# Python Tutorial II – Objects and Classes

Mark A. Austin

University of Maryland

*austin@umd.edu*

*ENCE 201, Fall Semester 2023*

April 29, 2024

# Overview

1. Working with Objects and Classes
2. Data Hiding and Encapsulation
3. Relationships Among Classes
4. Inheritance Mechanisms
5. Composition of Object Models
6. Working with Groups of Objects
7. Case Study: GeoModeling Spatial Entities

# Working with Objects and Classes

# Working with Objects and Classes

**Working with Objects and Classes:**

- Collections of objects share similar traits (e.g., data, structure, behavior).
- Collections of objects will form relationships with other collections of objects.

### Definition of a Class

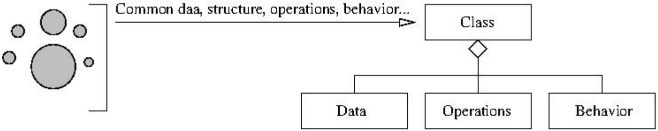
A class is a specification (or blueprint) of an object's structure and behavior.

### Definition of an Object

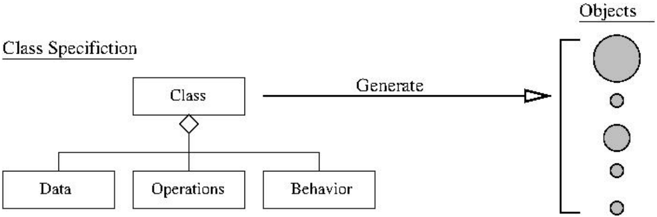
An object is an instance of a class.

# Working with Objects and Classes

## From Collections of Objects to Classes:



## Generation of Objects from Class Specifications:



# Working with Objects and Classes

**Principles for Development of Reusable Code:**

- **Inheritance:** Create new (specialized) classes from existing classes through mechanism of concept extension.
- **Encapsulation:** Hide some details of a class from other (external) classes.
- **Polymorphism:** Use common operation in different ways depending on details of data input.

**Key Design Tasks**

- Identify objects and their attributes and functions,
- Establish relationships among the objects,
- Implement and test the individual objects,
- Assemble and test the system.

# Example 1. Working with Points

## A Very Simple Class in Python

```
# =======================================================
# Point.py: Create point objects ...
#
# Modified by: Mark Austin                  October, 2020
# =======================================================

import math

class Point:

    def __init__(self, xCoord=0, yCoord=0):
        self.__xCoord = xCoord
        self.__yCoord = yCoord

    # compute distance between two points ...

    def distance(self, second):
        x_d = self.__xCoord - second.__xCoord
        y_d = self.__yCoord - second.__yCoord
        return (x_d**2 + y_d**2)**0.5

    # return string represention of object ...

    def __str__(self):
        return "( %6.2f, %6.2f ) " % ( self.__xCoord, self.__yCoord )
```

# Example 1. Working with Points

## Create and Print two Point Objects

```
    pt1 = Point( 0.0, 0.0 )
    pt2 = Point( 3.0, 4.0 )

    print("--- pt1 = %s ..." % (pt1) )
    print("--- pt2 = %s ..." % (pt2) )
```

## Output:

```
--- pt1 = (   0.00,   0.00 )  ...
--- pt2 = (   3.00,   4.00 )  ...
```

## Compute Distance between Two Points

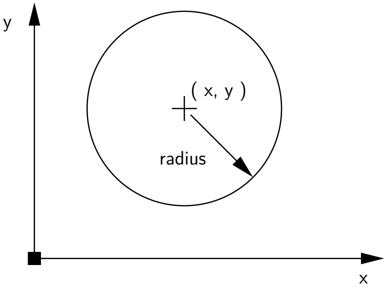
```
    distance = pt1.distance(pt2)
    print("--- Distance between pt1 and pt2 --> %.2f ..." % (distance) )
```

## Output:

```
--- Distance between pt1 and pt2 --> 5.00 ...
```

# Example 2. Working with Circles

A circle can be described by the `(x,y)` position of its center and by its `radius`.



There are numerous things we can do with circles:

- Compute their circumference, perimeter or area,
- Check if a point is inside a circle.

# Example 2. Working with Circles

```
# =======================================================
# Circle.py: Simplified modeling of a circle ...
#
# Written by: Mark Austin                   October, 2020
# =======================================================

import math

class Circle:
  radius = 0
  area   = 0
  perimeter = 0

  def __init__(self, x, y, radius):
    self.radius    = radius
    self.area      = math.pi*radius*radius
    self.perimeter = 2.0*math.pi*radius
    self.x = x
    self.y = y

  # Set circle radius, recompute area and perimeter ...

  def setRadius(self, radius):
    self.radius = radius
    self.area   = math.pi*radius*radius
    self.perimeter = 2.0*math.pi*radius
```

# Example 2. Working with Circles

```

    # Print details of circle ...

    def printCircle(self):
      print("--- Circle: (x,y) = (%.2f, %.2f): radius = %.2f: area = %.2f: perimeter = %.2
            % ( self.x, self.y, self.radius, self.area, self.perimeter ) )
```

## Create and Print two Circle Objects

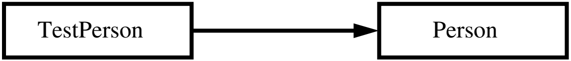
```
      x = Circle( 0.0, 0.0, 3.0 )
      y = Circle( 1.0, 2.0, 4.0 )
      x.printCircle()
      y.printCircle()
```

## Output:

```
--- Circle: (x,y) = (0.00, 0.00): radius = 3.00: area = 28.27
--- Circle: (x,y) = (1.00, 2.00): radius = 4.00: area = 50.27
```

# Example 3. Object Model of a Person

**Part I: Program Architecture.** The TestPerson will create objects of type Person.



## Part II: Person Object Model:

```
# ========================================================
# Person.py: Simplified model of a person ...
#
# Written by: Mark Austin                   October, 2022
# ========================================================

class Person:
  age = 0
  ssn = 0

  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname  = lname

  def printname(self):
    print("--- Name: {:s}, {:s}".format( self.firstname, self.lastname) )
```

# Example 3. Object Model of a Person

**Part II: Person Object Model:** (Continued) ...

```
  # Get first and last names ...

  def getFirstName(self):
    return self.firstname

  def getLastName(self):
    return self.lastname

  # Set/print age ...

  def setAge(self, age):
    self.age = age

  def printAge(self):
    print("--- Age = {:d} ".format(self.age) )

  # Set/print social security number ...

  def setSSN(self, ssn ):
    self.ssn = ssn

  def printSSN(self):
    print("--- Social Security No: {:d} ...".format(self.ssn) )
```

# Example 3. Object Model of a Person

## Part III: Person Test Program:

```
# =====================================================
# TestPerson.py: Test program for person objects ...
# =====================================================

from Person import Person

# main method ...

def main():
    print("--- Enter TestPerson.main()         ... ");
    print("--- =============================== ... ");

    # Exercise methods in class Person ...

    x = Person( "Angela", "Austin" )
    x.printname()

    print("--- First name: {:s} ".format( x.getFirstName() ) )
    print("--- Family name: {:s} ".format( x.getLastName() ) )

    # Initialize attribute values ..

    x.setAge(29)
    x.setSSN(123456789)

    # Print attribute values ..
```

# Example 3. Test Program for Person Object Model

## Part III: Person Test Program: (Continued) ...

```
    x.printAge()
    x.printSSN()

    print("--- =============================== ... ");
    print("--- Finished TestPerson.main()      ... ");

# call the main method ...

main()
```

## Output:

```
--- Enter TestPerson.main()         ...
--- =============================== ...
--- Name: Angela, Austin
--- First name: Angela
--- Family name: Austin
--- Age = 29
--- Social Security No: 123456789
--- =============================== ...
--- Finished TestPerson.main()      ...
```

# Example 3. Object Model of a Person

## Part IV: Files before Program Execution:

```
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 Person.py
-rw-r--r--  1 austin  staff   847 Feb 18 13:26 TestPerson.py
```

## Part IV: Files after Program Execution:

```
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 Person.py
-rw-r--r--  1 austin  staff   847 Feb 18 13:26 TestPerson.py
drwxr-xr-x  4 austin  staff   128 Feb 18 13:27 __pycache__

./__pycache__:
total 16
-rw-r--r--  1 austin  staff  1476 Feb 18 13:27 Person.cpython-37.pyc
```

**Note:** When TestPerson imports Person, python builds a compiled bytecode for Person (with .pyc extension).

Subsequent imports will be easier and faster.

