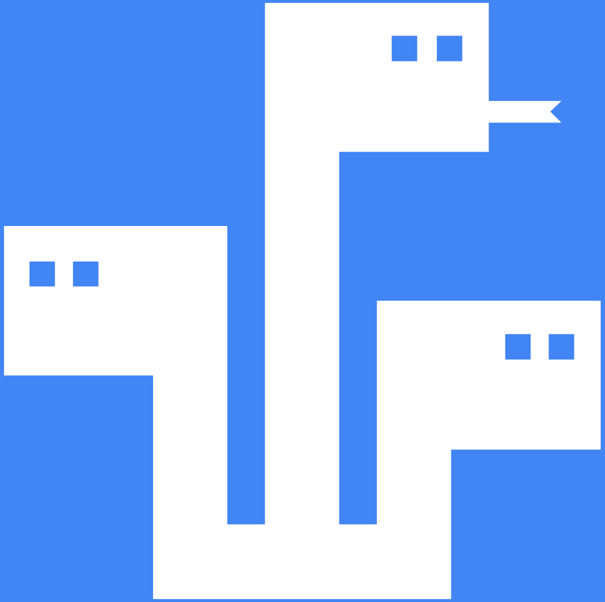
Classes

Python - Nick Reynolds
April 21, 2017



This Class

- Object Oriented Design
- Class Constructors
- Class Instances
- Class Inheritance

