Intro to Coding with Python– Classes Pt 3

Dr. Ab Mosca (they/them)

Slides based off slides courtesy of Jordan Crouser (https://jcrouser.github.io/)

Plan for Today

- Recap classes
- Object-Oriented Programming
    - Child classes
    - Inheritance

RECAP: **`class`** definitions ("blueprints")

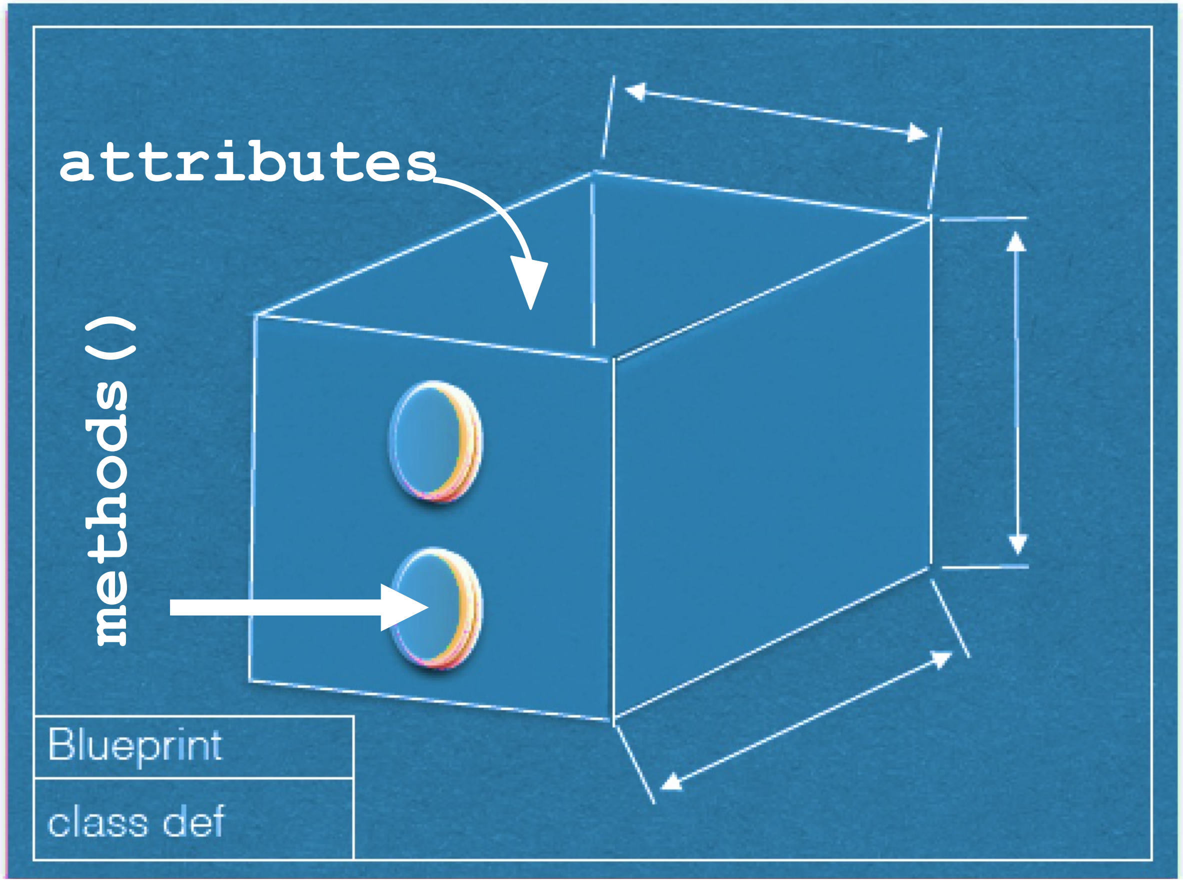
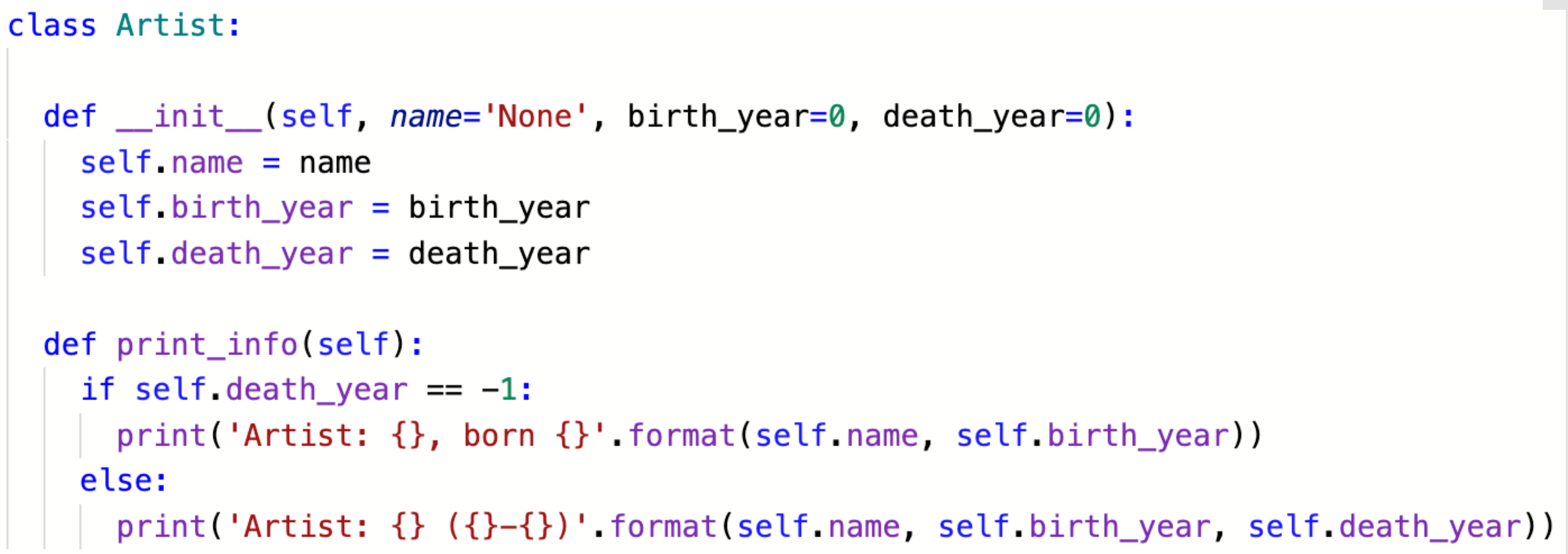


