Objects and Classes

Sarah Holderness

Author

@dr_holderness

Object Oriented Programming

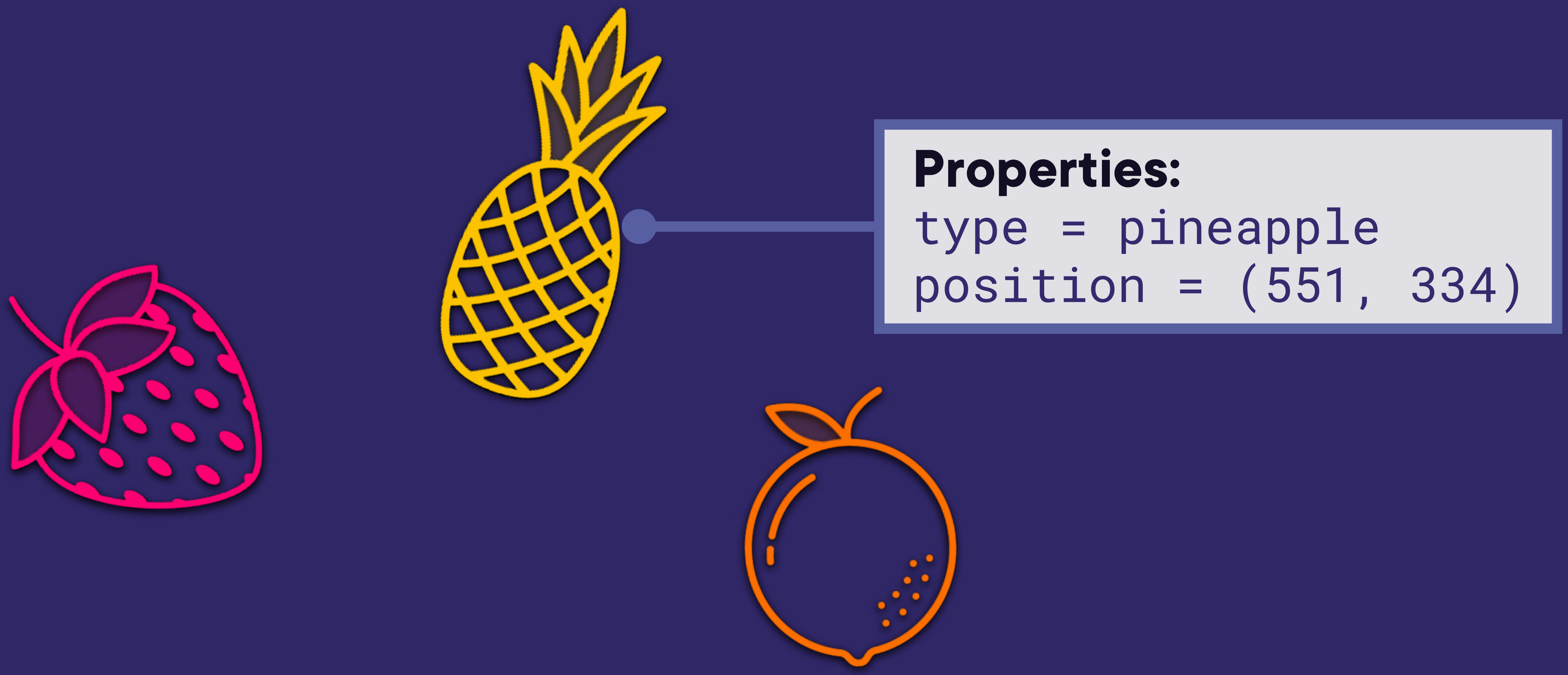
Designing computer programs to be organized around data or objects.

Why?

As programs get large and complicated, one person can't remember every detail. Organizing pieces of the program into objects makes it easier to understand and use.

Object Oriented Programming Example

Fruit Ninja - a piece of fruit in the game could be an object. And we need to know information about each fruit object like fruit type and position.

