Code For China
Lesson 7

Today’s Overview

- Classes & Inheritance
- Break 🤗
- Final Project Info
- Breakout Project

Classes

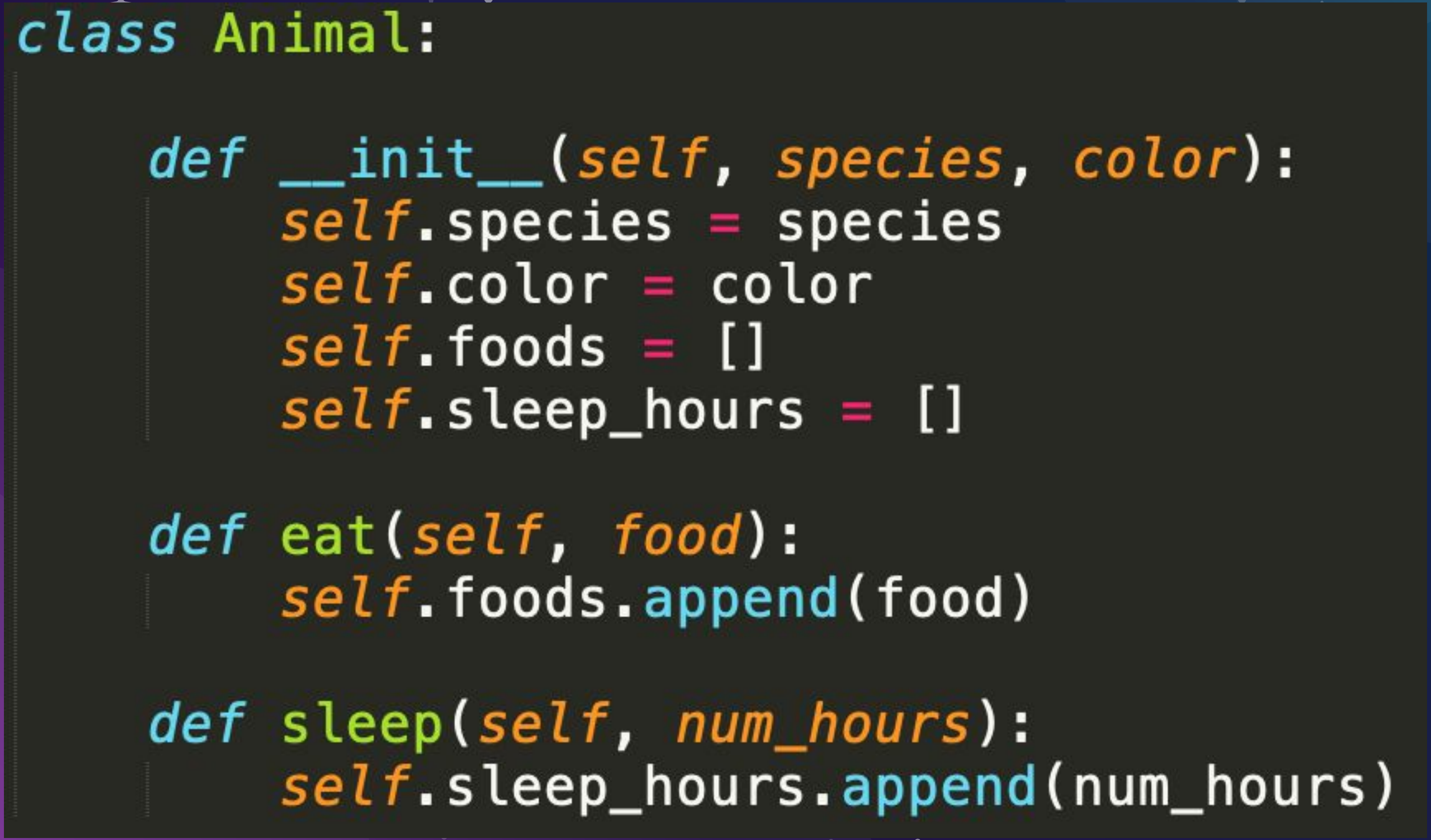
- So far, we have used **variables**
- What if we have a bunch of variables that are ***all related***?
- We can organize these variables using **classes**

For example...

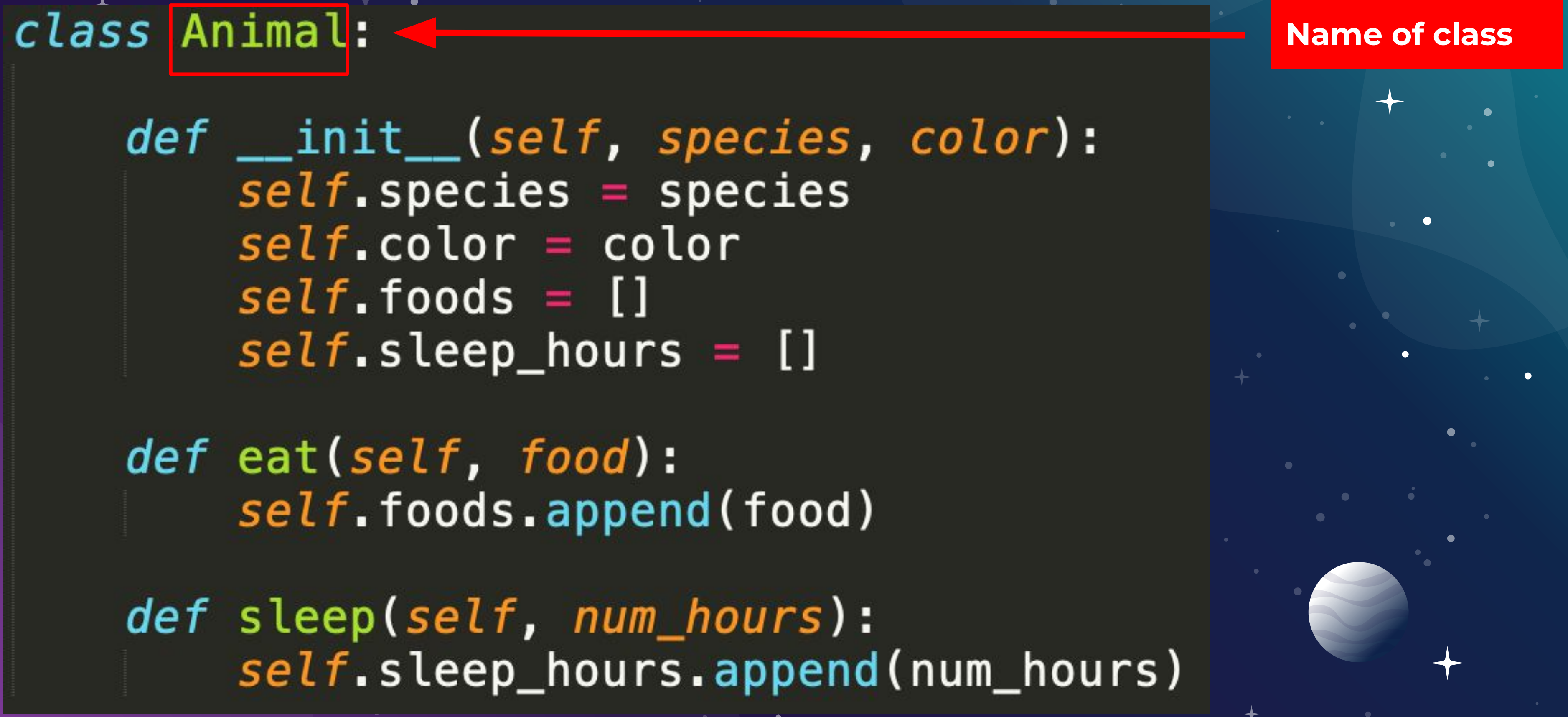
动物
Animals!

