Object Oriented Programming in Python

Lecture 8 & 9

Background

- Object Oriented Programming (OOP) allows decomposition of a problem into a number of units called objects and then build the data and functions around these objects. It emphasis more on the data than procedure or functions

Comparison Between Procedural & Object Oriented Programming

	Procedural Oriented Programming	Object Oriented Programming
Based On	In Pop, entire focus is on data and functions	Oops is based on a real world scenarios. Whole program is divided into small parts called object
Reusability	Limited Code reuse	Code reuse
Approach	Top down approach	Object focused Design
Access specifiers	Not any	Public, Private and Protected
Data movement	Data can move freely from functions to function in the system	In oops, objects can move and communicate with each other through member functions
Data Access	In pop, most function uses global data for sharing that can be accessed freely from function to function in the system	In oops, data cannot move freely from method to method, it can be kept in public or private so we can control the access of data.
Data Hiding	In pop, so specific way to hide data, so little bit less secure	It provides data hiding, so much more secure.
Overloading	Not possible	Function and Operator Overloading
Example-Languages	C, VB, Fortran, Pascal	C++, Python, Java, C#
Abstraction	Uses abstraction at procedure level	Uses abstraction at class and object level

What do objects consist of?

- An object in programming is an entity or “variable” that has two entities attached to it: data and things that act on that data. The data are called **attributes** of the object, and the functions attached to the object that can act on the data are called **methods** of the object

