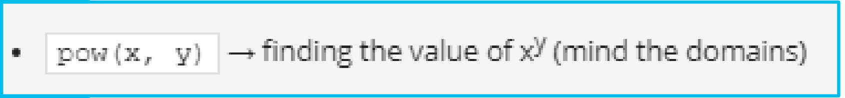
```
True
True
True
True
```

https://www.w3schools.com/python/module_math.asp

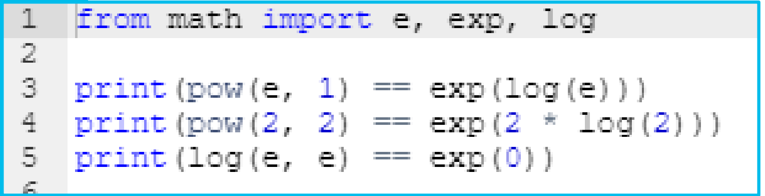
Selected functions from the math module



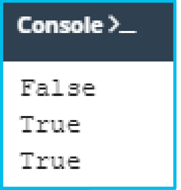
- `e` → a constant with a value that is an approximation of Euler's number (e)
- `exp(x)` → finding the value of e^x;
- `log(x)` → the natural logarithm of x
- `log(x, b)` → the logarithm of x to base b
- `log10(x)` → the decimal logarithm of x (more precise than `log(x, 10)`)
- `log2(x)` → the binary logarithm of x (more precise than `log(x, 2)`)



- `pow(x, y)` → finding the value of x^y (mind the domains)



```python
from math import e, exp, log

print(pow(e, 1) == exp(log(e)))
print(pow(2, 2) == exp(2 * log(2)))
print(log(e, e) == exp(0))
```

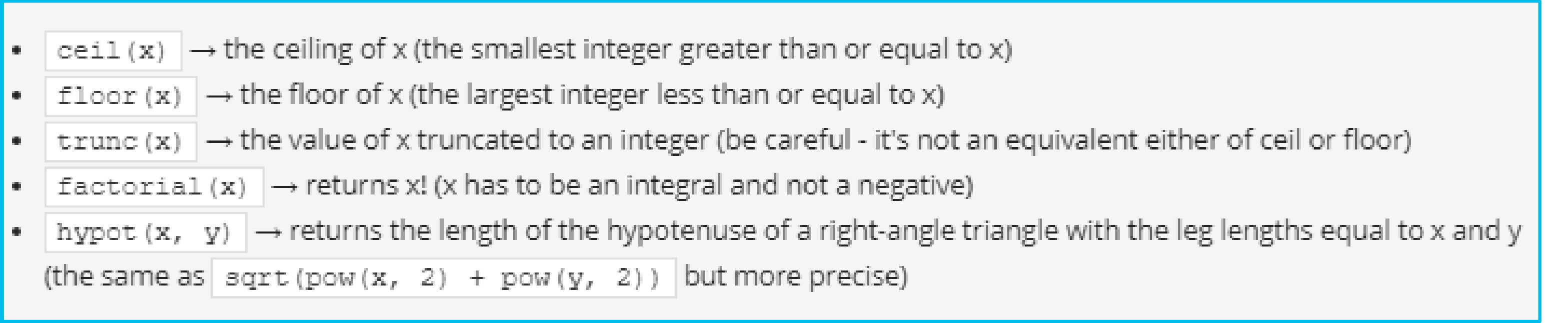


Console >_

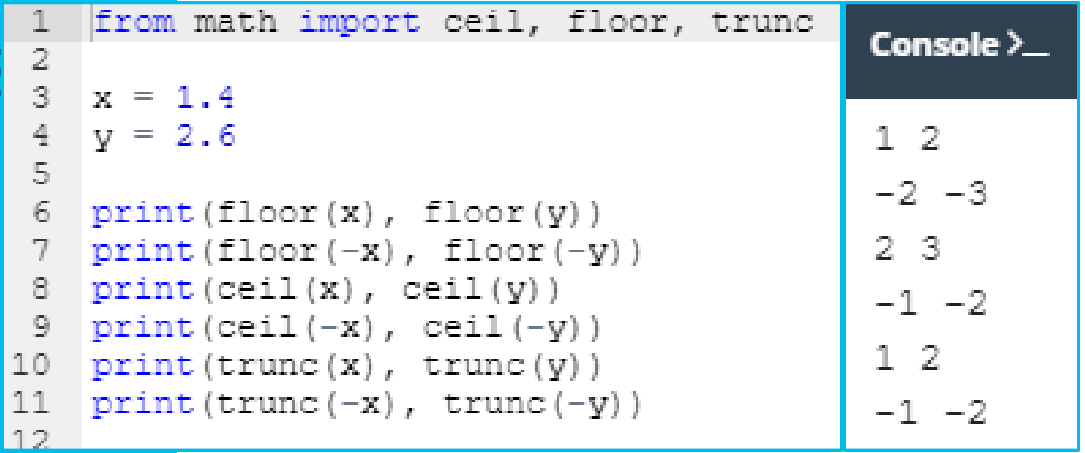
```
False
True
True
```

https://www.w3schools.com/python/module_math.asp

Selected functions from the math module



- `ceil(x)` → the ceiling of x (the smallest integer greater than or equal to x)
- `floor(x)` → the floor of x (the largest integer less than or equal to x)
- `trunc(x)` → the value of x truncated to an integer (be careful - it's not an equivalent either of ceil or floor)
- `factorial(x)` → returns x! (x has to be an integral and not a negative)
- `hypot(x, y)` → returns the length of the hypotenuse of a right-angle triangle with the leg lengths equal to x and y (the same as `sqrt(pow(x, 2) + pow(y, 2))` but more precise)



```python
from math import ceil, floor, trunc

x = 1.4
y = 2.6

print(floor(x), floor(y))
print(floor(-x), floor(-y))
print(ceil(x), ceil(y))
print(ceil(-x), ceil(-y))
print(trunc(x), trunc(y))
print(trunc(-x), trunc(-y))
```

Console >_

```
1 2
-2 -3
2 3
-1 -2
1 2
-1 -2
```

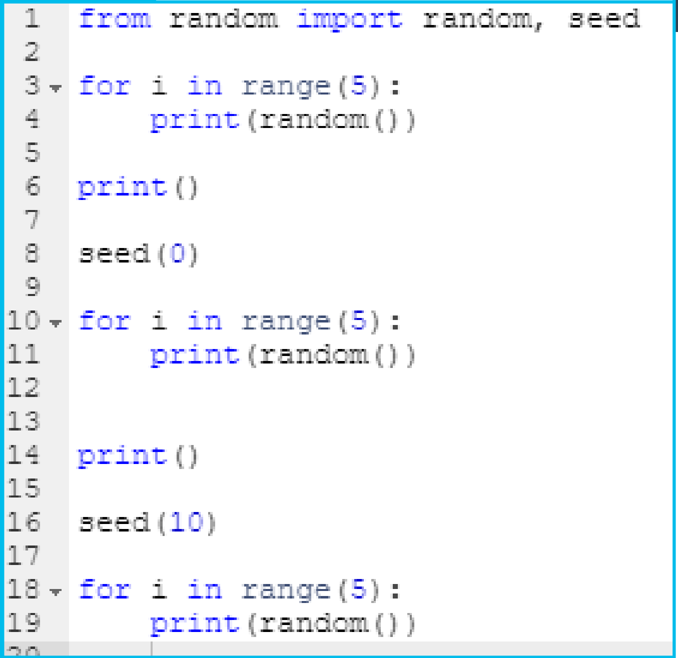
https://www.w3schools.com/python/module_math.asp

Is there real randomness in computers?