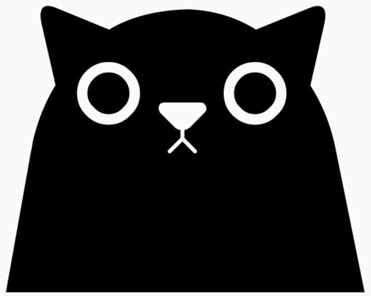
Object Oriented

Design Pattern

Everything is an Object

- Design pattern
- Highly structured
- Focused on reusability

A ***dog*** is an object



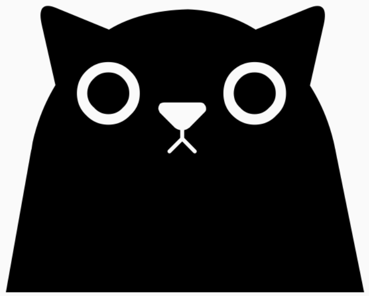
A ***cat*** is an object

These are both ***animal*** objects

Objects have Data

- Data that the object keeps to itself

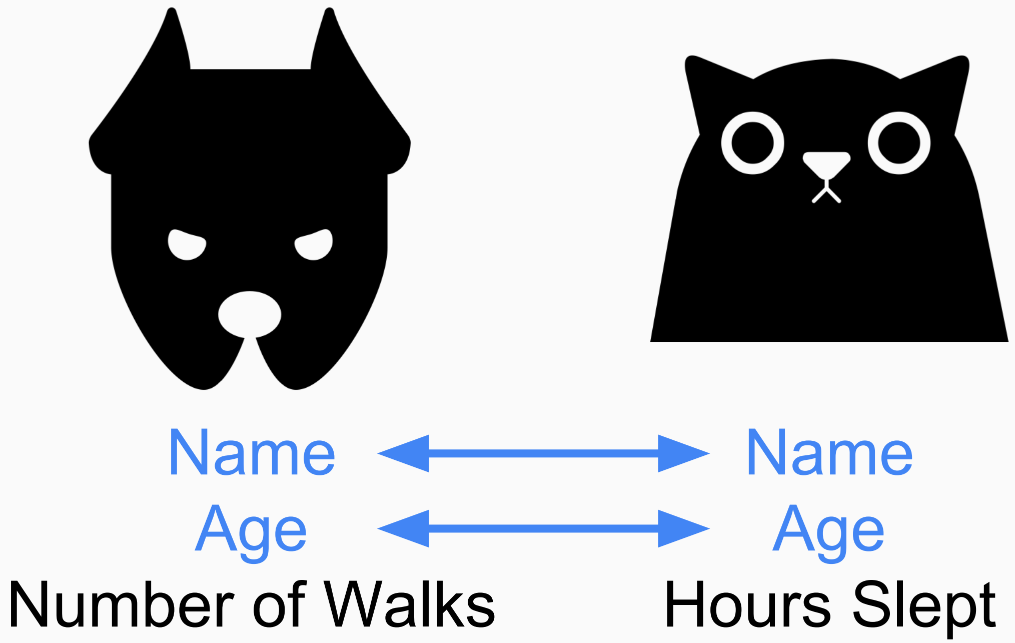
Name
Age
Number of Walks



Name
Age
Hours Slept

Objects have Data

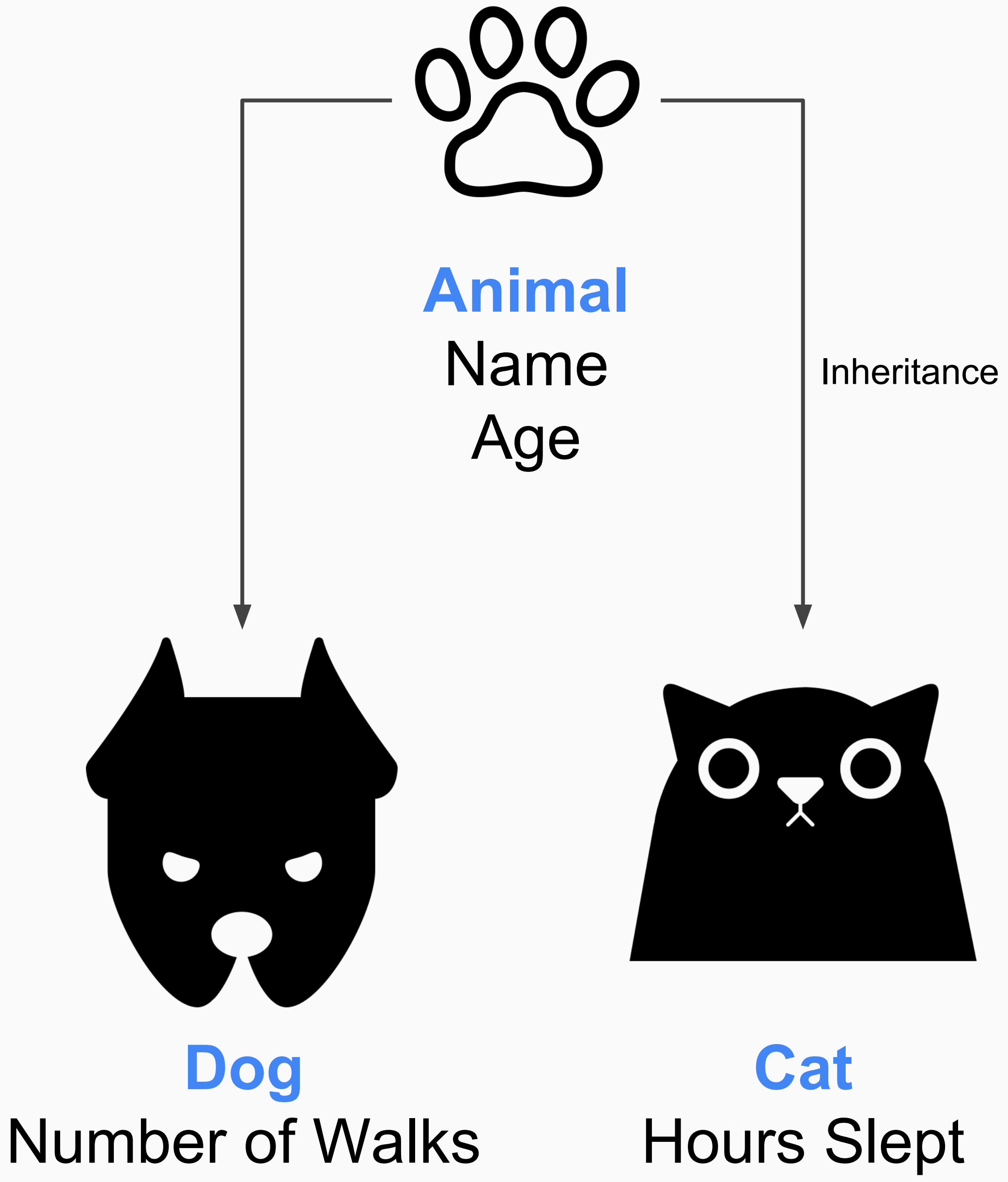
- Data that the object keeps to itself



General properties of an animal

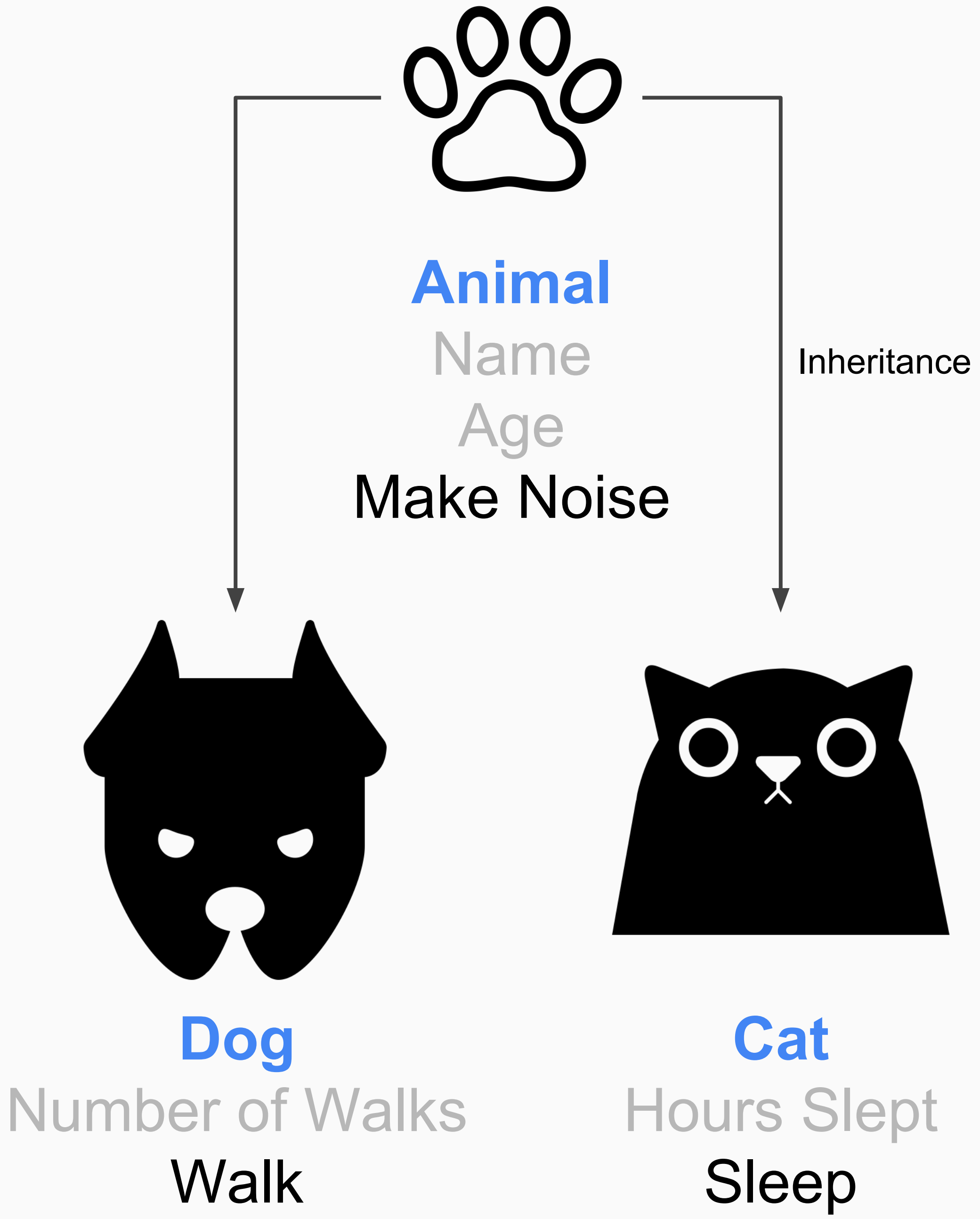
Inheritance

- Objects have “parents” and “children”
- Children (Dog, Cat) get properties from their parents



