Python Programming: Class Inheritance

Learning Objectives

After this lesson, you will be able to…

- Implement inheritance.
- Describe what has been inherited from one class to another.
- Overwrite variables and methods.

Discussion: Similar Classes

`Phone` is a class — there are hundreds of types of phones.

- What attributes and functions would a `Phone` have?

What about an `iPhone`? Or `android_phone`?

- `iPhone` and `android_phone` would be objects of the `Phone` class.
- But, there are different types of iPhones and Android phones.
- Should `IPhone` and `AndroidPhone` be classes themselves?

What would you do?

Introduction: Inheritance

`AndroidPhone` and `IPhone` are separate classes *and* in the `Phone` class.

This is called **inheritance**: making classes that are subsets of other classes.

`Phone` is the **parent** class. It's a regular class! All phones:

- Have a phone number.
- Can place phone calls.
- Can send text messages.

`IPhone` is a **child** class. The child class **inherits** methods and properties from the parent class but can also define its own functionality. iPhones uniquely:

- Have an `unlock` method that accepts a fingerprint.
- Have a `set_fingerprint` method that accepts a fingerprint.

We Do: Inheritance

All phones have a phone number, can place phone calls, and can send text messages.

Start a new file, `Phone.py`. In it, let's start and test the class:

```
class Phone:

  def __init__(self, phone_number):
    self.number = phone_number


  def call(self, other_number):
    print("Calling from", self.number, "to", other_number)


  def text(self, other_number, msg):
    print("Sending text from", self.number, "to", other_number)
    print(msg)
```

We Do: `IPhone` Class

Underneath the `Phone` class definition, let's create the `IPhone` class.

```
class IPhone(Phone):
  # Class definitions accept a parameter specifying what class they inherit
  def __init__(self, phone_number):
    # super()` calls the `init` defined in the parent class.
    super().__init__(phone_number)
    # Now self.number is set, because that's what happens in the parent Phon
```

We Do: `IPhone` Class

iPhones uniquely:

- Have an `unlock` method that accepts a fingerprint.
- Have a `set_fingerprint` method that accepts a fingerprint.

```
class IPhone(Phone):
  def __init__(self, phone_number):
    super().__init__(phone_number)

    # Under the call to super, we can define unique IPhone variables.
    # Regular Phone objects won't have this!
    self.fingerprint = None

# Here are methods unique to IPhone objects:
def set_fingerprint(self, fingerprint):
  self.fingerprint = fingerprint
```

Side Discussion: Edge Cases

Look at:

```
def unlock(self, fingerprint=None):
  if (fingerprint == self.fingerprint):
    print("Phone unlocked. Fingerprint matches.")
  else:
    print("Phone locked. Fingerprint doesn't match.")
```

What if `self.fingerprint` is currently `None`? We need to account for this!

```
def unlock(self, fingerprint=None):
  if (self.fingerprint == None):
    print("Phone unlocked. No fingerprint needed.")
  elif (fingerprint == self.fingerprint):
    print("Phone unlocked. Fingerprint matches.")
  else:
    print("Phone locked. Fingerprint doesn't match.")
```

When programming, always watch for **edge cases**. This isn't specific to classes!

We Do: Testing `IPhone`

Add some test lines at the bottom:

```
my_iphone = IPhone(151)
my_iphone.unlock()
my_iphone.set_fingerprint("Jory's Fingerprint")
my_iphone.unlock()
my_iphone.unlock("Jory's Fingerprint")

# And we can call the Phone methods:
my_iphone.call(515)
my_iphone.text(51121, "Hi!")
```

Try it! Then, try this. Why does it fail?

```
# Let's try a Phone object on an iPhone method.
test_phone.unlock()
```

Quick Recap: Inheritance

- A class can inherit from another class — a parent class and a child class.
- The child class can declare its own variables and methods, but it also has access to all the parents'.

```
## Parent class: A regular class ##
class Phone:
  def __init__(self, phone_number):
    self.number = phone_number

  def call(self, other_number):
    print("Calling from", self.number, "to", other_number)

test_phone = Phone(5214) # It's a regular class!
test_phone.call(515)
```

I Do: Overwriting Attributes

Next up: Overwriting attributes!

Let's switch to a new example. You don't need to follow along.

Here's a regular `Building` class:

```
class Building(object):
 # Class variables
 avg_sqft = 12500
 avg_bedrooms = 3

# No __init__ - there are no instance variables to declare!
# This is possible in any class, not just inheritance. (Building is a norma

def describe_building(self):
 print('Avg. Beds:', self.avg_bedrooms)
 print('Avg. Sq. Ft.:', self.avg_sqft)
```

I Do: Inheriting Building

Inheriting from `Building`, we can create a `Mansion` class.

```
# Call in the parent, Building, to the class definition.
class Mansion(Building):
  # Our child class definition goes here.
  # Will have the same class variables, instance variables, and methods as M
```

Overwriting Variables

What if we want the class variables to have different values? We can set new ones. Remember, child classes do not affect the parent class.

```
class Mansion(Building):

    # Overwrite the class variables.

    avg_sqft = 6

    avg_bedrooms = 1


    # We don't have a call to super __init__. Why?

    # There's no __init__ in the parent to call!


### Now, let's try it out. ###

# This still has the old values.

my_building = Building()
```

Discussion: Child Class Methods

In the `Building` class, we have:

```
def get_avg_price(self):
 price = self.avg_sqft * 5 + self.avg_bedrooms * 15000
 return price
```

What if a `Mansion`'s price calculation is different? What do you think we can do?

Overwriting Methods

We know that we can overwrite variables. Turns out, we can also overwrite methods!

```
class Mansion(Building):


 def get_avg_price(self):
  return 1000000


mans = Mansion()
bldg = Building()


bldg.get_avg_price()
# # returns `self.avg_sqft * 5 + self.avg_bedrooms * 15000`
```

Quick Review

When we make child classes, we can overwrite class variables and methods.

```
class Building(object):

    # Class variables

    avg_sqft = 12500

    avg_bedrooms = 3


    def get_avg_price(self):

        price = self.avg_sqft * 5 + self.avg_bedrooms * 15000

        return price



class Mansion(Building):
```

Knowledge Check

Consider the following classes:

```
class Animal(object):
    def is_mammal(self):
        return True
    def is_alive(self):
        return True


class Grasshopper(Animal):
    def is_small(self):
        return True
```

You instantiate two objects: `bug = Grasshopper()` and `cat = Animal()`. Which of the following instance methods are available for each?

Summary and Q&A

Inheritance:

- Allows us to make classes using other classes as templates.
- Has a **parent** class (`Phone`) and a **child** class (`IPhone`).
    - The parent class is still a usable class!

Child classes:

- `inherit` methods and properties from a parent class.
- Have access to all of the functionality of its parent.
- Can have new attributes and methods.
    - They won't be available to the parent.
- Can overwrite values from the parent class.