Python | Main course

Session 14

- Modules
- Packages
- PIP
- Types

by Mohammad Amin H.B. Tehrani

www.maktabsharif.ir

Modules

Modules

Intro

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

Snapp example:

```python
# users.py: Users module

class User: pass

class Driver(User): pass

class Passenger(User): pass
```

```python
# vehicles.py :Vehicles module

class Vehicle: pass

class Car(Vehicle): pass

class Motor(Vehicle): pass
```

```python
# main.py : main module

import users
import vehicles

def main():
    ...
```

Using a module

'import' Statement:

You can use any Python source file as a module by executing an **import** statement in some other Python source file. The import has the following syntax:

```
import module_name1 [, module_name2, module_name3, ...]
```

```
import dill
import pickle
import re
```

=

```
import dill, pickle, re
```

‘as’ keyword

If the module name is followed by **as,** then the name following as is bound directly to the imported module.

```
import module_name1 as new_module_name
```

```
import dill as DILL
import pickle as P_
import re as regex
```

=

```
import dill as DILL, pickle as P_, re as regex
```

from ... import ...

Python's **from** statement lets you **import** specific attributes from a module into the current namespace. The from...import has the following syntax:

```
from module_name1 import some_variable[, some_function, some_class]
```

```
from math import tan, sin, pow
```

```
from math import tan as tangent, sin as sinus, pow as power
```

It is also possible to import **all names** from a module into the current namespace by using the following import statement

```
from module_name1 import *
```

```
from math import *
```

What's the difference between:
import math
and
from math import *
?

Packages

Packages

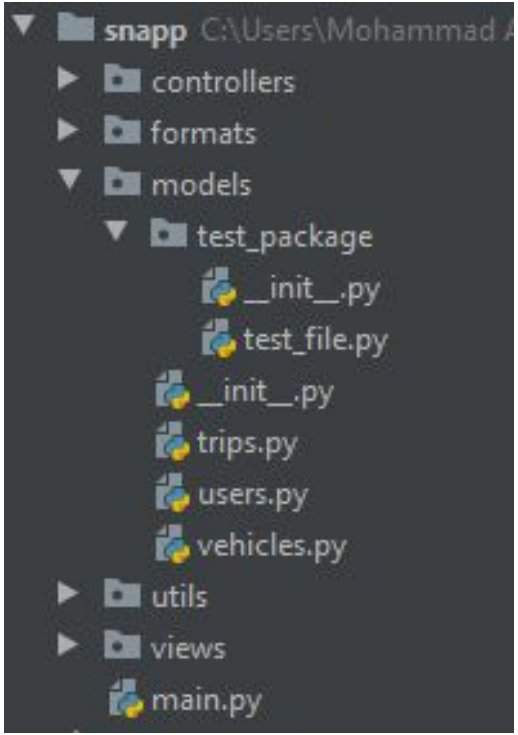
Intro

A **package** is basically a **directory** with Python files and a file with the name **__init__.py .**
This means that every directory inside of the Python path, which contains a file named **__init__.py**, will be treated as a package by Python. It's possible to put several modules into a Package.

Packages are a way of structuring Python's module namespace by using "dotted module names".

Example:

```
from models.test_package.test_file import TestClass
```



Packages

___init___.py

The __init__.py file makes Python treat directories containing it as modules. Furthermore, this is the first file to be loaded in a module, so you can use it to execute code that you want to run each time a module is loaded, or specify the submodules to be exported.

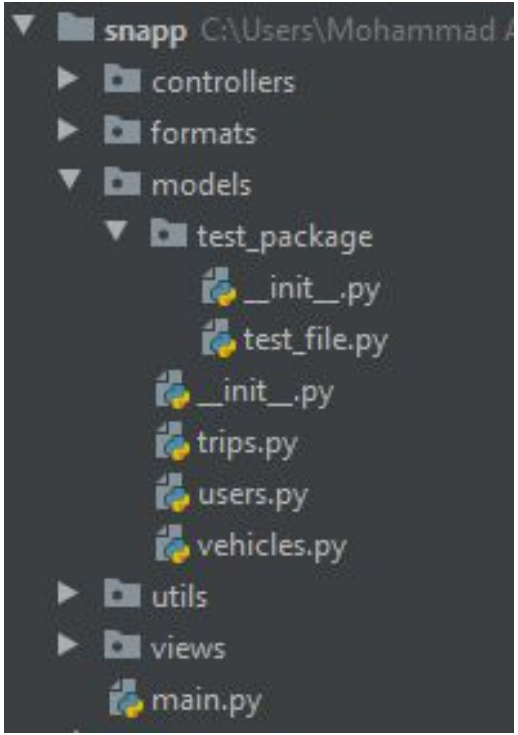
```
# models/__init__py

print(f"\n====== Module {__name__} =======")
print('Path:', __path__)
print('File:',__file__)
print('Name:',__name__)
print('Package:',__package__)
```

```
# main.py

import models
```

Output???



