THE UNIVERSITY OF SYDNEY

# INFO1112 A1 - Just a friendly reminder

> **Note**
>
> Students and tutors often have great suggestions to specifications. While no major changes will be made after release, this assignment specification may be clarified up to **Week 4, 27/08/2023**. Revised versions will be clearly marked and accompanying announcements made to Edstem.

In this assignment, you'll be creating a basic application called "Jafr" (short for "Just a friendly reminder"). This application helps multiple users manage their tasks and meetings on a Unix-like OS (a popular choice of OS in industry where developers might share a computer system or host web applications).

Jafr is Unix-friendly. This means that

1. Users interact with Jafr by typing commands in a command-line interface.
2. Jafr assumes that all the tasks and meetings are stored in text files that are otherwise managed by users of the shared system. Users simply edit these files themselves when they want to make changes outside of Jafr.

You will implement Jafr in Python and write a simple start up script in Bash. You will then write I/O end to end tests for Jafr.

These specifications first describe each behaviour of Jafr. The final sections describe error handling, how to write tests for Jafr and provide some hints.

## Overview

Jafr is designed to run whenever a user opens their terminal at the beginning of their day. Users can choose to view reminders that are relevant to the current day, or make changes. Changes can include sharing reminders with other users.

There are two kinds of reminders: tasks and meetings.

# Setup

Jafr primarily relies on two text files for each user: `tasks.md` and `meetings.md`. These text files are placed inside a master directory of the user's choosing.

The user chooses their master directory inside a JSON file called `user-settings.json` located at `~/.jafr/user-settings.json`. You may consider `~/.jafr/` a 'hidden' directory, for Jafr's internal use only.

> **Hint**
>
> Notice that the hidden directory `.jafr/` is inside a user's home directory which can be symbolically referred to by `~`.
> You can fetch the path referred to by `~` in Python by using `os.path.expanduser('~')`

`user-settings.json` has a single key value pair storing the absolute path to the master directory.

Sample `user-settings.json`:

```
{
        "master": "/home/dailystuff"
}
```

> **Help! What's a JSON file?**
>
> JSON is a universal file format for easy data reading and writing. There are two kinds of data structures possible to write in JSON: objects and arrays.
> Curly braces are used to define an object: a collection of name/value pairs (exactly like a dictionary in Python). Square brackets are used to define an array: an ordered list of values (exactly like a list in Python).
> You may use Python's `json` library in your implementation to read JSON files. See json.load()

> **Note**
>
> `tasks.md` and `meetings.md` for each user are given inside your scaffold. Assume the user creates these themselves using their preferred text editor.
> The `~/.jafr/` directory for each user is also given inside your scaffold. You do not have to handle the case where `~/.jafr/user-settings.json` is missing for any user. Assume Jafr has some installation script that handles this, outside of the scope of your assignment.

## Text files containing reminders

The two text files inside the master directory for each user are as follows.

### `tasks.md`

This text file contains dot pointed tasks with the following format. Dates follow DD/MM/YY, or more precisely the C standard format `%d/%m/%y` (see the datetime docs). You will only ever have to handle dates in the years 1969 - 2068 (inclusive).

```
- <task description> Due: <due date> <completion status>
```

For example

```
- Complete INFO1112 A1 Due: 01/10/23 not complete
- Acquire Twitter Due: 30/10/23 complete
- Study linux namespaces Due: 30/09/23 not complete
```

> **Hint**
>
> Notice that a task must end with `complete` or `not complete`!
> Moreover, the format implies that a task description should never contain the string `Due:`. You do not have to handle the case where a user does this.

### `meetings.md`

This text file contains dot pointed meetings with the following format. Times follow HH:mm, or more precisely the C standard format `%H:%M` (see the datetime docs).

```
- <meeting description> Scheduled: <scheduled time> <scheduled date>
```

For example

```
- Michael Mai's welcome party Scheduled: 18:00 25/08/23
- A1 marking meeting Scheduled: 09:00 01/09/23
```

> **Hint**
>
> You do not have to handle the case where a user places `Scheduled:` inside the meeting description.
> Further, as suggested by the links above, it will be easiest to use `datetime` to handle all dates/times!

# Usage

Jafr runs when `jafr.py` is executed by the Python interpreter. There is one command line argument which will contain a path (absolute or relative) to a given `passwd` file. More on this below.

