Object-Oriented Programming: Objects, Classes & Methods

CSCI 120 Intro to Computing

Define a class

- A *class* is a blueprint – a definition of a data type.
  - specifies the attributes and methods of that type
- Classes provide a means of bundling data and functionality together.
  - Each class instance can have attributes attached to it for maintaining its state.
  - Each class instance can also have methods (defined by its class) for modifying its state.

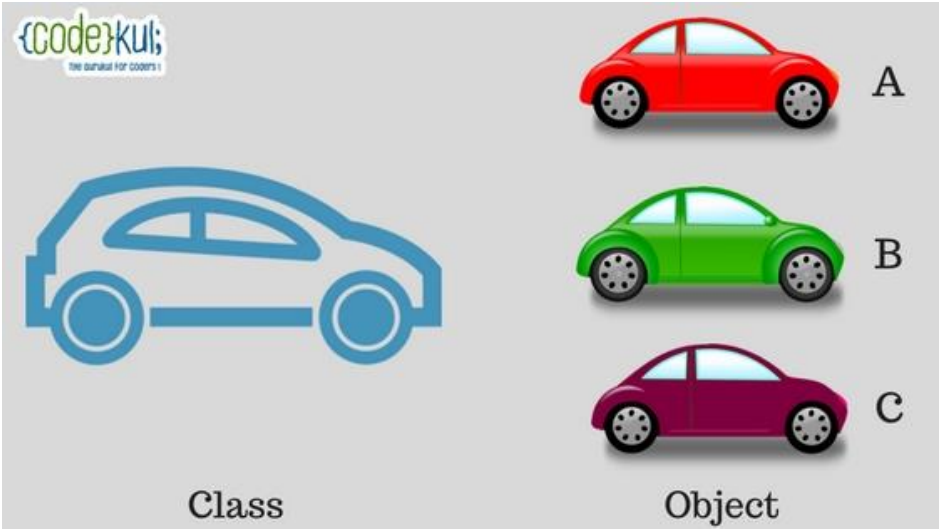
Examples:

Class: Coordinate (Has x and y)

Instance: (500, 300) (35, 48)

Class: Car (Has attributes of a color and mileage)

Instance: A specific car. Red with 79000 miles.



Attributes and Methods

- data attributes
  - think of attributes as states that make up the class
  - for example, a coordinate is made up of two numbers
- methods (procedural attributes)
  - think of methods as functions that only work with this class
  - methods can modify the states

- Example:
  - Class: Car (Has attribute of mileage)
  - Object (instance): A specific car. Red with 79000 miles.

Class name

Class constructor (optional)

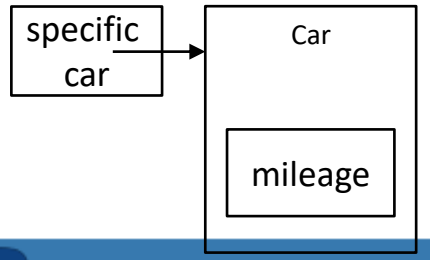
```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance
```

Define and initialize a "state"

Note:
When a class defines an __init__() method, class instantiation automatically invokes __init__() for the newly-created class instance.

Method to modify the state

"**self**" represents the instance of the class. By using the "self" keyword we can access the attributes and methods of the class in python.

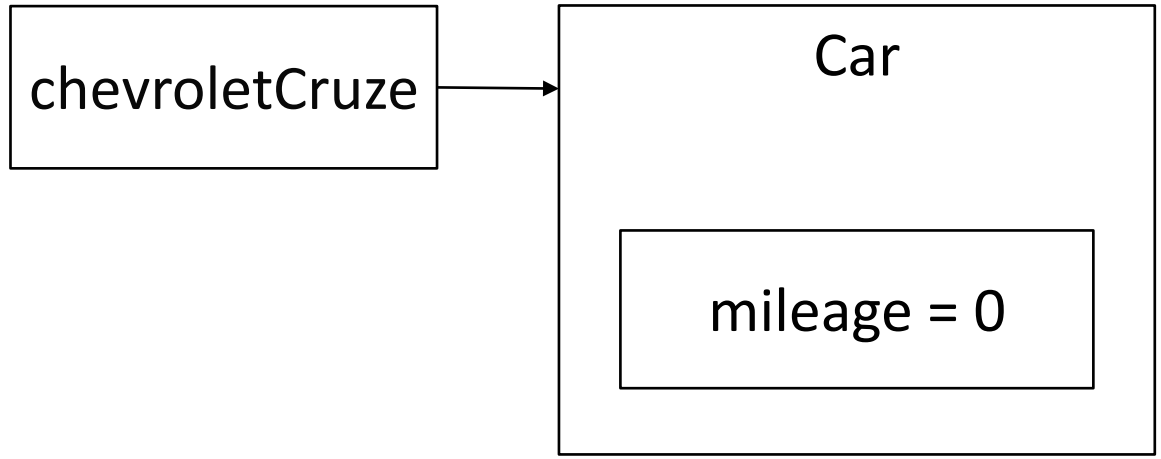


- Example:
  - Class: Car (Has attribute of mileage)
  - Object (instance): A specific car. Red with 79000 miles.

