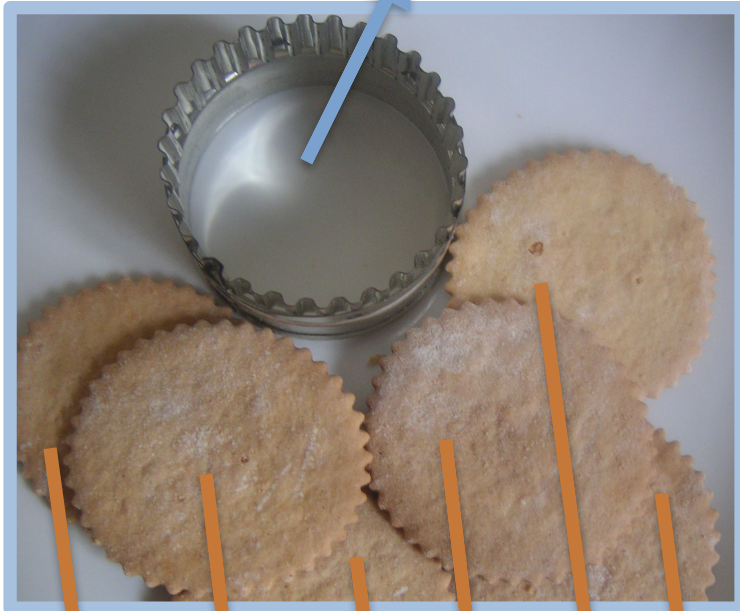
CS65: Introduction to Computer Science

Classes and Objects

Md Alimoor Reza
Assistant Professor of Computer Science

Review: Object Oriented Programming (OOP)

From a blueprint of Cookie-cutter



Cookie[1] , Cookie[2], Cookie[3], Cookie[4], Cookie[5], Cookie[6] are created

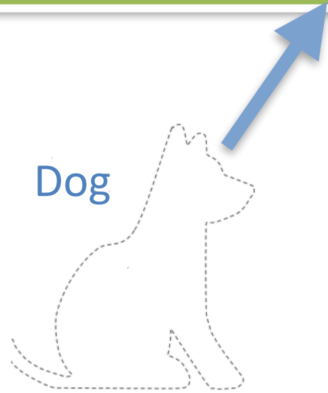
- Class is a blueprint or template that defines what attribute and methods Objects can have

- **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

Review: Object Oriented Programming (OOP)

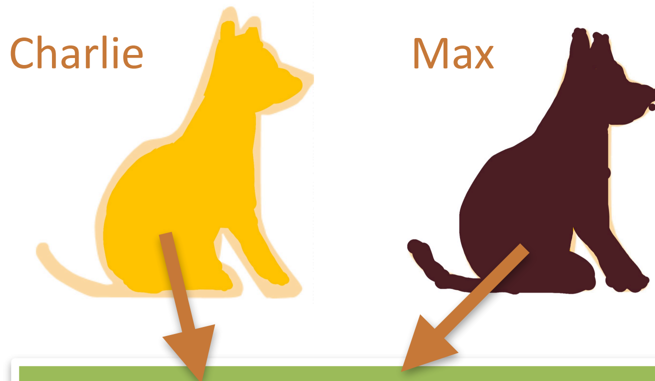
From a blueprint of Dog



- Class is a blueprint or template that defines what attribute and methods Objects can have

- **Charlie and Max** are made from a **Dog** template — **Objects** are made from a **Class** template

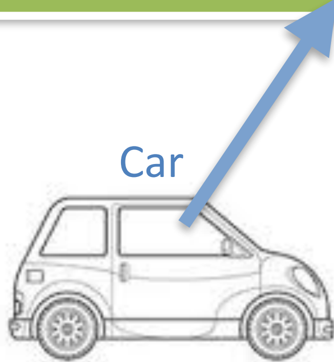
- **Class** is a shape with which many individual Objects can be created — in the same way Dog is a shape with which Charlie, Max, etc can be created



Charlie and Max are created

Review: Object Oriented Programming (OOP)

From a blueprint of Car



Fiat$_1$ and Fiat$_2$ are created

- Class is a blueprint or template that defines what attribute and methods **Objects** can have
- **Fiat$_1$** and **Fiat$_2$** made with a **Car** template — **Objects** are made from a **Class** template
- Class is a shape with which many individual **Objects** can be created — in the same way Car is a shape with which **Fiat$_1$** and **Fiat$_2$** can be created

