CLASSES IN PYTHON

WHO IS THIS FOR?

a beginner to intermediate Python programmer

WHAT SHOULD YOU ALREADY KNOW?

a basic understanding of Python

a good understanding of functions

WHAT WE WILL COVER

01
why use classes and not functions

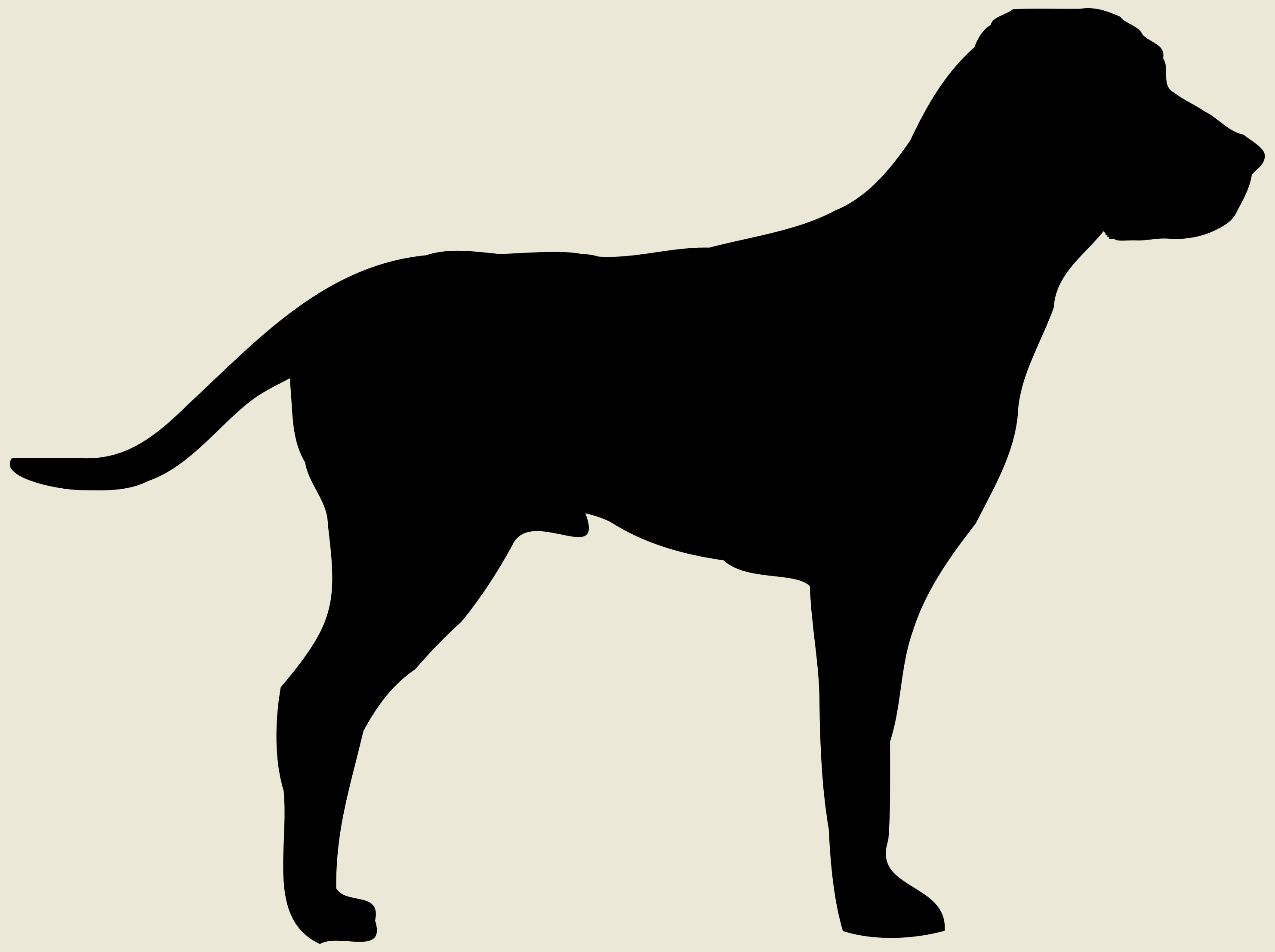
02
define and use a simple class

03
store and use data

04
write and use simple methods in a class to explore

VOCABULARY

CLASS



a general outline of what you want to model

OBJECT

an instance of that model

CLASSES VS FUNCTIONS

FUNCTIONS

Make programs easier to

- read
- write
- test
- fix

CLASSES

Can do all the things a function can do but more!

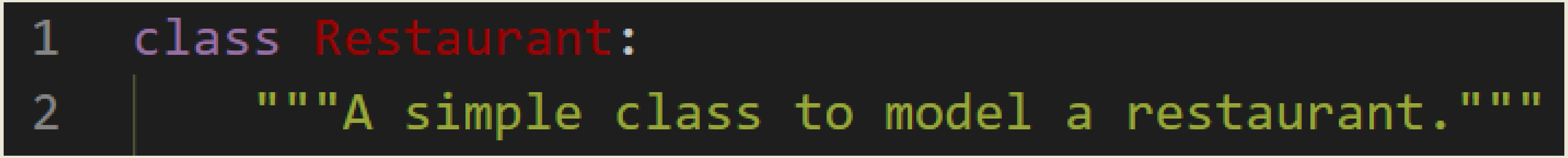
- hold multiple data values
- hold more than one function
- functions can interact with one another

DEFINE AND USE A SIMPLE CLASS

LET'S MAKE A CLASS TO MODEL A RESATAURANT

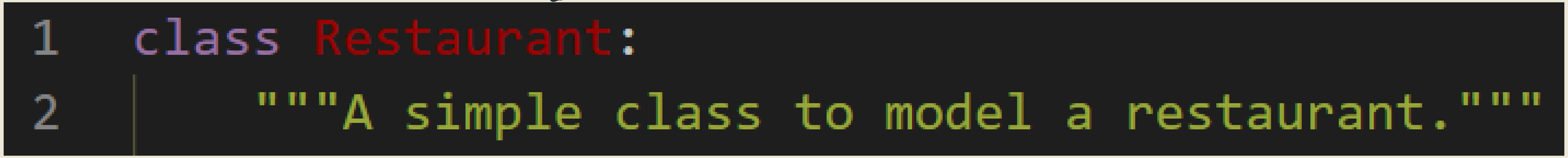
DEFINE A CLASS

use the keyword **class**



```python
class Restaurant:
    """A simple class to model a restaurant."""
```

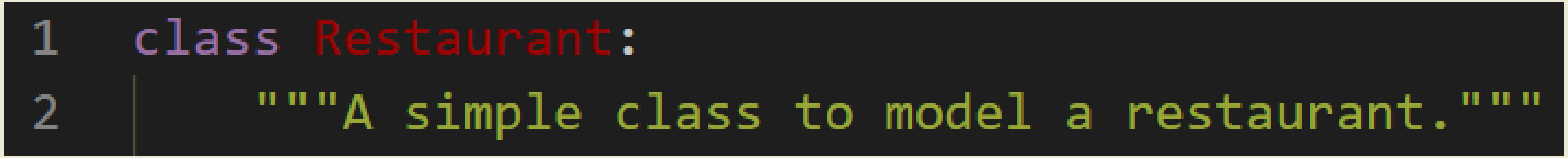
follow it with the **Name** of your class



```python
class Restaurant:
    """A simple class to model a restaurant."""
```

NOTE:

