Inheritance

Syntax for Inheritance

```
class ExampleParent:
        def __init__(self):
                [...]

        def funcA(self):
                [...]
```

Start by creating a class, as we've seen before. This will be our Parent Class.

Let's say we have a simple class here with an __init__() method and some function funcA.

Syntax for Inheritance

```
class ExampleParent:
        def __init__(self):
                [...]

        def funcA(self):
                [...]

#######################################

class ExampleChild(ExampleParent):
        def __init__(self):
                super().__init__(self)

        def funcB(self):
                [...]
```

We can create a class called the Child class that will *inherit* properties from the Parent class.

Syntax for Inheritance

```
class ExampleParent:
        def __init__(self):
                [...]

        def funcA(self):
                [...]

#######################################

class ExampleChild(ExampleParent):
        def __init__(self):
                super().__init__(self)

        def funcB(self):
                [...]
```

To inherit the Parent class, we simply place the name of our Parent class in parentheses here.

Syntax for Inheritance

```
class ExampleParent:
        def __init__(self):
                [...]

        def funcA(self):
                [...]

#######################################

class ExampleChild(ExampleParent):
        def __init__(self):
                super().__init__(self)

        def funcB(self):
                [...]
```

We can also use the __init__() function of the Parent class to help initialize our Child class. We do this by using super(). In this case, super() refers to the ExampleParent class.

Syntax for Inheritance

```
class ExampleParent:
        def __init__(self):
                [...]

        def funcA(self):
                [...]

#######################################

class ExampleChild(ExampleParent):
        def __init__(self):
                super().__init__(self)

        def funcB(self):
                [...]
```

Both the Parent and the Child class will be able to call funcA().

Syntax for Inheritance

```
class ExampleParent:
        def __init__(self):
                [...]

        def funcA(self):
                [...]

#######################################

class ExampleChild(ExampleParent):
        def __init__(self):
                super().__init__(self)

        def funcB(self):
                [...]
```

However, only the Child class will be able to call funcB.

Example time!

Pet class (Parent)

```
class Pet:
    def __init__(self, name, num_legs):
        self.name = name
        self.num_legs = num_legs

    def is_two_legged(self):
        return self.num_legs == 2

    def is_four_legged(self):
        return self.num_legs == 4

    def is_cute(self):
        return True
```

Our Pet class comes with an __init__() function and three other functions.

Pet class (Parent)

```
class Pet:
    def __init__(self, name, num_legs):
        self.name = name
        self.num_legs = num_legs

    def is_two_legged(self):
        return self.num_legs == 2

    def is_four_legged(self):
        return self.num_legs == 4

    def is_cute(self):
        return True
```

We can write child classes for specific types of pets! (Dogs, cats, etc).

Pet class (Parent)

```
class Pet:
    def __init__(self, name, num_legs):
        self.name = name
        self.num_legs = num_legs

    def is_two_legged(self):
        return self.num_legs == 2

    def is_four_legged(self):
        return self.num_legs == 4

    def is_cute(self):
        return True
```

Let's write a class called Dog that inherits the Pet class, but also has some dog-specific functionality.

Dog class (Child)

```python
class Pet:
    def __init__(self, name, num_legs):
        self.name = name
        self.num_legs = num_legs

    def is_two_legged(self):
        return self.num_legs == 2

    def is_four_legged(self):
        return self.num_legs == 4

    def is_cute(self):
        return True
```

```python
class Dog(Pet):
    def __init__(self, name, num_legs, is_a_good_doggy=True):
        self.is_good = is_a_good_doggy
        super().__init__(self, name, num_legs)

    def sit(self):
        print("*sits*")

    def speak(self):
        print("Woof")

    def shake_hand(self):
        if(self.is_four_legged()):
            print("*shakes hand*")
        else:
            print("Can you just give me treats instead?")
```

Great! So here’s what our classes look like.

Dog class (Child)

```python
class Pet:
    def __init__(self, name, num_legs):
        self.name = name
        self.num_legs = num_legs

    def is_two_legged(self):
        return self.num_legs == 2

    def is_four_legged(self):
        return self.num_legs == 4

    def is_cute(self):
        return True
```

```python
class Dog(Pet):
    def __init__(self, name, num_legs, is_a_good_doggy=True):
        self.is_good = is_a_good_doggy
        super().__init__(self, name, num_legs)

    def sit(self):
        print("*sits*")

    def speak(self):
        print("Woof")

    def shake_hand(self):
        if(self.is_four_legged()):
            print("*shakes hand*")
        else:
            print("Can you just give me treats instead?")
```

The class on the left is our original Pet class. The class on the right is our Dog class, which inherits the Pet class.

