# 3 Days Training on Python3

## **Day 3 : Module 9**

Muhammad Saufy Rohmad

# Module 8 (90 minutes)

Objectives

1. Python Modules and Packages

# 1. Python Modules and Packages

- Modules and packages are two constructs used in Python to organise larger programs.
- This module introduces modules in Python, how they are accessed, how they are define and how Python finds modules etc.
- It also explores Python packages and sub-packages.

# 1. Python Modules and Packages(2)

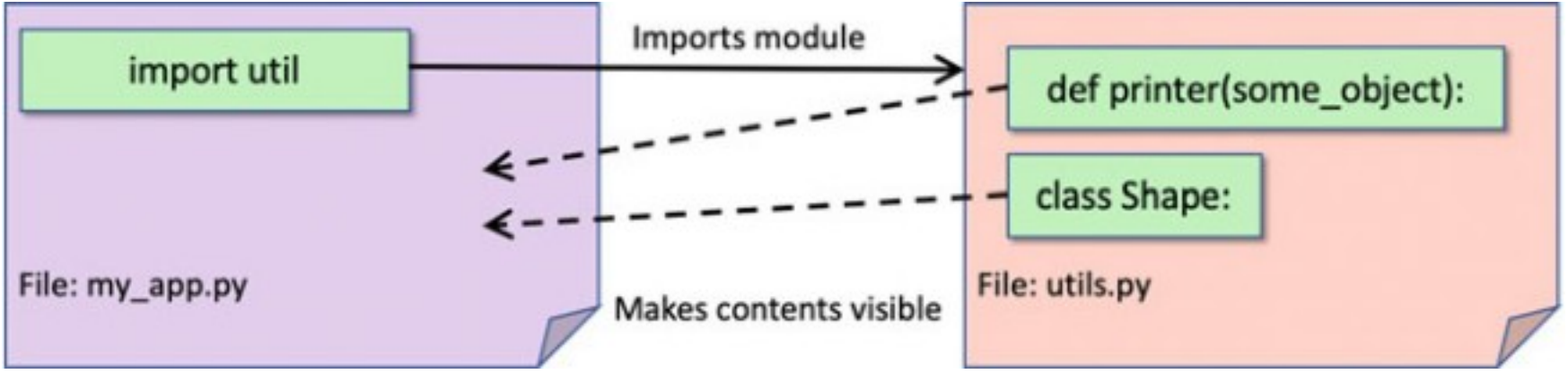
- A module allows you to group together related functions, classes and code in general.
- It is useful to organise your code into modules when the code either becomes large or when you want to reuse some elements of the code base in multiple project
- Breaking up a large body of code from a single file helps with simplifying code maintenance and comprehensibility of code, testing, reuse and scoping code
- These are for:
  - Simplicity
  - Maintenance
  - Testing
  - Reusability
  - Scoping

# 1. Python Modules and Packages(3)

- In Python a module equates to a file containing Python code. A module can contain
  - Function
  - Classes
  - Variables
  - Executable Code
  - Attributes associated with the module such as its name
- The name of a module is the name of the file that it is defined in (minus the suffix ‘.py’). For example, the following diagram illustrates a function and a class defined within a file called utils.py:

# 1. Python Modules and Packages(4)

- my_app.py is a script file that use utils.py



# 1. Python Modules and Packages(5)

- Write 2 modules files in same directory as your code.

  ***yourname_mod1.py***

  - Write any function here

  ***yourname_mod2.py***

  - Write any function here

- Write another program ***test_mod.py*** to call the function in your module files.

# 1. Python Modules and Packages(5)

- from <module name> import *
- from utils import Shape
- import utils as utilities
- from utils import printer as myfunc
- from utils import Shape, printer as myfunc

# 1. Python Modules and Packages(6)

- By default, any element in a module whose name starts with an underbar ('_') is hidden when a wildcard import of the contents of a module is performed
- In this way certain named items can be hidden unless they are explicitly imported.

```
def _special_function():
    print('Special function')
```

# 1.1 Module Properties

- Every module has a set of properties that can be used to find what features it provides, what its name is, what (if any) its documentation string is etc.
- These properties are considered special as they all start, and end, with a double underbar ('__'). These are:
  - __name__ the name of the module
  - __name__ the name of the module
  - __file__ the file in which the module was defined.