# Data Hiding and Encapsulation

# Hiding Information

## Data Hiding

Data Hiding is isolation of the client from a part of program implementation. Some objects in the module are kept internal, invisible, and inaccessible to the user.

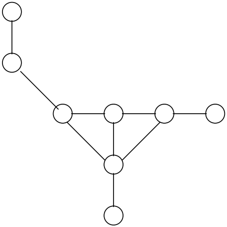
## Principle of Information Hiding

The principle of information hiding states that information which is likely to change (e.g., over the lifetime of a software/systems package) should be hidden inside a module.

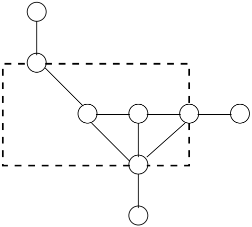
**Key Advantages**

- Prevents accidental linkage to incorrect data.
- It heightens the security against hackers that are unable to access confidential data.

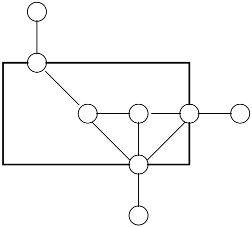
# Data Hiding and Encapsulation



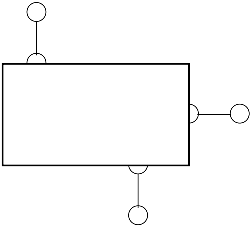
Unstructured Components



Aggregation



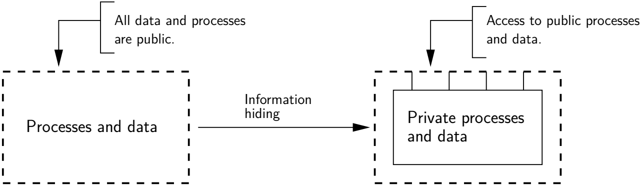
Designer's view of Aggregation



Encapsulation – User's view of Abstraction

# Data Hiding and Encapsulation

**Application.** Process for Implementation of Information Hiding.


All data and processes are public.
Access to public processes and data.
Processes and data
Information hiding
Private processes and data

**Data Hiding in Python** (Private and Protected) ...

- Data hiding is implemented by using a double underscore before (prefix) the attribute name. Making an attribute private hides it from users.
- Use of a single underscore makes the variable/method protected. The variables/methods will be available to the class, and all of its subclasses.

# Example 4. Revised Circle Object Model

## Part I: Revised Circle Object Model

```
# ============================================================
# Circle.py: Implementation of circle model with encapsulation
# (hiding) of circle parameters and properties.
#
# Written by: Mark Austin                        October, 2020
# ============================================================

import math

class Circle:
  __radius = 0           # <-- private parameters ....
  __area   = 0
  __perimeter = 0

  def __init__(self, x, y, radius):
    self.__radius    = radius
    self.__area      = math.pi*radius*radius
    self.__perimeter = 2.0*math.pi*radius
    self.__x = x
    self.__y = y

  # Set circle coordinates ...

  def setX(self, x):
    self.__x = x
```

# Example 4. Revised Circle Object Model

## **Part I: Revised Circle Object Model** (Continued) ...

```
  def setY(self, y):
    self.__y = y

  # Set circle radius, recompute area and perimeter ...

  def setRadius(self, radius):
    self.__radius = radius
    self.__area   = math.pi*radius*radius
    self.__perimeter = 2.0*math.pi*radius

  # Get circle parameters ...

  def getX(self):
    return self.__x

  def getY(self):
    return self.__y

  def getRadius(self):
    return self.__radius

  def getArea(self):
    return self.__area

  def getPerimeter(self):
    return self.__perimeter
```

# Example 4. Revised Circle Object Model

## Part I: Revised Circle Object Model (Continued) ...

```
  # String represention of circle ...

  def __str__(self):
     return "--- Circle: (x,y) = (%.2f, %.2f): radius = %.2f: area = %.2f:
            perimeter = %.2f" % ( self.__x, self.__y, self.__radius,
            self.__area, self.__perimeter )
```

## Part II: Test Program for Circle Object Model

```
# ======================================================
# TestCircles.py: Exercise circle objects.
#
# Written by: Mark Austin                  December 2022
# ======================================================

from Circle import Circle

# main method ...

def main():
    print("--- Enter TestCircles.main()        ... ");
    print("--- =============================== ... ");

    print("--- Part 1: Create and print circle ... ");

    x = Circle( 0.0, 0.0, 3.0 )
    print(x)
```

# Example 4. Revised Circle Object Model

## Part II: Test Program for Circle Object Model (Continued) ...

```
    print("--- =============================== ... ");
    print("--- Finished TestCircles.main()     ... ");

# call the main method ...

main()
```

## Part III: Program Output

```
--- Enter TestCircles.main()        ...
--- =============================== ...
--- Circle: (x,y) = (0.00, 0.00): radius = 3.00: area = 28.27
--- =============================== ...
--- Finished TestCircles.main()     ...
```

# Relationships Among Classes

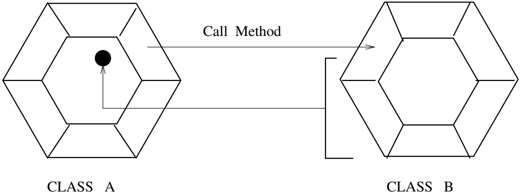
# Relationships Among Classes

## Motivation

- Classes and objects by themselves are not enough to describe the structure of a system.
- We also need to express relationships among classes.
- Object-oriented software packages are assembled from collections of classes and class-hierarchies that are related in three fundamental ways.

# Relationships Among Classes

1. **Use:** Class A uses Class B (method call).



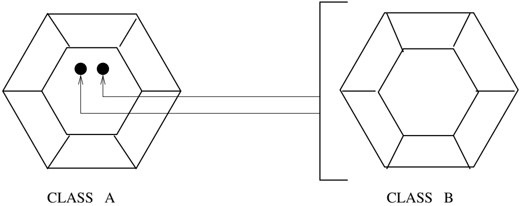
Class A uses Class B if a method in A calls a method in an object of type B.

### Example

```
import math

dAngle = math.sin ( math.PI / 3.0 );
```

# Relationships Among Classes

**2. Containment (Has a):** Class A contains a reference to Class B.



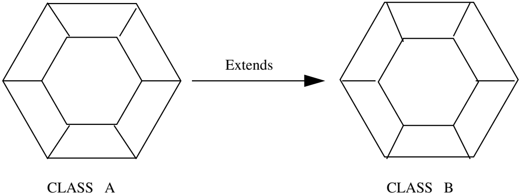
Clearly, containment is a special case of use (i.e., see Item 1.).

**Example**

```
class LineSegment
   self.start = Point() ...
   self.end   = Point() ...
```

# Relationships Among Classes

**3. Inheritance (Is a):** In everyday life, we think of inheritance as something that is received from a predecessor or past generation. Here, Class B inherits the data and methods (extends) from Class A.



