Queen Mary
University of London

DR ADRIAN BEVAN

PRACTICAL MACHINE LEARNING

INTRODUCTORY PYTHON

LECTURE PLAN

- Resources and requirements
- Example
- Hello World
- Operators
- Commands:
  - Conditionals
  - Loops
  - Built in commands
- Functions
- Classes
- matplotlib
- NumPy
- Summary
- Suggested Reading

These slides focus on the use of Python in order to ensure that there is a common background to build on for the remaining weeks.

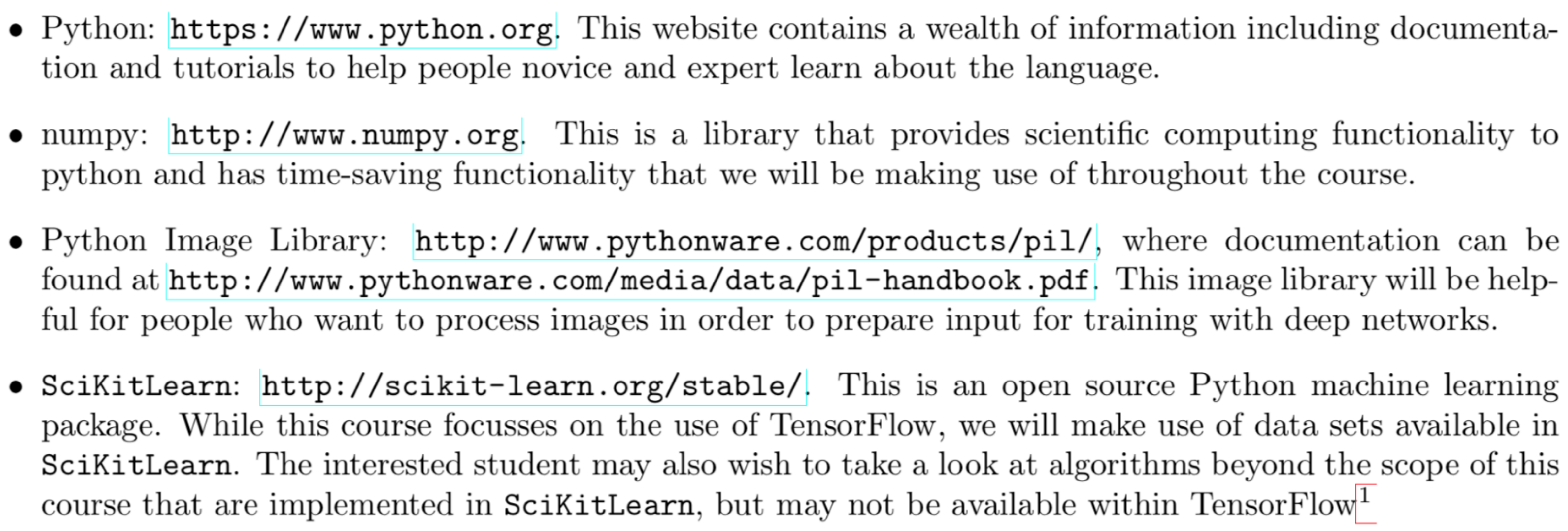
Once we have covered Python basics we will move on to TensorFlow basics, and then on to machine learning with TensorFlow.

People who can already code Python will find this a relaxing start.

QMUL Summer School:	https://www.qmul.ac.uk/summer-school/
Practical Machine Learning QMplus Page:	https://qmplus.qmul.ac.uk/course/view.php?id=10006

RESOURCES AND REQUIREMENTS

- The following resources will be of use to you

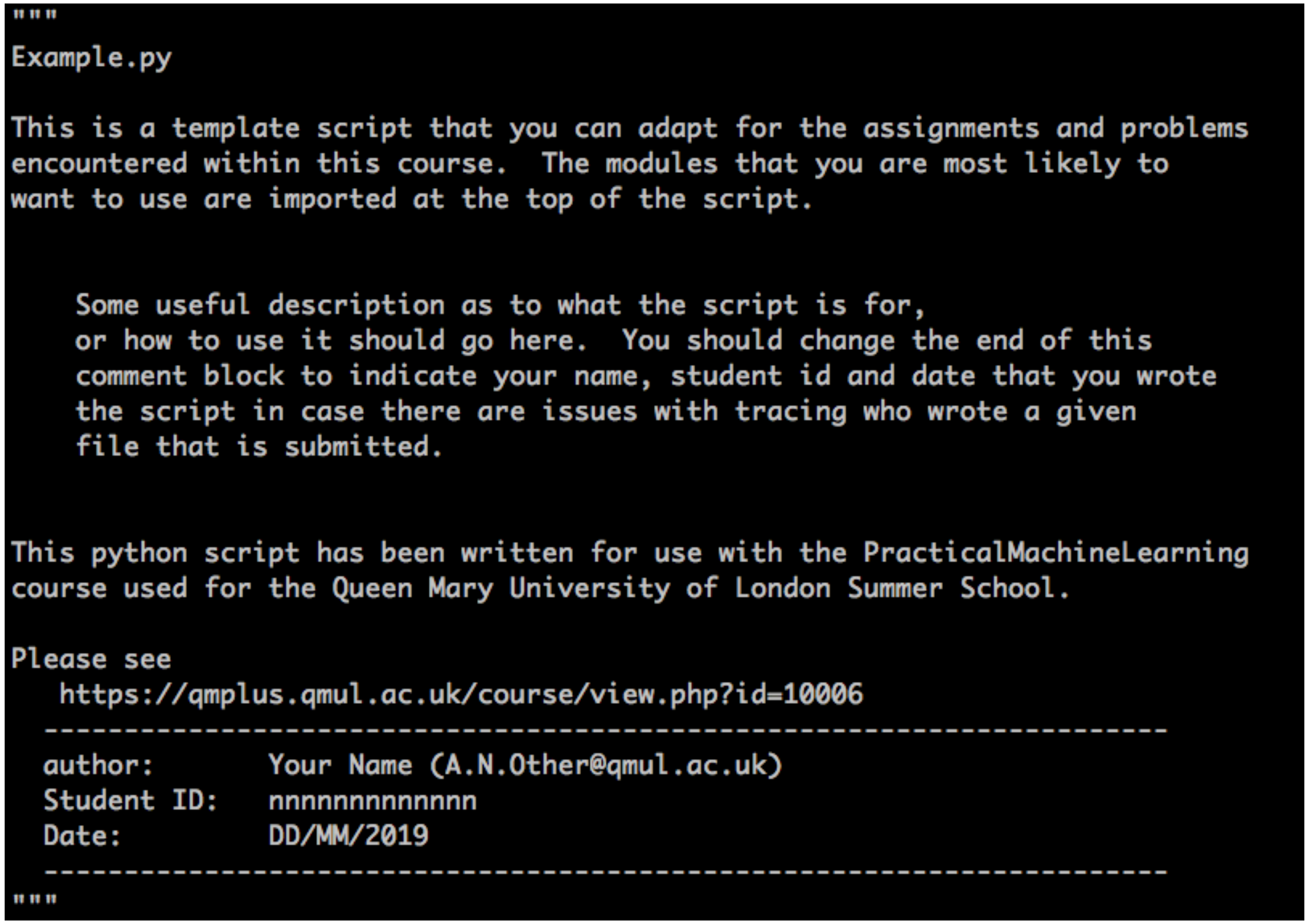


- Python: `https://www.python.org`. This website contains a wealth of information including documentation and tutorials to help people novice and expert learn about the language.
- numpy: `http://www.numpy.org`. This is a library that provides scientific computing functionality to python and has time-saving functionality that we will be making use of throughout the course.
- Python Image Library: `http://www.pythonware.com/products/pil/`, where documentation can be found at `http://www.pythonware.com/media/data/pil-handbook.pdf`. This image library will be helpful for people who want to process images in order to prepare input for training with deep networks.
- `SciKitLearn`: `http://scikit-learn.org/stable/`. This is an open source Python machine learning package. While this course focusses on the use of TensorFlow, we will make use of data sets available in `SciKitLearn`. The interested student may also wish to take a look at algorithms beyond the scope of this course that are implemented in `SciKitLearn`, but may not be available within TensorFlow[1]

- We will use standard python libraries along with numpy, TensorFlow and SciKitLearn for numerical work.
- Matplotlib will be used for plotting outputs; PIL is an alternative you may encounter.

EXAMPLE

- The `Example.py` file is a simple skeleton example python script.

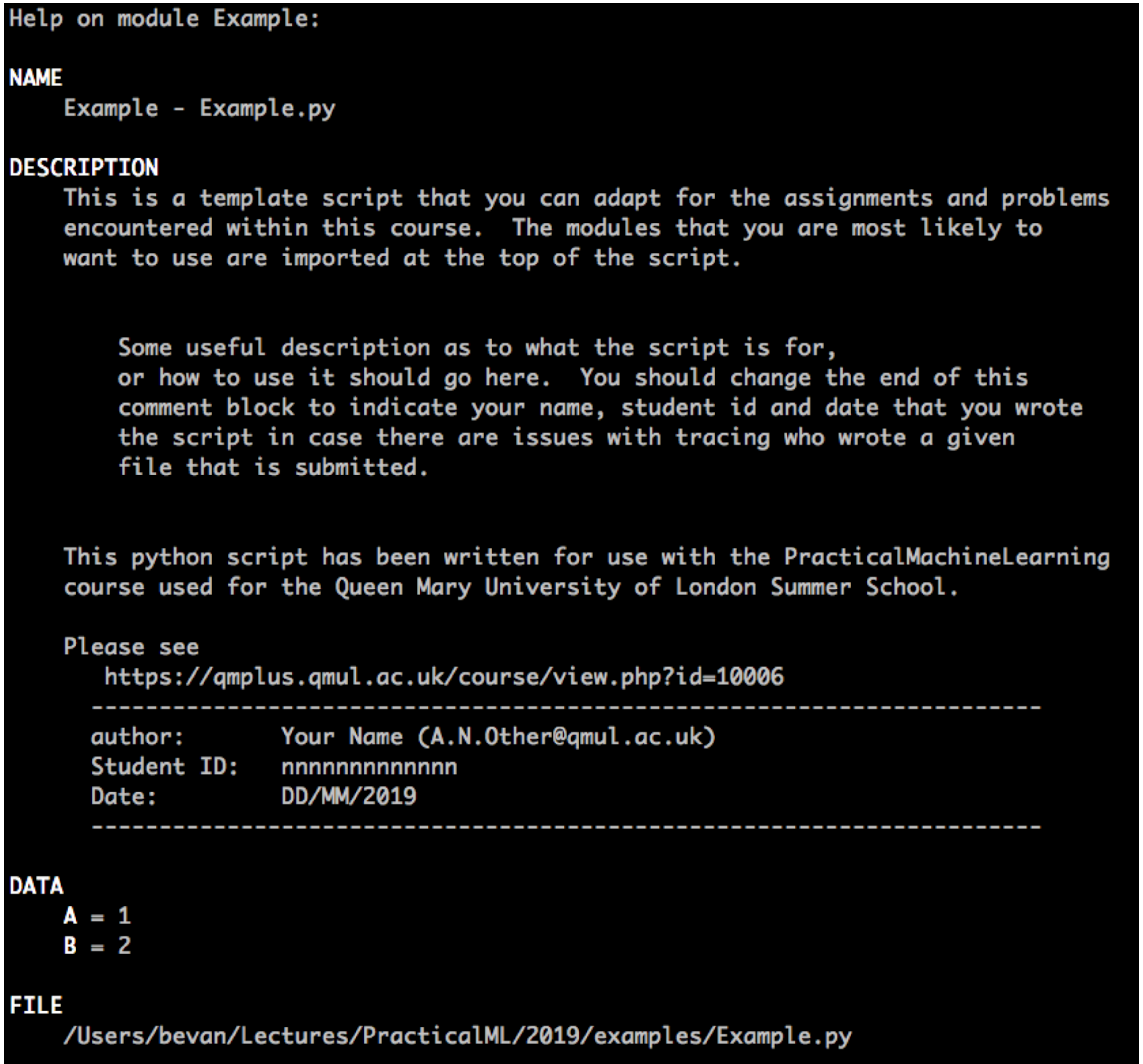


```
"""
Example.py

This is a template script that you can adapt for the assignments and problems
encountered within this course.  The modules that you are most likely to
want to use are imported at the top of the script.


    Some useful description as to what the script is for,
    or how to use it should go here.  You should change the end of this
    comment block to indicate your name, student id and date that you wrote
    the script in case there are issues with tracing who wrote a given
    file that is submitted.


This python script has been written for use with the PracticalMachineLearning
course used for the Queen Mary University of London Summer School.

Please see
   https://qmplus.qmul.ac.uk/course/view.php?id=10006
  ------------------------------------------------------------------------
  author:       Your Name (A.N.Other@qmul.ac.uk)
  Student ID:   nnnnnnnnnnnnn
  Date:         DD/MM/2019
  ------------------------------------------------------------------------
"""
```

The text contained in the pair of triple quotes (“””) is a comment. If you `import Example` and then execute the command `help(Example)` you will see a print out generated from this.

EXAMPLE

If you `import Example` and then execute the command `help(Example)` you will see this print out:



```
Help on module Example:

NAME
    Example - Example.py

DESCRIPTION
    This is a template script that you can adapt for the assignments and problems
    encountered within this course.  The modules that you are most likely to
    want to use are imported at the top of the script.


        Some useful description as to what the script is for,
        or how to use it should go here.  You should change the end of this
        comment block to indicate your name, student id and date that you wrote
        the script in case there are issues with tracing who wrote a given
        file that is submitted.


    This python script has been written for use with the PracticalMachineLearning
    course used for the Queen Mary University of London Summer School.

    Please see
      https://qmplus.qmul.ac.uk/course/view.php?id=10006
     ----------------------------------------------------------------------
     author:        Your Name (A.N.Other@qmul.ac.uk)
     Student ID:    nnnnnnnnnnnn
     Date:          DD/MM/2019
     ----------------------------------------------------------------------

DATA
    A = 1
    B = 2

FILE
    /Users/bevan/Lectures/PracticalML/2019/examples/Example.py
```

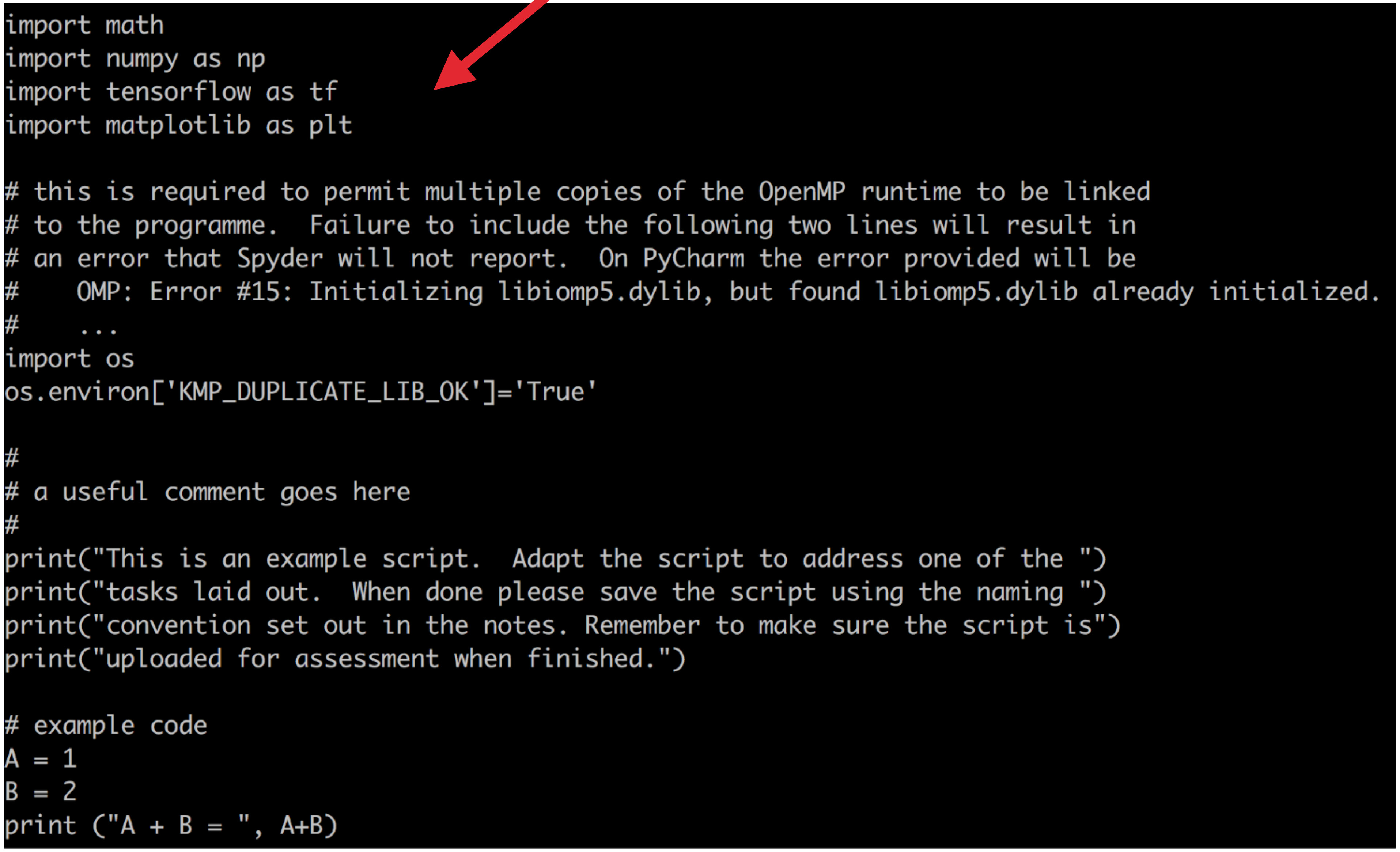
Replace the description with something useful.

Complete the author, Student ID and Data fields for your work.

EXAMPLE

- The `Example.py` file is a simple skeleton example python script.

Import Python libraries at the start of the script. These are four useful libraries we will be using a lot.

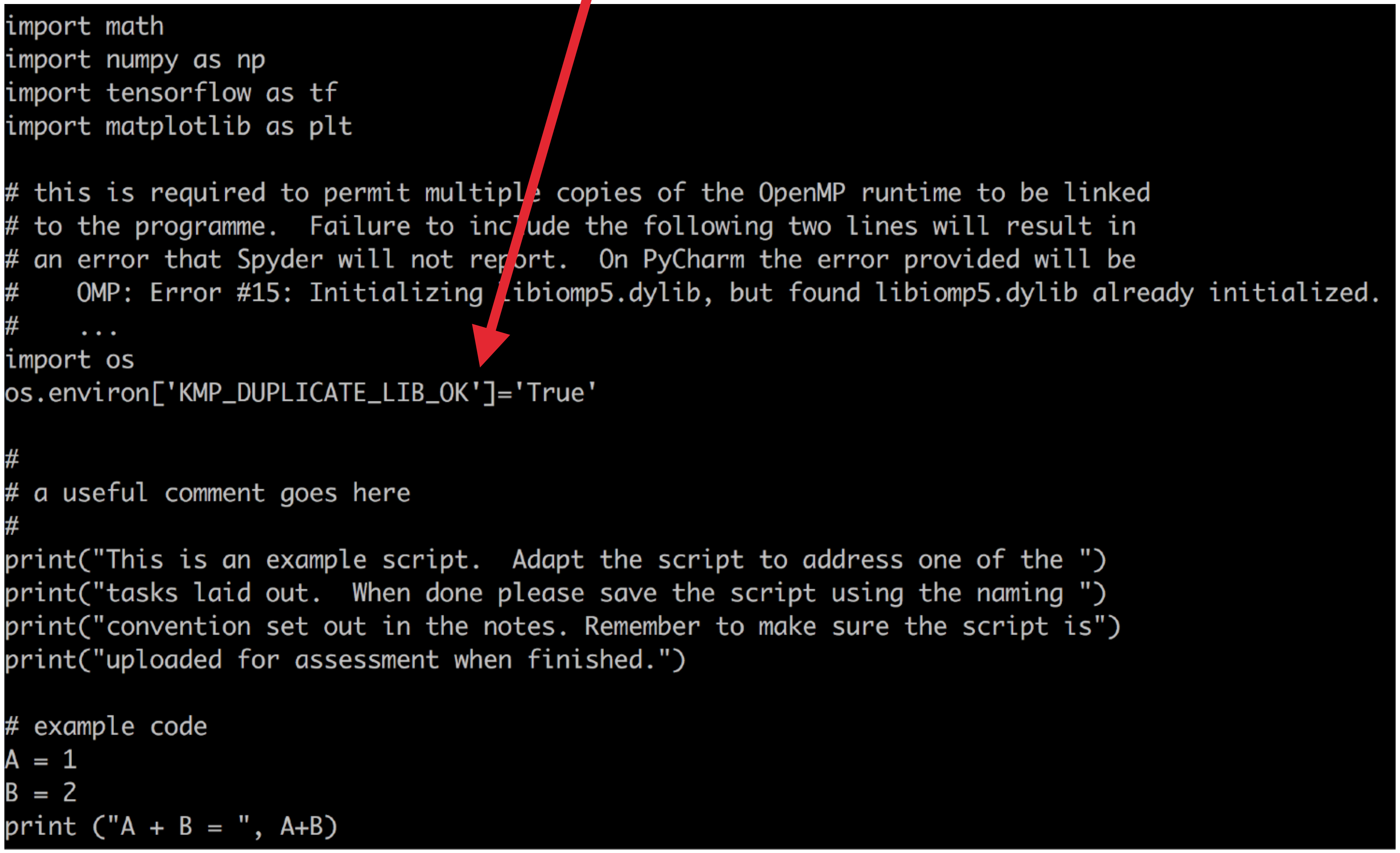


```
import math
import numpy as np
import tensorflow as tf
import matplotlib as plt

# this is required to permit multiple copies of the OpenMP runtime to be linked
# to the programme.  Failure to include the following two lines will result in
# an error that Spyder will not report.  On PyCharm the error provided will be
#    OMP: Error #15: Initializing libiomp5.dylib, but found libiomp5.dylib already initialized.
#    ...
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

#
# a useful comment goes here
#
print("This is an example script.  Adapt the script to address one of the ")
print("tasks laid out.  When done please save the script using the naming ")
print("convention set out in the notes. Remember to make sure the script is")
print("uploaded for assessment when finished.")

# example code
A = 1
B = 2
print ("A + B = ", A+B)
```

EXAMPLE

- The `Example.py` file is a simple skeleton example python script.

