Classes

Creating and Using a Class

```python
class Dog:
    """A simple attempt to model a dog."""

    def __init__(self, name, age):
        """Initialize name and age attributes."""
        self.name = name
        self.age = age

    def sit(self):
        """Simulate a dog sitting in response to a command."""
        print(f"{self.name} is now sitting.")

    def roll_over(self):
        """Simulate rolling over in response to a command."""
        print(f"{self.name} rolled over!")

my_dog = Dog('Willie', 6)
your_dog = Dog('Lucy', 3)

print(f"My dog's name is {my_dog.name}.")
print(f"My dog is {my_dog.age} years old.")
my_dog.sit()
my_dog.roll_over()

print(f"\nYour dog's name is {your_dog.name}.")
print(f"Your dog is {your_dog.age} years old.")
your_dog.sit()
your_dog.roll_over()
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$ pyth
on dog.py
My dog's name is Willie.
My dog is 6 years old.
Willie is now sitting.
Willie rolled over!

Your dog's name is Lucy.
Your dog is 3 years old.
Lucy is now sitting.
Lucy rolled over!
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$
```

Modifying Attribute Values

```python
class Car:
    """A simple attempt to represent a car."""

    def __init__(self, make, model, year):
        """Initialize attributes to describe a car."""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print(f"This car has {self.odometer_reading} miles on it.")

    def update_odometer(self, mileage):
        """
        Set the odometer reading to the given value.
        Reject the change if it attempts to roll the odometer back.
        """
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        """Add the given amount to the odometer reading."""
        self.odometer_reading += miles

my_used_car = Car('subaru', 'outback', 2015)
print(my_used_car.get_descriptive_name())

my_used_car.update_odometer(23_500)
my_used_car.read_odometer()

my_used_car.increment_odometer(100)
my_used_car.read_odometer()
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$ pyth
on car.py
2015 Subaru Outback
This car has 23500 miles on it.
This car has 23600 miles on it.
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$
```

Inheritance

```python
class Car:
    """A simple attempt to represent a car."""

    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def read_odometer(self):
        print(f"This car has {self.odometer_reading} miles on it.")

    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        self.odometer_reading += miles
```

```python
class Battery:
    """A simple attempt to model a battery for an electric car."""

    def __init__(self, battery_size=75):
        """Initialize the battery's attributes."""
        self.battery_size = battery_size

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")

    def get_range(self):
        """Print a statement about the range this battery provides."""
        if self.battery_size == 75:
            range = 260
        elif self.battery_size == 100:
            range = 315

        print(f"This car can go about {range} miles on a full charge.")
```

```python
class ElectricCar(Car):
    """Represent aspects of a car, specific to electric vehicles."""

    def __init__(self, make, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(make, model, year)
        self.battery = Battery()

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")

my_tesla = ElectricCar('tesla', 'model s', 2019)
print(my_tesla.get_descriptive_name())
my_tesla.battery.describe_battery()
my_tesla.battery.get_range()
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$ pyth
on electric_car_with_battery.py
2019 Tesla Model S
This car has a 75-kWh battery.
This car can go about 260 miles on a full charge.
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$
```

Importing Classes

Importing a Single Class

- `car_1.py`

```python
class Car:
    """A simple attempt to represent a car."""

    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def read_odometer(self):
        print(f"This car has {self.odometer_reading} miles on it.")

    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        self.odometer_reading += miles
```

```python
class Battery:
    """A simple attempt to model a battery for an electric car."""

    def __init__(self, battery_size=75):
        """Initialize the battery's attributes."""
        self.battery_size = battery_size

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")

    def get_range(self):
        """Print a statement about the range this battery provides."""
        if self.battery_size == 75:
            range = 260
        elif self.battery_size == 100:
            range = 315

        print(f"This car can go about {range} miles on a full charge.")
```

```python
class ElectricCar(Car):
    """Represent aspects of a car, specific to electric vehicles."""

    def __init__(self, make, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(make, model, year)
        self.battery = Battery()

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")
```

- my_car.py

```
from car_1 import Car

my_new_car = Car('audi', 'a4', 2019)
print(my_new_car.get_descriptive_name())

my_new_car.odometer_reading = 23
my_new_car.read_odometer()
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$ pyth
on my_car.py
2019 Audi A4
This car has 23 miles on it.
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$
```

