**Technical Docs – Assessment Planner (With Outlook agendas)**

Pepijn Alofs

Table of Contents

[Short Description 1](#_Toc183067402)

[Program Flow by script 2](#_Toc183067403)

[GUI interface (assessmentScheduling.py) 2](#_Toc183067404)

[Outlook availability (availability.py) 2](#_Toc183067405)

[Basic Operations and Sheet read-in (functions.py) 5](#_Toc183067406)

[Optimization/scheduling (functionScrtip.py) 6](#_Toc183067407)

[File management (input/output) 7](#_Toc183067408)

[Assessor information (.xlsx) 7](#_Toc183067409)

## Short Description

The assessment scheduling tool is designed to create optimized schedules for assessment sessions, balancing assessor availability, predefined goals, and program-specific requirements. It begins with user interaction through a graphical interface (GUI) in **assessmentScheduling.py**, allowing the selection of an input Excel file containing essential data such as assessor capacities, activities, and scheduling constraints. The GUI initiates the process through **start\_scheduling()**, organizes the input data using **load\_data()**, and integrates calendar-based availability through **retrieve\_calenders()** if enabled. A scheduling algorithm implemented in **makeSchedule()** then generates a proposed schedule that adheres to both operational constraints and program goals.

The input data includes detailed information about internal and external assessors, such as their monthly capacities, allowable activities, specific program affiliations, and availability. For assessors using Outlook calendars, availability is extracted from ICS links via **clean\_timezone\_ics()** in **availability.py**. These links, published by assessors through Outlook, provide a dynamic and accurate representation of free and busy time slots. The tool processes these calendars in **get\_calendar()** and **get\_free\_time()** to generate a comprehensive schedule of free time, accounting for recurring events and overlapping appointments using **merge\_overlapping\_events()**. This ensures that only genuinely available time slots are considered for scheduling.

The scheduling process integrates several layers of constraints managed by functions in **functionScript.py**. Assessors are limited to a maximum of one activity per day and are restricted from consecutive-day scheduling. Activities are grouped logically, such that sessions within an "Assessment Afternoon" are scheduled together. Weekly and monthly capacities are enforced, and specific unavailability dates, including office-wide unavailability such as holidays or events, are respected. The tool minimizes the use of external assessors combined with trying to match the desired number of monthly sessions per program.

The optimization algorithm in **makeSchedule()** processes the input data and constraints to produce a schedule that aligns with program-specific goals. It prioritizes minimizing external assessor usage while aiming to meet or approach session targets for each program, a balance achieved through the objective function **model.Minimize()**. The output includes three Excel sheets: the proposed schedule, a capacity usage report comparing available capacity with actual allocation, and a goal comparison sheet showing how well scheduling objectives were achieved. These sheets are created and updated using **xlsxwriter**, featuring active formulas for dynamic updates if the user modifies the final schedule.

## Program Flow

### GUI interface (assessmentScheduling.py)

* Start\_scheduling\_threaded()

Side function that allows the GUI to show status updates without freezing

* Open\_file()

Response to clicking the Browse button in GUI: Select single excel file

* log\_message(message)

Display string *message* in GUI textbox

* **start\_scheduling()**

Response to clicking the Start Scheduling button = main function

What happens:

1. Organize the input data (e.g. dates) in the right format
2. Call retrieve\_calenders (if indicated to take into account Outlook availability)
3. Optimize the schedule through makeSchedule

solutionDf, capacityUsage, goal\_comparison\_df, scheduleText =

makeSchedule(startDate, endDate, selectedFile, output\_text, check\_calender=check\_calender\_var.get(), want\_ics=False)

1. When finished, opens a file browsing window with a suggested filename for the schedule, and lets user put the file where wanted.
2. When file location is selected, it writes 3 sheets:
   1. solutionDf (schedule) as it comes from makeSchedule
   2. capacityUsage (which is the comparison of available capacity staff/externals and the final usage and to which activity it has been allocated, per month)

First it writes to the file, then it’s directly loaded back using the library *xlsxwriter* (only way to use this library is with existing files), to write active formulas in the cells (i.e. updates if the schedule is later adapted by the user)

* 1. goal\_comparison\_df (which is the overview of initial goals set per program, with reports on how many were actually scheduled, per month and per program). This is also loaded back to provide active formulas based on the schedule sheet and any changes.

1. The file is opened automatically and the application is closed.

### Outlook availability (availability.py)

* log\_message(message, output\_text)

Add string *message* to existing text in GUI textbox and update status to user

* clean\_timezone\_ics(ics\_text)

1 Outlook calender (iCalender: .ICS) file used strange/incorrect time zones, this function allows you to reset them to the general/common TZID:Romance Standard Time

Availability flow in order of process:

* retrieve\_calenders(selectedFile, output\_text, start\_date, end\_date, links='resources/calenders.json')

Main function that manages the flow. **Loads calenders.json**, which is a json that should contain the ICS links in the following format:

*{*

*"Person1": "https://outlook.office365.com/owa/calendar/anonymous1@ormittalent.be/uniqueidentifier1/calendar.ics",*

*"Person2": "https://outlook.office365.com/owa/calendar/anonymous2@ormittalent.be/uniqueidentifier2/calendar.ics"*

*}*

The link can be provided by a staff member via:

1. Navigate to Outlook **web version**, the calender or email pages are both okay
2. Navigate to **Settings** (gear icon top right)
3. In the Left pane there’s some options, select **Calendar**
4. In the Calendar pane, select ‘**Shared calendars’**
5. In the “Shared calendars” page there’s a section “**Publish** a calender” (! Not ‘share' calender)
6. Select your ORMIT agenda in the dropdown
7. From the sharing options, choose “Can view when I’m busy”. This again means no details will be shared about the events in your calender, but I will be able to see your event slots and your free slots.
8. Copy the **ICS** link that appears and send this to me.

In this way, the published links will update (with a +-30 min. delay) automatically at any call. Besides the calendar links, the **selected excel file is loaded** and used as a basis to which the available days will be added as rows. This excel file should have n+2 sheets: n internal assessors, 1 sheet for ‘External’ and 1 called ‘Extra’ where the goals per month are noted down (more under ‘Input Files – Assessment info Excel file’)

The function then for all internal assessors that are also in the calenders.json:

* get\_calender(calenders, key, str(start\_date), str(end\_date))

Takes the ICS links, internal staff members name (key) and start and end date.

It uses the library *requests* to collect a live version of the ICS file:

*ical\_string = response.text # Get the content of the ICS file as a string*

*a\_calendar = icalendar.Calendar.from\_ical(ical\_string) #Extract workable file*

**! Important**

*events = recurring\_ical\_events.of(a\_calendar).between(start\_date, end\_date)*

Recurrent events in ICS files are not written out for every day they re-occur, only first time with a note that they are recurrent. This function makes sure they are handled properly and written out for every day they re-occur.

The next steps are well-commented, they reshape the events variable (all events, including both single day and multiple day spanning ones) into a DataFrame that is readable and equal to their Outlook calendar (all events that should be found online as well): *expanded\_event\_df*.

**! Important**

The following step determines **which will be marked as unavailable**:

*df = df[df['Status'] != 'FREE']*

There is 4 labels possible in Outlook for scheduled events: Free, Busy, Tentative & Out of Office. **Currently, all events except those marked ‘Free’ are assumed to be unavailable**. An important last step is

*df\_merged = merge\_overlapping\_events(df)*

To make dealing with overlapping and double/triple-scheduled moments during days, all events on a day are merged together into 1 big event (e.g. 10:30-13:15, 12:00-14:00 and 13:30-15:00 in 1 person’s agenda on a day is turned into 1 event from 10:30-15:00). The resulting schedule (with busy moments, but no overlapping events anymore) is then transformed into a schedule of available moments, by:

* get\_free\_time(schedule\_df, staff\_member)

Is quite extensively commented, but for every day, looks for the 3 types of free time slots:

1. Before the first event
2. Between events during the day
3. After the last event

It does not assume any working hours (e.g. between 9-19:00 only), because the function that fits the requested assessment events works with clearly defined time slots.

Then, the **schedule with free moments** (avail\_gen), if not empty, is further processed:

* *find\_assessment\_slot(free\_time\_df, starttime, endtime, dayofweek)*

This works with:

* free time schedule (df with columns ['Event Name', 'Start Date', 'Start Time', 'End Time', 'Staff'])
* Start and end time of the event you want to schedule (strings HH:MM:SS, e.g. "12:00:00")
* Day of the week you want to schedule on (list of integer(s):

**!** 0=Monday, 1=Tuesday … 4=Friday, e.g. for Thursday [3], for multiple [0,3,4])

**! Important**

The **unavailable days work with almost the same coding** (same number values but not in a list): Monday is 0 and not 1!

This function outputs a **list of dates** that represent the date that the event can occur on, i.e. the free time schedule has space on that weekday, during that time period. ATM only assuming single day events.