Ensure that this environment variable is set to avoid problems when running on Spyder or PyCharm.

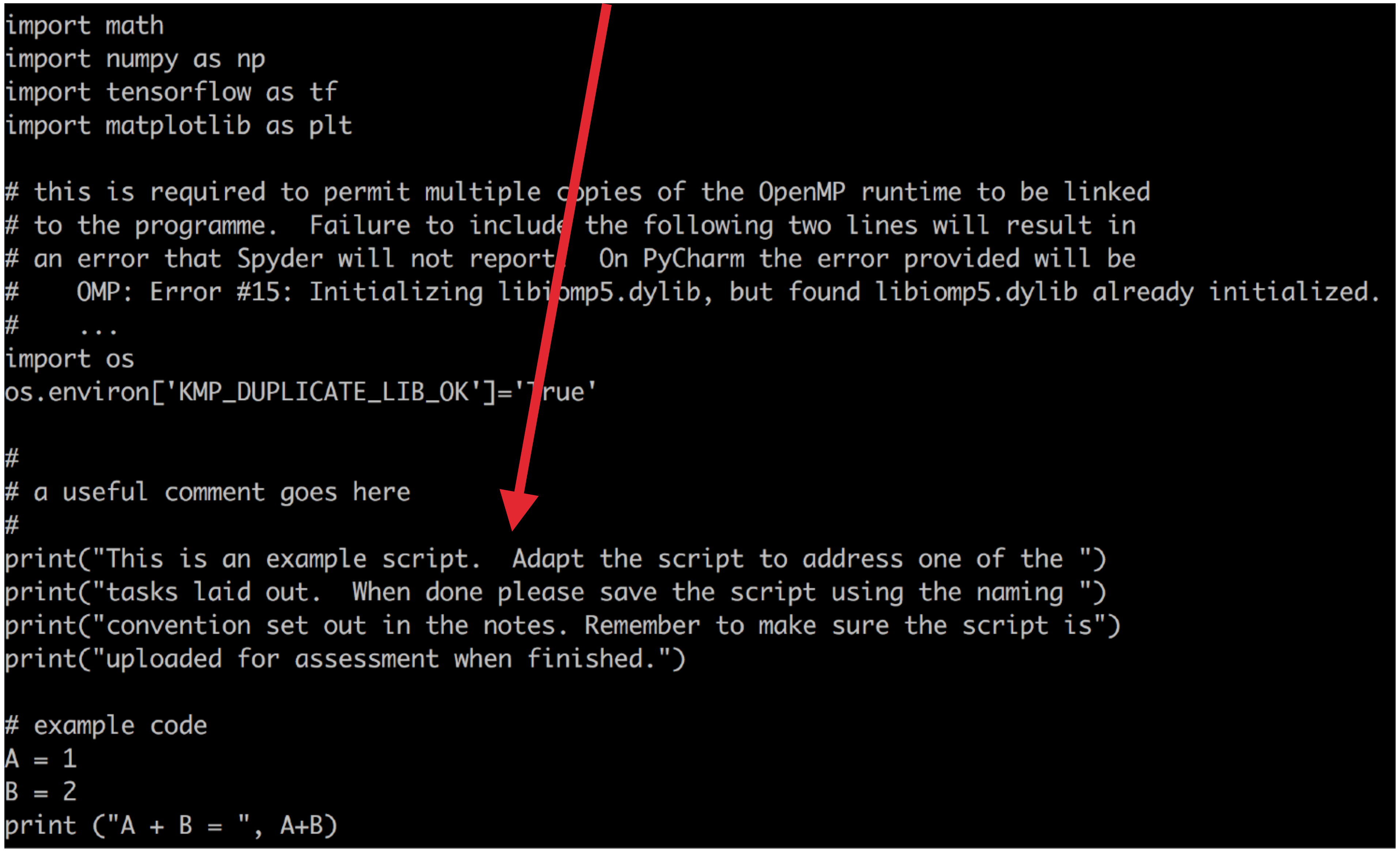


```
import math
import numpy as np
import tensorflow as tf
import matplotlib as plt

# this is required to permit multiple copies of the OpenMP runtime to be linked
# to the programme.  Failure to include the following two lines will result in
# an error that Spyder will not report.  On PyCharm the error provided will be
#    OMP: Error #15: Initializing libiomp5.dylib, but found libiomp5.dylib already initialized.
#    ...
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

#
# a useful comment goes here
#
print("This is an example script.  Adapt the script to address one of the ")
print("tasks laid out.  When done please save the script using the naming ")
print("convention set out in the notes. Remember to make sure the script is")
print("uploaded for assessment when finished.")

# example code
A = 1
B = 2
print ("A + B = ", A+B)
```

EXAMPLE

- The `Example.py` file is a simple skeleton example python script.

Print out some messages to the user when running the script.

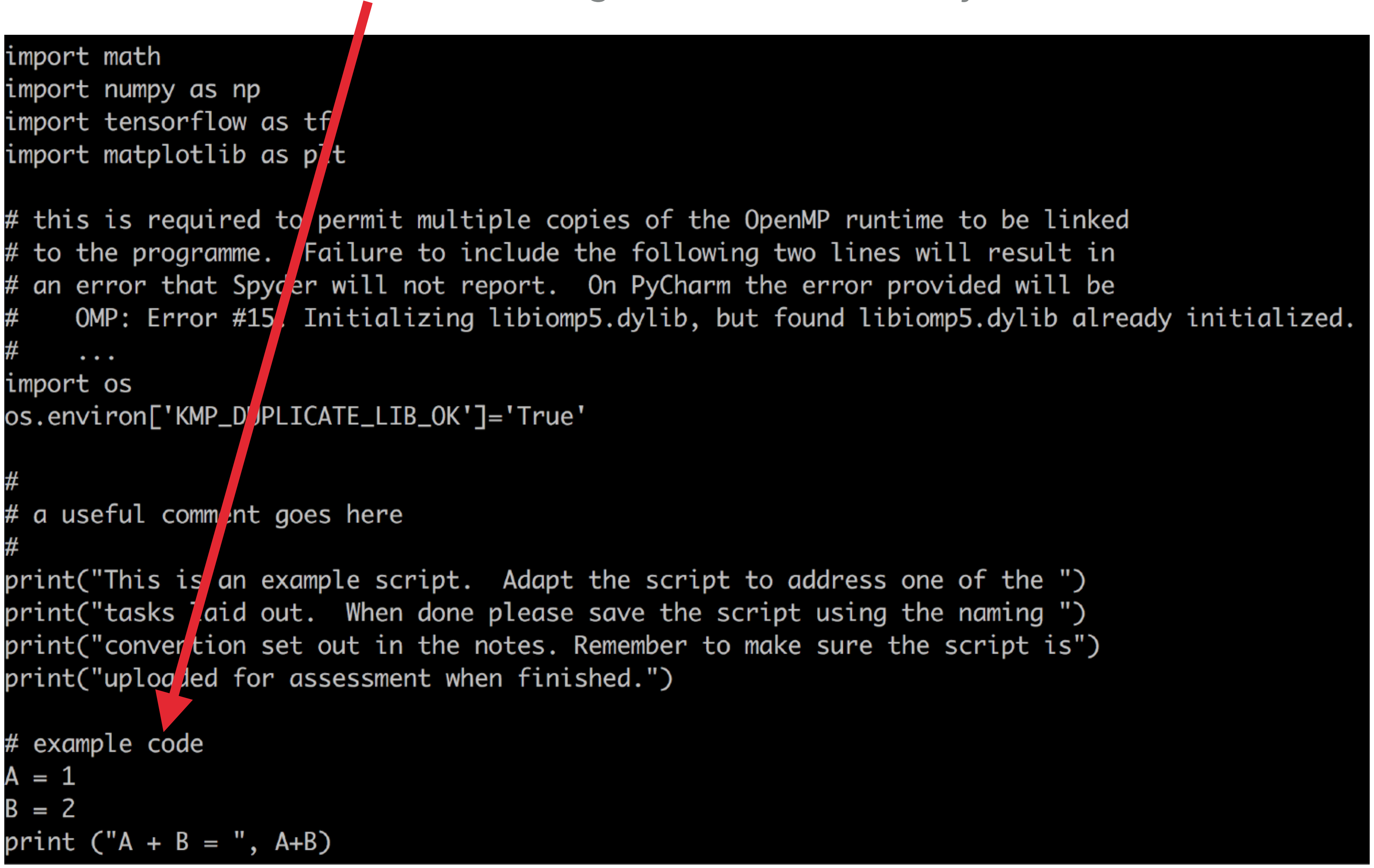


```
import math
import numpy as np
import tensorflow as tf
import matplotlib as plt

# this is required to permit multiple copies of the OpenMP runtime to be linked
# to the programme.  Failure to include the following two lines will result in
# an error that Spyder will not report.  On PyCharm the error provided will be
#    OMP: Error #15: Initializing libiomp5.dylib, but found libiomp5.dylib already initialized.
#    ...
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

#
# a useful comment goes here
#
print("This is an example script.  Adapt the script to address one of the ")
print("tasks laid out.  When done please save the script using the naming ")
print("convention set out in the notes. Remember to make sure the script is")
print("uploaded for assessment when finished.")

# example code
A = 1
B = 2
print ("A + B = ", A+B)
```

EXAMPLE

- The `Example.py` file is a simple skeleton example python script.

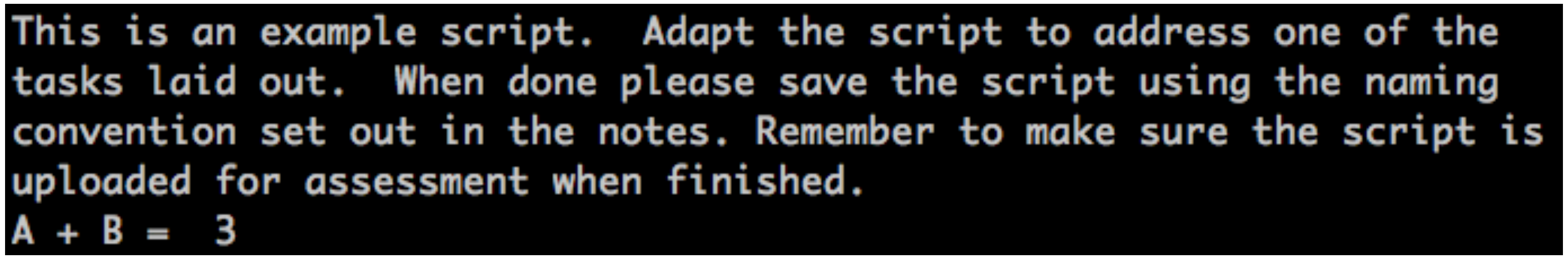
Do something (more on this shortly)



```
import math
import numpy as np
import tensorflow as tf
import matplotlib as plt

# this is required to permit multiple copies of the OpenMP runtime to be linked
# to the programme.  Failure to include the following two lines will result in
# an error that Spyder will not report.  On PyCharm the error provided will be
#    OMP: Error #15: Initializing libiomp5.dylib, but found libiomp5.dylib already initialized.
#    ...
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

#
# a useful comment goes here
#
print("This is an example script.  Adapt the script to address one of the ")
print("tasks laid out.  When done please save the script using the naming ")
print("convention set out in the notes. Remember to make sure the script is")
print("uploaded for assessment when finished.")

# example code
A = 1
B = 2
print ("A + B = ", A+B)
```

EXAMPLE

- We can run the `Example.py` script in Spyder to obtain the following output:



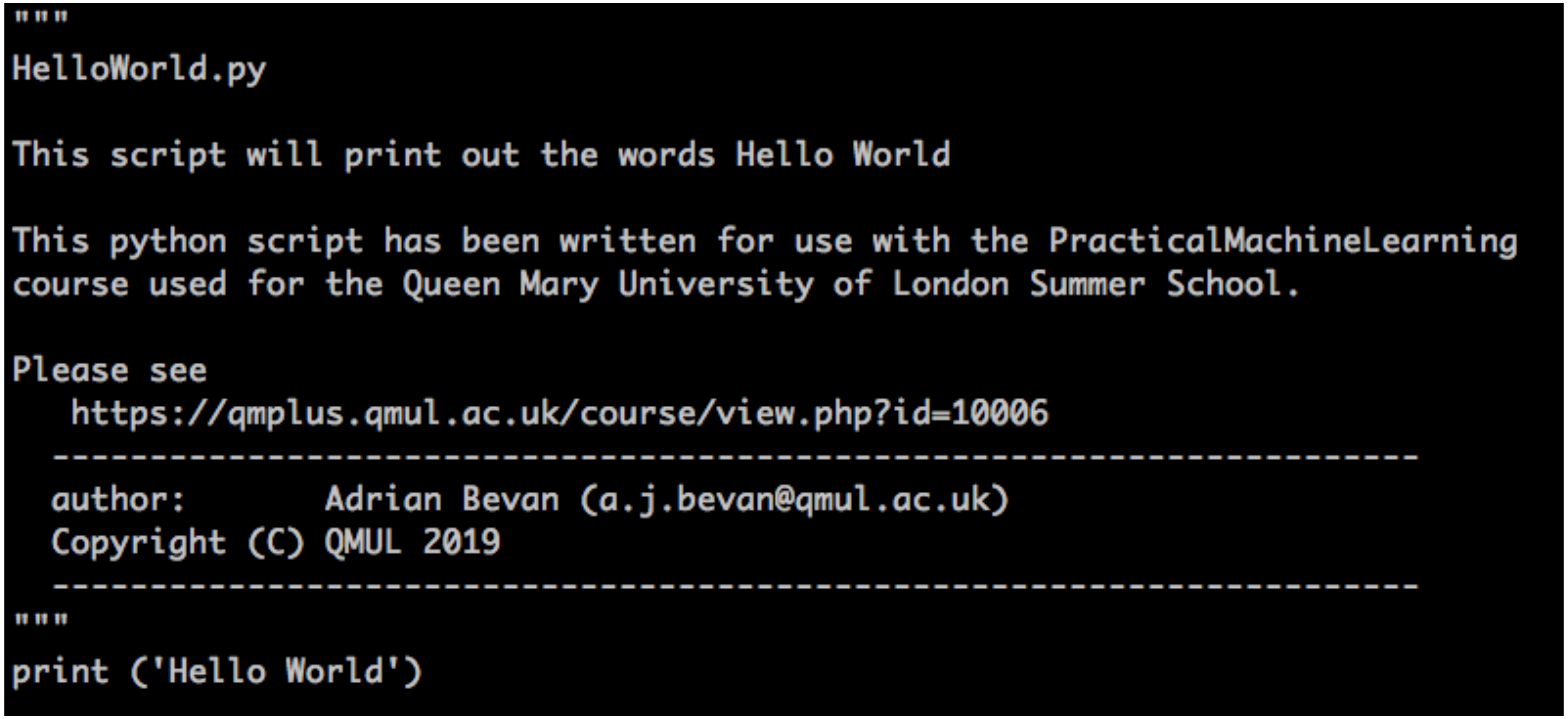
```
This is an example script.  Adapt the script to address one of the
tasks laid out.  When done please save the script using the naming
convention set out in the notes. Remember to make sure the script is
uploaded for assessment when finished.
A + B =  3
```

- This can be used as a skeleton to build on when working through the problems we will encounter in the course.
- It is important to [***i.e. you are expected to***]:
    - Provide documentation (the comment field at the top of `Example.py` is part of this).
    - Provide authorship information (to make sure you get assigned marks for your work you should make sure your files contain your name and your student number. A data stamp can help identify your work.

HELLO WORLD

- `HelloWorld.py`

- This is the canonical example to learn how to code when starting to pick up a new language.
  - Provides the basics of how to use the language in terms of getting a simple programme running that prints out a message.

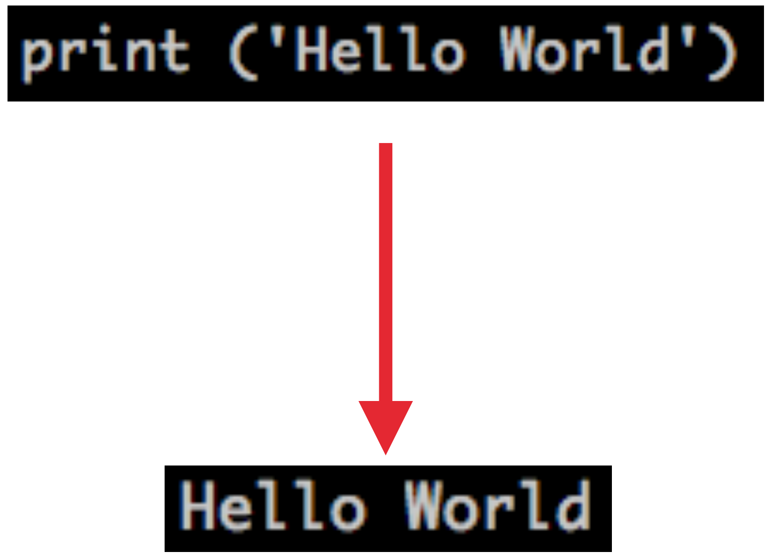


```
"""
HelloWorld.py

This script will print out the words Hello World

This python script has been written for use with the PracticalMachineLearning
course used for the Queen Mary University of London Summer School.

Please see
   https://qmplus.qmul.ac.uk/course/view.php?id=10006
  ----------------------------------------------------------------------
  author:       Adrian Bevan (a.j.bevan@qmul.ac.uk)
  Copyright (C) QMUL 2019
  ----------------------------------------------------------------------
"""
print ('Hello World')
```

HELLO WORLD

- `HelloWorld.py`

- This is the canonical example to learn how to code when starting to pick up a new language.
  - Provides the basics of how to use the language in terms of getting a simple programme running that prints out a message.



```
print ('Hello World')
```

```
Hello World
```

When run this script will simply display the message 'Hello World'.

This is not a particularly exciting example, but you should all be able to run this script to get this output.

It confirms that your setup is working.

