LOKESHVISWA M

26/07/2025

1.Create a class BankAccount in Python with private attributes __accountno,__name, __balance.

Add

parameterized constructor

methods:

deposit(amount)

withdraw(amount)

set_accountno

get_accountno

set_name

get_name

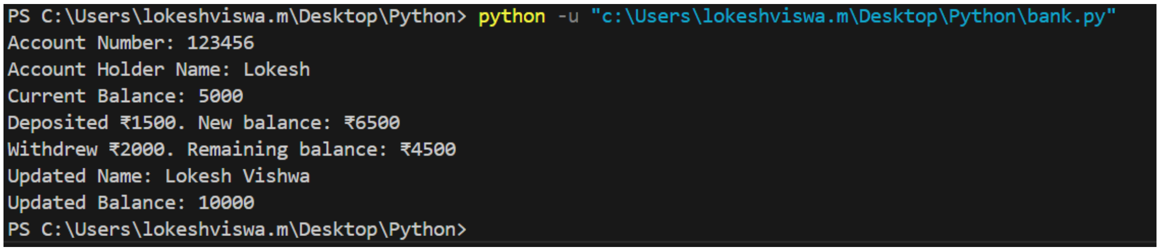
get_balance()

set_balance()

CODE:

```
class BankAccount:
    def __init__(self, accountno, name, balance):
        self.__accountno = accountno
        self.__name = name
        self.__balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            print(f"Deposited ₹{amount}. New balance: ₹{self.__balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.__balance:
                self.__balance -= amount
                print(f"Withdrew ₹{amount}. Remaining balance: ₹{self.__balance}")
            else:
                print("Insufficient balance.")
        else:
            print("Withdrawal amount must be positive.")

    def set_accountno(self, accountno):
        self.__accountno = accountno

    def get_accountno(self):
        return self.__accountno

    def set_name(self, name):
        self.__name = name

    def get_name(self):
        return self.__name

    def set_balance(self, balance):
        if balance >= 0:
            self.__balance = balance
        else:
            print("Balance cannot be negative.")

    def get_balance(self):
        return self.__balance


account = BankAccount(123456, "Lokesh", 5000)

print("Account Number:", account.get_accountno())
print("Account Holder Name:", account.get_name())
print("Current Balance:", account.get_balance())

account.deposit(1500)
account.withdraw(2000)

account.set_name("Lokesh Vishwa")
account.set_balance(10000)

print("Updated Name:", account.get_name())
print("Updated Balance:", account.get_balance())
```

OUTPUT:



```
PS C:\Users\lokeshviswa.m\Desktop\Python> python -u "c:\Users\lokeshviswa.m\Desktop\Python\bank.py"
Account Number: 123456
Account Holder Name: Lokesh
Current Balance: 5000
Deposited ₹1500. New balance: ₹6500
Withdrew ₹2000. Remaining balance: ₹4500
Updated Name: Lokesh Vishwa
Updated Balance: 10000
PS C:\Users\lokeshviswa.m\Desktop\Python>
```

2.How will you define a static method in Python?Explore and give an example.

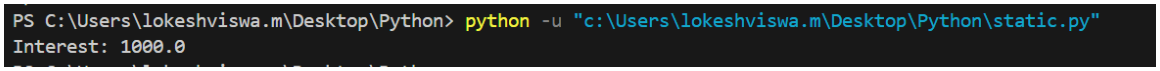
In Python, a static method is defined using the @staticmethod decorator. It belongs to the class, not the instance, and does not access self or cls. It's like a regular function, but lives in the class's namespace.

```
class MyClass:

    @staticmethod

    def my_static_method():

        # No access to self or cls

        print
```

EXAMPLE:

```
class BankUtility:

    @staticmethod

    def calculate_interest(principal, rate, time):

        """Simple Interest = (P × R × T) / 100"""

        return (principal * rate * time) / 100


interest = BankUtility.calculate_interest(10000, 5, 2)

print("Interest:", interest)
```

OUTPUT:



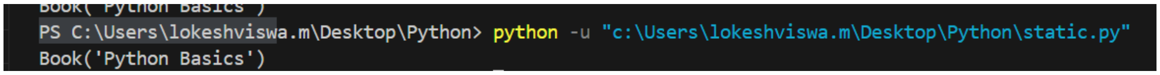
```
PS C:\Users\lokeshviswa.m\Desktop\Python> python -u "c:\Users\lokeshviswa.m\Desktop\Python\static.py"
Interest: 1000.0
```

3.Give examples for dunder methods in Python other than __str__ and __init__ .

1. __repr__()

```
class Book:

    def __init__(self, title):

        self.title = title


    def __repr__(self):

        return f"Book('{self.title}')"


b = Book("Python Basics")

print(repr(b))
```

OUTPUT:

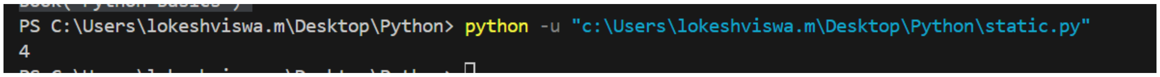


```
Book('Python Basics')
PS C:\Users\lokeshviswa.m\Desktop\Python> python -u "c:\Users\lokeshviswa.m\Desktop\Python\static.py"
Book('Python Basics')
```

2. __len__()

```
class MyList:

    def __init__(self, items):

        self.items = items


    def __len__(self):

        return len(self.items)


ml = MyList([1, 2, 3, 4])

print(len(ml))
```

