# 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

**script.py**

```
python codes
python codes
python codes
python codes

python codes
python codes
python codes
python codes

python codes
python codes
python codes
```

code block1

code block2

code block3

-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 codes
python codes
python codes

python codes
python codes
python codes
python codes

python codes
python codes
python codes
```

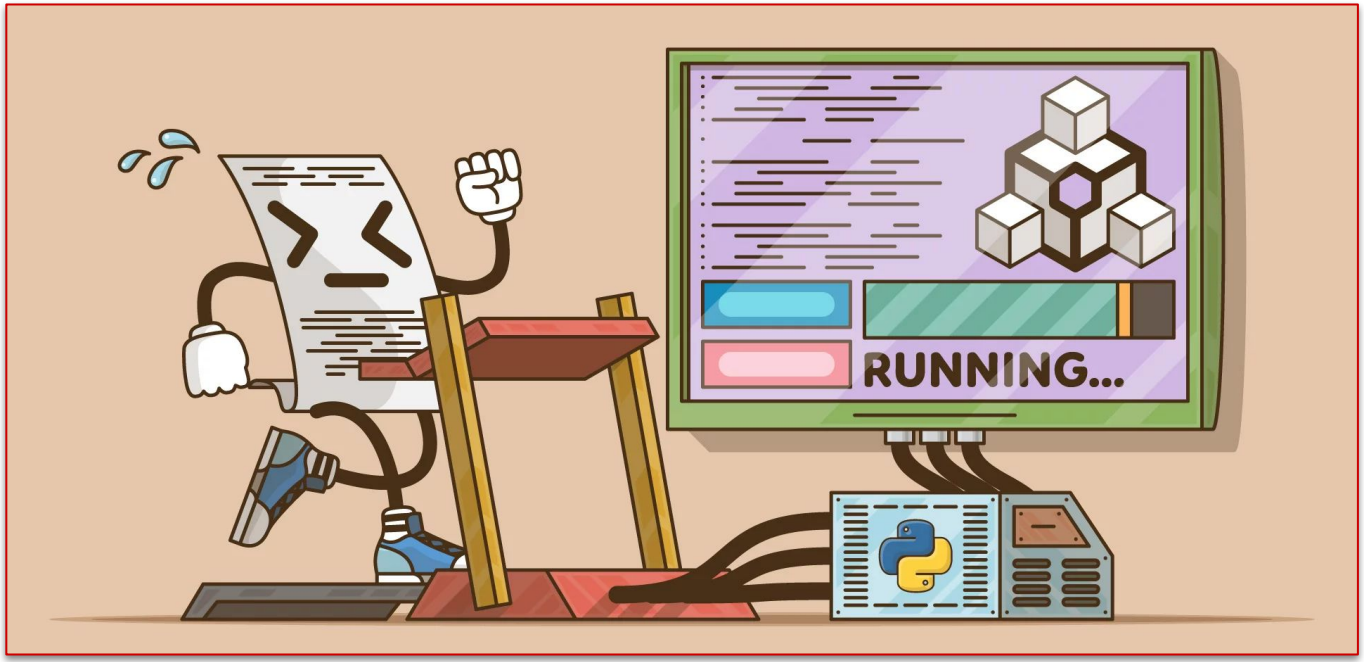
code block1

code block2

code block3

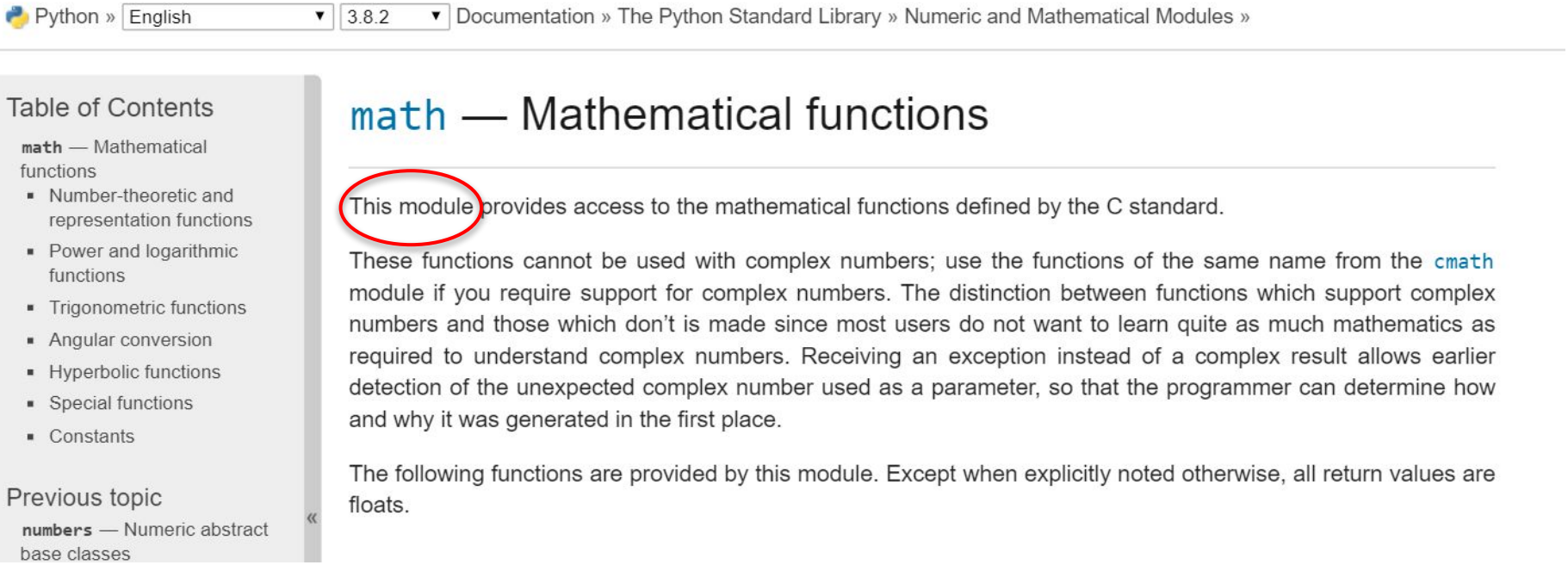
# What is Script? (review)

- Script



# What is Module?

- Module



# What is Module?

- Summary

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)

- Loading modules :

