Python Intermediate

OOP - Methods, Getters and Setters

{codenation}®

Learning Objectives

- ✓ To add methods to our classes
- ✓ To use setters to give our objects new properties
- ✓ To use getters to retrieve information about our objects

Activity recap

How did you get on with the activity?

Was it annoying to have to print out the same sentence again and again for each superhero?

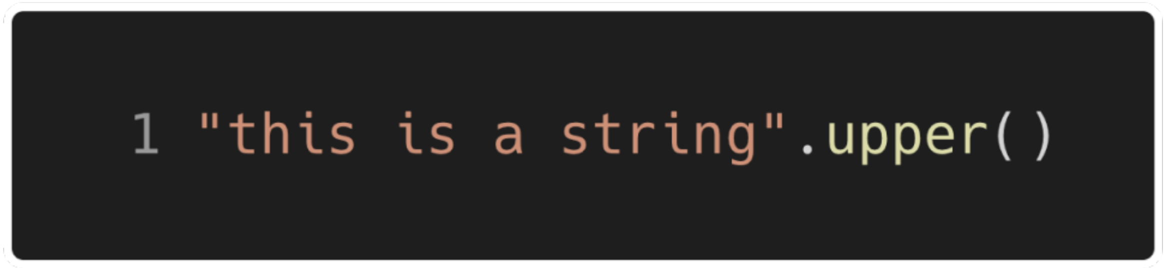
Methods

Methods

We said before that objects are collections of properties and methods.

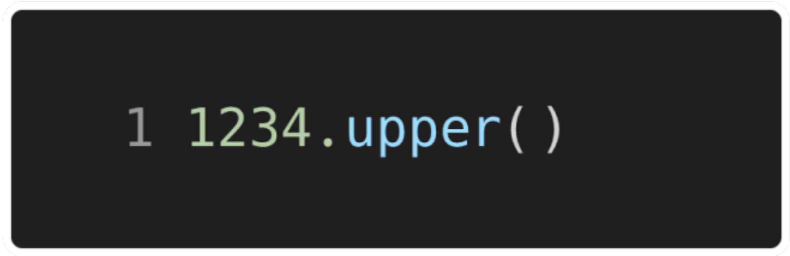
So far – we've only looked at properties.

Methods



```python
"this is a string".upper()
```

✓

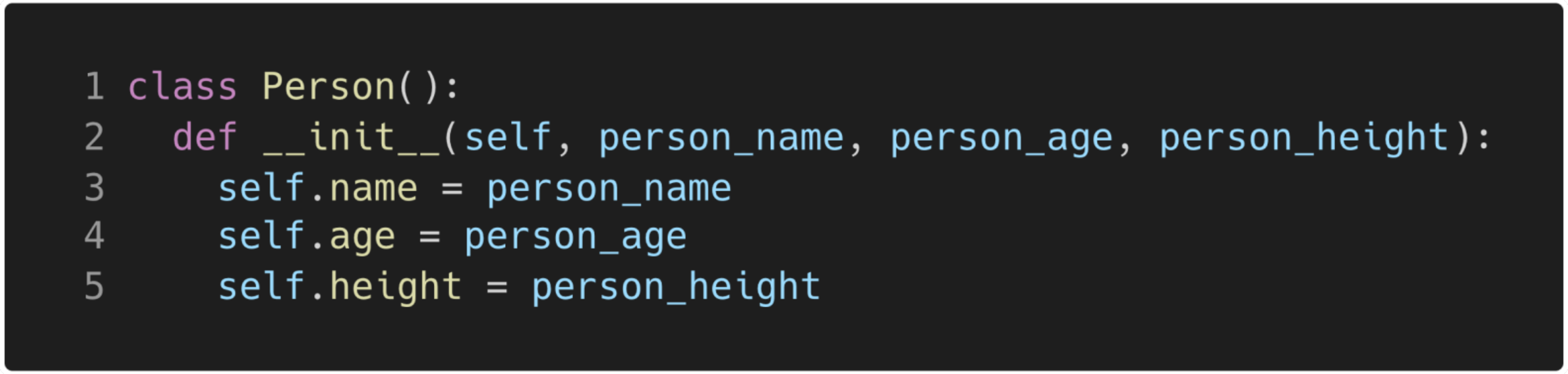


```python
1234.upper()
```

✗

Methods are functions attached to an object.
Let's look back at your person class..

