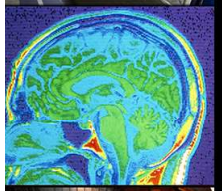
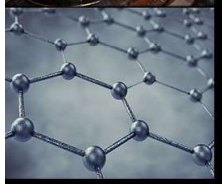





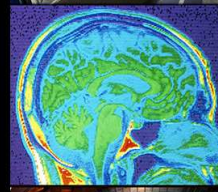
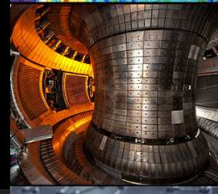


Lecture 1
Introduction to Python

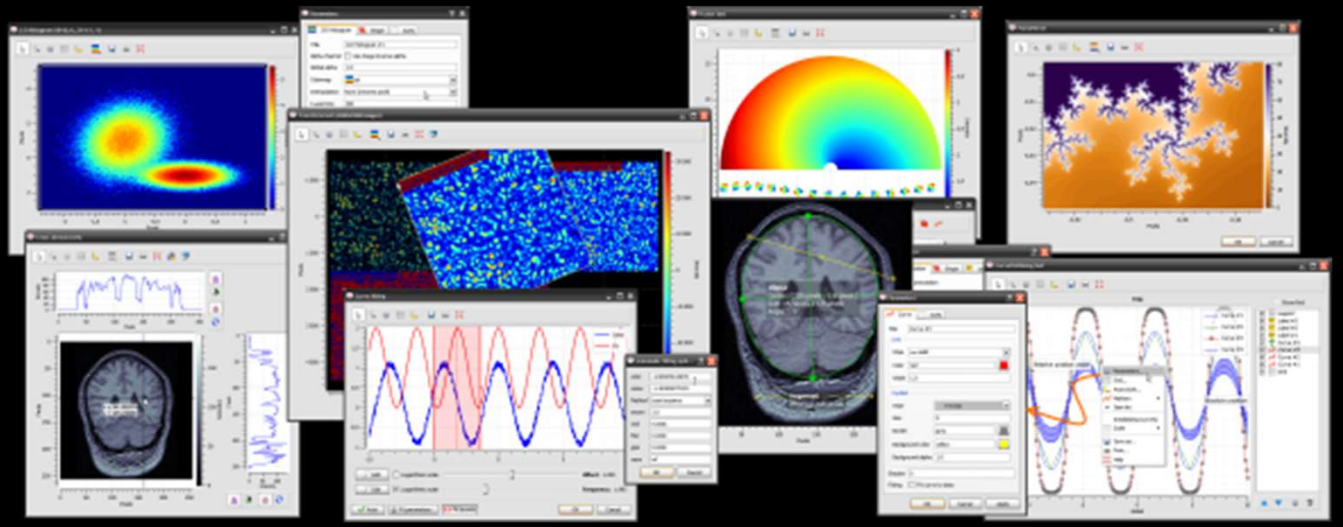
lecturer Alexander Gorbunov
Email: agorbunov@hse.ru

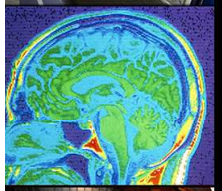


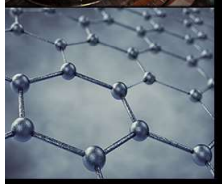


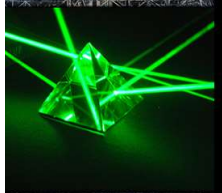





1.1 Introduction



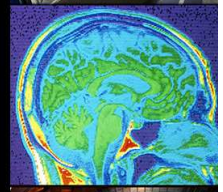
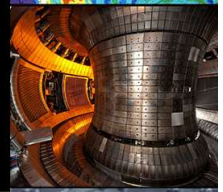






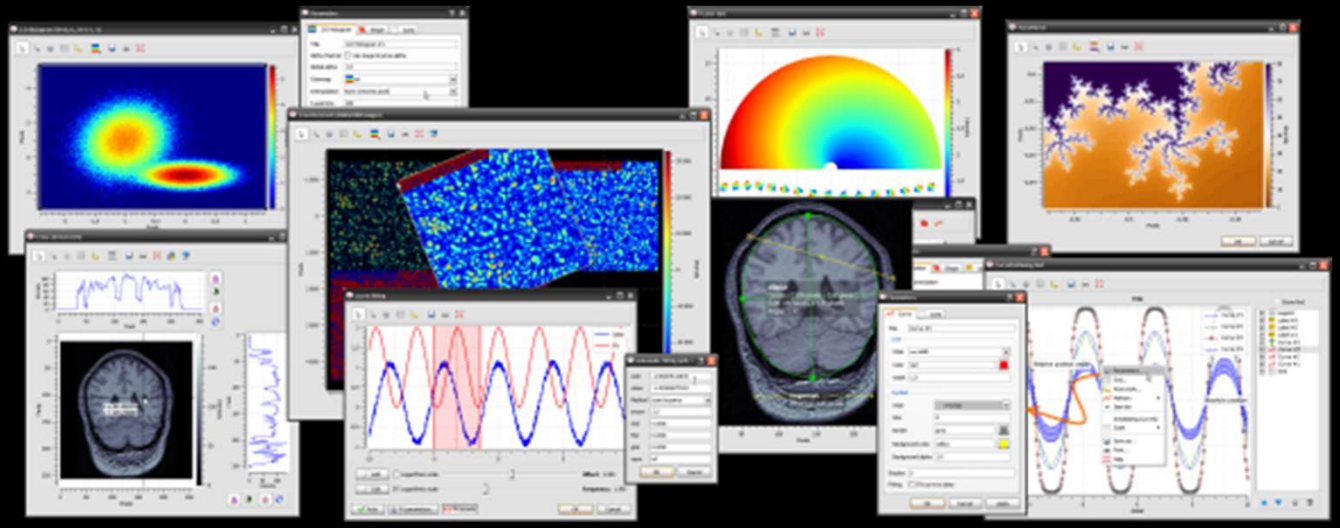


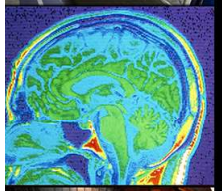


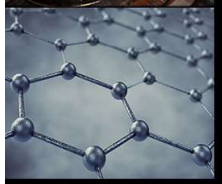




Python

Python is a very powerful programming language. I, like many other scientists, use Python every day to process data, analyse results and create visualisations, like the ones below:







The Course

Module 1

Introduction

What is Python

Basic Objects lists, tuples, dictionaries

Strings

Functions

Reading and Writing Files

Module 2

Introduction to Numpy

Numpy Functions

Introduction to pandas

Pandas Functions

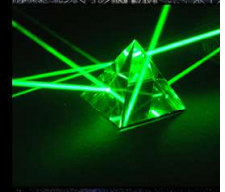
Introduction to Matplotlib

Matplotlib Plots

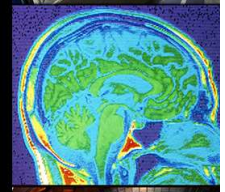
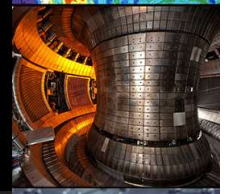
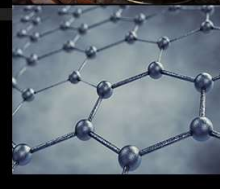
Module 3

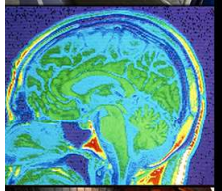
Business intelligence

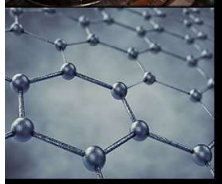
Introduction to Tableau, Qlik Sense, Power BI



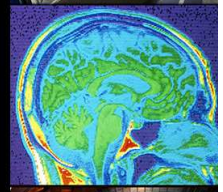
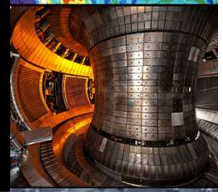










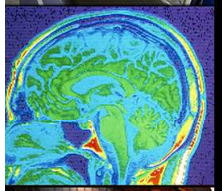

Exercises

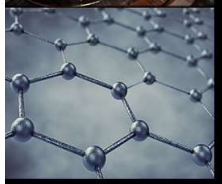
The only real way to learn Python is to try it yourself.

The more you practise the better you will be.

So the emphasis is on **you** to try the exercises.

You are encouraged to work together and help each other.





Google is your friend

Pretty much every problem you have, someone else will have had that problem

Use google…

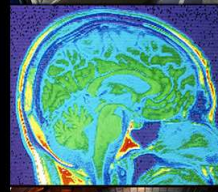
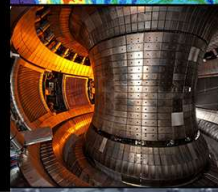
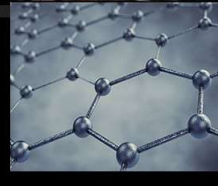
BUT

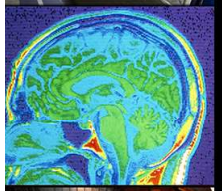
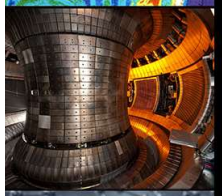
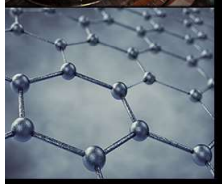
Do not copy and paste, you must understand every line in your code before using it

Blindly copying and pasting is an easy way to get many problems!!





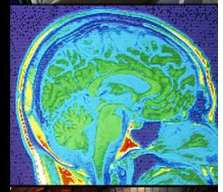
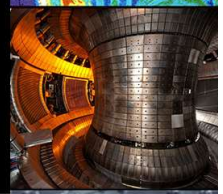







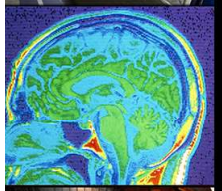
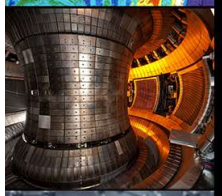
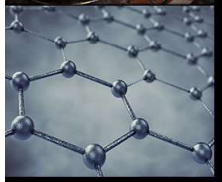


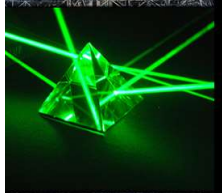




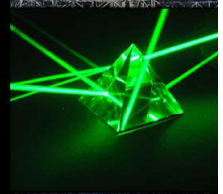




1.2 What is Python?

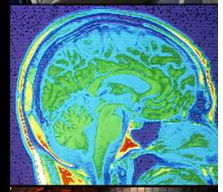















First, what is 'Programming'?