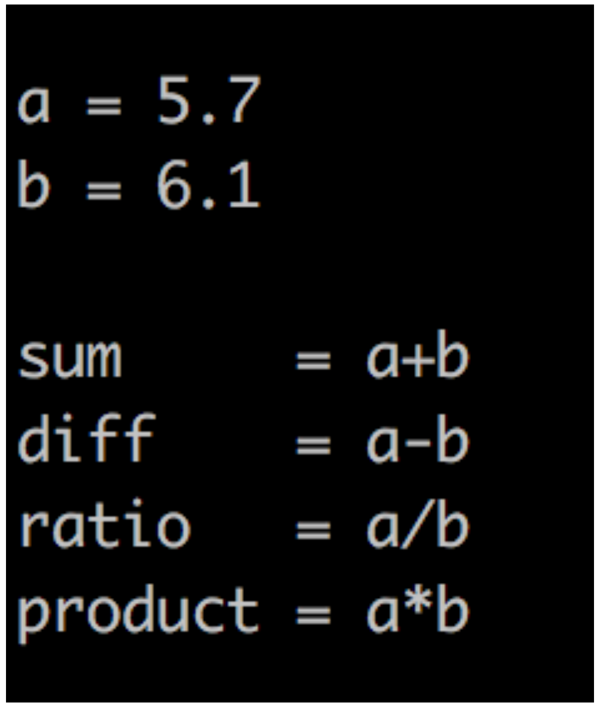
OPERATORS

- `Operators.py`

- Mathematical operations are required for this course; these are
  - addition: $A + B$
  - subtraction: $A - B$
  - multiplication: $A \times B$
  - division: A/B
  - Assignment: =
- The operations will be performed on *scalar* quantities as well as higher order representations such as vectors, matrices and tensors quantities.
  - Here we focus on operations on scalar quantities.

OPERATORS

- Operators.py

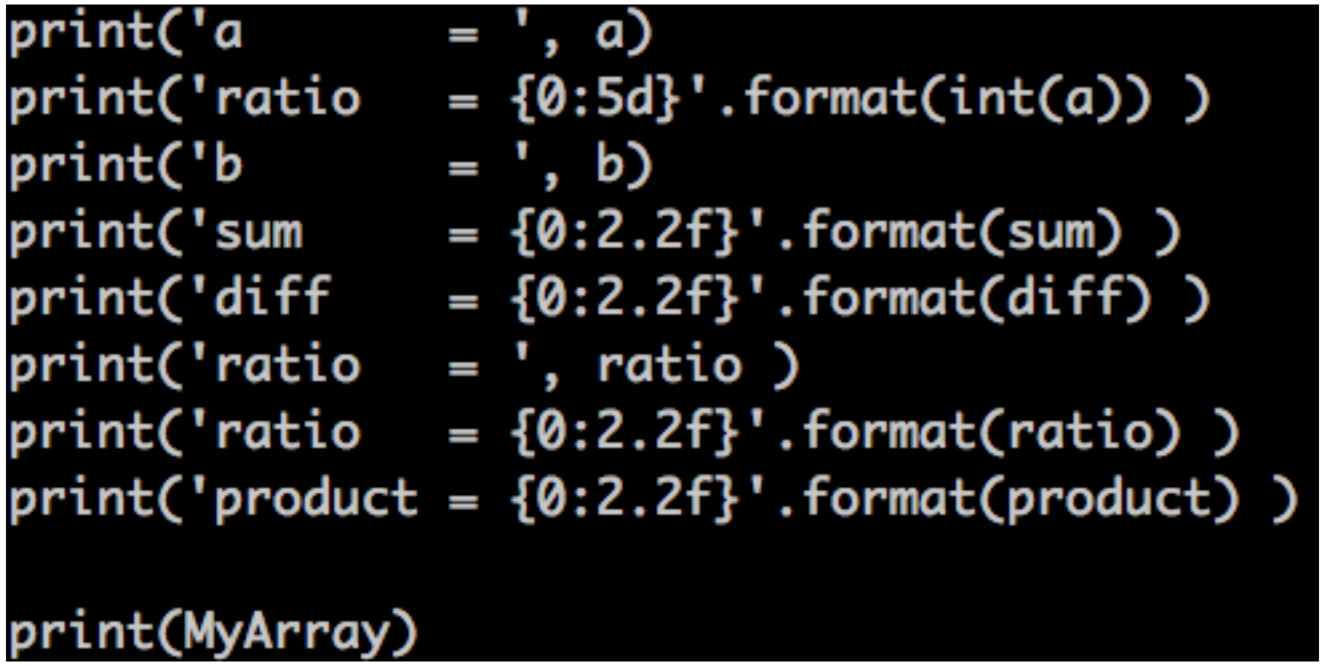


```
a = 5.7
b = 6.1

sum     = a+b
diff    = a-b
ratio   = a/b
product = a*b
```

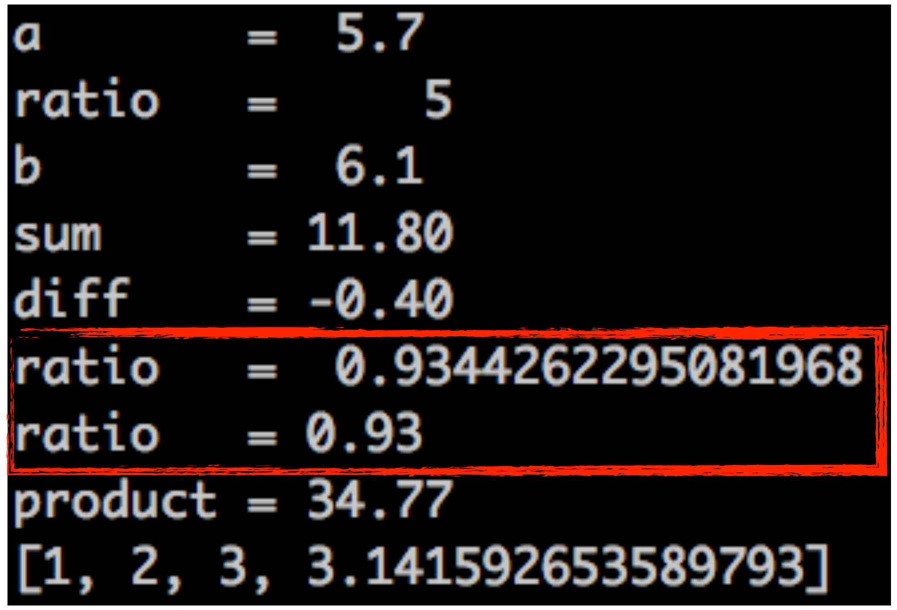
- We can assign values to variables called a and b. These are scalar numbers.
- Having initialised these variables we can use them to compute new quantities which can be assigned to new variables.

OPERATORS

- Operators.py



```
print('a       = ', a)
print('ratio   = {0:5d}'.format(int(a)) )
print('b       = ', b)
print('sum     = {0:2.2f}'.format(sum) )
print('diff    = {0:2.2f}'.format(diff) )
print('ratio   = ', ratio )
print('ratio   = {0:2.2f}'.format(ratio) )
print('product = {0:2.2f}'.format(product) )

print(MyArray)
```



```
a       =  5.7
ratio   =      5
b       =  6.1
sum     = 11.80
diff    = -0.40
ratio   =  0.9344262295081968
ratio   = 0.93
product = 34.77
[1, 2, 3, 3.141592653589793]
```

- This example uses print statement where python decides the formatting of the output, and where the user specifies the format.
- Use the format syntax to set the precision of the output [1].

[1] https://www.python.org/dev/peps/pep-3101/

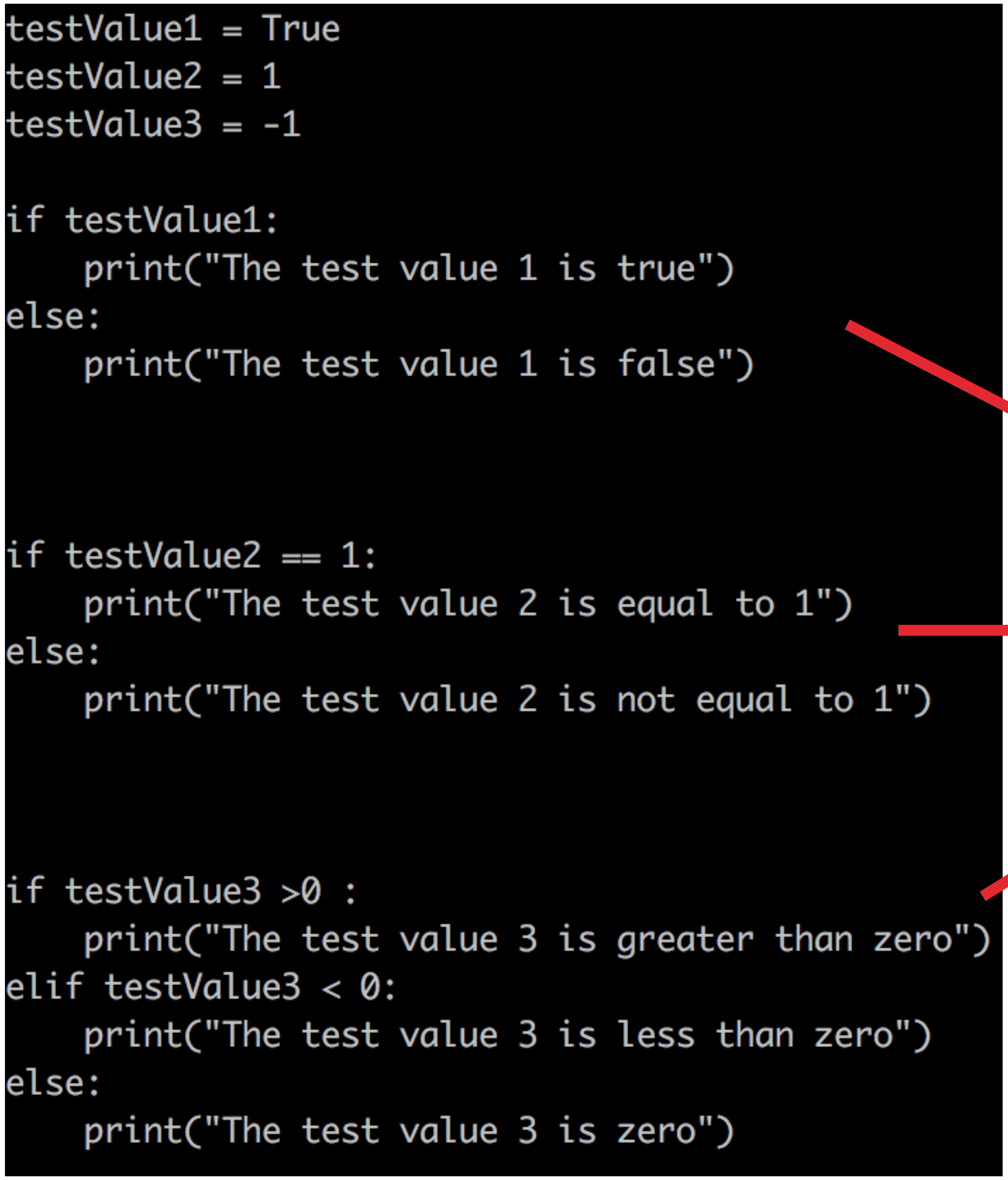
COMMANDS: CONDITIONALS

- `Conditionals.py`

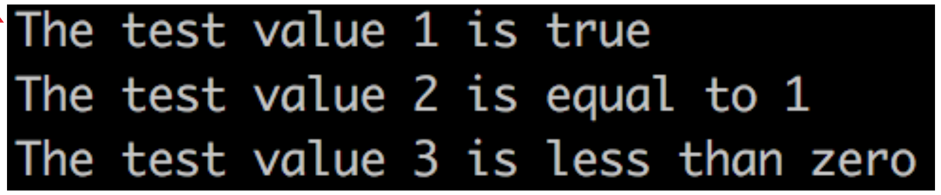
- Conditionals are required to enable decision making.
- Based on the value of some quantity (e.g. a boolean flag that has possible values of `True` and `False`) we can make a decision to do something or not.
- With a scalar value that is discrete or continuous we can use the value of a variable to do different things based on the values of that variable.

COMMANDS: CONDITIONALS

- Conditionals.py



```
testValue1 = True
testValue2 = 1
testValue3 = -1

if testValue1:
    print("The test value 1 is true")
else:
    print("The test value 1 is false")



if testValue2 == 1:
    print("The test value 2 is equal to 1")
else:
    print("The test value 2 is not equal to 1")



if testValue3 >0 :
    print("The test value 3 is greater than zero")
elif testValue3 < 0:
    print("The test value 3 is less than zero")
else:
    print("The test value 3 is zero")
```



```
The test value 1 is true
The test value 2 is equal to 1
The test value 3 is less than zero
```

COMMANDS: LOOPS

- `Loops.py`

- Having the ability to repeat commands is useful, and we will see this in the context of optimising parameters over some number of epochs for our neural networks.

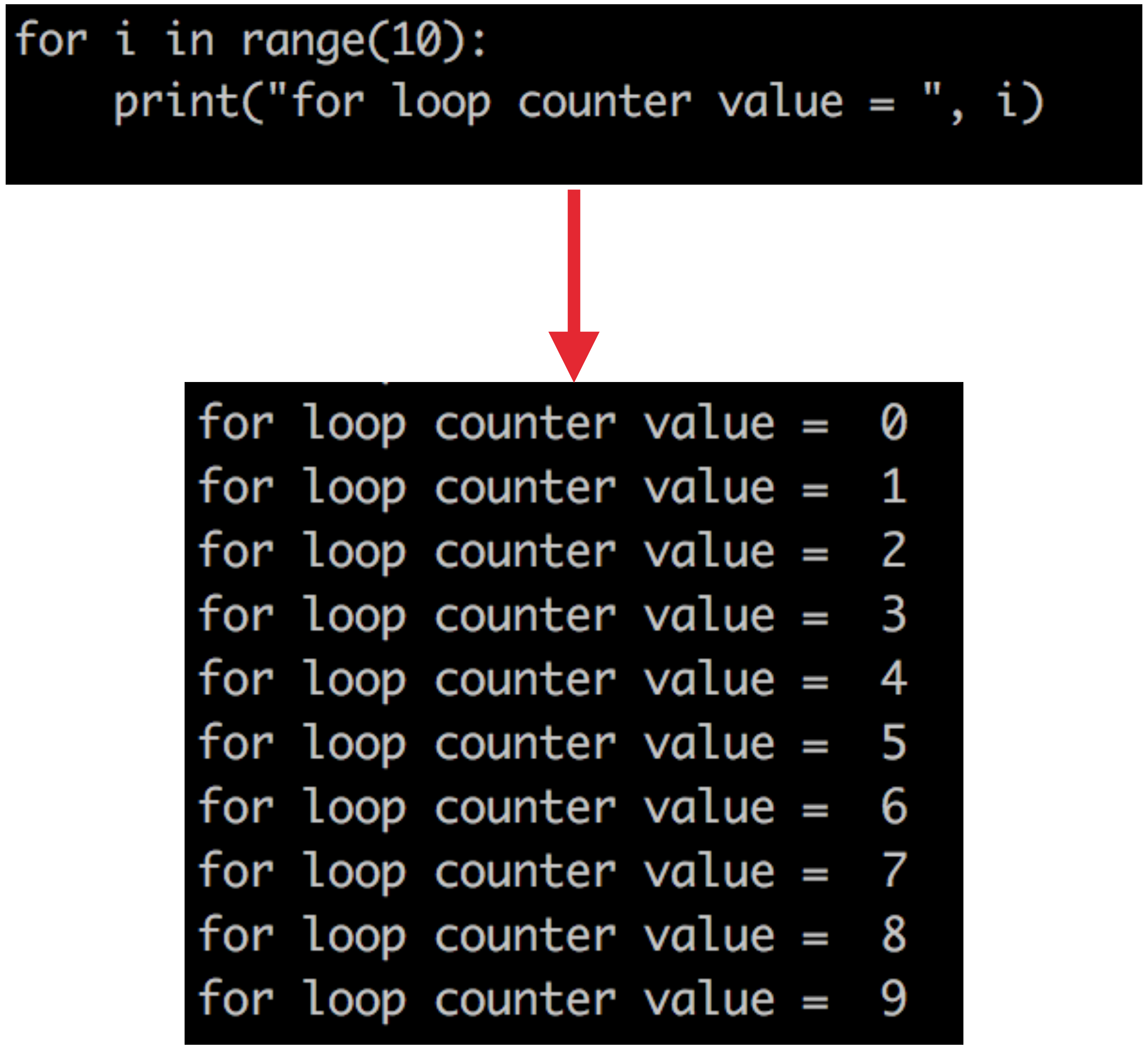
- This iteration is controlled using looping constructs within the Python language[1].
  - `for`
  - `while`

[1]We could write a recursive function to address this issue, however this would be more complicated than the example given for computing a factorial. Such an approach would add unnecessary complication, and the overhead of making a function call. It is often more elegant and efficient to use loops.

COMMANDS: LOOPS

- `Loops.py`

- An example `for` loop:



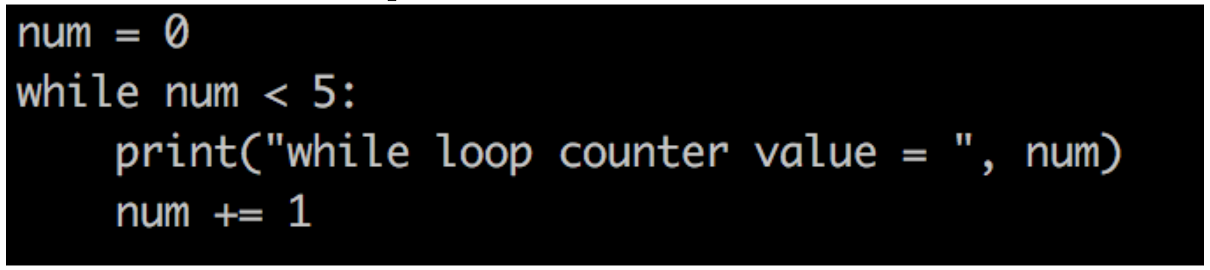
```
for i in range(10):
    print("for loop counter value = ", i)
```

```
for loop counter value =  0
for loop counter value =  1
for loop counter value =  2
for loop counter value =  3
for loop counter value =  4
for loop counter value =  5
for loop counter value =  6
for loop counter value =  7
for loop counter value =  8
for loop counter value =  9
```

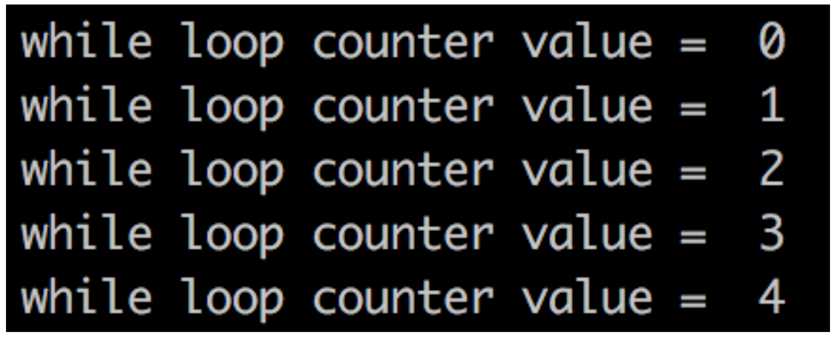
COMMANDS: LOOPS

- `Loops.py`

- An example `while` loop:



```
num = 0
while num < 5:
    print("while loop counter value = ", num)
    num += 1
```



```
while loop counter value =  0
while loop counter value =  1
while loop counter value =  2
while loop counter value =  3
while loop counter value =  4
```