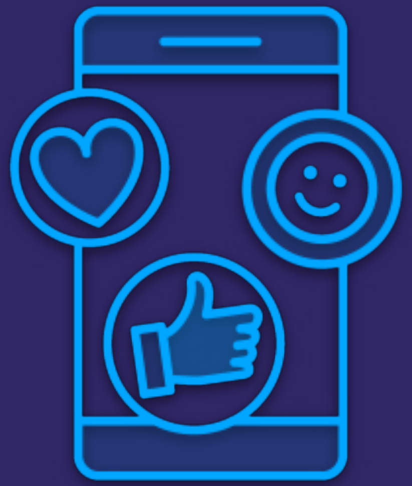


Properties:

```
type = pineapple
position = (551, 334)
```

Objects Have State and Behavior

Look around you, what objects do you see?



Phone

State	Behavior
Model	Ringing
Color	Receiving notifications
Storage	Sending data

Dog

State	Behavior
Name	Barking
Breed	Whining
Hungry	Wagging Tail

Designing a Robot Dog Toy

State	Behavior
Name	Barking
Breed	Whining
Hungry	Wagging Tail

Defining the Robot Dog Class

```
class Robot_Dog:

    def __init__(                   ):
```

The `__init__()` method lets us initialize our robot's properties

State	Behavior
Name	Barking
Breed	Whining
Hungry	Wagging Tail

Defining the Robot Dog Class

```
class Robot_Dog:

    def __init__(self, name, breed):
```

self is required it refers to the instance you're creating

Also pass in the properties you want to initialize as parameters

Defining the Robot Dog Class

```
class Robot_Dog:

    def __init__(self, name, breed):
        self.name  = name
        self.breed = breed
```

Initialize the properties of the new object, `self`, to the passed in values

This object's properties

Defining the Robot Dog Class

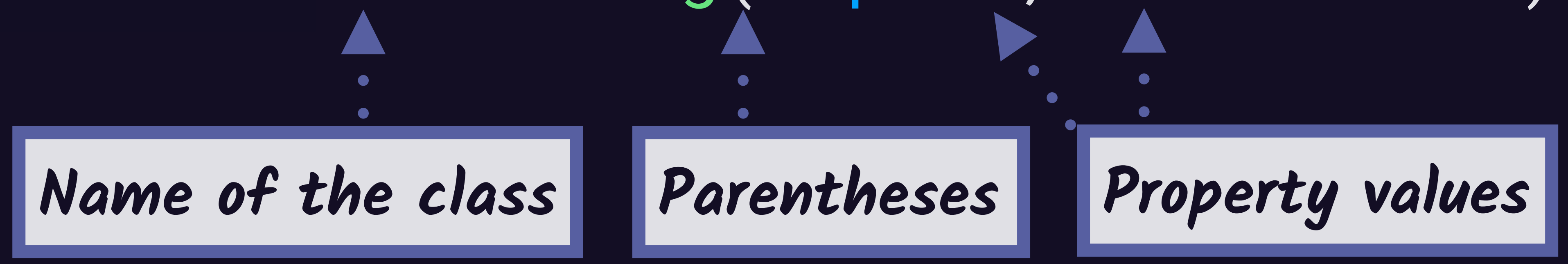
```
class Robot_Dog:

    def __init__(self, name_val, breed_val):
        self.name  = name_val
        self.breed  = breed_val
```

Initialize the properties of the new object, `self`, to the passed in values

This object's properties

Creating a Robot Dog Object

```python
class Robot_Dog:
    def __init__(self, name_val, breed_val):
        self.name = name_val
        self.breed = breed_val

# Main program
         Robot_Dog('Spot', 'Chihuaha')
```



Name of the class

Parentheses

Property values

Creating a Robot Dog Object

```
class Robot_Dog:
    def __init__(self, name_val, breed_val):
        self.name = name_val
        self.breed = breed_val

# Main program
my_dog = Robot_Dog('Spot', 'Chihuaha')
print(my_dog.name)
print(my_dog.breed)
```

We can print the dog object's property values to check our class is working.

