# *Classes & Object Orientation*

**Problem 1.** Define a class which holds a list of numbers, and can perform a series of operations on them:

- Average
- Maximum and minimum
- Return a random number
- Adding and removing numbers

The class should take a list of numbers as its initial state when it is initialized.

**Problem 2.** Define a `Person` class, which takes their `name` as an argument when it initializes. The person should have a method, `greet`, which introduces to the user.

**Problem 3.** Implement an ATM, which keeps track of the `total` amount of money it has.

The user should be able to interact with the ATM, and it provide options until they would like to exit. Implement some method, `Interact`, which allows the user to interact with the ATM.

Hint: this sounds a lot like a `while` loop...

It should have some methods that the user can select via this interface, which you can implement as you think makes most sense:

1. Withdraw
2. Deposit
3. Current balance
4. Exit

**Problem 4.** *Part 1:* Implement a `Snail` class which can `stepForwards` and `stepBackwards`. `stepForwards` and `stepBackwards` should take an argument describing the *distance* that the Snail moves. The snail should know where it is, so keep track of the snail's position, by keeping an instance attribute of how far it has moved.

The snail should have a unique identifier. Use a *class attribute* to keep track of how many snails exist, and use an *instance attribute* to keep track of which number the snail is.

*Part 2:* Make five snails. Write a loop which sees how far the snails go after fifty iterations, and prints the state of the race nicely.

The snails should move randomly forwards and backwards, and should move random distances, too. You can decide how this randomness is implemented! Two ideas we have had are to implement a method which returns a random direction and distance for the snail, or alternatively, to generate these random choices in the loop.

**Problem 5.** Make a class that defines a point with an x and y co-ordinate. These should be passed as arguments when the class is initialized.

Use this class to define shapes. Develop:

- A circle class, which holds a `Point` and some radius. The Circle should have methods for calculating its area and circumference.
- A parallellogram class, which can represent any class made of four edges, and edges for these angles. Then develop a Rectangle subclass, which takes two points and works out the other points to use when initialising the Parallelogram, and a Square which operates similarly for Rectangles. You should be able to calculate areas of your parallelograms.

**Problem 6.** Implement a `Set` class with the following methods:

1. a method which checks if an element is in the set
2. a method which adds an element to a set without duplicates
3. a method which coverts a list into a set.

**Problem 7.** Implement a class called `DynamicProgramming`. When an object of this class is initialized, it takes a function `f` as an argument and it sets up a dictionary internally which stores the inputs and outputs of the function `f` in a similar manner to problem 4 in the "dictionaries" problem set. You should implement the `__call__` method which takes the input `x` of `f` and returns the value of `f(x)`. Implementing the `__call__` method allows one to use an object of the class as if it were a function; for example, if `a = DynamicProgramming(f)`, for some function `f`, then we may call the `__call__` method of `a` on an input `x` either by writing `a.__call__(x)` or by writing `a(x)`.

To demonstrate:

```
class DoubleClass:
    def __call__(self, n):
        return n * 2

d = DoubleClass()
d(6)     # => 12
```

Here, `d` is an instance of `DoubleClass`, but when we try to call it, Python uses the `__call__` function we wrote.

**Problem 8.** Extend your `Person` class to include a person's height.

We want to sort people by their height, using the built-in method `sorted`. To do this, we need to be able to compare two people and say whether one is bigger than the other.

Implement `__lt__(self, other)`, a method which returns `True` if and only if `self` is smaller than `other`. You should also implement `__eq__(self, other)`, which returns whether two people are of equal height.

Using `functools.total_ordering` as described at this link, create a sorted list of people with heights.