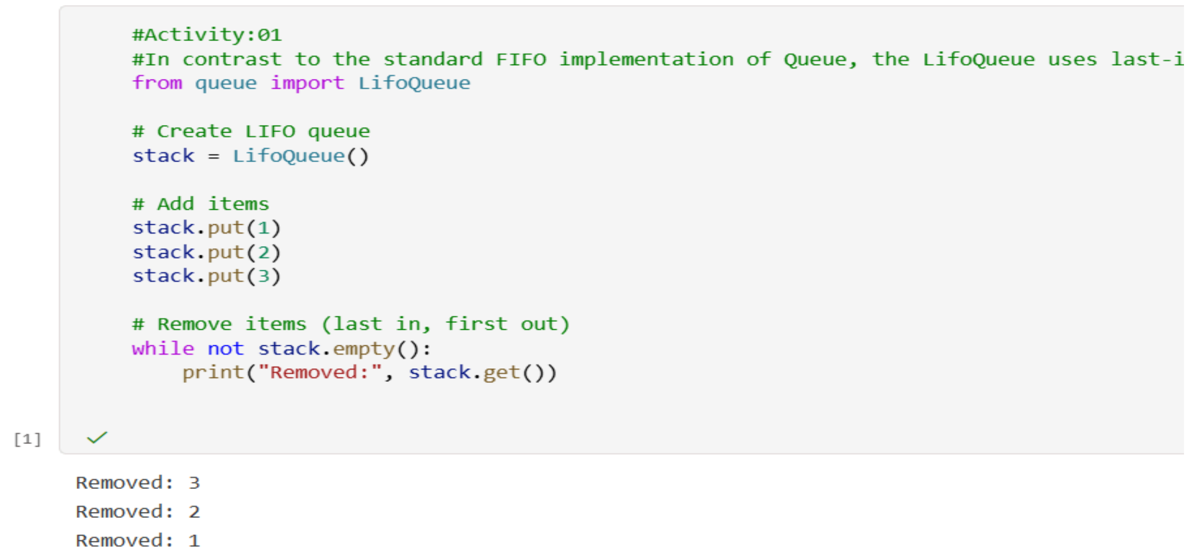
LAB#04

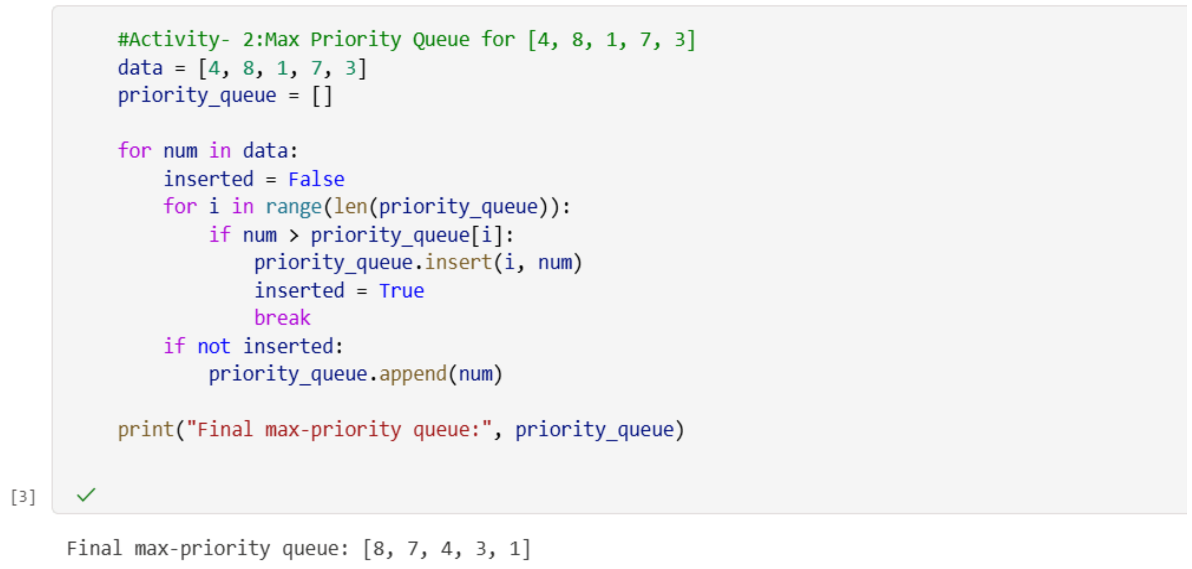
IMPLEMENTING PRIORITY QUEUE



```
#Activity:01
#In contrast to the standard FIFO implementation of Queue, the LifoQueue uses last-i
from queue import LifoQueue

# Create LIFO queue
stack = LifoQueue()

# Add items
stack.put(1)
stack.put(2)
stack.put(3)

# Remove items (last in, first out)
while not stack.empty():
    print("Removed:", stack.get())
```