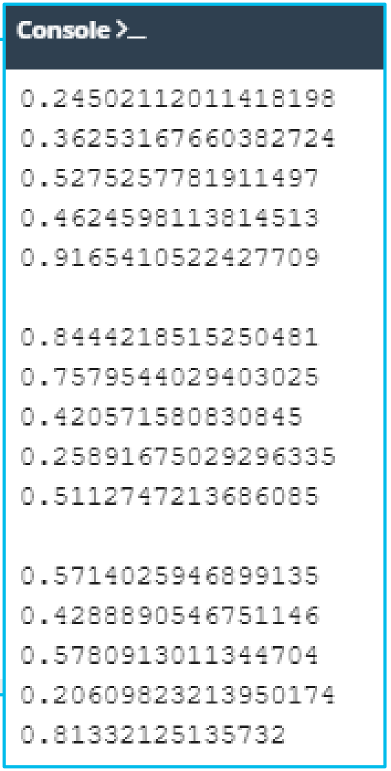
Selected functions from the random module

import random

random.random()

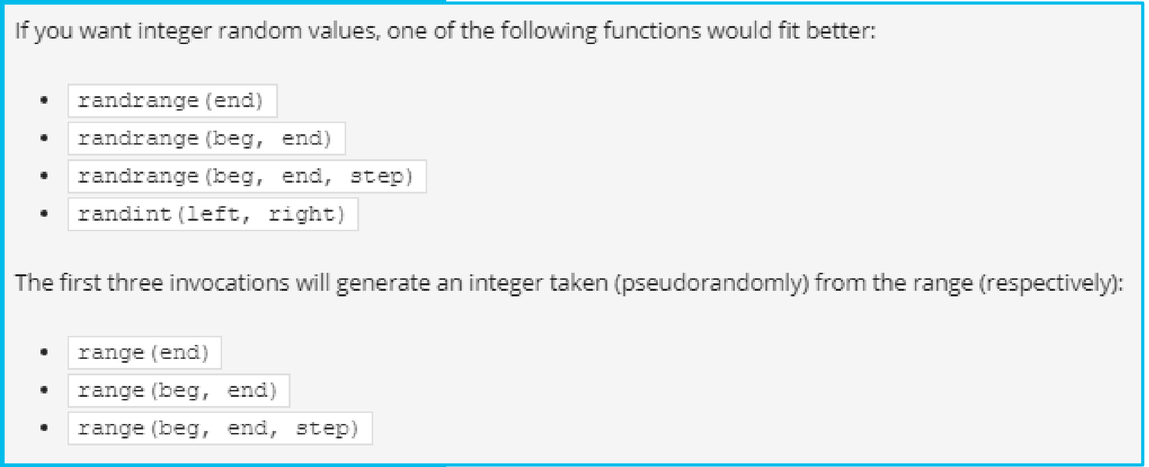


```
from random import random, seed

for i in range(5):
    print(random())

print()

seed(0)

for i in range(5):
    print(random())


print()

seed(10)

for i in range(5):
    print(random())
```



Console >_

```
0.24502112011418198
0.36253167660382724
0.5275257781911497
0.4624598113814513
0.9165410522427709

0.8444218515250481
0.7579544029403025
0.420571580830845
0.25891675029296335
0.511274721368085

0.5714025946899135
0.4288890546751146
0.5780913011344704
0.20609823213950174
0.81332125135732
```


If you want integer random values, one of the following functions would fit better:

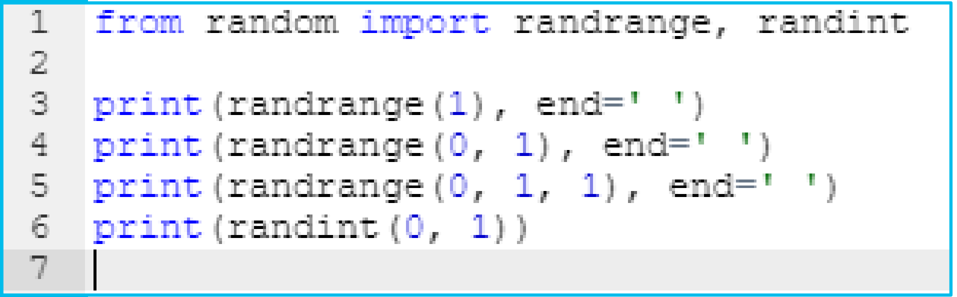
- `randrange(end)`
- `randrange(beg, end)`
- `randrange(beg, end, step)`
- `randint(left, right)`

The first three invocations will generate an integer taken (pseudorandomly) from the range (respectively):

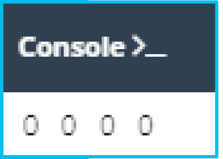
- `range(end)`
- `range(beg, end)`
- `range(beg, end, step)`

Selected functions from the random module: continued

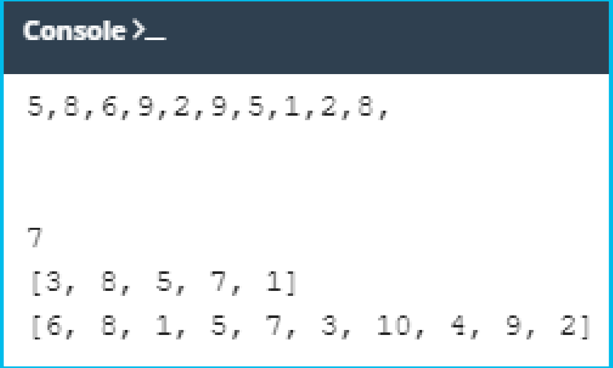
The randrange and randint functions



```python
from random import randrange, randint

print(randrange(1), end=' ')
print(randrange(0, 1), end=' ')
print(randrange(0, 1, 1), end=' ')
print(randint(0, 1))

```

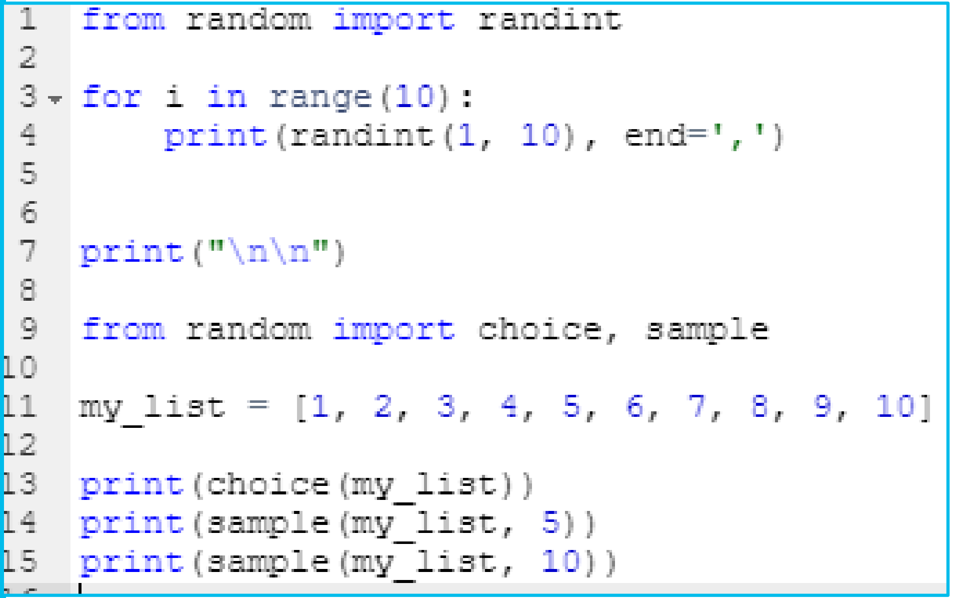


```
0 0 0 0
```



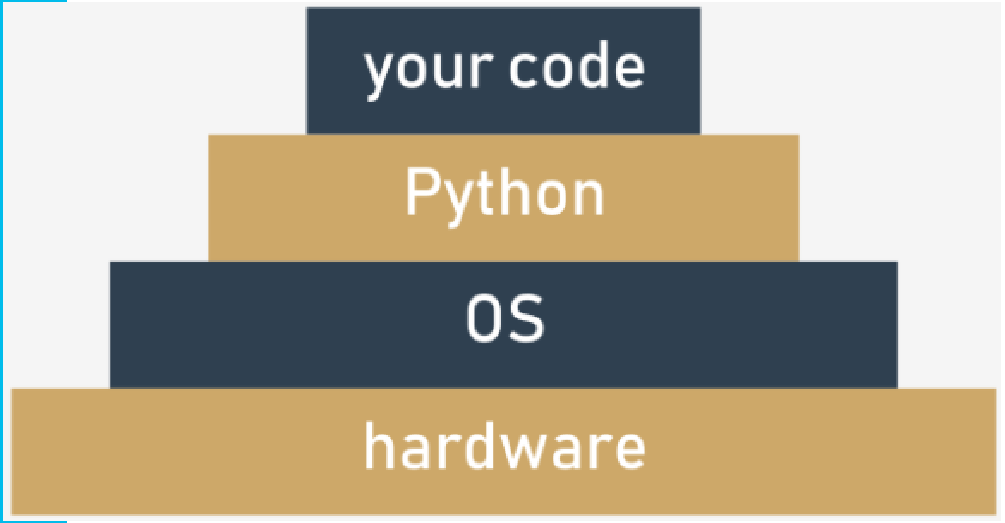
```
5,8,6,9,2,9,5,1,2,8,



7
[3, 8, 5, 7, 1]
[6, 8, 1, 5, 7, 3, 10, 4, 9, 2]
```