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance


chevroletCruze = Car()
chevroletCruze.drive(500)
chevroletCruze.drive(300)
chevroletCruze.drive(200)
```

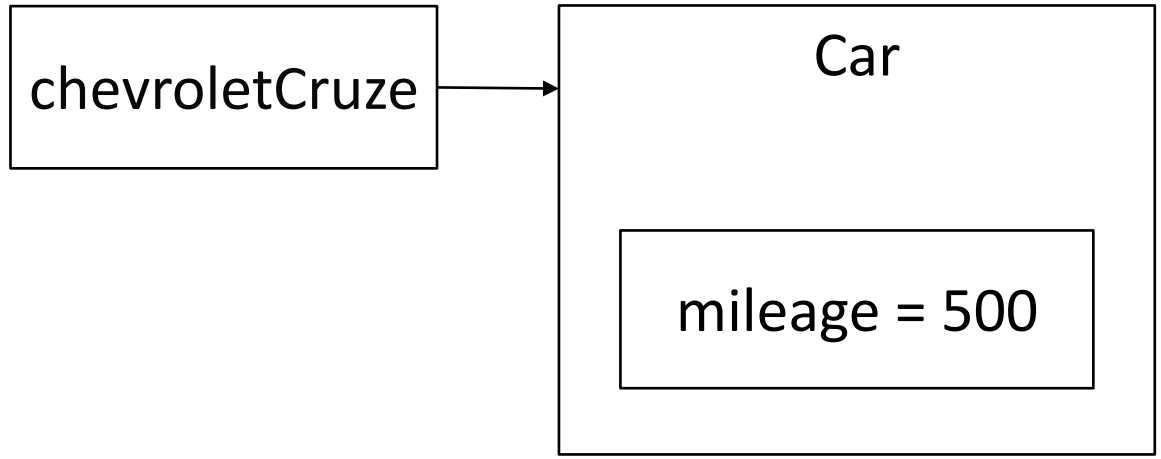
- Example:
  - Class: Car (Has attribute of mileage)
  - Object (instance): A specific car. Red with 79000 miles.

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance


chevroletCruze = Car()
chevroletCruze.drive(500)
chevroletCruze.drive(300)
chevroletCruze.drive(200)
```



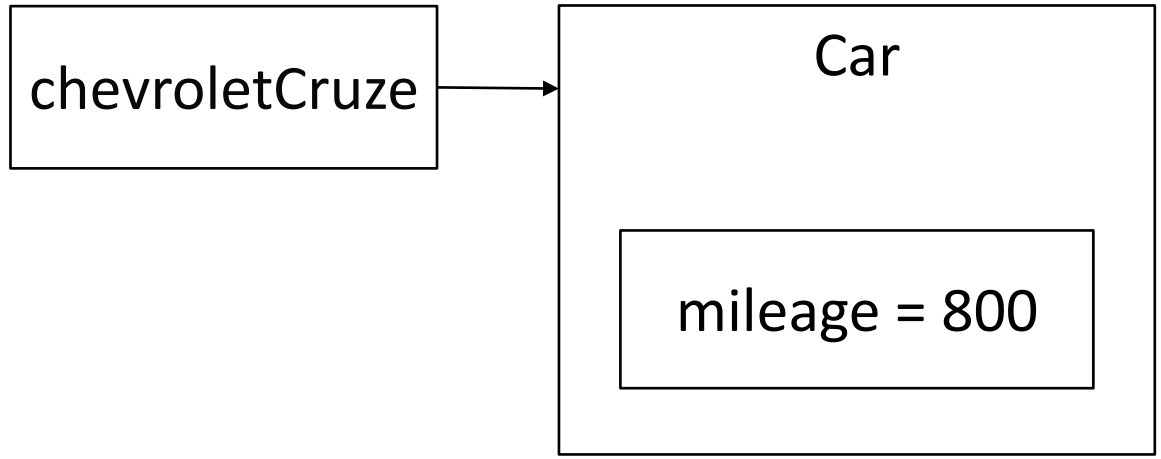
- Example:
  - Class: Car (Has attribute of mileage)
  - Object (instance): A specific car. Red with 79000 miles.

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance


chevroletCruze = Car()
chevroletCruze.drive(500)
chevroletCruze.drive(300)
chevroletCruze.drive(200)
```



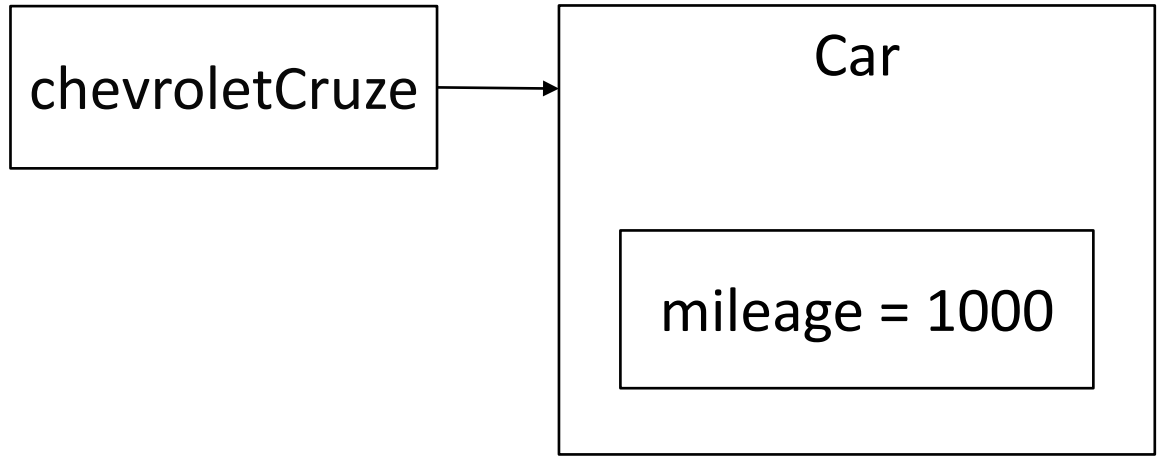
- Example:
  - Class: Car (Has attribute of mileage)
  - Object (instance): A specific car. Red with 79000 miles.

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance


chevroletCruze = Car()
chevroletCruze.drive(500)
chevroletCruze.drive(300)
chevroletCruze.drive(200)
```



- Example:
  - Class: Car (Has attribute of mileage)
  - Object (instance): A specific car. Red with 79000 miles.