Methods

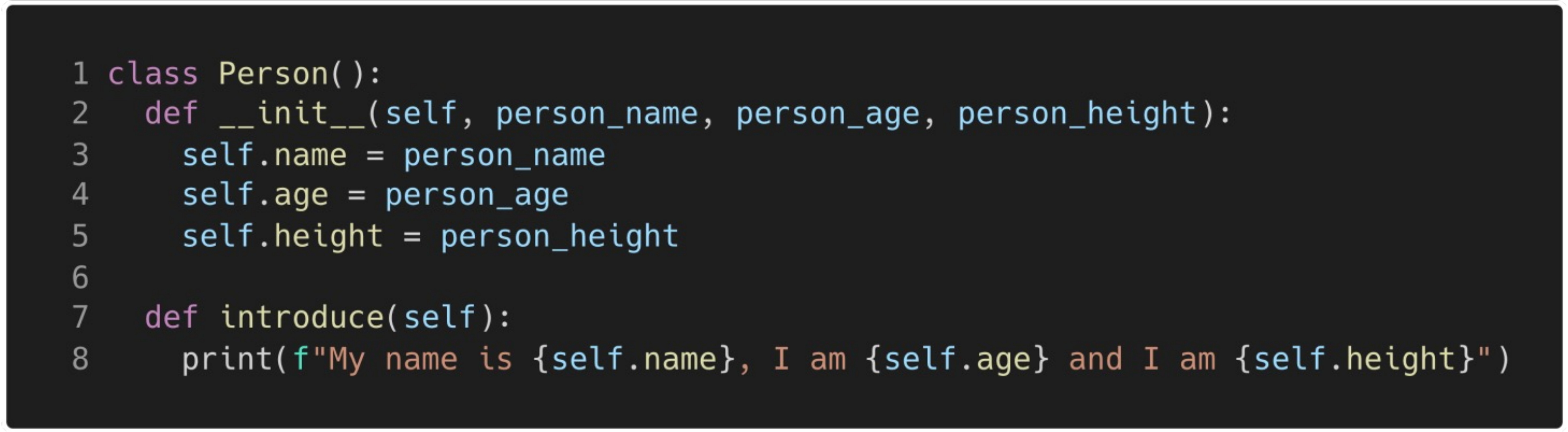


```python
class Person():
  def __init__(self, person_name, person_age, person_height):
    self.name = person_name
    self.age = person_age
    self.height = person_height
```

I used these properties to write out a sentence about the person object I made – but every person will want to introduce themselves.

How can I make this easier?

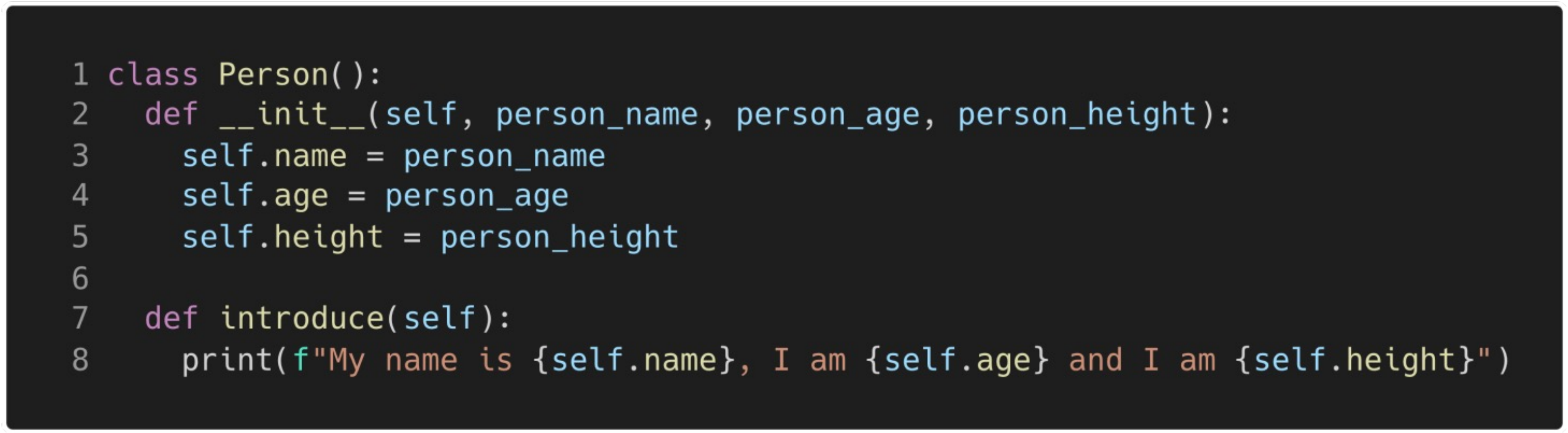
Methods



```python
class Person():
  def __init__(self, person_name, person_age, person_height):
    self.name = person_name
    self.age = person_age
    self.height = person_height

  def introduce(self):
    print(f"My name is {self.name}, I am {self.age} and I am {self.height}")
```

We've seen function syntax before – but this time, rather than being declared in the global scope, it's only with the Person class.