The dot lets you access the object's properties and methods

Creating a Robot Dog Object

```
class Robot_Dog:
    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

# Main program
my_dog = Robot_Dog('Spot', 'Chihuaha')
print(my_dog.name)
print(my_dog.breed)
```

```
> Spot
  Chihuahua
```

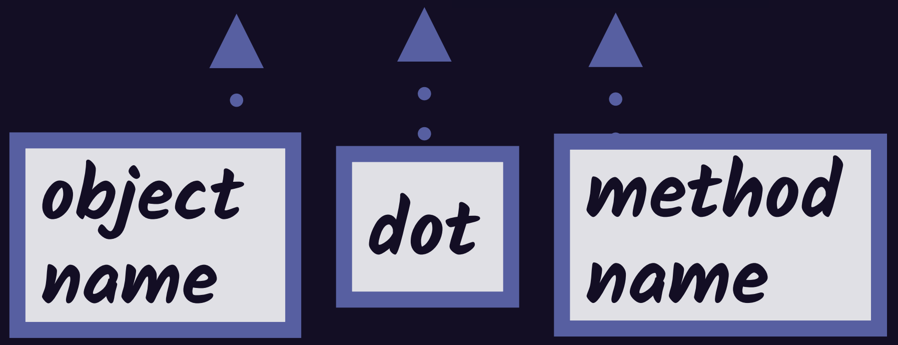
Creating a Class Method

```
class Robot_Dog:
    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    def bark(self):
        print('Woof Woof!')
```

We create a class method just like a function. Except a method has `self` *as the first parameter.*

Calling a Class Method

```python
class Robot_Dog:
    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    def bark(self):
        print('Woof Woof!')

# Main program
my_dog = Robot_Dog('Spot', 'Chihuaha')
print(my_dog.name)
print(my_dog.breed)
```



object name

dot

method name

```
> Spot
  Chihuahua
  Woof Woof!
```

Up Next:

Demo:

Create Classes to Manage a Company's Payroll

Class Inheritance

Sarah Holderness

Author

@dr_holderness

Relationships in Object Oriented Programming

Has-a

A company **has** employees

```python
class Company:
    def __init__(self):
        self.employees = []
```

```python
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def calculate_paycheck(self):
        return self.salary/52
```

Relationships in Object Oriented Programming

Has-a

We've seen this already

A company **has** employees

A robot **has a** battery

Is-a

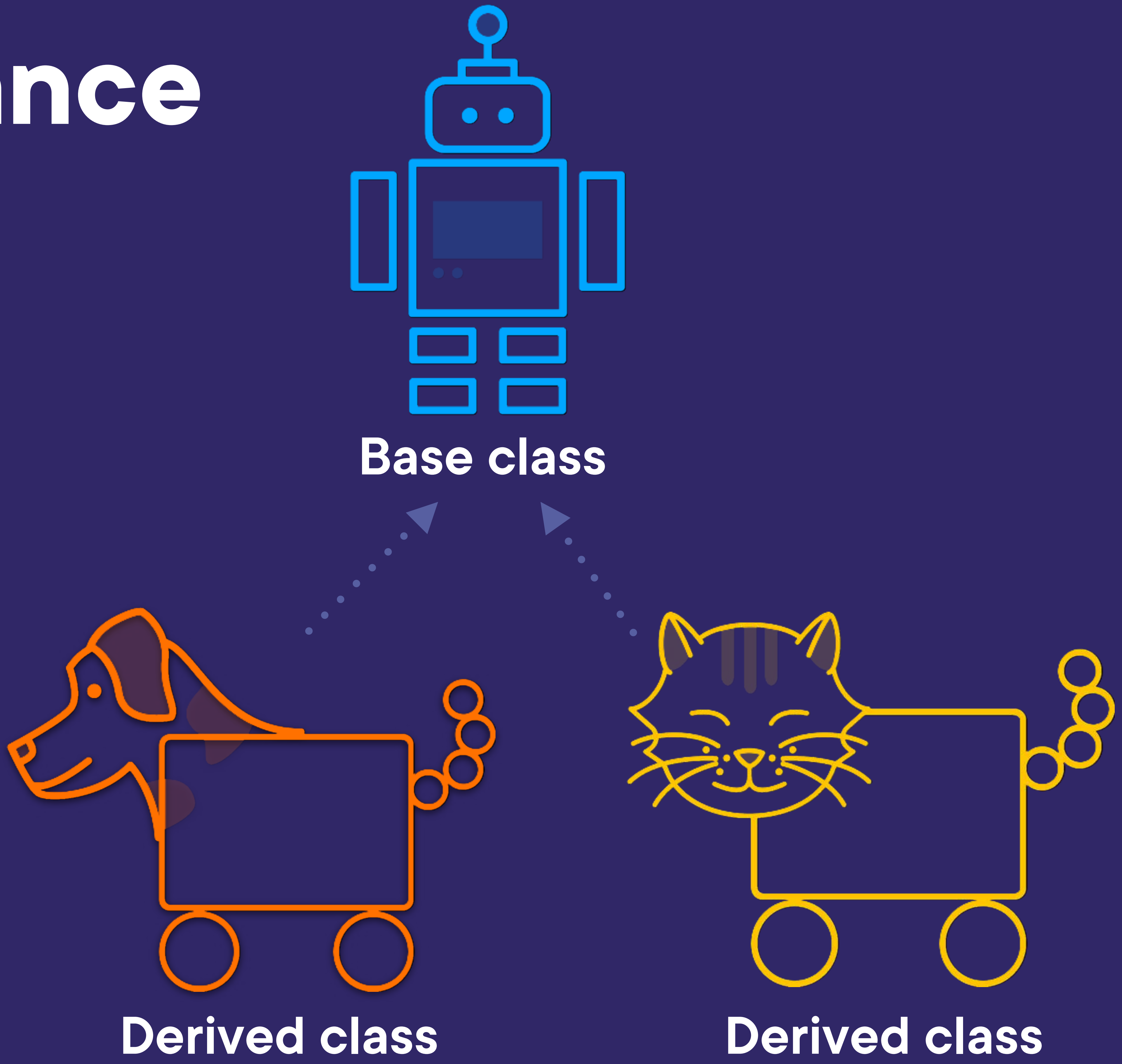
This is called inheritance. Let's take a closer look...