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance


chevroletCruze = Car()
chevroletCruze.drive(500)
chevroletCruze.drive(300)
chevroletCruze.drive(200)
```



- Example:
  - Class: Car (Has attribute of mileage)
  - Object (instance): A specific car. Red with 79000 miles.

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance
```

```
chevroletCruze = Car()
chevroletCruze.drive(500)
chevroletCruze.drive(300)
chevroletCruze.drive(200)
```

- Python always passes the object as the first argument
  - convention is to use **self** as the name of the first argument of all methods
- The "." operator is used to access any attribute
  - an attribute of an object
  - a method (procedural attribute) of an object

Is this correct?

```
class Car:
    def __init__(self):
        mileage = 0

    def drive(self, distance):
        mileage += distance

redChevyCruze = Car()
redChevyCruze.drive(79000)
print(redChevyCruze.mileage)
```

This line does not produce any error, but it just creates a variable mileage. *mileage* is not an attribute of the car object.

This line gives an error, since mileage created in __init__() cannot be accessed here.

How to print mileage?

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance

    def print_mileage(self):
        print(________) # 1


chevroletCruze = Car()
chevroletCruze.print_mileage()
print(____________) #2
```

	#1	#2
•	A mileage	chevroletCruze.mileage
•	B mileage	mileage
•	C self.mileage	self.mileage
•	D self.mileage	mileage
•	E self.mileage	chevroletCruze.mileage

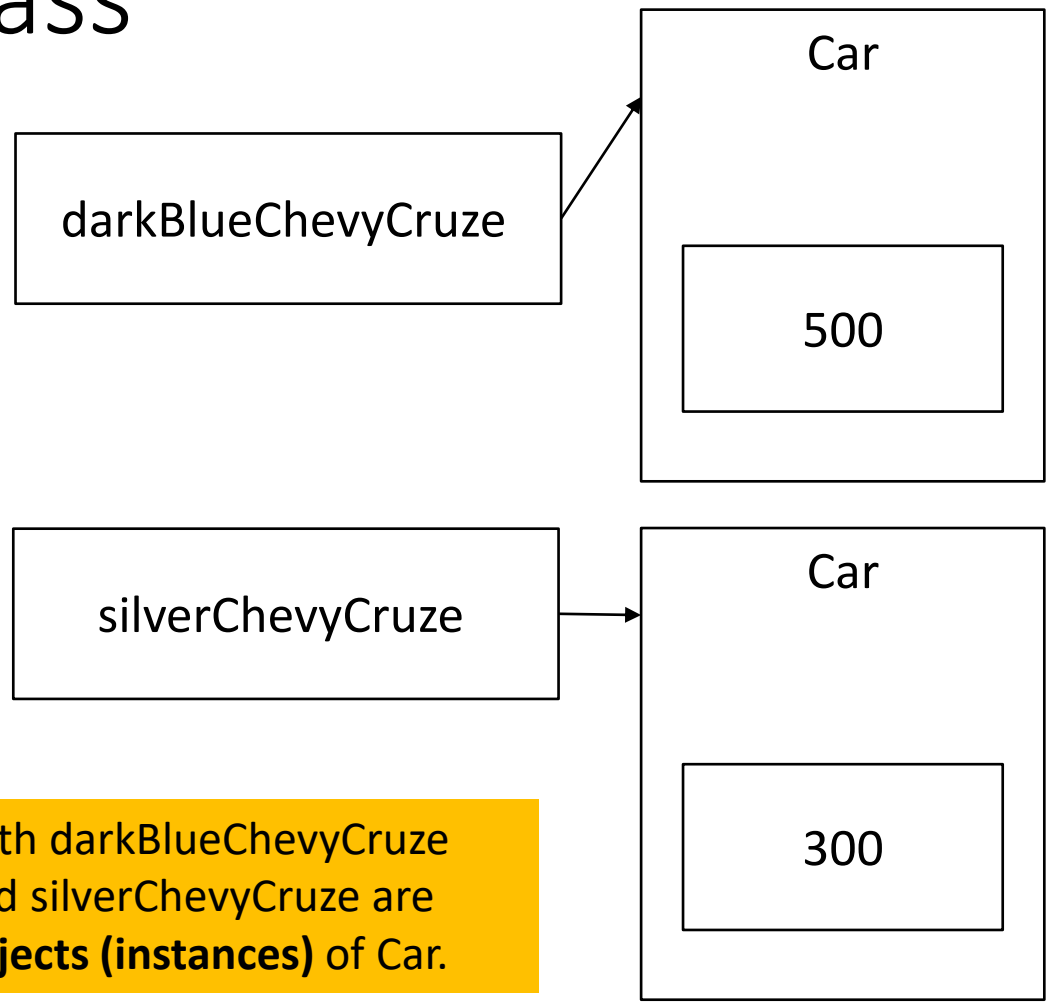
How to print mileage?

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance

    def print_mileage(self):
        print(self.mileage) # 1


chevroletCruze = Car()
chevroletCruze.print_mileage()
print(chevroletCruze.mileage) #2
```

	#1	#2
•	A mileage	chevroletCruze.mileage
•	B mileage	mileage
•	C self.mileage	self.mileage
•	D self.mileage	mileage
•	E self.mileage	chevroletCruze.mileage

Multiple Objects of a Class

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance

darkBlueChevyCruze = Car()
darkBlueChevyCruze.drive(500)
print(darkBlueChevyCruze.mileage) 500

silverChevyCruze = Car()
silverChevyCruze.drive(100)
silverChevyCruze.drive(200)
print(silverChevyCruze.mileage) 300
```



Both darkBlueChevyCruze and silverChevyCruze are **objects (instances)** of Car.

Object as param: What will be printed?

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance

    def mileage_diff(self, car):
        print(self.mileage - car.mileage)

car1 = Car()
car1.drive(100)
car2 = Car()
car2.drive(300)
car1.mileage_diff(car2)
car2.mileage_diff(car1)
```

- A -200 200
- B 200 200
- C 0 0
- D 200 0
- E 0 -200

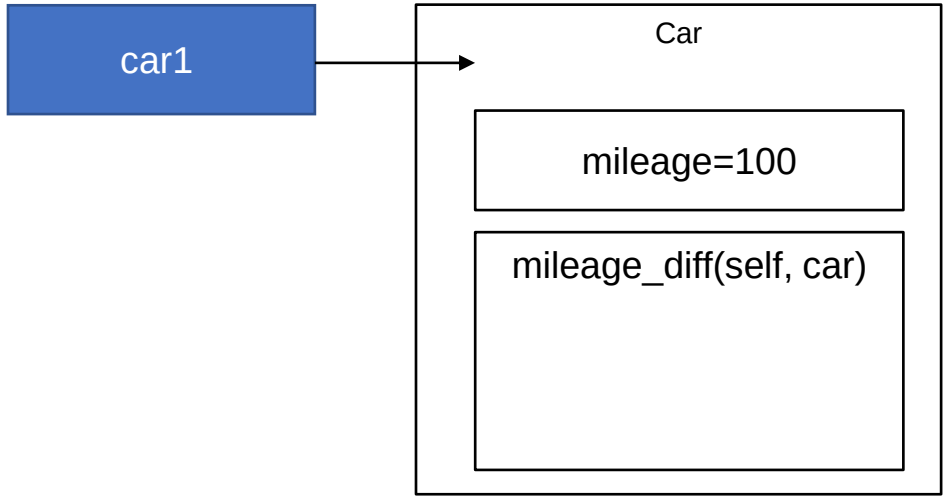
Object as param: What will be printed?

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance

    def mileage_diff(self, car):
        print(self.mileage - car.mileage)

car1 = Car()
car1.drive(100)
car2 = Car()
car2.drive(300)
car1.mileage_diff(car2)
car2.mileage_diff(car1)
```

- A -200 200
- B 200 200
- C 0 0
- D 200 0
- E 0 -200

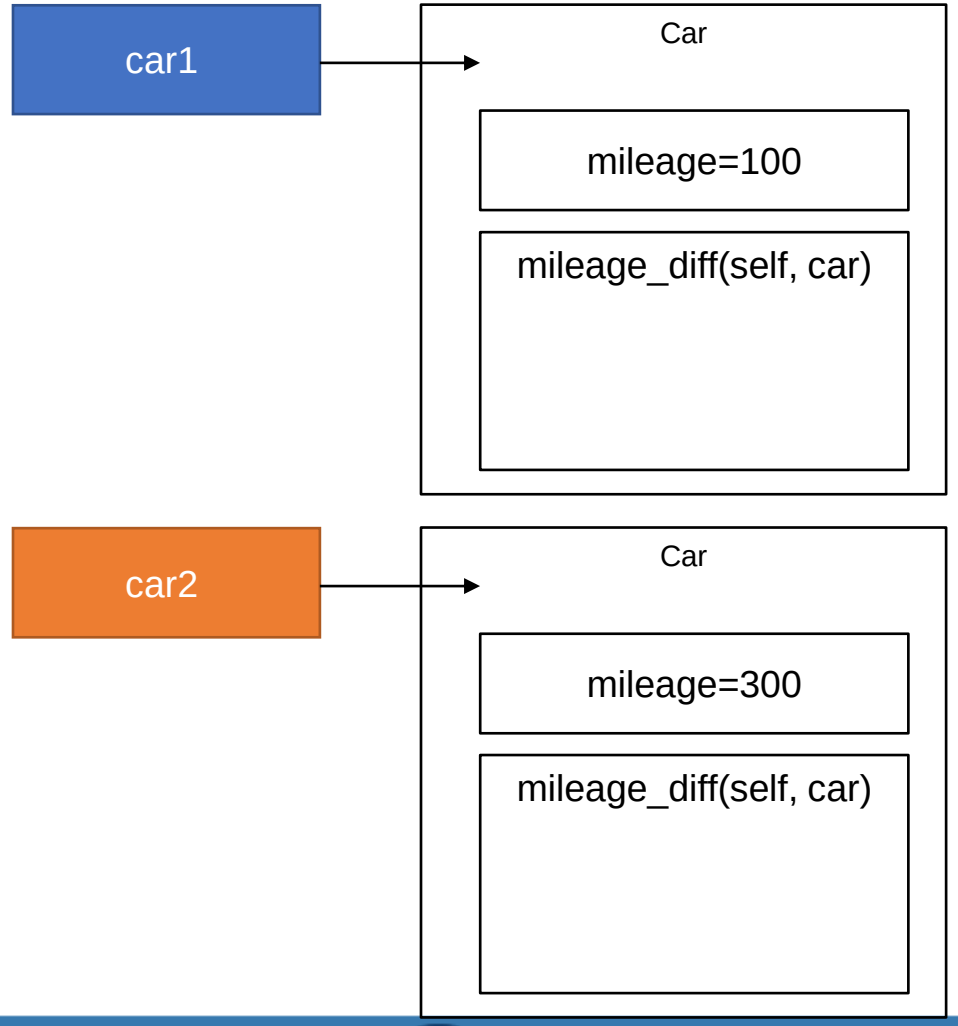
What will be printed?

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance

    def mileage_diff(self, car):
        print(self.mileage - car.mileage)

car1 = Car()
car1.drive(100)
car2 = Car()
car2.drive(300)

car1.mileage_diff(car2)    # -200
car2.mileage_diff(car1)    # 200
```



