Data Structures in Python Chapter 1

1. Introduction - Review Python
2. Objects and References
3. **Object-Oriented Programming**
4. OOP - Fraction Example
5. OOP - Classes
6. Exceptions 1, 2
7. JSON

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204 Handong Global University

내 아들들을 먼 곳에서 이끌며 내 딸들을 땅 끝에서 오게 하며 내 이름으로 불려지는 모든 자 곧 내가 내 영광을 위하여 창조한 자를 오게 하라 그를 내가 지었고 그를 내가 만들었노라 (사43:6-7)

그런즉 너희가 먹든지 마시든지 무엇을 하든지 다 하나님의 영광을 위하여 하라 (고전10:31)

Agenda

- Topics:
  - Objects - State and Behavior
  - Classes
  - Constructors
  - Methods & Self
  - Point class
  - Saving a class file and the module Geometry.py
  - Data Field Encapsulation

- References:
  - Problem Solving with Algorithms and Data Structures using Python
    - Chapter 1.13 Object-Oriented Programming in Python

Exercise

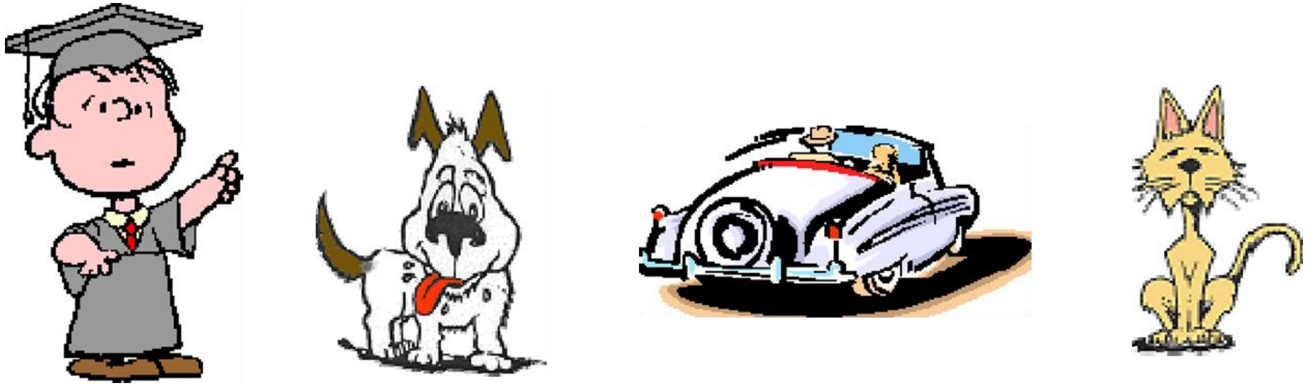
- What is the output of the following code fragment?

```
x = ['a', 'b', 'c']
y = x
z = ['a', 'b', 'c']
print (x == y)
print (x is y)
print (x == z)
print (x is z)
```

```
x = 'Hello'
y = x
z = 'Hello'
print (x == y)
print (x is y)
print (x == z)
print (x is z)
```

Object Oriented Programming(OOP)

