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.

```python
class Time(object):

    def set_time(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))

a = Time()
a.set_time(13, 43, 6)
```

- Can we create *and* automatically initialise an object in one step?
- 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.
- We replace our old `set_time()` method with a suitable `__init__()` method giving the following `Time` class implementation.

```python
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.

```python
a = Time()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [3], in <module>
----> 1 a = Time()
```

```
TypeError: __init__() missing 3 required positional arguments: 'hour', 'minute', and '
```

- 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.
- How do we supply the arguments expected by `__init__()`?
- We supply them in the only place we can i.e. as arguments when creating a `Time` object as follows.

```
a = Time(13, 43, 6)
a.show_time()
```

```
The time is 13:43:06
```

- When we call the `Time` class now the following takes place:

  1. An empty instance of the `Time` class is created,
  2. this empty object is passed along with `13`, `43` and `6` to the `__init__()` method,
  3. the `__init__()` method initialises the object with the supplied arguments,
  4. a reference to 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 normally 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__()` parameter values

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

```
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.

```
a = Time()
a.show_time()

a = Time(16)
a.show_time()

a = Time(16, 30)
a.show_time()

a = Time(16, 30, 59)
a.show_time()
```

```
The time is 00:00:00
The time is 16:00:00
The time is 16:30:00
The time is 16:30:59
```

Another special method : `__str__()`

- Another special method that we typically implement is `__str__()`.
- Whenever Python sees `print(class_instance)` it checks whether the class in question defines a method named `__str__()`.
- If it does the method is invoked (and passed a copy of the object as usual in `self`).
- What is printed is the string *returned* by the `__str__()` method.
- We can replace our `show_time()` method with this special method to make printing times handier and more intuitive.
- Below find the updated class and a demonstration of the method in action.

```
class Time(object):

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

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

```python
a = Time()
print(a)

a = Time(16)
print(a)

a = Time(16, 30)
print(a)

a = Time(16, 30, 59)
print(a)
```

```
The time is 00:00:00
The time is 16:00:00
The time is 16:30:00
The time is 16:30:59
```