Packages

___init___.py > Example

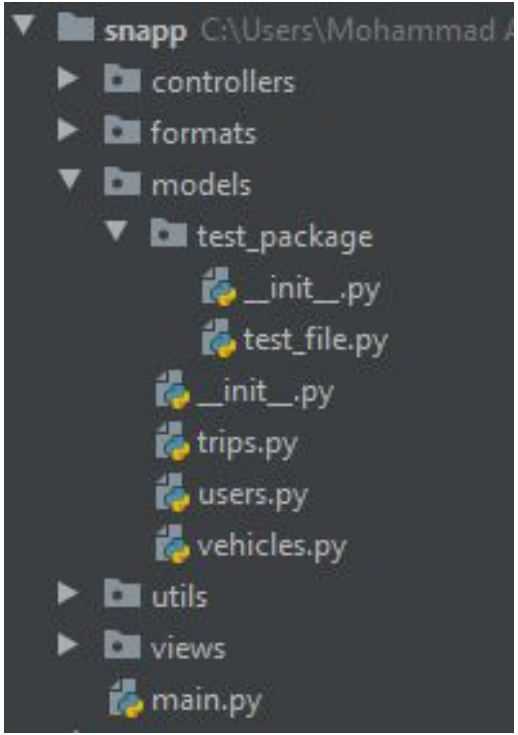
```python
# models/__init__py

print(f"\n====== Module {__name__} =======")
print('Path:', __path__)
print('File:',__file__)
print('Name:',__name__)
print('Package:',__package__)
```

```python
# main.py

import models
```

```
====== Module models =======
Path: ['C:\\Users\\~\\PycharmProjects\\snapp\\models']
File: C:\Users\~\PycharmProjects\snapp\models\__init__.py
Name: models
Package: models
```



Some special variables on Packaging

- **___name___ :** Name of module imported
- **___file___:** Absolute file directory to file imported
- **___package___:** Package name
- **___class___:** Name of class (If class was imported)

UserManager example: packaging

Maktab Sharif

UserManager: Packaging

Use packages and modules for organize your code.

- exceptions.py
- models.py
- menus.py
- main.py
- ...

```
USER MANAGER PROGRAM
     1. Register
     2. Login
Enter option:
```

```
USER MANAGER > REGISTER
>> phone:
>> password:
>> name:
>> email(Optional):
Registered Successfully!
```

```
USER MANAGER > LOGIN
>> phone:
>> password:
ERROR: Invalid password
```

PIP

PIP

Intro

Python Installs Packages

pip is a package-management system written in Python used to install and manage software packages. It connects to an online repository of public and paid-for private packages, called the Python Package Index.

Syntax

- if set in env variables:

```
pip ...
```

- From python interpreter:

```
python -m pip ...
```

Commands (pip -h)

install	Install packages.
download	Download packages.
uninstall	Uninstall packages.
freeze	Output installed packages in requirements format.
list	List installed packages.
show	Show information about installed packages.
check	Verify installed packages have compatible dependencies.
config	Manage local and global configuration.
search	Search PyPI for packages.
cache	Inspect and manage pip's wheel cache.
wheel	Build wheels from your requirements.
hash	Compute hashes of package archives.
completion	A helper command used for command completion.
debug	Show information useful for debugging.
help	Show help for commands.

Typing

Intro

Maktab Sharif

Python is a dynamically typed language. That means it is not necessary to declare the type of a variable when assigning a value to it.
Coding in a dynamically typed language like Python surely is more flexible, but one might want to annotate data types of objects and enforce type constraints. If a function only expects integer arguments, throwing strings into the function might crash the program.

In python we can define Types for variables and parameters , …
But it's NOT Strict Type checking yet.
It's mean than we can recommend our code user to use the type.

```
x: int = 123
y: float
b: bool = True
```

```
x: int = 12.3
y: float =
b: bool = range(1, 10)
```

Expected type 'int', got 'float' instead

Syntax

Maktab Sharif

Variable Typing:

<var_name> **: <var_type>** = value

Function Typing:

```
def func_name(arg1:type1,arg2:type2, ...) -> return_type:
    pass
```

```
def func(x: int, y: float, z: str = 'akbar'):
  res: str
  res = int(x*y)*z
  return res
```

Using class types:


```
class User:
    id: int
    name: str
    phone: str
    ...


def register(u: User):
    ...


def login(username: str, password: str) -> User:
    ...
```

‘typing’ module

Maktab Sharif

import

```
import typing
```

Some useful types:

- List
- Tuple
- Any
- Union
- Optional
- Literal
- Dict
- Set

Example: Translator

Maktab Sharif

Translator

Install ‘translators’ package using pip and write a console program that gets a file path from user and translate the file content to user defined target language (default: fa).

- Install and Use **‘translators’** package
- Log all of programm events -> INFO level
- Use Object-Oriented design.
- Prepare **save** method to save content to a file.
- Try to handle all of exceptions (like invalid file path and etc)
- Try to organize your code using packaging

Final example: WikiPedia Translator

Maktab Sharif

WikiPedia Translator

Create class (WikiTranslator) that get a title of a wikipedia article and translate the summary to a target language

- Install and Use **'wikipedia'** package
- Install and Use **'translators'** package
- Search all possible articles and suggest to user (using wikipedia.search())
- Log all of programm events -> INFO level
- Prepare **save** method to save content to a file.
- Try to handle all of exceptions
- Try to organize your code using packaging
- Write a console program finally (prepare menus & etc)

Advanced topics

- Dynamic import (__import__ function)
- * PIP freeze & PIP -r
- PIP Wheel