OOPS (Last Class)

Colab Link - https://colab.research.google.com/drive/1xfYU8Qv5BwKftUFQ0yq5vVDQlW1VoSrl?usp=sharing

```
class Student:
  pass
```

```
s1 = Student()
```

```
s1
```

```
<__main__.Student at 0x7f574cfc7f50>
```

```
type(s1)
```

```
__main__.Student
```

```
s2 = Student()
```

```
s2
```

```
<__main__.Student at 0x7f574ad261d0>
```

```
# attributes/properties
```

```
s1.name = "Rahul"
```

```
s2.name = "Anant"
```

```
s1.name
```

```
'Rahul'
```

```
s2.name
```

```
'Anant'
```

```
class Student:
  # dunder function (init)
  def __init__(self):
    self.name = "placeholder"
```

```
s1 = Student()
```

```
s1.name
```

```
    'placeholder'
```

```
s2 = Student()
```

```
s2.name
```

```
    'placeholder'
```

```
s1.name = "Anant"
```

```
s1.name
```

```
    'Anant'
```

```
s2.name
```

```
    'placeholder'
```

```
class Student:
  # dunder function (init)
  def __init__(self):
    self.name = "placeholder"
```

```
s = Student()
print(s)
```

```
    <__main__.Student object at 0x7f574acedad0>
    <__main__.Student object at 0x7f574acedad0>
```

```
class Student:
  # dunder function (init)
  def __init__(self):
    self.name = "placeholder"

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

```
s = Student()
```

```
print(s)
```

```
    Student's name is placeholder
```

```
class List:
  def __init__(self):
    pass

  def __str__(self):
    return
```

```
l = [1, 2, 3, 4]
```

```
print(l)
```

```
    [1, 2, 3, 4]
```

```
class Student:
  # dunder function (init)
  def __init__(self):
    name = "placeholder"
```

```
s = Student()
```

```
s.name
```

```
    ---------------------------------------------------------------------------
    AttributeError                            Traceback (most recent call last)
    <ipython-input-43-98b019fbfe94> in <module>()
    ----> 1 s.name

    AttributeError: 'Student' object has no attribute 'name'

     SEARCH STACK OVERFLOW
```

```
class Student:
  # dunder function (init)
  def __init__(self, name_value):
    self.name = name_value

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

```
s = Student("Anant")
```

```
s.name
```

```
    'Anant'
```

```
s2 = Student("Parth")
```

```
s2.name
```

```
'Parth'
```

```
print(s)
```

```
Student's name is Anant
```

```
print(s2)
```

```
Student's name is Parth
```

```
s3 = Student()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-51-88c31a65a5cc> in <module>()
----> 1 s3 = Student()

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

SEARCH STACK OVERFLOW

```
class Student:
  # dunder function (init)
  def __init__(self, name_value="placeholder"):
    self.name = name_value

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

```
s = Student("Anant")
print(s)
```

```
Student's name is Anant
```

```
s = Student()
print(s)
```

```
Student's name is placeholder
```

```
class Vehicle:
    def __init__(self, name):
      self.name = name
```

```
v = Vehicle("minivan")
```

```
v.name
```

```
'minivan'
```

```
Vehicle.__init__(v, "suv")
```

```
v.name
```

```
'suv'
```

```
class Student:
  def __init__(name_value, self):
    name_value.name= self
```

```
s = Student("Anant")
```

```
s.name
```

```
'Anant'
```

```
class Student:
  # dunder function (init)
  def __init__(self, new_name, new_roll_num):
    self.name = new_name
    self.roll_num = new_roll_num

  def __str__(self):
    return f"Student's name is {self.name} and roll number is {self.roll_num}"
```

```
s1 = Student("Anant", 1)
s2 = Student("Mudit", 2)
```

