1_classes

November 27, 2023

1 Classes and Objects

[1]:
```python
# include the required modules

import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_format = 'retina'

plt.rcParams.update({'font.size': 12,
                     'axes.titlesize': 18,
                     'axes.labelsize': 16,
                     'axes.labelpad': 14,
                     'lines.linewidth': 1,
                     'lines.markersize': 10,
                     'xtick.labelsize' : 16,
                     'ytick.labelsize' : 16,
                     'xtick.top' : True,
                     'xtick.direction' : 'in',
                     'ytick.right' : True,
                     'ytick.direction' : 'in',})
```

A very useful programming concept of modern programming is object oriented programming. In all the programs we wrote till now, we have designed our program around functions i.e. blocks of statements which manipulate data. This is called the procedure-oriented way of programming.

There is another way of organizing your program which is to combine data and functionality and wrap it inside something called an object. This is called the object oriented programming paradigm, which will be useful especially for larger programs.

Classes and objects are the two main aspects of object oriented programming. * A class creates a new type where objects are instances of the class.

- Objects can store data using ordinary variables that belong to the object. Variables that belong to an object or class are referred to as properties. Objects can also have functionality coming from functions that belong to a class and are called methods.
- Properties are of two types - they can belong to each instance/object of the class or they can belong to the class itself. They are called instance variables and class variables respectively.

1.1 Definition of Classes

We create a new class using the `class` statement and the name of the class and a colon `:`. This is followed by an indented block of statements which form the body of the class.

```
class Class_name:
    statements ###
```

In the following example, we have an empty block which is indicated using the pass statement.

```
[2]: class Colloid:
         pass   # An empty block
```

You can then create a new object of the class Colloid by The print command will just inform you about the memory address at which the object is stored.

```
[4]: p = Colloid()
     print(p)
```

```
<__main__.Colloid object at 0x7f332704bbb0>
```

1.2 Class Methods

Methods are functions that belong to a class.

Note: The self parameter

Class methods have an extra first parameter at beginning of the parameter list, but you do not give a value for this parameter when you call the method, Python will provide it. This particular variable refers to the object itself, and by convention, it is given the name **`self`**. Thus, even a method without arguments has a single argument:

```
class Colloid:
    def type(self):
        print('I am a plastic colloid')
```

```
[5]: class Colloid:
         def type(self):
             print('I am a plastic colloid')

     p = Colloid()
     p.type()

     b=Colloid()
     b.type()
```

```
I am a plastic colloid
I am a plastic colloid
```

1.2.1 The `__init__` method

The `__init__` method is run as soon as an object of a class is instantiated (i.e. created). The method is useful to do any initialization (i.e. passing initial values to your object) you want to do with your object. Notice the double underscores both at the beginning and at the end of the name.

[20]:
```python
class Colloid:

    def __init__(self, R):
        self.R = R

    def get_size(self):
        return(self.R)
```

[21]:
```python
p=Colloid(5)
b=Colloid(2)
```

[22]:
```python
print('Colloid radius is {} µm '.format(str(b.get_size())))
```

```
Colloid radius is 2 µm
```

Besides the `__init__`method, which is commonly called `constructor`, there is also the `__del__` method, which is called b´when an object is deleted. We will use that further down.

1.2.2 The `__str__` method

The `__str__` method that is invoked when a simple string representation of the class is needed, as for example when printed.

[23]:
```python
class Colloid:

    def __init__(self, R):
        self.R = R

    def get_size(self):
        return(self.R)

    def __str__(self):
        return('I am a plastic colloid of radius %.1f' % (self.R))
```

[25]:
```python
p=Colloid(5)
```

[26]:
```python
print(p)
```

```
I am a plastic colloid of radius 5.0
```

1.3 Class and object variables

The data part, i.e. properties, are ordinary variables that are bound to the namespaces of the classes and objects. This means that these names are valid within the context of these classes and objects only.

There are two types of fields - `class variables` and `object variables` which are classified depending on whether the class or the object owns the variables respectively.

- **Class variables** are shared - they can be accessed by all instances of that class. There is only one copy of the class variable and when any one object makes a change to a class variable, that change will be seen by all the other instances.
- **Object variables** are owned by each individual object/instance of the class. In this case, each object has its own copy of the field i.e. they are not shared and are not related in any way to the field by the same name in a different instance.

Here is an example:

[27]:
```
class Colloid:

    number = 0 # definition of a class variable

    def __init__(self, R):
        self.R = R
        Colloid.number=Colloid.number+1


    def __del__(self):
        Colloid.number=Colloid.number-1
```

[28]:
```
# create a particle with radius 4
p1=Colloid(3)

# create a second particle with radius 10
p2=Colloid(12)
```

[29]:
```
# print the radius of both particles
print(p1.R,p2.R)
```

```
3 12
```

[30]:
```
# print the particle number
print(p1.number, p2.number)
```

```
2 2
```

[31]:
```
# delete one particle object and print the
del(p2)
```

[32]:
```
print(p1.number)
```

```
1
```