Introduction to Object-Oriented Programming in Python

A. G. Peddle and B. A. Wingate

Introduction

- Object-Oriented Programming (OOP) provides a way to organise your code
- Lets you write cleaner, more portable code which is both more correct and easier to maintain than procedural code
- This is not a lecture in algorithms like you've been doing so far
- This is your first foray into *software engineering*: the proper design of software

What is OOP?

- Until now, you've done procedural programming: program consists of procedures like functions which act in sequence on data
- OOP bundles data structures and procedures together into things called *objects*
- A object is then a logical partition of ideas, to isolate some data that makes sense to put together with some things it makes sense to do to it

Everything in Python is an object with a type.

(We have been using object oriented programming all term!)

OOP vs. Procedural Programming

- Think of it as a way to *organise* your code, with some very powerful ideas which will save you a lot of work
- You'll still do decisions, loops, functions, and everything else you've learned – this is just a more powerful way of assembling all the moving parts

For Example...

- In Python, *everything is an object*
- For example, consider a `list`:

```
>>> A = [1, 2, 3]
>>> A.append(4)
>>> print(A)
    A = [1, 2, 3, 4]
```

- Here `list` is a *class*: a blueprint for making objects
- `A` is an object: a particular instance of the class
- `A` has attributes: `int`s 1, 2, 3, and 4
- It has methods, e.g. `append`, which is a special function that acts only on this particular object

So How Do We Actually Do It?

- What are these really? They are constructed using the **`class`** keyword.
- Objects have **`types`**.
- Objects are comprised of *methods* (actions, verbs) and *attributes* (data, nouns)
- Methods of an object are basically functions (you already know these) but they are bound to a particular object and act natively on the attributes of that object

Building Our Own Classes

- It is important to understand that built-in types are all objects, and newly created objects are *types*
- It is important to be able to create our own new objects...but how?
- We use **`classes`** to create the blueprint for an object
- Class definitions keep to all of the familiar indent rules we're used to

The Point Class

- Let's think about how we would represent a point
- You can think of a point in 2-space as a pair of x- and y-coordinates
- It makes sense to apply certain operations only to points, like checking whether or not they're the origin

Point Class

```
class Point:
    def __init__(self, x_in, y_in):
        self.x = x_in
        self.y = y_in
```

- The `class` keyword tells Python that we want everything inside the indent to be a class
- `self` is a special variable that the object uses to refer to itself
- `__init__` is automatically included as a method for any object. Here, we're changing it to set the x and y coordinates of the point.
- We can do a bit more with this example...

Powerful Ideas

- OOP relies on some powerful ideas to save you work
- Two of these are *inheritance* and *composition*
- Inheritance is what you do when you create a new object as a subclass of an old one – it is an *is-a* relationship
- Composition is when you build a new object from other ones – it is a *has-a* relationship

Inheritance First

- We use inheritance whenever we want to make a new object which is a *special case* of an old one
- When we inherit, we automatically get everything in the superclass, and we are free to change, modify, or keep methods and attributes as we desire
- In practice, we often start with a standard Python class, like a `list` or a `dictionary` and add some new functionality
- We'll do a simpler example here

Composition

- Composition lets us create an object from other ones
- A simple example of this is how a `list` contains 'smaller' objects as attributes, even if these objects are just `int`s
- We can re-use our point class from earlier to make a triangle, which is *composed* of three points (vertices)

Encapsulation

- Encapsulation is another big idea in object-orientation
- It is the idea that the implementation of an object is hidden and known only to itself
- This means objects can easily be improved or corrected without breaking the rest of the code
- It also means that you don't have to know or care *how* an object is implemented, just how to use it (unless it's your own!)

Workshop!

- In this workshop we will create our own Person, Captain, and Ship classes and get to see how inheritance and composition work.
- By working with classes ourselves, it will make it easier to understand object oriented programming when we see it both in Python and other languages.