Being good at programming is not simply about knowing a programming language, it is about:

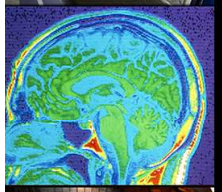
- understanding a problem conceptually and being able to translate it into a computer program
- knowing how to fix your program when it does not work
- writing a program that is fast enough (**not** the fastest possible)
- writing a program that can be understood by other people (or by yourself in a year or two)

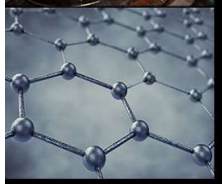
In science, the last point is important:

- reproducible research
- open science

Likewise in industry many companies demand that you write code that can be read and understood by your colleagues.

Graduates who can combine math and science skills with programming are in very high demand on the job market.



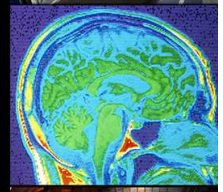
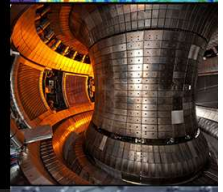


Why choose Python over other languages?

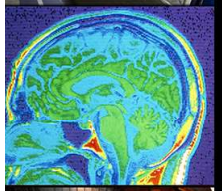
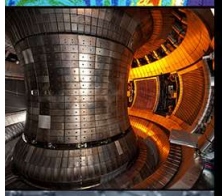
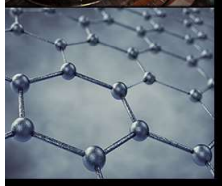
- It's free!
- It's relatively easy to pick up
- It has a clean and simple syntax which is easy to read
- It gives detailed errors by default which makes it easier to fix bugs
- It can be used interactively
- Many employers want python programmers
- Huge standard library
- It can interact with existing C/Fortran code when speed is important
- Nearly every problem you have will have been done before (Google is your friend)
- Generally good quality documentation
- Platform agnostic (works on Windows, Mac, Linux etc)
- Very common use of BSD/MIT-style licensing with third-party modules
- Emphasis on code readability (see PEP-8)
- Useful for a really broad range of programming tasks from little shell scripts to enterprise web applications to scientific uses; it may not be as good at any of those as a purpose-built programming language but it can do all of them, and do them well (e.g. you don't see web apps written as bash scripts nor do you see Linux package managers written in Java)







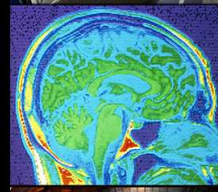
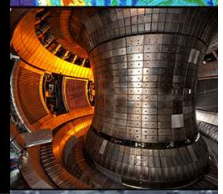




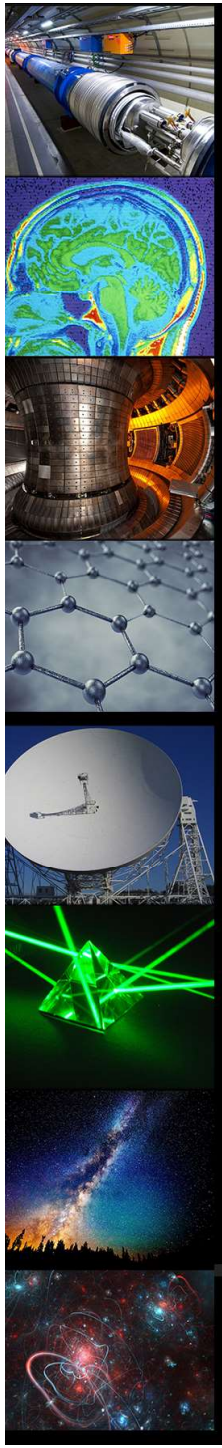
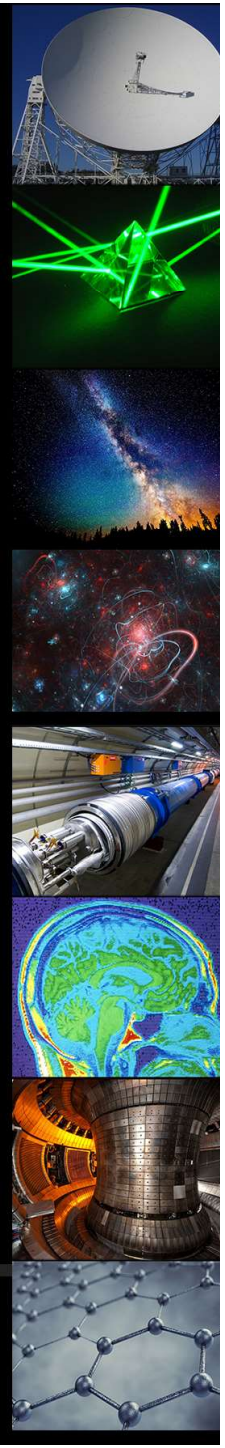






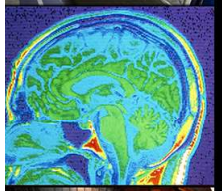


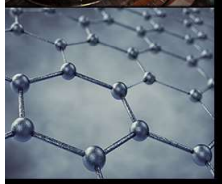



A note on Python Versions

We will be using Python version 3.7.x

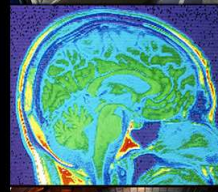
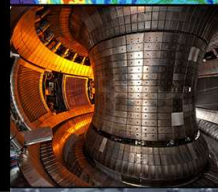










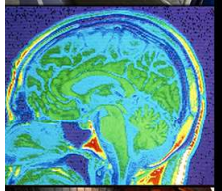

Installing Anaconda 1

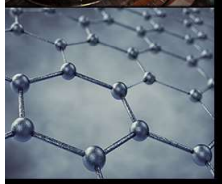
There are many ways of using python.

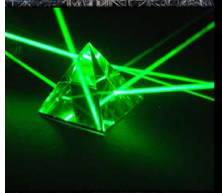
You are free to use any version distribution of python.

However, you may need to install modules that do not come with your chosen distribution.

Anaconda comes will all modules that we will be using and in easy to install and use.





Installing Anaconda 2a

To install:

1) Go to https://www.anaconda.com/distribution/#download-section
2) 2) Choose your installer (Windows, Mac or Linux should detect your version automatically)

For Windows 64 bit:

Windows 64-Bit Python 3.7 Graphical Installer - Size: 366M

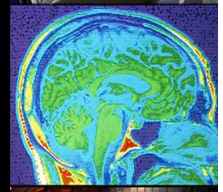
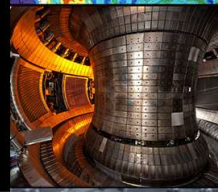
For Windows 32 bit:

Windows 32-bit - Python 3.7 - Graphical Installer

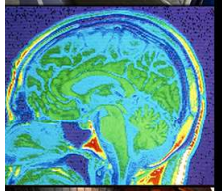
After downloading the installer, double click the .exe file and follow the instructions on the screen.

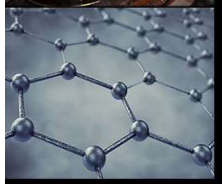


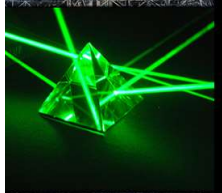






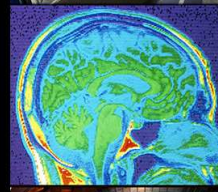
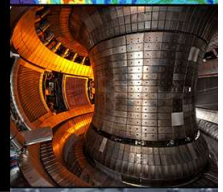











Installing Anaconda 2b

To install:

1) Go to https://www.anaconda.com/distribution/#download-section

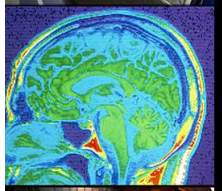
2) Choose your installer (Windows, Mac or Linux should detect your version automatically)

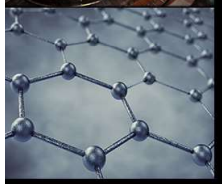
For Mac:

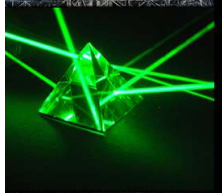
Mac OS X - 64-Bit Python 3.7 Graphical Installer - Size: 275M (OSX 10.7 or higher)

After downloading the installer, double click the .pkg file and follow the instructions on the screen.

COMMAND-LINE INSTALLS:
After downloading the installer, in the shell execute: bash Anaconda-2.1.0-MacOSX-x86_64.sh Note that you should type "bash" regardless of whether or not you are actually using the bash shell.





Installing Anaconda 2c

To install:

1) Go to https://www.anaconda.com/distribution/#download-section

2) Choose your installer (Windows, Mac or Linux should detect your version automatically)

For Linux 64 bit:

Linux 64-Bit - Python 2.7

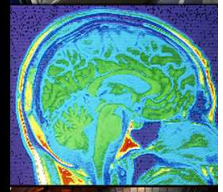
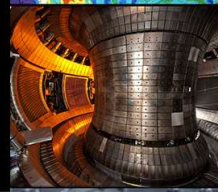
For Linux 32 bit:

Linx 32-bit

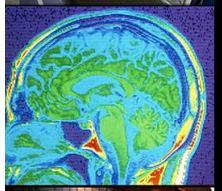
After downloading the installer, in the shell execute: bash Anaconda-2.1.0-Linux-x86.sh Note that you should type "bash" regardless of whether or not you are actually using the bash shell.

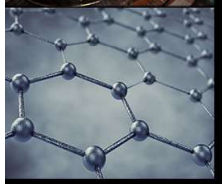








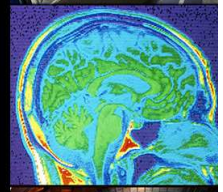
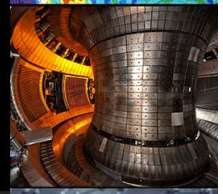