A robot dog **is a** robot

A robot cat **is a** robot

Inheritance

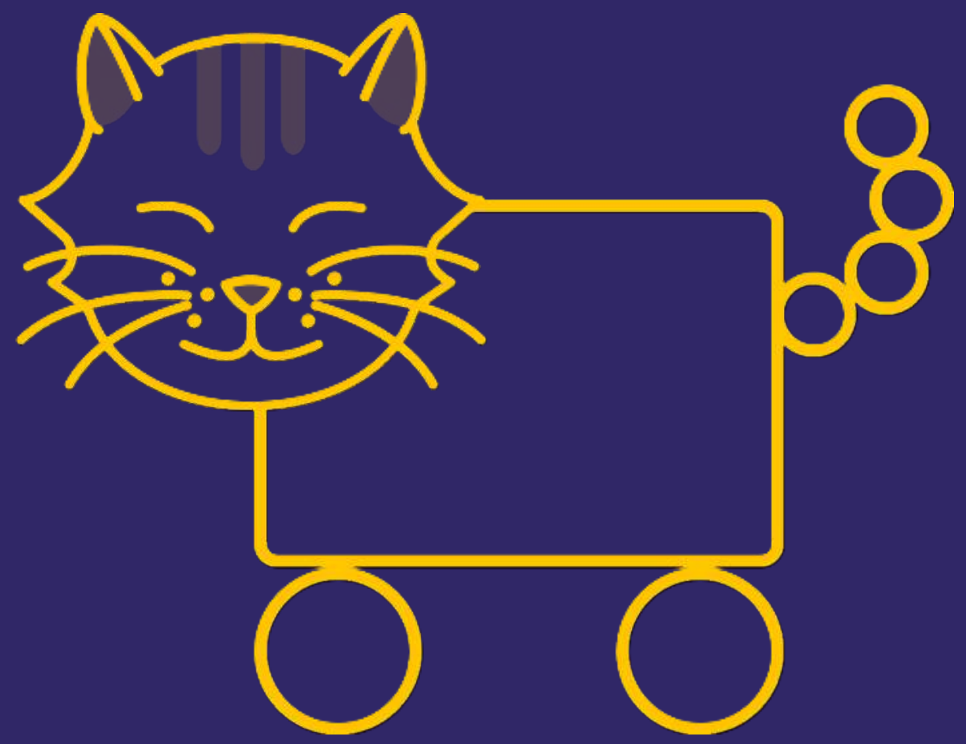
Hierarchy of classes that share properties and methods