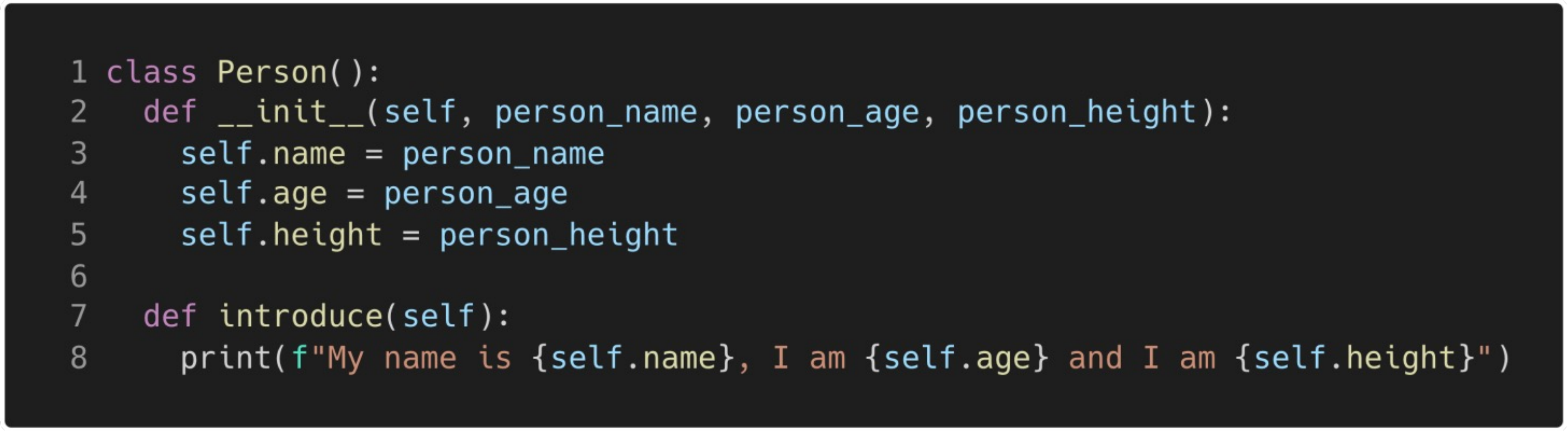
Methods



```python
class Person():
  def __init__(self, person_name, person_age, person_height):
    self.name = person_name
    self.age = person_age
    self.height = person_height

  def introduce(self):
    print(f"My name is {self.name}, I am {self.age} and I am {self.height}")
```

This function can only work when referenced with a Person object, and it becomes a method.

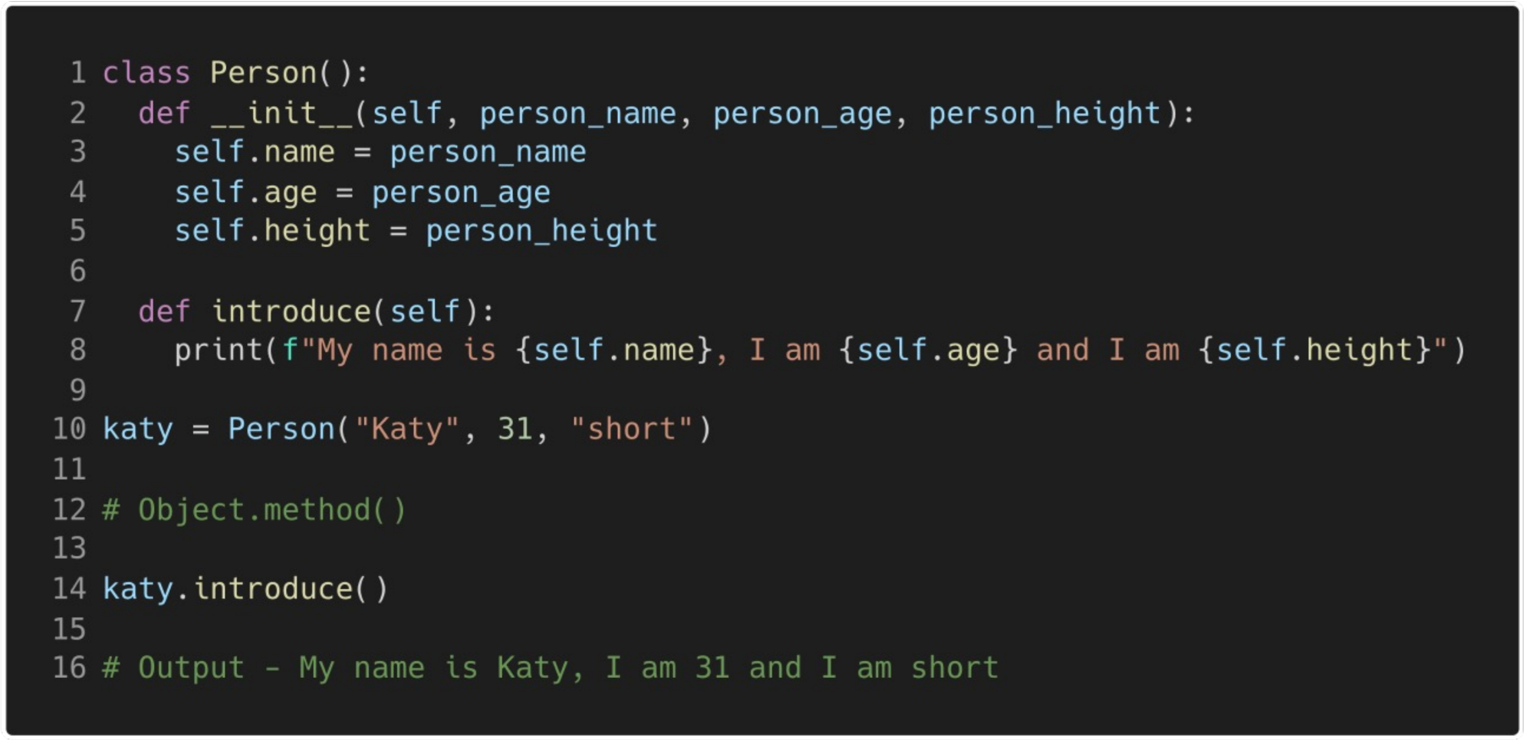
Methods



```python
class Person():
  def __init__(self, person_name, person_age, person_height):
    self.name = person_name
    self.age = person_age
    self.height = person_height

  def introduce(self):
    print(f"My name is {self.name}, I am {self.age} and I am {self.height}")
```