Image courtesy Dominique Thiebaut, Smith College

10 Minute activity: `Artist class`

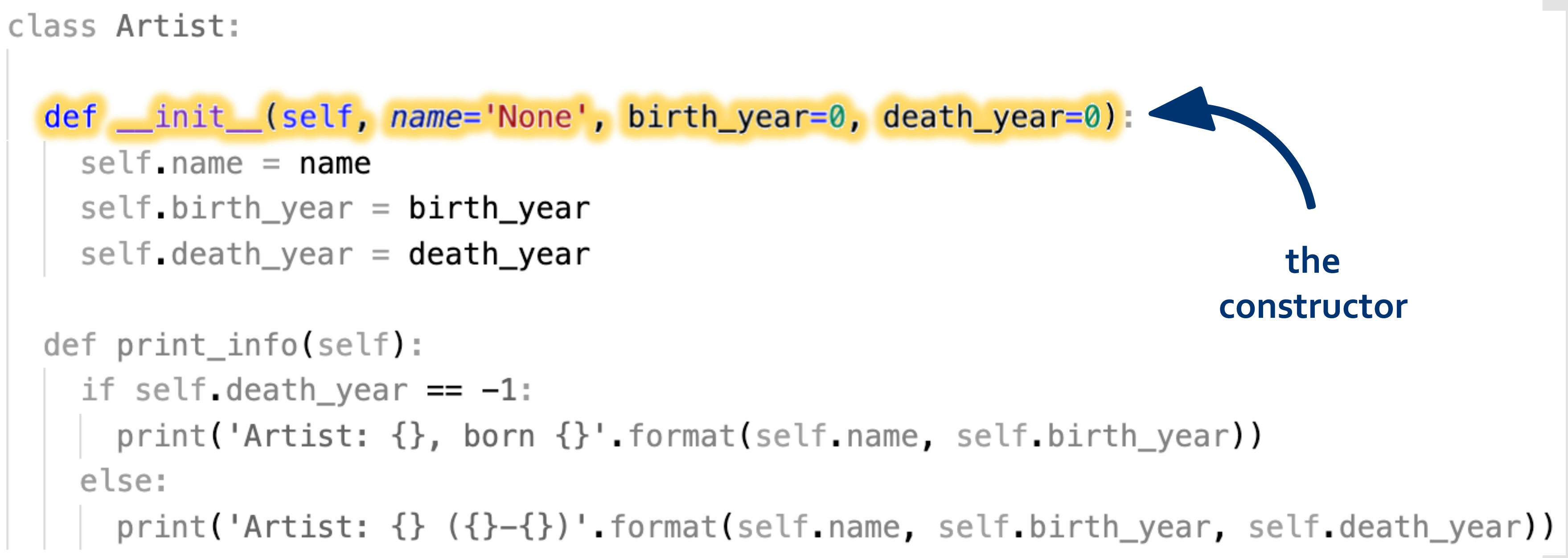
- Define an `Artist` class
- An Artist should have the attributes:
    - name
    - birth year
    - death year
- An Artist should have the method:
    - print_info that prints:
        - “Artist: <name>, born: < birth year>” if the artist is alive and
        - “Artist: <name>, < birth year> - <death year>” if the artist is dead

Coding the **Artist class**



```
class Artist:

  def __init__(self, name='None', birth_year=0, death_year=0):
    self.name = name
    self.birth_year = birth_year
    self.death_year = death_year

  def print_info(self):
    if self.death_year == -1:
      print('Artist: {}, born {}'.format(self.name, self.birth_year))
    else:
      print('Artist: {} ({}-{})'.format(self.name, self.birth_year, self.death_year))
```

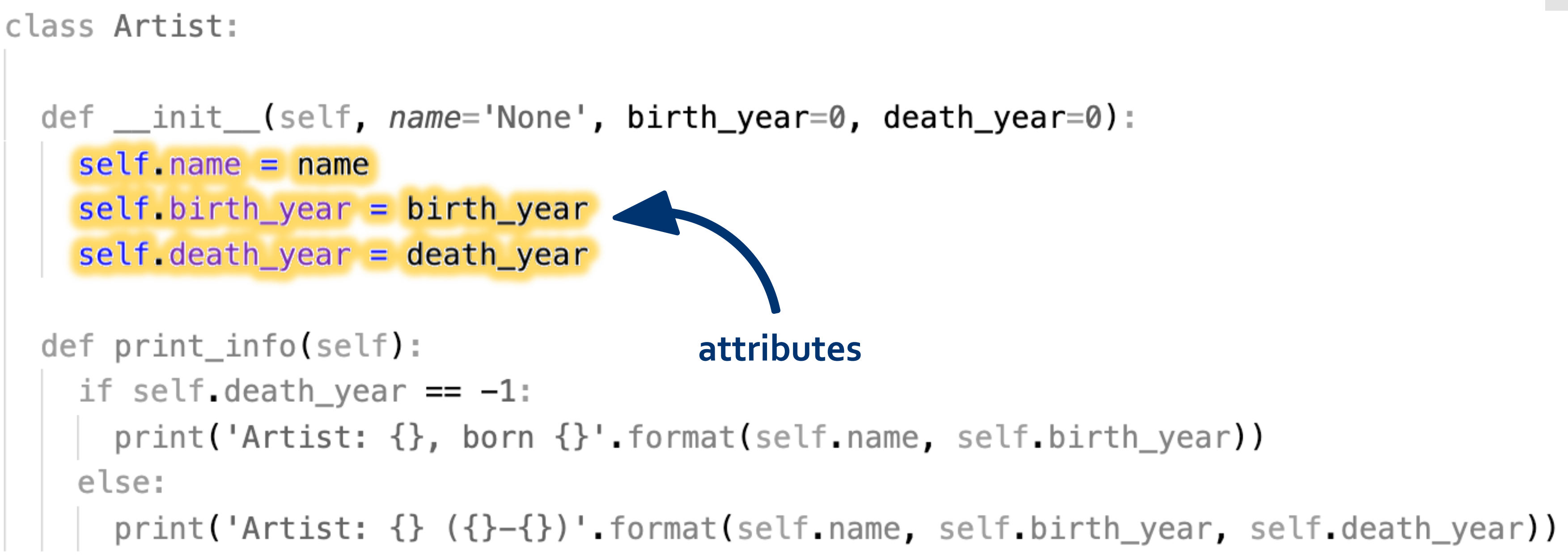
Coding the **Artist class**



```
class Artist:

  def __init__(self, name='None', birth_year=0, death_year=0):
    self.name = name
    self.birth_year = birth_year
    self.death_year = death_year

  def print_info(self):
    if self.death_year == -1:
      print('Artist: {}, born {}'.format(self.name, self.birth_year))
    else:
      print('Artist: {} ({}-{})'.format(self.name, self.birth_year, self.death_year))
```

