Minia University
Faculty of Computers & information

Artificial Neural Networks and Deep Learning

Slides By:

T.A. Sarah Osama Talaat

✉ E-mail: SarahOsama.fci@gmail.com

Slides were prepared based on set of references mentioned in the last slide

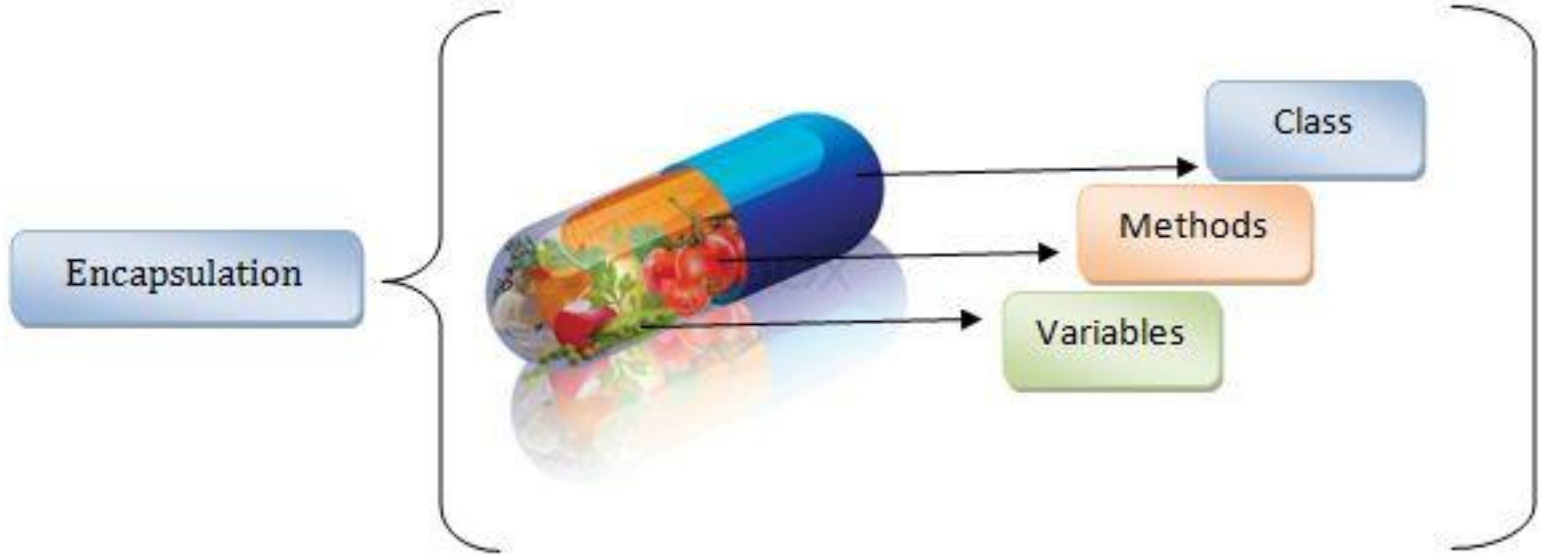
Agenda

- ❑Object Oriented Programming
  - ❑Terminologies
  - ❑Creating the class
  - ❑Creating the instances
  - ❑Inheritance

Let’s Start

Object Oriented Programming:
Terminologies

- ***Encapsulation*** is an Object Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse.



Object Oriented Programming:
Terminologies

- ❑ *Class:* a **user-defined prototype** for an object that defines a set of **attributes** that characterize any object of the class. The attributes are data members (class variables and instance variables) and **methods**, accessed via **dot notation.**

- ❑ *Class variable:* a variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods.

Object Oriented Programming:

Terminologies

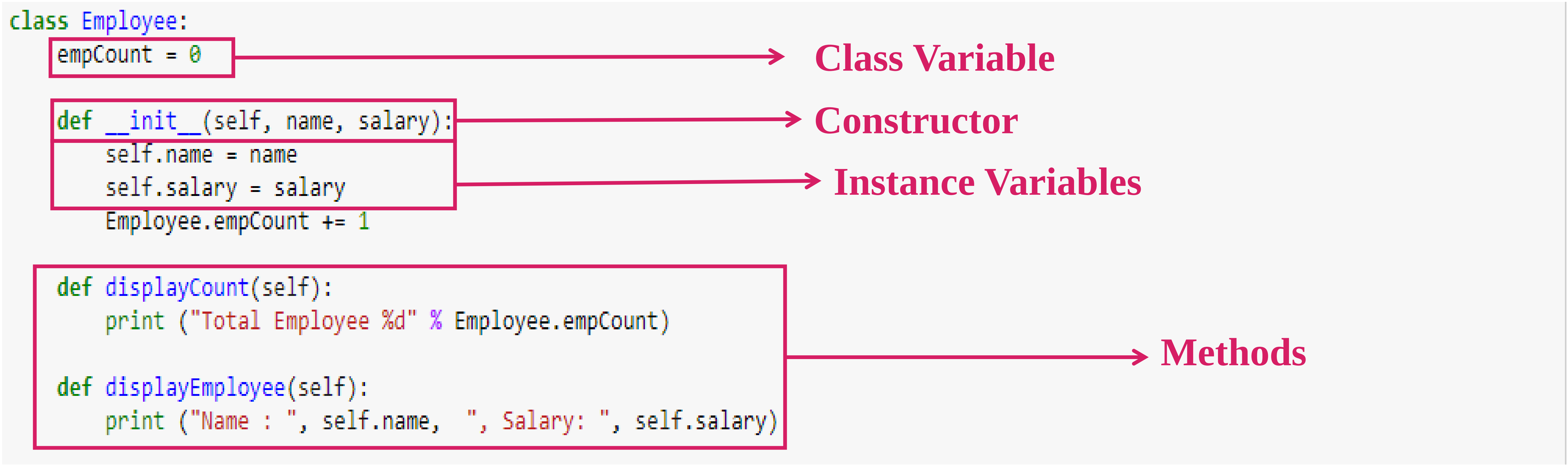
- ***Instance:*** an individual object of a certain class.
- ***Instance variable:*** a variable that is defined inside a method and belongs only to the current instance of a class.
- ***Inheritance:*** the transfer of the characteristics of a class (***parent class***) to other classes that are derived from it (***children classes***).

Object Oriented Programming:
Creating The Class

- **Syntax:**

```
class ClassName:
    class_suite
```

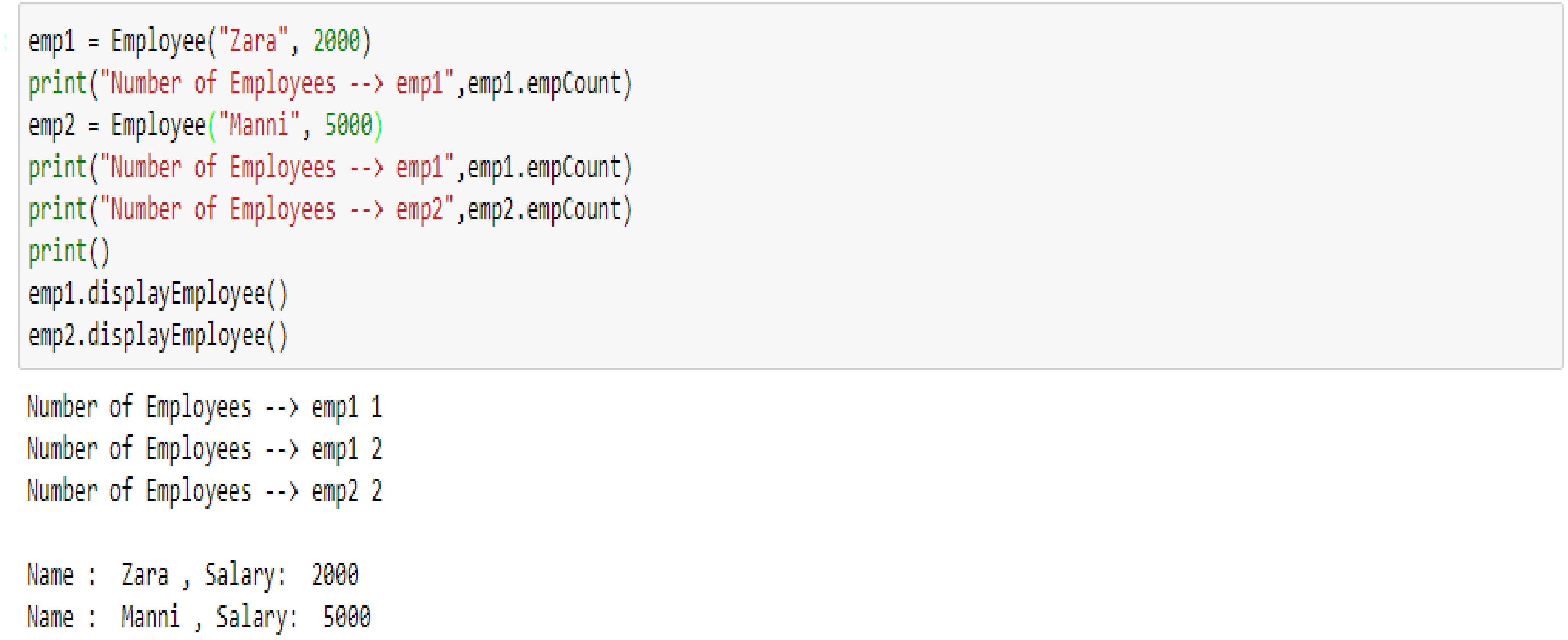
- **Example:**



```
class Employee:
    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)
```

Object Oriented Programming: Creating The Instances

❑Creating Instances:

