Classes in Python

May 9, 2022

Administrative Notes

More on Project 3 - it’s due May 16 - next Monday night

Reminder: final exam Monday, May 23, 3:30 - 5:30 pm, ITE 104

- Worth 200 points
    - 16 T/F or MC, 4 points each - 64 points
    - 13 short answer, 7 points each - 91 points
    - 3 programming problems - 15 points each - 45 points

Sample Final will be out later this week

Next Monday (May 16) will be a review

Classes

This material is NOT on the test, but you need to be aware of this part of computer science

“Classes”, “objects”, “object-oriented programming” - buzzwords, but what are they really?

Abstractions that make programming easier

Help programmers think about ‘objects’ with properties to which they can relate

- A “car” has properties like engine type; speed; fuel consumption; number of seats; …
- It can also have properties like “who’s the driver; who are the passengers”

We want to implement “objects” to make programming easier

Done slightly differently in each programming language

Python is somewhat unique in that it is not inherently an object-oriented language

- Classes are not a fundamental part of Python
- You can do a lot of programming in Python without ever getting into classes/objects - and most people do!!

But since this is a first programming course, students need to be introduced to the topic

Classes in Python

First you have to define a class. Use the reserved word class, followed by the name of the class - any valid Python variable name. In CMSC 201, we use UpperCamelCase for class names, to make it apparent to the reader

```
class MyClass:

    """A simple example class"""

    i = 12345



    def f(self):

        return 'hello world'
```

Class instantiation

To create an object that is an instance of a class, use an assignment statement

```
x = MyClass()
```

But that produces an empty object, which is not really useful. So Python defines a “constructor method” that lets you create a new object with properties that you want

```
class Car:
   def __init__(self):
       self.make = 'Toyota'
       self.model = 'Camry'
       self.vin = '123412341234'
       self.license_plate = '3AB1234'
```

Two underscore characters

Class “car”

(from the previous slide)

That code will let you define an object that is of class ‘car’ and has the properties that you want.

But that means every car you create is a Toyota Camry with *that* VIN and *that* license plate - which is likely not what you want

So you can define the class using a skeleton

Skeleton:

```
class Car:
   def __init__(self, make, model, vin, license_plate):
       self.make =  make
       self.model = model
       self.vin =  vin       self.license_plate = license_plate
```

Now you can create a car:

It looks like a function call

my_car = Car.(‘Toyota’, ‘Corolla’, ‘12345’, ‘GoDogs’)

The arguments get mapped to the variables in the class

your_car = Car.(‘Honda’, ‘CRV’, ‘98765’, ‘GoTerps’)

Accessing those variables

my_car.make

your_car.model

my_car.vin

Methods

Functions can be defined inside of classes to operate on the variables and values internal to the class - these are called “methods” - you’ve seen that word before!!!

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

A method to add a passenger to a c

```
def add_passenger(self, passenger):
    self.passengers.append(passenger)
```

```
class Car:
    def __init__(self, make, model, vin, license_plate
        self.make = make
        self.model = model
        self.vin = vin
        self.license_plate = license_plate
        self.passengers = []
```

Add a passenger to my_car

```
eric = Passenger('Eric')
my_car.add_passenger(eric)
```