How does that all look like in code?

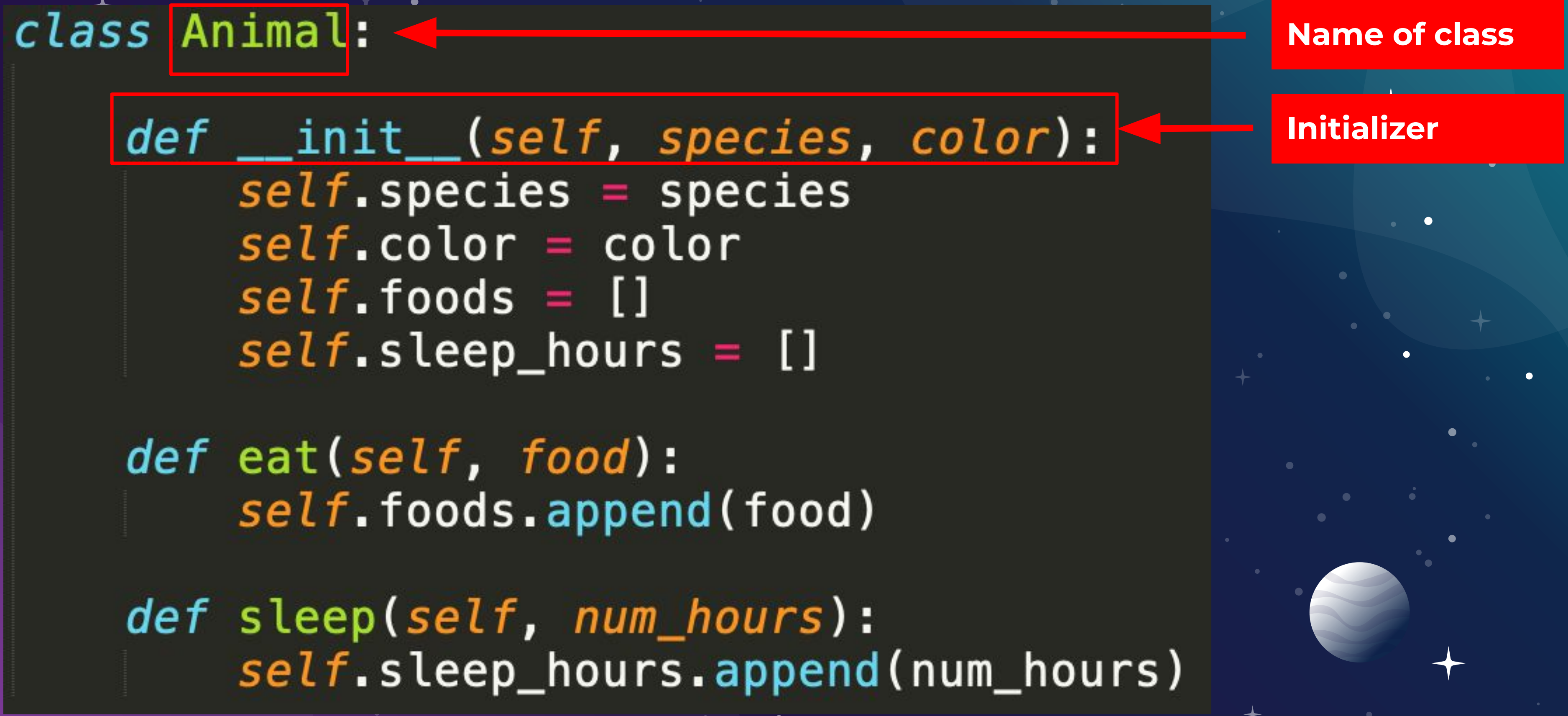
[example]



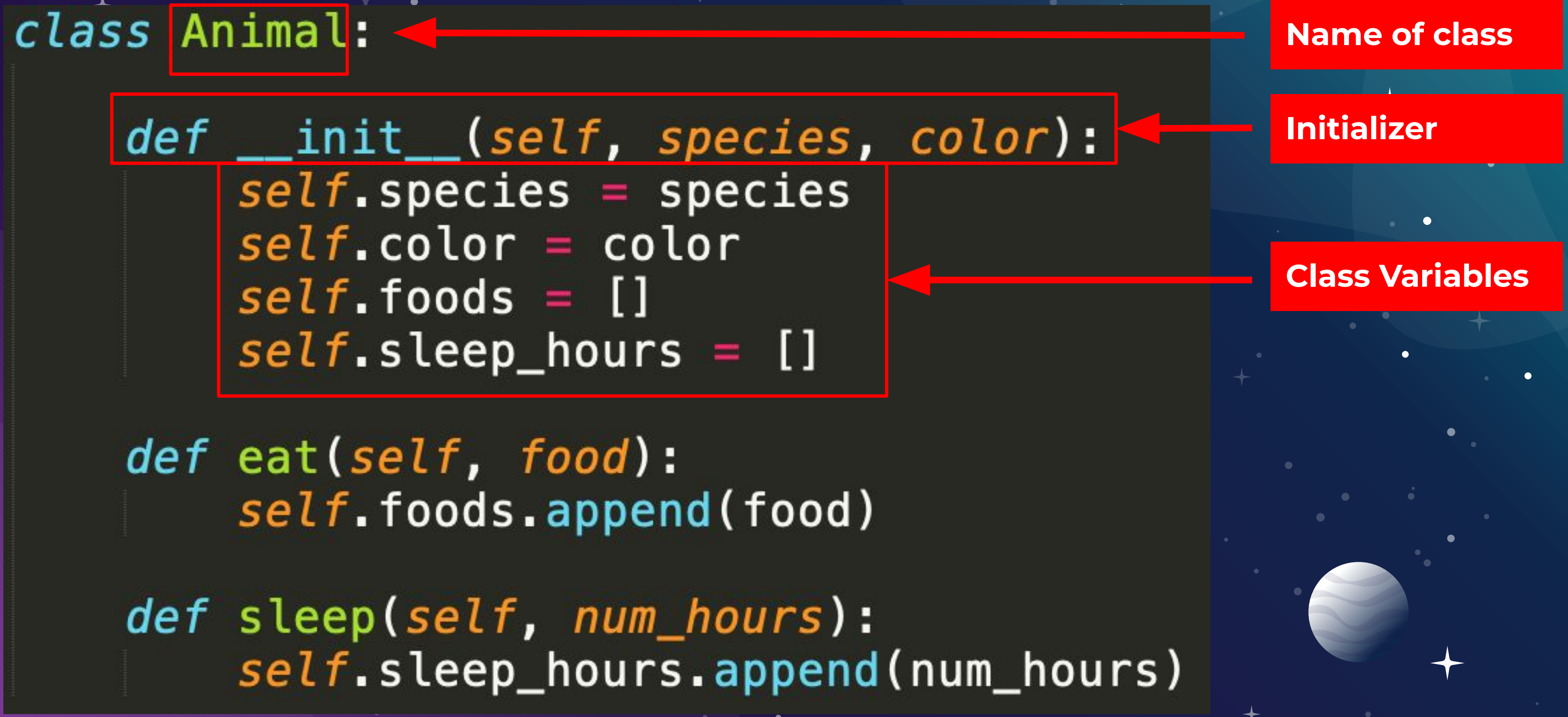
```python
class Animal:

    def __init__(self, species, color):
        self.species = species
        self.color = color
        self.foods = []
        self.sleep_hours = []

    def eat(self, food):
        self.foods.append(food)

    def sleep(self, num_hours):
        self.sleep_hours.append(num_hours)
```



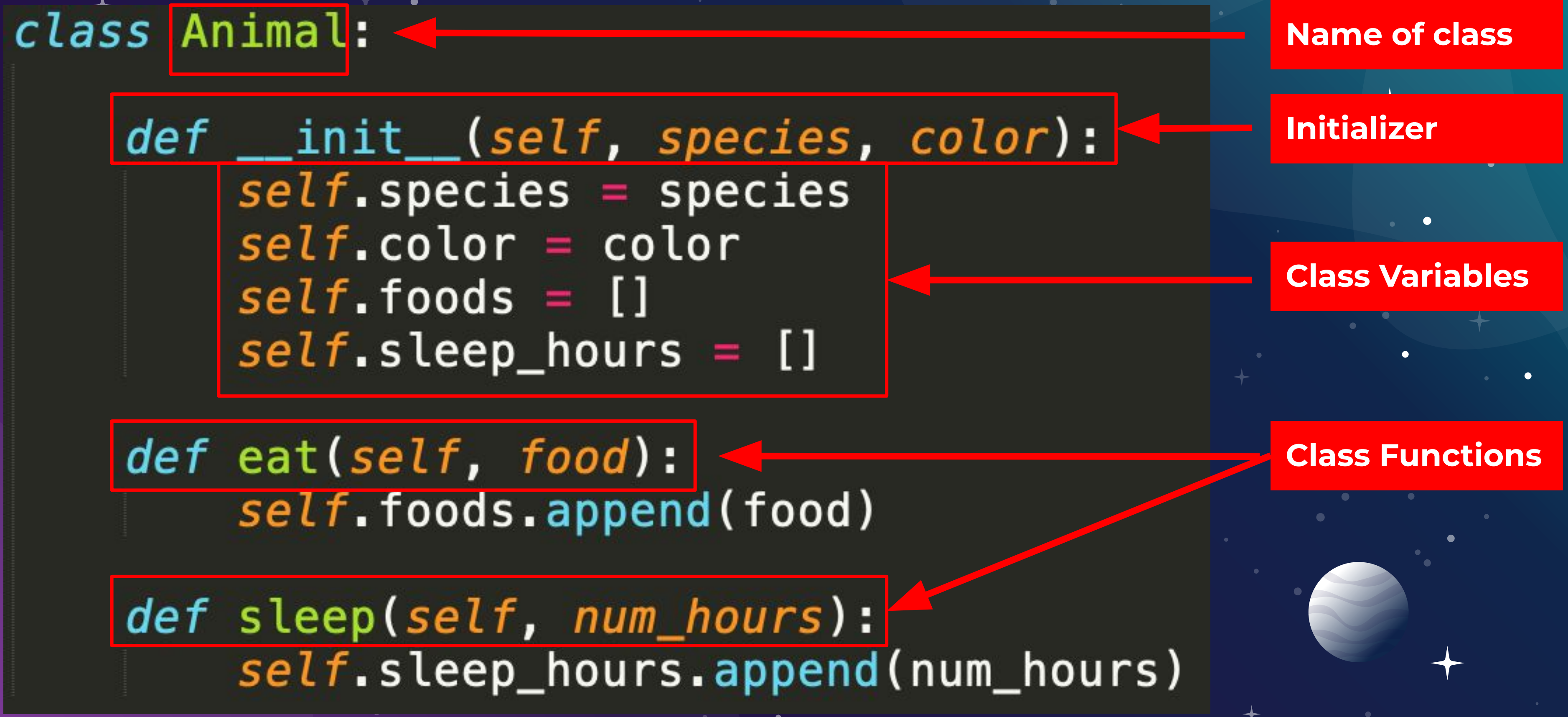
```
class Animal:

    def __init__(self, species, color):
        self.species = species
        self.color = color
        self.foods = []
        self.sleep_hours = []

    def eat(self, food):
        self.foods.append(food)

    def sleep(self, num_hours):
        self.sleep_hours.append(num_hours)
```