Selected functions from the random module: continued



```
from random import randint

for i in range(10):
    print(randint(1, 10), end=',')


print("\n\n")

from random import choice, sample

my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(choice(my_list))
print(sample(my_list, 5))
print(sample(my_list, 10))
```

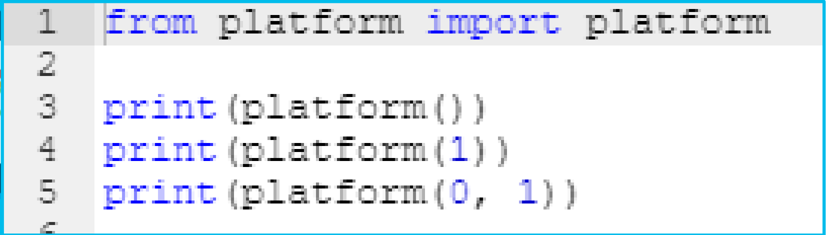
How to know where you are?



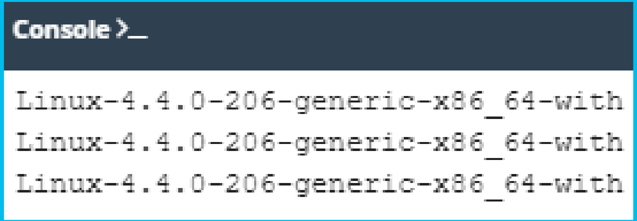
Selected functions from the random module: continued

The platform function

```
platform(aliased = False, terse = False)
```



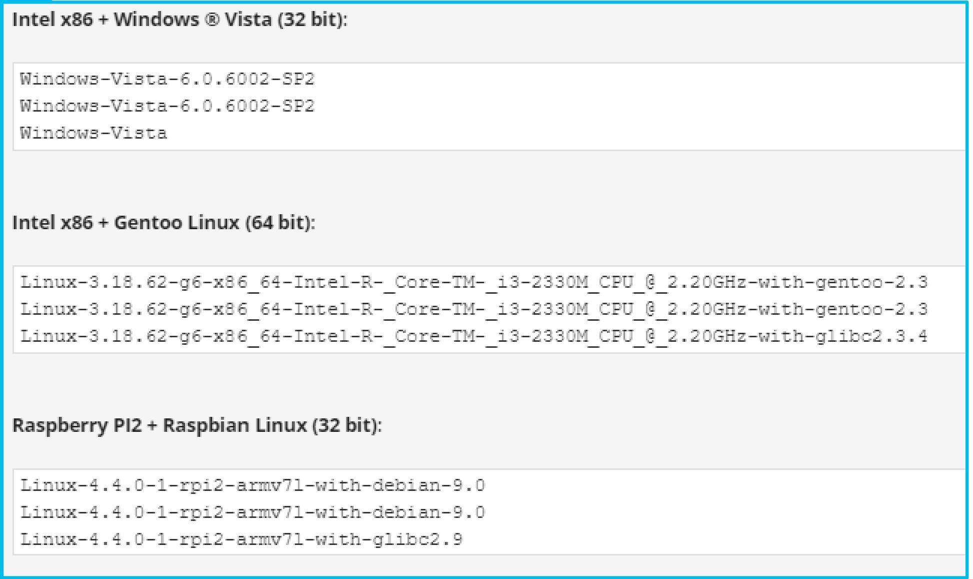
```
from platform import platform

print(platform())
print(platform(1))
print(platform(0, 1))
```



```
Linux-4.4.0-206-generic-x86_64-with
Linux-4.4.0-206-generic-x86_64-with
Linux-4.4.0-206-generic-x86_64-with
```

And now:

- aliased → when set to True (or any non-zero value) it may cause the function to present the alternative underlying layer names instead of the common ones;
- terse → when set to True (or any non-zero value) it may convince the function to present a briefer form of the result (if possible)



Intel x86 + Windows ® Vista (32 bit):

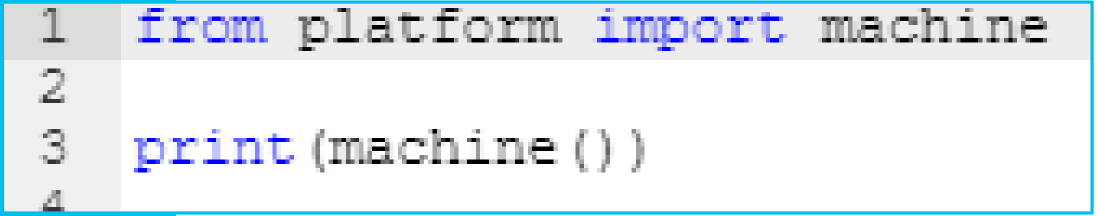
```
Windows-Vista-6.0.6002-SP2
Windows-Vista-6.0.6002-SP2
Windows-Vista
```

Intel x86 + Gentoo Linux (64 bit):

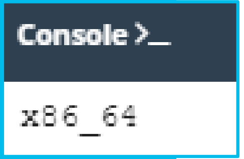
```
Linux-3.18.62-g6-x86_64-Intel-R-_Core-TM-_i3-2330M_CPU_@_2.20GHz-with-gentoo-2.3
Linux-3.18.62-g6-x86_64-Intel-R-_Core-TM-_i3-2330M_CPU_@_2.20GHz-with-gentoo-2.3
Linux-3.18.62-g6-x86_64-Intel-R-_Core-TM-_i3-2330M_CPU_@_2.20GHz-with-glibc2.3.4
```

Raspberry PI2 + Raspbian Linux (32 bit):

```
Linux-4.4.0-1-rpi2-armv7l-with-debian-9.0
Linux-4.4.0-1-rpi2-armv7l-with-debian-9.0
Linux-4.4.0-1-rpi2-armv7l-with-glibc2.9
```



```python
from platform import machine

print(machine())
```

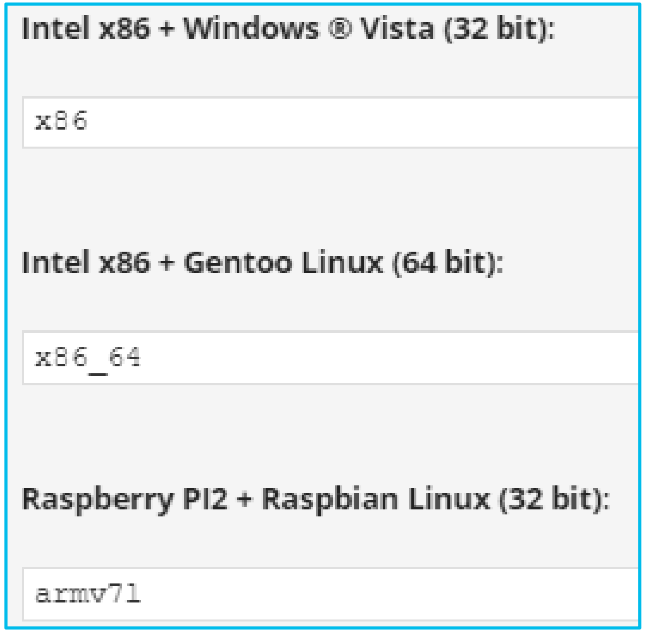


Console >_

```
x86_64
```

Selected functions from the random module: continued

The machine function



Intel x86 + Windows ® Vista (32 bit):

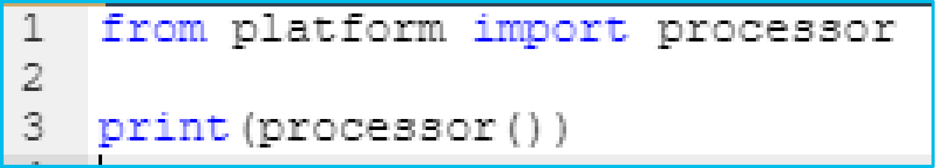
```
x86
```

Intel x86 + Gentoo Linux (64 bit):