For example

```
python3 jafr.py passwd
```

Jafr first displays relevant reminders (tasks followed by meetings), before showing a menu. The menu contains the following.

```
What would you like to do?
1. Complete tasks
2. Add a new meeting.
3. Share a task.
4. Share a meeting.
5. Change Jafr's master directory.
6. Exit
```

A user chooses one option only.

```
<menu num>
```

This invokes the relevant behaviour, described below. If the user enters `6`, Jafr exits. After completing a behaviour, Jafr returns to the menu.

For example

```
Just a friendly reminder! You have these tasks to finish today.
- Read INFO1112 A1 specs
- Fix bug 1 inside Jafr
- Study ELEC1601

These tasks need to be finished in the next three days!
- Shower by 03/08/23
- Organise paul's brithday by 03/08/23

You have the following meetings today!
- Michael Mai's welcome party at 18:00
- Resume writing workshop at 09:00
- Jafr dev meeting at 13:30

You have the following meetings scheduled over the next week!
- Barbenheimer marathon on 06/08/23 at 17:00
- Academic advice on 02/08/23 at 14:30
```

```
- ELEC1601 group meeting on 03/08/23 at 11:00

What would you like to do?
1. Complete tasks
2. Add a new meeting.
3. Share a task.
4. Share a meeting.
5. Change Jafr's master directory.
6. Exit
```

# Displaying tasks

Jafr will write two views of tasks to standard output. The first is a view of all tasks that are due today that have not been completed. The second is a view of all tasks that are due in the upcoming three days that have not been completed.

Today's view has the following format.

```
Just a friendly reminder! You have these tasks to finish today.
- <task description>
- <task description>
[...]
```

For example

```
Just a friendly reminder! You have these tasks to finish today.
- Read INFO1112 A1 specs
- Fix bug 1 inside Jafr
- Study ELEC1601
```

The upcoming three days' view has the following format.

```
These tasks need to be finished in the next three days!
- <task description> by <due date>
- <task description> by <due date>
[...]
```

For example

```
These tasks need to be finished in the next three days!
- Shower by 03/08/23
- Organise paul's birthday by 04/08/23
```

> **Note**
>
> Listed tasks are simply displayed in the order that they appear in `tasks.md`

> **Hint**
>
> The "upcoming" three days' view does not include the current day. Instead, "upcoming" implies the three days following the current day.

## Displaying meetings

Jafr will write two views of meetings to standard output. The first is a view of all events that are scheduled today. The second is a view of all events that are scheduled in the upcoming 7 days.

Today's view has the following format.

```
You have the following meetings today!
- <meeting description> at <scheduled time>
- <meeting description> at <scheduled time>
[...]
```

For example

```
You have the following meetings today!
- Michael Mai's welcome party at 18:00
- Resume writing workshop at 09:00
- Jafr dev meeting at 13:30
```

The upcoming 7 days' view has the following format.

```
You have the following meetings scheduled over the next week!
- <meeting description> on <scheduled date> at <scheduled time>
- <meeting description> on <scheduled date> at <scheduled time>
[...]
```

For example

```
You have the following meetings scheduled over the next week!
- Barbenheimer marathon on 06/08/23 at 17:00
- Academic advice on 02/08/23 at 14:30
- ELEC1601 group meeting on 03/08/23 at 11:00
```

> **Note**
>
> Listed meetings are simply displayed in the order that they appear in `meetings.md`

## Changing the user's master directory

Jafr allows the user to change their chosen master directory that contains `tasks.md` and `meetings.md`.

```
Which directory would you like Jafr to use?
```

The user enters an absolute path.

```
<absolute path>
```

Jafr should replace the `master` object in `~/.jafr/user-settings.json` appropriately.

> **Hint**
>
> See json.dump().

Jafr then writes a confirmation message to standard output.

```
Master directory changed to <absolute path>.
```

For example

```
Which directory would you like Jafr to use?
/home/paul/atreides_work
Master directory changed to /home/paul/atreides_work.
```

> **Hint**
>
> Jafr does not move `tasks.md` or `meetings.md` when changing the master directory. You can assume the user handles this themselves. This also allows the user to have multiple directories containing reminders and have Jafr focus on one at a time.

