In [2]:
```
class Student:
    pass
```

In [3]:
```
gaurav = Student()
```

In [4]:
```
gaurav
```

Out[4]:
```
<__main__.Student at 0x21a8e65b820>
```

In [5]:
```
type(gaurav)
```

Out[5]:
```
__main__.Student
```

In [6]:
```
gaurav.name = "golu"
```

In [7]:
```
gaurav.name
```

Out[7]:
```
'golu'
```

In [8]:
```
tajindar = gaurav
```

In [9]:
```
print(tajindar, gaurav)
```

```
<__main__.Student object at 0x0000021A8E65B820> <__main__.Student object at 0x0000021A8E65B820>
```

In [10]:
```
print(tajindar.name, gaurav.name)
```

```
golu golu
```

In [11]:
```
gaurav.name = 'narad'
```

In [12]:
```
tajindar.name
```

Out[12]:
```
'narad'
```

In [13]:
```
class Student:
    def __init__(self):
        self.name = None
```

In [14]:
```
gaurav = Student()
```

In [17]:
```
print(gaurav.name)
```

```
None
```

In [21]:
```
class Student:
    def __init__(self):
        print(self)
        self.name = "golu"
```

In [22]:
```
gaurav = Student()
```

```
<__main__.Student object at 0x0000021A8E749370>
```

In [23]:
```
gaurav.name
```

Out[23]:
```
'golu'
```

In [24]:
```
gaurav.name = "narad"
```

In [25]:
```
gaurav.name
```

Out[25]:
```
'narad'
```

In [26]:
```
gaurav
```

Out[26]:
```
<__main__.Student at 0x21a8e749370>
```

In [42]:
```
class Student:
    def __init__(self):
        self.name = "golu"

    def __str__(self):
        return f"student is {self.name}"
```

In [39]:
```
gaurav = Student()
```

```
student is golu
```

In [40]:
```
print(gaurav)
```

```
student is golu
```

In [41]:
```
gaurav
```

Out[41]:
```
<__main__.Student at 0x21a8e749850>
```

In [43]:
```
s1 = Student()
s2 = Student()
```

```
s3 = Student()
```

In [44]:
```
print(s1.name)
print(s2.name)
print(s3.name)
```

```
golu
golu
golu
```

In [45]:
```
s2.name = "Mudit"
s3.name = "Priya"

print(s1.name)
print(s2.name)
print(s3.name)
```

```
golu
Mudit
Priya
```

In [46]:
```
class Student:
    def __init__(self, newName):
        self.name = newName

    def __str__(self):
        return f"student is {self.name}"
```

In [47]:
```
s1 = Student("Anant")
s2 = Student("Mudit")
s3 = Student("Priya")
```

In [48]:
```
print(s1.name)
print(s2.name)
print(s3.name)
```

```
Anant
Mudit
Priya
```

In [49]:
```
s2 = Student()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-49-e2d9b8a0dd10> in <module>
----> 1 s2 = Student()

TypeError: __init__() missing 1 required positional argument: 'newName'
```

In [50]:
```
class Student:
    def __init__(self, newName):
        self.name = newName
        self.rollNum = 0
```

```
    def __str__(self):
        return f"Student is {self.name}"
```

In [51]:
```
class Student:
    def __init__(self, newName, rollNum):
        self.name = newName
        self.rollNum = rollNum

    def __str__(self):
        return f"Student is {self.name}"

s1 = Student("Anant", 101)
```

In [52]:
```
print(s1.name)
print(s1.rollNum)

# Let's also print string representation of s1

print(s1)
```

```
Anant
101
Student is Anant
```

In [53]:
```
class Student:
    def __init__(self, newName, newRollNum=-1): # default value of rollNum is -1
        self.name = newName
        self.rollNum = newRollNum

    def __str__(self):
        return f"{self.rollNum}. {self.name}"

s1 = Student("Anant")
s2 = Student("Mudit")
s3 = Student("Priya", 103)
```

In [54]:
```
print(s1)
print(s2)
print(s3)
```

```
-1. Anant
-1. Mudit
103. Priya
```

In [55]:
```
class Student:
    def __init__(self, newName="NO_NAME", newRollNum):
        self.name = newName
        self.rollNum = newRollNum
```

```
  File "<ipython-input-55-69f809b3bea9>", line 2
    def __init__(self, newName="NO_NAME", newRollNum):
                 ^
SyntaxError: non-default argument follows default argument
```

