Infrastructure as Code- Walkthrough

Python Object Oriented Coding

Contents

Last Updated: 6 October 2022

A **walkthrough** is intended to bring you through a technical exercise. A walkthrough shows you how I completed a task in a particular context, on specific systems, at a point in time. The document is a good basic guide, but you should always confirm that the information is current and if a more recent best practice exists, you should make yourself aware of it.

Introduction

This walkthrough is provided to demonstrate how to get started with Object Oriented (OO) coding in Python. So far, we have concentrated on functional programming. The fundamental units of coding our functions, and we create a main program to call these functions in the correct sequence. In scripting, this is a perfect paradigm for most of the code we're going to write.

But there is an entirely different paradigm which organizes our code around self-contained objects, which encapsulate data, properties or attributes, and functions or methods. Where we create large projects, which are complex, and will be maintained by many people, this is a more scalable way to organize. On a large project, I can have groups working separately on different objects, giving me better scalability and code reuse.

In this approach, a class is a template used to create objects. We instantiate the class to create instances of an individual object. Everything related to this class is contained inside an object, and only selected attributes and methods are exposed to the outside. In electronics terms, an object is a black box. It provides an abstraction to the complexity which it contains.

Imagine I am writing an instrument system for an airplane, and I create an instrument class. Instruments have certain characteristics or properties (Name, Type, etc) that all instruments have. If I want to verify communications, I could have a method to check a cyclical redundancy check (CRC).

By making the barometer class a sub-class of the instrument class, the barometer class can inherit all the properties and methods of the instrument class. All the properties and methods particular to students can be encapsulated in the student class. My airspeed class could also inherit the properties and methods of all instruments and have additional properties and methods specific to the pitot sensor.

Methods can be extended and overridden; this is polymorphism. Objects can share behaviors and functions can have different meanings and usages depending on the context.

In our coding so far, you have extensively used this functionality! Every time you use the format **object.method** you are using this. For example

```
my_string = "jor"
my_proper_string = my_string.capitalize()
print(my_proper_string)
```

Appending, sorting are all examples of methods which change the contents of an object.

Deliverables

You will be required to keep a list of commands with notes as to why you are using them as well as results of any testing you have done. You may be asked to present these notes as evidence of work completed or as part of a lab book. Check with your lecturer!

Prerequisites

1. I will carry out my exercises using a Windows 11 desktop. You can carry out this work on almost any modern platform.
2. You should have already installed Python as per my notes.
3. You should have already installed VSC as per my notes.
4. You have created a directory to keep example files in and you are ready to code.
    a. This could be on your OneDrive, in these examples, I am using **OneDrive\Python\Exercises_08**

Anatomy of a simple class

At the start, the anatomy and syntax of a class is not intuitive.

```
"""
Simple Class by JOR, by convention, use camel case to name classes
"""

# Create a class
class JORzClass():

    # Constructor, called whenever an instance of the class is created.
    def __init__(self, my_greeting):
        print("Running constructor for JORzClass")
        # Create attributes and set initial values
        self.message = my_greeting

    def my_method(self):
        print(self.message)

my_class1 = JORzClass("Morning JOR!")
my_class1.my_method()
print(type(my_class1))
```

Firstly, I used the **class** keyword to define the class JORzClass. Note the use of camel case as the class name and the use of snake case for functions. A function within an object is called a method.

The **self** keyword causes a lot of confusion! It is used to represent an instance (object) of the given class. Remember, we can create many objects from a class, and each will have its own unique attributes. We can use *self.something* to refer to attributes within an instance of an object.

The code block **def __init__(self):** is the constructor and is called whenever an instance of this object is created, and the first parameter is **self**. We can also pass parameters to the object at this point, I am using **my_greeting**. The print statement will execute, letting us know that we are running the constructor. For use with the class, I make the attribute **self.message** equal to the parameter **my_greeting** to use it in the object.

When I create a function or method, I pass the **self** attribute first, letting Python know that it is a method of the class.

We saw earlier in this module how to use the **type** method. What do you expect the last line in my script to do?

Exercise

Write and test a class like the one above and save it as **oo1.py**

In addition to **MyClass1**, write some code to create and exercise an instance of your class called **MyClass2**.

Building a class from scratch

I've been coding in Python for many years, and when I start building an OO project, I use a template. Let us start building a template you can use for future projects. Save this exercise as **oo2.py**.

