Lecture 7.2 : Object-oriented programming: Special methods¶

Introducing special methods: `__init__()`

- So far we have been instantiating our `Time` objects and initialising them in a two step process as follows:

```
>>> from time_v02 import Time
>>> a = Time()
>>> a.set_time(13, 43, 6)
```

- Can we combine these two steps into one? Can we create *and* automatically initialise an object in one step? It turns out we can if in our class we define a *special method* called `__init__()` (there are two underscores before and after `init`). If a class contains an `__init__()` method then that method is automatically called immediately an object of that class is created. If we replace our old `set_time()` method with a suitable `__init__()` method our `Time` class becomes:

```
# time_v03.py
class Time(object):

    def __init__(self, hour, minute, second):
        self.hour = hour
        self.minute = minute
        self.second = second

    def show_time(self):
        print('The time is {:02d}:{:02d}:{:02d}'.format(self.hour,
                                                       self.minute,
                                                       self.second))
```

- Now if we try to create a `Time` object as before, we get an error:

```
>>> from time_v03 import Time
>>> a = Time()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: __init__() missing 3 required positional arguments: 'hour', 'minute'
```

- We get an error because `__init__()` will be automatically called upon object creation and it expects four arguments to be passed to it. One argument is automatically supplied (the object that becomes `self`) meaning we must supply three. Where do we supply the arguments expected by `__init__()`? We supply then in the only place we can i.e. as arguments to the `Time()` function as follows:

```
>>> from time_v03 import Time
>>> a = Time(13, 43, 6)
>>> a.show_time()
The time is 13:43:06
```

- When the line highlighted above is executed the following takes place:

    0. An empty instance of the `Time` class is created,
    1. this empty object is passed along with `13`, `43` and `6` to the `__init__()` method,
    2. the `__init__()` method initialises the object with the supplied arguments,
    3. a reference to the the new and now initialised object is returned and assigned to `a` by the caller

- Note that `__init__()` is a special method. The fact that it is special is indicated by the double underscore prefix and suffix. Special methods are not called directly. Thus *under normal circumstances* we will not call `__init__()` directly.

- From now on any classes we write will typically contain an `__init__()` method. A suitable `__init__()` method is one of the first things we should start thinking about when writing a new class.

Default `__init__()` argument values

- Note an `__init__()` method is just like any other function in that it supports *default arguments*. This is very handy. It means we can initialise a new object to some default state when the user does not supply any arguments during object instantiation. Thus our *final* `__init__()` method looks like this:

```
# time_v04.py
class Time(object):

    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    def show_time(self):
        print('The time is {:02d}:{:02d}:{:02d}'.format(self.hour,
                                                       self.minute,
                                                       self.second))
```

- Now we can instantiate our `Time` objects with zero, one, two or three arguments. Any missing arguments will take on the default values specified in the `__init__()` method:

```
>>> from time_v04 import Time
>>> a = Time()
>>> a.show_time()
The time is 00:00:00
>>> a = Time(16)
>>> a.show_time()
The time is 16:00:00
>>> a = Time(16, 30)
>>> a.show_time()
The time is 16:30:00
>>> a = Time(16, 30, 59)
>>> a.show_time()
The time is 16:30:59
```