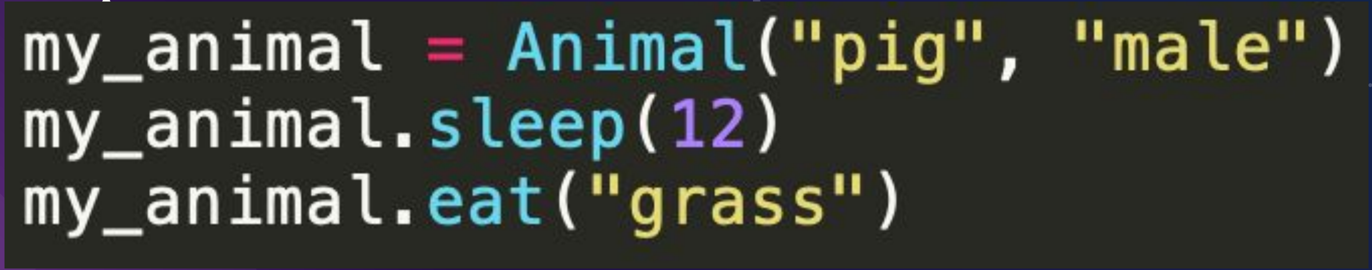
```
class Animal:

    def __init__(self, species, color):
        self.species = species
        self.color = color
        self.foods = []
        self.sleep_hours = []

    def eat(self, food):
        self.foods.append(food)

    def sleep(self, num_hours):
        self.sleep_hours.append(num_hours)
```



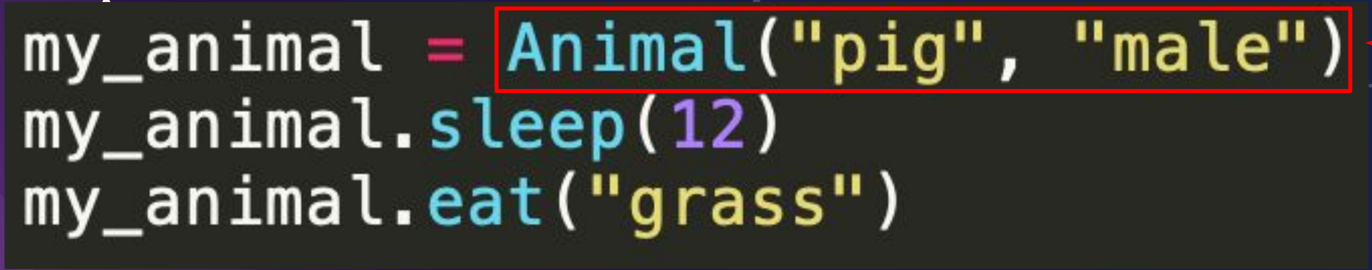
```
class Animal:

    def __init__(self, species, color):
        self.species = species
        self.color = color
        self.foods = []
        self.sleep_hours = []

    def eat(self, food):
        self.foods.append(food)

    def sleep(self, num_hours):
        self.sleep_hours.append(num_hours)
```



```python
class Animal:

    def __init__(self, species, color):
        self.species = species
        self.color = color
        self.foods = []
        self.sleep_hours = []

    def eat(self, food):
        self.foods.append(food)

    def sleep(self, num_hours):
        self.sleep_hours.append(num_hours)
```

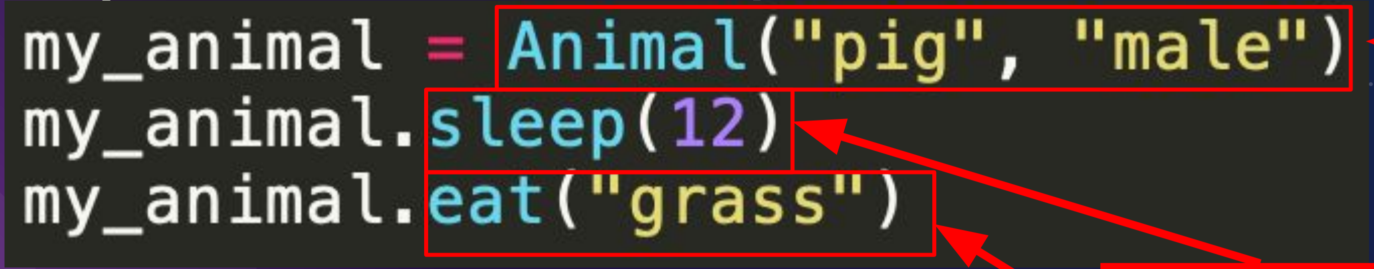


```python
my_animal = Animal("pig", "male")
my_animal.sleep(12)
my_animal.eat("grass")
```



```
my_animal = Animal("pig", "male")
my_animal.sleep(12)
my_animal.eat("grass")
```

Creating an **instance** of the Animal class



```
my_animal = Animal("pig", "male")
my_animal.sleep(12)
my_animal.eat("grass")
```

