Alexandra Ralevski

Dec 5th, 2020

Foundations of Python

Assignment 08

Github link

Classes in Python

# Introduction

In Module08, I will be learning about the different types of classes in Python, as well as how to format and modify them.

# Classes and Objects

Classes are a way of grouping code (data and functions) together. Most classes are designed to either focus on data or processing. When data is in a class the data are called Fields. Data in a class is defined using **variables or constants**. You can use a class directly from memory or indirectly as a reference to the class’s code. If you use it indirectly, you create an object instance of the class and use the object’s variable. The advantage of this is that you can create multiple object instances that each have a different memory address, and are therefore separate from each other. This is useful when you want to hold different but related data, for example data on different customers from a store. In general, you use a class directly if it’s focused on processing data, and indirectly if it’s made for storing data.

# Formatting Classes

Classes follow a general design pattern, which is as follows:

**class** MyClassName(MyBaseClassName- optional):

# -- **Fields** –

# -- **Constructor** –

# -- **Attributes** –

# -- **Properties** –

# -- **Methods** –

Fields are variables within a class. The class constructor is a special type of method that sets the initial values of the class. Attributes are similar to fields except that they are ‘invisible’ fields that hold internal data. Properties are a way of managing fields/attributes. Method is a generic name for any type of function that you use to create tasks within your class.

## Instance methods

Instance methods are used when storing data, for example different types of customer data. When you want to use a class indirectly and you need to use an object instance, you must use the keyword ‘self’ within the method. The design pattern for a constructor is as follows:

def\_\_init\_\_(self, parameter, parameter… =''):

Example:

def\_\_init\_\_(self, first\_name):

#Attributes

self.strFirstName = first\_name

## Static Methods

When a class focuses on processing data, ‘static’ methods, or those that will only be referenced once, should be used. You use the keyword @staticmethod if you want to call a method directly from a class, without creating an object first. For example:

class Math(object):

@staticmethod

def Add(Value1, Value2):

return Value1 + Value2

You can call the method using the name of the class and method:

Sum = Math.Add(5,6)

print(Sum)

# Properties and Methods

A method is a function inside of a class. A property is used to manage fields or attributes. It usually consists of two properties, “getters”, which get data, and “setters”, which set data. Setters contain code for validation and error handling. The design pattern is as follows:

@name\_of\_method.setter

Example:

@first\_name.setter

def first\_name(self, value)

Getter properties use code to format a field or attribute’s data, and uses the @property directive. For example:

@property

def first\_name(self):

return str(self.\_\_first\_name).title()

# Summary

In summary, I have learned about the different ways in which classes can be constructed, and the different components that are contained in a class.