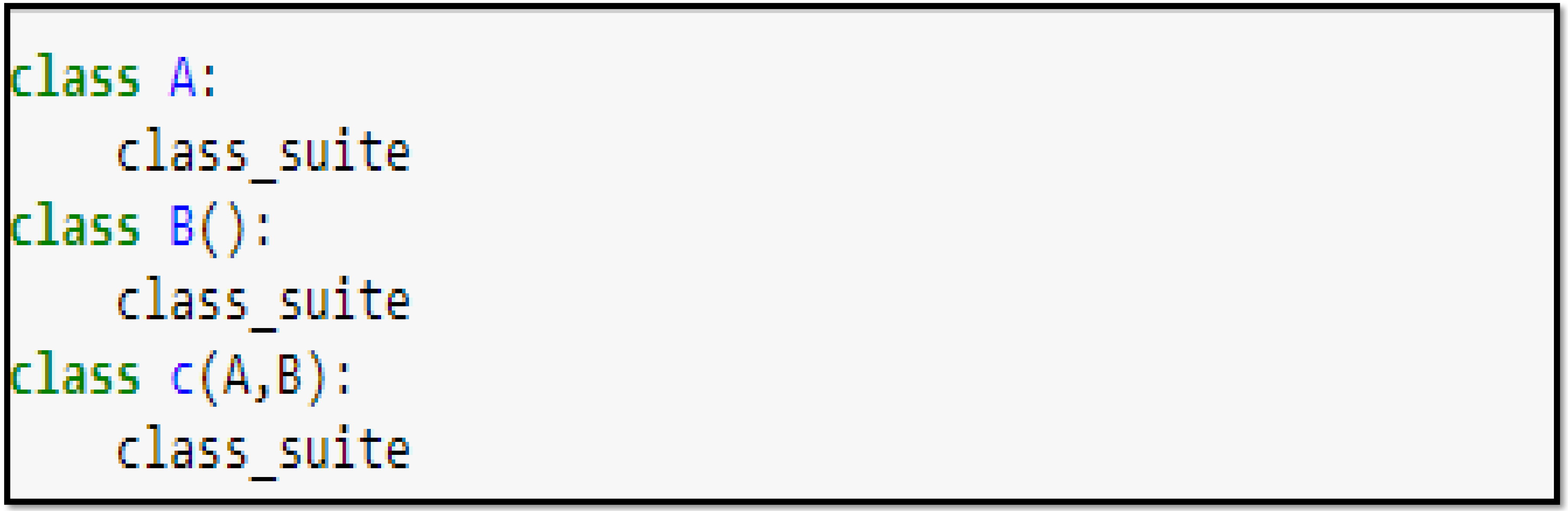


```
emp1 = Employee("Zara", 2000)
print("Number of Employees --> emp1",emp1.empCount)
emp2 = Employee("Manni", 5000)
print("Number of Employees --> emp1",emp1.empCount)
print("Number of Employees --> emp2",emp2.empCount)
print()
emp1.displayEmployee()
emp2.displayEmployee()
```

```
Number of Employees --> emp1 1
Number of Employees --> emp1 2
Number of Employees --> emp2 2

Name :  Zara , Salary:  2000
Name :  Manni , Salary:  5000
```

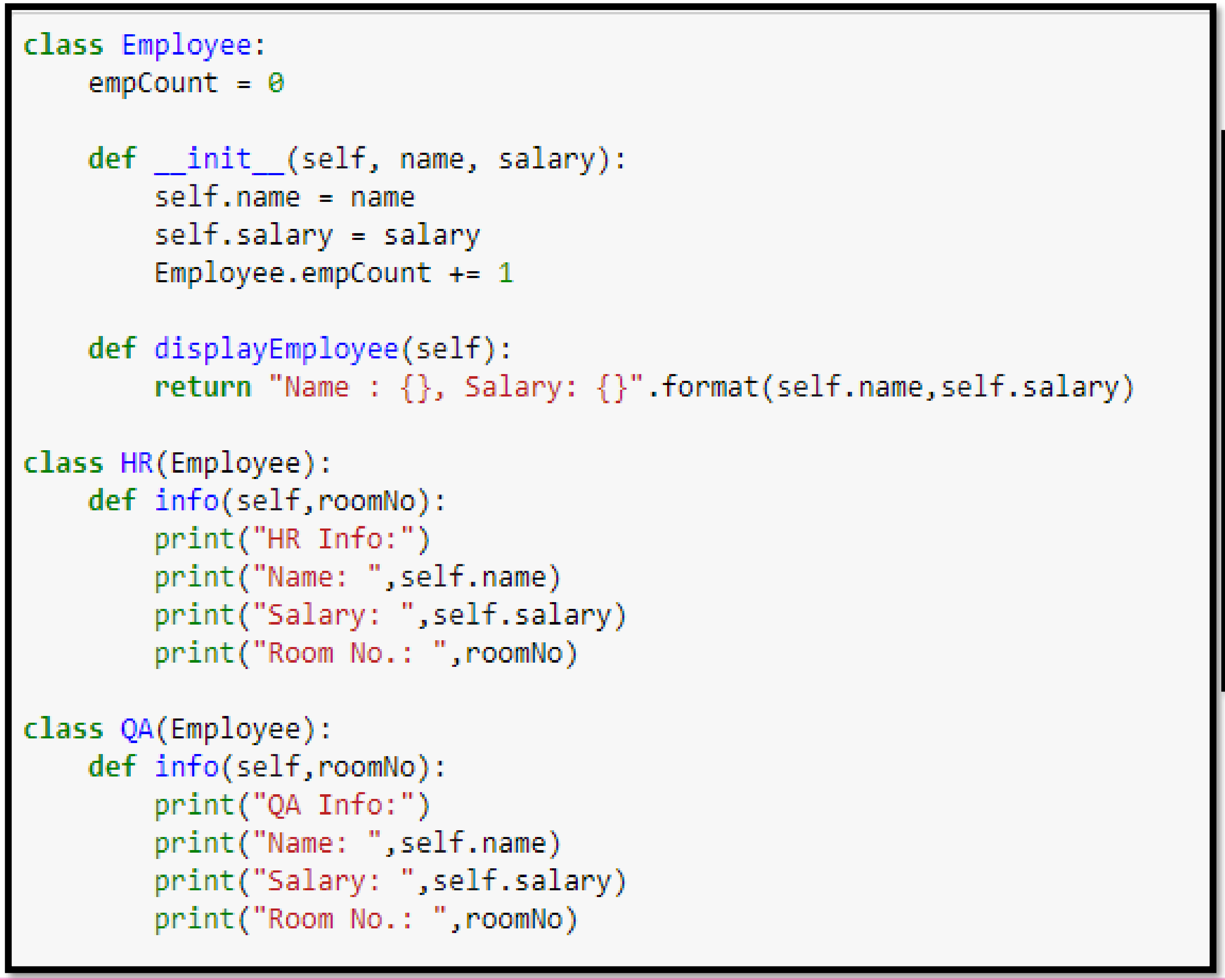
Object Oriented Programming:
Inheritance