Objects have actions

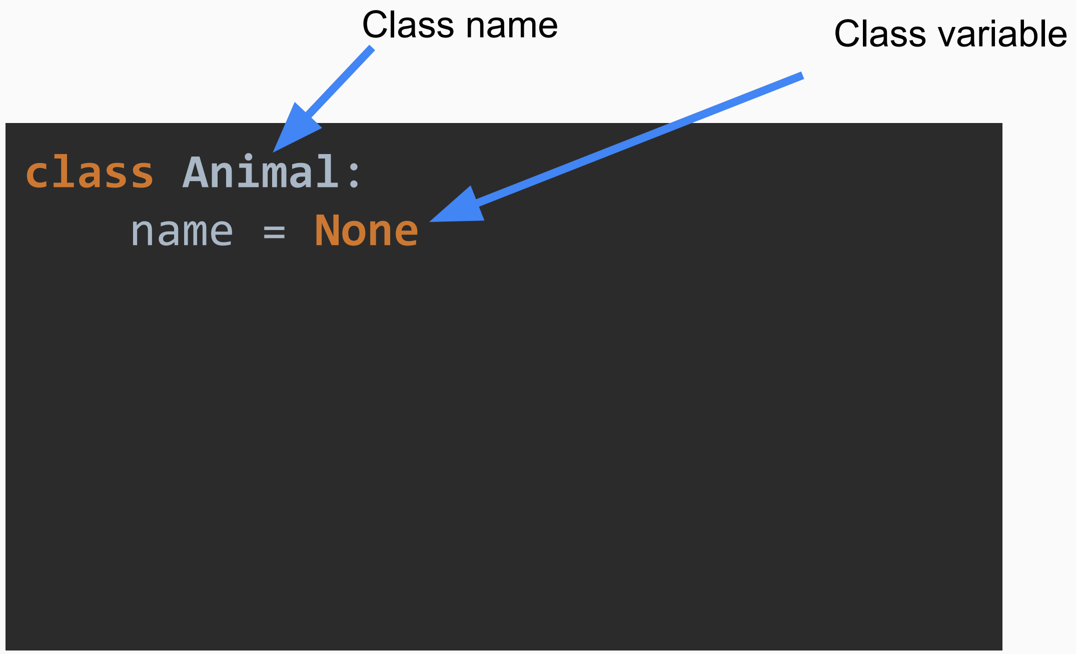
- Methods (functions) for that object
- They are inherited from parents too



Am I a vet now?

No

Python Classes



```python
class Animal:
    name = None
```

Python Classes

Class name

Class variable

```python
class Animal:
    name = None

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

Constructor

This is the setup part of the class

Happens once when the class is created

Python Classes

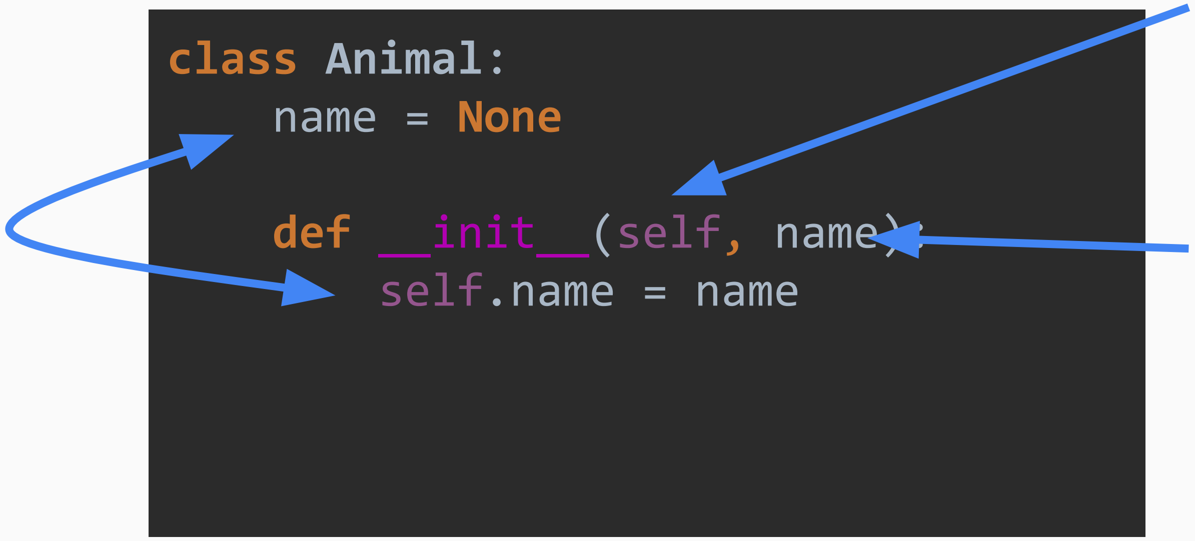
```
class Animal:
    name = None