the constructor

Coding the **Artist class**

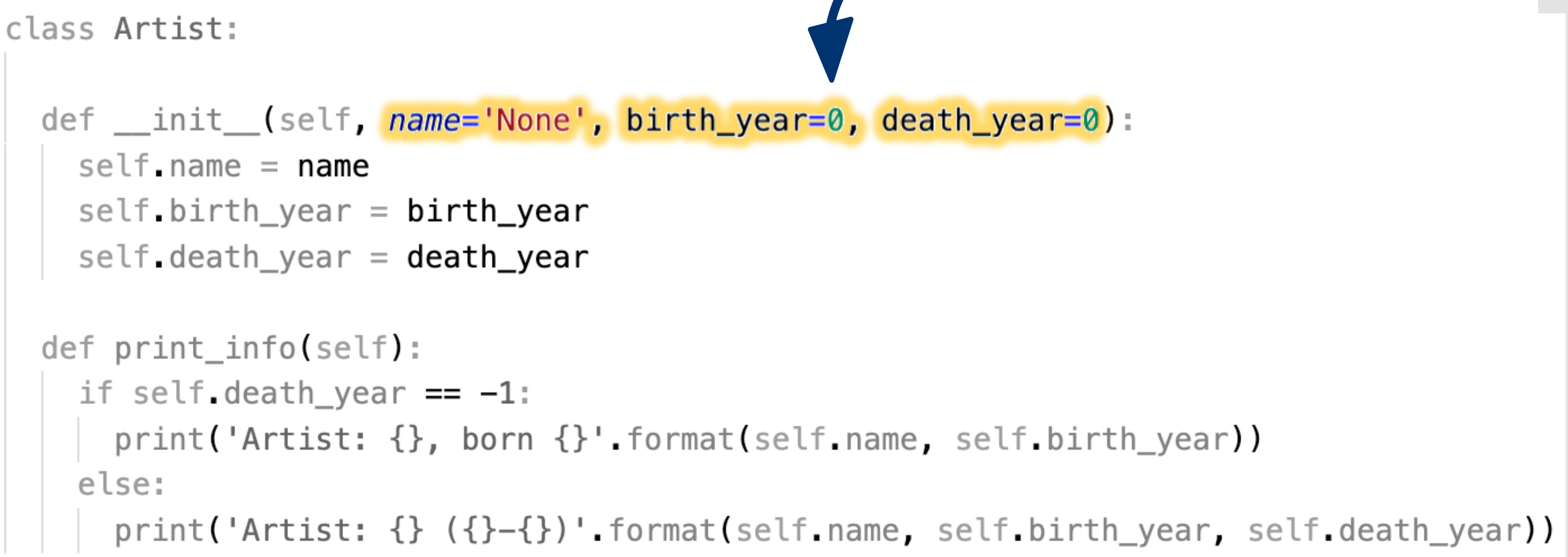


```
class Artist:

  def __init__(self, name='None', birth_year=0, death_year=0):
    self.name = name
    self.birth_year = birth_year
    self.death_year = death_year

  def print_info(self):
    if self.death_year == -1:
      print('Artist: {}, born {}'.format(self.name, self.birth_year))
    else:
      print('Artist: {} ({}-{})'.format(self.name, self.birth_year, self.death_year))
```

