VIJAY M
weekly 3

27-07-25

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()

BankAccount.py

```
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}. New 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
```

```
if __name__ == "__main__":
    acc = BankAccount(10123456, "Alice", 1000)
    print("Account No:", acc.get_accountno())
    print("Name:", acc.get_name())
    print("Balance:", acc.get_balance())
    acc.set_name("Alice Smith")
    acc.set_balance(1500)
    acc.deposit(500)
    acc.withdraw(300)
    print("\nUpdated Account Details:")
    print("Account No:", acc.get_accountno())
    print("Name:", acc.get_name())
    print("Balance:", acc.get_balance())
```

```
Account No: 10123456
Name: Alice
Balance: 1000
Deposited 500. New balance: 2000
Withdrew 300. New balance: 1700

Updated Account Details:
Account No: 10123456
Name: Alice Smith
Balance: 1700
>
```

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. Static methods belong to the class, not to instances, and do not receive the self or cls parameters by default.

Why Use Static Methods?

They are utility methods that don't need access to instance (self) or class (cls) data.

Used for operations related to the class but not dependent on instance variables.

Syntax

```
class MyClass:
    @staticmethod
    def my_static_method():
        print("This is a static method.")
```

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

In Python, dunder methods (short for "double underscore" methods) are special methods with names surrounded by double underscores (e.g., __len__, __add__). These methods allow you to define or customize the behavior of your objects when they are used with built-in Python operations.

Below are some commonly used dunder methods (excluding __init__ and __str__) with examples:

__repr__ – Official string representation (used for debugging)

```
class Person:
    def __init__(self, name):
        self.name = name
    def __repr__(self):
        return f"Person(name='{self.name}')"
p = Person("Alice")
print(repr(p))    # Output: Person(name='Alice')
```

__len__ – Define behavior for len() function

```
class Basket:
    def __init__(self, items):
        self.items = items
    def __len__(self):
        return len(self.items)
b = Basket(['apple', 'banana', 'orange'])
```

```
print(len(b))    # Output: 3
```

_eq__ – Define behavior for equality comparison (==)

```
class Book:
    def __init__(self, title):
        self.title = title
    def __eq__(self, other):
        return self.title == other.title
b1 = Book("Python Basics")
b2 = Book("Python Basics")
print(b1 == b2)    # Output: True
```

__call__ – Make an instance callable like a function

```
class Greeter:
    def __init__(self, name):
        self.name = name
    def __call__(self):
        print(f"Hello, {self.name}!")
g = Greeter("Alice")
g()    # Output: Hello, Alice!
```

__del__ – Destructor (called when object is deleted)

```
class Sample:
    def __del__(self):
        print("Object is being destroyed")
obj = Sample()
del obj    # Output: Object is being destroyed
```

Implement Stack with class in Python.

```
class Stack:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return len(self.items) == 0
    def push(self, item):
        self.items.append(item)
        print(f"Pushed: {item}")
    def pop(self):
        if not self.is_empty():
            removed = self.items.pop()
            print(f"Popped: {removed}")
            return removed
        else:
            print("Stack is empty.")
            return None
    def peek(self):
        if not self.is_empty():
            return self.items[-1]
        else:
            print("Stack is empty.")
            return None
    def size(self):
        return len(self.items)
```

```
    def display(self):
        if not self.is_empty():
            print("Stack (top to bottom):", self.items[::-1])
        else:
            print("Stack is empty.")
if __name__ == "__main__":
    s = Stack()
    s.push(10)
    s.push(20)
    s.push(30)
    s.display()
    print("Top item:", s.peek())
    s.pop()
    s.display()
    print("Stack size:", s.size())
```

```
========== RESTART: C:/Users/vijay.m/Documents/Weekly
Pushed: 10
Pushed: 20
Pushed: 30
Stack (top to bottom): [30, 20, 10]
Top item: 30
Popped: 30
Stack (top to bottom): [20, 10]
Stack size: 2
```

Explore some supervised and unsupervised models in ML.

Supervised Learning Models

Supervised learning uses labeled data — data where the input and output (target) are both known — to train a model.

Types:

Classification: Predict categories (e.g., spam or not spam)

Regression: Predict continuous values (e.g., price, temperature)

Common Supervised Models:

Linear Regression

Logistic Regression

Decision Tree

Random Forest

Support Vector Machine (SVM)

K-Nearest Neighbors (KNN)

Unsupervised Learning Models

🔍 Unsupervised learning uses unlabeled data — the model tries to find hidden patterns or structure in the data without guidance.

Types:

Clustering: Grouping similar data points

Dimensionality Reduction: Reducing number of features for simplicity/visualization

Common Unsupervised Models:

K-Means Clustering

DBSCAN

Principal Component Analysis (PCA)

Hierarchical Clustering

Gaussian Mixture Models (GMM)