

# CS65: Introduction to Computer Science

Classes and Objects



Md Alimoor Reza  
Assistant Professor of Computer Science

# Python Data Types

- **Primitive data types:**

- integer, floating point, boolean, etc

```
cost = 100  
score = 10.56  
is_present_in_class = True
```

- **Complex data types:**

- Sequence (String, List, Tuple)
- Dictionary

```
cost_list = [100, 200, 300]  
last_name = "Reza"  
cipher_mapping = {'a':'d', 'b':'e', 'c':'f'}
```

- **Custom data type:**

- Class

```
student = ???  
person = ???  
exam = ???  
pet = ???
```

# Class: Custom Data Type

- **Class is a Custom data type:**
  - We can create our own **new types** with a **class definition**
  - **Class definition** is a 'blueprint' of what should be included in a value of this type
- **Same operations can be done on this custom data type:**
  - Can create a variable of this **new type**
  - Change the value of the variable of this **new type**
  - Create List or Tuple with variables of this **new type**

# Topics

- What is a ‘Class’?
- What is an ‘Object’?
- Difference/connection between Class and Object
- Class components
  - Initializer/Constructor
  - Attributes
  - Methods

# Object Oriented Programming (OOP)

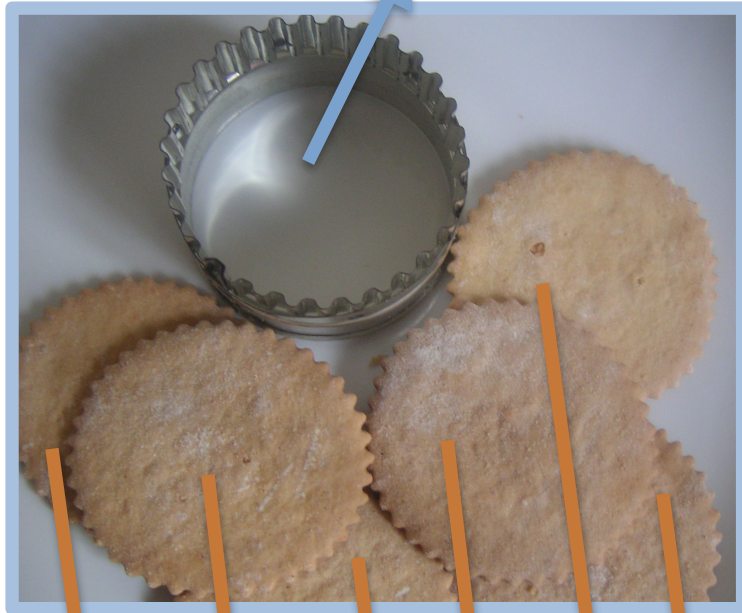
- Focus so far (up to previous lecture) has been on basic algorithmic programming structures
- Now, we will turn our focus on how data is stored and manipulated
  - Object Oriented Programming
  - We will still be developing algorithms
- It's a different way of thinking about writing code

# Object Oriented Programming (OOP)

- **Class** is the blueprint or template for creating an object
- **Object**: a combination of data and associated procedures created based on the blueprint of class
- **Object Oriented Programming (OOP)** is centered on creating Objects