In [56]:

```
class Student:
    def __init__(self, newRollNum, newName="NO_NAME"):
        self.name = newName
        self.rollNum = newRollNum
```

In [57]:

```
class Student:
    def __init__(self, newName): # counter inside `__init__`
        self.name = newName
        self.counter = 0 # initialize counter to 0
        self.counter += 1 # increment counter by 1
        self.rollNum = self.counter # assign counter to rollNum

    def __str__(self):
        return f"{self.rollNum}. {self.name}"
```

In [58]:

```
s1 = Student("Anant")
s2 = Student("Mudit")
s3 = Student("Priya")

print(s1)
print(s2)
print(s3)
```

1. Anant
1. Mudit
1. Priya

In [63]:

```
class Student:

    counter = 0 # initialize

    def __init__(self, newName):
        self.name = newName
        Student.counter += 1 # increment counter when new object is created
        self.rollNum = Student.counter # assign roll number to counter

    def __str__(self):
        return f"{self.rollNum}. {self.name}"
```

In [64]:

```
s1 = Student("Anant")
s2 = Student("Mudit")
s3 = Student("Priya")

print(s1)
print(s2)
print(s3)
```

1. Anant
2. Mudit
3. Priya

In [65]:

```
Student.counter = 1000

print(Student.counter)
```

```
print(s1.counter)
print(s2.counter)
print(s3.counter)
```

```
1000
1000
1000
1000
```

In [69]:

```
class Student:

    counter = 0

    def __init__(self, newName):
        self.name = newName
        Student.counter += 1
        self.rollNum = Student.counter

    # To give student's introduction
    def intro():
        print(f"Hello! My name is {self.name}")

    def __str__(self):
        return f"{self.rollNum}. {self.name}"

s1 = Student("Anant")
s1.intro()
```

```
Hello! My name is 1. Anant
```

In [ ]:

In [67]:

```
class Student:

    counter = 0

    def __init__(self, newName):
        self.name = newName
        Student.counter += 1
        self.rollNum = Student.counter

    def intro(self):
        print(f"Hello! My name is {self.name}")

    def __str__(self):
        return f"{self.rollNum}. {self.name}"

s1 = Student("Anant")
s1.intro()
```

```
Hello! My name is Anant
```

In [70]:

```
print(s1)
```

```
1. Anant
```

In [ ]:

```
Break : 10 40 pm
```

In [71]:
```
class Account:

    counter = 0

    def __init__(self, openingBal=0):
        Account.counter += 1
        self.id = Account.counter
        self.bal = openingBal

    # Ask Ques: What should be the parameters of deposit()?
    def deposit(self, amount):
        self.bal += amount

    def __str__(self):
        # Ask Ques: Is it going to print or return a string?
        return f"Account {self.id} has Rs. {self.bal}"
```

In [72]:
```
a1 = Account(100)
a2 = Account()

print(a1)
print(a2)
```

```
Account 1 has Rs. 100
Account 2 has Rs. 0
```

In [73]:
```
a1.deposit(50)

print(a1)
print(a2)
```

```
Account 1 has Rs. 150
Account 2 has Rs. 0
```

In [74]:
```
class Account:

    counter = 0

    def __init__(self, openingBal=0):
        Account.counter += 1
        self.id = Account.counter
        self.bal = openingBal

    def deposit(self, amount):
        self.bal += amount

    def withdraw(self, amount):
        self.bal -= amount

    def __str__(self):
        return f"Account {self.id} has Rs. {self.bal}"
```

In [75]:
```
a1 = Account(100)
```

```
a2 = Account()

print(a1)
print(a2)
```

```
Account 1 has Rs. 100
Account 2 has Rs. 0
```

In [76]:

```
a1.withdraw(50)

print(a1)
print(a2)
```

```
Account 1 has Rs. 50
Account 2 has Rs. 0
```

In [77]:

```
a2.withdraw(50)

print(a1)
print(a2)
```

```
Account 1 has Rs. 50
Account 2 has Rs. -50
```

