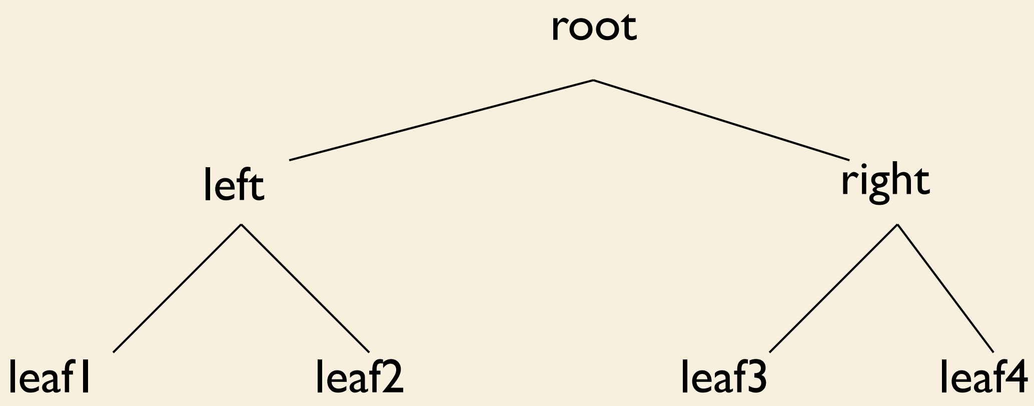
INHERITANCE (ERFÐIR)

CLASSES

- Remember the relationship between a class and its instances
  - a class can have many instances, each made initially from the constructor of the class
  - the methods an instance can call are initially shared by all instances of a class
- Classes can also have a separate relationship with other classes
  - These relationships form a hierarchy

CLASSES

- The hierarchy forms what is called a tree in computer science



root

left

right

leaf1

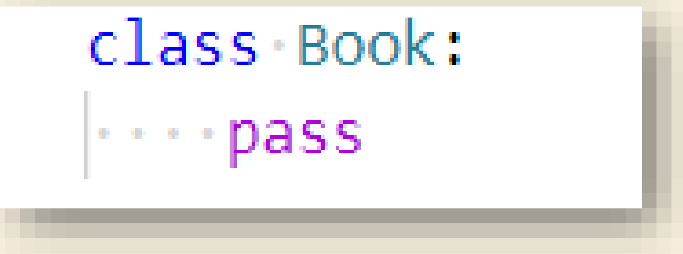
leaf2

leaf3

leaf4

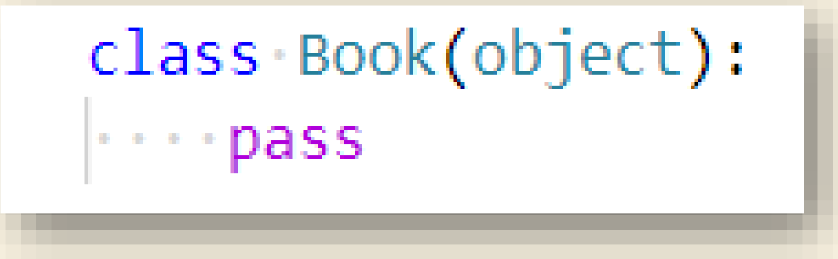
CLASSES

- when we create a class, which is itself another object, we can state how it is related to other classes
- the relationship we can indicate is the class that is 'above' it in the hierarchy
- The top class in Python is called `object`
  - It is predefined by Python and it always exists
  - Every class we create are related to this class and implicitly inherit it



```python
class Book:
    pass
```

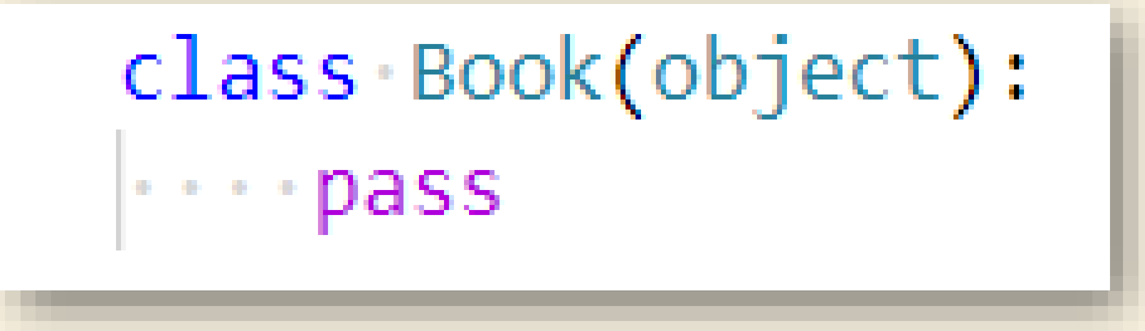
=



```python
class Book(object):
    pass
```