attributes

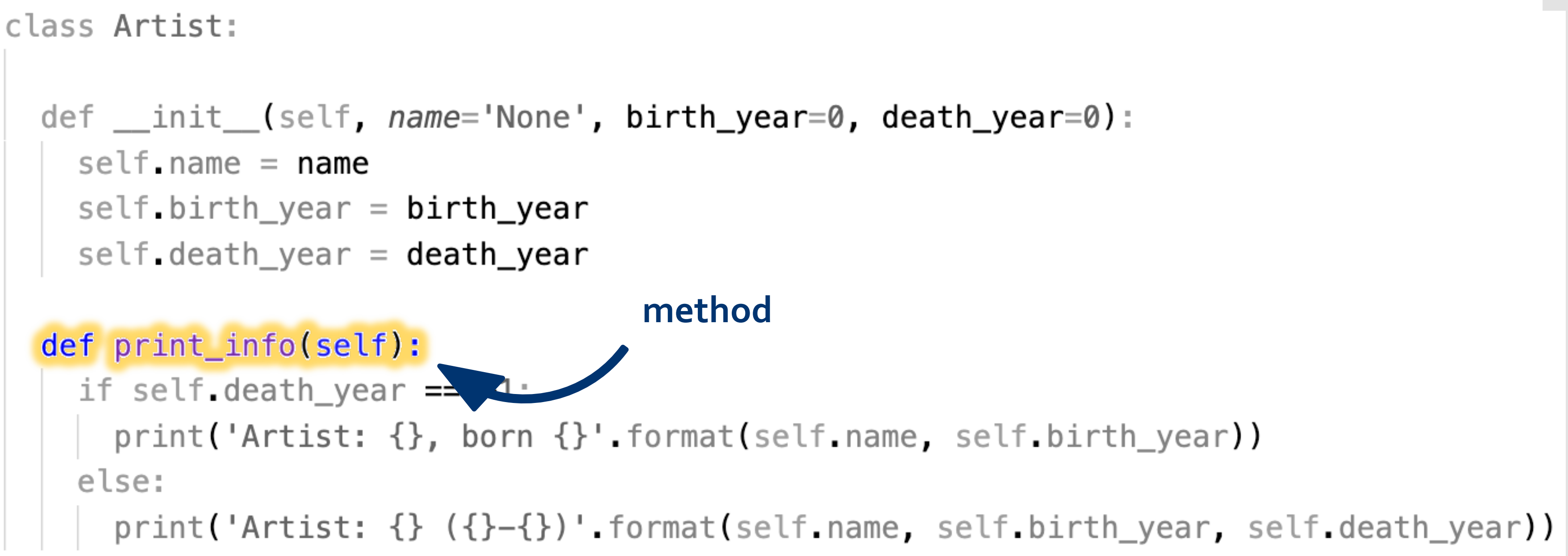
Coding the **Artist class**

default values



```
class Artist:

    def __init__(self, name='None', birth_year=0, death_year=0):
        self.name = name
        self.birth_year = birth_year
        self.death_year = death_year

    def print_info(self):
        if self.death_year == -1:
            print('Artist: {}, born {}'.format(self.name, self.birth_year))
        else:
            print('Artist: {} ({}-{})'.format(self.name, self.birth_year, self.death_year))
```

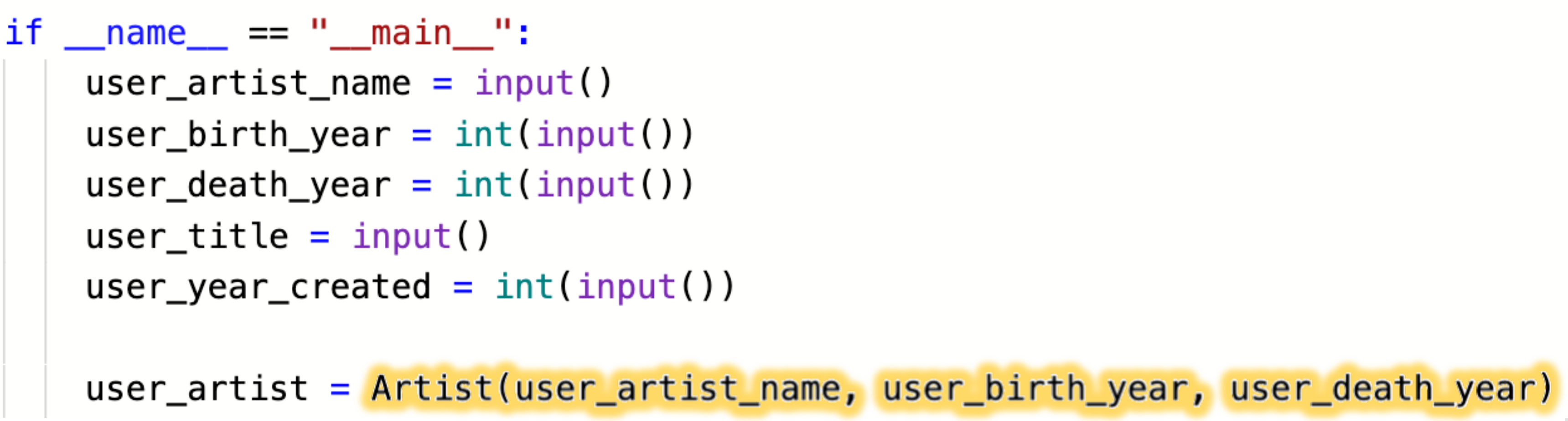
Coding the `Artist` class



```
class Artist:

    def __init__(self, name='None', birth_year=0, death_year=0):
        self.name = name
        self.birth_year = birth_year
        self.death_year = death_year

    def print_info(self):
        if self.death_year == -1:
            print('Artist: {}, born {}'.format(self.name, self.birth_year))
        else:
            print('Artist: {} ({}-{})'.format(self.name, self.birth_year, self.death_year))
```