## Two Examples from Python

```
class ColoredCircle (Circle) ....
class Student (Person) ....
```

# Inheritance Mechanisms

# Inheritance Mechanisms

## Inheritance Structures

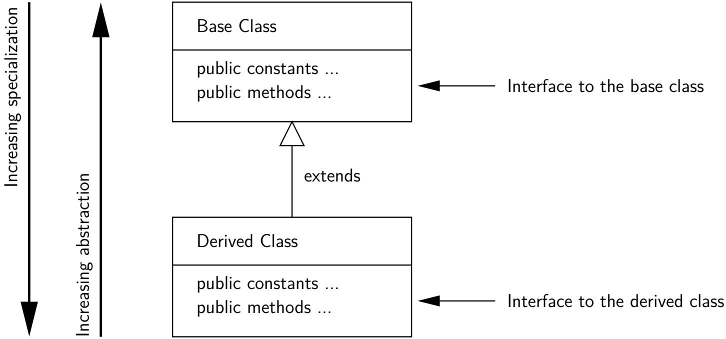
Inheritance structures allow you to capture common characteristics in one model artifact and permit other artifacts to inherit and possibly specialize them. Class hierarchies are explicitly designed for customization through extension.

In this approach to development:

- Forces us to identify and separate the common elements of a system from those aspects that are different/distinct.
- Commonalities are captured in a super-class and inherited and specialized by the sub-classes.
- Inherited features may be overridden with extra features designed to deal with exceptions.

# Base and Derived Classes

**Goal:** Avoid duplication and redundancy of data in a problem specification.



# Base and Derived Classes

Points to note:

- A class in the upper hierarchy is called a superclass (or base, parent class).
- A class in the lower hierarchy is called a subclass (or derived, child, extended class).
- The classes in the lower hierarchy inherit all the variables (static attributes) and methods (dynamic behaviors) from the higher-level classes.

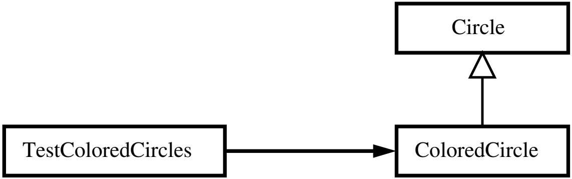
# Base and Derived Classes

## Python Syntax:

```
# ------------------------------------
# Base Class ...
# ------------------------------------

class BaseClass:

      # Constructor of Base Class

      # Base class variables and methods ...

# ------------------------------------
# Derived class extends Base Class ...
# ------------------------------------

class DerivedClass( BaseClass ):

      # Constructor of Derived Class

      # Derived class variables and methods ...
```

# Example 5. Model Colored Circles by Extending Circle

**Part I: Program Architecture.** The TestCircle program will create objects of type ColoredCircle.



Circle Attributes:

- _x, _y, _radius, _area, _perimeter.

ColoredCircle Attributes:

- _color.

# Example 5. Model Colored Circles by Extending Circle

## **Part IIa: Circle Object Model** (with Protected Variables)

```
# ============================================================
# Circle.py: Implementation of circle model with protection of
# circle parameters and methods.
#
# Written by: Mark Austin                        October, 2020
# ============================================================

import math

class Circle:
  _radius = 0
  _area   = 0
  _perimeter = 0

  def __init__(self, x, y, radius):
    self._radius    = radius
    self._area      = math.pi*radius*radius
    self._perimeter = 2.0*math.pi*radius
    self._x = x
    self._y = y

  # Set circle coordinates ...

  def setX(self, x):
    self._x = x

  def setY(self, y):
```

# Example 5. Model Colored Circles by Extending Circle

## **Part IIa: Circle Object Model** (Continued) ...

```
    self._y = y

  # Set circle radius, recompute area and perimeter ...

  def setRadius(self, radius):
    self._radius = radius
    self._area    = math.pi*radius*radius
    self._perimeter = 2.0*math.pi*radius

  # Get circle parameters ...

  def getX(self):
    return self._x

  def getY(self):
    return self._y

  def getRadius(self):
    return self._radius

  def getArea(self):
    return self._area

  def getPerimeter(self):
    return self._perimeter
```

# Example 5. Model Colored Circles by Extending Circle

## **Part IIa: Circle Object Model** (Continued) ...

```
   # String represention of circle ...

   def __str__(self):
      return "--- Circle: (x,y) = (%.2f, %.2f): radius = %.2f: area = %.2f: perimeter = %
             self._x, self._y, self._radius, self._area, self._perimeter )
```

# Example 5. Model Colored Circles by Extending Circle

## Part IIb: Colored Circle Object Model

```
# =========================================================
# ColoredCircle.py: Extend circle to create coloredcircles.
#
# Written by: Mark Austin                     October, 2022
# =========================================================

from Circle import Circle

class ColoredCircle(Circle):
  def __init__(self, x, y, radius, color):
    Circle.__init__(self, x, y, radius)
    self._color = color

  # Set/get color ...

  def setColor(self, color):
    self._color = color

  def getColor(self):
    return self._color

  # String representation of colored circle ...

  def __str__(self):
    return "--- ColoredCircle: (x,y) = (%4.1f, %4.1f): radius = %5.2f: area = %6.2f: col
            self._x, self._y, self._radius, self._area, self._color )
```

# Example 5. Model Colored Circles by Extending Circle

## Part II: Colored Circle Test Program

```
# =======================================================
# TestColoredCircles.py: Exercise colored circle objects.
#
# Written by: Mark Austin                   December 2022
# =======================================================

from Circle import Circle
from ColoredCircle import ColoredCircle

# main method ...

def main():
    print("--- Enter TestCircles.main()        ... ");
    print("--- =============================== ... ");

    print("--- Part 1: Create and print circle ... ");

    x = Circle( 0.0, 0.0, 3.0 )
    print(x)

    print("--- Part 2: Create and print colored circle ... ");

    y = ColoredCircle( 0.0, 0.0, 0.0, "blue" )
    print(y)
    y.setRadius(1.0)
    print(y)
    y.setRadius(2.0)
```

# Example 5. Model Colored Circles by Extending Circle

## **Part II: Colored Circle Test Program** (Continued) ...

```
    print(y)

    print("--- Part 3: Change coordinates and color ... ");

    y.setX( 1.0 )
    y.setY( 1.0 )
    y.setColor("red" )
    y.setRadius(3.0)

    print(y)

    print("--- =============================== ... ");
    print("--- Finished TestCircles.main()      ... ");

# call the main method ...

main()
```

# Example 5. Model Colored Circles by Extending Circle

## Part III: Abbreviated Output:

```
--- Enter TestCircles.main()        ...
--- =============================== ...
--- Part 1: Create and print circle ...
--- Circle: (x,y) = (0.00, 0.00): radius = 3.00: area = 28.27: perimeter = 18.85
--- Part 2: Create and print colored circle ...
--- ColoredCircle: (x,y) = ( 0.0,  0.0): radius =  0.00: area =  0.00: color = blue
--- ColoredCircle: (x,y) = ( 0.0,  0.0): radius =  1.00: area =  3.14: color = blue
--- ColoredCircle: (x,y) = ( 0.0,  0.0): radius =  2.00: area = 12.57: color = blue
--- Part 3: Change coordinates and color ...
--- ColoredCircle: (x,y) = ( 1.0,  1.0): radius =  3.00: area = 28.27: color = red
--- =============================== ...
--- Finished TestCircles.main()     ...
```

**Source Code:** See: python-code.d/inheritance/

# Example 5. Model Colored Circles by Extending Circle

**Part IV: Files before Program Execution:**

```
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 Circle.py
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 ColoredCircle.py
-rw-r--r--  1 austin  staff   847 Feb 18 13:26 TestColoredCircles.py
```

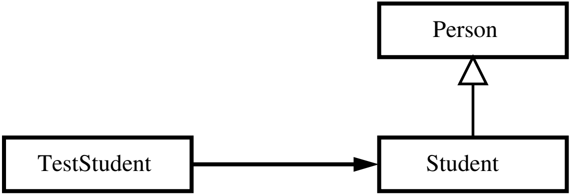
**Part IV: Files after Program Execution:**

```
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 Circle.py
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 ColoredCircle.py
-rw-r--r--  1 austin  staff   847 Feb 18 13:26 TestColoredCircles.py
drwxr-xr-x  4 austin  staff   128 Feb 18 13:27 __pycache__

./__pycache__:
total 16
-rw-r--r--  1 austin  staff  1476 Feb 18 13:27 Circle.cpython-37.pyc
-rw-r--r--  1 austin  staff  1476 Feb 18 13:27 ColoredCircle.cpython-37.pyc
```

**Note:** Python builds compiled bytecodes for Circle and ColoredCircle (with .pyc extension).

# Example 6. Student is an Extension of Person

**Part I: Program Architecture.** The TestStudent program will create objects of type Student.



Person Attributes:

- _firstname, _lastname, _age (age), _ssn (social security), _dob (date of birth).

