CS: Objects in Python

Inheritance

Extending & Overriding

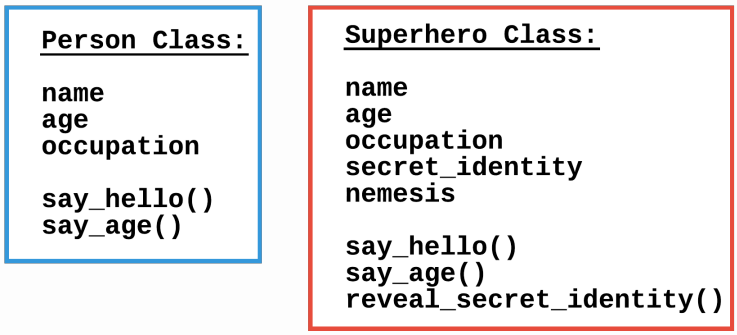
Learning Objectives - Extending & Overriding

- **Define the terms extending and overriding**
- **Extend the parent class with new method**
- **Override methods from the parent class with new functionality**

Extending a Class

Extending the `__init__` Method

The idea of inheritance is to borrow from a parent class and then add on functionality. Up until now, we have talked about borrowing from a parent class. The process of adding functionality to a child class is known as either extending or overriding. Extending a class means that new attributes and methods are given to the child class.



Superhero and Person Classes

The code below will first call upon the `__init__` method (using `super()`) of the parent class to create the attributes `name`, `age`, and `occupation`. The `__init__` method is extended when the attribute `secret_identity` is added to the `Superhero` class.

```
class Superhero(Person):
  def __init__(self, name, age, occupation, secret_identity):
    super().__init__(name, age, occupation)
    self.secret_identity = secret_identity

hero = Superhero("Spider-Man", 17, "student", "Peter Parker")
print(hero.secret_identity)
```

▼ **Inheritance is a One-Way Street**

Inheritance shares attributes and methods from the parent class to the child class. When a child class is extended, it cannot share the new additions with their parent class. In the code above, `Superhero` has access to `name`, but `Person` does not have access to `secret_identity`.

challenge

Try this variation:

- Rewrite the `Superhero` class so that it extends the `Person` class by adding the attribute `nemesis`, Doc Octopus.
  ▼ **Solution**

```
class Superhero(Person):
  def __init__(self, name, age, occupation, secret_identity, nemesis):
    super().__init__(name, age, occupation)
    self.secret_identity = secret_identity
    self.nemesis = nemesis

hero = Superhero("Spider-Man", 17, "student", "Peter Parker", "Doc Octopus")
print(hero.nemesis)
```

Extending a Class by Adding New Methods

Another way to extend a class is to create new methods that are unique to the child class. For example, the `say_hello` method will give the superhero's name, but it will not divulge their secret identity. Create the method `reveal_secret_identity` to print the attribute `secret_identity`.

```
class Superhero(Person):
  def __init__(self, name, age, occupation, secret_identity, nemesis):

    super().__init__(name, age, occupation)
    self.secret_identity = secret_identity

  def reveal_secret_identity(self):
    print(f"My real name is {self.secret_identity}.")

hero = Superhero("Spider-Man", 17, "student", "Peter Parker", "Doc Oct

hero.reveal_secret_identity()
```

challenge

Try this variation:

- Create the method `say_nemesis` that prints the string:
  `My nemesis is Doc Octopus.`.
  ▼ **Solution**

```
class Superhero(Person):
  def __init__(self, name, age, occupation, secret_identity, nemesis):
    super().__init__(name, age, occupation)
    self.secret_identity = secret_identity
    self.nemesis = nemesis

  def reveal_secret_identity(self):
    print(f"My real name is {self.secret_identity}.")

  def say_nemesis(self):
    print(f"My nemesis is {self.nemesis}.")

hero = Superhero("Spider-Man", 17, "student", "Peter Parker", "Doc Octopus")
hero.say_nemesis()
```

Method Overriding

Overriding a Method

Extending a class means adding new attributes or methods to the child class. Another way to add new functionality to a child class is through method overriding. Overriding a method means to inherit a method from the parent class, keep its name, but change the contents of the method.

Extend the `Superhero` class by overriding the `say_hello`. Make a new instance of the class and call the method.

```
def say_hello(self):
  print(f"I am {self.name}, and criminals fear me.")

hero = Superhero("Storm", "30", "Queen of Wakanda", "Ororo Munroe", "S

hero.say_hello()
```

▼ **Differentiating Overriding and Extending**

The difference between extending and overriding can be slight. Both approaches are used to make a child class unique from the parent class. Overriding deals with changing a pre-existing method from the parent class, while extending deals with adding new methods and attributes.

challenge

Try this variation:

- Override the `say_age` method so that it prints the string:
  `Young or old, I will triumph over evil.`
  ▼ **Solution**

```
def say_age(self):
  print(f'Young or old I will trimph over evil')

hero = Superhero('Storm', 30, 'Queen of Wakanda', 'Oro Monroe', 'Shadow King
hero.say_hello()
hero.say_age()
```

What Happens When You Override a Method?

If you can override a method from the parent class, what happens to the original method? Using the `help` function, we can see a graphical representation of the parent and child classes. Enter the code below and run the program.

```
print(help(Superhero))
```

Notice that there is no section that says `Methods inherited from Person:`. Does that mean that the `Superhero` can no longer use the original `say_age` and `say_hello` methods? No, the child class can still call the methods from the parent class. However, calling `say_hello` or `say_age` will not print the same string as the parent class. Instead, using the `super()` keyword gives a child class access to the original methods. Add the following method to the `Superhero` class and then call it.

```
def old_say_hello(self):
  super().say_hello()

hero.old_say_hello()
```

challenge

Try this variation:

- Add the method `old_say_age` to the `Superhero` class and then call it.

▼ **Solution**

```
def old_say_age(self):
  super().say_age()

hero.old_say_age()
```