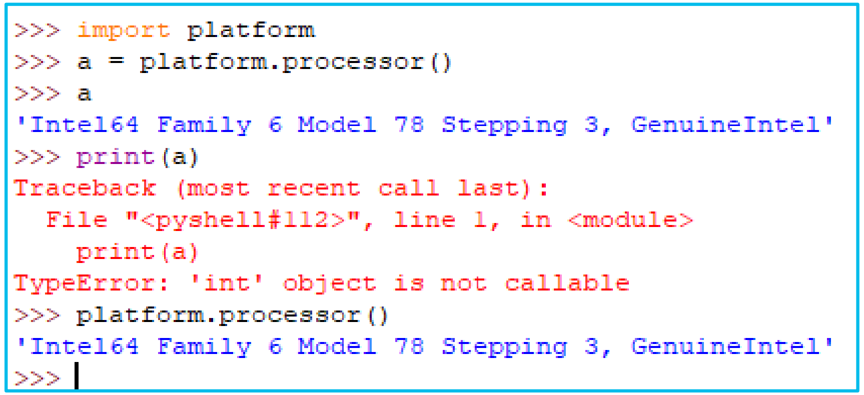
```
x86_64
```

Raspberry PI2 + Raspbian Linux (32 bit):

```
armv7l
```



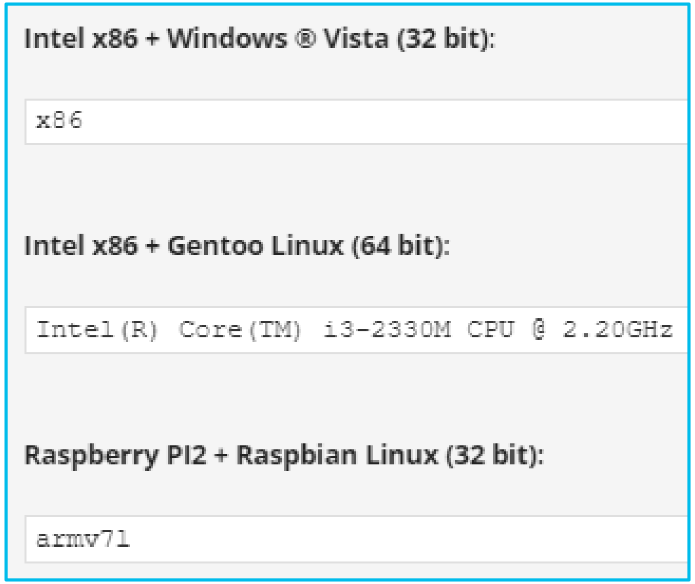
```python
from platform import processor

print(processor())
```



```
>>> import platform
>>> a = platform.processor()
>>> a
'Intel64 Family 6 Model 78 Stepping 3, GenuineIntel'
>>> print(a)
Traceback (most recent call last):
  File "<pyshell#112>", line 1, in <module>
    print(a)
TypeError: 'int' object is not callable
>>> platform.processor()
'Intel64 Family 6 Model 78 Stepping 3, GenuineIntel'
>>> 
```

Selected functions from the random module: continued

The processor function



Intel x86 + Windows ® Vista (32 bit):

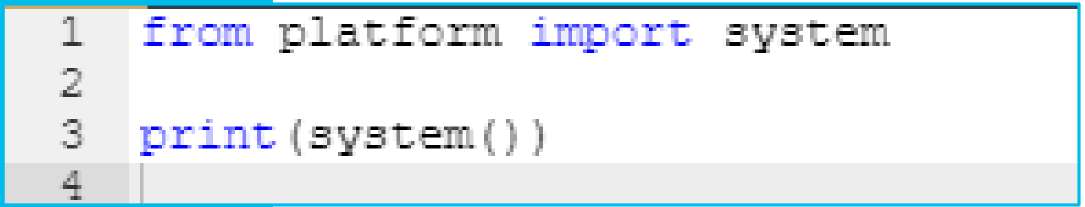
```
x86
```

Intel x86 + Gentoo Linux (64 bit):

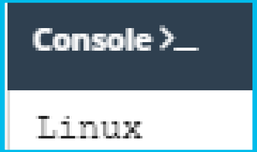
```
Intel(R) Core(TM) i3-2330M CPU @ 2.20GHz
```

Raspberry PI2 + Raspbian Linux (32 bit):

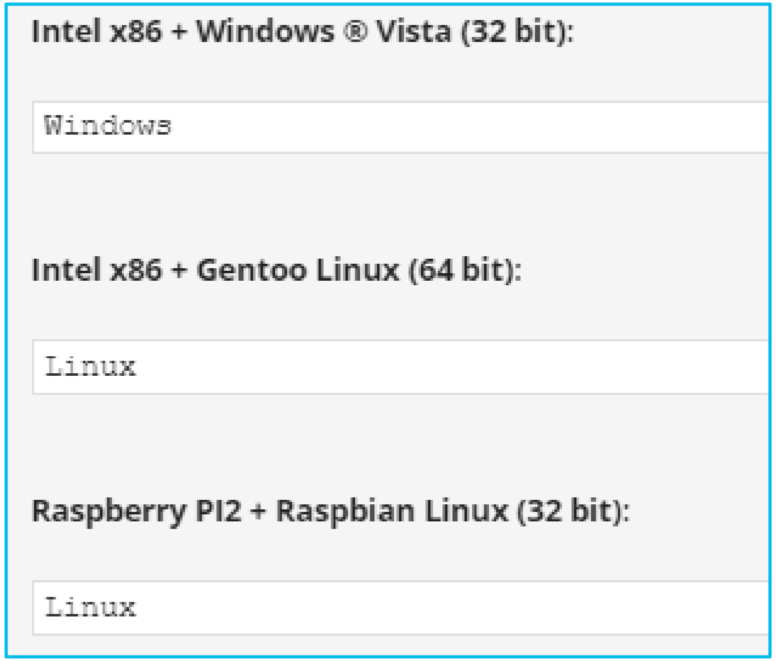
```
armv7l
```



```python
from platform import system

print(system())
```



```
Linux
```



Intel x86 + Windows ® Vista (32 bit):

```
Windows
```

Intel x86 + Gentoo Linux (64 bit):

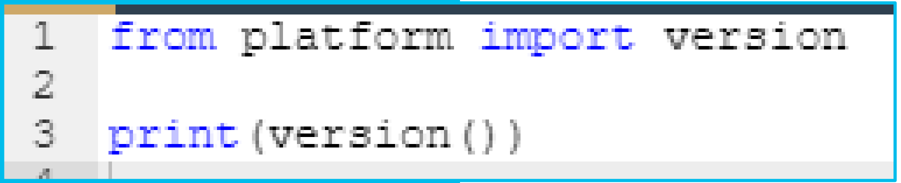
```
Linux
```

Raspberry PI2 + Raspbian Linux (32 bit):

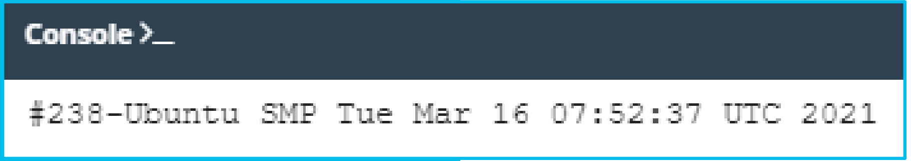
```
Linux
```

Selected functions from the random module: continued

The system function



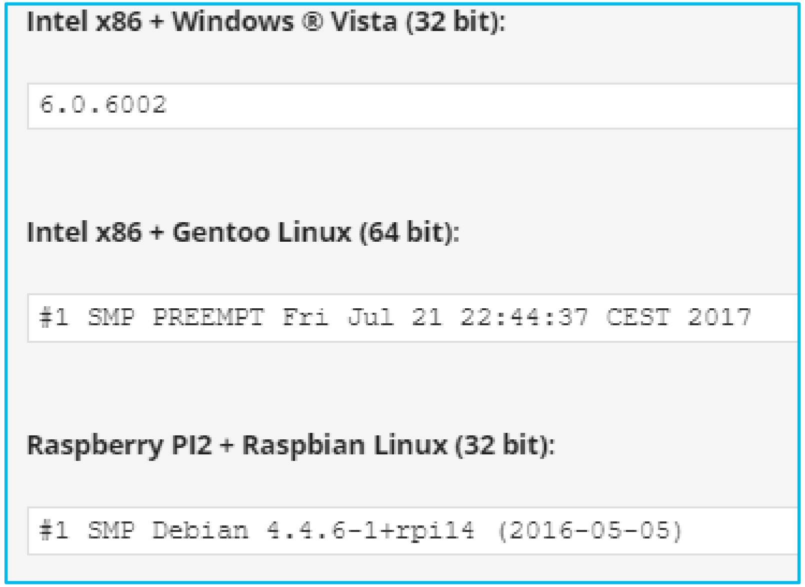
```python
from platform import version

print(version())
```



```
#238-Ubuntu SMP Tue Mar 16 07:52:37 UTC 2021
```

Selected functions from the random module: continued

The version function



Intel x86 + Windows ® Vista (32 bit):

```
6.0.6002
```

Intel x86 + Gentoo Linux (64 bit):

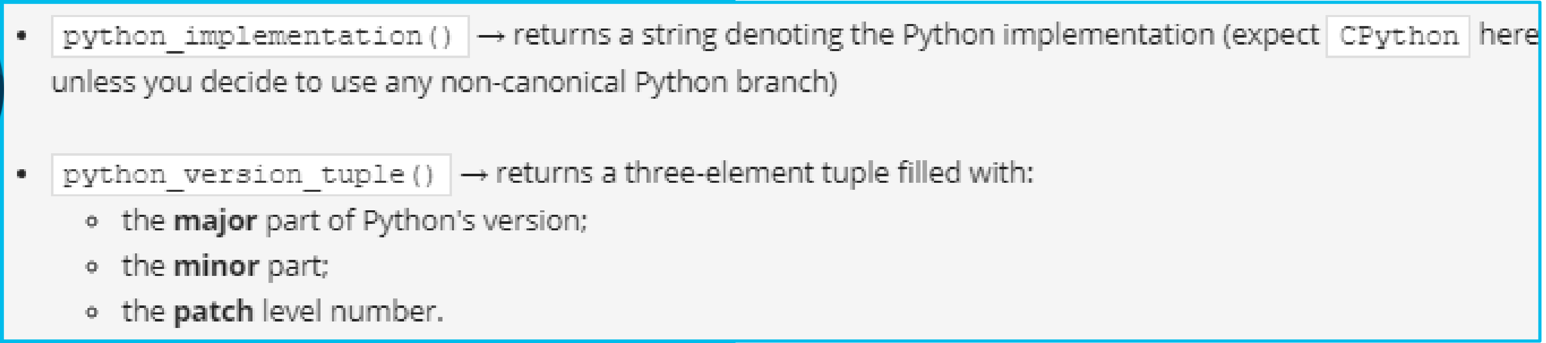
```
#1 SMP PREEMPT Fri Jul 21 22:44:37 CEST 2017
```

Raspberry PI2 + Raspbian Linux (32 bit):

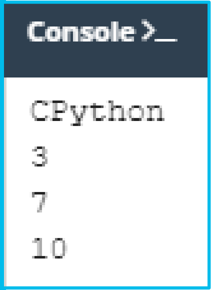
```
#1 SMP Debian 4.4.6-1+rpi14 (2016-05-05)
```

Selected functions from the random module: continued

The python_implementation and the python_version_tuple functions

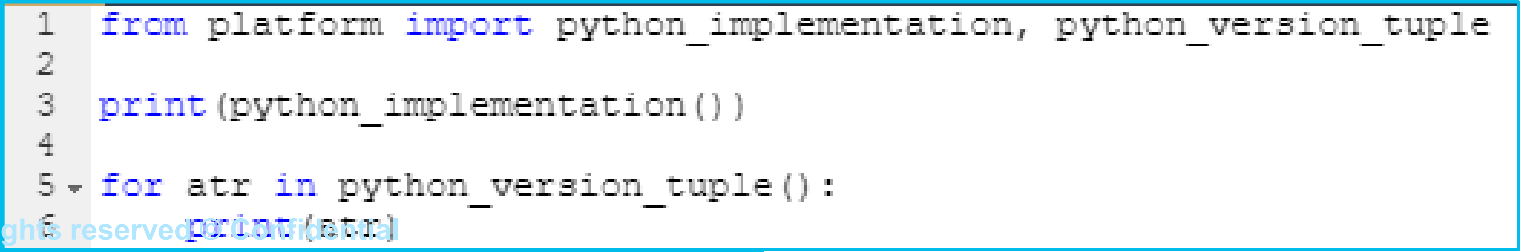


- `python_implementation()` → returns a string denoting the Python implementation (expect `CPython` here unless you decide to use any non-canonical Python branch)
- `python_version_tuple()` → returns a three-element tuple filled with:
  - the **major** part of Python's version;
  - the **minor** part;
  - the **patch** level number.



Console >_

```
CPython
3
7
10
```



```python
from platform import python_implementation, python_version_tuple

print(python_implementation())

for atr in python_version_tuple():
    print(atr)
```

Python Module Index

You can read about all standard Python modules here:

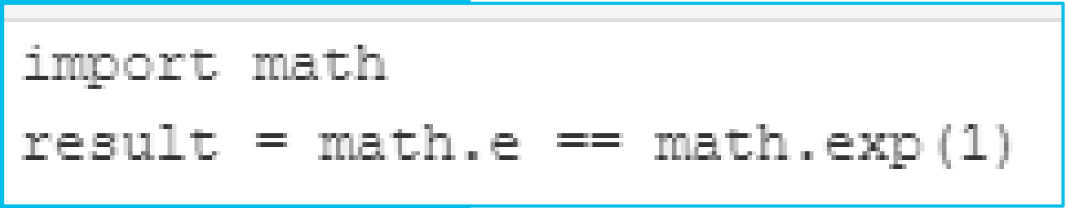
https://docs.python.org/3/py-modindex.html

Key takeaways

1. A function named **dir()** can show you a list of the entities contained inside an imported module. For example:

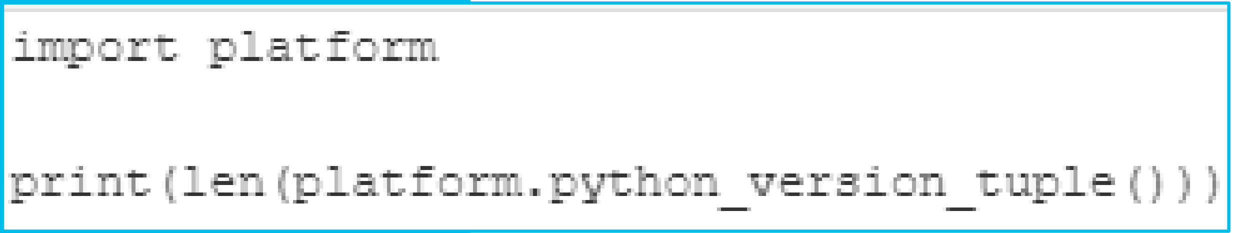
- **import os**
- **dir(os)**

prints out the list of all the **os** module's facilities you can use in your code.
2. The **math** module couples more than 50 symbols (functions and constants) that perform mathematical operations (like sin(), pow(), factorial()) or providing important values (like π and the Euler symbol e).
3. The **random** module groups more than 60 entities designed to help you use pseudo-random numbers. Don't forget the prefix "random", as there is no such thing as a real random number when it comes to generating them using the computer's algorithms.
4. The **platform** module contains about 70 functions which let you dive into the underlaying layers of the OS and hardware. Using them allows you to get to know more about the environment in which your code is executed.
5. Python Module Index (is a community-driven directory of modules available in the Python universe. If you want to find a module fitting your needs, start your search there.



```
import math
result = math.e == math.exp(1)
```

Examples

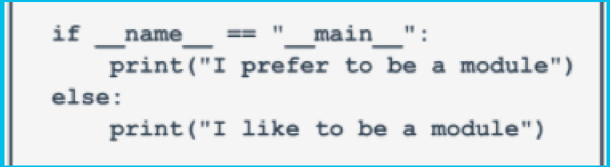


```
import platform


print(len(platform.python_version_tuple()))
```

Home work 7_0

while True:
input()
if s == exit: break - lol

if "" == version:
platform.funcname()



```
if __name__ == "__main__":
    print("I prefer to be a module")
else:
    print("I like to be a module")
```

