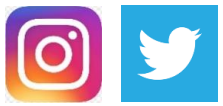
Computational Thinking with Programming

Lecture - 21

Introduction to Object Oriented Programming in Python

@cse_bennett @csebennett



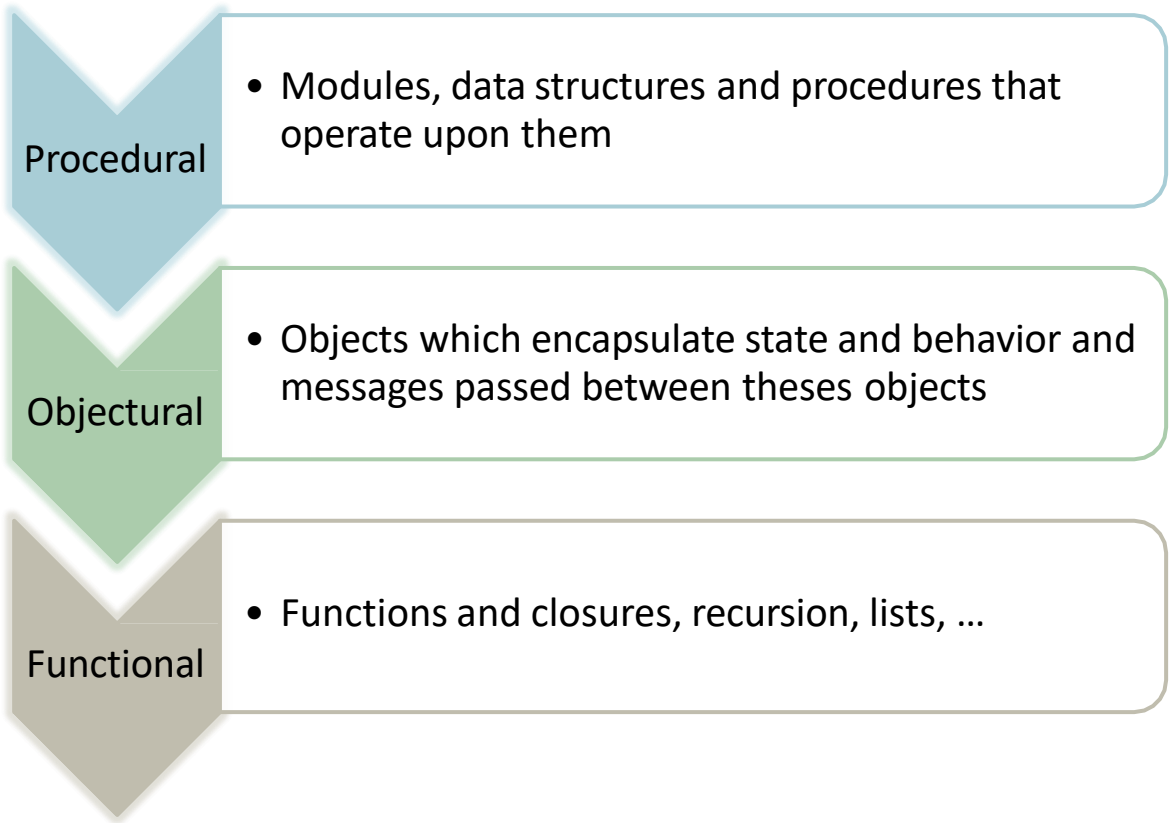
Today…

- Last Session:
  - Sets in Python.
- Today's Session:
  - Object Oriented Programming:
    - Fundamentals.
    - Oops in Python.
- Hands on Session with Jupyter Notebook:
  - We will practice on the objects programming in Jupyter Notebook.

Object Oriented Programming Basics

Programming Paradigms

Before diving deep into the concept of Object Oriented Programming, let's talk a little about all the programming paradigms which exist in this world.



Procedure Oriented Vs Object Oriented Programming

- Procedural programming creates a step by step program that guides the application through a sequence of instructions. Each instruction is executed in order.
- Procedural programming also focuses on the idea that all algorithms are executed with functions and data that the programmer has access to and is able to change.
- Object-Oriented programming is much more similar to the way the real world works; it is analogous to the human brain. Each program is made up of many entities called objects.
- Instead, a message must be sent requesting the data, just like people must ask one another for information; we cannot see inside each other's heads.

Summarized the differences

Procedure Oriented Programming	Object Oriented Progrmming
In POP, program is divided into small parts called **functions**	In OOP, program is divided into parts called **objects**.
POP follows **Top Down approach**	OOP follows **Bottom Up approach**
POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**
In POP, Overloading is not possible	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading
In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function, it can be kept public or private so we can control the access of data.
POP does not have any access specifier	OOP has access specifiers named Public, Private, Protected, etc.