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

Functions always have a reference to self (i.e. the class itself)

The constructor can take as many parameters as you want, currently we just take name

Python Classes

Setting the **class** variable to the **function** variable



```
class Animal:
    name = None

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

Functions always have a reference to self (i.e. the class itself)

The constructor can take as many parameters as you want, currently we just take name

Python Classes

Setting the **class** variable to the **function** variable

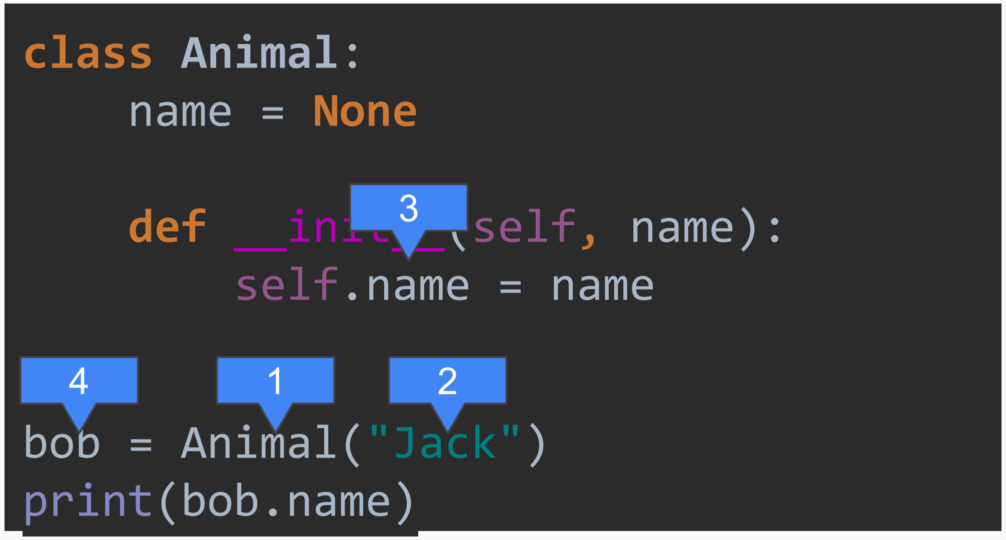
```
class Animal:
    name = None

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


bob = Animal("Bob")
print(bob.name)
```

Functions always have a reference to self (i.e. the class itself)

The constructor can take as many parameters as you want, currently we just take name

What does that do?

- Think of the class as a blueprint for an object
- An instance is the usage of a class



```python
class Animal:
    name = None

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

bob = Animal("Jack")
print(bob.name)
```

1. Make an ***instance*** of the class Animal
2. Pass the ***constructor*** its name “Jack”
3. The ***constructor*** sets the animal's name as “Jack”
4. The ***instance*** is stored in the variable bob
5. We print the name from bob

What will it print?

Python Classes - with Functions

```
class Animal:
    name = None

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

    def noise(self):
        print("Moo")