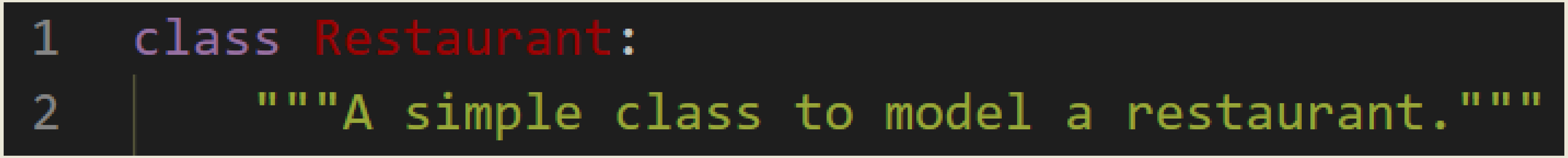
capitalized first letter



```python
class Restaurant:
    """A simple class to model a restaurant."""
```

NOTE:

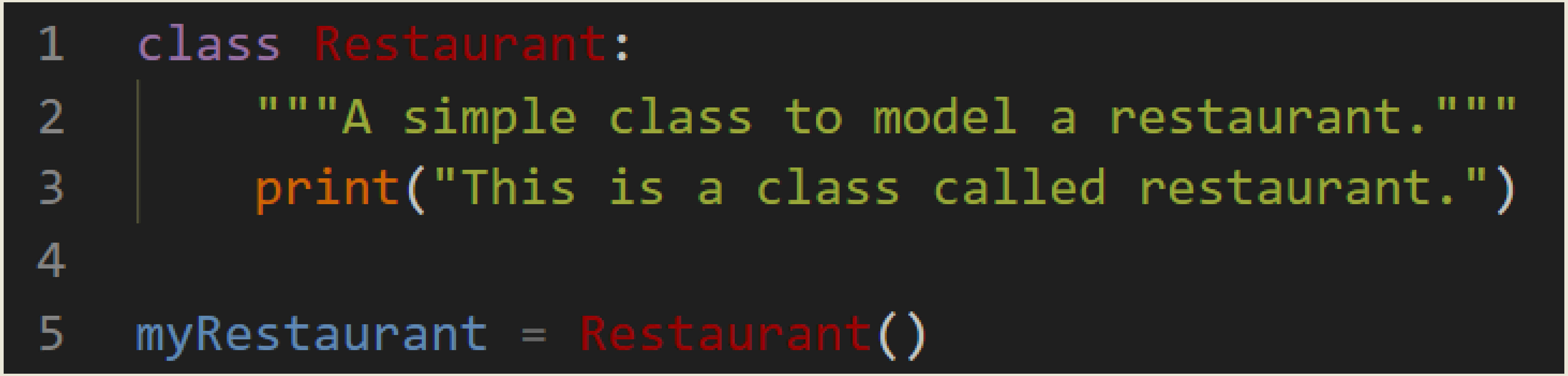
no parethesis



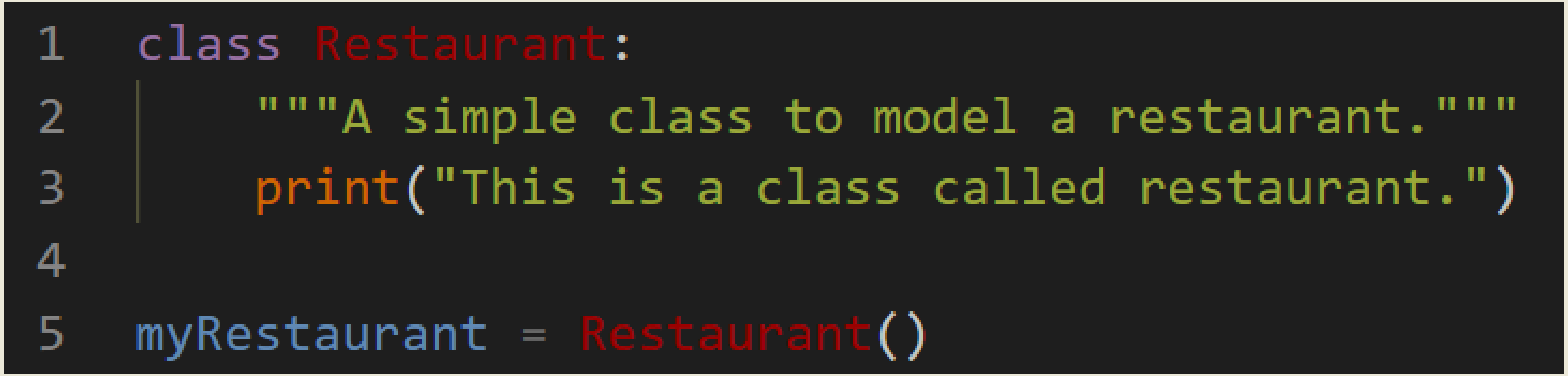
```python
class Restaurant:
    """A simple class to model a restaurant."""
```

LET'S GET THE CLASS TO PRINT OUT SOMETHING

HERE'S OUR CLASS SO FAR



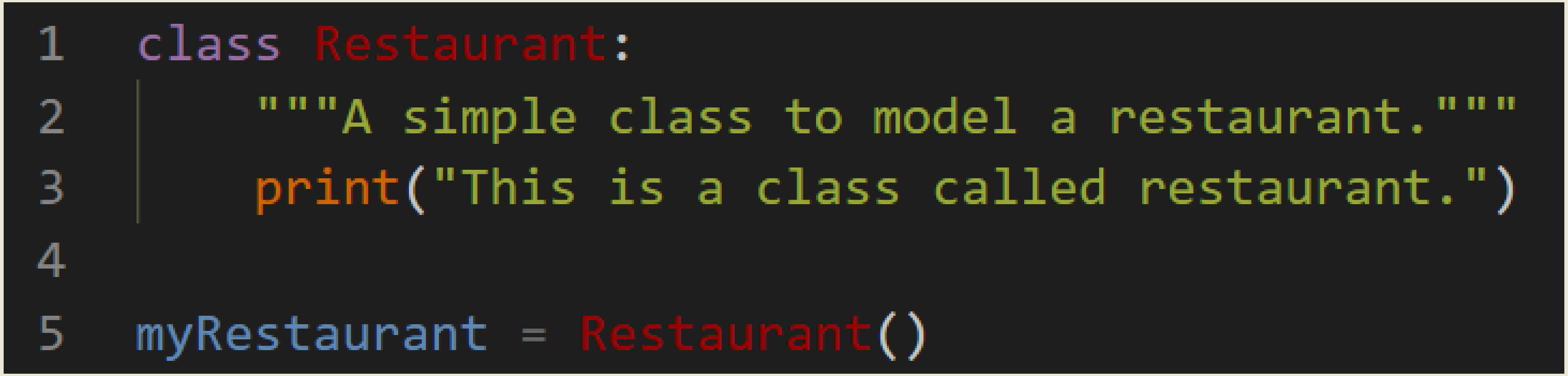
```python
class Restaurant:
    """A simple class to model a restaurant."""
    print("This is a class called restaurant.")

myRestaurant = Restaurant()
```



```python
class Restaurant:
    """A simple class to model a restaurant."""
    print("This is a class called restaurant.")

myRestaurant = Restaurant()
```

assign an object to a class

INSTANTIATE



```python
class Restaurant:
    """A simple class to model a restaurant."""
    print("This is a class called restaurant.")

myRestaurant = Restaurant()
```