It takes one parameter – self.
The instance of the object it is working on.

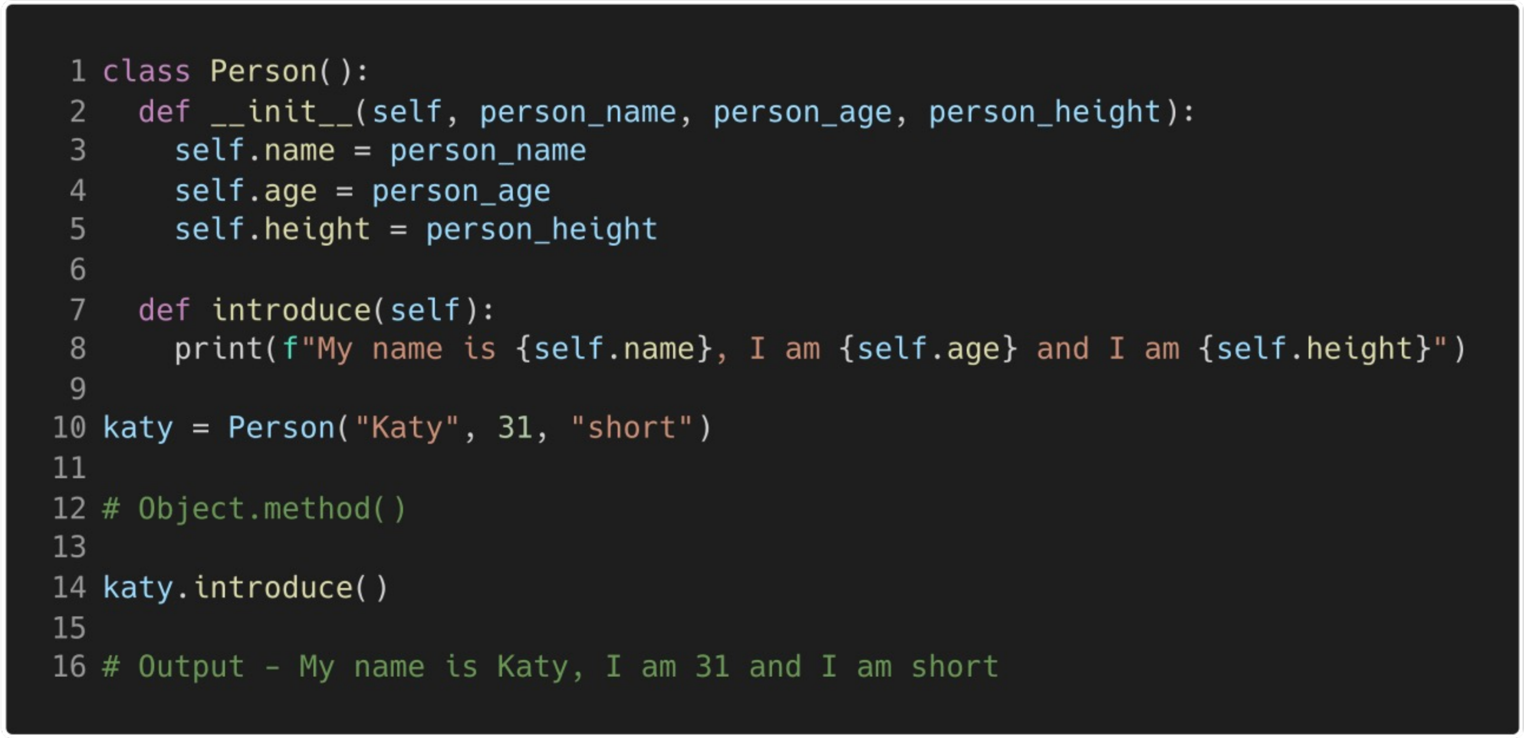
Methods



```python
class Person():
  def __init__(self, person_name, person_age, person_height):
    self.name = person_name
    self.age = person_age
    self.height = person_height

  def introduce(self):
    print(f"My name is {self.name}, I am {self.age} and I am {self.height}")

katy = Person("Katy", 31, "short")

# Object.method()

katy.introduce()

# Output - My name is Katy, I am 31 and I am short
```

I access this method using dot notation.

