Python Bootcamp

Workshop 5

8/16

Youth In Code
Justin Liu

Roadmap

- Classes and Objects: An Intro to Object-Oriented Programming
- Debugging
- Constructors
- Inheritance
- Accessing object functions and variables
- Getters and setters
- Debugging
- 2 Practice Problems

Classes and Objects

- Objects are an encapsulation of variables and functions into a single entity
- Objects get their variables and functions from classes
- Classes are templates/blueprints for objects
- A formalized definition: obj = *instance* of a class

Constructors

- __init__ (self, a, b): used for the initialization of an instance of a class
- The __init__() function is called whenever an object is created
- You can pass in other parameters to the init and all other functions within a class
- The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class
- It doesn't have to be named self, but it is customary to do so
- It's obligatory to pass in the self parameter - Python specific

Inheritance

- Classes normally have functions (and variables) defined within them, implicit that when you create an object, that object will *inherit* all the global methods and variables from its parent class
  - Inheritance Hierarchy

Class Example

Say, for instance, we have a class named ‘Person’:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Output:
John
36

name and age are two variables automatically created when the object is instantiated

To access a specific class instance’s variables and functions, use the obj name followed by a period to “go 1 level into that directory/subset”

Class Example

- What happens when we create another instance of the same class?

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)
p2 = Person("Mike", 19)

print(p1.name)
print(p2.name)
```

Output:
John
Mike

Object Functions

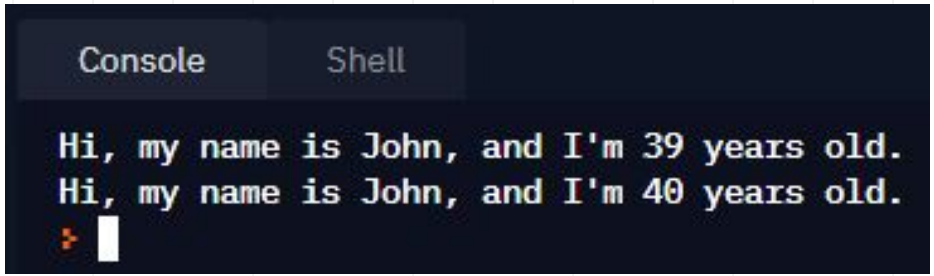
- To access a function inside of an object you use notation similar to accessing a variable

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def print_description(self):
    print("Hi, my name is " + self.name + ", and I'm " + str(self.age) + " years old.")

  def change_age(self, new_age):
    self.age = new_age

p1 = Person("John", 39)
p1.print_description()
p1.change_age(40)
p1.print_description()
```



Getters and Setters

- Getters retrieve data (often local/private variables that can't be accessed globally/publically)
  - Use a return statement
  - No args required (excluding self)
- Setters update data (also often local/private variables that can't be accessed globally/publically)
  - Our function from the previous slide, change_age(), is an example of a setter, where self.age = new_age

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def change_age(self, new_age):
    self.age = new_age

  def get_name(self):
    return self.name

p1 = Person("John", 39)

print(p1.get_name()) # Output: John
```

Modules for handling lists and dictionaries

- Numpy
  - Serves as the foundation for scientific computing and is used in data science and machine learning
  - Great alternatives to Python lists, are fast, easy to work with, and give users the opportunity to perform calculations across matrices/tensors
- Pandas
  - Key data structure: DataFrame
  - DataFrames allow you to store and manipulate tabular data in rows of observations and columns of variables -- great for handling csv files

A quick note on *debugging*

- Your code will almost never work the first time
- The process of debugging (finding + fixing your errors) is a critical skill to have as a programmer → similar to problem-solving
- Print statements are useful for debugging
- You can also print out your variables
  - Lets you validate if that variable is holding information you think it is supposed to be holding
- Some IDEs have built-in debugging tools such as breakpoints
  - You may experiment with Replit's

Practice Problems

1. Define a class “Calculator” and give it 4 functions: add, substract, multiply, and divide. Each function takes in 2 floats. Each function performs its respective operation on these 2 floats. In the constructor, initialize these 2 floats randomly. Create getters and setters for both floats. If you please, you may create a getter such that it returns a list of the 2 floats (so you don’t have to create 2 getters), and you may create a setter such that it updates both floats in a single function (to also avoid redundancy).

2. Create a class “Student”. Initialize it with 4 attributes: a 6-digit ID, first name, last name, and grade. Create a function to print out all 4 of these properties in a cohesive sentence. Instantiate 3 students and call your print function on all 3 of them. Store all 3 students in a dictionary, where the **key is their ID #**, and the **value is the Student object itself**. Loop through the dictionary, printing out each student’s ID # followed by their 1 sentence description. You do not have to make getters and setters for this problem.