## Completing tasks

Jafr allows the user to mark tasks as completed. The user is first prompted for which task they would like to complete. All `not complete` tasks are shown and numbered, in the order they appear in `tasks.md`.

```
Which task(s) would you like to mark as completed?
1. <task description> by <due date:DD/MM/YY>
2. <task description> by <due date:DD/MM/YY>
[...]
```

The user then selects task(s) by their number, separated by whitespace.

```
<task num> [<task num> ... <task num>]
```

Jafr should modify `tasks.md` appropriately and write a message to standard output. Tasks inside `tasks.md` are modified in place (in the same line).

```
Marked as complete.
```

For example

```
Which task(s) would you like to mark as completed?
1. Shower by 26/07/23
2. Invite friend by 27/07/23
3. Read INFO1112 A1 specs by 28/07/23
1 3
Marked as complete.
```

If all tasks are already complete then Jafr just writes the following to standard output.

```
No tasks to complete!
```

## Adding new meetings

Jafr allows users to add meetings. The user is first prompted for a meeting description, then a date, then a time.

```
Please enter a meeting description:
<meeting description>
Please enter a date:
<scheduled date>
Please enter a time:
<scheduled time>
Ok, I have added <meeting description> on <scheduled date> at <scheduled
time>.
```

Jafr should then modify `meetings.md` appropriately. A meeting is appended to the bottom of `meetings.md` as follows.

```

##### added by you
```

```
- <meeting>
```

The user is also prompted to optionally enter people to share the meeting with.

```
Would you like to share this meeting? [y/n]:
Who would you like to share with?
<user ID> <user name>
<user ID> <user name>
[...]
```

See more about sharing below.

## Sharing tasks and meetings

Jafr allows users to share tasks or meetings from their own `tasks.md` and `meetings.md` files with other users.

The user is first prompted for which task (or meeting) they would like to share. They are shown all tasks (or meetings) regardless of completion or scheduled date.

```
Which task would you like to share?
1. <task description> by <due date>
2. <task description> by <due date>
[...]
```

The analogous meetings option contains numbered lines with `<meeting description> on <scheduled date> at <scheduled time>`.

The user then selects *one* task (or meeting) by its number in the shown list.

```
<num>
```

The user is then prompted for the user IDs with whom they would like to share their selection. They should not be shown their own user ID here.

```
Who would you like to share with?
<user ID> <user name>
<user ID> <user name>
[...]
```

The user then selects users by their user ID, separated by whitespace.

```
<user ID> [<user ID> ... <user ID>]
```

Jafr should then append the selected task or meeting to the other user's `tasks.md` or `meetings.md` appropriately, as follows.

```
##### shared by <user name>
- <task>
```

> **Hint**
>
> The above heading `##### shared by <user name>` is always created, regardless of previous sharing history! You may assume the user will clean up their text file in their own time, after noticing added meetings.
> This applies to **Adding new meetings** in your own `meetings.md` as above too.

A confirmation message is finally written to standard output ( `Task shared.` or `Meeting shared.` ).

For example

```
Which task would you like to share?
1. Fix bug 1 in Jafr by 08/08/23
2. Apply for research grant by 10/08/23
3. Shower by 03/08/23
1
Who would you like to share with?
0001 michaelmai
0002 hazemelalfy
0003 paulatreides
0004 prathampurohit
0001 0004
Task shared.
```

> **Note**
>
> Other users' `tasks.md` and `meetings.md` will be inside their own master directory, as listed in their `~/.jafr/user-settings.json`.

## How does Jafr find other users' home directories?

As you may have realised, in order for Jafr to append tasks/meetings to other users' `.md` files, it must be able to locate the `user-settings.json` of each user. However the location of `user-settings.json` depends on the current user's home directory! How will the application find other users' home directories?

All users' home directories will be contained inside a given `passwd` file.

### `passwd` file

You will be given a `passwd` file (it is already present in your scaffold). This is a text file where each line denotes a different user on the shared computer system. Each line contains a user's username, hashed password, user ID, group ID, user ID info, home directory, and default shell.

```
<username>:<password>:<user ID>:<group ID>:<user ID info>:<home
directory>:<default shell>
```

Example `passwd` file:

```
michaelmai:x:0001:8888:staff user:/michael:/bin/bash
hazemelalfy:x:0002:8888:staff user:/hazem:/bin/bash
paulatreides:x:0003:1112:student user:/paul:/bin/bash
prathampurohit:x:0004:8888:staff user:/pratham:/bin/bash
```

**Help! What's a `passwd` file?**

`passwd` files are universal in Unix-like systems. They always contain the above information. This is how the OS remembers information about each user! However, usually the file is stored at `/etc/passwd`. Take a look at `/etc/passwd` on your own system!
You may notice that there is some unnecessary information in the `passwd` file. Jafr only requires the username, user ID and home directory.

**Hint**

You can find the current user's username using the `environ` object from the `os` module. https://docs.python.org/3/library/os.html#os.environ
This a dictionary containing environment variables, exposed by the `os` module. You can access the username as follows.
`os.environ['USER']`
**Please use this method. Others have been found to not work correctly on Edstem.**

## Running Jafr when Bash is started

Jafr is designed to be run whenever a terminal running Bash is started. This section explains how this can be achieved.

Most Unix-like operating systems shipped with Bash allow users to customise a Bash script that is run whenever Bash is started. This script is called `.bashrc` and is stored inside the user's home directory (i.e. at `~/.bashrc`).

Once you have read the Usage section, have a go at modifying `~/.bashrc` on your system so that it runs Jafr when bash is started. Your submission must include a `.bashrc` file that is able to do this.

You can check for the expected behaviour as follows.

1. Ensure your terminal runs bash.
2. Open a terminal window. On some systems you may need to run `bash` yourself (see Update below).
3. See if Jafr displays its views correctly.

> **Update**
>
> Some systems (which include Edstem's Arch and macOS) may not run `.bashrc` in login shells. These are shells that are started upon logging in.
> However, on *any* system you may start Bash again by simply executing `bash`. This will be a "non-login" shell and will always run `.bashrc`.

You may provide comments inside a `readme.md` to your marker about any expectations you have for this to work correctly.

> **Hint**
>
> In order to be awarded this section of the assignment, you may assume `jafr.py` and `passwd` exists inside every user's home directory.
> Remember `.bashrc` is a universal script. This means you can find plenty of help by looking this up online.

# Error handling

## Setup

You do not have to handle a missing `~/.jafr` directory or `~/.jafr/user-settings.json` file for any user.

If either `tasks.md` or `meetings.md` are missing from the user's chosen master directory then Jafr does not display any reminders or show the menu. Jafr writes the following message to standard error before simply exiting.

```
Missing tasks.md or meetings.md file.
```

If the user's chosen master directory does not exist, then Jafr does not display any reminders or show the menu. Jafr writes the following message to standard error before exiting.

```
Jafr's chosen master directory does not exist.
```

## Text files

`tasks.md` and `meetings.md` can contain any amount of text, and not necessarily only tasks/meetings. A line is only considered a task or meeting if it is a dot point (the line starts with `-`).

> **Note**
>
> Any amount of indentation is allowed before a dot point!

For example, suppose `tasks.md` contains

```
### School tasks
- Go to school Due: 01/08/23 not complete
- English homework Due: 02/08/23 not complete
        - English homework introduction Due: 01/08/23 not complete

### Chores
- Take out trash Due: 01/08/23 not complete

##### shared by hazemelalfy
- Study linux namespaces Due: 04/08/23 not complete
```

Then there are five tasks recognisable by Jafr.

> **Help! Why do I have to parse these files for dot points?**
>
> Jafr is designed to rely on text files that are otherwise freely editable by users. Different users may have different methods of organising their tasks or meetings. Think of Jafr as a parser that adapts to each user's context.
> The `re` module may be helpful here.

## Displaying tasks and meetings

Any malformed tasks or meetings are skipped when displaying them. No error message is given.

> **Hint**
>
> For those new to Python, a `try` block will be useful here to simply skip malformed tasks/meetings after a dot point.

## Menu

If the user enters an invalid `<menu num>`, then Jafr prompts them with a single-line explanation written to standard output, before allowing input again.

> **Hint**
>
> It is up to you what explanation is given. Our only requirement is that it fits in one line and is appropriate.

## Completing tasks and adding new meetings

Any malformed tasks or meetings are skipped when displaying them. No error message is given.

If the user enters an invalid option at any point, Jafr prompts them with a single-line explanation written to standard output, before allowing input again.

Specifically, if the user enters an invalid;

- `<task num>`
- `<meeting description>`
- `<scheduled date>`
- `<scheduled time>`,

then they are prompted with a single-line explanation.

## Sharing tasks and meetings

Any malformed tasks or meetings are skipped when displaying them. No error message is given.

If the user enters an invalid option at any point, Jafr prompts them with a single-line explanation written to standard output, before allowing input again.

Specifically, if the user enters an invalid;

- `<num>`
- `<user ID>`

then they are prompted with a single-line explanation.

You do not have to handle a malformed `passwd` file.

## Changing the user's master directory

You do not have to handle any invalid user input here. If the chosen master directory ends up being malformed in `user-settings.json`, the error handling in **Setup** above will at least cover this when opening Jafr.

# Writing your own tests

You will write I/O end-to-end tests for Jafr inside a directory `tests/`. You must test

1. Displaying reminders.
2. Completing tasks.
3. Adding new meetings.

You do **NOT** have to test

1. Sharing tasks and meetings (including sharing after adding new meetings)
2. Changing the user's master directory
3. Running Jafr when Bash is started

It is not required in Assignment 1 for your tests to be automated via a testing Bash script. (Doing so is still encouraged, for the bold and the brave. It may be required in Assignment 2, so this is an opportunity to receive feedback.)