method

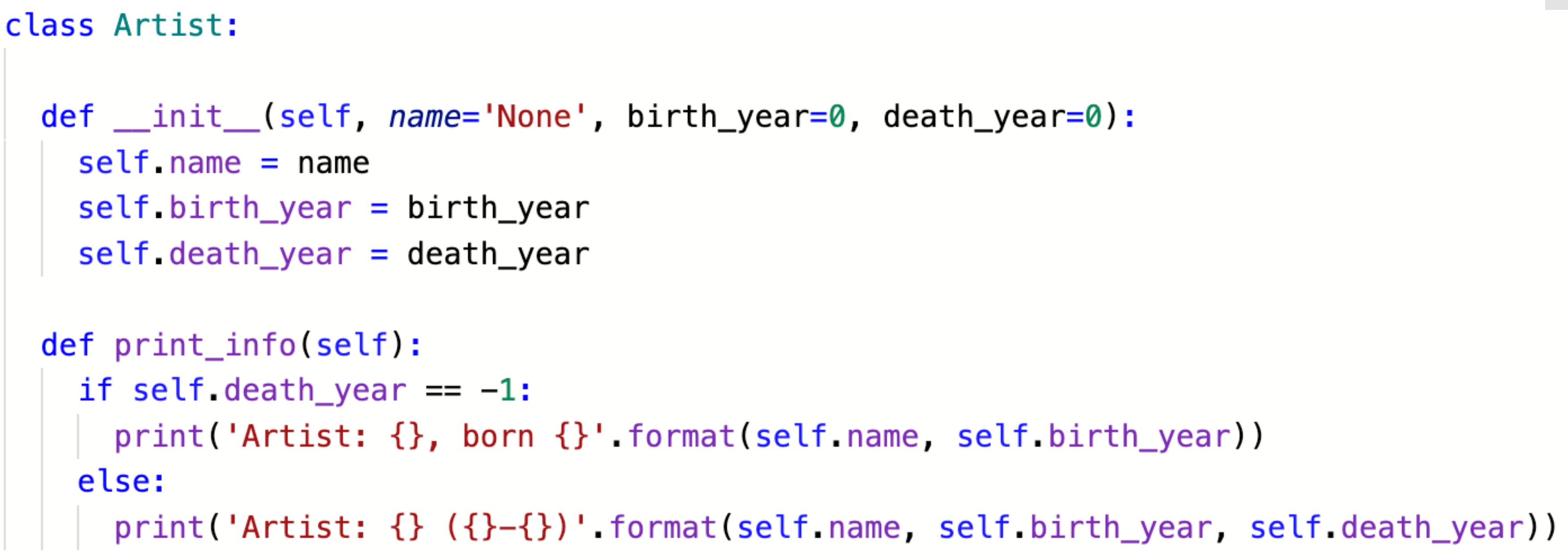
Creating an **Artist instance**



```python
if __name__ == "__main__":
    user_artist_name = input()
    user_birth_year = int(input())
    user_death_year = int(input())
    user_title = input()
    user_year_created = int(input())

    user_artist = Artist(user_artist_name, user_birth_year, user_death_year)
```

Lots of possible **Artists**

All from the **same blueprint**



```python
class Artist:

  def __init__(self, name='None', birth_year=0, death_year=0):
    self.name = name
    self.birth_year = birth_year
    self.death_year = death_year

  def print_info(self):
    if self.death_year == -1:
      print('Artist: {}, born {}'.format(self.name, self.birth_year))
    else:
      print('Artist: {} ({}-{})'.format(self.name, self.birth_year, self.death_year))
```

Inheritance

Motivation

Dog

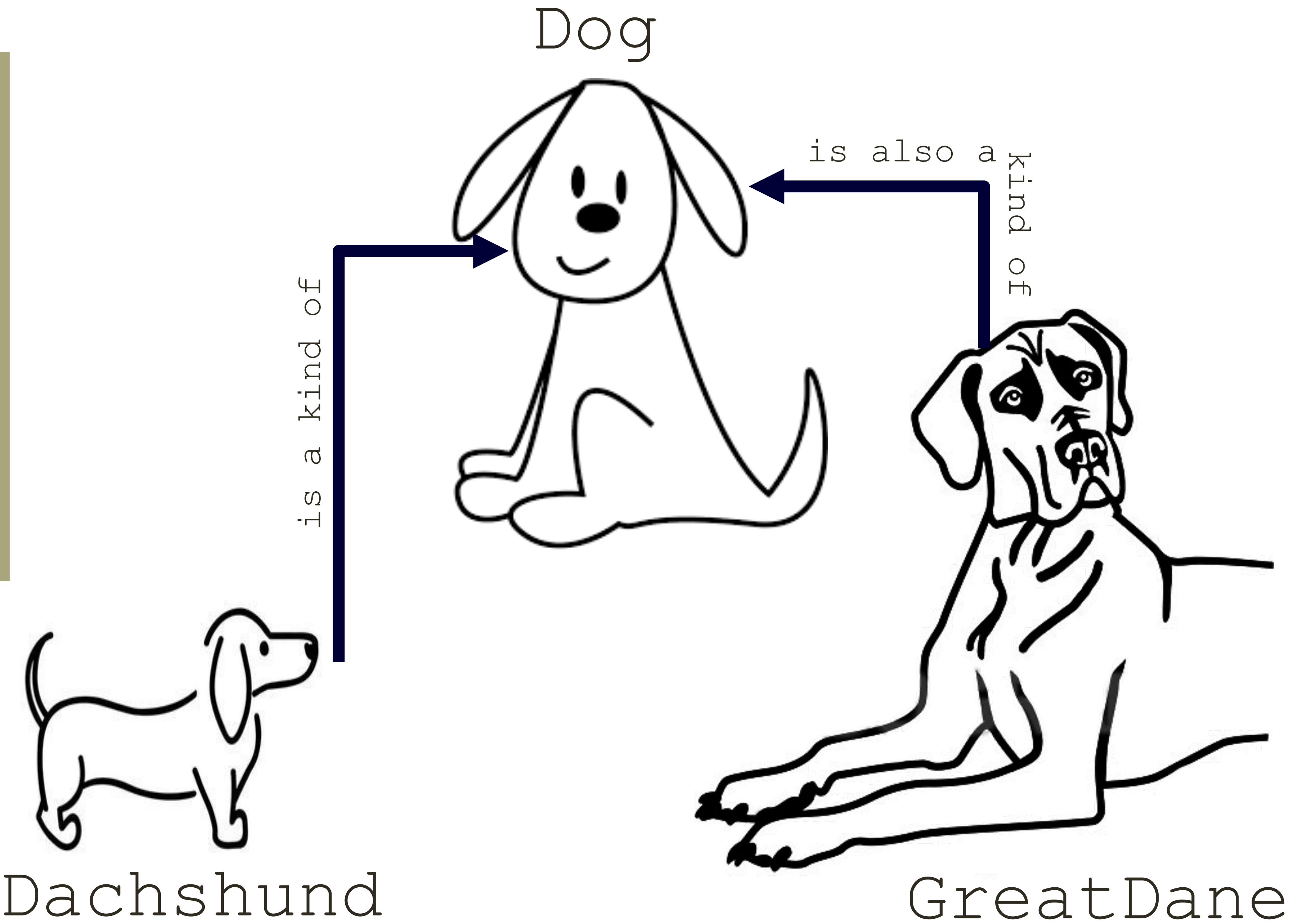
10 minute exercise: the `Dog` class

- Write a class called **Dog**, with a constructor that takes in the following parameters:

name (the dog's name)

age (the dog's age in years)

