COE 302
Object Oriented Programming with Python

Akinola Oluwole

soakinola@abuad.edu.ng

Content

- Why Python for OOP?
- Basics of Python
- …

Why Python for OOP

- Python is open source scripting language developed by Guido van Rossum in the early 1990s
- Everything in Python is an Object. Python is an OOP language. You can write programs in Python either in a procedural way or in an object-oriented way.
- Python doesn't support **strong encapsulation**, which is only one of many features associated with the term "object-oriented".
- Python is similar to MATLAB, OCTAVE or even R.
- Huge library of third party module and extensive support library.

Basics of Python

- **Installation**
- **Variables**
- **Using the Python Interpreter**
- **Python Comments**
- **Keywords in Python**
- Booleans in Python
- **Operators**
- Regular Expressions in Python
- Exception Handling in Python
- Strings in Python
- Lists in Python
- Dictionaries in Python
- **Loops**
- **Functions**
- **Conditional statements**
- **Modules (Math module)**

Basics of Python: Installation and Interpreter

- **Installation**

Download python from https://www.python.org/downloads/

From command prompt,

```
pip install jupyter # will install jupyter notebook
c\> jupyter notebook # Default browser load the Ipython notebook environment
```

Or

Install VS Code from https://www.code.visualstudio.com

- **Python Interpreter**

The Python interpreter is usually installed in **/usr/local/bin/python** on machines where it is available.

Basics of Python: Variables and Comments

- **Variables**
- In Python, variables are a storage placeholder for texts and numbers.

```
primary = 100

name = “Akinola Oluwole”
```

- **Python Comments**
- When working with any programming language, you include comments in the code to notate your work.

```
# This is a comment

’’’This is a multiline

comment’’’
```

Basics of Python: Keywords and Operators

- **Keywords**

Keywords in Python are reserved words that cannot be used as ordinary identifiers. They must be spelled exactly as they are written.

```
return, lambda, True, False, None
```

```
help("keywords") # list all keywords in python
```

- **Operators**
- Python includes the +, -, *, /, % (modulus), and ** (exponentiation) operators.

```
a += b, a = a + b #Add
```

```
a *= b, a = a*b #product
```

```
a **= b, a = a**b #power/exponetial
```

```
a %= b, a = a%b
```

Basics of Python: Conditions and Loops

- **Conditions**
- Very often we want to check the conditions and change the behaviour of the program. Conditions tests if a something is True or False, and it uses Boolean values.

```
if condition:                 elif condition:                       else:
      block                         block                                 block
```

- **Loops**

A function is something you can call (possibly with some parameters, the things you put in the parentheses), which performs an action and returns a value.

```
while condition:              else:                            for target in sequence:
      block                         block                            block
```

Basics of Python: Functions and Modules

- **Functions**

A function is something you can call (possibly with some parameters, the things you put in the parentheses), which performs an action and returns a value.

```
def func():
        print("hello")
```

Modules

- Python modules makes the programming a lot easier. It’s basically a file that consist of already written code
- **`import <module>`** `# import math`
- **`from <module> import <thing>`** `# from math import cos`
- **`import <module> as <alias>`** `# import math as m`
- **`from <module> import <thing> as <alias>`** `# from math import cos as c`

Basics of Python: Example 1

$$A = P(1 + (r/100))^n$$

Calculating the interest on a bank deposit. where P is the initial deposit (the principal), r is the yearly interest rate given in percent, n is the number of years, and A is the final amount. The task is now to write a program that computes A for given values of *P=100*, *r=5.0* and *n=7*.

```
primary = 100
r = 5.0
n = 7
amount = primary * (1+r/100)**n
print(amount)
```

Basics of Python: Example 2

Print Hello, name Statement

```
Print("Hello, My name is Oluwole!“)
```

or

```
hello = "Hello, My name is Oluwole!"
print(hello)
```

Basics of PYTHON: Example 3

$$f(x) = \frac{1}{\sqrt{2\pi}s}\exp\left[-\frac{1}{2}\left(\frac{x-m}{s}\right)^2\right]$$

Evaluating the bell-shaped Gaussian function
for $m = 0, s = \ 2,$ and $x = \ 1.,$

```
from math import sqrt, pi, exp
m = 0
s = 2
x = 1.0
f = 1/(sqrt(2*pi)*s) * exp(-0.5*((x-m)/s)**2)
print(f)
```

Python Object Oriented Programming (OOP)

- OOP is an approach to software development in which the structure of the software is based on objects interacting with each other to accomplish a task.
- This interaction takes the form of messages passing back and forth between the objects.
- In response to a message, an object can perform an action.

