ENGR 1330: Computational Thinking with Data Science

Lesson 6: Class, Objects, and File Handling

Dinesh S. Devarajan
Whitacre College of Engineering
Texas Tech University

Topic Outline

- Class and Objects in Python
- File Handling in Python

Objectives

- To understand the use of classes and objects to do effective coding in Python

- To understand the basic idea of how to manipulate the data in a file using file handling options in Python

Computational Thinking Concepts

Class and Objects ⇨ Decomposition, Abstraction

File handling ⇨ Decomposition

Class and Objects in Python

Object-Oriented Programming

- What is Object-Oriented Programming (OOP)?

  - ✓ Useful paradigm where classes define concepts and objects are instance of classes

  - ✓ Way of thinking and implementing code

Object-Oriented Programming

- How would you describe an apple to a person?
  - ✓ It is a fruit
  - ✓ It has color and flavor
- How would you describe an apple to a computer?
  - ✓ OOP comes in handy to communicate with computers

Object-Oriented Programming

- How would you describe an apple to a computer?
  - ✓ Define a class called ‘Apple’ that contains the characteristics of an apple
  - ✓ Define an instance of that ‘Apple’ class called an object
- You can create many instances and hence, many objects for the ‘Apple’ class

Object-Oriented Programming

- Think of class as a blueprint to build a house
- You can build many houses (objects) using a single blueprint (class)

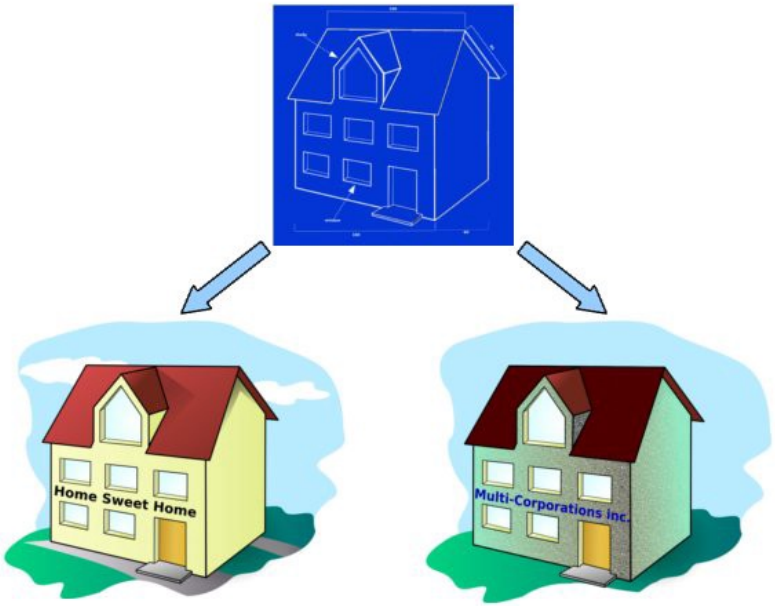


Figure Source: https://medium.com/@trulymhvu/everything-is-an-object-in-python-29d3aae8de5

Object-Oriented Programming

- Core concept: Attributes and methods

- Attributes: Characteristics associated to a type
  - ✓ E.g. color and flavor of an apple

- Methods: Functions associated to a type
  - ✓ E.g. cutting an apple into 4 slices

Object-Oriented Programming

- A more relevant example: Accessing a file that contains data

- Attributes: Characteristics associated to a type
  - ✓ E.g. file name, size, and creation date

- Methods: Functions associated to a type
  - ✓ E.g. reading and modifying the data in a file

Built-In Classes and Objects

- Guess what?....
  - ✓ Numbers, strings, lists, and dictionaries are all objects in Python
  - ✓ Each of them was an instance of a class

```
In [3]: print(type(0))
        <class 'int'>
```
class ‘int’

```
In [9]: print(type(""))
        <class 'str'>
```
class ‘str’

(Demo)

```
In [13]: print(type([1, 2, 3, 4]))
         <class 'list'>
```
class ‘list’