Dog class (Child)

```python
class Pet:
    def __init__(self, name, num_legs):
        self.name = name
        self.num_legs = num_legs

    def is_two_legged(self):
        return self.num_legs == 2

    def is_four_legged(self):
        return self.num_legs == 4

    def is_cute(self):
        return True
```

```python
class Dog(Pet):
    def __init__(self, name, num_legs, is_a_good_doggy=True):
        self.is_good = is_a_good_doggy
        super().__init__(self, name, num_legs)

    def sit(self):
        print("*sits*")

    def speak(self):
        print("Woof")

    def shake_hand(self):
        if(self.is_four_legged()):
            print("*shakes hand*")
        else:
            print("Can you just give me treats instead?")
```

Notice that in our Dog class, our ***__init__()*** function takes on a few more parameters than the Pet class's init function. This is okay! We can simply initialize our Dog-specific properties (namely, ***self.is_good***), and then we can initialize the properties that all Pets would have (namely, ***num_legs*** and ***name***).

Dog class (Child)

```python
class Pet:
    def __init__(self, name, num_legs):
        self.name = name
        self.num_legs = num_legs

    def is_two_legged(self):
        return self.num_legs == 2

    def is_four_legged(self):
        return self.num_legs == 4

    def is_cute(self):
        return True
```

```python
class Dog(Pet):
    def __init__(self, name, num_legs, is_a_good_doggy=True):
        self.is_good = is_a_good_doggy
        super().__init__(self, name, num_legs)

    def sit(self):
        print("*sits*")

    def speak(self):
        print("Woof")

    def shake_hand(self):
        if(self.is_four_legged()):
            print("*shakes hand*")
        else:
            print("Can you just give me treats instead?")
```

Notice also that in our Dog class, we can call functions from our Pet class. More specifically, in the ***shake_hand()*** function, we call the Pet class's ***is_four_legged()*** function. Dogs with fewer than four legs (like mine!) have a hard time shaking hands, so it's good that we can call our Pet class's functions as helpers for our Dog class's functions.

Using our Examples

```python
class Pet:
    def __init__(self, name, num_legs):
        self.name = name
        self.num_legs = num_legs

    def is_two_legged(self):
        return self.num_legs == 2

    def is_four_legged(self):
        return self.num_legs == 4

    def is_cute(self):
        return True
```

```python
class Dog(Pet):
    def __init__(self, name, num_legs, is_a_good_doggy=True):
        self.is_good = is_a_good_doggy
        super().__init__(self, name, num_legs)

    def sit(self):
        print("*sits*")

    def speak(self):
        print("Woof")

    def shake_hand(self):
        if(self.is_four_legged()):
            print("*shakes hand*")
        else:
            print("Can you just give me treats instead?")
```

Let's create a new Pet. We don't know if this pet is a dog, cat, unicorn, or anything in between. All we know is that this pet has four legs and is named "Firestar".

```
>>> import Dog, Pet

>>> firestar = Pet("Firestar", 4)

>>> firestar.is_two_legged()
False

>>> firestar.is_four_legged()
True

>>> firestar.is_cute()
True

>>>firestar.sit()
AttributeError: 'Pet' object has no attribute 'sit'
```

```python
class Pet:
    def __init__(self, name, num_legs):
        self.name = name
        self.num_legs = num_legs

    def is_two_legged(self):
        return self.num_legs == 2

    def is_four_legged(self):
        return self.num_legs == 4

    def is_cute(self):
        return True
```

```python
class Dog(Pet):
    def __init__(self, name, num_legs, is_a_good_doggy=True):
        self.is_good = is_a_good_doggy
        super().__init__(self, name, num_legs)

    def sit(self):
        print("*sits*")

    def speak(self):
        print("Woof")

    def shake_hand(self):
        if(self.is_four_legged()):
            print("*shakes hand*")
        else:
            print("Can you just give me treats instead?")
```

Let's create a Dog now. This Dog is named Floof.

```
>>> import Dog, Pet
>>> floof = Dog("Floof", 4)

>>> floof.is_two_legged()
False

>>> floof.is_four_legged()
True

>>> floof.is_cute()
True

>>> floof.sit()
*sits*

>>> floof.speak()
Woof

>>> floof.shake_hand()
*shakes hand*
```

```python
class Pet:
    def __init__(self, name, num_legs):
        self.name = name
        self.num_legs = num_legs

    def is_two_legged(self):
        return self.num_legs == 2

    def is_four_legged(self):
        return self.num_legs == 4

    def is_cute(self):
        return True
```

```python
class Dog(Pet):
    def __init__(self, name, num_legs, is_a_good_doggy=True):
        self.is_good = is_a_good_doggy
        super().__init__(self, name, num_legs)

    def sit(self):
        print("*sits*")

    def speak(self):
        print("Woof")

    def shake_hand(self):
        if(self.is_four_legged()):
            print("*shakes hand*")
        else:
            print("Can you just give me treats instead?")
```

Finally, let's create an object to represent my dog, Oreo. He has three legs, so we have to specify that.

>>> oreo = Dog("Oreo", 3)

>>> oreo.is_two_legged(), oreo.is_four_legged()
(False, False)

>>> oreo.is_cute()
True

>>> oreo.sit()
sits

>>> oreo.speak()
Woof

>>>oreo.shake_hand()
Can you just give me treats instead?