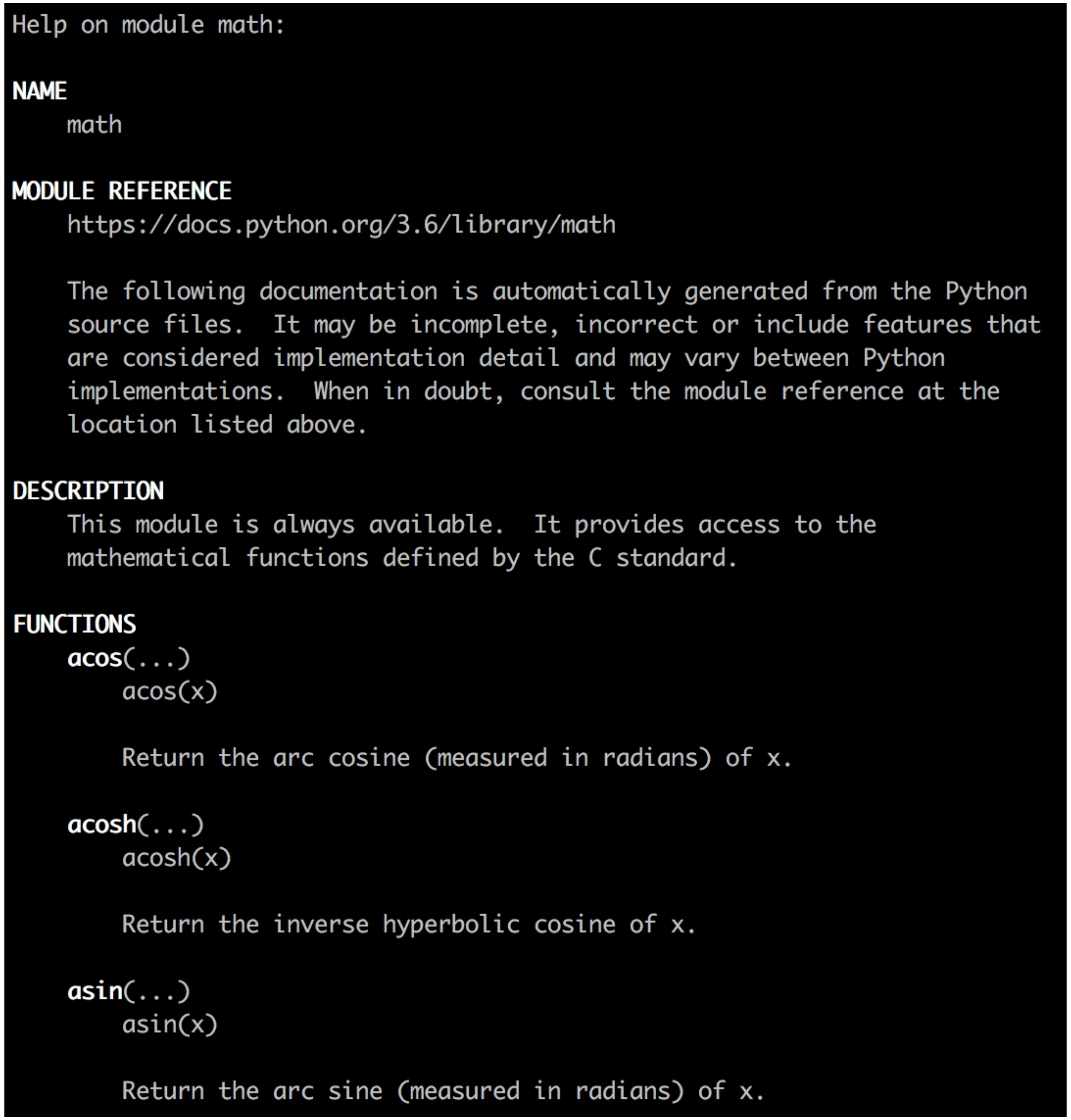
COMMANDS: BUILD IN COMMANDS

- Python modules contain commands that will be of use.
- To access these libraries you will need to import them into your python script.
- We saw `import math` in `Example.py` this is an interesting module that contains a number of useful functions.
- In the Spyder console window type

```
import math

help(math)
```

COMMANDS: BUILD IN COMMANDS

- You should see

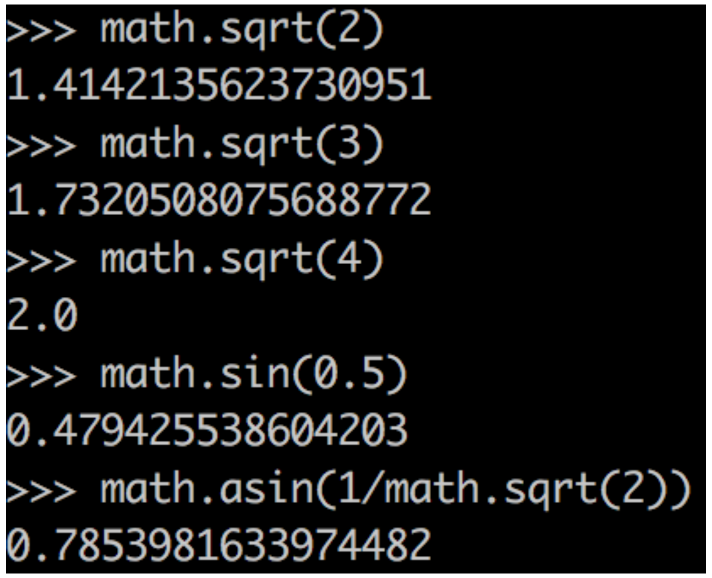


```
Help on module math:

NAME
    math

MODULE REFERENCE
    https://docs.python.org/3.6/library/math

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.

    acosh(...)
        acosh(x)

        Return the inverse hyperbolic cosine of x.

    asin(...)
        asin(x)

        Return the arc sine (measured in radians) of x.
```

etc.

COMMANDS: BUILD IN COMMANDS

- These commands operate on scalar quantities. That is you take some number as input and use that when using the built in command.



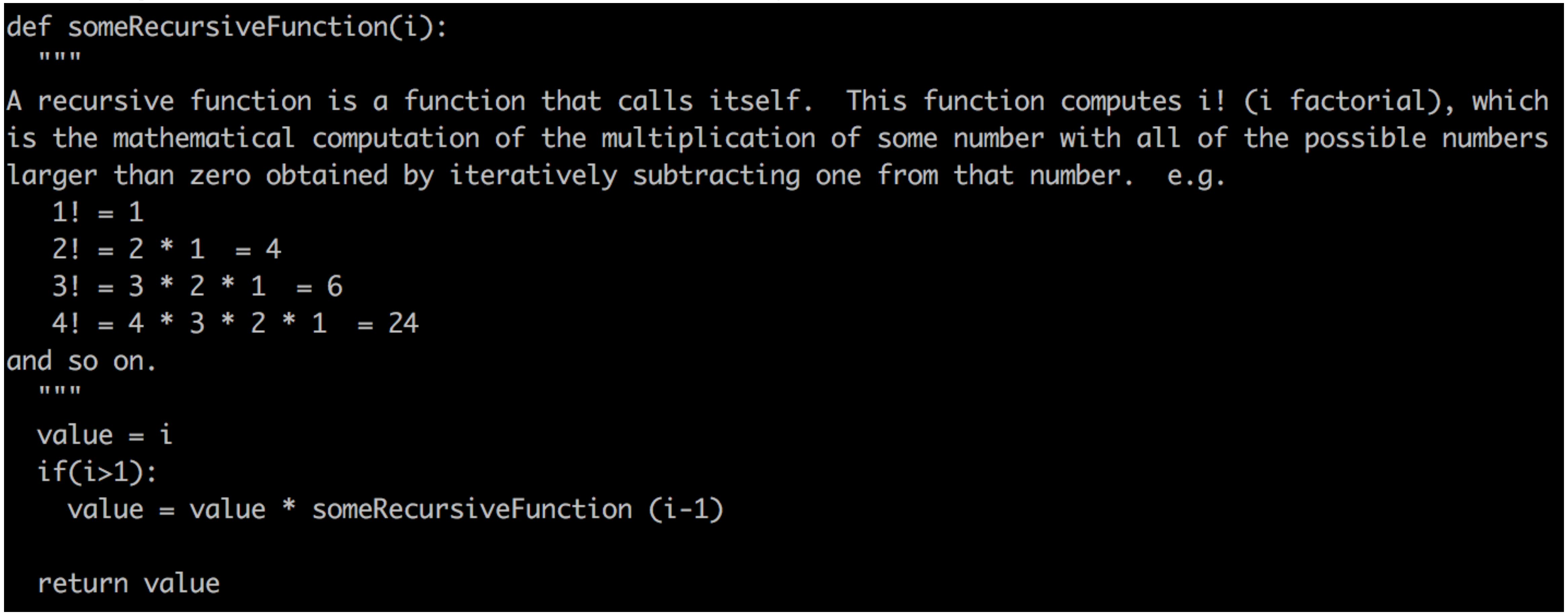
```
>>> math.sqrt(2)
1.4142135623730951
>>> math.sqrt(3)
1.7320508075688772
>>> math.sqrt(4)
2.0
>>> math.sin(0.5)
0.479425538604203
>>> math.asin(1/math.sqrt(2))
0.7853981633974482
```

- The built in commands in math are accessed by prefixing the command with `math.`
- These are all examples of calling a function.
- Also see `BuiltInCommands.py`.

FUNCTIONS

- `Functions.py`

- Functions are defined using the `def` keyword followed by the name of the function.
- A list of arguments passed to the function are included in parentheses, followed by a colon.
- The scope of the function is denoted by an indentation of commands.



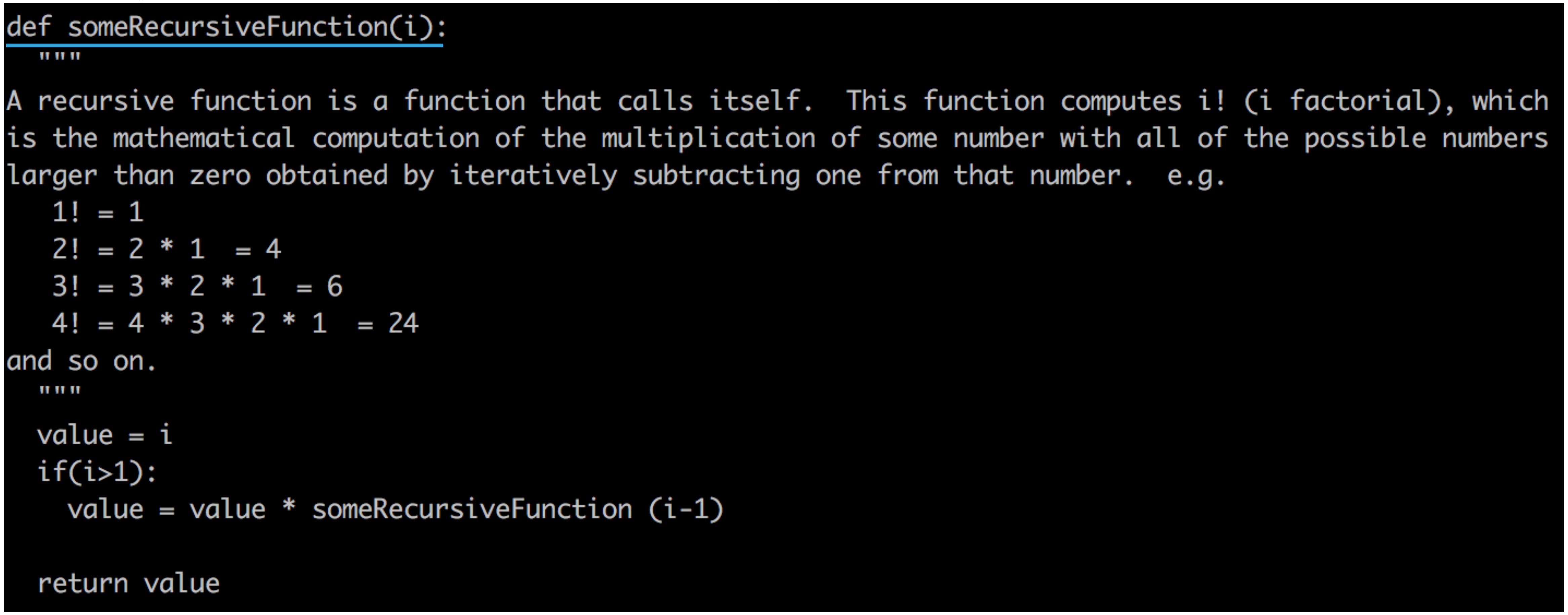
```
def someRecursiveFunction(i):
  """
A recursive function is a function that calls itself.  This function computes i! (i factorial), which
is the mathematical computation of the multiplication of some number with all of the possible numbers
larger than zero obtained by iteratively subtracting one from that number.  e.g.
   1! = 1
   2! = 2 * 1  = 4
   3! = 3 * 2 * 1  = 6
   4! = 4 * 3 * 2 * 1  = 24
and so on.
  """
  value = i
  if(i>1):
    value = value * someRecursiveFunction (i-1)

  return value
```

- These commands are only executed when the function is called.

FUNCTIONS

- `Functions.py`

- Functions are defined using the `def` keyword followed by the name of the function.
- A list of arguments passed to the function are included in parentheses, followed by a colon.
- The scope of the function is denoted by an indentation of commands.



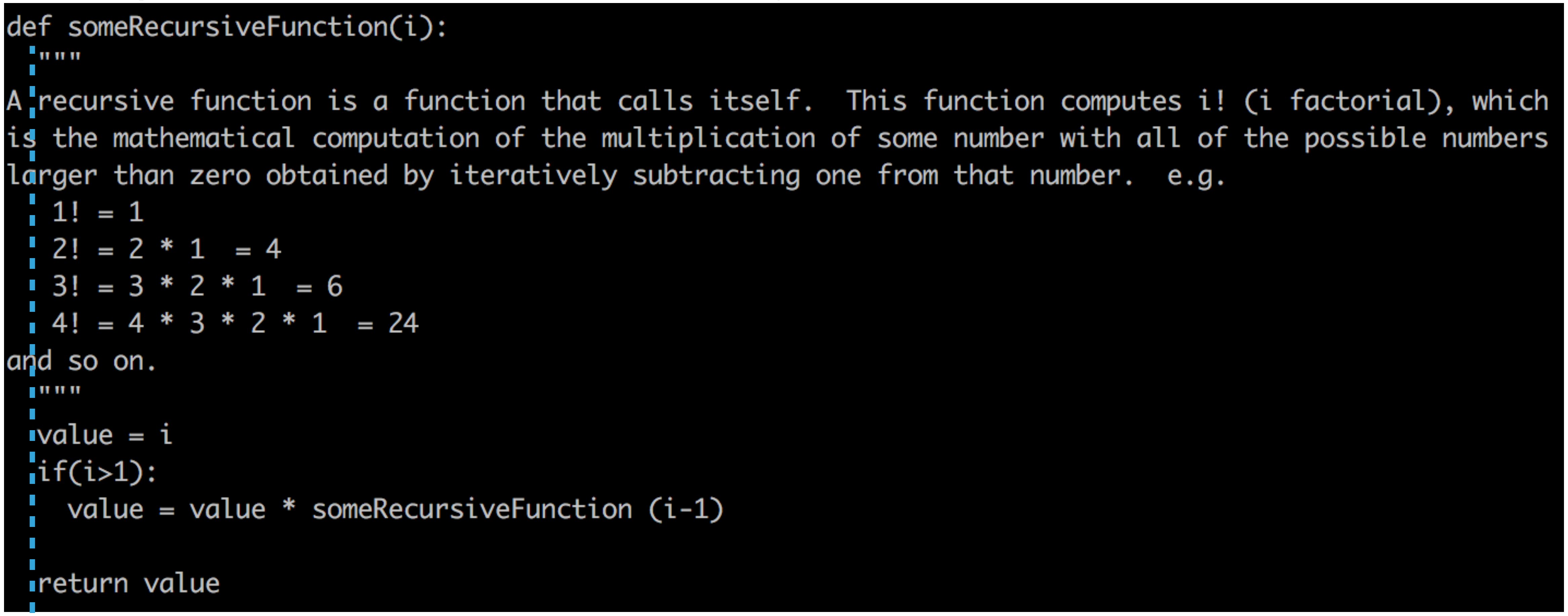
```
def someRecursiveFunction(i):
  """
A recursive function is a function that calls itself.  This function computes i! (i factorial), which
is the mathematical computation of the multiplication of some number with all of the possible numbers
larger than zero obtained by iteratively subtracting one from that number.  e.g.
   1! = 1
   2! = 2 * 1  = 4
   3! = 3 * 2 * 1  = 6
   4! = 4 * 3 * 2 * 1  = 24
and so on.
  """
  value = i
  if(i>1):
    value = value * someRecursiveFunction (i-1)

  return value
```

- These commands are only executed when the function is called.

FUNCTIONS

- `Functions.py`

- Functions are defined using the `def` keyword followed by the name of the function.
- A list of arguments passed to the function are included in parentheses, followed by a colon.
- The scope of the function is denoted by an indentation of commands.



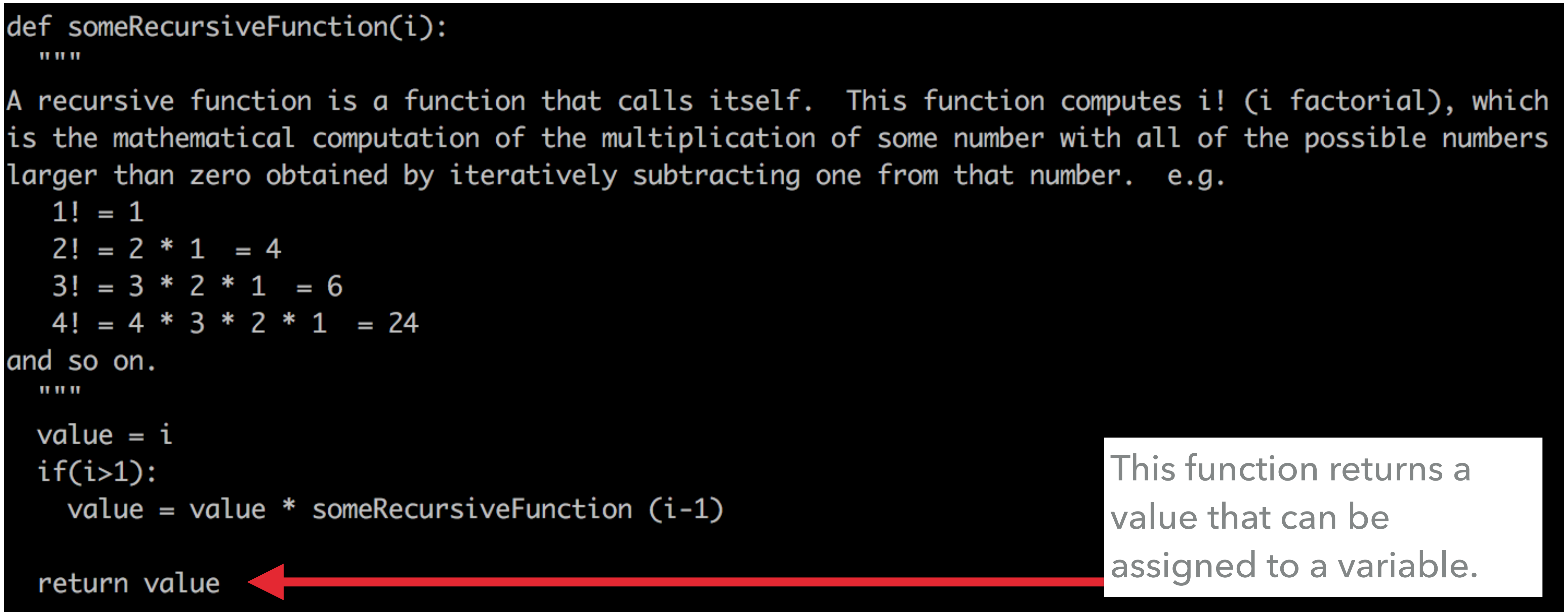
```
def someRecursiveFunction(i):
    """
A recursive function is a function that calls itself.  This function computes i! (i factorial), which
is the mathematical computation of the multiplication of some number with all of the possible numbers
larger than zero obtained by iteratively subtracting one from that number.  e.g.
   1! = 1
   2! = 2 * 1  = 4
   3! = 3 * 2 * 1  = 6
   4! = 4 * 3 * 2 * 1  = 24
and so on.
    """
    value = i
    if(i>1):
      value = value * someRecursiveFunction (i-1)

    return value
```

- These commands are only executed when the function is called.

