

Python Tutorial II – Objects and Classes

Mark A. Austin

University of Maryland

austin@umd.edu

ENCE 201, Spring Semester 2025

February 23, 2025

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

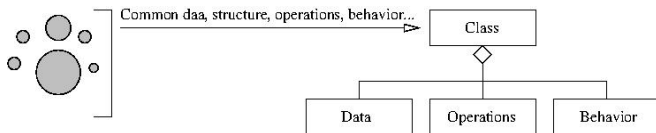
A **class** is a **specification** (or blueprint) of an object's structure and behavior.

Definition of an Object

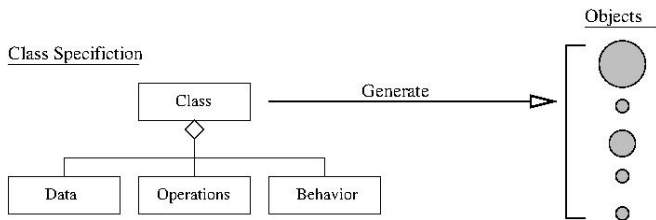
An **object** is an **instance** of a class.

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

```

1  # =====
2  # Point.py: Create point objects ...
3  #
4  # Modified by: Mark Austin                                October, 2020
5  # =====
6
7  import math
8
9  class Point:
10
11     def __init__(self, xCoord=0, yCoord=0):
12         self.__xCoord = xCoord
13         self.__yCoord = yCoord
14
15     # compute distance between two points ...
16
17     def distance(self, second):
18         x_d = self.__xCoord - second.__xCoord
19         y_d = self.__yCoord - second.__yCoord
20         return (x_d**2 + y_d**2)**0.5
21
22     # return string representation of object ...
23
24     def __str__(self):
25         return "( %6.2f, %6.2f )" % ( self.__xCoord, self.__yCoord )

```

Example 1. Working with Points

Create and Print two Point Objects

```

8      pt1 = Point( 0.0, 0.0 )
9      pt2 = Point( 3.0, 4.0 )
10
11      print("--- pt1 = %s ..." % (pt1) )
12      print("--- pt2 = %s ..." % (pt2) )

```

Output:

```

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

```

Compute Distance between Two Points

```

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

```

Output:

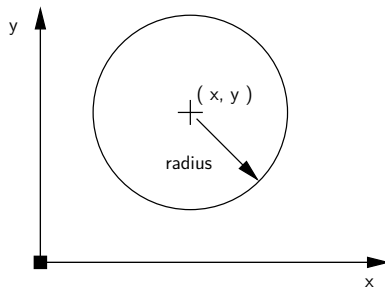
```

--- 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

```

1  # =====
2  # Circle.py: Simplified modeling of a circle ...
3  #
4  # Written by: Mark Austin                                October, 2020
5  # =====
6
7  import math
8
9  class Circle:
10     radius = 0
11     area   = 0
12     perimeter = 0
13
14     def __init__(self, x, y, radius):
15         self.radius    = radius
16         self.area      = math.pi*radius*radius
17         self.perimeter = 2.0*math.pi*radius
18         self.x = x
19         self.y = y
20
21     # Set circle radius, recompute area and perimeter ...
22
23     def setRadius(self, radius):
24         self.radius = radius
25         self.area   = math.pi*radius*radius
26         self.perimeter = 2.0*math.pi*radius

```

Example 2. Working with Circles

```

27
28     # Print details of circle ...
29
30     def printCircle(self):
31         print("--- Circle: (x,y) = (%.2f, %.2f): radius = %.2f: area = %.2f: perimeter = %.2f"
32               % ( self.x, self.y, self.radius, self.area, self.perimeter ) )

```

Create and Print two Circle Objects

```

1     x = Circle( 0.0, 0.0, 3.0 )
2     y = Circle( 1.0, 2.0, 4.0 )
3     x.printCircle()
4     y.printCircle()

```

Output:

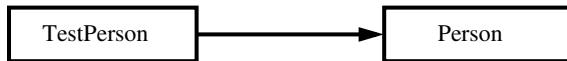
```

--- 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:

```
1  # =====
2  # Person.py: Simplified model of a person ...
3  #
4  # Written by: Mark Austin                      October, 2022
5  # =====
6
7  class Person:
8      age = 0
9      ssn = 0
10
11     def __init__(self, fname, lname):
12         self.firstname = fname
13         self.lastname = lname
14
15     def printname(self):
16         print("--- Name: {:s}, {:s}".format( self.firstname, self.lastname) )
```

Example 3. Object Model of a Person

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

```

17
18     # Get first and last names ...
19
20     def getFirstName(self):
21         return self.firstname
22
23     def getLastName(self):
24         return self.lastname
25
26     # Set/print age ...
27
28     def setAge(self, age):
29         self.age = age
30
31     def printAge(self):
32         print("--- Age = {:d} ".format(self.age) )
33
34     # Set/print social security number ...
35
36     def setSSN(self, ssn ):
37         self.ssn = ssn
38
39     def printSSN(self):
40         print("--- Social Security No: {:d} ...".format(self.ssn) )

```

Example 3. Object Model of a Person

Part III: Person Test Program:

```

1  # =====
2  # TestPerson.py: Test program for person objects ...
3  # =====
4
5  from Person import Person
6
7  # main method ...
8
9  def main():
10     print("--- Enter TestPerson.main()          ... ");
11     print("--- ===== ... ");
12
13     # Exercise methods in class Person ...
14
15     x = Person( "Angela", "Austin" )
16     x.printname()
17
18     print("--- First name: {:s} ".format( x.getFirstName() ) )
19     print("--- Family name: {:s} ".format( x.getLastName() ) )
20
21     # Initialize attribute values ..
22
23     x.setAge(29)
24     x.setSSN(123456789)
25
26     # Print attribute values ..

```

Example 3. Test Program for Person Object Model

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

```

28     x.printAge()
29     x.printSSN()
30
31     print("--- ===== ... ");
32     print("--- Finished TestPerson.main()      ... ");
33
34     # call the main method ...
35
36     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.

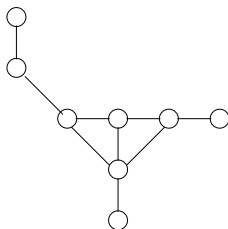
Principle of Information Hiding

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**.

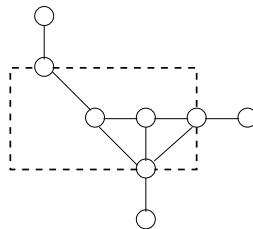
Key Advantages

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

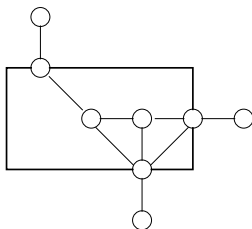
Data Hiding and Encapsulation



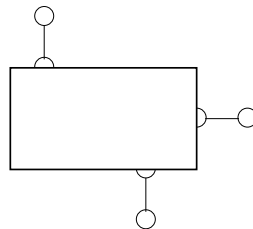
Unstructured Components



Aggregation



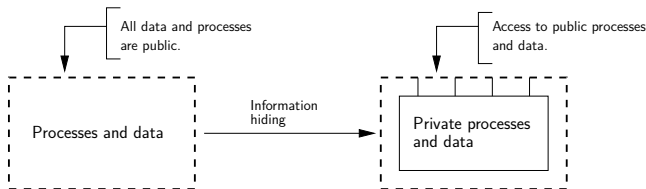
Designer's view of Aggregation



Encapsulation – User's view of Abstraction

Data Hiding and Encapsulation

Application. Process for Implementation of Information Hiding.



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

