Loading Modules

Table of Contents

1 Fundamentals of Modules

What is the **module** and what is it used for?

Students, write your response!

Fundamentals of Modules

- We give some explanations about the subject by staying within the frame drawn by related Python documents. *Scripts* and *modules* have essentially identical structures in terms of creation and are the files with a **.py** extension, containing some Python codes, statements, operations, and functions.

Tips :

The difference between these two terms :

- In fact, the difference between them depends on **how** and **for what purpose** you use this file with **.py** extension.

Fundamentals of Modules

script.py

```
python codes
python codescode block1
python codes
python codes

python codes
python codescode block2
python codes
python codes

python codes
python codescode block3
python codes
```

-A python file.

-You can open and edit then run it as a whole.

-You can import (load) it and then call a function or a variable and use it partially.

module.py

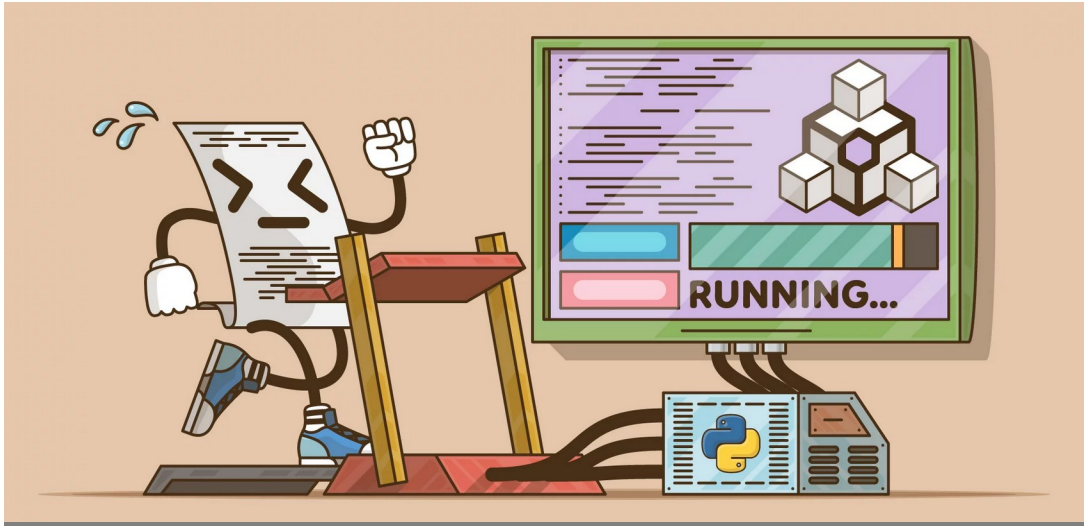
```
python codes
python codescode block1
python codes
python codes

python codes
python codescode block2
python codes
python codes

python codes
python codescode block3
python codes
```

What is Script? (review)

- The file you created consisting of codes, definitions and a list of operations that can be read and interpreted in the future is known as **script**.

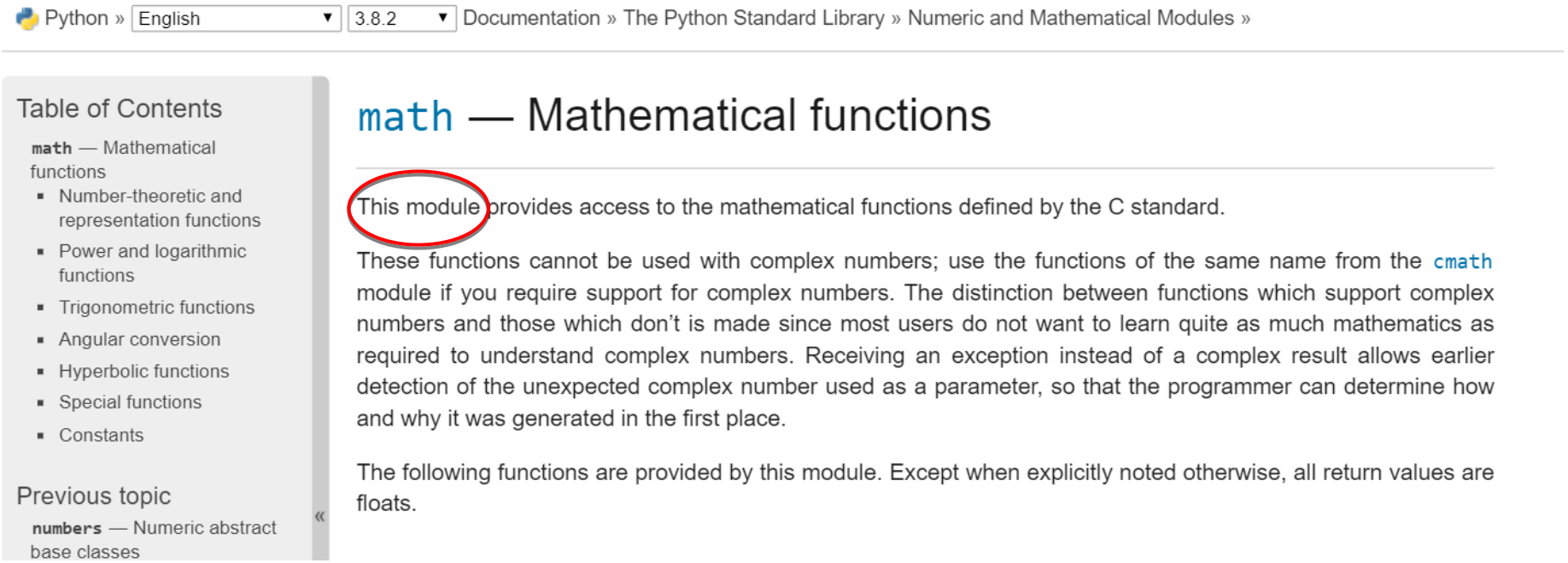


What is Module? (review)

- As your program gets longer, you may want to split it into several files for easier maintenance. You may also want to use a handy function that you've written in several programs without copying its definition into each program.
- To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a **module**.