Methods



```python
class Person():
  def __init__(self, person_name, person_age, person_height):
    self.name = person_name
    self.age = person_age
    self.height = person_height

  def introduce(self):
    print(f"My name is {self.name}, I am {self.age} and I am {self.height}")

katy = Person("Katy", 31, "short")

# Object.method()

katy.introduce()

# Output - My name is Katy, I am 31 and I am short
```

Methods are a structured and reusable way to allow objects of a specific class to perform a task.

Activity 1

Make a similar method for your heroes to introduce themselves.

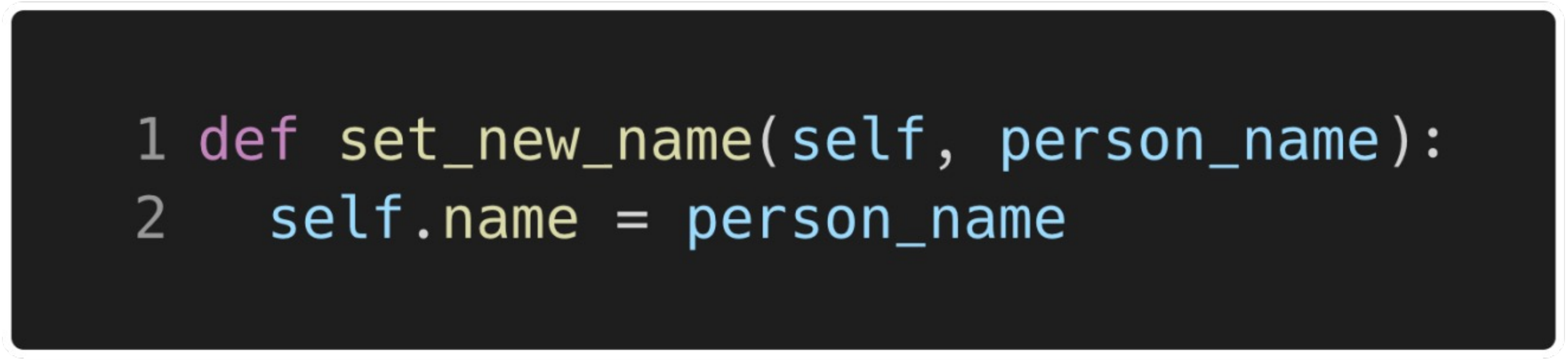
Getters and Setters

Getters and setters

We might want to give our objects more properties after instantiation or change an existing property.

We can use a kind of function called a setter to set new properties on our object.

Getters and setters

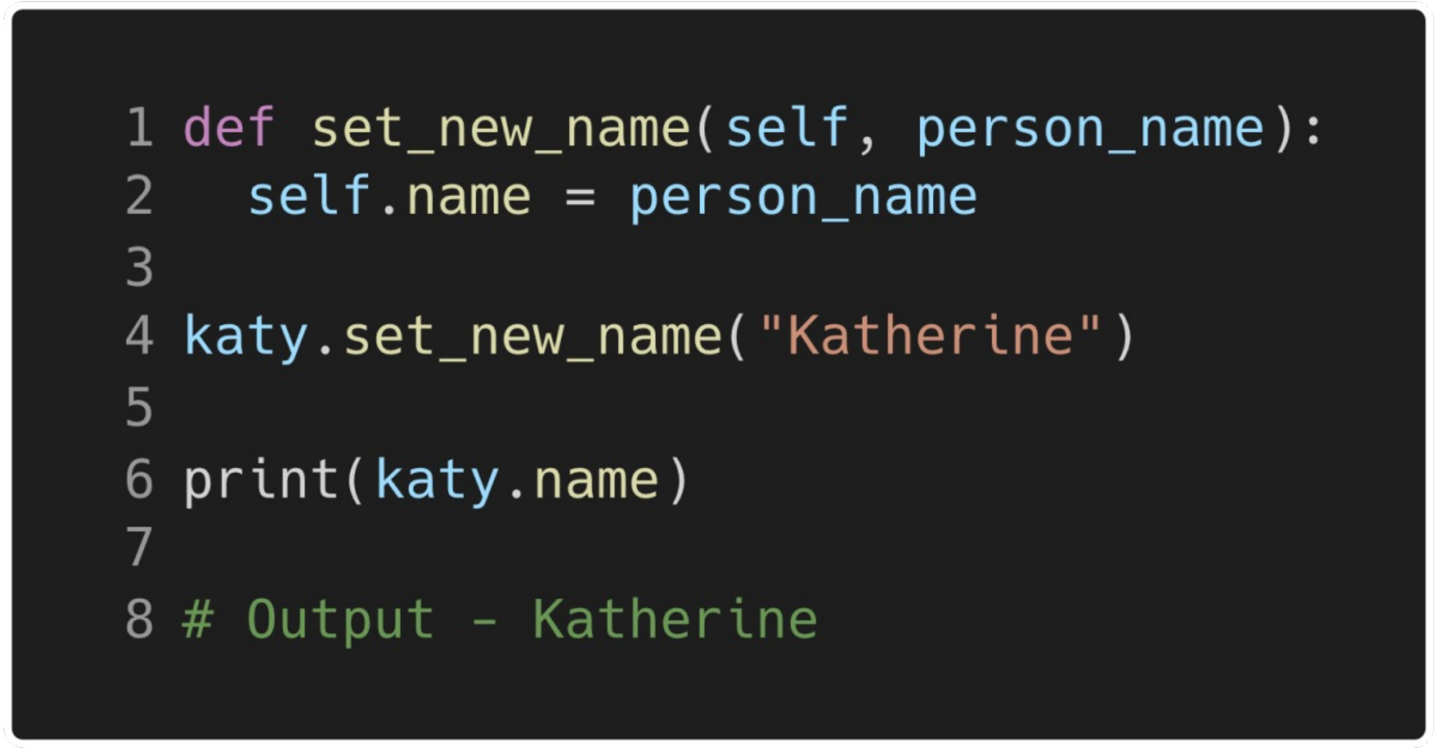


```python
def set_new_name(self, person_name):
  self.name = person_name
```