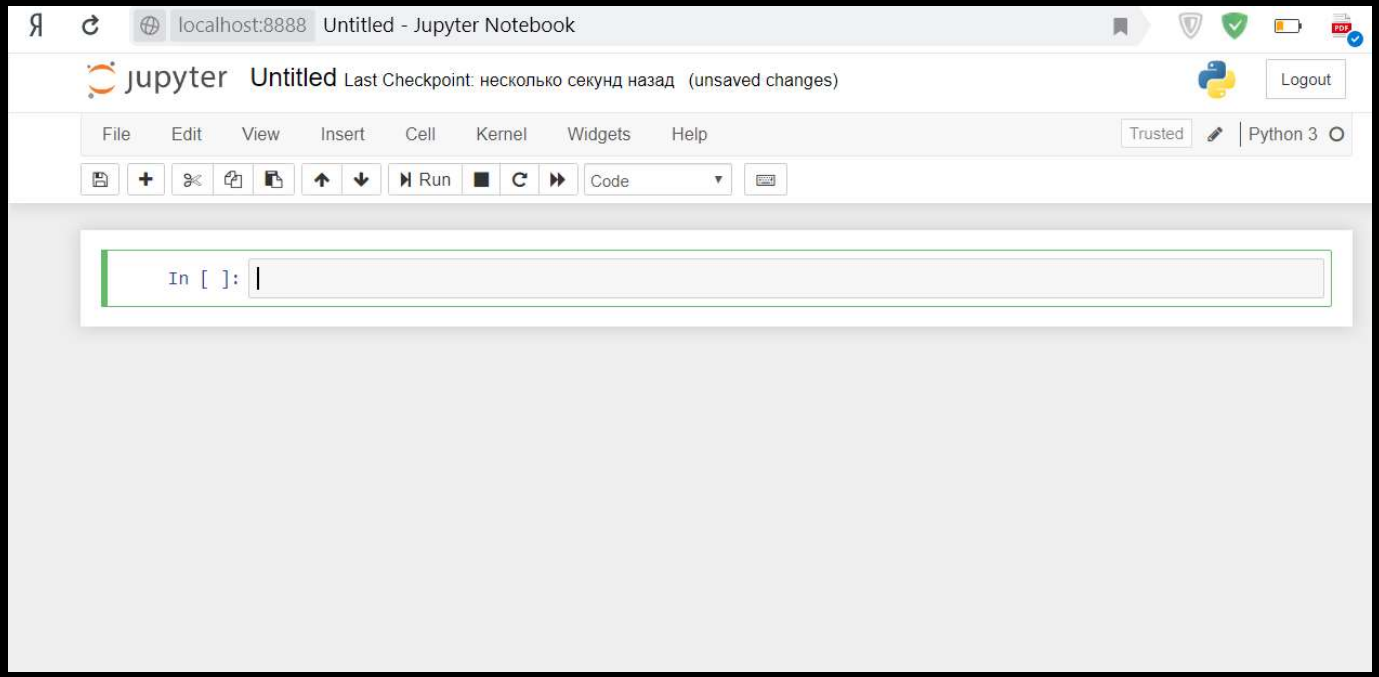





Using Anaconda 2

The interface should look something like as follows:

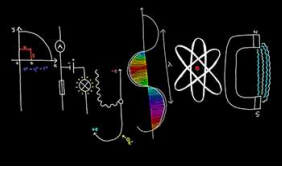


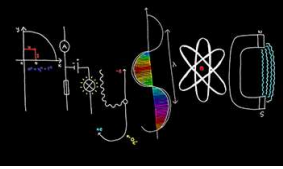
First line of code

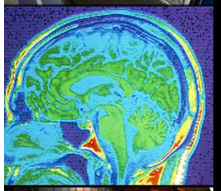
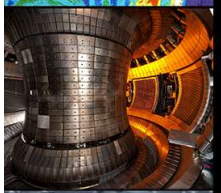
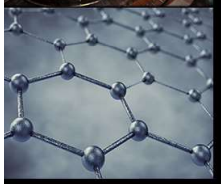
In the console type:

```
In [2]: print ("Hello World!")

Hello World!


You are now a programmer!
```

The print function will allow us to display information from our code.





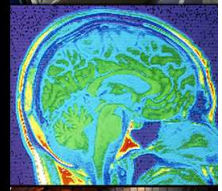
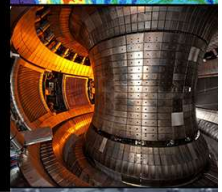









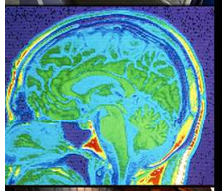


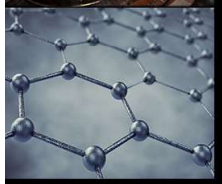


1.3 Basic Objects 1:

Numbers, Strings and Lists

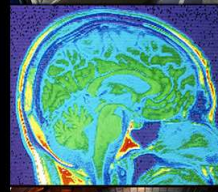
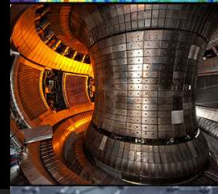






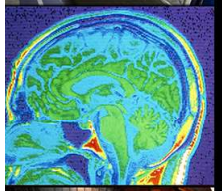


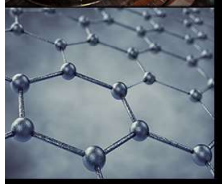


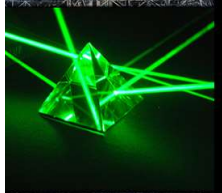




Basic Numeric Types

The basic numeric data types are similar to those found in other programming languages.

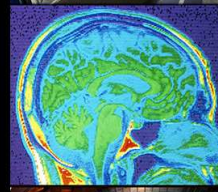
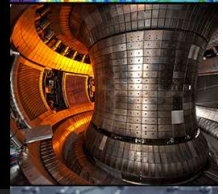








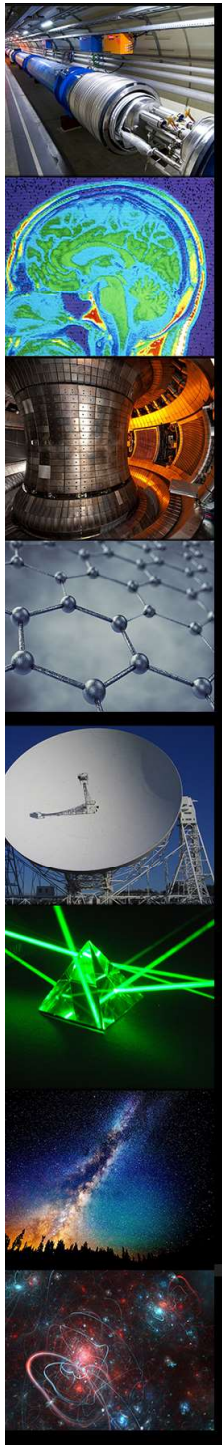
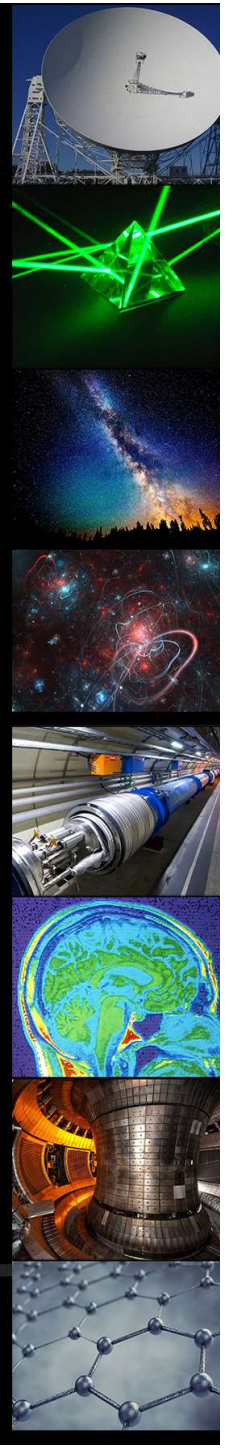



Integers (int):

Real whole numbers (can be positive or negative)

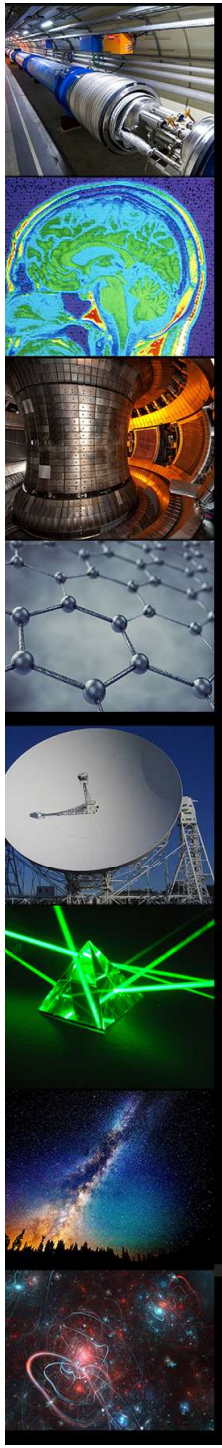
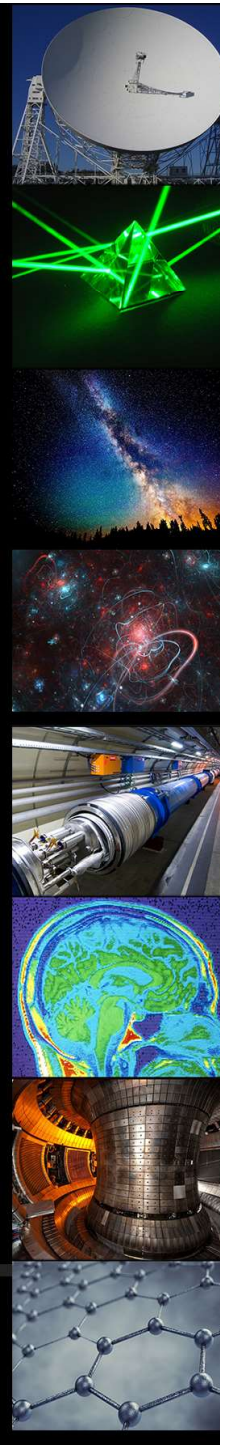
```
In [1]: i = 1
        j = 219089
        k = -21231

In [3]: print(i, j, k)

        1 219089 -21231
```

Floating point values (float)

Real rational numbers (i.e. can include decimal numbers)

```
In [5]: a = 4.3
        b = -5.2111222
        c = 3.1e33

In [6]: print(a, b, c)

        4.3 -5.2111222 3.1e+33
```

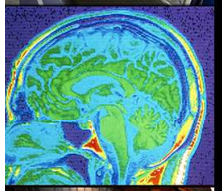
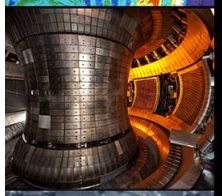
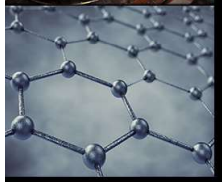



Complex values (complex):

Real part and imaginary part

i.e. 1.0 + 2.0i

```
In [7]: d = complex(4.0, -1.0)

In [11]: print( d )

         (4-1j)
```

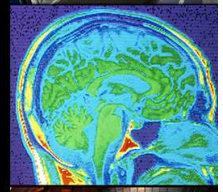
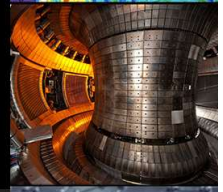













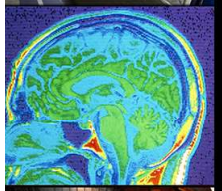

Manipulation 1

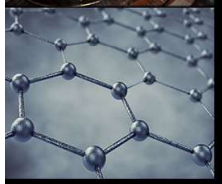
Manipulating these objects behaves the way you would expect, so an operation (+, -, /, *, **, etc.) on two values of the same type produces another value of the same type, while an operation on two values with different types produces a value of the more 'advanced' type:

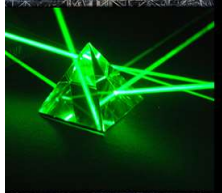
i.e.

- adding two integers gives an integer:

```
In [9]: 1 + 3

Out[9]: 4
```





Manipulation 2

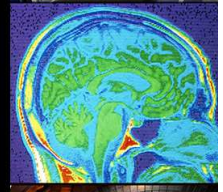
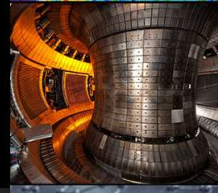
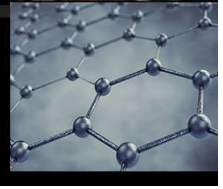
Manipulating these objects behaves the way you would expect, so an operation (+, -, /, *, **, etc.) on two values of the same type produces another value of the same type, while an operation on two values with different types produces a value of the more 'advanced' type:

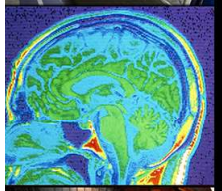
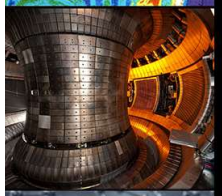
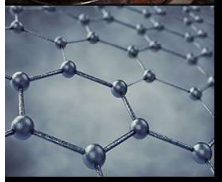
- Multiplying two floats gives a float:

```
In [10]: 3.0 * 2.0

Out[10]: 6.0
```







Manipulation 3

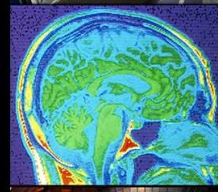
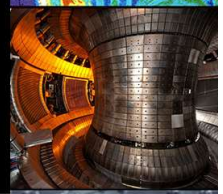
Manipulating these objects behaves the way you would expect, so an operation (+, -, /, *, **, etc.) on two values of the same type produces another value of the same type, while an operation on two values with different types produces a value of the more 'advanced' type:

Subtracting two complex numbers gives a complex number:

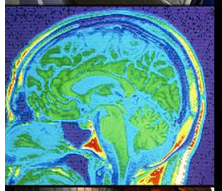
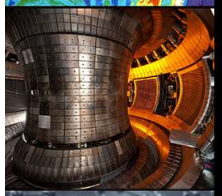
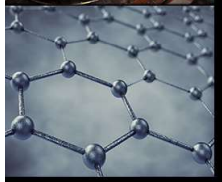
```
In [12]: complex(2.0, 4.0) - complex(1.0, 6.0)
Out[12]: (1-2j)
```







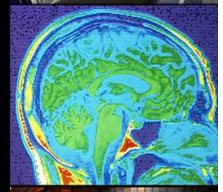
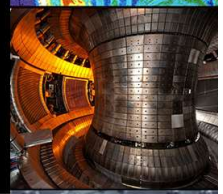












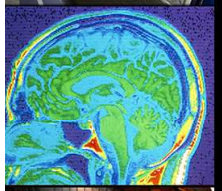



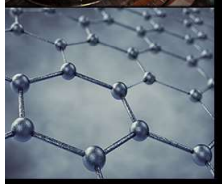

Manipulation 4

- Multiplying an integer with a float gives a float:

```
In [13]: 4 * 9.2

Out[13]: 36.8

In [14]: 4 * 10.0

Out[14]: 40.0

In [16]: 3 * 10.2

Out[16]: 30.599999999999998
```

Note the rounding precision in this last one!

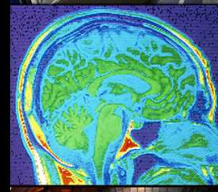
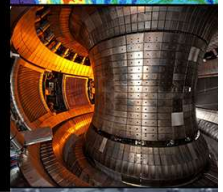











Manipulation 5

- Multiplying a float with a complex number gives a complex number:

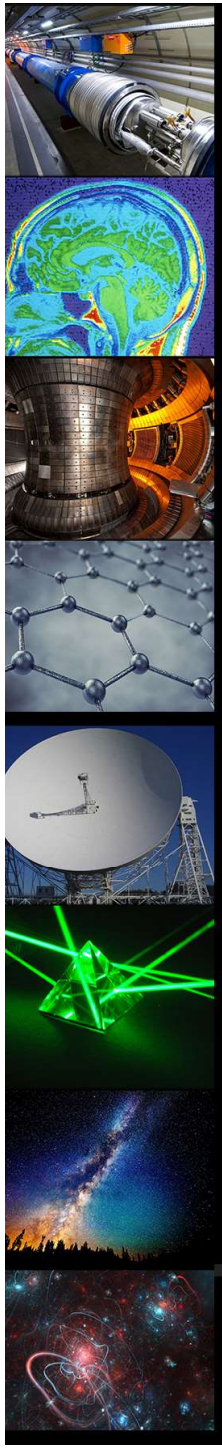
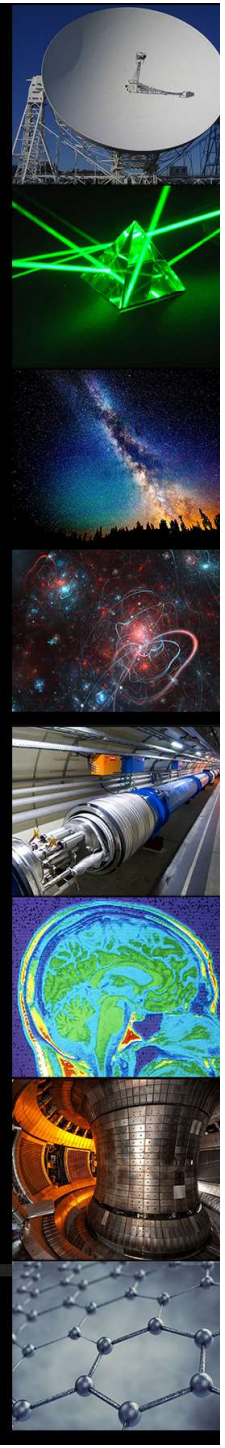
```
In [17]: 3.0 * complex(-1.0, 4.0)
```

```
Out[17]: (-3+12j)
```

- Multiplying an integer and a complex number gives a complex number:

```
In [18]: 8 * complex(-3.3, 1)
```

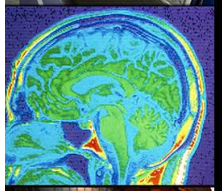
```
Out[18]: (-26.4+8j)
```

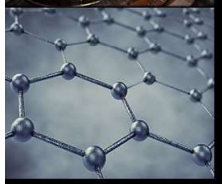



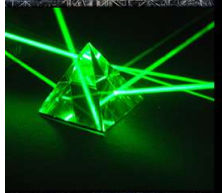
Manipulation 6

However be awake that **integer division** is not the same as **float division**

```
In [19]: 3/2

Out[19]: 1

In [24]: 3.0/2.0

Out[24]: 1.5
```

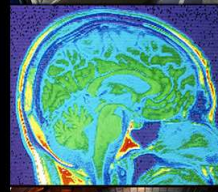
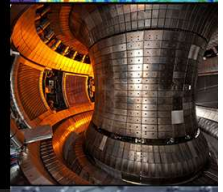











Manipulation 7

A way to prevent this is to **cast** (convert) at least one of the integers in the division to a float:

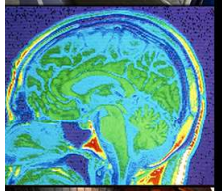
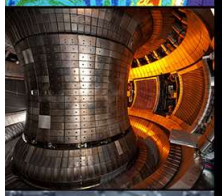
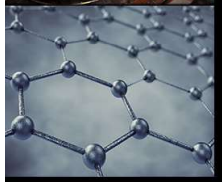
```
In [22]: 3 / float(2)
Out[22]: 1.5
```

or

```
In [23]: float(3) / 2
Out[23]: 1.5
```

or

```
In [26]: 3 / 2.0
Out[26]: 1.5
```

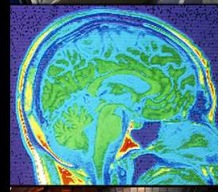
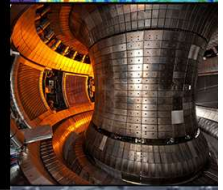











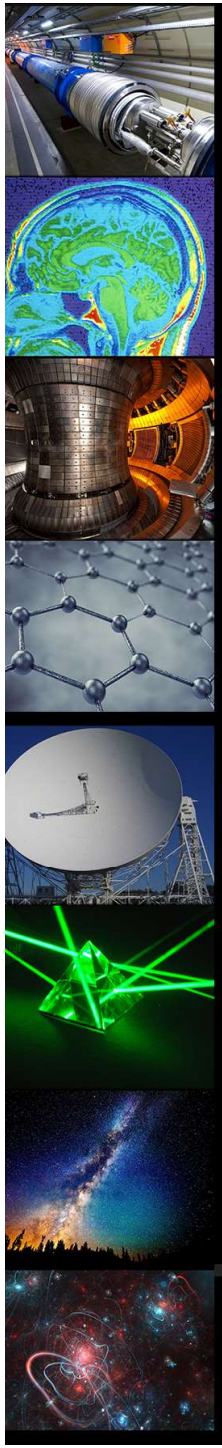
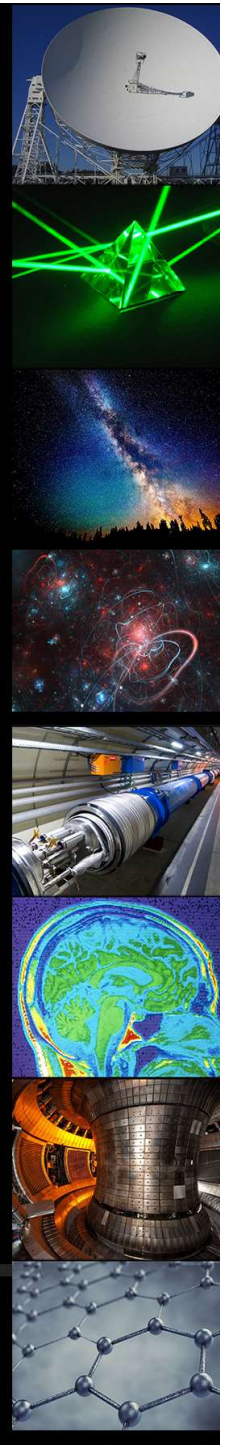



Exercise 1.3a

Divide 10 by 7 and take the square of the result

(Hint: the operator for taking a power is **)

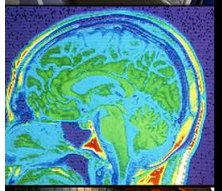
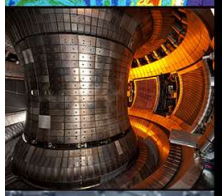
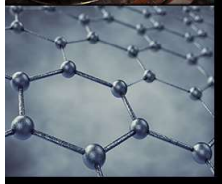



Solution 1.3a

Divide 10 by 7 and take the square of the result

(Hint: the operator for taking a power is **)

```
In [27]: (10/7.0)**2

Out[27]: 2.0408163265306123
```

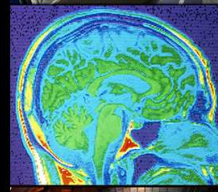
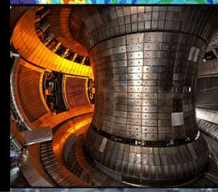













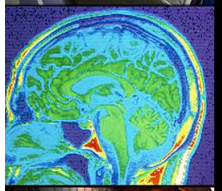
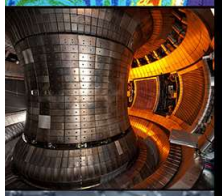
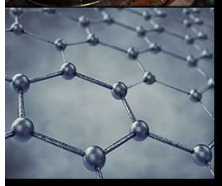




Strings 1

Strings (str) are sequences of characters:

```
In [28]: s = "Hello my name is Fred"
```

You can use either single quotes ('), double quotes ("), or triple quotes ''' or """) to enclose a string (triple quotes for multi-line strings).