> **Hint**
>
> We recommend creating tests locally, then uploading them at once in a zip file to Edstem. This will make it easier to handle the hidden directories and edit text files.

The `tests/` directory will contain

- `.in` files
- `.out` files
- a `passwd` file with test users of your own construction.
- directories that will be "home directories" of your test users. Each of these require
  - an appropriate `.jafr/user-settings.json` file.

- `tasks.md` and `meetings.md` files

> **Hint**
>
> "Home directories" of your test users will need to be defined in your `tests/passwd` file. While `tests/passwd` may not be used by your tests (which do not need to cover sharing), it will still be useful for your marker.

A test is constructed by creating a `.in` file and a `.out` file. The filename prefix to these should describe the test.

The `.in` file should contain all user input that will be written to standard input. The corresponding `.out` file should contain output that is expected to be written by Jafr to standard output.

The other files and directories present in `tests/` are shared across all tests. You may write three to five test users and a handful of test cases for each user. That is, each test case should choose a user to rely on.

You can use your own tests as follows.

1. Run `python3 jafr.py tests/passwd`.
2. Run each test by manually inspecting the appropriate `.in` and `.out` file. Enter input from the `.in` file. Compare your program's actual output to the `.out` file.

Any changes to `tasks.md` and `meetings.md` files made by your tests will not be captured with a `.in` and `.out` file. You do not have to capture these changes in your tests.

You may include any comments about your testing in a `test_readme.md` to your marker if you wish, such as current date and current user for each test.

See Daniel's video on Canvas for more about manual testing. See pinned post #284 on Edstem for more help around changing users for each test while on your local machine.

# Further hints and where to start

> **Hint**
>
> The allowed libraries themselves provide hints on how you can make this assignment easier! See **Submission and marking** below.

**Hint**

You might have noticed that tasks and meetings are saved in Markdown files. These are just text files (Unix users' favourite type of files), with the additional benefit of mark up via very simple formatting. Markdown happens to be used everywhere across software documentation, and so you can learn it if you like. However it's not necessary at all for this assignment.

**Hint**

**This hint only matters if you're aiming to submit automated end-to-end tests that can run on Ed** (not required).
Awkwardly, because of how Ed submission workspaces work, each test user's home directory path in `tests/passwd` would need to begin with `/home/tests/`.
Note the default user's home directory on Ed is `/home`.

Here's a suggested order of what to work on in this assignment.

1. Draw a picture for yourself of how Jafr interacts with the filesystem.
2. Plan your code. What functions will you write?
3. Work on setup and displaying reminders. Write tests.
4. Work on making modifications to tasks/meetings. Write tests.
5. Work on sharing tasks/meetings. Write tests.
6. Work on `.bashrc`. You might be surprised how simple the solution is here, but it takes some time to understand what is required.
7. Write final tests. Try to catch bugs in your own code and fix them.

# Submission and marking

An Edstem workspace will be made available for you to test and submit your code. Public test cases will be released up to **25/08/23**. For Assignment 1, there will be no private or hidden test cases.

## Files to submit

Your submission should include

```
jafr.py
.bashrc
tests/
```

Optionally, your submission may include

```
readme.md
test_readme.md
```

You can remove staff's comments present in the scaffold's `readme.md` or just add your own at the top of the file.

## Allowed libraries

```
sys
json
datetime
re
os
typing
```

All other libraries are disallowed.

The following functions are banned inside the `os` module. See thread #468 on Edstem for more discussion.

```
os.walk()
os.fwalk()
os.listdir()
```

## Marks weighting

This assignment will be marked out of 10. Each component is weighted as follows.

- Displaying reminders: 20% (2 marks)
- Completing tasks, adding new meetings and changing master directory: 30% (3 marks)
- Sharing tasks and meetings: 20% (2 marks)
- `.bashrc`: 10% (1 mark)
- `/tests` 20% (2 marks)

> ⚠ **Warning**
>
> Any attempts to deceive the automatic marking system will result in an immediate zero for the entire assignment. Negative marks can be assigned if you do not properly follow the assignment description, or your code is unnecessarily or deliberately obfuscated.

# Academic Declaration

By submitting this assignment, you declare the following:

*I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work and has not been copied from other sources or been previously submitted for award or assessment.*

*I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgment from other sources, including published works, the Internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.*

*I realize that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.*

*I acknowledge that the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.*