Final Project: Functional Programming
2023 - 2024

1. Description

The goal of the final project is to implement a program to manage the agenda of several people. Each individual agenda contains the name of the owner and a list of events. Each `Event` is a list of descriptors, which can appear in any order. `Descriptors` of an event cannot be repeated but some of them can be missing when the event is created. The possible descriptors are (the required ones are marked with *):

- `Name*`: a `String` describing the event. Required.
- `Type*`: with four possible values: `personal`, `health`, `work` and `management`.
- `Date`: the date of the event: `day` (`Int`), `month` (`String`), `year` (`Int`), `workday` (`Bool`)
- `Time*`: the time of the event. It will be a triplet with (`start`, `end`, `duration`) where `start` and `end` will be `hour` (`Int`) and `minutes` (`Int`) and `duration` will be `minutes` (`Int`). An event will always have `duration`, either explicitly or calculated by the difference between the final and initial hours.
- `Participants*`: a list with the persons involved in the event. The owner of the agenda will be always the first element of this list.
- `Repetitions*`: the number of times the event is repeated, it can be `punctual` if it is not repeated, `daily` if it is repeated every day at the same hour, `weekday` if it is repeated every working day but not on weekends, or `weekly`, if it is repeated every week at the same day and hour. When the event is not `punctual` the `Date` indicates the first occurrence of the event and an integer parameter indicates the number of repetitions.

2. Work to perform

1. Types to represent all the former concepts. An exhaustive type definition is required, use the most appropriate way (`type`, `data`, `newtype`) for each of them.
2. A function to show the events in a pretty way, following the order of descriptors as presented above.
3. An `isWorkingDate Date [Date]` function that receives a `Date` and a list of holidays and returns if the day is weekday or weekend/holiday by considering leap years and the fact that 1st January 2022 was Saturday.
4. A `searchEvents [Descriptor] Agenda` function that returns all the events of the `Agenda` with such list of `Descriptors`. It should be able for example to return events that start at a given time, with a given frequency, involving some persons, etc.
5. A `deleteEvents [Descriptor] Agenda` function that returns an `Agenda` where the events having those descriptors are removed.
6. A `completeTime Event` function that if the `Event` has not `duration` inserts it by considering the `start` and `end` hours. If it has `start` and

`duration` it will insert the `end` and vice versa. If it has just `duration` it will do nothing, as the `Event` has not been scheduled yet. Assume no event will spawn over different days.

7. A `listAgenda Agenda` function that returns all the `Events` of an `Agenda` sorted from earliest `Date` and `Time` to latest. Events which are not yet scheduled will be shown at the end and sorted by their `name` in alphabetical order.
8. An `insert Event Agenda` function that for events fully specified in terms of `Date` and `Time` returns the `Agenda` with the `Event` included if it does not overlap with any other. If it overlaps the original `Agenda` will be returned. If the `Event` is not fully specified, it will raise an error.
9. A `scheduleEvent Event Agenda StartDate EndDate` that inserts an `Event` with no `Date` in the `Agenda` and schedules it as earliest as possible between `StartDate` and `EndDate`. If the `Event` cannot be scheduled the original `Agenda` will be returned. Notice that the event can be partially or fully defined in terms of `Time`. Do not consider repetitions.
10. An `allPossibleSchedules Event Agenda StartDate EndDate` that returns a list of `Agendas` with all the possible slots where the `Event` can be scheduled. Do not consider repetitions either.
11. A `scheduleRepeatedEvent Event Agenda StartDate EndDate` that schedules an event as earliest as possible taking into account its repetitions and ensuring that at least the first repetition is scheduled in the interval (the second, third, etc. ones can be inside or outside the interval). If the `Event` cannot be scheduled the original `Agenda` will be returned.
12. A `scheduleSharedEvent Event [Agenda] StartDate EndDate` that schedules the event in all the agendas so it can take place simultaneously. It will return a list with the new agendas or the original list if it cannot be scheduled.
13. Functions that allow the user to create and see an `Agenda`, and to search, create, insert, delete and schedule `Events` in one or several `Agendas`.

The former functions must work for any input of the right type, including empty lists where applicable. They must also consider cases as `StartDate` not being before `EndDate`, etc. Any auxiliary function considered necessary can be created.

3. Delivery Rules

The following rules are **compulsory.** Not following them can result in the work not being considered for evaluation. The final project must be performed in groups of two students. Each group must upload to Aula Global an ipynb file containing the source code before the deadline. The name of the file must be “name-initials1-name-initials2.zip” (for example if the students are Lucía Pérez Gómez and Juan García Jiménez, the file will be named lpg-jgj.ipynb). Both students must upload the file.

In addition to the code, the file must include text cells that will include at least:

- A brief summary of the document.
- Description of the types created.
- General description of the functions.
- Description of the work performed, functionalities included, parts not performed.

- Conclusions:
    - Final summary of the work performed.
    - Main problems found when implementing the program.
    - Personal comments and feedback, including criticism of the final project.

4. Evaluation

The final project will be evaluated from 0 to 10 points.

The following items will be considered to mark the final project:

- Appropriate types design.
- Reimplementation of function in the derived types.
- Correct execution of the program and compliance with the guidelines stated by this document.
- Quality of the implemented code.
- Use of the syllabus of the course (ask before using any function not presented during the course).
- Inclusion of code comments.
- General description of the implemented code in the documentation. (Section 3).
- Adhesion to the delivery rules (Section 3).

The following penalties will be also applied:

- 1 point if not having enough comments.
- 1 point if functions' types are not specified or are too generic.

Copies will be seriously penalized: both copying and copied groups will be excluded from continuous evaluation in addition to any other administrative measure that the University could take. Two final projects will be considered a copy even if the similarity between them is limited to a single function. Automatic tools will be used to check for plagiarisms.

The practical exercise will be corrected in an oral presentation. Both students will have to defend their work in front of the professor.