Exercises and Challenges

These exercises are suggested activities for you to practice your Python with. Some have a set goal in mind, while others are more open-ended - there's plenty to do here, so pick something that looks interesting, and give it a go! Alternatively, feel free to come up with your own activities to try.

Text-Based Games

Try making a game in Python that uses only the text prompt (`input`) and text output (`print`). This could range from something quite simple to something quite in-depth and complex! Some possible types of game are given below - or you could come up with your own!

Think carefully about how you will structure your game. While you could just dive in with conditions and loops, and hard-code all aspects of your game, this risks becoming messy for larger projects. Consider whether you might be able to organise your data (questions, levels, etc) into data structures such as lists or objects, and use those throughout your program - this might make it much easier to modify and expand!

Simple: Quiz

Create a quiz game, where the player is asked various questions that they must answer.

- You could have the same questions in the same order, or perhaps a large selection of questions from which a few are chosen at random!
- Will the user simply have to enter the correct answer, or will it be multiple choice?
- Perhaps questions could be generated, rather than pre-written - such as a maths quiz. How would you go about doing this?

Advanced: Text-Based Adventure

In a text-based adventure game, the player is given a description of the scene, and enter what they want to do next, typically as part of some large story or adventure. They might have to solve puzzles, fight creatures, or unravel a conspiracy.

- Think what actions the player might want to take. What if they enter something your program doesn't recognise, or that doesn't make sense?
- Perhaps the player could have an inventory of items that they could pick up to use on their quest?
- You will probably find it useful to look at the advanced string manipulation functions available in Python: https://docs.python.org/3/library/stdtypes.html#string-methods. You can also treat strings as lists of individual characters, to extract parts of a string:

```
x = "Hello World"
print(x[0])         # Outputs "H"
print(x[4])         # Outputs "o"
print(x[:5])        # Outputs "Hello"
print(x[3:7])       # Outputs "lo W"
```

- If you want more information about the genre, and some examples / inspiration, take a look at https://en.wikipedia.org/wiki/Interactive_fiction.

Sorting Algorithms

Python provides functions that will sort lists for us. However, if you're interested in the ways in which sorting is actually performed, you might wish to try implementing your own. We have already seen one approach: Bubble Sort, a quite inefficient sorting algorithm. Below are some other sorting algorithms for you to try.

Once you have written a sorting algorithm, you'll want to make sure it works. Try creating a program that generates a random list of values and sorts them using your algorithm, then check the results. This will be easiest if you write your sorting algorithm as a function.

Intermediate: Insertion Sort

https://en.wikipedia.org/wiki/Insertion_sort

Insertion sort works by ensuring that the first *i* elements of the list are sorted before adding in another, starting from the trivial case *i = 1*, and increasing *i* until *i = n*, where *n* is the length of the list. When *i* is increased, the newly added element is *inserted* into the appropriate place in the sorted list, which is possible because we already know the rest of that list to be sorted.

Advanced: Quicksort

https://en.wikipedia.org/wiki/Quicksort

Quicksort is a highly efficient sorting algorithm in widespread use. It was one of many inventions of noted Oxford computer scientist Sir Tony Hoare (http://www.cs.ox.ac.uk/people/tony.hoare/).

It works by choosing an element within the list as a 'pivot' - the rest of the list is split into two smaller lists, one containing values less than the pivot value, and the other containing values greater than the pivot value. Each of these lists is then sorted using Quicksort - this is *recursion* - before being stuck back together at the end. A sorted list, plus a value higher than every value in that list, plus a sorted list higher than that value, must clearly be sorted! The recursion's terminating condition is a list containing 0 or 1 elements, which automatically counts as sorted.

Turtle

We have previously seen the `turtle` module - a fun module that lets us control a 'turtle' around the screen, leaving a trail behind it. Below are some suggested activities to try with this module.

You will want to refer to the documentation throughout these activities:
https://docs.python.org/3/library/turtle.html

Simple: Shapes and Patterns

Have the turtle draw out some shapes and patterns, by moving and changing direction.

