```
class Person:
  def __init__(self):
    self.name=input("Enter name:")
  def __del__(self):
    print("R.I.P.", self.name)
```

```
p=Person()
```

```
Enter name:Joker
```

```
p=Person()
```

```
Enter name:wonder woman
R.I.P. Joker
```

```
k=Person()
```

```
Enter name:superman
```

```
p=Person()
```

```
Enter name:zeena
R.I.P. wonder woman
```

```
p=Person()
```

```
Enter name:heena
```

```
# code style-2, industry standard
# means to create classes that dooes not take inputs itself, allows re-usability
class Human:
  def __init__(self, name, gender): # constructor method
    print("object created:", id(self))
    self.gender=gender
    self.name=name
  def __str__(self): # printer method, returns string
    return "Hi I am a "+self.gender+" called "+self.name
```

```
h=Human() # your've to pass arguments while creating class
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-a6c145001559> in <cell line: 1>()
----> 1 h=Human()

TypeError: __init__() missing 2 required positional arguments: 'name' and 'gender'
```

SEARCH STACK OVERFLOW

```
h=Human("Heman", "male")
```

```
object created: 139696790534656
```

```
print(h)
```

```
Hi I am a male called Heman
```

```
n=input("Name:")
g=input("Gender:")
h=Human(n, g)
```

```
Name:shaktiman
Gender:male
object created: 139697213586112
```

```
# code style-1, basic user
class Human:
  def __init__(self): # constructor method
    print("object created:", id(self))
    self.gender=input("Enter gender: ")
    self.name=input("Enter name: ")
  def __str__(self): # printer method, returns string
    return "Hi I am a "+self.gender+" called "+self.name
```

visibility / access specifier

```
class Person:
  def __init__(self,gender,name,number):
    self.gender=gender # public
    self._name=name # private
    self.__number=number # strong private
```

```
p=Person("male", "amar", 9821601163)
```

```
p.gender
```

```
'male'
```

```
p._name
```

```
'amar'
```

```
p.__number # strong private member not directly accessible to class object
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
```

```
class Person:
  def __init__(self,gender,name,number):
    self.gender=gender
    self._name=name
    self.__number=number
  def vibhishan(self): # strong private members using this member function of same class
    print(self.__number)
```

```
p=Person("male", "amar", 9821601163)
```

```
p.vibhishan()
```

```
9821601163
```

```
class Person:
  def __init__(self,gender,name,number):
    self.gender=gender
    self._name=name
    self.__number=number
  def _vibhishan(self): # made this method private
    print(self.__number)
```

```
p=Person("male", "amar", 9821601163)
```

```
p._vibhishan()
```

```
9821601163
```

```
class Person:
  def __init__(self,gender,name,number):
    self.gender=gender
    self._name=name
    self.__number=number
  def __vibhishan(self): # made this method strong private
    print(self.__number)
```

```
p=Person("male", "amar", 9821601163)
```

```
p.__vibhishan() # throws error as it is strong private method, thus not accessible to outside class
```

```python
class Human:
  count=0 # class variable
  def __init__(self, name, gender): # constructor
    print("object created:", id(self))
    self.gender=gender # instance variable
    self.name=name # instance variable
    Human.count+=1

  def __str__(self):
    return "Hi I am a "+self.gender+" called "+self.name

  def population(self):
    print("TOtal humans:", Human.count)
```

```python
h=Human("amar", "male")
```

```
object created: 139696790445408
```

```python
h2=Human("amrita", "female")
```

```
object created: 139696790444928
```

```python
h3=Human("samrita", "female")
```

```
object created: 139696790444736
```

```python
Human.count
```

```
3
```

Inheritance

```python
class A:
  def mA1(self):
    print("mA1 called")
  def _mA2(self):
    print("mA2 called")
  def __mA3(self):
    print("mA3 called")
```

```python
class B:
  def mB1(self):
    print("mB1 called")
  def _mB2(self):
    print("mB2 called")
  def __mB3(self):
    print("mB3 called")
```

```python
obj=B()
```

```
obj.mB1()
```

```
mB1 called
```

```
obj._mB2
```

```
<bound method B._mB2 of <__main__.B object at 0x7f0db178f250>>
```

```
obj.__mB3() # cannot be accessed directly, would need another class memebr to access
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-76-4c49ff3981da> in <cell line: 1>()
----> 1 obj.__mB3()

AttributeError: 'B' object has no attribute '__mB3'
```

