CS 162P
Inheritance

Inheritance is the act of deriving a new class from an existing class. The existing class is normally called the parent class and the new class that is created from it is called the child class. While class composition is referred to as a “has a” relationship, inheritance represents an “is a” relationship. A car “has a” engine, but “is a” vehicle.

The child class inherits the behaviors (methods) and data (variables) of the parent class. Thus, it is possible to create a parent with basic functionality and then create children that specialize in one way or another. For example, consider a college where there are people. This could be the base or parent class that includes such things as name, identifier, address, age, and image. Then one derived or child class could be student, which adds the variable courses with appropriate methods. Another child class could be instructor, which adds the variables office hours and courses with appropriate methods. A third child class could be manager, which adds the variable peopleManaged. Each of these child classes, student, instructor, or manager, has all of the characteristics of a person plus the specialized features of its own class.

Creating a Child Class

When creating a new class, it can either be created using inheritance or created as a stand-alone class. The classes created previously in this course have all been stand-alone classes, they have not had a parent.

In the example below, the class Vehicle is a parent class, and the class Bicycle inherits from Vehicle.

```
class Vehicle:
    def   init   (self):
        self.  wheels = 0

    def setWheels(self, wheels):
        self.  wheels = wheels

    def getWheels(self):
        return self.  wheels

    def getPower(self):
        return "undefined"

class Bicycle(Vehicle):
    def   init   (self):
        super().setWheels(2)

    def getPower(self):
        return "pedaling"
```

In this example, Bicycle is shown as being a child of Vehicle by the parentheses containing the parents when declaring that Bicycle is a class, class Bicycle(Vehicle):.

Because Bicycle is a child of Vehicle, it inherits the methods setWheels and getWheels from Vehicle and the following code works, even though Bicycle does not explicitly include either of the two methods.

```
def main():
    bike = Bicycle()

    print(bike.getWheels())
    print(bike.getPower())
```

Since a Bicycle has two wheels while a Vehicle has none, it is necessary for the __init__ method for Bicycle to set that parameter in its parent. Since __wheels is a private variable, it is necessary for Bicycle to use the setWheels() method to set it. Also, to make sure that the proper instance of Vehicle gets its wheels set to two, the super() method is called.

Protected Access

Previously it was explained that adding two underscores before a method or variable made it private while leaving them off made it public. Public methods and variables can be accessed by anyone. Private methods and variables can only be accessed by members of the class itself. There is a third access level protected. It is not strictly enforced in Python but is a convention. If a method or variable starts with a single underscore it is considered protected and should only be accessed by members of the class and its children.

Child Parent Access

A child class includes all public and protected methods of the parent class. The child class can directly access any public or protected variables of the parent class. The child class cannot see or use any private variables or methods of the parent class. If the child is to access such private variables, it must use public methods to do so. For example, in the Vehicle class above, the variable wheels is private, so the Bicycle class is not able to directly access it so it must use setWheels or getWheels.

Method Overriding

It is possible for a child class to add new methods or variables or to redefine existing parent methods. Adding new methods or variables is done in the same way as defining methods or variables in any class and has nothing to do with inheritance per se.

When a method in a child class has the same name and parameters as a method in the parent, it is said to override the parent method. That is, if an object of type child invokes the method the child version is called. If an object of type parent invokes the method, the parent method is called.

An example of method overriding is the getPower method in Vehicle and Bicycle. Here objects of each type are shown invoking the same named method, but with differing results.

Program:

```
def main():
    thing = Vehicle()
    bike = Bicycle()

    print(thing.getPower())
    print(bike.getPower())
```

Output:

```
undefined
pedaling
```

Parent __init__

The parent __init__ method is called before the child __init__ to initialize the parent portion of the object first. Sometimes it is necessary for the child to initialize the parent as shown here.

```
class Parent:
    def   init   (self, name= "", age= 0):
        self.  name = name
        self.  age = age

    def getName(self):
        return self.  name

    def getAge(self):
        return self.  age

    def setName(self, name):
        self.  name = name

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

class Child(Parent):
    def   init    (self, name, age, hobby):
        super().setAge(age)
        super().setName(name)
        self.  hobby = hobby

    def getHobby(self):
        return self.  hobby

    def setHobby(self, hobby):
        self.__hobby = hobby
```

This shows one way to set the name and age of the Parent class from the Child, calling the public setters of the Parent or super in the __init__ method of the Child.

A second method is to call the Parent's __init__ method in the Child's __init__ method as shown below.

```
class Child(Parent):
    def __init__(self, name, age, hobby):
        super().__init__(name, age)
        self.__hobby = hobby
```