Student Attributes:

- _gpa (grade point average).

# Example 6. Student is an Extension of Person

## **Part IIa: Person Object Model** (with Protected Variables)

```
# ==================================================================
# Person.py: Simple model of a Person. The scope of variables
# _age, _ssn, and _dob are protected to Person and all subclasses.
#
# Written by: Mark Austin                              November 2022
# ==================================================================

from datetime import date

class Person:
   _age = 0   # <-- age ...
   _ssn = 0   # <-- social security number ...
   _dob = 0   # <-- date of birth ...

   # Constructor method ...

   def __init__(self, fname, lname, dob ):
     self._firstname = fname
     self._lastname  = lname
     self._dob       = dob
     self._age       = self.calculateAge()

   # Get first and last names ...

   def getFirstName(self):
     return self._firstname
```

# Example 6. Student is an Extension of Person

## **Part IIa: Person Object Model** (Continued) ...

```
  def getLastName(self):
    return self._lastname

  # Set/get date of birth ...

  def setDob(self, dob):
    self._dob = dob

  def getDob(self, dob):
    return self._dob

  # Calculate age ...

  def calculateAge(self):
     today = date.today()
     age   = today.year - self._dob.year - ((today.month, today.day) < (self._dob.month
     return age

  # Set/get/print age ...

  def setAge(self, age):
    self._age = age

  def getAge(self):
    return self._age
```

# Example 6. Student is an Extension of Person

## **Part IIa: Person Object Model** (Continued) ...

```

  # Set/get/print social security number ...

  def setSSN(self, ssn ):
    self._ssn = ssn

  def getSSN(self):
    return self._ssn

  # return string represention of object ...

  def __str__(self):
      return "Person: {:6.2f} {:6.2f}: age = {:f} ".format( self._firstname,
                                                            self._lastname,
                                                            self._age )
```

# Example 6. Student is an Extension of Person

## Part Ib: Student Object Model

```
# ============================================================
# Student.py: A Student is a specialization of Person ...
# ============================================================

from Person import Person

class Student(Person):
   _gpa = 0

   # Parameterized constructor ...

   def __init__(self, fname, lname, dob, year):
     Person.__init__(self, fname, lname, dob)
     self._graduationyear = year

   # Set/get gpa ...

   def setGpa(self, gpa):
     self._gpa = gpa

   def getGpa(self):
     return self._gpa
```

# Example 6. Student is an Extension of Person

## Part Ib: Student Object Model

```
   # Boolean to confirm person is a student ...

   def isStudent(self):
      return True

   # Assemble string represention of student ...

   def __str__(self):
      studentinfo = [];
      studentinfo.append("\n");
      studentinfo.append("--- Student: {:s} {:s} ...  \n".format( self._firstname,
                                                                  self._lastname));
      studentinfo.append("--- ------------------------------------------------- \n");
      studentinfo.append("---   Gpa = {:6.2f} ... \n".format( self._gpa));
      studentinfo.append("---   Age = {:6d} ... \n".format( self._age));
      studentinfo.append("---   Graduation year = {:d} ... \n".format(
                                                           self._graduationyear ));
      studentinfo.append("--- ------------------------------------------------- ");
      return "".join(studentinfo);
```

# Example 6. Student is an Extension of Person

## Part II: Student Test Program

```
# =====================================================
# TestStudent.py: Exercise methods in Student class ...
#
# Written by: Mark Austin                 November 2022
# =====================================================

from Student import Student
from datetime import date

# main method ...

def main():
    print("--- Enter TestStudents.main()             ... ");
    print("--- ==================================== ... ");

    print("--- Part 1: Create student Angela Austin ...")

    y = Student( "Angela", "Austin", date(2002,3,2) ,2023)
    y.setGpa(3.5)
    y.setSSN(1234)

    print("--- Part 2: Retrieve student parameters ...")

    print("---  First Name: {:s}".format( y.getFirstName() ) )
    print("---  Last Name: {:s}".format( y.getLastName() ) )
    print("---  Age = {:d}".format( y.getAge() ) )
    print("---  Social Security Number = {:d}".format( y.getSSN() ) )
```

# Example 6. Student is an Extension of Person

## **Part II: Student Test Program** (Continued) ...

```
    print("---  Is student: {:s}".format( str( y.isStudent()) ) )

    print("--- Part 3: Assemble string representation of student ...")

    print( y.__str__() )

    print("--- ===================================== ... ");
    print("--- Finished TestStudents.main()          ... ");

# call the main method ...

main()
```

# Example 6. Student is an Extension of Person

## Part III: Abbreviated Output:

```
--- Part 1: Create student Angela Austin ...
--- Part 2: Retrieve student parameters ...
---
---  First Name: Angela
---  Last Name: Austin
---  Age = 20
---  Social Security Number = 1234
---  Is student: True
---
--- Part 3: Assemble string representation of student ...
---
--- Student: Angela Austin ...
--- -------------------------------------------------
---   Gpa =   3.50 ...
---   Age =     20 ...
---   Graduation year = 2023 ...
--- -------------------------------------------------
```

**Source Code:** See: python-code.d/inheritance/

# Example 6. Student is an Extension of Person

## Part IV: Files before Program Execution:

```
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 Person.py
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 Student.py
-rw-r--r--  1 austin  staff   847 Feb 18 13:26 TestStudents.py
```

## Part IV: Files after Program Execution:

```
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 Person.py
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 Student.py
-rw-r--r--  1 austin  staff   847 Feb 18 13:26 TestStudents.py
drwxr-xr-x  4 austin  staff   128 Feb 18 13:27 __pycache__

./__pycache__:
total 16
-rw-r--r--  1 austin  staff  1476 Feb 18 13:27 Person.cpython-37.pyc
-rw-r--r--  1 austin  staff  1476 Feb 18 13:27 Student.cpython-37.pyc
```

**Note:** Python builds compiled bytecodes for Student and Person (with .pyc extension).

# Mutiple Inheritance Mechanisms

**Multiple Inheritance Structures**

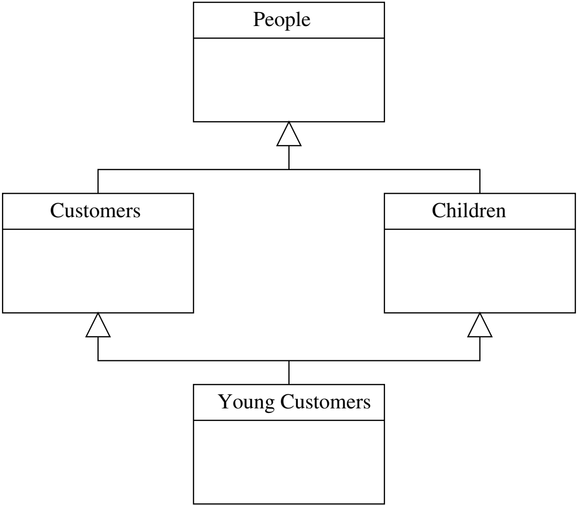
- In a multiple inheritance structure, a class can inherit properties from multiple parents.
- The downside is that properties and/or operations may be partially or fully contradictory.

