Introduction to Computer Programming

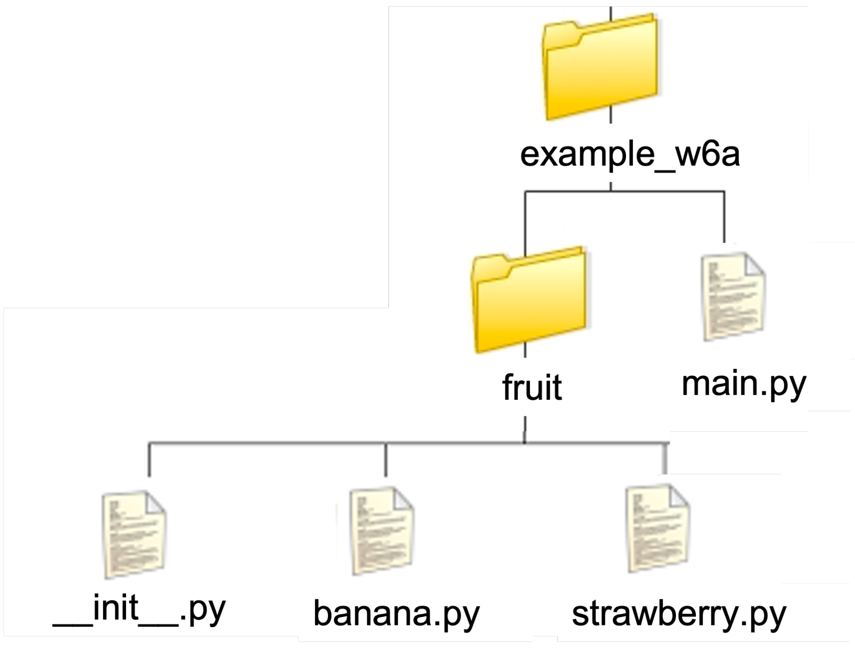
Week 6.2: Importing files from different locations

University of BRISTOL

Importing a file from a different directory

Downstream file location

Example: Importing from a downstream sub-directory (a **package**)



Example: Importing from a downstream sub-directory (a **package**)

```
example_w6a/
|
├── main.py
└── fruit/
    ├── __init__.py
    ├── banana.py
    └── strawberry.py
```

`.` is used to indicate a sub-directory downstream of the current location:

```
import subfolder.file
import folder.subfolder.file
```

Remember: Everything after `import` is stored in the local namespace and must be used to prepend any objects (variables, functions etc) from the imported module.

File contents

`main.py`

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

`fruit/___init__.py`

```
# (empty file)
```

`fruit/_banana.py`

```
word = 'banana'
```

`fruit/_strawberry.py`

```
word = 'strawberry'
```

The longer namespace when packages are imported can make code long and difficult to read.

Example Renaming `fruit.strawberry --> strawb` to make code shorter and neater

main.py

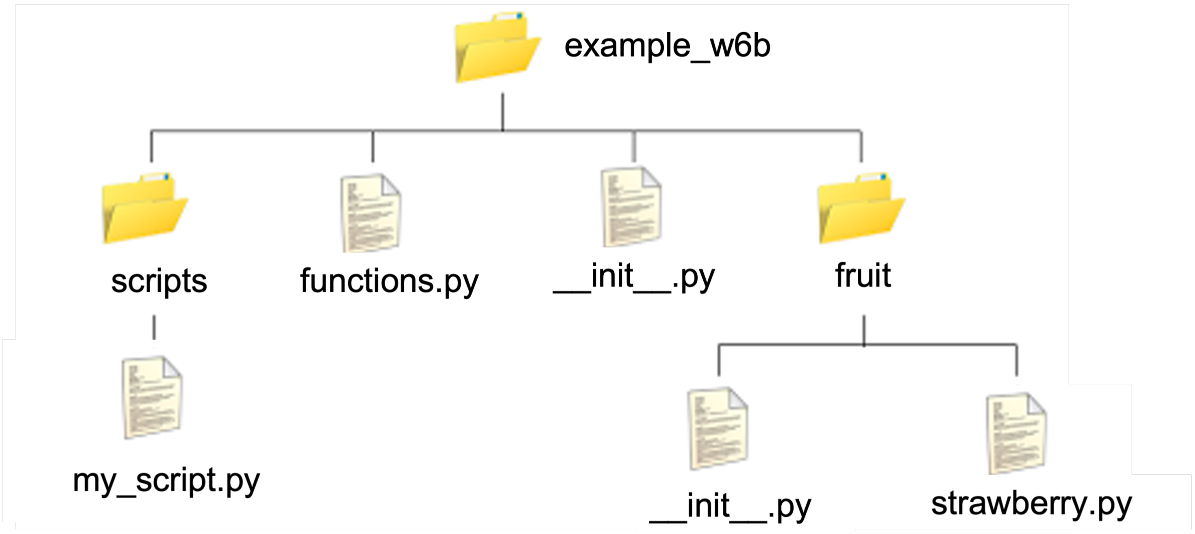
```
import fruit.strawberry as strawb
# OR
# from fruit import strawberry as strawb

print(strawb.word)
```

