Babraham Bioinformatics

Understanding Object Oriented Programming in Python

Exercises

version 2020-08

Licence

Exercise 1

1.1

i) Define a simple class called `Individual`.

ii) Add an initialisation method which initialises the `self.character_name` instance attribute.

iii) Add an access method to the class that returns `self.character_name`. Call this method `get_character_name()`.

iv) 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.

v) Create another instance, which should be assigned to the variable `individual2`. Set the name to 'Tobias'.

vi) Print the character name of `individual1` and `individual2` to the screen using the appropriate method.

vii) 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.

i) 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.)

ii) Create a predicate method `is_happy` to return the status of `self.happy`.

iii) Create a modification method named `switch_mood()` that changes `self.happy` from `True` to `False` (and vice versa).

iv) 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.

v) Create `individual3` with character name initialised to 'Lucille'

vi) Write some code to try out these methods/attributes of Buster and Tobias.

vii) Save all this code to a script called `oop2.py`.

1.3

i) 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.)

ii) 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]`

iii) Write additional code to verify the class is working as expected.

iv) 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.

i) 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.

ii) Create access methods to return the values for the attributes added in the previous step

iii) 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`.

iv) 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.

v) Write code to verify this is working as expected.

vi) Save the script as `star_wars1.py`.

Exercise 3

i) 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`.

ii) Write some code to check this has worked

iii) Save the script as `star_wars2.py`.

Exercise 4

i) Override the `speak` method in the class `droid` to return "Beep Beep Beep".

ii) Write code to check this is working.

iii) Save the script as `star_wars3.py.`

Exercise 5*

i) 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 / height^2 (with mass in kilograms and height in metres).

ii) 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.

iii) Identify the biological with the highest body mass index

iv) Save the script as `star_wars4.py`.