Principles of OOP

- Encapsulation
- Information Hiding
- Abstraction
- Inheritance

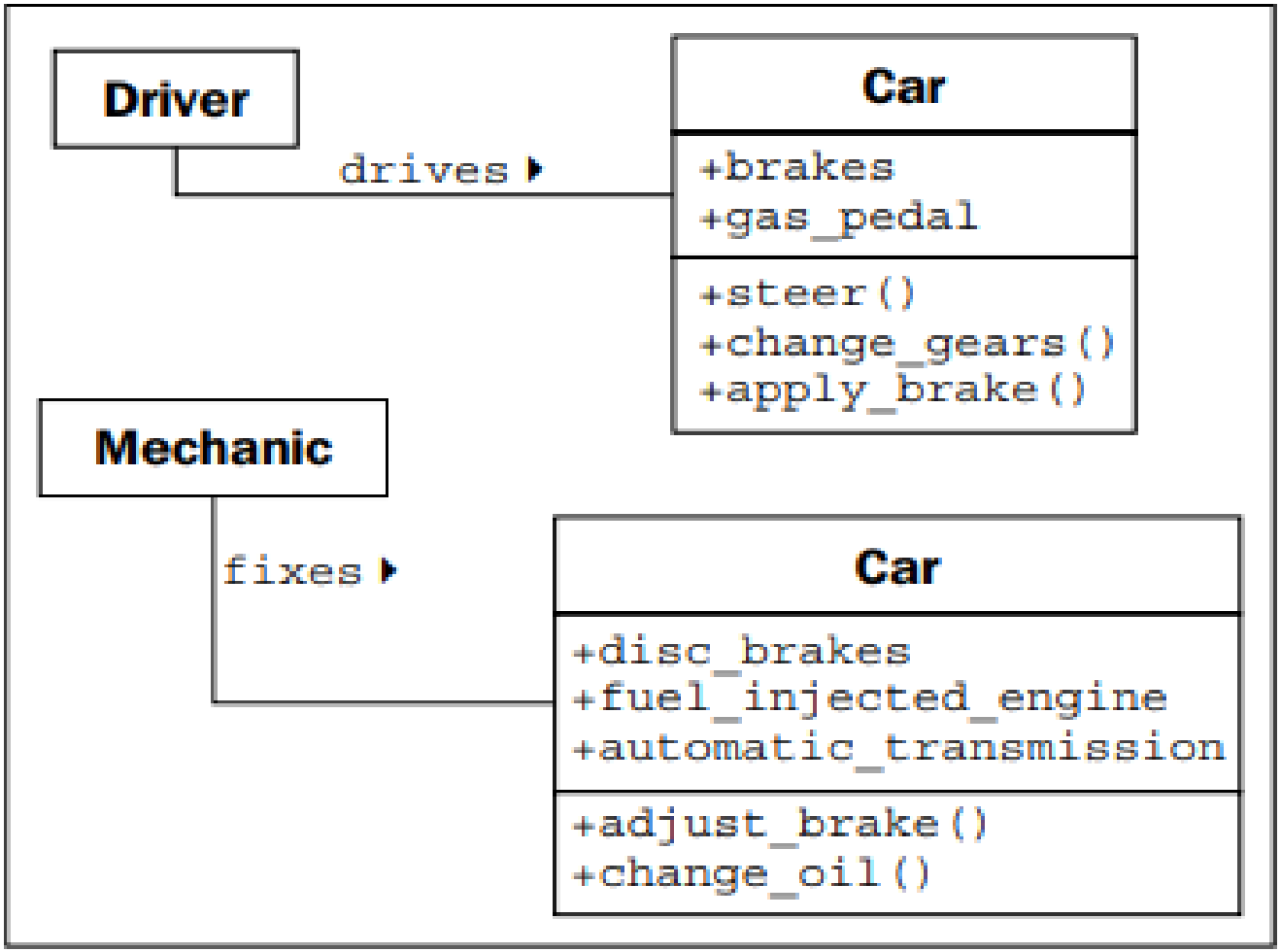
Information Hiding & Encapsulation

- This process of hiding the implementation, or functional details, of an object is suitably called information hiding.
- It is also sometimes referred to as encapsulation, but encapsulation is actually a more all-encompassing term

Abstraction

- Abstraction is another object-oriented concept related to encapsulation and information hiding.
- Simply put, abstraction means dealing with the level of detail that is most appropriate to a given task.
- It is the process of extracting a public interface from the inner details.
- Example: A driver of a car needs to interact with steering, gas pedal, and brakes. The workings of the motor, drive train, and brake subsystem don't matter to the driver. A mechanic, on the other hand, works at a different level of abstraction, tuning the engine and bleeding the breaks. Here's an example of two abstraction levels for a car:

Example: Abstraction



Composition

- Composition is the act of collecting several objects together to create a new one. Composition is usually a good choice when one object is part of another object.
- Example: A car is composed of an engine, transmission, starter, headlights, and windshield, among numerous other parts. The engine, in turn, is composed of pistons, a crank shaft, and valves. In this example, composition is a good way to provide levels of abstraction

Aggregation

- Aggregation is almost exactly like composition. The difference is that aggregate objects can exist independently. It would be impossible for a position to be associated with a different chess board, so we say the board is composed of positions. But the pieces, which might exist independently of the chess set, are said to be in an aggregate relationship with that set.

Inheritance

- Inheritance, also called generalization, allows us to capture a hierarchal relationship between classes and objects. For instance, a ‘fruit’ is a generalization of ‘orange’. Inheritance is very useful from a code reuse perspective.

Polymorphism

- Polymorphism is the ability to treat a class differently depending on which subclass is implemented.
- Poly-morphism means many forms. That is, a thing or action is present in different forms or ways. One good example of polymorphism is constructor overloading in classes.

How Python Object Works

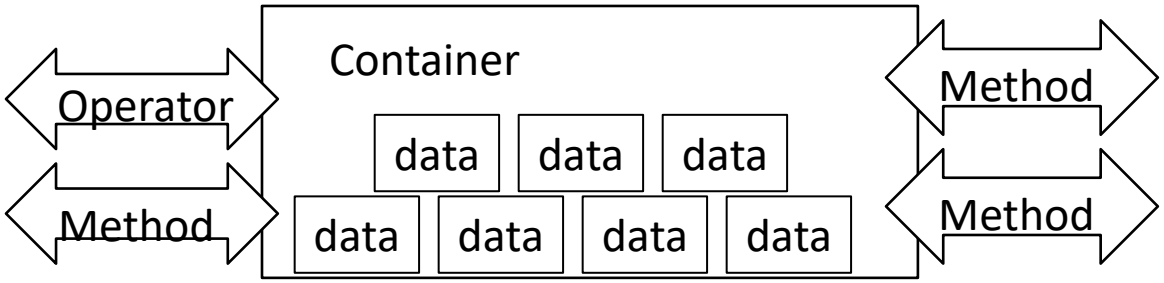
- In OOP, the specific realizations are called object instances, while the common pattern is called a class. In Python, this common pattern or template is defined by the class statement.
- So, in summary, objects are made up of attributes and methods, the structure of a common pattern for a set of objects is called its class, and specific realizations of that pattern are called “instances of that class.”s