# Object Oriented Programming (OOP)

Cookie-cutter is a Class



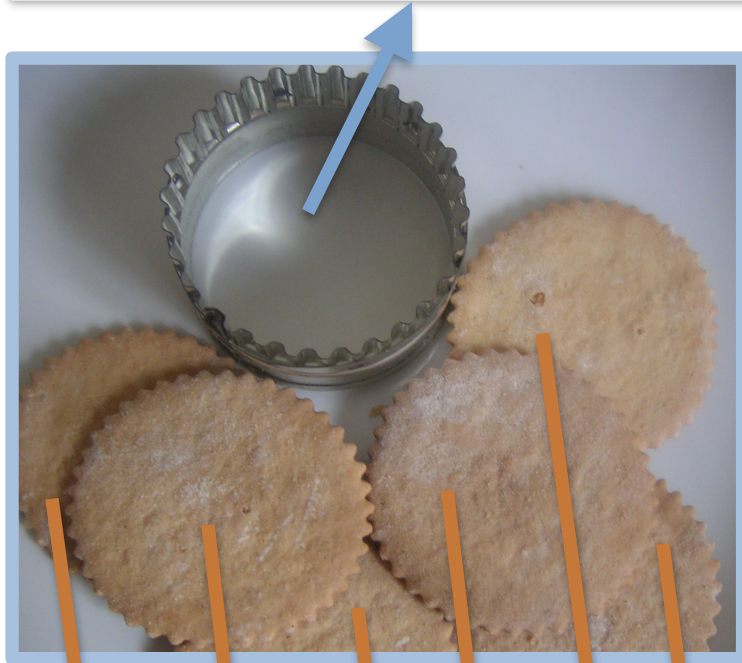
Cookies are objects



Cookies are being curved out based on the cookie-cutter's shape

# Object Oriented Programming (OOP)

From a blueprint of **Cookie-cutter**

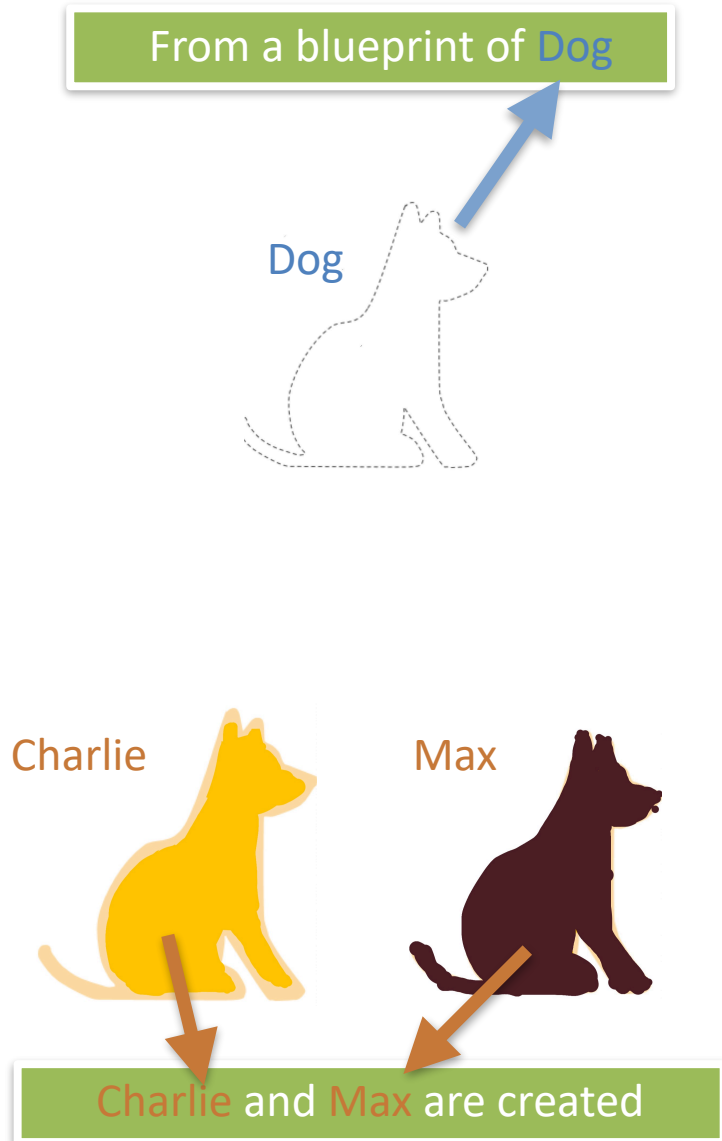


Cookie<sup>1</sup>, Cookie<sup>2</sup>, Cookie<sup>3</sup>,  
Cookie<sup>4</sup>, Cookie<sup>5</sup>, Cookie<sup>6</sup>  
are created

- **Class** is a blueprint or template that defines what attribute and methods **Objects** can have
- **Cookies** are made with a **Cookie-cutter** — **Objects** are made from a **Class**
- **Class** is a shape with which many individual **Objects** can be created — in the same way **Cookie-cutter** is a shape with which many **cookies** can be created



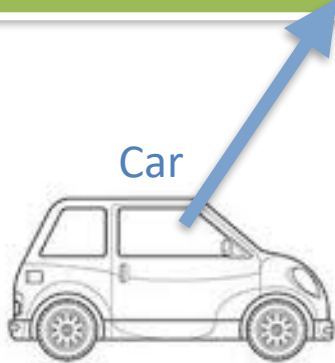
# Object Oriented Programming (OOP)