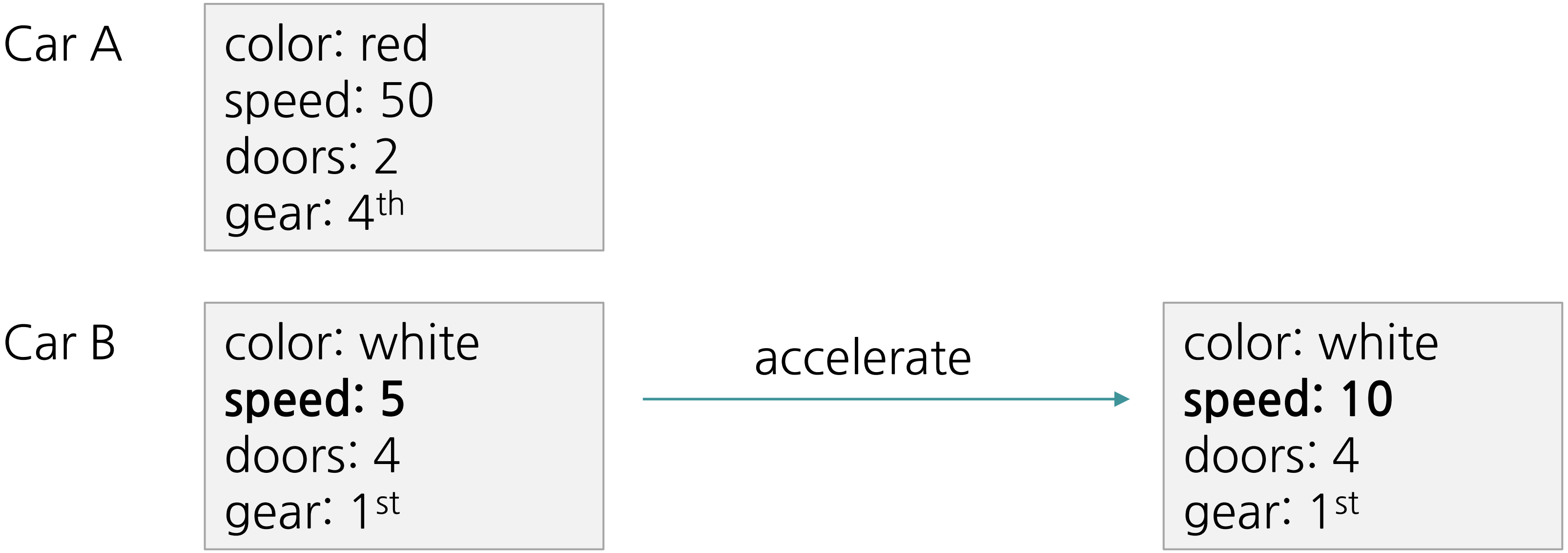
- An **object** represents an entity in the real world that can be distinctly identified, e.g., students, dogs, cars, cats, books.



- Object Oriented Programming(OOP) involves the use of objects to create programs.

Objects

- Cars may have:
  - **information:** color, current speed, current gear, etc.
  - **function:** accelerate, brake, change gear, reverse, etc.



Car A

color: red
speed: 50
doors: 2
gear: 4th

Car B

color: white
speed: 5
doors: 4
gear: 1st

accelerate

color: white
speed: 10
doors: 4
gear: 1st

Object State and Behavior

- Every real world object has:
  - State — information that the object stores.
  - Behavior — functionality of the object, i.e., what the object can do.

- Example:
  - Consider a system managing university students.
  - A student object has:
    - State — id, name, age, contact number, address, stage, grade, completed courses, current courses, advisor, faculty, …
    - Behavior — enroll in a new course, change contact number, change enrollment, choose degree, …
  - A person object has:
    - State - id, name, age, contact number, address, …
    - Behavior - eat, drink, wear, talk, work, meet, swim, run, drive, …

Object is state + behavior

- A software object's state is represented by its variables, called **data fields.**
- A software object implements its behavior with **methods.**
  - Every object is a bundle of variables and related methods.
  - We make an object perform actions by invoking the methods on that object.
- Example:

```
    my_list = [ 1, 2, 3 ]
    my_list.reverse()
```

In a Program

- Our program consists of many different objects.
- **Two objects of the same kind would have the same set of behaviors, but independent state information.**
  - Two string objects store different words, but can perform same methods, e.g., lower(), split(), index(), etc.

- For an object in our program
  - State — is defined by variables (data fields).
  - Behaviors — is defined by methods (actions).

- The definition of a particular kind of objects is called a **class**. Once created, an object is **an instance of a class**.