SEARCH STACK OVERFLOW

```
class B(A): # class B is child class which inherits from parent class A
  def mB1(self):
    print("mB1 called")
  def _mB2(self):
    print("mB2 called")
  def __mB3(self):
    print("mB3 called")
```

```
obj=B()
```

```
obj.mA1()
```

```
mA1 called
```

```
obj._mA2() # private can be inherited
```

```
mA2 called
```

```
obj.__mA3() # strong private cannot be inherited
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-75-9c3b559fb3f9> in <cell line: 1>()
----> 1 obj.__mA3()

AttributeError: 'B' object has no attribute '__mA3'
```

SEARCH STACK OVERFLOW

```
class A:
  def my(self):
    print("A's my")
```

```
class B:
  def my(self):
    print("B's my")
class C:
  def my(self):
    print("C's my")
```

```
class X(A, B, C): # multiple inheritance follows FCFS, so common members of only class A are inherited
  def myself(self):
    print("hi X")
```

```
obj=X()
obj.my()
```

```
A's my
```

```
class X(B, C, A): # multiple inheritance follows FCFS, so common members of only class B are inherited
  def myself(self):
  def myself(self):
    print("hi X")
```

```
obj=X()
obj.my()
```

```
B's my
```

```
class X(C, A, B): # multiple inheritance follows FCFS
  def myself(self):
    print("hi X")
```

```
obj=X()
obj.my()
```

```
C's my
```

```
class A:
  def my(self): # common method
    print("A's my")
  def myA(self): # different method
    print("A's")
class B:
  def my(self): # common method
    print("B's my")
  def myB(self): # different method
    print("B's")
class C:
  def my(self): # common method
    print("C's my")
  def myC(self): # different method
    print("C's")
```

```
class X(C, A, B): # exact same method gets inherited only once, but different methods are inherited, following FCFS for inheritance
  def myself(self):
    print("hi X")
```

```
obj=X()
```

```
obj.my()
```

```
C's my
```

```
obj.myA()
```

```
A's
```

```
obj.myB()
```

```
B's
```

```
obj.myC()
```

```
C's
```

```
# override
class Parent:
  def welcome(self):
    print("hello how are you how do you do bla bla")
  def other(self):
    print("Other things")
class Child(Parent):
  def chill(self):
    print("just chilling")
```

```
obj=Child()
```

```
obj.chill()
obj.other()
obj.welcome()
```

```
just chilling
Other things
hello how are you how do you do bla bla
```

```
# override
class Parent:
  def welcome(self): # parent method welcome
    print("hello how are you how do you do bla bla")
  def other(self):
    print("Other things")
class Child(Parent):
  def welcome(self): # overrides parent welcome() definition, also exhibits polymorphism
    print("hi")
  def chill(self):
    print("just chilling")
```

```
obj=Child()
```

```
obj.welcome()
```

```
hi
```

```
objc=Child()
objp=Parent()
```

```
objp.welcome()
```

```
hello how are you how do you do bla bla
```

```
objc.welcome()
```

```
hi
```

```
# create a class circle, with a method  redr() to read radius and area() to print area
class Circle:
  def readR(self, radius):
    self.radius=radius
  def area(self):
    area=3.14*self.radius*self.radius
    return area
circ=Circle()
r=float(input("Enter radius: "))
circ.readR(r)
circ.area()
```

```
Enter radius: 5.6
98.4704
```

```
# class emp
# id, name, salary --strong private
# setDetail() , printDetail() --private
class Emp:
  def _setDetails(self, id, name, salary):
    self.__id=id
    self.__name=name
    self.__salary=salary