← **At the end of each file**

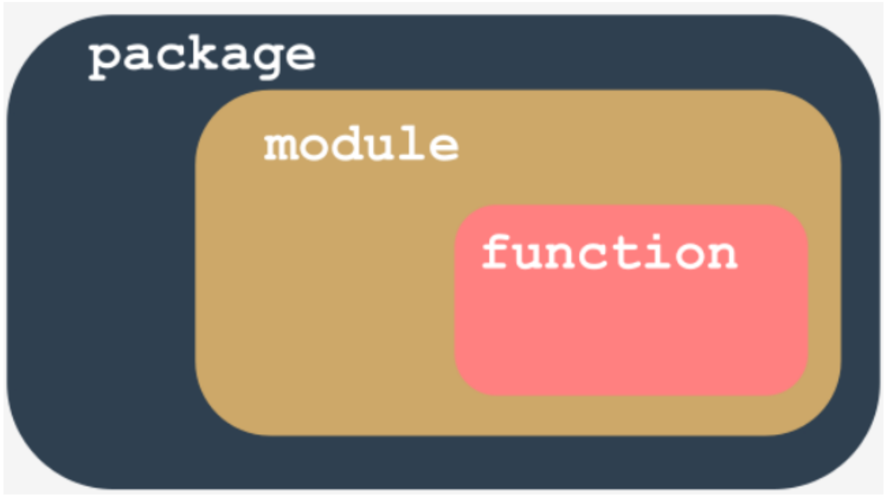
Write a program that, at the user's request:

- Defines **your python version**
- Determines the **capacity of the system**(32/64)
- Defines the **processor**
- And everything that interests you
- In response, the program issues comments, for example, if you have **Linux** - 'Wow, you makin' a hacker! ' Etc.
- By entering the word **exit** - the program **is interrupted**.

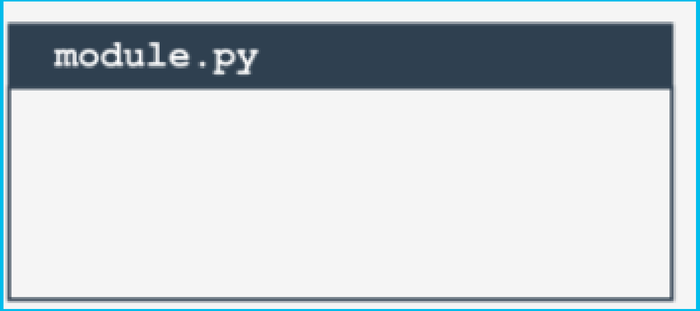
**Also, if desired:

- ***write all data about your system to a file, and output it from the file upon request.

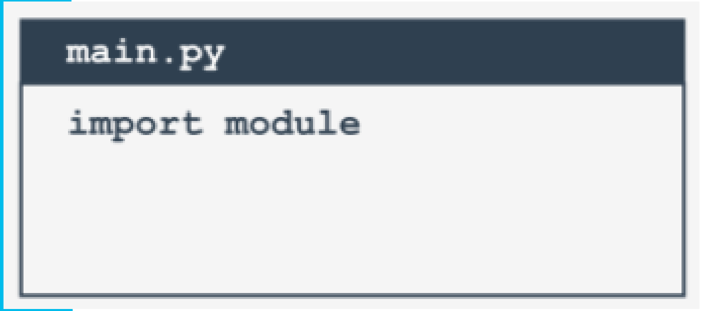
What is a package?
main.py



What is a package? __pycache__

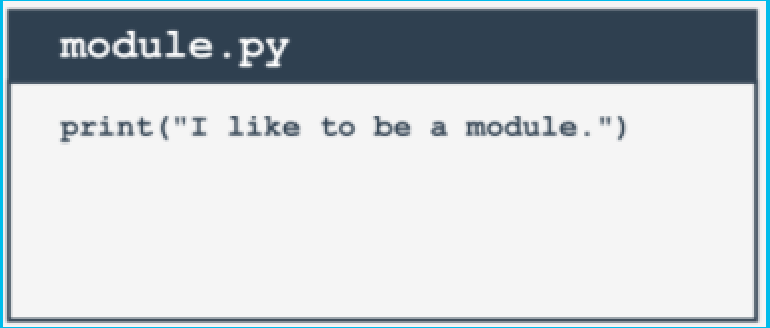


```
module.py
```



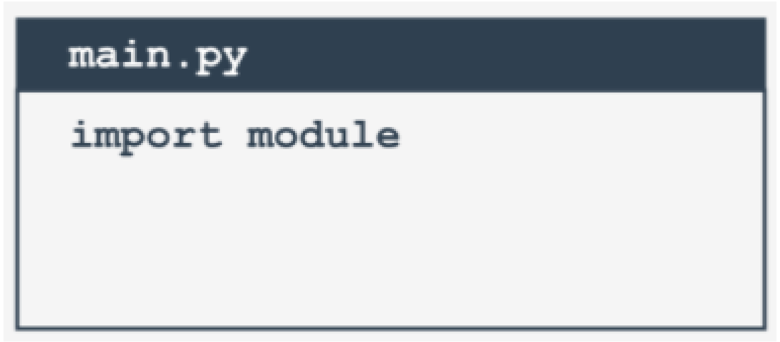
```
main.py
import module
```

What is a package? __pycache__



module.py

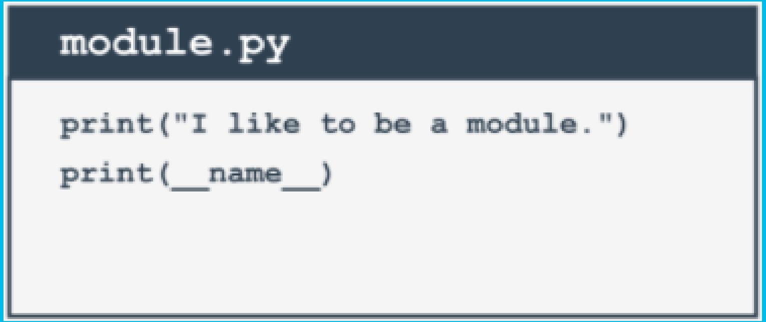
```
print("I like to be a module.")
```



main.py

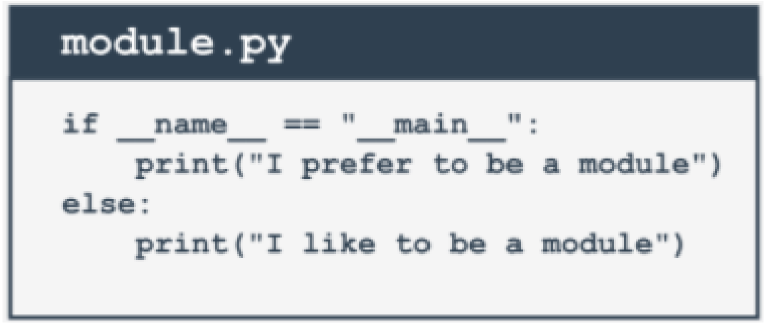
```
import module
```

What is a package?



module.py

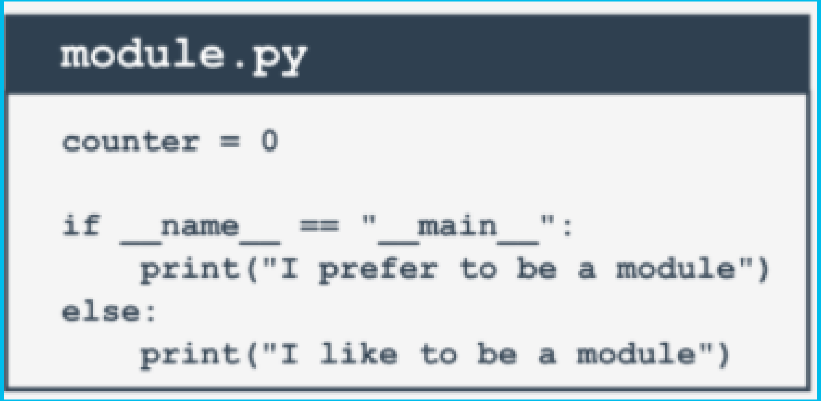
```
print("I like to be a module.")
print(__name__)
```



module.py

```
if __name__ == "__main__":
    print("I prefer to be a module")
else:
    print("I like to be a module")
```

