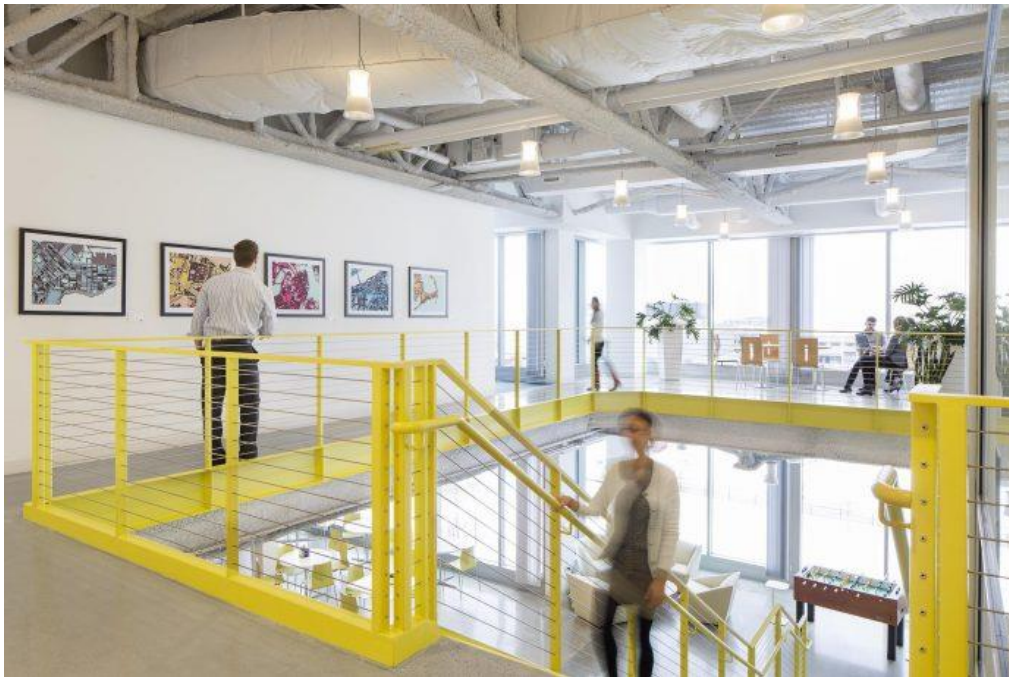


DR. ALVIN'S PUBLICATIONS

UPDATING STAFF RECORDS USING OBJECT ORIENTED PROGRAMMING

WITH PYTHON
DR. ALVIN ANG



1 | PAGE

COPYRIGHTED BY DR ALVIN ANG
WWW.ALVINANG.SG

CONTENTS

I. OOP is Unpopular.....	3
II. Chapter 0: Understanding Instance / Init / Self	4
A. Creating the Parent Class	5
B. An Object is an Instance of a Class	5
C. Calling Out the 'Instance'	6
D. Calling Out the 'Class'.....	6
III. Chapter 1: Using only 1 Parent Class (Full Time Employees Only)	7
A. Creating the Employee Class.....	8
B. Inputting 3 Employees with their Salaries	10
C. Displaying the 3 Employees Names and Salaries	11
D. Counting the Current Number of Employees.....	12
E. Deleting away Employee 1	13
F. Recounting the Current Number of Employees	13
IV. Chapter 2: Using 1 Parent Class (Employee) and 2 Child Classes (Full time vs Part Time) 14	
A. Creating the Employee Parent Class.....	15
B. Creating the FT Child Class.....	16
C. Creating the PT Child Class	17
D. Keying in 2 FT Employees and 1 PT Employee.....	18
E. Displaying Employee 1 Attributes	18
F. Displaying FT Count.....	19
G. Displaying PT Rate	19
H. Delete Ally Away.....	20
I. Delete Belinda Away	20
J. How Many Employees Left?	21
About Dr. Alvin Ang	22

I. OOP IS UNPOPULAR

- No, object-oriented programming (OOP) is not dead.
- But it is significantly less ubiquitous than it used to be.
- <https://medium.com/machine-words/the-rise-and-fall-of-object-oriented-programming-d67078f970e2#:~:text=No%2C%20object%2Doriented%20programming%20>

II. CHAPTER 0: UNDERSTANDING INSTANCE / INIT / SELF

https://www.alvinang.sg/s/Updating_HR_Records_using_Object_Oriented_Programming_by_Dr_Alvin_Ang_Ch_0_Understanding_Instance_In.ipynb

Chapter 0: Understanding Instance / Init / Self

Let's say you have a class `ClassA` which contains a method `methodA` defined as:

```
def methodA(self, arg1, arg2):  
    # do something
```

and `ObjectA` is an instance of this class.