FUNCTIONS

- `Functions.py`

- Functions are defined using the `def` keyword followed by the name of the function.
- A list of arguments passed to the function are included in parentheses, followed by a colon.
- The scope of the function is denoted by an indentation of commands.



```
def someRecursiveFunction(i):
  """
A recursive function is a function that calls itself.  This function computes i! (i factorial), which
is the mathematical computation of the multiplication of some number with all of the possible numbers
larger than zero obtained by iteratively subtracting one from that number.  e.g.
   1! = 1
   2! = 2 * 1  = 4
   3! = 3 * 2 * 1  = 6
   4! = 4 * 3 * 2 * 1  = 24
and so on.
  """
  value = i
  if(i>1):
    value = value * someRecursiveFunction (i-1)

  return value
```

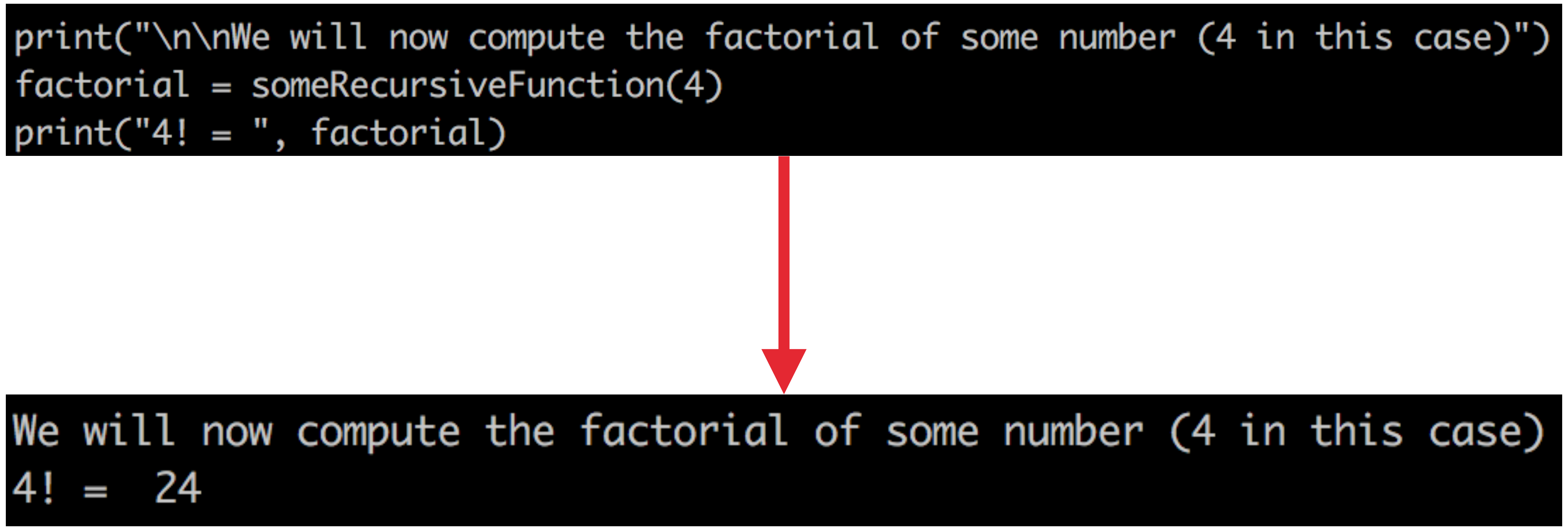
This function returns a value that can be assigned to a variable.

- These commands are only executed when the function is called.

FUNCTIONS

- `Functions.py`

- These commands are only executed when the function is called.



```
print("\n\nWe will now compute the factorial of some number (4 in this case)")
factorial = someRecursiveFunction(4)
print("4! = ", factorial)
```

```
We will now compute the factorial of some number (4 in this case)
4! =  24
```

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

- The function call provides the expected result.

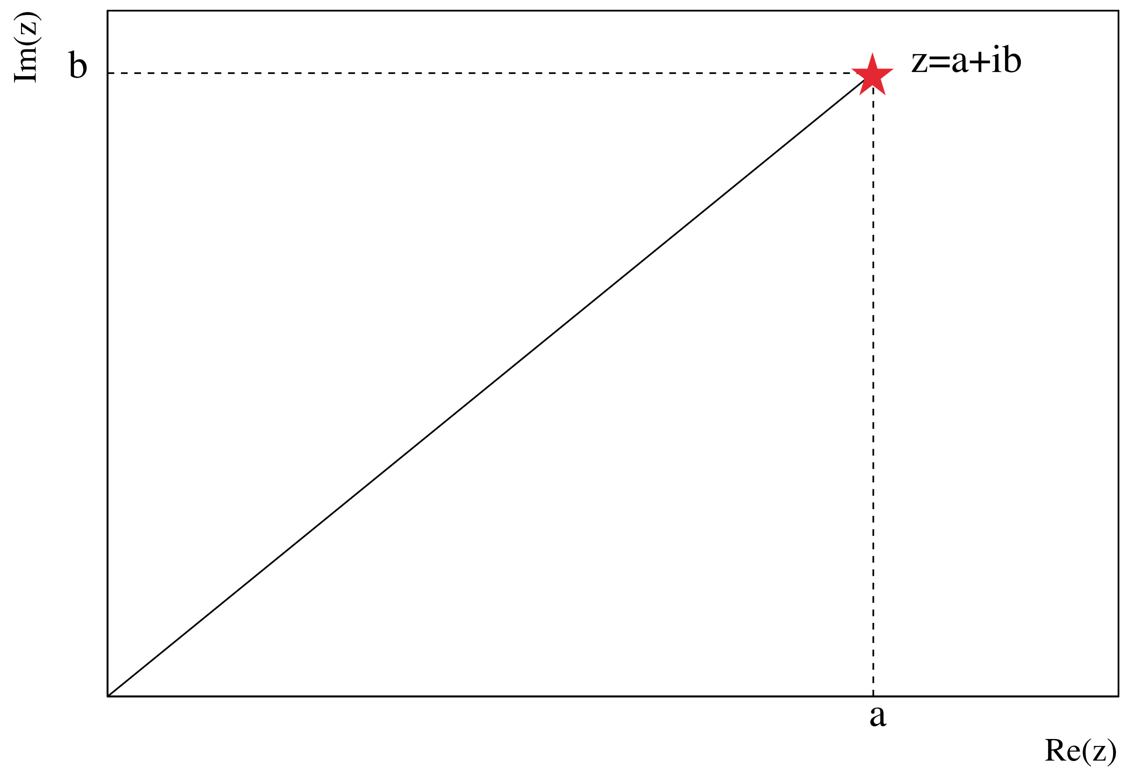
- The script includes a second function (`someFunction`) that calculates a few quantities and prints them out.

CLASSES

- Code quickly becomes disorganised and hard to maintain. One thing that helps keep code maintainable is a class.
- This encapsulates code in an object of some defined type.
- classes have:
  - member functions (functions that can be called only for instances of a class)
  - data members (variables of the instance of a class)
- Typically related to some given context.

CLASSES

- Consider a complex number.
    - Represented by two pieces of information: a real number and an imaginary number[1].
    - These number can be thought of in terms of the distance along the horizontal and vertical axes of a complex plane.
    - We can write a class to represent the complex number.



$$
\begin{aligned}
z &= a + ib \\
&= re^{i\theta} \\
&= r(cos\theta + i\sin\theta)
\end{aligned}
$$

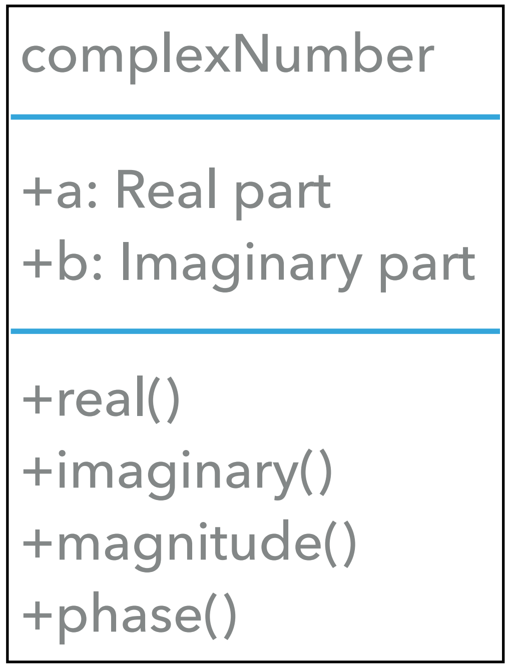
$$r = \sqrt{a^2 + b^2}$$

$$\theta = \mathrm{atan}(b/a)$$

[1]An imaginary number i = √-1, such that i^2 = (-1)(-1) = 1

CLASSES

- `Classes.py`

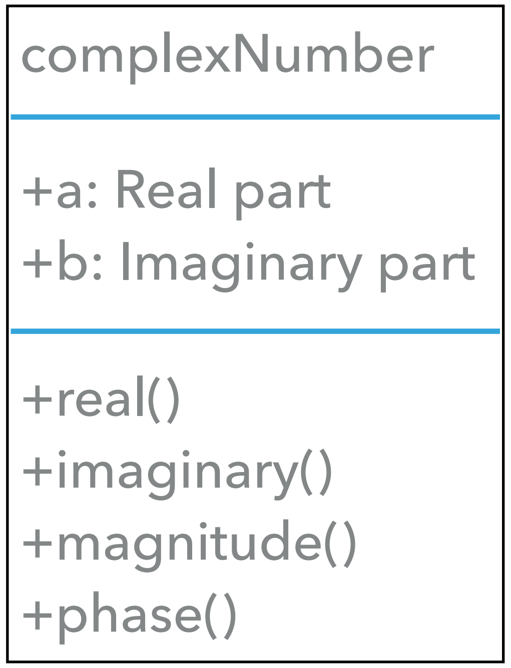


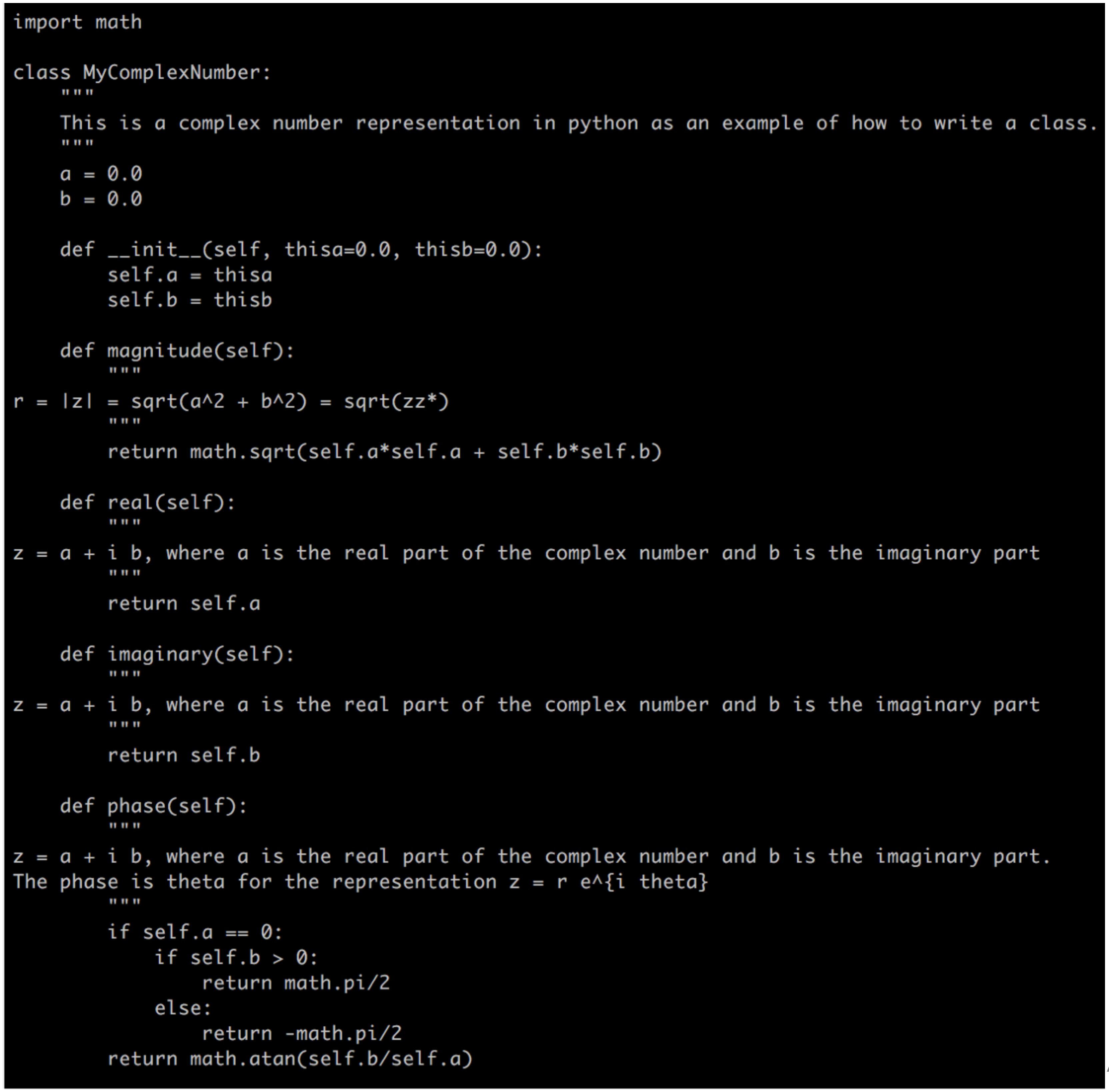
- To represent a complex number we need a class (the class name is complexNumber)
- This requires two data members; we can choose to use cartesian (a, b) or polar (r, θ) coordinates.
- The class will allow us to obtain the values of a, b, r and θ.[1]

[1]If this was a Physics course we might want to include additional functions related to complex number arithmetic. However, before getting to excited about that we should also note that Python already has a complex number class. See https://docs.python.org/3.0/library/cmath.html.

CLASSES

- Classes.py

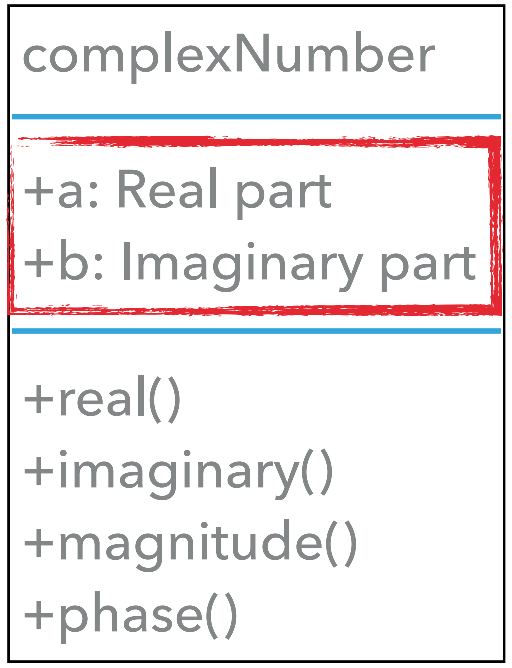


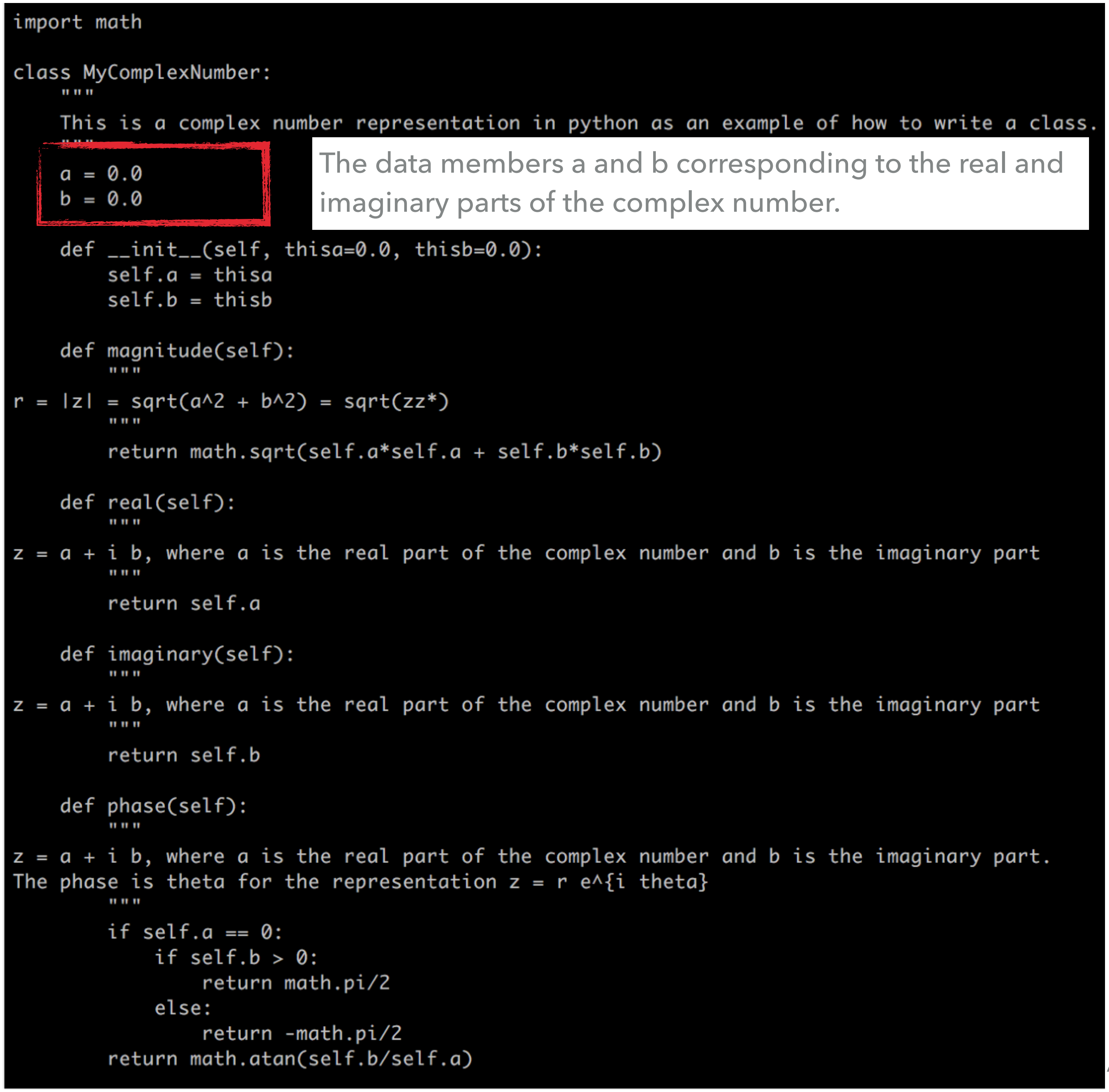


```
import math

class MyComplexNumber:
    """
    This is a complex number representation in python as an example of how to write a class.
    """
    a = 0.0
    b = 0.0

    def __init__(self, thisa=0.0, thisb=0.0):
        self.a = thisa
        self.b = thisb

    def magnitude(self):
        """
r = |z| = sqrt(a^2 + b^2) = sqrt(zz*)
        """
        return math.sqrt(self.a*self.a + self.b*self.b)

    def real(self):
        """
z = a + i b, where a is the real part of the complex number and b is the imaginary part
        """
        return self.a

    def imaginary(self):
        """
z = a + i b, where a is the real part of the complex number and b is the imaginary part
        """
        return self.b

    def phase(self):
        """
z = a + i b, where a is the real part of the complex number and b is the imaginary part.
The phase is theta for the representation z = r e^{i theta}
        """
        if self.a == 0:
            if self.b > 0:
                return math.pi/2
            else:
                return -math.pi/2
        return math.atan(self.b/self.a)
```