The code below is the simplest I can create!

```
"""
Class template by JOR

Revision History
06OCT22: Alpha
"""

class MyTemplate():
    pass

# Instantiate the class
my_object = MyTemplate()
# Check the object and type
print(type(my_object))
```

The **pass** keyword just means "do nothing!".

Next, we will add the constructor, a dunder method called **__init__**, passing the **self** keyword to connect this method to the instance of the class. I replace the keyword pass with

```
# Constructor, called whenever an instance of the class is created.
    def __init__(self) -> None:
        print("Constructor ran")
```

Objects have attributes, we normally pass these to an object as arguments when we instantiate. I edit the constructor as shown to add two attributes. I pass two argument values to the object and I set them to the object attributes **self.attr1** and **self.attr2**.

```
# Constructor, called whenever an instance of the class is created.
    def __init__(self, attribute1: str, attribute2: bool) -> None:
        print("Constructor ran")
        # Take in an argument and assign it to a meaningful attribute name
        self.attr1 = attribute1
        self.attr2 = attribute2
```

I must also edit the line where I instantiate the class, otherwise I am missing the two positional arguments. This will create a type error.

```
# Instantiate the class
my_object = MyTemplate("John", True)
```

In VSC, if I type **my_object.** I am now offered these attributes.

Class object attributes

Sometimes we need values which will be the same for all instances of the class. Imagine I'm writing some code for a geographical information system (GIS). The earth is abstracted as an ellipsoid, with a defined semi major axis, and semi minor axis. These will not change.

I can define a class object attribute in the same way that we used to define constants in other programming languages. I add the following code before the constructor.

```
# Define a class object attribute, it will be the same for any instance of the class
semi_major_axis = 6378137
semi_minor_axis = 6356752
```

We don't need to use the **self** keyword, as these values will be the same for every instance of the class. The following code at the end tests that this works.

```
# Instantiate the class
my_object = MyTemplate("John", True)
# Check the object
print(my_object.semi_major_axis, my_object.semi_minor_axis)
```

To generalize the template, change

- **semi_major_axis** to **class_object_attribute1**
- **semi_minor_axis** to **class_object_attribute2**

Then change the print statement to accommodate these changes and retest.

Methods

Methods are functions within the class which will act on attributes of the class. The simplest implementation just looks like a function. I add the method under the __init code block at the same indentation as the __init__ method.

```
    def my_method1(self):
        if self.attr2:
            print(f"Good morning {self.attr1}")
        else:
            print(f"No greeting {self.attr1}")
```

I can then call that method by the following line at the end of my code.

```
my_object.my_method1()
```

I use the object attributes to make a decision and to give feedback.

I could also pass an argument to a method at runtime.

```
    def my_method2(self, my_name:str):
        if self.attr2:
            print(f"Good morning {my_name}")
        else:
            print(f"No greeting {my_name}")
```

I can then call this method by the following line at the end of my code.

```
my_object.my_method2("Giovanni")
```

As demonstrated with functions in previous notes, I can set a default value for an argument.

Exercise

Create and document a template for your own use, such that whenever you need a simple class, you can just copy the source code. Call it **class_template.py**.

Inheritance

If a project is complex enough to justify the overhead of writing it OO, it's probably complex enough to justify a little advanced planning and the use of inheritance. Classic books on OO design will specify Unified Modeling Language (UML) for this kind of work. In practice, I'm just not seeing that used in industry.

In this section, I'm going to build up a basic class model on which I can build network automation code and utilities. Don't worry too much about the details, I do not assume you know anything about network hardware. This is just a slightly contrived context I'm using to build code which might actually be useful in another module!

I am going to start off with the assumption but there are characteristics in common with all network devices.

```
# In any complex application, create a base class which will never be instantiated.
class Device():

    # Define a class object attribute, it will be the same for any instance of the class
    pi = 3.142

    # Constructor, called whenever an instance of the class is created.
    def __init__(self) -> None:
        print("Running constructor for base class")
        # Create attributes and set initial values
        self.debug = False

    def run(self):
        raise NotImplementedError("This is an abstract class, do not instantiate")

    def calculate_crc(self, frame:str)->int:
        print("Checking CRC from base")
        # Put real code in here, this is a dummy value for initial setup
        crc = 123456789
        return crc
```

The first new idea is that of an abstract class. I am never going to instantiate an object called **device**, I'm going to want objects like firewalls, routers, and switches. I'm only using the abstract class to hold the attributes and methods common to all devices. For example, I will always want a CRC check, I have created a placeholder using a print statement (I might just have used **pass**).