Behaviors

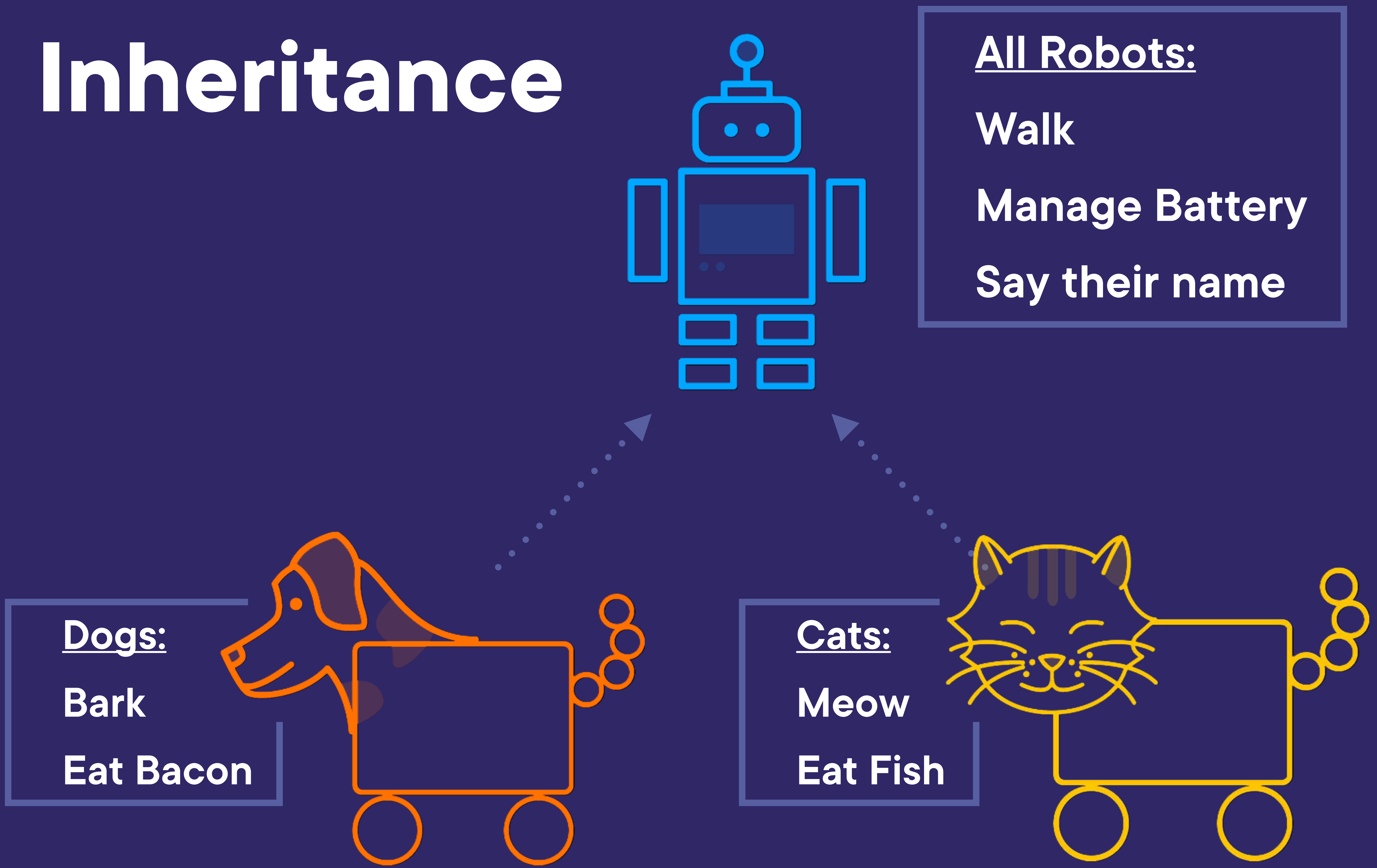
- Walk
- Manage battery
- Say their name
- Bark
- Eat Bacon

The same!



