# Python OOP

## Table of Contents

## 11_OOP

### 01_first_time_oops.py

```python
class FirstOops:
    def intro(self):
        print(f"My name is {self.name}")

a=FirstOops()
a.name='Anonymous'
a.intro()
```

### 02_class_vs_instance_attr.py

```python
class Employee:
    salary=1000 # this is class attribute

ram=Employee()
hari=Employee()

# creating instance attribute
ram.salary=200
# hari.salary=500
print(ram.salary)
print(hari.salary)


# note: instance attribute get first priority than class attribute
```

### 03_staticmethod_self.py

```python
class Employee:
    ''' if you don't use any parameter in method of class then you can
self or @staticmethod. Both will do the same job
    '''
    def greet(self):
        print("Good morning")
    @staticmethod
    def drink():
        print("I like coffee")

    # you can use any no. of @staticmethod
    @staticmethod
    def good_day():
        print("Have a good day")

a=Employee()
a.greet()
a.drink()
a.good_day()
```

### 04_init.py

```python
class Employee:
    # it runs automatically after object is created
    # def __init__(self):
    #     print("I run automatically as soon as object is created")
    name='hari'
    def __init__(self, name):
        print(f"I am {name} ")
        print(f"I am {self.name} ")
    def greet(self):
        print("Good morning")

    @staticmethod
    def drink():
        print("I like coffee")

a=Employee("Anonymous")
```

## 11_OOP/practice sets

### 01_class_programmer.py

```python
'''
Create a class programmer for storing information of few programmers
working at microsoft.
'''
```

```python
class Programmer:
    company='Microsoft'
    def __init__(self, name, age, language):
        print(f'Name: {name}')
        print(f'Age: {age}')
        print(f'Language: {language}')
        print(f'Company: {self.company}')
        print("\n")

hari=Programmer('Hari',20,'python')
ram=Programmer('Ram',22,'java')
shyam=Programmer('Shyam',19,'php')
```

## 02_calculator.py

```python
'''
Write a class calculator capable  of finding square,
cube and square root of a number.
'''
# 1st method
class Calculator1:
    def square(self,num):
        print(f"Square of {num} is {num**2}")
    def cube(self,num):
        print(f"Cube of {num} is {num**3}")
    def square_root(self,num):
        print(f"Square root of {num} is {num**0.5}")

a=Calculator1()
num=4
print("1st method: ")
a.square(num)
a.cube(num)
a.square_root(num)
print("\n")

# 2nd method
class Calculator2:
    def __init__(self,num):
        self.num=num
    def square(self):
        print(f"Square of {self.num} is {self.num**2}")
    def cube(self):
        print(f"Cube of {self.num} is {self.num**3}")
    def square_root(self):
        print(f"Square root of {self.num} is {self.num**0.5}")

print("2nd method: ")
a=Calculator2(4)
a.square()
a.cube()
a.square_root()
```

## 03.py

```python
'''
Create a class witha a class attribute 'a'; create an object from it and
set 'a' directly using object a=0. Does this change the class attribute ?
'''

# Lets' see this practically

class Employee:
    a='ram'
    def get_a(self):
        print(self.a)

first=Employee()
first.a=0
first.get_a() # 0

# 1st way to check
second=Employee()
second.get_a() # ram

# 2nd way to check
print(Employee.a)

# but if i do something like this then it will change class attribute a
Employee.a=0

# 1st way to check
second=Employee()
second.get_a() # ram

# 2nd way to check
print(Employee.a)

# conclusion: it changes only instance attribute for first
```

## 04_staticmethod.py

```python
'''
Add a static method to problem 2 to greet the user with hello
'''
class Calculator2:
    def __init__(self,num):
        self.num=num
    def square(self):
        print(f"Square of {self.num} is {self.num**2}")
    def cube(self):
        print(f"Cube of {self.num} is {self.num**3}")
    def square_root(self):
        print(f"Square root of {self.num} is {self.num**0.5}")
    @staticmethod
    def greet():
        print('Hello')
print("2nd method: ")
a=Calculator2(4)
a.square()
a.cube()
a.square_root()
a.greet()
```

### 05_train.py

```python
'''
Write a class Train chich has methods to book a ticket, get status [no. of
seats] and get fare information of trains running under Nepalese Railways.
'''

class Train:
    seats=20
    def book_ticket(self):
        if self.seats>0:
            print('Ticket is booked successfully')
            self.seats=self.seats-1
        else:
            print('Sorry,the train is full')
    def get_status(self):
        print(f"No. of seats available is {self.seats}")
    def get_fare_info(self):
        print('Chormara to Bharatpur: 200 rupees')
        print('Kathmandu to Jhapar: 2000 rupees')
        print('Surkhet to Pokhara: 1500 rupees')

a=Train()
a.book_ticket()
a.book_ticket()
a.get_status()
a.get_fare_info()
print('*****************')

b=Train()
b.get_status()
```

### 06_changing_self.py

```python
'''
Can you change the self parameter inside a class to something else (say
'ram'). Try changing self to 'slf' or 'ram'. Seet the effects
'''

class Employee:
    name="Ram"
    def a(self):
        print('Hello',self.name)

    def b(ram):
        print('Hello',ram.name)

    def c(slf):
        print('Hello',slf.name)

    def greet(slf):
        print('Hi')

emp=Employee()
emp.a()
emp.b()
emp.c()
emp.greet()

# Conclusion: Yes it it possible. But we shoudn't use it as it may confuse
other programmer while reading such code. So as a general rule of thumb we
should use self.
```

## Root