**Example**

- People is a generalization of Children and Customers.
- Young customers inherits properties from Customers and Children.

**Note.** Python supports use of multiple inheritance. Java explicitly prevents multiple inheritance – instead, it allows classes to have multiple interfaces.

# Mutiple Inheritance Mechanisms



# Mutiple Inheritance Mechanisms

## Python Syntax:

```
class People:

      # People constructor ...
      # People variables, and methods ...

class Customers (People):

      # Customers constructor ...
      # Customers variables, and methods ...

class Children (People):

      # Children constructor ...
      # Children variables, and methods ...

class YoungCustomers( Customers, Children ):

      # YoungCustomer constructor ...
      # YoungCustomer variables, and methods ...
```

# Composition of Object Models

# Composition of Object Models

**Definition**

Composition is known as is a part of or is a relationship.

The member object is a part of the containing class and the member object cannot survive or exist outside the enclosing or containing class or doesn't have a meaning after the lifetime of the enclosing object.

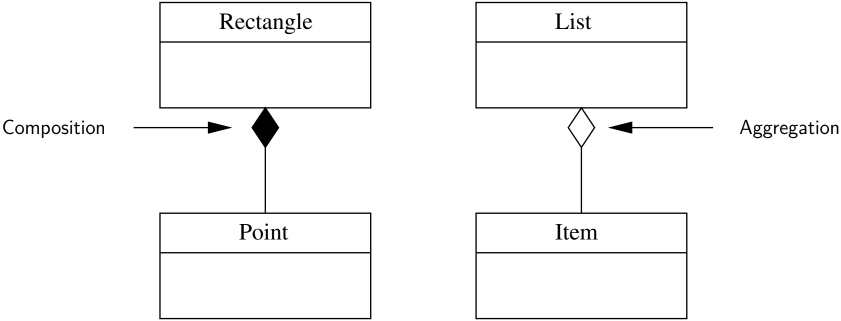
**Is it Aggregation or Composition?**

- Ask the question: if the part moves, can one deduce that the whole moves with it in normal circumstances?

**Example:** A car is composition of wheels and an engine. If you drive the car to work, hopefully the wheels go too!

# Composition of Object Models

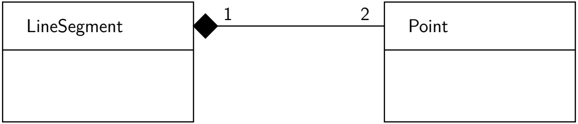
## Notation for Aggregation and Composition



**Recall:** Aggregation is all about grouping of things ...

# Example 7. Modeling Line Segments

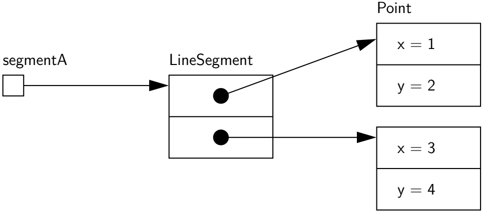
## Model Composition



Creating a line segment object with:

```
segmentA = LineSegment( 1, 2, 3, 4 );
```

should give a layout of memory:



# Example 7. Modeling Line Segments

## Part I: Line Segment Object Model

```
# ====================================================================
# LineSegment.py: Line segments are defined by end points (x1, y1) and
# (x2, y2). Compute length and angle of the line segment in radians.
#
# Written by: Mark Austin                                October, 2022
# ====================================================================

import math

from Point import Point

class LineSegment:
  __length = 0
  __angle   = 0

  def __init__(self, x1, y1, x2, y2 ):
    self.__pt1 = Point(x1,y1)                  # <-- Object composition ...
    self.__pt2 = Point(x2,y2)                  # <-- Object composition ...
    self.__length = self.__pt1.distance(self.__pt2)
    self.__angle   = self.getAngle()

  # Compute angle (radians) for coordinates in four quadrants ....

  def getAngle(self):
     dX = self.__pt2.get_xCoord() - self.__pt1.get_xCoord();
     dY = self.__pt2.get_yCoord() - self.__pt1.get_yCoord();
```

# Example 7. Modeling Line Segments

## **Part I: Line Segment Object Model** (Continued) ...

```

        if dY >  0.0 and dX == 0.0:
           angle = math.pi/2.0
        if dY >= 0.0 and dX > 0.0:
           angle = math.atan( dY/dX )
        if dY >= 0.0 and dX < 0.0:
           angle = math.pi + math.atan( dY/dX )
        if dY < 0.0 and dX < 0.0:
           angle = math.pi + math.atan( dY/dX )
        if dY < 0.0 and dX >= 0.0:
           angle = 2*math.pi + math.atan( dY/dX )

        return angle

    # String represention of line segment ...

    def __str__(self):
        x1 = self.__pt1.get_xCoord();
        y1 = self.__pt1.get_yCoord();
        x2 = self.__pt2.get_xCoord();
        y2 = self.__pt2.get_yCoord();
        return "--- LineSegment: (x1,y1) = (%5.2f, %5.2f), (x2,y2) = (%5.2f, %5.2f),
                    angle = %.2f, length = %.2f" % ( x1, y1, x2, y2, self.__angle, self.__l
```

# Example 7. Modeling Line Segments

## Part II: Line Segment Test Program

```
# ===================================================
# TestLineSegment.py: Exercise line segment class ...
# ===================================================

from LineSegment import LineSegment

# main method ...

def main():
    print("--- Enter TestLineSegment.main()     ... ");
    print("--- ============================== ... ");

    print("--- Part 1: Create test line segment ... ");

    segmentA = LineSegment( 1.0, 2.0,  3.0,  4.0 )
    print(segmentA)

    print("--- Part 2: Sequence of line segments ... ");

    a = LineSegment( 0.0, 0.0,  3.0,  0.0 )
    print(a)
    b = LineSegment( 0.0, 0.0,  3.0,  3.0 )
    print(b)
    c = LineSegment( 0.0, 0.0,  0.0,  3.0 )
    print(c)
    d = LineSegment( 0.0, 0.0, -3.0,  3.0 )
    print(d)
```

# Example 7. Modeling Line Segments

## Part II: Line Segment Test Program (Continued) ...

```
    e = LineSegment( 0.0, 0.0, -3.0,  0.0 )
    print(e)

    print("--- =============================== ... ");
    print("--- Finished TestLineSegment.main() ... ");

# call the main method ...

main()
```

## Part III: Abbreviated Program Output:

```
--- Part 1: Create test line segment ...
--- LineSegment: (x1,y1) = ( 1.00,  2.00), (x2,y2) = ( 3.00,  4.00), angle = 0.79, length = 2.83
--- Part 2: Sequence of line segments ...
--- LineSegment: (x1,y1) = ( 0.00,  0.00), (x2,y2) = ( 3.00,  0.00), angle = 0.00, length = 3.00
--- LineSegment: (x1,y1) = ( 0.00,  0.00), (x2,y2) = ( 3.00,  3.00), angle = 0.79, length = 4.24
--- LineSegment: (x1,y1) = ( 0.00,  0.00), (x2,y2) = ( 0.00,  3.00), angle = 1.57, length = 3.00
--- LineSegment: (x1,y1) = ( 0.00,  0.00), (x2,y2) = (-3.00,  3.00), angle = 2.36, length = 4.24
--- LineSegment: (x1,y1) = ( 0.00,  0.00), (x2,y2) = (-3.00,  0.00), angle = 3.14, length = 3.00
```

**Source Code:** See: python-code.d/classes/

# Working with Groups of Objects

# Pathway From Objects to Groups of Objects

## Data Structures

Now that we know how to create objects, the next subject is how to organize collections of objects so that they are easy to store, easy to find, and easy to modify?