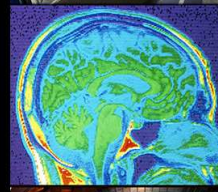
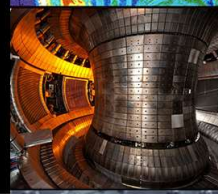











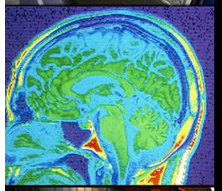


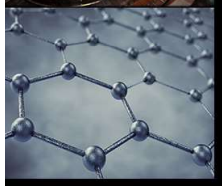




Strings 2

To include single or double quotes inside a string, you can either use the opposite quote to enclose the string:

```
In [29]: "I'm Fred"

Out[29]: "I'm Fred"

In [30]: '"Hello Fred"'

Out[30]: '"Hello Fred"'
```





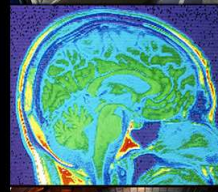
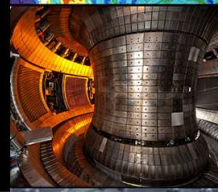
Strings 3

or you can *escape* them:

```
In [31]: 'I\'m Fred'

Out[31]: "I'm Fred"

In [32]: "\"Hello Fred\""

Out[32]: '"Hello Fred"'
```



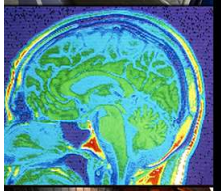
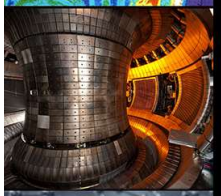
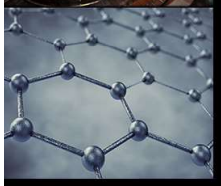



Strings 4

You can also access individual characters or sequences of characters (Note in python *indexing* starts at zero - so the first characster is at position 0)

```
In [34]: s = "Hello my name is Fred"
         s[4]

Out[34]: 'o'

In [36]: s[9:13]

Out[36]: 'name'
```

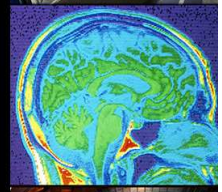
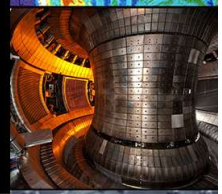













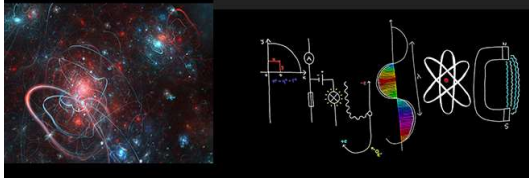



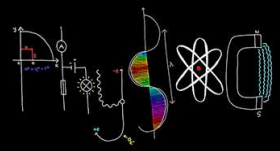

Strings 5

Note that strings are **immutable**, that means you cannot change the value of any characters without creating a new string

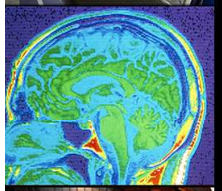
```
In [37]: s[4] = 'r'
```

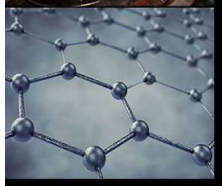
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-37-fd2bb09889bc> in <module>()
----> 1 s[4] = 'r'

TypeError: 'str' object does not support item assignment
```

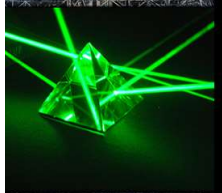








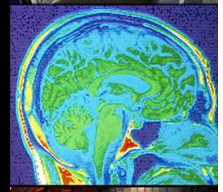
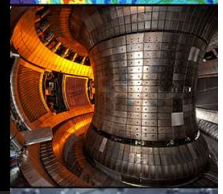










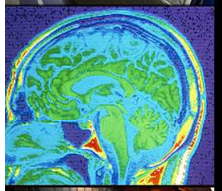
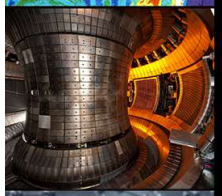
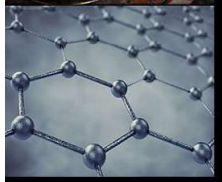

Strings 6

You can use the "+" operator to combine strings:

```
In [38]: "hello" + " " + "world!"

Out[38]: 'hello world!'
```

You can use the "*" operator to repeat strings

```
In [39]: "this is text, " * 3

Out[39]: 'this is text, this is text, this is text, '
```

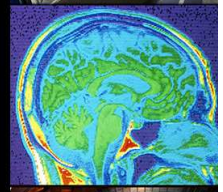
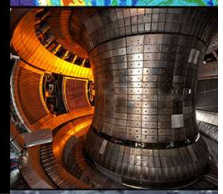













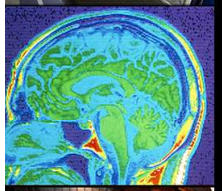
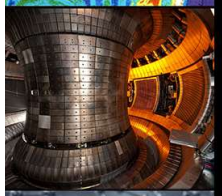
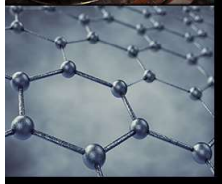




Strings 7

Strings like most **objects** have many **methods** associated with them, here are a few:

- upper (an uppercase version of the string)

```
In [42]: s = 'hello fred'
         s.upper()

Out[42]: 'HELLO FRED'
```

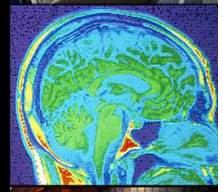













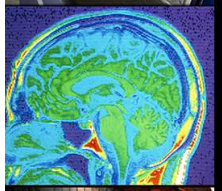
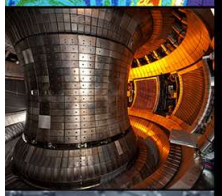
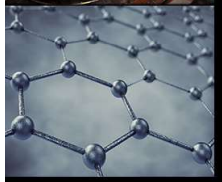


Strings 8

Strings like most **objects** have many **methods** associated with them, here are a few:

- index(substring) (An integer giving the position of the sub-string)

```
In [45]: s = 'hello fred'
         s.index('fred')

Out[45]: 6
```

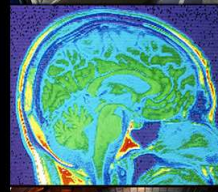
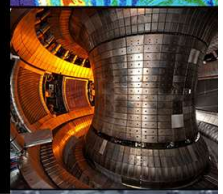














Strings 10

Like most **objects** strings have many **methods** associated with them, here are a few:

- split(character) (A list of substrings splitting at the character defined)

```
In [46]: s = 'this is a test of a test'
         s.split(' ')

Out[46]: ['this', 'is', 'a', 'test', 'of', 'a', 'test']
```

Strings 11

- format(value1, value2) (Formats numbers using {} notation)

  " This is text the value is {x1: y1} +/- {x2: y2}".format(value1, value2)

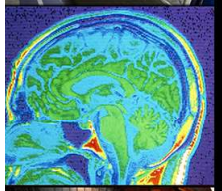
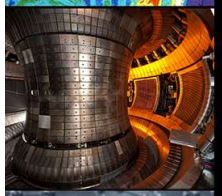
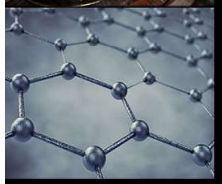
  where x1 is an index of the location of value1 where y1 is the format to print value 1 in

formats are:

```
.1f is for a float with 1 decimal place
.2f is for a float with 2 decimal places
05d is for a integer with 4 leading zeros
```

```
In [52]: "The temperature equals {0:.2f} +/- {1:.2f}".format(23.456789, 0.071232)

Out[52]: 'The temperature equals 23.46 +/- 0.07'

In [63]: "The bank number was {0: 010d}".format(123)

Out[63]: 'The bank number was  000000123'
```

More about strings: https://docs.python.org/2/library/string.html

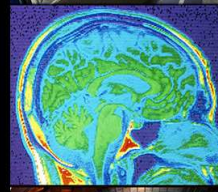
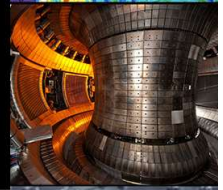









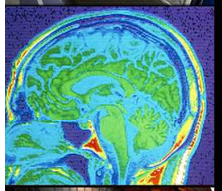


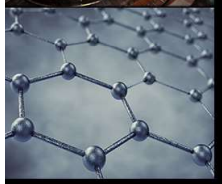



Exercise 1.3b

Store your first name in a variable called *name* and print a hello message to yourself (e.g. "Hello John!")

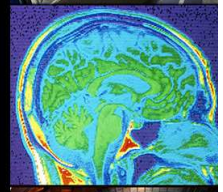
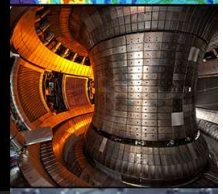








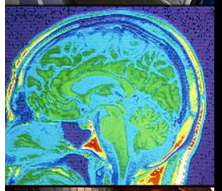


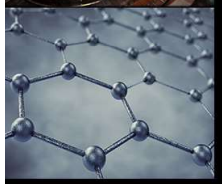



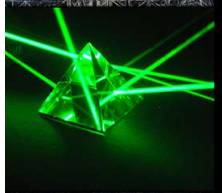

Solution 1.3b

Store your first name in a variable called *name* and print a hello message to yourself (e.g. "Hello John!")

```
In [57]: name = "John"

In [58]: print("Hello " + name + "!")

         Hello John!