CLASSES

- Classes.py



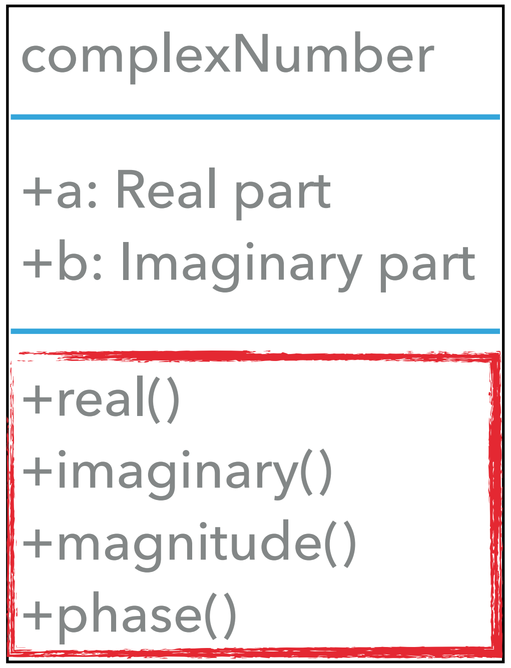


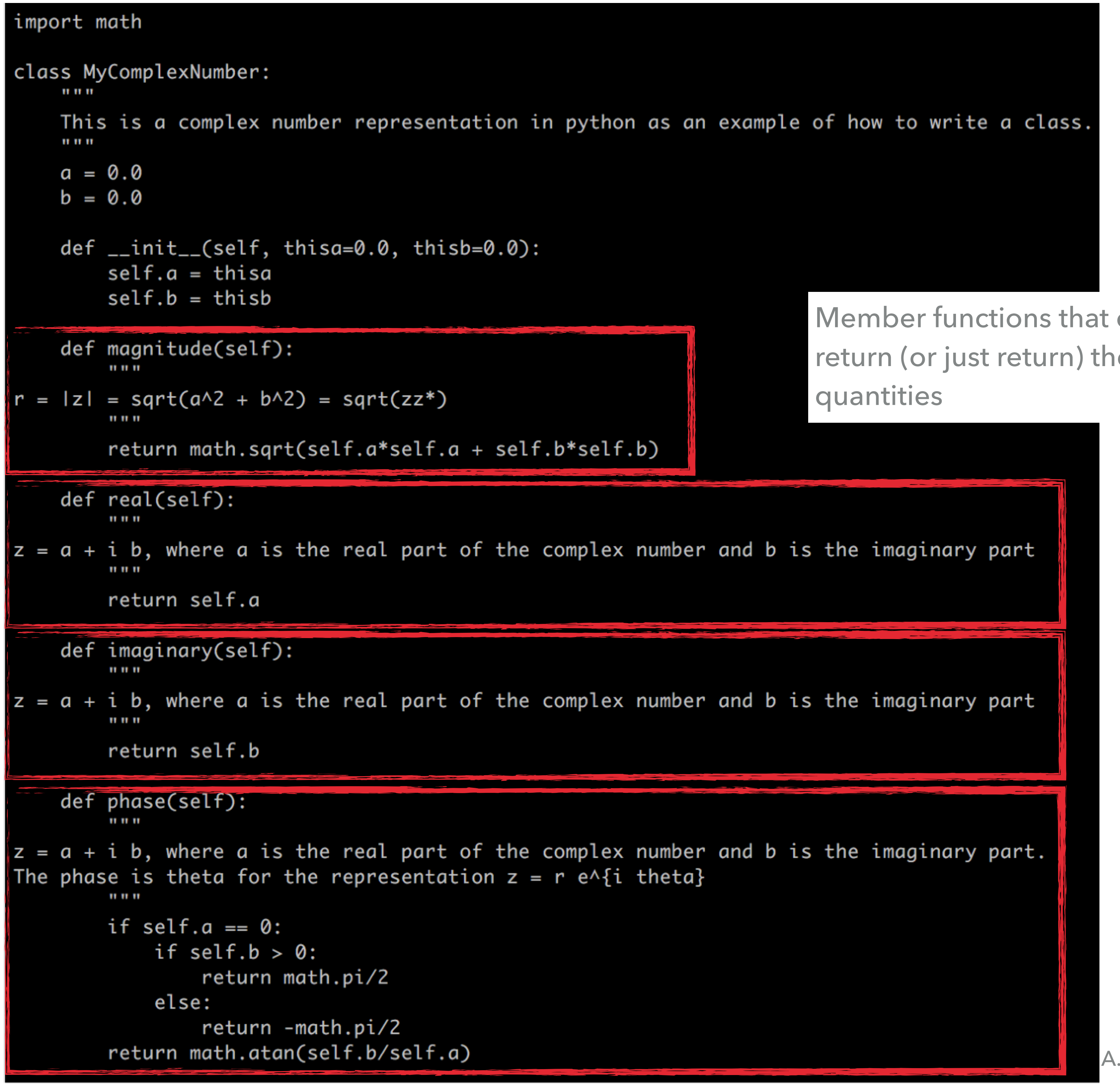
```
import math

class MyComplexNumber:
    """
    This is a complex number representation in python as an example of how to write a class.
    """
    a = 0.0
    b = 0.0

    def __init__(self, thisa=0.0, thisb=0.0):
        self.a = thisa
        self.b = thisb

    def magnitude(self):
        """
r = |z| = sqrt(a^2 + b^2) = sqrt(zz*)
        """
        return math.sqrt(self.a*self.a + self.b*self.b)

    def real(self):
        """
z = a + i b, where a is the real part of the complex number and b is the imaginary part
        """
        return self.a

    def imaginary(self):
        """
z = a + i b, where a is the real part of the complex number and b is the imaginary part
        """
        return self.b

    def phase(self):
        """
z = a + i b, where a is the real part of the complex number and b is the imaginary part.
The phase is theta for the representation z = r e^{i theta}
        """
        if self.a == 0:
            if self.b > 0:
                return math.pi/2
            else:
                return -math.pi/2
        return math.atan(self.b/self.a)
```

The data members a and b corresponding to the real and imaginary parts of the complex number.

CLASSES

- `Classes.py`





```
import math

class MyComplexNumber:
    """
    This is a complex number representation in python as an example of how to write a class.
    """
    a = 0.0
    b = 0.0

    def __init__(self, thisa=0.0, thisb=0.0):
        self.a = thisa
        self.b = thisb

    def magnitude(self):
        """
r = |z| = sqrt(a^2 + b^2) = sqrt(zz*)
        """
        return math.sqrt(self.a*self.a + self.b*self.b)

    def real(self):
        """
z = a + i b, where a is the real part of the complex number and b is the imaginary part
        """
        return self.a

    def imaginary(self):
        """
z = a + i b, where a is the real part of the complex number and b is the imaginary part
        """
        return self.b

    def phase(self):
        """
z = a + i b, where a is the real part of the complex number and b is the imaginary part.
The phase is theta for the representation z = r e^{i theta}
        """
        if self.a == 0:
            if self.b > 0:
                return math.pi/2
            else:
                return -math.pi/2
        return math.atan(self.b/self.a)
```

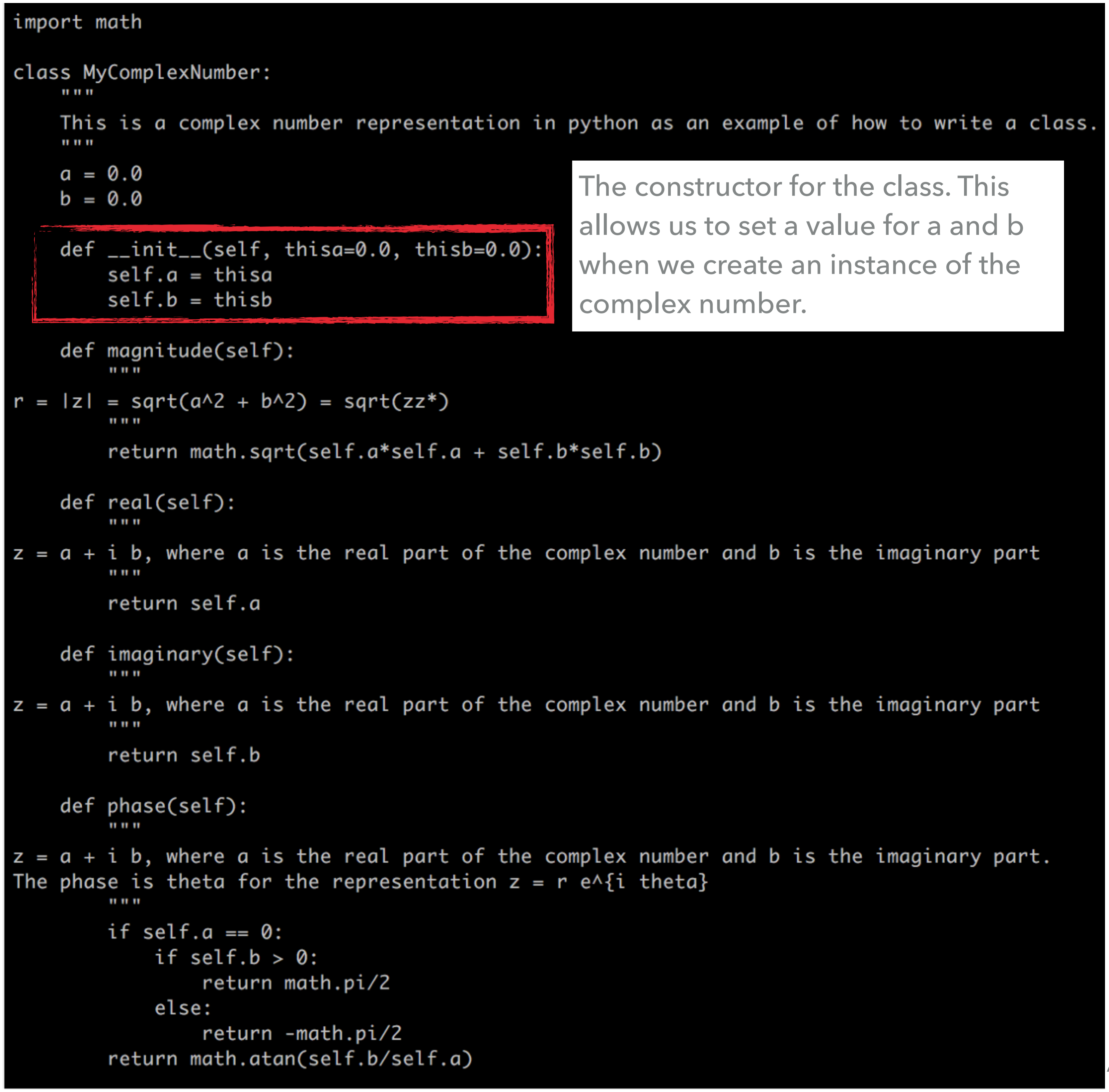
Member functions that calculate and return (or just return) the requested quantities

CLASSES

- Classes.py

complexNumber
+a: Real part +b: Imaginary part

+real()
+imaginary()
+magnitude()
+phase()



```
import math

class MyComplexNumber:
    """
    This is a complex number representation in python as an example of how to write a class.
    """
    a = 0.0
    b = 0.0

    def __init__(self, thisa=0.0, thisb=0.0):
        self.a = thisa
        self.b = thisb

    def magnitude(self):
        """
r = |z| = sqrt(a^2 + b^2) = sqrt(zz*)
        """
        return math.sqrt(self.a*self.a + self.b*self.b)

    def real(self):
        """
z = a + i b, where a is the real part of the complex number and b is the imaginary part
        """
        return self.a

    def imaginary(self):
        """
z = a + i b, where a is the real part of the complex number and b is the imaginary part
        """
        return self.b

    def phase(self):
        """
z = a + i b, where a is the real part of the complex number and b is the imaginary part.
The phase is theta for the representation z = r e^{i theta}
        """
        if self.a == 0:
            if self.b > 0:
                return math.pi/2
            else:
                return -math.pi/2
        return math.atan(self.b/self.a)
```

The constructor for the class. This allows us to set a value for a and b when we create an instance of the complex number.

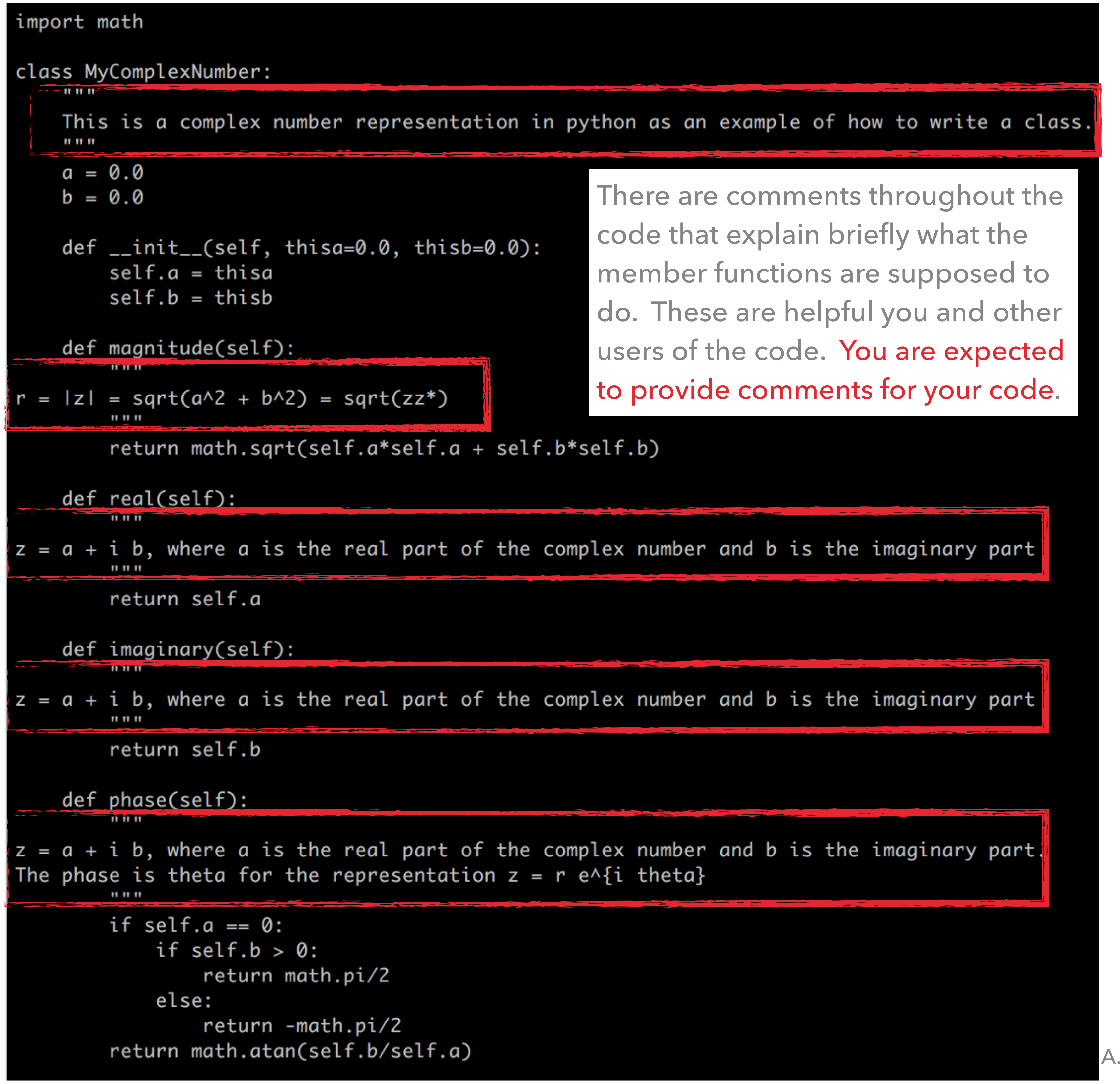
CLASSES

- `Classes.py`

complexNumber

+a: Real part
+b: Imaginary part

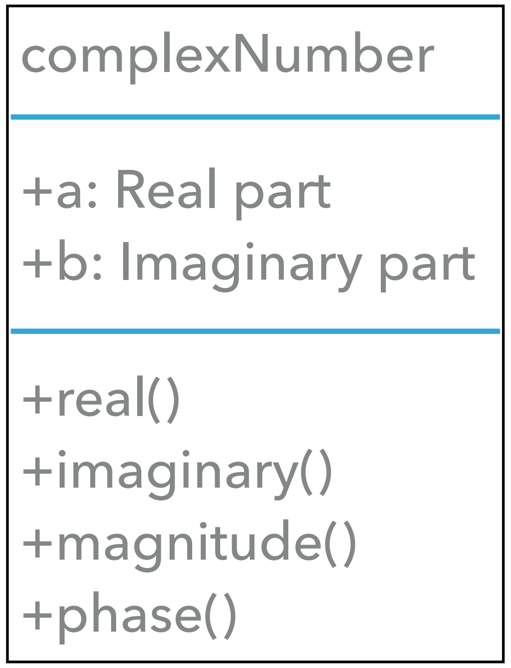
+real()
+imaginary()
+magnitude()
+phase()



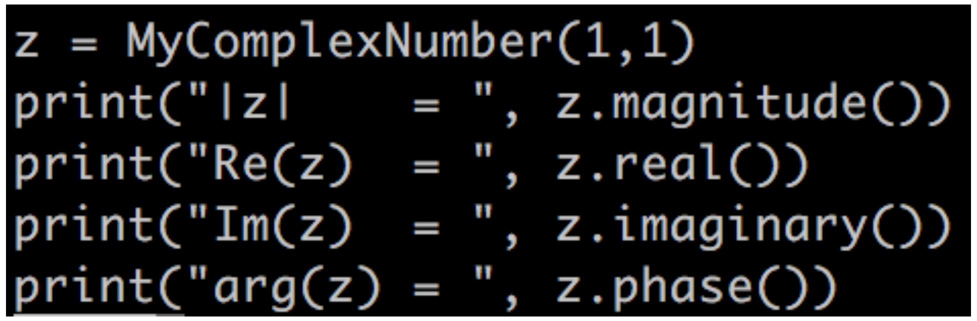
```
import math

class MyComplexNumber:
    """
    This is a complex number representation in python as an example of how to write a class.
    """
    a = 0.0
    b = 0.0

    def __init__(self, thisa=0.0, thisb=0.0):
        self.a = thisa
        self.b = thisb

    def magnitude(self):
        """
r = |z| = sqrt(a^2 + b^2) = sqrt(zz*)
        """
        return math.sqrt(self.a*self.a + self.b*self.b)

    def real(self):
        """
z = a + i b, where a is the real part of the complex number and b is the imaginary part
        """
        return self.a

    def imaginary(self):
        """
z = a + i b, where a is the real part of the complex number and b is the imaginary part
        """
        return self.b

    def phase(self):
        """
z = a + i b, where a is the real part of the complex number and b is the imaginary part.
The phase is theta for the representation z = r e^{i theta}
        """
        if self.a == 0:
            if self.b > 0:
                return math.pi/2
            else:
                return -math.pi/2
        return math.atan(self.b/self.a)
```

There are comments throughout the code that explain briefly what the member functions are supposed to do. These are helpful you and other users of the code. You are expected to provide comments for your code.

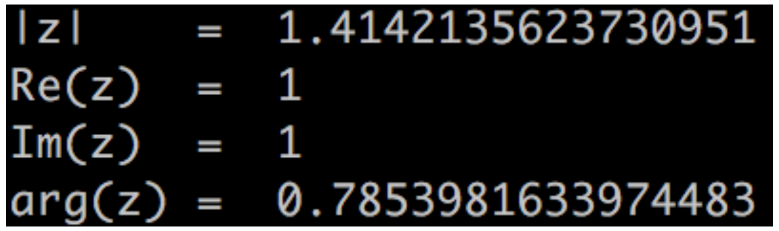
CLASSES



- `Classes.py`



```
z = MyComplexNumber(1,1)
print("|z|    = ", z.magnitude())
print("Re(z)  = ", z.real())
print("Im(z)  = ", z.imaginary())
print("arg(z) = ", z.phase())
```



```
|z|    =  1.4142135623730951
Re(z)  =  1
Im(z)  =  1
arg(z) =  0.7853981633974483
```

The print statement is formatted in such a way that the code is straightforward to read… and the output is easy to interpret.

- z is instantiated as an object of type MyComplexNumber, with a=1 and b=1.
- The values evaluated for the different member functions are printed out.

CLASSES: EXERCISE

- Make a copy of the `Examples.py` script and call this `YOURNAME_Complex.py`.
- Incorporate the code from the Classes.py script and adapt this to do the following:
  - Allow a user to change the real and imaginary parts by calling functions to:
    - (i) set new values for either a or b;
    - (ii) set new values for r and θ.
  - Add printouts of function calls to demonstrate that the modifications work.
  - Make sure the code is documented (comments, self-evident printouts, etc.)

Hint - to do this you need to add three functions to the class: **one to change a**, **one to change b** and **one to use a value of r and θ to update a and b**.[1]

See Week 1 Assignments

[1]If you are an experienced coder you may wish to re-write the class to use values of r and θ as the data members.

