CSCA08H Assignment 1: Airplane Tickets

Goals of this Assignment

- Use the Function Design Recipe to plan, implement, and test functions.
- Write function bodies using variables, numeric types, strings, and conditional statements. (You can do this whole assignment with only the concepts from Weeks 1, 2, and 3 of the course.)
- Learn to use Python 3, Wing 101, provided starter code, a checker module, and other tools.

Starter code

For this assignment, we are giving you some files, including Python starter code files. Please download the Assignment 1 Files and extract the zip archive.

There are two starter code Python files and a Python program that helps you check (**not fully test!**) your solutions for Python style violations:

- `tickets.py`

  This file contains some code that sets up your use of constants from `constants.py`, a helper function for you to use (see the table of functions to implement below), **solutions** for several required functions for you to use as examples, as well as **headers and docstrings** (but not bodies) for several function you are to write, to help you get started. Your job is to complete this file.

- `constants.py`

  This file contains some useful constants that you should use in your solution. You will **not** modify this file.

- `a1_checker.py`

  This is a checker program that you should use to check your code. See below for more information about `a1_checker.py`.

The Airline Ticket

The typical airplane boarding pass contains many different pieces of information, including passenger information, seat, flight number, flight time, and so on. In this assignment, we'll work with our own version of information you might find on a boarding pass.

We'll refer to the data we will work with just as a "ticket". A ticket is a string, where different pieces of the string represent different kinds of information. Our ticket will include the following information:

- The date the ticket is for, including year, month, and day.
- Departure and arrival airport codes of the flight the ticket is for.
- The seat assigned to the ticket, including the row number and the seat within that row.
- The passenger's frequent flyer number, if applicable.

A valid ticket will have either 17 or 21 characters. Below, we describe what the different index ranges mean in a ticket.

- The date of the ticket consists of three pieces of information: year, month, and day.
    - Characters at indices from `YR` to `YR+3`, inclusive, represent the year the ticket is for. In a valid year, these four characters must be digits. We do not place any other restrictions on the validity of the year field.
    - Characters at indices `MON` and `MON+1` represent the month the ticket is for. In a valid month, these two characters must be digits that represent a number between 1 and 12, inclusive.
    - Characters at indices `DAY` and `DAY+1` represent the day the ticket is for. In a valid day, these two characters must be digits that represent a number between 1 and the maximum number of days for the given month and year. Recall that months January, March, May, July, August, October and December have 31 days, months April, June, September, and November have 30 days, and the month of February has 28 days in a non-leap year and 29 days in a leap year. A leap year is a positive integer multiple of 4 (except for years evenly divisible by 100, but not by 400).
- Characters at indices from `DEP` to `DEP+2`, inclusive, represent the code of the departure airport. In a valid code, these three characters must be letters.

- Characters at indices from `ARR` to `ARR+2`, inclusive, represent the code of the arrival airport. In a valid code, these three characters must be letters. In a valid ticket, departure and arrival airports must be different.
- Characters at indices `ROW` and `ROW+1` represent the row number of the ticket. In a valid row number, these two characters must be digits.
- The character at index `SEAT` represents the seat in a given row. A valid seat is the value of one of the constants `SA`, `SB`, `SC`, `SD`, `SE`, or `SF`, which for us represent seats "A", "B", "C", "D", "E", and "F", but can be translated into a different language by changing the values of these constants. Our aircrafts all have six seats in each row, with three seats on each side of the aisle: `SA`, `SB`, `SC` on one side, and `SD`, `SE`, and `SF` on the other. Seats `SA` and `SF` are window seats, seats `SB` and `SE` are middle seats, and seats `SC` and `SD` are aisle seats.
- Characters starting at index `FFN` are special and represent the frequent flyer number for the passenger this ticket belongs to, if they have one. The value of `FFN` must be 17 on a valid ticket, meaning the frequent flyer number always appears as the last record in the ticket string. Some tickets do no have a record for frequent flyer number, meaning the corresponding string is the empty string `''`. Some tickets have a record for frequent flyer number, in which case this number appears between indices 17 and 20, inclusive. The empty string is a valid frequent flyer number. A valid non-empty frequent flyer number must contain exactly four digits, and the sum of the first three digits modulo 10 must be the same as the last (fourth) digit.
- Here are some examples of valid tickets.
    - Suppose the values of our constants are `YR = 0`, `MON = 4`, `DAY = 6`, `DEP = 8`, `ARR = 11`, `ROW = 14`, `SEAT = 16`, and `FFN = 17`. The ticket below is for December 21, 2023 for a flight from YYZ to YEG in seat 25F. The passenger's flyer number is 4442. (Note that `4 + 4 + 4 mod 10 is 2`).