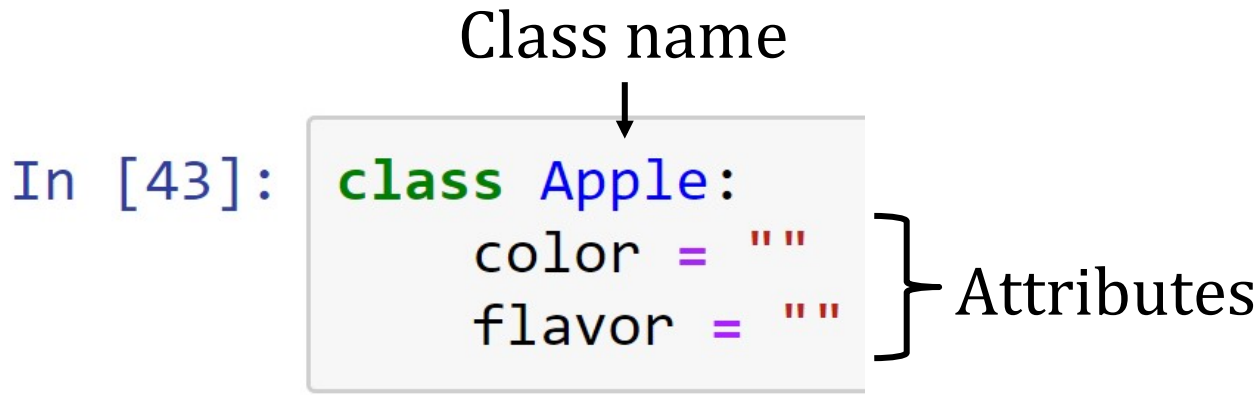
In-Built Classes and Objects

- dir(“ ”): To display all the methods associated with the string class
  - ✓ upper(): Creates an uppercase version of a string
  - ✓ count(): Counts the number of occurrences of a substring
- help(“ ”): Tells us how to use the methods associated with the string class

User-Defined Classes

- We have been using in-built classes and objects so far
  - ✓ We will now define our own classes and objects

- Creating a class named 'Apple' with attributes color and flavor



(Demo)

User-Defined Objects

- Creating objects (new instances) for the 'Apple' class

Object name

```
In [35]: gala = Apple()
         gala.color = "red-yellow"
         gala.flavor = "sweet"
```

Assigning attributes

Object name

```
In [40]: cripps = Apple()
         cripps.color = "pinkish-red"
         cripps.flavor = "sweet-tart"
```

Assigning attributes

(Demo)

Methods

- Methods: Functions that operate on the attributes of a specific instance of a class

Parameter: represents the instance that the method is being executed on

```
In [2]: class Dog:
            def sound(self):
                print("woof! woof!")

        fudge = Dog()
        fudge.sound()

        maple = Dog()
        maple.sound()
```

Method

(Demo)

Instance Variables

- Instance variables: Variables that have different values for different instances of the same class

Instance variable

```
In [74]: class Dog:
             name = ""
             def sound(self):
                 print("woof! I am {}! woof!".format(self.name))

         fudge = Dog()
         fudge.name = "Fudge"
         fudge.sound()

         maple = Dog()
         maple.name = "Maple"
         maple.sound()
```

(Demo)

Instance Variables

- Methods can also be used to do mathematical operations to return values

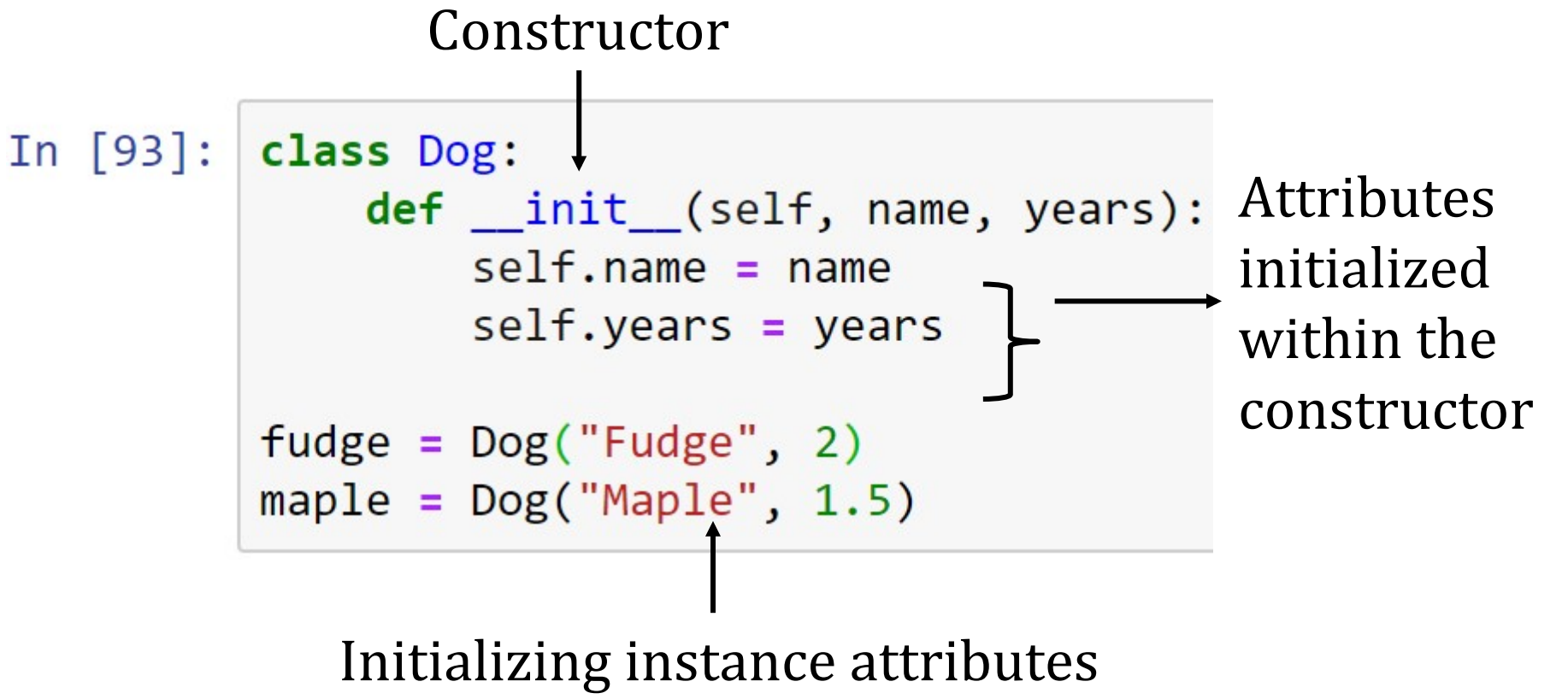
Instance variable

```
In [75]: class Dog:
             years = 0
             def dog_years(self):
                 return self.years*9

         fudge = Dog()
         fudge.years = 2
         print(fudge.dog_years())

         maple = Dog()
         maple.years = 1.5
         print(maple.dog_years())
```