Agenda

- Defining a Class and Creating Objects from it

- Accessing Object Attributes and Methods
  - Attributes/fields (characteristics of an object)
    - **how to access and change those**
  - Methods (behavior)
    - `__init__()` and `__str__()` methods
    - **other methods**
    - **how to add a method in a class**

- Scope and Visibility

Drake UNIVERSITY

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

Drake UNIVERSITY

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

Drake UNIVERSITY

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 + ",
    " +  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 knows 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):
    …
    …
    …
```

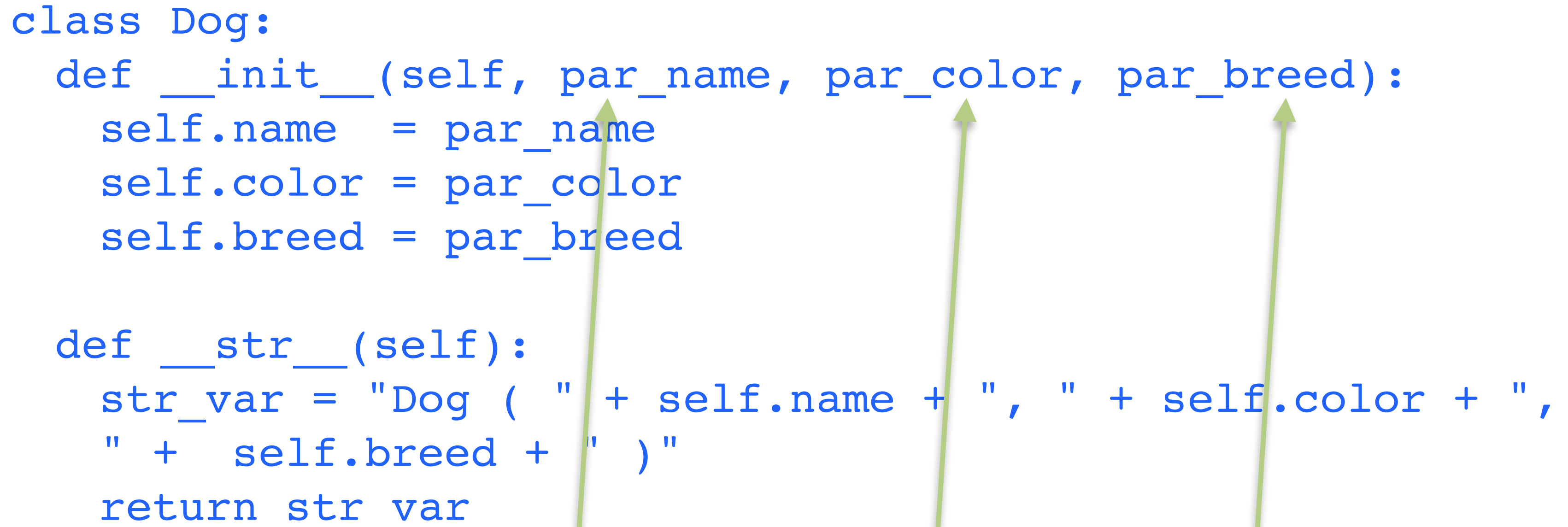
- Create an instance charlie_obj (an object) from the Dog class template

```
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 + ",
    " +  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

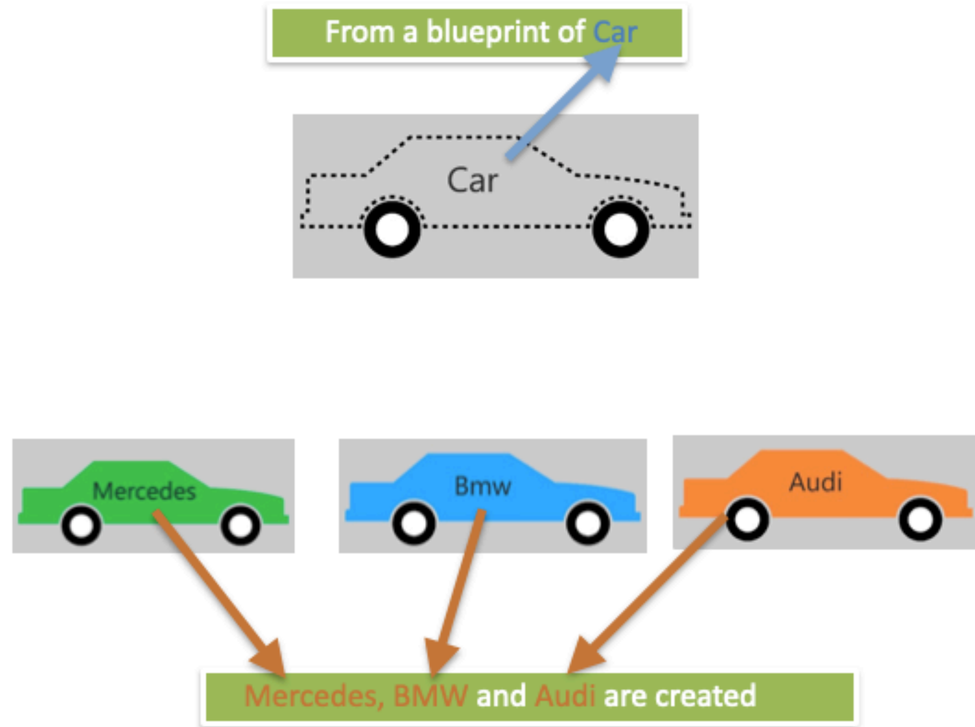
- Each object is an instance of the class, hence each variable that exists inside the object is called instance variable:

    - `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:
    …
    …
    …
```

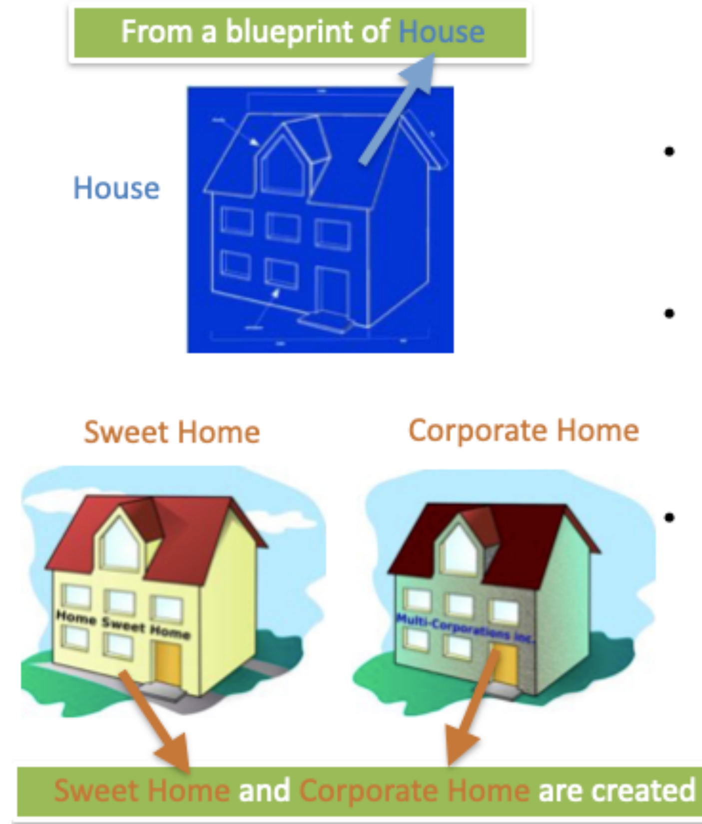


- Create an instance `bmw_obj` (an object) from the `Car` class template
- Create an instance `mercedes_obj` (an object) from the `Car` class template
- Create an instance `audi_obj` (an object) from the `Car` class template

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).

Agenda

- Defining a Class and Creating Objects from it

- Accessing Object Attributes and Methods
  - Attributes/fields (characteristics of an object)
    - **how to access and change those**
  - Methods (behavior)
    - `__init__()` and `__str__()` methods
    - **other methods**
    - **how to add a method in a class**

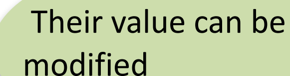
- Scope and Visibility

Accessing Object Attributes

- Recall that inside the class definition, the keyword `self` tells Python that the variable following the **dot(.)** references <u>a particular attribute/field</u>
  - Example: in the `Dog` class, `self.name, self.color, self.breed`

- Using similar notation, to access a attribute using the created instance, use **dot** notation
  - <u>Format:</u> `my_instance.attribute_name`

- Object attributes are **<u>mutable</u>**



Accessing Object Attributes

```python
# -------------- 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 + ", " +  self.breed + " )"
        return str_var


# ------------- creating object (instance of Class) -------
charlie_obj  = Dog("Charlie", "yellow", "golden retriever")
max_obj      = Dog("Max",     "brown",  "golden retriever")


# ------------- accessing object attributes ---------------
print("-----------------------------------")
print("chalie_obj.name:  ", charlie_obj.name)
print("chalie_obj.color: ", charlie_obj.color)
print("chalie_obj.breed: ", charlie_obj.breed)
```

Use . (Dot) after the object name