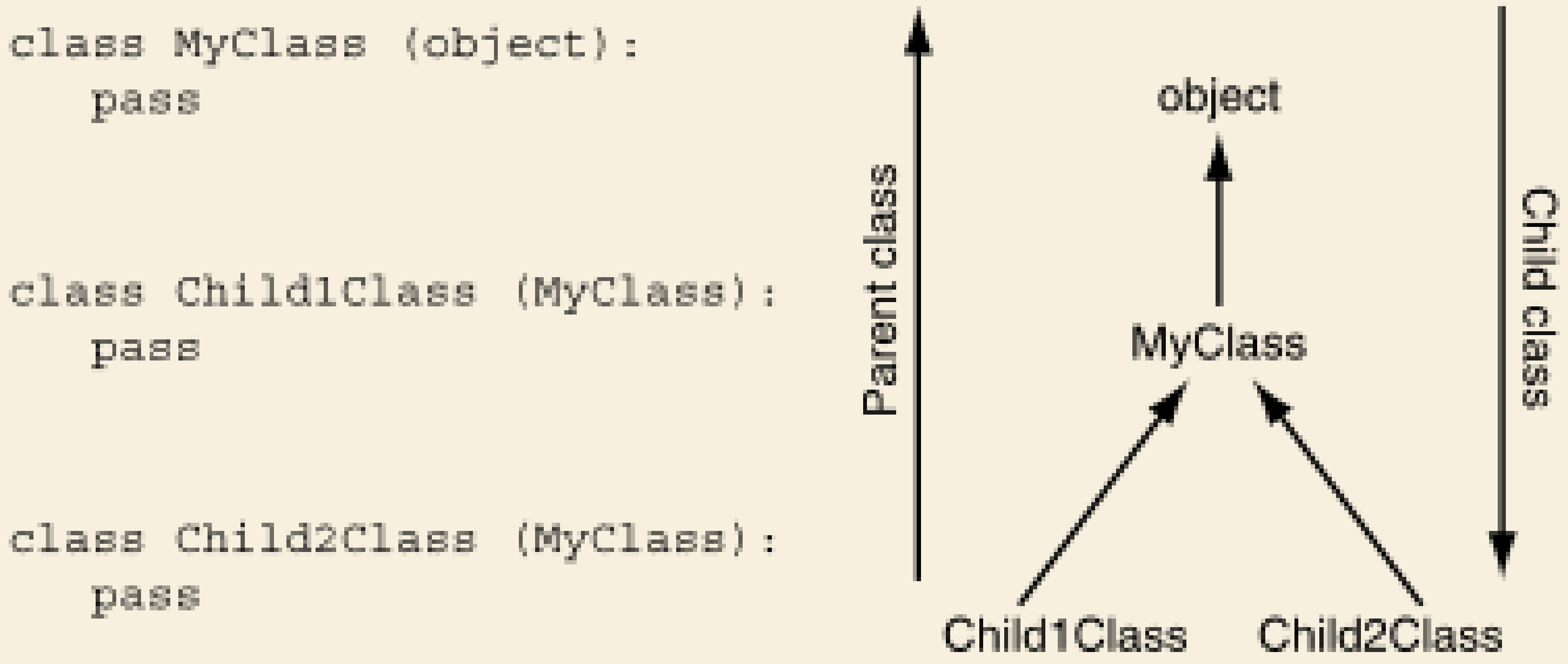
CLASSES

name of the class above this class in the hierarchy



```python
class Book(object):
    pass
```