bob = Animal("Bob")
bob.noise()  # Prints Moo
```

Functions always have a reference to self (i.e. the class itself)

We can create multiple instances

- Think of the class as a blueprint for an object
- An instance is the usage of a class

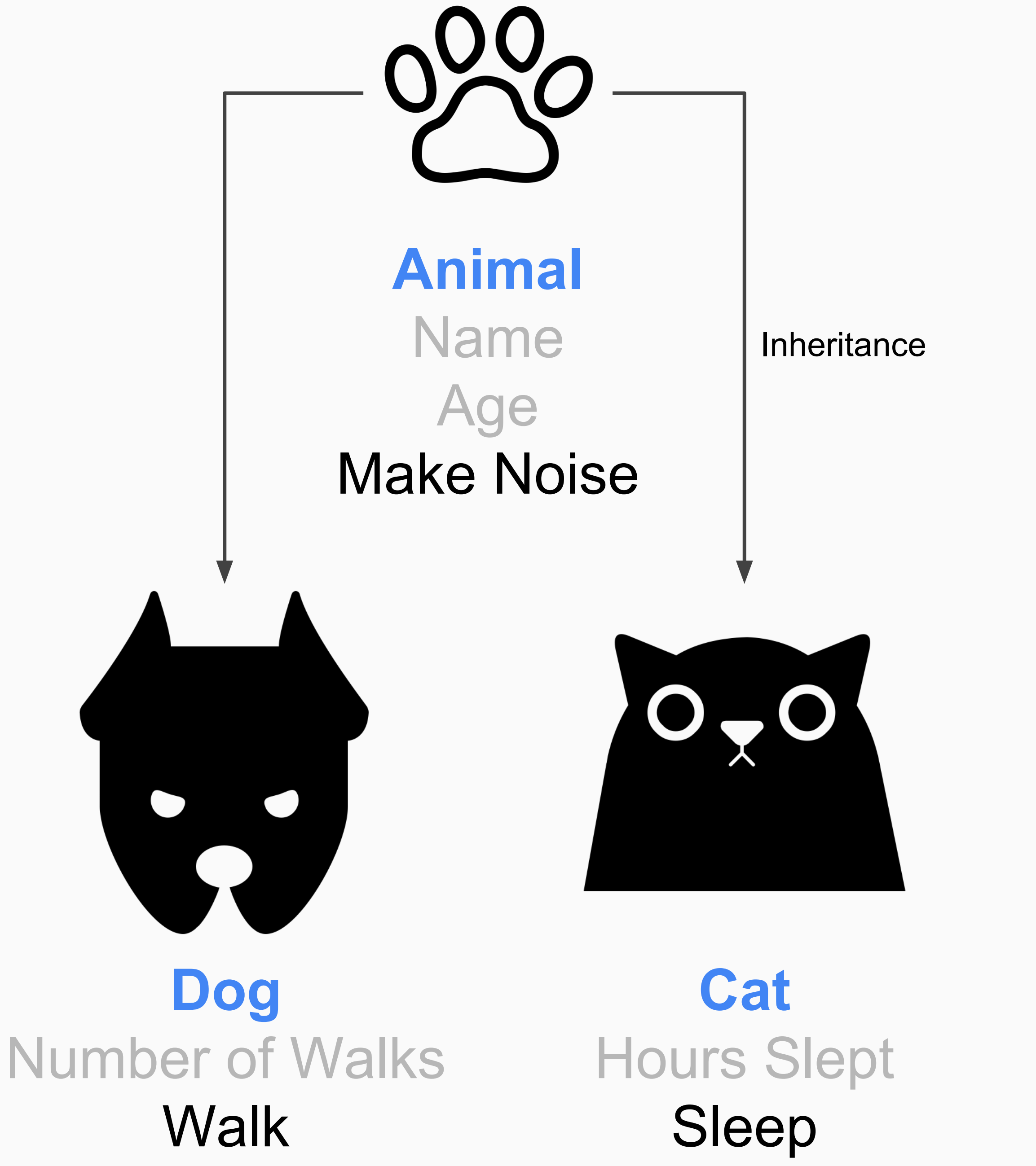
```python
class Animal:
    name = None