Upstream file location

Example: Importing from an upstream directory

If a file is located *upstream* of a script being run, it cannot automatically be found by the Python interpreter.



```
example_w6b/
 │
 ├── scripts/
 │   └── my_script.py
 │
 ├── fruit/
 │   ├── __init__.py
 │   └── strawberry.py
 │
 ├── functions.py
 └── __init__.py
```

This is because Python only looks for modules and packages in its **import path**.

This is a list of locations:

- current directory
- contents of `PYTHONPATH` variable (a list of user defined directories)
- standard directories automatically set up when python installs

To view the path we can use the `sys` module which installs with Python:

```
import sys

print(sys.path)
```

To add a *location* to the path from within a python program we can use `sys` .

`../` is used to indicate a location one directory upstream of the current location.

Example: In `my_script.py` :

```
import sys
sys.path.append('../')      # append path with directory on
e level up

import functions            # files from this location can
now be imported
import fruit.strawberry
```

`../` can be used in combination with directory names to form a path to the file to import.

Example: In `my_script.py` :

```
import sys
sys.path.append('../')       # append path with directory one level up
sys.path.append('../fruit') # append path with different directory at same level

import functions            # files from these locations can now be imported
import strawberry
```

```
example_w6b/
│
├── scripts/
│   └── my_script.py
│
├── fruit/
│   ├── __init__.py
│   └── strawberry.py
│
├── functions.py
└── __init__.py
```

A directory can be removed in a similar way:

```
sys.path.remove('../')
```

Summary

- Everything after `import` is stored in the local namespace and must be used to prepend any objects (variables, functions etc) from the imported module.
- Import from a **downstream** location by seperating directory names with `.` : `import folder.subfolder.file`
- If a file is located **upstream** of a script being run, it cannot automatically be found by the Python interpreter.
- Import from an uspstream location by using `sys` to add locations to the Python path

```
example_w6a/
|
├── main.py
└── fruit/
    ├── __init__.py
    ├── banana.py
    └── strawberry.py
```

In `main.py` ...

Importing submodule:

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

Renaming :

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

```
example_w6a/
|
├── main.py
└── fruit/
    ├── __init__.py
    ├── banana.py
    └── strawberry.py
```

In `main.py` ...

Importing variable:

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

Renaming

```
from fruit.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` .

```
example_w6b/
│
├── scripts/
│   └── my_script.py
│
├── fruit/
│   ├── __init__.py
│   └── strawberry.py
│
├── functions.py
└── __init__.py
```

Try it yourself

Example 1b:

Add the content shown:

`functions.py`

```
def letters(word):
    for w in word:
        print(w)
```

`strawberry.py`

```
word = 'strawberry'
```

Example 1c:

Within `my_script.py` , use the function `letters` to print the letters of the word 'Python' on seperate lines.

Solution: Example 1c

```
my_script.py

   import sys
   sys.path.append('../')

   import functions

   functions.letters('Python')
```

Try it yourself

Example 1d:

Within `my_script.py` , use the function `letters` to print the letters of the variable `word` imported from strawberry.py

Solution: Example 1c

Add directory one level up to path

```
my_script.py

   import sys
   sys.path.append('../')

   import functions
   import fruit.strawberry

   functions.letters(fruit.strawberry.word)
```

Add directory one level up and different directory to path

```
my_script.py
```

```
import sys
sys.path.append('../')
sys.path.append('../fruit')

import functions
import strawberry

functions.letters(strawberry.word)
```

Renaming and importing individual objects

`my_script.py`

```
import sys
sys.path.append('../')
sys.path.append('../fruit')

import functions as f
from strawberry import word

f.letters(word)
```

In [ ]: