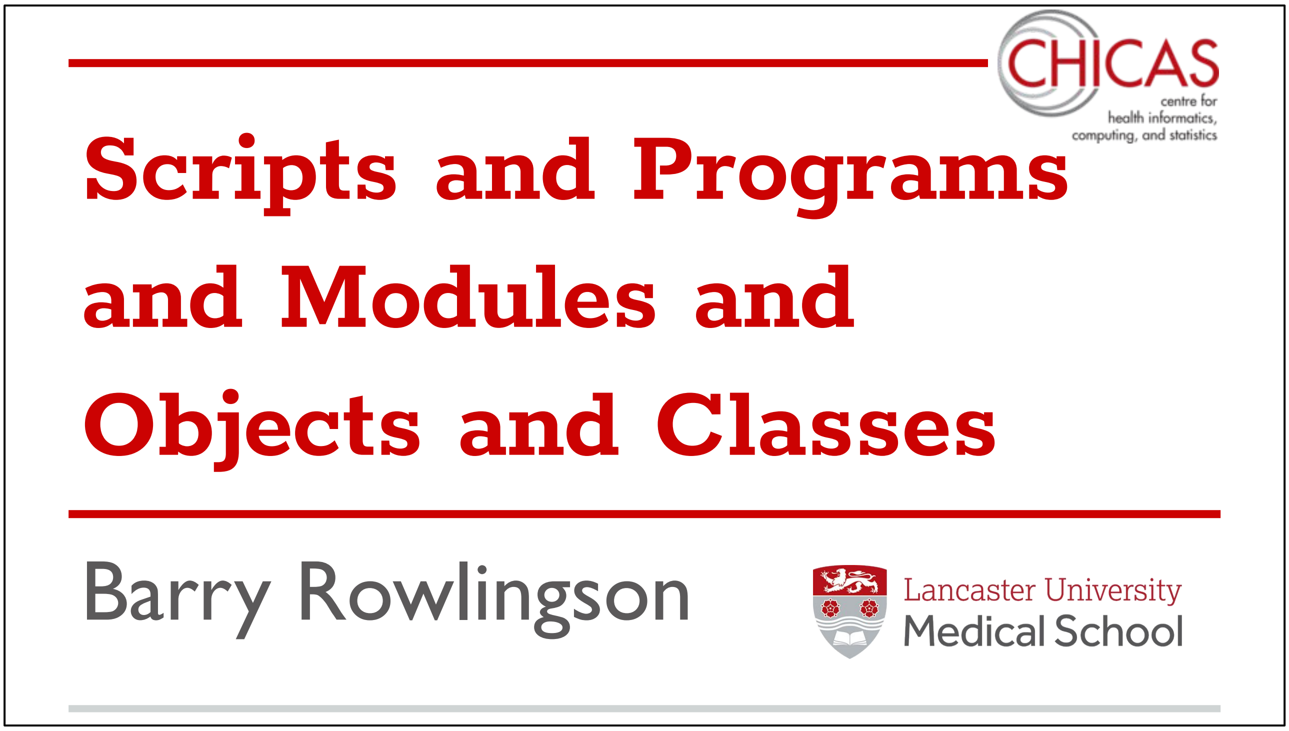


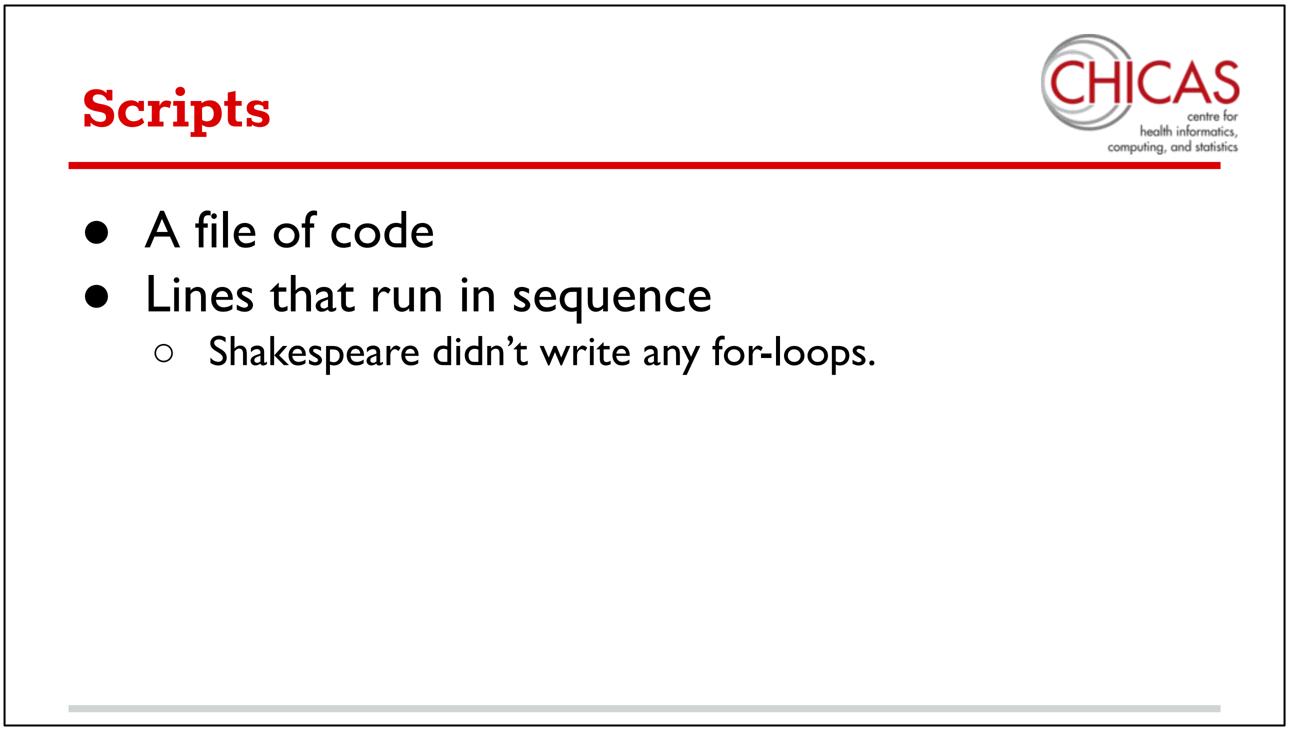
CHICAS
centre for health informatics, computing, and statistics

Scripts and Programs and Modules and Objects and Classes

Barry Rowlingson

Lancaster University Medical School

Last time we saw how to write functions in python. This time we'll expand on that to show how objects and classes work, since they are the main way of writing complex programs.



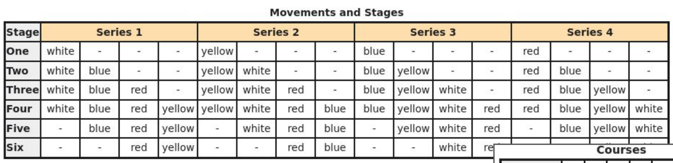
Scripts

- A file of code
- Lines that run in sequence
  - Shakespeare didn’t write any for-loops.

But first, what is a program? What is a script? I tend to use “script” for something that is a simple sequence of commands - in any programming language really - without any branching or looping. A bit like the script for a play.

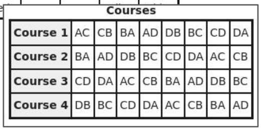
Scripts

- A file of code
- Lines that run in sequence
    - Shakespeare didn’t write any for-loops.
    - Although modern theatre...



Movements and Stages

Stage	Series 1				Series 2				Series 3				Series 4			
One	white	-	-	-	yellow	-	-	-	blue	-	-	-	red	-	-	-
Two	white	blue	-	-	yellow	white	-	-	blue	yellow	-	-	red	blue	-	-
Three	white	blue	red	-	yellow	white	red	-	blue	yellow	white	-	red	blue	yellow	-
Four	white	blue	red	yellow	yellow	white	red	blue	blue	yellow	white	red	red	blue	yellow	white
Five	-	blue	red	yellow	-	white	red	blue	-	yellow	white	red	-	blue	yellow	white
Six	-	-	red	yellow	-	-	red	blue	-	-	white	re	[illegible]	[illegible]	[illegible]	[illegible]



Courses

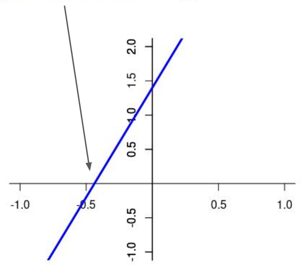
Course 1	AC	CB	BA	AD	DB	BC	CD	DA
Course 2	BA	AD	DB	BC	CD	DA	AC	CB
Course 3	CD	DA	AC	CB	BA	AD	DB	BC
Course 4	DB	BC	CD	DA	AC	CB	BA	AD

“Quad I” by Samuel Beckett

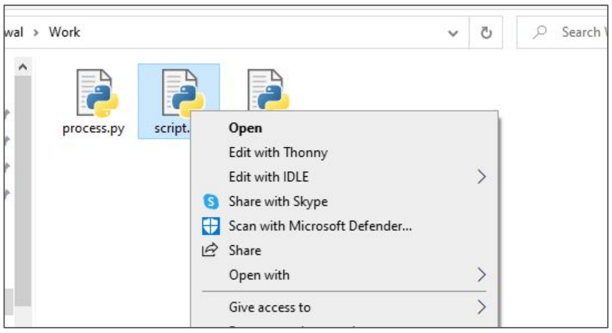
Except of course modern theatre has pieces that are more like a complex program...

Running A Script

```
gradient = 3.2
intercept = 1.4

x_zero = -intercept/gradient

print("X0 is ",x_zero)
```



Double-click/Open…



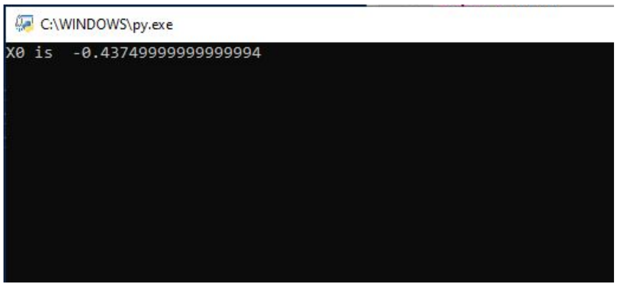
...dont blink!

So here’s a little script, its four lines and it solves a simple mathematical problem. Where does this line hit the X-axis? If I save that in a file in windows and double-click or Open it, it runs, but it opens a window and closes it immediately so I can’t see the result!

Wait...

```
gradient = 3.2
intercept = 1.4

x_zero = -intercept/gradient

print("X0 is ",x_zero)

input()
```

Waits for user input - press return

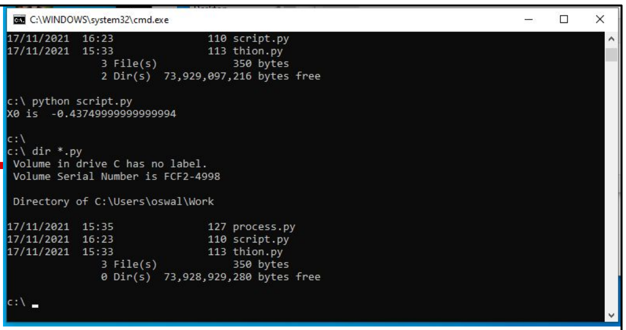


```
C:\WINDOWS\py.exe
X0 is  -0.43749999999999994
```

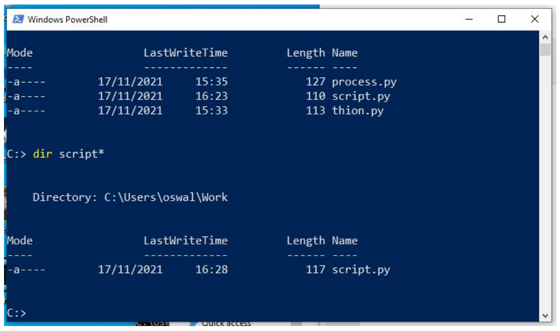
I could make the program pause by using the `input` function which accepts a text input typed by the user. If I do that I do see the result and I can press return so that the script finishes and the window closes.

Use a System Shell

- Windows CMD
- Windows Powershell
- Mac Terminal
- Linux Terminal


C:\WINDOWS\system32\cmd.exe

```
17/11/2021  16:23               110 script.py
17/11/2021  15:33               113 thion.py
               3 File(s)            350 bytes
               2 Dir(s)  73,929,097,216 bytes free

c:\ python script.py
X0 is  -0.43749999999999994

c:\
c:\ dir *.py
 Volume in drive C has no label.
 Volume Serial Number is FCF2-4998

 Directory of C:\Users\oswal\Work

17/11/2021  15:35               127 process.py
17/11/2021  16:23               110 script.py
17/11/2021  15:33               113 thion.py
               3 File(s)            350 bytes
               0 Dir(s)  73,928,929,280 bytes free

c:\ _
```

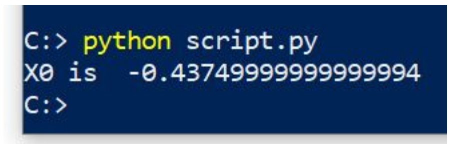

Windows PowerShell

```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        17/11/2021     15:35            127 process.py
-a----        17/11/2021     16:23            110 script.py
-a----        17/11/2021     15:33            113 thion.py


C:> dir script*


    Directory: C:\Users\oswal\Work


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        17/11/2021     16:28            117 script.py


C:>
```

But a better way is to run it from a system shell, or command line. In Windows you can use CMD or COMMAND or the “PowerShell”. Mac and Linux systems use a “Terminal” or “Console” program.

Run with Python

From a system shell or command line:



```
C:> python script.py
X0 is  -0.43749999999999994
C:>
```

And if I want to run that same script from the command line I give it as an argument to python. Then python reads the file, runs the code which prints the result, and comes back to the command prompt again. This script isn't very flexible though - can we make it do more?

Run with arguments

Imported the **`sys`** module...

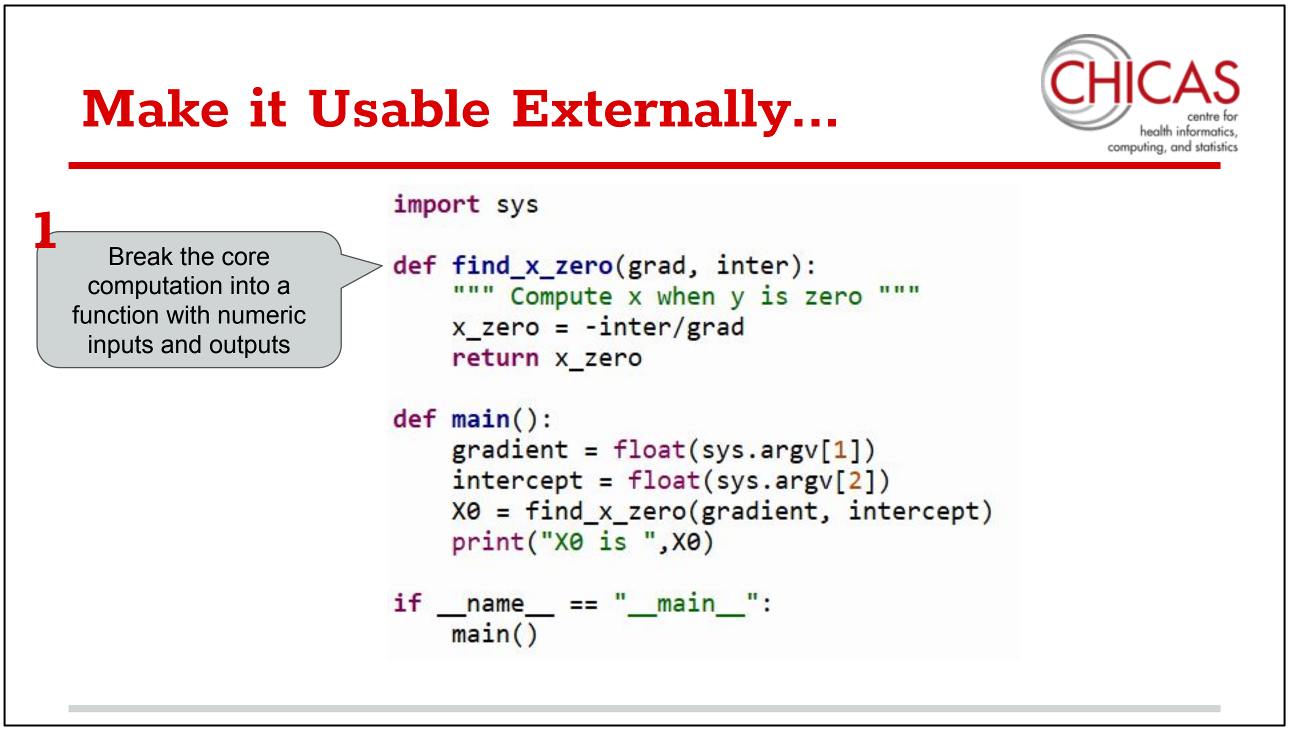
```python
import sys

gradient = float(sys.argv[1])
intercept = float(sys.argv[2])

x_zero = -intercept/gradient

print("X0 is ",x_zero)
```

Get two values from **`sys.argv`** and convert to floating point decimal...

...now we can run this with any two values for gradient and intercept.

```
C:> python script.py 3.2 1.4
X0 is  -0.43749999999999994
C:> python script.py 5.2 1.4
X0 is  -0.2692307692307692
```

Running a script with arguments is generally the next step in script development on the way to complex programs. The `sys` module gives you access to a list called `sys.argv`, which contains values given after the file name on the python command line. The values are always character text, so if we want numbers we have to convert them. Now I can run the same script with any given gradient and intercept and get a result printed.



Make it Usable Externally...

CHICAS centre for health informatics, computing, and statistics

1 Break the core computation into a function with numeric inputs and outputs

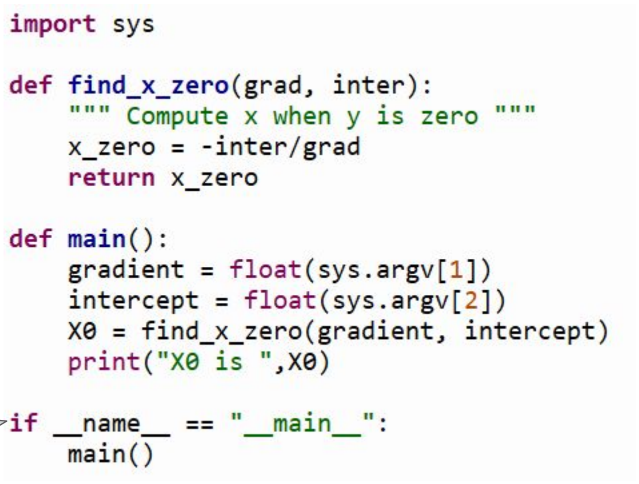
```
import sys

def find_x_zero(grad, inter):
    """ Compute x when y is zero """
    x_zero = -inter/grad
    return x_zero

def main():
    gradient = float(sys.argv[1])
    intercept = float(sys.argv[2])
    X0 = find_x_zero(gradient, intercept)
    print("X0 is ",X0)

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

The next level of complexity is making the code available from outside the script. First I'm taking the core computational process into a new function. This takes two numbers and returns the zero-crossing point.

Make it Usable Externally...

1 Break the core computation into a function with numeric inputs and outputs

2 This runs is `True` when run from the command line - when this is the “main” program:



```python
import sys

def find_x_zero(grad, inter):
    """ Compute x when y is zero """
    x_zero = -inter/grad
    return x_zero

def main():
    gradient = float(sys.argv[1])
    intercept = float(sys.argv[2])
    X0 = find_x_zero(gradient, intercept)
    print("X0 is ",X0)

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

Next is this bit of special python magic. This condition is True if the code is being run from a command line, or from a double-click “Open”.

Make it Usable Externally...

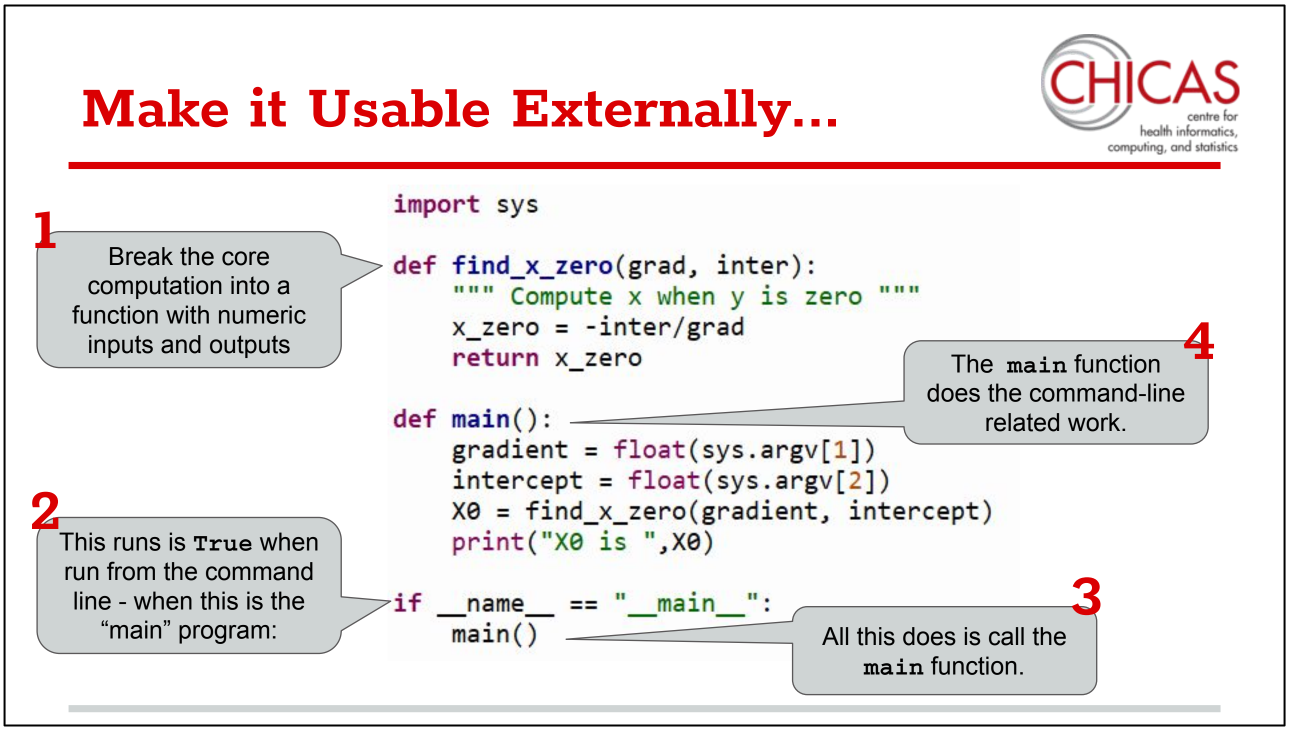
```python
import sys

def find_x_zero(grad, inter):
    """ Compute x when y is zero """
    x_zero = -inter/grad
    return x_zero

def main():
    gradient = float(sys.argv[1])
    intercept = float(sys.argv[2])
    X0 = find_x_zero(gradient, intercept)
    print("X0 is ",X0)

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

1 Break the core computation into a function with numeric inputs and outputs

2 This runs is `True` when run from the command line - when this is the “main” program:

3 All this does is call the `main` function.

Then if the program is being run as a main program, we call the main function.



Make it Usable Externally...

CHICAS centre for health informatics, computing, and statistics

```
import sys

def find_x_zero(grad, inter):
    """ Compute x when y is zero """
    x_zero = -inter/grad
    return x_zero

def main():
    gradient = float(sys.argv[1])
    intercept = float(sys.argv[2])
    X0 = find_x_zero(gradient, intercept)
    print("X0 is ",X0)

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

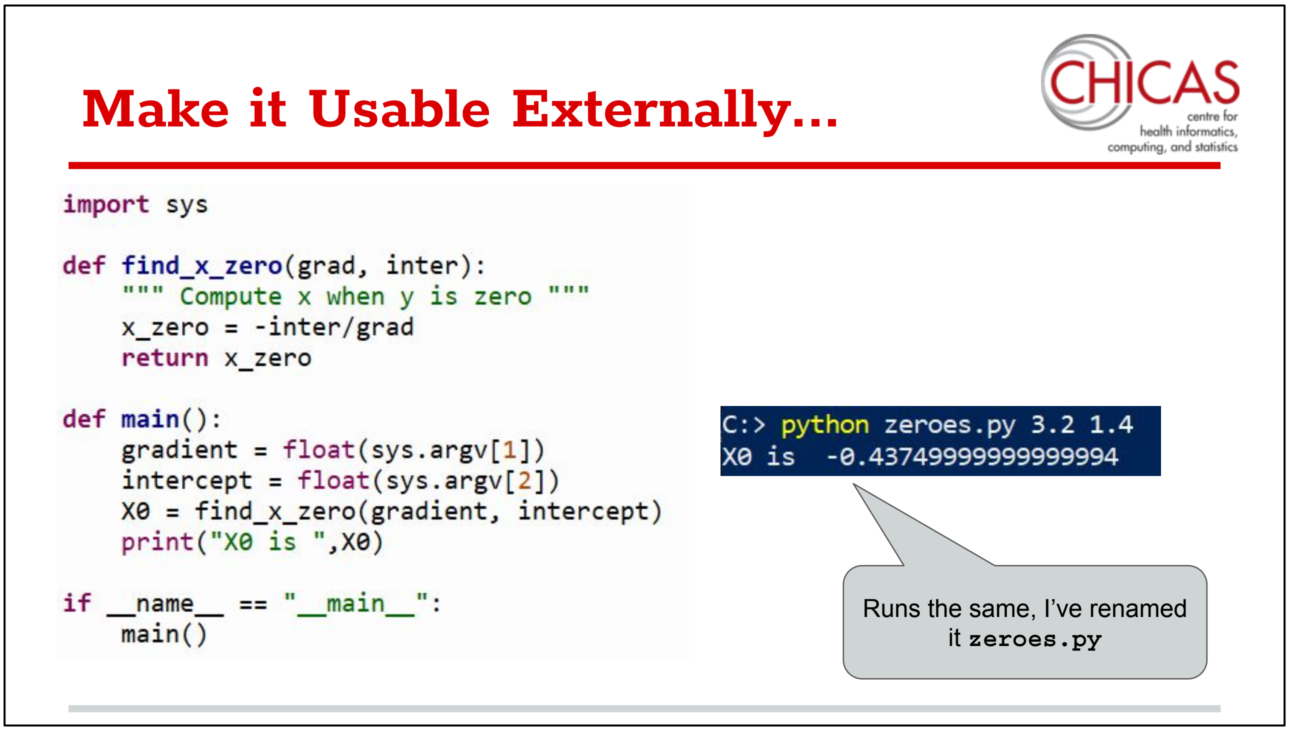
1 Break the core computation into a function with numeric inputs and outputs

2 This runs is `True` when run from the command line - when this is the “main” program:

3 All this does is call the `main` function.

4 The `main` function does the command-line related work.

The main function does all the work related to command-line processing. It gets `sys.argv` and calls the computational function and then prints the result.



Make it Usable Externally...

```python
import sys

def find_x_zero(grad, inter):
    """ Compute x when y is zero """
    x_zero = -inter/grad
    return x_zero

def main():
    gradient = float(sys.argv[1])
    intercept = float(sys.argv[2])
    X0 = find_x_zero(gradient, intercept)
    print("X0 is ",X0)

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

```
C:> python zeroes.py 3.2 1.4
X0 is  -0.43749999999999994
```

Runs the same, I've renamed it **`zeroes.py`**

Although there's more here, the program still runs the same as it did before - I can feed two arguments and get the zero crossing point.

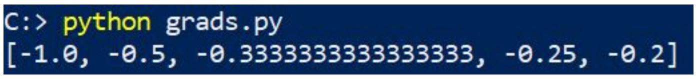
Using it from another file

grads.py

```
from zeroes import find_x_zero

gs = [1,2,3,4,5]

X0s = [find_x_zero(grad=g, inter=1) for g in gs]

print(X0s)
```

Looks in zeroes.py

Calls the function inside a list comprehension

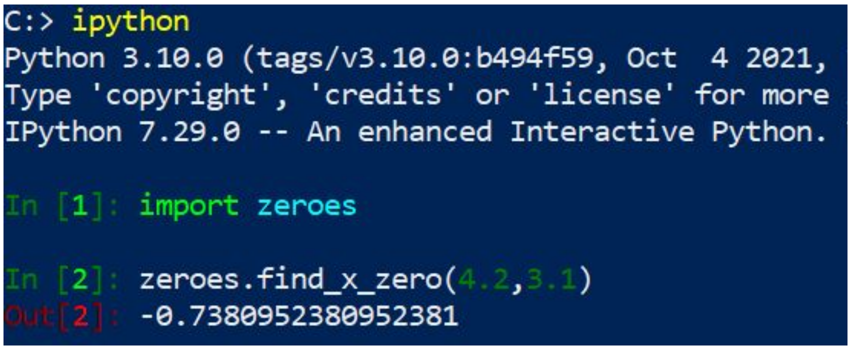


```
C:> python grads.py
[-1.0, -0.5, -0.3333333333333333, -0.25, -0.2]
```

But I can now *import* the computational function from another file. When imported, the "if __name__ == "__main__"" test is False, so the main() function doesn't run. But the file that imports it can now use the functions defined. So here is a new file that uses that function for a set of values defined in a list.

Using it interactively…

e.g. from an **`ipython`** session or a notebook



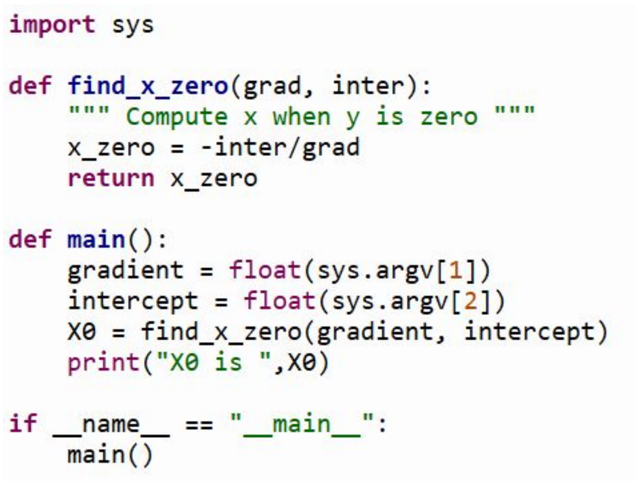
```
C:> ipython
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021,
Type 'copyright', 'credits' or 'license' for more
IPython 7.29.0 -- An enhanced Interactive Python.

In [1]: import zeroes

In [2]: zeroes.find_x_zero(4.2,3.1)
Out[2]: -0.7380952380952381
```

Similarly you can import from an interactive session and use it there. Here it is being imported and ran from an ipython session.

Its a module!

This has gone from being a fixed script to a fully-reusable Python module!

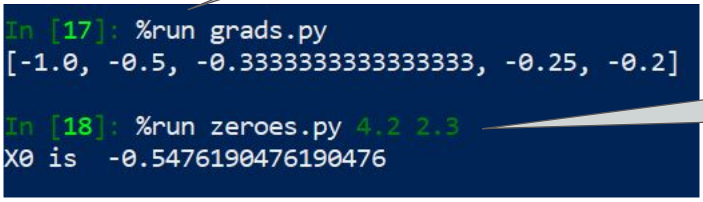
(Compare with writing packages in R…)



```python
import sys

def find_x_zero(grad, inter):
    """ Compute x when y is zero """
    x_zero = -inter/grad
    return x_zero

def main():
    gradient = float(sys.argv[1])
    intercept = float(sys.argv[2])
    X0 = find_x_zero(gradient, intercept)
    print("X0 is ",X0)

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

Running scripts from ipython

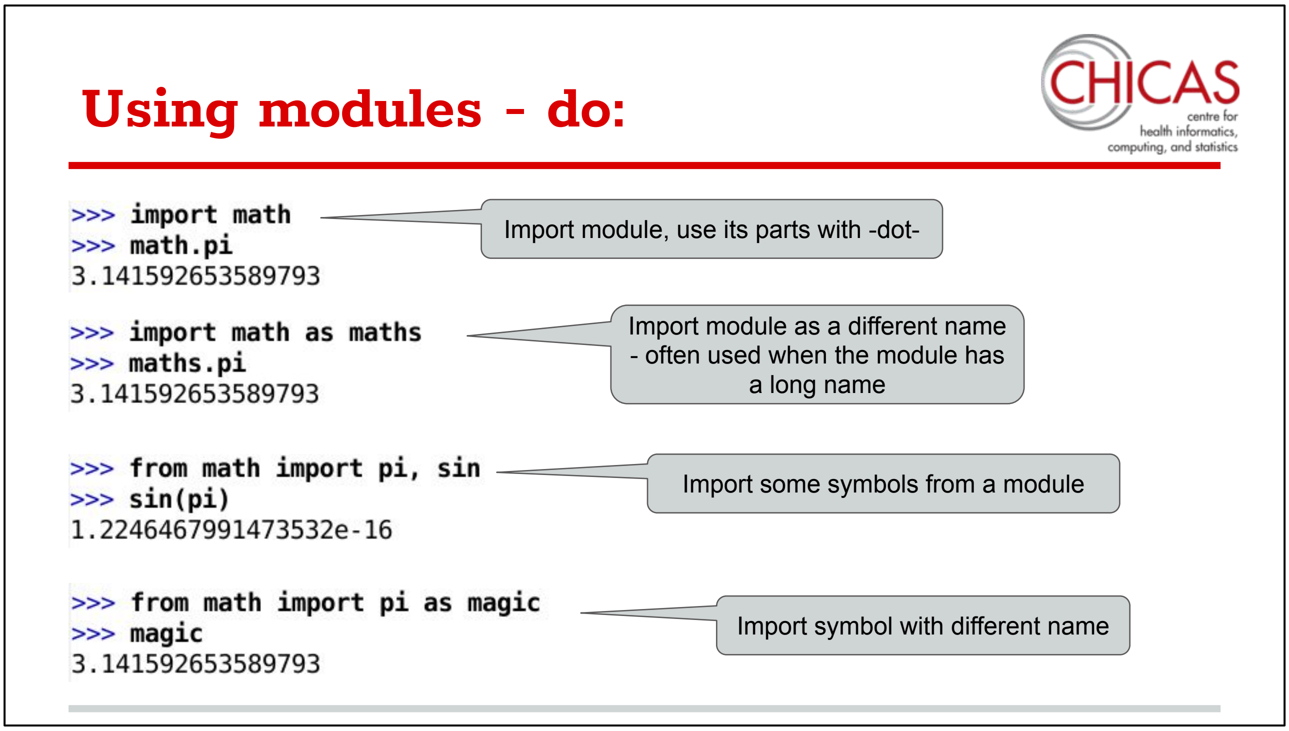
Things starting `%` are **`ipython`** "magic" functions. Not Python



```
In [17]: %run grads.py
[-1.0, -0.5, -0.3333333333333333, -0.25, -0.2]

In [18]: %run zeroes.py 4.2 2.3
X0 is  -0.5476190476190476
```

Can run Python scripts with argument without needing the system shell.

Note that if you are using the ipython interpreter you can run scripts as "main" programs using the %run "magic" function. You can pass arguments etc without needing a system shell.



Using modules - do:

```
>>> import math
>>> math.pi
3.141592653589793
```

Import module, use its parts with -dot-

```
>>> import math as maths
>>> maths.pi
3.141592653589793
```

Import module as a different name - often used when the module has a long name

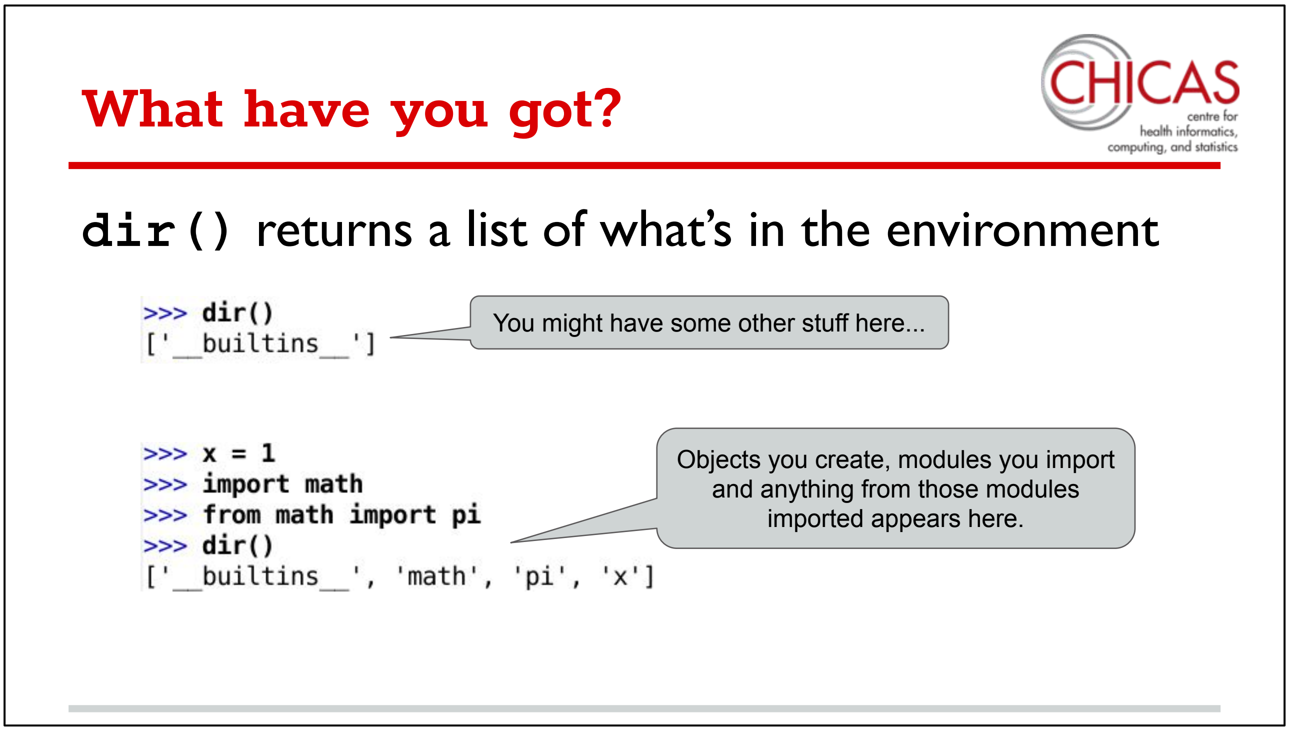
```
>>> from math import pi, sin
>>> sin(pi)
1.2246467991473532e-16
```

Import some symbols from a module

```
>>> from math import pi as magic
>>> magic
3.141592653589793
```

Import symbol with different name

Time for some more formalities on using modules. You can import a module and then use its parts with a dot. Or you can import a module *as* as different name and use its parts with that name. Useful if the name is long and saves typing, or if it clashes with the same name from another module. Or you can just import some symbols from a module, either as their own name, or renamed with “as” again. Note that all these methods allow you to track down any named object to its source. Every name is either part of a module.name combo or can be tracked back to its module import.



What have you got?

`dir()` returns a list of what’s in the environment

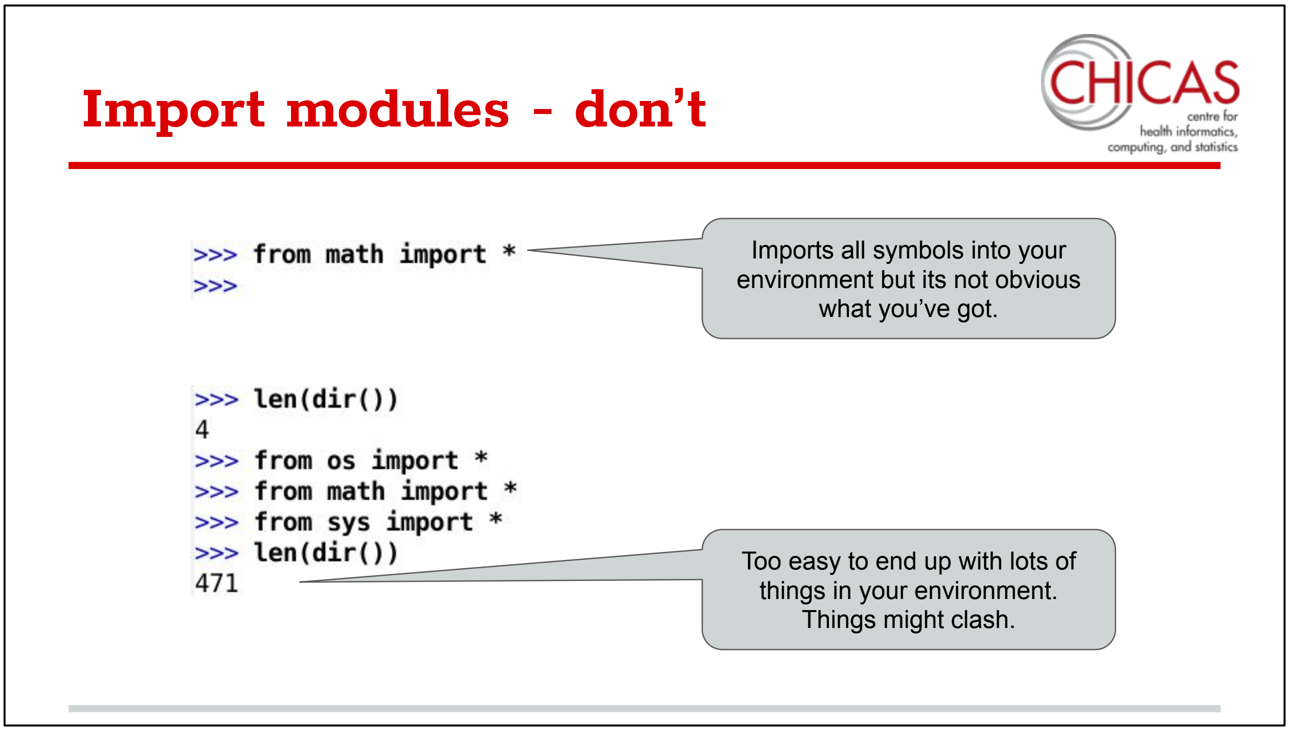
```
>>> dir()
['__builtins__']
```

You might have some other stuff here...

```
>>> x = 1
>>> import math
>>> from math import pi
>>> dir()
['__builtins__', 'math', 'pi', 'x']
```

Objects you create, modules you import and anything from those modules imported appears here.

There’s a useful `dir` function that is a bit like R’s `ls()` function to tell you what’s in some environment. Python keeps a few important things in the environment, but when you create or import things you can see them with `dir()`.



Import modules - don't

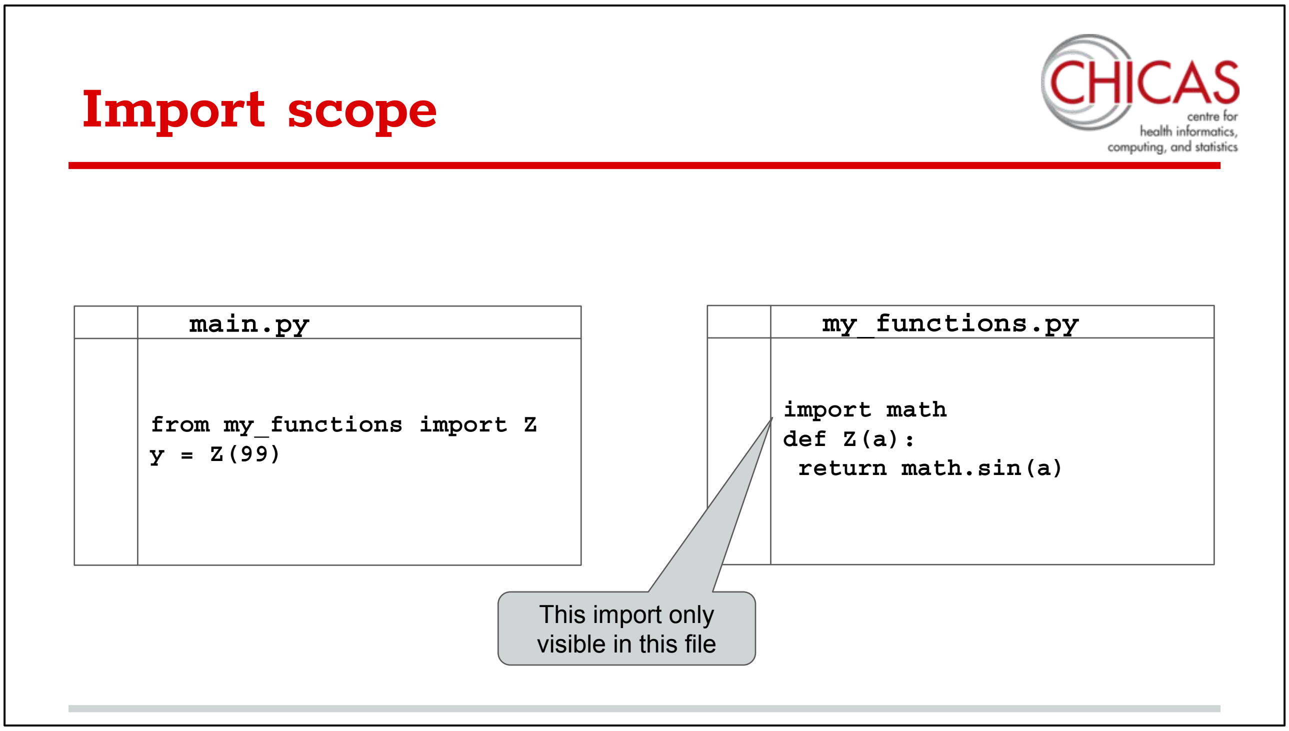
```
>>> from math import *
>>>
```

Imports all symbols into your environment but its not obvious what you've got.

```
>>> len(dir())
4
>>> from os import *
>>> from math import *
>>> from sys import *
>>> len(dir())
471
```

Too easy to end up with lots of things in your environment. Things might clash.

One thing you shouldn't ever do is "import *" - this brings in all the symbols from a module. Not only can this clutter your namespace with potentially hundreds of symbols, but it also breaks the back-tracking so you can't tell where a name has come from, and this can make fixing bugs tricky.



Import scope

main.py

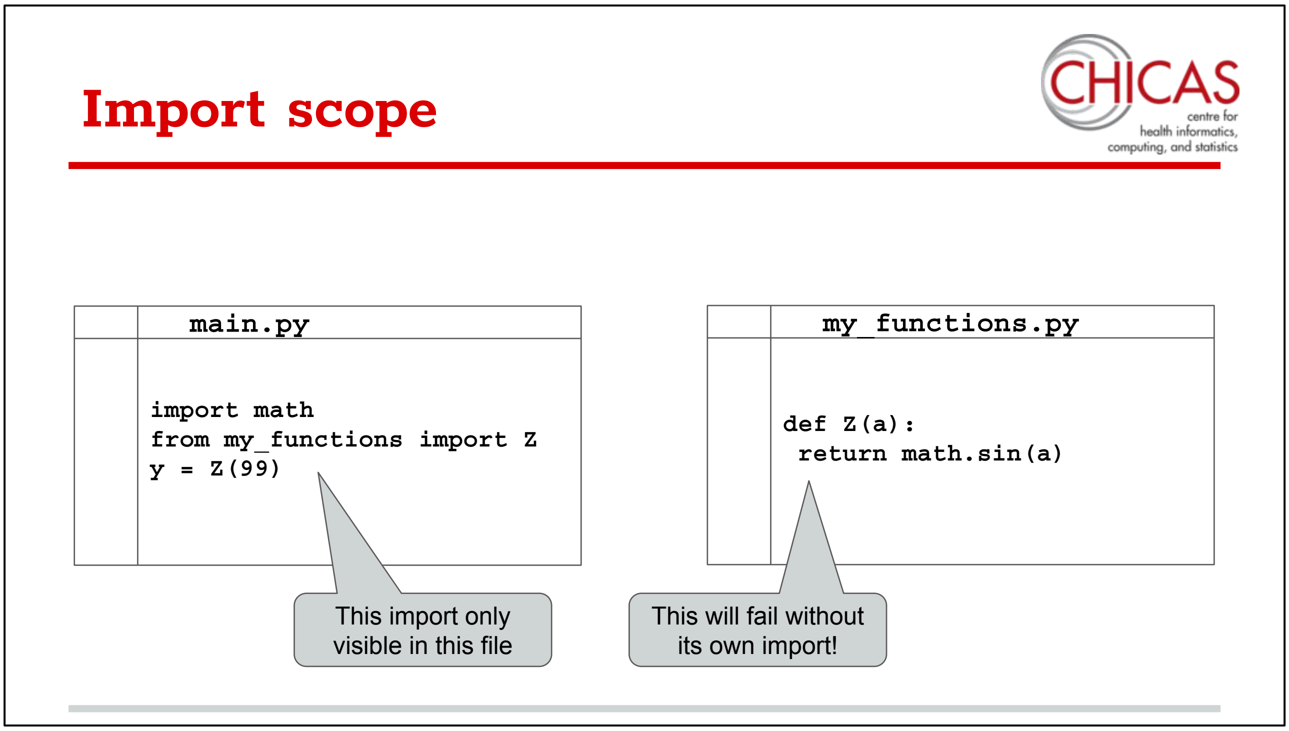
```
from my_functions import Z
y = Z(99)
```

my_functions.py

```
import math
def Z(a):
 return math.sin(a)
```

This import only visible in this file

Another difference between R’s packages and Python’s modules is where the package loading is valid. In python imports have “module scope” - they are only visible with the module that has it.



Import scope

main.py

```
import math
from my_functions import Z
y = Z(99)
```

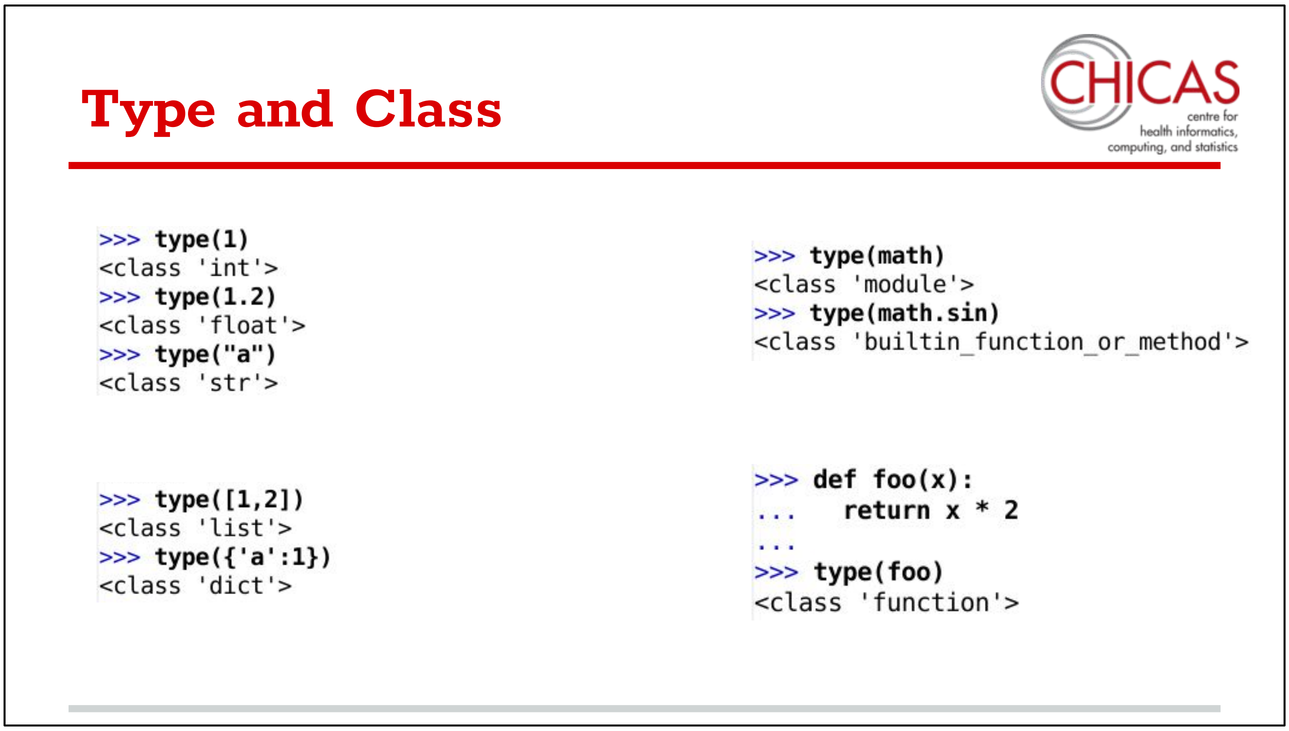
This import only visible in this file

my_functions.py

```
def Z(a):
 return math.sin(a)
```

This will fail without its own import!

If you import math over here, then the other function will fail. This is unlike R, where library(pkg) adds the package to the search list for all R code in that session. Which means sometimes its impossible to know which R function is actually getting called….



Type and Class

CHICAS centre for health informatics, computing, and statistics

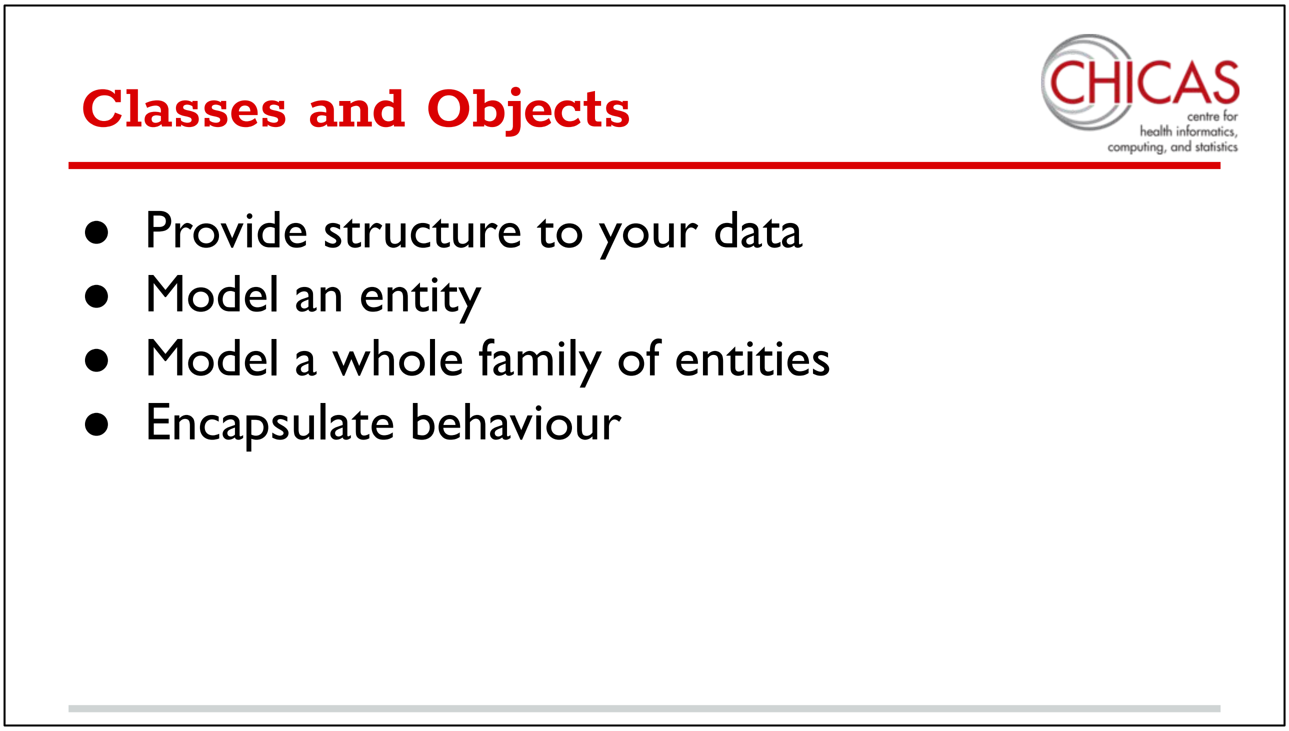
```
>>> type(1)
<class 'int'>
>>> type(1.2)
<class 'float'>
>>> type("a")
<class 'str'>
```

```
>>> type(math)
<class 'module'>
>>> type(math.sin)
<class 'builtin_function_or_method'>
```

```
>>> type([1,2])
<class 'list'>
>>> type({'a':1})
<class 'dict'>
```

```
>>> def foo(x):
...     return x * 2
...
>>> type(foo)
<class 'function'>
```

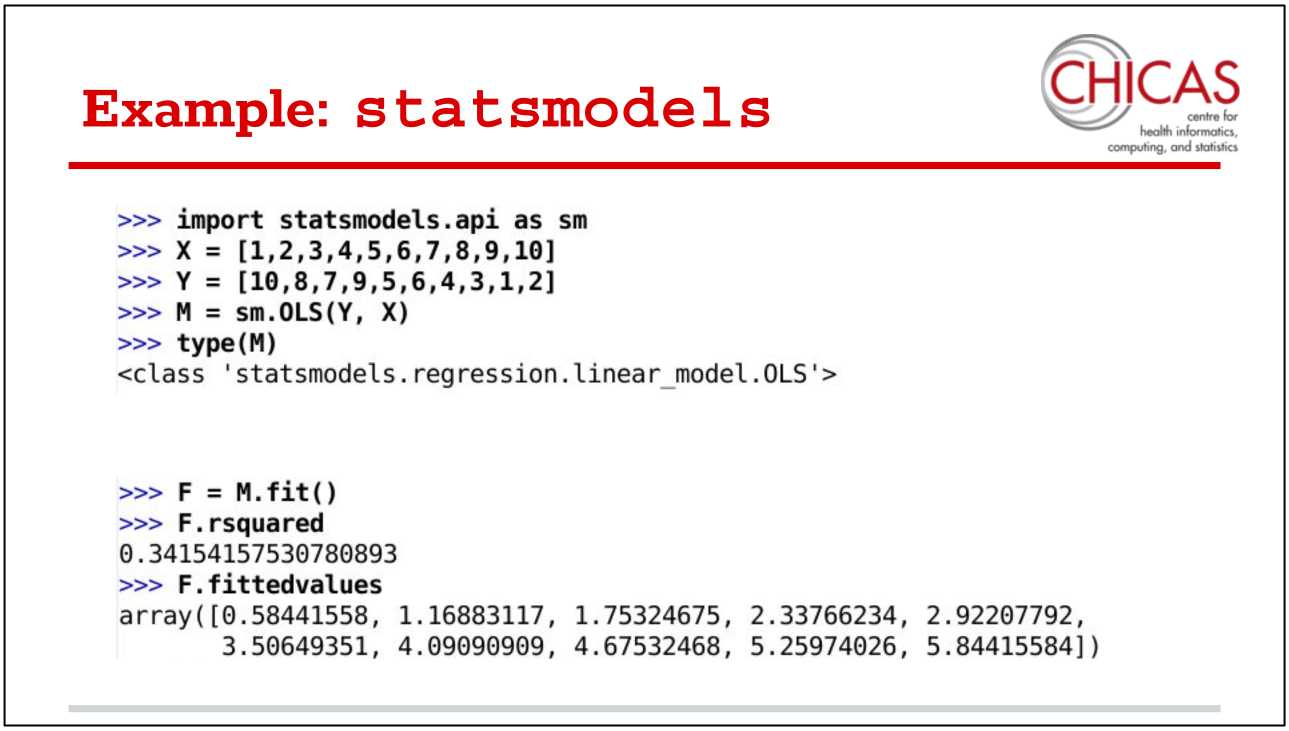
Next we need to look at the different types of data we get in Python. The words “type” and “class” are used interchangeably, and in Python version 3 are the same thing. In version 2 there were separate “types” and “classes” but that’s all in the past now. To find out what sort of thing an object is, run the `type()` function on it. Here’s some of the classes we’ve seen already.



Classes and Objects

- Provide structure to your data
- Model an entity
- Model a whole family of entities
- Encapsulate behaviour

In python the key to building systems is the definition and use of objects. We use them to provide structure to data, and to add behaviour to data, such that the behaviour is closely coupled to teh data. There's also the ability to model whole families and structures of data to avoid replicating code across similar things.



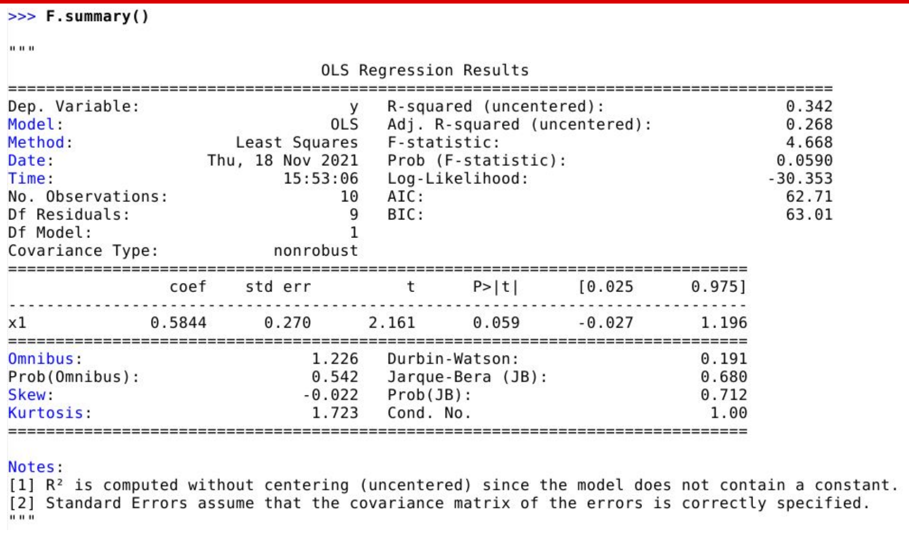
Example: `statsmodels`

centre for health informatics, computing, and statistics

```
>>> import statsmodels.api as sm
>>> X = [1,2,3,4,5,6,7,8,9,10]
>>> Y = [10,8,7,9,5,6,4,3,1,2]
>>> M = sm.OLS(Y, X)
>>> type(M)
<class 'statsmodels.regression.linear_model.OLS'>


>>> F = M.fit()
>>> F.rsquared
0.34154157530780893
>>> F.fittedvalues
array([0.58441558, 1.16883117, 1.75324675, 2.33766234, 2.92207792,
       3.50649351, 4.09090909, 4.67532468, 5.25974026, 5.84415584])
```

Here’s a typical example. The `statsmodels` package is an add-on that can fit simple (and slightly less simple) statistical models. From reading the docs you can discover that to fit a simple ordinary least-squares model you need statsmodels.api.OLS, and conventionally you’d import statsmodels.api as sm and then you can call `sm.OLS` with the Y and X values from a list.
What you get back is a new type, in this case a statsmodel.regression.linear_model.OLS object. This has a “fit” method which returns the parameters of the fit such as the fitted values and the R-squared value.

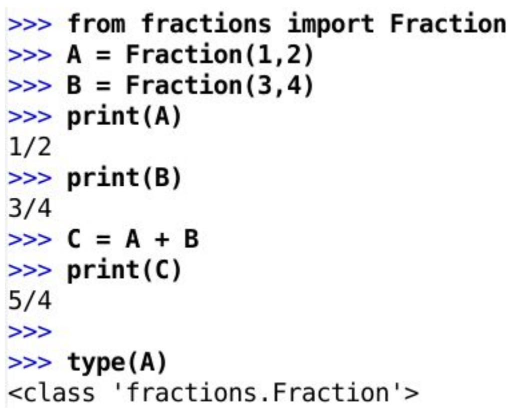
Fit summary



```
>>> F.summary()

"""
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.342
Model:                            OLS   Adj. R-squared (uncentered):              0.268
Method:                 Least Squares   F-statistic:                              4.668
Date:                Thu, 18 Nov 2021   Prob (F-statistic):                      0.0590
Time:                        15:53:06   Log-Likelihood:                         -30.353
No. Observations:                  10   AIC:                                      62.71
Df Residuals:                       9   BIC:                                      63.01
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.5844      0.270      2.161      0.059      -0.027       1.196
==============================================================================
Omnibus:                        1.226   Durbin-Watson:                   0.191
Prob(Omnibus):                  0.542   Jarque-Bera (JB):                0.680
Skew:                          -0.022   Prob(JB):                        0.712
Kurtosis:                       1.723   Cond. No.                         1.00
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
"""
```

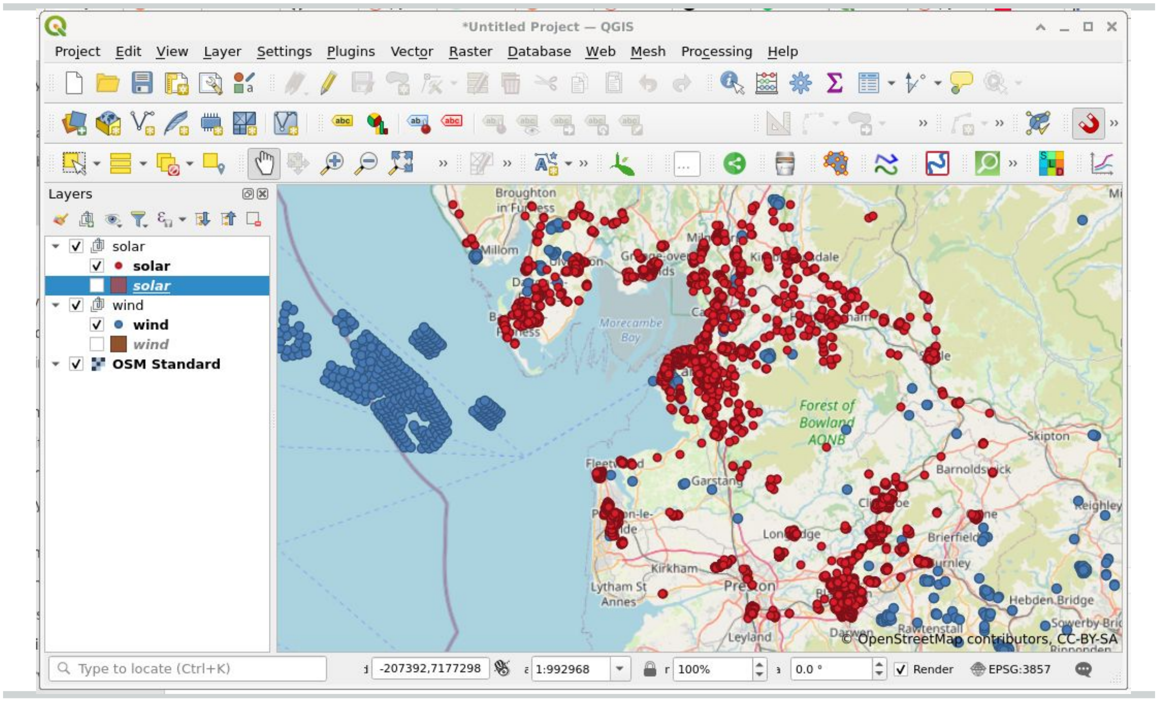
That fit is an object of another class and has its own methods, such as summary(), which prints out the same sort of statistical info that R's summary does on a linear model output.

More types

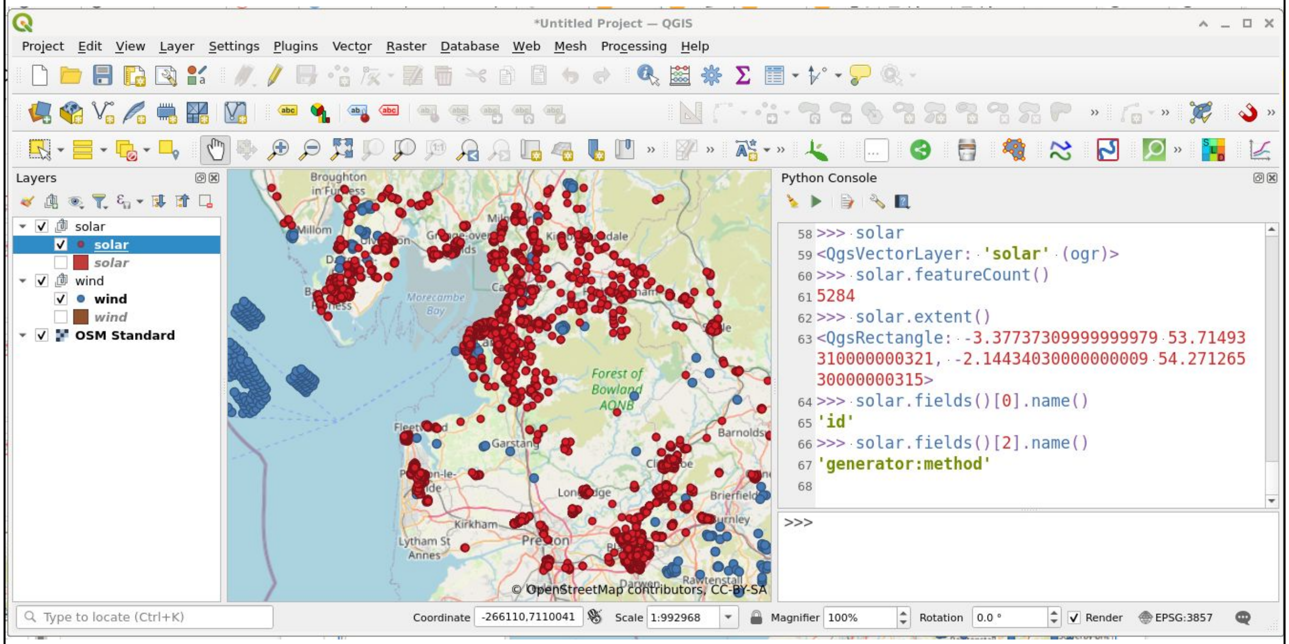


```
>>> from fractions import Fraction
>>> A = Fraction(1,2)
>>> B = Fraction(3,4)
>>> print(A)
1/2
>>> print(B)
3/4
>>> C = A + B
>>> print(C)
5/4
>>>
>>> type(A)
<class 'fractions.Fraction'>
```

As another example, if you want to work with fractions, you can import the Fraction function from the built-in fractions module, and then define Fraction objects, and do arithmetic with them and preserve their fractional nature. Useful if you want to work out what (3/16) + (2/3) is without resorting to your school maths books (its 41/48).



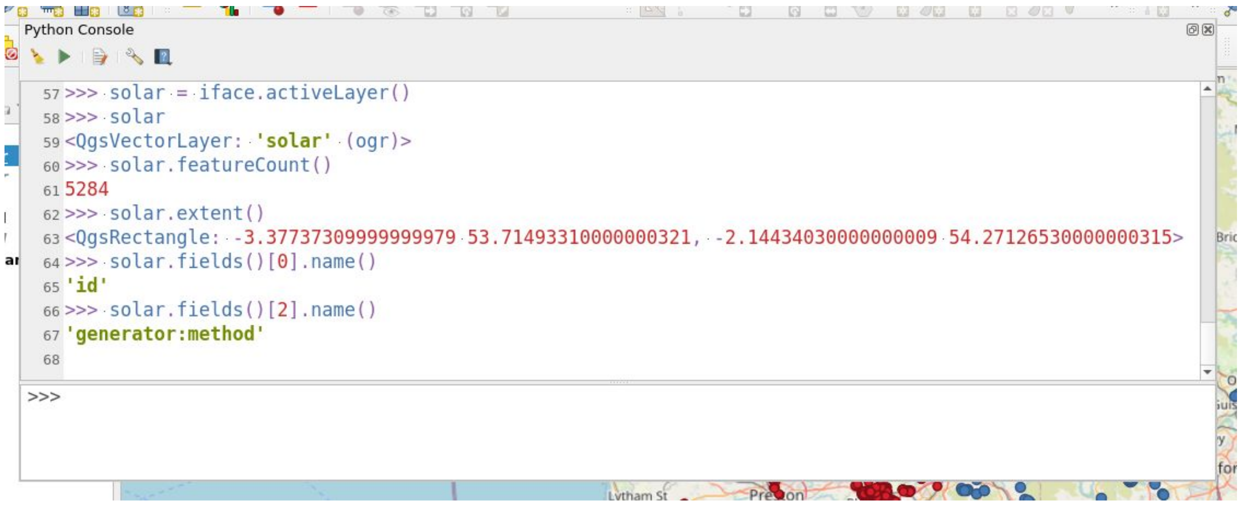
As an illustration of how you can “build big” with objects and classes, this is QGIS, an open-source geographic information system, used for making maps and analysing spatial data. Here’s a background map layer with some solar power points in red and some wind power points in blue.



```
>>> solar
<QgsVectorLayer: 'solar' (ogr)>
>>> solar.featureCount()
5284
>>> solar.extent()
<QgsRectangle: -3.37737309999999979 53.71493
310000000321, -2.14434030000000009 54.271265
30000000315>
>>> solar.fields()[0].name()
'id'
>>> solar.fields()[2].name()
'generator:method'

>>>
```

Coordinate -266110,7110041 Scale 1:992968 Magnifier 100% Rotation 0.0 °

QGIS has embedded in it a python interpreter that you can open and get a python command line...



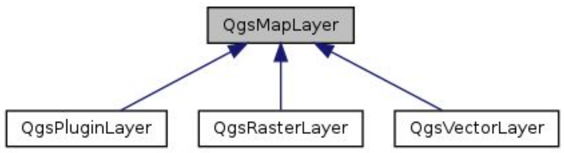
Python Console

```
>>> solar = iface.activeLayer()
>>> solar
<QgsVectorLayer: 'solar' (ogr)>
>>> solar.featureCount()
5284
>>> solar.extent()
<QgsRectangle: -3.37737309999999979 53.71493310000000321, -2.14434030000000009 54.27126530000000315>
>>> solar.fields()[0].name()
'id'
>>> solar.fields()[2].name()
'generator:method'

```

```
>>>
```

And everything in the application appears as Python objects here. For example I can get a list of the data layers, and then run methods on those layers to get things like the number of points (the “feature count”) or the extremes of latitude and longitude - the “extent” - as another class, this time a QgsRectangle. I can also get more details on each point like the ID and the method of generation.

Documentation

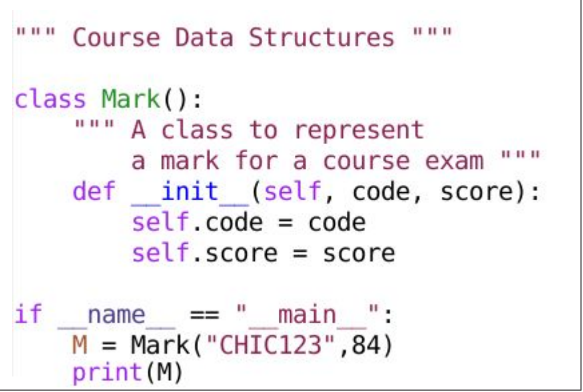


	This is the method that does the actual work of drawing the layer
virtual void	drawLabels (QgsRenderContext &rendererContext)
	Draw labels.
virtual QgsRectangle	extent ()
	Return the extent of the layer.
bool	isValid ()
QString	publicSource () const
const QString &	source () const
	Returns the source for the layer

This is all documented for all the classes that QGIS supports, and there are diagrams that show how classes relate to each other. For example anything that can appear on the map is a sub-class of “QgsMapLayer”. All layers have an “extent” so this is documented here. Other types of layers will have these methods (more are there, not shown) and their own specific methods for their class.

Defining Classes

	course1.py



```
""" Course Data Structures """

class Mark():
    """ A class to represent
        a mark for a course exam """
    def __init__(self, code, score):
        self.code = code
        self.score = score

if __name__ == "__main__":
    M = Mark("CHIC123",84)
    print(M)
```

Let’s see how we can define our own classes (or “types”). We’ll do this in a new file (although you can do this interactively at a Python command line prompt…)

Defining Classes

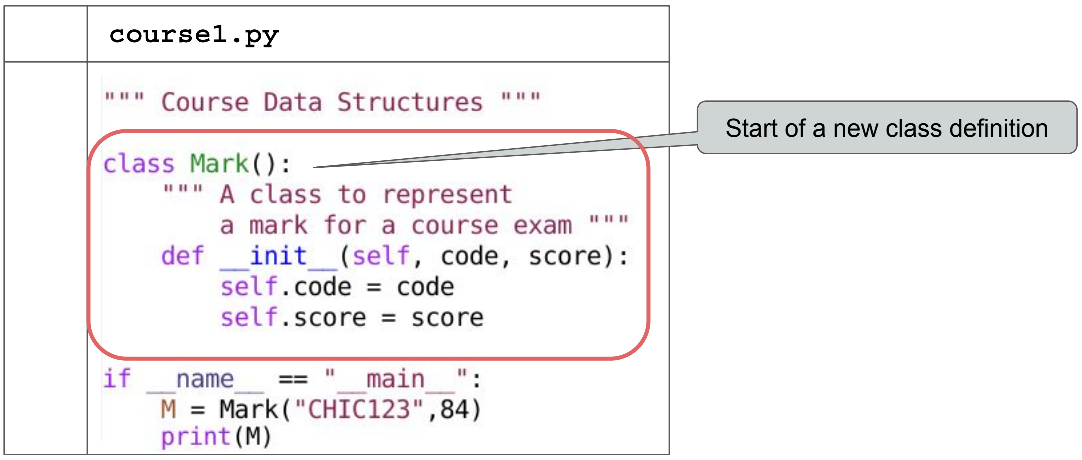
course1.py

```
""" Course Data Structures """

class Mark():
    """ A class to represent
        a mark for a course exam """
    def __init__(self, code, score):
        self.code = code
        self.score = score

if __name__ == "__main__":
    M = Mark("CHIC123",84)
    print(M)
```

Module docstring

So this is a new module, and you can start with a documentation string that will appear if you do help(course1).

Defining Classes

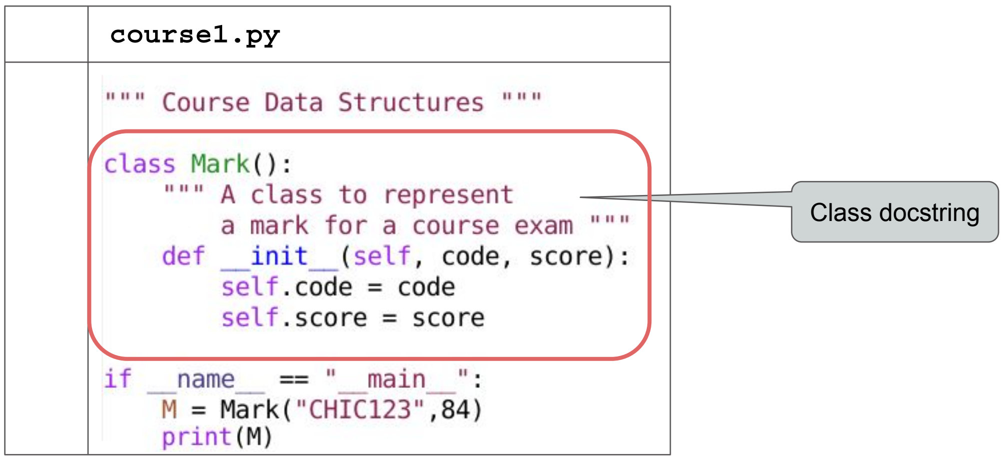


course1.py

```
""" Course Data Structures """

class Mark():
    """ A class to represent
        a mark for a course exam """
    def __init__(self, code, score):
        self.code = code
        self.score = score

if __name__ == "__main__":
    M = Mark("CHIC123",84)
    print(M)
```

A class definition is introduces with the “class” keyword, and conventionally classes start with a capital letter, but its not compulsory. The indented sections is then the class definition.

Defining Classes

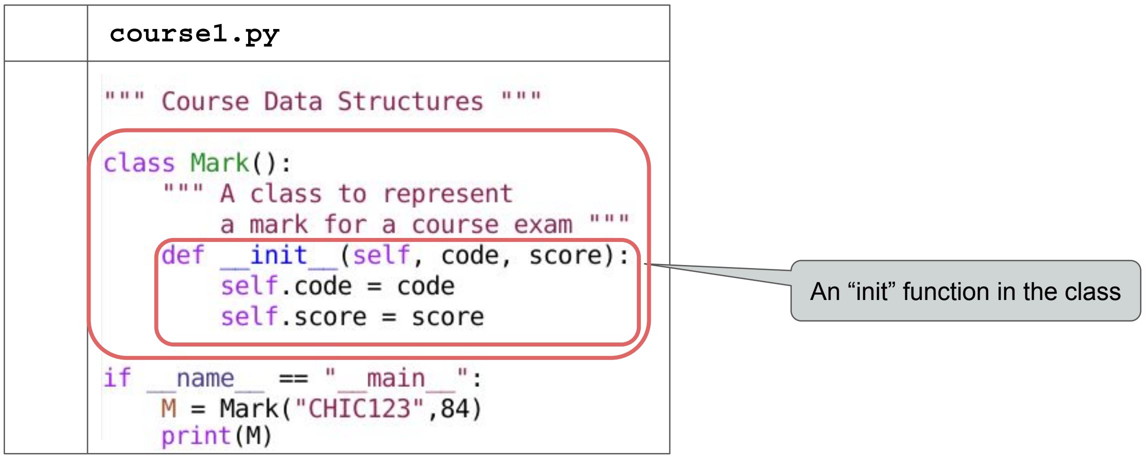


course1.py

```
""" Course Data Structures """

class Mark():
    """ A class to represent
        a mark for a course exam """
    def __init__(self, code, score):
        self.code = code
        self.score = score

if __name__ == "__main__":
    M = Mark("CHIC123",84)
    print(M)
```

Within this there’s a docstring that appears on help(course1.Mark)

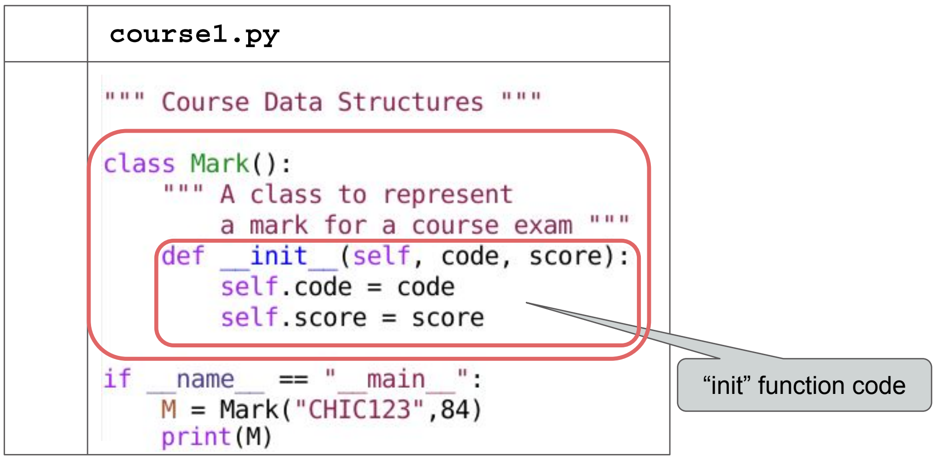
Defining Classes


course1.py

```
""" Course Data Structures """

class Mark():
    """ A class to represent
        a mark for a course exam """
    def __init__(self, code, score):
        self.code = code
        self.score = score

if __name__ == "__main__":
    M = Mark("CHIC123",84)
    print(M)
```

Then we define a function that is called when one of these objects gets created, it has to be called __init__ and is referred to as “the init function” or “the dunder-init” function or sometimes “the constructor”.

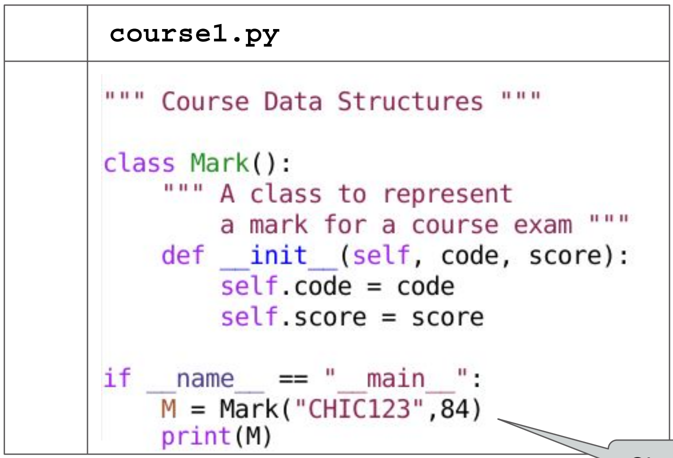
Defining Classes


course1.py

```
""" Course Data Structures """

class Mark():
    """ A class to represent
        a mark for a course exam """
    def __init__(self, code, score):
        self.code = code
        self.score = score

if __name__ == "__main__":
    M = Mark("CHIC123",84)
    print(M)
```

The init function has at least one argument, “self”, which is going to be the object we are constructing. Other arguments can be passed in to build the object. In this case we pass in a course code and an exam score, and store them as attribute of the “self” object using dot-notation.

Defining Classes



course1.py

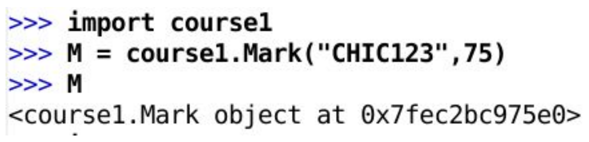
```python
""" Course Data Structures """

class Mark():
    """ A class to represent
        a mark for a course exam """
    def __init__(self, code, score):
        self.code = code
        self.score = score

if __name__ == "__main__":
    M = Mark("CHIC123",84)
    print(M)
```

Simple test for now

Its also a good idea to put a little test in the “main” block of a module just to check it works as expected. This can always be expanded later.

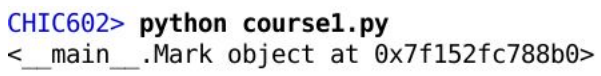
Running

Can import and test:



```
>>> import course1
>>> M = course1.Mark("CHIC123",75)
>>> M
<course1.Mark object at 0x7fec2bc975e0>
```

or run from command line:



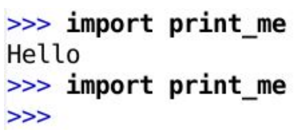
```
CHIC602> python course1.py
<__main__.Mark object at 0x7f152fc788b0>
```

So now I can import that and create new course Mark objects, or I can run the little test from the command line.

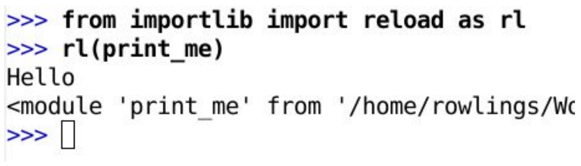
Reloading

- When developing interactively, a second import doesn't change anything!

```
print_me.py

# print "Hello" to the screen
print("Hello")
```



```
>>> import print_me
Hello
>>> import print_me
>>>
```

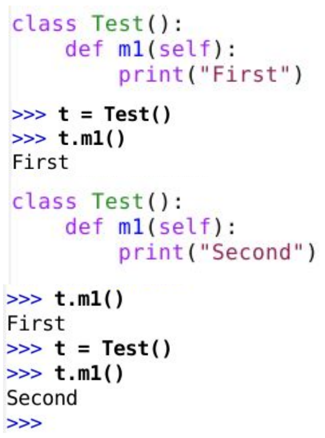


```
>>> from importlib import reload as rl
>>> rl(print_me)
Hello
<module 'print_me' from '/home/rowlings/W
>>> 
```

A couple of points that can catch new users: first if you import something twice, nothing happens the second time! So if you are editing files and need to do a second import, you actually have to use the "reload" function from the "importlib" module.

Gotcha

- Objects use the class definition at the time they were created...

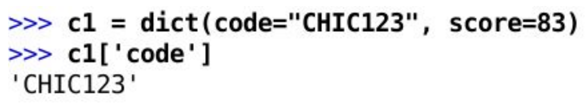


```
class Test():
    def m1(self):
        print("First")
>>> t = Test()
>>> t.m1()
First
class Test():
    def m1(self):
        print("Second")
>>> t.m1()
First
>>> t = Test()
>>> t.m1()
Second
>>>
```

Another “gotcha” is that objects keep their class definition at the time they were created, so if you change the definition it won’t change the behaviour of previously-created objects. Here you see that the m1 method on “t” still prints the First value even though I’ve redefined the class to print “Second”. This is another good reason to test your code from the command line where you can run everything from a clean start.

Why classes?

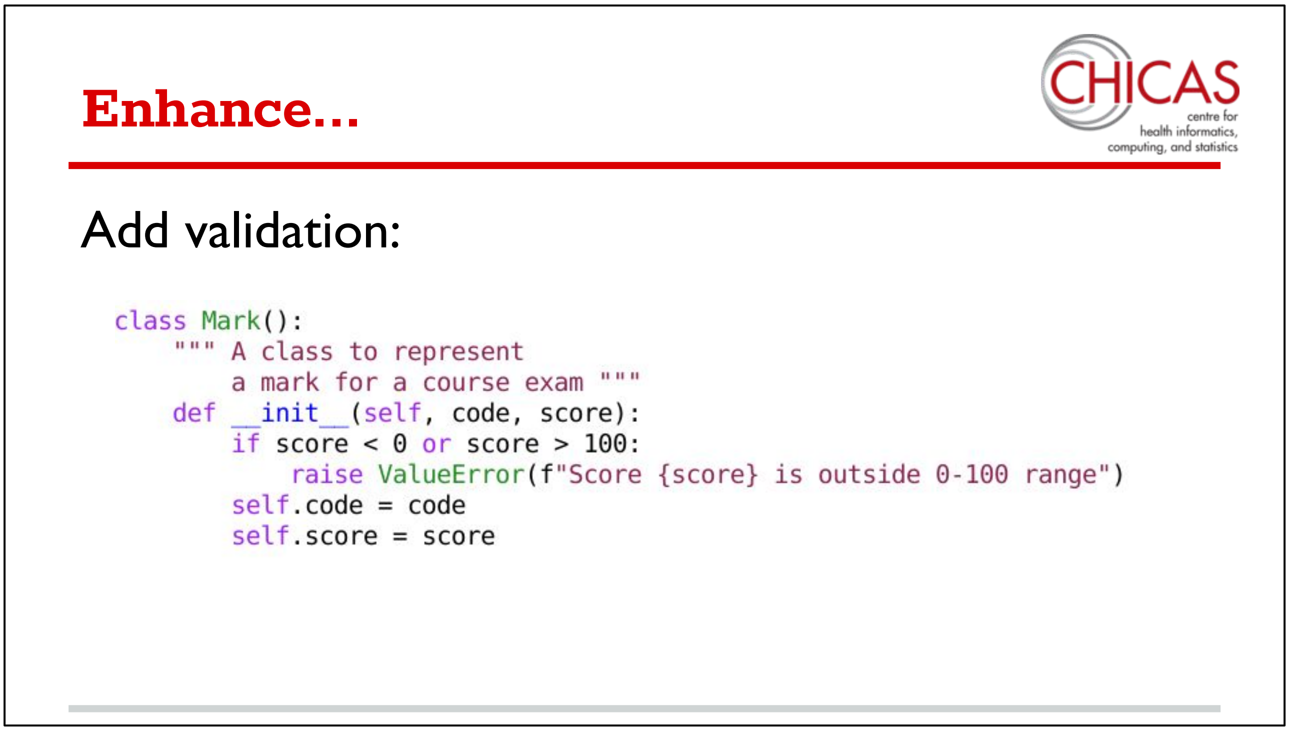
Could use a dictionary for this…



```
>>> c1 = dict(code="CHIC123", score=83)
>>> c1['code']
'CHIC123'
```

… why not?

If you didn’t know about classes yet, you could have stored this info in a dictionary. So why not? What do classes give you?



Enhance...

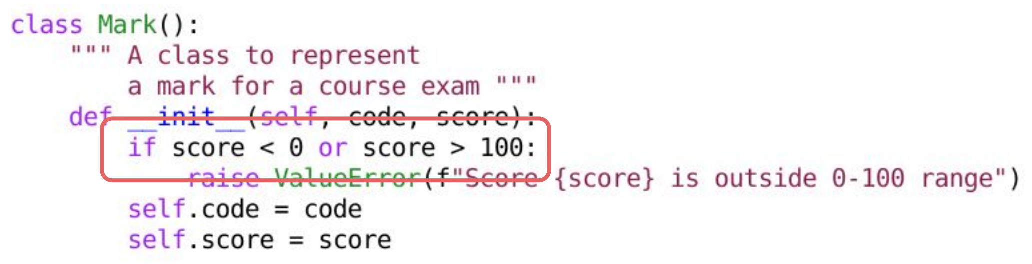
Add validation:

```
class Mark():
    """ A class to represent
        a mark for a course exam """
    def __init__(self, code, score):
        if score < 0 or score > 100:
            raise ValueError(f"Score {score} is outside 0-100 range")
        self.code = code
        self.score = score
```

For one thing we can add validation to our constructor.

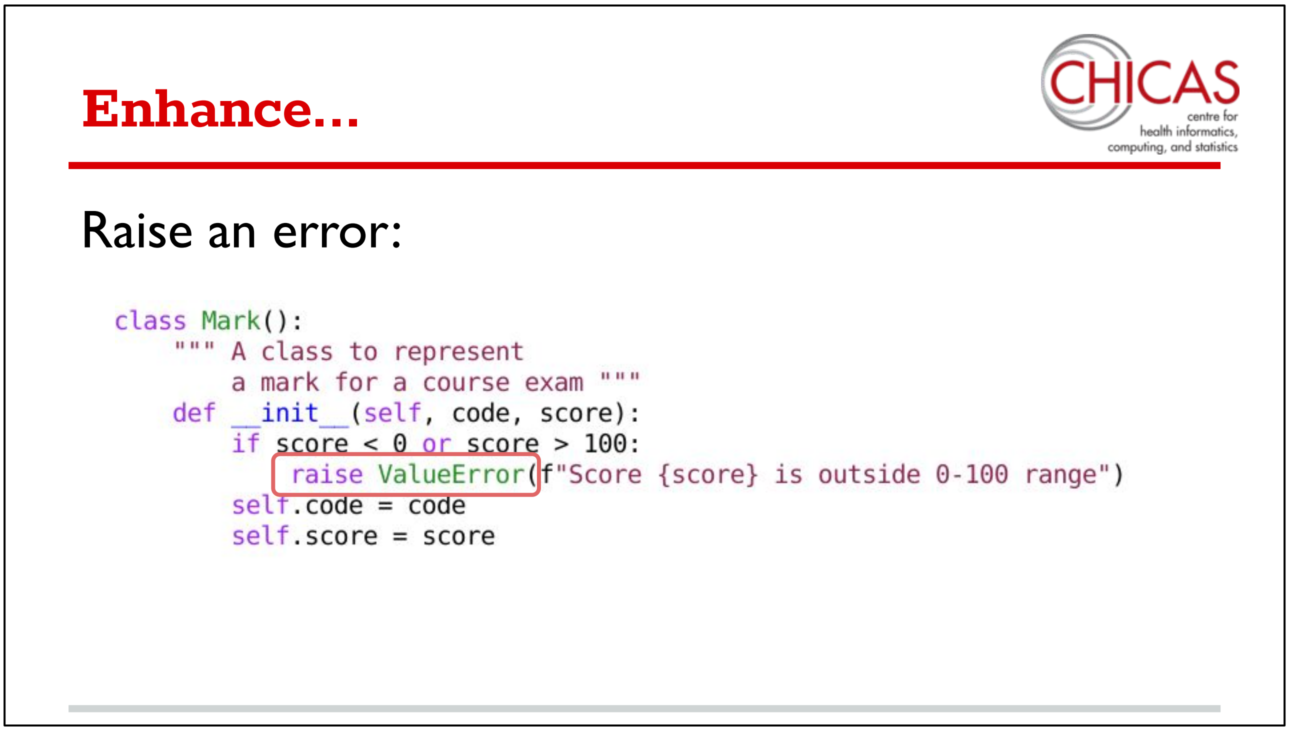
Enhance…

Test inputs:



```
class Mark():
    """ A class to represent
        a mark for a course exam """
    def __init__(self, code, score):
        if score < 0 or score > 100:
            raise ValueError(f"Score {score} is outside 0-100 range")
        self.code = code
        self.score = score
```

I'll check the score is between 0 and 100.

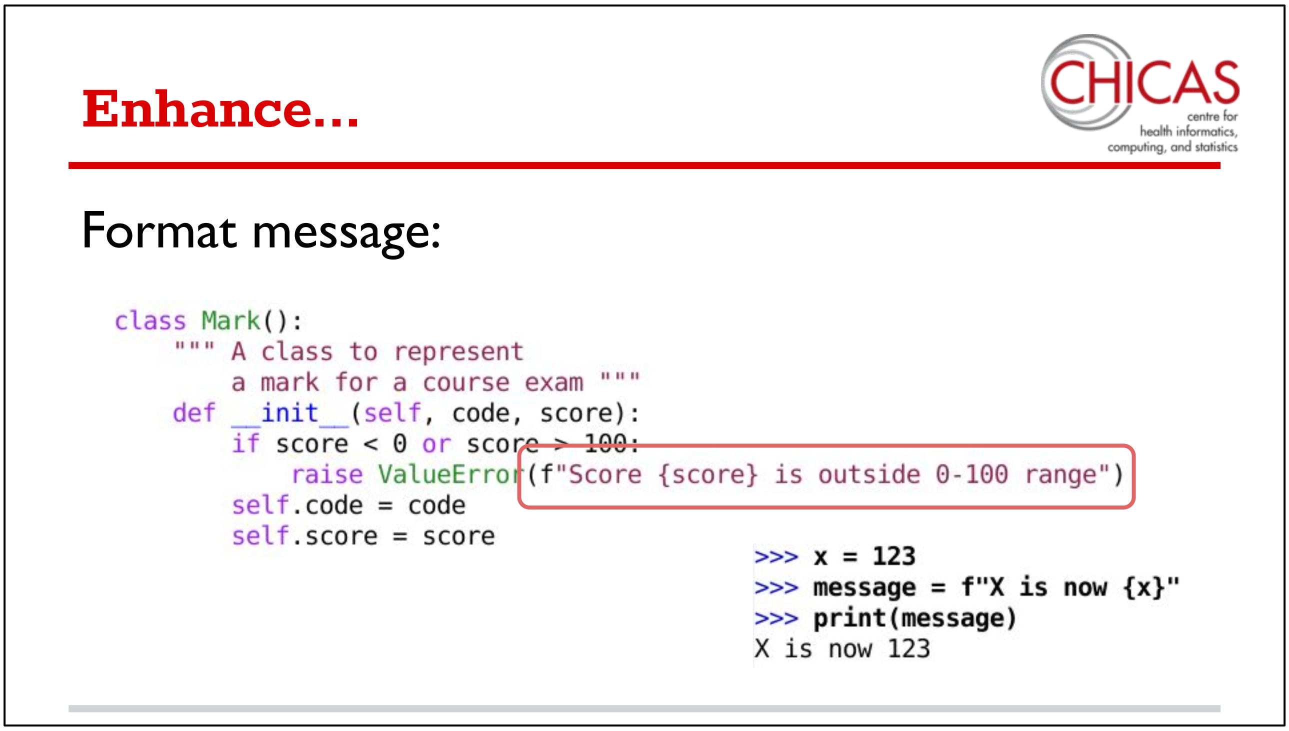


Enhance...

Raise an error:

```
class Mark():
    """ A class to represent
        a mark for a course exam """
    def __init__(self, code, score):
        if score < 0 or score > 100:
            raise ValueError(f"Score {score} is outside 0-100 range")
        self.code = code
        self.score = score
```

And if it isn’t I use python’s “raise” statement to generate an error. This is the thing to do when something “exceptional” happens which the code can’t deal with. Don’t ever exit the program, raise an error will exit the program unless this is being called from something that *can* deal with the error. The “ValueError” here is one of Python’s built-in error classes.



Enhance…

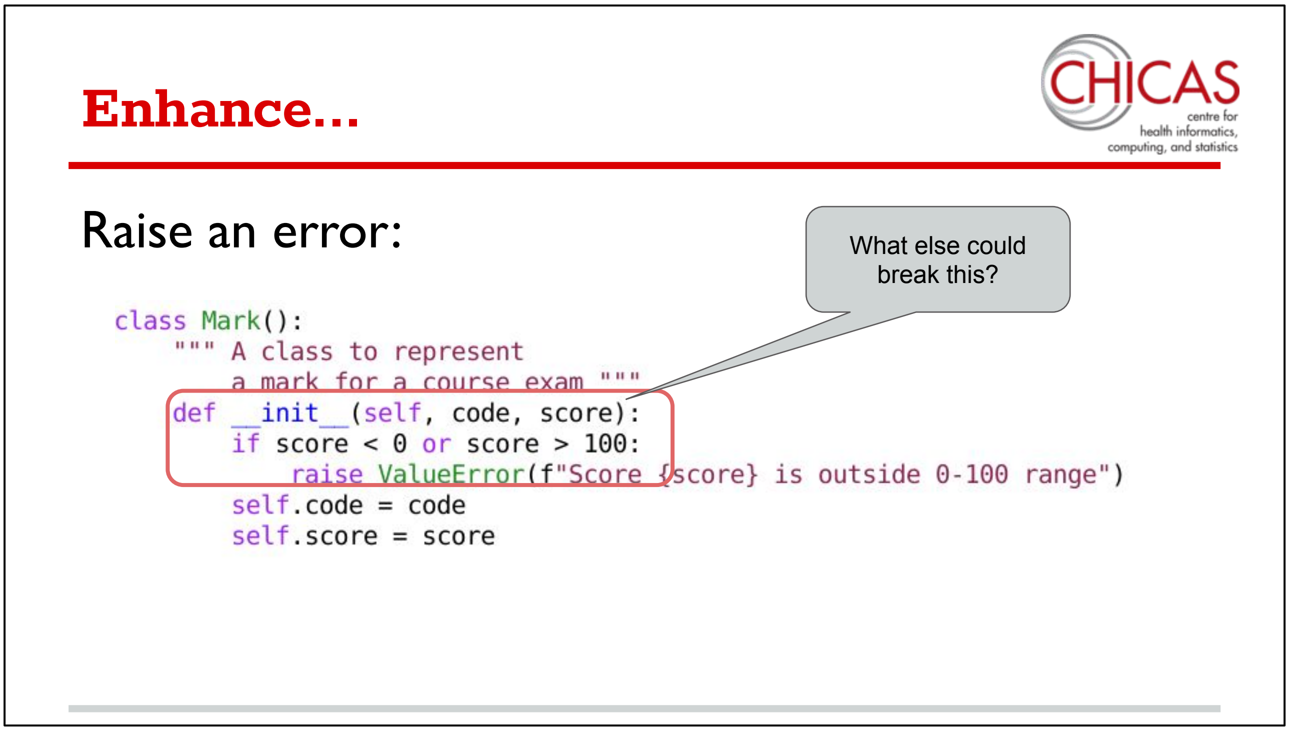
CHICAS
centre for
health informatics,
computing, and statistics

Format message:

```
class Mark():
    """ A class to represent
        a mark for a course exam """
    def __init__(self, code, score):
        if score < 0 or score > 100:
            raise ValueError(f"Score {score} is outside 0-100 range")
        self.code = code
        self.score = score
```

```
>>> x = 123
>>> message = f"X is now {x}"
>>> print(message)
X is now 123
```

The other new thing here is the “format string”, or “f-string”. Unlike a plain quoted string, this will replace things in curly brackets with the result of evaluated expressions.



Enhance…

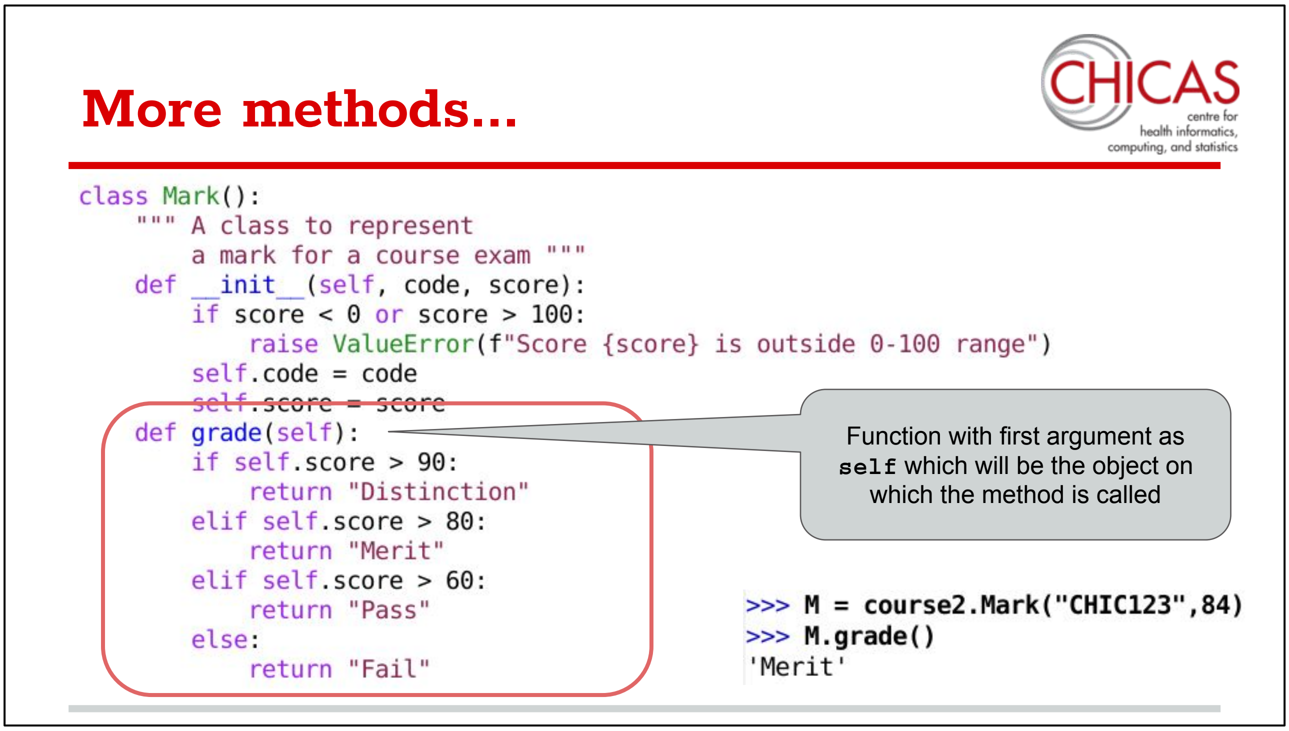
centre for health informatics, computing, and statistics

Raise an error:

```
class Mark():
    """ A class to represent
        a mark for a course exam """
    def __init__(self, code, score):
        if score < 0 or score > 100:
            raise ValueError(f"Score {score} is outside 0-100 range")
        self.code = code
        self.score = score
```

What else could break this?

Its instructive to think about what could break this. Many things don't need handling or testing for since they'll raise an error that you'd only want to raise anyway, for example if a string was passed in instead of a number. The numeric comparison will fail and Python will raise an error. Is this meaningful and useful? Try it.



More methods...

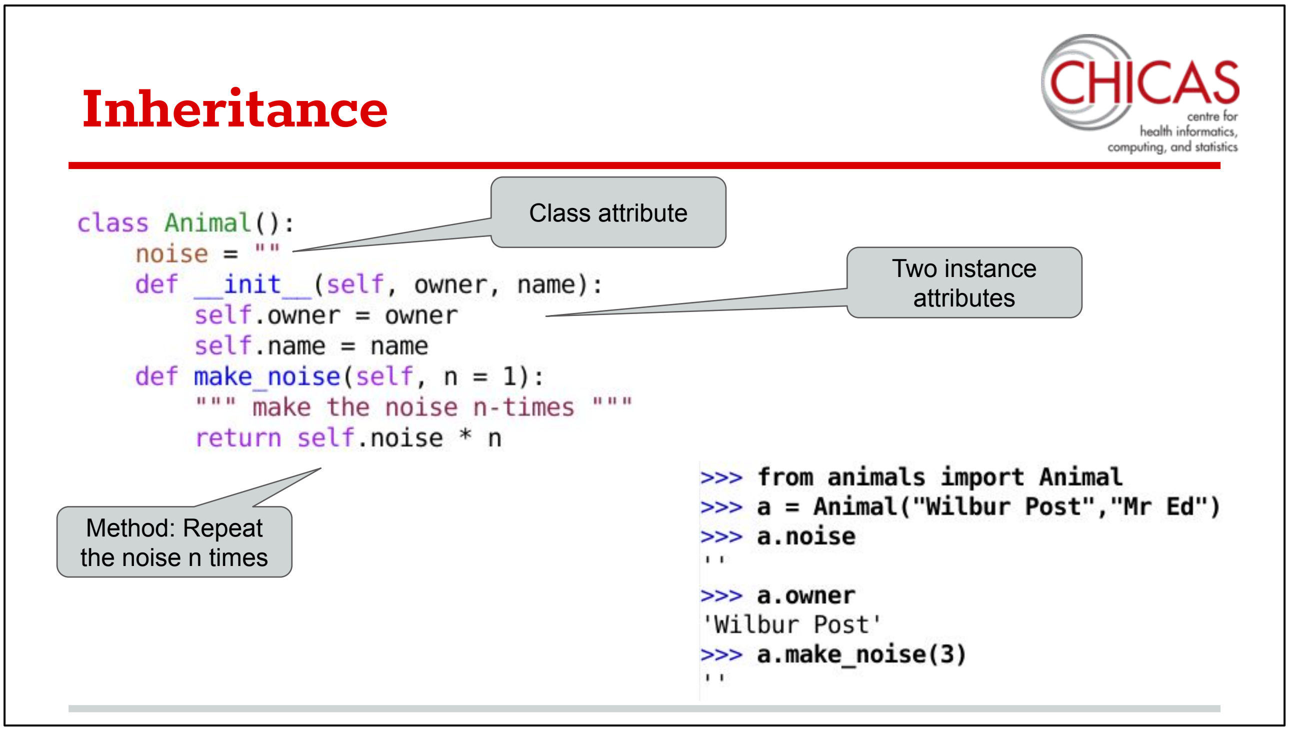
centre for health informatics, computing, and statistics

```python
class Mark():
    """ A class to represent
        a mark for a course exam """
    def __init__(self, code, score):
        if score < 0 or score > 100:
            raise ValueError(f"Score {score} is outside 0-100 range")
        self.code = code
        self.score = score
    def grade(self):
        if self.score > 90:
            return "Distinction"
        elif self.score > 80:
            return "Merit"
        elif self.score > 60:
            return "Pass"
        else:
            return "Fail"
```

Function with first argument as `self` which will be the object on which the method is called

```
>>> M = course2.Mark("CHIC123",84)
>>> M.grade()
'Merit'
```

The other thing we can do with a class that we can’t do with a dictionary is to add methods to it, for example here to apply some grading scheme to the course mark. You could write a function outside the class system to return this grade when given a dictionary but its important to see that the grade is something “bound” to an course mark, so its worth defining it as a method. If it were a separate function then you could call it with something as an argument that wasn’t a course mark, and you might have to test and handle that. This way, grades can only be worked out for course marks. Now this might be a restriction that later needs reconsidering, in which case we may have to think more about our class structure and a possible hierarchy, or a new class for grades themselves...



Inheritance

CHICAS centre for health informatics, computing, and statistics

```
class Animal():
    noise = ""
    def __init__(self, owner, name):
        self.owner = owner
        self.name = name
    def make_noise(self, n = 1):
        """ make the noise n-times """
        return self.noise * n
```

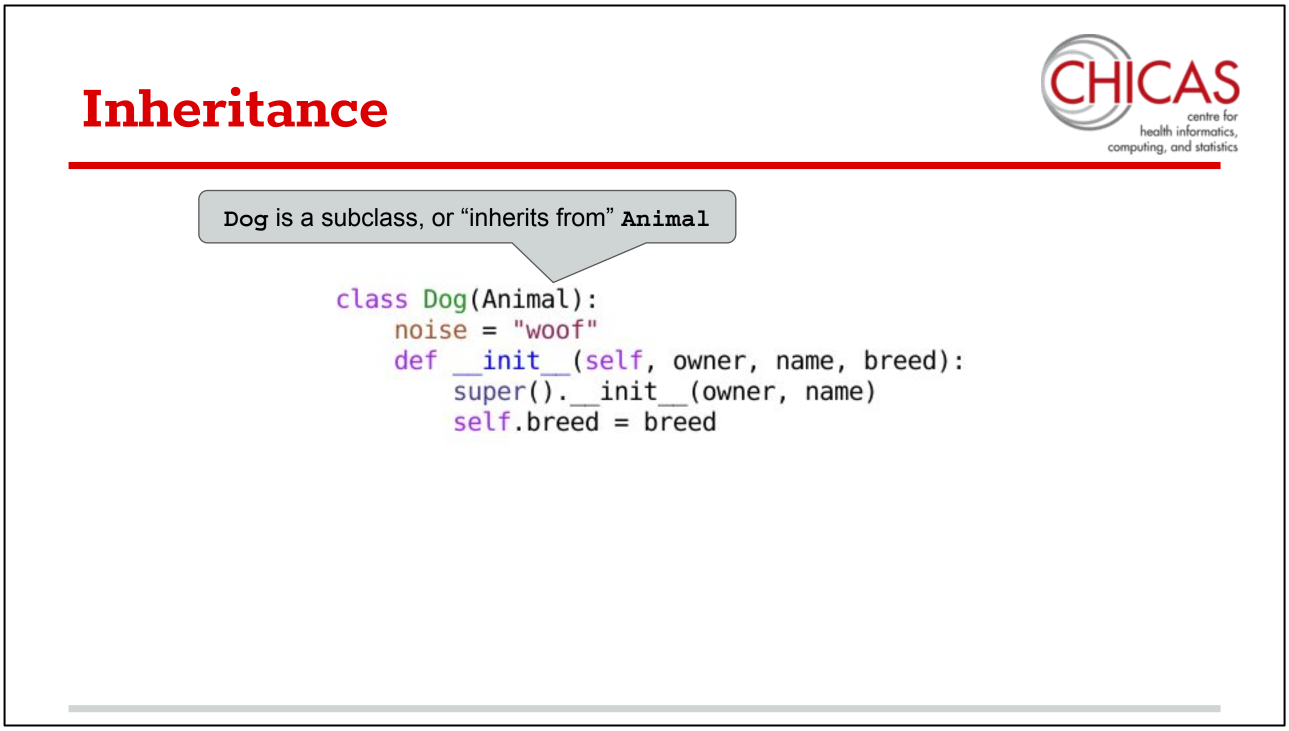
Class attribute

Two instance attributes

Method: Repeat the noise n times

```
>>> from animals import Animal
>>> a = Animal("Wilbur Post","Mr Ed")
>>> a.noise
''
>>> a.owner
'Wilbur Post'
>>> a.make_noise(3)
''
```

The other thing we can do with classes is to build a hierarchy. This “Animal” class stores the name of the animal and the name of the owner. It also keeps the noise of the animal. Animals don’t make individual noises, so its a property of the animal rather than a particular animal (or “instance” of the class). This class can then be used to create animal objects, and the “make_noise” method can be used to repeat the noise a number of times.



Inheritance

`Dog` is a subclass, or “inherits from” `Animal`

```python
class Dog(Animal):
    noise = "woof"
    def __init__(self, owner, name, breed):
        super().__init__(owner, name)
        self.breed = breed
```

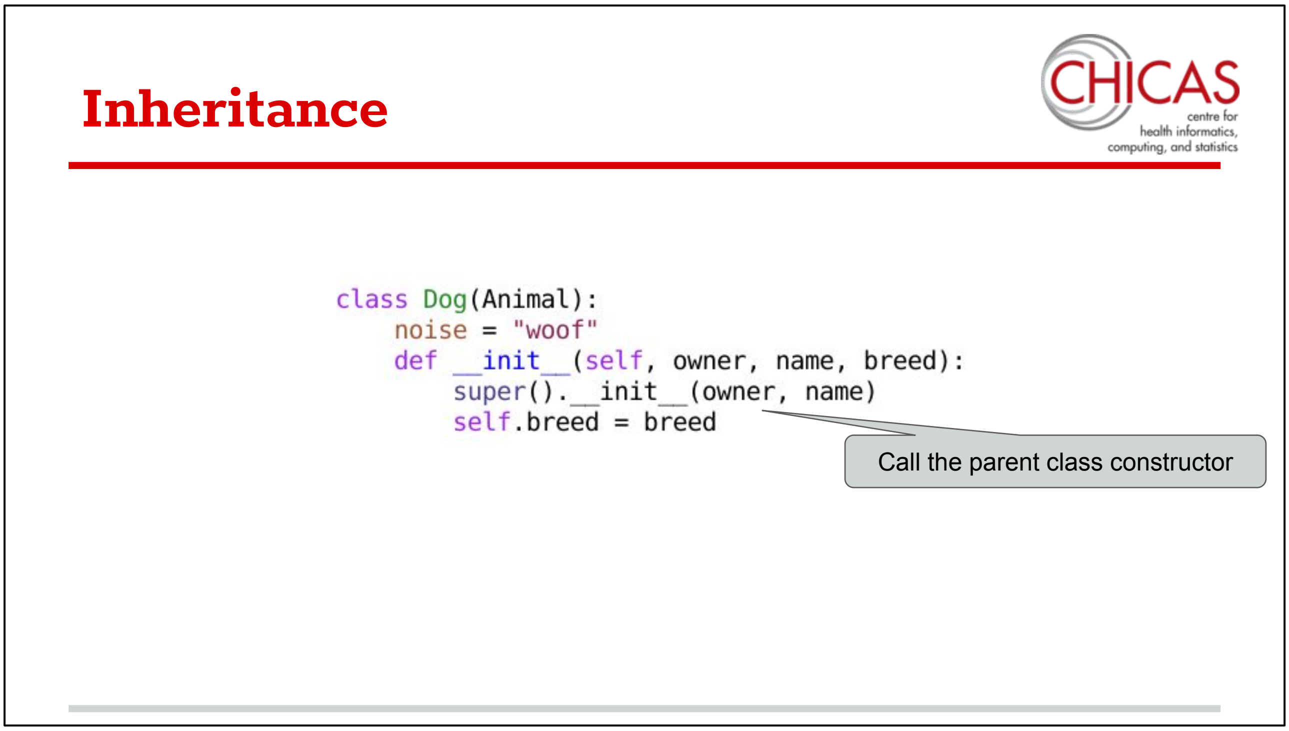
But now we want to specialise the animal class. We can create a class for dogs by *inheriting* from the Animal class.

Inheritance

Override the noise

```
class Dog(Animal):
    noise = "woof"
    def __init__(self, owner, name, breed):
        super().__init__(owner, name)
        self.breed = breed
```

We override the noise in the Animal class with the typical dog noise.

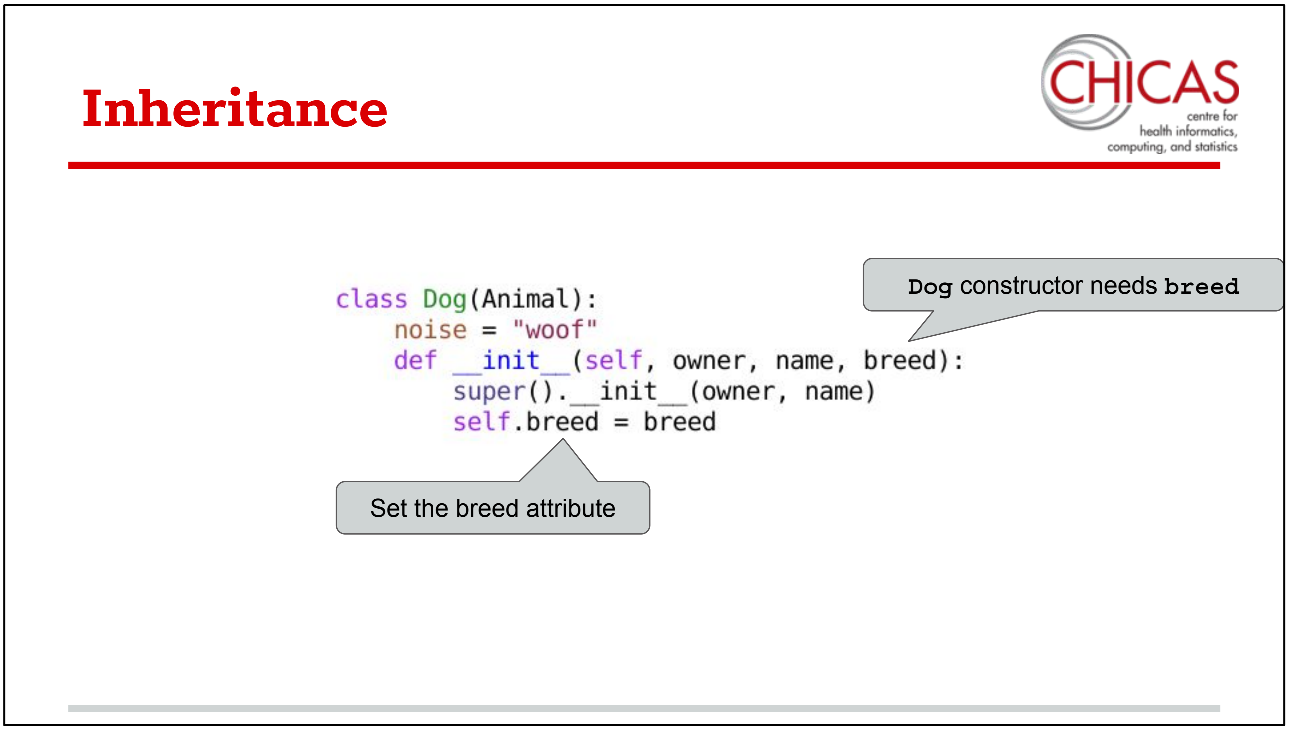


Inheritance

```
class Dog(Animal):
    noise = "woof"
    def __init__(self, owner, name, breed):
        super().__init__(owner, name)
        self.breed = breed
```

Call the parent class constructor

The dog init method has to call the init method of the parent class, and we get that with the super() function rather than explicitly saying “Animal” here in case we ever change the parent class (for example, we might need “Mammal” between Animal and Dog).