```
1  # =====
2  # Circle.py: Implementation of circle model with encapsulation
3  # (hiding) of circle parameters and properties.
4  #
5  # Written by: Mark Austin                                October, 2020
6  # =====
7
8  import math
9
10 class Circle:
11     __radius = 0                # <-- private parameters ....
12     __area   = 0
13     __perimeter = 0
14
15     def __init__(self, x, y, radius):
16         self.__radius   = radius
17         self.__area     = math.pi*radius*radius
18         self.__perimeter = 2.0*math.pi*radius
19         self.__x = x
20         self.__y = y
21
22     # Set circle coordinates ...
23
24     def setX(self, x):
25         self.__x = x
```

Example 4. Revised Circle Object Model

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

```
27     def setY(self, y):
28         self.__y = y
29
30     # Set circle radius, recompute area and perimeter ...
31
32     def setRadius(self, radius):
33         self.__radius = radius
34         self.__area = math.pi*radius*radius
35         self.__perimeter = 2.0*math.pi*radius
36
37     # Get circle parameters ...
38
39     def getX(self):
40         return self.__x
41
42     def getY(self):
43         return self.__y
44
45     def getRadius(self):
46         return self.__radius
47
48     def getArea(self):
49         return self.__area
50
51     def getPerimeter(self):
52         return self.__perimeter
```

Example 4. Revised Circle Object Model

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

```
54     # String representation of circle ...
55
56     def __str__(self):
57         return "--- Circle: (x,y) = (%.2f, %.2f): radius = %.2f: area = %.2f:
58             perimeter = %.2f" % ( self.__x, self.__y, self.__radius,
59             self.__area, self.__perimeter )
```

Part II: Test Program for Circle Object Model

```
1  # =====
2  # TestCircles.py: Exercise circle objects.
3  #
4  # Written by: Mark Austin                      December 2022
5  # =====
6
7  from Circle import Circle
8
9  # main method ...
10
11 def main():
12     print("--- Enter TestCircles.main()          ... ");
13     print("--- ===== ... ");
14
15     print("--- Part 1: Create and print circle ... ");
16
17     x = Circle( 0.0, 0.0, 3.0 )
18     print(x)
```

Example 4. Revised Circle Object Model

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

```
20     print("--- ===== ... ");
21     print("--- Finished TestCircles.main()      ... ");
22
23     # call the main method ...
24
25     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

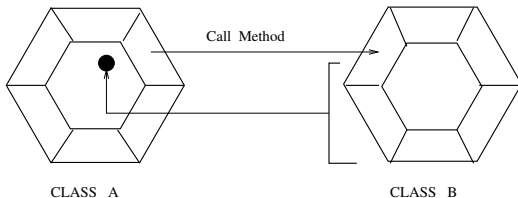
Relationships Among Classes

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.

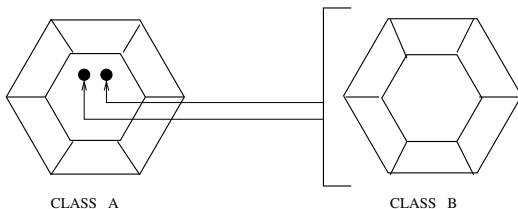
Example

```
import math

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

Relationships Among Classes

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



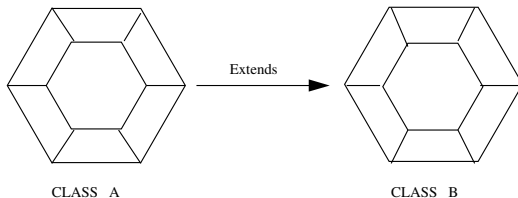
Clearly, containment is a special case of use (i.e., see Item 1.).

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

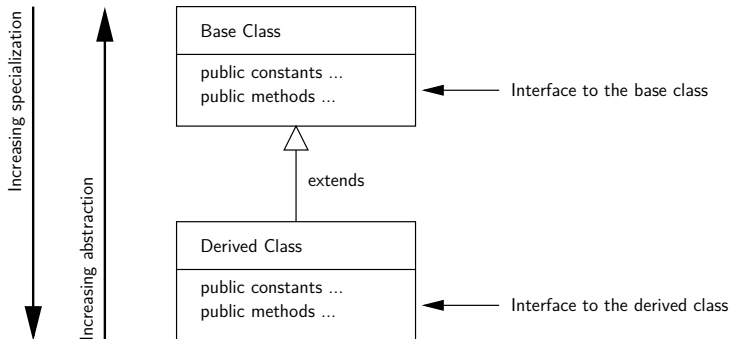
Inheritance structures allow you to capture **common characteristics** in one model artifact and permit other artifacts to inherit and possibly specialize them. Class hierarchies are explicitly designed for **customization through extension**.

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**.

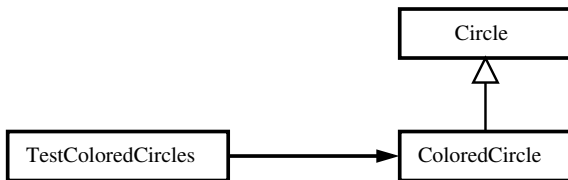
Base and Derived Classes

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)

```
1  # =====
2  # Circle.py: Implementation of circle model with protection of
3  # circle parameters and methods.
4  #
5  # Written by: Mark Austin                                October, 2020
6  # =====
7
8  import math
9
10 class Circle:
11     _radius = 0
12     _area   = 0
13     _perimeter = 0
14
15     def __init__(self, x, y, radius):
16         self._radius = radius
17         self._area    = math.pi*radius*radius
18         self._perimeter = 2.0*math.pi*radius
19         self._x = x
20         self._y = y
21
22     # Set circle coordinates ...
23
24     def setX(self, x):
25         self._x = x
26
27     def setY(self, y):
```

Example 5. Model Colored Circles by Extending Circle

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

```

28         self._y = y
29
30         # Set circle radius, recompute area and perimeter ...
31
32         def setRadius(self, radius):
33             self._radius = radius
34             self._area = math.pi*radius*radius
35             self._perimeter = 2.0*math.pi*radius
36
37         # Get circle parameters ...
38
39         def getX(self):
40             return self._x
41
42         def getY(self):
43             return self._y
44
45         def getRadius(self):
46             return self._radius
47
48         def getArea(self):
49             return self._area
50
51         def getPerimeter(self):
52             return self._perimeter

```

Example 5. Model Colored Circles by Extending Circle

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

```
54     # String representation of circle ...
55
56     def __str__(self):
57         return "--- Circle: (x,y) = (%.2f, %.2f): radius = %.2f: area = %.2f: perimeter = %
58             self._x, self._y, self._radius, self._area, self._perimeter )
```

Example 5. Model Colored Circles by Extending Circle

Part IIb: Colored Circle Object Model

```

1  # =====
2  # ColoredCircle.py: Extend circle to create coloredcircles.
3  #
4  # Written by: Mark Austin                                October, 2022
5  # =====
6
7  from Circle import Circle
8
9  class ColoredCircle(Circle):
10     def __init__(self, x, y, radius, color):
11         Circle.__init__(self, x, y, radius)
12         self._color = color
13
14     # Set/get color ...
15
16     def setColor(self, color):
17         self._color = color
18
19     def getColor(self):
20         return self._color
21
22     # String representation of colored circle ...
23
24     def __str__(self):
25         return "--- ColoredCircle: (x,y) = (%4.1f, %4.1f): radius = %5.2f: area = %6.2f: col
26             self._x, self._y, self._radius, self._area, self._color )

```

Example 5. Model Colored Circles by Extending Circle

Part II: Colored Circle Test Program

```
1  # =====
2  # TestColoredCircles.py: Exercise colored circle objects.
3  #
4  # Written by: Mark Austin                                December 2022
5  # =====
6
7  from Circle import Circle
8  from ColoredCircle import ColoredCircle
9
10 # main method ...
11
12 def main():
13     print("--- Enter TestCircles.main()          ... ");
14     print("--- ===== ... ");
15
16     print("--- Part 1: Create and print circle ... ");
17
18     x = Circle( 0.0, 0.0, 3.0 )
19     print(x)
20
21     print("--- Part 2: Create and print colored circle ... ");
22
23     y = ColoredCircle( 0.0, 0.0, 0.0, "blue" )
24     print(y)
25     y.setRadius(1.0)
26     print(y)
27     y.setRadius(2.0)
```


Example 5. Model Colored Circles by Extending Circle

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

```

28     print(y)
29
30     print("--- Part 3: Change coordinates and color ... ");
31
32     y.setX( 1.0 )
33     y.setY( 1.0 )
34     y.setColor("red" )
35     y.setRadius(3.0)
36
37     print(y)
38
39     print("--- ===== ... ");
40     print("--- Finished TestCircles.main()      ... ");
41
42     # call the main method ...
43
44     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:

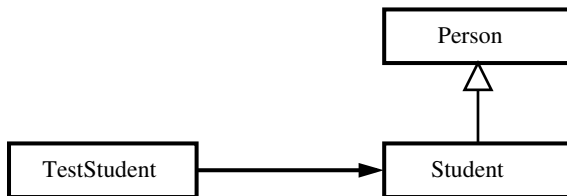
```
-rw-r--r-- 1 austin staff 903 Feb 18 13:21 Circle.py
-rw-r--r-- 1 austin staff 903 Feb 18 13:21 ColoredCircle.py
-rw-r--r-- 1 austin staff 847 Feb 18 13:26 TestColoredCircles.py
drwxr-xr-x 4 austin staff 128 Feb 18 13:27 __pycache__

./__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)

```

1  # =====
2  # Person.py: Simple model of a Person. The scope of variables
3  # _age, _ssn, and _dob are protected to Person and all subclasses.
4  #
5  # Written by: Mark Austin November 2022
6  # =====
7
8  from datetime import date
9
10 class Person:
11     _age = 0      # <-- age ...
12     _ssn = 0     # <-- social security number ...
13     _dob = 0     # <-- date of birth ...
14
15     # Constructor method ...
16
17     def __init__(self, fname, lname, dob ):
18         self._firstname = fname
19         self._lastname  = lname
20         self._dob       = dob
21         self._age       = self.calculateAge()
22
23     # Get first and last names ...
24
25     def getFirstName(self):
26         return self._firstname

```

Example 6. Student is an Extension of Person

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

```

27
28     def getLastName(self):
29         return self._lastname
30
31     # Set/get date of birth ...
32
33     def setDob(self, dob):
34         self._dob = dob
35
36     def getDob(self, dob):
37         return self._dob
38
39     # Calculate age ...
40
41     def calculateAge(self):
42         today = date.today()
43         age = today.year - self._dob.year - ((today.month, today.day) < (self._dob.month, self._dob.day))
44         return age
45
46     # Set/get/print age ...
47
48     def setAge(self, age):
49         self._age = age
50
51     def getAge(self):
52         return self._age

```

Example 6. Student is an Extension of Person

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

```

53      # Set/get/print social security number ...
54
55
56      def setSSN(self, ssn ):
57          self._ssn = ssn
58
59      def getSSN(self):
60          return self._ssn
61
62      # return string representation of object ...
63
64      def __str__(self):
65          return "Person: {:6.2f} {:6.2f}: age = {:f} ".format( self._firstname,
66                                                                self._lastname,
67                                                                self._age )

```

Example 6. Student is an Extension of Person

Part Ib: Student Object Model

```
1  # =====
2  # Student.py: A Student is a specialization of Person ...
3  # =====
4
5  from Person import Person
6
7  class Student(Person):
8      _gpa = 0
9
10     # Parameterized constructor ...
11
12     def __init__(self, fname, lname, dob, year):
13         Person.__init__(self, fname, lname, dob)
14         self._graduationyear = year
15
16     # Set/get gpa ...
17
18     def setGpa(self, gpa):
19         self._gpa = gpa
20
21     def getGpa(self):
22         return self._gpa
```


Example 6. Student is an Extension of Person

Part Ib: Student Object Model

```

24  # Boolean to confirm person is a student ...
25
26  def isStudent(self):
27      return True
28
29  # Assemble string representation of student ...
30
31  def __str__(self):
32      studentinfo = [];
33      studentinfo.append("\n");
34      studentinfo.append("--- Student: {:s} {:s} ... \n".format( self._firstname,
35                                                                self._lastname));
36      studentinfo.append("--- ----- \n");
37      studentinfo.append("---   Gpa = {:6.2f} ... \n".format( self._gpa));
38      studentinfo.append("---   Age = {:6d} ... \n".format( self._age));
39      studentinfo.append("---   Graduation year = {:d} ... \n".format(
40                                                                self._graduationyear ));
41      studentinfo.append("--- ----- ");
42      return "".join(studentinfo);

```

Example 6. Student is an Extension of Person

Part II: Student Test Program