In [78]:

```
class Account:

    counter = 0

    def __init__(self, openingBal=0):
        Account.counter += 1
        self.id = Account.counter
        self.bal = openingBal

    def deposit(self, amount):
        if amount > 0:  # condition added to deposit
            self.bal += amount

    def withdraw(self, amount):
        if amount > 0 and self.bal >= amount:  # condition added to withdraw
            self.bal -= amount

    def __str__(self):
        return f"Account {self.id} has Rs. {self.bal}"
```

In [79]:

```
a1 = Account(100)
a2 = Account()
a1.deposit(50)
print(a1)
a1.withdraw(10)
print(a1)

print(a1)
print(a2)
```

```
Account 1 has Rs. 150
Account 1 has Rs. 140
```

```
Account 1 has Rs. 140
Account 2 has Rs. 0
```

In [ ]:

```
__repr__
```

In [80]:

```
class Account:
    counter = 0
    def __init__(self, openingBal=0):
        Account.counter += 1
        self.id = Account.counter
        self.bal = openingBal

    def deposit(self, amount):
        if amount >= 0:
            self.bal += amount

    def withdraw(self, amount):
        if amount >= 0 and self.bal >= amount:
            self.bal -= amount

    def __str__(self):
        return f"Acc {self.id} has {self.bal}"
```

In [81]:

```
class SavingsAccount(Account):
    pass

class CurrentAccount(Account):
    pass

sa1 = SavingsAccount()
ca1 = CurrentAccount()
```

In [82]:

```
class Account:
    counter = 0
    def __init__(self, openingBal=0):
        Account.counter += 1
        self.id = Account.counter
        self.bal = openingBal
        self.numTrans = 0
        self.maxTrans = 2 # new

    def deposit(self, amount):
        # do you understand why < and not <=?
        if amount >= 0 and self.numTrans < self.maxTrans: # new
            self.bal += amount
            self.numTrans += 1 # new

    def withdraw(self, amount):
        if amount >= 0 and self.bal >= amount and self.numTrans < self.maxTrans: # new
            self.bal -= amount
            self.numTrans += 1 # new

    def __str__(self):
        return f"Acc {self.id} has {self.bal}"
```

In [83]:
```
class SavingsAccount(Account):
    pass

class CurrentAccount(Account):
    pass

sa1 = SavingsAccount()
ca1 = CurrentAccount()
print(sa1)
sa1.deposit(100)
print(sa1)
sa1.deposit(100)
print(sa1)
sa1.deposit(100)
print(sa1)
```

```
Acc 1 has 0
Acc 1 has 100
Acc 1 has 200
Acc 1 has 200
```

In [84]:
```
ca1.deposit(100)
print(ca1)
ca1.deposit(100)
print(ca1)
ca1.deposit(100)
print(ca1)
```

```
Acc 2 has 100
Acc 2 has 200
Acc 2 has 200
```

In [85]:
```
class Account:
    counter = 0
    def __init__(self, openingBal=0):
        Account.counter += 1
        self.id = Account.counter
        self.bal = openingBal
        self.numTrans = 0
        self.maxTrans = 2

    def deposit(self, amount):
        if amount >= 0 and self.numTrans < self.maxTrans:
            self.bal += amount
            self.numTrans += 1

    def withdraw(self, amount):
        if amount >= 0 and self.bal >= amount and self.numTrans < self.maxTrans:
            self.bal -= amount
            self.numTrans += 1

    def __str__(self):
        return f"Acc {self.id} has {self.bal}"
```

In [86]:
```
class SavingsAccount(Account):
    pass
```

```
class CurrentAccount(Account):
    self.maxTrans = 5
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-86-960c46939396> in <module>
      2     pass
      3
----> 4 class CurrentAccount(Account):
      5     self.maxTrans = 5

<ipython-input-86-960c46939396> in CurrentAccount()
      3
      4 class CurrentAccount(Account):
----> 5     self.maxTrans = 5

NameError: name 'self' is not defined
```

In [87]:

```
class SavingsAccount(Account):
    pass

class CurrentAccount(Account):
    def __init__(self):  # new
        self.maxTrans = 5
```

In [88]:

```
sa1 = SavingsAccount()
ca1 = CurrentAccount()

print(sa1)
sa1.deposit(100)
print(sa1)

print(ca1)  # new
ca1.deposit(100)  # new
print(ca1)  # new
```

```
Acc 1 has 0
Acc 1 has 100
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-88-f2c54f1d00e6> in <module>
      6 print(sa1)
      7
----> 8 print(ca1)  # new
      9 ca1.deposit(100)  # new
     10 print(ca1)  # new

<ipython-input-85-83e23e1186e1> in __str__(self)
     19
     20     def __str__(self):
---> 21         return f"Acc {self.id} has {self.bal}"

AttributeError: 'CurrentAccount' object has no attribute 'id'
```

