Python 11

Classes

The advantages of writing classes and writing functions are very similar.

When we write functions we group core Python functions and methods to create a unique collection statements that occur in a specific order.

These new functions make our code easier to read and to write, especially if you will use the function many times.

A conceptual difference between a function and a class is that a function usually does one thing, while a class will do many related things to help solve a problem.

What is a class really, what does it do? A class doesn't really do anything except for setting a list of rules for creating a new custom object. Every time you use the class you are creating an instance of a type of object.

You have been using classes to create objects

You have already been using classes to create objects. Here we are using the `open` function to create two instances of a file object. One instance holds information about a FASTA file while the other holds information about a GFF file.

```
fa_input = open("somedata.fa")
gff_input = open("somedata.gff")
```

attributes and methods

Classes create objects, these objects will have attributes and methods associated with them.

methods

Methods are functions which belong to objects of a particular class.

attributes

Attributes are variables that are associated with an object of a particular class.

Creating a Class

Defining a class is straightforward.

The first step is to decide what attributes and what methods it will have.

Create a DNARecord Class.

When we create a class, we are really setting up a series of rules that a DNARecord object must follow.

DNARecord Rules:

1. DNARecord must have a sequence [attribute]
2. DNARecord must have a name [attribute]
3. DNARecord must have an organism [attribute]
4. DNARecord will be able to calculate AT content [method]
5. DNARecord will be able to calculate the reverse complement [method]

Here is the first, but not final draft of our class. We will go through each section of this code below:

```
### START of CLASS DNARecord ###
class DNARecord(object):
  # define class attributes
  sequence = 'ACGTAGCTGACGATC'
  gene_name = 'ABC1'
  species_name = 'Drosophila melanogaster'

  # define methods
  def reverse_complement(self):
    replacement1 = self.sequence.replace('A', 't')
    replacement2 = replacement1.replace('T', 'a')
    replacement3 = replacement2.replace('C', 'g')
    replacement4 = replacement3.replace('G', 'c')
    reverse_comp = replacement4[::-1]
    return reverse_comp.upper()

  def get_AT(self):
    length = len(self.sequence)
    a_count = self.sequence.count('A')
    t_count = self.sequence.count('T')
    at_content = (a_count + t_count) / length
    return at_content
```

```
### END of CLASS DNARecord ###


### Outside class defintion ###
## Create a new DNARecord Object
dna_rec_obj = DNARecord()

## Use New DNARecord object
print('Created a record for ' + dna_rec_obj.gene_name + ' from ' +
dna_rec_obj.species_name)
print('AT is ' + str(dna_rec_obj.get_AT()))
print('complement is ' + dna_rec_obj.reverse_complement())
```

Now let's go through each section:

We start with the keyword `class`, followed by the name of our class `DNARecord` with the name of the base class in parentheses `object`.

```
class DNARecord(object):
```

Then we define class attributes. These are variables with data that belongs to the class, and therefore to any object that is created using this class

```
  # define class attributes
  sequence = 'ACGTAGCTGACGATC'
  gene_name = 'ABC1'
  species_name = 'Drosophila melanogaster'
```

Next, we define our class methods:

```
  # define methods
  def reverse_complement(self):
    replacement1 = self.sequence.replace('A', 't')
    replacement2 = replacement1.replace('T', 'a')
    replacement3 = replacement2.replace('C', 'g')
    replacement4 = replacement3.replace('G', 'c')
    reverse_comp = replacement4[::-1]
    return reverse_comp.upper()

  def get_AT(self):
    length = len(self.sequence)
```

```
    a_count = self.sequence.count('A')
    t_count = self.sequence.count('T')
    at_content = (a_count + t_count) / length
    return at_content
```

The methods are using an argument called `self`, i.e., `length = len(self.sequence)`. This is a special variable that you use inside a class. With it you can access all the data that is contained inside the object when it is created.

