

Name: Rajani Sankapal  
Roll No: C-19  
Course No:SPP II(Python)

Experiment No :9

**Title:Write Python programs to create and use classes,constructors, and destructors.**

**1. Student Class: Create a class Student with name and marks. Include methods to display details, change school name (class method), calculate grade (static method), and a destructor message.**

```
class Student:
    school = "GHS"
    def __init__(self, name, marks):
        self.name, self.marks = name, marks
        print(f"Constructed: {name}")
    @classmethod
    def change_school(cls, n): cls.school = n
    @staticmethod
    def get_grade(m): return 'A' if m >= 90 else 'B'
    def show(self):
        print(f"{self.name}: {self.marks} ({self.get_grade(self.marks)} @ {Student.school})")
    def __del__(self):
        print(f"Destructed: {self.name}")
s1 = Student("Alice", 92)
s1.show(); Student.change_school("New Academy")
s2 = Student("Bob", 65)
s2.show()
```

Constructed: Alice  
Alice: 92 (A @ GHS)  
Constructed: Bob  
Bob: 65 (B @ New Academy)  
Destructed: Alice  
Destructed: Bob

**2. Employee Salary: Create a class Employee with emp\_id, name, and salary. Include methods to calculate yearly salary, update company name (class method), calculate bonus (static method), and destructor message.**

```
class Employee:
    company = "TechCo"
    def __init__(self, i, n, s):
        self.id, self.name, self.salary = i, n, s
        print(f"Hired: {n}")
    @classmethod
    def update_company(cls, new_c): cls.company = new_c
    @staticmethod
    def calculate_bonus(s): return s * 0.10 if s > 50000 else 0
    def show_details(self):
        y = self.salary * 12; b = Employee.calculate_bonus(self.salary)
        print(f"{self.name} ({self.id}): Yearly=${y:,.0f}, Bonus=${b:,.0f} @ {Employee.company}")
    def __del__(self): print(f"Terminated: {self.name}")
# Demonstration (All on three lines)
e1 = Employee(101, "Jane", 60000); e2 = Employee(102, "Mark", 45000)
Employee.update_company("NewCorp")
e1.show_details(); e2.show_details(); del e1
```

Hired: Jane  
Hired: Mark  
Jane (101): Yearly=\$720,000, Bonus=\$6,000 @ NewCorp  
Mark (102): Yearly=\$540,000, Bonus=\$0 @ NewCorp  
Terminated: Jane  
Terminated: Mark

Name: Rajani Sankapal

Roll No: C-19

Course No:SPP II(Python)

**3. Bank Account: Create a class BankAccount with acc\_no and balance. Include methods for deposit/withdrawal, update bank branch (class method), calculate interest (static method), and destructor message.**

```
class BankAccount:
    branch = "Main St"
    def __init__(self, acc, bal=0.0): self.acc_no, self.balance = acc, bal; print(f"Open: {acc}")
    def deposit(self, amt):
        if amt > 0: self.balance += amt; print(f"Dep: ${amt:,.2f}. Bal: ${self.balance:,.2f}")
        else: print("Deposit failed.")
    def withdraw(self, amt):
        if 0 < amt <= self.balance: self.balance -= amt; print(f"Wdr: ${amt:,.2f}. Bal: ${self.balance:,.2f}")
        else: print("Withdrawal failed.")
    @classmethod
    def update_bank_branch(cls, new_b): cls.branch = new_b; print(f"Branch updated to {new_b}")
    @staticmethod
    def calculate_interest(p, r=0.035): return p * r
    def __del__(self): print(f"Closed: {self.acc_no}")
    # Demonstration
    a1 = BankAccount(1001, 500); a2 = BankAccount(1002, 1200)
    a1.deposit(250); a2.withdraw(50); a1.withdraw(900) # Fails due to
    insufficient funds
    BankAccount.update_bank_branch("Downtown HQ")
    print(f"A1 Int: ${BankAccount.calculate_interest(a1.balance):,.2f}. A2 Branch: {a2.branch}")
    del a1
```

```
Open: 1001
Open: 1002
Dep: $250.00. Bal: $750.00
Wdr: $50.00. Bal: $1,150.00
Withdrawal failed.
Branch updated to Downtown HQ
A1 Int: $26.25. A2 Branch: Downtown HQ
Closed: 1001
Closed: 1002
```

**4. Rectangle Operations: Create a class Rectangle with length and width. Include methods to calculate area/perimeter, update unit (class method), compare two rectangles (static method), and destructor message.**

```
class Rectangle:
    unit = "cm"
    def __init__(self, l, w): self.length, self.width = l, w; print(f"Init: {l}x{w}")
    def calculate_area(self): return self.length * self.width
    def calculate_perimeter(self): return 2 * (self.length + self.width)
    @classmethod
    def update_unit(cls, new_unit): cls.unit = new_unit; print(f"Unit set to {new_unit}")
    @staticmethod
    def compare_rectangles(r1, r2): return r1.calculate_area() == r2.calculate_area()
    def __del__(self): print(f"Destruct: {self.length}x{self.width}")
    r1 = Rectangle(10, 5); r2 = Rectangle(8, 7)
    print(f"R1 Area: {r1.calculate_area()} {Rectangle.unit}²")
    print(f"R2 Perimeter: {r2.calculate_perimeter()} {Rectangle.unit}")
    print(f"Equal Area? {Rectangle.compare_rectangles(r1, r2)}")
    Rectangle.update_unit("m")
    del r2
```

```
Init: 10x5
Init: 8x7
R1 Area: 50 cm²
R2 Perimeter: 30 cm
Equal Area? False
Unit set to m
Destruct: 8x7
Destruct: 10x5
```

Name: Rajani Sankapal

Roll No: C-19

Course No: SPP II(Python)

**5. Car Information: Create a class Car with brand, model, and price. Include methods to display details, update total cars (class method), calculate discounted price (static method), and destructor message.**

```
class Car:
    total_cars = 0
    def __init__(self, b, m, p):
        self.brand, self.model, self.price = b, m, p
        Car.total_cars += 1
        print(f"Constructed: {b} {m}")
    def display_details(self):
        print(f"{self.brand} {self.model} - ${self.price:,.0f}. Total in stock: {Car.total_cars}")
    @classmethod
    def update_total_cars(cls, count):
        cls.total_cars = count
        print(f"Class Update: Stock count set to {count}")
    @staticmethod
    def calculate_discounted_price(price, discount):
        return price * (1 - discount)
    def __del__(self):
        Car.total_cars -= 1
        print(f"Destructed: {self.brand} {self.model}. Remaining: {Car.total_cars}")

# Demonstration
c1 = Car("Honda", "Civic", 25000); c2 = Car("Tesla", "Model Y", 55000)
c1.display_details()
print(f"Discounted Price (15%): ${Car.calculate_discounted_price(c2.price, 0.15):,.0f}")
Car.update_total_cars(100)
c2.display_details()
del c1
```

Constructed: Honda Civic  
Constructed: Tesla Model Y  
Honda Civic - \$25,000. Total in stock: 2  
Discounted Price (15%): \$46,750  
Class Update: Stock count set to 100  
Tesla Model Y - \$55,000. Total in stock: 100  
Destructed: Honda Civic. Remaining: 99  
Destructed: Tesla Model Y. Remaining: 98

**6. Circle Calculations: Create a class Circle with radius. Include methods to calculate area/circumference, update  $\pi$  value (class method), validate radius (static method), and destructor message.**

```
class Circle:
    PI = 3.14159
    def __init__(self, r): self.radius = r; print(f"Init: r={r}")
    def area(self): return self.PI * self.radius ** 2
    def circumference(self): return 2 * self.PI * self.radius
    @classmethod
    def update_pi(cls, p): cls.PI = p; print(f"PI set to {p}")
    @staticmethod
    def validate_radius(r): return r > 0
    def __del__(self): print(f"Destruct: r={self.radius}")

c1 = Circle(7); c2 = Circle(3)
print(f"C1 Area: {c1.area():.2f}")
print(f"C2 Circumference: {c2.circumference():.2f}")
print(f"Radius 7 is valid: {Circle.validate_radius(7)}")
Circle.update_pi(3.14)
print(f"C1 New Area: {c1.area():.2f}") del c2
```

Init: r=7  
Init: r=3  
C1 Area: 153.94  
C2 Circumference: 18.85  
Radius 7 is valid: True  
PI set to 3.14  
C1 New Area: 153.86  
Destruct: r=3  
Destruct: r=7

Name: Rajani Sankapal

Roll No: C-19

Course No:SPP II(Python)

**7. Book Details: Create a class Book with title, author, and price. Include methods to display details, update publisher (class method), check price limit (static method), and destructor message.**

```
class Book:
    publisher = "Penguin Books"
    def __init__(self, t, a, p): self.title, self.author, self.price = t, a, p; print(f"Constructed: {t}")
    def display_details(self): print(f"'{self.title}' by {self.author} - ${self.price:.2f}. Publisher: {Book.publisher}")
    @classmethod
    def update_publisher(cls, new_pub): cls.publisher = new_pub; print(f"Class Update: Publisher set to {new_pub}")
    @staticmethod
    def check_price_limit(price, limit=30.0): return price < limit
    def __del__(self): print(f"Destructed: {self.title}")
b1 = Book("1984", "G. Orwell", 12.50)
b2 = Book("War and Peace", "L. Tolstoy", 45.99)
b1.display_details()
print(f"Price under $30 limit? {Book.check_price_limit(b2.price)}")
Book.update_publisher("Vintage Classics")
b2.display_details()
del b1
```

```
Constructed: 1984
Constructed: War and Peace
'1984' by G. Orwell - $12.50. Publisher: Penguin Books
Price under $30 limit? False
Class Update: Publisher set to Vintage Classics
'War and Peace' by L. Tolstoy - $45.99. Publisher: Vintage Classics
Destructed: 1984
Destructed: War and Peace
```

**8. Laptop Specifications: Create a class Laptop with brand, model, and ram. Include methods to display specs, update warranty (class method), check RAM sufficiency (static method), and destructor message.**

```
class Laptop:
    warranty_years = 2
    def __init__(self, b, m, r): self.brand, self.model, self.ram = b, m, r;
    print(f"Constructed: {b} {m}")
    def display_specs(self): print(f"{self.brand} {self.model} ({self.ram}GB). Warranty: {Laptop.warranty_years} yrs")
    @classmethod
    def update_warranty(cls, years): cls.warranty_years = years; print(f"Class Update: Warranty set to {years} years")
    @staticmethod
    def check_ram_sufficiency(ram_gb): return ram_gb >= 16
    def __del__(self): print(f"Destructed: {self.brand} {self.model}")
# Demonstration
l1 = Laptop("Dell", "XPS 13", 16)
l2 = Laptop("Apple", "M3 Max", 64)
l1.display_specs()
print(f"Is {l2.ram}GB sufficient (>=16GB)? {Laptop.check_ram_sufficiency(l2.ram)}")
Laptop.update_warranty(3)
l1.display_specs()
del l2
```

```
Constructed: Dell XPS 13
Constructed: Apple M3 Max
Dell XPS 13 (16GB). Warranty: 2 yrs
Is 64GB sufficient (>=16GB)? True
Class Update: Warranty set to 3 years
Dell XPS 13 (16GB). Warranty: 3 yrs
Destructed: Apple M3 Max
Destructed: Dell XPS 13
```

**9. Payroll System: Create a class Payroll with name and basic\_salary. Include methods to calculate total salary, update HRA percentage (class method), calculate tax (static method), and destructor message.**

