Mini Project: To-do list

In this mini project, you're going to create a console program to keep a to-do list with various tasks and their deadlines. The program should allow the user to add, delete and browse tasks on the list. Most importantly, the program should keep its records in a text file so that the data is not lost when the user closes the program and opens it up again.

1. Main menu

The main menu should allow the user to show existing tasks, add a new task, complete a task (= remove it from the todo list) or exit the program.

```
== TODO LIST ==
[1] show tasks
[2] add task
[3] complete task
[4] exit
Your choice:
```

2. Showing tasks

The program should show the current to-do list with all the tasks loaded from a text file. For each task, we should show its id, task description and its deadline. After showing the list, the program should go back to the main menu. Example below:

```
[YOUR TASKS]
f7755225-c6e9-4747-bc18-bc3602823325 | clean up the room | tomorrow
7ceb038b-117c-4abc-827d-a142506c8a1f | go to the shop | today
```

If there are no tasks in the list, the program should show a corresponding message:

```
[YOUR TASKS]
Empty list
```

3. Adding a task

When adding a new task, the program should ask the user what the task is and what the deadline is. Then, the program should generate a unique, random id for the given task and store it in a text file. After saving the task, the program should go back to the main menu. Example below:

```
[ADD TASK]
What is the task? go to the shop
What is the deadline? today
```

4. Completing a task

Completing a task should remove it from the list. The program should first show the list of all existing tasks (similar to point 2), and then ask for the id of the task to complete. After completing the task, the program should go back to the main menu. Example below:

```
[COMPLETE TASK]

[YOUR TASKS]
5a9285bb-40f9-4512-909b-e59b5fe514de | clean up the room | tomorrow
1e18f69d-dbd4-4b0f-964a-efaa713d0f0f | go to the shop | today

Enter id to complete: 5a9285bb-40f9-4512-909b-e59b5fe514de
```

If there are no tasks in the list, the program should simply show a proper message and then immediately go to the main menu.

```
[COMPLETE TASK]

[YOUR TASKS]
Empty list

No tasks to complete
```

5. Handling errors

Try to include some error handling in your program: think about some problems with user input and about file handling errors.

6. Other details

If there is anything else about the program not specified in this instruction, you are free to solve it as you considered appropriate. Remember that in programming there are always different ways you can solve the same problem. :)

7. Sample output

```
== TODO LIST ==
[1] show tasks
[2] add task
[3] complete task
[4] exit
Your choice: 1

[YOUR TASKS]
Empty list

== TODO LIST ==
[1] show tasks
[2] add task
[3] complete task
[4] exit
Your choice: 2

[ADD TASK]
What is the task? study Polish
What is the deadline? Wednesday

== TODO LIST ==
[1] show tasks
[2] add task
[3] complete task
[4] exit
Your choice: 1

[YOUR TASKS]
64078209-8d84-4204-96e6-bc1f00360a4b | study Polish | Wednesday

== TODO LIST ==
[1] show tasks
[2] add task
[3] complete task
[4] exit
Your choice: 3

[COMPLETE TASK]

[YOUR TASKS]
64078209-8d84-4204-96e6-bc1f00360a4b | study Polish | Wednesday

Enter id to complete: 64078209-8d84-4204-96e6-bc1f00360a4b
```

```
== TODO LIST ==
[1] show tasks
[2] add task
[3] complete task
[4] exit
Your choice: 4
```

6. Hints & tips

1. For this program, you will need to work with a text file. You will need to be able to open the text file to read tasks from it, and you will need to be able to add new tasks to the file. You can use the **append** ('a') mode to add new tasks at the end of the file. If you use the write (‘w’) mode with an existing file with some tasks inside, you will lose them.

2. You will need to think about a data format so that you can easily add, show and delete tasks from the file. One suggestion is to have one task per file line, and then to separate the id, the task and the deadline by a fixed character, for example a semicolon ;
Example file content:

```
9413d147-37cd-4e14-95fb-9a955f588a7f;fix the car;Thursday
5a8882f4-db98-46cd-8f9e-b757d41fdfa3;read the book;weekend
```

If you decide on a data format like this, it will be easy for you to read the file: you can use the `readlines()` function to get a list of lines (= a list of tasks) and then you can split a single line into three separate elements: id, task and deadline. To that end, you can simply use:

```
line.split(';')
```

Remember that if you choose the semicolon as the field delimiter, you will need to make sure that the user does not use semicolons inside the task description and/or deadline!

3. Each task should have an automatically generated id. If you don't add ids, it will be difficult for the user to delete tasks (how can they indicate which task should be deleted, then?). You must make sure that the id is unique for each task. How can you do that?

    a. You could assign id=1 to the first task, id=2 to the second task, and so on. However, you will need to decide what to do if a task is deleted. If you have three tasks with ids 1, 2, and 3, what should happen when you delete task 2? Should the next new task be assigned id=2 or id=4? How do you keep track of that?

    b. [recommended] In modern programming, we typically don't use option (a). Instead, we try to generate random unique identifiers such as

    ```
    64078209-8d84-4204-96e6-bc1f00360a4b
    ```

It's not easy to come up with an algorithm to generate unique ids on your own, but fortunately Python has a module for that. Such ids are typically called UUIDs. You can read more about them on Wikipedia. The module that you can import is **uuid**. It's actually very simple to use. If you want a new unique id as a string, you can simply invoke `str(uuid.uuid4())`. There are also other generation options, this one is just a suggestion.

If you are still not sure how to write the program, remember that you can always take a look at the sample solution that I've prepared for you. It is available as a downloadable resource in the same lecture.