- Walk
- Manage battery
- Say their name
- Meow
- Eat Fish

Inheritance



All Robots:

Walk

Manage Battery

Say their name

Dogs:

Bark

Eat Bacon

Cats:

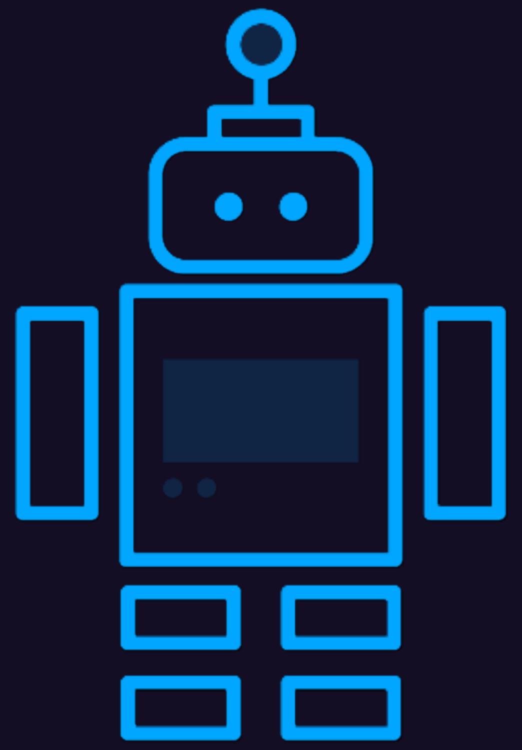
Meow

Eat Fish

Creating a Parent Class

```
class Robot:

    def __init__(self, name):
        self.name = name
        self.position = [0,0]
        print('My name is', self.name)

    def walk(self, x):
        self.position[0] = self.position[0] + x
        print('New position:', self.position)
```

The same as just creating a regular class



<u>All Robots:</u>

Walk

Say their name

Creating the Child Class

```
class Robot_Dog(Robot):
```

The Parent class we're inheriting from goes in parentheses

If we leave out the `__init__()` method it will call the parent's `__init__()` method by default.

Creating the Child Class

```
class Robot_Dog(Robot):

    def make_noise(self):
        print('Woof Woof!')
```

Only Robot_Dogs *not* Robots *can call this method.*

Creating a Robot_Dog Object

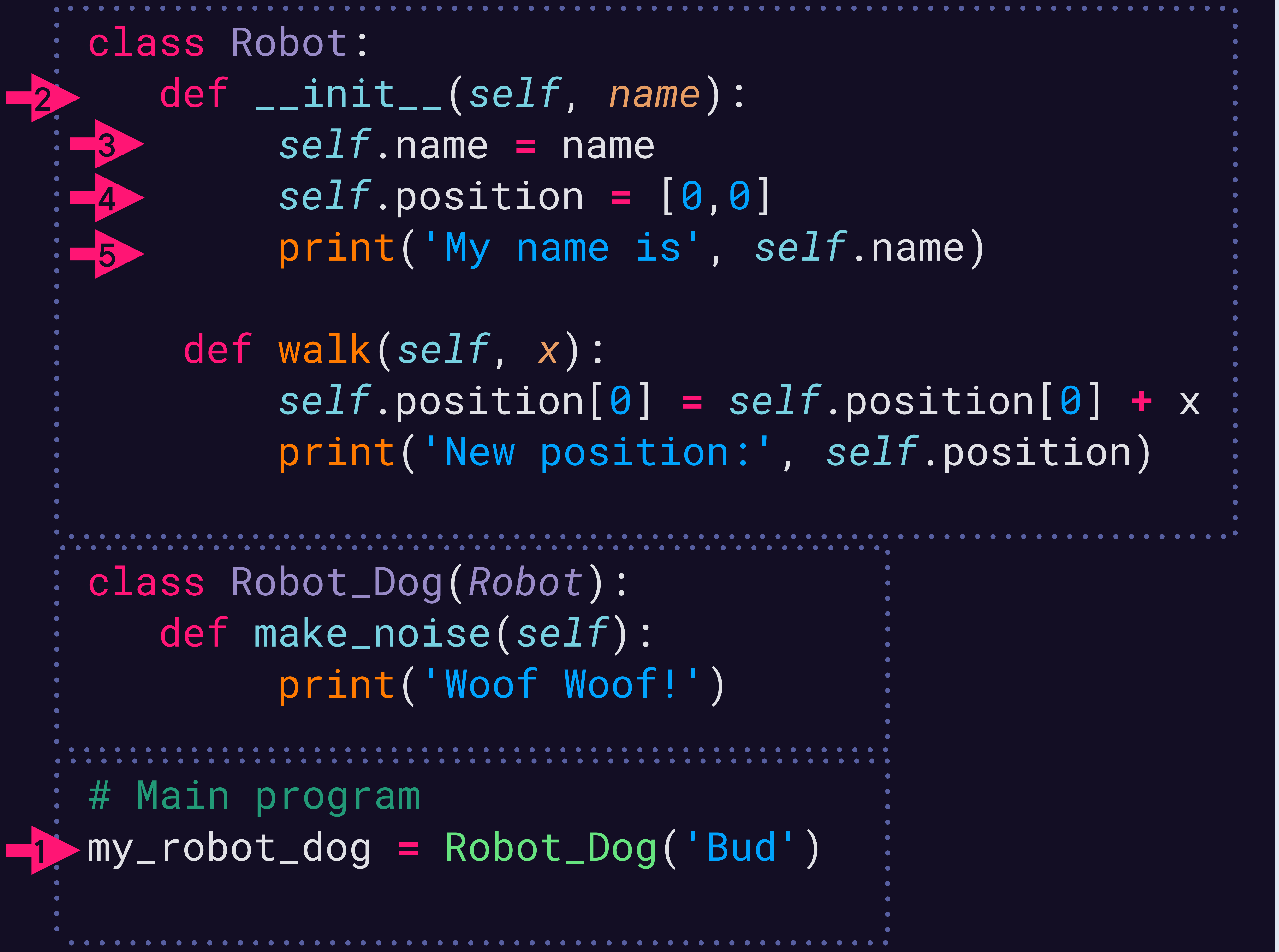
```python
class Robot_Dog(Robot):

    def make_noise(self):
        print('Woof Woof!')


my_robot_dog = Robot_Dog('Bud')
```

