11.3 Directories

When you create a new file by opening it and writing, the new file goes in the current directory (wherever you were when you ran the program). Similarly, when you open a file for reading, Python looks for it in the current directory.

If you want to open a file somewhere else, you have to specify the **path** to the file, which is the name of the directory (or folder) where the file is located:

```
>>>   f = open("/usr/share/dict/words","r")
>>>   print f.readline()
Aarhus
```

This example opens a file named `words` that resides in a directory named `dict`, which resides in `share`, which resides in `usr`, which resides in the top-level directory of the system, called `/`.

You cannot use `/` as part of a filename; it is reserved as a delimiter between directory and filenames.

The file `/usr/share/dict/words` contains a list of words in alphabetical order, of which the first is the name of a Danish university.

11.4 Pickling

In order to put values into a file, you have to convert them to strings. You have already seen how to do that with `str`:

```
>>> f.write (str(12.3))
>>> f.write (str([1,2,3]))
```

The problem is that when you read the value back, you get a string. The original type information has been lost. In fact, you can't even tell where one value ends and the next begins:

```
>>>   f.readline()
'12.3[1, 2, 3]'
```

The solution is **pickling**, so called because it "preserves" data structures. The `pickle` module contains the necessary commands. To use it, import `pickle` and then open the file in the usual way:

```
>>> import pickle
>>> f = open("test.pck","w")
```

To store a data structure, use the `dump` method and then close the file in the usual way:

```
>>> pickle.dump(12.3, f)
>>> pickle.dump([1,2,3], f)
>>> f.close()
```

Then we can open the file for reading and load the data structures we dumped:

```
>>> f = open("test.pck","r")
>>> x = pickle.load(f)
>>> x
12.3
>>> type(x)
<type 'float'>
>>> y = pickle.load(f)
>>> y
[1, 2, 3]
>>> type(y)
<type 'list'>
```

Each time we invoke `load`, we get a single value from the file, complete with its original type.

11.5 Exceptions

Whenever a runtime error occurs, it creates an **exception**. Usually, the program stops and Python prints an error message.

For example, dividing by zero creates an exception:

```
>>> print 55/0
ZeroDivisionError: integer division or modulo
```

So does accessing a nonexistent list item:

```
>>> a = []
>>> print a[5]
IndexError: list index out of range
```

Or accessing a key that isn't in the dictionary:

```
>>> b = {}
>>> print b['what']
KeyError: what
```