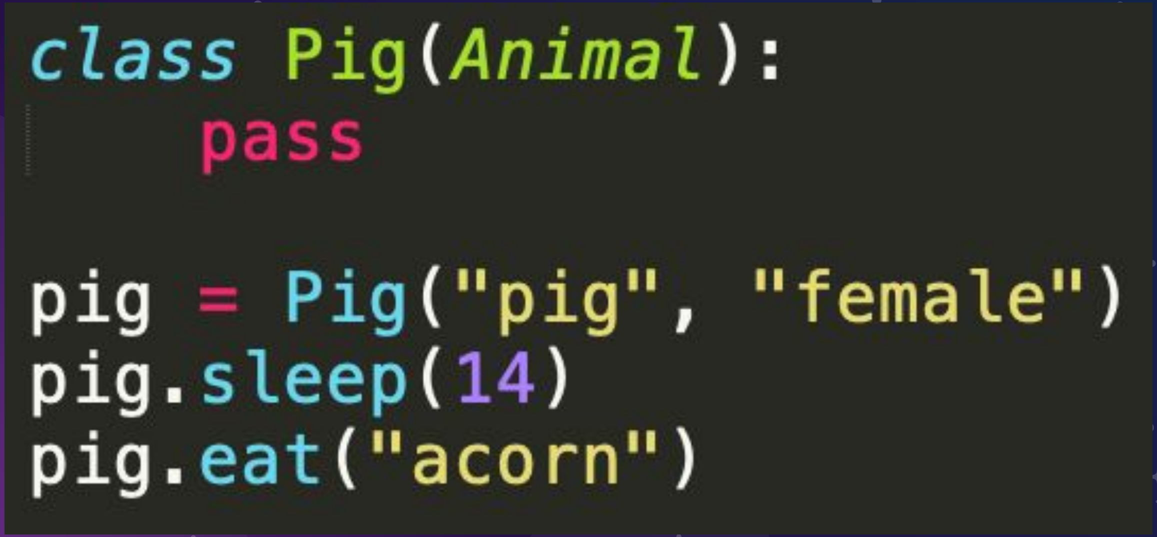
Creating an instance of the Animal class

Calling class functions on the instance

Inheritance - 继承

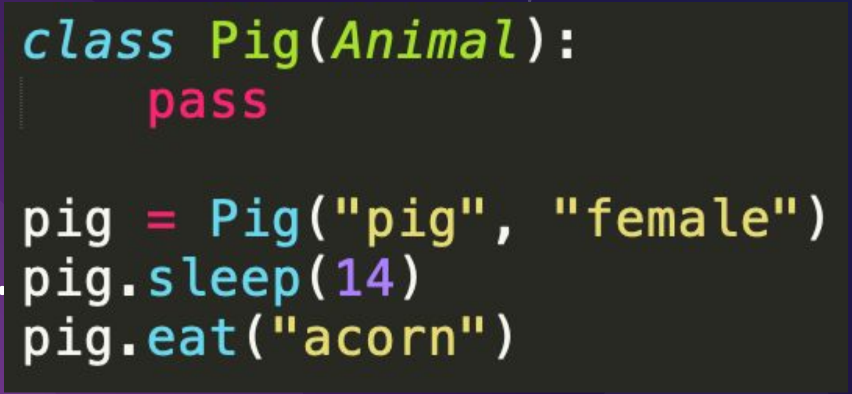
- Classes can **inherit properties and functions** from other classes
- This way, we can be lazy and not have to rewrite code 😛

For example, Pig is a type of animal. It can inherit the properties of the Animal class.



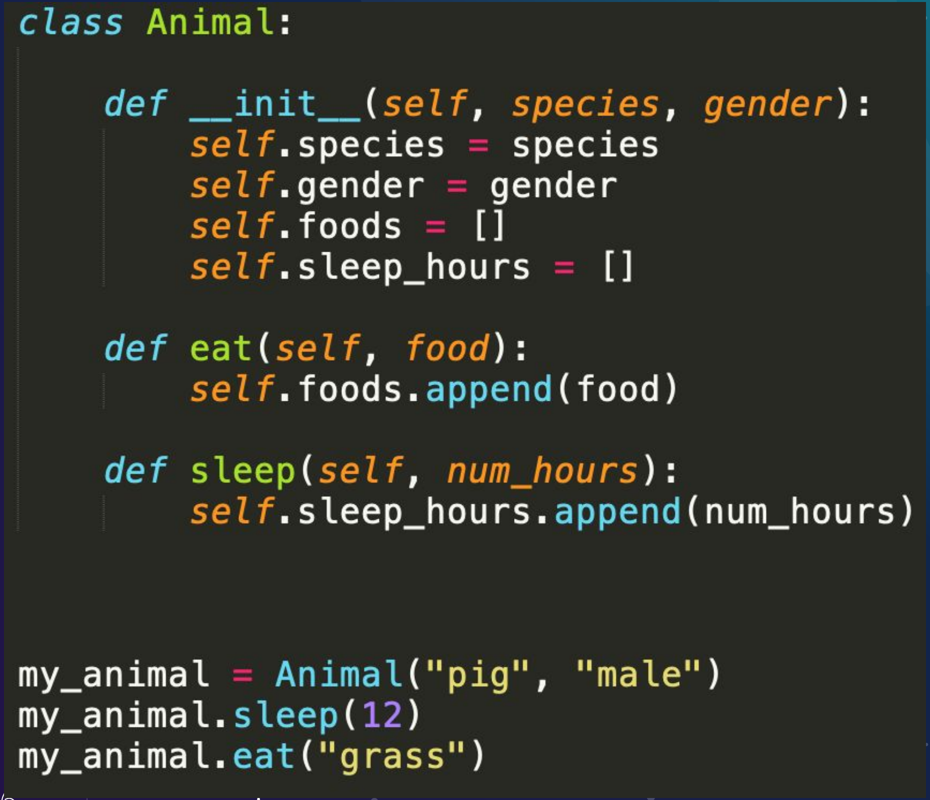
```
class Pig(Animal):
    pass

pig = Pig("pig", "female")
pig.sleep(14)
pig.eat("acorn")
```