OUTPUT:



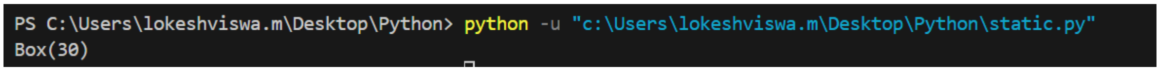
```
Book('Python Basics')
PS C:\Users\lokeshviswa.m\Desktop\Python> python -u "c:\Users\lokeshviswa.m\Desktop\Python\static.py"
4
PS C:\Users\lokeshviswa.m\Desktop\Python> 
```

3. __add__()

class Box:

```
    def __init__(self, volume):

        self.volume = volume


    def __add__(self, other):

        return Box(self.volume + other.volume)


    def __repr__(self):

        return f"Box({self.volume})"


b1 = Box(10)

b2 = Box(20)

b3 = b1 + b2

print(b3)
```

OUTPUT:



```
PS C:\Users\lokeshviswa.m\Desktop\Python> python -u "c:\Users\lokeshviswa.m\Desktop\Python\static.py"
Box(30)
```

4) Explore some supervised and unsupervised models in ML.

Supervised Learning Models

Supervised learning uses labeled data (i.e., inputs with known outputs) to train a model.

1. **Linear Regression**

Type: Regression

Use: Predicting continuous values

Example: Predict house prices based on size, location, etc.

from sklearn.linear_model import LinearRegression

2. **Logistic Regression**

Type: Classification

Use: Predict binary outcomes (0 or 1)

Example: Spam detection, disease prediction

from sklearn.linear_model import LogisticRegression

3. **Decision Tree**

Type: Classification / Regression

Use: Easy to interpret; splits data based on rules

Example: Loan approval, exam pass/fail prediction

from sklearn.tree import DecisionTreeClassifier

4. **Random Forest**

Type: Classification / Regression

Use: Ensemble of decision trees → more accurate

Example: Credit scoring, fraud detection

from sklearn.ensemble import RandomForestClassifier

5. **Support Vector Machine (SVM)**

Type: Classification

Use: Finds the best separating boundary

Example: Face detection, text classification

from sklearn.svm import SVC

Unsupervised Learning Models

Unsupervised learning works on unlabeled data to find hidden patterns or groupings.

1. **K-Means Clustering**

Type: Clustering

Use: Groups similar data points into clusters

Example: Customer segmentation, image compression

from sklearn.cluster import KMeans

2. **Hierarchical Clustering**

Type: Clustering

Use: Builds a tree of clusters

Example: Gene analysis, social network grouping

```
from scipy.cluster.hierarchy import dendrogram, linkage
```

3. Principal Component Analysis (PCA)

Type: Dimensionality Reduction

Use: Reduces data features while keeping most variance

Example: Image compression, noise reduction

```
from sklearn.decomposition import PCA
```

4. DBSCAN (Density-Based Spatial Clustering)

Type: Clustering

Use: Finds clusters based on data density

Example: Anomaly detection, spatial data analysis

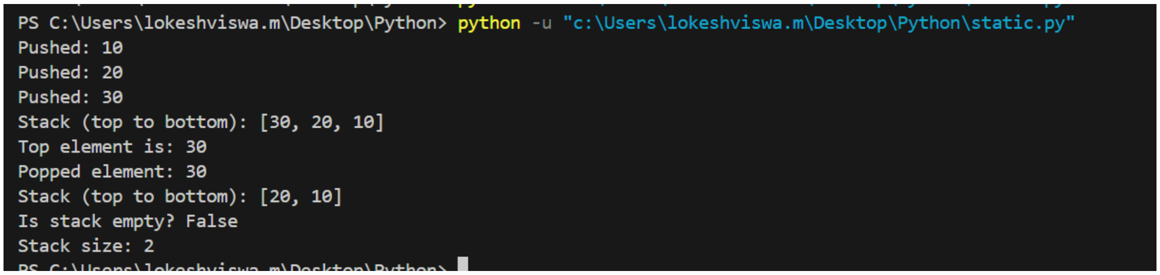
```
from sklearn.cluster import DBSCAN
```

5) Implement Stack with class in Python.

CODE:

```
class Stack:
    def __init__(self):
        self.stack = []



    def push(self, item):
        self.stack.append(item)
        print(f"Pushed: {item}")



    def pop(self):
        if self.is_empty():
            print("Stack is empty. Cannot pop.")
            return None
        return self.stack.pop()
    def peek(self):
        if self.is_empty():
            print("Stack is empty. Nothing to peek.")
            return None
        return self.stack[-1]


    def is_empty(self):
        return len(self.stack) == 0


    def size(self):
        return len(self.stack)


    def display(self):
        print("Stack (top to bottom):", list(reversed(self.stack)))
s = Stack()

s.push(10)
s.push(20)
s.push(30)

s.display()

print("Top element is:", s.peek())
print("Popped element:", s.pop())

s.display()

print("Is stack empty?", s.is_empty())

print("Stack size:", s.size())
```

OUTPUT:



```
PS C:\Users\lokeshviswa.m\Desktop\Python> python -u "c:\Users\lokeshviswa.m\Desktop\Python\static.py"
Pushed: 10
Pushed: 20
Pushed: 30
Stack (top to bottom): [30, 20, 10]
Top element is: 30
Popped element: 30
Stack (top to bottom): [20, 10]
Is stack empty? False
Stack size: 2
PS C:\Users\lokeshviswa.m\Desktop\Python>
```