Python Class

- A **class** is the structure we use to define a category of objects.
  It defines the **state and behavior** of a category of objects.
- A class **is a template or blueprint defining t**he date fields and actions (methods) that any instance (object) of that class can have.
- For an object in our program
  - State — is defined by variables (data fields).
  - Behaviors — is defined by methods (actions).
- Analogies for class and object:
  - Factory mold and products produced from that mold
  - Blueprint and apartment building units
  - Cookie cutter and cookies

Classes

- Python has a number of built-in classes
  - list, dict, set, int, float, boolean, str
- We can define our own classes
  - creates a new type of object in Python

```
class name_of_the_class:
    # definition of the class goes here
    # initializer
    # methods
```

- Classes consist of:
  - **state** variables (sometimes called instance variables)
  - **methods** (functions that are linked to a particular instance of the class)

Example

- An example:

```
class foo:
    a, b, c = 0, "ace", (1,2)
```

multiple assignments

- Instantiating Classes
  - A class is instantiated by calling the class object:

```
obj = foo()
print(obj.a)
print(obj.b)
print(obj.c)
```

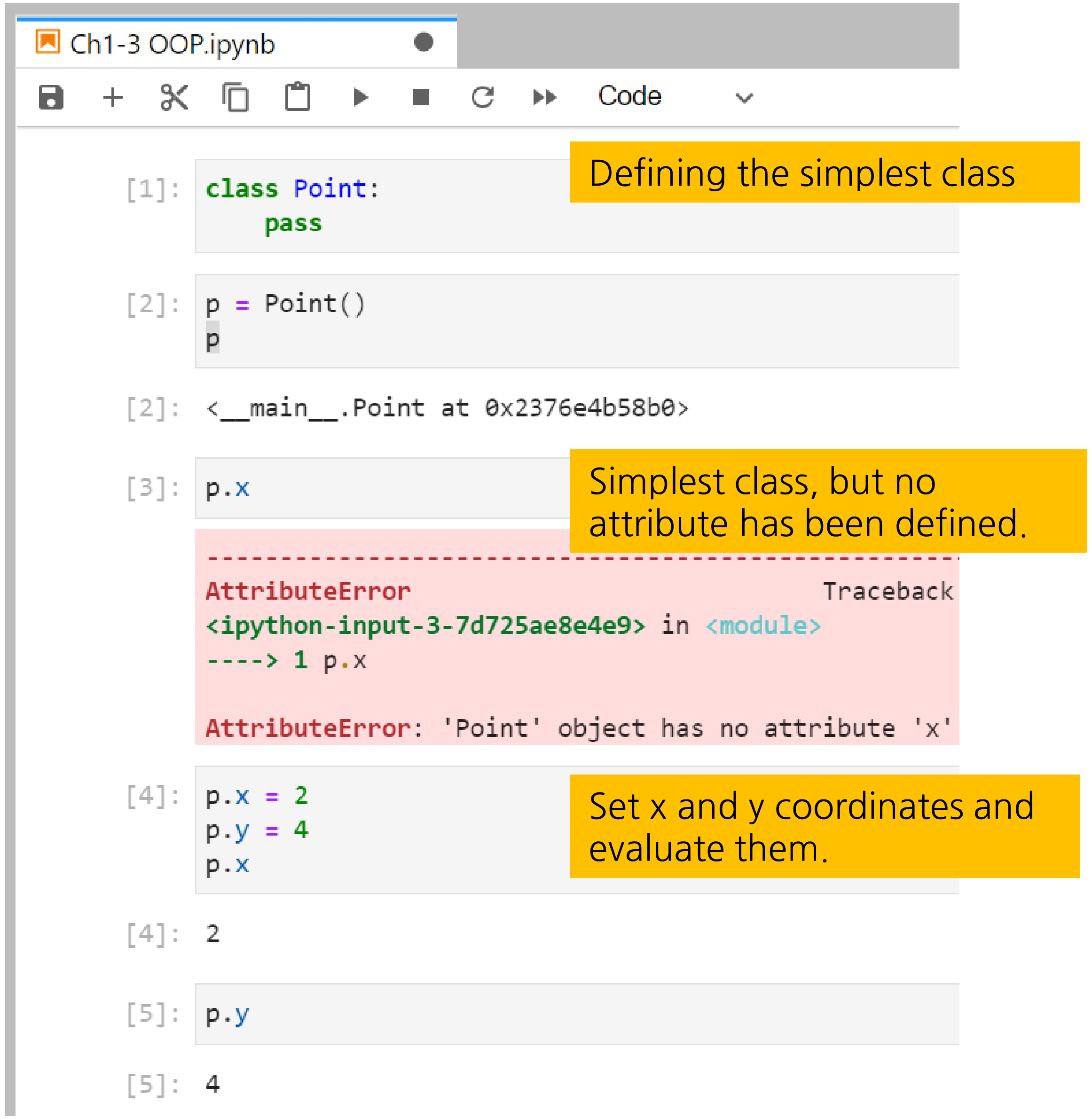
```
0
ace
(1,2)
```

The simplest class possible

- A simple example:

```
class Point:
    pass
```

  - “pass” is a statement that does nothing. It is often used as a placeholder when developing code



```
[1]: class Point:
         pass
```

Defining the simplest class

```
[2]: p = Point()
     p

[2]: <__main__.Point at 0x2376e4b58b0>
```

```
[3]: p.x

---------------------------------------------------------------------------
AttributeError                            Traceback
<ipython-input-3-7d725ae8e4e9> in <module>
----> 1 p.x

AttributeError: 'Point' object has no attribute 'x'
```

Simplest class, but no attribute has been defined.

```
[4]: p.x = 2
     p.y = 4
     p.x

[4]: 2
```

Set x and y coordinates and evaluate them.

```
[5]: p.y

[5]: 4
```

Constructors

- Each class should contain a **constructor** method
  - Name of the method is __init__
  - The method always has at least one parameter, called self
  - Self is a reference to the object that we are creating
  - The constructor can have other parameters

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

It creates an object in the memory for the class.

```
p = Point(5,7)
```

Accessing Objects

- After an object is created, you can access its data fields and invoke its methods using the dot operator (.), also known as the **object member access operator**.
