823G5 Programming in Python

Aim

To familiarise yourself with

- Using multi-dimensional lists
- Stacks
- Tuples
- Dictionaries

Exercises

Download the `Lab6Exercises` ZIP file. It contains some basic code for you to develop. The code is a set of Python source code file (`.py`). You will need to open a new PyCharm project and import the `.py` files into it. Refer to the Lab 1 instructions if you cannot remember how to do this.

Working with multi-dimensional lists

Some problem domains may be conveniently thought of as 2 dimensional or more array like structures (effectively, matrices). For example, a tic-tac-toe board is a 3x3 structure with 3 rows and 3 columns. In Python we can represent such multi-dimensional structures as "lists of lists". Here is an example…

```
a = [ [1, 2, 3], [4, 5, 6] ]
```

This is an 2D list with 2 rows and 3 columns. We refer to an entry in the structure using two sets of square brackets:

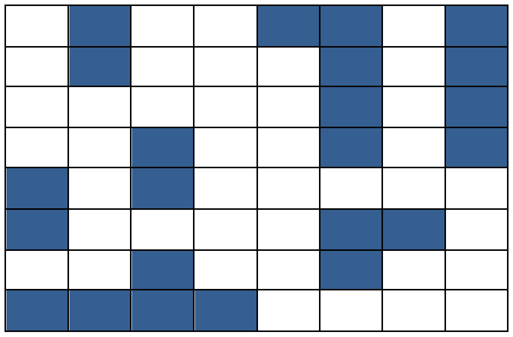
`a[0][0]` means "row 0, column 0": 1 in this case
`a[1][2]` means "row 1, column 2": 6 in this case

We can conveniently display this using a double iteration loop such as:

```
for row in a:
    for elem in row:
        print(f'{row},{elem}')
```

Coding challenge 1: Random maze search

Open the `challenge1.py` file. It contains the start of a program that can automatically search a maze using a simple random walk strategy. The maze is an 8 x 8 structure that can be visualized like this:



We start the search at the top left of the maze (list location (row,column) = (0,0)) and we want to find a path to the bottom right (list location (row,column) = (7,7)). We can only traverse the light-coloured squares (these entries have the value 0 in the 2D list). The dark-coloured squares are rocks (value 1 in the 2D list).

We need to:

- Use random navigation to discover a path from the start to the end points of the maze.
- We can only travel in one of four directions: north, south, east and west.
- We travel in straight lines unless we meet an obstacle.
- We start by moving east.
- If we find that our path is blocked, we choose another direction until we can move.
- We cannot move through the boundary of the maze.
- We keep track of our legal moves using a simple stack. Each time we move in a certain direction, we push a string to a stack to record our journey.

You need to implement the `find_path()` and `display_path()` functions.

A suggested answer is provided in Canvas. As a variation, use a queue instead of a stack. The stack will have our complete path in reverse order, a queue would solve this problem.

Coding challenge 2

For this challenge, we shall create a simple contacts application. Contacts will be represented as key/value pairs in a dictionary.

Open the `challenge2.py` file. Keys and values can be any valid singular Python entity. In most practical cases, keys are simple things like a single string or a number. Values can usefully be things like tuples. That is what our example here has. The 3-tuple value stores three pieces of contact information; an email address, the position in the Acme company and the telephone extension number.

There are 4 functions in the file to be completed. You can see clearly from the Python docstring in each case what they are supposed to do.

Take note also of the comments in the file on the nature of global variables. A global variable has scope over the whole of your program. This can be useful where you have data that is used by many functions, methods and classes. But it should be used sparingly and only where it represents a clear and sensible solution to a problem.

To create a global variable, either:

- Initialise it outside the scope of any function, method or class definition.
- Initialise it explicitly using the global keyword.

In the example here, the contact dictionary is established as a global variable.

An example solution is available on Canvas.

Dr. Benjamin Evans
B.D.Evans@sussex.ac.uk