When is a module not a module?

- it is often useful to create a module
  - for yourself and others to use in the future
  - to subdivide the large problem set into a number of smaller modules

- sometimes a module might be able to operate as a stand alone program
  - consider autoftp could be organised as a module

When is a module not a module?

- ```
  if __name__ == "__main__":
      main()
  ```

- which means run the function `main` if this module is explicitly invoked by the user
  - note that it is not run if this module was imported
  ```

Example times module

```
#!/usr/bin/python3

import sys

def multiplyby10(value):
    return value+"0"

if __name__ == "__main__":
    if len(sys.argv) == 2:
        print("testing the times module")
        print(multiplyby10(sys.argv[1]))
```

Example program

```
#!/usr/bin/python3

import times, sys

if len(sys.argv) == 2:
    print("importing the times module")
    print(times.multiplyby10(sys.argv[1]))
```

Example program

- note that the module times takes a string and adds a ’0’ to the left hand side
  - effectively multiply by 10

- note it also uses the `if __name__ ==` condition which only calls the multiply routine if this module was invoked as the main program by the user

Example program

```
./prog.py 12
importing the times module
120
```

```
./times.py 12
testing the times module
120
```

- exercise for the reader, add a function to perform divide and modulus of a numerical integer string

printf

- if any C programmer laments the lack of a `printf` function, you can roll your own:

- **mylibc.py**

```
#!/usr/bin/python3

import sys

#
#  printf - keeps C programmers happy :-)
#

def printf (format, *args):
    sys.stdout.write (str (format) % args)
    sys.stdout.flush ()
```

- please create this file (module) as it will be very useful when you start the coursework

printf

- **mytest.py**

```
#!/usr/bin/python3

from mylibc import printf

printf ("hello world\n")
printf ("an int: %d\n", 42)
printf ("a float: %f\n", 3.1415927)
```

- why does the output for a float differ from the constant value?

printf

mytest2.py

```
#!/usr/bin/python3

from mylibc import printf

printf ("hello world\n")
printf ("an int: %d\n", 42)
printf ("a float: %f\n", 3.1415927)

printf ("a float: %19.19f\n", 3.1415927)
```

Tutorial

- type in the printf example given during the lecture and check that it works

- create the file `mylibc.def` and also create the test programs
  - try running the test programs
  - you have created your first Python module `mylibc.def`

- try out any other examples from this weeks lecture notes