DS2030 Data Structures and Algorithms for Data Science
Week-5 Practice Set

September 10th, 2025

1 Task Management System using Min-Heap

You are required to implement a simple **Task Management System** using a **Min-Heap**. Each task has a description and a priority (integer where lower number means higher priority).

Attributes

Each task is stored as a tuple:

- `priority`: Integer priority of the task.
- `description`: String describing the task.

Methods

- `add_task(priority, description)` : Insert a new task into the system.
- `execute_task()` : Remove and return the task with the highest priority.
- `peek_next_task()` : Return the task with the highest priority without removing it.
- `show_tasks()` : Display all tasks in heap order.
- `change_priority(description, new_priority)` : **Challenge** — Change the priority of a task based on description.
- `get_task_count()` : Return the total number of tasks in the system.

Starter Code

```
class TaskManager:
    def __init__(self):
        self.heap = []

    def add_task(self, priority, description):
        """
        Insert a new task maintaining min-heap property
        """
        pass

    def execute_task(self):
        """
        Remove and return the task with the highest priority
        """
        pass

    def peek_next_task(self):
        """
        Return the task with the highest priority without removing it
        """
        pass

    def show_tasks(self):
        """
```

```
        Print the tasks in the heap
        """
        pass

    def change_priority(self, description, new_priority):
        """
        Find a task by description and change its priority
        """
        pass

    def get_task_count(self):
        """
        Return the number of tasks in the system
        """
        return len(self.heap)
```

Testing

```
def test_task_manager():
    tm = TaskManager()
    tm.add_task(3, "Complete assignment")
    tm.add_task(1, "Fix server bug")
    tm.add_task(2, "Prepare presentation")

    print("All Tasks:")
    tm.show_tasks()

    print("\nNext Task:")
    print(tm.peek_next_task())

    print("\nExecuting Task:")
    print(tm.execute_task())
    tm.show_tasks()

    print("\nChanging priority of 'Complete assignment' to 0...")
    tm.change_priority("Complete assignment", 0)
    tm.show_tasks()

    print("\nTotal tasks in system:", tm.get_task_count())

test_task_manager()
```

Sample Output

```
All Tasks:
[(1, 'Fix server bug'), (3, 'Complete assignment'), (2, 'Prepare presentation')]

Next Task:
(1, 'Fix server bug')

Executing Task:
(1, 'Fix server bug')
[(2, 'Prepare presentation'), (3, 'Complete assignment')]

Changing priority of 'Complete assignment' to 0...
[(0, 'Complete assignment'), (2, 'Prepare presentation')]

Total tasks in system: 2
```

2 Event Logger using Max-Heap

You are required to implement an **Event Logger** system using a **Max-Heap**. Each event has a description and a severity level (integer where higher number means higher severity).

Attributes

Each event is stored as a tuple:

- `severity`: Integer severity level of the event.
- `description`: String describing the event.

Methods

- `log_event(severity, description)` : Insert a new event into the logger.
- `get_most_severe()` : Remove and return the event with the highest severity.
- `peek_most_severe()` : Return the most severe event without removing it.
- `show_events()` : Display all events in heap order.
- `update_severity(description, new_severity)` : **Challenge** — Update the severity of an event given its description.
- `get_event_count()` : Return the total number of events in the system.

Starter Code

```
class EventLogger:
    def __init__(self):
        self.heap = []

    def log_event(self, severity, description):
        """
        Insert a new event maintaining max-heap property
        """
        pass

    def get_most_severe(self):
        """
        Remove and return the event with the highest severity
        """
        pass

    def peek_most_severe(self):
        """
        Return the most severe event without removing it
        """
        pass

    def show_events(self):
        """
        Print the events in the heap
        """
        pass

    def update_severity(self, description, new_severity):
        """
        Find an event by description and change its severity
        """
        pass
```

```
    def get_event_count(self):
        """
        Return the number of events in the system
        """
        return len(self.heap)
```

Testing

```
def test_event_logger():
    logger = EventLogger()
    logger.log_event(5, "Low disk space warning")
    logger.log_event(10, "System crash detected")
    logger.log_event(7, "High memory usage")

    print("All Events:")
    logger.show_events()

    print("\nMost Severe Event:")
    print(logger.peek_most_severe())

    print("\nHandling Most Severe Event:")
    print(logger.get_most_severe())
    logger.show_events()

    print("\nUpdating severity of 'Low disk space warning' to 12...")
    logger.update_severity("Low disk space warning", 12)
    logger.show_events()

    print("\nTotal events in system:", logger.get_event_count())

test_event_logger()
```

Sample Output

```
All Events:
[(10, 'System crash detected'), (5, 'Low disk space warning'), (7, 'High memory usage')]

Most Severe Event:
(10, 'System crash detected')

Handling Most Severe Event:
(10, 'System crash detected')
[(7, 'High memory usage'), (5, 'Low disk space warning')]

Updating severity of 'Low disk space warning' to 12...
[(12, 'Low disk space warning'), (7, 'High memory usage')]

Total events in system: 2
```