Data Processing and Analysis in Python
Lecture 10
Classes

UNIVERSITY OF MARYLAND

ROBERT H. SMITH SCHOOL OF BUSINESS

DR. ADAM LEE

Classes and Objects

- Programmers who use objects and classes know:
  - The interface that can be used with a class
  - The state of an object
  - How to **instantiate** a class to obtain an object
- Objects are **abstractions**
  - Package their state and methods in a single entity that can be referenced with a name
- Class definition is like a **blueprint** for each of the objects of that class and contains:
  - Definitions of all of the methods that its objects recognize
  - Descriptions of the data structures used to maintain the state of an object

Class Definitions

- Syntax of a simple class definition:
  class <class name>(<parent class name>):
  <method definition−1>
  …
  <method definition−*n*>
- Class name is a Python identifier
  - Typically capitalized
- Python classes are organized in a tree-like class **hierarchy**
  - At the top, or root, of this tree is the **Object** class
  - Some terminology: child/subclass **inherits** parent/base/superclass

Method Definitions

- Method definitions are indented below class header
- Syntax of method definitions similar to functions

  ```
  def method(self, ...):
    ["""<doc-string>"""]
    <method body>
  ```

  - Each method definition must include a first parameter named **self**
  - Can have required and/or default arguments, return values, create/use temporary variables
  - Returns None when no return statement is used
- Usage: class.method(arguments)

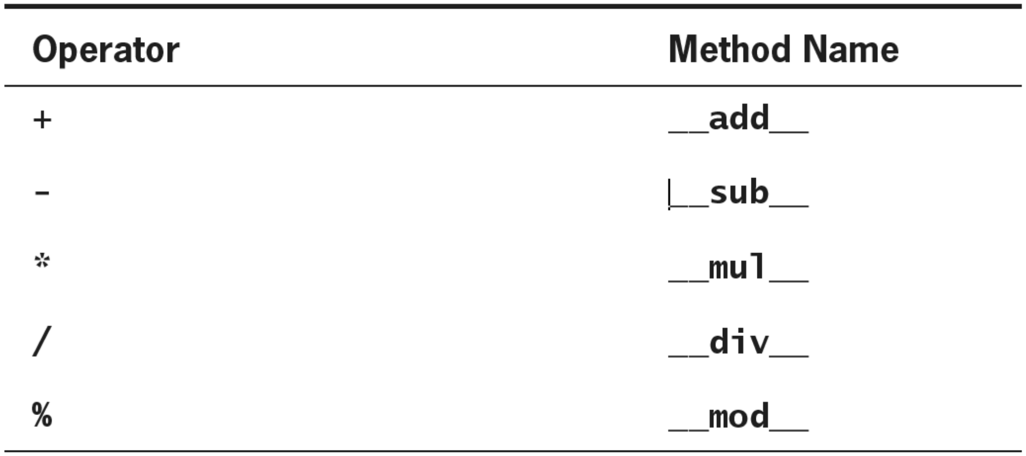
__init__(self, …) Method

- Most classes include a special method named **__init__**
- This method is the class's **constructor**, because it is run automatically when a user instantiates the class
- The purpose of the constructor is to initialize an individual object's **attributes**