This setter takes the self parameter so it knows which object to change, and the person_name parameter.

It re-assigns the name property to the new value.

Getters and setters



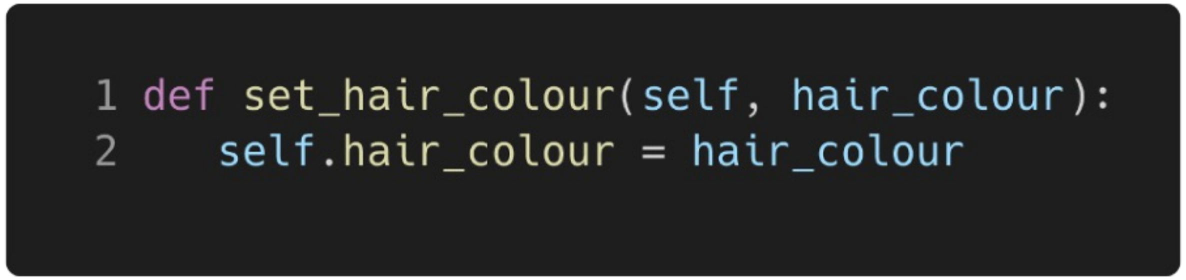
```python
def set_new_name(self, person_name):
  self.name = person_name

katy.set_new_name("Katherine")

print(katy.name)

# Output - Katherine
```

As always, we access it with dot notation.

Getters and setters

We could also set a new property on our object.

We can all this method to give our object a hair colour if we want to.



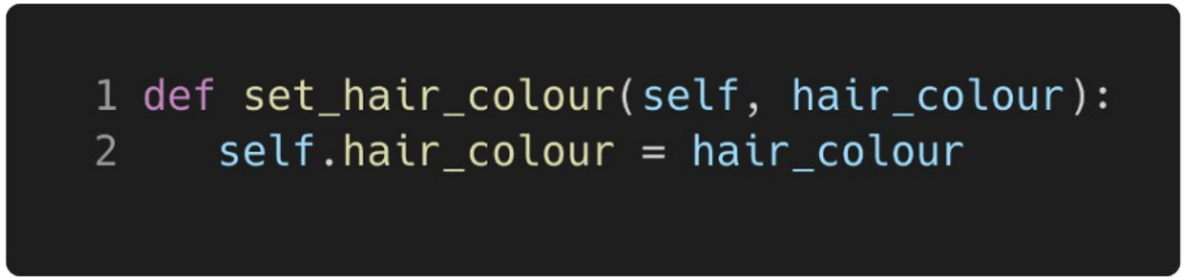
```python
def set_hair_colour(self, hair_colour):
    self.hair_colour = hair_colour
```

Getters and setters

Our person objects need a name, age, and height to be created.

The hair_colour setter makes hair_colour optional.

We can add it if we need to.

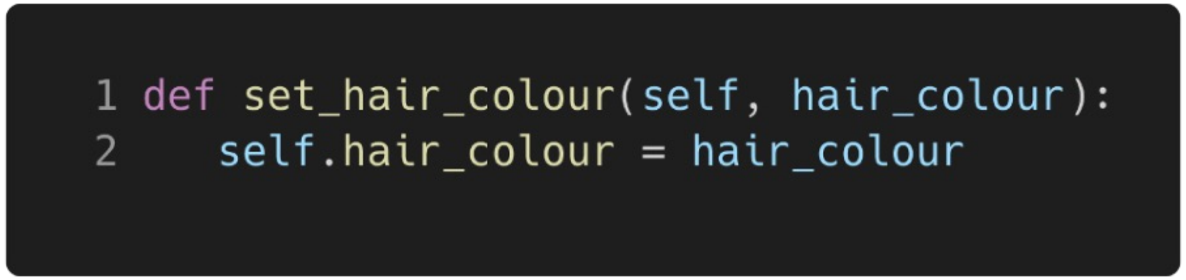


```python
def set_hair_colour(self, hair_colour):
    self.hair_colour = hair_colour
```

Getters and setters

The O in SOLID stands for the Open-Closed Principle.

Our classes should be open to expansion but closed for modification.