# 1.1 Module Properties(2)

- You can also obtain a list of the contents of a module once it has been imported using the dir(<module-name>) function. For example:

```
import utils
print(utils.__name__)
print(utils.__doc__)
print(utils.__file__)
print(dir(utils))
```

- Note that the executable print statement is still executed as this is run when the module is loaded into Python's current runtime; even though all we do it then access some of the module's properties.
- Mostly module properties are used by tools to help developers; but they can be a useful reference when you are first encountering a new module.

# 1.2 Standard Modules

- Python comes with many built-in modules as well as many more available from third parties.
- Of particular use is the sys module, which contains a number of data items and functions that relate to the execution platform on which a program is running.
- Some of the sys module’s features are shown below, including sys.path(), which lists the directories that are searched to resolve a module when an import statement is used. This is a writable value, allowing a program to add directories (and hence modules) before attempting an import.

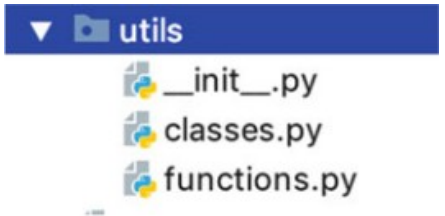
# 1.2 Standard Modules(2)

```
import sys
print('sys.version: ', sys.version)
print('sys.maxsize: ', sys.maxsize)
print('sys.path: ', sys.path)
print('sys.platform: ', sys.platform)
```

- Modules also can run as a normal script

# 1.3 Python Packages

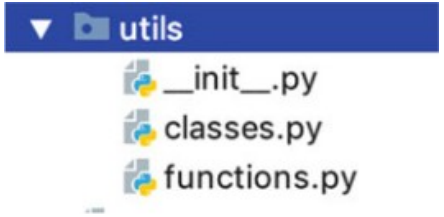
- Python allows developers to organize modules together into packages, in a hierarchical structure based on directories.
- A package is defined as
  - a directory containing one or more Python source files and
  - an optional source file named __init__.py. This file may also contain code that is executed when a module is imported from the package.
  - For example, the following picture illustrates a package utils containing two modules classes and functions.



# 1.3 Python Packages(2)

- In this case the __init__.py file contains package level initialisation code:

  print('utils package')



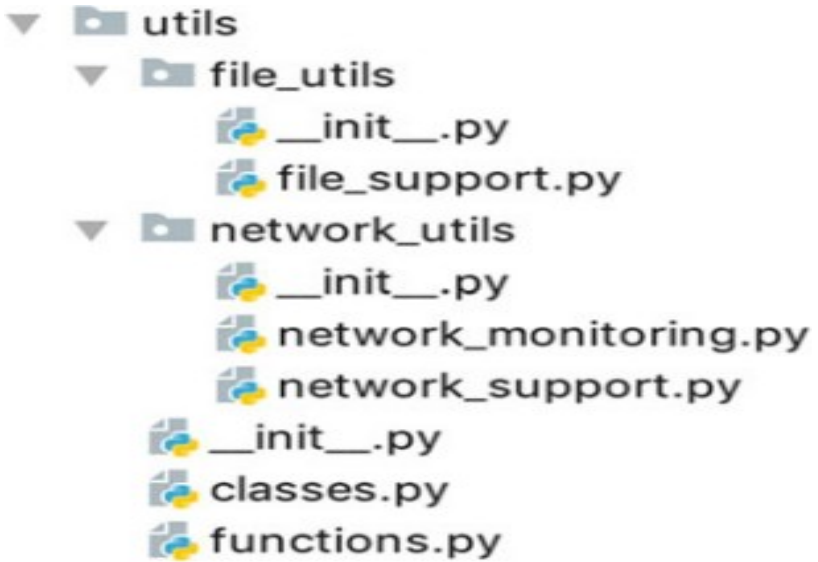

- In __init__.py, write

  from utils.functions import *

  from utils.classes import *

# 1.3 Python Packages(3)

- Packages can contain sub packages to any depth you require. For example, the following diagram illustrates this idea:



# Exercise

- Create your own package
- Put in a directory ***yourname***
- Write __init__.py file
- Write ***module1.py*** and ***module2.py***
- Copy the directory to sys.path
- Import your modules with import yourname.module1 and import yourname.module2 in your program

# 2. Monkey Pacthing and Attributes Lookup

xxx