ITP 115– Programming in Python

Software Objects

What is the Main Idea?

- We have been using different type of variables that were defined by the creators of Python
  - e.g. list, dictionary, string, etc.

- These are useful, but not for every situation

- Wouldn't it be great if we could **create our own type of variable**?
  - e.g. Dog, Color, Song, Student

Object-Oriented Programming (OOP)

- A different way of thinking about programming
- A modern methodology used in the creation of the majority of new, commercial software
- The basic building block is the **software object**
  - just called an **object**

Python is Object-Oriented

- We called list, string, dictionaries "containers"
  - Officially they are **objects**!

- They are made of *attributes* or *variables*

- Provide us *methods* to use

```
myList = ["cat", "yeti"]
my_list.sort()

msg = "hello world"
print(msg.upper())
```

Real-Life Objects

- OOP lets you represent real-life objects as software objects
  - Examples: checking account, alien spacecraft, person, house
- Objects can also represent abstract ideas
  - Examples: color, words, shapes
- Like real-life objects, software objects combine characteristics (*attributes*) and behaviors (*methods*)

Consider a vehicle

- What *attributes* does a vehicle have?
  - Think "facts about a vehicle"

- What *behaviors* can a vehicle perform?
  - Think "actions a vehicle can do"

Consider a vehicle

Attributes

- Number of wheels
- License plate number
- Fuel capacity
- Horsepower
- Color
- Make
- Model
- Year

Behaviors (*Actions*)

- Turn Right
- Turn Left
- Turn on A/C
- Turn off A/C
- Accelerate
- Decelerate
- Fill tank
- Honk Horn

Ex: One vehicle object

- Data
  - Number of wheels = 4
  - License plate number = LUIGI
  - Fuel capacity = 18 gallons
- Methods
  - Accelerate
  - Decelerate
  - Fill tank

Ex: Another vehicle object

- Data
  - Number of wheels = 3
  - License plate number = HEELZ
  - Fuel capacity = 6 gallons
- Methods
  - Accelerate
  - Decelerate
  - Fill tank

Both were vehicles

- Our "vehicle" object can describes all of these

Objects

- Objects are created (*instantiated*) from a definition called a **`class`**
  - An object is not a class—it is the realization of a class

- A programmer can create many objects from the same **`class`**
  - Each object (*instance*) instantiated from the same class will have a similar structure

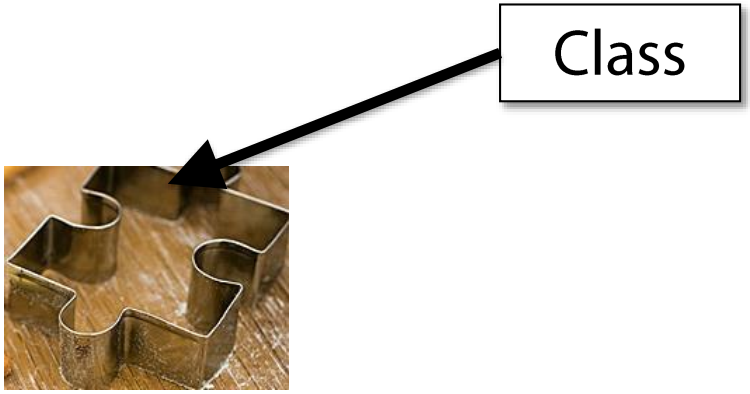
Classes

- Classes are like blueprints
  - A class is not an object—it is the design for an object

- Classes are code that defines *attributes* and *methods*

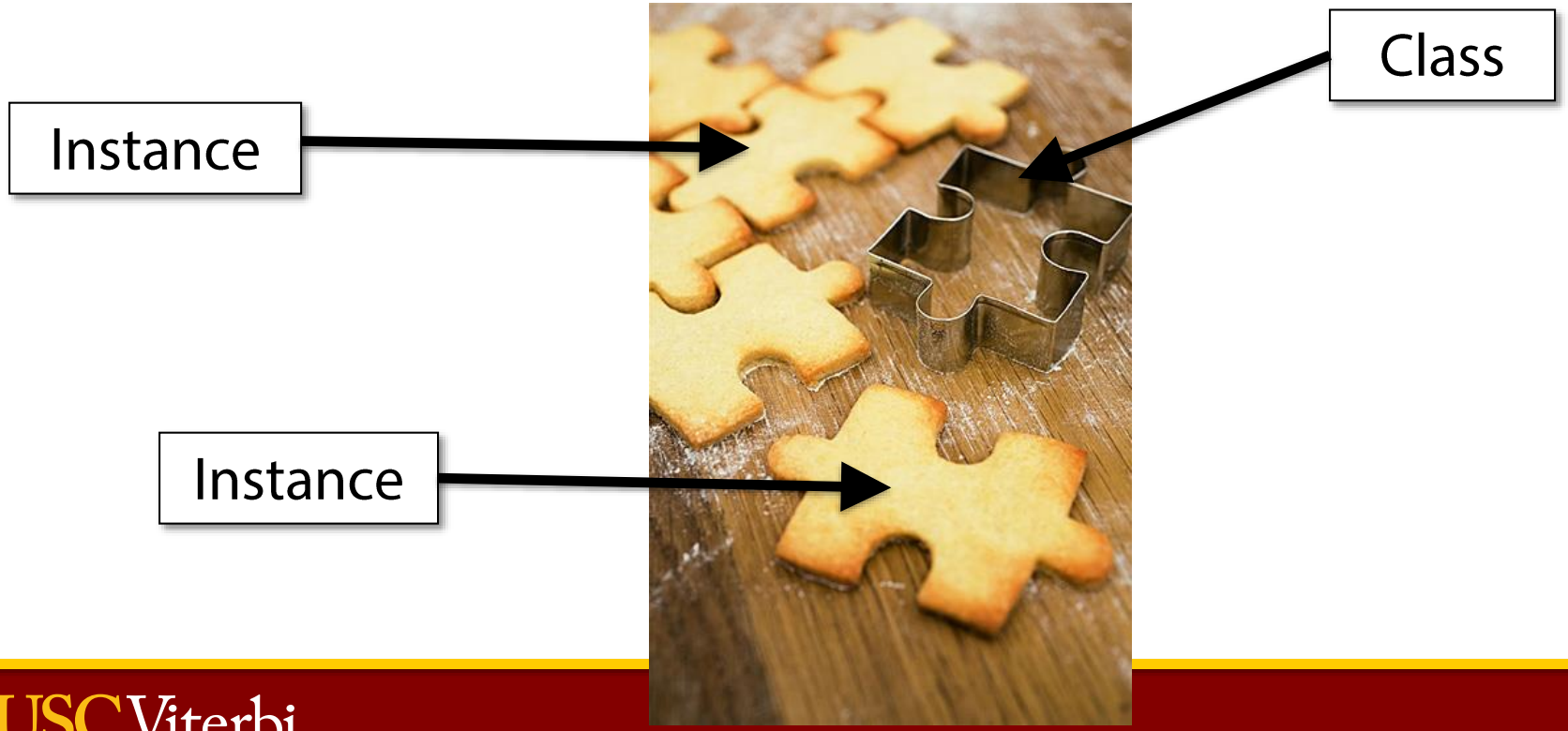
Classes and Instances

- Think of a **class** as a cookie cutter



Classes and Instances

- Think of a **class** as a cookie cutter
- **Objects** (also called **instances**) are the cookies



Very Loose Analogy

- Remember how we separately **define** function and then **call** the function?