**Approach:** Two-step procedure:

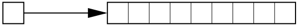
- Choose an appropriate mathematical formalism.
- Develop software to support each formalism.

As a starting point, of objects can be organized into:

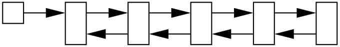
- Arrays
- Linked lists and queues (lists in Python).
- HashMaps (dictionaries in Python).
- Trees and Graphs.

# Memory Layout: Arrays, Lists, Queues, Trees, and Graphs

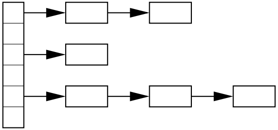
Arrays



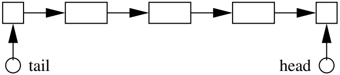
Linked List



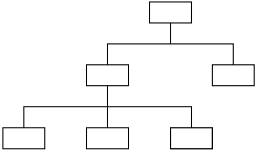
Hash Map



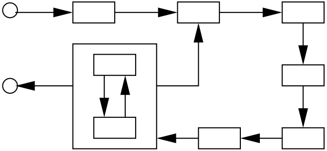
Queues



Trees



Graphs



# Linear and Nonlinear Data Structures

**Linear Data Structure:**

- Items are arranged in a linear fashion.
- Simple to implement.

---

**Examples:**

- **Array:** Sequential arrangement of data elements paired with the index of the data element.
- **Linked List:** Each data element contains a link to another element along with the data present in it.
- **Stack:** LIFO (last in First Out) or FILO (First in Last Out).
- **Queue:** Similar to Stack, but the order of operation is only FIFO (First In First Out).

# Linear and Nonlinear Data Structures

**Nonlinear Data Structure:**

- Items are not ordered in any particular way.
- Often, items are often organized into hierarchies.

---

**Examples:**

- **Binary Tree:** Each data element can be connected to maximum two other data elements and it starts with a root node.
- **Hash Table:** Retrieves values using keys rather than index from a data element.
- **Graph:** Arrangement of vertices and nodes where some of the nodes are connected to each other through links.

# Python Builtin Data Structures

**Lists:**

- Lists are used to store multiple items in a single variable.
- A list may store multiple types (heterogeneous) of elements.

**Dictionary:**

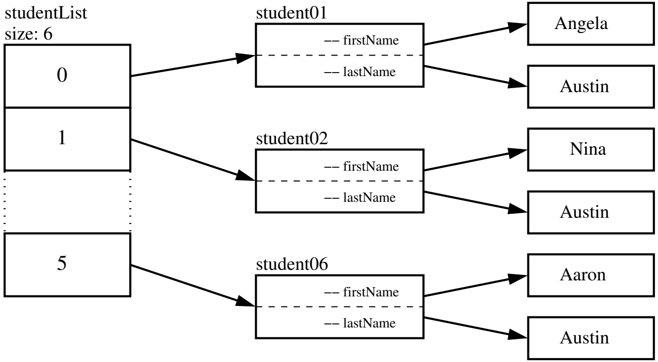
- Dictionaries store data values as key:value pairs.
- As of Python 3.7, a dictionary is a collection which is ordered, changeable and do not allow duplicates.

**Set:**

- Sets store multiple items in a single variable.
- A set is a collection which is unordered, unchangeable (but you can remove items and add new items) and unindexed.

# Example 8: Create List of Student Objects

## Part I: Program Architecture



Assemble list of six students. Sort and print by name and gpa.

# Example 8: Create List of Student Objects

## Part II: Assemble Student Objects ...

```
# ====================================================================
# TestStudents02.py: Assemble list of students ...
#
# Written by: Mark Austin                                February 2023
# ====================================================================

from Student import Student
from datetime import date

# main method ...

def main():
    print("--- Enter TestStudents02.main()                         ... ");
    print("--- =================================================== ... ");

    print("--- ")
    print("--- Part 1: Create student objects ...")

    student01 = Student( "Angela", "Austin", date(2002, 3, 2), 2023)
    student01.setGpa(3.5), student01.setSSN(1234)

    student02 = Student( "Nina", "Austin", date(2001, 4, 12), 2025)
    student02.setGpa(3.2), student02.setSSN(2134)

    student03 = Student( "David", "Austin", date(2000, 6, 8), 2025)
    student03.setGpa(2.9), student03.setSSN(2143)
```

# Example 8: Create List of Student Objects

## Part II: Assemble Student Objects ...

```

    student04 = Student( "Marie", "Austin", date(2005, 8, 5), 2026)
    student04.setGpa(3.9), student04.setSSN(1243)

    student05 = Student( "Albert", "Austin", date(1999, 10, 20), 2026)
    student05.setGpa(3.8), student05.setSSN(3124)

    student06 = Student( "Aaron", "Austin", date(2002, 12, 2), 2026)
    student06.setGpa(4.0), student06.setSSN(1131)

    print("--- ")
    print("--- Part 2: String description of student parameters ...")

    print( student01.__str__() )
    print( student02.__str__() )
    print( student03.__str__() )
    print( student04.__str__() )
    print( student05.__str__() )
    print( student06.__str__() )

    print("--- ")
    print("--- Part 3: Add students to list ... ")

    studentList = [];
    studentList.append(student01)
    studentList.append(student02)
    studentList.append(student03)
```

# Example 8: Create List of Student Objects

## Part II: Assemble Student Objects ...

```
    studentList.append(student04)
    studentList.append(student05)
    studentList.append(student06)

    print("--- ")
    print("--- Part 4: Print contents of list ... ")

    i = 0
    for student in studentList:
       print ("---   list01[{:d}]: {:6s} --> {:.2f} ...".format( i, student.getFirstName
       i = i + 1

    print("--- ")
    print("--- Part 5: Sort list items by first name ... ")

    sort_values = sorted( studentList, key = lambda x: x._firstname )

    i = 0
    for student in sort_values:
       print ("---   list01[{:d}]: {:6s} --> {:.2f} ...".format( i, student.getFirstName
       i = i + 1

    print("--- ")
    print("--- Part 6: Sort list items by gpa ... ")

    sort_values = sorted( studentList, key = lambda x: x._gpa )

    i = 0
```

# Example 8: Create List of Student Objects

## Part II: Assemble Student Objects ...

```
    for student in sort_values:
       print ("---   list01[{:d}]: {:6s} --> {:.2f} ...".format( i, student.getFirstName
       i = i + 1

    print("--- ================================================== ... ");
    print("--- Finished TestStudents02.main()                     ... ");

# call the main method ...

main()
```

# Example 8: Create List of Student Objects

## Part III: Abbreviated Output:

```
--- Enter TestStudents02.main()                         ...
--- =================================================== ...
--- Part 1: Create student objects ...
--- Part 2: String description of student parameters ...

--- Student: Angela Austin ...
--- -------------------------------------------------
---   Gpa = 3.50, Age = 20, Graduation year = 2023 ..
--- -------------------------------------------------

--- Student: Nina Austin ...
--- -------------------------------------------------
---   Gpa = 3.20, Age = 21, Graduation year = 2025 ..
--- -------------------------------------------------

--- Student: David Austin ...
--- -------------------------------------------------
---   Gpa = 2.90, Age = 22, Graduation year = 2025 ..
--- -------------------------------------------------
```

# Example 8: Create List of Student Objects

**Part III: Abbreviated Output:** (Continued) ...

```
--- Student: Marie Austin ...
--- -------------------------------------------------
---   Gpa = 3.90, Age = 17, Graduation year = 2026 ..
--- -------------------------------------------------

--- Student: Albert Austin ...
--- -------------------------------------------------
---   Gpa = 3.80, Age = 23, Graduation year = 2026 ..
--- -------------------------------------------------

--- Student: Aaron Austin ...
--- -------------------------------------------------
---   Gpa = 4.00, Age = 20, Graduation year = 2026 ..
--- -------------------------------------------------

--- Part 4: Print contents of list ...
---
---   list01[0]: Angela --> 3.50 ...
---   list01[1]: Nina   --> 3.20 ...
---   list01[2]: David  --> 2.90 ...
```