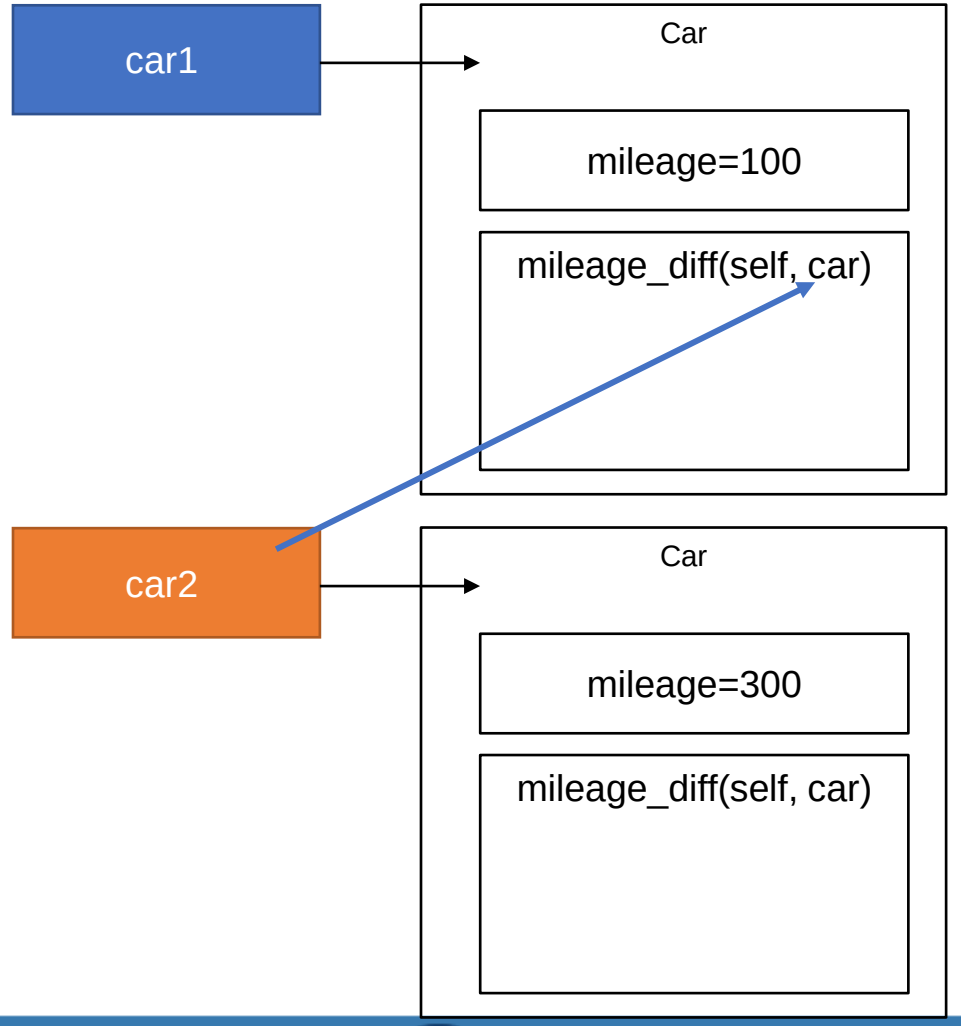
What will be printed?

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance

    def mileage_diff(self, car):
        print(self.mileage - car.mileage)

car1 = Car()
car1.drive(100)
car2 = Car()
car2.drive(300)

car1.mileage_diff(car2)    # -200
car2.mileage_diff(car1)    # 200
```



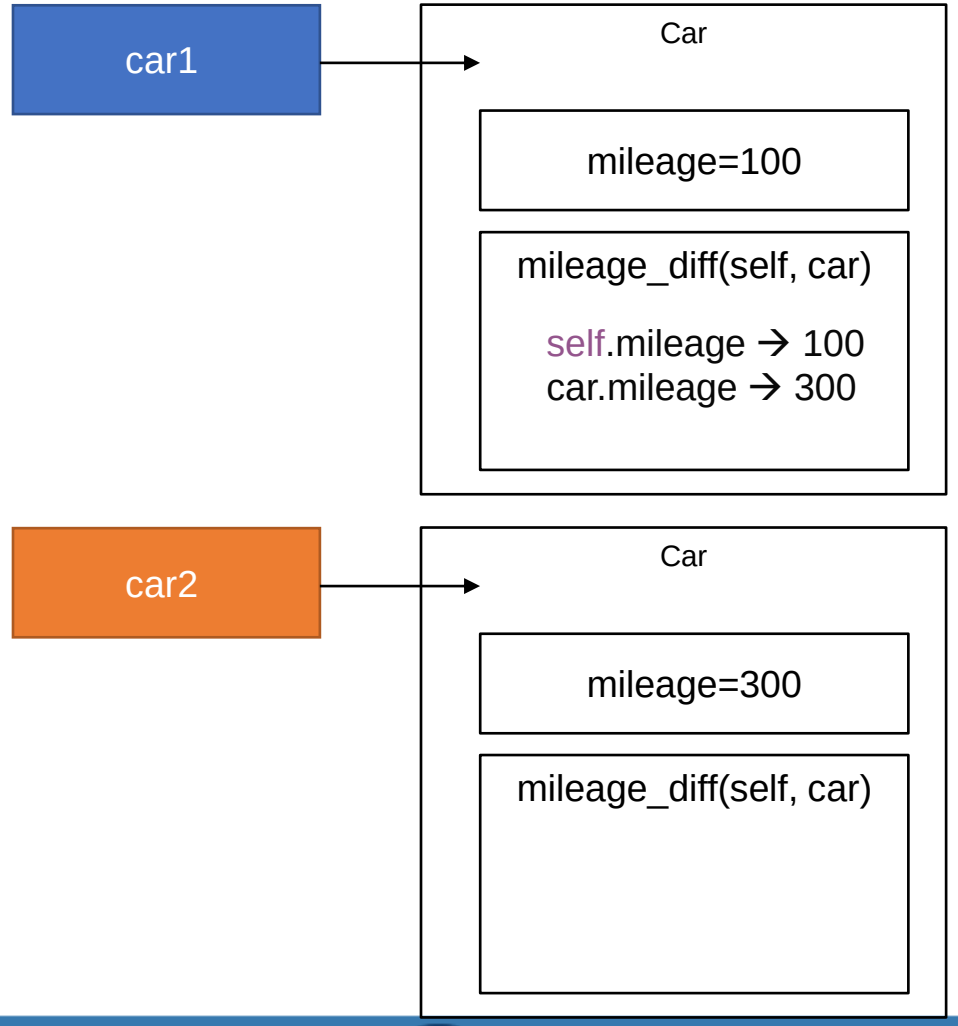
What will be printed?

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance

    def mileage_diff(self, car):
        print(self.mileage - car.mileage)

car1 = Car()
car1.drive(100)
car2 = Car()
car2.drive(300)

car1.mileage_diff(car2)    # -200
car2.mileage_diff(car1)    # 200
```