Create a `Robot_Dog` *object by calling the constructor*

Creating a Robot_Dog Object

```
class Robot_Dog(Robot):

    def make_noise(self):
        print('Woof Woof!')



my_robot_dog = Robot_Dog('Bud')
my_robot_dog.walk(10)
my_robot_dog.make_noise()
```

We can call any of Robot_Dog’s *or* Robot’s *methods.*



```python
class Robot:
    def __init__(self, name):
        self.name = name
        self.position = [0,0]
        print('My name is', self.name)

    def walk(self, x):
        self.position[0] = self.position[0] + x
        print('New position:', self.position)

class Robot_Dog(Robot):
    def make_noise(self):
        print('Woof Woof!')

# Main program
my_robot_dog = Robot_Dog('Bud')
```

```
> python3 robots.py

My name is Bud!
```

```
class Robot:
   def __init__(self, name):
        self.name = name
        self.position = [0,0]
        print('My name is', self.name)

2   def walk(self, x):
3       self.position[0] = self.position[0] + x
4       print('New position:', self.position)

class Robot_Dog(Robot):
   def make_noise(self):
        print('Woof Woof!')

my_robot_dog = Robot_Dog('Bud')
1 my_robot_dog.walk(10)
```

```
> python3 robots.py

My name is Bud!
New position: [10, 0]
```

```
class Robot:
   def __init__(self, name):
        self.name = name
        self.position = [0,0]
        print('My name is', self.name)

    def walk(self, x):
        self.position[0] = self.position[0] + x
        print('New position:', self.position)
```

```
class Robot_Dog(Robot):
2  def make_noise(self):
3       print('Woof Woof!')
```

```
my_robot_dog = Robot_Dog('Bud')
my_robot_dog.walk(10)
1 my_robot_dog.make_noise()
```

```
> python3 robots.py

My name is Bud!
New position: [10, 0]
Woof Woof!
```

We can see with inheritance how seamlessly the methods are called from either the parent class or the child class.

Method Overriding

```python
class Robot:
    ...
    def eat(self):
        print("I'm hungry!")

class Robot_Dog:
    ...

my_robot_dog = Robot_Dog('Bud')
my_robot_dog.eat()
```

```
> python3 robots.py

My name is Bud!
I’m hungry!
```

Method Overriding

```python
class Robot:
    ...
    def eat(self):
        print("I'm hungry!")

class Robot_Dog:
   ...
    def eat(self):
        print('I like bacon!')

my_robot_dog = Robot_Dog('Bud')
my_robot_dog.eat()
```

```
> python3 robots.py

My name is Bud!
I like bacon!
```

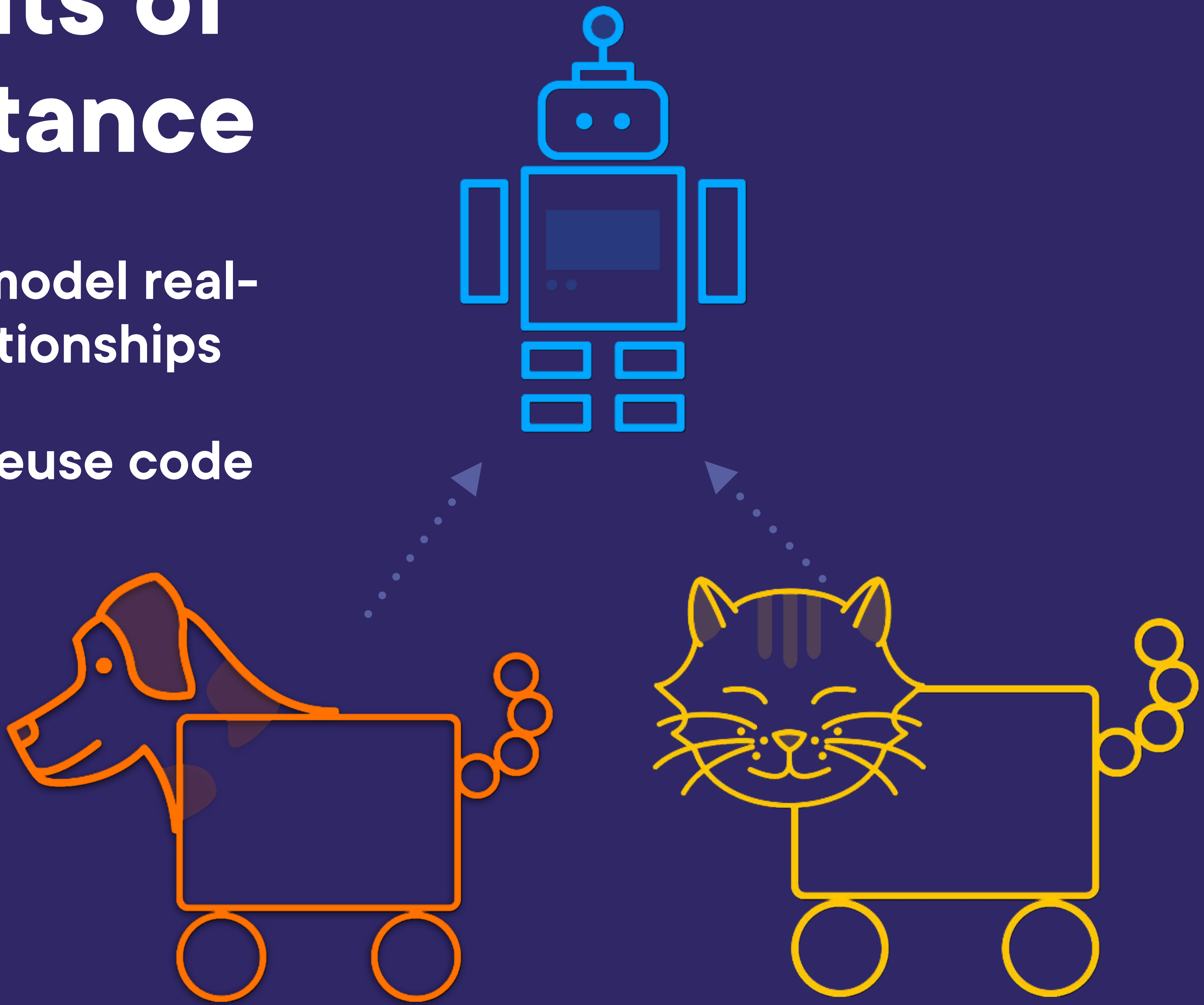
Calling super()

```python
class Robot:
    ...
    def eat(self):
        print("I'm hungry!")

class Robot_Dog:
   ...
    def eat(self):
        super().eat()
        print('I like bacon!')

my_robot_dog = Robot_Dog('Bud')
my_robot_dog.eat()
```

```
> python3 robots.py

My name is Bud!
I’m hungry
I like bacon!
```

Benefits of Inheritance

- It lets us model real-world relationships
- It lets us reuse code



Up Next:

Demo on Inheritance