Command Line Calendar

So far, you've used Python to build a variety of things, including calculators and games. In this project, we'll build a basic calendar that the user will be able to interact with from the command line. The user should be able to choose to:

- View the calendar
- Add an event to the calendar
- Update an existing event
- Delete an existing event

The program should behave in the following way:

1. Print a welcome message to the user
2. Prompt the user to view, add, update, or delete an event on the calendar
3. Depending on the user's input: view, add, update, or delete an event on the calendar
4. The program should never terminate unless the user decides to exit

Let's begin!

Command Line Calendar

1. As in previous projects, it's good practice to let other developers know what your program does.

Begin by including a multi-line comment that starts on line 1 that describes what your program will do. You can use the instructions above to help you write the comment.

2. On the next line, use a function import to import `sleep` from the `time` module.

3. Since this is a calendar program, we'll need to access date and time quite often. We've been using the familiar `datetime` function, but sometimes it's necessary to use other functions. Let's try a new function out.

On the next line, use a function import to import `strftime` from the `time` module.

4. Actually, it's repetitive to import two different functions from the same module on two different lines. Let's fix that.

Go ahead and delete that last two lines of code you wrote (the function imports). Whenever possible, it's better to not repeat yourself when coding.

Use a function import to import both `sleep` and `strftime` from the `time` module, all in one line of code.

5. Great, our program is a little more concise now.

The instructions ask to build a calendar that starts with a welcome message for the user. It'd be nice for the calendar to know who the user is.

On the next line, add a constant variable that stores the user's first name as a string. Set it equal to your name (or another name).

6. What data structure can we use to store calendar events? You learned about lists and dictionaries in the Python course, so we'll have to choose one of those two.

   To decide, let's think about what a calendar requires. Ideally, a calendar allows users to at least associate an event with a date, as a pair.

   That sounds a lot like the functionality that a dictionary provides, so we'll use a dictionary to build the calendar. Our calendar will store the dates as keys and the events as values.

7. On the next line, create an empty dictionary called `calendar`.

8. Great! We have the structures in place that we'll use to build the rest of the program. Let's start by adding the welcome message.

   On the next line, create a function called `welcome()`.

9. On the next line, inside of the function, print a welcome message to the user. Use concatenation to include the message and the user's first name.

10. Next, print a message to let the user know the calendar is opening. On the next line, sleep the program for 1 second.

11. It's time to use the new `strftime` function we imported. The cool thing about the `strftime` function is that printing different units of time (months, days, years, hours, etc.) is simpler than having to call separate functions for each aspect (the way `datetime` works).

    Look at the `strftime` documentation to understand how to use it (the "Directive" table is especially useful). Reviewing documentation is a common practice among professional software engineers.

    On the next line, print the current date in the following format: `Full weekday name Month Day, Year`. Use concatenation and `strftime` to help you.

12. On the next line, print the current time in the following format: `H:M:S`. Use concatenation and `strftime` to help you.

    Next, sleep the program for 1 second.

13. You succesfully used a new function, congrats! On the next line, print `"What would you like to do?"` to the user.

14. Perfect! Our welcome message function is now complete. Let's start building the calendar's functionality.

    Create a new function called `start_calendar()`.

15. When the calendar starts, the first thing we'd like to do is welcome the user. On the next line, inside of `start_calendar()`, call the `welcome()` function.

16. The project instructions ask that the project terminate only when the user voluntarily exits the program. In this case, we can use a `while` loop, since while loops will continue running as long as a condition is true.

    On the next line, create a variable called `start` and set it equal to `True`. Next, add a `while` loop that uses `start` as the Boolean condition.

    Since `start` is `True`, we have ensured that this loop will continue to run, unless `start` changes value.

17. Now we'll start building the most important part of this program, the actual calendar (along with its required behavior).

    Inside of the `while` loop, prompt the user to enter `A to Add, U to Update, V to View, D to Delete, X to Exit:`. Store their input in a variable called `user_choice`.

    On the next line, convert `user_choice` to upper case.

☐ **18.** Great! We now have the user's input. The instructions require that a user be able to:

- View the calendar
- Update the calendar
- Add to the calendar
- Delete from the calendar

Let's start with the behavior that will be the easiest to implement: viewing the calendar.

☐ **19.** Keeping inside the while loop, add an `if` statement that checks if the user's choice is `V` (for View).

☐ **20.** If the user would like to view the calendar, first we have to make sure the calendar contains events. Otherwise, we'll print the calendar for them.

Inside of the current `if` statement, add another `if` statement that checks if there are no dates (keys) in the calendar (i.e. less than 1 key).

You can use the `.keys()` function on `calendar`, and use the `len()` function to check the length (size) of the keys.

Inside of the new `if` statement, print a message to the user letting them know the calendar is empty.

☐ **21.** That will take care of notifying the user if their calendar is empty. However, if it's not empty, we need to print the calendar for them.

Add an `else` block that corresponds to the `if` block you just added. Inside of the `else` block, print the calendar.

☐ **22.** Perfect - you just built a fifth of the functionality that's needed! Keep in mind that what you just coded represents the general flow of how each behavior will function. Now let's add functionality to update the calendar.

Add an `elif` block (corresponding to the first `if` block you coded) that checks if the user's choice is `U` (for Update).

☐ **23.** On the next line, prompt the user for the date with the following: `"What date? "`. Store their input into a variable called `date`.