In [59]: print("Hello {0}!".format(name))

         Hello John!
```





Lists 1

There are several kinds of ways of storing sequences in Python, the simplest being the *list* which is simply a sequence of *any* Python object.

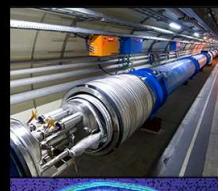
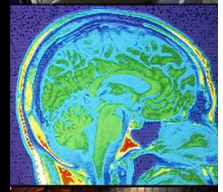
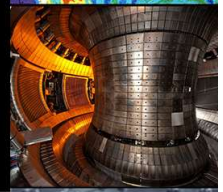
```
In [1]: x = [4, 5.5, "Fred"]
```

Again note that in Python, indexing is *zero-based*, which means that the first element in a list is zero, and not one (like in for example Matlab)

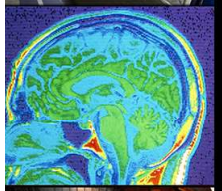
```
In [2]: x[0]
Out[2]: 4

In [3]: x[1]
Out[3]: 5.5

In [4]: x[2]
Out[4]: 'Fred'
```

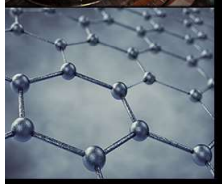


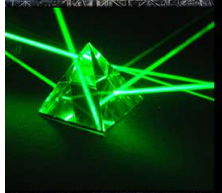






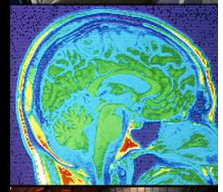
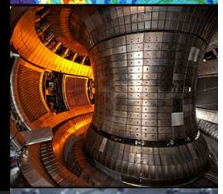










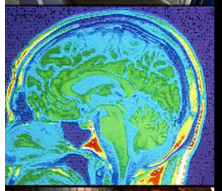

Lists 2

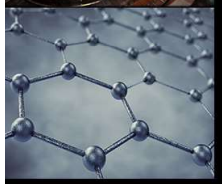
There are several kinds of ways of storing sequences in Python, the simplest being the *list* which is simply a sequence of *any* Python object.

Values in a list **can** be changed, and it is also possible to append or insert elements:

(**append** and **insert** are **methods** associated with a list)

```
In [5]: x[1] = -2.2
In [6]: x
Out[6]: [4, -2.2, 'Fred']
In [7]: x.append(-4000)
In [8]: x
Out[8]: [4, -2.2, 'Fred', -4000]
In [9]: x.insert(1, 3.141592)
In [10]: x
Out[10]: [4, 3.141592, -2.2, 'Fred', -4000]
```

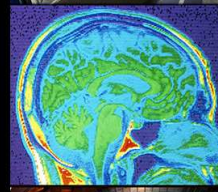
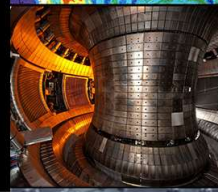










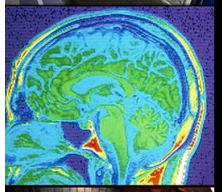
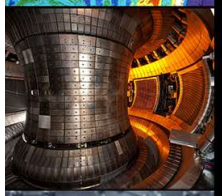
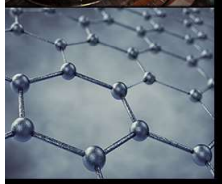




Lists 3

There are several kinds of ways of storing sequences in Python, the simplest being the *list* which is simply a sequence of *any* Python object.

Also very useful is that it is possible to access list items with negative indices, which counts from the end: -1 is the last **element**, -2 is the second from last **element** etc)

```
In [11]: x[-1]

Out[11]: -4000
```

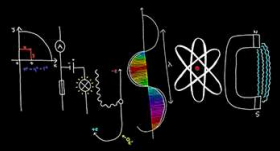
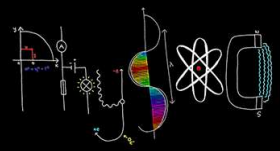


Lists 4

There are several kinds of ways of storing sequences in Python, the simplest being the *list* which is simply a sequence of *any* Python object.

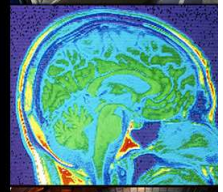
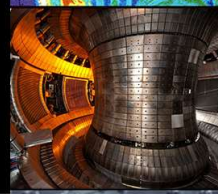
You can also select slices from a list with the start:end:step syntax. Be aware than the last element is **NOT** included

```
In [12]: x[0:2]
Out[12]: [4, 3.141592]

In [13]: x[:2]  # ''start'' defaults to zero
Out[13]: [4, 3.141592]

In [14]: x[2:]  # ''end'' defaults to the last elemen
         t
Out[14]: [-2.2, 'Fred', -4000]

In [16]: x[::2]  # specify a step size
Out[16]: [4, -2.2, -4000]
```

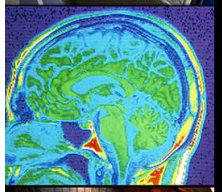



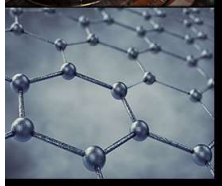








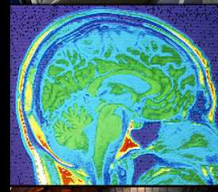
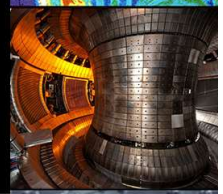










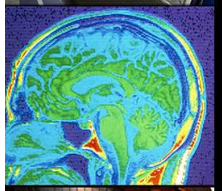
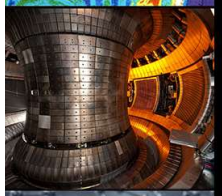
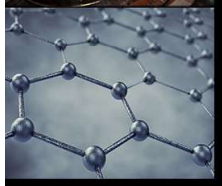




Lists 5

There are several kinds of ways of storing sequences in Python, the simplest being the *list* which is simply a sequence of *any* Python object.

Note if you try and access an element that does not exist, you will get a IndexError

```
In [17]: x = [1, 4, 5]

In [18]: x[0]

Out[18]: 1

In [19]: x[5]

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-19-e8c2945f243d> in <module>()
----> 1 x[5]

IndexError: list index out of range
```

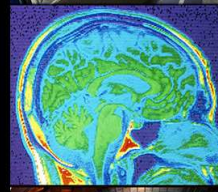
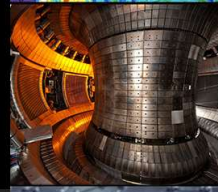













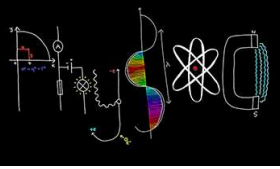
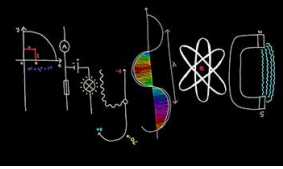




Exercise 1.3c

Store all numbers from 1 to 10 in a list, and print only the even numbers

Solution 1.3c

```
In [1]: mylist = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

In [81]: print mylist[1], mylist[3], mylist[5], mylis
         t[7], mylist[9]

         2 4 6 8 10

In [82]: print(mylist[1::2])

         [2, 4, 6, 8, 10]
```

Functions that operate on lists: len

Python has a function **len** that makes it easy to find the length of a sequence

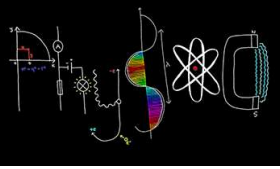
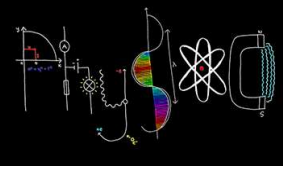
```
In [83]: len([1, 2, 3, 4, 5])
```

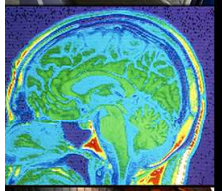
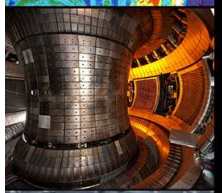
```
Out[83]: 5
```

Note that since a string is a sequence, **len** can be used to find the number of characters of a string (it's length):

```
In [84]: len("Hello Fred")
```

```
Out[84]: 10
```




Functions that operate on lists: sum and sort

You can use the **sum** functino to add the items of a list (if it contains only numbers)

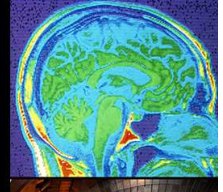
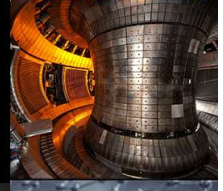
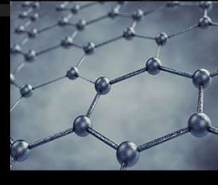
```
In [85]: sum([4, 10, 6])

Out[85]: 20
```

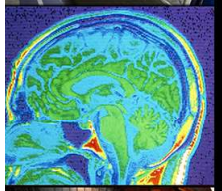
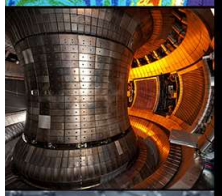
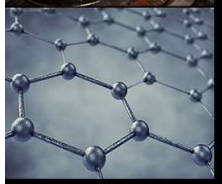
You can use the **sort** method to sort lists:

```
In [95]: x = [4, 10, 2, 5, 15]
         x.sort()
         x

Out[95]: [2, 4, 5, 10, 15]
```





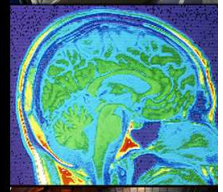
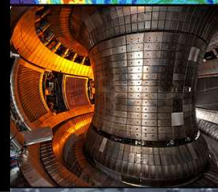














A note on Python objects

Most things in Python are objects.

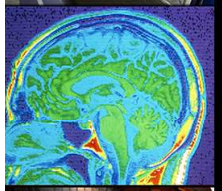
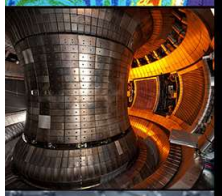
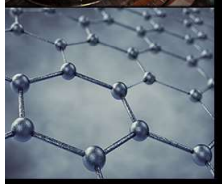
But what is an object?

Every thing we have showed you (integers, floats, strings etc.) are actually objects with a **type** and
associated **attributes** and **methods**.

For example:

x = [1, 3, 4, 5]

x.append() is a **method** of a list object (x)

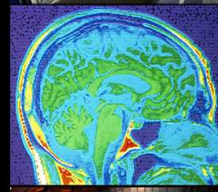
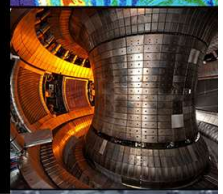














A note on Python objects

To understand the attributes and methods of an object, use **tab** completion in Ipython to inspect objects.

mylist = [3, 1, 4, 5]

mylist.<TAB>

This will show the available **attributes** and **methods** for the Python list mylist.

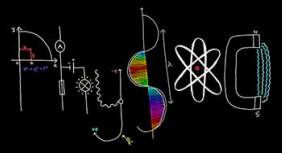
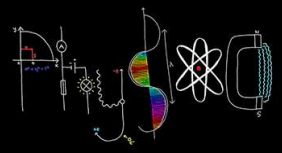
A note on Python objects

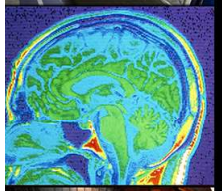
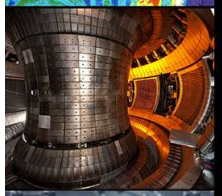
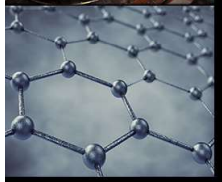
Using completion and help is a very efficient way to learn and later remember object **methods** and **attributes**

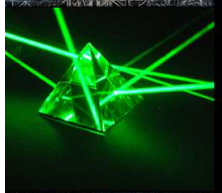
```
In [2]: x.
x.append    x.copy      x.extend    x.insert
 x.remove    x.sort
x.clear     x.count     x.index     x.pop
 x.reverse
```

If you want to know what a **method** does, you can use a question mark (to access the help files)

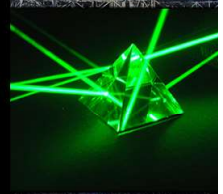
```
In [20]: x.append?
Type:        builtin_function_or_method
String form: <built-in method append of lis
t object at 0x0000000380B108>
Docstring:   L.append(object) -- append obj
ect to end
```

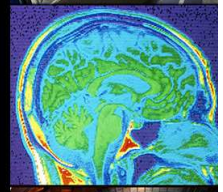
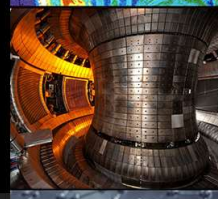













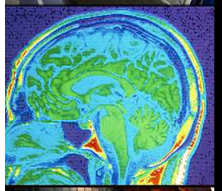



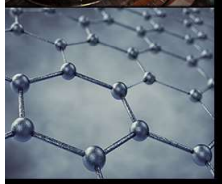

Dynamic typing

Unlike many other programming languages where types **have** to be declared for variables, Python is dynamically typed.

This means that variables can change their type during the course of a program:

```
In [98]: a = 1
         type(a)

Out[98]: int

In [99]: a = 2.3
         type(a)

Out[99]: float

In [100]: a = 'Hello'
          type(a)

Out[100]: str
```

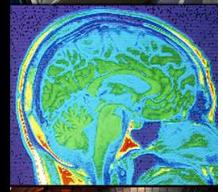
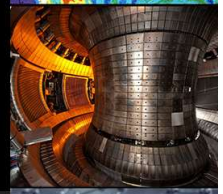










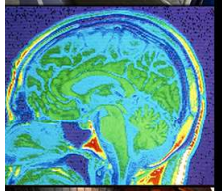
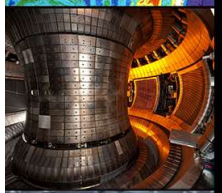
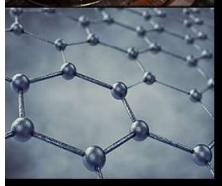



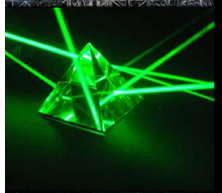

Converting between types 1

There may be cases where you want to convert a string to a float, or an integer to a string, etc.

For this you can simply use the int(), float(), and str() functions:

```
In [101]: x = '1'

In [106]: type(x)

Out[106]: str

In [108]: int(x)

Out[108]: 1

In [109]: type(int(x))

Out[109]: int
```

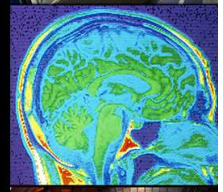
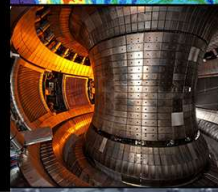














Converting between types 2

There may be cases where you want to convert a string to a float, or an integer to a string, etc.

For example:

```
In [110]: '5' + '4.31'

Out[110]: '54.31'
```

is different from:

```
In [111]: int('5') + float('4.31')

Out[111]: 9.309999999999999
```

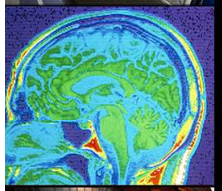
Converting between types 3

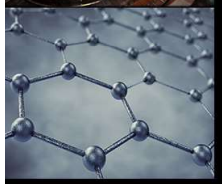
There may be cases where you want to convert a string to a float, or an integer to a string, etc.

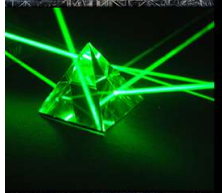
Similarly:

```
In [112]: x = 1.0
          x

Out[112]: 1.0

In [113]: str(x)

Out[113]: '1.0'
```

1.3B More on Strings





String Formatting 1

Strings are an important objects.

A string is what we call a group of characters (letters numbers and symbols), words and sentences.

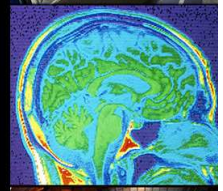
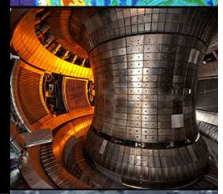
There are many format options with strings

The most basic way to define a string is using quotation marks

```
In [5]: x = "This is a basic string"
        y = 'This is also a string'
        z = ('This is also a string')
        w = """
        This is a string
        that can be on many lines
        """
```






String Formatting 2

And these can be viewed using the print command

In [94]:
```
print (x)
print x
print (y)
print(z)
print (w)
```

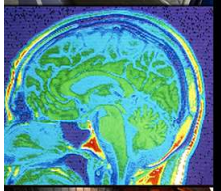
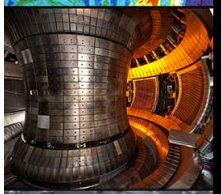
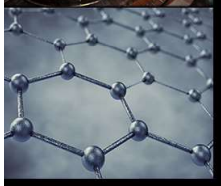
```
this is a string. It has no numbers
this is a string. It has no numbers
This is a string. It has the numbers 123
12345

This is a string
that can be on many lines
```

String Formatting 3

Or printed together:

In [100]:
```
print (x + z)

print (x + ' ' + z)

print x, z
```

```
this is a string. It has no numbers12345
this is a string. It has no numbers 12345
this is a string. It has no numbers 12345
```

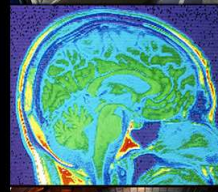
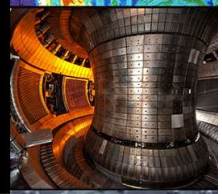














The “format” method 1

- format(value1, value2) (Formats numbers using {} notation)

  " This is text the value is {x1: y1} +/- {x2: y2}".format(value1, value2)

  where x1 is an index of the location of value1 where y1 is the format to print value 1 in

formats are:

.1f is for a float with 1 decimal place

.2f is for a float with 2 decimal places

05d is for a integer with 4 leading zeros

In [14]:
```
x = 1.2345678
y = 102.9809020903019

print ('I only want 2 decimal places: {0:.2f}').format(x)

print ('Now I want 1 decimal places: {0:.1f}, 3 decimal places: {1:.3f} and 4 decimal places: {0:4f}').format(x, y)

print ('And 3 leading zeros: {0:03d} and {1:03d}').format(int(x),int(y))
```

```
I only want 2 decimal places: 1.23
Now I want 1 decimal places: 1.2, 3 decimal places:
 102.981 and 4 decimal places: 1.234568
And 3 leading zeros: 001 and 102
```

The “format” method 2

Note you can only use leading zeros on integers:

In [17]:
```
x = 99
y = 100.2342
print ('I am a integer: {0:05d}').format(x)
```

```
I am a integer: 00099
```

In [20]:
```
print ('I am not an integer: {0:05d}').forma
t(y)
```

```
---------------------------------------------------
-----------------------
ValueError                                Traceback
 (most recent call last)
<ipython-input-20-344dc46411b2> in <module>()
----> 1 print ('I am not an integer: {0:05d}').form
at(y)

ValueError: Unknown format code 'd' for object of t
ype 'float'
```

Adding tabs and new lines 1

It may also be useful to add tabs and newlines

- '\n' adds a new line
- '\t' add a new tab

These are known as Escape Characters

In [3]:
```
x = ' We may want a new line \n\n Here is a
newline'
print (x)
```

```
 We may want a new line

 Here is a newline
```

In [2]:
```
y = ' We may want a tab \t\t\t here are thre
e! '
print( y )
```

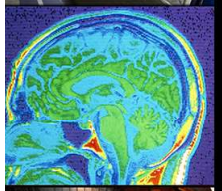
```
 We may want a tab                    here are t
hree!
```

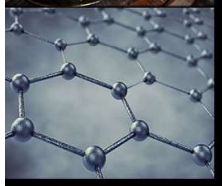
Adding tabs and new lines 2

To view a string with the escape characters in we can use the "repr()" function:

In [4]:
```
print(repr(x) )
print(repr(y) )
```

```
' We may want a new line \n\n Here is a newline'
' We may want a tab \t\t\t here are three! '
```

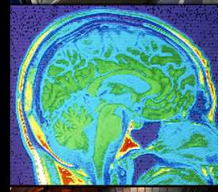
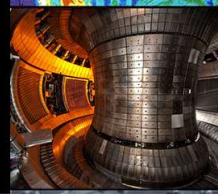











Special Characters 1

You may not be able to use some characters that are reserved (such as \ for \n and \t)

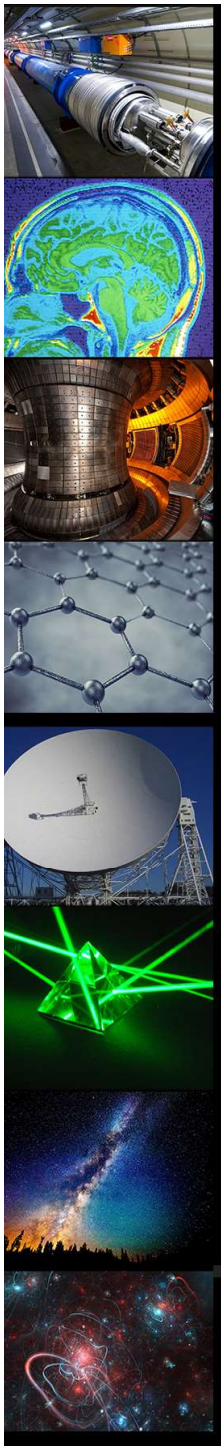
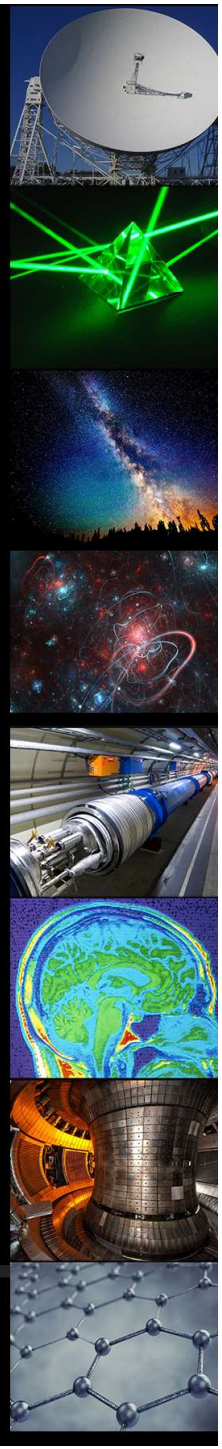
To use \ you will need to write the escape character \ first:

In [30]:
```
print ('I want to use a slash \')
```