The History of OOP

- OOP concepts started surfacing in the mid-1960s with a programming language called Simula and in the 1970s with Smalltalk.
- In the mid-1980s OOP languages such as C++ became popular with mainstream computer programmers.
- OOP continued to grow in popularity in the 1990s, with the advent of Java and in 2002 with Microsoft C# (pronounced C-sharp)
- Today OOP languages continue to flourish and are a mainstay of modern programming.

Why Use OOP?

- OOP concepts started surfacing in the mid-1960s with a programming language called Simula and in the 1970s with Smalltalk.
- In the mid-1980s OOP languages such as C++ became popular with mainstream computer programmers.
- OOP continued to grow in popularity in the 1990s, with the advent of Java and in 2002 with Microsoft C# (pronounced C-sharp)
- Today OOP languages continue to flourish and are a mainstay of modern programming.

Challenges with Procedural Programming

- Existing functionality was hard to alter without adversely affecting all of the system’s functionality.
- New programs were essentially built from scratch. Consequently, there was little return on the investment of previous efforts.
- It was hard to translate business models into programming models.
- Structural programming worked well in isolation but did not integrate well with other systems.
- As a result, many business software developers turned to object-oriented methods and programming languages.

Benefits with OOP Programming

- a more intuitive transition from business-analysis models to software-implementation models
- the ability to maintain and implement changes in the programs more efficiently and rapidly
- the ability to create software systems more effectively using a team process, allowing specialists to work on parts of the system
- the ability to reuse code components in other programs and purchase components written by third-party developers to increase the functionality of existing programs with little effort
- better integration with loosely coupled distributed-computing systems
- the ability to create a more intuitive graphical-user interface for the users

OOP Terminologies: Class

A blueprint to create objects. It defines the **data (attributes)** and **functionality (methods)** of the objects. You can access both attributes and methods via the dot notation.

Example

```
class Student:
    # class attribute
    take_lecture =True
```

OOP Terminologies: Object

(Object=instance): Often, an object corresponds to a thing in the real world. An example is the object **Wole** that is created according to the class definition **Lecturer**. An object consists of an arbitrary number of attributes and methods, encapsulated within a single unit.

Example

```
# Object instance
Obj_stud = Student()
Obj_stud.name = “Wole”
```

OOP Terminologies: Instantiation

The process of creating an object of a class. This is done with the constructor method __init__(self, …). The _init_ method is run as soon as an object of a class is created. This method is useful for passing initial value to your objects. the self parameter is included so that our initializer (__init__) has a reference to the new object being instantiated.

Example

```
# constructor
def __init__(self, name):
   # instance attribute
   self.name = name
```

OOP Terminologies: Method

A subset of the overall functionality of an object. The method is defined similarly to a function (using the keyword "def") in the class definition. An object can have an arbitrary number of methods.

Example

```
# method
def Course(self):
    print(“Result")
```

OOP Terminologies: Self

The first argument when defining any method is always the **self** argument. This argument specifies the instance on which you call the method. **self** gives the Python interpreter the information about the concrete instance. To define a method, you use **self** to modify the instance attributes. But to call an instance method, you do not need to specify **self**.

Example

```
# method
def Course(self):
    print(“Result")
```

OOP Terminologies: Encapsulation

Encapsulation is the process in which no direct access is granted to the data; instead, it is hidden. To gain access to the data, user interact with the object responsible for the data.

By encapsulating data, you make the data of your system more secure and reliable.

Binding together data and functionality that manipulates the data. Encapsulation keeps the data safe from misuse and changes in the code can be done independently.

