from queue import PriorityQueue, Queue

def get\_average(iterable):

    return sum(iterable) / len(iterable)

# Base Scheduling Class

class BaseScheduling:

    def \_\_init\_\_(self, no\_of\_process: str, arrival\_times: list, burst\_times: list) -> None:

        self.no\_of\_process = int(no\_of\_process)

        self.arrival\_times = list(map(int, arrival\_times))

        self.burst\_times = list(map(int, burst\_times))

        self.cpu\_time = 0

        self.waiting\_queue = Queue()

        self.gantt\_list: dict = {}

        self.completion\_time: list[tuple] = list()

        self.turnaround\_time: list = list()

        self.waiting\_time: list = list()

    def check\_arrival(self, priorities) -> None:

        for process, (arrival\_t, priority) in enumerate(zip(self.arrival\_times, priorities)):

            if arrival\_t == self.cpu\_time:

                self.waiting\_queue.put\_nowait((priority, process))

    def set\_turnaround\_time(self) -> None:

        for (\_, completion), arrival in zip(self.completion\_time, self.arrival\_times):

            self.turnaround\_time.append(completion - arrival)

    def set\_waiting\_time(self) -> None:

        for tat, burst in zip(self.turnaround\_time, self.burst\_times):

            self.waiting\_time.append(tat - burst)

    def get\_completion\_time(self) -> list:

        return self.completion\_time

    def get\_turnaround\_time(self) -> list:

        return self.turnaround\_time

    def get\_waiting\_time(self) -> list:

        return self.waiting\_time

    def get\_average\_turnaround\_time(self) -> float:

        return get\_average(self.turnaround\_time)

    def get\_average\_waiting\_time(self) -> float:

        return get\_average(self.waiting\_time)

# Shortest Job First Scheduling

class SJF\_Scheduling(BaseScheduling):

    def \_\_init\_\_(self, no\_of\_process: str, arrival\_times: list, burst\_times: list) -> None:

        super().\_\_init\_\_(no\_of\_process, arrival\_times, burst\_times)

        self.waiting\_queue = PriorityQueue()

    def start(self) -> None:

        while self.no\_of\_process > 0:

            # initial waiting queue

            self.check\_arrival(self.burst\_times)

            # starting process

            while not self.waiting\_queue.empty():

                process\_is\_complete = False

                burst\_t, process = self.waiting\_queue.get()

                # print(f"process {process}, burst {burst\_t}")  # TEST LINE

                self.gantt\_list[self.cpu\_time] = f"P{process + 1}"

                # print(f"{self.gantt\_list=}")  # TEST LINE

                # if current process is not complete

                while not process\_is\_complete:

                    self.cpu\_time += 1

                    burst\_t -= 1

                    # if more than 1 process is available

                    if self.no\_of\_process > 1:

                        self.check\_arrival(self.burst\_times)

                    # if current process is complete

                    if burst\_t == 0:

                        process\_is\_complete = True

                        self.no\_of\_process -= 1

                        self.completion\_time.append((process, self.cpu\_time))

                        self.completion\_time = sorted(self.completion\_time)

                        continue

                    # checking the queue for new process

                    if not self.waiting\_queue.empty():

                        dequed = self.waiting\_queue.get()

                        # if new process has lower burst time

                        if burst\_t > dequed[0]:

                            self.waiting\_queue.put\_nowait((burst\_t, process))

                            burst\_t, process = dequed

                            self.gantt\_list[self.cpu\_time] = f"P{process + 1}"

                            continue

                        self.waiting\_queue.put\_nowait(dequed)

            # if there is no process in the queue, then cpu is idle.

            if self.waiting\_queue.empty():

                self.cpu\_time += 1

        # print(f"{self.gantt\_list=}")  # TEST LINE

        # print(f"{self.completion\_time=}")  # TEST LINE

        self.set\_turnaround\_time()

        self.set\_waiting\_time()

# Priority Scheduling

class Priority\_Scheduling(BaseScheduling):

    def \_\_init\_\_(self, no\_of\_process: str, arrival\_times: list, burst\_times: list, priorities: list) -> None:

        super().\_\_init\_\_(no\_of\_process, arrival\_times, burst\_times)

        self.priorities = list(map(int, priorities))

        self.waiting\_queue = PriorityQueue()

    def start(self) -> None:

        while self.no\_of\_process > 0:

            # initial waiting queue

            self.check\_arrival(self.priorities)

            # starting process

            while not self.waiting\_queue.empty():

                process\_is\_complete = False

                priority, process = self.waiting\_queue.get()

                # print(f"{priority=} {process=}")  # TEST LINE

                burst\_t = self.burst\_times[process]

                # print(f"{burst\_t=}")  # TEST LINE

                self.gantt\_list[self.cpu\_time] = f"P{process + 1}"

                # print(f"{self.gantt\_list=}")  # TEST LINE

                # if current process is not complete

                while not process\_is\_complete:

                    self.cpu\_time += 1

                    burst\_t -= 1

                    # if more than 1 process is available

                    if self.no\_of\_process > 1:

                        self.check\_arrival(self.priorities)

                    # if current process is complete

                    if burst\_t == 0:

                        process\_is\_complete = True

                        self.no\_of\_process -= 1

                        self.completion\_time.append((process, self.cpu\_time))