  def _printDetails(self):
    print("ID:", self.__id, "Name:", self.__name, "Salary:", self.__salary)
e=Emp()
print("Enter details of Employee:")
e._setDetails(int(input("Enter ID: ")), input("Enter Employee Name: "), float(input("Enter Salary: ")))
e._printDetails()
```

```
Enter details of Employee:
Enter ID: 111
Enter Employee Name: Surya
Enter Salary: 74745
ID: 111 Name: Surya Salary: 74745.0
```

```python
# class emp
# id, name, salary --strong private
# setDetail() , printDetail() --private
# change setDetails to constructor
class Emp:
  def __init__(self, id, name, salary):
    self.__id=id
    self.__name=name
    self.__salary=salary
  def _printDetails(self):
    print("ID:", self.__id, "Name:", self.__name, "Salary:", self.__salary)

print("Enter details of Employee:")
e=Emp(int(input("Enter ID: ")), input("Enter Employee Name: "), float(input("Enter Salary: ")))
e._printDetails()
```

```
Enter details of Employee:
Enter ID: 11
Enter Employee Name: Surya
Enter Salary: 45625
ID: 11 Name: Surya Salary: 45625.0
```

```python
# class emp
# id, name, salary --strong private
# setDetail() , printDetail() --private
# change setDetails to constructor
# replace printDeails with printer
class Emp:
  def __init__(self, id, name, salary):
    self.__id=id
    self.__name=name
    self.__salary=salary
  def __str__(self):
    return "ID: "+str(self.__id)+" Name: "+self.__name+" Salary: "+str(self.__salary)

print("Enter details of Employee:")
e=Emp(int(input("Enter ID: ")), input("Enter Employee Name: "), float(input("Enter Salary: ")))
print(e)
```

```
Enter details of Employee:
Enter ID: 123
Enter Employee Name: Surya
Enter Salary: 45645
ID: 123 Name: Surya Salary: 45645.0
```

```python
# class emp
# id, name, salary --strong private
# setDetail() , printDetail() --private
# change setDetails to constructor
# replace printDeails with printer
# empID should be auto generating
```

```python
class Emp:
  empid=202300
  def __init__(self, name, salary):
    self.__id=Emp.empid
    print("Employee ID: ", self.__id)
    self.__name=name
    self.__salary=salary
    Emp.empid+=1
  def __str__(self):
    return "ID: "+str(self.__id)+" Name: "+self.__name+" Salary: "+str(self.__salary)

print("Enter details of Employee:")
e=Emp(input("Enter Employee Name: "), float(input("Enter Salary: ")))
print(e)
```

```
Enter details of Employee:
Enter Employee Name: Surya
Enter Salary: 45454
Employee ID:  202300
ID: 202300 Name: Surya Salary: 45454.0
```

```python
e=Emp(input("Enter Employee Name: "), float(input("Enter Salary: ")))
print(e)
```

```
Enter Employee Name: Varun
Enter Salary: 26000
Employee ID:  202301
ID: 202301 Name: Varun Salary: 26000.0
```

```python
e=Emp(input("Enter Employee Name: "), float(input("Enter Salary: ")))
print(e)
```

```
Enter Employee Name: Tarun
Enter Salary: 45415
Employee ID:  202302
ID: 202302 Name: Tarun Salary: 45415.0
```

```python
# class emp
# id, name, salary --strong private
# setDetail() , printDetail() --private
# change setDetails to constructor
# replace printDeails with printer
# empID should be auto generating
# store details of 5 employees
class Emp:
  empid=202300
  def __init__(self, name, salary):
    self.__id=Emp.empid
    print("Employee ID: ", self.__id, "\n")
    self.__name=name
    self.__salary=salary
    Emp.empid+=1
  def __str__(self):
    return "ID: "+str(self.__id)+" Name: "+self.__name+" Salary: "+str(self.__salary)