- Similarly, we **define a class** and then **instantiate an object** from the class

OBJECTS IN PYTHON

Designing a Class

- When you are designing a programming solution, think about design
  - Does it *make sense* to use objects?
- If so, consider which attributes and methods a class will need
  - Attributes are variables *every* object will store
  - Methods are functions that are *every* object can perform

Defining a Class

- By convention, name all classes using **U**pper**C**amel**C**ase

- Classes are defined "globally"
  - Aligned to far left
  - Outside of / separate from `main` or any functions

Defining a Class

- Syntax:

```
class ClassName(object):
    # rest of the code to define class
```

  – where `ClassName` is the name of the class

Defining a Class

- Example:

```
class Vehicle(object):
  ...

def main():
  ...

main()
```

Creating an Instance of a Class

- We just *defined* a general blueprint for a class called **Vehicle**, but we want to create an actual object

- To use a class, we need to create an instance of the class (also called ***instantiating an object***)

Creating an Instance of a Class

- Syntax

```
varName = ClassName()
```

  - *varName* is the name of a variable that will store the object
  - *ClassName* is the name of the class

Creating an Instance of a Class

- Example:

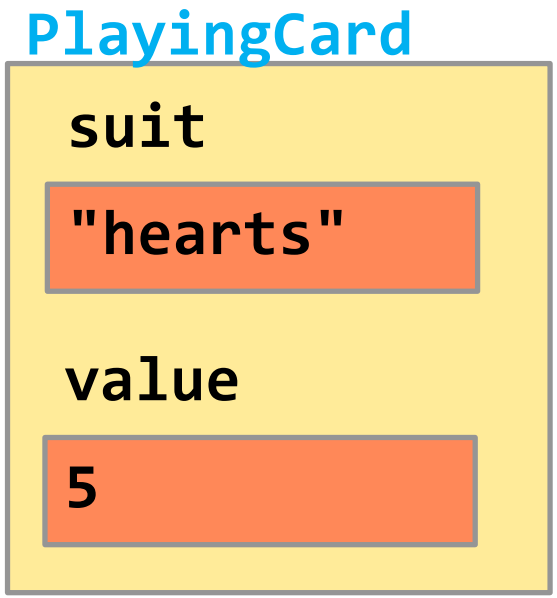
```
class Vehicle(object):
  ...

def main():
  v1 = Vehicle()

main()
```

Attributes

- Recall: we can store other variables inside a list or dictionary variable
- We can also store other variables inside an object
- Variables contained inside an object are called **attributes**



In other languages, attributes are often called *instance variables* or *member variables*

Attributes and Constructor

- We use a **constructor** to define what attributes will exist inside a object

- A constructor is **method** that is used to create an instance of an object
  - A **method** is basically a **function** that is part of a class (more later)

Constructors

- Syntax:

```
def __init__(self):
```

- Constructors have **no return value**
- Constructors are **called automatically** when you create an object
- We will explain **self** in a moment…

Unlike C++ or Java, attributes are defined in the constructor

Attributes Example

```
class Vehicle(object):
 def __init__(self, makeParam, modelParam):
```

Attributes Example

```
class Vehicle(object):
 def __init__(self, makeParam, modelParam):
   self.make = makeParam
   self.model = modelParam
   self.mpg = 0
   # note that we will calculate mpg later, but
   # we need to create the attribute now
```

Attributes Example

```
class Vehicle(object):
 def __init__(self, makeParam, modelParam):
   self.make = makeParam
   self.model = modelParam
   self.mpg = 0

def main():
   car1 = Vehicle("Ford", "Fiesta")
```

Attributes Example

```
class Vehicle(object):
 def __init__(self, makeParam, modelParam):
   self.make = makeParam
   self.model = modelParam
   self.mpg = 0

def main():
   car1 = Vehicle("Ford", "Fiesta")
```

When we instantiate an object, Python **automagically** calls the **`__init__`** constructor

Attributes Example

```
class Vehicle(object):
 def __init__(self, makeParam, modelParam):
   self.make = makeParam
   self.model = modelParam
   self.mpg = 0

def main():
   car1 = Vehicle("Ford", "Fiesta")
```