OOP Terminologies: Encapsulation

Example

```
# method
class Student:
       def __init__(self, name, matric, cgpa=0):
              self.name = name
              self.matric = matric
              self.cgpa = cgpa
       def show(self):
              print(self.name)
              print(self.matric)
              print(self.cgpa)
stud = Student("Wole", 123, 4.6)
#accessing using class method
stud.show()
#Access directly from outside
print(self.name)
print(self.matric)
print(self.cgpa)
```

OOP Terminologies: Attribute

A variable defined for a class (class attribute) or for an object (instance attribute). You use attributes to package data into enclosed units (class or instance.

Instance Attributes are unique to each object. Any Student object we create will be able to store its **name**, **matric** and **cgpa**. We can change attribute of a Student, without affecting any other Student objects we've created. class attributes.

Class Attributes are unique to each class. Each instance of the class will have this attribute. It's sometimes used to specify a default value that all objects should have after they've been instantiated.

OOP Terminologies: Attribute

Class attribute (=class variable, static variable, static attribute) A variable that is created statically in the class definition and that is shared by all class objects.

Instance attribute(=instance variable)

A variable that holds data that belongs only to a single instance. Other instances do not share this variable (in contrast to class attributes). In most cases, you create an instance attribute x in the constructor when creating the instance itself using the self keywords (e.g. self.grade = “A”).

Dynamic attribute

An instance attribute that is defined dynamically during the execution of the program and that is not defined within any method. For example, you can simply add a new attribute reg_course to any object o by calling stud. reg_course = True.

OOP Terminologies: Attribute

```
class Student:
    dept = “Computer Engineering”
    def __init__(self, name, matric, cgpa=0):
        …
        …
stud = Student(“Wole”, 123, 4.6)#instantiation
#access the instance attributes with dot notation
print(stud.name)
stud.reg_course = True
print(stud.reg_course)
```

OOP Terminologies: Method Overloading

define a method in a way that there are multiple option call to it. For example for class X, you define a method **`f(...)`** that can be called in three ways: **`f(a)`**, **`f(a,b)`**, **or** **`f(a,b,c)`**. To this end, you can define the method with default parameters (e.g. **`f(a, b=None, c=None)`**.

The Characteristics of OOP: Objects

- Object is a structure for incorporating data and the procedures for working with that data. For example, in tracking data associated with product inventory, a product object created. this is responsible for maintaining and using the data pertaining to the products.
- The object-oriented programming breaks the programming task into objects, which combine data (known as attributes) and behaviors/functions (known as methods). Thus, there are two main components of the OOP: class and object.
- The class is a blueprint to define a logical grouping of data and functions. It provides a way to create data structures that model real-world entities.

The Characteristics of OOP: Abstraction

- When constructing objects in OOP applications, it is important to incorporate this concept of abstraction. The objects include only the information that is relevant in the context of the application. For example, in a shipping application, a product object constructed with attributes such as size and weight. Incorporating a color attribute, would be extraneous or ignored. On the other hand, when constructing an order-entry application, the color could be important and would be included as an attribute of the product object.

The Characteristics of OOP: Polymorphism

- Polymorphism is the ability of two different objects to respond to the same request message in their own unique way.
- For example, Train a dog to respond to the command bark and a bird to respond to the command chirp. Another example is to send a print message to a printer object that would print the text on a printer, and send the same message to a screen object that would print the text to a window on the computer screen.
- Another good example of polymorphism is the use of words in the English language. Words have many different meanings, but through the context of the sentence you can deduce which meaning is intended.
- In OOP, polymorphism is implemented through **overloading**. This can can be implemented with different methods of an object that have the same name. The object can then tell which method to implement depending on the context (in other words, the number and type of arguments passed) of the message.

The Characteristics of OOP: Inheritance

- Most real-life objects can be classified into hierarchies. For example, all dogs have common characteristics such as four legs and fur. Their breeds further classify them into subgroups with common attributes such as size and demeanor.
- Objects can also be classified according to their function. For example, there are commercial vehicles and recreational vehicles. There are trucks and passenger cars or cars according to their make and model.
- Inheritance in OOP is used to classify the objects in your programs according to common characteristics and function. It also makes programming easier because it enables combination of general characteristics into a parent object and inherit these characteristics in the child objects.

Reference

- https://www.quora.com/Why-is-Python-an-object-oriented-language?top_ans=101893010
- https://www.pythonforbeginners.com/basics/python-quick-guide
- https://realpython.com/python-keywords/