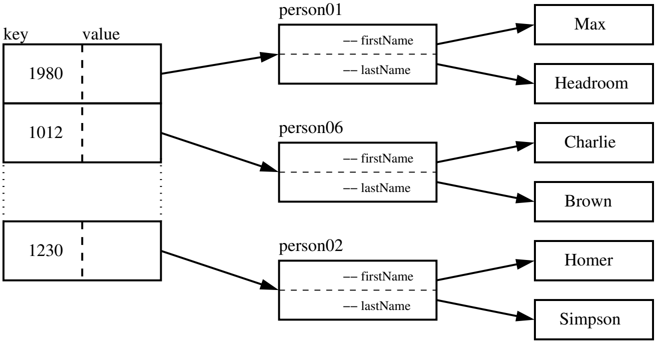
# Example 8: Create List of Student Objects

**Part III: Abbreviated Output:** (Continued) ...

```
---   list01[3]: Marie  --> 3.90 ...
---   list01[4]: Albert --> 3.80 ...
---   list01[5]: Aaron  --> 4.00 ...
---
--- Part 5: Sort list items by first name ...
---   list01[0]: Aaron  --> 4.00 ...
---   list01[1]: Albert --> 3.80 ...
---   list01[2]: Angela --> 3.50 ...
---   list01[3]: David  --> 2.90 ...
---   list01[4]: Marie  --> 3.90 ...
---   list01[5]: Nina   --> 3.20 ...
---
--- Part 6: Sort list items by gpa ...
---   list01[0]: David  --> 2.90 ...
---   list01[1]: Nina   --> 3.20 ...
---   list01[2]: Angela --> 3.50 ...
---   list01[3]: Albert --> 3.80 ...
---   list01[4]: Marie  --> 3.90 ...
---   list01[5]: Aaron  --> 4.00 ...
```

# Example 9: Dictionary of Fictional Characters

## Part I: Program Architecture



Assemble dictionary of six cartoon characters (key = SSN, value = reference to object). Convert dictionary to list, then sort by age.

# Example 9: Dictionary of Fictional Characters

## Part II: Dictionary of Fictional Characters

```
# ===============================================================
# TestDictionary03.py: Create dictionary of objects ...
#
# Last Modified:                                    February 2023
# ===============================================================

from Person import Person

# main method ...

def main():
    print("--- Enter TestDictionary03.main()    ... ");
    print("--- ======================================== ... ");

    # Create cartoon characters ...

    print ("--- Part 01: Create cartoon character objects ...")

    person01 = Person( "Max", "Headroom" )
    person01.setAge(42)
    person01.setSSN(1980)

    person02 = Person( "Homer", "Simpson" )
    person02.setAge(55)
    person02.setSSN(1230)
```

# Example 9: Dictionary of Fictional Characters

## Part II: Dictionary of Fictional Characters:

```
    person03 = Person( "Bart", "Simpson" )
    person03.setAge(35)
    person03.setSSN(1231)

    person04 = Person( "Yogi", "Bear" )
    person04.setAge(65)
    person04.setSSN(1111)

    person05 = Person( "Charlie", "Brown" )
    person05.setAge(72)
    person05.setSSN(1012)

    print ("--- ")
    print ("--- Part 02: Print sample objects ...")
    print ("--- ")

    print("--- person01 --> {:s} ...".format(person01.__str__() ))
    print("--- person05 --> {:s} ...".format(person05.__str__() ))

    print ("--- ")
    print ("--- Part 03: Assemble dictionary of cartoon characters ...")

    cartoon = {}
    cartoon[ person01.getSSN() ] = person01
    cartoon[ person02.getSSN() ] = person02
    cartoon[ person03.getSSN() ] = person03
    cartoon[ person03.getSSN() ] = person03
```

# Example 9: Dictionary of Fictional Characters

## Part II: Dictionary of Fictional Characters:

```
    cartoon[ person04.getSSN() ] = person04
    cartoon[ person05.getSSN() ] = person05

    print ("--- ")
    print ("--- Part 04: Retrieve items from dictionary ...")
    print ("--- ")

    key = 1980
    personItem = cartoon.get(key)
    print("--- key = {:d} --> {:s} ...".format( key, personItem.__str__() ) )

    key = 1230
    personItem = cartoon.get(key)
    print("--- key = {:d} --> {:s} ...".format( key, personItem.__str__() ) )

    key = 1231
    personItem = cartoon.get(key)
    print("--- key = {:d} --> {:s} ...".format( key, personItem.__str__() ) )

    key = 1111
    personItem = cartoon.get(key)
    print("--- key = {:d} --> {:s} ...".format( key, personItem.__str__() ) )

    key = 1012
    personItem = cartoon.get(key)
    print("--- key = {:d} --> {:s} ...".format( key, personItem.__str__() ) )
```

# Example 9: Dictionary of Fictional Characters

## Part II: Dictionary of Fictional Characters:

```
    print ("--- ")
    print ("--- Part 04: Convert dictionary to list ...")

    keysList = list( cartoon.keys() )
    cartoonlist = [];
    for person in keysList:
       cartoonlist.append( cartoon.get(person) )

    print ("--- ")
    print ("--- Part 05: Sort list of cartoon items by age ...")
    print ("--- ")

    sorted_items = sorted( cartoonlist )

    i = 1
    for person in sorted_items:
       print ("---   person[%d]: %s --> %s ..." %( i, person.getFirstName(), person.getA
       i = i + 1

    print("--- ======================================== ... ");
    print("--- Leave TestDictionnary03.main()           ... ");

# call the main method ...

main()
```

# Example 9: Dictionary of Fictional Characters

## Part III: Abbreviated Output:

```
--- Enter TestDictionary03.main()    ...
--- ======================================== ...
--- Part 01: Create cartoon character objects ...
---
--- Part 02: Print sample objects ...
---
--- person01 --> Person: Max Headroom: age = 42.00  ...
--- person05 --> Person: Charlie Brown: age = 72.00  ...
---
--- Part 03: Assemble dictionary of cartoon characters ...
---
--- Part 04: Retrieve items from dictionary ...
---
--- key = 1980 --> Person: Max Headroom: age = 42.00  ...
--- key = 1230 --> Person: Homer Simpson: age = 55.00  ...
--- key = 1231 --> Person: Bart Simpson: age = 35.00  ...
--- key = 1111 --> Person: Yogi Bear: age = 65.00  ...
--- key = 1012 --> Person: Charlie Brown: age = 72.00  ...
```

# Example 9: Dictionary of Fictional Characters

**Part III: Abbreviated Output:** (Continued) ...

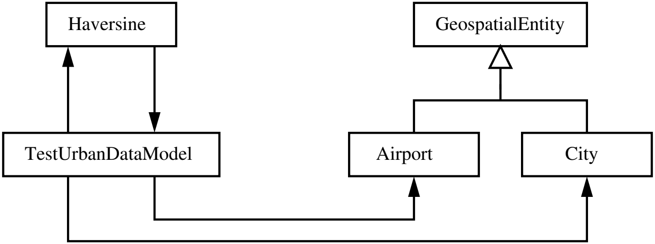
```
--- Part 05: Convert dictionary to list ...
---
--- Part 06: Sort list of cartoon items by age ...
---
---   person[1]: Bart --> 35 ...
---   person[2]: Max --> 42 ...
---   person[3]: Homer --> 55 ...
---   person[4]: Yogi --> 65 ...
---   person[5]: Charlie --> 72 ...
--- ======================================== ...
--- Leave TestDictionnary03.main()           ...
```

