### Modules

Modules are single python files that can be imported and used inside another file.

#### 1. Import

We use import statement to import a python file into another python file; it will run when the module is imported.  
Inside the other python file, all the modules’ functions and classes can be accessed by module\_name.function\_name.

For example, import a file called my\_program.py, write import my\_program in your code.

# math module

import math

print(math.e) # Euler's number

# >> 2.718281828459045

Convention says you should write all your import statements on the top of the page.

#### 2. From

We can import specific functions from a module with the from statement.

There is no need to call module\_name.function\_name when using the importing from; you need only call the function regularly since it has been directly imported to your application.

Example:

from mymodule import myfunction

from math import floor

print(floor(9.8923))

# >> 9

# Instead of

import math

print(math.floor(9.8923))

# >> 9

#### 3. As

We can use aliases to change the name of the module inside the program.

Example:  
import mymodule as mod

# Calculus of the square root of a number.

# alias called 'm'

import math as m

print(m.sqrt(121))

print(m.sqrt(729))

# >> 11

# >> 27

print(m)

# >> <module 'math' from '/usr/local/lib/python3.8/lib-dynload/math.cpython-38-x86\_64-linux-gnu.so'>

print(dir(m))

# >> ['\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']

#### 4. Pip

Pip is a python packages manager; it can download and install packages alone; it takes them automatically from pypi.org and installs a downloaded package. You can use it from the terminal.

* Package

A package is made up of multiple python files and can even include libraries in C or C++; it’s an entier folder.

* \_\_init\_\_.py

When looking at the structure of the packages, you’ll see a file called \_\_init\_\_.py, this is the old syntax to create python packages, it’s not necessary today, but most of the packages are built like this. This file is executed when the package is loaded.  
Every module in the package can be loaded alone by using from my\_package import my\_module

#### 5. Some Modules Examples

* time
* random
* requests
* os

#### 6. Example Using The Modules Faker And Tabulate

Main Function

if \_\_name\_\_ == "\_\_main\_\_":

Create fake data using faker

From the terminal, install the Faker package with the command: pip install Faker

In your Python program, import Faker and have it create a file of fake dates

from faker import Faker

fake = Faker()

print(fake.date())

Write these to two different files.

**Create a fake Date**

from faker import Faker

fake = Faker()

def create\_file(num):

f = open("datefile" + num + ".txt", "w")

for \_ in range(100):

f.write(f"{fake.date()}\n")

f.close()

print(fake.date())

create\_file('1')

create\_file('2')

**Finds the median between two dates.**

import sys

import typing

from tabulate import tabulate

def main(file1, file2):

file1\_contents = extract\_file\_contents(file1)

file2\_contents = extract\_file\_contents(file2)

display\_files(file1\_contents, file2\_contents)

dates = sorted(set(file1\_contents).union(file2\_contents))

print(dates[len(dates) // 2])

def display\_files(file1\_contents, file2\_contents):

table = {

'file1': file1\_contents,

'file2': file2\_contents

}

print(tabulate(table))

def extract\_file\_contents(file\_path: str) -> typing.List[str]:

"""

Extract file contents and strip whitespaces from each row.

: param file\_path: The path to the file to extract

: return: A list of date data rows

"""

with open(file\_path, 'r') as f:

file\_contents = f.readlines()

new\_file\_contents = []

for date\_data\_record in file\_contents:

date\_data\_record = date\_data\_record.strip()

if date\_data\_record:

new\_file\_contents.append(date\_data\_record)

return new\_file\_contents

if \_\_name\_\_ == '\_\_main\_\_':

args = sys.argv[1:]

if len(args) != 2:

print(f"Expected two arguments as files. Got {len(args)} instead.")

sys.exit(1)

main(\*args)

### Modules

One advantage of functions is the way they separate blocks of code from your main program. By using descriptive names for your functions, your main program will be much easier to follow.

### What Is A Module?

You can go a step further by storing your functions in a separate file called a **module** and then **importing** that module into your main program.

An import statement tells Python to **make the code in a module available** in the currently running program file.

