Custom Types

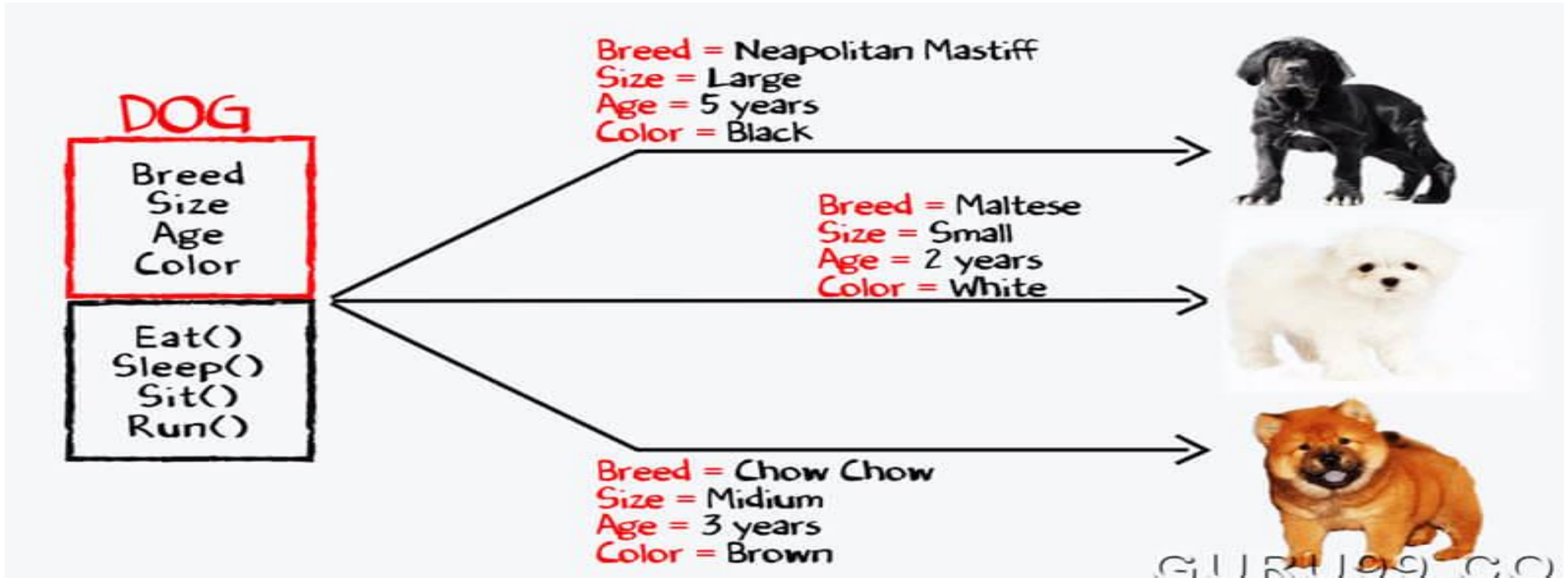
Introduction

- Built-in data types
  - Simple : integers, floating-point numbers, and Booleans
  - Complex : lists, tuples, dictionaries, sets

- Custom data types
  - Python provides the ability to design custom types which more closely model the problem at hand