[1] ✓

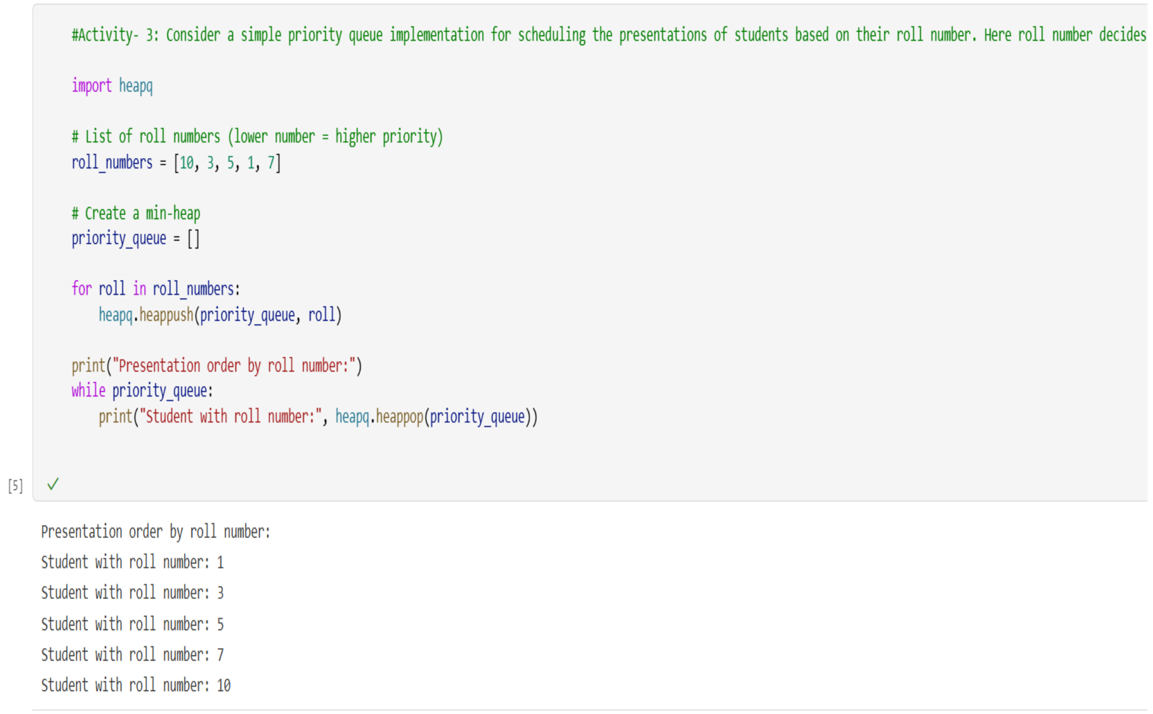
```
Removed: 3
Removed: 2
Removed: 1
```



```
#Activity- 2:Max Priority Queue for [4, 8, 1, 7, 3]
data = [4, 8, 1, 7, 3]
priority_queue = []

for num in data:
    inserted = False
    for i in range(len(priority_queue)):
        if num > priority_queue[i]:
            priority_queue.insert(i, num)
            inserted = True
            break
    if not inserted:
        priority_queue.append(num)

print("Final max-priority queue:", priority_queue)
```