Pig class

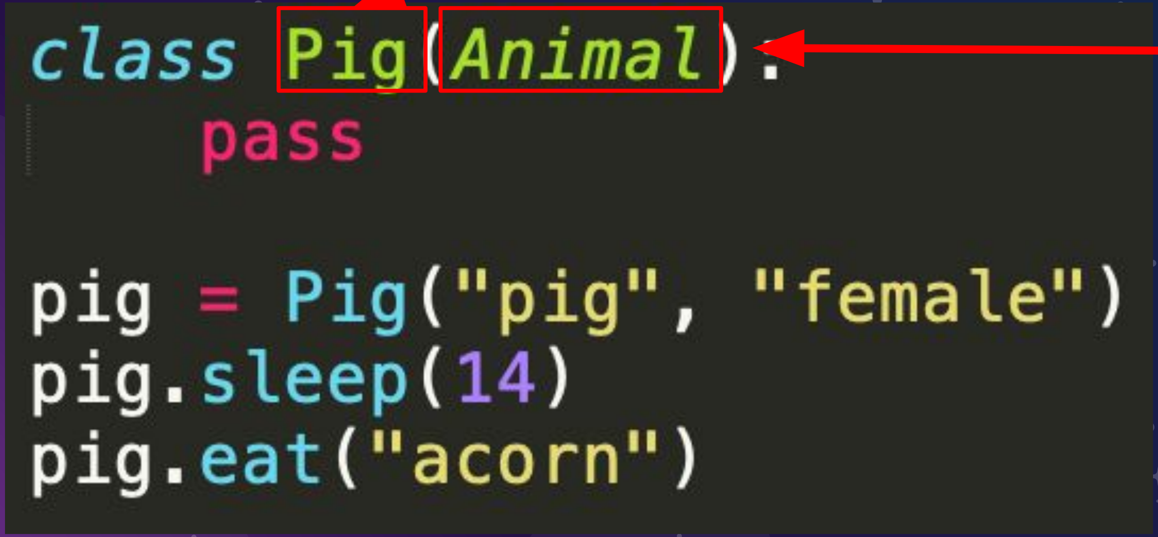


```python
class Pig(Animal):
    pass

pig = Pig("pig", "female")
pig.sleep(14)
pig.eat("acorn")
```

Animal class



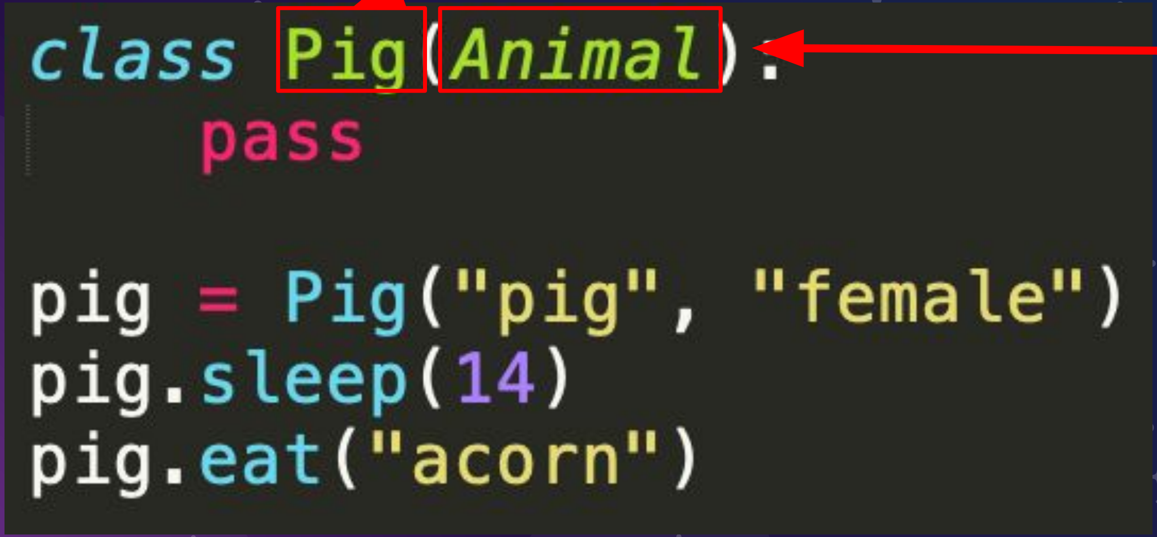
```python
class Animal:

    def __init__(self, species, gender):
        self.species = species
        self.gender = gender
        self.foods = []
        self.sleep_hours = []

    def eat(self, food):
        self.foods.append(food)

    def sleep(self, num_hours):
        self.sleep_hours.append(num_hours)



my_animal = Animal("pig", "male")
my_animal.sleep(12)
my_animal.eat("grass")
```



```python
class Pig(Animal):
    pass

pig = Pig("pig", "female")
pig.sleep(14)
pig.eat("acorn")
```

Name of class

Class it inherits from



```python
class Pig(Animal):
    pass

pig = Pig("pig", "female")
pig.sleep(14)
pig.eat("acorn")
```

Name of class

Class it inherits from

Calling Animal class functions on the Pig instance

Pig inherits the `init`, `sleep`, and `eat` functions from Animal, so we don't have to do anything!

We can also add properties / functions specific to the `Pig` class

What are some characteristics or actions of pigs that are unique / no other animals do?

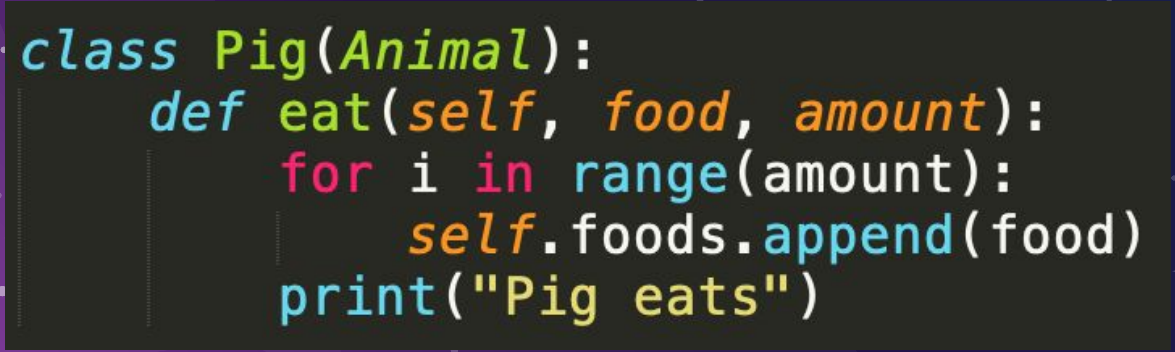
We can also override functions from the inherited `Animal` class

We can also override functions from the inherited `Animal` class

`Pig` inherits `eat` and `sleep` functions from `Animal`. Compared to other animals, pigs eat a lot. Perhaps we want to make its own `eat` function.

