Object-Oriented Programming

Objects

As opposed to procedural programming, which views data as passive and functions as active, OOP views things as objects, which are active.

Objects have fields (variables) which store data, and methods (functions) which do things.

Class Definitions

Class definitions are templates for creating objects, defining what methods and fields they have.

Class definitions allow you to create two objects from the same class definition, e.g. two bank accounts. These bank accounts will have the same structure, but have their own copies of any data, allowing (e.g.) each bank account to have an individual balance.

Methods

The syntax for calling a method *Deposit* of an object *a* is as follows:

```
a.Deposit(arguments)
```

All method definitions must have 'self' as their first argument (in Python it technically doesn't have to be called 'self', but it's good convention), which allows each method to access all of the object's data and functions.

When calling a method you only have to provide subsequent arguments—the 'self' argument is taken care of internally.

The `__init__` Method

When an object is created from a class, the `__init__` function of that class is called to create that object and define its fields.

This function must also have 'self' as its first argument, which feels a bit weird.

On the Ideology of Object-Oriented Programming

It would be difficult to really show how useful OOP is within small examples, but here's something:

- *A user should only access the methods of an object, rather than accessing the fields directly.*

(e.g. the user should not type `a._balance += 200` and should instead type `a.Deposit(200)`)

Handouts:

- Handout 21 - Object Oriented Programming