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

```
Student's name is Anant and roll number is 1
Student's name is Mudit and roll number is 2
```

```
class Student:
  counter = 0 # class variable
  # dunder function (init)
  def __init__(self, new_name):
    self.name = new_name
    Student.counter += 1
    self.roll_num = Student.counter

  def __str__(self):
    return f"Student's name is {self.name} and roll number is {self.roll_num}"
```

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

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

```
Student's name is Anant and roll number is 1
Student's name is Mudit and roll number is 2
Student's name is Mohit and roll number is 3
```

```
Student.counter
```

```
3
```

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

```
3
3
3
```

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

```
1
2
3
```

```
Student.counter = 1000
```

```
Student.counter
```

```
1000
```

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

```
1000
1000
1000
```

```
class Student:

    counter = 100

    def __init__(self, newName):
        self.name = newName
        Student.counter += 1
        self.rollNum = Student.counter
    def __str__(self):
        return f"{self.rollNum}. {self.name}"

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

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

```
101. Anant
102. Mudit
103. Priya
--------------------------------------------------
```

```
s1.counter = 10000
```

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

```
103
10000
103
103
```

```
class Vehicle:
    country = "India"

    def __init__(self, name, mileage):
        self.name = name
        self.mileage = mileage

    def __str__(self):
        return 'Vehicle Name={} \nMileage={}'.format(self.name, self.mileage)

v1 = Vehicle("minivan", 10)

print(v1.country)
v1.country = "USA"
print(Vehicle.country)
print(v1.country)
```

```
India
India
USA
```

```
class Student:
    counter = 0

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

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

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

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

```
     Hello my name is Anant
```

```
# create a class Account
# id, bal (get it as an argument)
# a1, id=1, bal=100
# a2 id=2, bal=0


class Account:

  counter = 0

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

  def __str__(self):
    return f"Account Number: {self.id}, Account Balance: {self.bal}"


a1 = Account(100)
a2 = Account()


print(a1)
print(a2)
```

```
     Account Number: 1, Account Balance: 100
     Account Number: 2, Account Balance: 0
```

```
class Account:

  counter = 0

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

  def __str__(self):
    return f"Account Number: {self.id}, Account Balance: {self.bal}"

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

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

```
     Account Number: 1, Account Balance: 150
     Account Number: 2, Account Balance: 0
```

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

    def __str__(self):
      return f"Account Number: {self.id}, Account Balance: {self.bal}"

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

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

  a1 = Account(100)
  a2 = Account()
  a1.deposit(50)
  a2.withdraw(30)
  print(a1)
  print(a2)
```

```
     Account Number: 1, Account Balance: 150
     Account Number: 2, Account Balance: 0
```

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

    def __str__(self):
      return f"Account Number: {self.id}, Account Balance: {self.bal}"
    def deposit(self, amount):
      if amount > 0:
        self.bal += amount

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

    def __repr__(self):
      return f"{self.id}"

    # def __myspecialdunder__(self):
    #   print("This is special")

  a1 = Account(100)
  a2 = Account()
  a1.deposit(50)
  a2.withdraw(30)
  print(a1)
  print(a2)
```

```
    Account Number: 1, Account Balance: 150
    Account Number: 2, Account Balance: 0
```

```
str(a2)
```

```
    'Account Number: 2, Account Balance: 0'
```

```
a2.__str__()
```

```
    'Account Number: 2, Account Balance: 0'
```

```
repr(a2)
```

```
    '2'
```

```
a2.__repr__()
```

```
    '2'
```

```
# Inheritance
```

```
class Account:
    counter = 0
    def __init__(self, openingBal=0):
        Account.counter += 1
        self.id = Account.counter
        self.bal = openingBal
        self.num_transactions = 0
        self.max_transactions = 5