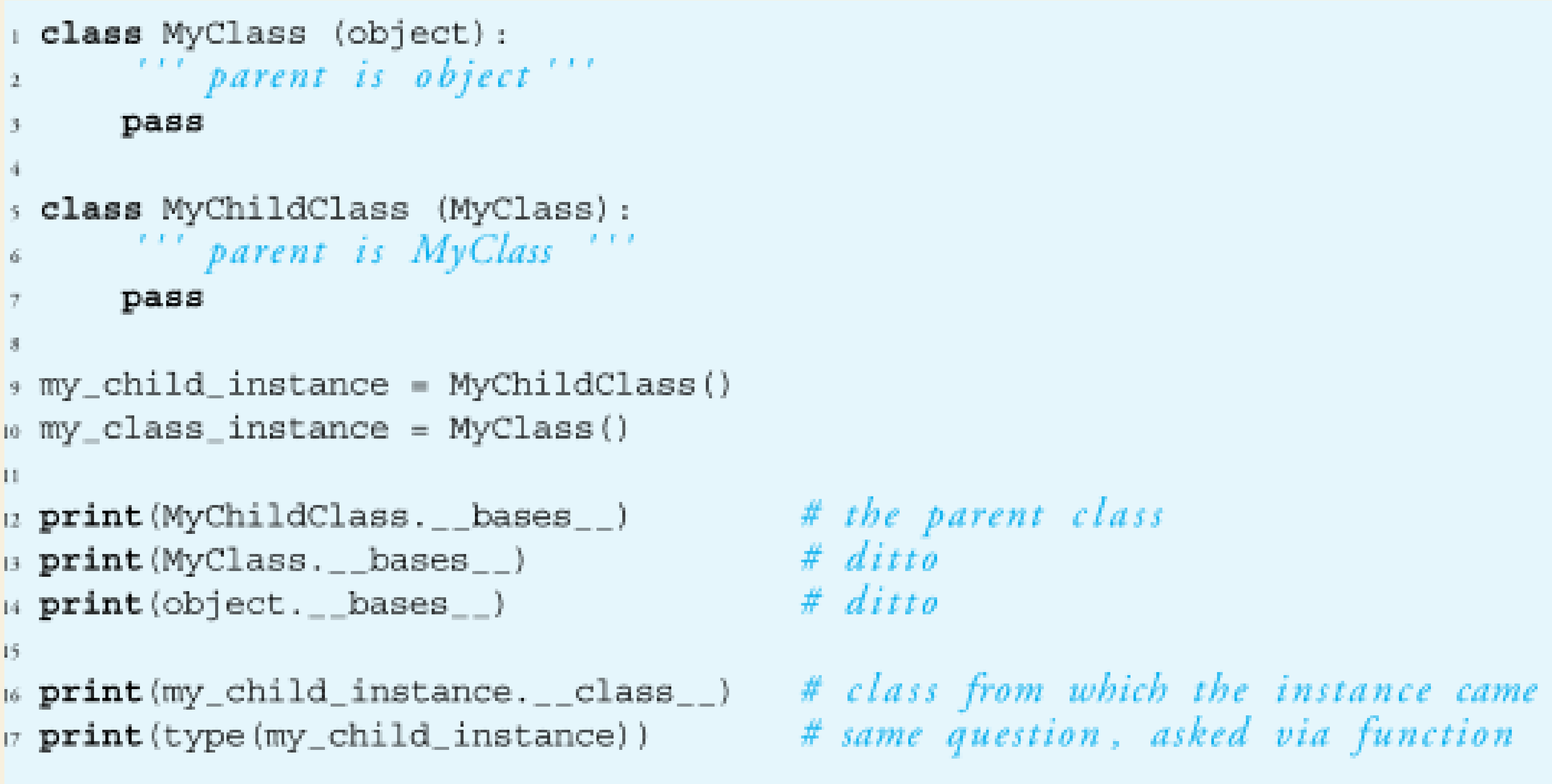
CLASSES



```
class MyClass (object):
   pass


class Child1Class (MyClass):
   pass


class Child2Class (MyClass):
   pass
```

FIGURE 12.1 A simple class hierarchy.

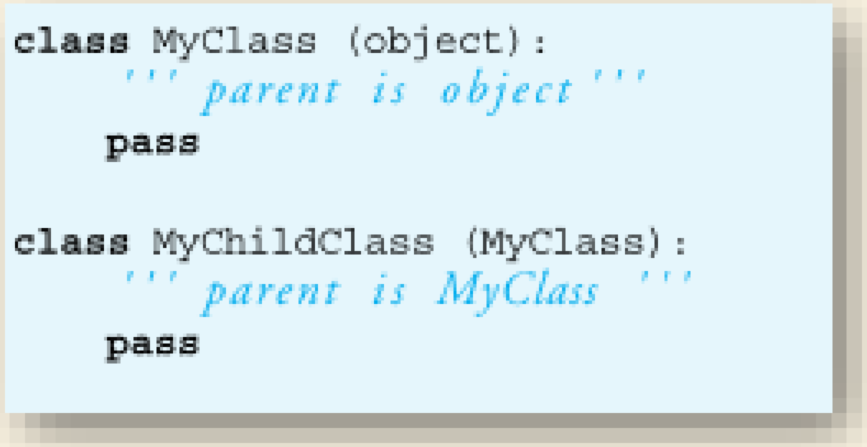
CLASSES



```
class MyClass (object):
    ''' parent is object '''
    pass

class MyChildClass (MyClass):
    ''' parent is MyClass  '''
    pass

my_child_instance = MyChildClass()
my_class_instance = MyClass()

print(MyChildClass.__bases__)        # the parent class
print(MyClass.__bases__)             # ditto
print(object.__bases__)              # ditto

print(my_child_instance.__class__)   # class from which the instance came
print(type(my_child_instance))       # same question, asked via function
```

CLASSES

- The class hierarchy imposes an ***is-a*** relationship between classes
  - – MyChildClass ***is-a*** (or is a kind of) MyClass
  - – MyClass ***is-a*** (or is a kind of) object
  - – Object has as a subclass MyClass
  - – MyChildClass has as a superclass MyClass



```
class MyClass (object):
    ''' parent is object'''
    pass

class MyChildClass (MyClass):
    ''' parent is MyClass '''
    pass
```

CLASSES

- This hierarchy arrangement helps saving/re-use of code
- Superclass code contains general code that is applicable to many subclasses
- A subclass uses code from it‘s superclass (via sharing) but specializes code for itself when and if necessary

CLASSES

The Scope for objects, the full story

1. Look in the current object for the attribute
2. If not in that object, look to the object's class to see if there is a shared class attribute
3. If not in the object's class, look up the hierarchy of that class for the attribute (in the parent class)
4. If you hit the class object, then the attribute does not exist

CLASSES

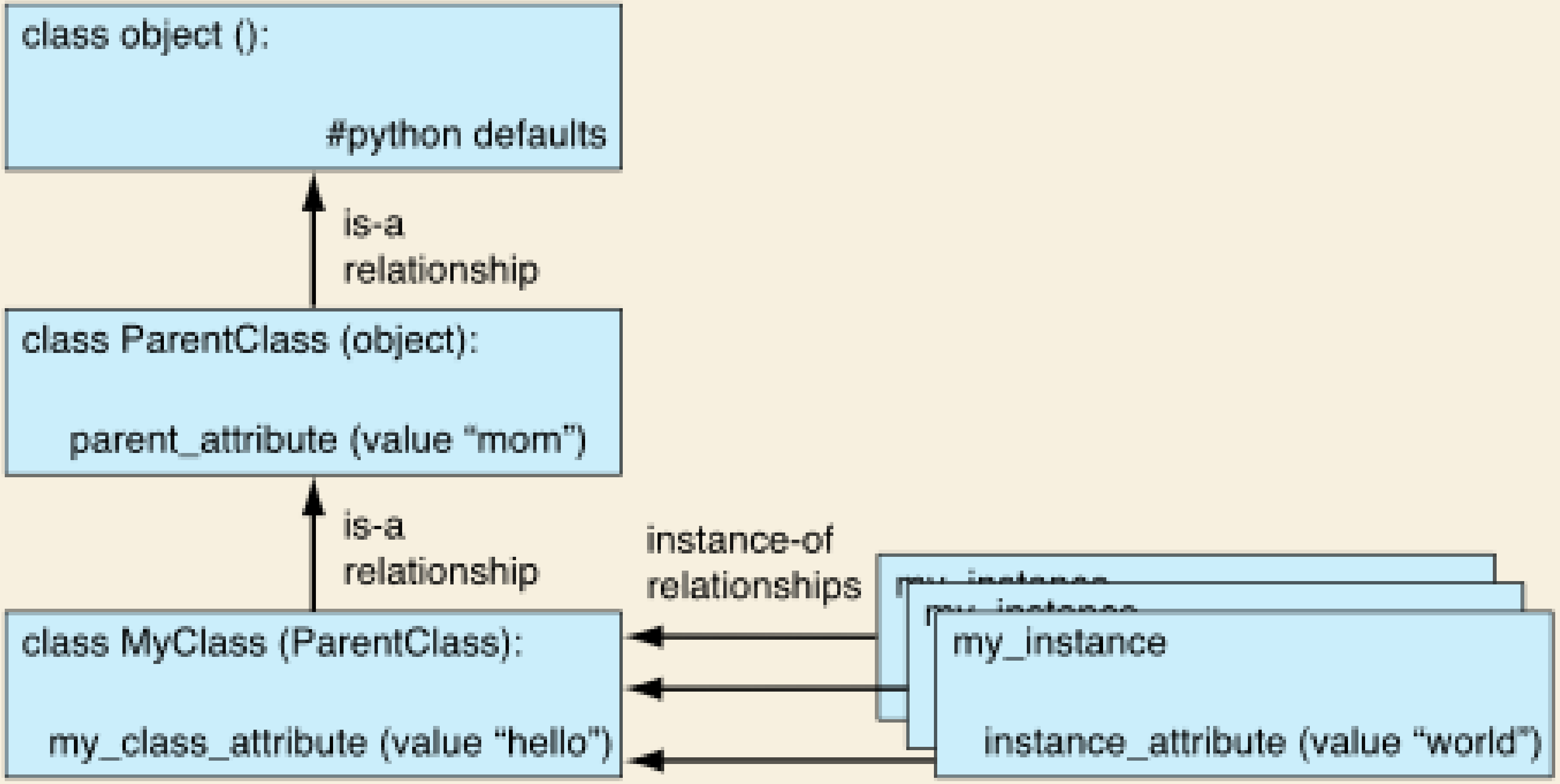

class object ():
#python defaults
is-a
relationship
class ParentClass (object):
parent_attribute (value "mom")
is-a
relationship
instance-of
relationships
class MyClass (ParentClass):
my_class_attribute (value "hello")
my_instance
instance_attribute (value "world")