Example of how objects work: Strings

- Python strings (like nearly everything else in Python) are objects. Thus, built into Python, there (implicitly) is a class definition of the string class, and every time you create a string, you are using that definition as your template. That template defines both attributes and methods for all string objects, so whatever string you've created, you have that set of data and functions attached to your string which you can use

What Is a Class?

- A class is a data type that allows for data storage and a set of methods to access the data.
- Recall the diagram of a container or a list in the last module:



When we use a list, we can store data in it and we can call methods such as append, sort to do work with the data.

- A container is a specific type of class, therefore the diagram of a class is the same as above, with data storage and methods.
- The Python built-in data types that we've been using: int, str, list, etc. are each a specific type of class.
- In fact, every data type in Python is a class, and Python is called an object oriented language.

Advantage of Classes

- It is good to have class (both in real life and in programming).
- There are 2 main advantages to using classes:
  1. Classes allow us to model real life entities, which makes the design of the software more intuitive.
  2. Classes can be re-used, which makes software development faster because we can take advantage of existing classes.
- In the next slides we will go over each of the advantages in more detail.

Modeling with Classes (1 of 2)

Advantage 1: Classes allow us to model real life entities, which makes the design of the software more intuitive.

- In real life, entities such as a school, a car, a basketball player, a dog, a movie… all have specific behaviors. We expect a dog to bark and a basketball player to shoot hoops.
- Because a class allows for both data and the methods to access the data, the methods effectively control the behavior of a class. A list class has a method to sort data, but a set class does not have a sort method because data in a set are not in any particular order. By not having the sort method, we enforce the unordered data behavior of a set.
- When we create our own class, let's say BankAccount, we will have a withdraw() and a deposit() methods, because these are behavior of a bank account.

Modeling with Classes (2 of 2)

- Having data mimic or model real life entities makes it easier to design the software.
- If we want to write software for a bank, it is intuitive to think of a Bank class with methods such as set_interest_rate(), advertise(), maintain_website(), invest(), etc.
  The Bank class can contain other classes such as CheckingAccount class, SavingsAccount class, SafeDepositBox class, etc. And each of these classes will have its own methods or behavior.
  The design of the bank software becomes more “natural” when we have classes that model real life entities of a bank.

Class Re-Use (1 of 2)

Advantage 2: Classes can be re-used, which makes software development faster because we can take advantage of existing classes.

- Some general purpose classes, such as the list class, can be useful in many situations.
- Someone wrote the list class, and the rest of us can use it without having to write the code for it. This makes our own coding effort shorter.
- Python supports class re-use by providing many software modules. Each module is a set of classes that do work in a specific area. For example, we've used the Turtle module for graphics output, without having to write a single line of graphics code.

Class Re-Use (2 of 2)

- Python also encourages class re-use by supporting inheritance.
- Working with inheritance is beyond the scope of CIS 40, but it's good to have an overview of this important concept of object oriented programming or *OOP*.
- When an existing class is a general format of the class that we want to create, *inheritance* lets us re-use the existing class.
- Example: A Cat class is already written with cat-type methods or behavior. We want to create a Manx class to represent the Manx breed of cats.

  When we create the Manx class, the class *inherits* from the Cat class, which means that it automatically contains all the methods of the Cat class. We don't need to re-write the meow() method or the scratch() method. We only need to write the methods that are specialized to the Manx breed, such as the ~~tail()~~ tail = false method.