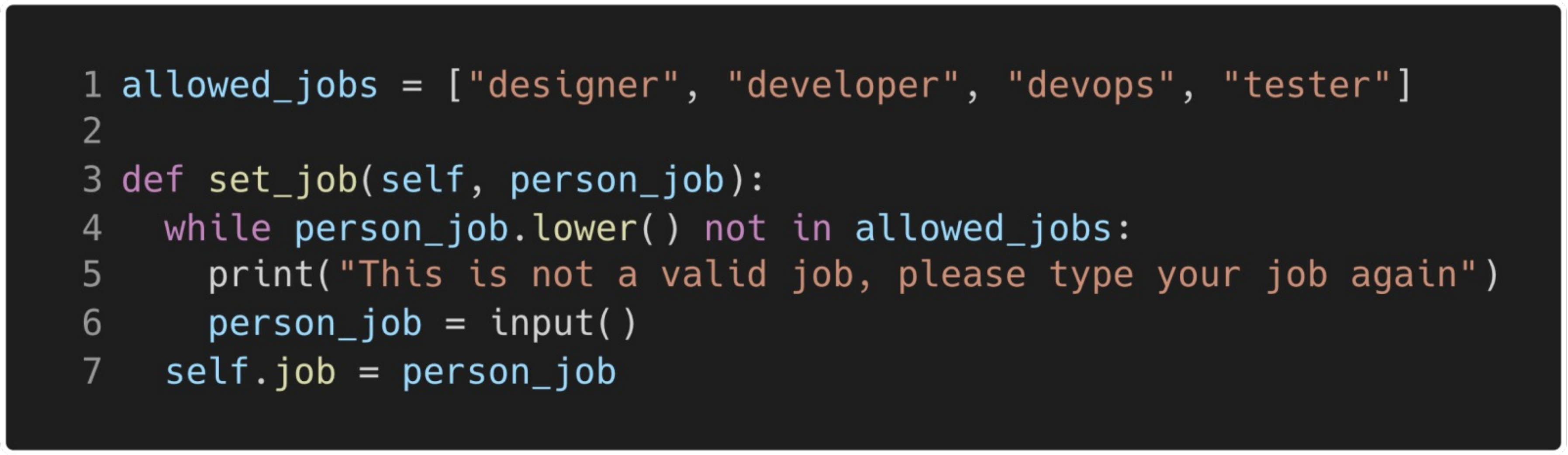
```python
def set_hair_colour(self, hair_colour):
    self.hair_colour = hair_colour
```

Getters and setters

Lots of people could be using your object – it would be dangerous to **expose** them to the code which controls your object – but they can **safely** make changes to it using setters, without allowing them access to the data.

Setters can also be used to **validate** the data.

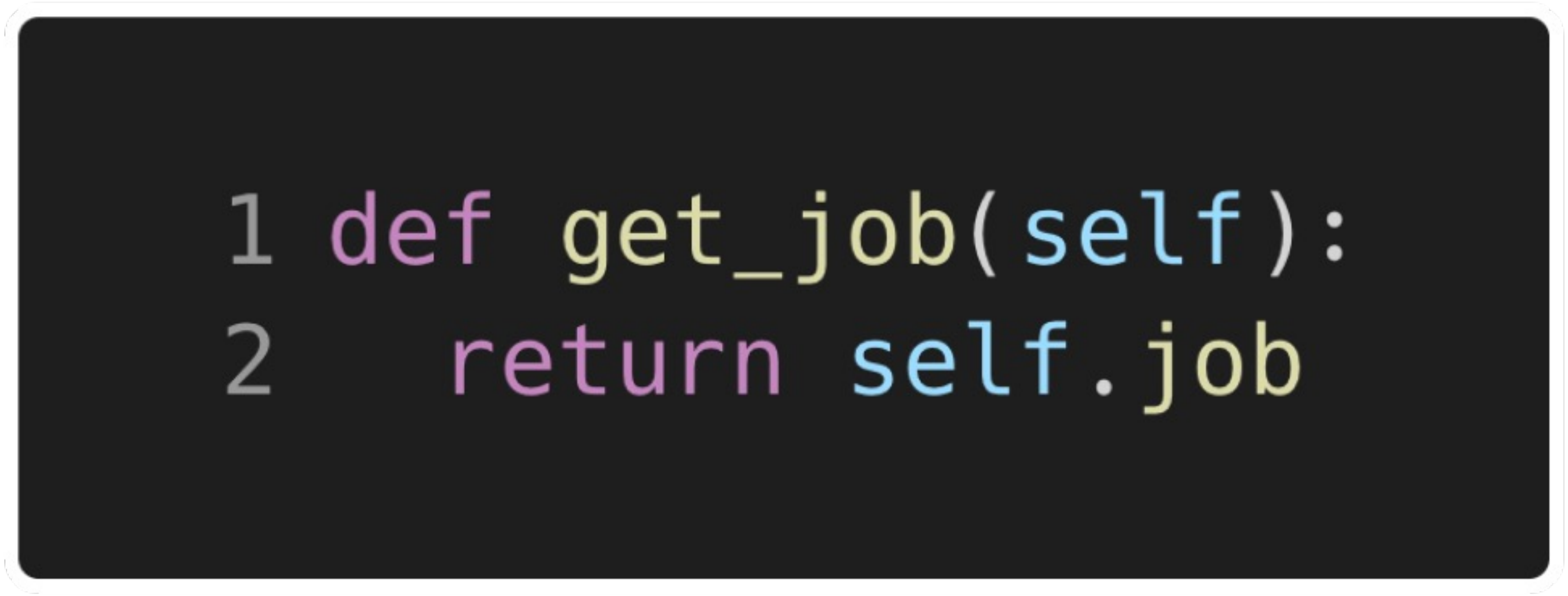
Getters and setters



```python
allowed_jobs = ["designer", "developer", "devops", "tester"]

def set_job(self, person_job):
  while person_job.lower() not in allowed_jobs:
    print("This is not a valid job, please type your job again")
    person_job = input()
  self.job = person_job
```