- **Class** is a blueprint or template that defines what attribute and methods **Objects** can have
- **Charlie and Max** are made from the template of a **Dog** — **Objects** are made from a **Class**
- **Class** is a shape with which many individual **Objects** can be created — in the same way **Dog** is a shape with which **Charlie, Max, etc** can be created

# Object Oriented Programming (OOP)

From a blueprint of **Car**



- **Class** is a blueprint or template that defines what attribute and methods **Objects** can have
- **Fiat<sub>1</sub>** and **Fiat<sub>2</sub>** made with a **Car** — **Objects** are made from a **Class**
- **Class** is a shape with which many individual **Objects** can be created — in the same way **Car** is a shape with which **Fiat<sub>1</sub>** and **Fiat<sub>2</sub>** can be created

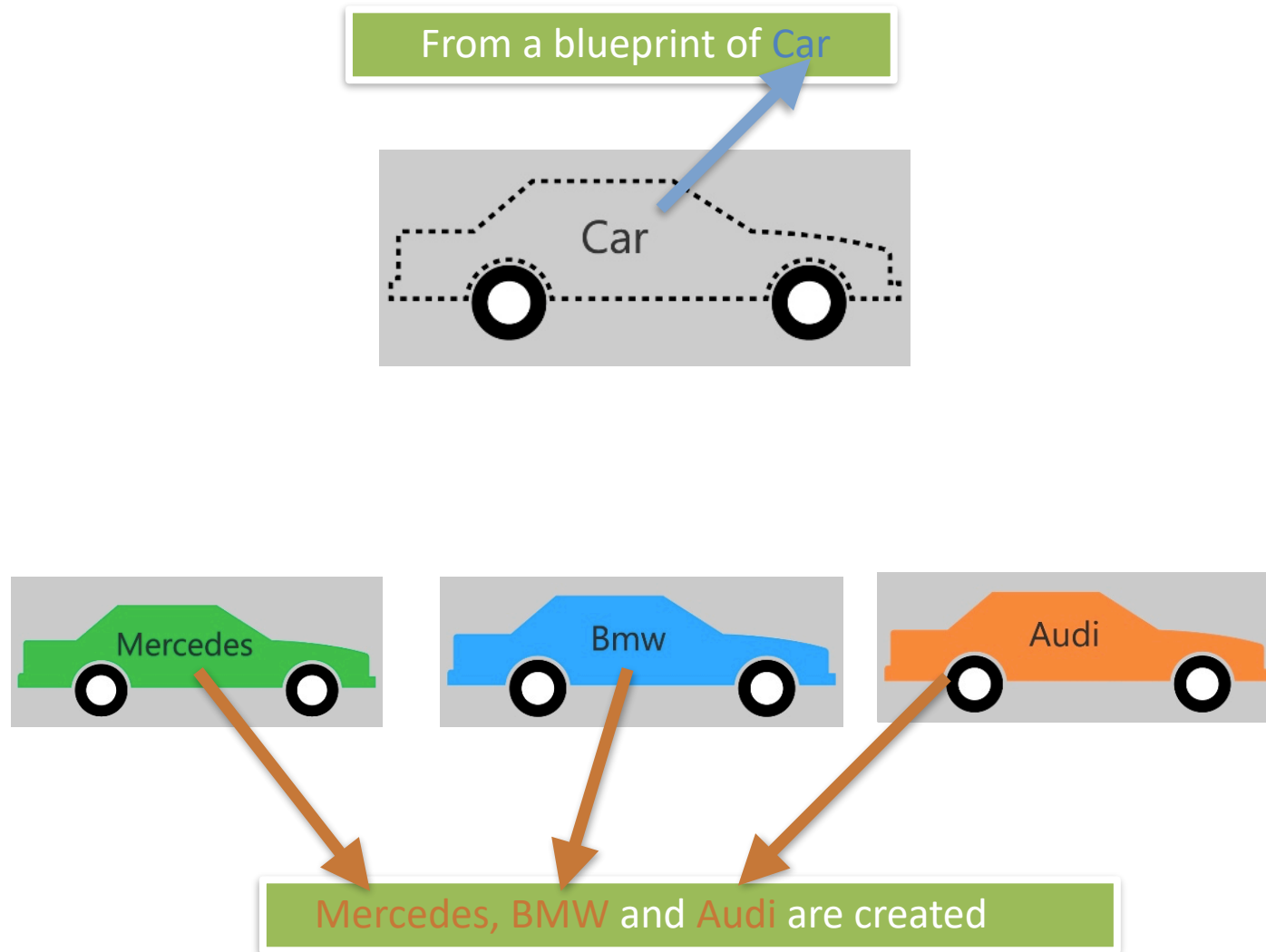
**Fiat<sub>1</sub>**

**Fiat<sub>2</sub>**



**Fiat<sub>1</sub>** and **Fiat<sub>2</sub>** are created

# Object Oriented Programming (OOP)



# Object Oriented Programming (OOP)

From a blueprint of **House**

House



Sweet Home

Corporate Home



Sweet Home and Corporate Home are created

- **Class** is a blueprint or template that defines what attribute and methods **Objects** can have
- **Sweet Home** and **Corporate Home** are made with a **House** template — **Objects** are made from a **Class** template
- **Class** is a shape with which many individual **Objects** can be created — in the same way **House** is a shape with which **Sweet Home** and **Corporate Home** can be created

# Object Oriented Programming (OOP)

- Object Oriented Programming (OOP) is centered on creating Objects
  - **Object**: a combination of *data components* and associated *procedures/functions*
    - Data components are called attributes or fields
    - Procedures/functions are called methods
- A **class** is the blueprint of an object
  - Defines attributes/fields and methods associated with an object
  - Classes are useless without objects
- These concepts lead to easily developing code that can be reusable

# Class and Object

- **Class** - a custom data type
- **Object** - an “instance” of a **class**
- Analogy:
  - `[15, 16, 17]` is value of **List** type
  - `“Computer Science”` is value of **String** type
  - `100.2345` is value of **Float** type
  - **Mercedes** is an instance of **Car Class** type
  - **Charlie** is an instance of **Dog Class** type

# Class

- Class: code that specifies the attributes and methods of a particular type of object
  - Similar to a blueprint of a house or a cookie cutter
- Instance: an *object* (a variable) created from a class
  - Similar to a specific house built according to the blueprint or a specific cookie
  - There can be many instances of one class
- Objects are interchangeably called instances

# Class Definition

- Class definition: set of statements that define a class's methods and data attributes
  - Format: begin with `class ClassName:`
    - Use `class` keyword
    - Class names often start with uppercase letter
  - Method definition like any other python function definition
    - `self` parameter: required in every method in the class – references *the specific object* that the method is working on



# Class definition (initializer method)

- Initializer method: automatically executed when an instance of the class is created. It is also known as constructor
  - Initializes object's attributes and assigns `self` parameter to the object that was just created
  - format: `def __init__(self):`
  - usually the first method in a class definition

# Class definition (`__str__` method)

- `__str__` method:
  - automatically executed when an instance of the class is **printed**
- `__str__` method should return a string
  - when the object is printed, the contents of the `__str__` method will be output

# Example: Dog Class

- Initializer (also known as ‘constructor’) method name is `__init__`
- Heads up: double underscores on both sides ( `__init__` )
- Must have at least one first formal parameter `self`
- May have more parameters beside `self`
- Primary task is to define attributes of the class

```
class Dog:
    def __init__(self, par_name, par_color, par_breed):
        self.name = par_name
        self.color = par_color
        self.breed = par_breed

    def __str__(self):
        str_var = "Dog ( " + self.name + ", " + self.color + ", "
        str_var = str_var + self.breed + " )"
        return str_var
```

# Class definition

- Object: a combination of data components and associated procedures
  - data components are called attributes or fields
  - functions are called methods
- The keyword self tells Python that the variable following the period (.) *references* a particular attribute of the defined class
  - Example: in the `Dog` class, `self.name`, `self.color`, `self.breed`

# Object Instantiation

- To create a new ‘instance of a class’ (a.k.a known as *object*) call the initializer method
  - Format: `my_instance = ClassName(arg1, arg2)`
- To call any of the methods with the created instance, use **dot** notation
  - Format: `my_instance.method()`
  - Because the `self` parameter references the specific instance of the class, the method will affect that instance only
    - Reference to `self` is passed automatically

# Example: Dog Class

- Class definition

```
class Dog:
    def __init__(self, par_name, par_color, par_breed):
        ...
        ...
        ...
```

- Create an instance `charlie_obj` (an object) from the `Dog` class template

```
charlie_obj = Dog("Charlie", "yellow", "golden retriever")
```

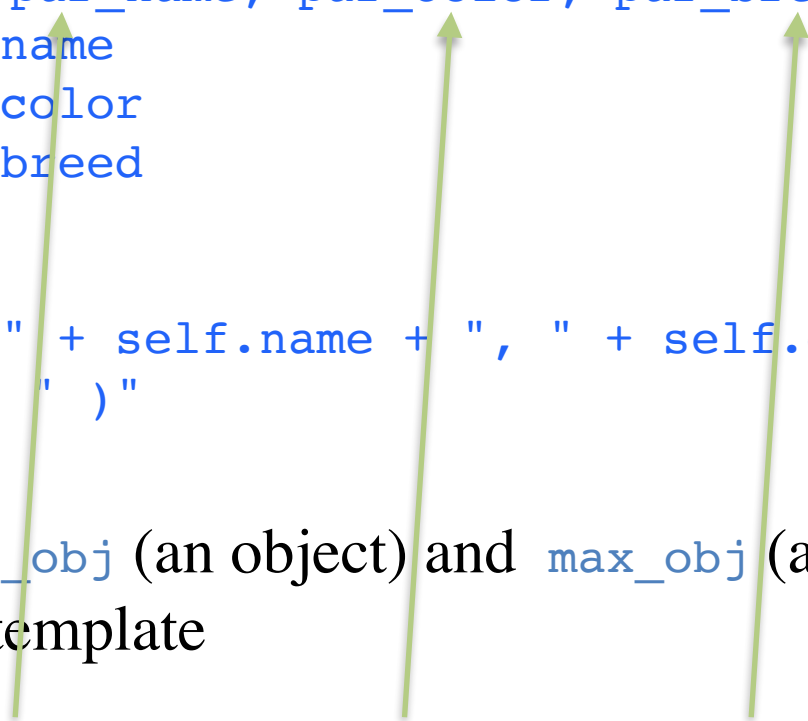
- This calls the `__init__` definition of the class `Dog`
- Note that `self` parameter seems to have been supplied some other way

# Example: Dog Class

- Class definition

```
class Dog:
    def __init__(self, par_name, par_color, par_breed):
        self.name = par_name
        self.color = par_color
        self.breed = par_breed

    def __str__(self):
        str_var = "Dog ( " + self.name + ", " + self.color + ", "
        str_var = str_var + self.breed + " )"
        return str_var
```



- Create an instance `charlie_obj` (an object) and `max_obj` (another object) from the `Dog` class template

```
charlie_obj = Dog("Charlie", "yellow", "golden retriever")
max_obj     = Dog("Max",      "brown",  "golden retriever")
```

# Example: Dog Class

- Class definition

```
class Dog:
    def __init__(self, par_name, par_color, par_breed):
        self.name = par_name
        self.color = par_color
        self.breed = par_breed
```

- Create an instance `charlie_obj` (an object) from the `Dog` class template

```
charlie_obj = Dog("Charlie", "yellow", "golden retriever")
max_obj     = Dog("Max",      "brown",  "golden retriever")
```

- Each object is an instance of the class, hence each variable that exists inside the object is called instance variable:

- `charlie_obj` has its own `name`, its own `color`, and its own `breed`
- `max_obj` has its own `name`, its own `color`, and its own `breed`



# Exercise: Create Person Class

- Class definition

```
class:  
...  
...  
...
```

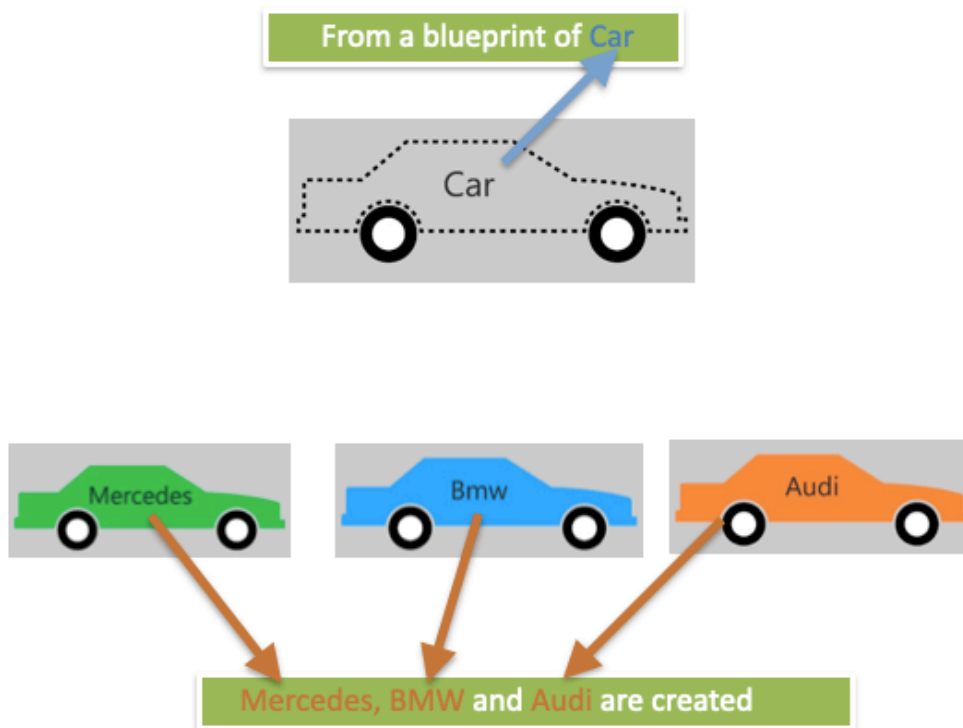
- Create an instance `reza_obj` (an object) from the `Person` class template
- Create an instance `chris_obj` (an object) from the `Person` class template
  
- Each object is an instance of the class, hence each variable that exists inside the object is called instance variable:

- `reza_obj` has its own `?`, its own `?` and its own `?`
- `chris_obj` has its own `?`, its own `?` and its own `?`

# Exercise: Create Car Class

- Class definition

```
class:  
...  
...  
...
```

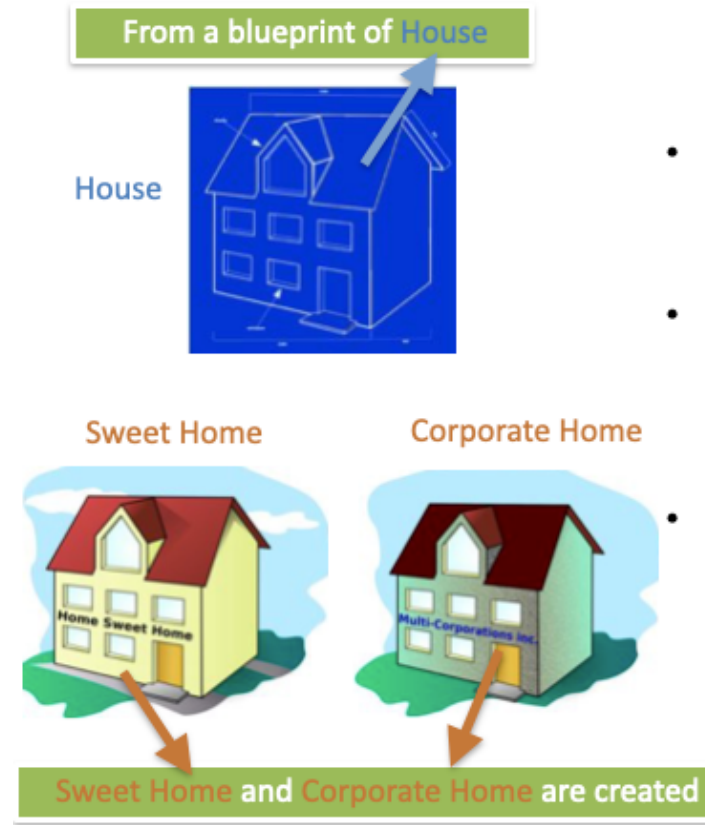


- Create an instance `bmw_obj` (an object) from the `Car` class template
- Create an instance `mercedes_obj` (an object) from the `Car` class template
- Create an instance `audi_obj` (an object) from the `Car` class template

# Exercise: Create House Class

- Class definition

```
class:  
...  
...  
...
```



- Create an instance `sweet_home_obj` (an object) from the `House` class
- Create an instance `corporate_home_obj` (an object) from the `House` class

# Summary: Classes and Objects

- **Step 1: Class definition (blueprint)**

- Class definition tells Python how the new data type works.

- **Step 2: Object instantiation (creation)**

- An object must be instantiated (created) from the class definition, to fill in instance variables, before it can be used.

- **Step 3: Object manipulation (use)**

- Once object exists, we can read/write its data (access its attributes or fields), and use its behaviors (call its methods).