Inheritance

The 2002 ACM Turing Award was given to two Norwegian computer scientists, for their early work in object-oriented programming. In the 1960s, Nygaard and Dahl designed a language for simulations they called SIMULA. SIMULA-67 was the first language to use the object-oriented paradigm that is now widely adopted in the field.

In addition to his work in computer science, Kristen Nygaard also took an active role in making sure that computers served the interests of workers, not just their employers. In 1990, Nygaard's social activism was recognized in the form of the Norbert Wiener Award for Social and Professional Responsibility.

Object-Oriented Programming

Class Hierarchies

- Much of the power of modern object-oriented languages comes from the fact that they support class hierarchies. Any class can be designated as a subclass of some other class, which is called its superclass.
- Each subclass represents a specialization of its superclass. If you create an object that is an instance of a class, that object is also an instance of all other classes in the hierarchy above it in the superclass chain.
- When you define a new class in Python, it automatically inherits the behavior of its superclass.
- Although Python supports multiple inheritance in which a class can inherit behavior from more than one superclass, most class hierarchies use single inheritance in which each class has a unique superclass.

The GObject Hierarchy

Specifying Inheritance in Python

- Subclass definitions in Python differ from standalone class definitions in two ways:
  1. A subclass definition includes the name of its superclass in parentheses after the name of the subclass.
  2. The constructor for the subclass explicitly calls the constructor for its superclass before performing its own initialization.
- Each of these features is illustrated in the following pattern:

```python
class subclass (superclass):
    def __init__(self, parameters):
        superclass.__init__(self, any relevant parameters)
```

Designing an Employee Hierarchy

- The Employee class I introduced on Friday assumed that all employees can be represented using a structure that includes the same fields for every employee. When you work in the real world, data structures tend to be more complex.
- For example, most companies have several different types of employees. Some employees are paid hourly, others are paid a commission based on their sales volume, and others are paid a fixed salary. The program that writes the payroll checks must take these differences into account.
- In an object-oriented design, a general class called Employee will be broken down into several subclasses representing the different employee types. Shared behavior is implemented at the level of the base class, but behavior that differs depending on the employee type is implemented in each subclass.
Classes in the Employee Hierarchy

- The following diagram shows a simple Employee hierarchy:

```
Employee
  getName()
  getTitle()
  getPay()

SalariedEmployee
  setSalary(salary)
  getPay()

CommissionedEmployee
  setBaseSalary(dollars)
  setCommissionRate(rate)
  setSalesVolume(dollars)
  getPay()

HourlyEmployee
  setHourlyRate(wage)
  setHoursWorked(hours)
  getPay()
```

- All employees share the getName and getTitle methods. The getPay method is defined for all employees, but implemented differently by each subclass.

Exercise: CommissionedEmployee

- The CommissionedEmployee subclass is paid a fixed base salary plus a specified commission on the total sales volume for which that employee is responsible.
- These values are set for a CommissionedEmployee object by calling the methods setBaseSalary, setCommissionRate, and setSalesVolume.
- The CommissionedEmployee subclass defines its own version of the getPay method that calculates the correct pay based on these stored values.
- In the language of object-oriented programming, the getPay method in the subclass overrides the definition in Employee, which is defined at that level to inform clients that all Employee objects respond to that method.

Exercise: Design a Subclass

- Last year’s CS qual included the following question (the code for the Car class appears on the next slide):

```
Figure 1 presents some code for a Car class. Write the code for a new class, Delivery, used to represent delivery trucks for a particular company. Delivery trucks are like cars, but rather than allowing for customization, these trucks should always have a gas tank that holds 50 gallons and gets 15 miles per gallon. Furthermore, there should be an additional method, idle, which takes as input a time measured in minutes and represents the truck idling at a stop for that many minutes. The truck, when idling, burns two gallons per hour. (If a truck is asked to idle for long enough to burn more gas than there is in the tank, the tank just becomes emptied.)
```

- Spend the next five minutes or so talking about this problem with your neighbors and writing out a solution in Python.