For each question, paste your solution in the text field under the description.

For programming questions:

- All answers should be provided as working Python code
- You should perform your own tests to verify that the code is correct. However test cases need not be provided with your answer. If you do provide test cases, you should identify those clearly (for example with comments).
- for maximum credits, the provided answer should be in the best possible complexity class. Less efficient answers will be given partial credits.
- It is not necessary to comment on your code, it is more important that the code is clean and easy to read.

The maximum possible total number of points is 60. 24 points correspond to grade 3, 36 points correspond to grade 4, and 48 points correspond to grade 5.

Allowed aids:

- You can use the Self-Practice website in sandbox mode to test your code. From the Safe Exam Browser, you should click the "Python Sandbox" button on the bottom left of your screen to open it. So will open the self-practice website that we used in the course in another window. On the self-practice website, it is intended for you to use the "Exam Sandbox" exercise block, which you can find at the bottom of the exercise list. Note that your code won't be saved if you switch exercise in the self-practice website, or if you close or reload its window. Every time you click on "Python Sandbox", a new window will appear. It is advised to have one Self-Practice window open for each question and switch between windows using Alt+Tab or Option+Tab.
- You can refer to your own notes and books. **These notes or books must be on paper.**

Disallowed aids: anything else. In particular, it is not allowed to use electronic devices which can access:

- AI tools (Copilot, ChatGPT, Llama, ...)
- Communication tools (chat, mail, social media, ...)
- Electronic documentation (Google, stack overflow, python guides or manuals, ...)
- Your own notes in electronic format

Question 1

Assume that a timestamp is represented as a tuple `(hour,minute)`. Assume further that each component of the tuple is an integer, and that the `minute` component is in the interval 0 to 59. Write a function `advance` which takes an arbitrary timestamp as parameter and returns a new timestamp which is advanced by 10 minutes towards the future.

Answer

```
# assumption: a timestamp can refer to different days
# and thus hours may be >24.

def advance(t):
    (h,m) = t
    m = m + 10
    if m >= 60:
        h=h+1
        m=m-60

    return (h,m)
```

Question 2

Your user has many agenda files, and each of them have the same format. Each line of the file has the form: `Start Stop Task`, where `Start` and `Stop` are integers and `Task` is a string without spaces. For example, "agenda.txt" might look like this:

```
9 13 exam
12 14 lunch
15 17 dance-class
18 22 meet-friends
```

In this example, the `exam` is considered to happen from 9 to 13 included.

Write two python functions:

1. `load_agenda(file_name)`, that reads the agenda and returns a list of tuples with `Start`, `Stop` and `Task`, using appropriate data types.
2. `tabulate_agenda(agenda)` which takes the result of the previous function and returns a list of lists `result`, such that `result[t]` is the list of tasks happening at time `t`. Note that tasks can overlap. For instance, assuming the above agenda, `result[13] = ['exam','lunch']`.

You can test both functions like so:

```
print(tabulate_agenda(load_agenda("agenda.txt")))
```

Answer

```
def load_agenda(file_name):
    result = []
    with open(file_name) as f:
        for l in f:
            start,stop,task = l.split()
            result.append((int(start),int(stop),task))
    return result

def tabulate_agenda(a):
    result = []
    for start,stop,task in a:
        while len(result) <= stop:
            result.append([])
        for i in range(start,stop+1):
            result[i].append(task)
    return result

print(tabulate_agenda(load_agenda("agenda.txt")))
```

Question 3

Write a class `Tracker` which keeps track of tasks done during a day. It must have the following methods:

- `__init__(self,now)` It initializes the tracker and sets the current time to `now` . You can assume that time can be represented by a single integer, as in Question 2.
- `tick(self,n)` It advances the time by `n` units of time.
- `start(self,task)` It records that `task` is starting. A warning message is printed if the task is already started.
- `stop(self,task)` . It records that `task` stops. A warning message is printed if the task is not started.
- `stop_all(self)` It records that all ongoing tasks stop.
- `print_agenda(self)` It prints the agenda for the day in the format of `agenda.txt` (see Question 2). If there are tasks which are not stopped, they should not be printed in this format. A warning message should be printed in this case.
- Attention: a task with the same name can occur twice.

Here is a run which reproduces the contents of `agenda.txt` from Question 2:

```
from q3 import Tracker
a = Tracker(9)
a.start("exam")
a.tick(3)
a.start("lunch")
a.tick(1)
a.stop("exam")
a.tick(1)
a.stop("lunch")
a.tick(1)
a.start("dance-class")
a.tick(1)
a.tick(1)
a.stop("dance-class")
a.tick(1)
a.start("meet-friends")
a.tick(4)
a.stop_all()
a.print_tasks()
```

Answer

```
class Tracker:
    def __init__(self,now):
        self.open = {}
        self.done = []
        self.time = now
    def tick(self,n):
        self.time = self.time + n
    def start(self,task):
        if task in self.open:
            print("Warning: task already started")
        self.open[task] = self.time
    def stop(self,task):
        if task not in self.open:
            print("Warning: task not started")
        else:
            self.done.append((self.open[task],self.time,task))
            self.open.pop(task)
    def stop_all(self):
        for t,start in self.open.items():
            self.done.append((start,self.time,t))
        self.open = {}
    def print_tasks(self):
        if len(self.open) > 0:
            print("Warning: there are currently open tasks")
        for (start,stop,task) in self.done:
            print(start,stop,task)

# from q3 import Tracker
a = Tracker(9)
a.start("exam")
a.tick(3)
a.start("lunch")
a.tick(1)
a.stop("exam")
a.tick(1)
a.stop("lunch")
a.tick(1)
a.start("dance-class")
a.tick(1)
a.tick(1)
a.stop("dance-class")
a.tick(1)
```

```
a.start("meet-friends")
a.tick(4)
a.start("exam")
a.tick(1)
a.stop_all()
a.print_tasks()
```