Name: Rajani Sankapal  
Roll No: C-19  
Course No:SPP II(Python)

```
class Payroll:
    HRA_percent = 0.15 # Default HRA is 15% of basic salary
    def __init__(self, n, s): self.name, self.basic_salary = n, s;
    print(f"Init: {n}")
    def total_salary(self): return self.basic_salary * (1 +
    Payroll.HRA_percent)
    @classmethod
    def update_hra(cls, p): cls.HRA_percent = p; print(f"HRA set to
    {p*100:.0f}%")
    @staticmethod
    def calculate_tax(salary, rate=0.10): return salary * rate
    def __del__(self): print(f"Destruct: {self.name}")
    # Demonstration
    p1 = Payroll("John Doe", 50000)
    ts = p1.total_salary()
    print(f"{p1.name}'s Total Salary: ${ts:,.2f}")
    print(f"Calculated Tax (10%): ${Payroll.calculate_tax(ts):,.2f}")
    Payroll.update_hra(0.20)
    p2 = Payroll("Jane Smith", 70000)
    print(f"{p2.name}'s New Total: ${p2.total_salary():,.2f}")
    del p1
```

```
Init: John Doe
John Doe's Total Salary: $57,500.00
Calculated Tax (10%): $5,750.00
HRA set to 20%
Init: Jane Smith
Jane Smith's New Total: $84,000.00
Destruct: John Doe
Destruct: Jane Smith
```

**10. Temperature Converter: Create a class Temperature with a Celsius value. Include methods to convert to Fahrenheit, set boiling point (class method), validate temperature (static method), and destructor message.**

```
class Temperature:
    BOILING_POINT = 100
    def __init__(self, c): self.celsius = c; print(f"Temp object created for
    {c}°C")
    def to_fahrenheit(self): return (self.celsius * 9/5) + 32
    @classmethod
    def set_boiling_point(cls, bp): cls.BOILING_POINT = bp; print(f"Class
    Update: Boiling point set to {bp}°C")
    @staticmethod
    def validate_temp(celsius): return celsius >= -273.15 # Absolute zero
    def __del__(self): print(f"Destructor called for {self.celsius}°C")
    # Demonstration
    t1 = Temperature(25)
    print(f"25°C is {t1.to_fahrenheit():.2f}°F")
    print(f"Is temp valid? {Temperature.validate_temp(t1.celsius)}")
    Temperature.set_boiling_point(99)
    t2 = Temperature(105)
    print(f"Boiling Point is now: {Temperature.BOILING_POINT}°C")
    del t1
```

```
Temp object created for 25°C
25°C is 77.00°F
Is temp valid? True
Class Update: Boiling point set to 99°C
Temp object created for 105°C
Boiling Point is now: 99°C
Destructor called for 25°C
Destructor called for 105°C
```

**11. Product Management: Create a class Product with name, price, and quantity. Include methods to calculate total cost, update tax rate (class method), validate quantity (static method), and destructor message.**

Name: Rajani Sankapal  
Roll No: C-19  
Course No: SPP II(Python)

class Product:

TAX\_RATE = 0.07 # Default tax rate is 7%

def \_\_init\_\_(self, n, p, q): self.name, self.price, self.quantity = n, p, q; print(f"Product added: {n}")

def calculate\_total\_cost(self):

"""Calculates total cost including tax."""

subtotal = self.price \* self.quantity

return subtotal \* (1 + Product.TAX\_RATE)

@classmethod

def update\_tax\_rate(cls, rate): cls.TAX\_RATE = rate; print(f"Class Update: Tax rate set to {rate\*100:.2f}%")

@staticmethod

def validate\_quantity(q): return q > 0

def \_\_del\_\_(self): print(f"Destructor called for product:  
{self.name}")

# Demonstration

p1 = Product("Laptop Bag", 50.00, 2)

p2 = Product("Mouse Pad", 15.00, 5)

print(f"{p1.name} valid quantity? {Product.validate\_quantity(p1.quantity)}")

print(f"{p1.name} Total Cost: \${p1.calculate\_total\_cost():.2f}")

Product.update\_tax\_rate(0.085) # Update tax to 8.5%

print(f"New Tax Rate: {Product.TAX\_RATE\*100:.2f}%")

print(f"{p2.name} Total Cost: \${p2.calculate\_total\_cost():.2f}")

