Programming in Python

Part Three: OOP

Definitions

What's OOP?

Object-oriented programming (OOP) is a method of structuring a program by bundling related properties and behaviors into individual **objects**.

What's an "object"?

Properties define the state of the object. This is the data that the object stores. This data can be a built-in type like int, or even our own custom types.

Behaviors are the actions our object can take. Oftentimes, this involves using or modifying the properties of our object.

You have already encountered several objects. What are they??

Strings, lists, tuples, dictionaries and even numbers in Python are all objects.

Example

An object could also represent a person with properties such as age and address, and with behaviors like walking, talking, breathing, and running. Another object may represent an email with properties like recipient list, subject, and body, and behaviors like adding attachments and sending.

Procedural & Functional Programming

Procedural Programming: Structures a program like a recipe, in that it provides a set of steps, in the form of functions and code blocks, that flow sequentially in order to complete a task.

Functional Programming: Functional programming (FP) is about passing data from function to function to function to get a result. In FP, functions are treated as data, meaning you can use them as parameters, return them, build functions from other functions, and build custom functions.

Class & Object

Class

Classes are used to create user-defined data structures. Classes define functions called methods, which identify the behaviors and actions that an object created from the class can perform with its data.

Why Class?

- To represent complex information (a notch above the primitive data structures)
- Example: Managing students in a school
- Make code more manageable
- To handle large amounts of data

Class is a “blueprint” of an object

Object

Object is an instance of a class.

An object is built from a class.

Class is a form.
Object is a form filled with information.

Pythonic Class & Object

Define a Class

```
class Student:

    pass
```

Class Names are usually capitalized (PascalCase). If the class name is complex number, this is how you write it: “ComplexNumber”.

__init__ constructor

```
class Student:

    def __init__(self, name, grade):

        self.name = name

        self.grade = grade
```

`__ init__()` sets the initial state of the object by assigning the values of the object's properties. That is, `__init__()` initializes each new instance of the class.

name and grade are instance attributes.

Class attribute?

```
class Student:
    university = “NYU”
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade
```

`university` is a class attribute.

Demo

objects_instantiation.py

Will this work?

```
class Student:

    def __init__(self, name, grade):

        self.name = name

        self.grade = grade

student_obj = Student()
```

Demo

object_args.py, class_var.py, default_param.py

Objects are mutable!!

Demo

object_mutable.py

Instance methods

Instance methods are functions that are defined inside a class and can only be called from an instance of that class. Just like `__init__()`, an instance method first parameter is always self.

Demo

instance_method.py

Code??

Here's how the output should look like:

```
Given the length as {length} and
breadth as {breadth}, the area of
    the rectangle is {area}.
```

Create a class called "rectangle area" that accepts length and breadth arguments and prints the area of the rectangle.

Inheritance

What's Inheritance?

Inheritance is passing on from *x* to *y*.

Inheritance in Python is when one class picks up the attributes and methods of the other.

Newly formed classes are called child classes, and the classes that child classes are derived from are called parent classes.

Overriding & Extending

Overriding: Overriding is when you modify what you've inherited.

Extending: Extending is when you add things to what you've inherited.

Demo

override.py, extend.py

Multiple Inheritance

Multiple inheritance is when a class can inherit attributes and methods from more than one parent class. This can allow programs to reduce redundancy, but it can also introduce a certain amount of complexity as well as ambiguity, so it should be done with thought to overall program design.

Demo

multiple_inheritance.py

Encapsulation

What's Encapsulation?

Encapsulation acts as a protective shield that puts restrictions on accessing variables and methods directly, and can prevent accidental or unauthorized modification of data.

Demo

encapsulation.py

Polymorphism

What's Polymorphism?

Polymorphism is the ability to leverage the same interface for different underlying forms such as data types or classes.

```
len("hello!")       # returns 6 as result

len([11, 21, 33])     # returns 3 as result
```

Code??

Test your code with:

```
rectangle_one = Rectangle(3, 4)
rectangle_two = Rectangle(4, 4)
print(area_difference(rectangle_one,
        rectangle_two))
```

Write a function, area_difference, that takes two Rectangle instances (note: "Rectangle" is a class) as parameters and returns the signed difference in area between them.

"signed difference" means that rather than always returning a positive number, the sign of the return value should be negative if the first rectangle is smaller.

Can you have more than one class in a file?

Can multiple objects be created from the same class?