# Case Study

## (GeoModeling Spatial Entities)

# Case Study: GeoModeling Spatial Entities

**Geospatial Data Model:** Create city and airport models. Use Haversine formula to compute distances between entities.



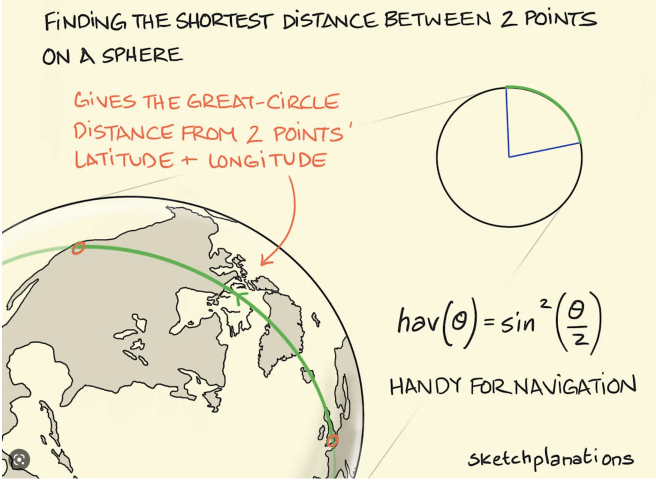
**Geospatial Attributes:** latitude, longitude, elevation.

**City Attributes:** name, population, state, country.

**Airport Attributes:** name, airport code.

# Case Study: GeoModeling Spatial Entities

## Haversine Formula


FINDING THE SHORTEST DISTANCE BETWEEN 2 POINTS ON A SPHERE
GIVES THE GREAT-CIRCLE DISTANCE FROM 2 POINTS' LATITUDE + LONGITUDE
$hav(\theta) = \sin^2\left(\frac{\theta}{2}\right)$
HANDY FOR NAVIGATION
sketchplanations

# Case Study: GeoModeling Spatial Entities

## **Haversine Formula:** Source code ...

```
# =========================================================================
# Haversine.py. Small class that provides approximate distance (km) between
# two points using the Haversine formula.
#
# Call in a static context:
#
# Haversine.distance(47.6788206, -122.3271205,
#                    47.6788206, -122.5271205) --> 14.973190481586224 [km]
#
# earthRadius = 6372.8;     # Earth radius in KM
# earthRadius = 3959.87433 # Earth radius in miles.
#
# Written by: Jason Winn (http://jasonwinn.org)
# Modified by: Mark Austin                                    February 2023
# =========================================================================

from math import radians, cos, sin, asin, sqrt

class Haversine:

   # =====================================
   # Compute haversine distance ...
   # =====================================

   @staticmethod
   def distance(lat1, lon1, lat2, lon2):
```

# Case Study: GeoModeling Spatial Entities

**Haversine Formula:** Source code ...

```
        earthRadius = 3959.87433 # Earth radius in miles.
        dLat = radians(lat2 - lat1)
        dLon = radians(lon2 - lon1)
        lat1 = radians(lat1)
        lat2 = radians(lat2)

        a = sin(dLat/2)**2 + cos(lat1)*cos(lat2)*sin(dLon/2)**2
        c = 2*asin(sqrt(a))

        return earthRadius * c
```

**Source Code:** See: python-code.d/geospatial/

# Case Study: GeoModeling Spatial Entities

## Compute Distance between Washington DC and NYC

```
# ===========================================================
# TestHaversine.py: Small test program for haversine formula.
# ===========================================================

from Haversine import Haversine
from City import City
from Airport import Airport

# main method ...

def main():
    print("--- Enter TestHaversine.main()      ... ");
    print("--- =============================== ... ");

    print("--- Part 1: Create sample cities and airports ... ");

    city01 = City( "Washington DC", 38.907192, -77.036871, 410.0,  5 )
    city02 = City(     "Baltimore", 39.290385, -76.612189, 480.0, 10 )
    city03 = City(      "New York", 40.712784, -74.005941, 265.0, 10 )

    airport01 = Airport( "Baltimore-Washington", "BWI", 39.177404, -76.668392, 148.0 );
    airport02 = Airport( "Washington Dulles",    "IAD", 38.952934, -77.447741, 313.0 );

    print("--- Part 2: Print details of cities and airports ... ");

    print(city01);    print(city02); print(city03)
```

# Case Study: GeoModeling Spatial Entities

## Compute Distance between Washington DC and NYC

```
    print(airport01); print(airport02)

    print("--- Part 3: Compute distances between locations ... ");

    # Compute distance between Washington DC and Baltimore ...

    lat1 = city01.getLatitude(); lon1 = city01.getLongitude()
    lat2 = city02.getLatitude(); lon2 = city02.getLongitude()
    d1 = Haversine.distance(lat1, lon1, lat2, lon2)

    print("--- Distance: Washington DC to Baltimore --> {:f} miles ..".format(d1))

    # Compute distance between Washington DC and New York ...

    lat1 = city01.getLatitude(); lon1 = city01.getLongitude()
    lat2 = city03.getLatitude(); lon2 = city03.getLongitude()

    d1 = Haversine.distance(lat1, lon1, lat2, lon2)

    print("--- Distance: Washington DC to New York --> {:f} miles ..".format(d1))

    # Compute distance between IAD and BWI ...

    lat01 = airport01.getLatitude(); lon01 = airport01.getLongitude()
    lat02 = airport02.getLatitude(); lon02 = airport02.getLongitude()

    d1 = Haversine.distance( lat01, lon01, lat02, lon02)
```

# Case Study: GeoModeling Spatial Entities

## Compute Distance between Washington DC and NYC

```

    code01 = airport01.getAirportCode()
    code02 = airport02.getAirportCode()
    print("--- Distance: {:s} to {:s} --> {:f} miles ..".format( code01, code02, d1))

    print("--- =============================== ... ");
    print("--- Leave TestHaversine.main()       ... ");

# call the main method ...

main()
```

**Source Code:** See: python-code.d/geospatial/

# Case Study: GeoModeling Spatial Entities

## Abbreviated Output:

```
--- Enter TestHaversine.main()      ...
--- =============================== ...
--- Part 1: Create sample cities and airports ...
--- Part 2: Print details of cities and airports ...

--- City: Washington DC ...
--- -------------------------------------------------
---   Latitude   =    38.907192 ...
---   Longitude  =   -77.036871 ...
---   Elevation (highest) = 410.00 ft ...
---   Population =   5.00 ...
--- -------------------------------------------------

--- City: Baltimore ...
--- -------------------------------------------------
---   Latitude   =    39.290385 ...
---   Longitude  =   -76.612189 ...
---   Elevation (highest) = 480.00 ft ...
---   Population =  10.00 ...
--- -------------------------------------------------

--- City: New York ...
--- -------------------------------------------------
---   Latitude   =    40.712784 ...
---   Longitude  =   -74.005941 ...
---   Elevation (highest) = 265.00 ft ...
---   Population =  10.00 ...
--- -------------------------------------------------
```

# Case Study: GeoModeling Spatial Entities

**Abbreviated Output:** (Continued) ...

```
--- Airport: Baltimore-Washington (BWI) ...
--- -------------------------------------------------
---   Latitude   =    39.177404 ...
---   Longitude  =   -76.668392 ...
---   Elevation (highest) = 148.00 ft ...
--- -------------------------------------------------

--- Airport: Washington Dulles (IAD) ...
--- -------------------------------------------------
---   Latitude   =    38.952934 ...
---   Longitude  =   -77.447741 ...
---   Elevation (highest) = 313.00 ft ...
--- -------------------------------------------------

--- Part 3: Compute distances between locations ...

--- Distance: Washington DC to Baltimore --> 34.931571 miles ..
--- Distance: Washington DC to New York --> 203.608912 miles ..
--- Distance: BWI to IAD --> 44.605415 miles ..

--- =============================== ...
--- Leave TestHaversine.main()      ...
```

# References

- ....
- ....