

Introduction to Computer Programming

Week 6.1: Importing Python files



University of
BRISTOL

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 python **module** is then `import` ed for use in a python 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>)

In [2]:

```
1 import math
2
3 print(math.pi)
4
```

3.141592653589793

Module:

A python file containing python code - variables, functions, classes etc - (designed to be imported).

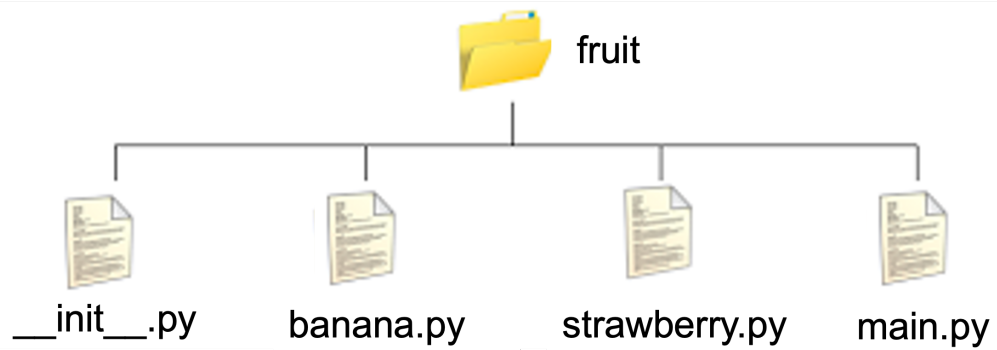
Package:

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

Script:

A top level file, run as an program (not designed to be imported).

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



`__init__.py` :

Required to make Python treat a directory as a package.

Can be empty or execute initialization code for a package.

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 (variables) of `banana.py` and `strawberry.py` are imported and can be used within the `main.py` program.

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.

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

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

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

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

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

`__init__.py`

```
# (empty file)
```

`banana.py`

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

`main.py`

```
import banana  
banana.peel()
```

When `banana` is imported, the function `peel` is imported.

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

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

`__init__.py`

```
# (empty file)
```

`banana.py`

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

`main.py`

```
import banana  
b = banana.Banana()  
b.peel()
```

When `banana` is imported, the class `Banana` is imported.

Changing the module name in the local namespace

We can change the name of the imported module e.g. to make it shorter:

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

Whatever comes after `import` is added to the local namespace

In main.py, you can change the lines:

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

to

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

Note: In this example `word` is now the only part of `strawberry` that has been imported.

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 <module name> 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 unknowingly reassign the functionality of a variable or function, effecting the behaviour of your program.

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

Example: Square Root

`math` and `cmath` modules contain a function named `sqrt`.

Both functions compute the square root of the input.

- `math.sqrt`, from the package, `math`, gives an error if the input is a negative number. It does not support complex numbers.
- `cmath.sqrt`, from the package, `cmath`, supports complex numbers.

In [6]:

```
1 from math import *  
2 from cmath import *  
3  
4 print(sqrt(4))  
5 print(sqrt(-1))
```

(2+0j)

1j

Summary

- **Module:** A python file containing python code (variables, functions, classes etc) (designed to be imported).
- **Package:** A file directory (folder) containing python files (and other directories).
- **Script:** A top level file, run as an program (not designed to be imported).
- `__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 objects (variables, functions etc) from the imported module, to use them in the current program.
- We can rename packages when they are imported.
- Individual objects (variables, functions etc) can be imported.

Importing module:

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

Renaming :

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

Importing object (e.g. variable, function, class...):

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

Importing and renaming object

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

In-class Demos

Try it yourself**Example 1a:**

Create the file structure shown below within a new folder called `lecture_6` .

Add the content shown within each file.

```
lecture_6/  
├── __init__.py  
├── capitals.py  
└── main.py
```

`__init__.py`

```
# (empty file)
```

`capitals.py`

```
Japan = ('Japan', 'Tokyo')  
Germany = ('Germany', 'Berlin')
```

`main.py`

```
# (empty file)
```

Example 1b:

Within `main.py` , print the output below:

The capital of Japan is Tokyo

F strings

(formatted string literals)

Strings that have:

- an f at the beginning
- curly braces containing expressions that are replaced with their values

```
In [7]: 1 A = 10
        2
        3 B = 6
        4
        5 print(f'I have', A, 'apples and', B, 'pears')
        6
        7 print(f'I have {A} apples and {B} pears')
```

I have 10 apples and 6 pears

I have 10 apples and 6 pears

Try it yourself

Example 1c:

Within `main.py`, print the output below:

The capital of Germany begins with B

Example 1d:

Can you make the code in `main.py` any more concise?