> *It has always been the calm leadership of the officer class that has made the British Army what it is.*

Introduction to Object Oriented Programming

In the previous section, we looked at ADTs. In this section, we extend the idea of ADTs to classes, objects and object oriented programming.

Classes and Objects

Object oriented programming is a style of programming based on **objects**. Objects, like ADTs, contain **data** (state) and **methods** (the behaviour of the object). The methods of an object can be simple — consisting of accessors, mutators and tests. In this case, we can think of such objects as ADTs as we have seen before. On the other hand, some objects can have methods that encode sophisticated behaviours — in this case, we would **not** consider such objects as ADTs.

All objects of the same kind (e.g. string objects) contain the same kind of state (e.g. all strings are a sequence of characters) and have the same methods. We can abstract the essence of such a collection into a **class**. A class describes the abstract characteristics of all the objects that are **instances** of the class. In other words, a class describes the state and methods any instance should have. For example, the string class describes what characteristics any string object should have — some representation of the characters of a string and various methods to access and process the characters of the string.

Design in an object oriented language consists of two major parts: finding suitable classes in existing modules (libraries); and designing our own classes when suitable classes do not exist.

In the remainder of this section we will restrict ourselves to the first part — using existing classes. In particular we will reconsider strings and look at another kind of sequence — lists. We will also look at file objects.

Methods

Methods are similar to functions we have seen before but are accessed in a different way. Because methods (and state) are really part of an object. We use the 'dot notation' to access a given method of a given object. The examples below illustrates the use of this notation.

```
>>> s = 'spam spam spam spam'
>>> s.find(' ')
4
>>> s.partition(' ')
('spam', ' ', 'spam spam spam')
```

Notice that these two methods have the same functionality as the `find` and `partition` functions that we wrote in the previous notes.

Let's have a look at another string method that is available:

```
>>> s.split(' ')
['spam', 'spam', 'spam', 'spam']
>>> help(s.split)
Help on built-in function split:

split(...)
    S.split([sep [,maxsplit]]) -> list of strings

    Return a list of the words in the string S, using sep as the
    delimiter string.  If maxsplit is given, at most maxsplit
    splits are done. If sep is not specified or is None, any
    whitespace string is a separator and empty strings are
    removed from the result.

>>> s.split()
['spam', 'spam', 'spam', 'spam']
>>> s.split(' ', 2)
['spam', 'spam', 'spam spam']
>>>
```

The second example shows the use of the help command to be able to find out more information on what the method does. The next two examples use the `split` method of strings in more interesting ways. `split` returns a **list**, we will get to lists in a short while.

To be able to find out all the methods available to a particular object the dir command is useful.

```
>>> dir(s)
['__add__', '__class__', '__contains__', '__delattr__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__', '__le__',
'__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', '_formatter_field_name_split', '_formatter_parser',
'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs',
'find', 'format', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace',
'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

As can be seen `dir(s)` lists all the methods available for strings. For now, we can ignore the methods with the double underscores we will get back to these later. Once we know the existence of a method, we can use help to get more information on that method, as we did with the `split` method above.

Method Calling Syntax

objectname.*methodname*(*arg-list*)

where *objectname* is the variable whose value is the object we wish to call the method on, *methodname* is the name of the method and *arg-list* is the list of arguments of the method.