

Understanding Object Oriented Programming in Python

Exercises

version 2020-08

# Licence

This manual is © 2020, Steven Wingett & Simon Andrews.

This manual is distributed under the creative commons Attribution-Non-Commercial-Share Alike 2.0 licence. This means that you are free:

* to copy, distribute, display, and perform the work
* to make derivative works

Under the following conditions:

* Attribution. You must give the original author credit.
* Non-Commercial. You may not use this work for commercial purposes.
* Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a licence identical to this one.

Please note that:

* For any reuse or distribution, you must make clear to others the licence terms of this work.
* Any of these conditions can be waived if you get permission from the copyright holder.
* Nothing in this license impairs or restricts the author's moral rights.

Full details of this licence can be found at

<http://creativecommons.org/licenses/by-nc-sa/2.0/uk/legalcode>

# Exercise 1

## 1.1

1. Define a simple class called Individual.
2. Add an initialisation method which initialises the self.character\_name instance attribute.
3. Add an access method to the class that returns self.character\_name. Call this method get\_character\_name().
4. Create an instance of the character class and assign it to the variable individual1. This class instance should be assigned the character\_name ‘Buster’ on initialisation.
5. Create another instance, which should be assigned to the variable individual2. Set the name to ‘Tobias’.
6. Print the character name of individual1 and individual2 to the screen using the appropriate method.
7. Save this to a script called oop1.py.

## 1.2

Let’s build on our individual class a little more to make it more interesting.

1. On initialisation, set the instance attribute self.happy to True. This should be done by default (i.e. no parameter needs to be passed on instantiation in order to do this.)
2. Create a predicate method is\_happy to return the status of self.happy.
3. Create a modification method named switch\_mood()that changes self.happy from True to False (and vice versa).
4. Create a method called speak() that returns “Hello, I am [self.name]” or ‘Go away!’, depending on whether self.happy is set to True or False respectively.
5. Create individual3 with character name initialised to ‘Lucille’
6. Write some code to try out these methods/attributes of Buster and Tobias.
7. Save all this code to a script called oop2.py.

## 1.3

1. Add a class attribute called self.Counter that records the number of Individual instances created. This should be incremented by a class method called AddOne(). This way we can keep track of the total number of individuals. The current count total should be assigned to the instance variable self.id on instantiation. (Hint: we did this for the counting sheep example in the manual.)
2. Create \_\_str\_\_ and \_\_repr\_\_ methods to give a human-readable representation of each instance of individual. It should return: individual: [self.id self.character\_name]
3. Write additional code to verify the class is working as expected.
4. Save your updated code to a file named oop3.py.

# Exercise 2

We are now going to build on our class Individual some more and we are going to create a population of individuals using the data sheet Star\_Wars\_Data.txt (the data were extracted from the R dply package). In this list you will see the categories: Name, Height, Mass, Homeworld, Species.

1. Our individual class already has an attribute to store names. But let’s now create self.height, self.mass and self.homeworld attributes. These need to be set on instantiation of the individual object.
2. Create access methods to return the values for the attributes added in the previous step
3. In the species column we see there are droids (robots) and living species (e.g. organisms). These will have slightly different properties, so create sub-classes of Individual called Droid and Biological.
4. Add a species attribute to the Biological class. This needs to be specified on instantiation of a Biological. Also, add an access method to return the species value.
5. Write code to verify this is working as expected.
6. Save the script as star\_wars1.py.

# Exercise 3

1. Read in the data file Star\_Wars\_Data.txt and create either droid or biological class instances using the data in the sheet. These newly created objects should be stored in a list named population.
2. Write some code to check this has worked
3. Save the script as star\_wars2.py.

# Exercise 4

1. Override the speak method in the class droid to return “Beep Beep Beep”.
2. Write code to check this is working.
3. Save the script as star\_wars3.py.

# Exercise 5\*

1. Add a get\_bmi() method to the biological class, which returns the Body Mass Index of a biological. (Body Mass Index is a simple calculation using a person's height and weight. The formula is BMI = mass / height2 (with mass in kilograms and height in metres).
2. Iterate over the population list, identifying instances of the biological class (the function isinstance() may help you with this) and record their body mass index values.
3. Identify the biological with the highest body mass index
4. Save the script as star\_wars4.py.