```

1  # =====
2  # TestStudent.py: Exercise methods in Student class ...
3  #
4  # Written by: Mark Austin                      November 2022
5  # =====
6
7  from Student import Student
8  from datetime import date
9
10 # main method ...
11
12 def main():
13     print("--- Enter TestStudents.main()          ... ");
14     print("--- ===== ... ");
15
16     print("--- Part 1: Create student Angela Austin ...")
17
18     y = Student( "Angela", "Austin", date(2002,3,2) ,2023)
19     y.setGpa(3.5)
20     y.setSSN(1234)
21
22     print("--- Part 2: Retrieve student parameters ...")
23
24     print("--- First Name: {:s}".format( y.getFirstName() ) )
25     print("--- Last Name:  {:s}".format( y.getLastName() ) )
26     print("--- Age =      {:d}".format( y.getAge() ) )
27     print("--- Social Security Number = {:d}".format( y.getSSN() ) )

```

Example 6. Student is an Extension of Person

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

```
28     print("--- Is student: {:s}".format( str( y.isStudent()) ) )
29
30     print("--- Part 3: Assemble string representation of student ...")
31
32     print( y.__str__() )
33
34     print("--- ===== ... ");
35     print("--- Finished TestStudents.main()          ... ");
36
37     # call the main method ...
38
39     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](#)).

Multiple Inheritance Mechanisms

Multiple Inheritance Structures

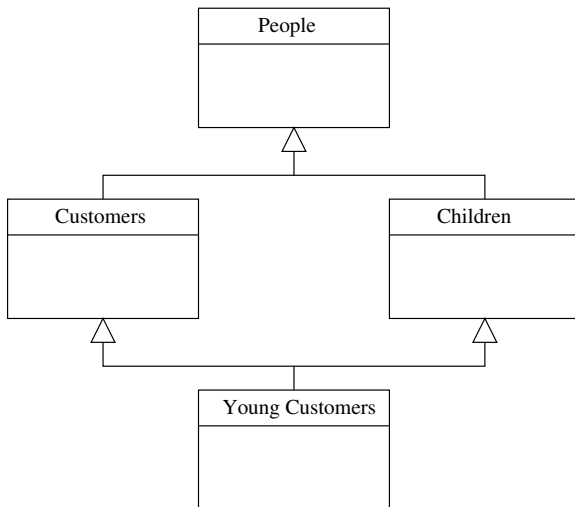
- In a multiple inheritance structure, a class can inherit properties from multiple parents.
- The downside is that properties and/or operations may be partially or fully contradictory.

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.

Multiple Inheritance Mechanisms



Multiple 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.

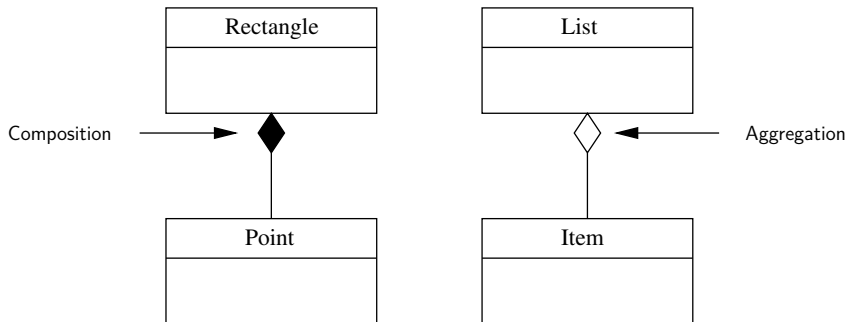
Is it Aggregation or Composition?

- 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

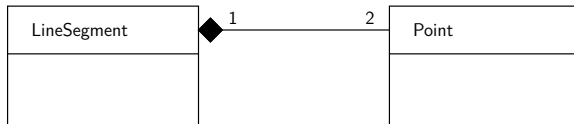
Notation for Aggregation and Composition



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

Example 7. Modeling Line Segments

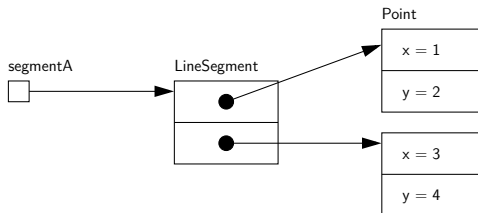
Model Composition



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

```
1  # =====
2  # LineSegment.py: Line segments are defined by end points (x1, y1) and
3  # (x2, y2). Compute length and angle of the line segment in radians.
4  #
5  # Written by: Mark Austin October, 2022
6  # =====
7
8  import math
9
10 from Point import Point
11
12 class LineSegment:
13     __length = 0
14     __angle = 0
15
16     def __init__(self, x1, y1, x2, y2 ):
17         self.__pt1 = Point(x1,y1) # <-- Object composition ...
18         self.__pt2 = Point(x2,y2) # <-- Object composition ...
19         self.__length = self.__pt1.distance(self.__pt2)
20         self.__angle = self.getAngle()
21
22     # Compute angle (radians) for coordinates in four quadrants ....
23
24     def getAngle(self):
25         dX = self.__pt2.get_xCoord() - self.__pt1.get_xCoord();
26         dY = self.__pt2.get_yCoord() - self.__pt1.get_yCoord();
```

Example 7. Modeling Line Segments

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

```

27
28     if dY > 0.0 and dX == 0.0:
29         angle = math.pi/2.0
30     if dY >= 0.0 and dX > 0.0:
31         angle = math.atan( dY/dX )
32     if dY >= 0.0 and dX < 0.0:
33         angle = math.pi + math.atan( dY/dX )
34     if dY < 0.0 and dX < 0.0:
35         angle = math.pi + math.atan( dY/dX )
36     if dY < 0.0 and dX >= 0.0:
37         angle = 2*math.pi + math.atan( dY/dX )
38
39     return angle
40
41     # String representation of line segment ...
42
43     def __str__(self):
44         x1 = self.__pt1.get_xCoord();
45         y1 = self.__pt1.get_yCoord();
46         x2 = self.__pt2.get_xCoord();
47         y2 = self.__pt2.get_yCoord();
48         return "---- LineSegment: (x1,y1) = (%5.2f, %5.2f), (x2,y2) = (%5.2f, %5.2f),
49             angle = %.2f, length = %.2f" % ( x1, y1, x2, y2, self.__angle, self.__l

```

Example 7. Modeling Line Segments

Part II: Line Segment Test Program

```
1  # =====
2  # TestLineSegment.py: Exercise line segment class ...
3  # =====
4
5  from LineSegment import LineSegment
6
7  # main method ...
8
9  def main():
10     print("--- Enter TestLineSegment.main()    ... ");
11     print("--- ===== ... ");
12
13     print("--- Part 1: Create test line segment ... ");
14
15     segmentA = LineSegment( 1.0, 2.0,  3.0,  4.0 )
16     print(segmentA)
17
18     print("--- Part 2: Sequence of line segments ... ");
19
20     a = LineSegment( 0.0, 0.0,  3.0,  0.0 )
21     print(a)
22     b = LineSegment( 0.0, 0.0,  3.0,  3.0 )
23     print(b)
24     c = LineSegment( 0.0, 0.0,  0.0,  3.0 )
25     print(c)
26     d = LineSegment( 0.0, 0.0, -3.0,  3.0 )
27     print(d)
```