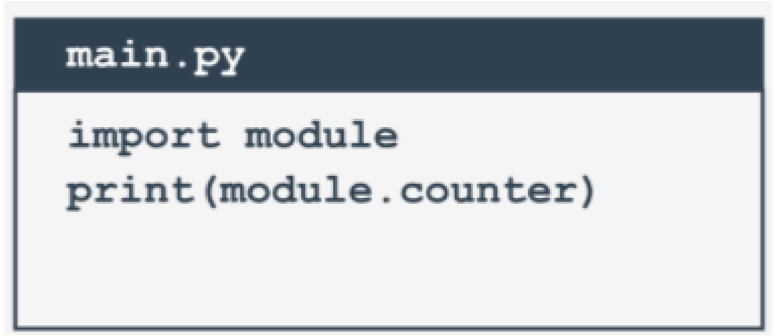
What is a package? __init__



module.py

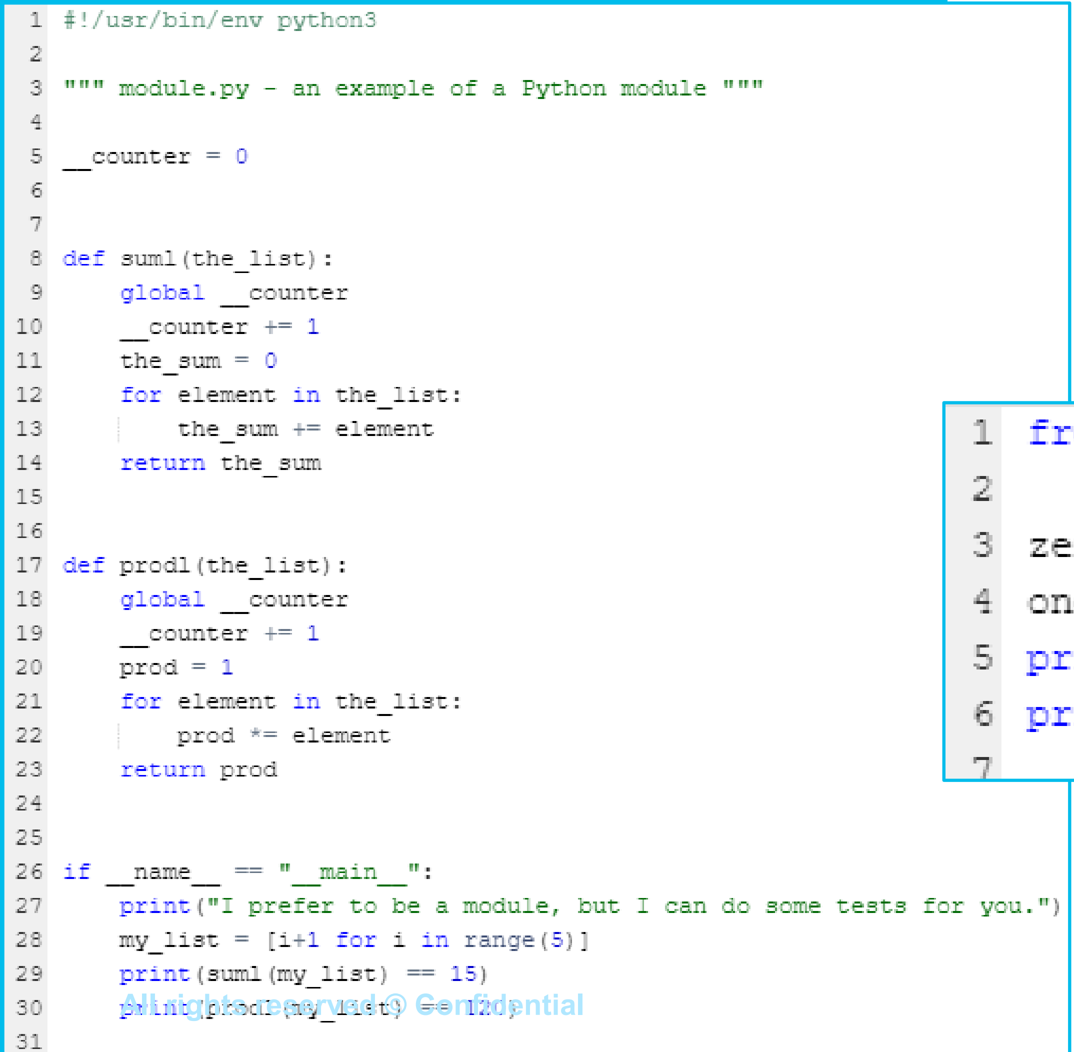
```
counter = 0

if __name__ == "__main__":
    print("I prefer to be a module")
else:
    print("I like to be a module")
```

- myVariable
- my_variable



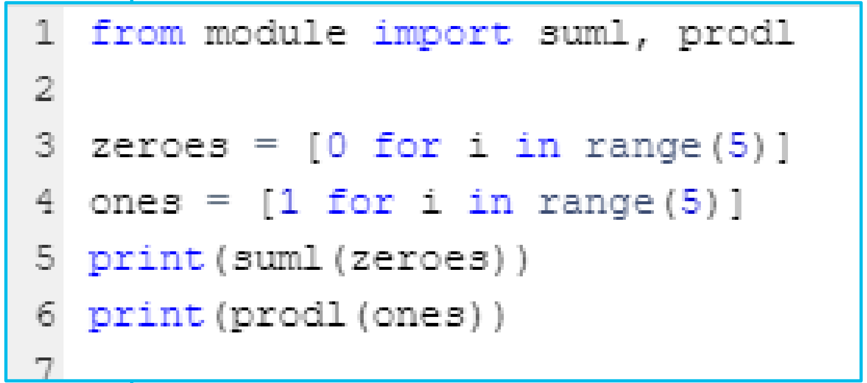
main.py

```
import module
print(module.counter)
```



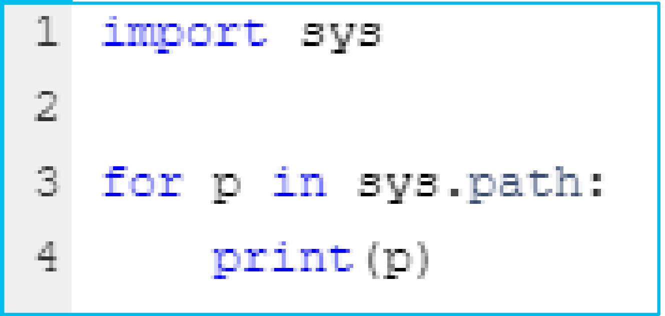
```python
#!/usr/bin/env python3

""" module.py - an example of a Python module """

__counter = 0


def suml(the_list):
    global __counter
    __counter += 1
    the_sum = 0
    for element in the_list:
        the_sum += element
    return the_sum


def prodl(the_list):
    global __counter
    __counter += 1
    prod = 1
    for element in the_list:
        prod *= element
    return prod


if __name__ == "__main__":
    print("I prefer to be a module, but I can do some tests for you.")
    my_list = [i+1 for i in range(5)]
    print(suml(my_list) == 15)
    print(prodl(my_list) == 120)

```

- main.py
- from sys import path
- path.append(..\\modues)

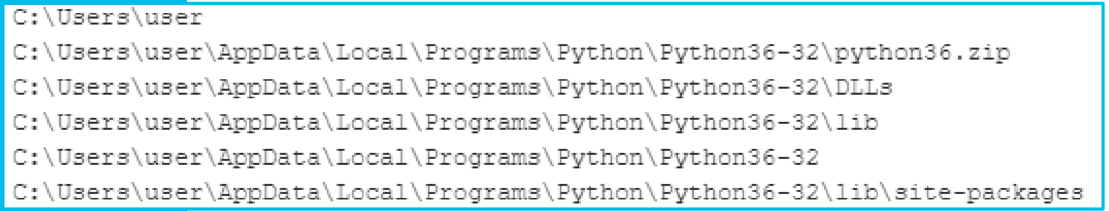


```python
from module import suml, prodl

zeroes = [0 for i in range(5)]
ones = [1 for i in range(5)]
print(suml(zeroes))
print(prodl(ones))

```

What is a package?

