Object-Oriented Programming

Dr. Dongchul Kim

Part I Intro

Object

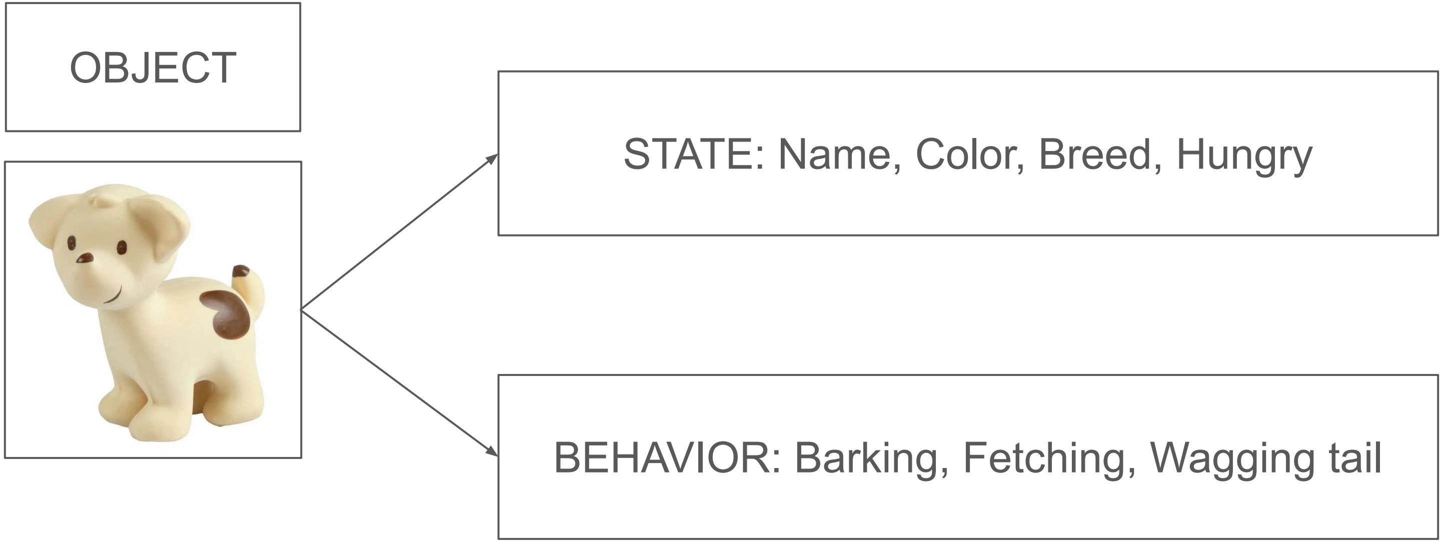
Python is an Object-Oriented Programming (OOP) language, where the fundamental building blocks are objects.

Objects represent entities with characteristics (properties/state) and actions (methods). In essence, objects encapsulate both states and behaviors.

Each object is an instance of a class, and **classes** define the blueprint for creating objects.

In the real world, we encounter numerous objects, such as cars, dogs, and humans, each having its unique set of properties and behaviors.

For instance, if we consider a dog as an object, its properties or state include attributes like name, breed, and color, while its behaviors encompass actions like barking, wagging its tail, and running.



OBJECT

STATE: Name, Color, Breed, Hungry

BEHAVIOR: Barking, Fetching, Wagging tail

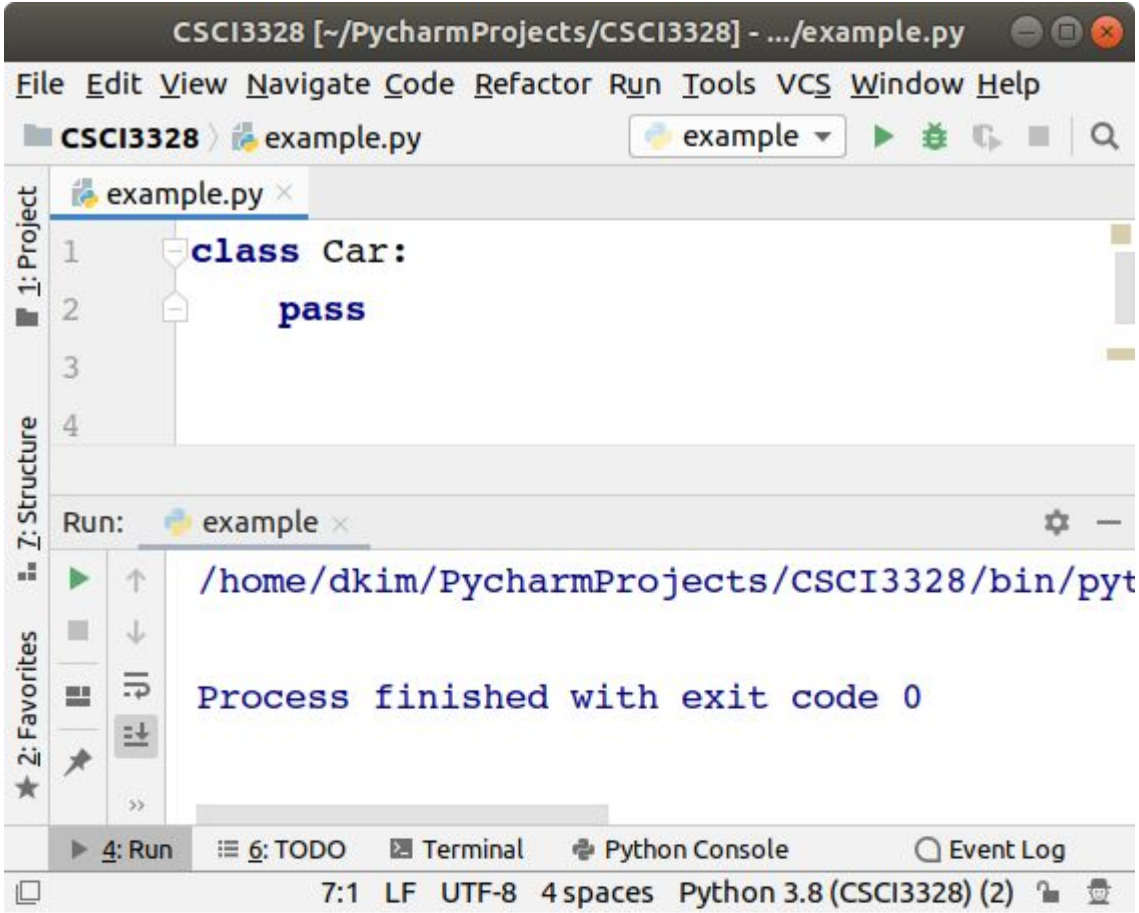
Class

A class can be defined as a template/blueprint that describes the behaviors/states of the object.

Objects in OOP have a state and behavior. Software object's states are stored in **attributes** and behaviors are shown via **methods** (functions).

Before we talk about attributes and methods, let’s define a class first now.

```
class Car:  # class name is Car

    pass    # we don’t define anything for now here.
```


CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328
example.py
example
1: Project
7: Structure
2: Favorites
Run:
example
4: Run
6: TODO
Terminal
Python Console
Event Log
7:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

```
class Car:
    pass
```

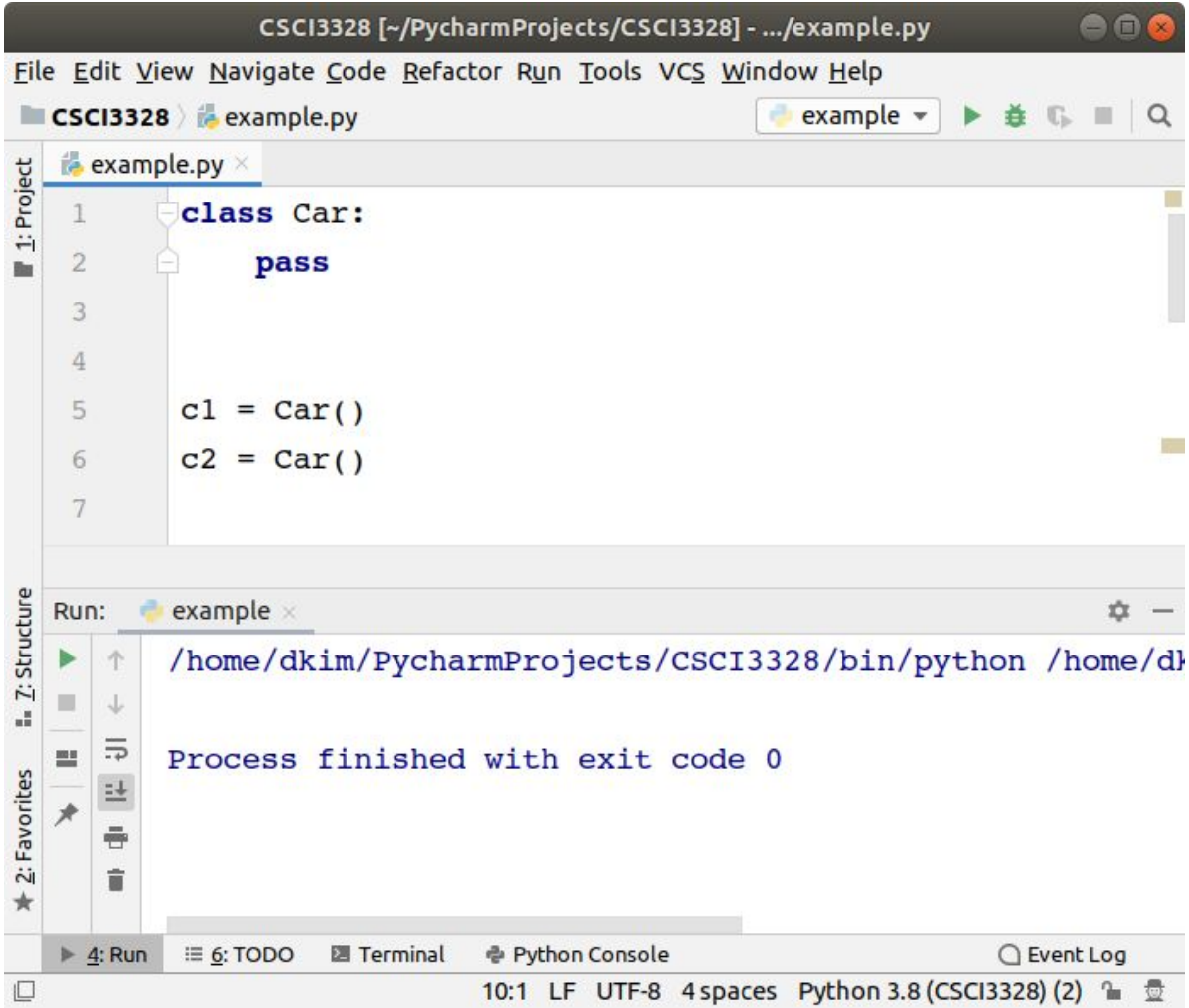
```
/home/dkim/PycharmProjects/CSCI3328/bin/pyt

Process finished with exit code 0
```

Creating an object given a class

Once we defined a class, we can instantiate it to create a new object from that class. We say the new object has the type of the class it was instantiated from.

```
class Car:  # class name is  Car
    pass    # we don’t define anything for now here.
c1 = Car()   # our object is c1
c2 = Car()   # Another Car object, c2
```

We can create multiple objects from the same class, and each object will be unique. They will all have the same type, but they can store different values for their individual properties.



CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328
example.py
example
1: Project
7: Structure
2: Favorites
Run:
example
4: Run
6: TODO
Terminal
Python Console
Event Log
10:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

```python
class Car:
    pass


c1 = Car()
c2 = Car()

```

```
/home/dkim/PycharmProjects/CSCI3328/bin/python /home/dk

Process finished with exit code 0
```

Property/Attribute of Object

There are two ways to define an attribute of the object.

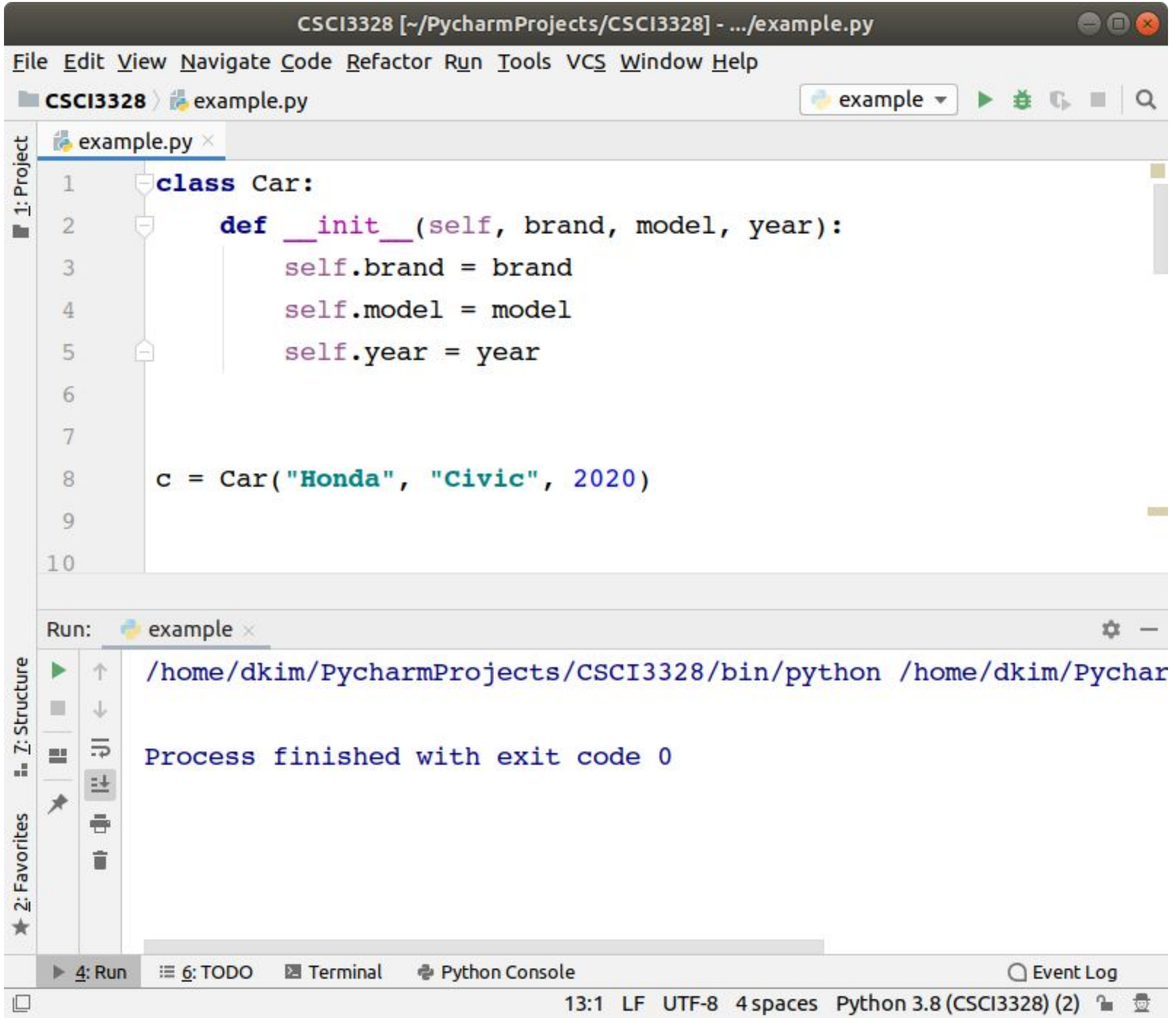
1. Instance Attribute

Instance attributes are specific to each individual object, where an object is also known as an instance. Take the case of a car object: each one can have its own distinct brand, model, and year. Modifying any of these attributes in one car object does not impact the attributes of any other car objects that have been created.