We can also override functions from the inherited `Animal` class

`Pig` inherits `eat` and `sleep` functions from `Animal`. Compared to other animals, pigs eat a lot. Perhaps we want to make its own `eat` function.

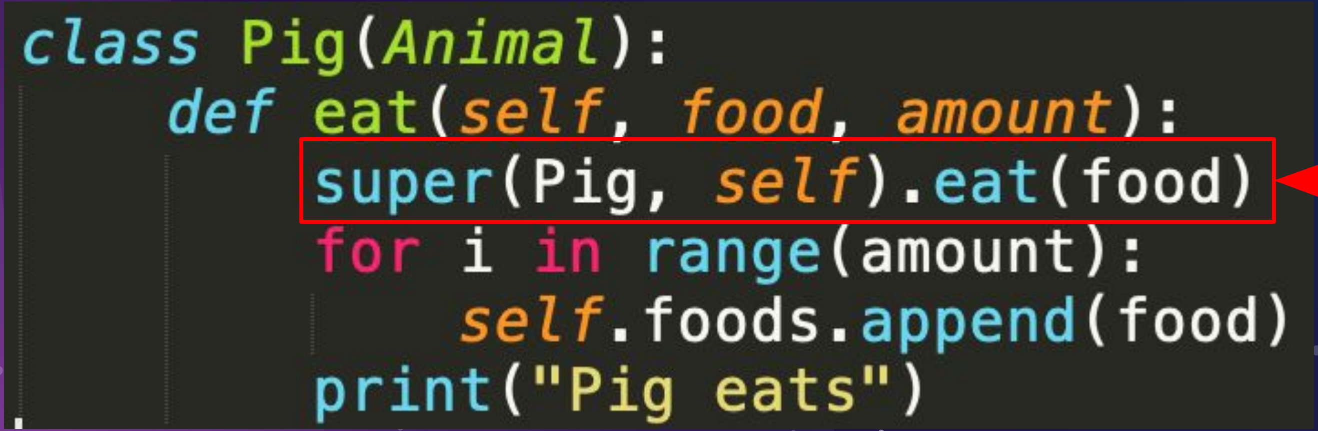


```python
class Pig(Animal):
    def eat(self, food, amount):
        for i in range(amount):
            self.foods.append(food)
        print("Pig eats")
```

Here, `self.foods` is not defined in Pig, but it is defined in Animal, (which Pig inherits from), thus we can use it

What about both: keep the **eat** function in **Animal** & also make a new **eat** function in **Pig**?

SOLUTION: Use **super**



```
class Pig(Animal):
    def eat(self, food, amount):
        super(Pig, self).eat(food)
        for i in range(amount):
            self.foods.append(food)
        print("Pig eats")
```

Using super

CODING TIME!

Activity: Design your own `class`

- Design 2 classes (or more)
- Think of the hierarchies & add **inheritance** (at least 1 class must inherit, but you can always do more 🤓)
- Examples: Cars, Humans, Books...and anything else you think of

This assignment is NOT about writing the correct code. It is about understanding inheritance and the relationship between classes.

15 min break 🥳

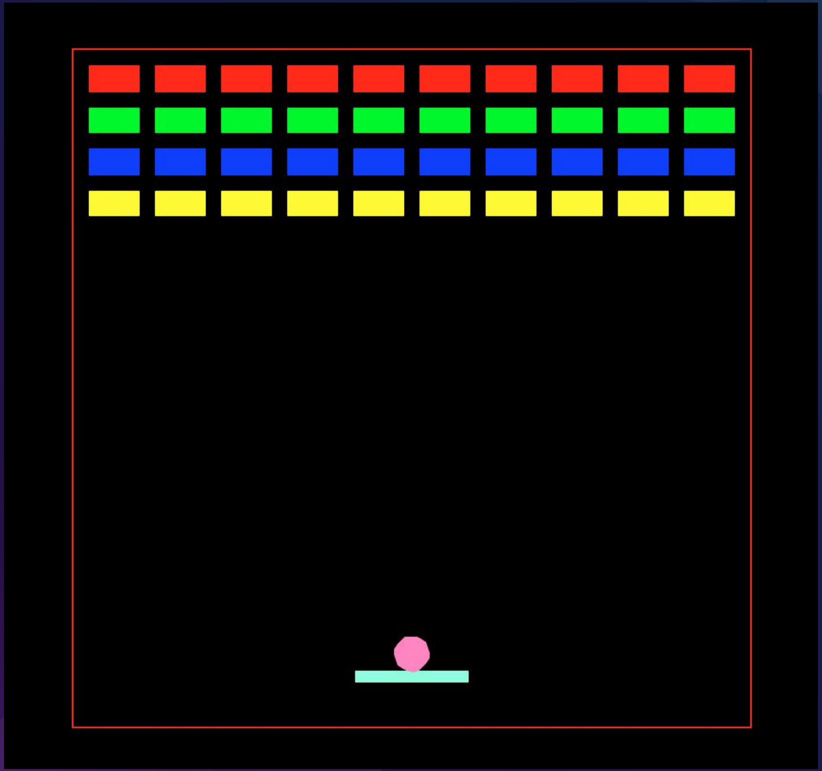
Structure for Future CS Classes

- First, finish Breakout
- Classes will be:
    - Short lectures on interesting topics (AI, Unity3D, algorithms, etc.)
    - Time to work on final projects with teacher help

Final Project

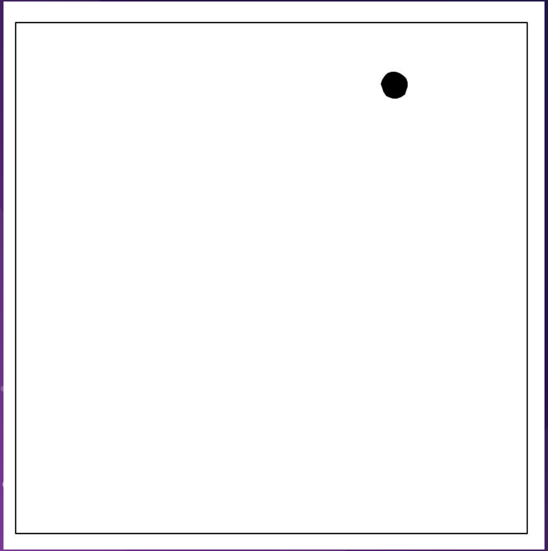
- **Goal:**
  - Code a Game
  - Can be new or extending something we did in class (e.g. Breakout)
- **Tools**
  - Can use turtle and what we learned in class
  - Can also use other libraries if you want
- Teams of 1-3
- Final Presentation
  - Share your project with parents / classmates / teachers
- Talk to us if you need ideas or help 😊