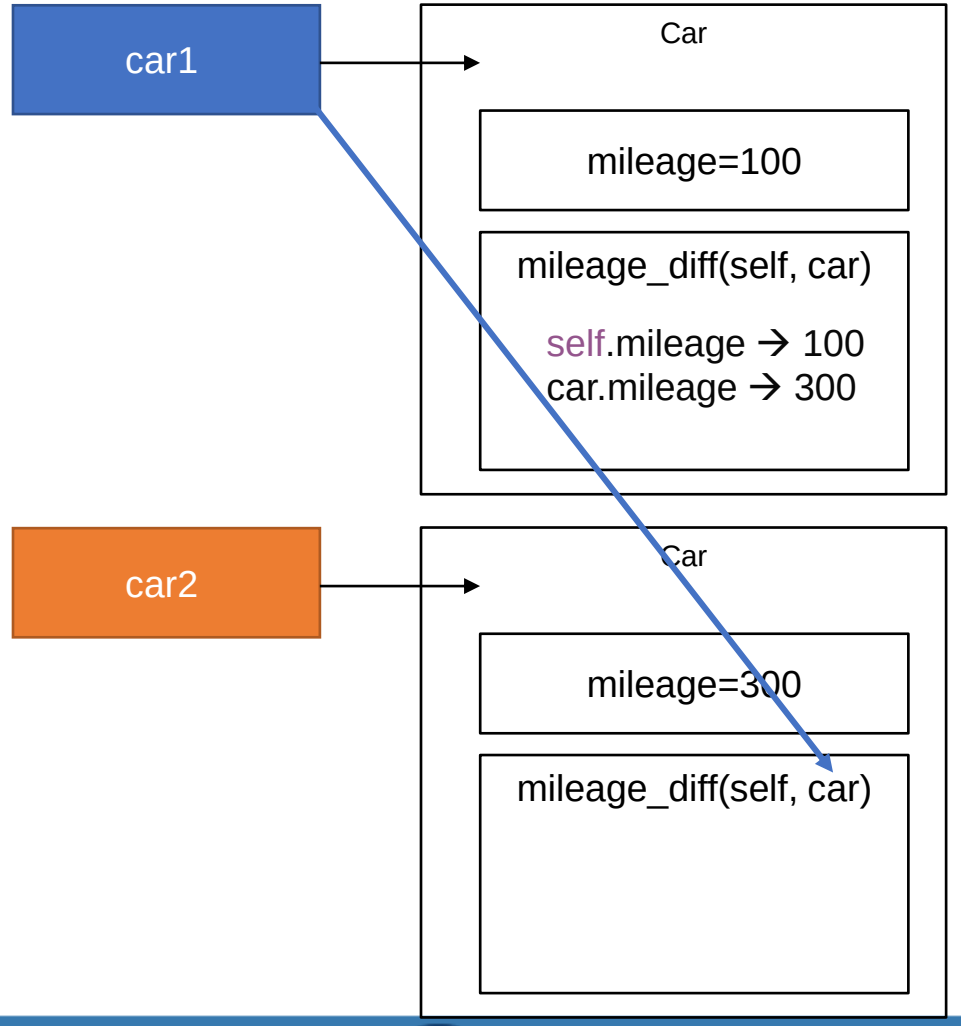
What will be printed?