Introduction

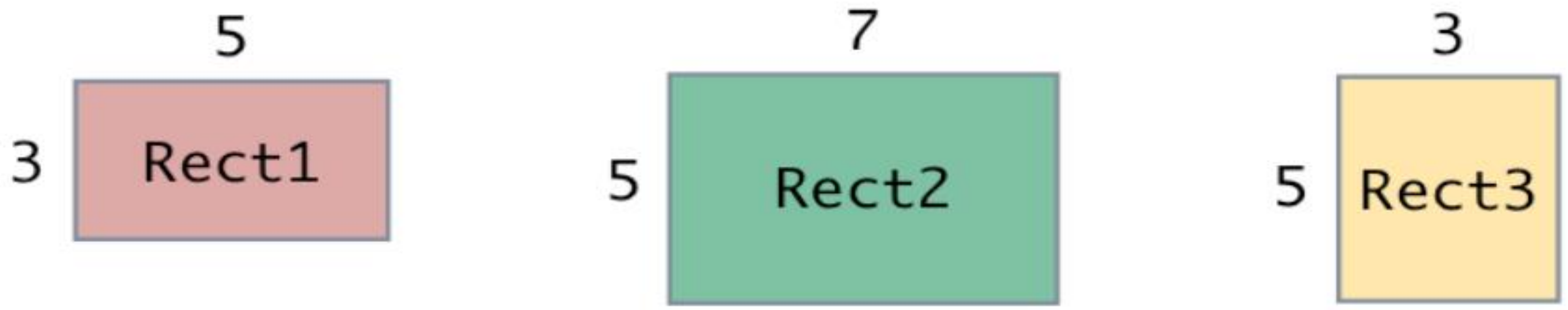
- A software object generally contains:
  - data (instance variables or attributes)
  - functionality (methods)
- The instance variables (attributes) and methods of an object comprise its members.
- The class of an object defines the object's basic structure and capabilities.



Example

- width
- height
- colour

- area()



```
width=5
height=3
colour=(200,0,0)
area() -> 15
```

```
width=7
height=5
colour=(0,200,0)
area() -> 35
```

```
width=3
height=5
colour=(255,200,0)
area() -> 15
```

General form of a class definition

```
class name :
    block
```

Example: car