- You could create polygons, or spirograph-like patterns, or anything you can think of!
- Try varying how the turtle draws by adjusting the pen size and colour, to make your drawings more interesting.

Intermediate: Controllable Turtle

Try using the module's in-built mouse input to direct the turtle. For instance, you could get the turtle to head to where you click, or to draw a specific pattern at some position.

Intermediate: Multiple Turtles

It is possible to operate multiple turtles on-screen at the same time. To create some new turtle objects, and assign them to variables, use:

```
x = turtle.Turtle()
y = turtle.Turtle()
```

Now `x` and `y` are both turtles, and can be controlled individually.

```
x.forward(100)
y.left(125)
y.forward(70)
```

Try using multiple turtles for something!

- You could create multiple patterns in different parts of the screen.
- You could create a local multiplayer game, where each player takes it in turns to control their turtle to achieve some objective.

Graphical User Interfaces

Sometimes you might want your program to have a more user-friendly interface than just a text prompt. One Python module that lets you create Graphical User Interfaces (GUIs) is `tkinter`. Refer to the documentation at https://docs.python.org/3/library/tkinter.html when attempting these exercises.

Intermediate: Simple GUI

By looking at the documentation and studying the example program given, try making your own very simple GUI, with various elements that perform different actions.

- To begin with, you can just copy the example program, and see what it does.
- Then, try tweaking it - do you adjustments do what you expect them to?
- The documentation linked does not contain all the detail that there is, but at the beginning it lists further resources for you to explore, to see what else can be done with this module.

Advanced: GUIs for Other Exercises

Try augmenting one of the other exercises on this sheet, or from any of the lessons, with a custom GUI! Possibilities include:

- A control panel for turtle.
- A fancy interface for a text-based adventure.
- A fancy interface for an online system.

Networking and the Internet

There are various ways of using Python with networks and the Internet. Networking is an interesting subject area, but can sometimes be complicated and fiddly. Giving a course on networking is slightly beyond our scope, so if you are unfamiliar with networks you may want to do some research online before / while attempting these exercises!

Intermediate: Downloading from the Internet

Take a look at the documentation for the `urllib.request` module, and see if you can work out how to download a web page or file hosted online, and output its contents to the screen.

- https://docs.python.org/3/library/urllib.request.html
- You may find the 'Examples' section of most use. Note that this uses some advanced Python syntax that we haven't seen - either work out how to use it, or find a way to avoid having to.
- Once you can print the downloaded data to the screen, try saving it to a file instead. Then you can open the downloaded files with the appropriate program.
- (Note: please be courteous when using programs that interface with websites you do not own, and avoid doing anything malicious, such as spamming requests [denial of service]. If you want a 'safe' website to practice with, use *ox.compsoc.net*.)

Advanced: Networking with Sockets

We can perform more advanced networking using *sockets* with the `socket` module in Python. Technical details of the `socket` module can be found in the documentation: https://docs.python.org/3/library/socket.html; however, for a step-by-step walkthrough of what sockets are and how to use them, take a look at this tutorial: http://www.binarytides.com/python-socket-programming-tutorial/. Work your way through the tutorial, to implement both a more complex website downloader, and then your own very simple server.

- The tutorial uses `try … except …` blocks, which we haven't seen before. Essentially, the code in the `try` block is run as normal; however, if it crashes for any reason, then the `except` block is run. This is called *catching an exception*.
- If you manage to get the server example working, and you understand it, try writing a very simple chat client! This is an advanced but quite rewarding exercise.
- To test a server on your own computer, you can refer to it using the address 'localhost', or equivalently the IP address 127.0.0.1. Unfortunately, you will not be able to test your server between your computers over *eduroam*, due to security limitations.
- (Writing networking code can be fun when it works, although frustrating when it doesn't. However, unless you have a lot of experience, it is probably best not to expose your own programs to the internet immediately - there are myriad security considerations that we won't go into here.)