```
  File "<ipython-input-30-124020873f05>", line 1
    print ('I want to use a slash \')
                                      ^
SyntaxError: EOL while scanning string literal
```

In [32]:
```
# Correct way
print ('I want to use a slash \\')
```

```
I want to use a slash \
```

Special Characters 2

We can then do as many slashes as we need:

```
In [36]: print ('Many slashes \\\\\\\\')

Many slashes \\\\
```

We can also tell python we want the string to be interpreted exactly the way we have written it (known as a raw string)

We can do this by using the following notation:

```
In [37]: print(r'Many slashes \\\\\\\\')

Many slashes \\\\\\\\
```

For more escape characters see here:

http://www.tutorialspoint.com/python/python_strings.htm

String Methods 1

There are many methods that you can use on strings. Here are just a few:

```
In [41]:
x = 'this is a string. It has no numbers'
y = 'This is a string. It has the numbers 123'
z = '12345'
```

```
- capitalize()
            Capitalizes the first letter of a string
- count(str, start, end)
            Counts how many times str occurs between index start and index end
            Returns integer
- endswith(str, start, end)
            Determines if the string ends with str (between index start and index end)
            Returns True if so and False otherwise
```

```
In [45]: # capitalize
         print(x.capitalize())

         This is a string. it has no numbers
```

String Methods 2

There are many methods that you can use on strings. Here are just a few:

```
In [41]:
x = 'this is a string. It has no numbers'
y = 'This is a string. It has the numbers 123'
z = '12345'
```

```
- find(str, start, end)
          Determine if str occurs in string (between index start and index end)
          Returns index if found or -1 if not found
```

```
In [47]: # find example 1
         print(x.find('string'))

10

In [49]: # find example 2
         print(z.find('string'))

-1

In [73]: # find example 3
         print(y[11:30])
         print(y.find('string', 11, 30))

tring. It has the n
-1
```

String Methods 3

There are many methods that you can use on strings. Here are just a few:

```
In [41]:
x = 'this is a string. It has no numbers'
y = 'This is a string. It has the numbers 123'
z = '12345'
```

```
- isalnum()
            Determines if string has at least 1 character and all characters are alphanumeric
            Return True if so or False if not
- isalpha()
            Determines if string has at least 1 character and all characters are alphabetic
            Returns True if so or False if not
- isdigit()
            Determines if string has at least 1 character and all characters are numeric
            Returns True if so or False if not
```

```
In [76]: # isalpha exanple
         print(x.isalpha())

False
```

```
In [77]: # isdigit example
         print(z.isdigit())

True
```

String Methods 4

There are many methods that you can use on strings. Here are just a few:

```
In [41]:
x = 'this is a string. It has no numbers'
y = 'This is a string. It has the numbers 123'
z = '12345'
```

```
- len(string)
            Determines how many characters there are in the string
            Returns integer
- lower()
            Changes all uppercase letters to lowercase letters
- max(string)
            Returns the max alphabetical character from string
- min(string)
            Returns the minimum alphabetical character from string
- replace(old, new)
            replaces any instances of substring old with string new
```

```
In [60]:  # length example
          len(x)

Out[60]:  35

In [62]:  #replace example
          print(x.replace('s', 'P'))

          thiP iP a Ptring. It haP no numberP
```

String Methods 5

There are many methods that you can use on strings. Here are just a few:

```
In [41]:
x = 'this is a string. It has no numbers'
y = 'This is a string. It has the numbers 123'
z = '12345'
```

- split(str)
  creates a list of substrings where the original string is split by str
- startswith(str, start, end)
  Determins if the string stars with str (between index star and index end)
  Returns True if so and False otherwise
- swapcase()
  Inverts case for all letters in string
- upper()
  Changes all lowercase letters to uppercase letters

```
In [68]:
# split example
my_list = x.split('.')
print(my_list)

print("The first element is: ")
print(my_list[0])
```

```
['this is a string', ' It has no numbers']
The first element is:
this is a string
```

String Methods 6

There are many methods that you can use on strings. Here are just a few:

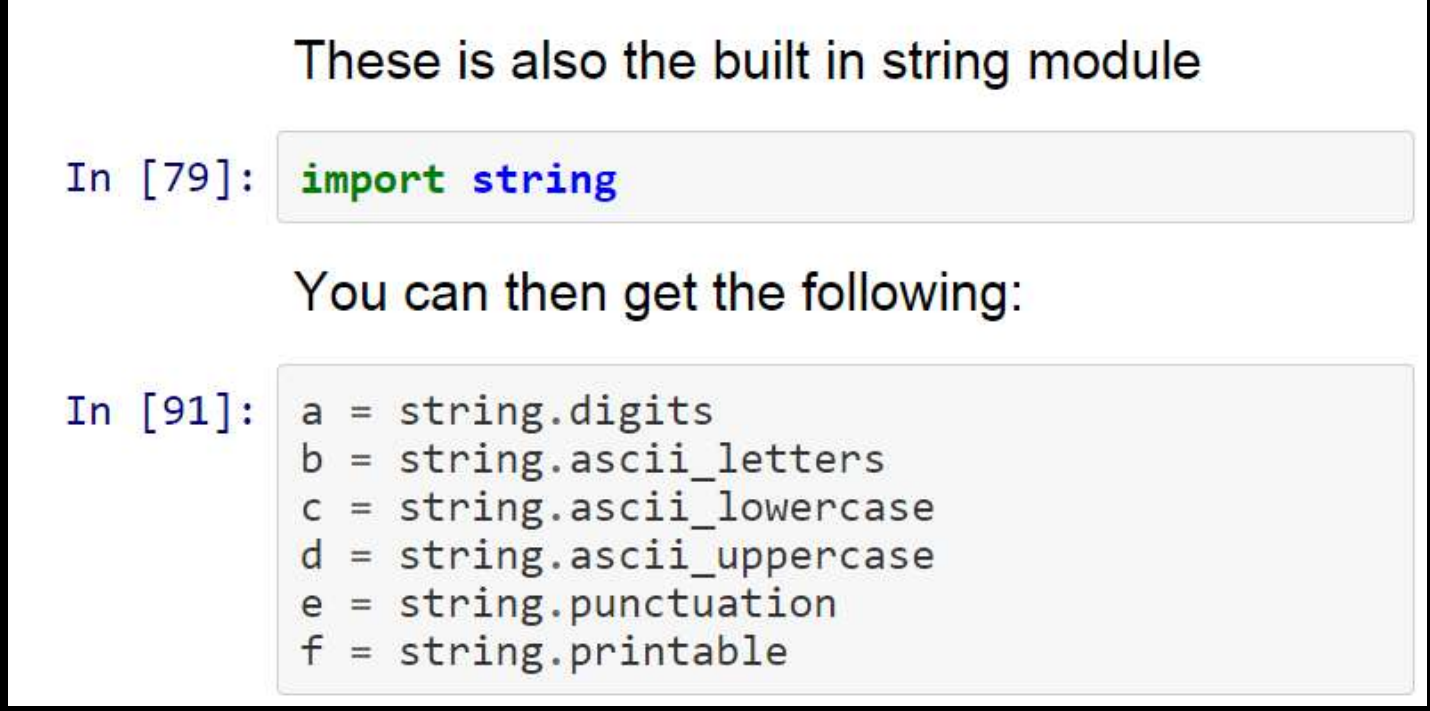
```
In [41]:
x = 'this is a string. It has no numbers'
y = 'This is a string. It has the numbers 123'
z = '12345'
```

```
- split(str)
            creates a list of substrings where the original string is split by str
- startswith(str, start, end)
            Determins if the string stars with str (between index star and index end)
            Returns True if so and False otherwise
- swapcase()
            Inverts case for all letters in string
- upper()
            Changes all lowercase letters to uppercase letters
```

```
In [78]: #upper and lower example
         print(y.upper())

         print(y.lower())

         print(y.swapcase())

THIS IS A STRING. IT HAS THE NUMBERS 123
this is a string. it has the numbers 123
tHIS IS A STRING. iT HAS THE NUMBERS 123
```

The String Module 1

These is also the built in string module



The String Module 2

These is also the built in string module

```
In [91]: a = string.digits
         b = string.ascii_letters
         c = string.ascii_lowercase
         d = string.ascii_uppercase
         e = string.punctuation
         f = string.printable
```

```
In [86]: print( a )
```

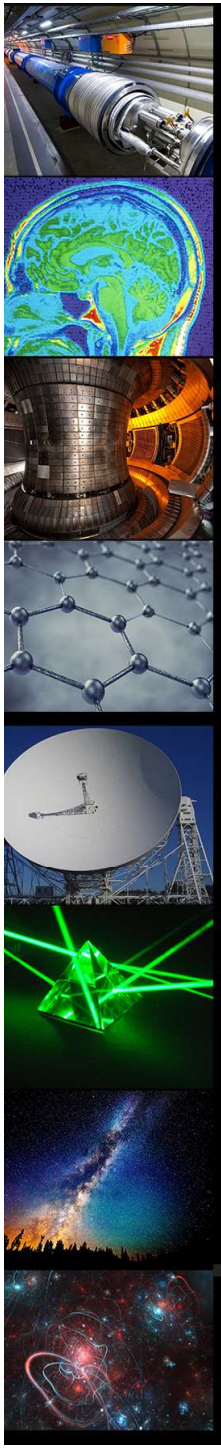
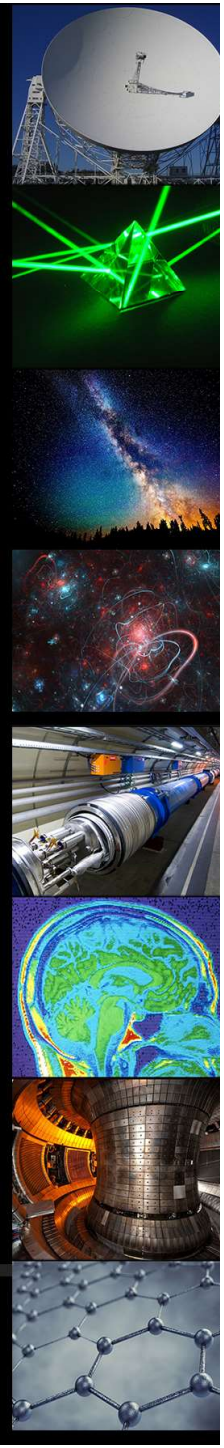
```
0123456789
```

```
In [87]: print (b )
```

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXY
Z
```

```
In [88]: print (c )
```

```
abcdefghijklmnopqrstuvwxyz
```

The String Module 3

These is also the built in string module

```
In [91]: a = string.digits
         b = string.ascii_letters
         c = string.ascii_lowercase
         d = string.ascii_uppercase
         e = string.punctuation
         f = string.printable
```

```
In [89]: print d
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
In [92]: print e
```

```
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

```
In [93]: print f
```

```
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNO
PQRSTUVWXYZ!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```