This setter function ensures the job typed in is listed in the allowed jobs.
If it isn’t, it will keep prompting the user for a new response.
This ensures the new property fulfils a requirement.

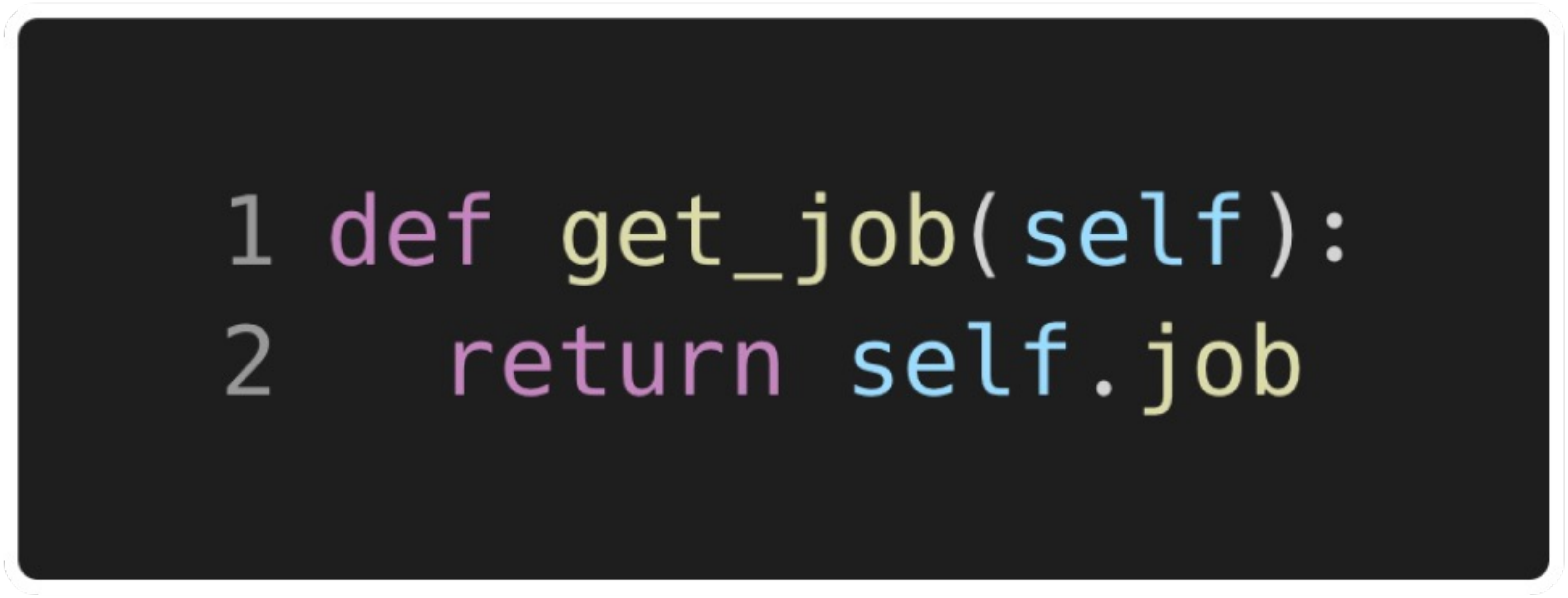
Getters and setters



```python
def get_job(self):
  return self.job
```

Getters retrieve information from an object.

Getters and setters



```python
def get_job(self):
  return self.job
```

**Getters should always return the information.
This allows us to work with it as we see fit.
Returning the value is more flexible than simply printing it.**

Getters and setters

Notice how we named our getters and setters?

They included the word get and set!

This isn’t necessary but is best practise.

Learning Objectives

- ✓ To add methods to our classes
- ✓ To use setters to give our objects new properties
- ✓ To use getters to retrieve information about our objects

{codenation}®

Challenge 1

Make a method for your superheroes called **transform**, which prints out a message saying your hero has transformed from their real-life persona to their hero persona.

For example, “Peter Parker has transformed into Spiderman!”.

Challenge 2

Write a **setter** to give your superhero a secret lair and set a lair for all four of your heroes using the setter.