Now when `ObjectA.methodA(arg1, arg2)` is called, python internally converts it for you as:

```
ClassA.methodA(ObjectA, arg1, arg2)
```

The `self` variable refers to the object itself.

<https://stackoverflow.com/questions/2709821/what-is-the-self-parameter-in-class-methods>

A. CREATING THE PARENT CLASS

```
class Animal:

    color = 'white'
    legs = 4

#color and legs are Class Variables

    def __init__(self,color,legs):
        self.color = color
        self.legs = legs

# self.color and self.legs are Instance Variables

    def instanceType(self):
        print('This {} animal has {} legs'.format(self.color,self.legs))

#notice that the 'self.' points back to the 'Instance'

    def classType(self):
        print('This {} animal has {} legs'.format(Animal.color,Animal.legs))

#notice that the 'Animal.' points back to the 'Class'
```

B. AN OBJECT IS AN INSTANCE OF A CLASS

A) An Object is an Instance of a Class

```
[ ] Alvin = Animal('HUMAN', 3)

# 'Alvin' is now an Object = an Instance (child) of the Class 'Animal' (parent)
```

C. CALLING OUT THE 'INSTANCE'

B) Calling Out the 'Instance'

```
[ ] Alvin.instanceType()
```

```
This HUMAN animal has 3 legs
```

D. CALLING OUT THE 'CLASS'

C) Calling Out the 'Class'

```
[ ] Alvin.classType()
```

```
This white animal has 4 legs
```

THE END

https://www.alvinang.sg/s/Updating_HR_Records_using_Object_Oriented_Programming_by_Dr_Alvin_Ang_Ch_1_Using_only_1_Parent_Class.ipynb

Understanding The Concept of 'Class' vs 'Instance' Variables

Chapter 1: Understanding 'Class' vs 'Instance' variables

- 'Class' Variables == Global Variables
- 'Instance' Variables == Local Variables

A. CREATING THE EMPLOYEE CLASS

1a) Creating the Employee Class

- 1st Function: def **init**
 - is needed for inputs
 - it only deals with *Class Variables* because

```
Employee.empCount is a Class Variable
```

- 2nd Function: def **del**
 - is needed to delete Employees away
 - it only deals with *Class Variables* because

```
Employee.empCount is a Class Variable
```

- 3rd Function: def **Display**
 - is needed to display Employees Name and Salary
 - it only deals with *Instance Variables* because

```
self.name, self.salary are Instant Variables
```

- 4th Function: def **Count**
 - is needed to Count the current number of Employees
 - it only deals with *Class Variable* because

```
Employee.empCount is a Class Variable
```



```

class Employee:
    empCount = 0 #Class Variable

#-----
#1st function: Init
#-----
    def __init__(self,name,salary):
        self.name = name #Instance Variable
        self.salary = salary #Instance Variable
        Employee.empCount = Employee.empCount+1 #Class Variable

#-----
#2nd function: Del
#-----
    def __del__(self):
        Employee.empCount = Employee.empCount-1

#-----
#3rd function: Display
#-----
    def Display(self):
        print("{} salary is {}".format(self.name,self.salary))

        #self.name and self.salary are Instance Variables

#-----
#4th function: Count
#-----
    def Count(self):
        print("Number of Employee is {}".format(Employee.empCount))

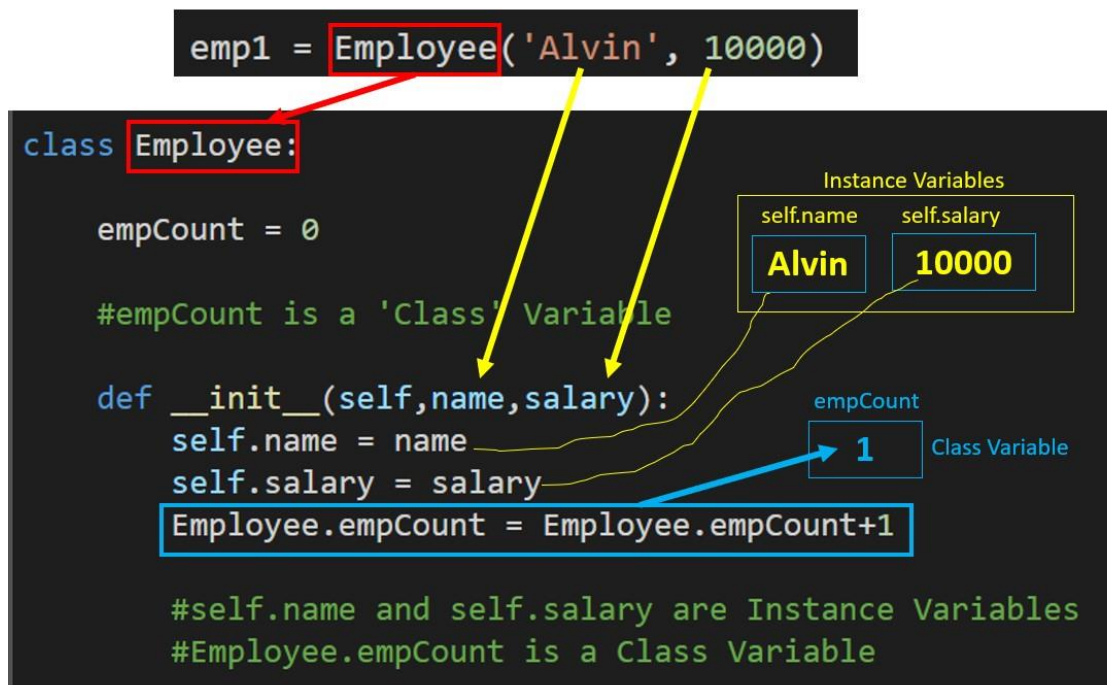
        #Employee.empCount is a Class Variable