```
>>> %Run my_dog.my.py

  -----------------------------------
  chalie_obj.name:   Charlie
  chalie_obj.color:  yellow
  chalie_obj.breed:  golden retriever
```

Drake UNIVERSITY

Changing/Updating Object Attributes

```python
# ------------- 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 + ", " +  self.breed + " )"
        return str_var


# ------------- creating object (instance of Class) -------
charlie_obj  = Dog("Charlie", "yellow", "golden retriever")
max_obj      = Dog("Max",     "brown",  "golden retriever")

# ------------- changing object attributes ---------------
charlie_obj.name    = "Caramel"
charlie_obj.color   = "gray"

print("--------------------------------------")
print("Changing object attributes ...")
print("--------------------------------------")
print("chalie_obj.name:  ", charlie_obj.name)
print("chalie_obj.color: ", charlie_obj.color)
print("chalie_obj.breed: ", charlie_obj.breed)
```

Mutable — their values can be modified

```
>>> %Run my_dog.my.py

 --------------------------------------
 Changing object attributes ...
 --------------------------------------
 chalie_obj.name:   Caramel
 chalie_obj.color:  gray
 chalie_obj.breed:  golden retriever
```

Agenda

- Defining a Class and Creating Objects from it
- Accessing Object Attributes and Methods
  - Attributes/fields (characteristics of an object)
    - **how to access and change those**
  - Methods (behavior)
    - `__init__()` and `__str__()` methods
    - **other methods**
    - **how to add a method in a class**
- Scope and Visibility

Drake UNIVERSITY

Adding a New Method

Add another for 'age' of the Dog class

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

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

    # new method
    def update_age(self, year):

        self.age = self.age + year
```

Pass along the argument for 'age'

```
# ------------ creating object (instance of Class) -------
charlie_obj  = Dog("Charlie", "yellow", "golden retriever", 3)
max_obj      = Dog("Max",     "brown",  "golden retriever", 4)
```

Drake UNIVERSITY

Calling a Method from an Object

- To call any of method of an object, use **dot** notation
  - `my_instance.method(arg1, arg2)`
  - the `self` parameter references the *specific object/instance* of the class, the method will affect this *object/instance*
  - reference to `self` is passed automatically

Accessing Object Methods

- To call any of methods using the created instance, use **dot** notation
  - `my_instance.method(arg1, arg2)`

```
# ------------ creating object (instance of Class) -------
charlie_obj  = Dog("Charlie", "yellow", "golden retriever", 3)
max_obj      = Dog("Max",      "brown",  "golden retriever", 4)



# ------------ accessing object attributes --------------
print("------------------------------------")
print("chalie_obj.name:  ", charlie_obj.name)
print("chalie_obj.color: ", charlie_obj.color)
print("chalie_obj.breed: ", charlie_obj.breed)
print("chalie_obj.age:   ", charlie_obj.age)

charlie_obj.update_age(2)
print("chalie_obj.age: ", charlie_obj.age)
```

Use . (Dot) after the object name

```
>>> %Run my_dog.my.py

  --------------------------------------
  chalie_obj.name:   Charlie
  chalie_obj.color:  yellow
  chalie_obj.breed:  golden retriever
  chalie_obj.age:    3
  chalie_obj.age:  5
```

Exercise: Create a Point Class

- A Point is a 2 dimensional coordinate pair expressed by `(x, y)`
  - <u>Attributes/fields:</u>
    - **`x`** `and` **`y`** values in 2D coordinate system
    - Optional attributes: **`color`**, **`radius`**
    - Assuming that we want to draw a circle centered at (`x, y`)
  - <u>Methods:</u>
    - def `shift(self, x_offset, y_offset):`
      - takes two additional parameters besides `self`
        - `x_offset`
        - `y_offset`
    - This method should change the location (`x, y`) of a point object by shifting it to a new location (`x+x_offset, y+y_offset`)

Exercise: Add Another Method

- Add another method:
  - reset_to_origin(): that can change the location `(x, y)` of a point to location `(0, 0)`

Exercise: Creating Objects of Point Class

- Create two objects of Point class
  - The first object should be at the location (1, 2) and
  - the second one at (3, 4).
  - For each object, feel free to pick any value of your choice for the remaining attributes – color and radius.

- Print the two objects of the Point class

Exercise: Creating Objects of Point Class

- You should change the location of the first point from (1, 2) to (11, 12) using an appropriate method

- Similarly, change the location of the second point to (103, 104).

- To verify the correctness, show the updated objects using print()

Demo: Point Class