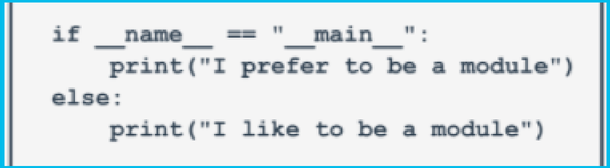


```python
import sys

for p in sys.path:
    print(p)
```



```
C:\Users\user
C:\Users\user\AppData\Local\Programs\Python\Python36-32\python36.zip
C:\Users\user\AppData\Local\Programs\Python\Python36-32\DLLs
C:\Users\user\AppData\Local\Programs\Python\Python36-32\lib
C:\Users\user\AppData\Local\Programs\Python\Python36-32
C:\Users\user\AppData\Local\Programs\Python\Python36-32\lib\site-packages
```

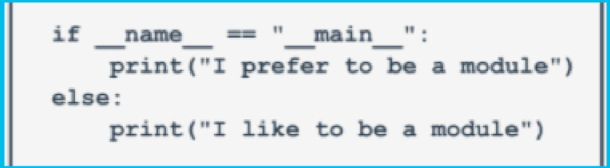
Your first module demo HW 7_1



```
if __name__ == "__main__":
    print("I prefer to be a module")
else:
    print("I like to be a module")
```

← At the end of each file

Your first package demo
HW 7_2



```
if __name__ == "__main__":
    print("I prefer to be a module")
else:
    print("I like to be a module")
```

← At the end of each file

Key takeaways

1. While a module is designed to couple together some related entities (functions, variables, constants, etc.), a package is a container which enables the coupling of several related modules under one common name. Such a container can be distributed as-is (as a batch of files deployed in a directory sub-tree) or it can be packed inside a zip file.

2. During the very first import of the actual module, Python translates its source code into the semi-compiled format stored inside the py files, and deploys these files into the __**pycache**__ directory located in the module's home directory.

3. If you want to instruct your module's user that a particular entity should be treated as private (i.e. not to be explicitly used outside the module) you can mark its name with either the _ or __ prefix. Don't forget that this is only a recommendation, not an order.

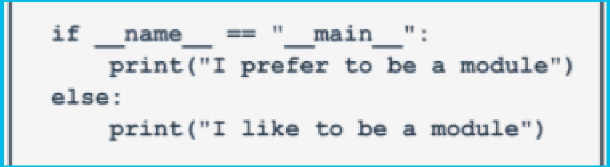
4. # The names shabang, shebang, hasbang, poundbang, and hashpling describe the digraph written as #!, used to instruct Unix-like OSs how the Python source file should be launched. This convention has no effect under MS Windows.

5. If you want convince Python that it should take into account a non-standard package's directory, its name needs to be inserted/appended into/to the import directory list stored in the path variable contained in the sys module.

6. A Python file named __**init**__**.py** is implicitly run when a package containing it is subject to import, and is used to initialize a package and/or its sub-packages (if any). The file may be empty, but must not be absent.

Home work 7_3
ToDO list

USE PACKAGE



```
if __name__ == "__main__":
    print("I prefer to be a module")
else:
    print("I like to be a module")
```

← **At the end of each file**

Data:

- format: [task_number][task_name]
- Input what you want to do while input not equal "stop"

I want to eat
I want to play
and so on

- Exit by word - **stop**

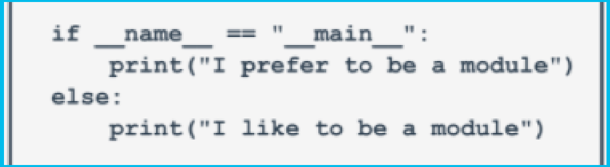
Functions:

- display_task_list(list_task)
- enter_task(list_task)

Home work 7_3

ToDO list

USE PACKAGE



```
if __name__ == "__main__":
    print("I prefer to be a module")
else:
    print("I like to be a module")
```

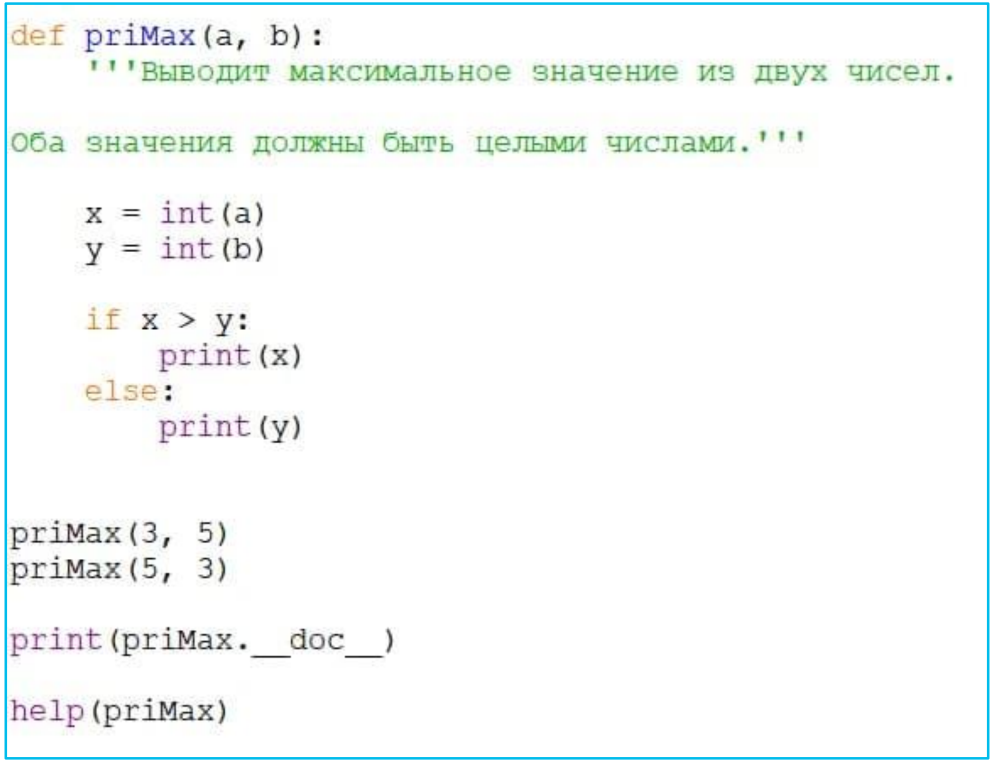
← At the end of each file

packages -- the package todolist
functions inside the modules

Use structure like:

```
Python_projects|
     todo_list/|
               |- modules/
               |- packages/|
               |           |-todolist/
               |                      |__init__.py
               |                      | show_task/ ...py
               |                      | enter_task/ ...py
               |                      |*write_to_file_task/ ...py
               |
               |- main.py – main()
              *|-sources/ *.txt
```

Comments



```
def priMax(a, b):
    '''Выводит максимальное значение из двух чисел.

Оба значения должны быть целыми числами.'''

    x = int(a)
    y = int(b)

    if x > y:
        print(x)
    else:
        print(y)


priMax(3, 5)
priMax(5, 3)

print(priMax.__doc__)

help(priMax)
```

ЗАДАНИЯ

1) Прорешать всю классную работу
2) Выполнить все домашние задания

Почитать:

1) Byte of Python - стр. 78-84

****) Structuring Your Project:**

https://docs.python-guide.org/writing/structure/

Крайний срок сдачи 10/10 в 21:00 (можно раньше, но не позже)

ЗАДАНИЯ

Название файлов, которые вы отправляете мне в telegram:

Vasia_Pupkin_class_work_L7_P0.py

Vasia_Pupkin_L7_0.zip/rar – внутри него папки:

todolist, Example modules, Example packages,

и файл: Vasia_Pupkin_L7_0_platform.py

Формат сообщения которое вы присылаете мне

(после полного выполнения домашнего задания, только один раз) в Telegram:

Добрый день/вечер. Я Вася Пупкин, и это мои домашние задания к лекции 7 часть 0 про Модули и Пакеты.

И отправляете файл с классной работой и архив с ДЗ

Крайний срок сдачи 10/10 в 21:00 (можно раньше, но не позже)

https://docs.github.com/articles/using-pull-requests

Tap to links if you want to know more

Work with files:
https://www.youtube.com/watch?v=oRr_bEXJbV0
https://www.w3schools.com/python/python_ref_file.asp

Books for great peoples:
992 pages of "real" python

993 pages of "real" python

Watch this channel, useful things:
https://www.youtube.com/c/egoroffchannel/playlists

https://www.w3schools.com/python/default.asp

Q&A

Create your
possibilities.
Bye bye.