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

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

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

    def __repr__(self):
        return f"{id}"


class SavingsAccount(Account):
  pass


class CurrentAccount(Account):
  pass
```

```
s1 = SavingsAccount()
```

```
c1 = CurrentAccount()
```

```
print(s1)
```

```
    Acc 1 has Rs.0
```

```
print(c1)
```

```
    Acc 2 has Rs.0
```

```
s1.deposit(100)
```

```
print(s1)
```

```
    Acc 1 has Rs.100
```

```
s1.deposit(20)
```

```
print(s1)
```

```
Acc 1 has Rs.120
```

```
s1.deposit(40)
```

```
print(s1)
```

```
Acc 1 has Rs.160
```

```
s1.deposit(100)
```

```
print(s1)
```

```
Acc 1 has Rs.260
```

```
s1.deposit(50)
```

```
print(s1)
```

```
Acc 1 has Rs.310
```

```
s1.deposit(100)
```

```
print(s1)
```

```
Acc 1 has Rs.310
```

```
class Account:
    counter = 0
    def __init__(self, openingBal=0):
        Account.counter += 1
        self.id = Account.counter
        self.bal = openingBal
        self.num_transactions = 0
        self.max_transactions = 5

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

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

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

class SavingsAccount(Account):
  pass

class CurrentAccount(Account):
  def __init__(self):
    self.max_transactions = 100

s1 = SavingsAccount()
s1.deposit(100)
s1.deposit(4)
print(s1)
```

```
Acc 1 has Rs.104
```

```
c1 = CurrentAccount()
c1.deposit(100)
c1.deposit(4)
print(c1)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-174-c3483529a6e0> in <module>()
      1 c1 = CurrentAccount()
----> 2 c1.deposit(100)
      3 c1.deposit(4)
      4 print(c1)

<ipython-input-172-a727d00559aa> in deposit(self, amount)
      9 
     10     def deposit(self, amount):
---> 11         if amount >= 0 and self.num_transactions < self.max_transactic
     12             self.bal += amount
     13             self.num_transactions += 1

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

SEARCH STACK OVERFLOW

```
class Account:
    counter = 0
    def __init__(self, openingBal=0):
        Account.counter += 1
        self.id = Account.counter
        self.bal = openingBal
        self.num_transactions = 0
        self.max_transactions = 2

    def deposit(self, amount):
        if amount >= 0 and self.num_transactions < self.max_transactions:
            self.bal += amount
            self.num_transactions += 1
    def withdraw(self, amount):
        if amount >= 0 and self.bal >= amount and self.num_transactions < self.max_
            self.bal -= amount
            self.num_transactions += 1

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

    def __repr__(self):
        return f"{id}"

class SavingsAccount(Account):
  pass

class CurrentAccount(Account):
  def __init__(self):
    super().__init__()
    self.max_transactions = 5

s1 = SavingsAccount()
s1.deposit(100)
s1.deposit(4)
s1.deposit(1000)
print(s1)
```

```
Acc 1 has Rs.104
```

```
s1 = CurrentAccount()
s1.deposit(100)
s1.deposit(4)
s1.deposit(1000)
print(s1)
```

```
Acc 2 has Rs.1104
```

```
# Private Variables - __anant
# Polymporphism - calculate_interest()


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

    def __repr__(self):
        return f"{id}"

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}")


class Student:
  pass

s = Student()
s.name = "Anant"

l = [1, 2, 3, 4]
l.some_attr = "Anant" # asked by Shubham, need to check
```

```
class CustomList(list):
    pass

l = CustomList([1, 2, 3, 4])
l.some_attr = "Anant"
# https://stackoverflow.com/questions/4698493/can-i-add-custom-methods-attributes-t
# https://stackoverflow.com/questions/1285269/why-cant-you-add-attributes-to-object
```

SEARCH STACK OVERFLOW

```
l.__dict__
```

```
{'some_attr': 'Anant'}
```

`__customdunderemethods__` , shared by Lakshmi, need to check "again". https://www.geeksforgeeks.org/customize-your-python-class-with-magic-or-dunder-methods/ Use for retaining the inherited feature of the built-in class while providing customized class behavior https://stackoverflow.com/q/9302814/12266923