❑Syntax:

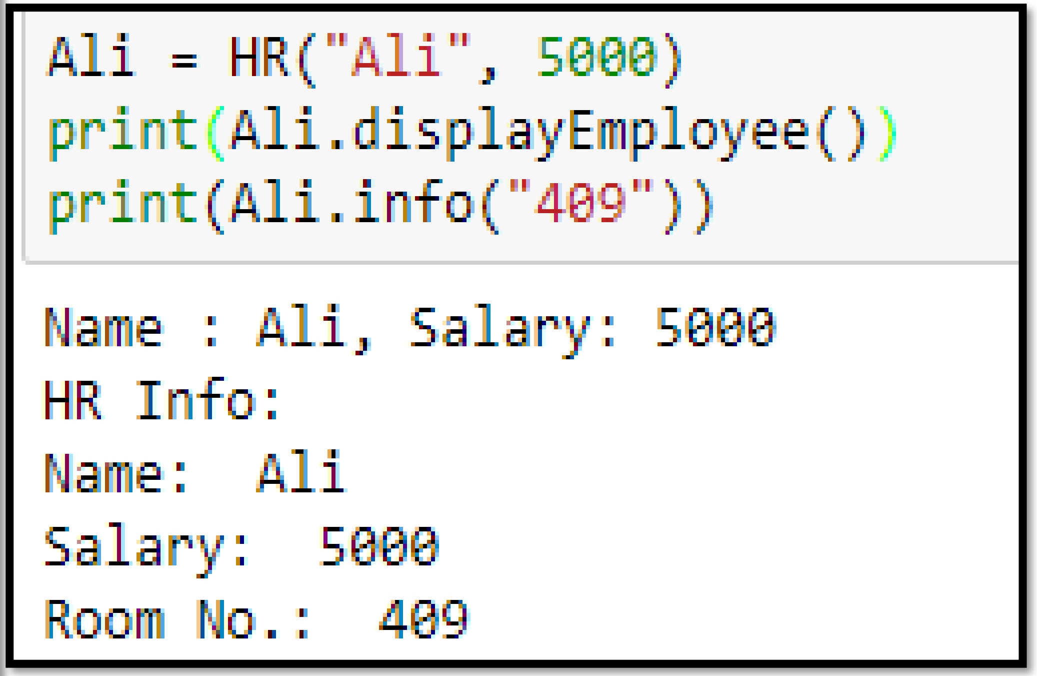


```
class A:
    class_suite
class B():
    class_suite
class c(A,B):
    class_suite
```

Object Oriented Programming: Inheritance



```
class Employee:
    empCount = 0

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

    def displayEmployee(self):
        return "Name : {}, Salary: {}".format(self.name,self.salary)

class HR(Employee):
    def info(self,roomNo):
        print("HR Info:")
        print("Name: ",self.name)
        print("Salary: ",self.salary)
        print("Room No.: ",roomNo)

class QA(Employee):
    def info(self,roomNo):
        print("QA Info:")
        print("Name: ",self.name)
        print("Salary: ",self.salary)
        print("Room No.: ",roomNo)
```

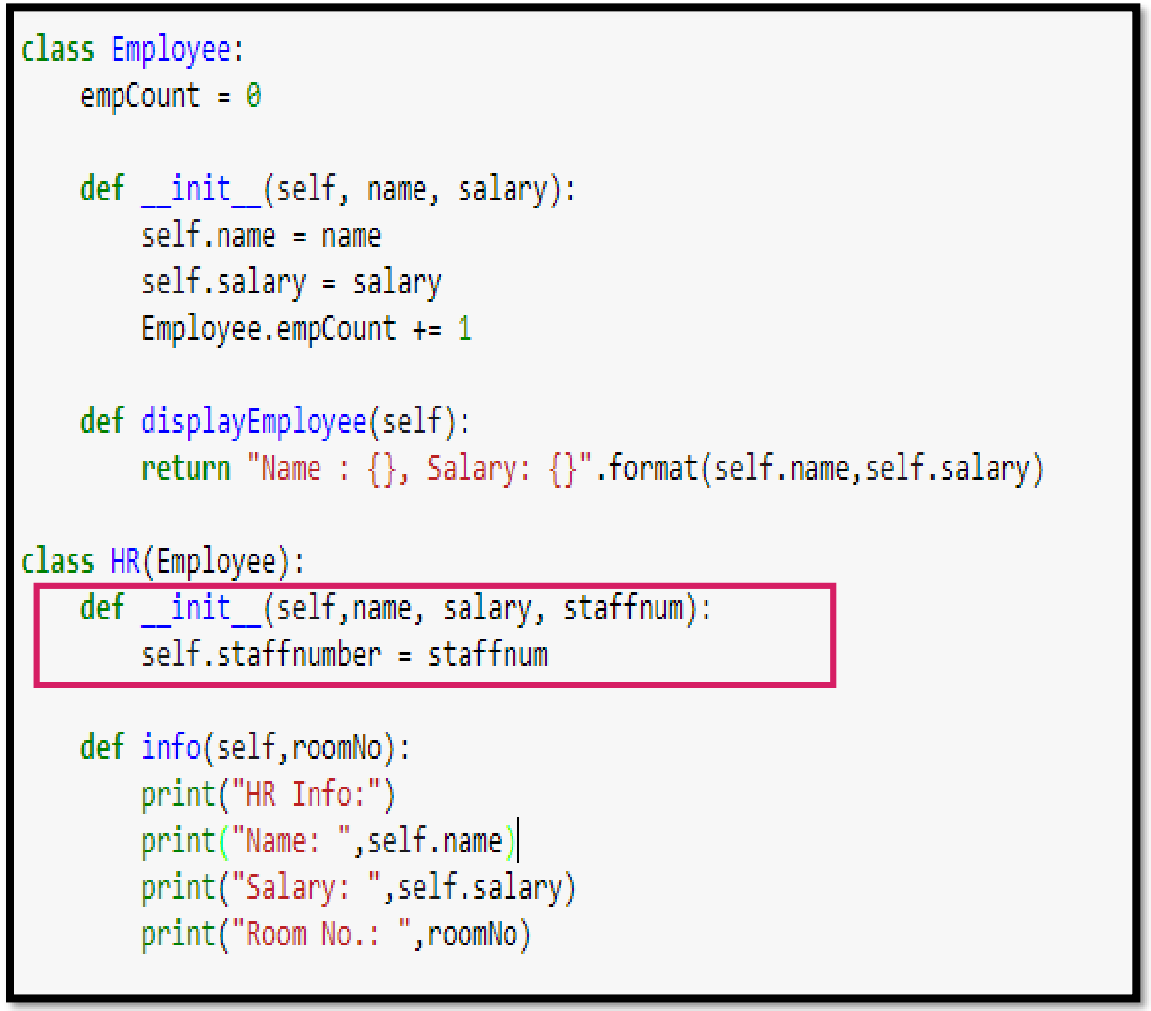


```
Ali = HR("Ali", 5000)
print(Ali.displayEmployee())
print(Ali.info("409"))
```

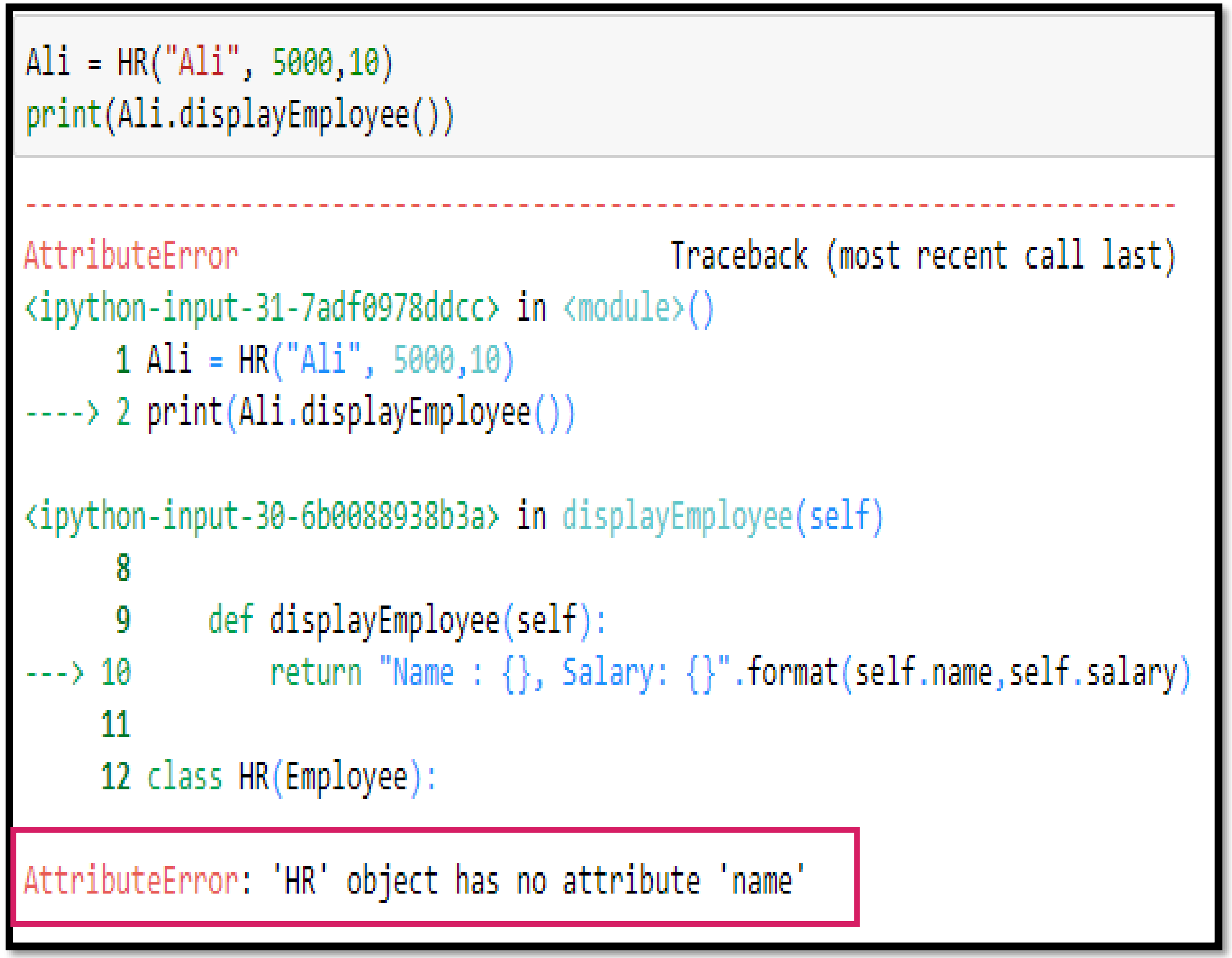
```
Name : Ali, Salary: 5000
HR Info:
Name:  Ali
Salary:  5000
Room No.:  409
```

Object Oriented Programming: Inheritance

❑Constructor:



```
class Employee:
    empCount = 0

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

    def displayEmployee(self):
        return "Name : {}, Salary: {}".format(self.name,self.salary)

class HR(Employee):
    def __init__(self,name, salary, staffnum):
        self.staffnumber = staffnum

    def info(self,roomNo):
        print("HR Info:")
        print("Name: ",self.name)
        print("Salary: ",self.salary)
        print("Room No.: ",roomNo)
```

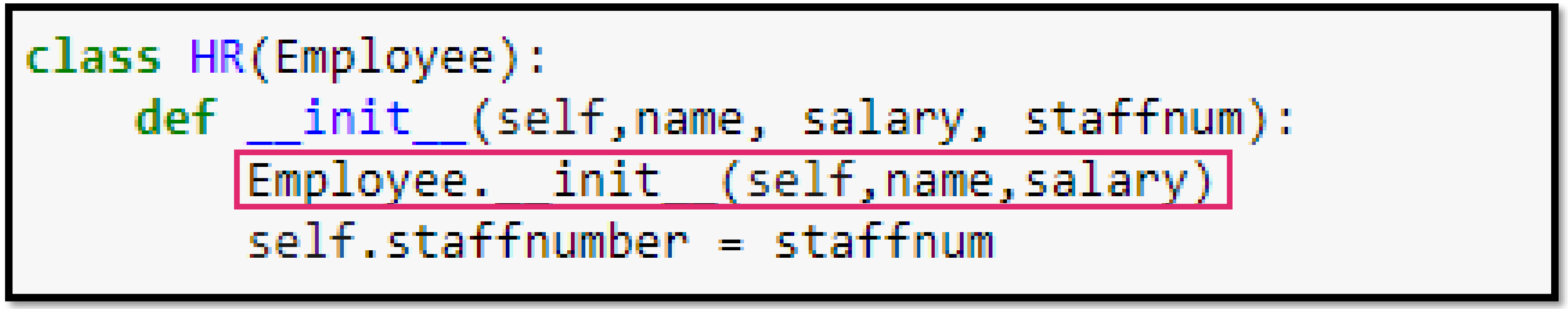


```
Ali = HR("Ali", 5000,10)
print(Ali.displayEmployee())
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-31-7adf0978ddcc> in <module>()
      1 Ali = HR("Ali", 5000,10)
----> 2 print(Ali.displayEmployee())

<ipython-input-30-6b0088938b3a> in displayEmployee(self)
      8
      9     def displayEmployee(self):
---> 10         return "Name : {}, Salary: {}".format(self.name,self.salary)
     11
     12 class HR(Employee):

AttributeError: 'HR' object has no attribute 'name'
```