```

```python
print("Enter details of Employee:")
emplist=[]
for i in range(0, 5):
  e=Emp(input("\nEnter Employee Name: "), float(input("Enter Salary: ")))
  emplist.append(e)
for i in range(0, 5):
  print(emplist[i])
```

```
Enter details of Employee:

Enter Employee Name: Surya
Enter Salary: 45655
Employee ID:  202300


Enter Employee Name: Nishant
Enter Salary: 154512
Employee ID:  202301


Enter Employee Name: Sahil
Enter Salary: 65254
Employee ID:  202302


Enter Employee Name: Prabjot
Enter Salary: 75858
Employee ID:  202303


Enter Employee Name: Ritwik
Enter Salary: 15655
Employee ID:  202304

ID: 202300 Name: Surya Salary: 45655.0
ID: 202301 Name: Nishant Salary: 154512.0
ID: 202302 Name: Sahil Salary: 65254.0
ID: 202303 Name: Prabjot Salary: 75858.0
ID: 202304 Name: Ritwik Salary: 15655.0
```

```python
# CG, 3i, and morningstar
# id, name, salary --strong private
# setDetail() , printDetail() --private
# change setDetails to constructor
# replace printDeails with printer
# empID should be auto generating
# store details of 5 employees
# menu driven to create, search, delete employee
class Emp:
    empid=202300
    def __init__(self, name, salary):
        self.__id=Emp.empid
        print("Employee ID: ", self.__id, "\n")
        self.__name=name
        self.__salary=salary
        Emp.empid+=1
    def __str__(self):
        return "ID: "+str(self.__id)+" Name: "+self.__name+" Salary: "+str(self.__salary)
    def getempid(self):
        return self.__id

emplist=[]
while True:
    print("1. Create \n2. Search\n3. Delete\n0. Exit")
    ch=int(input(" : "))
    if ch ==1:
        e=Emp(input("\nEnter Employee Name: "), float(input("Enter Salary: ")))
        emplist.append(e)
    elif ch==2: # search logic
        tempid=int(input("Enter ID: "))
        flag=False
        for employee in emplist:
            if employee.getempid()==tempid:
                print("Found")
                flag=True
                print(employee)
                break
        if flag==False:
            print("Not found")
    elif ch==3: # delete logic
        # read id to delete
        tempid=int(input("Enter ID: "))
        flag=False
        for clsele in range(len(emplist)):
            if emplist[clsele].getempid()==tempid:
                print("Found")
                flag=True
                print("Deleted: ", emplist.pop(clsele))
                break
        if flag==False:
            print("Not found")
    elif ch==0:
        print("Bye-Bye")
        break
    else:
        print("\nInvalid input\n")
```

```
1. Create
2. Search
3. Delete
0. Exit
 : 1

Enter Employee Name: aaaa
Enter Salary: 1000
Employee ID:  202300

1. Create
2. Search
3. Delete
0. Exit
 : 1

Enter Employee Name: bbbb
Enter Salary: 2000
Employee ID:  202301
```

```
1. Create
2. Search
3. Delete
0. Exit
 : 1

Enter Employee Name: cccc
Enter Salary: 3000
Employee ID:  202302

1. Create
2. Search
3. Delete
0. Exit
 : 1

Enter Employee Name: dddd
Enter Salary: 4000
Employee ID:  202303

1. Create
2. Search
3. Delete
0. Exit
 : 3
Enter ID: 202303
Found
Deleted:  ID: 202303 Name: dddd Salary: 4000.0
1. Create
2. Search
3. Delete
0. Exit
 : 2
Enter ID: 202303
Not found
1. Create
2. Search
3. Delete
```