On the line after that, prompt the user for the update with the following : `"Enter the update: "`. Store their input into a variable called `update`.

☐ **24.** On the next line, update `calendar` by adding the `update` to the `date` that the user specifies.

☐ **25.** Now that we've made the update possible, print a message to the user on the next line about the update being successful.

On the line after that, print the calendar.

☐ **26.** Great, we've built two of the five required calendar functionalities! It's time to add the next piece of behavior: adding to the calendar.

Add an `elif` block (again, corresponding to the first `if` block you coded) that checks if the user's choice is `A` (for Add).

☐ **27.** On the next line, prompt the user for their input with the following: `"Enter event: "`. Store their input into a variable called `event`.

On the line after that, prompt the user for their input with the following : `"Enter date (MM/DD/YYYY): "`. Store their input into a variable called `date`.

☐ **28.** When the user inputs a date, they must format it as `MM/DD/YYYY`, including the forward slashes. But what if the date they enter doesn't match the format we've specified? We have to handle that possibility.

On the next line, add an `if` statement that checks if the length of `date` is greater than 10.

Note: A date in the format `MM/DD/YYYY` contains 10 characters if you include the forward slashes.

☐ **29.** Great, but what if the user tries to schedule something for a year in the past? Calendars usually allow this, but our calendar won't.

Expand the `if` statement by using Boolean `or` and checking if the year the user entered occurs before the year that the `strftime` function returns (in other words, the current year). You can use the `<` operator to achieve this.

Since characters in a string can be accessed by index, use list slicing to retrieve the year from `date`.

Use `%Y` to retrieve the year from `strftime`.

Both years retrieved will be in the form of string, but we need to compare integers, so convert both using `int()`.

☐ **30.** On the next line, inside the `if` block, print a message to user indicating that an invalid date was entered.

☐ **31.** Next, still inside the `if` block, ask the user if they would like to try again by prompting them with: `"Try Again? Y for Yes, N for No: "`. Store their input in a variable called `try_again`.

Directly after that, convert `try_again` to uppercase.

☐ **32.** Next, add another `if` statement that checks if `try_again` is `Y` (for Yes).

If the user selects `Y`, we should allow the program to continue. On the next line use the `continue` keyword. The `continue` keyword will start the loop from the beginning again.

Learn more about the `continue` keyword here.

☐ **33.** Otherwise, if they select `N`, we have to exit the program.

Add a corresponding `else` block. Inside the block, set `start` equal to `False` to exit the loop and quit the program.

☐ **34.** Great! That takes care of invalid years. Let's let the user add events to the calendar now.

Add an `else` block that corresponds to the `if` block. On the next line, add to `calendar` by adding the `event` to the `date` the user specifies.

Then, on the next line, print a message saying the event was successfully added. After that, print the calendar.

☐ **35.** We're almost there! Only one more piece of functionality needs to be added: deleting an event.

Add an `elif` block (corresponding to the first `if` block you coded) that checks if the user's choice is `"D"` (for Delete).

☐ **36.** It doesn't make sense to delete from a calendar that's already empty! Let's make sure the user knows that.

On the next line, inside of the block, add an `if` statement that checks if there are no dates in the calendar (i.e. less than 1 key). You can use the `.keys()` function on our `calendar`, and use the `len()` function to check the length of the keys.

Inside of the `if` statement, print a message to the user letting them know the calendar is empty.

☐ **37.** Otherwise, we should let the user delete an event.

Add an `else` block. Then, on the next line, prompt the user for their input with the following: `"What event?"`. Store their input into a variable called `event`.

☐ **38.** In order to delete the event, we're going to have to search for it in the calendar. This will involve iterating through the dates (keys) and finding the matching event (value).

On the next line, still inside the `else` block, add a `for` loop that iterates through the keys using a `date` variable. Use `.keys()` to help you.

39. Now it's time to decide whether or not the event should be deleted. It should be deleted if we encounter the event during the iteration.

    On the next line, add an `if` statement that checks if `event` is equal to `calendar[date]`. This statement checks if the event exists.

40. Great! If the event does exists, we should delete it.

    On the next line, use the `del` statement to delete `calendar[date]`. Learn more about the `del` statement here.

41. On the next line, print a message to the user indicating that the event was successfully deleted. Then, directly after, print the calendar.

42. Oh right! We also have to check if a valid event was entered by the user.

    Add a corresponding `else` block. Inside, print a message indicating that an incorrect event was specified.

43. Finally, let's add some functionality that will let the user exit the program.

    Add an `elif` block (again, corresponding to the very first `if` block you coded) that checks if the user's choice is `"X"` (for Exit). On the next line, inside the block, exit the program by changing the value of `start`.

44. The homestretch! If a user enters garbage when prompted by the program, we should exit.

    This time, add an `else` block (again, corresponding to the very first `if` block you coded).

    Inside of the block, print a message indicating that an invalid command was entered. On the next line, exit the program.

45. The program won't run unless we call it. For the grand finale, call the `start_calendar()` function on the next line (outside of any function).

46. Let's see if the calendar works!

    First, click Save. Then, in the terminal, type the following command and press "Enter" on your keyboard:

    ```
    python Calendar.py
    ```

    Interact with the calendar, making sure to test all the functionality. If there is something that didn't work, check the step associated with that part of the project and see if you can debug it.

    Can you think of ways to improve or add to the calendar? Happy coding!