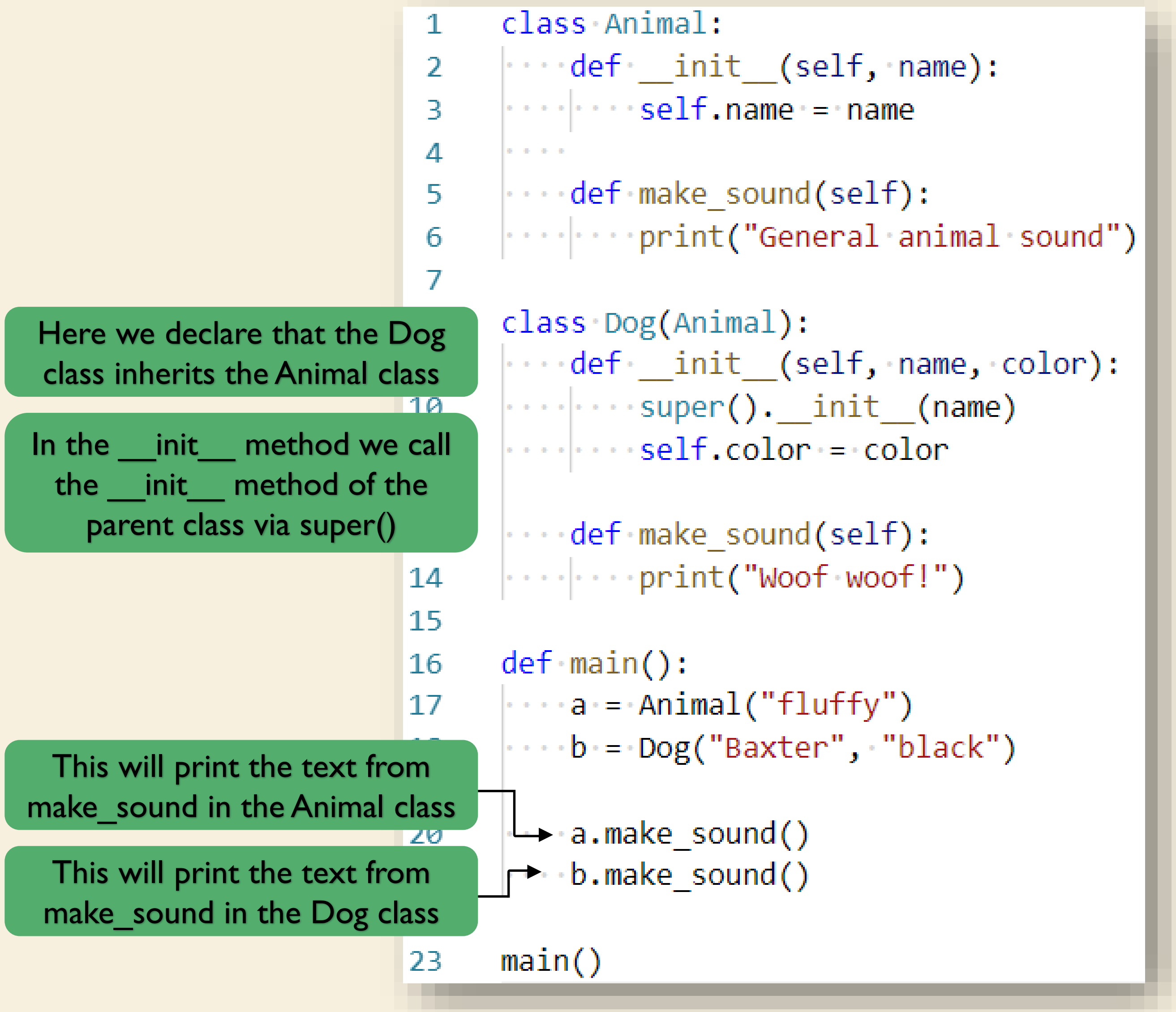
FIGURE 12.2 The players in the "find the attribute" game.

CLASSES

- builtins are objects too
  - One nice and easy way to use inheritance is to note that all the built-in types are objects
  - thus you can inherit the properties of built-in types and then modify how they get used in your subclass
  - you can also use any of the types you pull in as modules

CLASSES

- Lets see some examples
- Consider these two classes
- Notice that both Animal and Dog implement the make_sound method
    - In other words, the Dog class overwrites the make_sound method



```python
class Animal:
    def __init__(self, name):
        self.name = name
    
    def make_sound(self):
        print("General animal sound")

class Dog(Animal):
    def __init__(self, name, color):
        super().__init__(name)
        self.color = color

    def make_sound(self):
        print("Woof woof!")

def main():
    a = Animal("fluffy")
    b = Dog("Baxter", "black")

    a.make_sound()
    b.make_sound()

main()
```

Here we declare that the Dog class inherits the Animal class

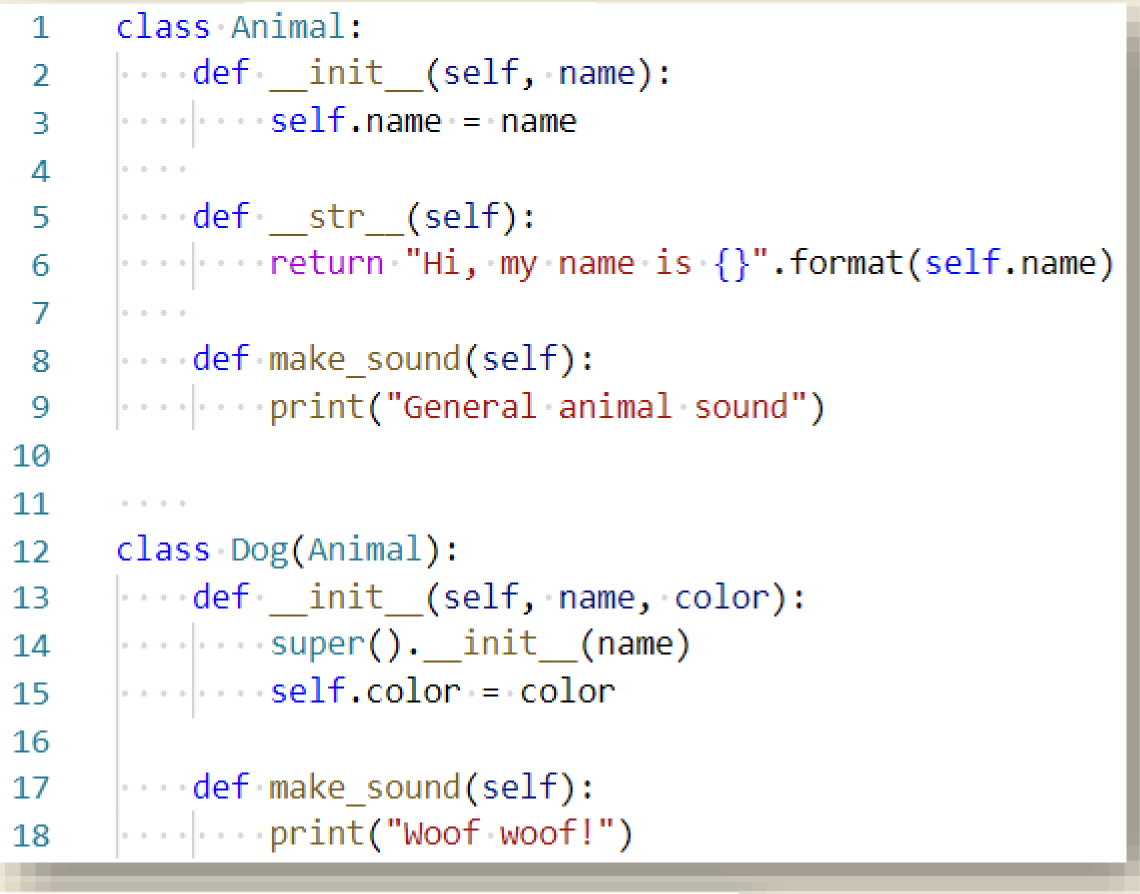
In the __init__ method we call the __init__ method of the parent class via super()

This will print the text from make_sound in the Animal class

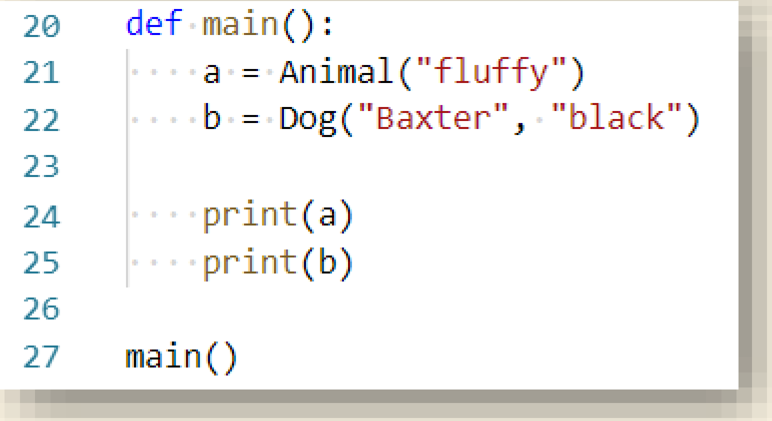
This will print the text from make_sound in the Dog class

CLASSES

- Here we have declared the __str__ method on the Animal class but not the Dog class
- What happens when the code on line 29 is executed?
  - Python checks the type of variable b and sees that it is of the type Dog.
  - Next it checks whether it has implemented the __str__ method. If it hasn´t it checks whether its parent class has implented that method
  - In this case the parent class has implemented the __str__ method so Python uses that implementation