  - For example, the following code accesses the x, y coordinates

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

```
p = Point(5,7)
print(p.x)
print(p.y)
```

Adding functionality

- Defining more methods
  - A method to shift a point by a given amount in horizontal and vertical directions

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def translate(self, dx, dy):
        self.x += dx
        self.y += dy
```

- Note: the method is named normally, but has the additional parameter (**self**) as the first parameter
  - All methods that are called on an instance of an object need the self parameter

Why “self’?

- Note that the first parameter is special. It is used in the implementation of the method, but not used when the method is called. So, what is this parameter self for? Why does Python need it?
- **self** is a parameter that represents **an object**.
  - Using self, you can access instance variables in an object. Instance variables are for storing data fields.
  - Each object is an instance of a class.
  - Instance variables are tied to specific objects.
  - Each object has its own instance variables. You can use the syntax self.x to access the **instance variable x** for the object self in a method.

Using the Point class

- Methods are defined to accept self as the first parameter

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def translate(self, dx, dy):
        self.x += dx
        self.y += dy
```

- We call the method using:

    object_name.method(params)

```
p = Point(0,0)
p.translate(3,4)
...
```

This call is equivalent with
Point.translate(p, 3, 4)

Exercise 1

- Write a method named halfway(target) which takes a Point as an argument and returns the halfway point between itself and the parameter Point.
- Sample Run:

```
p = Point(3, 4)
q = Point(5, 12)
r = p.halfway(q)
print(r.x, r.y)    #4.0 8.0
```

Exercise 2

- Write a method named **midpoint** which takes two Points as arguments and returns the middle point between them. Define this method as a part of Point class even though it is possible to exist outside of the class.
- Sample Run:

```
p = Point(3, 4)
q = Point(5, 12)
r = Point.midpoint(p, q)
print(r.x, r.y)    #4.0 8.0
```

Compare …

- Now, compare the midpoint() function and the halfway method
  - Midpoint takes two parameters but halfway takes one

```
p = Point(3, 4)
q = Point(5, 12)
r = Point.midpoint(p, q)
print(r.x, r.y)
```

```
p = Point(3, 4)
q = Point(5, 12)
r = p.halfway(q)
print(r.x, r.y)    #4.0 8.0
```

