```
Extra Python material
=====================
The following will be additional content, we highly recommend that you type and play around with the code in the Python interpretor
in your terminal, and refer to the official Python documentation - https://docs.python.org/3/contents.html

Lists
-----
sorting - there are two ways

First:-
>>> lotto = [5,2,6,1,3,4]
>>> lotto
[5,2,6,1,3,4]
>>> sorted(lotto)
[1, 2, 3, 4, 5, 6]
>>> lotto
[5, 2, 6, 1, 3, 4]

Now let's try this:-
>>> lotto.sort()
>>> lotto
[1, 2, 3, 4, 5, 6]

<alist>.sort() only works with lists
sorted(<object>) works with any iterable
An iterable is an object that has an __iter__ method which returns an iterator, An iterator is an object with __next__ method
Whenever you use a for loop, or map, or a list comprehension, etc. in Python,
the next method is called automatically to get each item from the iterator, therefore going through the process of iteration.

key functions
-------------
key parameter to specify function to be called on each list before making comparisons

>>> sorted("This is a beginners Python workshop".split())
['Python', 'This', 'a', 'beginners', 'is', 'workshop']

Now let's use a key to sort by lowercase
>>> sorted("This is a beginners Python workshop".split(), key=str.lower)
['a', 'beginners', 'is', 'Python', 'This', 'workshop']

This technique is fast because the key function is called exactly once for each input record.

tuple
-----
Before we get into sorting complex objects, introduce a tuple. It's another sequence data type like lists.
It's like a list where it's a number of values seperated by comma but instead of [], it uses ().
>>> a_tuple = (1,2)
>>> a_tuple
(1, 2)
>>> a_tuple[1]
2

A tuple is immutable, it's a fixed value, you can't change it.
>>> a_tuple[1] = 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment

But a tuple can containt mutable objects.
>>> b_tuple = ([1, 2, 3], [4, 5, 6])
>>> b_tuple[1]
[4, 5, 6]
>>> b_tuple[1][1]
5
>>> b_tuple[1][1]=7
>>> b_tuple
([1, 2, 3], [4, 7, 6])

Let's sort complex objects. We have students stored in a tuple with their first name, grade and age:-
```

```
>>> student_tuples = [
...     ('john', 'A', 15),
...     ('jane', 'B', 12),
...     ('steph', 'B', 10),
... ]
>>> student_tuples
[('john', 'A', 15), ('jane', 'B', 12), ('steph', 'B', 10)]

Let's sort by their age with the following key:-
>>> sorted(student_tuples, key=lambda student: student[2])
[('steph', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]

range
------
One other sequence data type, this is called range.
This is a constructor that has to be integers (numbers).
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(range(0, 30, 5))
[0, 5, 10, 15, 20, 25]
>>> list(range(0, 10, 3))
[0, 3, 6, 9]
>>> list(range(0, -10, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> list(range(0))
[]


Sets
----
It's another data type, it's an unordered collection with no duplicate elements.
Support mathematical operations like union, intersection, difference and symmetric
difference.

Used {} to create a set.

>>> animals = {"dog", "cat", "parrot", "snake"}
>>> animals
{'dog', 'snake', 'parrot', 'cat'}

>>> "dog" in animals
True
>>> "fish" in animals
False

Let's create another set:-
>>> my_animals = {"cat", "rabbit"}

In animals but not in my_animals:-
>>> animals - my_animals
{'dog', 'snake', 'parrot'}

In either animals or my_animals:-
>>> animals | my_animals
{'rabbit', 'dog', 'cat', 'snake', 'parrot'}

In both animals and my_animals:-
>>> animals & my_animals
{'cat'}

In either animals or my_animals but not both:-
>>> animals ^ my_animals
{'dog', 'parrot', 'rabbit', 'snake'}


List comprehension
------------------
A concise and better performance way of creating lists.

Most common way:-
>>> squares = []
>>> for x in range(10):
        squares.append(x**2)
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
List comprehension way:-
>>> squares_list = [x**2 for x in range(10)]
>>> squares_list
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Say given a matrix:
>>> matrix = [
... [1, 2, 3, 4],
... [5, 6, 7, 8],
... [9, 10, 11, 12],
... ]

One way to transpose rows and columns:
>>> transposed = []
>>> for i in range(4):
...     transposed.append([row[i] for row in matrix])
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

You can use comprehension to transpose rows and columns:
>>> [[row[i] for row in matrix] for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

Eep, not as legible, but it does its job.

Here's a better way with inbuilt functions:
>>> list(zip(*matrix))
[(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]
```