Featuers of OOP

- Ability to simulate real-world event much more effectively
- Code is reusable thus less code may have to be written
- Data becomes active
- Better able to create GUI (graphical user interface) applications
- Programmers are able to produce faster, more accurate and better- written applications

Fundamental concepts of OOP in Python

The four major principles of object orientation are:

- Encapsulation
- Data Abstraction
- Inheritance
- Polymorphism

Definitions

- Class - a template
- Method or Message - A defined capability of a class
- Field or attribute- A bit of data in a class
- Object or Instance - A particular instance of a class

Terminology: Class

Defines the abstract characteristics of a thing (object), including the thing's characteristics (its attributes, fields or properties) and the thing's behaviors (the things it can do, or methods, operations or features). One might say that a class is a blueprint or factory that describes the nature of something. For example, the class Dog would consist of traits shared by all dogs, such as breed and fur color (characteristics), and the ability to bark and sit (behaviors).

http://en.wikipedia.org/wiki/Object-oriented_programming

Terminology: Instance

One can have an instance of a class or a particular object. The instance is the actual object created at runtime. In programmer jargon, the Lassie object is an instance of the Dog class. The set of values of the attributes of a particular object is called its state. The object consists of state and the behavior that's defined in the object's class.

Object and Instance are often used interchangeably.

http://en.wikipedia.org/wiki/Object-oriented_programming

Terminology: Method

An object's abilities. In language, methods are verbs. Lassie, being a Dog, has the ability to bark. So bark() is one of Lassie's methods. She may have other methods as well, for example sit() or eat() or walk() or save_timmy(). Within the program, using a method usually affects only one particular object; all Dogs can bark, but you need only one particular dog to do the barking

Method and Message are often used interchangeably.

http://en.wikipedia.org/wiki/Object-oriented_programming

What is an Object..?

- Objects are the basic run-time entities in an object-oriented system.
- They may represent a person, a place, a bank account, a table of data or any item that the program must handle.
- When a program is executed the objects interact by sending messages to one another.
- Objects have two components:
  - Data (i.e., attributes)
  - Behaviors (i.e., methods)

Object Attributes and Methods Example

Object Attributes

- Store the data for that object
- Example (taxi):
  - Driver
  - OnDuty
  - NumPassengers
  - Location

Object Methods

- Define the behaviors for the object
- Example (taxi):
  - PickUp
  - DropOff
  - GoOnDuty
  - GoOffDuty
  - GetDriver
  - SetDriver
  - GetNumPassengers

Some Python Objects

```
>>> x = 'abc'
>>> type(x)
<class 'str'>
>>> type(2.5)
<class 'float'>
>>> type(2)
<class 'int'>
>>> y = list()
>>> type(y)
<class 'list'>
>>> z = dict()
>>> type(z)
<class 'dict'>
```