Use `self.attribute` format to retrieve the value of variables created within the class. Here we use `self.sequence` to retrieve the information stored in our attribute named `sequence`.

```
replacement1 = self.sequence.replace('A', 't')
```

Creating a DNARecord Object

The above class is a set of rules that need to be followed when creating a new DNARecord object.
Now let's create a new DNARecord object:

```
  dna_rec_obj = DNARecord()
```

`dna_rec_obj` is our new DNARecord object that was creating using the rules we put into place in the class definition.

Retrieving attribute values

Now that a new DNARecord object has been created, and assigned to the variable `dna_rec_obj`, we can access its attributes using the following format, `object.attribute_name`

To get the gene name of the object we created, we simply write `dna_rec_obj.gene_name`.

This is possible because within our class definition we create a `gene_name` variable.

Let's try it:

```
>>> dna_rec_obj.gene_name
'ABC1'
>>> dna_rec_obj.sequence
'ACGTAGCTGACGATC'
```

Using class methods

To call a method associated with our new object, we use a similar format `object.method_name`.

So to call the `get_AT()` method, we would use `dna_rec_obj.get_AT()`. This should look familiar, you have done used class methods over and over again: `some_string.count('A')`

Let's try it with our `dna_rec_obj`:

```
>>> dna_rec_obj.sequence
'ACGTAGCTGACGATC'
>>> dna_rec_obj.get_AT()
0.4666666666666667
```

Now let's use the `reverse_complement()` method

```
>>> dna_rec_obj.sequence
'ACGTAGCTGACGATC'
>>> dna_rec_obj.reverse_complement()
GATCGTCAGCTACGT
```

Wow!! Getting the reverse complement in one line is pretty nice!

Getting data into a new instance of our class

Great!!!

We can now create a DNARecord object and retrieve the object attributes and use the cool methods we created.

But..... It always contains the same gene_name, sequence, and species information 😟

Let's make our class more generic, or in other words, make it so that a user can provide a new gene name, gene sequence, and source organism everytime a DNARecord object is created.

__init__

To do this we need to add an `__init__` function to our Object Rules, or Class.

The `init` function will automatically get called when you create an object.

It contains specific instructions for creating a new DNARecord Object.

It specifies how many pieces of data we want to collect from the creator of a DNARecord object to use within a DNARecord object.

Below our **__init__** instructions indicate that we want to create object attributes called `sequence`, `gene_name`, and `species_name` and to set them with the values provided as arguments when the object was created.

Here is our new class definition and new object creation when using the **__init__** function:

```
#!/usr/bin/env python3
class DNARecord(object):

  # define class attributes
  def __init__(self, sequence, gene_name, species_name): ## note that '__init__' is
wrapped with two underscores
    #sequence = 'ACGTAGCTGACGATC'
    #gene_name = 'ABC1'
    #species_name = 'Drosophila melanogaster'
    self.sequence = sequence
    self.gene_name = gene_name
    self.species_name = species_name

  # define methods
  def reverse_complement(self):
    replacement1 = self.sequence.replace('A', 't')
    replacement2 = replacement1.replace('T', 'a')
    replacement3 = replacement2.replace('C', 'g')
    replacement4 = replacement3.replace('G', 'c')
    reverse_comp = replacement4[::-1]
    return reverse_comp.upper()

  def get_AT(self):
    length = len(self.sequence)
    a_count = self.sequence.count('A')
    t_count = self.sequence.count('T')
    at_content = (a_count + t_count) / length
    return at_content

## Create new DNARecord Objects with user defined data
dna_rec_obj_1 = DNARecord('ACTGATCGTTACGTACGAGT', 'ABC1', 'Drosophila melanogaster')
dna_rec_obj_2 = DNARecord('ATATATTATTATATTATA', 'COX1', 'Homo sapiens')

for d in [ dna_rec_obj_1, dna_rec_obj_2 ]:
  print('name:' , d.gene_name , ' ' , 'seq:' , d.sequence)
```

Output:

```
$ python3 dnaRecord_init.py
name: ABC1    seq: ACTGATCGTTACGTACGAGT
name: COX1    seq: ATATATTATTATATTATA
```

Now you can create as many DNASequence Objects as you like, each can contain information about a different sequence.

Link to Python 11 Problem Set