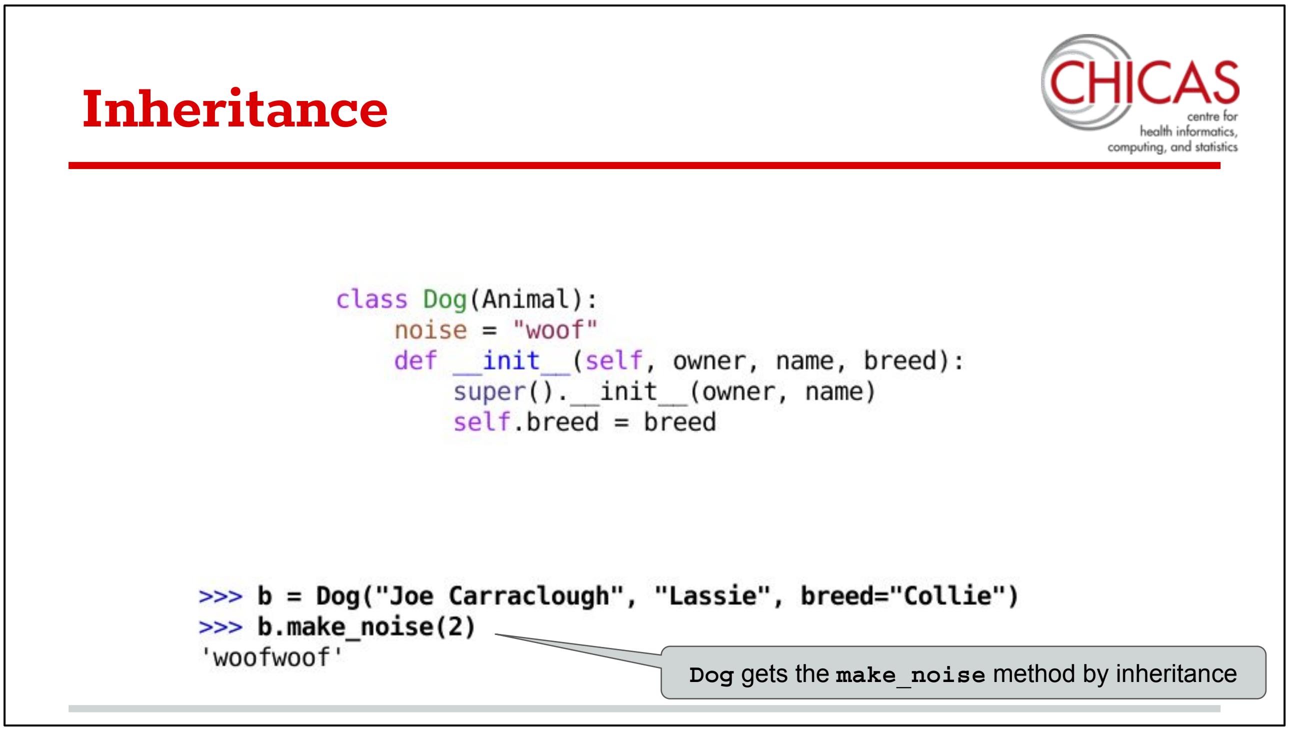
Inheritance

```
class Dog(Animal):
    noise = "woof"
    def __init__(self, owner, name, breed):
        super().__init__(owner, name)
        self.breed = breed
```

`Dog` constructor needs `breed`

Set the breed attribute

Our dog class is going to keep the breed of dog, so this is passed into the constructor and stored in the object.



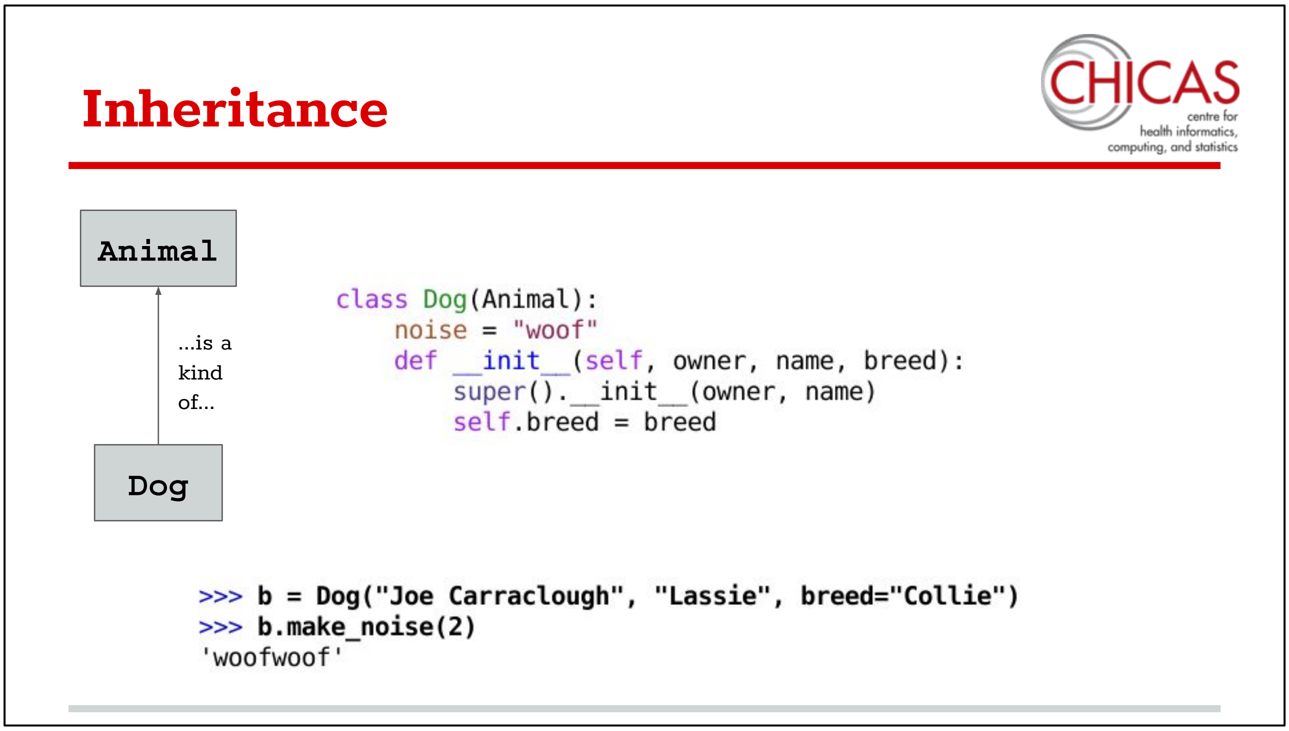
Inheritance

```
class Dog(Animal):
    noise = "woof"
    def __init__(self, owner, name, breed):
        super().__init__(owner, name)
        self.breed = breed
```

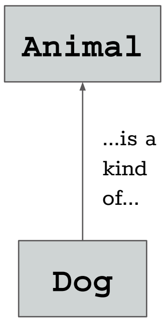
```
>>> b = Dog("Joe Carraclough", "Lassie", breed="Collie")
>>> b.make_noise(2)
'woofwoof'
```

`Dog` gets the `make_noise` method by inheritance

So now we can make a dog object with the owner name, dog name, and breed. Although we didn’t define the “make_noise” function in the Dog class, the dog still gets it because the Dog class inherits from the Animal class.



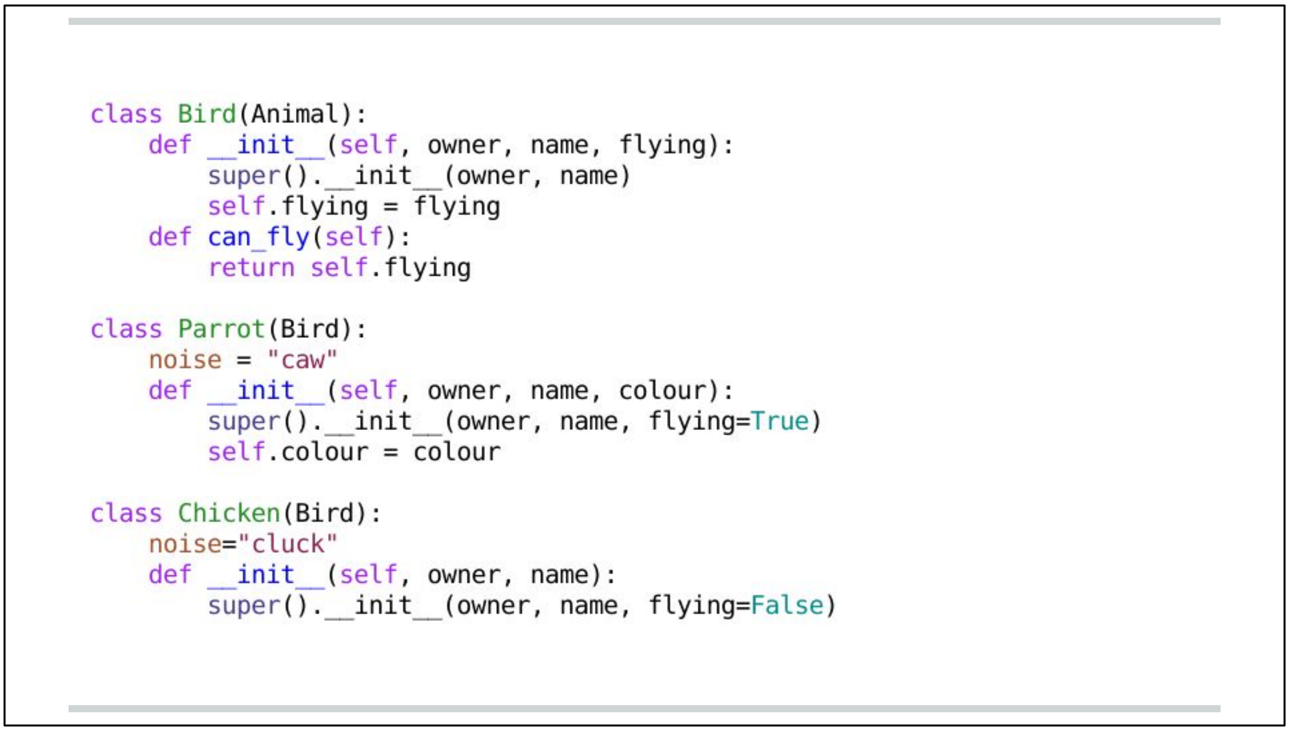
Inheritance



```
class Dog(Animal):
    noise = "woof"
    def __init__(self, owner, name, breed):
        super().__init__(owner, name)
        self.breed = breed
```

```
>>> b = Dog("Joe Carraclough", "Lassie", breed="Collie")
>>> b.make_noise(2)
'woofwoof'
```

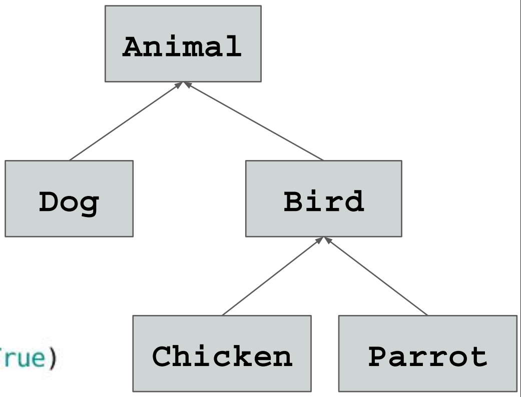
If we see that one class of objects "...is a kind of…" another class of objects then that's a good case for creating that more specialised class as a sub-class of the other.



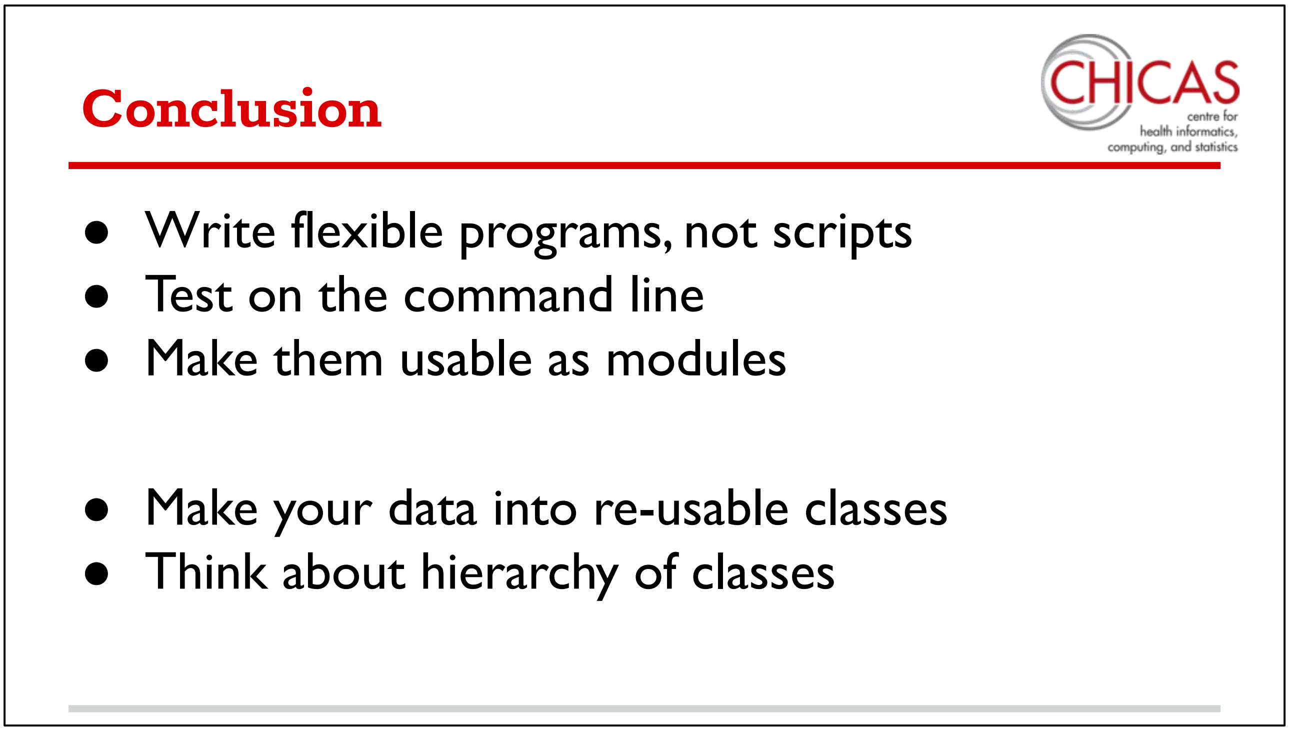
```python
class Bird(Animal):
    def __init__(self, owner, name, flying):
        super().__init__(owner, name)
        self.flying = flying
    def can_fly(self):
        return self.flying

class Parrot(Bird):
    noise = "caw"
    def __init__(self, owner, name, colour):
        super().__init__(owner, name, flying=True)
        self.colour = colour

class Chicken(Bird):
    noise="cluck"
    def __init__(self, owner, name):
        super().__init__(owner, name, flying=False)
```

And we can extend this to many levels. Birds are a kind of animal, and parrots and chickens are kinds of birds. Birds either fly or not, so we store a “flying” property in the bird class.

```python
class Bird(Animal):
    def __init__(self, owner, name, flying):
        super().__init__(owner, name)
        self.flying = flying
    def can_fly(self):
        return self.flying

class Parrot(Bird):
    noise = "caw"
    def __init__(self, owner, name, colour):
        super().__init__(owner, name, flying=True)
        self.colour = colour

class Chicken(Bird):
    noise="cluck"
    def __init__(self, owner, name):
        super().__init__(owner, name, flying=False)
```



So now we have an object hierarchy like this. Clear object data structures like this can help to clarify our thinking about data within our program, and mean that other can make more sense of our code.



Conclusion

- Write flexible programs, not scripts
- Test on the command line
- Make them usable as modules

- Make your data into re-usable classes
- Think about hierarchy of classes

In conclusion...