(Demo)

Special Methods: Constructors

- Constructors: Used to initialize instance attributes when an object is created



(Demo)

Discussion Exercise

- Can you now write a class such that the dog can say its name and age (in dog years) using constructors?

In [74]:
```
class Dog:
    name = ""
    def sound(self):
        print("woof! I am {}! woof!".format(self.name))

fudge = Dog()
fudge.name = "Fudge"
fudge.sound()

maple = Dog()
maple.name = "Maple"
maple.sound()
```

In [75]:
```
class Dog:
    years = 0
    def dog_years(self):
        return self.years*9

fudge = Dog()
fudge.years = 2
print(fudge.dog_years())

maple = Dog()
maple.years = 1.5
print(maple.dog_years())
```

In [93]:
```
class Dog:
    def __init__(self, name, years):
        self.name = name
        self.years = years

fudge = Dog("Fudge", 2)
maple = Dog("Maple", 1.5)
```

Discussion Exercise

- Solution:

In [120]:
```
class Dog:
    def __init__(self, name, years):
        self.name = name
        self.years = years
        self.dog_age = years*9

    def sound(self):
        print("woof! I am {} and I am {} dog years old! woof!".format(self.name, self.dog_age))

fudge = Dog("Fudge", 2)
maple = Dog("Maple", 1.5)
fudge.sound()
maple.sound()
```

(Demo)

Docstrings

- Docstrings: A brief comment that explains the purpose of the class and the methods used inside the class
- Docstrings are typed between triple quotes

In [131]:
```
class Dog:
    """This class enables the dog to say its name and age in dog years"""
    def __init__(self, name, years):
        """This function contains all the necessary attributes"""
        self.name = name
        self.years = years
        self.dog_age = years*9

    def sound(self):
        """This function enables the dog to speak"""
        print("woof! I am {} and I am {} dog years old! woof!".format(self.name, self.dog_age))

fudge = Dog("Fudge", 2)
maple = Dog("Maple", 1.5)
fudge.sound()
maple.sound()
```

represents docstrings

(Demo)

Docstrings

- Docstrings are useful for others to understand your code easily

- Using help(Class name) displays the docstrings that explains the user-defined classes and methods

```
In [132]: help(Dog)
```

(Demo)

File Handling in Python

File Handling

- open( ) function in Python is useful to work with files

- Different modes to open a file:
  - ✓ “r” – opens a file for reading
  - ✓ “w” – opens a file for writing
  - ✓ “a” – opens a file for appending
  - ✓ “x” – creates a specified file

Reading a File

- Reading a file named ‘sample.txt’

```
In [153]: sample_file = open("sample.txt", "r")
```

File object — File name — Mode

- Printing the contents of the file named ‘sample.txt’ using read( ) function

```
In [154]: print(sample_file.read())
```

(Demo)

Appending a File

- Appending a file named ‘sample.txt’

```
In [198]: sample_file = open("sample.txt", "a")
```

File object — File name — Mode

- Appending text using the write( ) function

```
In [199]: sample_file.write("\nMy hobbies are dancing and playing tennis")
```

(Demo)

Summary

- Concepts of class and objects in Python are covered

- Concepts of basic file handling modes in Python are covered