Exercise 3

- Write a method named reflect_x() which **returns** a new Point, one which is the reflection of the point about the x-axis.
  For example, Point(3, 4).reflect_x() is (3, -4)

Exercise 4

- Write a method named slope_to_origin() which returns the slope of the line joining the origin to the point.
- For example, Point(4,10).slope_to_origin() returns 2.5
  - The slope between two points is $a = \frac{y_2 - y_1}{x_2 - x_1}$

```
p = Point(4, 10)
print(p.slope_to_origin())    #2.5
```

```
print(Point(4, 10).slope_to_origin())    #2.5
```

How to save the code to a file in Jupyter-Lab or Notebook

- Use cell magic commands to write and read.
  - To create or overwrite: `%%writefile filename.py`
  - To append to an existing file: `%%writefile -a filename.py`
  - To load a file into a cell: `%load filename.py`

  - To import all classes in a file (or module): `import filename`
  - To import a class in a file(or module): `from filename import classname`

Place these cell magic commands at the first line of the code cell.

```
%%writefile Geometry.py

class Point:
    def __init__(self, x, y):
    ''' Constructs and initializes a point at x, y'''
        self.x = x
        self.y = y
    ...
```

```
from Geometry import Point

p = Point(5,7)
```

Saving the class

- Classes are designed to help build modular code
  - Can be defined within a module that also contains application code
  - Multiple classes can be defined in the same file.
- In this course, we will typically store each class in their own module
  - To use the class in another module, you will need to import the module.

Saved in a file called Geometry.py

```
class Point:
    def __init__(self, x, y):
    ''' Constructs and initializes a point at x, y '''
        self.x = x
        self.y = y

    def translate(self, dx, dy):
    ''' Translates this point, at x, y
        by dx, dy along the x, y axis. '''
        self.x += dx
        self.y += dy
```

```
from Geometry import Point

p = Point(5,7)
```

Exercise 5

- Define a class that will be used to represent a square with a given side length in Geometry.py.
  - Use cell magic command to append: `%%writefile -a Geometry.py`
  - Your class should include a constructor that will allow the square to be used as follows:

```
from Geometry import Square

side = 10
s = Square(side)
```

  - Add a method to the class to calculate the perimeter of the square. The following code shows how the method may be used.

```
print(s.perimeter())
```

```
40
```

Three different ways of importing a module

- There are a few different ways of importing a module. For example,

- Case 1: **`from Geometry import Point`**
  - This imports a specific class **`Point`** in **`Geometry.py`** module.
    You may use it directly, for example,
    `p = Point(1, 2)`
- Case 2: **`import Geometry`**
  - This imports all things defined in Geometry.py module.
  - You must specify the name of the module to use it such as
    `p = Geometry.Point(1, 2)`
- Case 3: **`import Geometry as gm`**
  - It just simplify the way of calling the module:
    `p = gm.Point(1, 2)`

Data Field Encapsulation

- To protect data.
- To make class easy to maintain.
- To prevent direct modifications of data fields, don't let the client directly access data fields.
- This is known as data field **encapsulation**.
  - This can be done by defining private data fields. In %thon, the private data fields are defined with **two leading underscores.**
  - You can also define a private method named with two leading underscores.

Example: Circle

- __radius : No direct access outside the Circle class

```
class Circle:
    def __init__(self, r):
        self.__radius = r

    def radius(self):
        return self.__radius
    ...
```

```
from Geometry import Circle

c = Circle(5)
print(c.__radius)
```

```
AttributeError:
'Circle' object has no attribute '__radius
```

```
print(c.radius())
```

```
5
```

- If a class is designed for other programs to use, to prevent data from being tampered with and to make the class easy to maintain, **define data fields private.**

Data Structures in Python
Chapter 1

1. Introduction - Review Python
2. Modelling Objects
3. **Object-Oriented Programming**
4. OOP - Fraction Example
5. OOP - Classes
6. Exceptions 1, 2
7. JSON