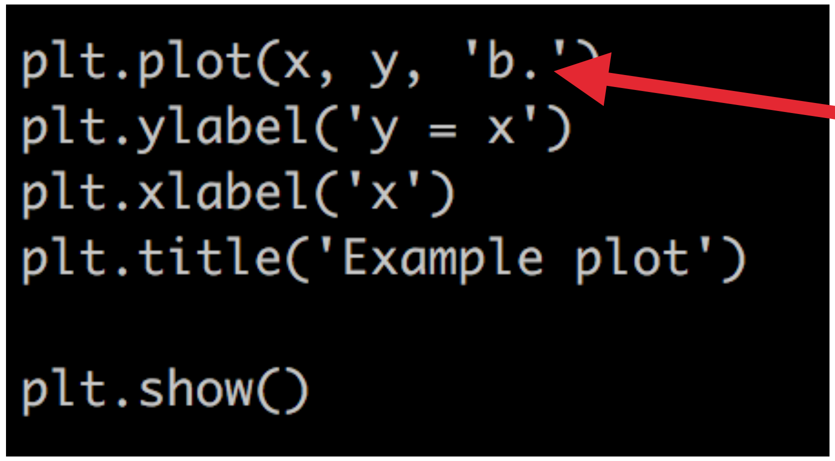
MATPLOTLIB: PLOTTING ‣ `Plotting.py`

- matplotlib is one of the plotting modules available for Python. This allows the user to make histograms, scatter plots and output more complicated graphics.
    - We are going to use pyplot in this course. See Ref [1] for more information about this module.
    - The example script shows you how to make a single plot and a group of plots, and how to save these to a file.
    - In addition to plotting graphs, there is an example of how to plot a histogram.

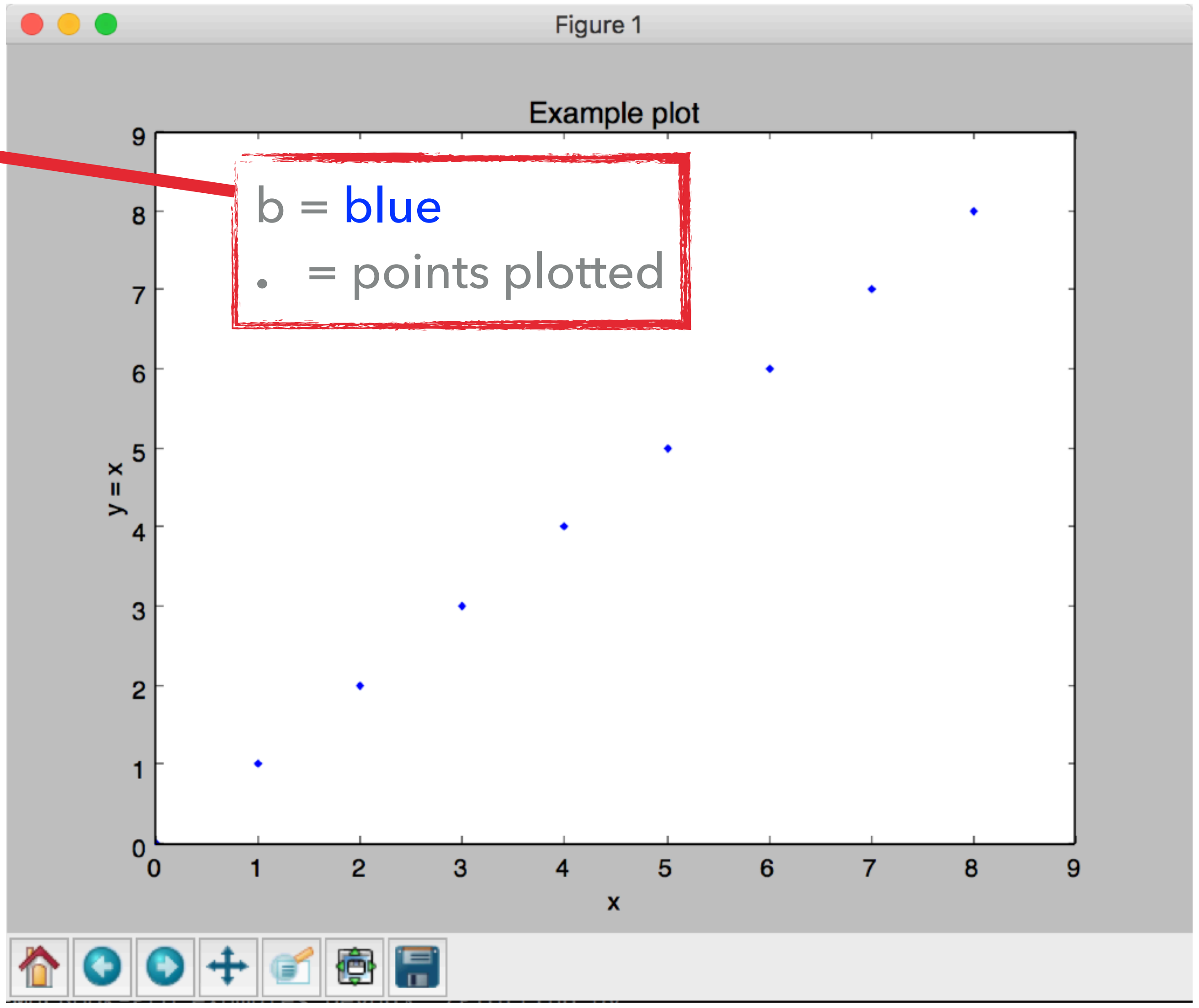
[1] See https://matplotlib.org/users/pyplot_tutorial.html for more details.

MATPLOTLIB: PLOTTING ‣ `Plotting.py`

- `import matplotlib.pyplot as plt`
- `x` and `y` are lists of data (pairs of x, y values)



```
plt.plot(x, y, 'b.')
plt.ylabel('y = x')
plt.xlabel('x')
plt.title('Example plot')

plt.show()
```



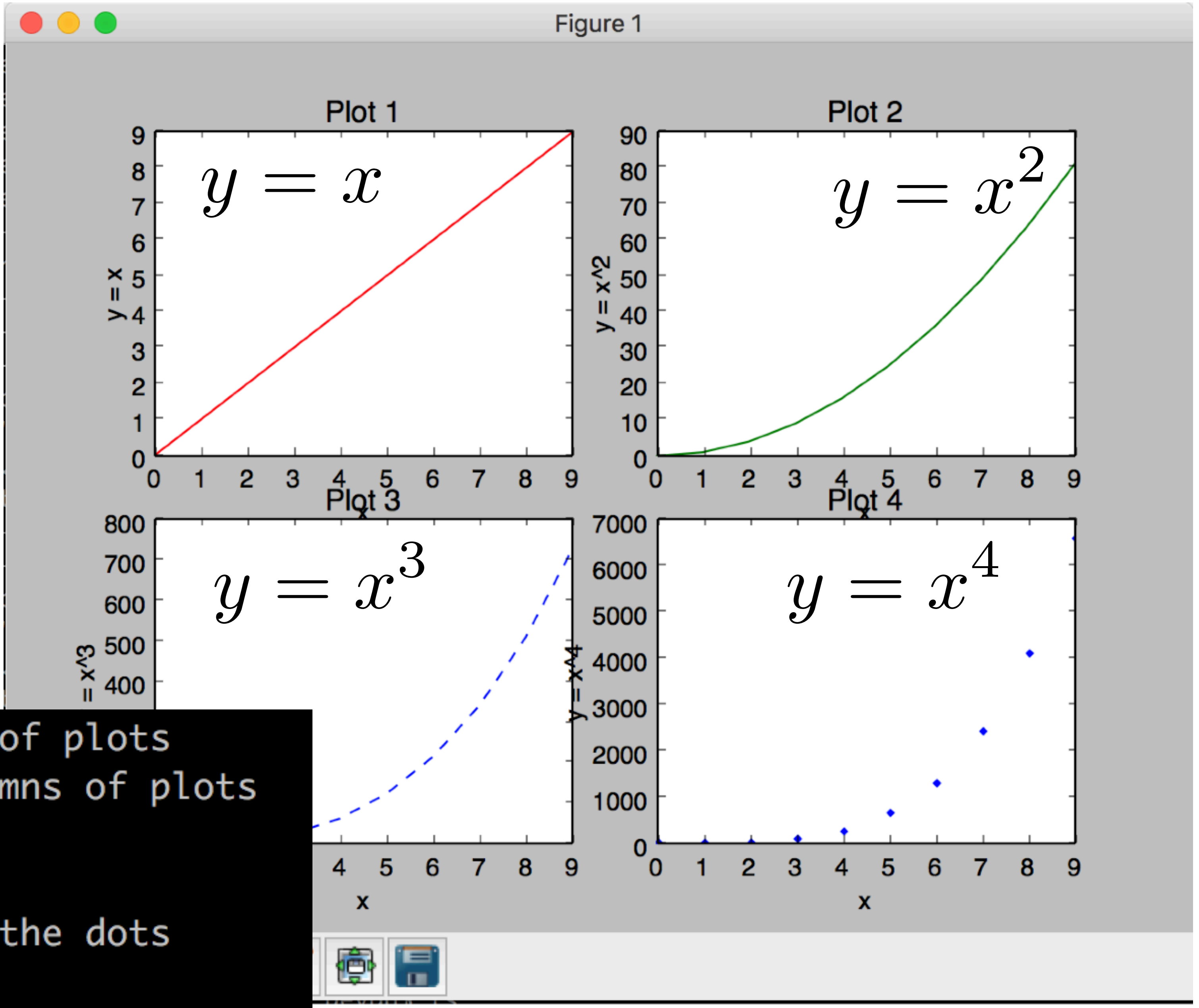
`plt.show()`

- results in the plot being displayed.

[1] See https://matplotlib.org/users/pyplot_tutorial.html for more details.

MATPLOTLIB: PLOTTING ‣ Plotting.py

- The second part of the script uses subplots to make a 2x2 array of plots to show four functions on different panels.



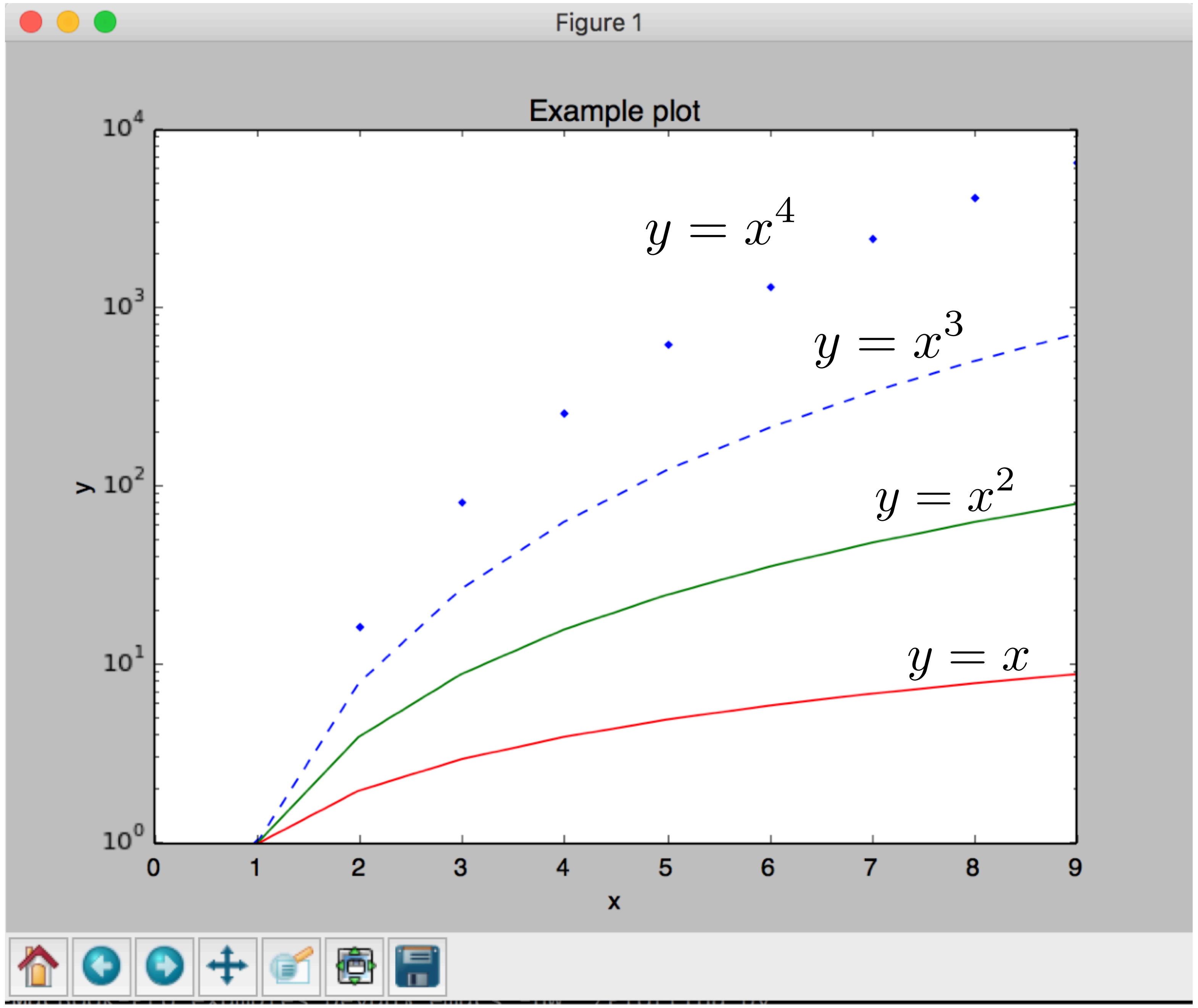
```
# the first number is the number of rows of plots
# the second number is the number of columns of plots
# the third number is the plot number
plt.subplot(2,2,1)
plt.plot(x, y, 'r') # a red line joining the dots
plt.ylabel('y = x')
plt.xlabel('x')
plt.title('Plot 1')
```

[1] See https://matplotlib.org/users/pyplot_tutorial.html for more details.

MATPLOTLIB: PLOTTING ▸ Plotting.py

- Sometimes it is better to overlay the plots.
- These functions are better represented on a log scale vertically.

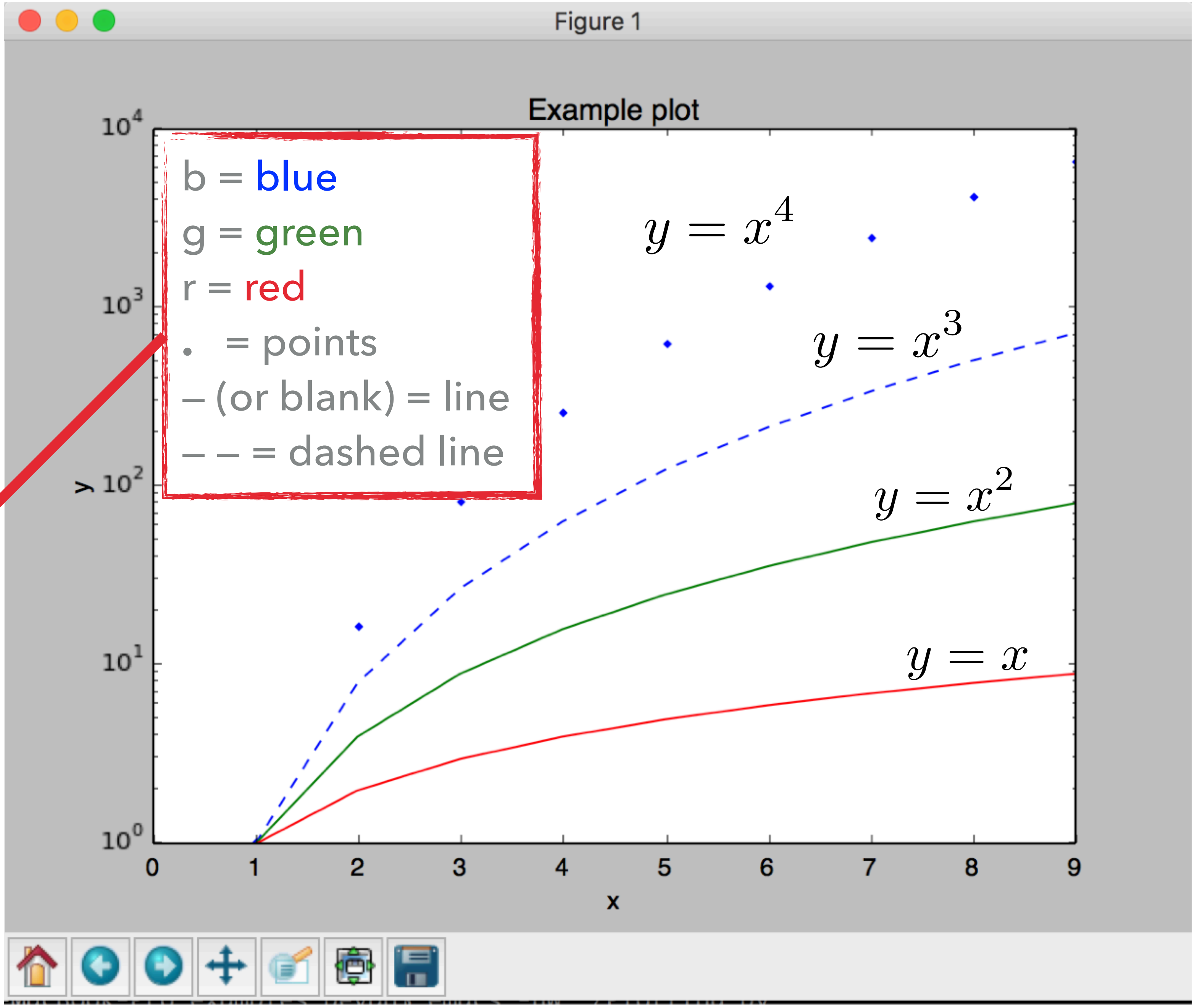
```
plt.plot(x, y4, 'b.')
plt.plot(x, y3, 'b--')
plt.plot(x, y2, 'g-')
plt.plot(x, y,  'r')
plt.ylabel('y')
plt.xlabel('x')
plt.yscale('log')
plt.title('Example plot')
```



MATPLOTLIB: PLOTTING ▸ `Plotting.py`

- Sometimes it is better to overlay the plots.
- These functions are better represented on a log scale vertically.

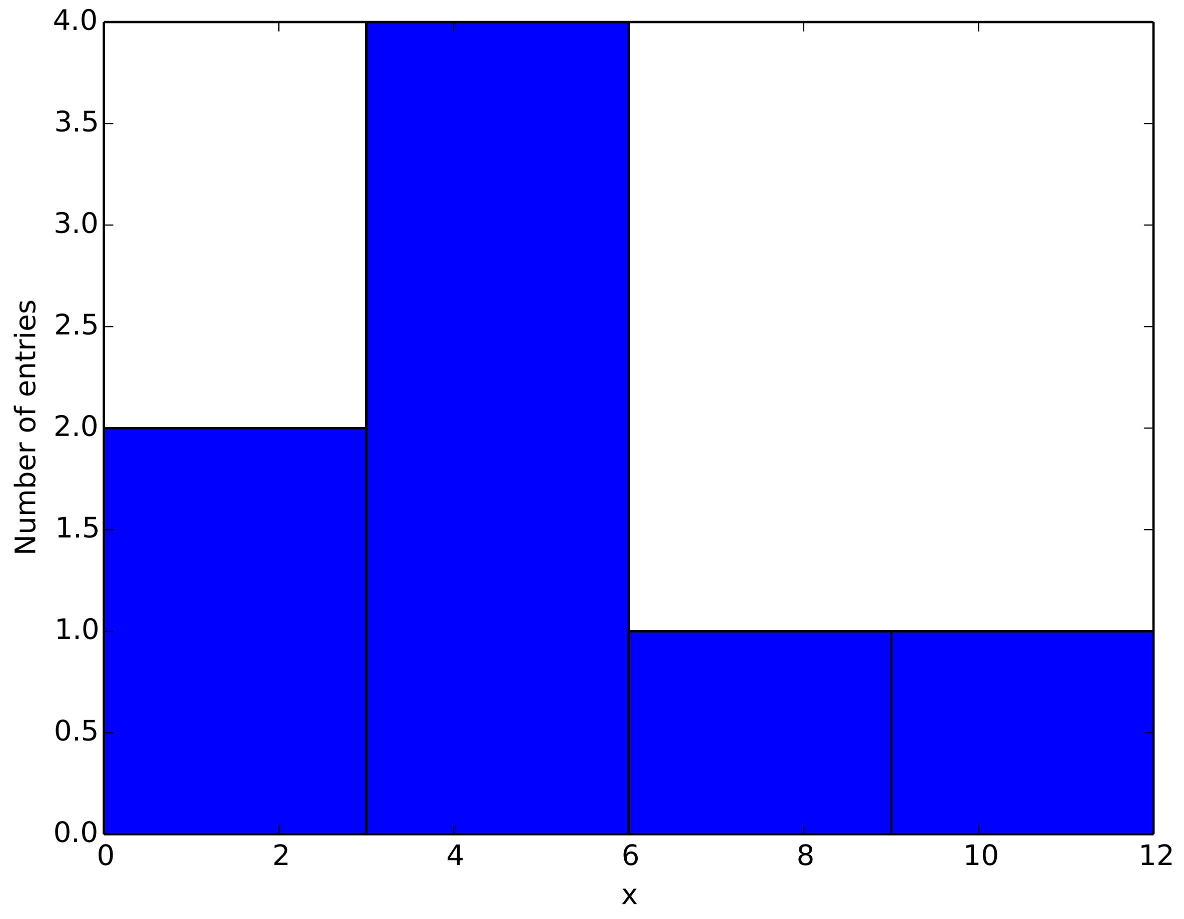
```
plt.plot(x, y4, 'b.')
plt.plot(x, y3, 'b--')
plt.plot(x, y2, 'g-')
plt.plot(x, y,  'r')
plt.ylabel('y')
plt.xlabel('x')
plt.yscale('log')
plt.title('Example plot')
```



MATPLOTLIB: PLOTTING ‣ `Plotting.py`

- Histograms are also useful visual representations of data.

```
data = [1, 2, 3, 4, 4.5, 4.7, 8, 10]

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
n, bins, patches = ax.hist(data, 5, (0, 15))
ax.set_xlabel("x")
ax.set_ylabel("Number of entries")
fig.savefig("Plotting.pdf")
```



[1] See https://matplotlib.org/users/pyplot_tutorial.html for more details.