In [89]:

```
class SavingsAccount(Account):
    pass

class CurrentAccount(Account):
    def __init__(self):
```

```
        super().__init__()  # new
        self.maxTrans = 5
```

In [90]:

```
sa1 = SavingsAccount()
ca1 = CurrentAccount()

print(sa1)
sa1.deposit(100)
print(sa1)

print(ca1)
ca1.deposit(100)
print(ca1)
```

```
Acc 2 has 0
Acc 2 has 100
Acc 3 has 0
Acc 3 has 100
```

In [91]:

```
print(sa1)
sa1.deposit(100)
sa1.withdraw(50)
sa1.deposit(100)
sa1.withdraw(50)
sa1.bal = 999999999  # new - manually setting the balance
print(sa1)
```

```
Acc 2 has 100
Acc 2 has 999999999
```

In [102…

```
class Account:
    counter = 0
    def __init__(self, openingBal=0):
        Account.counter += 1
        self.id = Account.counter
        self.__bal = openingBal  # new --> self.__bal
        self.numTrans = 0
        self.maxTrans = 2

    def deposit(self, amount):
        if amount >= 0 and self.numTrans < self.maxTrans:
            self.__bal += amount  # new --> self.__bal
            self.numTrans += 1

    def withdraw(self, amount):
        if amount >= 0 and self.__bal >= amount and self.numTrans < self.maxTrans:  # n
            self.__bal -= amount  # new --> self.__bal
            self.numTrans += 1

    def __str__(self):
        return f"Acc {self.id} has {self.__bal}"  # new --> self.__bal
```

In [93]:

```
class SavingsAccount(Account):
    pass

class CurrentAccount(Account):
```

```
    def __init__(self):
        super().__init__()
        self.maxTrans = 3


sa1 = SavingsAccount()
ca1 = CurrentAccount()
```

In [100…

```
print(sa1)
sa1.deposit(100)
sa1.withdraw(50)
sa1.deposit(100)
sa1.withdraw(50)
sa1.__bal = 999999999  # new - Manual change will NOT work
print(sa1)
print(sa1.__bal)
"__b"
__b
```

```
Acc 1 has 50
Acc 1 has 50
999999999
```

In [95]:

```
class Account:

    counter = 0

    def __init__(self, openingBal=0):
        Account.counter += 1
        self.id = Account.counter
        self.bal = openingBal
        self.numTrans = 0
        self.maxTrans = 2

    def deposit(self, amount):
        if amount >= 0 and self.numTrans < self.maxTrans:
            self.bal += amount
            self.numTrans += 1

    def withdraw(self, amount):
        if amount >= 0 and self.bal >= amount and self.numTrans < self.maxTrans:
            self.bal -= amount
            self.numTrans += 1

    def getInterest(self): # new
        pass


    def __str__(self):
        return f"Acc {self.id} has {self.bal}"  # new --> self.__bal
```

In [96]:

```
class SavingsAccount(Account):
    def __init__(self):
        super().__init__()

    def getInterest(self):  # new - Interest calculation for Savings Account
        interest = self.bal*0.07
```

```
        print(f"Interest on Account {self.id} is {interest}")


class CurrentAccount(Account):
    def __init__(self):
        super().__init__()
        self.maxTrans = 3

    def getInterest(self):  # new - Interest calculation for Current Account
        interest = (self.bal*0.05)/self.numTrans
        print(f"Interest on Account {self.id} is {interest}")
```

In [97]:

```
sa1 = SavingsAccount()
ca1 = CurrentAccount()

print(sa1)
sa1.deposit(100)
sa1.withdraw(50)
print(sa1)
sa1.getInterest()
```

```
Acc 1 has 0
Acc 1 has 50
Interest on Account 1 is 3.5000000000000004
```

In [98]:

```
print(ca1)
ca1.deposit(100)
ca1.deposit(100)
ca1.deposit(100)
print(ca1)
ca1.getInterest()
```

```
Acc 2 has 0
Acc 2 has 300
Interest on Account 2 is 5.0
```

In [ ]: