Introduction to Computer Programming

Week 6.1: Importing Python files

Modularity: Breaking large chunks of code into smaller, more manageable pieces.

Useful blocks of code (e.g. variables, functions, classes) can be stored in a python file (a **module**)

The module is then `import` ed for use in a program saved elsewhere on your computer.

Example The Python module, `math` installs with Python
[https://docs.python.org/3/library/math.html (https://docs.python.org/3/library/math.html)](https://docs.python.org/3/library/math.html)

In [2]:

```python
import math

print(math.pi)
```

```
3.141592653589793
```

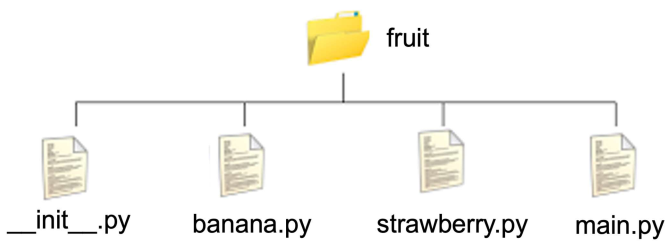
Module: A python file containing python code (variables, functions, classes etc).

Package: A file directory (folder) containing python files (and other directories).

Script: A top level file, run as an program (importing would run the program).

`__init__.py` : Required to make Python treat a directory as a package.
Can be empty or execute initialization code for a package.

Example: Three files in the same directory, `fruit`



Example: Four files in the same directory, `fruit`
Import variables from `banana.py` and `strawberry.py` into `main.py` :

```
fruit/
|
├── __init__.py
├── banana.py
├── strawberry.py
└── main.py
```

Contents of four files in the same directory:

---

init.py

```
# (empty file)
```

---

banana.py

```
word = 'banana'
```

---

strawberry.py

```
word = 'strawberry'
```

---

main.py

```
import banana
import strawberry
print(banana.word)
print(strawberry.word)
```

When we run `main.py` the contents of `banana.py` and `strawberry.py` are imported and can be used within the `main.py` program.

Example: Three files in the same directory, `fruit`
Import function from `banana.py` into `main.py` :

```
fruit/
|
├── __init__.py
├── banana.py
└── main.py
```

Contents of three files in the same directory:

---

init.py

```
# (empty file)
```

banana.py

```
def peel():
    print('Peel!')
```

main.py

```
# main.py
import banana
banana.peel()
```

Contents of three files in the same directory:

init.py

```
# (empty file)
```

banana.py

```
class Banana():
    def __init__(self):
        pass
    def peel(self):
        print('Peel!')
```

main.py

```
# main.py
import banana
b = Banana()
b.peel()
```

Namespaces

Each Python file has a local namespace.

This is a “symbol table” that contains the names of imported modules, packages etc.

When you import a package/module, the part after `import` gets added to the local namespace.

This part should be used to prepend all variables etc from the imported module, to use them in the current program.

Contents of four files in the same directory:

init.py

```
# (empty file)
```

banana.py

```
word = 'banana'
```

strawberry.py

```
word = 'strawberry'
```

main.py

```
import banana
import strawberry
print(banana.word)
print(strawberry.word)
```

We prepend `word` with the **namespace**, `strawberry` when we want to print `'strawberry'` .

We prepend `word` with the **namespace**, `banana` when we want to print `'banana'` .

The namespace indicates which module/package to import the variable/function etc from.

Changing the module name in the local namespace

In main.py, you can change the lines:

```
import strawberry
print(strawberry.word)
```

to

```
import strawberry as s
print(s.word)
```

Importing *individual items* from a module (by adding to the local namespace)

In main.py, you can change the lines:

```
import strawberry
print(strawberry.word)
```

to

```
from strawberry import word
print(word)
```

Importing *individual items* from a module - A word of warning!

A name can only have one associated value in a program.

Example: Importing two variables with the same name

```
from strawberry import word
from banana import word
```

Question: What will be the output of `print(word)` ?

Namespaces can be helpful - items (variables, functions) with the same name but from different modules can be used.

Importing *all contents* of a module

```
    from strawberry import *
    print(word)
```

Importing *all contents* of a module - A word of warning!

It is inadvisable to use `from ... import *` where you do not know the full content of a module
(e.g. a large module or a module written by a developer downloaded from the internet).

You may overshadow useful parts of your program such as built-in Python functions (`print` , `type` etc).

It may be appropraite to use * with a small, specific, user-defined module.

Summary

- **Module:** A python file containing python code (variables, functions, classes etc).
- **Package:** A file directory (folder) containing python files (and other directories).
- **Script:** A top level file, run as an program (importing would run the program).
- `__init__.py` **:** Required to make Python treat a directory as a package.

- When you import a package/module, the part after `import` should be used to prepend all variables, functions etc from the imported module, to use them in the current program.
- We can rename packages when they are imported.
- Individual variables, functions etc can be imported.

Renaming :

```
import fruit.strawberry as strawb # OR from fruit import strawberry as strawb
print(strawb.word)
```

Importing submodule:

```
from fruit import strawberry
print(strawberry.word)
```

Importing variable:

```
from fruit.strawberry import word
print(word)
```

Importing and rename variable

```
from fruit.strawberry import word as w
print(w)
```