```

B. INPUTTING 3 EMPLOYEES WITH THEIR SALARIES

1b) Inputting 3 Employees with their Salaries

```
[14] emp1 = Employee('Alvin', 10000)
      emp2 = Employee('Patrick', 6000)
      emp3 = Employee('Steven', 4000)
```

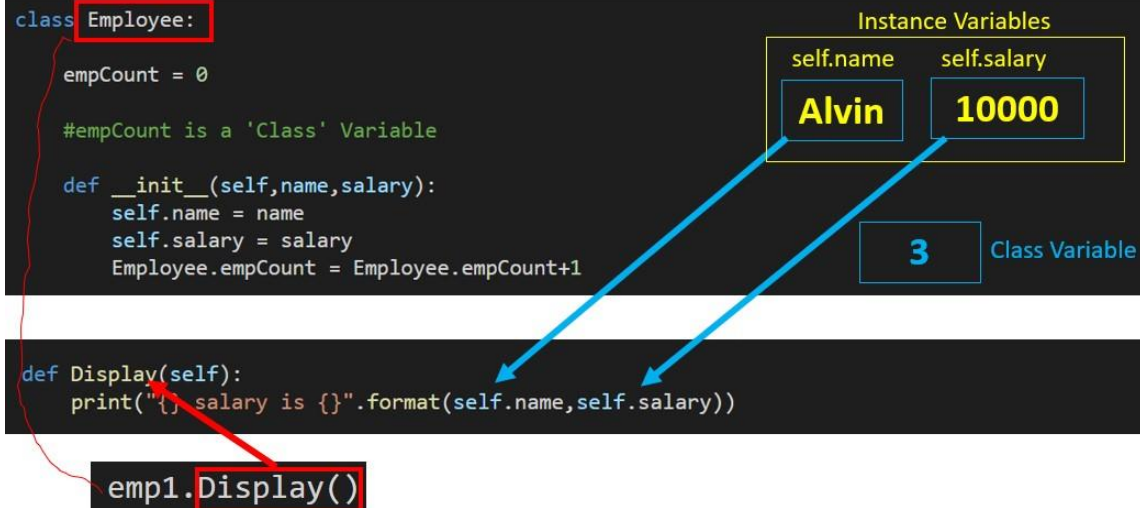


C. DISPLAYING THE 3 EMPLOYEES NAMES AND SALARIES

1c) Displaying the 3 Employees Names and Salaries

```
▶ emp1.Display()  
emp2.Display()  
emp3.Display()
```

```
↳ Alvin salary is 10000  
Patrick salary is 6000  
Steven salary is 4000
```



D. COUNTING THE CURRENT NUMBER OF EMPLOYEES

1d) Counting the Current Number of Employees

```
[16] emp1.Count()
```

```
#you are calling up a 'Class' Variable because  
#Employee.empCount is a Class Variable
```

```
Number of Employee is 3
```

E. DELETING AWAY EMPLOYEE 1

1e) Deleting away Employee 1

```
▶ del emp1
```

```
#you are calling up a 'Class' Variable because  
#Employee.empCount is a Class Variable
```