Storing your functions in a separate file allows you to reuse functions in many different programs. When you reserve your functions in individual files, you can share those files with other programmers without having to share your entire program.

Knowing how to import functions also allows you to use libraries of functions that other programmers have written.

There are several ways to import a module; we will describe them in this chapter. But first, let’s **create our own module**!!

### Creating A Module

A module is just a python file (a file with the .py extension).

Let’s create a file called pizza.py containing the function make\_pizza().

def make\_pizza(size, \*toppings):

"""

Summarize the pizza we are about to make.

"""

print("\nMaking a {}-inch pizza with the following toppings:".format(size))

for topping in toppings:

print("- " + topping)

We will create another file called making\_pizzas.py (in the same directory as pizza.py).  
This file will **import the function** we just created and made two calls to it.

### Importing An Entire Module

To import an entire module:

* use the import statement,
* followed by the name of the module (without the extension).

For example, to import the pizza.py file:

import pizza

When Python reads this file, import pizza tells Python to open the file **pizza.py** and copy all the functions from it into this program.

You don’t see code being copied between files because Python copies the code behind the scenes as the program runs.

All you need to know is that any function defined in **pizza.py** will now be **available** in **making\_pizzas.py**.

To call a function from an imported module, enter the name of the module you imported, pizza, followed by the name of the function, make\_pizza(), separated by a dot.

import pizza

pizza.make\_pizza(16, 'pepperoni')

pizza.make\_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')

### Importing Specific Functions

You can also import a specific function from a module. Here’s the general syntax for this approach:

from module\_name import function\_name

You can import as many functions as you want from a module by separating each function’s name with a comma:

from module\_name import function\_0, function\_1, function\_2

The making\_pizzas.py example would look like this if we want to import just the function we’re going to use:

from pizza import make\_pizza

make\_pizza(16, 'pepperoni')

make\_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')

### Using Alias

You can give a nickname to each module/function you import. This nickname, a short, unique alias (an alternate name similar to a nickname for the function), is defined as` keyword.

from pizza import make\_pizza as mp

The general syntax for a function is

from module\_name import function\_name as alias\_name

And for a module:

import module\_name as alias\_name

#### Exercise

Create a file called operators.py

Create 2 functions : addOperator(x,y) that returns the addition of 2 numbers, and divideOperator(x,y) that returns the division of 2 numbers

Create another file called calculator.py, and import the operators module. Call the 2 functions and display the results

Do this process 3 times :

* once by importing the whole module
* the second time by importing specific functions
* the third time by using alias

### Installing An External Module

Python has a huge open source community, it means a lot of developers put their source code online.

You can download their modules to be able to use their function.

Python has a built-in modules installer, called pip (sometimes called pip3).

To check your pip installation, enter the following command in a terminal:

* For Linux and Mac OS: pip3 --version
* For windows: python -m pip --version

Pip has a list of modules that he can download, available on [pypi](https://pypi.org/" \t "_blank).

To install one of the modules, check its name on pypi and use:

$ pip install module\_name

### Working With A Requirements File

A requirements file is used to make it easier for other developers to install the correct versions of the required Python package to run the Python code we’ve written.

Most python programs come with a text file, usually a requirements.txt containing a list of every required module used in your python code and their version.  
It is good practice to generate and share a requirements.txt file when uploading your code to Github  
A requirements.txt file looks like this:

appnope==0.1.0

backcall==0.1.0

beautifulsoup4==4.6.3

bleach==2.1.4

certifi==2018.8.24

chardet==3.0.4

Click==7.0

cycler==0.10.0

decorator==4.3.0

defusedxml==0.5.0

entrypoints==0.2.3

Flask==1.0.2

You can install every required module from a requirements.txt file with the pip install -r (filename) argument within pip.

pip install -r requirements.txt

You can generate a requirements file for your python project with pipreqs, install it with:

$ pip install pipreqs

To generate a requirements file, just run this in your project folder:

$ pipreqs .

There is also an alternate way to do this using pip freeze inside your project folder:

$ pip freeze > requirements.txt

This command creates the requirements.txt in the same way pipreqs would write the file.