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


bob = Animal("Jack")
print(bob.name)  # Prints Jack

david = Animal("John")
print(david.name)  # Prints John
```

bob and **david** both hold instances of the **Animal** class

Pen and Paper

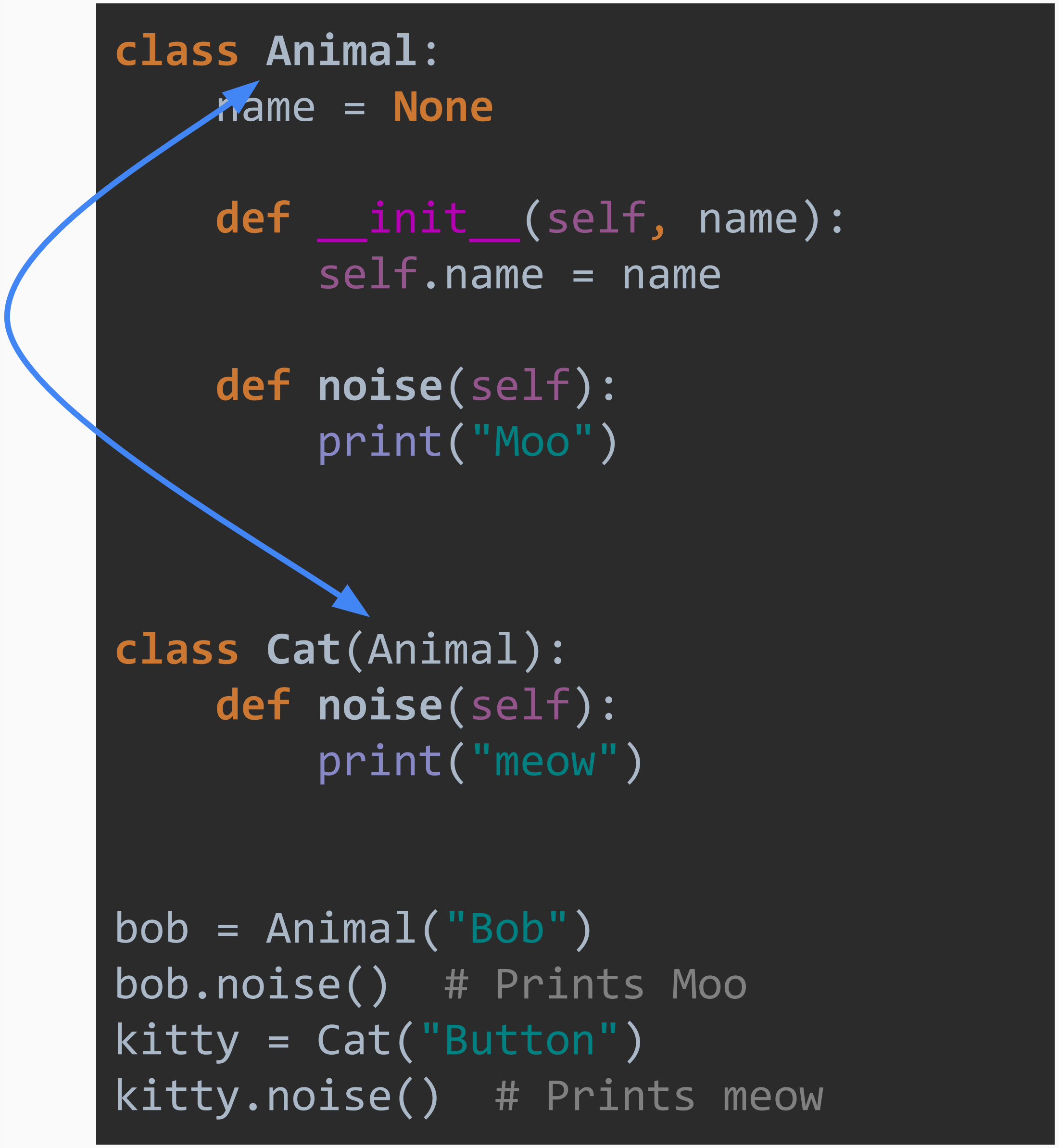
Checkpoint

Recall our inheritance example



Inheritance
(or parent classes)

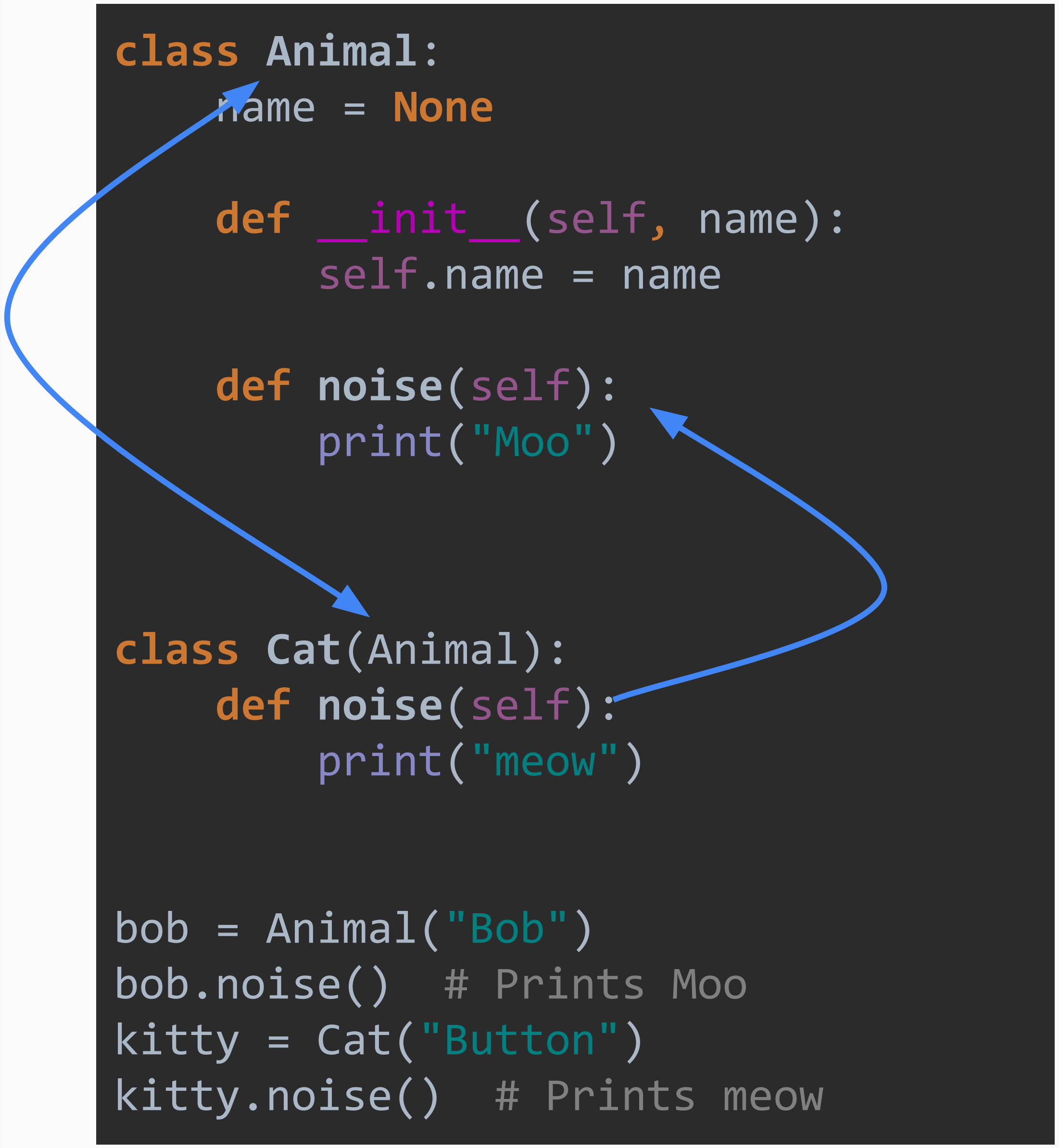
- Children inherit properties and methods from their parents



```python
class Animal:
    name = None

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

    def noise(self):
        print("Moo")


class Cat(Animal):
    def noise(self):
        print("meow")


bob = Animal("Bob")
bob.noise()  # Prints Moo
kitty = Cat("Button")
kitty.noise()  # Prints meow
```

Inheritance
(or parent classes)

- Children inherit properties and methods from their parents
- Subclass noise method overrides the parent's method



```python
class Animal:
    name = None

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

    def noise(self):
        print("Moo")


class Cat(Animal):
    def noise(self):
        print("meow")


bob = Animal("Bob")
bob.noise()  # Prints Moo
kitty = Cat("Button")
kitty.noise()  # Prints meow
```

Practical

References

- https://docs.python.org/3/tutorial/classes.html
- https://thenounproject.com/