This list of dates is concatenated to any other similar event types (e.g. curious cases that are planned on 3 days are retrieved by individual function calls, but then concatenated into 1 list). The 2 assessment types (assessmentAvailability for Assessment afternoons with Roleplay, Cases and PAPI interviews; caseAvailability for Curious Case sessions) are inserted into the DataFrame (of the original selected/loaded excel with all info) under the ‘Unavailability’ row. This is outputted to a default file: assessors2025\_available.xlsx that is used for the remainder of the run.

**! If someone’s** **calendar is unavailable/not provided** **full availability is assumed**.

### Basic Operations and Sheet read-in (functions.py)

* workingDays

Returns a dictionary that contains the list of %Y-%m-%d formatted working dates (i.e. week and not weekend day).

* Get\_mont\_number/get\_month\_name()

Allows switching between numerical months and their name

* **Load\_data(assessorExcel)**

Important function that reads in the information in the file selected. It goes over all the sheets (i.e. staff members, in addition to the ‘External’ and ‘Extra’ sheet). For staff members (internal and external) it stores their:

* Capacity for the given months (int)
* Activities they are allowed to assess (list of str)
* Whether they are allowed to do data cases (True/False)
* Whether they are part of the HR team (True/False)
* Which programs they are allowed to do assessment afternoons for (list of str)
* The list of dates (str) where they are available for the assessment afternoon slot
* The list of dates (str) where they are available for the curious cases

For the ‘Extra’ sheet it stores:

* The desired number of sessions (‘Candidate Goal – ‘)

**! Note: Candidate Goal is a slightly misleading term, but it refers to:**

* + **The number of sessions (1 to 1) for curious cases (i.e. 14 = 14 cases will have to be planned)**
  + **The number of sessions divided by 3 for assessment afternoons (i.e. 9 = 3 sessions will be planned)**
* Public Holidays (list of str of dates) which will be assumed unavailable for everyone
* Same for Office Events (everyone assumed unavailable on those days)

It returns:

* assessors: info on assessors as a dict by assessor
* program\_capacities: per month the goal per program
* office\_unavailabilities: list of str dates

### Optimization/scheduling (functionScrtip.py)

1. No Assessments on Consecutive Days:

Ensures assessors do not have activities scheduled on two consecutive weekdays (except for "External" assessors).

1. Uniqueness of Activity Types per Assessment:

Limits to one activity type per assessment (e.g., one PAPI, one Roleplay) except for "Curious" cases, which allow twice the same activity type.

1. One Activity per Day per Assessor

Each assessor participates in at most one activity per day.

1. Grouped Assessment Activities

Ensures all activities of an assessment type (except "Curious") are scheduled together in the same afternoon or none at all.

1. Assessor Monthly Capacity

Limits the total activities scheduled for an assessor in a month based on their predefined capacity.

1. Weekly Non-Working Days

Ensures no activities are scheduled for assessors on their predefined non-working weekdays.

1. Specific Unavailability:

Prevents scheduling for assessors on specific dates marked as unavailable (e.g., due to training).

1. Office Unavailability:

Prohibits scheduling any activities on days when the office is marked unavailable.

1. Calendar-Based Availability:

Ensures activities are scheduled only on dates when assessors are available according to their calendar (applicable if check\_calender is true).

1. Curious Case Scheduling Days:

Restricts "Curious" case activities to specific weekdays (Monday, Friday, mutually exclusive on Tuesday and Thursday) and disallows them on Wednesdays.

1. Weekly Activity Limits:

Maximum 2 activities per week for each assessor: 2 cases | 1 case & 1 assessment day | NOT 2 assessment days (too intense)

1. HR Team Member Presence:

Ensures at least one HR team member is present in each assessment session. (Currently inactive in the script due to capacity constraints.)

1. External Assessor Usage Minimization:

Includes a goal in the objective function to minimize the use of external assessors.

1. Program Goal Constraints:

Matches scheduled candidates to monthly program goals, allowing for under-scheduling but not over-scheduling.