Breakout Game



Steps

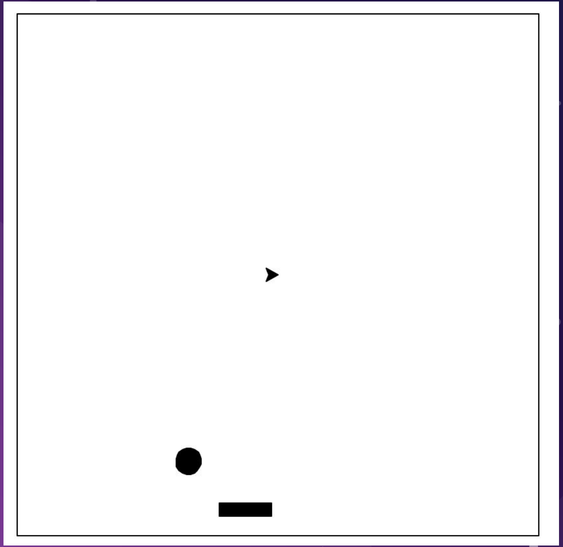
1. **First Make Ball Bounce**



- The ball begins with a random downward-facing angle and continues moving infinitely
- If it collides with the border:
  - If corner: reverse direction
  - If wall: switch angle

Steps

2. Next, add paddle



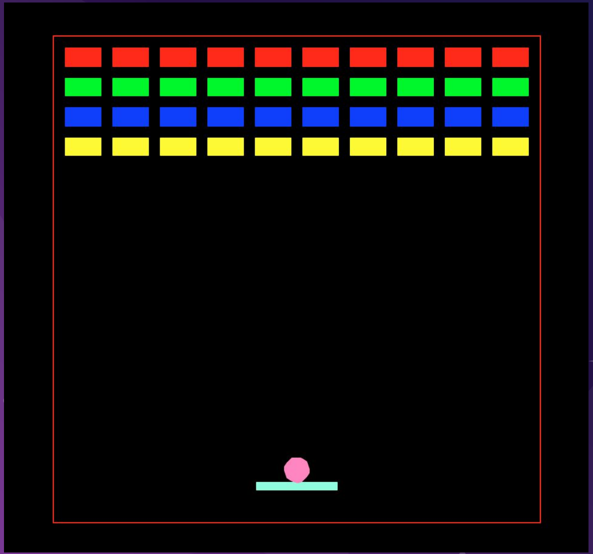
- Now, add the paddle
- User can move the paddle using the Left / Right keys
- The ball can now collide with the paddle and change direction (switch angle)

Hint:

- If you draw the paddle using paddle.shape(“square”), its coordinates are at the **center**
- Default square size is (20, 20)
- Must use: **paddle.shapesize(paddle_height/20.0, paddle_width / 20.0)**

Steps

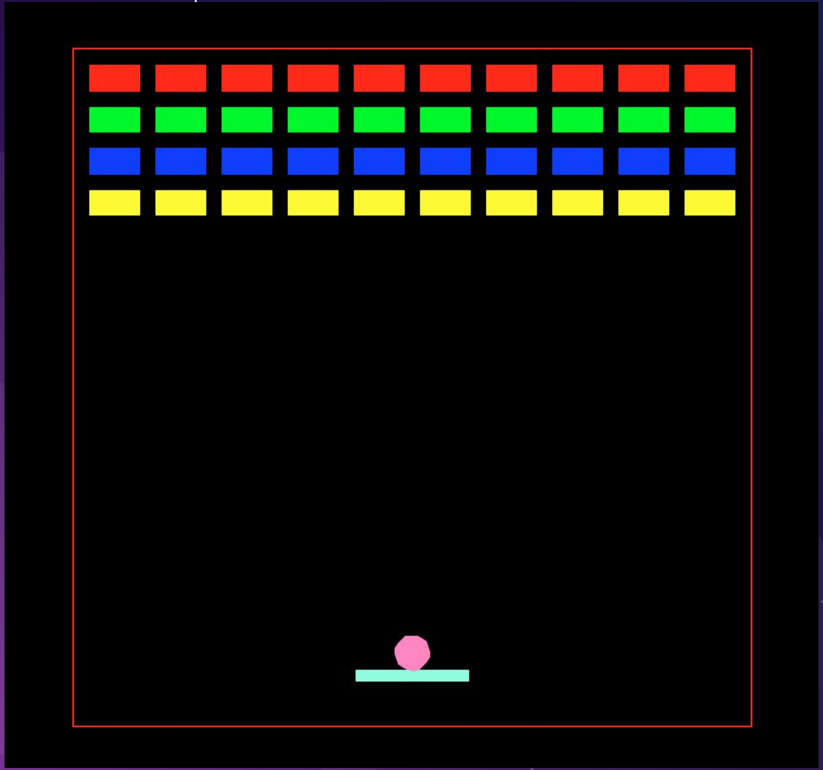
3. Complete Game



- Add the bricks
- Check for collisions with bricks, and remove the brick if hit by ball
- Add colors, other design choices

- *Extra*:
    - Tough bricks that require multiple hits to disappear
    - Add score label
    - Accelerate ball as game continues
    - Any other ideas!

Breakout Starter Code



github.com/jessicae5/CodeForChina

Turtle Commands

```
turtle.forward(length)
turtle.backward(length)
turtle.left(angle)
turtle.right(angle)
turtle.goto(x, y)
turtle.setheading(angle)
turtle.shape("turtle")
turtle.pensize(width)
turtle.showturtle()
turtle.hideturtle()
```

For more:
https://docs.python.org/3.3/library/turtle.html?highlight=turtle

Turtle setHeading()

```
turtle.setHeading(angle)
```