- my_electric_car.py

```
from car_1 import ElectricCar

my_tesla = ElectricCar('tesla', 'model s', 2019)

print(my_tesla.get_descriptive_name())
my_tesla.battery.describe_battery()
my_tesla.battery.get_range()
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$ pyth
on my_electric_car.py
2019 Tesla Model S
This car has a 75-kWh battery.
This car can go about 260 miles on a full charge.
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$
```

Importing Multiple Classes from a Module

- `my_cars.py`

```
from car_1 import Car, ElectricCar

my_beetle = Car('volkswagen', 'beetle', 2019)
print(my_beetle.get_descriptive_name())

my_tesla = ElectricCar('tesla', 'roadster', 2019)
print(my_tesla.get_descriptive_name())
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$ pyth
on my_cars.py
2019 Volkswagen Beetle
2019 Tesla Roadster
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$
```

Importing an Entire Module

- `my_cars_1.py`

```
import car_1

my_beetle = car_1.Car('volkswagen', 'beetle', 2019)
print(my_beetle.get_descriptive_name())

my_tesla = car_1.ElectricCar('tesla', 'roadster', 2019)
print(my_tesla.get_descriptive_name())
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$ pyth
on my_cars_1.py
2019 Volkswagen Beetle
2019 Tesla Roadster
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$
```

Importing a Module into a Module

- `car_2.py`

```python
class Car:
    """A simple attempt to represent a car."""

    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def read_odometer(self):
        print(f"This car has {self.odometer_reading} miles on it.")

    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        self.odometer_reading += miles


~
~
```

- electric_car_2.py

```
from car_2 import Car

class Battery:
    """A simple attempt to model a battery for an electric car."""

    def __init__(self, battery_size=75):
        """Initialize the battery's attributes."""
        self.battery_size = battery_size

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")

    def get_range(self):
        """Print a statement about the range this battery provides."""
        if self.battery_size == 75:
            range = 260
        elif self.battery_size == 100:
            range = 315

        print(f"This car can go about {range} miles on a full charge.")


class ElectricCar(Car):
    """Represent aspects of a car, specific to electric vehicles."""

    def __init__(self, make, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(make, model, year)
        self.battery = Battery()

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")


~
```

- my_cars_2.py

```
from car_2 import Car
from electric_car_2 import ElectricCar

my_beetle = Car('volkswagen', 'beetle', 2019)
print(my_beetle.get_descriptive_name())

my_tesla = ElectricCar('tesla', 'roadster', 2019)
print(my_tesla.get_descriptive_name())
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$ pyth
on my_cars_2.py
2019 Volkswagen Beetle
2019 Tesla Roadster
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$
```

Using Aliases

- `my_electric_car_1.py`

```
from car_1 import ElectricCar as EC

my_tesla = EC('tesla', 'model s', 2019)

print(my_tesla.get_descriptive_name())
my_tesla.battery.describe_battery()
my_tesla.battery.get_range()
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$ pyth
on my_electric_car_1.py
2019 Tesla Model S
This car has a 75-kWh battery.
This car can go about 260 miles on a full charge.
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_09$
```

The Python Standard Library

- The *Python standard library* is a set of modules included with every Python installation.
- Now that you have a basic understanding of how functions and classes work, you can start to use modules like these that other programmers have written.
- You can use any function or class in the standard library by including a simple import statement at the top of your file.

```
>>> from random import randint
>>> randint(1, 6)
1
>>> from random import choice
>>> players = ['charles', 'martina', 'michael', 'florence', 'eli']
>>> first_up = choice(players)
>>> first_up
'florence'
>>>
```

Styling Classes

- Class names should be written in *CamelCase*.
- To do this, capitalize the first letter of each word in the name, and don’t use underscores.
- Instance and module names should be written in lowercase with underscores between words.

- Every class should have a docstring immediately following the class definition.
- The docstring should be a brief description of what the class does, and you should follow the same formatting conventions you used for writing docstrings in functions.
- Each module should also have a docstring describing what the classes in a module can be used for.

- You can use blank lines to organize code, but don’t use them excessively.
- Within a class you can use one blank line between methods, and within a module you can use two blank lines to separate classes.

- If you need to import a module from the standard library and a module that you wrote, place the import statement for the standard library module first.
- Then add a blank line and the import statement for the module you wrote.
- In programs with multiple import statements, this convention makes it easier to see where the different modules used in the program come from.