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 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]:
```
import math

print(math.pi)

```

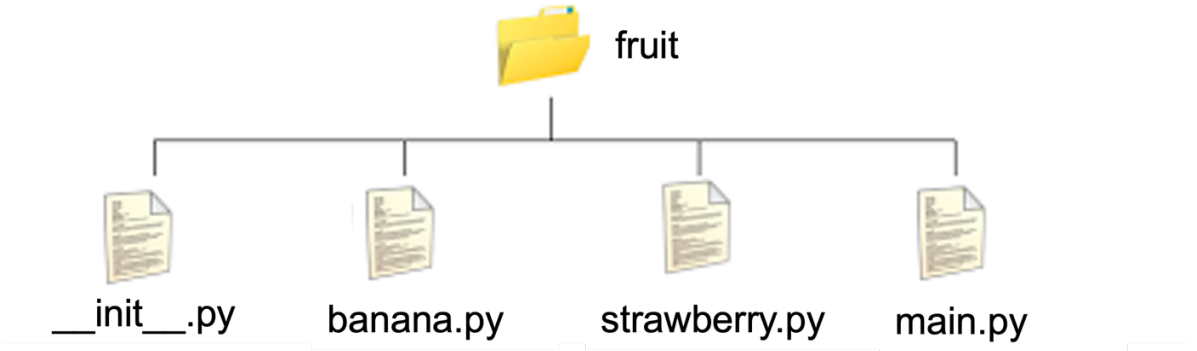
```
3.141592653589793
```

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 (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 **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.

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`

```
# 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`

```
# 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

Below are two functions, both 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 [7]:
```
from math import *
from cmath import *


print(sqrt(4))
print(sqrt(-1))
```

```
(2+0j)
1j
```

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 (not deigned to be imported, 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.

Importing module:

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

Renaming :

```
import strawberry as strawb # OR import strawberry as straw
b
print(strawb.word)
```

Importing variable:

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

Importing and rename variable

```
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`

Solution: Example 1b

main.py

```
import capitals

print(f'The capital of {capitals.Japan[0]} is {capitals.Jap
an[1]}')
```

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?

Solution: Example 1c

main.py

```
import capitals

print(f'The capital of {capitals.Germany[0]} begins with {c
apitals.Germany[1][0]}')
```

Solution: Example 1d

Changing the module name in the local namespace

main.py

```
import capitals as c

print(f'The capital of {c.Japan[0]} is {c.Japan[1]}')

print(f'The capital of {c.Germany[0]} begins with {c.German
y[1][0]}')
```

Solution: Example 1d

Importing *individual items* from a module

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

main.py

```
from capitals import Japan, Germany

print(f'The capital of {Japan[0]} is {Japan[1]}')

print(f'The capital of {Germany[0]} begins with {Germany[1]
[0]}')
```

Solution: Example 1d

Importing *individual items* from a module and renaming

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

```
main.py

   from capitals import Japan as J, Germany as G

   print(f'The capital of {J[0]} is {J[1]}')

   print(f'The capital of {G[0]} begins with {G[1][0]}')
```

In [ ]:

```
1
```