Python Project # 2 - Greedy Algorithm for a Scheduling Problem

In this project we want to use the Greedy algorithm to solve a scheduling problem. Assume that you have a summer internship where you work 8 hours per day. Your supervisor has given you a list of 7 tasks to complete and has prioritized these by ranking them by 1 through 10, where 10 is the most important task. You look at this list of tasks and estimate the time (in minutes) that you need to complete each one. Unfortunately you can't complete all 7 tasks in 8 hours so the goal is to determine which tasks to do.

We first assume that we use a Greedy Algorithm where we choose the task which has the highest priority to complete first. When that is done we choose the task to do which has the next highest priority that can be completed in the remaining time, etc. Your goal is to determine which tasks you can complete in the first 8 hour day.

What is our strategy for writing this code to determine which tasks to do on the first day?

Basically, this is just like the Greedy algorithm for the Knapsack problem where instead of picking the item with the highest value not already in the Knapsack (and which fits), we pick the task which has the highest priority and which is not already completed and which can be done in the remaining time.

We will construct our program in a step-by-step mammner.

Step 1. Input the problem specific data.

- I have defined the two task lists; one is for the priority of each task and the other is for the time in minutes that each task takes.
- Next define the number of tasks. Do not use a number (like 7) but rather use the **len()** command.
- Next define the amount of time in minutes that is available to complete the tasks on the first day.
- Output the information in a table-like format; later we will learn how to do this to make it look nice but for now just print out the task numbers from 1 to 7, their priority and the time to complete the task. I have added the commands to print the title and one line of the table. To print all the tasks you need to put this statement inside a **for** loop and modify it appropriately.

```
priority = [2, 6, 1, 7, 10, 8, 5]
time = [30, 200, 20, 70, 120, 60, 150]
#
# define number of tasks:     don't hardwire this but rather use the len() command

i=1
n_tasks = len(priority)
max_time =  max(time)
print(f"TASK      PRIORITY      TIME TO COMPLETE")
# you need to add an appropriate for loop here and modify the next print statement so that you can print out a
for i in range(0, n_tasks):
    n = i+1
    print(f"  {n}         {priority[i]}             {time[i]}")
```

```
TASK      PRIORITY      TIME TO COMPLETE
  1         2             30
  2         6             200
  3         1             20
  4         7             70
  5         10             120
  6         8             60
  7         5             150
```

Step 2. Write code to test finding the maximum value in a list and the index where it occurs.

To do this you must use the **max()** function on a given list and then use the list method **.index ()** to find the index of the location in the list where this occurs. Using a formatted print statement output your maximum value and the index where it occurs in the list.

I have provided a list to check your answer. Clearly the maximum value should be 11 which is the 5th item in the list and has an index of 4 (because the first item in the list has index = 0)

```
test_list = [4, 5, 3, 7, 11, 2]


max_value = max(test_list)  # use function max() to find maximum value in list
max_index = test_list.index(max_value)  # use method list.index () to find the index where this max occurs
print(f"The Index of the Maximum Value, {max_value}, is {max_index}")   # print out results in formatted state
```

```
The Index of the Maximum Value, 11, is 4
```

Step 3. Add statements to Step 2 to find the corresponding time that it takes for the task you found there. Print it out using a formatted statement.

I have added a sample time list for the sample list above to use to verify that your code is working.

This is easy to do because you already have the index where the maximum occurs so just print out the item in the other list which occurs at this index.

As before the maximum value should be 11 which is the 5th item in the list and has an index of 4 and this tasks takes 90 minutes.

```
test_list = [4,  5, 3, 7, 11, 2]
time_list = [30, 45, 20, 120, 90, 70]
max_value =  max(test_list)
max_index =  test_list.index(max_value)
task_time = time_list[max_index]   # input the time it takes to complete this task
print(f"The time it takes to complete the task of finding the Max Value, {max_value}, and the Index of the Max
The time it takes to complete the task of finding the Max Value, 11, and the Index of the Maximum, 4, is 90 minutes
```

Step 4. Conditional statement to test if the task chosen can be completed in the remaining time.

- I have entered random values (not for your specific problem) for the maximum time available and the amount of time already used.
- To your statements in Step 3 add a conditional statement to check to see if the task can be done in the remaining time. If it can, then update the time_used.
- Print out the task added (not its index but the number of the task so for this test problem it is 5) along with the current time used which is updated to 270;

```
max_time = 6 * 60  # this is not the maximum time available for your example but rather for testing
time_used = 350
#
# Following 6 statements are from Step 3
test_list = [4, 5, 3, 7, 11, 2]
time_list = [30, 45, 20, 120, 90, 70]
max_value =  max(test_list)
max_index =  test_list.index(max_value)
task_time = time_list[max_index]   # input the time it takes to complete this task
print(f"The time it takes to complete the task of finding the Max Value, {max_value}, and the Index of the Max

# Add conditional to see if this task can be completed in the allotted time
if task_time <= (max_time - time_used):
    time_used += task_time
    max_index = max_index + 1 # if conditional is true, i.e., task can be completed update time_used
    print(f"Task {max_index} added. Current time used: {time_used}.")
else:
    max_index = max_index + 1
    print(f"Task {max_index} cannot be completed in the remaining time.")# add print statements
```

```
The time it takes to complete the task of finding the Max Value, 11, and the Index of the Maximum, 4, is 90 minutes
Task 5 cannot be completed in the remaining time.
```