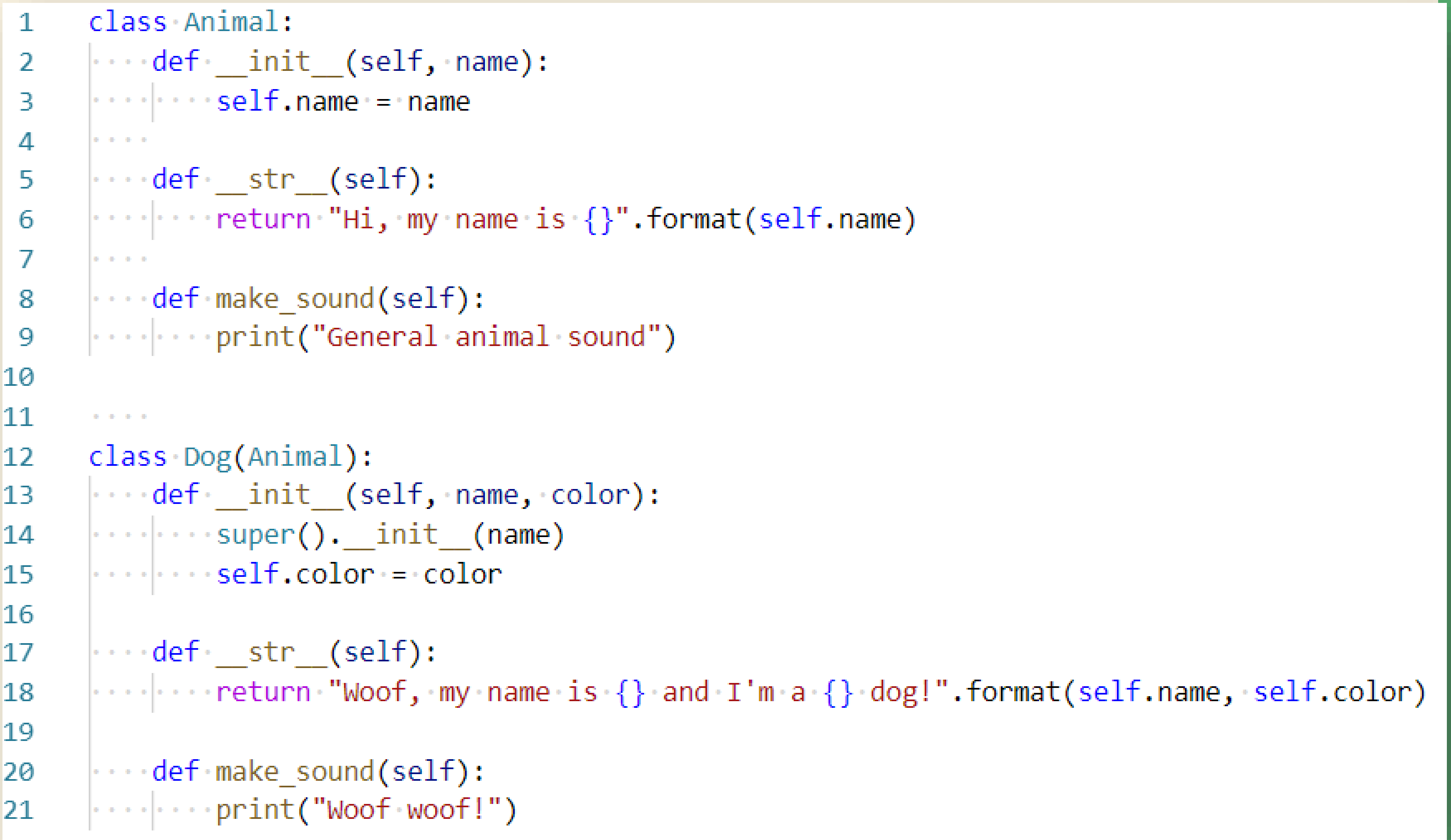
```
1   class Animal:
2       def __init__(self, name):
3           self.name = name
4       
5       def __str__(self):
6           return "Hi, my name is {}".format(self.name)
7       
8       def make_sound(self):
9           print("General animal sound")
10
11      
12  class Dog(Animal):
13      def __init__(self, name, color):
14          super().__init__(name)
15          self.color = color
16
17      def make_sound(self):
18          print("Woof woof!")
```



```
20  def main():
21      a = Animal("fluffy")
22      b = Dog("Baxter", "black")
23
24      print(a)
25      print(b)
26
27  main()
```