Motivation



Dog

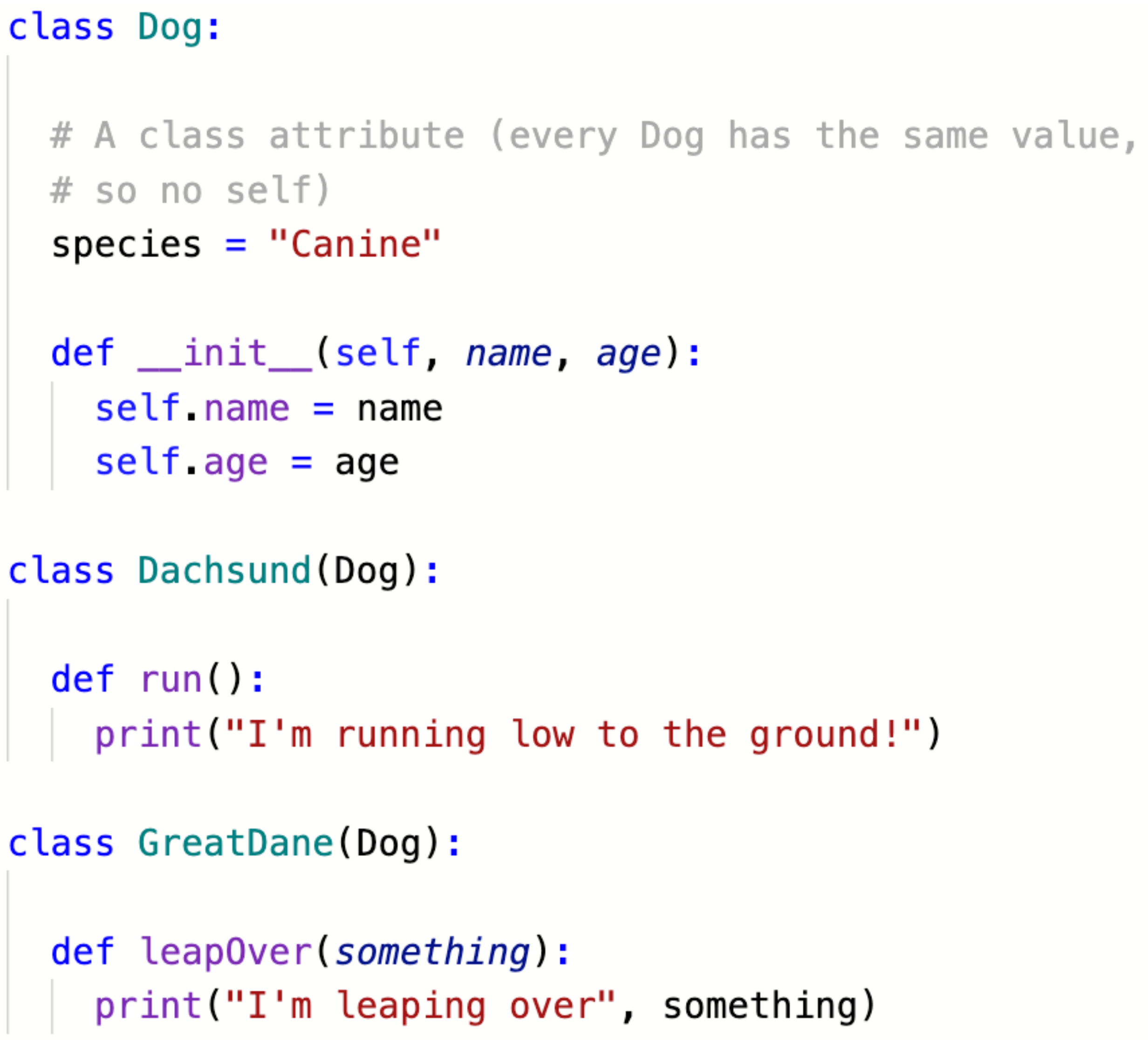
is a kind of

is also a kind of

Dachshund

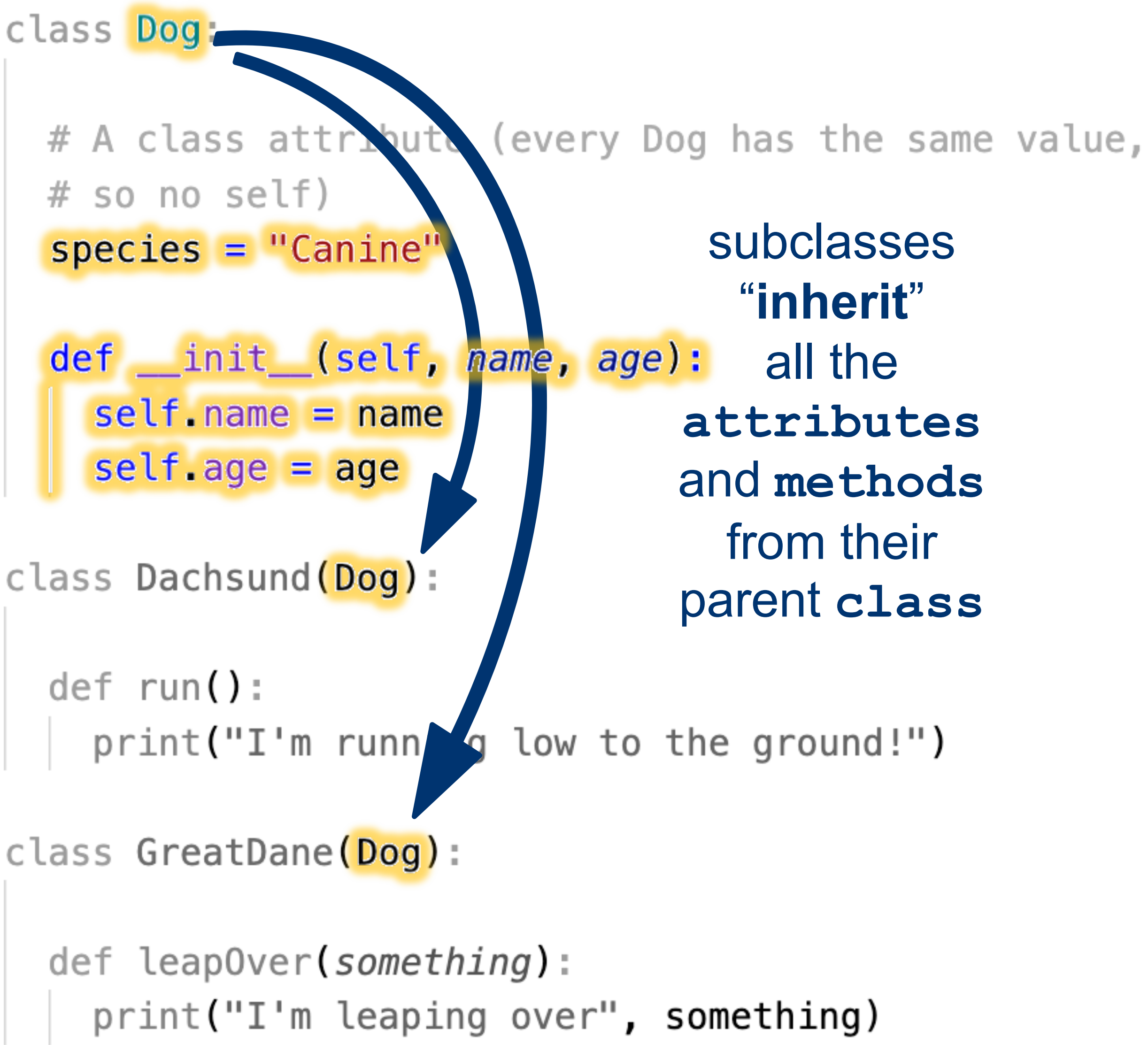
GreatDane

As (sub)classes



```
class Dog:

  # A class attribute (every Dog has the same value,
  # so no self)
  species = "Canine"

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

class Dachsund(Dog):

  def run():
    print("I'm running low to the ground!")

class GreatDane(Dog):

  def leapOver(something):