      `20231221YYZYEG25F4442`

    - Suppose now the values of our constants are `YR = 10`, `MON = 8`, `DAY = 6`, `DEP = 0`, `ARR = 3`, `ROW = 14`, `SEAT = 16`, and `FFN = 17`. The ticket below is for September 20, 2024 for a flight from YYZ to LAS in seat 16D. The passenger's flyer number is 4420. (Note that `4 + 4 + 2 mod 10 is 0`).

      `YYZLAS2009202416D4420`

Constants

Constants are special variables whose values should not change once assigned. A different naming convention (uppercase pothole) is used for constants, so that programmers know to not change their values. For example, in the starter code, the constant `YR` (for "year") is assigned the value 1 at the beginning of the module and the value of `YR` should never change in your code. When writing your code, if you need to use the index at which a ticket contains the year, you should use `YR` and not the value 1. The same goes for the other constant values.

Using constants simplifies code modifications and improves readability and changeability. For example, if we later decide to use French instead of English, we would only have to make the changes in one place (the `WINDOW`, `AISLE`, and `MIDDLE` assignment statements in the file `constants.py`), rather than throughout the program. This also makes our code more readable — whether we use `1` or any other index to store the year on the ticket, we write our code using the constant `YR` so it is clear to the reader what we mean.

You should **read the file `constants.py` carefully** to understand the purpose of each defined constant. You must use these constants in your code and not the literal values. For example, you must use `YR` instead of `0` to earn full marks.

What to do

In the starter code file `tickets.py`, complete the following function definitions. **Note that some are already provided for you to use as examples!** Use the Function Design Recipe that you have been learning in this course. We have included the type contracts in the following table; please read through the table to understand how the functions will be used.

We will be evaluating your docstrings in addition to your code. Please include **at least two examples** in each docstring. You will need to paraphrase the full descriptions of the functions to get an appropriate docstring description. Your docstring examples should be **valid doctests** — we will **run** them as part of evaluation of your program.

Task 1: Getting the Information (7 correctness marks)

Advice: Start working on these functions as soon as the assignment is released. You have already learned all material needed to complete them.

Functions to write for A1: Getting Ticket Info

Function name: (Parameter types) -> Return type	Full Description (paraphrase to get a proper docstring description)
`get_date: (str) -> str`	The parameter represents the ticket. The function should return the date on the given ticket in the format `YYYYMMDD`. You may assume the input ticket is valid. For example, if the constants have values `YR = 0`, `MON = 4`, `DAY = 6`, and the input ticket string is `'20231221YYZYEG25F4442'`, this function should return `'20231221'`. As another example, if the constants have values `YR = 10`, `MON = 8`, `DAY = 6`, and the input ticket string is `'YYZLAS2009202416D4420'`, this function should return `'20240920'`.
`get_year: (str) -> str`	The parameter represents the ticket. The function should return the year on the given ticket in the format `YYYY`. You may assume the input ticket is valid. See function `get_date` for examples.
`get_month: (str) -> str`	The parameter represents the ticket. The function should return the month on the given ticket in the format `MM`. You may assume the input ticket is valid. See function `get_date` for examples.
`get_day: (str) -> str`	The parameter represents the ticket. The function should return the day on the given ticket in the format `DD`. You may assume the input ticket is valid. See function `get_date` for examples.
`get_departure: (str) -> str`	The parameter represents the ticket. The function should return the departure airport code on the given ticket. You may assume the input ticket is valid.

	For example, if the constant `DEP` has value `8`, and the input ticket string is `'20231221YYZYEG25F4442'`, this function should return `'YYZ'`. As another example, if the constant `DEP` has value `0`, and the input ticket string is `'YYZLAS2009202416D4420'`, this function should return `'YYZ'`.
`get_arrival: (str) -> str`	The parameter represents the ticket. The function should return the arrival airport code on the given ticket. You may assume the input ticket is valid. For example, if the constant `ARR` has value `11`, and the input ticket string is `'20231221YYZYEG25F4442'`, this function should return `'YEG'`. As another example, if the constant `ARR` has value `3`, and the input ticket string is `'YYZLAS2009202416D4420'`, this function should return `'LAS'`.
`get_row: (str) -> int`	The parameter represents the ticket. The function should return the row number on the given ticket. You may assume the input ticket is valid. For example, if the constant `ROW` has value `14`, and the input ticket string is `'20231221YYZYEG25F4442'`, this function should return `25`.
`get_seat: (str) -> str`	The parameter represents the ticket. The function should return the seat on the given ticket. You may assume the input ticket is valid. For example, if the constant `SEAT` has value `16`, and the input ticket string is `'20231221YYZYEG25F4442'`, this function should return `'F'`.
`get_ffn: (str) -> str`	The parameter represents the ticket. The function should return the frequent flyer number on the given ticket. You may assume the input ticket is valid. For example, if the input ticket string is `'20231221YYZYEG25F4442'`, this function should return `'4442'`. If the input ticket string is `'20231221YYZYEG25F'`, this function should return `''`.