empty class / pass keyword

```
class Insan:
  pass
```

```
i=Insan()
```

setattr() getattr()

```
class Insan:
  pass
```

```
i=Insan()
```

```
setattr(i, "name", "xmax") # sets attributes
```

```
setattr(i, "__contact", 9821601163)
```

```
getattr(i, "age", "not given")
```

```
'not given'
```

super

```
# super
class A:
  def __init__(self):
    print("A")
class B(A):
  def __init__(self):
    super().__init__() # will call __init__() from super/parent class
    print("B")
obj=B()
```

```
A
B
```

```
# super in case of inheritance
class A:
  def __init__(self):
    print("A")
class B:
  def __init__(self):
    print("B")
class X(A, B):
  def __init__(self):
    super().__init__() # will call __init__() class A, by following FCFS in inheritance
    print("X")
obj=X()
```

```
A
X
```

```
# super
class A:
  def __init__(self):
    print("A")
class B:
  def __init__(self):
    print("B")
class X(B, A):
  def __init__(self):
    super().__init__() # will call __init__() class B, by following FCFS in inheritance
    print("X")
obj=X()
```

```
B
X
```

```
# super -  calling parent method from child class
class A:
  def __init__(self, d1):
    print("A", d1)
class B(A):
  def __init__(self, d1, d2):
    super().__init__(d1) # will call __init__() class A, by following FCFS in inheritance
    print("B", d2)
class X(B):
  def __init__(self, d1, d2, d3):
    super().__init__(d1, d2) # will call __init__() class B, by following FCFS in inheritance
    print("X", d3)
obj=X(11, 22, 33)
```

```
A 11
B 22
X 33
```

```
class Person:
  def __init__(self): # first definition for __init__()
    print("the one")
  def __init__(self): # second definition for __init__()
    print("the two")
  def __init__(self):# third definition for __init__()
    print("the three")
  def __str__(self):
    print("R.I.P.", self.name)
p=Person() # calls the latest definitnion of __init__(), thus overrides any previous definintion
p=Person()
```

```
the three
the three
```

```
class Triangle:
  def __init__(self, base, alt):
    self.__base=base
    self.__alt=alt
  def area(self):
    print("Area :", 0.5*self.__base*self.__alt)
tr=Triangle(3, 7)
tr.area()
```

```
Area : 10.5
```

```
#create a bank class
# has amount,accountno,name
# createaccount(constructor)-user only gives name and amount
# account number auto generated
```

```
#withdraw(amount):should not be -ve amount and min
# balance is 2000 else reject transection
#deposite(amount):amount can not be -ve
#checkbalance():shows balance and account holder name

'''
menu driven code to
1 create account
2 withdraw
3 deposite
4 check balance

1---->create by takingvalues and auto generate account number
2/3/4--->ask account number ,search account and then operate
'''
# author: Surya Dev Singh Jamwal
# date: 06 Apr 2023
# Title/purpose: mini bank-teller app
def notnegamtchk(amt):
    return amt>=0

class Bankacc:
    sban=4000
    def __init__(self, acholdernam, idprfnum):
        Bankacc.sban+=1
        self.__sbaccn=Bankacc.sban
        print("New Account created, SB Acc No :", self.__sbaccn)
        self.__accholdername=acholdernam
        self.__idproofnumber=idprfnum
        self.__balance=2500
        print("Rs. 2500 credited upon account opening of a/c no.", self.__sbaccn)

    def getaccno(self):
        return self.__sbaccn

    def withdrawalction(self, wamt, accno):
        if notnegamtchk(wamt):
            if self.__balance-wamt>2000:
                self.__balance-=wamt
                print("Withdrawal of", wamt, "from a/c", self.__sbaccn, ", updated balance:", self.__balance)
            else:
                print("A/c no:", self.__sbaccn, "has Insufficient Balance")

    def depositaction(self, damt, accno):
        if notnegamtchk(damt):
            self.__balance+=damt
            print("Deposit of", damt, "from a/c", self.__sbaccn, ", updated balance:", self.__balance)

    def balchk(self):
        print("a/c no:", self.__sbaccn, "balance:", self.__balance)


sbal=[] # list to store accounts
def validateaccnum(an):
    for ele in sbal:
        if ele.getaccno()==an:
            return True
            break
    return False

while True:
    print("\n0. Exit\n1. Create New Account\n2. Withdrawal\n3. Deposit\n4. Check Balance")
    inp=int(input(" : "))
    if inp==1: # create new account
        ba=Bankacc(input("Enter account holder name:"), int(input("Enter ID proof number:")))
        sbal.append(ba)
    elif inp==2: # Withdrawal
        an=int(input("Enter account number to withdraw: "))
        if validateaccnum(an):
            amt=int(input("Enter withdrawal amount : "))
            for ele in range(len(sbal)):
                if sbal[ele].getaccno()==an:
                    sbal[ele].withdrawalction(amt, an)
        else:
            print("Invalid account number")
    elif inp==3: # deposit
        an=int(input("Enter account number to deposit: "))
        if validateaccnum(an):
            amt=int(input("Enter deposit amount : "))
            for ele in range(len(sbal)):
                if sbal[ele].getaccno()==an:
                    sbal[ele].depositaction(amt, an)
        else:
            print("Invalid account number")
    elif inp==4: # check balance
        an=int(input("Enter account number to Check balance: "))
        if validateaccnum(an):
            for ele in range(len(sbal)):
                if sbal[ele].getaccno()==an:
                    sbal[ele].balchk()
        else:
            print("Invalid account number")
    elif inp==0:
        print("\nExiting Program, Byee")
        break
    else:
        print("\nInvalid Input, Try Again")
```

```
0. Exit
1. Create New Account
2. Withdrawal
3. Deposit
4. Check Balance
 : 5

Invalid Input, Try Again

0. Exit
1. Create New Account
2. Withdrawal
3. Deposit
4. Check Balance
 : 0
```

```
Exiting Program, Byee
```