[3] ✓

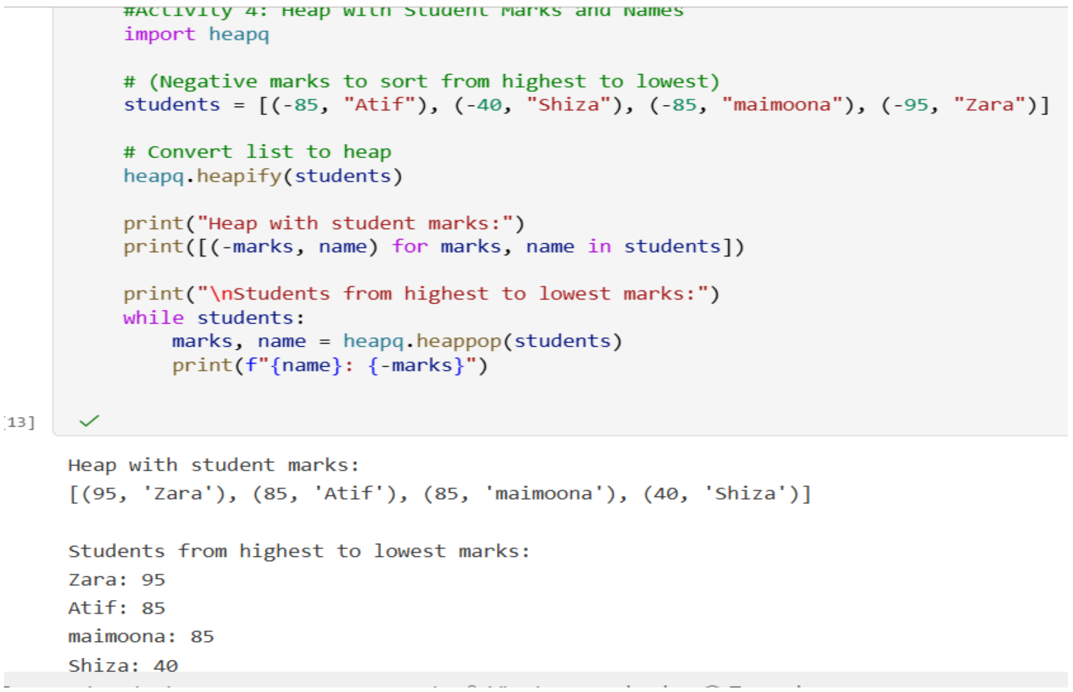
```
Final max-priority queue: [8, 7, 4, 3, 1]
```



```
#Activity- 3: Consider a simple priority queue implementation for scheduling the presentations of students based on their roll number. Here roll number decides

import heapq

# List of roll numbers (lower number = higher priority)
roll_numbers = [10, 3, 5, 1, 7]

# Create a min-heap
priority_queue = []

for roll in roll_numbers:
    heapq.heappush(priority_queue, roll)

print("Presentation order by roll number:")
while priority_queue:
    print("Student with roll number:", heapq.heappop(priority_queue))
```

[5] ✓

```
Presentation order by roll number:
Student with roll number: 1
Student with roll number: 3
Student with roll number: 5
Student with roll number: 7
Student with roll number: 10
```



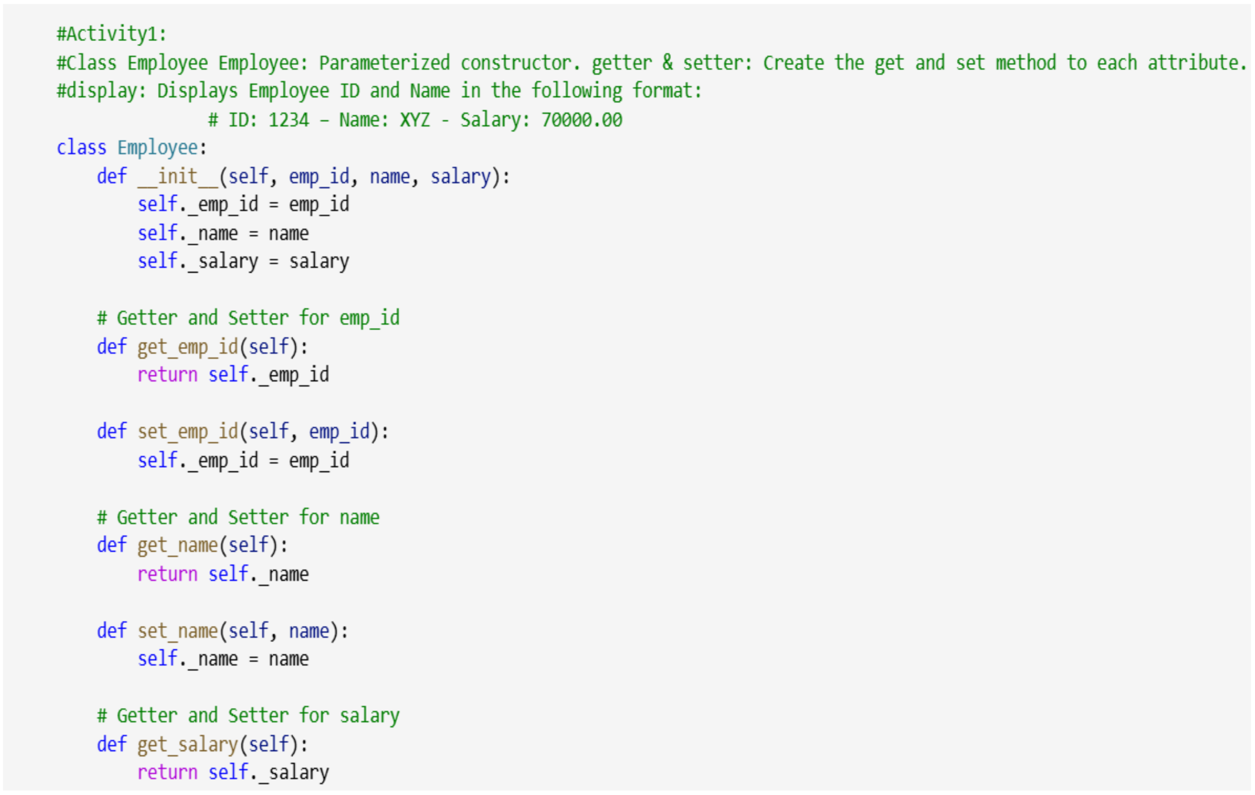
```
#Activity 4: Heap with Student Marks and Names
import heapq

# (Negative marks to sort from highest to lowest)
students = [(-85, "Atif"), (-40, "Shiza"), (-85, "maimoona"), (-95, "Zara")]

# Convert list to heap
heapq.heapify(students)

print("Heap with student marks:")
print([(-marks, name) for marks, name in students])

print("\nStudents from highest to lowest marks:")
while students:
    marks, name = heapq.heappop(students)
    print(f"{name}: {-marks}")
```

13] ✓

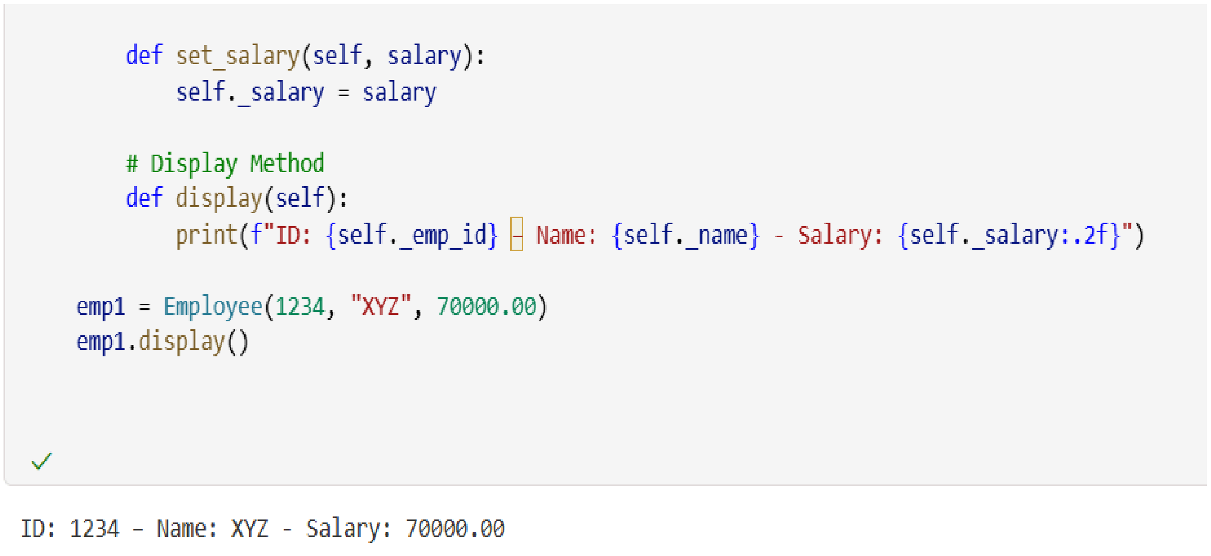
```
Heap with student marks:
[(95, 'Zara'), (85, 'Atif'), (85, 'maimoona'), (40, 'Shiza')]

Students from highest to lowest marks:
Zara: 95
Atif: 85
maimoona: 85
Shiza: 40
```