What is Module?

- The example of built-in module of Python. → `math` module



What is Module?

- These simple files, usually with a **.py** extension and containing Python statements and definitions, are called **modules**. The ability to load or import one module from another in Python is a unique feature that significantly reduces our programming processes, and that is what makes the Module system really

Tips :

- If you open and use this file (with a **.py** extension) *directly*, that is **script**, and
- If you *load* (import) this file (with a **.py** extension) and call any function from it, that's a **module** this time.

2 How to Load a Module?

How to Load a Module? (review)

- my_module is the name of the module we imported. When loading a module you can also use an **abbreviated nickname** for modules by using a keyword as. Let's give a nickname to my_module to load it :

```
1  import my_module as mym # loads my_module, we give a nickname to it
2
3  mym.my_function()  # we can use it the same way
4
5  print(mym.my_variable)
```

- In the example above, mym stands for the module my_module. For instance, imagine that there is a file called my_module.py named my_module. And for being importable, this file should be placed in the same directory as the file you are working on.

How to Load a Module? (review)

- You can also use keyword as here the same way as well. Consider this example :

```
1  from my_module import my_function as mfnc # we've imported my_function named
       mfnc
2
3  mfnc()  # we use the my_function's alias directly
```

- It is traditionally best to type each import syntax in separate lines and put them all at the beginning of the current module. Let's see it in an example :

```
1  import module_1
2  import module_2
3  import module_3
4
5  # The code stream of the current module starts here
```

How to Load a Module? (review)

- Initially, the Python importing mechanism **searches for** a module in the **current directory**, then the built-in modules are inspected and an error will be raised if nothing is found. The module becomes available under its name or alias after `import`ing and you can use the *dot notation* to access the *functions* and *variables* defined in it.
- Importing a function or variable defined in a module is a very common and useful method. We use the *keyword* `from` to use this option. Let's see how it works :

```
1  from my_module import my_function  # we've loaded only my_function from
       my_module
2
3  my_function()  # my_function can be used directly now at the current module
```

3 Built-in Modules

Built-in Modules

- Python comes with a huge library of standard modules many of which are built into the interpreter.
- These modules make our code more effective by providing useful functions and data structures.
- Another advantage of built-in modules which is invaluable to the user is that you can access these standard libraries from all operating systems in which Python is installed.
- Let's have a look at some of them if you want :

Built-in Modules (review)

- `math` is one of the most known and used modules. This module allows us to work with mathematical functions.

```
1  import math
2
3  print(dir(math))  # you can find out all names defined in this module
```

```
1  ['__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', 'trunc']
```

Built-in Modules

- **Task :**
  - Let's import `pi, factorial and log10` functions from `math` module,
  - Print the value of **pi, 4!** and **log10** of **1000** using these functions.

Built-in Modules

- The code block that you should type is as follows :

```
1  from math import pi, factorial, log10  # we'll use the functions directly
2
3  print(pi)  # it also contains several arithmetic constants
4  print(factorial(4))  # gives the value of 4!
5  print(log10(1000))  # prints the common logarithm of 1000
```

```
1  3.141592653589793
2  24
3  3.0
```

Built-in Modules

- `string` module is used for common string operations.
- **Task :**
  - Let's import `punctuation` and `digits` function from `string` module
  - Print the all *punctuation marks* and *digits chars* using these functions.

Built-in Modules

- The code block that you should type is as follows :

```
1  import string as stg  # we've used alias for 'string' module
2
3  print(stg.punctuation)  # prints all available punctuation marks
4  print(stg.digits)  # prints all the digits
```

```
1  !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
2  0123456789
```

Built-in Modules

- `datetime` is commonly used when working with date and time types.
- **Task :**
  - Let's import `today` function from `date` object and `now` function from `datetime` object all from `datetime` module,
  - Print the current *date* (yyyy-mm-dd) and *time* using these functions.

Built-in Modules

- The syntax of importing modules and calling functions are as follows :

```
1  import datetime
2
3  print(datetime.date.today())  # prints today's date (yyyy-mm-dd)
4  print(datetime.datetime.now())  # prints the current time in microseconds
```

```
1  2019-12-31
2  2019-12-31 15:03:31.303994
```

A sample output

Built-in Modules

- `random` is a module that contains functions that allow us to select randomly from various data types.
- **Task :**
  - Let's import `choice` function from from `random` module,
  - Print one of the element of the following `list` randomly.

```
1  city = ['Stockholm', 'Istanbul', 'Seul', 'Cape Town']
2
```

Built-in Modules

- The code and the output should be as follows :

```
1  from random import choice
2
3  city = ['Stockholm', 'Istanbul', 'Seul', 'Cape Town']
4  print(choice(city))
```

```
1  Istanbul
```

Built-in Modules

- **Task :** Using `datetime` module, write a program to calculate the following.
  - According to the general acceptance, the Prophet Muhammad was born on 22th April 571 AD, and died on 8th June 632 AD.
  - How many days have he lived in his life?

Built-in Modules

- The code and the output should be as follows :

```
1  from datetime import date
2
3  birth = date(571, 4, 22)
4  death = date(632, 6, 8)
5
6  life_day = date.toordinal(death)-date.toordinal(birth)
7  print(life_day)
8
```

Creating date object

Converting date to ordinal format (i.e. :0001, 01, 01 → 1)

Output

```
22327
```

THANKS!

Any questions?

You can find me at:

- @joseph
- joseph@clarusway.com