```python
class Car():
 def __init__(self, mk, md, yr):
  self.make = mk
  self.model = md
  self.year = yr

 def get_descriptive_name(self):
  long_name = str(self.year) + ' ' + self.make + ' ' + self.model
  return long_name.title()

my_new_car = Car('audi', 'a4', 2016)
print(my_new_car.get_descriptive_name())
```

```
    Type "help", "copyright", "credits" or "license" for more information.
>>> [evaluate car.py]
    2016 Audi A4
>>>
```

Constructor: __init__()

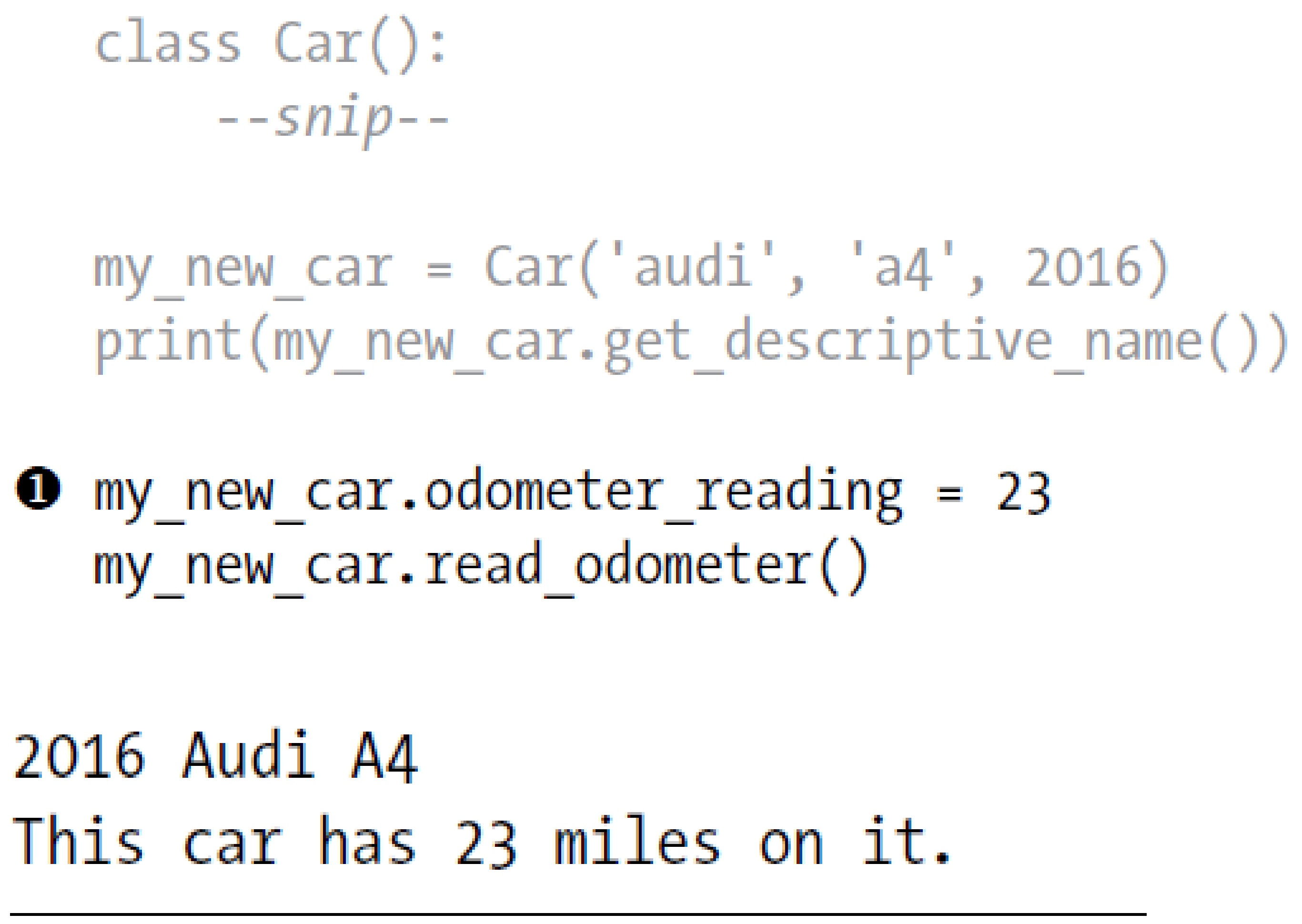
The __init__ method is run as soon as an object of a class is instantiated. Its aim is to initialize the object.