Step 5. Write the code for a Greedy algorithm where the "best" criteria is the item with the highest priority. To do this, input your problem specific data from Step 1 and add the appropriate lines of code from Step 4. Add comments to your code with a "#"

- First input problem specific data from Step 1 (just cut and paste)
- As discussed in the Knapsack Problem this requires a loop over all the tasks.
- Be sure to thing about what quantity needs to be initialized (i.e., set to 0) BEFORE the beginning of the loop
- In the loop you need to find the index where the maximum occurs and then find the corresponding time which you have already done in Step 3/4. Then you need to add the conditional statement from Step 4 to see if there is enough time remaining to complete this task. If so, print out results as in Step 4.
- In order to make sure that each time through the loop the code does not find the same task (i.e., the same maximum priority) we see the priority to 0 after we have checked this item to see if it can be done or not.

- I have added comments to remind you of what statements to add where; that is, for how the code is structured.

```
# Input problem specific data and print it out from Step 1
i= 0
priority = [2, 6, 1, 7, 10, 8, 5]
time = [30, 200, 20, 70, 120, 60, 150]
n_tasks = len(priority)
max_time =  max(time)
print(f"TASK      PRIORITY      TIME TO COMPLETE")

# Initialize any variables before loop
priorityc= priority.copy() #creates copy of original list
time_used = 0 #initializes time used as 0

# add for loop to go through all tasks

for i in range(n_tasks):
    # find highest priority left and task number it corresponds to from Step 3/4
    task_num = priorityc.index(max(priorityc))
    # zero out this entry in the priority list so you won't find it again
    priorityc[task_num] = 0

    # add conditional to check to see if task can be completed as in Step 4
    if (time[task_num] + time_used) <= max_time:
        # Update the time used
        time_used += time[task_num]
        # Print out the task and time_used
        print(f"The time it took to complete the task, {time[task_num]}, and total time used, {time_used}")
```

```
TASK      PRIORITY      TIME TO COMPLETE
The time it took to complete the task, 120, and total time used, 120
The time it took to complete the task, 60, and total time used, 180
```

Step 6 Making your code more efficient

- If the task is added and the time used = maximum time then there is no reason to go through the remaining items. So we would add another conditional (inside the first conditional) and if it is true, then we break out of the loop using the command **break**

```
# Input problem specific data and print it out from Step 1
i= 0
priority = [2, 6, 1, 7, 10, 8, 5]
time = [30, 200, 20, 70, 120, 60, 150]
n_tasks = len(priority)
max_time =  max(time)
print(f"TASK      PRIORITY      TIME TO COMPLETE")

# Initialize any variables before loop
priorityc= priority.copy()
time_used = 0

# add for loop to go through all tasks
for i in range(n_tasks):
    # find highest priority left and task number it corresponds to from Step 3/4
    task_num = priorityc.index(max(priorityc))
    # zero out this entry in the priority list so you won't find it again
    priorityc[task_num] = 0
    # add conditional to check to see if task can be completed as in Step 4
    if (time[task_num] + time_used) <= max_time:
        # Update the time used
        time_used += time[task_num]
        # Print out the task and time_used
        print(f"The time it took to complete the task, {time[task_num]}, and total time used, {time_used}")
    # if this conditional is satisfied then add another conditional to check if time_used = max_time; if so br
        if time_used == max_time:
            break
    # Print out the total time used for all tasks in day 1
print(f"Total time used for all tasks in day 1 is {time_used}")
```

```
TASK      PRIORITY      TIME TO COMPLETE
The time it took to complete the task, 120, and total time used, 120
The time it took to complete the task, 60, and total time used, 180
The time it took to complete the task, 20, and total time used, 200
Total time used for all tasks in day 1 is 200
```

Step 7. (Extra Credit) Now modify your code to use a Greedy algorithm where you choose the "best" task to do based on the priority divided by the time; choose the next task to be completed to be the item which has largest priority to time ratio.

```
# Input problem specific data
old_priority = [2, 6, 1, 7, 10, 8, 5]
time = [30, 200, 20, 70, 120, 60, 150]
# n_tasks =  len(old_priority)
#
# Create new priority list containing ratios
#
priority = []  # initialize the list which contains the new priorities which are original priority divided by

# time it takes to complete this task
max_time = 400
time_used = 0
# add for loop to create this list
for i in range(n_tasks):
    ratio = old_priority[i] / time[i]
    priority.append(ratio) # use method .append() to  create elements of the new priority list

while time_used < max_time and priority:
    task_num = priority.index(max(priority))
    if(time[task_num] + time_used) <= max_time:
        time_used += time[task_num]
    priority.pop(task_num)
    time.pop(task_num)

print(f"total time used is {time_used} minutes")

# Remainder of code should be the same; this is because we called the new list

# which contained the ratios the same as before.
```

```
total time used is 300 minutes
```

Why do you get different answers for the two algorithms?

The first algorith prioritized priority value, while algorithm 2 used priority value/time ratio.

Which strategy for the two Greedy algorithms allows you to complete the largest number of tasks in 8 hours?

By prioritizing the value/time ratio we are able to get more tasks done since we prioritize the value of the tasks compared to the time. Rather than just the value where we could be doing more valuable tasks, but sacrifice being less efficient with time.