OUTPUT

```
This is a class called restaurant.
```

STORE AND USE DATA IN A CLASS

LET'S HAVE OUR RESTAURANT CLASS STORE SOMETHING

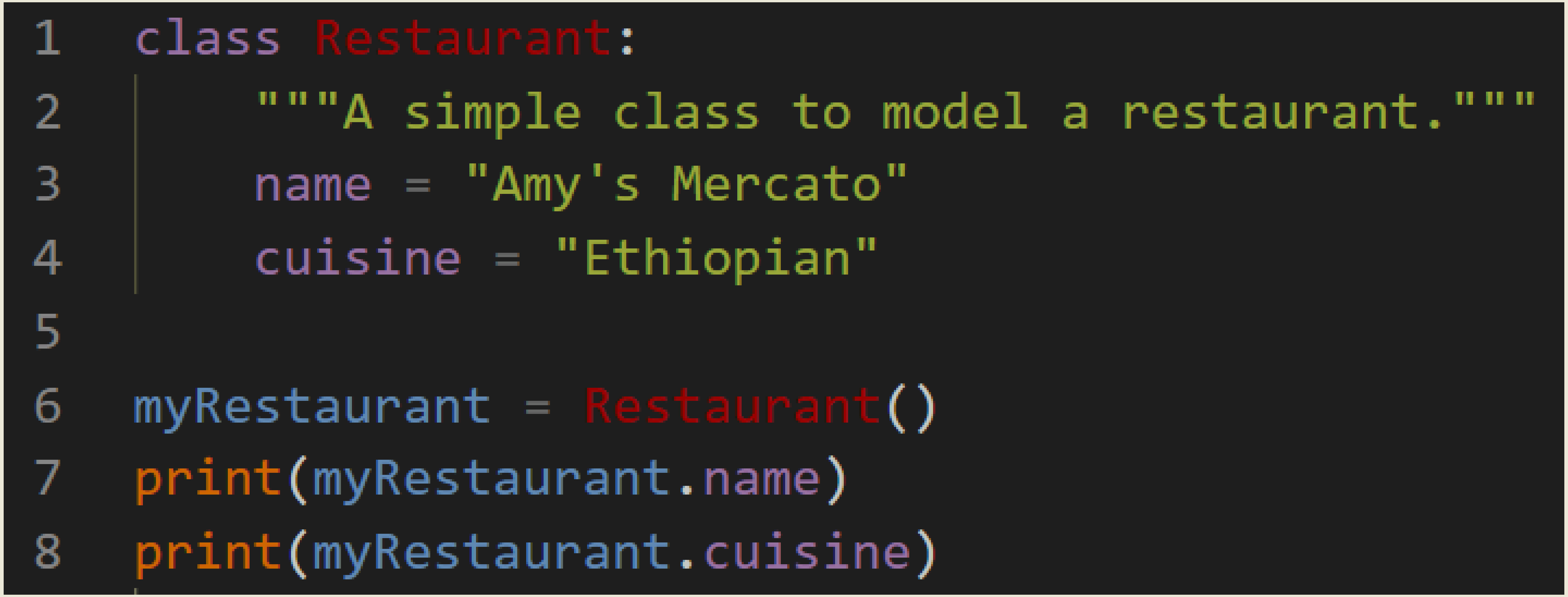
- name
- cuisine

TWO WAYS TO STORE DATA

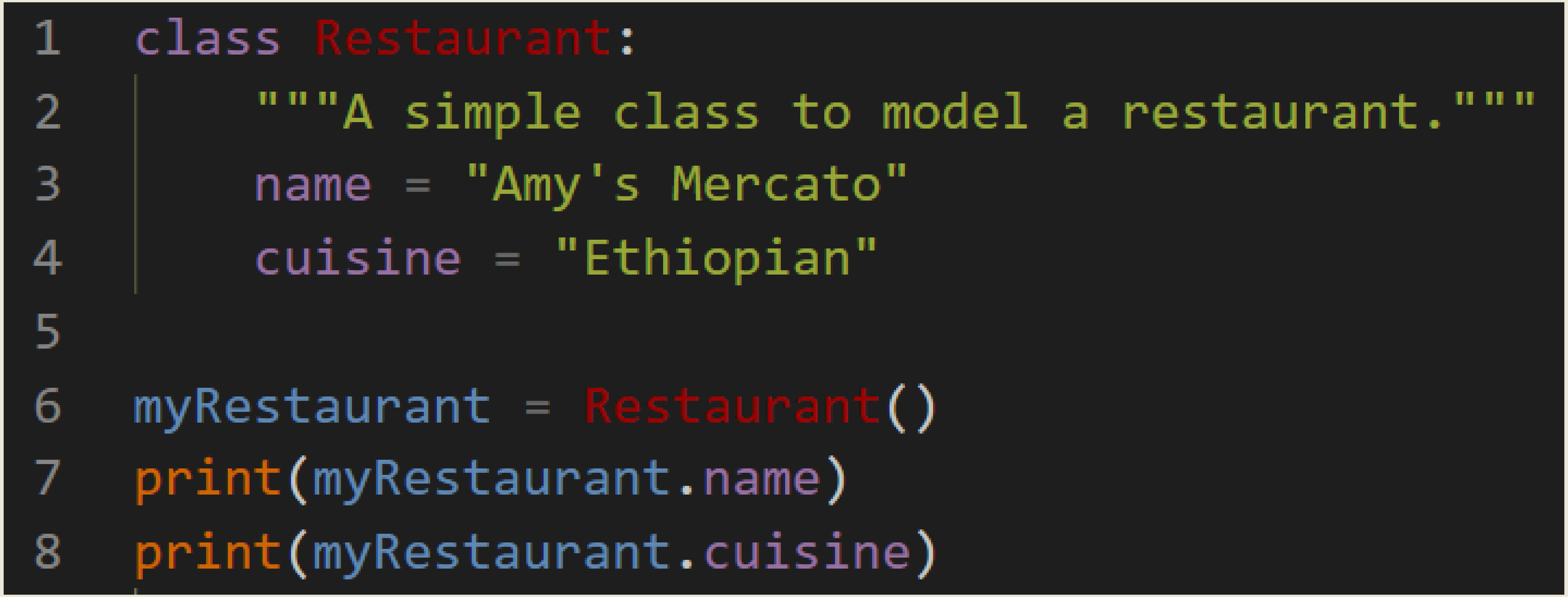
- Using class variables
- Using constructors