The first argument in the initialization maps to the second parameter

Attributes Example

```
class Vehicle(object):
 def __init__(self, makeParam, modelParam):
   self.make = makeParam
   self.model = modelParam
   self.mpg = 0

def main():
   car1 = Vehicle("Ford", "Fiesta")
```

The second argument in the initialization maps to the third parameter

And so on…

Attributes Example

```
class Vehicle(object):
 def __init__(self, makeParam, modelParam):
   self.make = makeParam
   self.model = modelParam
   self.mpg = 0

def main():
   car1 = Vehicle("Ford", "Fiesta")
```

But what is **self**?

Attributes Example

```
class Vehicle(object):
 def __init__(self, makeParam, modelParam):
   self.make = makeParam
   self.model = modelParam
   self.mpg = 0

def main():
   car1 = Vehicle("Ford", "Fiesta")
```

`self` refers to the **new, unique** object that was just created

Imagine a Conversation…

- Ron:
  - "I am so behind on my potions homework"
- Hermione:
  - "I am already finished. I can help you"

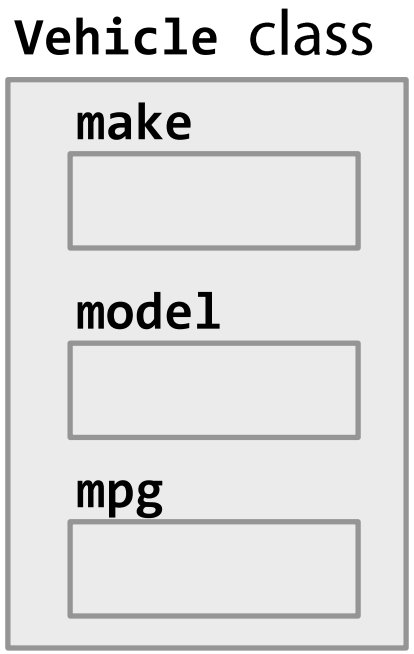
- To whom does I refer?
  - It depends on who is speaking
  - I is a way for people to refer to themselves

`self`

- `self` is a way for an object to refer to itself
- `self` always refers to a unique object that called a method (or was created by a constructor)
- Attributes are part of the object so it must be preceded by `self`
  - Attributes are stored inside of the object so they exist after the constructor ends
  - Unlike regular local variables in a function