NUMPY

- `NumPy.py`

- NumPy is a useful Python module for scientific computing. It allows us to interface with TensorFlow data types.
- The main object type us the NumPy array, which we will see later allows us to work with tensor objects in TensorFlow.
- To use this we need to import the corresponding module:

```
import numpy as np
```

https://docs.scipy.org/doc/numpy/user/basics.creation.html
https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html

NUMPY

- `NumPy.py`

- **Reminder:**
  - Single numbers are scalar objects (e.g. `x=3`, here `x` is a scalar)
  - Arrays are collection of numbers, we have used Python lists for arrays
    - e.g. `[1, 2, 3, 4, …, N]` until now; where this example has N dimensions (i.e. N elements).
  - Mathematically we can think of arrays as having a correspondence to a vector where each entry of the array corresponds to the value of one of the dimensions that the vector is used to describe.
    - e.g. the co-ordinate pair x and y can be written as a vector (x, y) and represented as a python array `[x, y]`.
- We can also write the elements of an N dimensional array as an array themselves (of dimension M). This allows us to represent an N x M dimensional object (matrix).
- Higher dimensional objects are tensors (e.g. a rank 3 tensor is an N x M x P object).

https://docs.scipy.org/doc/numpy/user/basics.creation.html
https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html

NUMPY

- `NumPy.py`

- Example: 2x2 matrix represented in a python list:
  - $[[\underbrace{\texttt{A, B}}_{\text{row 1}}], [\underbrace{\texttt{C, D}}_{\text{row 2}}]]$
- Example: 2x2 matrix represented in a numpy array:
  - `x = np.array([[A, B], [C, D]])`

- Example: a set of n data, each example being represented by two coordinate values (or features):
  - $[[x_1, y_1], [x_2, y_2], \ldots, [x_n, y_n]]$

https://docs.scipy.org/doc/numpy/user/basics.creation.html
https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html

NUMPY

- `NumPy.py`

- Create a unit matrix: use the identity function.
- e.g. a 5x5 unit matrix:

```
unit = np.identity(5)
print(unit)
```

$\longrightarrow$

```
[[ 1.   0.   0.   0.   0.]
 [ 0.   1.   0.   0.   0.]
 [ 0.   0.   1.   0.   0.]
 [ 0.   0.   0.   1.   0.]
 [ 0.   0.   0.   0.   1.]]
```

$$\parallel$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

- The eye function can also be used: `np.eye(5)` is equivalent to `np.identity(5)`

https://docs.scipy.org/doc/numpy/user/basics.creation.html
https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html

NUMPY

- `NumPy.py`

- Create a matrix with elements assigned to zero.
- e.g. a 2x3 matrix of zeros:
  - i.e. 2 rows and 3 columns

```
Zeros = np.zeros((2,3))
print(Zeros)
```

$\longrightarrow$

```
[[ 0.   0.   0.]
 [ 0.   0.   0.]]
```

$$\| \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

https://docs.scipy.org/doc/numpy/user/basics.creation.html
https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html

NUMPY

- `NumPy.py`

- Create a tensor with elements assigned to zero.
- e.g. a 2x3x4 array of zeros:

```
ZerosTensor = np.zeros((2,3,4))
print(ZerosTensor)
```

$\rightarrow$

```
[[[ 0.   0.   0.   0.]
  [ 0.   0.   0.   0.]
  [ 0.   0.   0.   0.]]

 [[ 0.   0.   0.   0.]
  [ 0.   0.   0.   0.]
  [ 0.   0.   0.   0.]]]
```

} row 1

} row 2

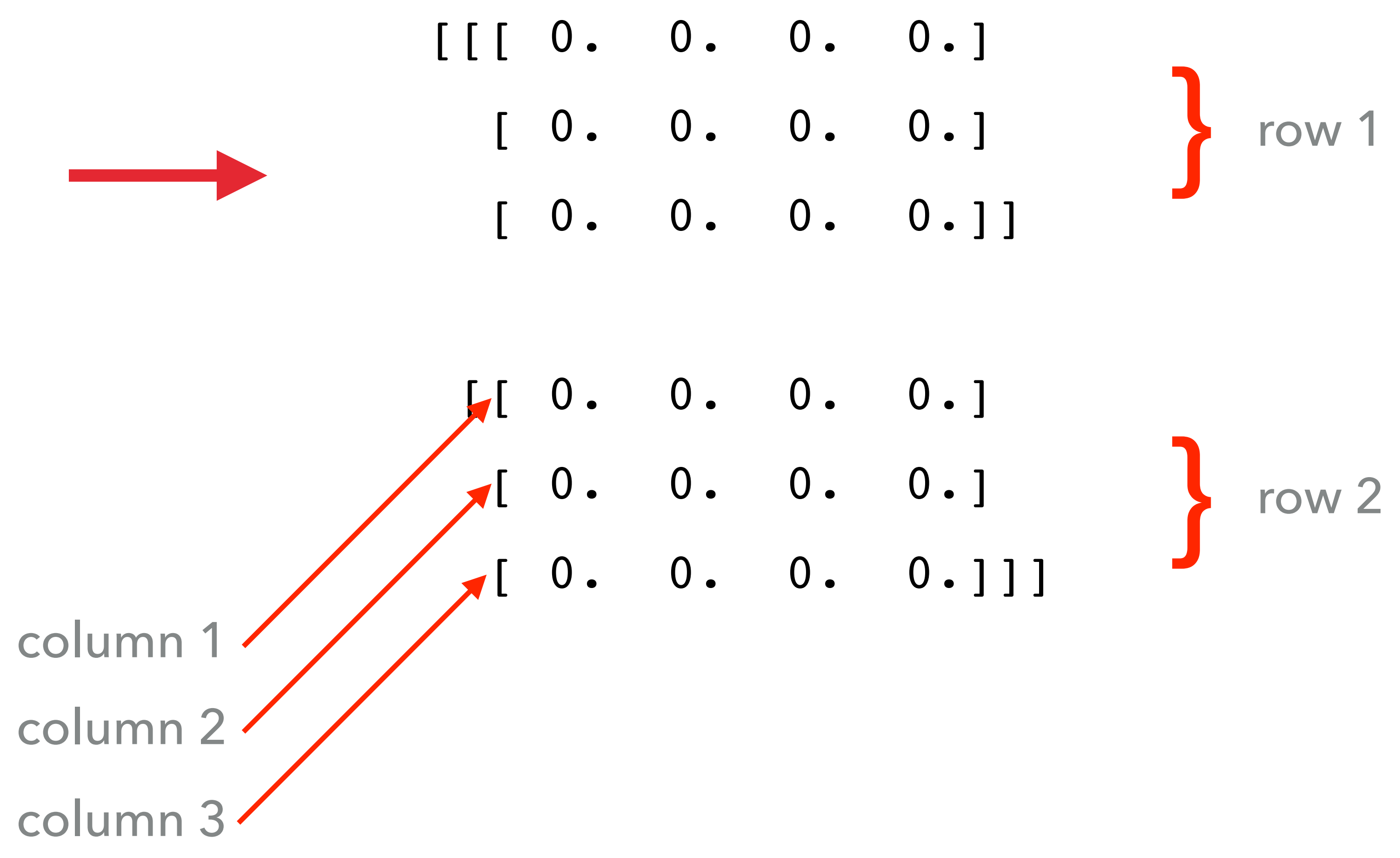
https://docs.scipy.org/doc/numpy/user/basics.creation.html
https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html

NUMPY

- `NumPy.py`

- Create a tensor with elements assigned to zero.
- e.g. a 2x3x4 array of zeros:

```
ZerosTensor = np.zeros((2,3,4))
print(ZerosTensor)
```



```
[[[ 0.   0.   0.   0.]
  [ 0.   0.   0.   0.]
  [ 0.   0.   0.   0.]]

 [[ 0.   0.   0.   0.]
  [ 0.   0.   0.   0.]
  [ 0.   0.   0.   0.]]]
```

https://docs.scipy.org/doc/numpy/user/basics.creation.html
https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html

NUMPY

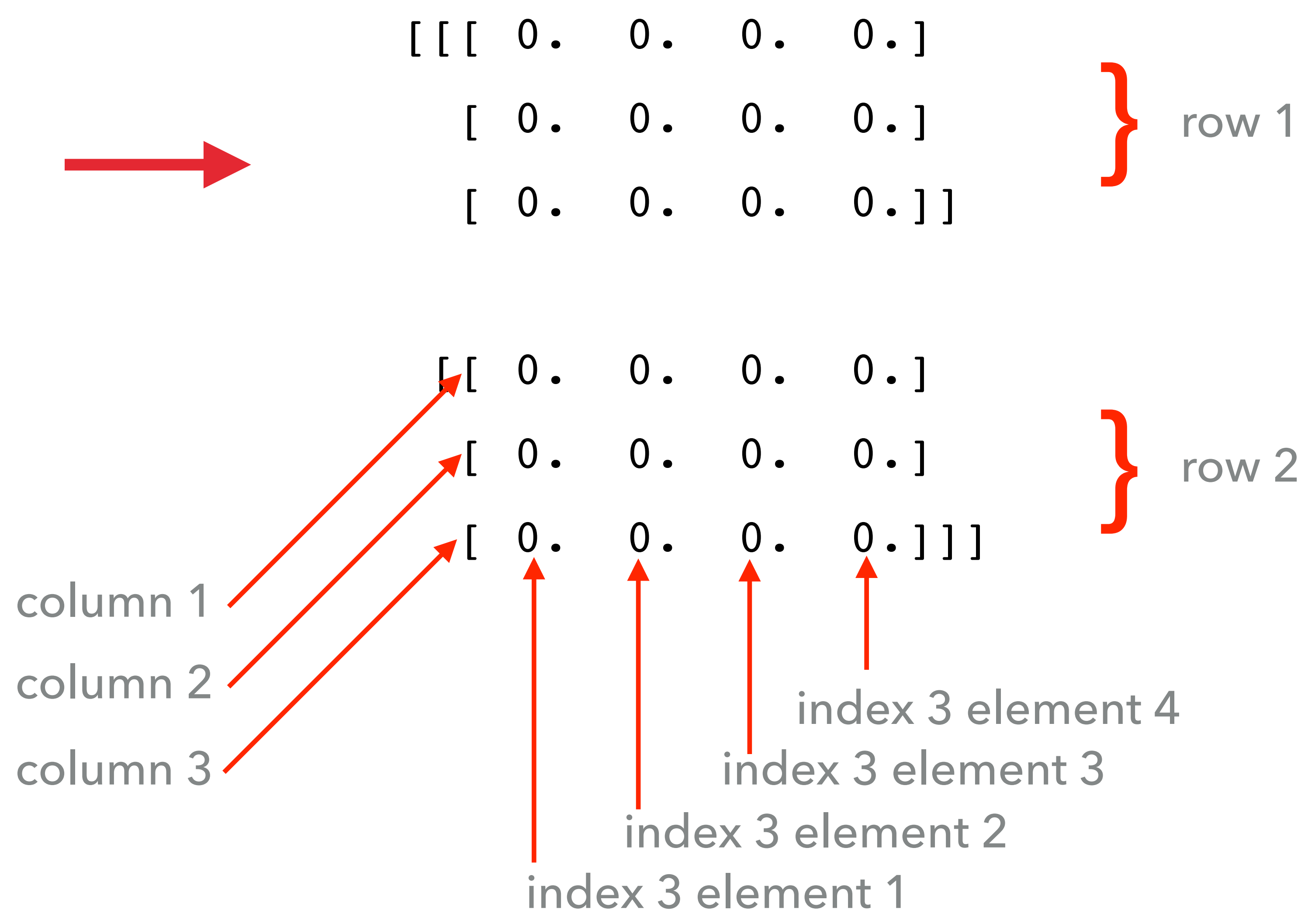
- NumPy.py

- Create a tensor with elements assigned to zero.
- e.g. a 2x3x4 array of zeros:

```
ZerosTensor = np.zeros((2,3,4))
print(ZerosTensor)
```

2D arrays usually are referred to has having rows (index 1) and columns (index 2).

The rank of a Tensors indicates the number of indices.



```
[[[ 0.   0.   0.   0.]
  [ 0.   0.   0.   0.]
  [ 0.   0.   0.   0.]]

 [[ 0.   0.   0.   0.]
  [ 0.   0.   0.   0.]
  [ 0.   0.   0.   0.]]]
```

https://docs.scipy.org/doc/numpy/user/basics.creation.html
https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html

NUMPY

- `NumPy.py`

- Create a tensor with randomly initialised elements.
- e.g. a 2x3x4 tensor:

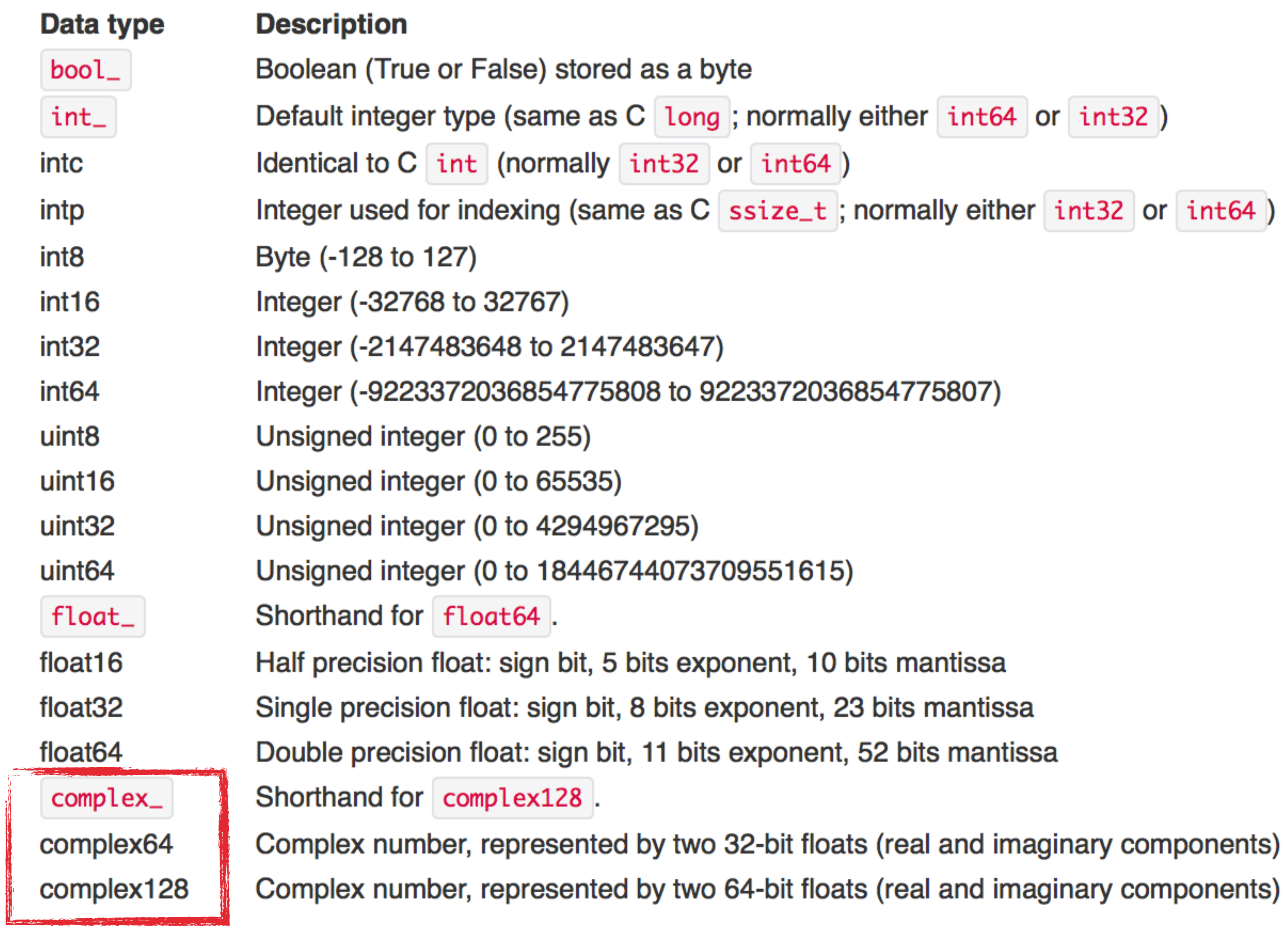
```
Rand = np.random.random((2, 3, 4))
print(Rand)
```

```
[[[ 0.37979962  0.78551892  0.34330952  0.92365559]
  [ 0.00454494  0.72722471  0.38566498  0.98557806]
  [ 0.33205225  0.03776655  0.68628128  0.49059066]]

 [[ 0.97233543  0.88084555  0.95404336  0.7690701 ]
  [ 0.3738863   0.76247708  0.02096017  0.88523764]
  [ 0.12331041  0.41162241  0.53865653  0.05036706]]]
```

https://docs.scipy.org/doc/numpy/user/basics.creation.html
https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html

NUMPY

- Data types



Data type	Description
`bool_`	Boolean (True or False) stored as a byte
`int_`	Default integer type (same as C `long`; normally either `int64` or `int32`)
intc	Identical to C `int` (normally `int32` or `int64`)
intp	Integer used for indexing (same as C `ssize_t`; normally either `int32` or `int64`)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
`float_`	Shorthand for `float64`.
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
`complex_`	Shorthand for `complex128`.
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

NumPy is part of SciPy and provides complex data type to facilitate scientific computing.

https://docs.scipy.org/doc/numpy/user/basics.types.html

NUMPY

- Elementwise operations can be performed on NumPy arrays; this can simplify calculations, and we will see parallels with TensorFlow ops.
- Operators +, -, / and * are element wise operators.

```
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6], [7, 8]])
z=x+y
print(z)
```

$\longrightarrow$

```
[[ 6  8]
 [10 12]]
```

- Similarly built in functions have been replicated in NumPy so that we can perform element-wise calculations.

NUMPY

- Elementwise operations can be performed on NumPy arrays; this can simplify calculations, and we will see parallels with TensorFlow ops.
- Matrix multiplication can easily be performed using NumPy's matmul function:

```
x = np.array([[1, 2], [3, 4]])
y = np.array([[0.5, 1], [0, 2]])
product = np.matmul(x,y)
print(product)
```

$\longrightarrow$

```
[[ 0.5   5. ]
 [ 1.5  11. ]]
```

SUMMARY

- We have worked through some elementary python to look at simple calculations, conditionals, loops, classes, and functions.
- We have also used matplotlib to make (and save) plots.
- We have use NumPy to help us perform calculations.
- The skills you pick up while working with these language constructs will underpin work with TensorFlow in the coming weeks.

SUGGESTED READING

- There are many good books on Python that you may wish to review. A general remark - use library resources to try out books to find one that has a style that you can work with. People are all different, and respond differently to different learning resources.
- In addition to the resources listed on page 3, and references given in the slides, you may find the following useful further reading:
  - O'Reilly books on Python
  - web searches (in addition to the URLs provided, when you get an error message it can be useful to just to a quick web search of that error).
  - https://stackoverflow.com may also be a useful resource.