Programming Assignment: Assignment 1: Time Zones

✔ Passed and verified · 42/42 points

ⓘ It looks like this is your first programming assignment. Learn more ✕

Deadline Pass this assignment by April 23, 11:59 PM PDT

Instructions

My submission

Discussions

Preface

Requirements

You need to use Python 3 for this assignment. Python 2 is not suitable.

Required Preparation

In order to complete this assignment, you'll probably need to have completed the second module. However, you can get a good start understanding the assignment even having completed just the first module.

Problem Domain

A *problem domain* is an area of expertise that needs to be understood in order to solve a problem. When programmers are hired, they usually need to learn about their customer's problem domain. For example, if a programmer is hired by a non-profit organization to help with their financial database, that programmer might have to learn how taxes apply to non-profit organizations.

Every assignment in this course will have a problem domain that you will need to learn about.

A1 Problem Domain: Coordinated Universal Time

The problem domain for this assignment involves time zones, and in particular Coordinated Universal Time (UTC), which is "the primary time standard by which the world regulates clocks and time" [Wikipedia]. As you know, there are many different time zones in the world. Wikipedia has a map of the time zones.

As of this writing, there are 40 time zones. One of them, UTC+00:00, is considered to be in the "middle" of the other time zones. All time zones have names, such as UTC+02:00, that indicate the number of hours and minutes they are away from UTC+00:00. For example, the Philippines are in time zone UTC+08:00 because clocks there are set 8 hours later than in time zone UTC+00:00. If it's noon in time zone UTC+00:00, it's 20:00 in time zone UTC+08:00.

Representing hours, minutes, and seconds using a float

In this assignment, we are sometimes going to represent hours and minutes and seconds together as a `float`. 1 hour will be represented as `1.0`, 1 hour and 30 minutes as `1.5`, and so on.

Preconditions

Some of the functions you will write assume that parameter values are in a certain range. The technical term for these restrictions is *precondition*: in order for the function to work, the precondition must be met. **A precondition is a warning to whoever calls the function that the function was designed to work only under those conditions.** When you see a precondition, that means we are guaranteeing that we will only call that function with values that meet the precondition. You can assume that the parameter values meet the preconditions, you do not need to check them. The preconditions are there to make your lives easier!

Floating-point numbers in this handout

This assignment involves `float` calculations, and as you know, these can be inexact. As an example, here is code copied from the Python shell:

```
>>> 7 / 3000
0.0023333333333333335
>>> 7 * (1 / 3000)
0.0023333333333333333
```

Because we leave it up to you to write some expressions, your functions may return values that are very slightly different from the examples in our docstrings. As long as they are very close, your code will be marked as correct; you don't need to make your code match our expected results exactly.

Print statements: don't use them

Nothing in the assignment requires print statements; your code will be marked as incorrect if you use them.

What to do

There are several functions that you will need to implement. We have listed the functions roughly in order of complexity. Each function body will be quite short. You can submit your assignment for feedback once every hour up until the deadline; we recommend that you do this at least once early on in order to make sure you can submit properly.

Step 1: Download the starter code

In this assignment, we are providing *starter code* that contains some function headers and docstrings and one complete function. Download it here:

a1.py

We strongly recommend that you do not proceed until you have opened the starter code in IDLE.

Step 2: Complete the body of function `seconds_difference`

Function name: **(Parameter types) -> Return type**	**Description**

`seconds_difference: (number, number) -> number`	The parameters are times in seconds. Return how many seconds later the second time is than the first. Please note: in `a1.py`, we have provided the completed docstring for this function, including example function calls with the expected return values.

We have written the header and docstring for you in the starter code. You need to write the body of the function. Because this is the first function, here are detailed instructions:

1. If you haven't done Step 1 above, do that now. In IDLE, find `seconds_difference` in `a1.py`.
2. Click in the space below the docstring to write the function body. Make sure that your code is indented to the same level as the docstring!
3. Type a `return` statement: "`return`" followed by the expression that performs the calculation described in the table above. (Remember that the `return` statement is used to give a value back to the caller of the function.)
4. Click `Run -> Run Module`. (This will load the code in your `a1.py` into the shell so that you can call the functions defined in it at the prompt.)
5. Copy and paste this call on the function you just completed: `seconds_difference(3600.0, 1800.0)` . With luck (and skill!), you'll see `-1800.0`. If you see a different number, that means your function is calculating the wrong value. If you get an error, make sure the word `return` is indented exactly the same amount as the docstring.
6. Once you have tried all the example calls and are happy with your function, go to the Assignment 1 page and submit your code. You can do this as soon as your function is doing what you expect. Go to the Assignment 1 page. Select the `My submission` tab and click `Create submission`. Click on `Assignment 1 automarking`, upload your .py file, and click on `Submit`. This will send your `a1.py` to our testing program, and our testing program will call each of the A1 functions and test whether they are producing the correct values.
7. To get your feedback, view your graded submission in the `Your Submissions` area of the `My submission` webpage.