Example 7. Modeling Line Segments

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

```

28     e = LineSegment( 0.0, 0.0, -3.0, 0.0 )
29     print(e)
30
31     print("--- ===== ... ");
32     print("--- Finished TestLineSegment.main() ... ");
33
34     # call the main method ...
35
36     main()

```

Part III: Abbreviated Program Output:

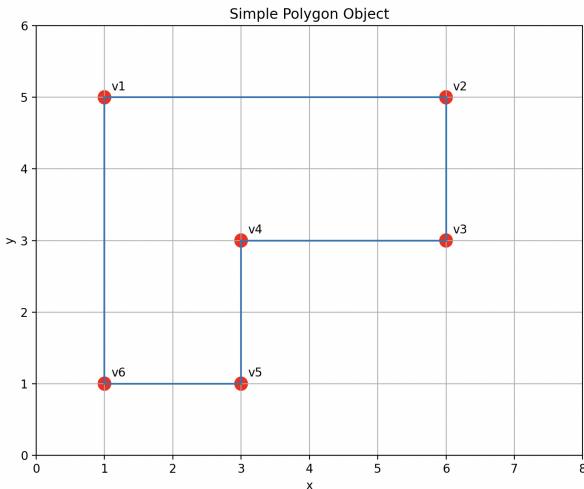
```

--- 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/

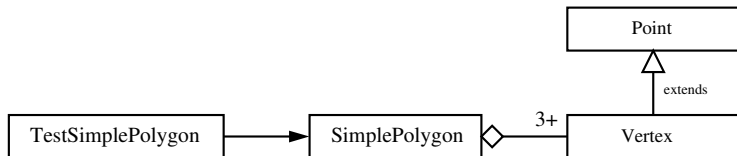
Example 8. Simple Polygon Object



Source Code: See: python-code.d/applications/shapes/TestSimplePolygon01.py

Example 8. Simple Polygon Object

Part I: Program Architecture



Point Attributes:

- `_xCoord`, `_yCoord`, ...

Vertex Attributes:

- `_label` ...

SimplePolygon Attributes:

- `v01`, `v02`, ... `v06`, `polygon01`, ...

Example 8. Simple Polygon Object

Part II: Abbreviated Program Output:

```
--- Enter TestSimplePolygon01.main()    ...
--- ===== ...

--- Part 1: Create list of vertices ...

--- Part 2: Assemble and print simple polygon object ...

--- SimplePolygon: L-shaped Polygon ...
--- -----
--- Vertex 1: (x,y) = ( 1.00,  5.00) ...
--- Vertex 2: (x,y) = ( 6.00,  5.00) ...
--- Vertex 3: (x,y) = ( 6.00,  3.00) ...
--- Vertex 4: (x,y) = ( 3.00,  3.00) ...
--- Vertex 5: (x,y) = ( 3.00,  1.00) ...
--- Vertex 6: (x,y) = ( 1.00,  1.00) ...
--- Perimeter   = 18.00 ...
--- -----
--- Part 3: Draw simple polygon ...

--- ===== ...
--- Finished TestSimplePolygon01.main() ...
```

Example 8. Simple Polygon Object

Part III: Point Object Model

```

1  # =====
2  # Point01.py: Bare-bones implementation of a Point class ...
3  #
4  # Written by: Mark Austin                      October, 2024
5  # =====
6
7  import math
8
9  class Point:
10
11     def __init__(self, xCoord=0, yCoord=0):
12         self._xCoord = xCoord
13         self._yCoord = yCoord
14
15     # Get/set X coordinate
16
17     def getX(self):
18         return self._xCoord
19
20     def setX(self, xCoord):
21         self._xCoord = xCoord
22
23     # Get/set Y coordinate
24
25     def getY(self):
26         return self._yCoord

```

Example 8. Simple Polygon Object

Part III: Point Object Model (Continued) ...

```

27
28     def setY(self, yCoord):
29         self._yCoord = yCoord
30
31         # Get current position
32
33     def get_position(self):
34         return self.getX(), self.getY()
35
36         # Change x & y coordinates by p & q
37
38     def move(self, p, q):
39         self._xCoord += p
40         self._yCoord += q
41
42         # Compute distance between two points ...
43
44     def distance(self, second):
45         x_d = self.getX() - second.getX()
46         y_d = self.getY() - second.getY()
47         return (x_d**2 + y_d**2)**0.5
48
49         # Return string representation of object ...
50
51     def __str__(self):
52         return "( %6.2f, %6.2f )" % ( self.getX(), self.getY() )

```

Example 8. Simple Polygon Object

Part III: Vertex Object Model

```
1  # =====
2  # Vertex.py: A vertex is a point with a label ...
3  # =====
4
5  from Point01 import Point
6
7  class Vertex(Point):
8      _label = ""
9
10     # Constructor method ...
11
12     def __init__(self, x, y) :
13         Point.__init__(self, x, y)
14         self._label = ""
15
16     # Set/get label ...
17
18     def setLabel(self, label ):
19         self._label = label
20
21     def getLabel(self):
22         return self._label
```

Example 8. Simple Polygon Object

Part III: Vertex Object Model (Continued) ...

```

23
24     # Assemble string representation of Vertex ...
25
26     def __str__(self):
27         vertexinfo = [];
28         vertexinfo.append("\n");
29         vertexinfo.append("--- Vertex: {:s} ... \n".format( self.getLabel()));
30         vertexinfo.append("--- ----- \n");
31         vertexinfo.append("---      Coordinate: (x,y) = {:s} ... \n".format( Point.__self__(
32         vertexinfo.append("--- ----- ");
33         return "".join(vertexinfo);

```

Example 8. Simple Polygon Object

Part IV: Simple Polygon Object Model

```

1  # =====
2  # SimplePolygon01.py: Bare-bones implementation of a simple polygon.
3  #
4  # Written by: Mark Austin                                October 2024
5  # =====
6
7  import math
8
9  from Vertex01 import Vertex
10 from matplotlib.patches import Circle
11 from matplotlib.lines import Line2D
12
13 class SimplePolygon:
14     area      = 0
15     perimeter = 0
16     name      = ""
17     coords    = []
18
19     # Constructor method ...
20
21     def __init__(self, vertexlist):
22         self.coords = vertexlist;           # <--- Assign vertex list to coords ...
23         self.perimeter = self.getPerimeter() # <--- Compute perimeter ...
24
25     # Set/get name ...
26
27     def setName(self, name):

```


Example 8. Simple Polygon Object

Part IV: Simple Polygon Object Model (Continued) ...