                        self.completion\_time = sorted(self.completion\_time)

                        continue

                    # checking the queue for new process

                    if not self.waiting\_queue.empty():

                        dequed = self.waiting\_queue.get()

                        # if new process has higher priority

                        if priority > dequed[0]:

                            self.waiting\_queue.put\_nowait((priority, process))

                            priority, process = dequed

                            self.gantt\_list[self.cpu\_time] = f"P{process + 1}"

                            continue

                        self.waiting\_queue.put\_nowait(dequed)

            # if there is no process in the queue, then cpu is idle.

            if self.waiting\_queue.empty():

                self.cpu\_time += 1

        # print(f"{self.gantt\_list=}")  # TEST LINE

        # print(f"{self.completion\_time=}")  # TEST LINE

        self.set\_turnaround\_time()

        self.set\_waiting\_time()

# Round Robin Scheduling

class RR\_Scheduling(BaseScheduling):

    def \_\_init\_\_(self, no\_of\_process: str, arrival\_times: list, burst\_times: list, time\_quantum: str) -> None:

        super().\_\_init\_\_(no\_of\_process, arrival\_times, burst\_times)

        self.TIME\_QUANTUM = int(time\_quantum)

    def start(self) -> None:

        while self.no\_of\_process > 0:

            # initial waiting queue

            self.check\_arrival(self.burst\_times)

            # starting process

            while not self.waiting\_queue.empty():

                process\_is\_complete = False

                burst\_t, process = self.waiting\_queue.get()

                # print(f"{process=} {burst\_t=}")  # TEST LINE

                self.gantt\_list[self.cpu\_time] = f"P{process + 1}"

                # print(f"{self.gantt\_list=}")  # TEST LINE

                # use time quantum for a process

                for \_ in range(self.TIME\_QUANTUM):

                    self.cpu\_time += 1

                    burst\_t -= 1

                    # if more than 1 process is available

                    if self.no\_of\_process > 1:

                        self.check\_arrival(self.burst\_times)

                    # if current process is complete

                    if burst\_t == 0:

                        process\_is\_complete = True

                        self.no\_of\_process -= 1

                        self.completion\_time.append((process + 1, self.cpu\_time))

                        self.completion\_time = sorted(self.completion\_time)

                        break

                # if current process is not complete

                if not process\_is\_complete:

                    self.waiting\_queue.put\_nowait((burst\_t, process))

            # if there is no process in the queue, then cpu is idle.

            if self.waiting\_queue.empty():

                self.cpu\_time += 1

        # print(f"{self.gantt\_list=}")  # TEST LINE

        # print(f"{self.completion\_time=}")  # TEST LINE

        self.set\_turnaround\_time()

        self.set\_waiting\_time()

def print\_output(schedule: BaseScheduling) -> None:

    completion\_time = [time for \_, time in schedule.get\_completion\_time()]

    turnaround\_time = schedule.get\_turnaround\_time()

    waiting\_time = schedule.get\_waiting\_time()

    average\_turnaround\_time = schedule.get\_average\_turnaround\_time()

    average\_waiting\_time = schedule.get\_average\_waiting\_time()

    print(f"Completion time         : {completion\_time}")

    print(f"Turnaround time         : {turnaround\_time}")

    print(f"Waiting time            : {waiting\_time}")

    print(f"Average turnaround time : {average\_turnaround\_time}")

    print(f"Average waiting time    : {average\_waiting\_time}")

def main():

    # ~~~~~ TEST VALUES : SJF\_Scheduling ~~~~~ #

    print("~~~~~ SJF\_Scheduling ~~~~~")

    no\_of\_process = input("Enter total number of processes: ")

    arrival\_time\_1 = input("Enter arrival times (space separated): ").split()

    burst\_time\_1 = input("Enter burst times (space separated): ").split()

    process\_sjf = SJF\_Scheduling(no\_of\_process, arrival\_time\_1, burst\_time\_1)

    process\_sjf.start()

    print\_output(process\_sjf)

    # ~~~~~ TEST VALUES : Priority\_Scheduling ~~~~~ #

    print("\n~~~~~ Priority\_Scheduling ~~~~~")

    no\_of\_process = input("Enter total number of processes: ")

    arrival\_time\_2 = input("Enter arrival times (space separated): ").split()

    burst\_time\_2 = input("Enter burst times (space separated): ").split()

    priorities = input("Enter priority values (space separated): ").split()

    process\_priority = Priority\_Scheduling(no\_of\_process, arrival\_time\_2, burst\_time\_2, priorities)

    process\_priority.start()

    print\_output(process\_priority)

    # ~~~~~ TEST VALUES : Round\_Robin ~~~~~ #

    print("\n~~~~~ Round\_Robin ~~~~~")

    no\_of\_process = input("Enter total number of processes: ")

    arrival\_time\_3 = input("Enter arrival times (space separated): ").split()

    burst\_time\_3 = input("Enter burst times (space separated): ").split()

    time\_quantum = input("Enter time quantum value: ")

    process\_rr = RR\_Scheduling(

        no\_of\_process, arrival\_time\_3, burst\_time\_3, time\_quantum)

    process\_rr.start()

    print\_output(process\_rr)

if \_\_name\_\_ == "\_\_main\_\_":

    main()