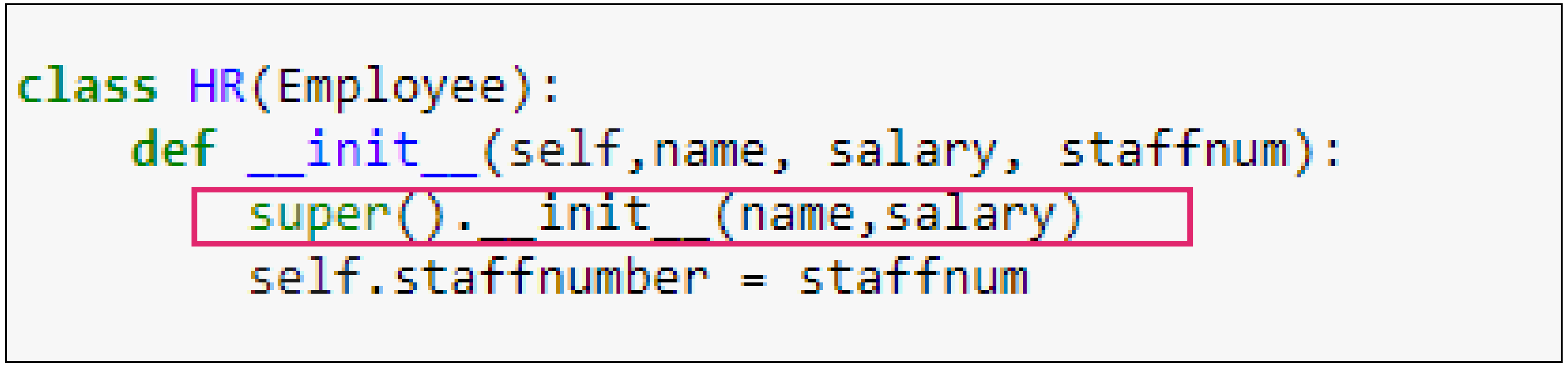
Object Oriented Programming:
Inheritance

❑Constructor:



```
class HR(Employee):
    def __init__(self,name, salary, staffnum):
        Employee.__init__(self,name,salary)
        self.staffnumber = staffnum
```

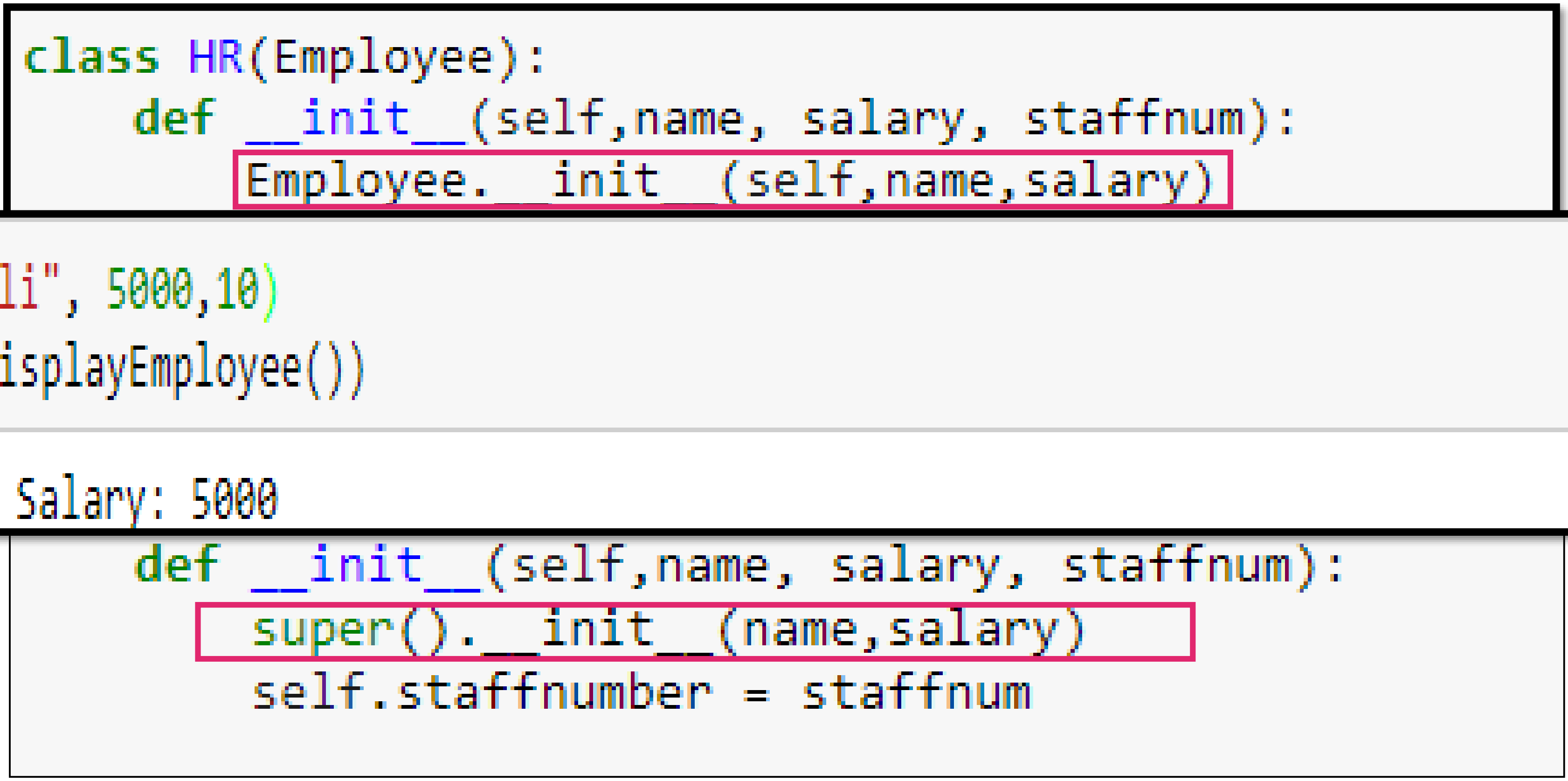
Or



```
class HR(Employee):
    def __init__(self,name, salary, staffnum):
        super().__init__(name,salary)
        self.staffnumber = staffnum
```