# Explicit destructor call

del p1

```
Product added: Laptop Bag
Product added: Mouse Pad
Laptop Bag valid quantity? True
Laptop Bag Total Cost: $107.00
Class Update: Tax rate set to 8.50%
New Tax Rate: 8.50%
Mouse Pad Total Cost: $81.38
Destructor called for product: Laptop Bag
Destructor called for product: Mouse Pad
```

**12. Employee Attendance: Create a class Attendance with emp\_name and days\_present. Include methods to display attendance, update total working days (class method), check bonus eligibility (static method), and destructor message.**

class Attendance:

TOTAL\_WORKING\_DAYS = 220 # Class variable for total days

def \_\_init\_\_(self, n, p): self.emp\_name, self.days\_present = n, p; print(f"Attendance recorded for: {n}")

def display\_attendance(self):

"""Displays attendance percentage."""

percent = (self.days\_present / Attendance.TOTAL\_WORKING\_DAYS) \* 100

print(f"{self.emp\_name}: {self.days\_present} days ({percent:.2f}% present)")

@classmethod

def update\_total\_working\_days(cls, days): cls.TOTAL\_WORKING\_DAYS = days; print(f"Class Update: Total working days set to {days}")

@staticmethod

def check\_bonus\_eligibility(days\_present, threshold=0.9):

"""Checks eligibility based on a threshold percentage."""

return (days\_present / Attendance.TOTAL\_WORKING\_DAYS) >= threshold

def \_\_del\_\_(self): print(f"Destructor called for attendance record of: {self.emp\_name}")

# Demonstration

e1 = Attendance("Alice", 200)

e2 = Attendance("Bob", 150)

e1.display\_attendance()

```
Attendance recorded for: Alice
Attendance recorded for: Bob
Alice: 200 days (90.91% present)
Alice eligible for bonus (90% threshold)? True
Class Update: Total working days set to 180
Bob eligible for bonus (90% threshold) after update? False
Bob: 150 days (83.33% present)
Destructor called for attendance record of: Alice
Destructor called for attendance record of: Bob
```

Name: Rajani Sankapal

Roll No: C-19

Course No: SPP II(Python)

```
print(f"Alice eligible for bonus (90% threshold)? {Attendance.check_bonus_eligibility(e1.days_present)}")
Attendance.update_total_working_days(180) # Shorter period
print(f"Bob eligible for bonus (90% threshold) after update?
{Attendance.check_bonus_eligibility(e2.days_present)}")
e2.display_attendance()
del e1
```

**13. CircleMath Operations: Create a class CircleMath with radius. Include methods to calculate area/circumference, update  $\pi$  (class method),**

```
import math
class CircleMath:
    PI = math.pi
    def __init__(self, r): self.radius = r; print(f"Circle object created with R={r}")
    def calculate_area(self): return self.PI * (self.radius ** 2)
    def calculate_circumference(self): return 2 * self.PI * self.radius
    @classmethod
    def update_pi(cls, p): cls.PI = p; print(f"Class Update: PI set to {p}")
    @staticmethod
    def validate_radius(r): return r > 0
    def __del__(self): print(f"Destructor called for CircleMath with R={self.radius}")

# Demonstration
c1 = CircleMath(5)
print(f"Area: {c1.calculate_area():.2f}")
print(f"Circumference: {c1.calculate_circumference():.2f}")
CircleMath.update_pi(3.14)
c2 = CircleMath(10)
print(f"Is R=10 valid?
{CircleMath.validate_radius(c2.radius)}")
print(f"New Area (using updated PI): {c2.calculate_area():.2f}")
del c1
```

Circle object created with R=5  
Area: 78.54  
Circumference: 31.42  
Class Update: PI set to 3.14  
Circle object created with R=10  
Is R=10 valid? True  
New Area (using updated PI): 314.00  
Destructor called for CircleMath with R=5  
Destructor called for CircleMath with R=10