```
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)
```

# How to Load a Module? (review)

- Loading a function in a module directly :

```
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
```

- Loading more than one module :

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

# 3 Built-in Modules

# Built-in Modules

Python » English 3.9.0 Documentation »

## Python Module Index

_ | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | z

| | |
|---|---|
| **_** | |
| __future__ | *Future statement definitions* |
| __main__ | *The environment where the top-level script is run.* |
| _thread | *Low-level threading API.* |
| **a** | |
| abc | *Abstract base classes according to :pep:`3119`.* |
| aifc | *Read and write audio files in AIFF or AIFC format.* |
| argparse | *Command-line option and argument parsing library.* |
| array | *Space efficient arrays of uniformly typed numeric values.* |
| ast | *Abstract Syntax Tree classes and manipulation.* |
| asynchat | *Support for asynchronous command/response protocols.* |
| asyncio | *Asynchronous I/O.* |
| asyncore | *A base class for developing asynchronous socket handling services.* |
| atexit | *Register and execute cleanup functions.* |
| audioop | *Manipulate raw audio data.* |
| **b** | |
| base64 | *RFC 3548: Base16, Base32, Base64 Data Encodings; Base85 and Ascii85* |

# Built-in Modules (review)

- `math`

```
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`

- **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`

- **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

- **Task :** Using `datetime` module, write a program to calculate the following.
  - According to the general acceptance, Ali 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
```

# 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 `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
```

4

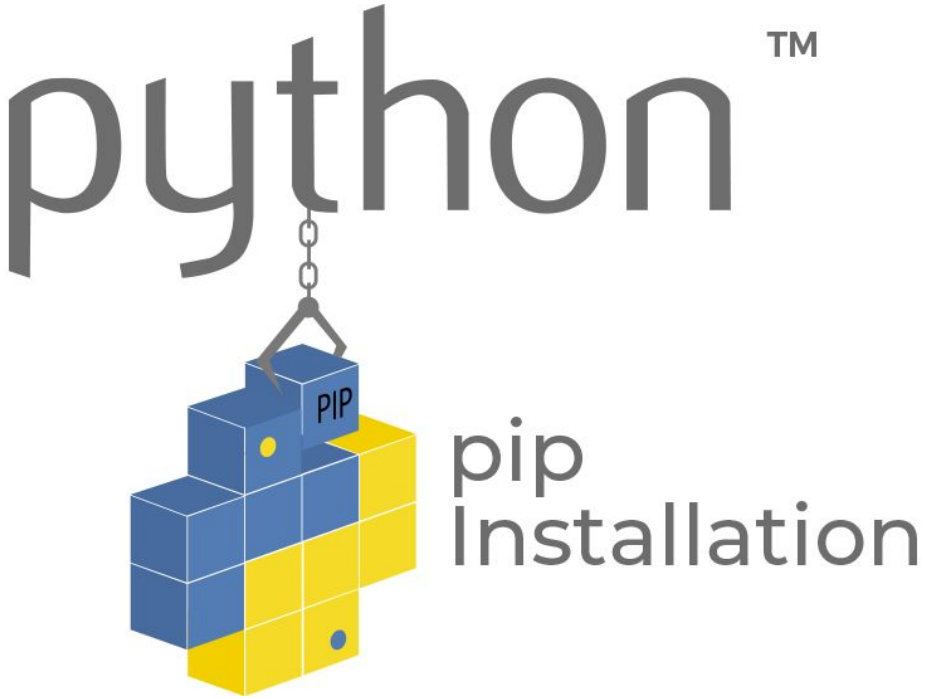
# `pip` - The Package Manager for Python

# How was your pre-class preparation? Did you understand `pip` issue?

Students, drag the icon!

# What is `pip`?

- `pip` is the standard package manager for Python.



# What is `pip`? (review)

- Now open your "**command prompt**", or "**jupyter cell**" and run the following syntax to make sure that you have `pip` installed.

```
pip --version

```

- This code should display your valid pip version which is 19.3.1 currently. The output will be :

```
pip 21.0.1 from c:\users\yd\appdata\local\programs\python\
python38-32\lib\site-packages\pip (python 3.8)
```

- If you have problems with installing or upgrading **`pip`**, you can follow the official guide for the best practice.

# Working with `pip` (review)

```
The formula syntax is : pip command options
```

`install`

- The most common and essential command of `pip` is of course `install`. The most common syntax is :

```
pip install my_package
```

- If you want, you can use this command by adding the version number to the end of the syntax as follows :

```
pip install my_package==3.2.1
```

# Working with `pip` (review)

`install`

```
pip install python==3.8.1
```

# Working with `pip` (review)

`list`

```
pip list
```

# Working with `pip` (review)

`show`

```
1  pip show my_package
```

# Working with `pip` (review)

`uninstall`

```
pip uninstall my_package
```

# Working with `pip`

- **Task :** Using `pip` command ;
  - **List** all packages already installed on your device,
  - **Install** `numpy` and `pandas` packages,
  - **Display** the information of these packages,
  - **List** all packages again that installed on your device.

**You don't need to know what *these packages* used for.**