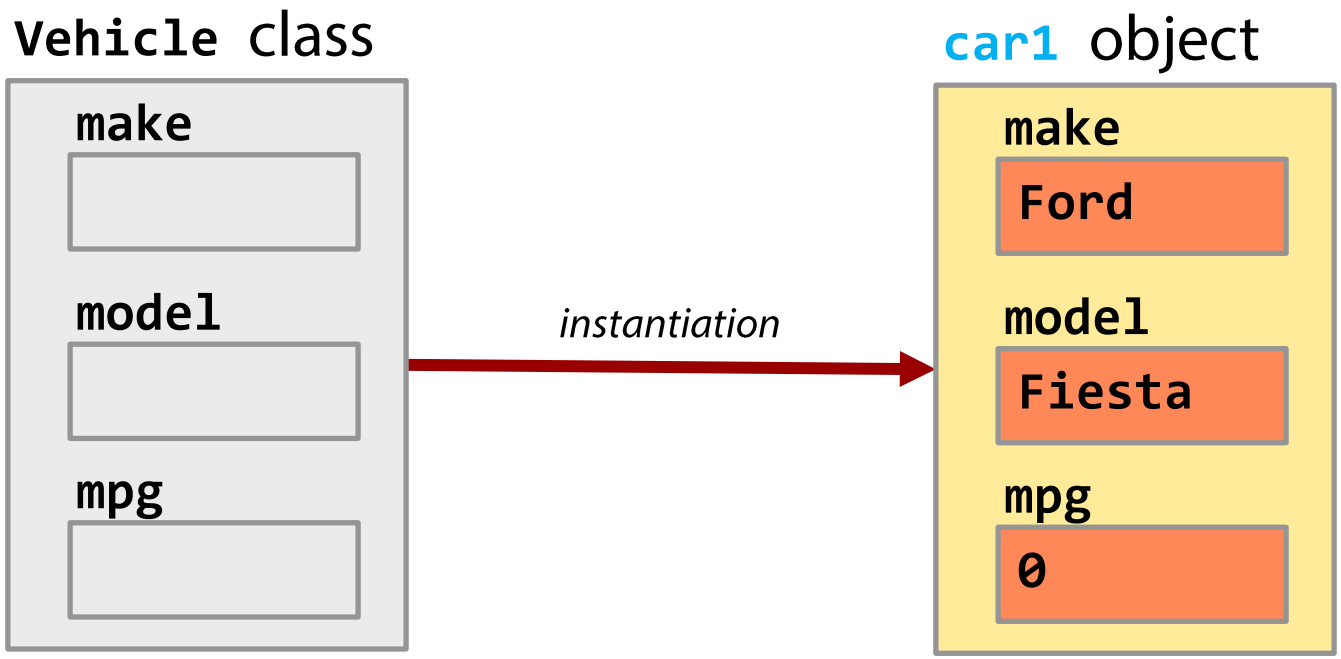
What Does All This Mean?

What Does All This Mean?



When we define a class, we describe the blueprint for what objects will look like

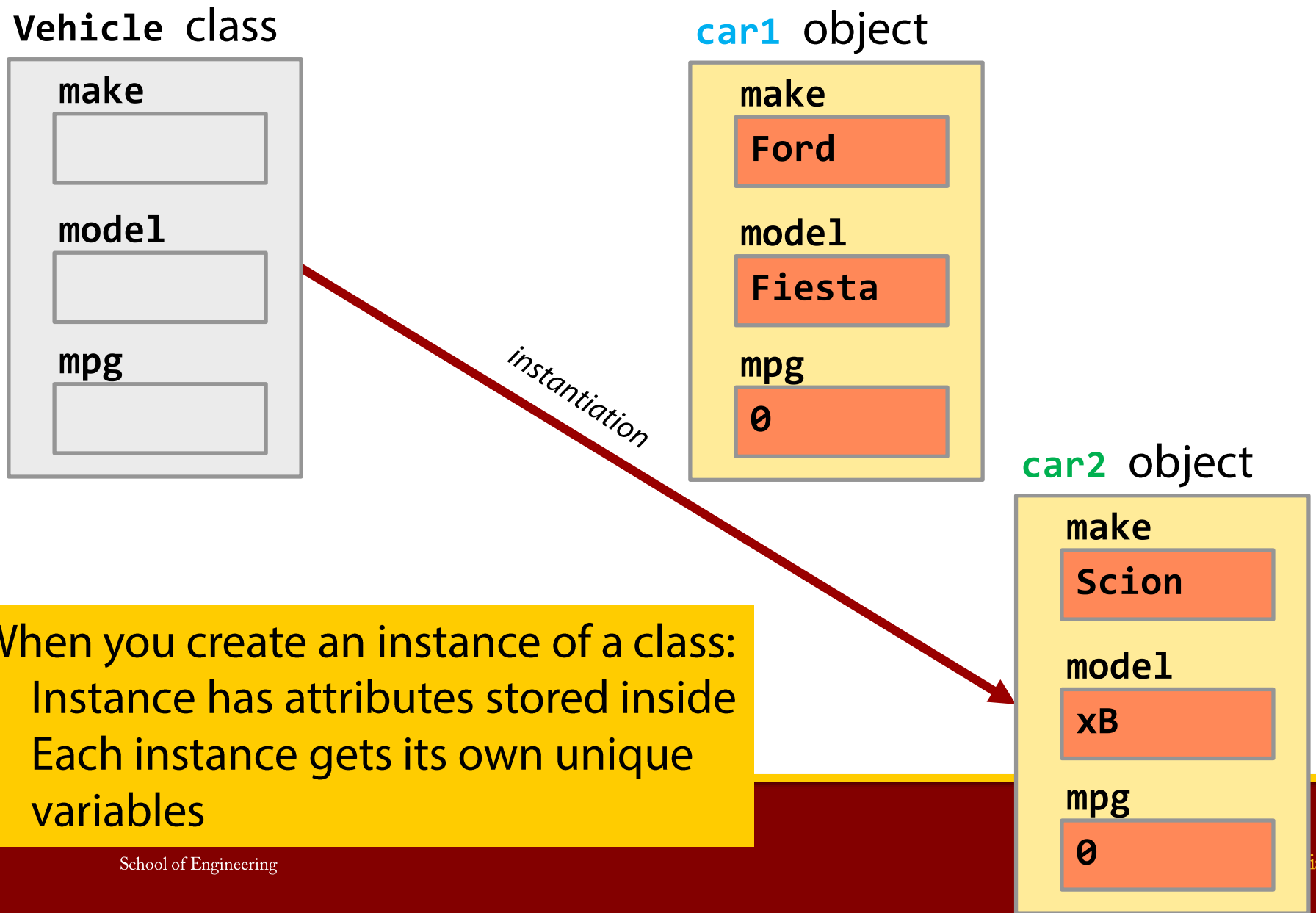
What Does All This Mean?