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance

    def mileage_diff(self, car):
        print(self.mileage - car.mileage)

car1 = Car()
car1.drive(100)
car2 = Car()
car2.drive(300)

car1.mileage_diff(car2)    # -200
car2.mileage_diff(car1)    # 200
```



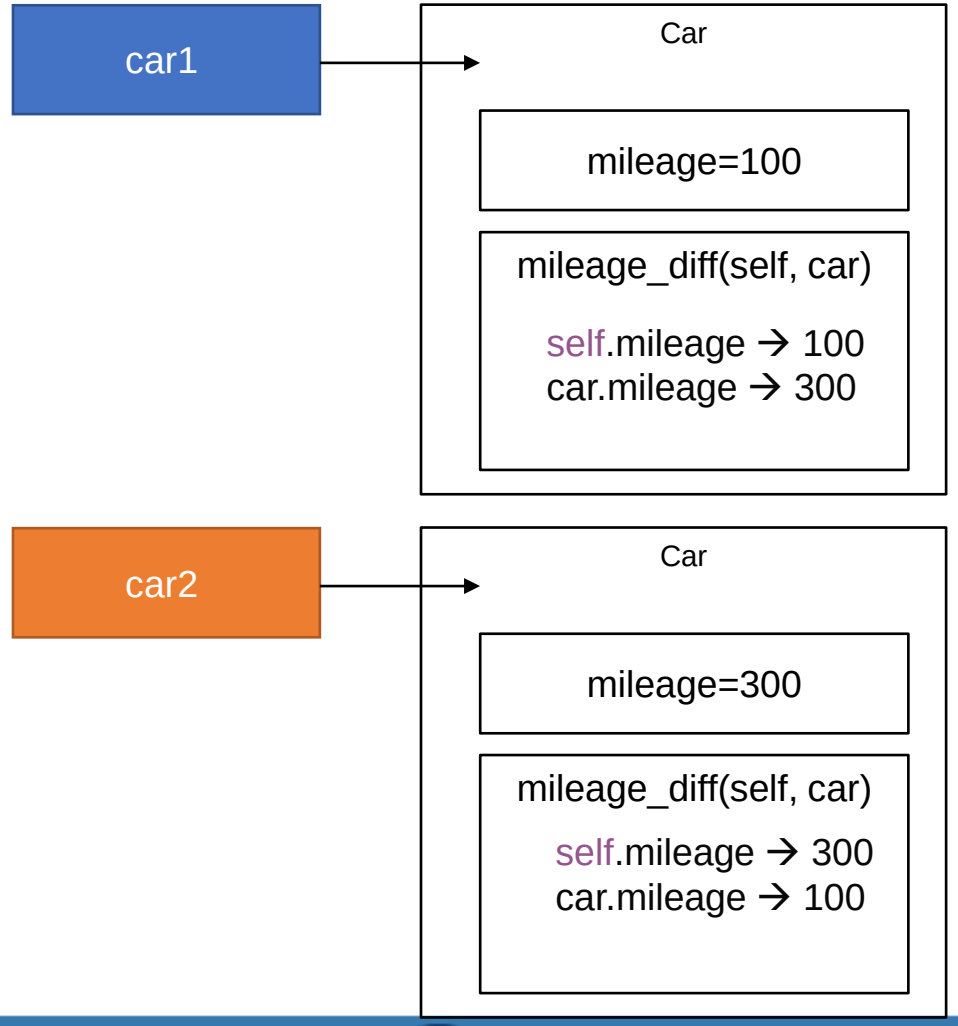
What will be printed?

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance

    def mileage_diff(self, car):
        print(self.mileage - car.mileage)

car1 = Car()
car1.drive(100)
car2 = Car()
car2.drive(300)

car1.mileage_diff(car2)    # -200
car2.mileage_diff(car1)    # 200
```



What will be printed?

```
class Car:
    def __init__(self):
        self.mileage = 0

    def drive(self, distance):
        self.mileage += distance

    def mileage_diff(self, car):
        print(self.mileage - car.mileage)

car1 = Car()
car1.drive(100)
car2 = Car()
car2.drive(300)

car1.mileage_diff(car2)    # -200
car2.mileage_diff(car1)    # 200
```



Initialize an Object

- __init__() can have other parameters to initialize attributes.

```
class Car:
    def __init__(self, make):
        self.make = make


car1 = Car(”Ford")
print(car1.make) # Ford
```

```
class Car:
    def __init__(self, make, year):
        self.make = make
        self.year = year


car1 = Car(”Ford”, 2005)
print(car1.year)   # 2005
```

Return

- Similar to functions, methods can return values

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def get_info(self):
        print(f”This is a {self.make}, model {self.model}, from {self.year}.")


car1 = Car(”Ford", ”Escape", 2015)
car1.get_info()    #  This is a Ford, model Escape, from 2015.
```

Return

- Similar to functions, methods can return values

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def get_info(self):
        return(f"This is a {self.make}, model {self.model}, from {self.year}.")


car1 = Car("Ford", "Escape", 2015)
info = car1.get_info()
print(info)                 #  This is a Ford, model Escape, from 2015.
```

Return

- The __repr__() function returns a printable representation of the given object.