Task 2: Validating the Information (12 correctness marks)

Advice: Start working on these functions as soon as the assignment is released. You can complete the headers and docstrings for all of them now, and then complete the implementations after the week of Sep 25 classes are done.

Functions to write for A1: Validating Ticket Info

Function name: (Parameter types) -> Return type	Full Description (paraphrase to get a proper docstring description)
`is_valid_seat: (str, int, int) -> bool`	The first parameter represents the ticket. The second parameter represents the number of the first row in the plane. The third parameter represents the number of the last row in the plane. The function should return True if and only if the seat, including the row number and the seat within that row, of this ticket is valid. See above for explanations of ticket validity. You may assume that the **format** of the ticket string is valid: see provided helper function `is_valid_ticket_format`. For example, if the values of the constants are `ROW = 14` and `SEAT = 16`, and the input ticket string is `'20230915YYZYEG12F1236'`, the second argument is `1`, and the third argument is `30`, then the function should return `True`. If, however, the input ticket string is `'20230915YYZYEG32F1236'`, the second argument is `1`, and the third argument is `30`, then the function should return `False`. Similarly, if the input ticket string is `'20230915YYZYEG12H1236'`, the second argument is `1`, and the third argument is `30`, then the function should return `False`.
`is_valid_ffn: (str) -> bool`	The parameter represents the ticket. The function should return True if and only if the frequent flyer number of this ticket is valid. See above for explanations of ticket validity. You may assume that the **format** of the ticket string is valid: see provided helper function `is_valid_ticket_format`.

`is_valid_date: (str) -> bool`	The parameter represents the ticket. The function should return True if and only if the date of this ticket is valid. See above for explanations of ticket validity. You may assume that the **format** of the ticket string is valid: see provided helper function `is_valid_ticket_format`. *Hint*: this is possibly the most challenging function in this assignment. You may want to implement it last.
`is_valid_ticket: (str, int, int) -> bool`	The first parameter represents the ticket. The second parameter represents the number of the first row in the plane. The third parameter represents the number of the last row in the plane. The function should return True if and only if the ticket is in valid format and all of the ticket information on this ticket is valid. See above for explanations of ticket validity. *Hint*: use the provided helper function `is_valid_ticket_format`.

Task 3: Analysing the Information (12 correctness marks)

Advice: Start working on these functions as soon as the assignment is released. You can complete the headers and docstrings for all of them now, and then complete the implementations after the week of Sep 25 classes are done.

Functions to write for A1: Analysing Ticket Info

Function name: (Parameter types) -> Return type	Full Description (paraphrase to get a proper docstring description)
`visits_airport: (str, str) -> bool`	The first parameter represents the ticket. The second parameter represents the airport code. The function should return `True` if and only if this flight either begins or ends in the given airport. You may assume the ticket is valid.
`connecting: (str, str) -> bool`	The parameters represent two tickets. The function should return `True` if and only if the two flights are connecting: the first flight arrives in the same airport as the departure point of the second flight, and the two flights are on the same dates. You may assume the tickets are valid.

`adjacent: (str, str) -> bool`	The parameters represent two tickets. The function should return `True` if and only if the seats on the two tickets are adjacent, i.e. they are next to each other in the same row. Seats that are across an aisle are not considered adjacent. You do not need to check the date, nor the departure/arrival airports in this function. You may assume the tickets are valid.
`behind: (str, str) -> bool`	The parameters represent two tickets. The function should return `True` if and only if the seats on the two tickets are one immediately behind another. You do not need to check the date, nor the departure/arrival airports in this function. You may assume the tickets are valid.
`get_seat_type: (str) -> str`	The parameter represents the ticket. The function should return the type of the seat on the given ticket: `WINDOW`, `MIDDLE`, or `AISLE`. You may assume the ticket is valid.

