Accessing Tuples. The Immutability of Tuples

This notebook looks at

1. how tuples can be accessed - using positive or negative indexes just like in the case of strings and lists. This is because tuples, like strings and lists, are an ordered collection of items.
2. the immutability of tuples - you cannot alter the contents of a tuple.

However, it is important to understand that if an individual element of a tuple is mutable, then the contents of that element can be altered inside a tuple.

Tuples can be accessed using indexes, just like we did for strings and lists. As in the case of strings and lists, the index begins at 0 and ends at a value one less than the number of items in the tuple. Negative indexes can also be used.

In [9]:

```
tuple1 = ('physics', 'chemistry', 1997, 2000)
tuple2 = (1, 2, 3, 4, 5, 6, 7 )
print(tuple1)
print(tuple2)
print(tuple1[3])
print(tuple2[-7])
```

```
('physics', 'chemistry', 1997, 2000)
(1, 2, 3, 4, 5, 6, 7)
2000
1
```

Like strings, tuples are also immutable. This means that individual elements of a tuple cannot be changed or removed. Remember however, this is not true for lists. Lists are mutable.

Once a tuple is created, we cannot change its contents by adding new values or deleting existing values. You cannot change elements in a tuple by replacing one element with another. Hence statements attempting to change a tuple are not valid and will generate errors.

In [10]:

```
my_str = 'this is a test'
my_int = 45
my_list =[4,8,9,10]
my_tuple = (my_str, my_int, my_list, 67, 'hello', [5,9,11])
my_list[3] = 18
print(my_list)
my_tuple[3] = 18 # will generate an error
```

```
[4, 8, 9, 18]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-8175b24afce5> in <module>
      5 my_list[3] = 18
      6 print(my_list)
----> 7 my_tuple[3] = 18 # will generate an error

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

Tuples are immutable. Therefore if individual elements of a tuple are assigned different values, the tuple still remains unchanged.

In [11]:

```
my_str = 'this is a test'
my_int = 45
my_list =[4,8,9,10]
```

```
my_tuple = (my_str, my_int, my_list, 67, 'hello', [5,9,11])
print(my_tuple)

my_str = 'this string is now different'
my_int = 78
my_list = [15, 20]

print(my_tuple)
print(my_list)
```

```
('this is a test', 45, [4, 8, 9, 10], 67, 'hello', [5, 9, 11])
('this is a test', 45, [4, 8, 9, 10], 67, 'hello', [5, 9, 11])
[15, 20]
```

However, it is important to understand that if the individual element of a tuple is mutable, than the contents of that element can be altered and this will be reflected in the contents of the tuple as well.
Look at the tuple, `my_tuple`, created below. `my_str`, `my_int`, the string literal and the integer constant are all immutable. However, `my_list`, the list constant is mutable. We can change the list contents inside the tuple as shown below.

In [12]:

```
my_str = 'this is a test'
my_int = 45
my_list =[4,8,9,10]

my_tuple = (my_str, my_int, my_list, 67, 'hello', [5,9,11])
print(my_tuple)

'''
If the contents of a mutable element in a tuple are changed,
this change is also reflected when we look at the tuple directly
'''
my_tuple[2][2] = 15
print(my_tuple)

my_list[3] = 16
print(my_tuple)
```

```
('this is a test', 45, [4, 8, 9, 10], 67, 'hello', [5, 9, 11])
('this is a test', 45, [4, 8, 15, 10], 67, 'hello', [5, 9, 11])
('this is a test', 45, [4, 8, 15, 16], 67, 'hello', [5, 9, 11])
```