    print("I'm leaping over", something)
```

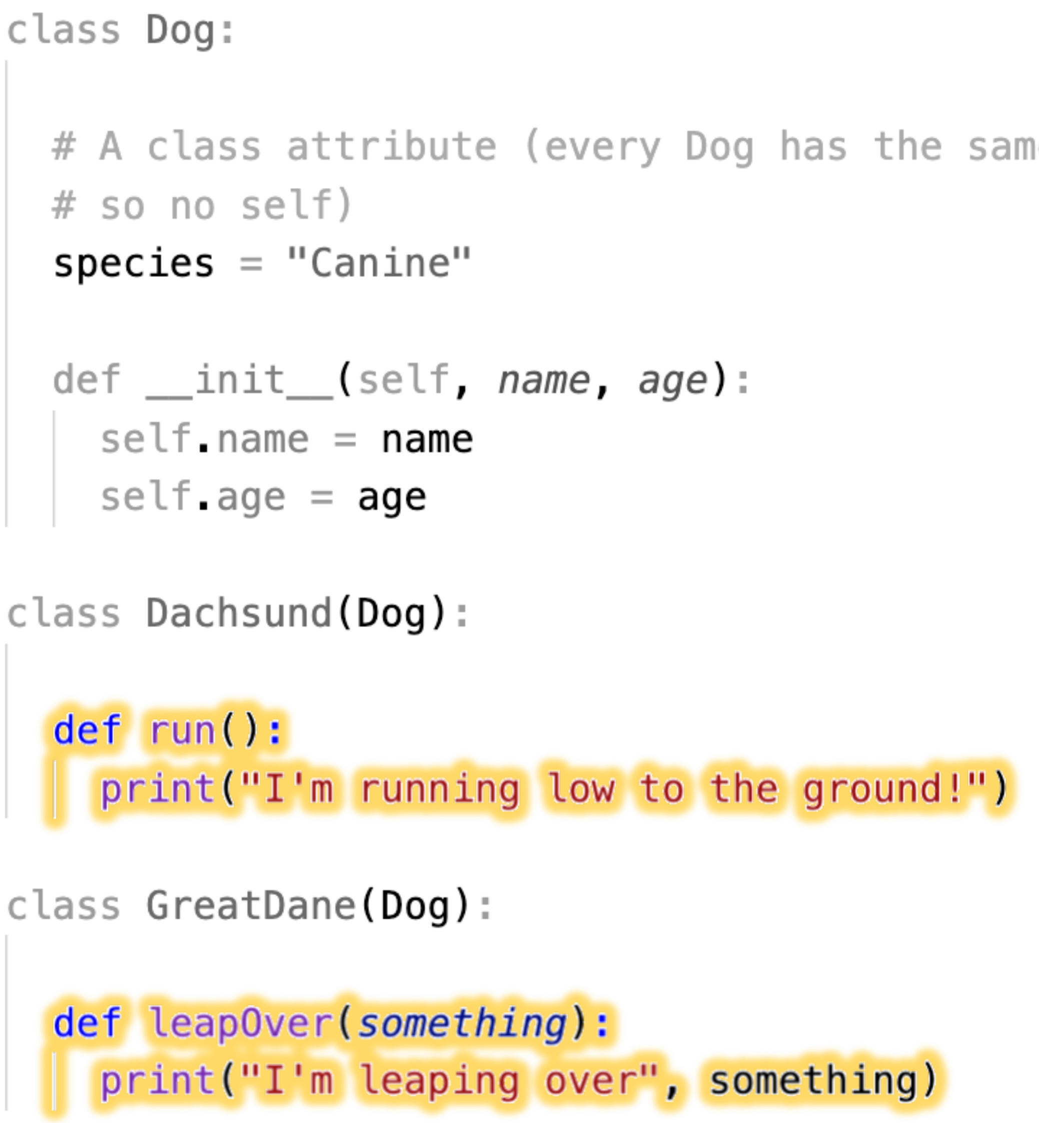
As (sub)classes



```
class Dog:

    # A class attribute (every Dog has the same value,
    # so no self)
    species = "Canine"

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

class Dachsund(Dog):

    def run():
        print("I'm running low to the ground!")

class GreatDane(Dog):

    def leapOver(something):
        print("I'm leaping over", something)
```

