SOC presents

Intermediate Python

Quick Admin Stuff

- **docsoc.co.uk/education** for all resources
  - Some errors have been fixed
  - Feedback form is now correct
- I will be posting code examples too
  - jackel119/python102 on GitHub

Recap from last time

- Classes and Objects: Person Class
  - A class has data, and methods which act on that data
  - A class is a template for objects

Person Class

```
class Person:

    Constructor
    def __init__(self, name):
        self.name = name  Field

    def greet(self):  Method
        print("Hello! My name is", self.name)
```

Person class

- We created a single class to encapsulate a Person, so we can create multiple people.
- Each Person object
  - can be created simply and easily
  - has the same functionality
- You should now **begin** to see why Object-Oriented Programming is useful

Example 2

- You want a simple, command line rock-paper-scissors game
- How would you do this?

Example 2

- You want a simple, command line rock-paper-scissors game
- How would you do this?
  - Option 1: Create a RockPaperScissors class to encapsulate the game

Example 2

- You want a simple, command line rock-paper-scissors game
- How would you do this?
  - Option 1: Create a RockPaperScissors class to encapsulate the game
  - If you were to host a board games night, what would you need?

Example 2

- You want a simple, command line rock-paper-scissors game
- How would you do this?
  - Option 2: Create a RockPaperScissors game class, as well as a Player class.
  - This allows us to separate (and possibly later reuse) the logic

Example 2

- You want a simple, command line rock-paper-scissors game
- How would you do this?
  - Option 2: Create a RockPaperScissors game class, as well as a Player class.
  - This allows us to separate (and possibly later reuse) the logic

How should do the two classes interact with each other now?

Interfaces (AKA an API)

A set of rules which define how a component should interact with another

Interfaces

- A very general programming concept - not Python specific
  - Other languages have features to **enforce** an interface
  - You will also hear about **Web APIs**
- An interface isn't good or bad by itself - it depends on the context and use cases
- Abstracts away the usage from the implementation
- **"Design"** of a program/application

Our game interface

RockPaperScissors class:

- moves() method to give a list of possible moves (Rock, Paper, Scissors)
- play() method to play the game, and prints out the winner

Our game interface

RockPaperScissors class:

- moves() method to give a list of possible moves (Rock, Paper, Scissors)
- play() method to play the game, and prints out the winner

HumanPlayer class:

- pick_action() method to select a move to play

Our game interface

RockPaperScissors class:

- moves() method to give a list of possible moves (Rock, Paper, Scissors)
- play() method to play the game, and prints out the winner

HumanPlayer class:

- pick_action() method to select a move to play

Once this has been agreed, we can now get started!

What if we want to add an AI Player class?

What design decisions have we made?

Menti

- Think about type signatures of methods
- How we create each object and use them
- Would we change our design if our use case was different?

What design decisions have we made?

- Think about type signatures of methods
- How we create each object and use them
- Would we change our design if our use case was different?
- What bad decisions could we have made?

What if we want a Tic-Tac-Toe game now?

What could we reuse?

What would we need to add/change?

Break Time

Inheritance

What if we want to have lots of classes that are similar in lots of ways but not exactly the same?

Inheritance

*We can have a Class **inherit** from another Class!*

*I.e. Students and Lecturers are both **Persons***

Student Class

```
class Student(Person):


    def __init__(self, name, age, subject):
        super().__init__(name, age)
        self.subject = subject
```

Student Class

```
class Student(Person):


    def __init__(self, name, age, subject):
        super().__init__(name, age)
        self.subject = subject
```

Superclass

Inheritance

Student is now a subclass of Person

Person now a superclass of Student

Student Class

```
class Student(Person):


    def greet(self):
        super().greet()
        print("I am studying", self.subject)
```

Student Class

```
class Student(Person):

    def greet(self):
        super().greet()
        print("I am studying", self.subject)
```

Can still access everything from the (parent) superclass

Inheritance

Is it possible to inherit from multiple classes at the same time?

OOP Recap

- Allows programs to be thought of as a lots of smaller, different components
- Allows you to write code once and reuse it multiple times
- Interfaces abstract away responsibility
- Easy to split work up
- “Design” of software
- Often the diff. Between “programmers” and “software engineers”

Numpy, Pandas, and Scientific Computation

Why is vanilla Python not great for scientific/mathematical computation?

Why is vanilla Python not great for scientific/mathematical computation?

Hint: think in terms of what you might wanna do to data, lists/tables of data, types of data, etc

Why vanilla Python is **bad** for data:

- Lists don't enforce types
- Side effects might happen if not careful
- No element-wise operations
- No support for "tables"
  - Could use nested lists, but difficult
  - Index by?
- *A million other reasons!*

Enter NumPy:

- Has a very powerful N-dimensional array object
  - Fast
  - Easy to generate
  - Can enforce types
  - Has TONS of useful methods/operations
- Linear Algebra (and Matrix operations) support
- Other useful functions as well

A quick note on importing/installing 3rd party packages:

- Python's package manager is called **pip**.
- There is a **pip2** and **pip3**, for **python2** and **python3** respectively. Make sure you are using the right one.
- Generally, the syntax to install is:
  - pip install <package>
  - pip uninstall <package>

Why vanilla Python is bad for data:

- Lists don't enforce types Numpy
- Side effects might happen if not careful Numpy
- No element-wise operations Numpy
- No support for "tables"
  - Could use nested lists, but difficult
  - Index by?
- *A million other reasons!*

Enter Pandas:

- Series object, similar to 1-D Numpy Array (actually built on top of it)
- DataFrame object, which represents a table
  - Has column names (which are accessible)
  - Row accessible
  - Again, LOTS of features
- Lots of other useful datatypes (dates, times, etc)
- Combined with Numpy, has anything and everything you will ever need for data processing

What about visualising data?

What about visualising data?
We have **matplotlib**

That's it for this week!

- docsoc.co.uk/education for all resources
- Next week(?), what should we look at? Either:
  - Web interaction via HTTP, using Web APIs, scripting
  - More data processing/statistics, perhaps with some data science/machine learning
  - Open to suggestions!