CS: Objects in Python

Inheritance

Multiple Inheritance

Learning Objectives - Multiple Inheritance

- **Define multiple inheritance**
- **Override the `__init__` method to inherit attributes from both parent classes**
- **Define the inheritance hierarchy of an object with more than one parent class**

Multiple Inheritance

Multiple Inheritance

Up until now, inheritance has been introduced as there being only one parent class. Multiple inheritance is when there are more than one parent class. We can define the class `Dinosaur` as one parent class, and `Carnivore` as the other parent class.

In single inheritance, the parent class appears between parentheses. For multiple inheritance, put both parent classes between parentheses.

```
class Dinosaur:
  def __init__(self, size, weight):
    self.size = size
    self.weight = weight

class Carnivore:
  def __init__(self, diet):
    self.diet = diet

class Tyrannosaurus(Dinosaur, Carnivore):
  pass
```

▼ **What does `pass` do?**
Python expects there to be code in the body of the `Tyrannosaurus` class. However, we just want this class to inherit from its parents. Using `pass` satisfies Python's need to have a body for the class, but `pass` doesn't do anything.

Now we can instantiate an object from the `Tyrannosaurus` class. This t-rex `tiny` that is 12 meters tall, weighs 14 metric tons, and eats whatever it wants.

```
tiny = Tyrannosaurus(12, 14, "whatever it wants")
print(tiny.size)
```

What Went Wrong?

In multiple inheritance, there are two `__init__` methods (one from each parent) that the class `Tyrannosaurus` inherits. Python is not sure how to take the list of parameters for `tiny` and divide them between the two `__init__` methods. That is why you saw an error message when you ran your code. Rewrite the instantiation of `tiny` as shown below.

```
tiny = Tyrannosaurus(12, 14)
print(tiny.size)
```

Why does it work now? The `Tyrannosaurus` class inherits from two classes. One parent has two parameters for its constructor, while the other has one parameter. Python throws an error when three parameters are passed to `Tyrannosaurus`, but runs just fine when two parameters are passed. Look at how the `Tyrannosaurus` class was defined.

```
class Tyrannosaurus(Dinosaur, Carnivore):
  pass
```

`Dinosaur` is listed as the first parent class, so Python uses the constructor from the `Dinosaur` class. Since `Dinosaur` takes two parameters, the `Tyrannosaurus` class works with two parameters.

challenge

Try this variation:

- Change the order of the parent classes when defining the `Tyrannosaurus` class. Instantiate an instance as, `tiny = Tyrannosaurus("whatever it wants")`. Finally, print out the `diet` attribute.

  ▼ **Solution**

```
class Tyrannosaurus(Carnivore, Dinosaur):
    pass

tiny = Tyrannosaurus("whatever it wants")
print(tiny.diet)
```

Override the init Method

Overriding the `__init__` Method

In order to have access to the attributes from all the parent classes, you need to override the `__init__` method of the child class. Previously, you used the `super()` keyword to refer to methods in the parent class. This **will not** work for multiple inheritance. Look at the code below:

```
class Tyrannosaurus(Dinosaur, Carnivore):
  def __init__(self, size, weight, diet):
    super().__init__(size, weight)
    super().__init__(diet)
```

Python does not know if `super()` is referring to the `Dinosaur` class or the `Carnivore` class. Overriding the `__init__` method for multiple inheritance requires a unique structure. Instead of `super()` use the name of the class. You must also pass `self` to the parent `__init__` method.

```
class Dinosaur:
  def __init__(self, size, weight):
    self.size = size
    self.weight = weight

class Carnivore:
  def __init__(self, diet):
    self.diet = diet

class Tyrannosaurus(Dinosaur, Carnivore):
  def __init__(self, size, weight, diet):
    Dinosaur.__init__(self, size, weight)
    Carnivore.__init__(self, diet)

tiny = Tyrannosaurus(12, 14, "whatever it wants")
```

▼ **Where is the output?**

You should see a green check mark that the code ran properly. There are

no `print` statements, so you should not see any output except for the green check mark.

challenge

Try this variation:

- Verify that the `__init__` method has been overridden by printing out the `size`, `weight`, and `diet` attributes for `tiny`.
  ▼ **Solution**

```
tiny = Tyrannosaurus(12, 14, "whatever it wants")
print(tiny.size)
print(tiny.weight)
print(tiny.diet)
```

Multiple Inheritance Hierarchy

Is an Instance or is a Subclass?

Like single inheritance, the `isinstance` function works with with multiple inheritance. The code below prints `True` if `obj` is an instance of the `D` class. It will print `False` if `obj` is not an instance of `D`. The code also prints another `True` because `D` is a subclass (child) of `A`.

```
class A:
  pass

class B:
  pass

class C:
  pass

class D(A, B):
  pass

obj= D()
print(isinstance(obj, D))
print(issubclass(D, A))
```

challenge

Try this variation:

- Change the `isinstance` function call to `print(isinstance(obj, A))`
- Change the `isinstance` function call to `print(isinstance(obj, B))`
- Change the `isinstance` function call to `print(isinstance(obj, C))`
- Change the `issubclass` function call to `print(issubclass(D, B))`
- Change the `issubclass` function call to `print(issubclass(D, C))`

Method Resolution Order

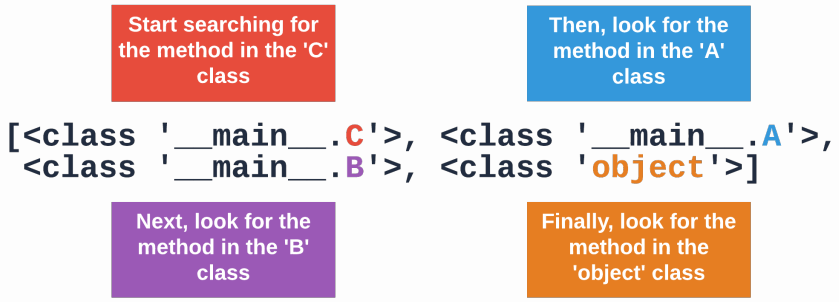
Look at the code below. Class `C` is the subclass of `A` and `B`. Both parent classes have a method called `hello` which prints either `Hello from class A` or `Hello from class B`. Class `C` overrides `hello` and calls `super().hello()`. The keyword `super()` refers to the `hello` method of the parent class. However, there are two parent classes. What do you think the result will be?

```
class A:
  def hello(self):
    print("Hello from class A")

class B:
  def hello(self):
    print("Hello from class B")

class C(A, B):
  def hello(self):
    super().hello()

obj= C()
obj.hello()
```

If `super()` cannot be used when overriding the `__init__` method, why does Python print `Hello from class A` and not an error message? Python has something called method resolution order (MRO). MRO is the way Python looks for methods in parent classes. Modify the end of your program so it looks like the code below. Then click the `TRY IT` button.

```
obj= C()
print(C.mro())
```

The code should print a list of object names. The order of objects in the list represents the order Python uses to search for a method. **Note:** all classes in Python are a child class of the `object` class.



MRO

So when class `C` says `super().hello()`, Python skips class `C` because of the `super()` keyword. Then it moves on to class `A`. The `hello` method is present in class `A`, so it prints `Class A`. Then Python stops searching the rest of the classes, which why `Class B` is not printed even though the `hello` method is present in class `B`.

challenge

Try this variation:

- Change the definition for class `C` so that `B` comes before `A`. Then print the output of the `mro()` method on class `C`.

```
class C(B, A):
  def describe(self):
    super().hello()

print(C.mro())
```

▼ **What is happening?**
MRO is defined by the order of parent classes. So `class C(A, B):` puts class `A` before class `B`. Writing `class C(B, A):` will search class `B` before class `A.

- Now that the MRO has been changed, call the `hello` method.

```
obj.hello()
```

▼ **What is happening?**
The output should now be `Hello from class B` since the MRO is changed to search class `B` first. Once Python finds the `hello` method in class `B`, it stops searching.

Extending and Overriding Methods

Extending a Class with Multiple Inheritance

Multiple inheritance has no effect on extending a child class. The `bonjour` method is not present in either parent class. There is no need for special syntax to extend `C`.

```
class A:
  def hello(self):
    print("Hello from class A")

class B:
  def hello(self):
    print("Hello from class B")

class C(A, B):
  def bonjour(self):
    print("Bonjour")

obj= C()
obj.bonjour()
```

challenge

Try this variation:

- Extend `C` with the method `goodbye` that prints `Goodbye`. Then call this method.
  ▼ **Solution**

```
class C(A, B):
  def goodbye(self):
    print("Goodbye")

obj= C()
obj.goodbye()
```

Overriding a Method with Multiple Inheritance

With the exception of the `__init__` method, overriding a method works the same in multiple inheritance as it does in single inheritance.

```
class A:
  def hello(self):
    print("Hello from class A")

class B:
  def hello(self):
    print("Hello from class B")

class C(A, B):
  def hello(self):
    print("Hello from class C")

obj= C()
obj.hello()
```

In single inheritance, you can use the `super()` keyword to invoke methods from the parent class. Because of method resolution order, you can do the same thing with multiple inheritance. The one exception is when both parent classes have a method with the same name.

```
class C(A, B):
  def hello(self):
    print("Hello from class C")
    super().hello()

obj= C()
obj.hello()
```

Because of MRO, `super()` refers to the `hello` method in class `A`. To refer to the `hello` method in class `B`, you will use the same format as when you overrode the `__init__` method.

```
class C(A, B):
  def hello(self):
    print("Hello from class C")
    super().hello()
    B.hello(self)

obj= C()
obj.hello()
```