Classes and Objects (1 of 2)

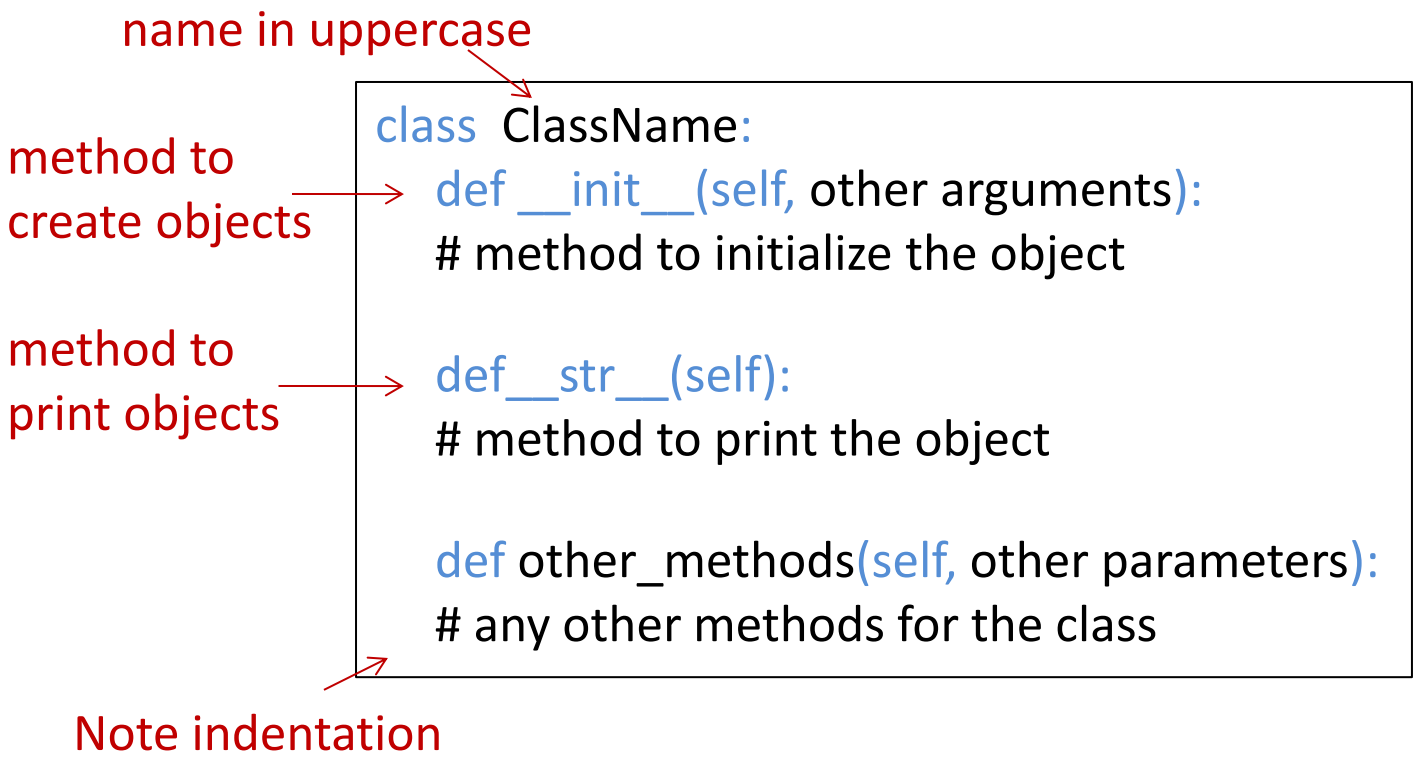
- So far we know that a class:
  - Allows for data storage and methods to access data, which specify the behavior of the class
  - Provides 2 main advantages for software development
- The third part of a class definition is: A class is a data type.
- The BankAccount class is a data type, just like int is a data type.
- When we want to store the value 5 in our code, we need to store it in a memory space that has the data type int.
- Likewise, when we want to store bank account data in our code, we need to store it in a memory space that has the data type BankAccount.
- The memory space that has a class data type is called an *object*.