OOPs (Private Variables and Polymorphism)

```
class Account:
    counter = 0
    def __init__(self, openingBal=0):
        Account.counter += 1
        self.id = Account.counter
        self.bal = openingBal
        self.num_transactions = 0
        self.max_transactions = 2

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

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

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

    def __repr__(self):
        return f"{id}"

class SavingsAccount(Account):
  pass

class CurrentAccount(Account):
  def __init__(self):
    super().__init__()
    self.max_transactions = 5

sa1 = SavingsAccount()
```

```
print(sa1)
```

```
     Acc 1 has Rs.0
```

```
print(ca1)
```

```
     Acc 2 has 0
```

```
sa1.deposit(100)
sa1.withdraw(150)
sa1.withdraw(50)
```

```
print(sa1)
```

```
     Acc 1 has Rs.50
```

```
sa1.bal
```

```
     50
```

```
sa1.bal = 999999
```

```
print(sa1)
```

```
     Acc 1 has Rs.999999
```

```
class Account:
    counter = 0
    def __init__(self, openingBal=0):
        Account.counter += 1
        self.id = Account.counter
        self.__bal = openingBal
        self.num_transactions = 0
        self.max_transactions = 2

    def deposit(self, amount):
        if amount >= 0 and self.num_transactions < self.max_transactions:
            self.__bal += amount
            self.num_transactions += 1

    def withdraw(self, amount):
        if amount >= 0 and self.__bal >= amount and self.num_transactions < self.ma
            self.__bal -= amount
            self.num_transactions += 1

    def __str__(self):
        return f"Acc {self.id} has Rs.{self.__bal}"

    def __repr__(self):
        return f"{id}"

class SavingsAccount(Account):
  pass

class CurrentAccount(Account):
  def __init__(self):
    super().__init__()
    self.max_transactions = 5

sa1 = SavingsAccount()

sa1.deposit(100)
sa1.withdraw(150)
sa1.withdraw(50)
```

```
print(sa1)
```

```
    Acc 1 has Rs.50
```

```
sa1.__bal = 999999
```

```
print(sa1)
```

```
    Acc 1 has Rs.50
```

```
sa1.__bal
```

```
    999999
```

```
# __bal --> _Account__bal
```

```
sa1._Account__bal
```

```
    50
```

```
sa1._Account__bal = 1000
print(sa1)
```

```
    Acc 1 has Rs.1000
```

Public, Protected and Private

```
# Polymorphism

# Savings Account - 3%
# Current Account - No interest


class Account:
    counter = 0
    def __init__(self, openingBal=0):
        Account.counter += 1
        self.id = Account.counter
        self.__bal = openingBal
        self.num_transactions = 0
        self.max_transactions = 2

    def deposit(self, amount):
        if amount >= 0 and self.num_transactions < self.max_transactions:
            self.__bal += amount
            self.num_transactions += 1

    def withdraw(self, amount):
        if amount >= 0 and self.__bal >= amount and self.num_transactions < self.ma
            self.__bal -= amount
            self.num_transactions += 1

    def __str__(self):
        return f"Acc {self.id} has Rs.{self.__bal}"

    def __repr__(self):
        return f"{id}"

    def get_interest(self):
      pass


class SavingsAccount(Account):
  def get_interest(self):
    return self._Account__bal*0.03

class CurrentAccount(Account):
  def __init__(self):
    super().__init__()
    self.max_transactions = 5

  def get_interest(self):
    return 0

s1 = SavingsAccount()
c1 = CurrentAccount()


print(s1)
```

```
    Acc 1 has Rs.0
```

```
s1.deposit(102)
c1.deposit(100)
print(s1)
print(c1)
```

```
Acc 1 has Rs.102
Acc 2 has Rs.100
```

```
c1.get_interest()
```

```
0
```

```
s1.get_interest()
```

```
3.06
```

✓ 0s completed at 21:51