USE CLASS VARIABLES

same data for every instance of the class

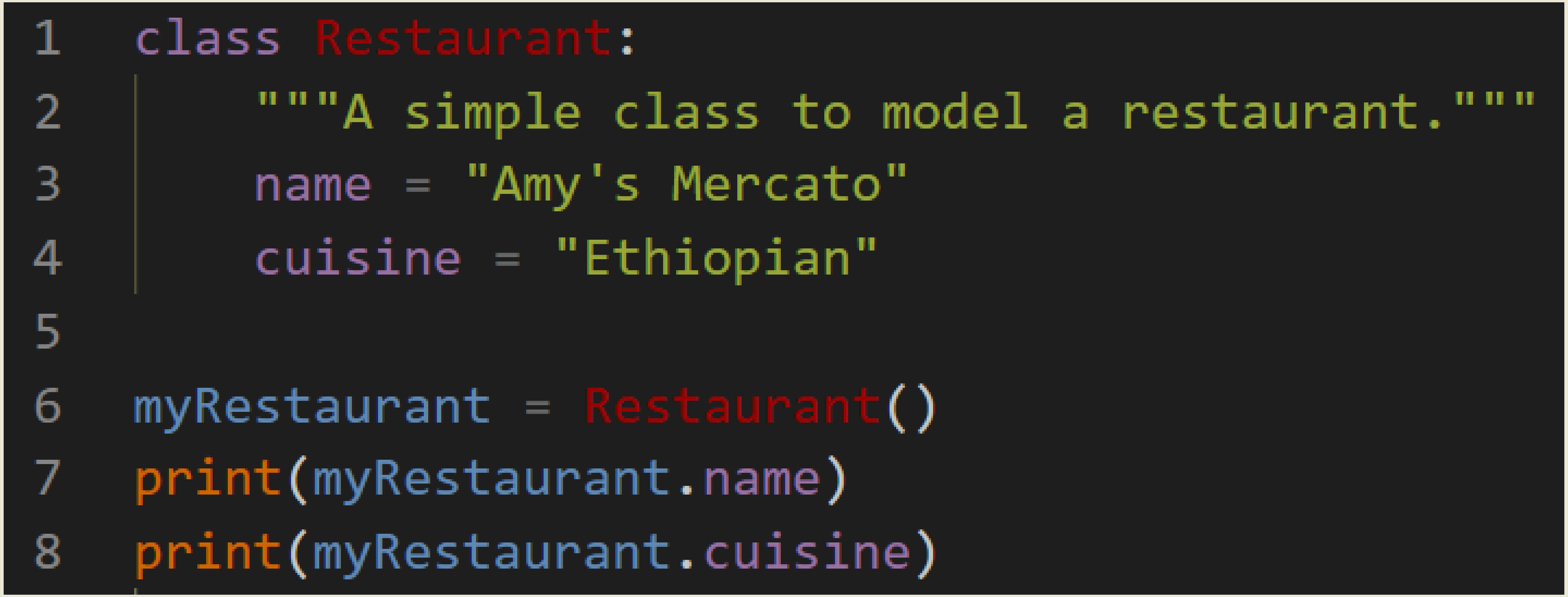


```python
class Restaurant:
    """A simple class to model a restaurant."""
    name = "Amy's Mercato"
    cuisine = "Ethiopian"

myRestaurant = Restaurant()
print(myRestaurant.name)
print(myRestaurant.cuisine)
```



```python
class Restaurant:
    """A simple class to model a restaurant."""
    name = "Amy's Mercato"
    cuisine = "Ethiopian"

myRestaurant = Restaurant()
print(myRestaurant.name)
print(myRestaurant.cuisine)
```

Access class variables with **object.variable**



```python
class Restaurant:
    """A simple class to model a restaurant."""
    name = "Amy's Mercato"
    cuisine = "Ethiopian"

myRestaurant = Restaurant()
print(myRestaurant.name)
print(myRestaurant.cuisine)
```

OUTPUT

```
Amy's Mercato
Ethiopian
```

USE A CONSTRUCTOR

```
                                          header0_initialPadding + 'px');
if ($(window).scrollTop() > header1_initialDistance) {
    if (parseInt(header1.css('padding-top'), 10) >= header1_initialPadding
        header1.css('padding-top', '' + $(window).scrollTop() -
    }
} else {
    header1.css('padding-top', '' + header1_initialPadding + 'px');
}

if ($(window).scrollTop() > header2_initialDistance) {
    if (parseInt(header2.css('padding-top'), 10) >= header2_initialPadding
        header2.css('padding-top', '' + $(window).scrollTop() -
    }
} else {
    2.css('padding-top', '' + header2_initialPadding
```

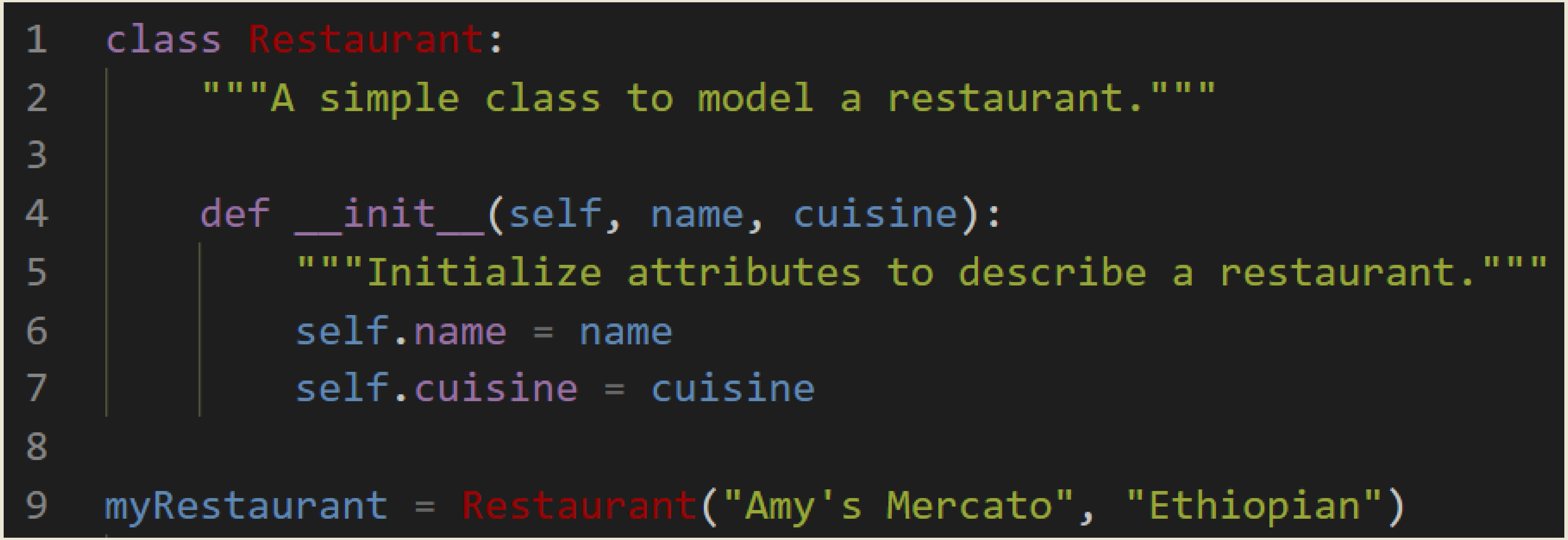
METHOD

a function that is inside a class

USE THE __init__ METHOD

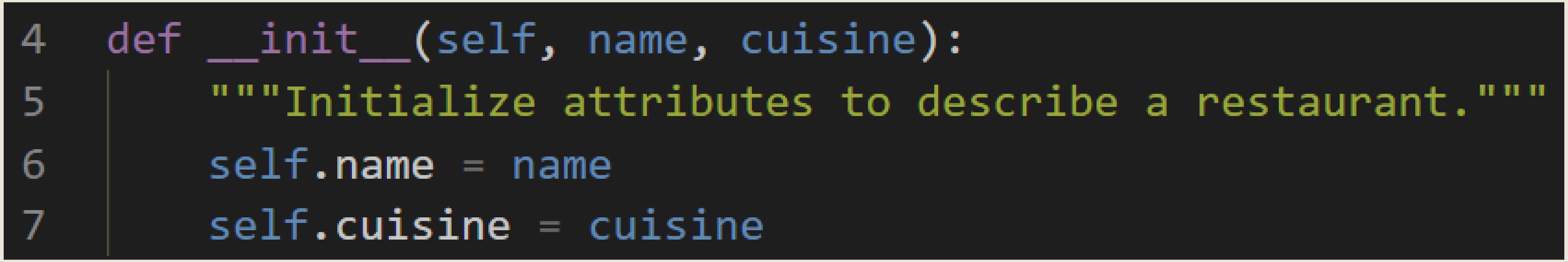
- referred to as constructor
- assign different values to class variables using arguments

HERE'S OUR CLASS WITH THE __init__ METHOD


```python
class Restaurant:
    """A simple class to model a restaurant."""

    def __init__(self, name, cuisine):
        """Initialize attributes to describe a restaurant."""
        self.name = name
        self.cuisine = cuisine

myRestaurant = Restaurant("Amy's Mercato", "Ethiopian")
```

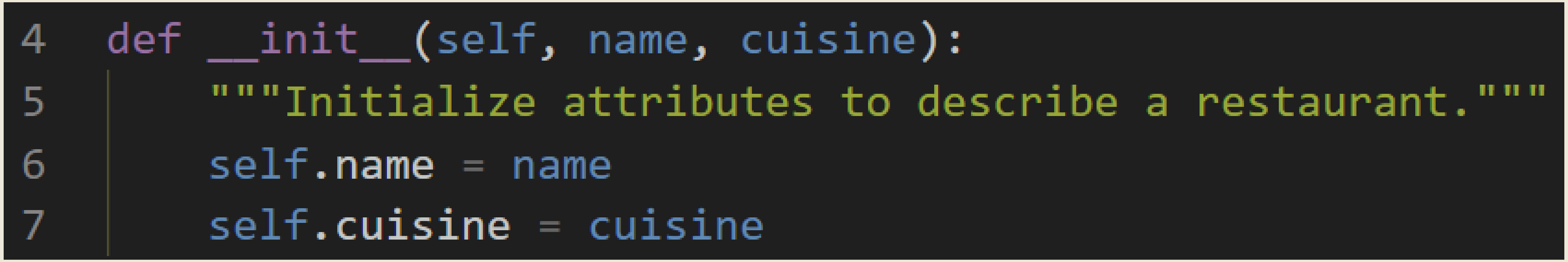
LET'S GET A CLOSER LOOK AT THE __init__ METHOD

start with def



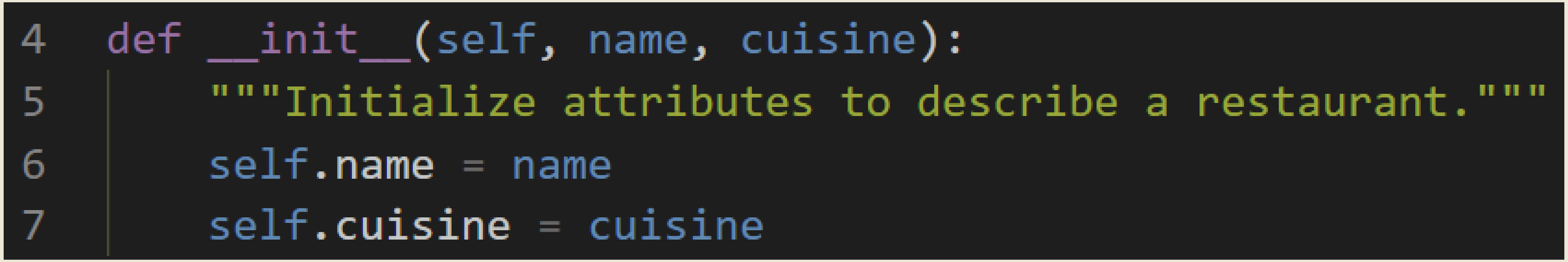
```python
def __init__(self, name, cuisine):
    """Initialize attributes to describe a restaurant."""
    self.name = name
    self.cuisine = cuisine
```

two underscores before and after



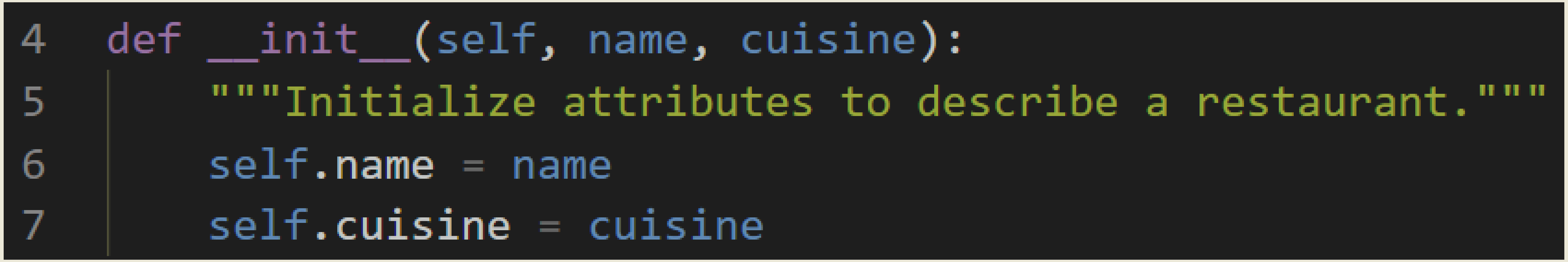
```python
def __init__(self, name, cuisine):
    """Initialize attributes to describe a restaurant."""
    self.name = name
    self.cuisine = cuisine
```

always gets a **self** argument

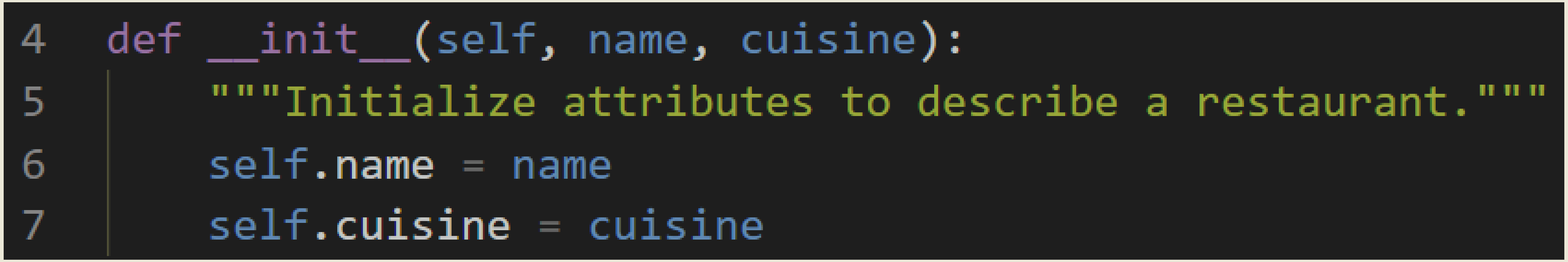


```python
def __init__(self, name, cuisine):
    """Initialize attributes to describe a restaurant."""
    self.name = name
    self.cuisine = cuisine
```

other arguments added after a comma

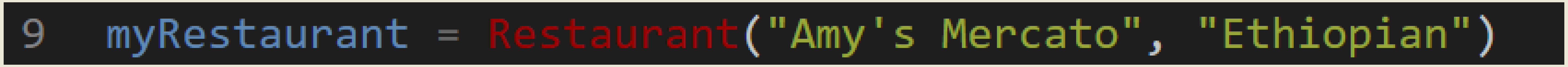


```python
def __init__(self, name, cuisine):
    """Initialize attributes to describe a restaurant."""
    self.name = name
    self.cuisine = cuisine
```



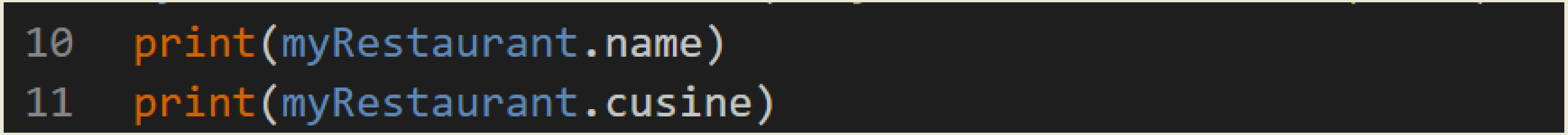
```python
def __init__(self, name, cuisine):
    """Initialize attributes to describe a restaurant."""
    self.name = name
    self.cuisine = cuisine
```

assign class variables from the arguments



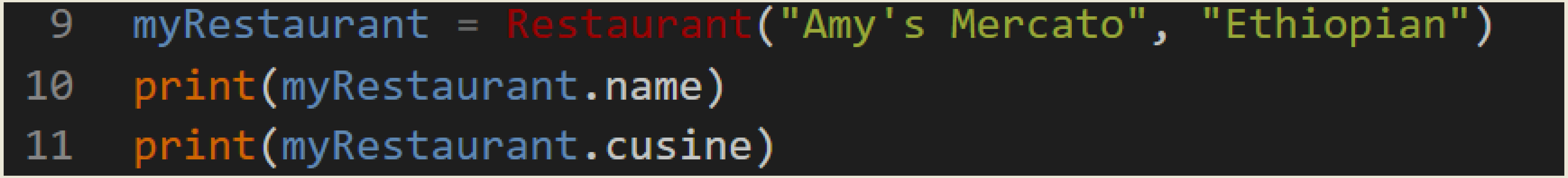
```python
myRestaurant = Restaurant("Amy's Mercato", "Ethiopian")
```

make objects by giving arguments



```python
print(myRestaurant.name)
print(myRestaurant.cusine)
```

access attributes the same way



```python
myRestaurant = Restaurant("Amy's Mercato", "Ethiopian")
print(myRestaurant.name)
print(myRestaurant.cusine)
```

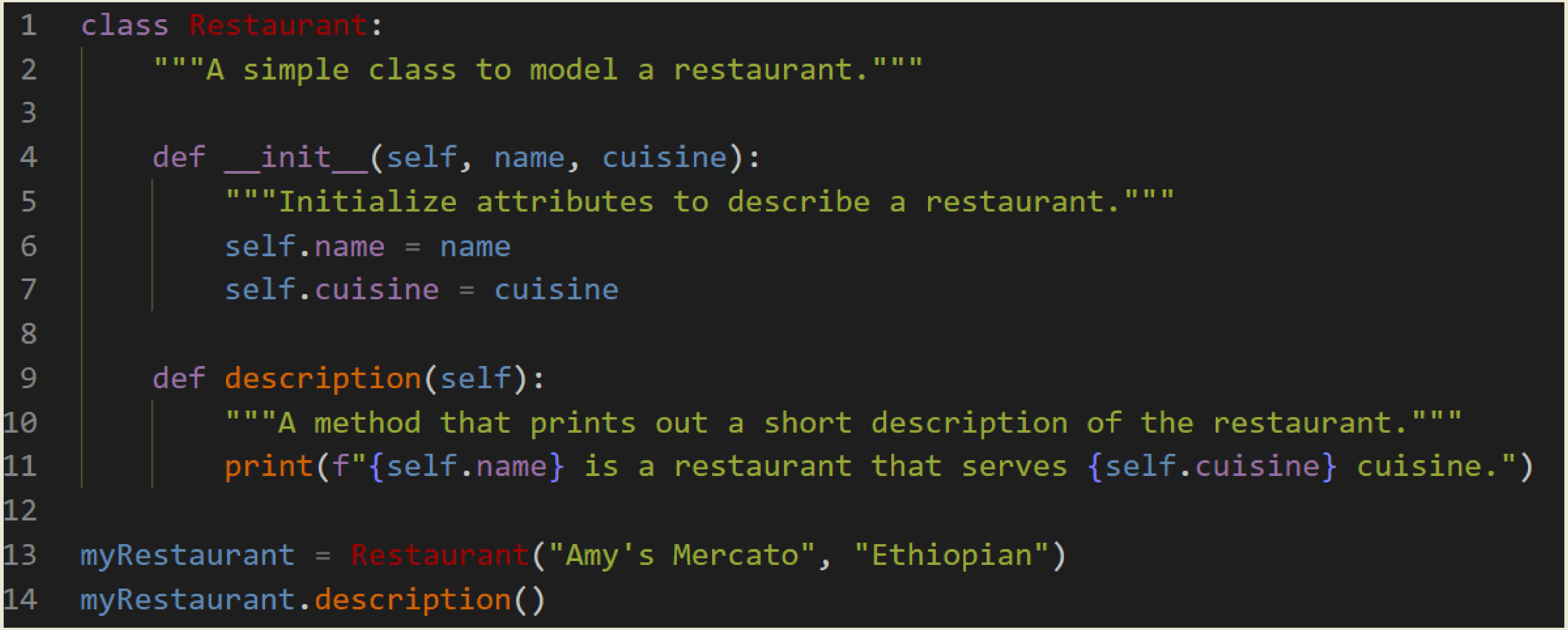
OUTPUT

```
Amy's Mercato
Ethiopian
```

LET'S SEE IF WE DO MORE

METHODS IN A CLASS

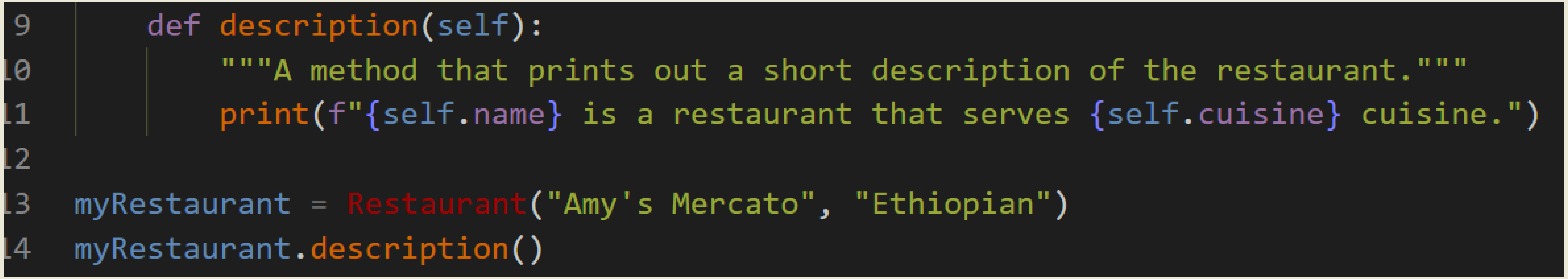
LET'S MAKE A METHOD TO PRINT OUT A SHORT DESCRIPTION OF THE RESTAURANT.



```python
class Restaurant:
    """A simple class to model a restaurant."""

    def __init__(self, name, cuisine):
        """Initialize attributes to describe a restaurant."""
        self.name = name
        self.cuisine = cuisine

    def description(self):
        """A method that prints out a short description of the restaurant."""
        print(f"{self.name} is a restaurant that serves {self.cuisine} cuisine.")

myRestaurant = Restaurant("Amy's Mercato", "Ethiopian")
myRestaurant.description()
```

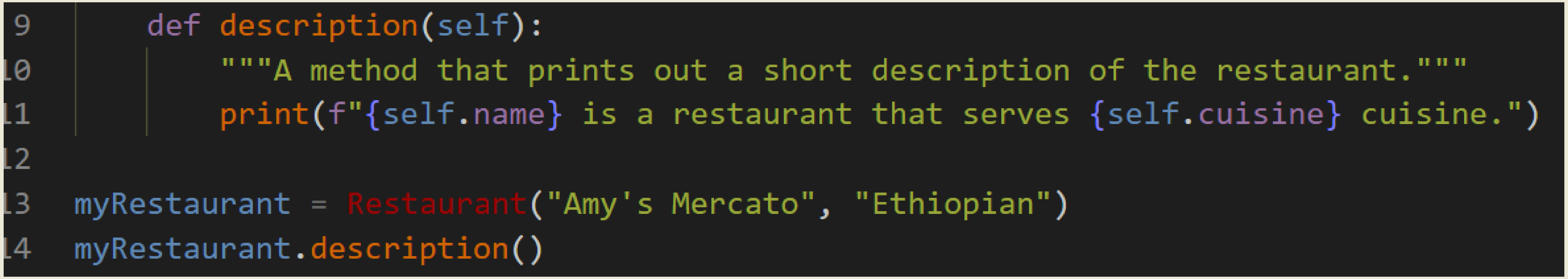
LET'S GET A CLOSER LOOK AT THE DESCRIPTION METHOD

first argument of a class is always self



```
    def description(self):
        """A method that prints out a short description of the restaurant."""
        print(f"{self.name} is a restaurant that serves {self.cuisine} cuisine.")

myRestaurant = Restaurant("Amy's Mercato", "Ethiopian")
myRestaurant.description()
```

refer to a method of a class

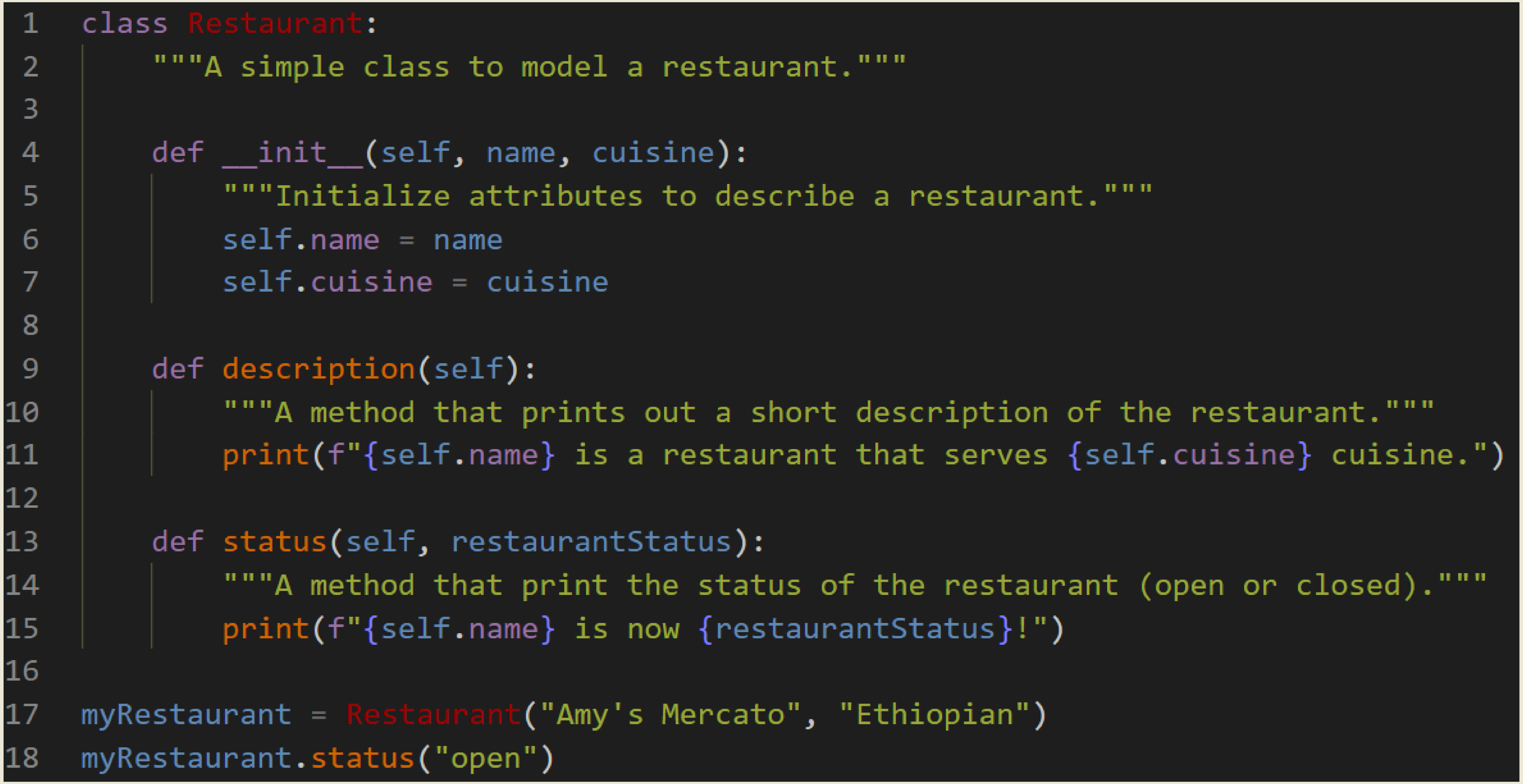


```python
    def description(self):
        """A method that prints out a short description of the restaurant."""
        print(f"{self.name} is a restaurant that serves {self.cuisine} cuisine.")

myRestaurant = Restaurant("Amy's Mercato", "Ethiopian")
myRestaurant.description()
```

OUTPUT

Amy's Mercato is a restaurant that serves Ethiopian cuisine.

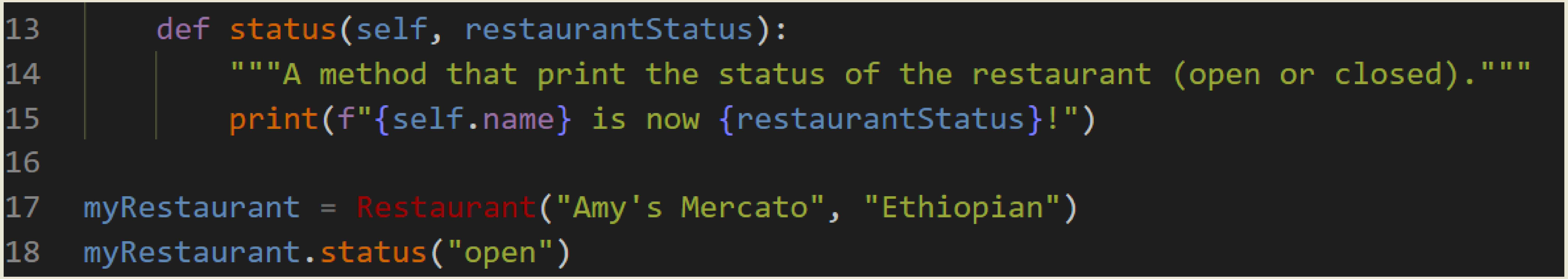
LET'S MAKE A METHOD TO PRINT OUT A MESSAGE THAT SAYS THAT OUR RESTAURANT IS OPEN.



```python
class Restaurant:
    """A simple class to model a restaurant."""

    def __init__(self, name, cuisine):
        """Initialize attributes to describe a restaurant."""
        self.name = name
        self.cuisine = cuisine

    def description(self):
        """A method that prints out a short description of the restaurant."""
        print(f"{self.name} is a restaurant that serves {self.cuisine} cuisine.")

    def status(self, restaurantStatus):
        """A method that print the status of the restaurant (open or closed)."""
        print(f"{self.name} is now {restaurantStatus}!")

myRestaurant = Restaurant("Amy's Mercato", "Ethiopian")
myRestaurant.status("open")
```

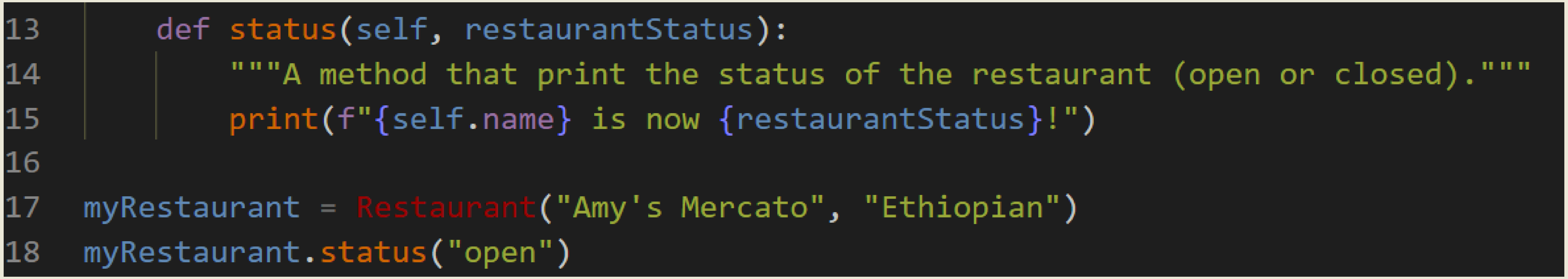
LET'S GET A CLOSER LOOK AT THE STATUS METHOD

other arguments added after a comma



```
    def status(self, restaurantStatus):
        """A method that print the status of the restaurant (open or closed)."""
        print(f"{self.name} is now {restaurantStatus}!")

myRestaurant = Restaurant("Amy's Mercato", "Ethiopian")
myRestaurant.status("open")
```

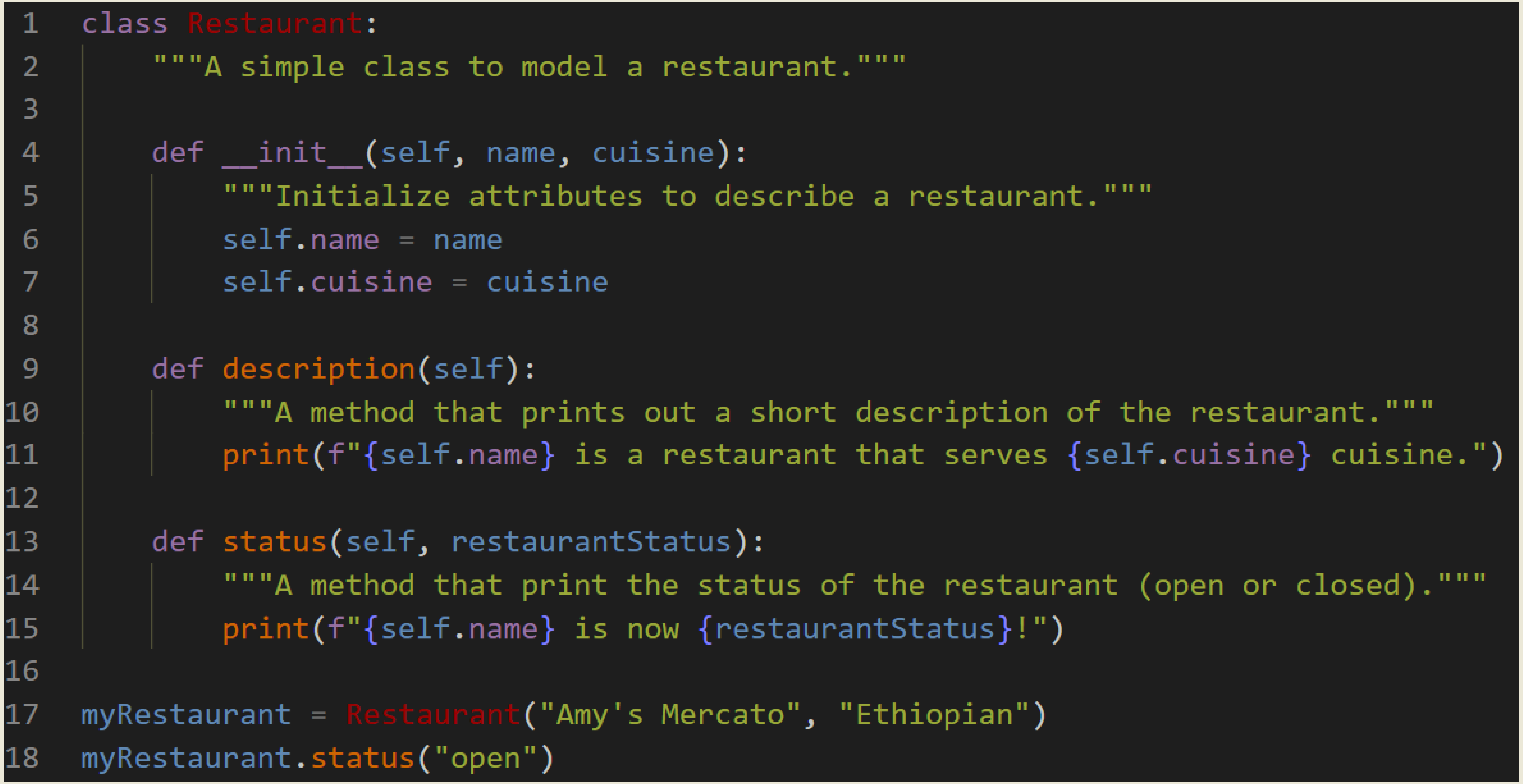
must give an argument when calling the method



```python
    def status(self, restaurantStatus):
        """A method that print the status of the restaurant (open or closed)."""
        print(f"{self.name} is now {restaurantStatus}!")

myRestaurant = Restaurant("Amy's Mercato", "Ethiopian")
myRestaurant.status("open")
```

OUTPUT

```
Amy's Mercato is now open!
```



```python
class Restaurant:
    """A simple class to model a restaurant."""

    def __init__(self, name, cuisine):
        """Initialize attributes to describe a restaurant."""
        self.name = name
        self.cuisine = cuisine

    def description(self):
        """A method that prints out a short description of the restaurant."""
        print(f"{self.name} is a restaurant that serves {self.cuisine} cuisine.")

    def status(self, restaurantStatus):
        """A method that print the status of the restaurant (open or closed)."""
        print(f"{self.name} is now {restaurantStatus}!")

myRestaurant = Restaurant("Amy's Mercato", "Ethiopian")
myRestaurant.status("open")
```

01
why use classes and not functions

02
define and use a simple class

03
Store and use data

04
get our class to do some things

ON YOUR OWN:

1. Make a class called User.
2. Make an __init__ method to store: first_name, last_name, and two more pieces of data that are typically stored in a user profile
3. Make a method called describe_user() that prints a summary of the user's information
4. Make another method called login_status() that receives a login status as an argument. Then print out a statement that describes the login status of the user.

BONUS: Add an attribute in the __init__ method called login_attempts. Then write a method called increment_login_attempts() that increments the value of login_attempts by 1. Write another method called reset_login_attempts() that resets the value of login_attempts to 0. Use increment_login_attempts() several times. Then print the value of login_attempts to make sure it was incremented properly. Then call reset_login_attempts(). Print login_attempts againto make sure it was reset to 0.

These problems were taken from *Python Crash Course 2nd Edition*