Classes and Objects (2 of 2)

- A class is a description of the data type and its behavior.
- An object is the actual memory space that stores data. An object is an *instance* of a class.
- From one class (or one description), many objects of that class can be created in memory to store data.
- Here are two common analogies to illustrate the relationship between a class and an object:
  - A class is like an architect's blue print of a house, a design on paper. The house is the object, it is built from the design. Many house "instances" can be built from one blue print.
  - A class is like a blank tax form provided by the IRS. When a person fills in the tax form to submit it, the completed form is an object. Many people can use the same tax form and fill it with their own data, creating many "objects" of the same tax form "data type."

Defining a Class (1 of 3)

- We are not limited to using classes that are built-in to the Python language. We can create or *define* our own class.
- Format for defining a class:



- Every method has self as the first parameter. self specifies the

Defining a Class (2 of 3)

- The __init__ method:

```
def __init__(self, other arguments):
    # method to initialize the object
```

  - The name starts and ends with 2 underscores (_).
  - When an object is instantiated or created, the __init__ method runs to initialize the object with data.
  - The data can be a default value or it can be passed in through the arguments.

- The __str__ method:

```
def__str__(self):
 # method to print the object
```

  - The name starts and ends with 2 underscores.
  - When the function print is used to print the object, the __str__ method runs to return a string of object data, which is printed to screen.
- The class typically has other methods to define its behavior.

Defining a Class (3 of 3)

- Example of a bank account class definition:

```
# Bank account class
class BankAcct():
    # initialize account number and balance
    def __init__(self, num, bal):
        self.num = num
        self.bal = bal

    # return string with object data to be printed
    def __str__(self):
        return "Account: " + str(self.num) + "\nBalance: " + str(self.bal)

    # return account number
    def getNumber(self):
        return self.num

    # return account balance
    def getBalance(self):
        return self.bal

    # allow a deposit
    def deposit(self, amt):
        self.bal = self.bal + amt
```

We must pass in the account number and balance when creating the object

The last 4 methods let us:

- Print the acct info as a string
- Get the acct number
- Get the acct balance
- Make a deposit

- Note that all object data variables start with: self.

Working With an Object (1 of 2)

- Before we instantiate or create an object from a class that we define, we must import the class. This is similar to how we import the Turtle class before we can use the turtle graphics.
- To instantiate an object:
  - Type the class name, which runs the __init__ method of the class. If the __init__ method has input parameters other than self, provide data for them.
  - The __init__ method returns the object, which needs to be assigned to a variable.
- To work with an object is similar to working with any of the Python objects: object_name.method_name(any parameters)

Working With an Object (2 of 2)

- Working with an example object of the BankAcct class:

```
# Bank account class
class BankAcct():
    # initialize account number a
    def __init__(self, num, bal):
        self.num = num
        self.bal = bal

    # return string with object c
    def __str__(self):
        return "Account: " + str(

    # return account number
    def getNumber(self):
        return self.num

    # return account balance
    def getBalance(self):
        return self.bal

    # allow a deposit
    def deposit(self, amt):
        self.bal = self.bal + amt
```

Code

```
# instantiate myAcct object
# with acct num 123 and $50
myAcct = BankAcct(123, 50)
# verify acct info
print(myAcct)
# deposit $100
myAcct.deposit(100)
# print acct balance
print("new balance is:", myAcct.getBalance() )
```

Output

```
Account: 123
Balance: 50
new balance is: 150
```

A Example of Python Class

```
class Person:

    def __init__(self,name):
        self.name = name

    def Sayhello(self):
        print 'Hello, my name is', self.name

    def __del__(self):
        print '%s says bye.' % self.name

A = Person('Yang Li')
```

This example includes
class definition, **constructor function**, **destructor function**, **attributes and methods definition** and **object definition.**
These definitions and uses will be introduced specifically in the following.