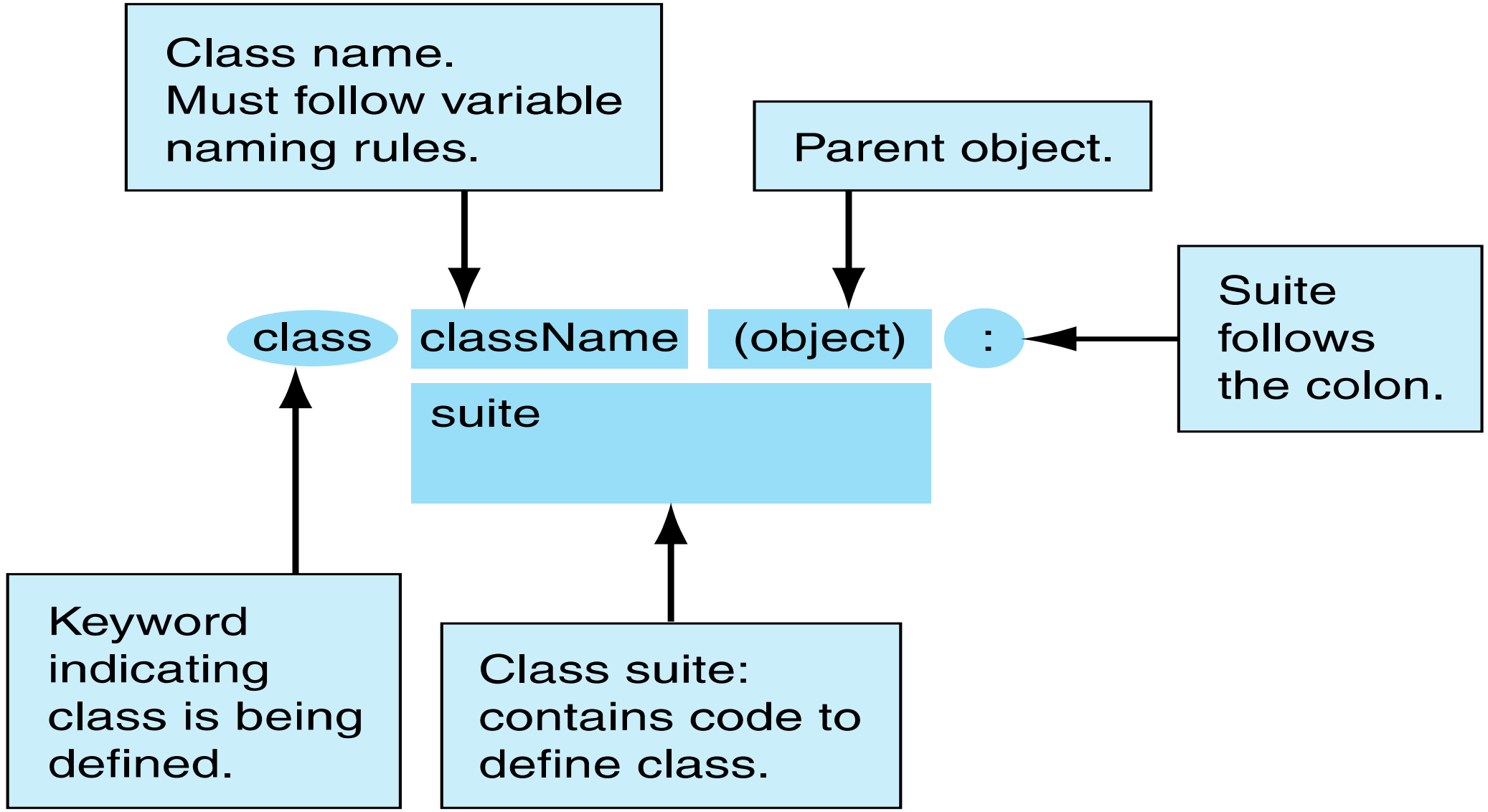
```
>>> dir(x)
[ … 'capitalize', 'casefold', 'center', 'count',
'encode', 'endswith', 'expandtabs', 'find',
'format', … 'lower', 'lstrip', 'maketrans',
'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
>>> dir(y)
[… 'append', 'clear', 'copy', 'count', 'extend',
'index', 'insert', 'pop', 'remove', 'reverse',
'sort']
>>> dir(z)
[…, 'clear', 'copy', 'fromkeys', 'get', 'items',
'keys', 'pop', 'popitem', 'setdefault', 'update',
'values']
```

What is a Class..?

- A class is a special data type which defines how to build a certain kind of object.
- The class also stores some data items that are shared by all the instances of this class
- Instances are objects that are created which follow the definition given inside of the class
- Python doesn’t use separate class interface definitions as in some languages
- You just define the class and then use it

Methods in Classes

- Define a method in a class by including function definitions within the scope of the class block
- There must be a special first argument **self** in all of method definitions which gets bound to the calling instance
- There is usually a special method called **__init__** in most classes


Class name.
Must follow variable naming rules.
Parent object.
class
className
(object)
:
Suite follows the colon.
suite
Keyword indicating class is being defined.
Class suite: contains code to define class.

A Simple Class def: Student

```
class student:
“““A class representing a student ”””
def__init__(self , n, a):
    self.full_name = n
    self.age = a
def get_age(self):        #Method
    return self.age
```

- Define class:
  - Class name, begin with capital letter, by convention
  - object: class based on (Python built-in type)
- Define a method
  - Like defining a function
  - Must **have a special first parameter**, self, which provides way for a method to refer to object itself

Instantiating Objects with '_init_'

- __init__is the default constructor
- __init__serves as a constructor for the class. Usually does some initialization work
- An__init__method can take any number of arguments
- However, the first argument self in the definition of __init__is special

Self

- The first argument of every method is a reference to the current instance of the class
- By convention, we name this argument **self**
- In init , self refers to the object currently being created; so, in other class methods, it refers to the instance whose method was called
- Similar to the keyword this in Java or C++
- But Python uses self more often than Java uses this
- You **do not** give a value for this parameter(self) when you call the method, Python will provide it.

Continue…

Self

…Continue

- Although you must specify self explicitly when defining the method, you don't include it when calling the method.
- Python passes it for you automatically

Defining a method:

(this code inside a class definition.)

```
def get_age(self, num):
    self.age = num
```

Calling a method:

```
>>> x.get_age(23)
```

Deleting instances: No Need to “free”

- When you are done with an object, you don’t have to delete or free it explicitly.
- Python has automatic garbage collection.
- Python will automatically detect when all the references to a piece of memory have gone out of scope.
- frees that memory.
- Generally works well, few memory leaks
- There’s also no “destructor” method for classes

Syntax for accessing attributes and methods

```
>>> f = student("Python", 14)

>>> f.full_name # Access attribute
"Python"

>>> f.get_age() # Access a method
14
```

Thank You

?