When you create an instance of a class:

- Instance has attributes stored inside

What Does All This Mean?



When you create an instance of a class:

- Instance has attributes stored inside
- Each instance gets its own unique variables

Putting it All Together

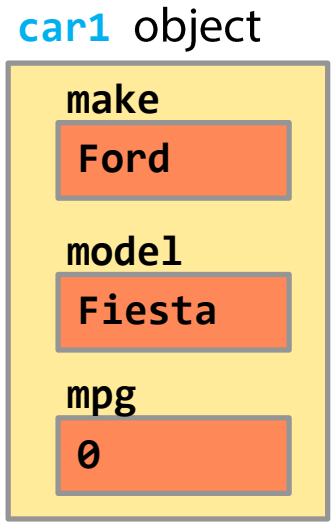
```
class Vehicle(object):
  def __init__(self, makeParam, modelParam):
    self.make = makeParam
    self.model = modelParam
    self.mpg = 0
```

Putting it All Together

```
class Vehicle(object):
  def __init__(self, makeParam, modelParam):
    self.make = makeParam
    self.model = modelParam
    self.mpg = 0

def main():
  car1 = Vehicle("Ford", "Fiesta")


main()
```

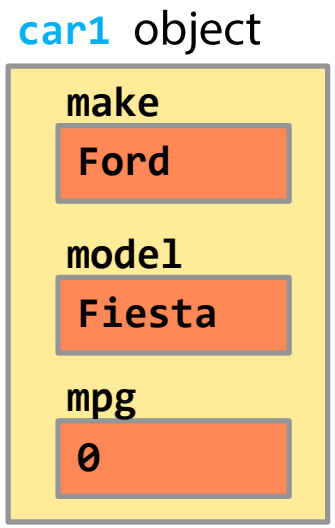


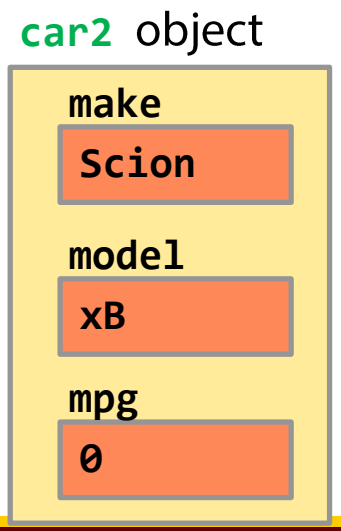
Putting it All Together

```
class Vehicle(object):
  def __init__(self, makeParam, modelParam):
    self.make = makeParam
    self.model = modelParam
    self.mpg = 0

def main():
  car1 = Vehicle("Ford", "Fiesta")
  car2 = Vehicle("Scion", "xB")

main()
```





