Week 13 - Friday

COMP 1800

Last time

- What did we talk about last time?
- Finished fish and bears example

Questions?

Assignment 9

Assignment 10

Inheritance

Inheritance

- The idea of inheritance is to take one class and generate a child class
- This child class has everything that the parent class has (members and methods)
- But, you can also add more functionality to the child
- The child can be considered to be a **specialized** version of the parent

Code reuse

- The key idea behind inheritance is safe code reuse
- You can use old code that was designed to, say, sort lists of **`Vehicle`**s, and apply that code to lists of **`Car`**s
- All that you have to do is make sure that **`Car`** is a subclass (or child class) of **`Vehicle`**

Inheritance Mechanics

Creating a subclass

- All this is well and good, but how do you actually create a subclass?
- Let's start by writing the **Vehicle** class

```
class Vehicle:
  def travel(self, destination):
    print('Traveling to', destination)
```

Extending a superclass

- We use put the superclass name in parentheses when making a subclass

```
class Car(Vehicle):
  def __init__(self, model):
     self.model = model

  def getModel(self):
     return self.model

  def startEngine(self):
     print('Vroooooom!')
```

- A **`Car`** can do everything that a **`Vehicle`** can, plus more

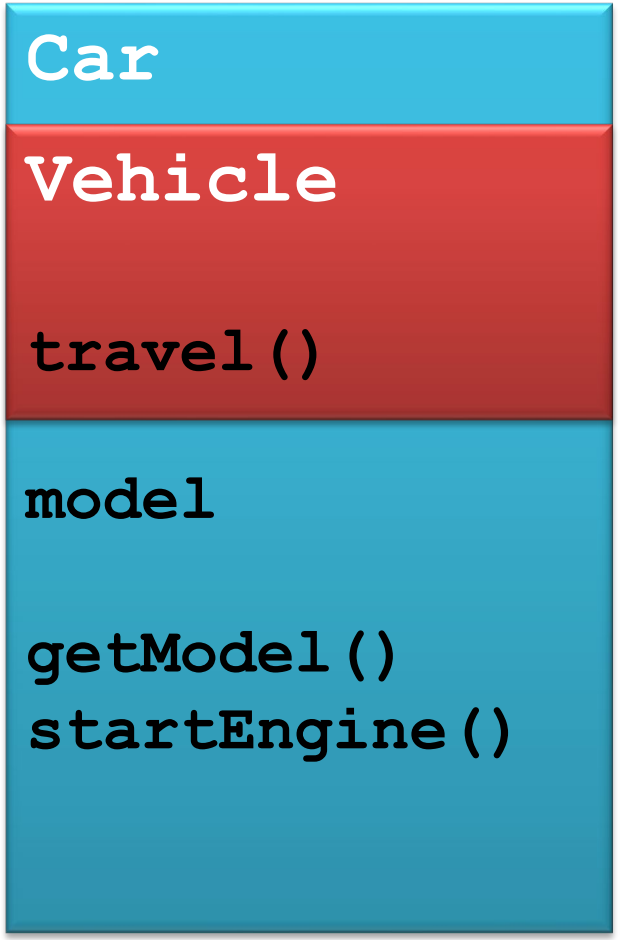
Power of inheritance

- There is a part of the **`Car`** class that knows all the **`Vehicle`** members and methods

```
car = Car('Camry')

#prints 'Camry'
print(car.getModel())

#prints 'Vroooooom!'
car.startEngine()

#prints 'Traveling to New York City'
car.travel('New York City')
```

A look at a Car

- Each **`Car`** object actually has a **`Vehicle`** object buried inside of it
- If code tries to call a method that isn't found in the **`Car`** class, it will look deeper and see if it is in the **`Vehicle`** class
- The outermost method will always be called



Calling the parent constructor

- If a class's parent has a constructor (the **`__init__()`** method), that constructor needs to get called too
  - That way, your parent gets set up correctly
- The best way to do that is to access the parent with the **`super()`** function
- Inside a class's constructor, it should call **`super().__init__()`**
  - Inserting arguments if appropriate

Parent example

- The **`Car`** class has a constructor that takes a model
- So, if we make a child class, it needs to call the parent constructor with a model

```
class RocketCar(Car):
  def __init__(self):
    super().__init__('Rocket Car')

  def fireRockets(self):
    print('Rockets firing!')
```

Overriding Methods

Adding to existing classes is nice…

- Sometimes you want to do more than add
- You want to change a method to do something different
- You can write a method in a child class that has the same name as a method in a parent class
- The child version of the method will always get called
- This is called **overriding** a method

Mammal example

- We can define the **`Mammal`** class as follows:

```
class Mammal:
  def makeNoise(self):
     print('Grunt!')
```

Mammal subclasses

- From there, we can define the **`Dog`**, **`Cat`**, and **`Human`** subclasses, overriding the **`makeNoise()`** method appropriately

```
class Dog(Mammal):
  def makeNoise(self):
     print('Woof')
```

```
class Cat(Mammal):
  def makeNoise(self):
     print('Meow')
```

```
class Human(Mammal):
  def makeNoise(self):
     print('Hello')
```

Work Time

Upcoming

Next time…

- Inheritance examples

Reminders

- Finish Assignment 9
  - **Due tonight by midnight!**
- Keep reading Chapter 12