subclasses “**inherit**” all the **attributes** and **methods** from their parent **class**

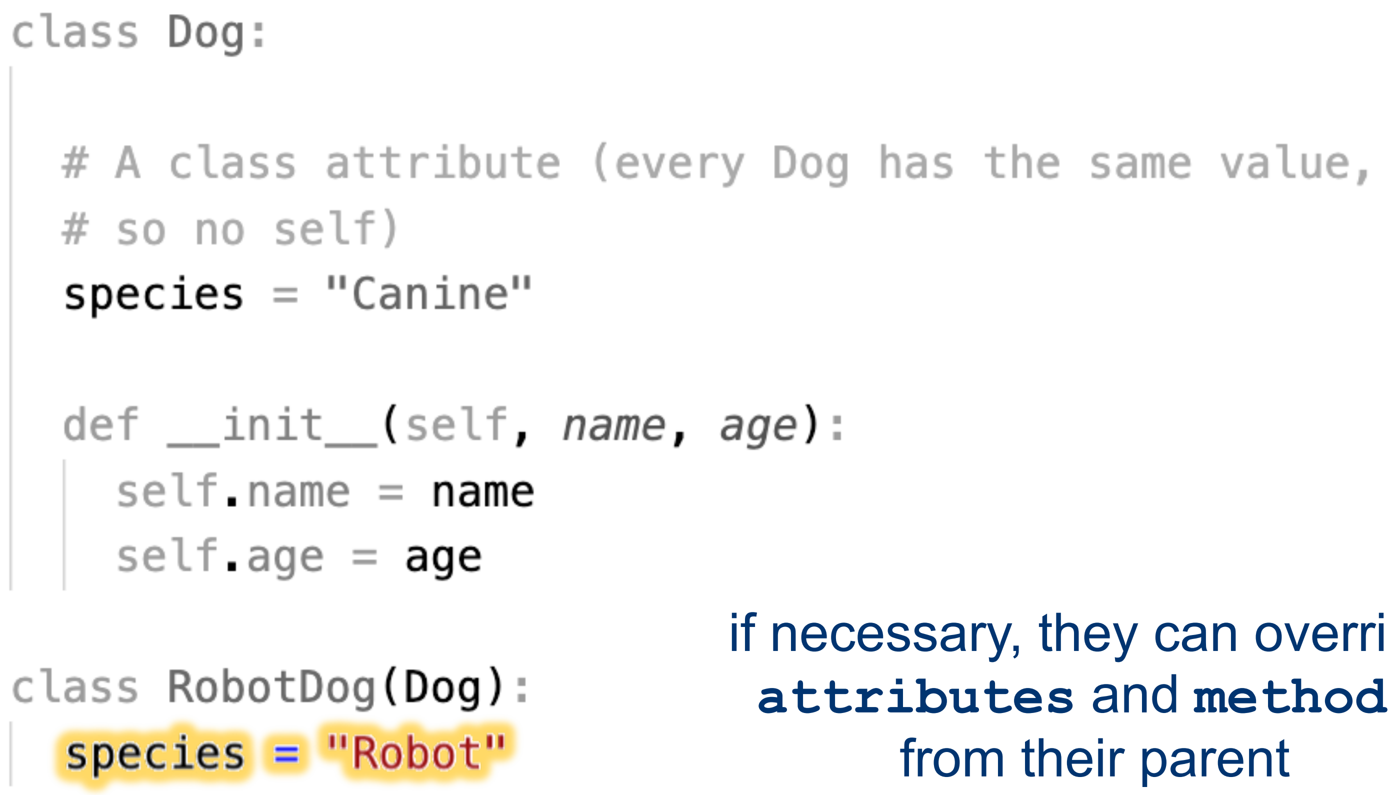
As (sub)**classes**



```
class Dog:

    # A class attribute (every Dog has the same value,
    # so no self)
    species = "Canine"

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

class Dachsund(Dog):

    def run():
        print("I'm running low to the ground!")

class GreatDane(Dog):

    def leapOver(something):
        print("I'm leaping over", something)
```

they can also have **their own attributes** and **methods** separate from their parent

As (sub)**classes**



```
class Dog:

    # A class attribute (every Dog has the same value,
    # so no self)
    species = "Canine"

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

class RobotDog(Dog):
    species = "Robot"
```

if necessary, they can override **attributes** and **methods** from their parent

Discussion

Why is this “inheritance” idea **useful**?

15 Minute Activity

Return to the `Playlist` class you wrote for your music library earlier.

Create a subclass, called `Radio` that fills a new `Playlist` with 10 random songs.

Create another subclass, called `ShuffleList` that has an additional method for randomly shuffling a `Playlist`.