```
28         self.name = name
29
30     def getName(self):
31         return self.name
32
33     # Compute polygon perimeter ...
34
35     def getPerimeter(self):
36
37         dperimeter = 0.0;
38         for i in range( len(self.coords)-1):
39             dperimeter += self.coords[i].distance( self.coords[i+1] );
40
41         lastnode = len( self.coords) - 1
42         dperimeter += self.coords[ lastnode ].distance( self.coords[0] );
43
44         return dperimeter;
45
46     # Draw simple polygon ...
47
48     def draw(self, ax):
49
50         # Draw polygon edges ...
51
52         for i in range( len(self.coords)-1):
53             xcoords = [ self.coords[i].getX(), self.coords[i+1].getX() ];
54             ycoords = [ self.coords[i].getY(), self.coords[i+1].getY() ];
55             ax.add_line( Line2D(xcoords, ycoords) )
```

Example 8. Simple Polygon Object

Part IV: Simple Polygon Object Model (Continued) ...

```

56
57     lastnode = len( self.coords ) - 1
58     xcoords = [ self.coords[0].getX(), self.coords[ lastnode ].getX() ];
59     ycoords = [ self.coords[0].getY(), self.coords[ lastnode ].getY() ];
60     ax.add_line( Line2D(xcoords, ycoords) )
61
62     # Draw polygon vertices as small circles ...
63
64     width = 0.1;
65     for i in range(len( self.coords )):
66         xcoord = self.coords[i].getX();
67         ycoord = self.coords[i].getY();
68         ax.add_patch( Circle( (xcoord, ycoord), width, facecolor='red') )
69
70     # Draw node labels ...
71
72     dx = 0.1; dy = 0.1
73     for i in range(len( self.coords )):
74         xcoord = self.coords[i].getX();
75         ycoord = self.coords[i].getY();
76         label = self.coords[i].getLabel();
77         ax.text( xcoord + dx, ycoord + dy, label )

```

Example 8. Simple Polygon Object

Part IV: Simple Polygon Object Model (Continued) ...

```

78
79  # String representation of simple polygon ...
80
81  def __str__(self):
82      polygoninfo = [];
83      polygoninfo.append("\n");
84      polygoninfo.append("--- SimplePolygon: {:s} ... \n".format( self.name ));
85      polygoninfo.append("--- ----- \n");
86
87      for i in range(len( self.coords )):
88          xc = self.coords[i].getX();
89          yc = self.coords[i].getY();
90          polygoninfo.append("--- Vertex {:2d}: (x,y) = ({:6.2f}, {:6.2f}) ... \n".for
91
92      polygoninfo.append("--- Perimeter      = {:6.2f} ... \n".format( self.getPerimeter()
93      polygoninfo.append("--- ----- ");
94      return "".join(polygoninfo);

```

Example 8. Simple Polygon Object

Part V: Simple Polygon Test Program

```
1  # =====
2  # TestSimplePolygon01.py: Exercise SimplePolygon class ...
3  # =====
4
5  from Vertex01 import Vertex
6  from SimplePolygon01 import SimplePolygon
7
8  import matplotlib.pyplot as plt
9
10 # main method ...
11
12 def main():
13     print("--- Enter TestSimplePolygon01.main()    ... ");
14     print("--- ===== ... ");
15
16     print("--- Part 1: Create list of vertices ... ");
17
18     v01 = Vertex ( 1.0, 5.0 ); v01.setLabel("v1");
19     v02 = Vertex ( 6.0, 5.0 ); v02.setLabel("v2");
20     v03 = Vertex ( 6.0, 3.0 ); v03.setLabel("v3");
21     v04 = Vertex ( 3.0, 3.0 ); v04.setLabel("v4");
22     v05 = Vertex ( 3.0, 1.0 ); v05.setLabel("v5");
23     v06 = Vertex ( 1.0, 1.0 ); v06.setLabel("v6");
24
25     print("--- Part 2: Assemble and print simple polygon object ... ");
```

Example 8. Simple Polygon Object

Part V: Simple Polygon Test Program (Continued) ...

```
27     polygon01 = SimplePolygon( [ v01, v02, v03, v04, v05, v06 ] )
28     polygon01.setName("L-shaped Polygon")
29
30     print( polygon01 )
31
32     print("--- Part 3: Draw simple polygon ... \n");
33
34     # Define Matplotlib figure and axis
35
36     fig, ax = plt.subplots()
37
38     polygon01.draw(ax)
39
40     plt.title('Simple Polygon Object')
41     plt.ylabel('y')
42     plt.xlabel('x')
43     plt.ylim( 0, 6 )
44     plt.xlim( 0, 8 )
45     plt.grid(True)
46     plt.show()
47
48     print("--- ===== ... ");
49     print("--- Finished TestSimplePolygon01.main()      ... ");
50
51     # call the main method ...
52
53     main()
```

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:

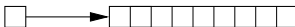
- Choose an appropriate **mathematical formalism**.
- Develop **software** to **support each formalism**.

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

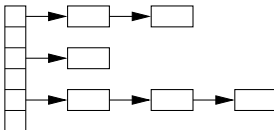
- Arrays
- Linked lists and queues (lists in Python).
- HashMaps (dictionaries in Python).
- Trees and Graphs.

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

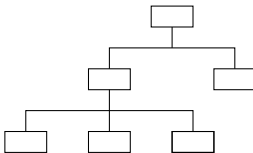
Arrays



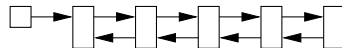
Hash Map



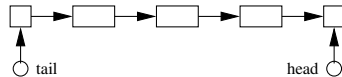
Trees



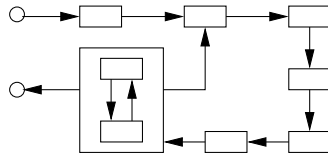
Linked List



Queues



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:

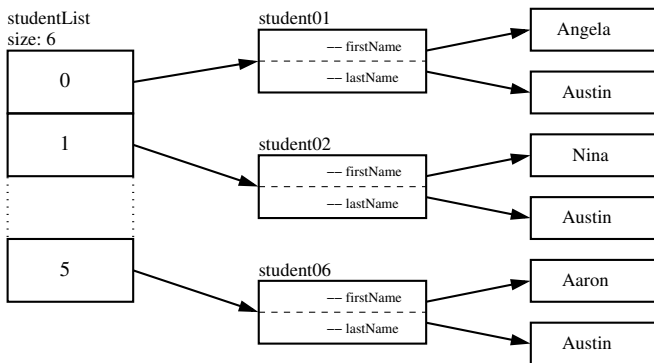
- Dictionaries store **data values** as **key:value pairs**.
- As of Python 3.7, a dictionary is a collection which is ordered, changeable and do not allow duplicates.

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 9: Create List of Student Objects

Part I: Program Architecture



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

Example 9: Create List of Student Objects

Part II: Assemble Student Objects ...

```

1  # =====
2  # TestStudents02.py: Assemble list of students ...
3  #
4  # Written by: Mark Austin                                February 2023
5  # =====
6
7  from Student import Student
8  from datetime import date
9
10 # main method ...
11
12 def main():
13     print("--- Enter TestStudents02.main()                ... ");
14     print("--- =====                                ... ");
15
16     print("--- ")
17     print("--- Part 1: Create student objects ...")
18
19     student01 = Student( "Angela", "Austin", date(2002, 3, 2), 2023)
20     student01.setGpa(3.5), student01.setSSN(1234)
21
22     student02 = Student( "Nina", "Austin", date(2001, 4, 12), 2025)
23     student02.setGpa(3.2), student02.setSSN(2134)
24
25     student03 = Student( "David", "Austin", date(2000, 6, 8), 2025)
26     student03.setGpa(2.9), student03.setSSN(2143)

```

Example 9: Create List of Student Objects

Part II: Assemble Student Objects ...

```
27
28     student04 = Student( "Marie", "Austin", date(2005, 8, 5), 2026)
29     student04.setGpa(3.9), student04.setSSN(1243)
30
31     student05 = Student( "Albert", "Austin", date(1999, 10, 20), 2026)
32     student05.setGpa(3.8), student05.setSSN(3124)
33
34     student06 = Student( "Aaron", "Austin", date(2002, 12, 2), 2026)
35     student06.setGpa(4.0), student06.setSSN(1131)
36
37     print("--- ")
38     print("--- Part 2: String description of student parameters ...")
39
40     print( student01.__str__() )
41     print( student02.__str__() )
42     print( student03.__str__() )
43     print( student04.__str__() )
44     print( student05.__str__() )
45     print( student06.__str__() )
46
47     print("--- ")
48     print("--- Part 3: Add students to list ... ")
49
50     studentList = [];
51     studentList.append(student01)
52     studentList.append(student02)
53     studentList.append(student03)
```

Example 9: Create List of Student Objects

Part II: Assemble Student Objects ...

```
54     studentList.append(student04)
55     studentList.append(student05)
56     studentList.append(student06)
57
58     print("--- ")
59     print("--- Part 4: Print contents of list ... ")
60
61     i = 0
62     for student in studentList:
63         print ("---   list01[{:d}]: {:6s} --> {:.2f} ...".format( i, student.getFirstName(), student.getGpa()))
64         i = i + 1
65
66     print("--- ")
67     print("--- Part 5: Sort list items by first name ... ")
68
69     sort_values = sorted( studentList, key = lambda x: x._firstname )
70
71     i = 0
72     for student in sort_values:
73         print ("---   list01[{:d}]: {:6s} --> {:.2f} ...".format( i, student.getFirstName(), student.getGpa()))
74         i = i + 1
75
76     print("--- ")
77     print("--- Part 6: Sort list items by gpa ... ")
78
79     sort_values = sorted( studentList, key = lambda x: x._gpa )
80
81     i = 0
```

Example 9: Create List of Student Objects

Part II: Assemble Student Objects ...

```
82     for student in sort_values:
83         print ("---   list01[{:d}]: {:6s} --> {:.2f} ...".format( i, student.getFirstName
84             i = i + 1
85
86     print("--- ===== ... ");
87     print("--- Finished TestStudents02.main()           ... ");
88
89     # call the main method ...
90
91     main()
```


Example 9: 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 9: 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 ...
```

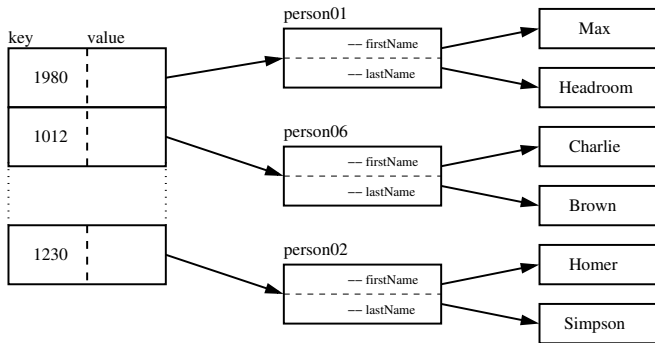
Example 9: Create List of Student Objects

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 10: 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 10: Dictionary of Fictional Characters

Part II: Dictionary of Fictional Characters

```
1  # =====
2  # TestDictionary03.py: Create dictionary of objects ...
3  #
4  # Last Modified: February 2023
5  # =====
6
7  from Person import Person
8
9  # main method ...
10
11 def main():
12     print("--- Enter TestDictionary03.main()    ... ");
13     print("--- ===== ... ");
14
15     # Create cartoon characters ...
16
17     print ("--- Part 01: Create cartoon character objects ...")
18
19     person01 = Person( "Max", "Headroom" )
20     person01.setAge(42)
21     person01.setSSN(1980)
22
23     person02 = Person( "Homer", "Simpson" )
24     person02.setAge(55)
25     person02.setSSN(1230)
```

Example 10: Dictionary of Fictional Characters

Part II: Dictionary of Fictional Characters:

```

27     person03 = Person( "Bart", "Simpson" )
28     person03.setAge(35)
29     person03.setSSN(1231)
30
31     person04 = Person( "Yogi", "Bear" )
32     person04.setAge(65)
33     person04.setSSN(1111)
34
35     person05 = Person( "Charlie", "Brown" )
36     person05.setAge(72)
37     person05.setSSN(1012)
38
39     print ( "--- " )
40     print ( "---- Part 02: Print sample objects ...")
41     print ( "---- " )
42
43     print( "---- person01 --> {:s} ...".format(person01.__str__() ) )
44     print( "---- person05 --> {:s} ...".format(person05.__str__() ) )
45
46     print ( "---- " )
47     print ( "---- Part 03: Assemble dictionary of cartoon characters ...")
48
49     cartoon = {}
50     cartoon[ person01.getSSN() ] = person01
51     cartoon[ person02.getSSN() ] = person02
52     cartoon[ person03.getSSN() ] = person03
53     cartoon[ person03.getSSN() ] = person03

```

Example 10: Dictionary of Fictional Characters

Part II: Dictionary of Fictional Characters:

```
54     cartoon[ person04.getSSN() ] = person04
55     cartoon[ person05.getSSN() ] = person05
56
57     print ("--- ")
58     print ("--- Part 04: Retrieve items from dictionary ...")
59     print ("--- ")
60
61     key = 1980
62     personItem = cartoon.get(key)
63     print("--- key = {:d} --> {:s} ...".format( key, personItem.__str__() ) )
64
65     key = 1230
66     personItem = cartoon.get(key)
67     print("--- key = {:d} --> {:s} ...".format( key, personItem.__str__() ) )
68
69     key = 1231
70     personItem = cartoon.get(key)
71     print("--- key = {:d} --> {:s} ...".format( key, personItem.__str__() ) )
72
73     key = 1111
74     personItem = cartoon.get(key)
75     print("--- key = {:d} --> {:s} ...".format( key, personItem.__str__() ) )
76
77     key = 1012
78     personItem = cartoon.get(key)
79     print("--- key = {:d} --> {:s} ...".format( key, personItem.__str__() ) )
```