__str__(self) Method

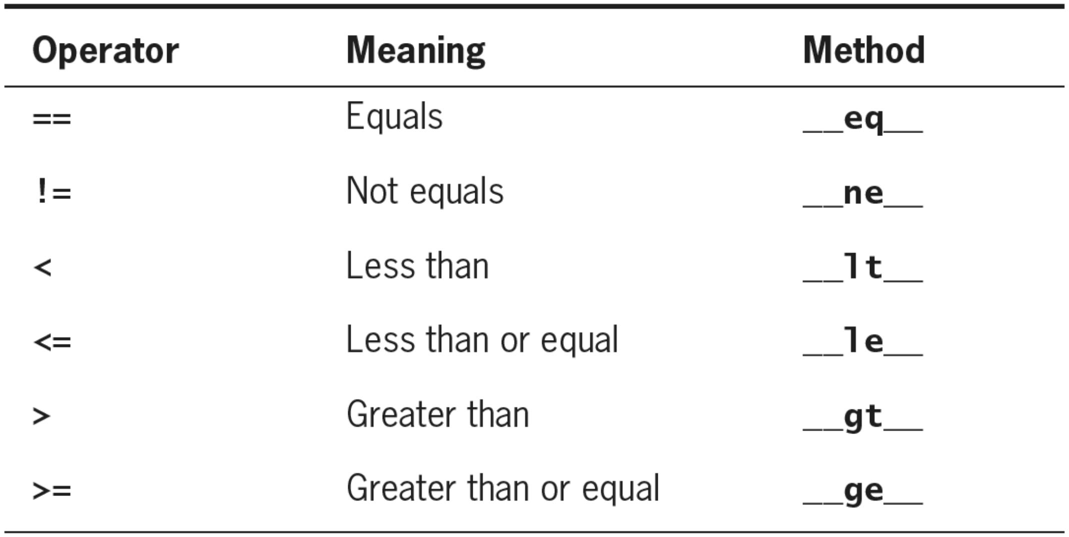
- Returns a string representation of an object's state
- When the str function is called with an object, that object's __str__ method is automatically invoked to obtain the string that str returns
  - The function call str(s) is equivalent to the method call s.__str__()
  - The function call print(s) also automatically runs str(s)
- Perhaps the most important use of __str__ is in debugging, when you often need to observe the state of an object after running another method

Operator Methods



Operator	Method Name
+	`__add__`
-	`__sub__`
*	`__mul__`
/	`__div__`
%	`__mod__`

Table 9-3 Built-in arithmetic operators and their corresponding methods



Operator	Meaning	Method
==	Equals	`__eq__`
!=	Not equals	`__ne__`
<	Less than	`__lt__`
<=	Less than or equal	`__le__`
>	Greater than	`__gt__`
>=	Greater than or equal	`__ge__`

Table 9-5 The comparison operators and methods

Example – Student

Student Method	**What It Does**
s = Student(name, number)	Returns a Student object with the given name and number of scores. Each score is initially 0
s.getName()	Returns the student's name
s.getScore(i)	Returns the student's *i*th score, i must range from 1 through the number of scores
s.setScore(i, score)	Resets the student's *i*th score to score, i must range from 1 through the number of scores
s.getAverage()	Returns the student's average score
s.getHighScore()	Returns the student's highest score
s.__str()__	Same as str(s). Returns a string representation of the student's information

Example – Student

```
>>> from student import Student
>>> s = Student("Maria", 5)
>>> print(s)
Name: Maria
Scores: 0 0 0 0 0
>>> s.setScore(1, 100)
>>> print(s)
Name: Maria
Scores: 100 0 0 0 0
>>> s.getHighScore()
100
>>> s.getAverage()
20
>>> s.getScore(1)
100
```

Rules of Thumb for Defining a Simple Class

- Before writing code, think about the **behavior and attributes** of the objects of the new class
  - What actions does an object perform?
  - How do these actions access or modify the object's state?
- Choose an appropriate **class name**, and develop a short list of the **methods** available to users
  - This interface should include appropriate **parameter names**
  - Brief descriptions of what the methods do
  - Avoid describing how the methods perform their tasks

Rules of Thumb for Defining a Simple Class

- Write a short script that appears to use the new class in an appropriate way
  - The script should **instantiate** the class and **call** all of its methods
  - This helps to clarify the interface of your class and serve as an initial test bed for it
- Choose the appropriate data structures to represent the **attributes** of the class
- Fill in the class template with a constructor (an **__init__** method) and an **__str__** method
  - As soon as you have defined these two methods, you can test your class by instantiating it and printing the resulting object

Rules of Thumb for Defining a Simple Class

- Complete and test the remaining methods incrementally
  - Work in a **bottom-up** manner
  - If one method depends on another, complete the second method first
- Remember to **document your code**
  - Include a **docstring** for the module, the class, and each method
  - Do not add docstrings as an afterthought
  - Write them as soon as you write a class header or a method header
  - Be sure to examine the results by running **help** with the class name

Polymorphism

- **Polymorphism** is the condition of occurrence in different forms
- It refers to the use of a single type (operator, method, or object) to represent different types in different scenarios
  - Polymorphic + operator: 1 + 2, 1.1 + 2.2, "one" + "two", etc.
  - Polymorphic len() function: len("123"), len([1, 2, 3]), etc.

- Python does not support **function overload**
  - To have multiple functions with the same name but with different signatures/implementations
  - The later one always overrides the prior