F. RECOUNTING THE CURRENT NUMBER OF EMPLOYEES

1f) Recounting the Current Number of Employees

```
[18] emp2.Count()
```

```
Number of Employee is 2
```

IV. CHAPTER 2: USING 1 PARENT CLASS (EMPLOYEE) AND 2 CHILD CLASSES (FULL TIME VS PART TIME)

https://www.alvinang.sg/s/Updating_HR_Records_using_Object_Oriented_Programming_by_Dr_Alvin_Ang_Ch_2_Using_1_Parent_Class_Empl.ipynb

Chapter 2: Using 1 Parent Class (Employee) and 2 Child Classes (Full Time vs Part Time Staff)

We want to:

- Create Employee PARENT CLASS
- Full Time Staff CHILD Class
- Full Time Staff has
 - Name
 - Salary
 - Leave
- Part Time Staff CHILD Class
- Part Time Staff has
 - Name
 - Hourly Rate
 - NO salary and leave attributes
- Staff Strength ONLY counts Full Time Staff

A. CREATING THE EMPLOYEE PARENT CLASS

2a) Creating the Employee PARENT Class

```
[1] class Employee:

    empCount = 0

    #-----
    #1st function: Init
    #-----

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

    #-----
    #2nd function: Del
    #-----

    def __del__(self):
        Employee.empCount = Employee.empCount-1
        print("{} has left the company".format(self.name))

    #-----
    #3rd function: Display Employee Count
    #-----

    def dispEmployeeCount(self):
        print("Number of Employee is {}".format(Employee.empCount))
```

B. CREATING THE FT CHILD CLASS

2b) Creating the FT Child Class

```
class FullTimeStaff(Employee):  
  
    #-----  
    #1st function: Init  
    #-----  
  
    def __init__(self,name,salary,leave):  
        super().__init__(name,salary)  
        self.leave = leave  
  
    #-----  
    #2nd function: Display Full Time Employee Attributes  
    #-----  
  
    def dispFTEmployee(self):  
        print("{} salary is {} and leave is {}".format(self.name,self.salary,self.leave))
```

- Note that the “Super Init” means we overwrite the Parent (Employee) class with values that will be input into the Child class here.

C. CREATING THE PT CHILD CLASS

2c) Creating the PT Child Class

```
class PartTimeStaff(Employee):  
#-----  
#1st function: Init  
#-----  
    def __init__(self,PTname,hrrate):  
        #super().__init__(name,0)  
        self.PTname = PTname  
        self.hrrate = hrrate  
  
#-----  
#2nd function: Display Part Time Employee Attributes  
#-----  
    def dispPTEmployee(self):  
        print("{} hourly rate is {}".format(self.PTname,self.hrrate))
```

- Note that this PT Child class is almost entirely on its own.
- Even though it takes after the Parent Class, we declare all local variables “PTname” and “hrrate” that has nothing to do with the Parent class.

D. KEYING IN 2 FT EMPLOYEES AND 1 PT EMPLOYEE

2d) Keying in 2 FT Employees & 1 PT Employee

```
✓ [4] emp1 = FullTimeStaff('Ally',5000,24)  
0s
```

```
✓ [5] emp2 = FullTimeStaff('Belinda',6000,21)  
0s
```

```
✓ [6] emp3 = PartTimeStaff('Jane',100)  
0s
```

E. DISPLAYING EMPLOYEE 1 ATTRIBUTES

2e) Displaying Employee 1 Attributes

```
✓ [▶] emp1.dispFTEmployee()  
0s
```

```
[↵] Ally salary is 5000 and leave is 24
```

F. DISPLAYING FT COUNT

▼ 2f) Displaying FT Count

```
✓  
0s [8] emp1.dispEmployeeCount()
```

```
Number of Employee is 2
```

G. DISPLAYING PT RATE

▼ 2g) Displaying PT Rate

```
✓  
0s [9] emp3.dispPTEmployee()
```

```
Jane hourly rate is 100
```

H. DELETE ALLY AWAY

▼ 2h) Delete Ally Away....

```
✓  
0s [10] del emp1
```

Ally has left the company

I. DELETE BELINDA AWAY

▼ 2i) Delete Belinda Away....

```
✓  
0s [11] del emp2
```

Belinda has left the company

2j) How Many Employees Left?

✓
0s [12] `emp3.dispEmployeeCount()`

Number of Employee is 0

THE END

ABOUT DR. ALVIN ANG



Dr. Alvin Ang earned his Ph.D., Masters and Bachelor degrees from NTU, Singapore. He is a scientist, entrepreneur, as well as a personal/business advisor. More about him at www.AlvinAng.sg.