LAB#03

CLASS & OBJECT:

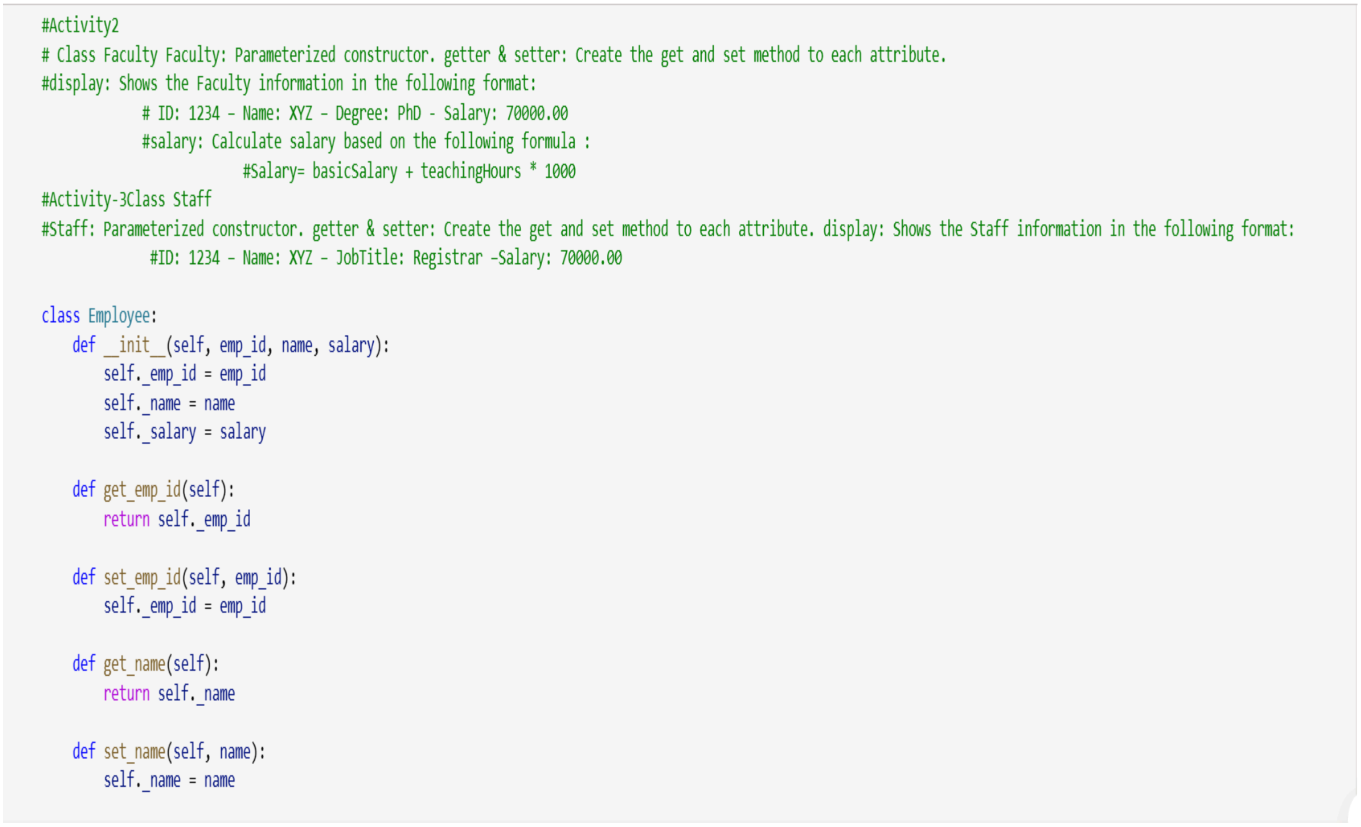


```
#Activity1:
#Class Employee Employee: Parameterized constructor. getter & setter: Create the get and set method to each attribute.
#display: Displays Employee ID and Name in the following format:
               # ID: 1234 – Name: XYZ - Salary: 70000.00
class Employee:
    def __init__(self, emp_id, name, salary):
        self._emp_id = emp_id
        self._name = name
        self._salary = salary

    # Getter and Setter for emp_id
    def get_emp_id(self):
        return self._emp_id

    def set_emp_id(self, emp_id):
        self._emp_id = emp_id

    # Getter and Setter for name
    def get_name(self):
        return self._name

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

    # Getter and Setter for salary
    def get_salary(self):
        return self._salary
```

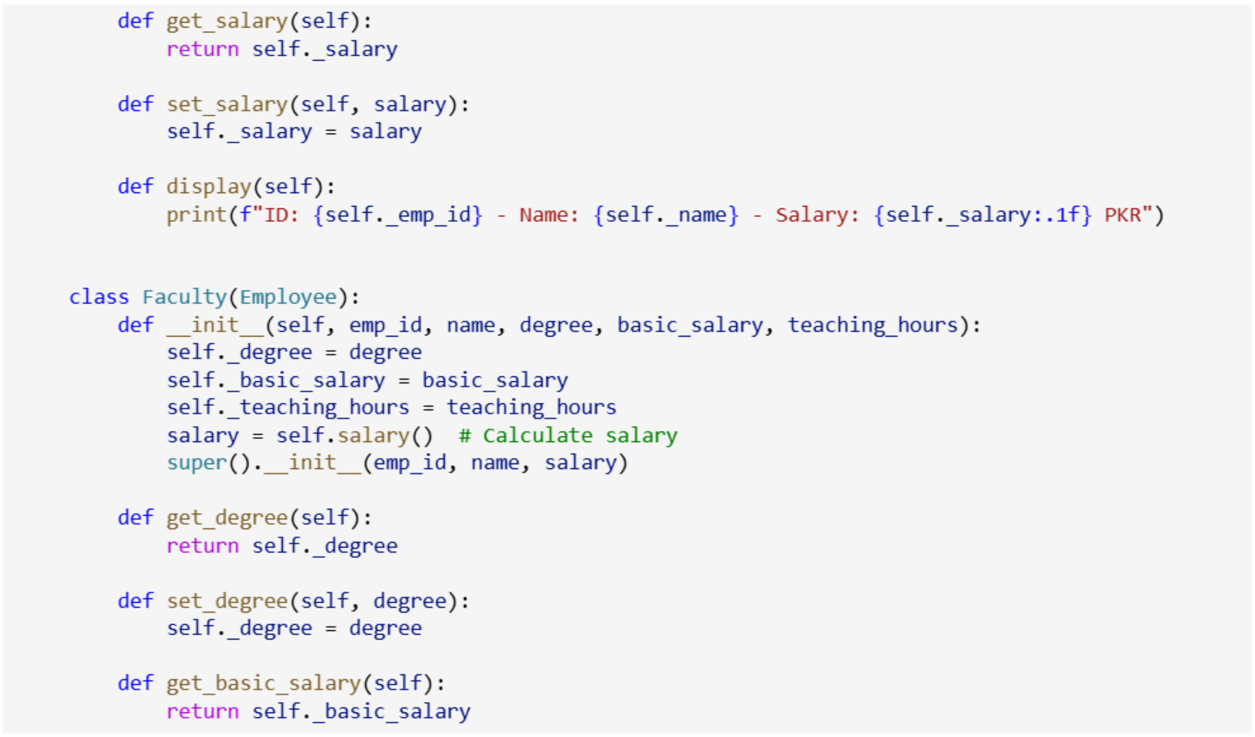


```
    def set_salary(self, salary):
        self._salary = salary

    # Display Method
    def display(self):
        print(f"ID: {self._emp_id} – Name: {self._name} - Salary: {self._salary:.2f}")

emp1 = Employee(1234, "XYZ", 70000.00)
emp1.display()
```