Task 4: Changing the Information (BONUS 4 correctness marks)

Functions to write for A1: Changing Ticket Info

Function name: (Parameter types) -> Return type	Full Description (paraphrase to get a proper docstring description)
`change_seat: (str, str, str) -> str`	*Note*: this function is not required; completing it will earn you bonus marks. The first parameter represents the ticket. The second parameter represents the row number. The third parameter represents the seat within that row. The function should return a new ticket that is in the same format as the input ticket, has the same departure, arrival, date, and frequent flyer number as the input ticket, and has a new seat information with the given row and seat. You may assume the ticket and the new seat information are valid. For example, if the values of the constants are `ROW = 14` and `SEAT = 16`, and the input ticket string is `'20230915YYZYEG12F1236'`, the second argument is `'24'`, and

	the third argument is `'A'`, then the function should return `'20230915YYZYEG24A1236'`.
`change_date: (str, str, str, str) -> str`	*Note*: this function is not required; completing it will earn you bonus marks. The first parameter represents the ticket. The second parameter represents the day. The third parameter represents the months. The last parameter represents the year. The function should return a new ticket that is in the same format as the input ticket, has the same departure, arrival, seat information, and frequent flyer number as the input ticket, and has a new date. You may assume the ticket and the new date information are valid. For example, if the values of the constants are `YEAR = 0`, `MON = 4`, and `DAY = 6`, and the input ticket string is `'20230915YYZYEG12F1236'`, the second argument is `'20'`, the third argument is `'10'`, and the last argument is `'2024'`, then the function should return `'20241020YYZYEG12F1236'`.

Using Constants

As we discuss in section Constants above, your code should make use of the provided constants. If the value of one of those constants were changed, and your program rerun, your functions should work with those new values.

For example, if `YR` were changed, then your functions should work according to the new number of bonus points that should be earned for guessing a consonant. Using constants in your code means that this happens automatically.

Your docstring examples should reflect the given **values** of the constants in the provided starter code, and **do not need to change**.

No Input or Output

Your `tickets.py` file should contain the starter code, plus the function definitions specified above. `tickets.py` must *not* include any calls to the `print` and `input` functions. Do *not* add any `import` statements. Also, do *not* include any function calls or other code outside of the function definitions.

A1 Checker

We are providing a checker module (`a1_checker.py`) that tests two things:

- whether your code follows the Python style guidelines, and
- whether your functions are named correctly, have the correct number of parameters, and return the correct types.

To run the checker, open `a1_checker.py` and run it. Note: the checker file should be in the **same** directory as your `tickets.py`, as provided in the starter code zip file. Be sure to scroll up to the top and read all messages!

If the checker passes for both style and types:

- Your code follows the style guidelines.
- Your function names, number of parameters, and return types match the assignment specification. **This does not mean that your code works correctly in all situations.** We will run a *different* set of tests on your code once you hand it in, so be sure to thoroughly test your code yourself before submitting.

If the checker fails, carefully read the message provided:

- It may have failed because your code did not follow the style guidelines. Review the error description(s) and fix the code style. Please see the PyTA documentation for more information about errors.
- It may have failed because:
    - you are missing one or more functions,
    - one or more of your functions are misnamed,
    - one or more of your functions have incorrect number or types of parameters, or
    - one of more of your functions return values of types that do not match the assignment specification.

  Read the error messages to identify the problematic functions, review the function specifications in the handout, and fix your code.

Make sure the checker passes before submitting.

Marking

These are the aspects of your work that may be marked for A1:

- **Coding style (20%):**
    - Make sure that you follow Python style guidelines that we have introduced and the Python coding conventions that we have been using throughout the semester. Although we do not provide an exhaustive list of style rules, the checker tests for style are complete, so if your code passes the checker, then it will earn full marks for coding style with one exception: *docstrings will be evaluated separately*. For each occurrence of a PyTA error, one mark (out of 20) deduction will be applied. For example, if a C0301 (line-too-long) error occurs 3 times, then 3 marks will be deducted.
    - All functions, including helper functions, should have complete docstrings including preconditions when you think they are necessary and at least two valid examples.
- **Correctness (80%):**

  Your functions should perform as specified. Correctness, as measured by our tests, will count for the largest single portion of your marks. Once your assignment is submitted, we will run additional tests not provided in the checker. Passing the checker **does not** mean that your code will earn full marks for correctness.

  Note that for full marks **all docstring examples you provide should run without producing errors**.

What to Hand In

The very last thing you do before submitting should be to run the checker program one last time.

Otherwise, you could make a small error in your final changes before submitting that causes your code to receive zero for correctness.

Submit `tickets.py` on MarkUs by following the instructions on the course website. Remember that spelling of filenames, including case, counts: your file must be named exactly as above.