Example: car

```python
class Car():
 def __init__(self, mk, md, yr):
  self.make = mk
  self.model = md
  self.year = yr
  self.odometer_reading = 0

 def get_descriptive_name(self):
  long_name = str(self.year) + ' ' + self.make + ' ' + self.model
  return long_name.title()

 def read_odometer(self):
  print("This car has " + str(self.odometer_reading) + " miles on it.")

my_new_car = Car('audi', 'a4', 2016)
print(my_new_car.get_descriptive_name())
my_new_car.read_odometer()
```

```
>>> [evaluate car.py]
    2016 Audi A4
    This car has 0 miles on it.
```

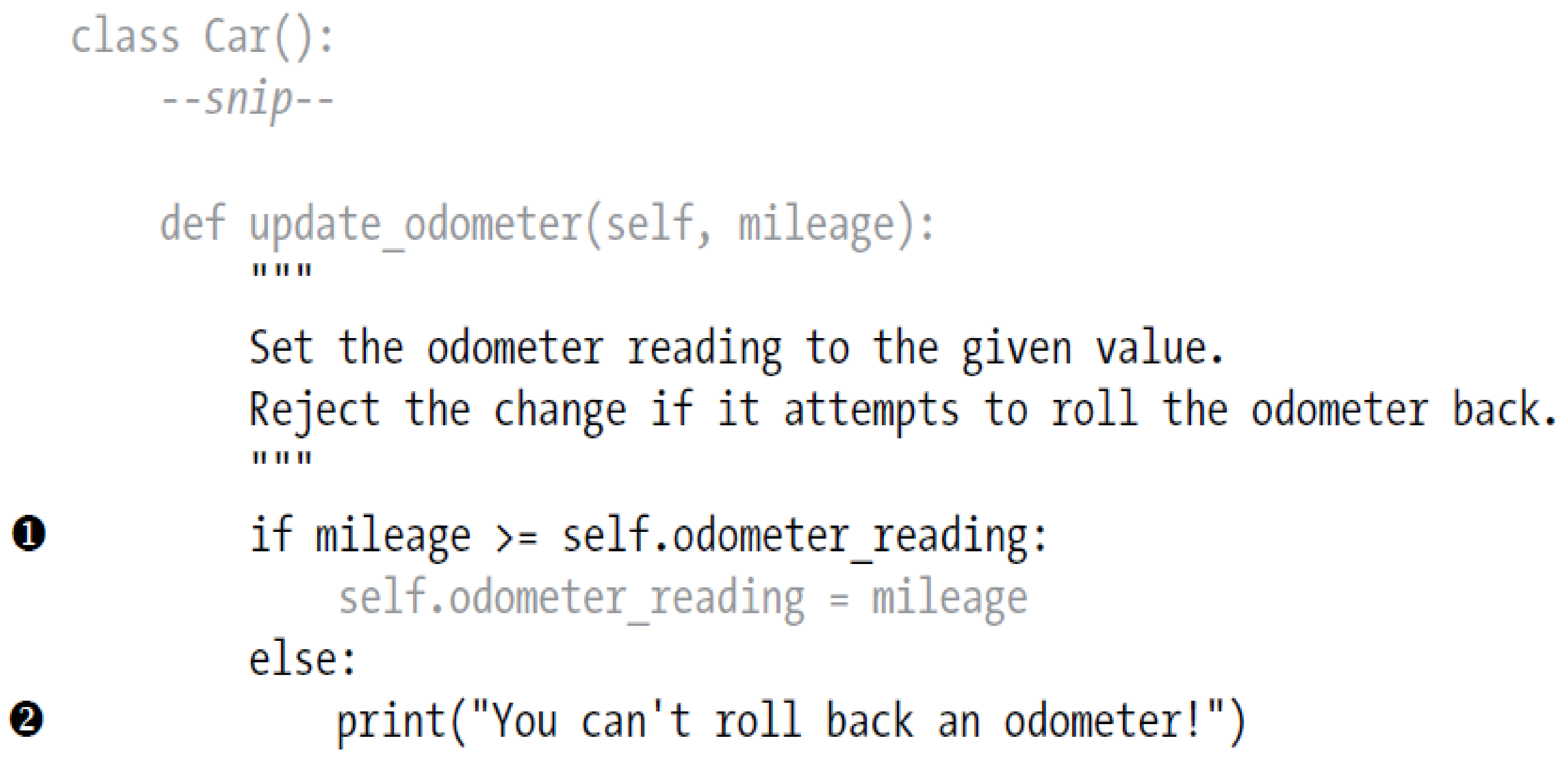
Example: car



```
  class Car():
      --snip--

  my_new_car = Car('audi', 'a4', 2016)
  print(my_new_car.get_descriptive_name())

❶ my_new_car.odometer_reading = 23
  my_new_car.read_odometer()
```