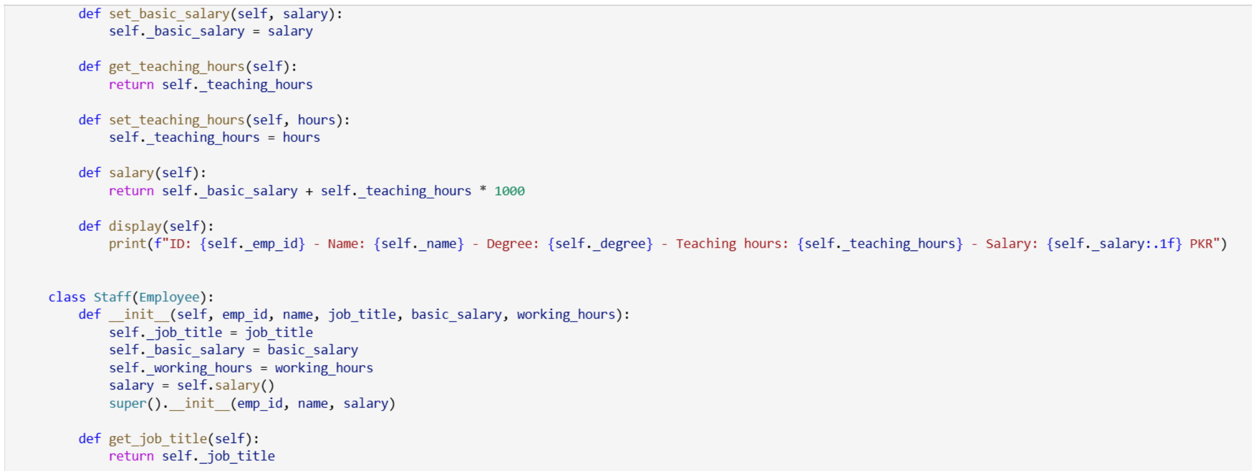
```
ID: 1234 – Name: XYZ - Salary: 70000.00
```



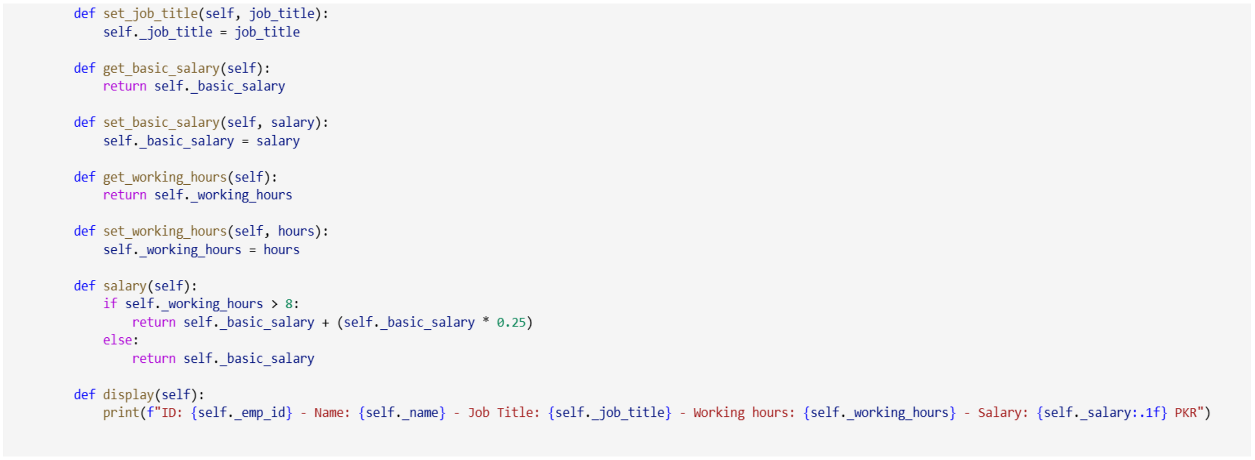
```
#Activity2
# Class Faculty Faculty: Parameterized constructor. getter & setter: Create the get and set method to each attribute.
#display: Shows the Faculty information in the following format:
             # ID: 1234 - Name: XYZ - Degree: PhD - Salary: 70000.00
             #salary: Calculate salary based on the following formula :
                          #Salary= basicSalary + teachingHours * 1000
#Activity-3Class Staff
#Staff: Parameterized constructor. getter & setter: Create the get and set method to each attribute. display: Shows the Staff information in the following format:
              #ID: 1234 - Name: XYZ - JobTitle: Registrar -Salary: 70000.00

class Employee:
    def __init__(self, emp_id, name, salary):
        self._emp_id = emp_id
        self._name = name
        self._salary = salary

    def get_emp_id(self):
        return self._emp_id

    def set_emp_id(self, emp_id):
        self._emp_id = emp_id

    def get_name(self):
        return self._name

    def set_name(self, name):
        self._name = name
```