CLASSES

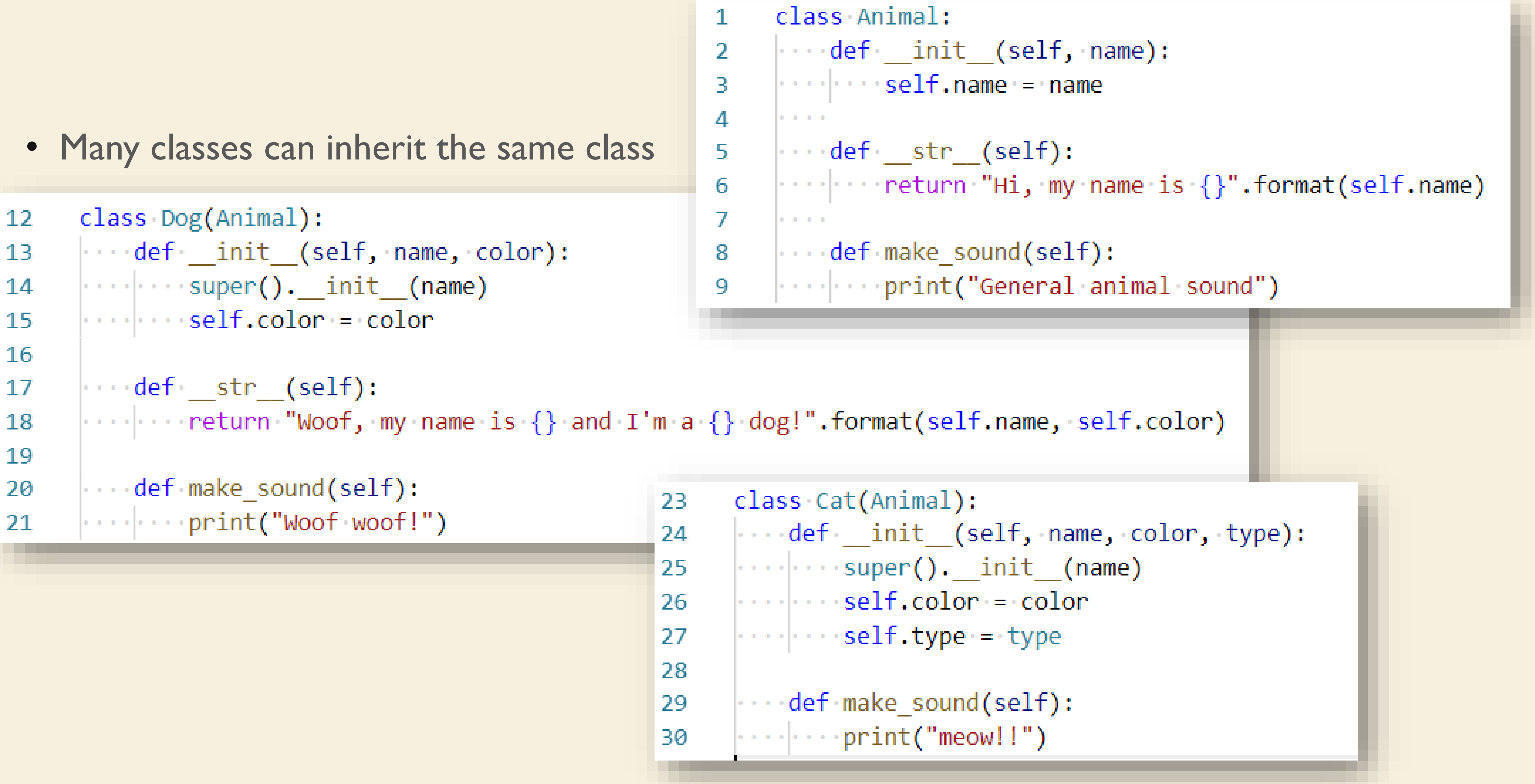
- If we wanted to we could overwrite the __str__() method on the Dog class like this
- This means that all derived classes of Animal could simply overwrite each method



```python
class Animal:
    def __init__(self, name):
        self.name = name
    
    def __str__(self):
        return "Hi, my name is {}".format(self.name)
    
    def make_sound(self):
        print("General animal sound")

    
class Dog(Animal):
    def __init__(self, name, color):
        super().__init__(name)
        self.color = color

    def __str__(self):
        return "Woof, my name is {} and I'm a {} dog!".format(self.name, self.color)

    def make_sound(self):
        print("Woof woof!")
```

CLASSES

- Many classes can inherit the same class



```python
class Animal:
    def __init__(self, name):
        self.name = name
    
    def __str__(self):
        return "Hi, my name is {}".format(self.name)
    
    def make_sound(self):
        print("General animal sound")
```

```python
class Dog(Animal):
    def __init__(self, name, color):
        super().__init__(name)
        self.color = color

    def __str__(self):
        return "Woof, my name is {} and I'm a {} dog!".format(self.name, self.color)

    def make_sound(self):
        print("Woof woof!")
```

```python
class Cat(Animal):
    def __init__(self, name, color, type):
        super().__init__(name)
        self.color = color
        self.type = type

    def make_sound(self):
        print("meow!!")
```

CLASSES

- Remember
  - It is a good idea to keep all the method that work the same for each animal within the parent class and methods that are unique to a specific animal within a derived class