Classes and Objects

Customizing QGIS with Python

Ujaval Gandhi
ujaval@spatialthoughts.com

Classes and Objects

Why Classes?

- Makes the code ‘modular’. Classes can be re-used and enhanced by other classes.
- Classes allow us to avoid code duplication
- Hides the implementation detail from the user of the class

What is a Class?

- A **class** can be thought of as a template.
- It contains certain functions and properties.
- Example
  - You have a class called `Car()`
  - A blueprint to build a car with given specifications
  - A class `Car()` which takes 2 parameters `color` and `type`
  - The class has `Car()` functions that know operate the car: `start()`, `drive()`, `stop()`

Objects

- To use a class in a program, you must construct an object from a class
  - This is called creating an `instance` of the class
- To create an instance of a class, you initialize it with some parameters
- A new **object** is constructed from the template using the supplied parameter.
  - A car object with parameters: color=blue and type=automatic
  - `my_car = Car('blue', 'automatic')`
- You can call class functions using that object
  - `my_car.start()` to start the car

Understanding Terminology

Let's define a Python class

```
class Car:

  def __init__(self, color, type):
      self.color = color
      self.type = type
      self.started = False
      self.stopped = False

  def start(self):
      print('Car Started')
      self.started = True
      self.stopped = False
```

Instance

- Instance is an object of a class
- You can create an instance of a class by calling the constructor

Creating an instance

```
class Car:

    def __init__(self, color, type):
        self.color = color
        self.type = type
        self.started = False
        self.stopped = False

    def start(self):
        print('Car Started')
        self.started = True
        self.stopped = False
```

```
my_car1 = Car('blue', 'automatic')

my_car2 = Car('red', 'manual')
```

Method

- Classes have functions
- When you call a function from an object it is called a `method`
- Methods are passed on the reference to the current object using the *self* keyword
    - If you see a function with the first parameter as *self*, it needs to be called on an object

Using a Method

```
class Car:

    def __init__(self, color, type):
        self.color = color
        self.type = type
        self.started = False
        self.stopped = False

    def start(self):
        print('Car Started')
        self.started = True
        self.stopped = False


my_car = Car('blue', 'automatic')

my_car.start()
```

Attributes

- Class can have variables
  - Class variables are called attributes

Instance Attributes

- Instance Attributes
  - They are associated with a particular object
  - Every object can have a different value
  - Defined inside the __init__() constructor

```
class Car:

    def __init__(self, color, type):
        self.color = color
        self.type = type
        self.started = False
        self.stopped = False
```

```
my_new_car = Car('blue', 'automatic')
my_old_car = Car('red', 'manual')
```

```
print(my_new_car.color)
print(my_old_car.color)
```

Class Attributes

- Class Attributes
  - They are associated with a class
  - Every objects will have the same value
  - Defined outside of the __init__() constructor
  - Can be accessed from a class

```
class Car:

    model = 'Civic'

    def __init__(self, color, type):
        self.color = color
        self.type = type
        self.started = False
        self.stopped = False


my_new_car = Car('blue', 'automatic')
my_old_car = Car('red', 'manual')

print(my_new_car.model)
print(my_old_car.model)

print(Car.model)
```

Inheritance

- Classes can be derived from another class.
- The derived class ‘inherits’ all features of the base class.
- This allows you to build a hierarchy of classes where classes get more and more specialized without having to implement all the features.

```
class Car:

    def __init__(self, color, type):
        self.color = color
        self.type = type
        self.started = False
        self.stopped = False

    def start(self):
        print('Car Started')
        self.started = True
        self.stopped = False
```

```
class Sedan(Car):

    def __init__(self, color, type, seats):
        super().__init__(color, type)
        self.seats = seats
```