**Goal:**

model.Minimize(external\_assessor\_count + constant\_goal\_weight \* sum(goal\_deviations))

***Minimize the external assessor******use*** *(in credits)* ***combined with as close to the number of session goals per program as possible****. Currently the weight of 1 works well. Focus can be shifted towards using more externals to meet the number of assessments more closely, or vice versa.*

Remaining section of the script organizes the proposed schedule, and delivers some additional side-statistics on the proposed schedule. Currently there is a maximum time for the optimization to run:

*solver.parameters.max\_time\_in\_seconds = 120 + 2\*days\_difference # Set the maximum time limit depending on date range*

## File management (input/output)

### Assessor information (.xlsx)

Structure/how to recreate if necessary:

* The .xlsx file should consists of n +2 sheets with n first names of assessors as sheet names + 1 for ‘External’ + 1 named ‘Extra’

Afbeelding met tekst, schermopname, nummer, Lettertype

Automatisch gegenereerde beschrijving

For every **internal and external** assessor: *Example*

|  |  |  |
| --- | --- | --- |
| **Key** | **Value** | **Description** |
| Activities | CURIOUS, ROLEPLAY, CASE, PAPI | Activities person is allowed to do  Options: CURIOUS, ROLEPLAY, CASE, PAPI (Datacase is below, not here!) |
| HR | TRUE | Is person part of HR? (TRUE/FALSE) |
| DATA | FALSE | Is person allowed to do Datacases? (TRUE/FALSE) |
| programs | MCP&DATA, Curious, Scrum Master | Which programs can this person assess?  Options: Should match program names in ‘Extra’ sheet (see below) |
| weeklyUnavailability | 2 | Days a person is always unavailable  Options: 0-4 (Mo-Friday) |
| Unavailability |  | Occasional unavailability (list of str dates Y-m-d format). If Outlook agenda is up-to-date, this is redundant. |
| Capacity - January | 16 | Capacity in **credits** for this month. Currently:   * PAPI costs 9 credits * Case (normal/data) costs 5 credits * Curious Case costs 3 credits   ! These **cannot be used partially**, so 16 is a difficult number (only 2x 5 + 2x 3 would be easy to match). Consider alternating for some people (e.g. instead of 7 every month, use 6 | 9 | 6 | 9 to be able to fully use available internal capacity\_ |
| Capacity - February | 16 |  |
| Capacity - March | 16 |  |
| Capacity - April | 16 |  |
| Capacity - May | 16 |  |
| Capacity - June | 16 |  |
| Capacity - July | 16 |  |
| Capacity - August | 9 | Example: From August this person will have less capacity available and will be planned less often. |
| Capacity - September | 9 |  |
| Capacity - October | 9 |  |
| Capacity - November | 9 |  |
| Capacity - December | 9 |  |

The **External sheet** is identical, but **you can put the capacity very high** (e.g. 100), since for everyone this represents the maximum that is allowed to be used. In the optimization one of the goals is to minimize external credit use, so even if you put a very high credit capacity, it does not imply you want a bias towards external assessors. Also, if you want the External to fill (or not) help with certain programs and or tasks, this should also be put here. Example:

Afbeelding met tekst, schermopname, nummer, Lettertype

Automatisch gegenereerde beschrijving

For the remaining **Extra sheet**:

Afbeelding met tekst, schermopname, nummer, Lettertype

Automatisch gegenereerde beschrijving

Example:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Key** | **Value** | | **MCP&DATA** | **Curious** | **Program1** | **Description** |
| Public Holidays |  | 2025-01-01, 2025-04-21, 2025-05-01 | | | | List of public holidays (that will be unavailable days for everyone) |
| Office Events |  | |  |  |  | “” |
| Candidate Goal - January |  | | 9 | 10 | 3 | This means the planner will try to plan 3 sessions for MCP&DATA, using the assessors that are allowed to assess this program; will try to plan 10 curious cases; and will try to plan 1 session for ‘Program1’ (these names for novel programs should be used everywhere, not their actual name, since the program can deal with ‘Program1-8’ but not unexpected program names). |
| Candidate Goal - February |  | | 9 | 8 | 3 |  |
| Candidate Goal - March |  | | 15 | 7 | 3 |  |
| Candidate Goal - April |  | | 18 | 5 | 3 |  |
| Candidate Goal - May |  | | 15 | 8 | 3 |  |
| Candidate Goal - June |  | | 12 | 7 | 0 | Note: A **value of 0 is allowed**, also all 0, then the optimization will simply not include the program. |
| Candidate Goal - July |  | | 9 | 3 | 0 |  |
| Candidate Goal - August |  | | 15 | 8 | 0 |  |
| Candidate Goal - September |  | | 18 | 9 | 0 |  |
| Candidate Goal - October |  | | 18 | 10 | 0 |  |
| Candidate Goal - November |  | | 12 | 3 | **1** | ! Currently **only multiples of 3 are used** **for the non-curious programs**! 3 implies 1 program will be planned, not 1! |
| Candidate Goal - December |  | | 6 | 6 | 0 |  |