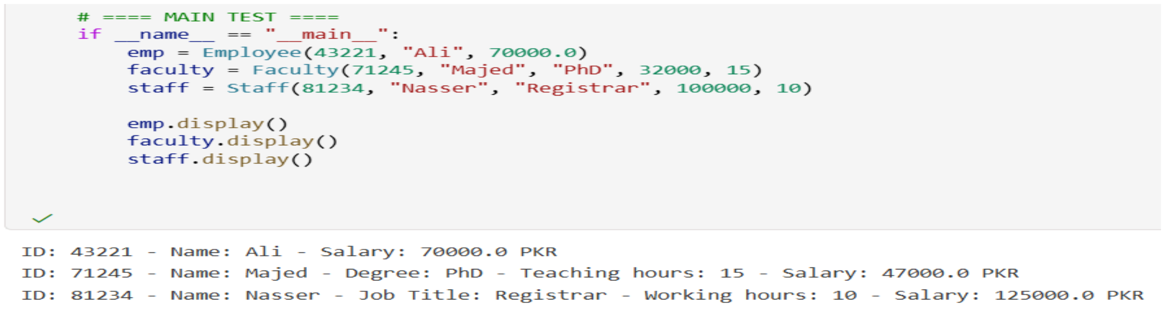
```
    def get_salary(self):
        return self._salary

    def set_salary(self, salary):
        self._salary = salary

    def display(self):
        print(f"ID: {self._emp_id} - Name: {self._name} - Salary: {self._salary:.1f} PKR")


class Faculty(Employee):
    def __init__(self, emp_id, name, degree, basic_salary, teaching_hours):
        self._degree = degree
        self._basic_salary = basic_salary
        self._teaching_hours = teaching_hours
        salary = self.salary()  # Calculate salary
        super().__init__(emp_id, name, salary)

    def get_degree(self):
        return self._degree

    def set_degree(self, degree):
        self._degree = degree

    def get_basic_salary(self):
        return self._basic_salary
```



```python
    def set_basic_salary(self, salary):
        self._basic_salary = salary

    def get_teaching_hours(self):
        return self._teaching_hours

    def set_teaching_hours(self, hours):
        self._teaching_hours = hours

    def salary(self):
        return self._basic_salary + self._teaching_hours * 1000

    def display(self):
        print(f"ID: {self._emp_id} - Name: {self._name} - Degree: {self._degree} - Teaching hours: {self._teaching_hours} - Salary: {self._salary:.1f} PKR")


class Staff(Employee):
    def __init__(self, emp_id, name, job_title, basic_salary, working_hours):
        self._job_title = job_title
        self._basic_salary = basic_salary
        self._working_hours = working_hours
        salary = self.salary()
        super().__init__(emp_id, name, salary)

    def get_job_title(self):
        return self._job_title
```



```python
    def set_job_title(self, job_title):
        self._job_title = job_title

    def get_basic_salary(self):
        return self._basic_salary

    def set_basic_salary(self, salary):
        self._basic_salary = salary

    def get_working_hours(self):
        return self._working_hours

    def set_working_hours(self, hours):
        self._working_hours = hours

    def salary(self):
        if self._working_hours > 8:
            return self._basic_salary + (self._basic_salary * 0.25)
        else:
            return self._basic_salary

    def display(self):
        print(f"ID: {self._emp_id} - Name: {self._name} - Job Title: {self._job_title} - Working hours: {self._working_hours} - Salary: {self._salary:.1f} PKR")
```



```python
# ==== MAIN TEST ====
if __name__ == "__main__":
    emp = Employee(43221, "Ali", 70000.0)
    faculty = Faculty(71245, "Majed", "PhD", 32000, 15)
    staff = Staff(81234, "Nasser", "Registrar", 100000, 10)

    emp.display()
    faculty.display()
    staff.display()
```

```
ID: 43221 - Name: Ali - Salary: 70000.0 PKR
ID: 71245 - Name: Majed - Degree: PhD - Teaching hours: 15 - Salary: 47000.0 PKR
ID: 81234 - Name: Nasser - Job Title: Registrar - Working hours: 10 - Salary: 125000.0 PKR
```