Inheritance in Python

CC3 – Object Oriented Programming

Introducing Inheritance

- To recall what this term means, **inheritance** allows us to create "is a" relationship between two or more classes.
- This allows us to abstract common logic into super-classes and manage specific details in the subclass.
- The superclass is the "parent" that has all the methods and attributes.
- The subclass is the "child" that inherits all the methods and attributes of the superclass.

Introducing Inheritance

- Technically, every class we create in Python uses inheritance.
- All Python classes are subclasses of the special class named *object*.
- This class does not really do much in terms of attributes and behaviors, but it does allow Python to treat all object in the same way.
- If we don't explicitly inherit from a different class, our classes will automatically inherit from object.

Introducing Inheritance

- A new class can be derived from an existing class as follows:

```
class MyClass(object):
    pass
```

- In Python 2.2 and Python 3 onwards, all classes implicitly inherit from the object class.
- This means we do not need to type object when defining the parent class.
- This was just shown to demonstrate how Python tells classes they can be inherited

Introducing Inheritance

- A new class can be derived from an existing class as follows:

```
class SuperClass(object):
        pass
class SubClass(SuperClass):
        pass
```

- This syntax tells Python that the “SubClass” should be derived from the “SuperClass”

Introducing Inheritance

- Now that we know how to define subclass and superclass, let us see how it can be used.
- The most straightforward use of inheritance is to add functionality to an existing class.
- Let us have some examples of how this can be used on the next slides.

Using Inheritance

- Here is a simple example of how inheritance can be implemented:

```
class Cat:
        def __init__(self, breed, color):
                self.breed = breed
                self.fur_color = color
        def meow (self):
                print("The cat meows.")
```

Using Inheritance

- Here is a simple example of how inheritance can be implemented:

```
class PersianCat(Cat):
      pass
persian_cat1 = PersianCat("Persian Cat", "White")
print(persian_cat1.fur_color)
Persian_cat1.meow()
```

Using Inheritance

- As seen in the previous example, the "`PersianCat()`" class inherits the attributes and methods of the "`Cat()`" class.
- This also includes the "`__init__`" method, which means the child class will also need to instantiate an object.
- When we would like to use the attributes in the "`Cat()`" class, we simply need to call them.
- We do not need to define the same attributes and methods in our child class "`PersianCat()`".

The super() function

- The **super** function returns the object as an instance of the parent class, which allows us to call the parent method directly.
- This essentially allows a child class to call all the methods of the parent class.
- This is meant to allow code reusability.
- This allows you to override methods in your child class, but still access the “original” methods in your parent class.
-

Using Inheritance

- We define a super class "Rectangle" with the following attributes and behaviors:

```
class Rectangle:
      def __init__ (self, width, height):
          self.width = width
          self.height = height
      def area(self):
          return self.width * self.height
       def perimeter (self):
          return 2 * (self.width * self.height)
```

Using Inheritance

- We can now define a subclass “Square”
- Since a square is simply a rectangle with equal sides, these two shapes share characteristics and can be inherited.
- Here is the source code example for the subclass “Square”:

```
class Square (Rectangle):
    def __init__(self, side):
        super().__init__(side, side)
        self.side = side
```

Using Inheritance

- We use the “super()” method to allow us to set the sides of the square with the help of the parent class.
- What is happening is we are asking the user to give a value for the square when it is instantiated.
- We then pass the value as both the height and width to the “Rectangle” class initializer.
- This allows us to set the sides of the square to be equal.

Using Inheritance

- Once you have your superclass and subclass linked together, you can then start creating objects for both classes.
- Keep in mind that objects created for their class are independent of each other.
- Here are some examples of objects created along with their outputs:
  - `rectangle_example = Rectangle (2, 3)`
  - `print(rectangle_example.area())`
  - `print (rectangle_example.perimeter())`

Using Inheritance

- We can now also create objects for our “Square()” class by passing only one value:
  - `square_example = Square.(5)`
  - `print (square_example.area())`
  - `Print (square_example.perimeter())`

Overriding Class Behavior

Object Oriented Programming

Overriding Class Behavior

- Inheritance allows us not to only add new behavior to existing classes but change behavior.
- **Overriding** means altering or replacing a method of the superclass with a new method in the subclass.

Overriding Class Behavior

- When overriding methods, you are usually only overriding methods of the same name.
- No special syntax is needed to do this.
- You simply need to recreate the method you would like to override.
- The subclass's newly created method is automatically called instead of the superclass's method.