**14. Student Marks: Create a class StudentMarks with name and marks. Include methods to calculate total and average, update passing marks (class method), assign grade (static method), and destructor message.**

```
class StudentMarks:
    PASSING_MARKS = 40 # Default passing mark per subject
    def __init__(self, n, m): self.name, self.marks = n, m; print(f"Record created for: {n}")
    def total_avg(self):
        """Calculates total and average marks."""
        total, avg = sum(self.marks), sum(self.marks) / len(self.marks)
        return total, avg
    @classmethod
    def update_passing_marks(cls, marks): cls.PASSING_MARKS = marks; print(f"Class Update: Passing
marks set to {marks}")
    @staticmethod
    def assign_grade(avg):
        """Assigns a letter grade based on average."""
```

Name: Rajani Sankapal

Roll No: C-19

Course No: SPP II(Python)

```
return 'A+' if avg >= 90 else ('A' if avg >= 75 else ('B' if avg >= 60 else 'C'))
```

```
def __del__(self): print(f"Destructor called for record of: {self.name}")
```

```
# Demonstration
```

```
s1 = StudentMarks("John Doe", [85, 90, 78])
```

```
t, a = s1.total_avg()
```

```
print(f"{s1.name}: Avg={a:.2f}. Grade:
```

```
{StudentMarks.assign_grade(a)}")
```

```
StudentMarks.update_passing_marks(50) # Update passing threshold
```

```
s2 = StudentMarks("Jane Smith", [45, 55, 60])
```

```
t, a = s2.total_avg()
```

```
print(f"New Pass: {StudentMarks.PASSING_MARKS}. Jane's Grade: {StudentMarks.assign_grade(a)}")
```

```
del s1
```

```
Record created for: John Doe
```

```
John Doe: Avg=84.33. Grade: A
```

```
Class Update: Passing marks set to 50
```

```
Record created for: Jane Smith
```

```
New Pass: 50. Jane's Grade: C
```

```
Destructor called for record of: John Doe
```

```
Destructor called for record of: Jane Smith
```

**15. Vehicle Information: Create a class Vehicle with type, brand, and speed.**

**Include methods to display details, update total vehicle count (class method),**

**check legal speed (static method), and destructor message.**

```
class Vehicle:
```

```
TOTAL_VEHICLES = 0
```

```
LEGAL_SPEED_LIMIT = 120 # Default limit in km/h
```

```
def __init__(self, t, b, s): self.type, self.brand, self.speed = t, b, s; Vehicle.TOTAL_VEHICLES += 1;
```

```
print(f"Vehicle registered: {b}")
```

```
def display_details(self): print(f"[{self.type}]
```

```
{self.brand}, Speed: {self.speed} km/h. Total fleet:
```

```
{Vehicle.TOTAL_VEHICLES}")
```

```
@classmethod
```

```
def update_total_vehicle_count(cls, count):
```

```
cls.TOTAL_VEHICLES = count; print(f"Class
```

```
Update: Total vehicle count set to {count}")
```

```
@staticmethod
```

```
def check_legal_speed(current_speed, limit=None):
```

```
"""Checks if the speed is below the legal limit."""
```

```
limit = limit if limit is not None else Vehicle.LEGAL_SPEED_LIMIT
```

```
return current_speed <= limit
```

```
def __del__(self): Vehicle.TOTAL_VEHICLES -= 1; print(f"Destructor called for vehicle: {self.brand}")
```

```
# Demonstration
```

```
v1 = Vehicle("Car", "Toyota", 90)
```

```
v2 = Vehicle("Truck", "Volvo", 130)
```

```
v1.display_details()
```

```
print(f"Toyota speed is legal? {Vehicle.check_legal_speed(v1.speed)}")
```

```
print(f"Volvo speed is legal? {Vehicle.check_legal_speed(v2.speed)}")
```

```
Vehicle.update_total_vehicle_count(10) # Update fleet size
```

```
v2.display_details()
```

```
del v1
```

```
Vehicle registered: Toyota
```

```
Vehicle registered: Volvo
```

```
[Car] Toyota, Speed: 90 km/h. Total fleet: 2
```

```
Toyota speed is legal? True
```

```
Volvo speed is legal? False
```

```
Class Update: Total vehicle count set to 10
```

```
[Truck] Volvo, Speed: 130 km/h. Total fleet: 10
```

```
Destructor called for vehicle: Toyota
```

```
Destructor called for vehicle: Volvo
```