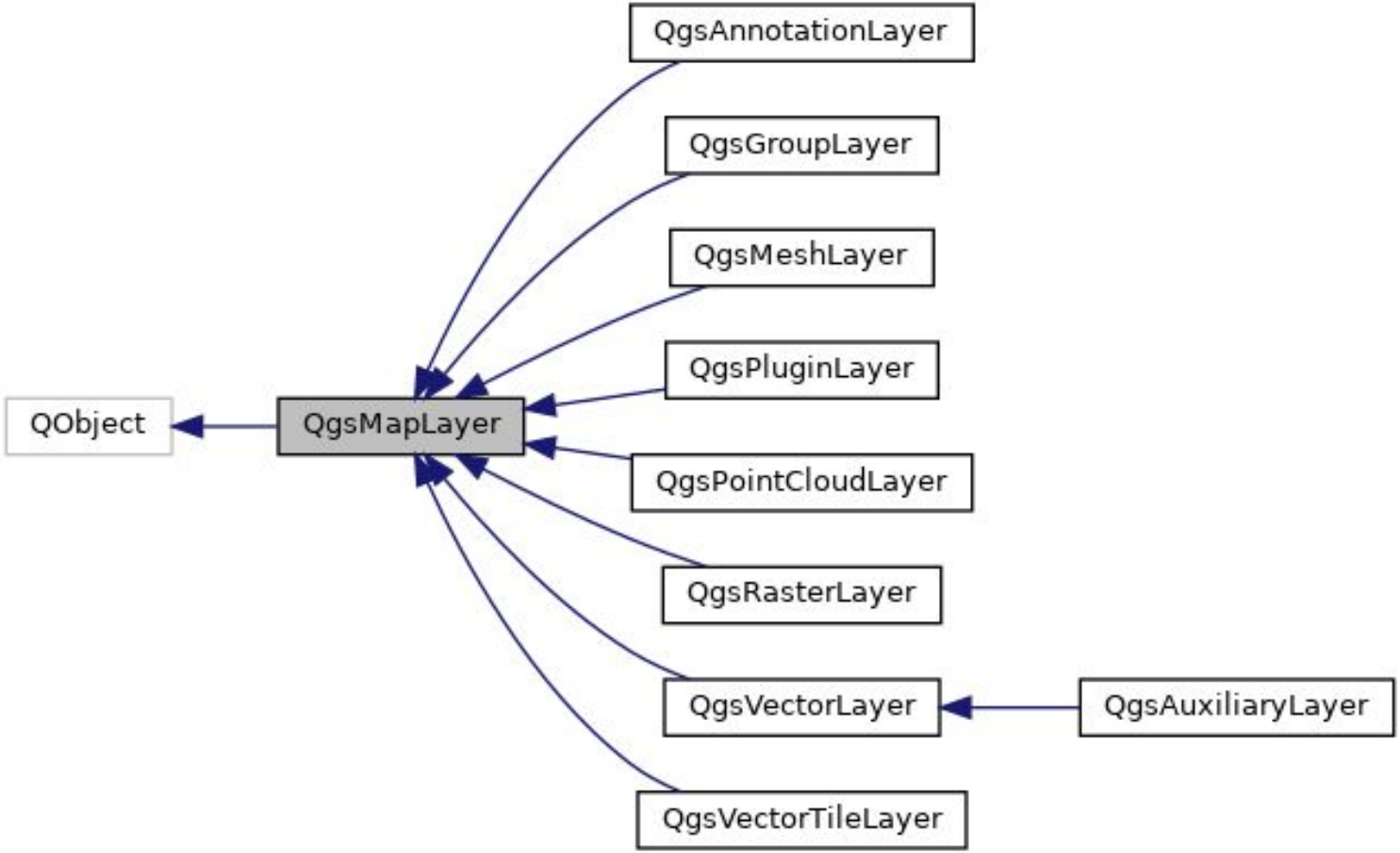
```
class ElectricSedan(Sedan):

    def __init__(self, color, type, seats,
                 range_km):
        super().__init__(color, type, seats)
        self.range_km = range_km
```

```
my_car = Sedan('blue', 'automatic', 5)
print(my_car.color)
print(my_car.seats)
my_car.start()
```

```
my_future_car = ElectricSedan(
            'red', 'automatic', 5, 500)
print(my_future_car.color)
print(my_future_car.seats)
print(my_future_car.range_km)
my_future_car.start()
```

Inheritance in PyQGIS

- All PyQGIS classes are derived from the Base Class called QObject()
- Example
  - QgsMapLayer() is the Base class for all map layer types.
    - QgsRasterLayer() is derived from QgsMapLayer()
    - QgsPointCloudLayer() is derived from QgsMapLayer()
    - QgsVectorLayer() is derived from QgsMapLayer()
      - QgsAuxiliaryLayer() is derived from QgsVectorLayer()


QgsAnnotationLayer
QgsGroupLayer
QgsMeshLayer
QgsPluginLayer
QObject
QgsMapLayer
QgsPointCloudLayer
QgsRasterLayer
QgsVectorLayer
QgsAuxiliaryLayer
QgsVectorTileLayer

Let's use a PyQGIS class

QgsDistanceArea

A general purpose distance and area calculator, capable of performing ellipsoid based calculations.

GUI Programming Concepts

Qt

- Qt is a free and open-source widget toolkit for creating graphical user interfaces as well as cross-platform applications.
- QGIS is built using the Qt platform. Both QT and QGIS itself have well-documented APIs that should be used when writing Python code to be run within QGIS.

PyQt

- Qt is written in C++
- PyQt is the Python API to Qt
- PyQt provides classes and functions to interact with Qt widgets.

PyQGIS

- QGIS is written using C++
- QGIS provides a Python API , commonly known as PyQGIS.
- PyQGIS integrates with PyQt and allows you to use QGIS and Qt classes.
- When we use PyQt or PyQGIS classes, it is executing the code in the C++ classes via the python bindings.

Let's use a PyQt class

QMessageBox

- The QMessageBox is a PyQt class for creating a dialog with buttons.
- We will create a confirmation dialog with ‘OK’ and ‘Cancel’ buttons.

Inheritance

- QObject is the most basic class in Qt.
- All Qt widgets and QGIS classes inherit from QObject.
- The most basic widget is the QWidget. QWidget contains most properties that are used to describe a window, or a widget, like position and size, mouse cursor, tooltips, etc.
- The QDialog class is the base class of dialog windows.
- QMessageBox is a specialized QDialog.