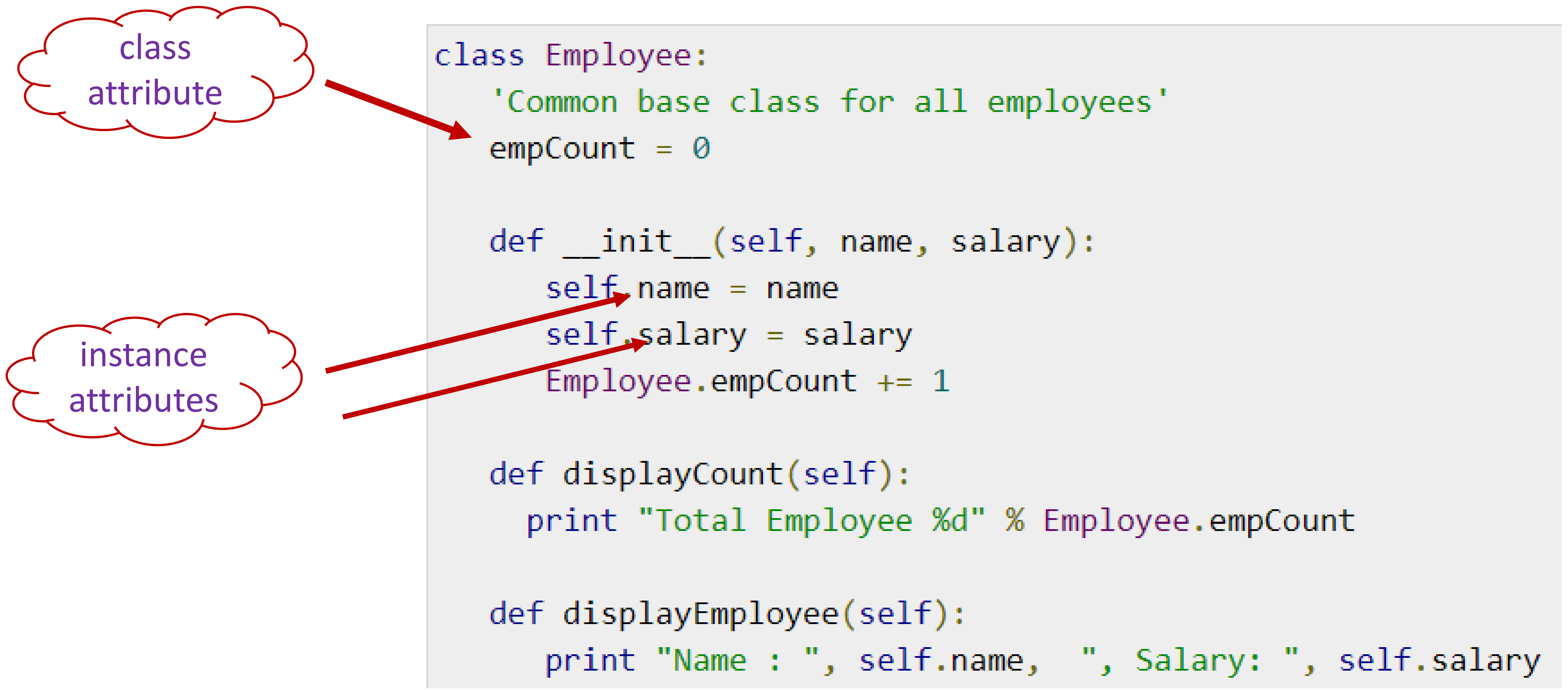
```
2016 Audi A4
This car has 23 miles on it.
```

Example: car



```
    class Car():
        --snip--

        def update_odometer(self, mileage):
            """
            Set the odometer reading to the given value.
            Reject the change if it attempts to roll the odometer back.
            """
❶           if mileage >= self.odometer_reading:
                self.odometer_reading = mileage
            else:
❷               print("You can't roll back an odometer!")
```

Example :employee



```
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print "Total Employee %d" % Employee.empCount

   def displayEmployee(self):
      print "Name : ", self.name,  ", Salary: ", self.salary
```

Example:employee

```
"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount
```

When the above code is executed, it produces the following result −

```
Name :  Zara ,Salary:  2000
Name :  Manni ,Salary:  5000
Total Employee 2
```

Example: employee

You can add, remove, or modify attributes of classes and objects at any time –

```
emp1.age = 7  # Add an 'age' attribute.
emp1.age = 8  # Modify 'age' attribute.
del emp1.age  # Delete 'age' attribute.
```

Instead of using the normal statements to access attributes, you can use the following functions –

- The **getattr(obj, name[, default])** – to access the attribute of object.
- The **hasattr(obj,name)** – to check if an attribute exists or not.
- The **setattr(obj,name,value)** – to set an attribute. If attribute does not exist, then it would be created.
- The **delattr(obj, name)** – to delete an attribute.

```
hasattr(emp1, 'age')    # Returns true if 'age' attribute exists
getattr(emp1, 'age')    # Returns value of 'age' attribute
setattr(emp1, 'age', 8) # Set attribute 'age' at 8
delattr(empl, 'age')    # Delete attribute 'age'
```