Example 10: Dictionary of Fictional Characters

Part II: Dictionary of Fictional Characters:

```

81     print ("--- ")
82     print ("--- Part 04: Convert dictionary to list ...")
83
84     keysList = list( cartoon.keys() )
85     cartoonlist = [];
86     for person in keysList:
87         cartoonlist.append( cartoon.get(person) )
88
89     print ("--- ")
90     print ("--- Part 05: Sort list of cartoon items by age ...")
91     print ("--- ")
92
93     sorted_items = sorted( cartoonlist )
94
95     i = 1
96     for person in sorted_items:
97         print ("---   person[%d]: %s --> %s ..." % ( i, person.getFirstName(), person.getA
98             i = i + 1
99
100    print ("--- ===== ... ");
101    print ("--- Leave TestDictionary03.main() ... ");
102
103    # call the main method ...
104
105    main()

```


Example 10: 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 10: Dictionary of Fictional Characters

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

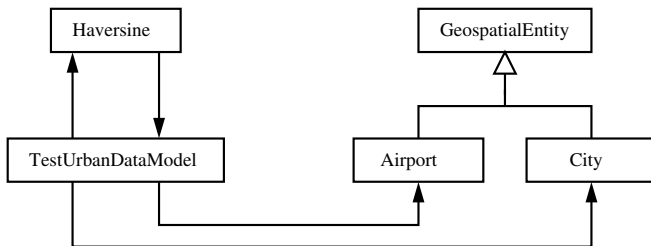
```
--- Part 05: Convert dictionary to list ...  
---  
--- Part 06: Sort list of cartoon items by age ...  
---  
---   person[1]: Bart --> 35 ...  
---   person[2]: Max --> 42 ...  
---   person[3]: Homer --> 55 ...  
---   person[4]: Yogi --> 65 ...  
---   person[5]: Charlie --> 72 ...  
--- ===== ...  
--- Leave TestDictionary03.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



The diagram shows a sphere representing Earth with a map of the North Atlantic and surrounding regions. A green great-circle arc connects two points marked with red dots: one in North America and one in Europe. A red arrow points from the text 'GIVES THE GREAT-CIRCLE DISTANCE FROM 2 POINTS' LATITUDE + LONGITUDE' to this arc.



The diagram shows a sphere with a green arc representing a great-circle path between two points. A blue line segment connects the two points, representing the straight-line distance through the sphere. A red arrow points from the text 'GIVES THE GREAT-CIRCLE DISTANCE FROM 2 POINTS' LATITUDE + LONGITUDE' to this arc.

$$\text{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right)$$

HANDY FOR NAVIGATION

sketchplanations

Case Study: GeoModeling Spatial Entities

Haversine Formula: Source code ...

```
1  # =====
2  # Haversine.py. Small class that provides approximate distance (km) between
3  # two points using the Haversine formula.
4  #
5  # Call in a static context:
6  #
7  # Haversine.distance(47.6788206, -122.3271205,
8  #                    47.6788206, -122.5271205) --> 14.973190481586224 [km]
9  #
10 # earthRadius = 6372.8;      # Earth radius in KM
11 # earthRadius = 3959.87433 # Earth radius in miles.
12 #
13 # Written by: Jason Winn (http://jasonwinn.org)
14 # Modified by: Mark Austin                                     February 2023
15 # =====
16
17 from math import radians, cos, sin, asin, sqrt
18
19 class Haversine:
20
21     # =====
22     # Compute haversine distance ...
23     # =====
24
25     @staticmethod
26     def distance(lat1, lon1, lat2, lon2):
```

Case Study: GeoModeling Spatial Entities

Haversine Formula: Source code ...

```
27     earthRadius = 3959.87433 # Earth radius in miles.
28     dLat = radians(lat2 - lat1)
29     dLon = radians(lon2 - lon1)
30     lat1 = radians(lat1)
31     lat2 = radians(lat2)
32
33     a = sin(dLat/2)**2 + cos(lat1)*cos(lat2)*sin(dLon/2)**2
34     c = 2*asin(sqrt(a))
35
36     return earthRadius * c
```

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

Case Study: GeoModeling Spatial Entities

Compute Distance between Washington DC and NYC

```
1  # =====
2  # TestHaversine.py: Small test program for haversine formula.
3  # =====
4
5  from Haversine import Haversine
6  from City import City
7  from Airport import Airport
8
9  # main method ...
10
11 def main():
12     print("--- Enter TestHaversine.main()      ... ");
13     print("--- ===== ... ");
14
15     print("--- Part 1: Create sample cities and airports ... ");
16
17     city01 = City( "Washington DC", 38.907192, -77.036871, 410.0, 5 )
18     city02 = City(      "Baltimore", 39.290385, -76.612189, 480.0, 10 )
19     city03 = City(      "New York", 40.712784, -74.005941, 265.0, 10 )
20
21     airport01 = Airport( "Baltimore-Washington", "BWI", 39.177404, -76.668392, 148.0 );
22     airport02 = Airport( "Washington Dulles",      "IAD", 38.952934, -77.447741, 313.0 );
23
24     print("--- Part 2: Print details of cities and airports ... ");
25
26     print(city01);      print(city02); print(city03)
```


Case Study: GeoModeling Spatial Entities

Compute Distance between Washington DC and NYC

```
27     print(airport01); print(airport02)
28
29     print("--- Part 3: Compute distances between locations ... ");
30
31     # Compute distance between Washington DC and Baltimore ...
32
33     lat1 = city01.getLatitude(); lon1 = city01.getLongitude()
34     lat2 = city02.getLatitude(); lon2 = city02.getLongitude()
35     d1 = Haversine.distance(lat1, lon1, lat2, lon2)
36
37     print("--- Distance: Washington DC to Baltimore --> {:f} miles ..".format(d1))
38
39     # Compute distance between Washington DC and New York ...
40
41     lat1 = city01.getLatitude(); lon1 = city01.getLongitude()
42     lat2 = city03.getLatitude(); lon2 = city03.getLongitude()
43
44     d1 = Haversine.distance(lat1, lon1, lat2, lon2)
45
46     print("--- Distance: Washington DC to New York --> {:f} miles ..".format(d1))
47
48     # Compute distance between IAD and BWI ...
49
50     lat01 = airport01.getLatitude(); lon01 = airport01.getLongitude()
51     lat02 = airport02.getLatitude(); lon02 = airport02.getLongitude()
52
53     d1 = Haversine.distance( lat01, lon01, lat02, lon02)
```

Case Study: GeoModeling Spatial Entities

Compute Distance between Washington DC and NYC

```
55
56     code01 = airport01.getAirportCode()
57     code02 = airport02.getAirportCode()
58     print("--- Distance: {:s} to {:s} --> {:f} miles ..".format( code01, code02, d1))
59
60     print("--- ===== ... ");
61     print("--- Leave TestHaversine.main()      ... ");
62
63     # call the main method ...
64
65     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

-

-