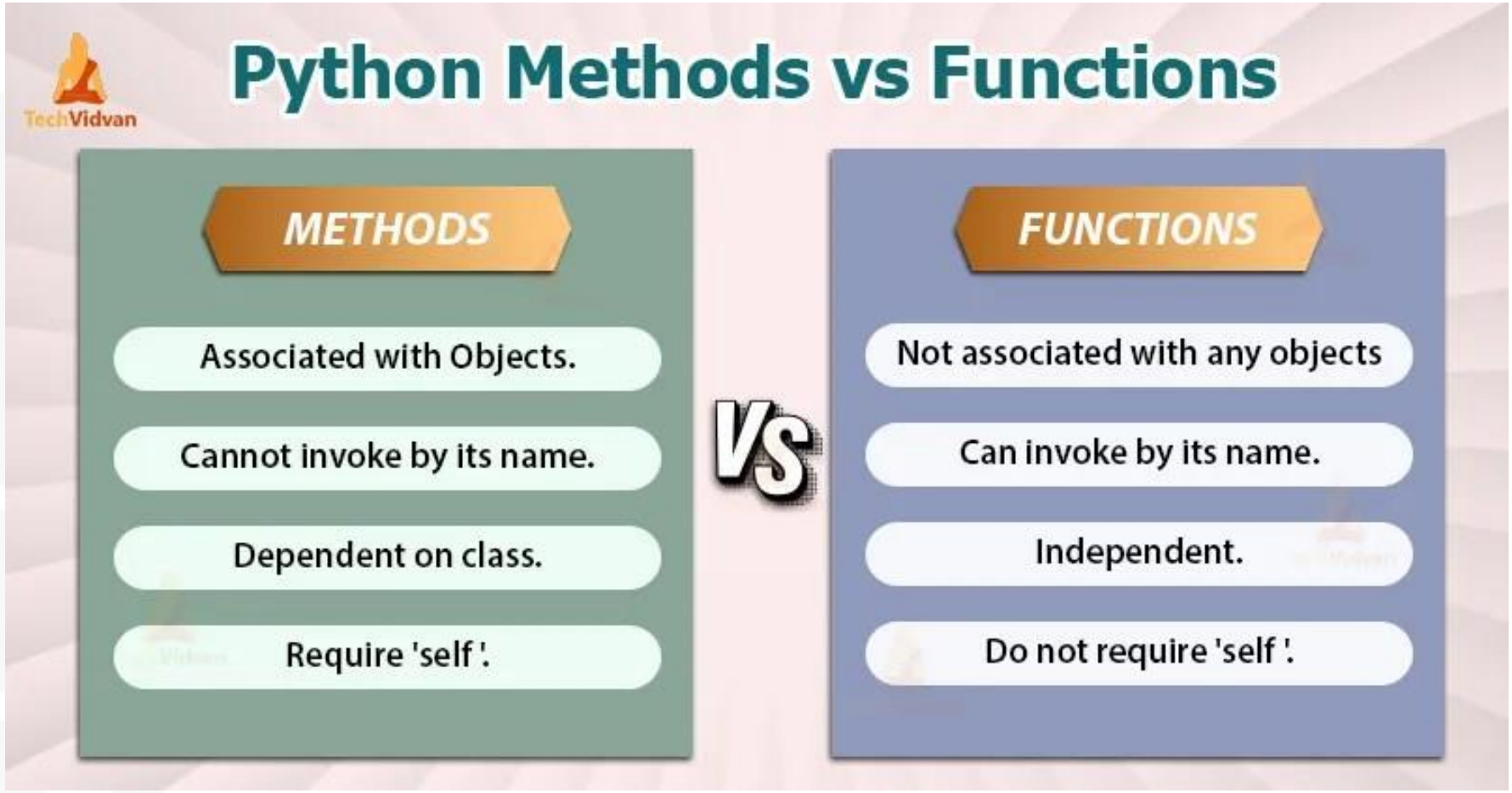
```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def __repr__(self):
        return (f"This is a {self.make}, model {self.model}, from {self.year}.")


car1 = Car("Ford", "Escape", 2015)
print(car1)          #  This is a Ford, model Escape, from 2015.
```

Method vs. Function

- Our drive method is part of the class
  - Thus, it is inside the Car object.
  - Must be called on the object: car.drive(300)
- Here's a ***function*** that takes two `car` objects as inputs:
  - it is *not* part of the class and is *not* inside Car objects

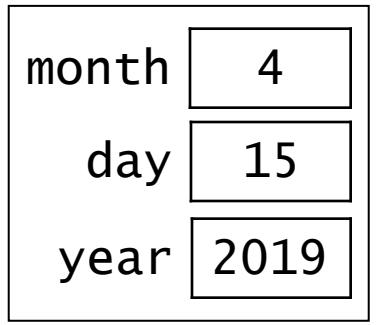
```
class Car:
    …

    def drive(self, distance):
        self.mileage += distance

def get_newer_car(car1, car2):
    if car1.year >= car2.year:
        return car1
    else:
        return car2

get_newer_car(car1, car2)
```

Method vs. Function



Date Class

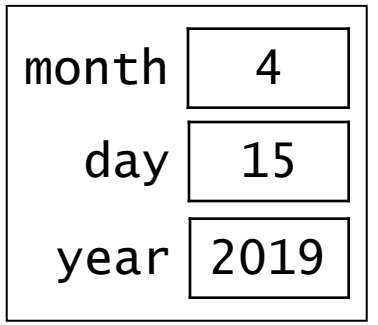
```
class Date:
    def __init__(self, month_param, day_param, year_param):
        """The constructor for the Date class"""
```



Example

Date Class

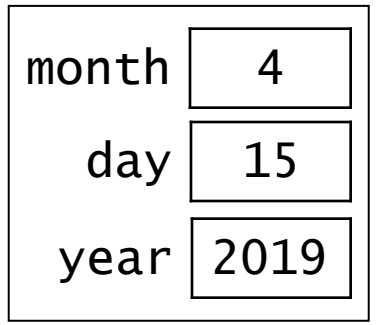
```
class Date:
    def __init__(self, month_param, day_param, year_param):
        """The constructor for the Date class"""



    def __repr__(self):
        """This method returns a string representation for the
           object of type Date that calls it (named self)."""

        s =  f'{self.month:02d}/{self.day:02d}/{self.year:04d}'
        return s
```



Example

Date Class

```
class Date:
    def __init__(self, month_param, day_param, year_param):
        …

    def __repr__(self):
        …
        return s

    def copy(self):
        new_date = Date(self.month, self.day, self.year)
        return new_date
```



Example

Date Class

- Example of how Date objects can be used:

```
>>> d = Date(12, 31, 2018)              # calls __init__
>>> print(d)                            # calls __repr__
12/31/2018
>>> d.advance_one_day()                 # a method you will write
                                        # nothing is returned!
>>> print(d)                            # d has been changed!
01/01/2019
```