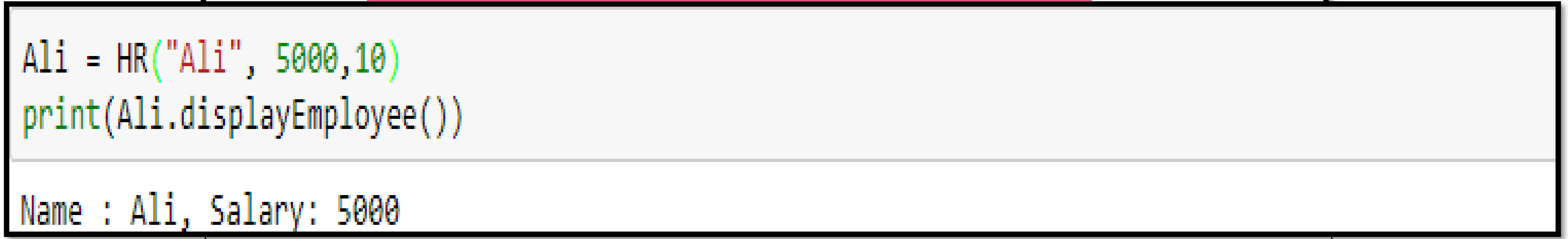
Object Oriented Programming:
Inheritance

❑Constructor:



```
class HR(Employee):
    def __init__(self,name, salary, staffnum):
        Employee.__init__(self,name,salary)
```

```
    def __init__(self,name, salary, staffnum):
        super().__init__(name,salary)
        self.staffnumber = staffnum
```



```
Ali = HR("Ali", 5000,10)
print(Ali.displayEmployee())
```

```
Name : Ali, Salary: 5000
```

Object Oriented Programming:
Inheritance

- **Access Modifiers:** Python has no privacy model, there are no access modifiers like in C++, C# or Java.

~~Public~~
~~Protected~~
~~Private~~

Object Oriented Programming: Inheritance

❑Destructor:

- Destructors are a very important concept in C++. But in Python, destructors are **needed much less**, because Python has a **garbage collector** that handles memory management. However, while memory is the **most common resource allocated,** it is **not the only one**. There are also **sockets** and database **connections** to be closed, files, buffers and caches flushed and a few more resources that need to be released when an object is done with them.

Object Oriented Programming: Inheritance

```
class Employee:
    empCount = 0

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

    def displayEmployee(self):
        return "Name : {}, Salary: {}".format(self.name,self.salary)

    def __del__(self):
        Employee.empCount -= 1
```

```
emp1=Employee('Ali',5000)
emp2=Employee('Khaled',9000)
emp3=Employee('Sara',12000)
```

```
print("Number of Employee = ", emp1.empCount)
```

```
Number of Employee =  3
```

```
del emp2
print("Number of Employee = ", emp1.empCount)
```

```
Number of Employee =  2
```

References

- https://realpython.com/blog/python/python3-object-oriented-programming/#python-object-inheritance

Any Questions!?

Thank you