- End lecture

Methods

- Classes can have methods (or behaviors)
- You can think of methods as *functions associated with an object*
- Methods are defined <u>inside</u> of the class
  - This is what makes them different from regular functions

Functions and Methods

- Functions are free-standing blocks of code that we can use

```
print("Hello world")
drink = input("What coffee do you want?")
```

- Methods are basically functions that are part of an object

```
word = word.upper()
numList = line.split(",")
```

Methods are like Other Functions

- Output
  - Can return a value
- Input
  - Can take input parameters
- Contents of the method are in a block (indented)

Methods

- Methods are part of the object <u>just like</u> attributes

- Methods can access the attributes defined in the constructor using **`self`**

Defining a Method

- Syntax:

```
def methodName(self):
```

  – `methodName` is the name of the method

- Every method special first parameter `self`
  – Provides a way for a method to refer to the object itself

Calling a Method

- Syntax

```
varName.methodName()
```

  - `varName` is the name of the variable object *(not the name of the class)*
  - `methodName` is the name of the method

Calling a Method

- Recall:
  - Every list object has a **sort** method

```
myList = ["cat", "yeti"]
myList.sort()
```

- Once we have created / instantiated an object, we can call methods

Calling a Method

- Example:

```
class Vehicle(object):
  def startEngine(self):
  ...

def main():
  v1 = Vehicle()
  v1.startEngine()
```

Calling a Method

- Example:

```
class Vehicle(object):
  def startEngine(self):
  ...

def main():
  v1 = Vehicle()
  v1.startEngine()
```

self refers to the object that called the method

Method Input and Output

- As with functions, methods can receive input parameters and return output values

- Syntax

```
def methodName(self, param1, param2, ...)
  ...
  return someVariable
```

Example: Method Input and Output

```
class Vehicle(object):
  def calcTripCost(self, miles):
    ... #perform some calculations
    return totalCost #new variable


def main():
  v1 = Vehicle()
  cost = v1.calcTripCost(100)
```

Example: Method Input and Output

```
class Vehicle(object):
  def calcTripCost(self, miles):
    ... #perform some calculations
    return totalCost #new variable

def main():
  v1 = Vehicle()
  cost = v1.calcTripCost(100)
```

self refers to the object that called the method

Example: Method Input and Output

```
class Vehicle(object):
  def calcTripCost(self, miles):
    ... #perform some calculations
    return totalCost #new variable

def main():
  v1 = Vehicle()
  cost = v1.calcTripCost(100)
```

Rest of the parameters go in order

Example: Method Input and Output

```
class Vehicle(object):
  def calcTripCost(self, miles):
    ... #perform some calculations
    return totalCost #new variable

def main():
  v1 = Vehicle()
  cost = v1.calcTripCost(100)
```

Return values just like we did with functions

`__str__()`

- Special method you can create that can be used to display the attributes of an object

- This called automatically whenever you attempt to "print" an instance
  - Pass instance as argument to print function

__str__()

- Syntax

```
def __str__(self):
    return "This will have some data"
```

- **Important:** This method ***returns*** a string; it does not print directly

__str__()

- Example

```
class Vehicle(object):
    ...
    def __str__(self):
        msg = "Make: " + self.make
        msg += "Model: " + self.model
        return msg
```

Printing an Instance

```
def main():
    v = Vehicle("Mario Kart", "Zip Zip")
    print(v)
```

Output

```
Make: Mario Kart
Model: Zip Zip
```