It all works, but there is no live code.

If someone executes the **run** method, it will raise an error.

```
my_devices = Device()
my_devices.calculate_crc("dummy")
my_devices.run()
```

What I will actually need are objects which abstract real network devices.

```
# Create child classes which can access the methods and properties of the base class
class Firewall(Device):

    # Constructor, called whenever an instance of the class is created.
    # Use parameter1 as a tag to identify the object
    def __init__(self, parameter1) -> None:
        # Call back to the parent class
        Device.__init__(self)
        print(f"Running constructor for {parameter1}")
        # Create attributes and set initial values
        self.parameter1 = parameter1
        self.test_message = ""

    def configure_firewall(self):
        print("Configuring Firewall")
```

I create the child class using **class *ChildClassName*(*ParentClassname*)**:

The command **class Firewall(Device):** creates a derived class inheriting from the parent class, **Device**. I create an instance of Firewall by using **def __init__(self, parameter1) -> None:**

I also need to initialize the base class using **Device.__init__(self)** when I create the Firewall class. You should understand all the other code.

I can access attributes and methods from either the parent or child class from an object based on the child class.

```
hostname = "firewall3"
my_firewall = Firewall(hostname)
my_firewall.calculate_crc("dummy")
my_firewall.configure_firewall()
```

It can be mind-blowingly confusing, but I can override methods in the base class from the derived class. Add the following code to the Firewall class and retest.

```
def calculate_crc(self, frame:str)->int:
        print("Checking CRC from child")
        # Put real code in here, this is a dummy value for initial setup
        crc = 123456789
        return crc
```

The method **calculate_crc** from the base is ignored and overridden.

Bringing it all together

My final file called devices.py looks like this.

```
"""
Parent class Template by JOR
"""
# In any complex application, create a base class which will never be instantiated.
class Device():

    # Define a class object attribute, it will be the same for any instance of the class
    pi = 3.142

    # Constructor, called whenever an instance of the class is created.
    def __init__(self) -> None:
        print("Running constructor for base class")
        # Create attributes and set initial values
        self.debug = False

    def run(self):
        raise NotImplementedError("This is an abstract class, do not instantiate")

    def calculate_crc(self, frame:str)->int:
        print("Checking CRC from base")
        # Put real code in here, this is a dummy value for initial setup
        crc = 123456789
        return crc


"""
Child class Template by JOR
"""
# Create child class which can access the methods and properties of the base class
class Firewall(Device):

    # Constructor, called whenever an instance of the class is created.
    # Use parameter1 as a tag to identify the object
    def __init__(self, parameter1) -> None:
        # Call back to the parent class
        Device.__init__(self)
        print(f"Running constructor for {parameter1}")
        # Create attributes and set initial values
        self.parameter1 = parameter1
        self.test_message = ""

    def configure_firewall(self):
        print("Configuring Firewall")

    def calculate_crc(self, frame:str)->int:
        print("Checking CRC from child")
        # Put real code in here, this is a dummy value for initial setup
        crc = 123456789
        return crc
```

I create a file called **main.py** in the same directory to use my code.

```
from devices import Firewall

# Create a firewall instance
firewall27 = Firewall("firewall27")
# Configure it
firewall27.configure_firewall()

# Create a firewall instance
firewall28 = Firewall("firewall28")
# Verify a CRC
firewall28.calculate_crc("dummy data")
```

All the complexity of the Device and Firewall classes are encapsulated and easily accessed.

Exercise

Add classes for other devices, switches, firewall and load balancers. Test the code by calling it from main.

Finally

This probably seemed madly complicated.

I have a confession to make.

When Microsoft introduced object-oriented programming with Visual C++ c. 1993, I gave up programming for years!

Object oriented coding can seem horrendous at first. It might be easier if we didn't teach functional programming before teaching OO. However, from years of writing production scripts and code, I do believe you need both.

If I need to do a task and it's simple, I write a 10-line script.

If I need to reuse old code, or I need something that's modular, I'll build it as a package using the functional programming paradigm.

But if I'm working on a very large project, in particular if I'm working as part of a team, I will always use object-oriented principles.

The bad news is that this was very much an introduction to OO programming for scripting purposes. Once you get into professional coding, you will realize that we've only scratched the surface here.