Instance Attribute

The `init` method is known as the initializer. It is automatically invoked when a class is instantiated. Its primary role is to ensure that the class has all the necessary attributes. Additionally, it is often employed to verify that the object is in a valid state upon instantiation, such as confirming that a user has not entered a negative year for a car.

```
class Car:

    def __init__(self, brand, model, year):

        self.brand = brand

        self.model = model

        self.year = year
```



CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328 example.py
example
example.py
1: Project
7: Structure
2: Favorites
Run: example
4: Run
6: TODO
Terminal
Python Console
Event Log
13:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

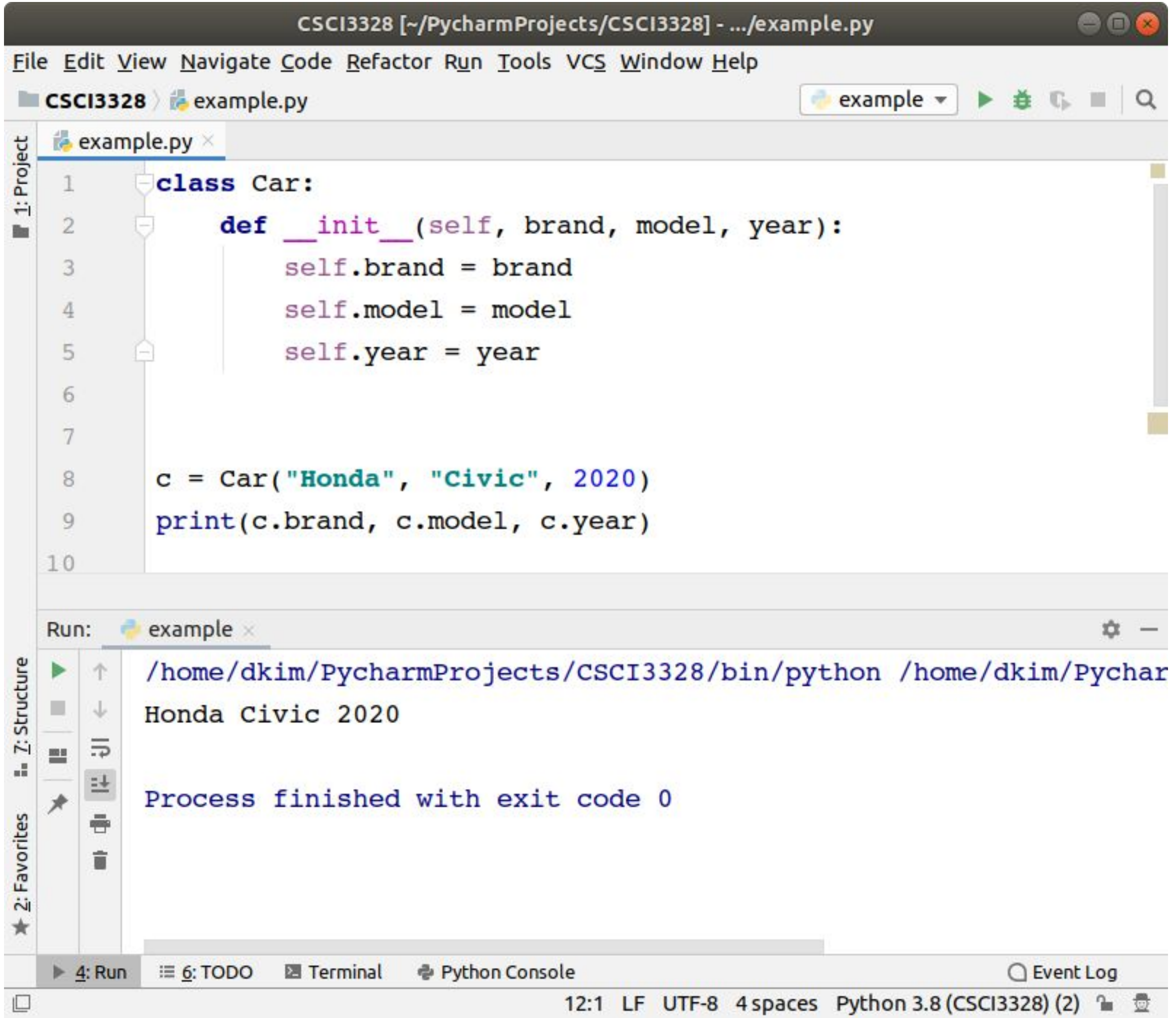
```python
class Car:
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year


c = Car("Honda", "Civic", 2020)
```

```
/home/dkim/PycharmProjects/CSCI3328/bin/python /home/dkim/Pychar

Process finished with exit code 0
```

Accessing instance attributes

After creating an object and its instance attribute, you can access the attribute using dot (.) operator. For example,

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

c = Car("Honda", "Civic", 2020)
print(c.brand, c.model, c.year)
```


CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328
example.py
example
example.py
1: Project
Run:
example
7: Structure
2: Favorites
4: Run
6: TODO
Terminal
Python Console
Event Log
12:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

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


c = Car("Honda", "Civic", 2020)
print(c.brand, c.model, c.year)
```

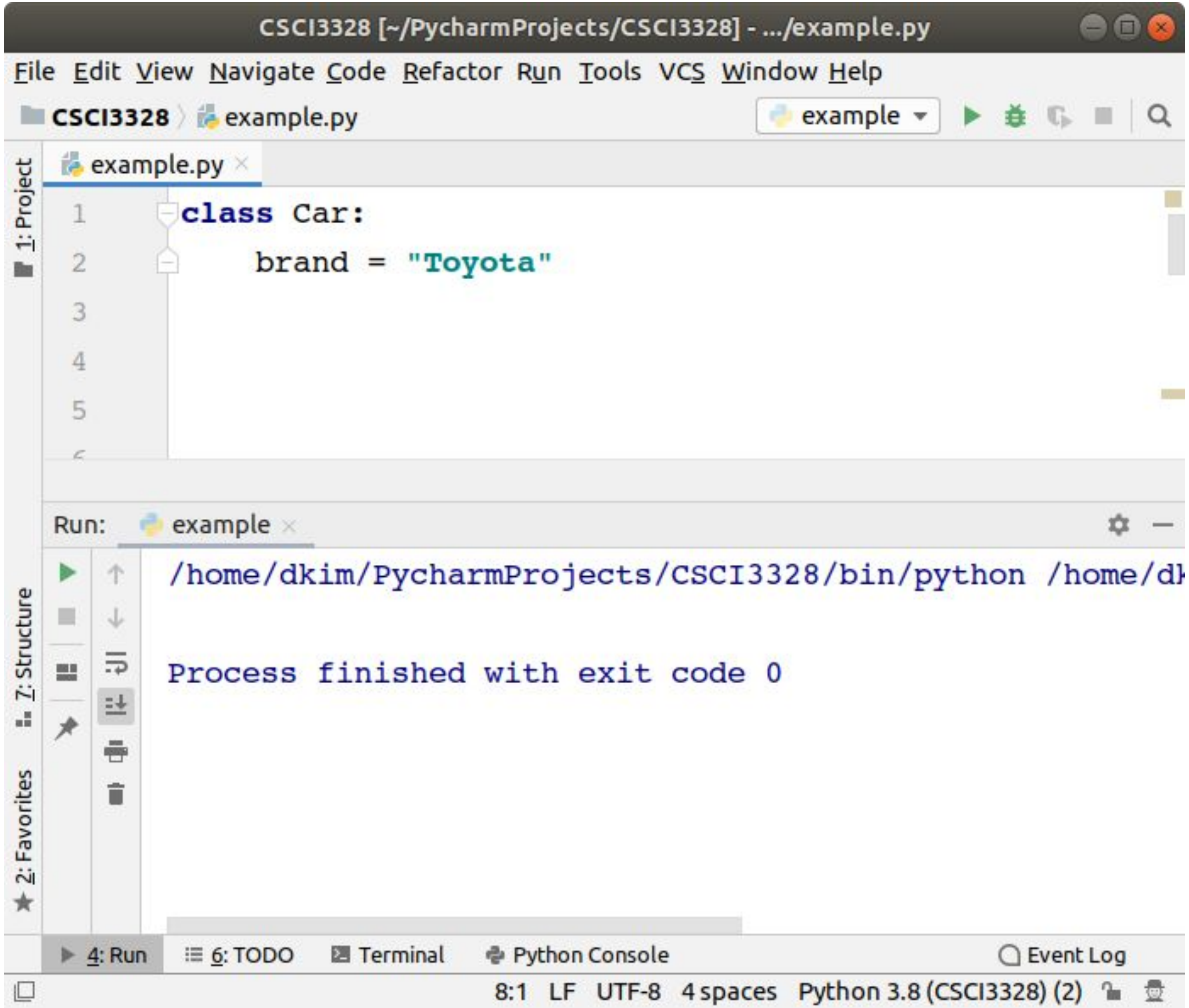
```
/home/dkim/PycharmProjects/CSCI3328/bin/python /home/dkim/Pychar
Honda Civic 2020

Process finished with exit code 0
```

Property/Attribute of Object

There are two ways to define an attribute of the object.

2. class attribute

```
class Car:

    # an attribute, "brand" is created

    # the attribute is assigned with the value "Toyota"

    brand = "Toyota"
```



CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328 example.py
example
example.py
1: Project
Run: example
7: Structure
2: Favorites
4: Run
6: TODO
Terminal
Python Console
Event Log
8:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

```python
class Car:
    brand = "Toyota"
```

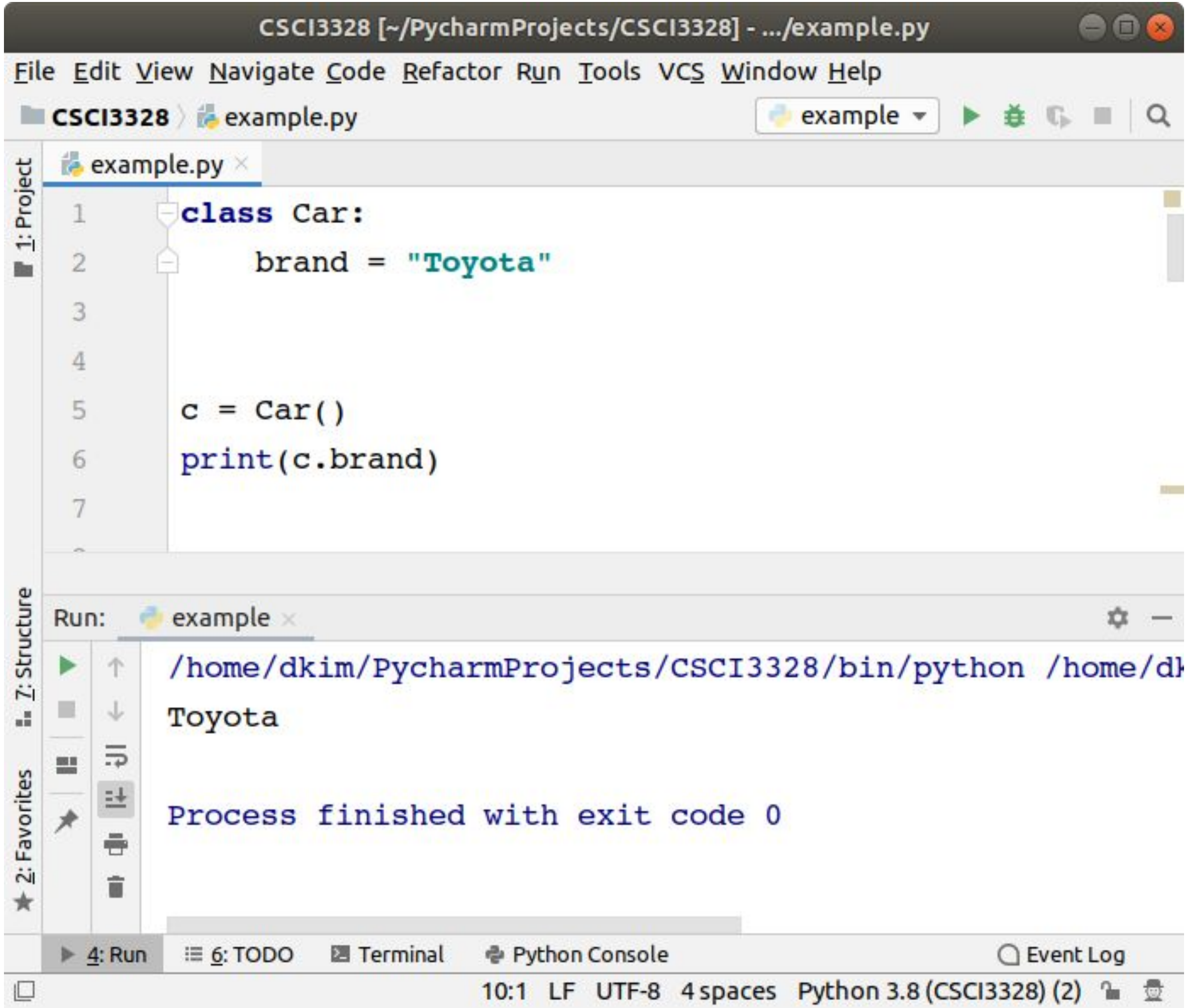
```
/home/dkim/PycharmProjects/CSCI3328/bin/python /home/dk

Process finished with exit code 0
```

Accessing class attributes

After creating an object and its attribute, you can access the attribute using dot (.) operator. For example,

```
class Car:

    brand = "Toyota"

c = Car()

print(c.brand)
```

For now, let’s use only instance attributes!



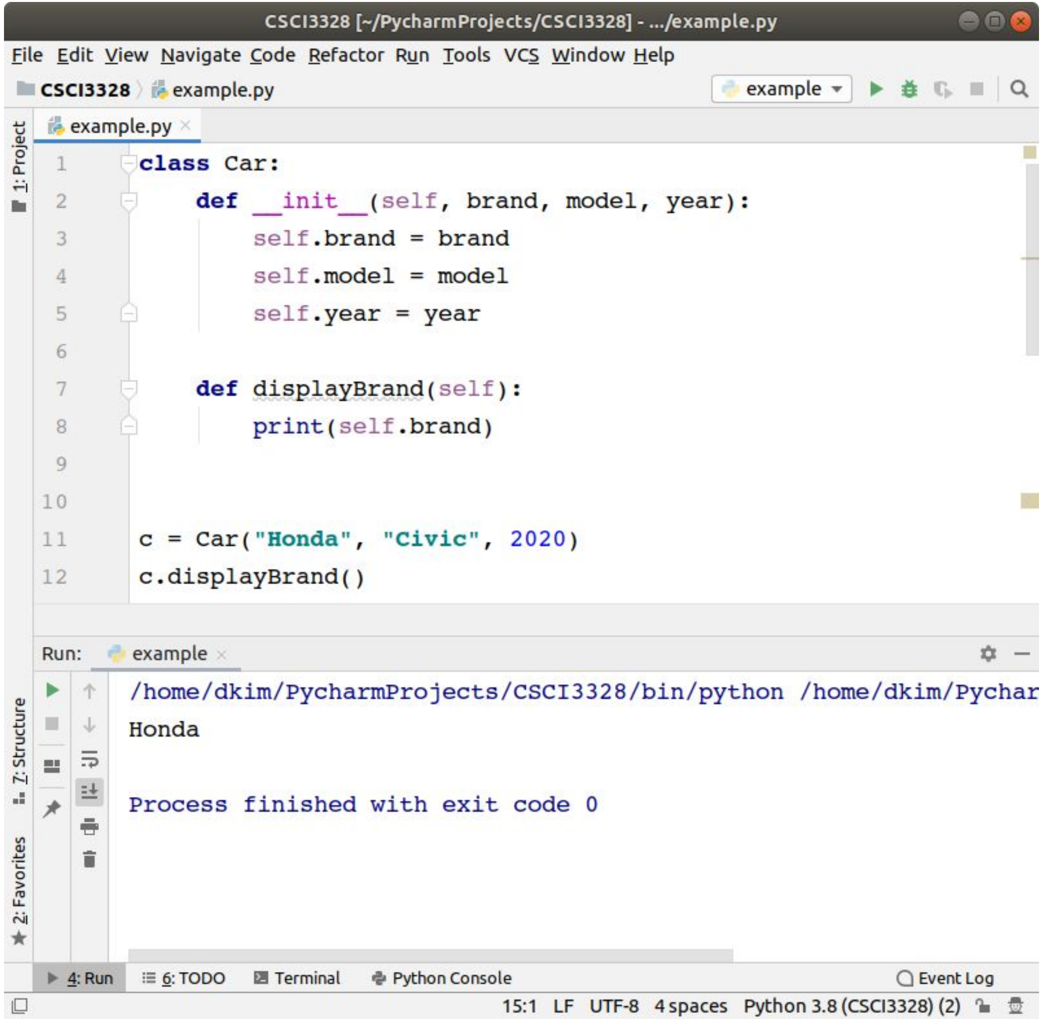
CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328
example.py
example
example.py
1: Project
7: Structure
2: Favorites
Run:
example
4: Run
6: TODO
Terminal
Python Console
Event Log
10:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

```python
class Car:
    brand = "Toyota"


c = Car()
print(c.brand)
```

```
/home/dkim/PycharmProjects/CSCI3328/bin/python /home/dk
Toyota

Process finished with exit code 0
```

Defining a method in a class

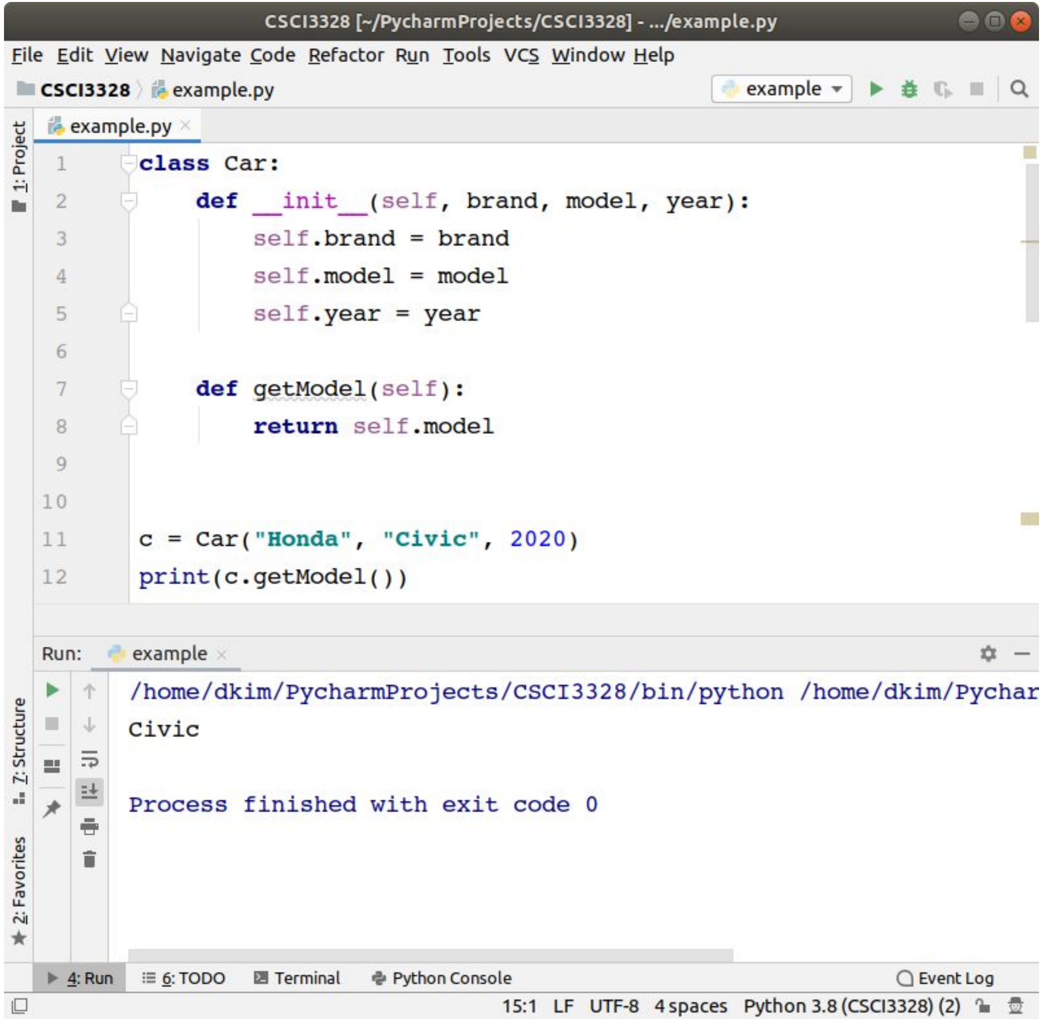
```
class Car:
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year

    def displayBrand(self):
        print(self.brand)
```



CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328 〉 example.py
example
example.py
1: Project
7: Structure
2: Favorites
Run:
example
4: Run
6: TODO
Terminal
Python Console
Event Log
15:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

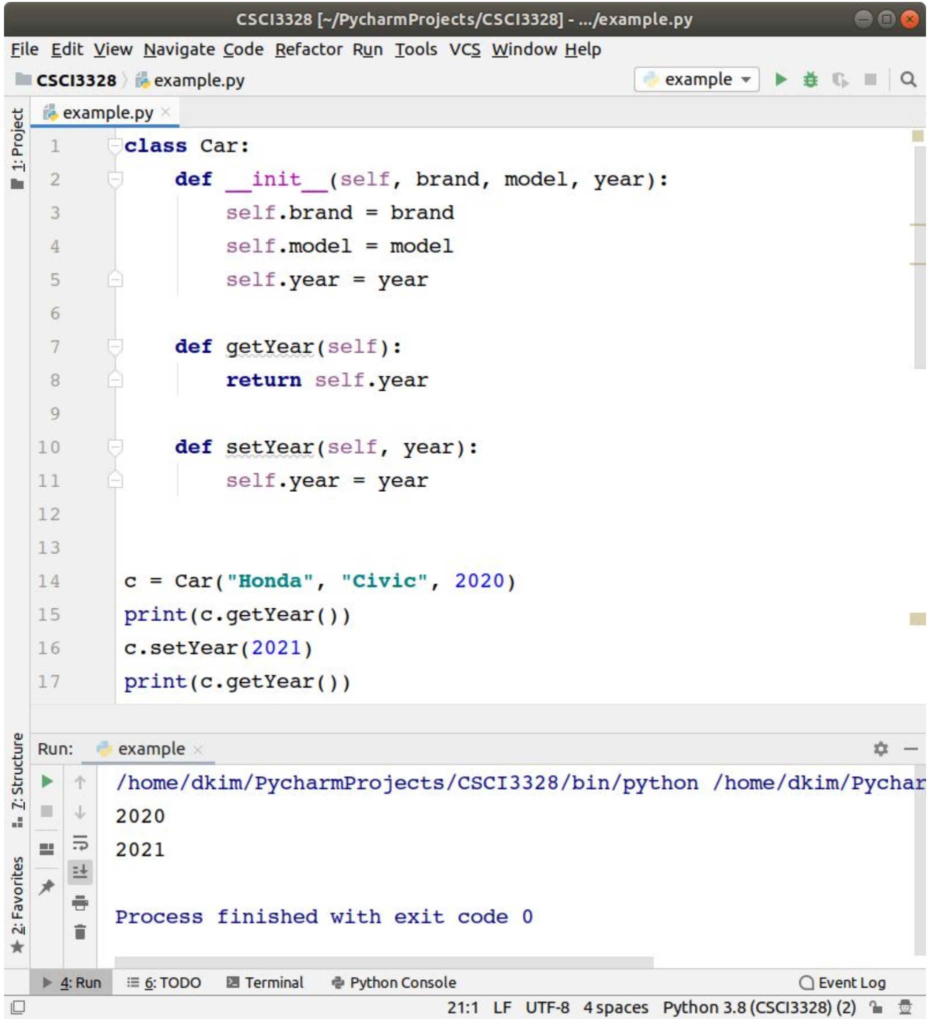
```python
class Car:
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year

    def displayBrand(self):
        print(self.brand)


c = Car("Honda", "Civic", 2020)
c.displayBrand()
```

```
/home/dkim/PycharmProjects/CSCI3328/bin/python /home/dkim/Pychar
Honda

Process finished with exit code 0
```


CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328 〉 example.py
example
example.py
1: Project
7: Structure
2: Favorites
Run: example
4: Run
6: TODO
Terminal
Python Console
Event Log
15:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

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

    def getModel(self):
        return self.model


c = Car("Honda", "Civic", 2020)
print(c.getModel())
```

```
/home/dkim/PycharmProjects/CSCI3328/bin/python /home/dkim/Pychar
Civic

Process finished with exit code 0
```

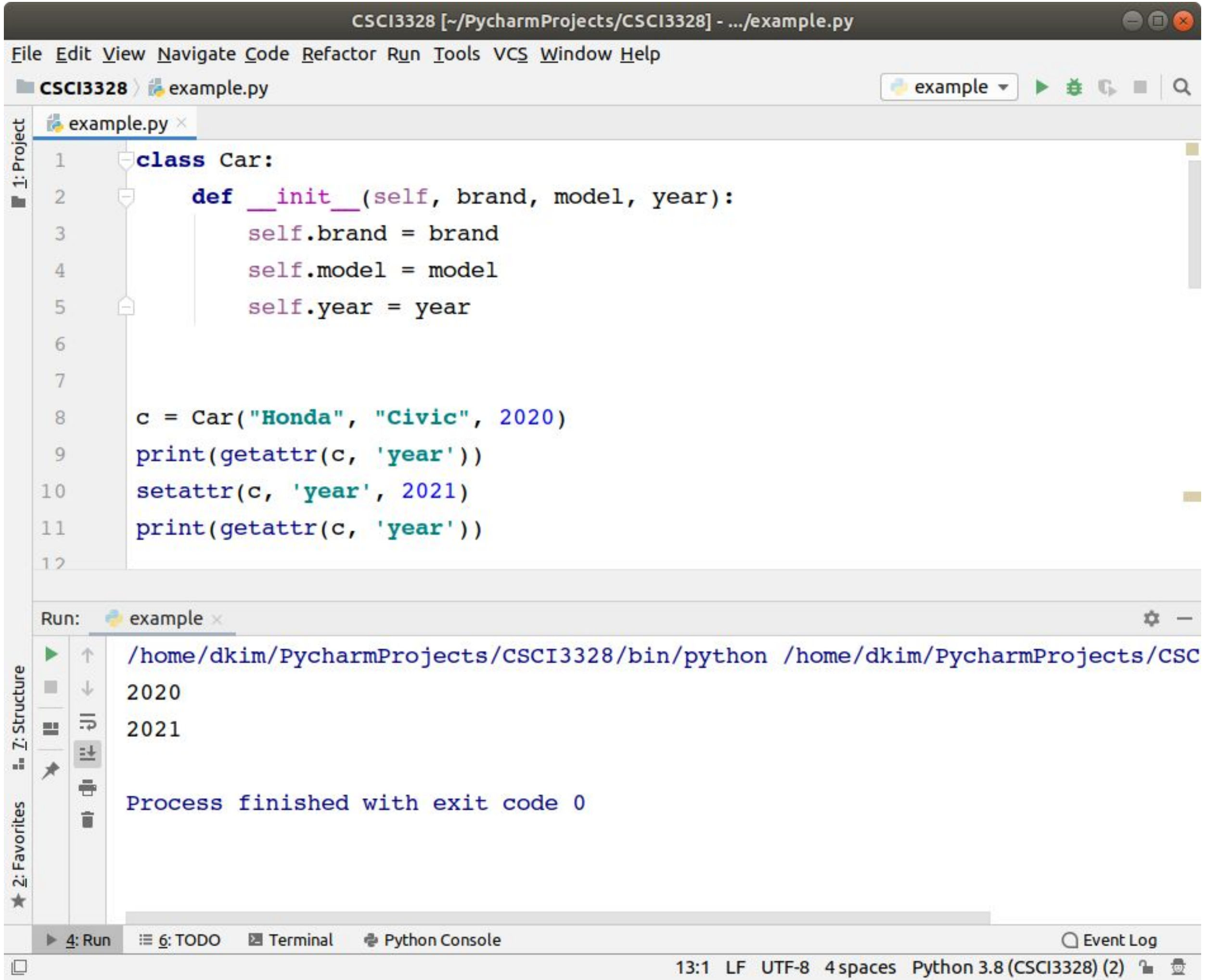

CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328 example.py
example
example.py
1: Project
7: Structure
2: Favorites
Run: example
4: Run
6: TODO
Terminal
Python Console
Event Log
21:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

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

    def getYear(self):
        return self.year

    def setYear(self, year):
        self.year = year


c = Car("Honda", "Civic", 2020)
print(c.getYear())
c.setYear(2021)
print(c.getYear())
```

```
/home/dkim/PycharmProjects/CSCI3328/bin/python /home/dkim/Pychar
2020
2021

Process finished with exit code 0
```

Built-in methods

```
hasattr(x, "attribute_name")       # Returns true if the attribute exists

getattr(x, "attribute_name")       # Returns value of the attribute

setattr(x, "attribute_name", new_value) # Set the attribute to a new value

delattr(x, "attribute_name")       # Delete the attribute
```



CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328 example.py
example
example.py
1: Project
7: Structure
2: Favorites
Run: example
4: Run
6: TODO
Terminal
Python Console
Event Log
13:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

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


c = Car("Honda", "Civic", 2020)
print(getattr(c, 'year'))
setattr(c, 'year', 2021)
print(getattr(c, 'year'))
```

```
/home/dkim/PycharmProjects/CSCI3328/bin/python /home/dkim/PycharmProjects/CSC
2020
2021

Process finished with exit code 0
```

Inheritance

Inheritance is a concept where we extend the functionality of a class to create new classes. There are many benefits of doing this. Foremost is to reuse existing code (called reusability).

The existing class has generic code that can be reused. This class is called parent, base, or super class.

We create a child class that would receive the definition from the parent class.

Let us consider a parent class, `Car`. This has properties and methods suitable to describe any `Car`.

Example

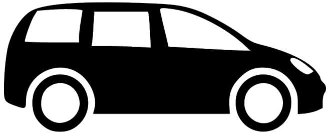
Car class

vin
brand
model
year

SportsCar class

Sedan class

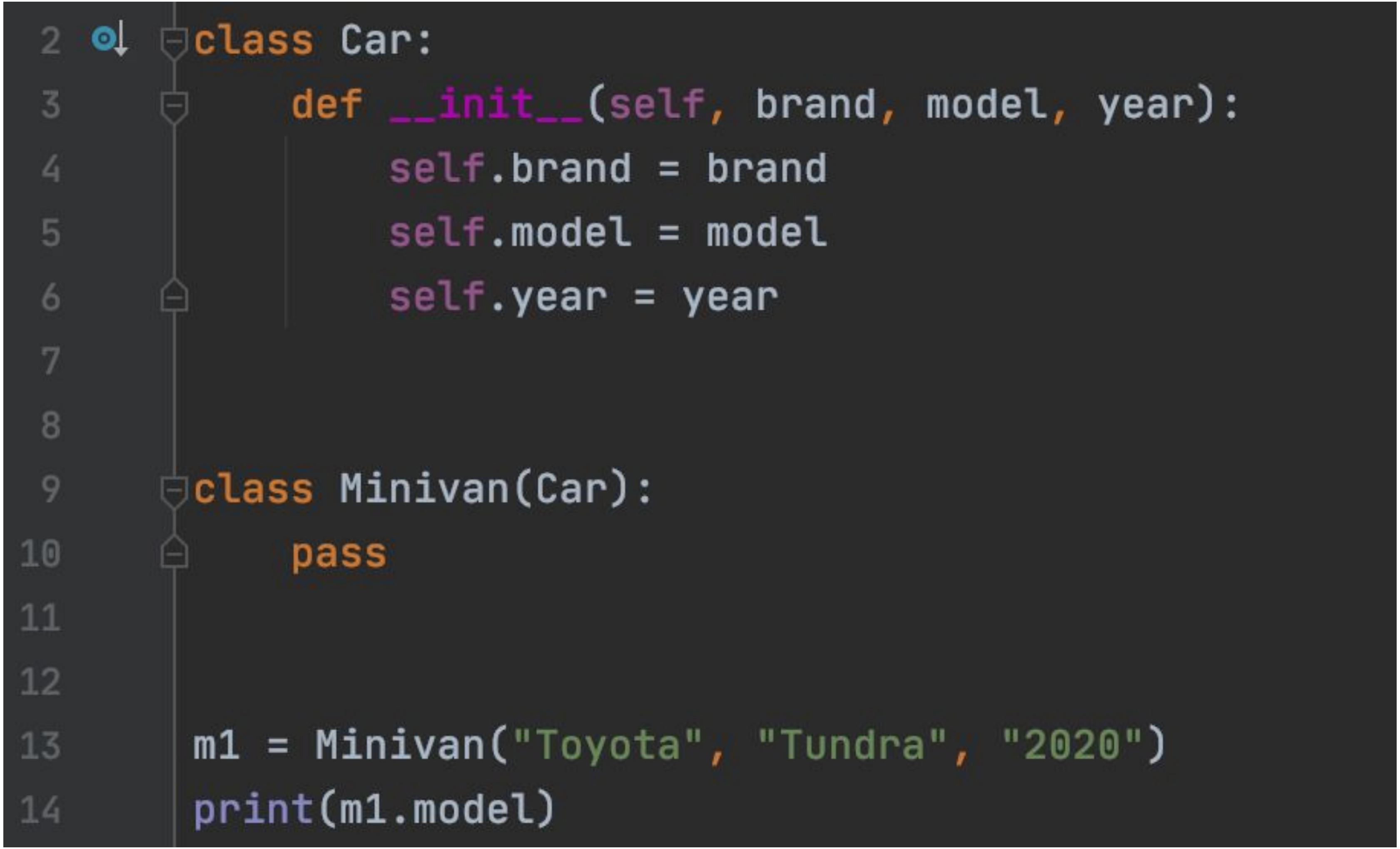
Pickuptruck class



Minivan class

Parent class and Child class

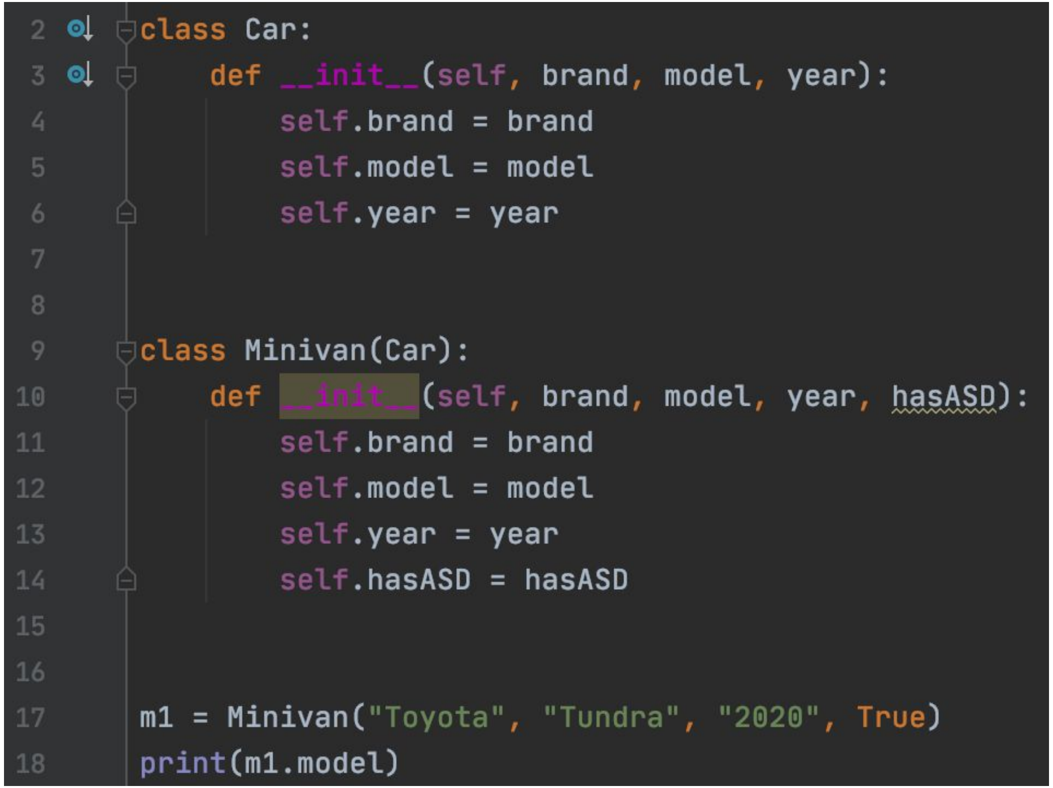
```
class Car:
    pass


class MiniVan(Car):
    pass
```



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


class Minivan(Car):
    pass


m1 = Minivan("Toyota", "Tundra", "2020")
print(m1.model)
```

Override

Inherited methods can be redefined in child class.

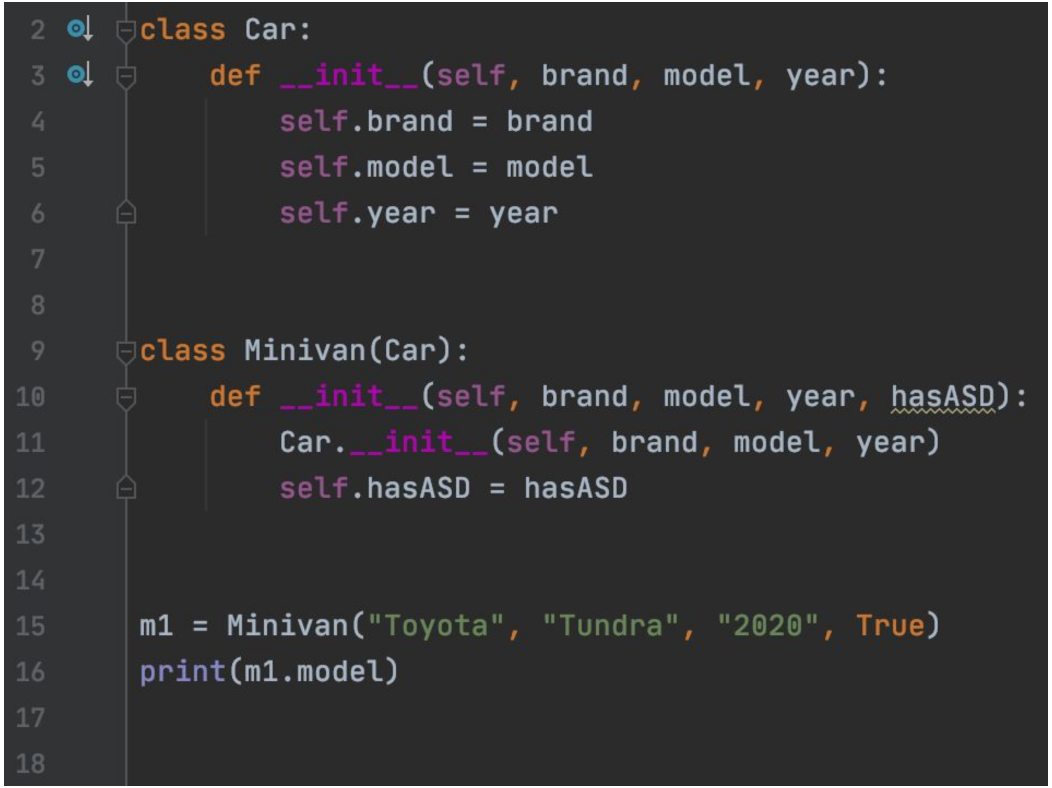


```python
class Car:
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year


class Minivan(Car):
    def __init__(self, brand, model, year, hasASD):
        self.brand = brand
        self.model = model
        self.year = year
        self.hasASD = hasASD


m1 = Minivan("Toyota", "Tundra", "2020", True)
print(m1.model)
```

Override

Inherited methods can be redefined in child class.

If you want to reuse the parent’s method too, you can put it under the overridden method.



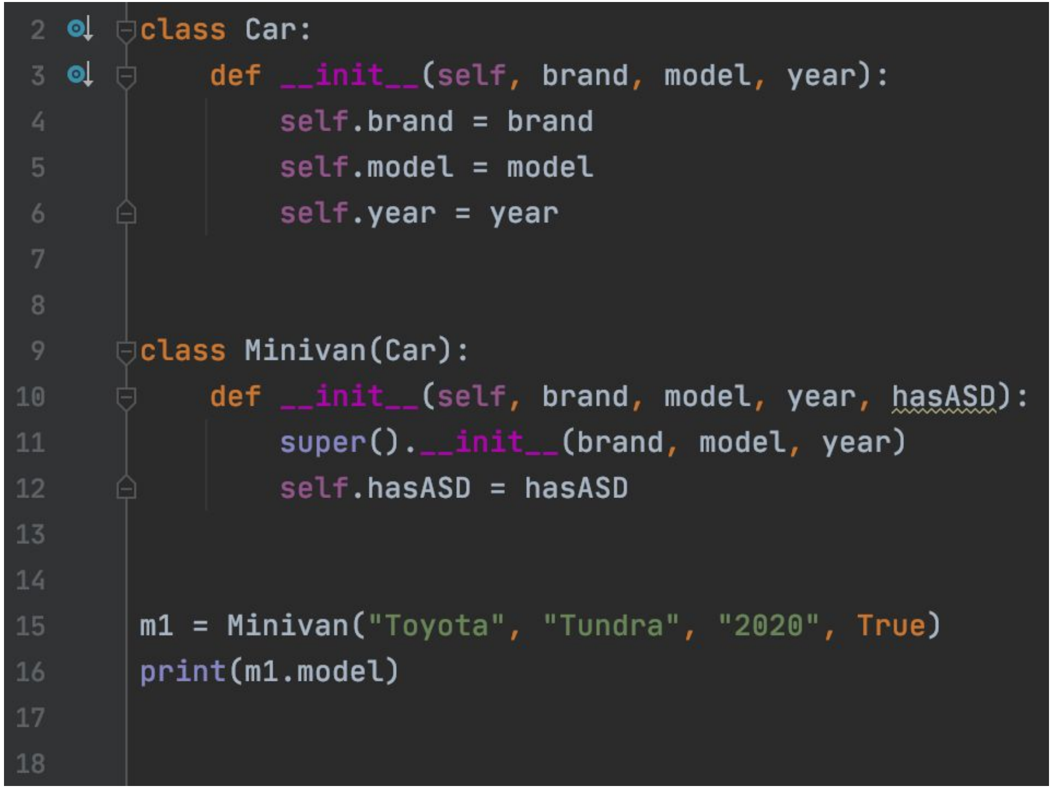
```python
class Car:
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year


class Minivan(Car):
    def __init__(self, brand, model, year, hasASD):
        Car.__init__(self, brand, model, year)
        self.hasASD = hasASD


m1 = Minivan("Toyota", "Tundra", "2020", True)
print(m1.model)
```

Override

Inherited methods can be redefined in child class.

If you want to reuse the parent’s method too, you can put it under the overridden method.

Instead of the parent’s class, you can use super() function.



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


class Minivan(Car):
    def __init__(self, brand, model, year, hasASD):
        super().__init__(brand, model, year)
        self.hasASD = hasASD


m1 = Minivan("Toyota", "Tundra", "2020", True)
print(m1.model)
```

Lab 12

Defining a Class:
Define a class named Car. This class should have an __init__ method that initializes three attributes: brand, model, and year.

Creating an Object:
Create an object of the Car class and initialize it with brand, model, and year. Print out each attribute individually.

Extending the Class with Inheritance:
Define a subclass of Car named Minivan. This subclass should have its own __init__ method that adds an additional attribute has_auto_sliding_door or hasASD (a boolean to indicate if the minivan has an auto sliding door). Use the super() function to inherit the __init__ method of the Car class.

Creating an Object of the Subclass:
Create an object of the Minivan class, initialize it with appropriate values including the has_auto_sliding_door attribute, and print out each attribute.

Hint: See the previous slide