Step 3: Complete the body of function `hours_difference`

Here are helpful hints:

- there are 60 seconds in 1 minute
- there are 60 minutes in 1 hour

Function name: **(Parameter types) -> Return type**	**Description**
`hours_difference:` `(number, number) -> float`	The parameters are times in seconds. Return how many *hours* later the second time is than the first. (Please note: in `a1.py`, we have provided the completed docstring for this function, including example function calls with the expected return values.)

Once you have finished writing the body of `hours_difference`, in IDLE, select `Run -> Run Module`. In the shell, test your function by running the examples from the docstring. If the example calls return the expected results, move on to Step 4. Otherwise, modify your code and repeat the tests.

Step 4: Complete the body of function `to_float_hours`

Function name: **(Parameter types) -> Return type**	**Description**

`to_float_hours:` `(int, int, int) -> float`	The first parameter is a number of hours, the second parameter is a time in minutes (between 0 and 59, inclusive), and the third parameter is a time in seconds (between 0 and 59, inclusive). Return the combined time as a `float` value. (Please note: in `a1.py`, we have provided the completed docstring for this function, including example function calls with the expected return values.)

Once you have finished writing the body of `to_float_hours`, in IDLE, select `Run -> Run Module`. In the shell, test your function by running the examples from the docstring. If the example calls return the expected results, move on to Step 5. Otherwise, modify your code and repeat the tests.

Step 5: Write functions `get_hours`, `get_minutes` and `get_seconds`

Read this section and make sure you understand all of it before you proceed.

We have not provided starter code for these three functions, although we have described them fully in the table below. **Follow the Function Design Recipe** as you develop these functions in `a1.py`.

`get_hours`, `get_minutes` and `get_seconds` are related: they are used to determine the hours part, minutes part and seconds part of a time in seconds.

For example:

```
>>> get_hours(3800)
1
>>> get_minutes(3800)
3
>>> get_seconds(3800)
20
```

In other words, if `3800` seconds have elapsed since midnight, it is currently `01:03:20` (hh:mm:ss).

Here is an overview of how we determined what the example function calls should return:

- There are 60 seconds in 1 minute and 60 minutes in 1 hour, so there are 60 * 60, or 3600, seconds in 1 hour.
- Because there are 3600 seconds in an hour, there is 1 full hour in 3800 seconds. There are 200 seconds remaining.
- Because there are 60 seconds in a minute, there are 3 full minutes in 200 seconds. There are 20 seconds remaining.
- Therefore, 3800 seconds is equivalent to 1 hour, 3 minutes and 20 seconds.

There are several correct ways to write these three function bodies. You may find operators `%` and `//` to be helpful. (If you want another hint, read the body of function `to_24_hour_clock` in the starter code. It has an example of using `%`.)

As an example of the approach you might use, lets assume your program is given a number and you want to work out the "units" portion (remember, the hundred, tens, units thing for decimal numbers?) Let's assume the number is 123.

First, we get rid of the hundreds column:

```
>>> 123 % 100
23
```

You can see that 100 goes in to 123 once and leaves us with 23.

Next, we get rid of the tens.

```
>>> 23 % 10
3
```

The number 10 divides in to 23 twice and leaves us with 3. We have achieved our goal: there are 3 "units".

Function name: **(Parameter types) -> Return type**	**Description**

`get_hours: (int) -> int`	The parameter is a number of seconds since midnight. Return the number of *hours* that have elapsed since midnight, as seen on a 24-hour clock. (You should call `to_24_hour_clock` to convert the number of full hours to a time on a 24-hour clock. This means that the return value should be in the range 0 to 23, inclusive.)
`get_minutes: (int) -> int`	The parameter is a number of seconds since midnight. Return the number of *minutes* that have elapsed since midnight as seen on a clock. (This means that the return value should be in the range 0 to 59, inclusive.)
`get_seconds: (int) -> int`	The parameter is a number of seconds since midnight. Return the number of *seconds* that have elapsed since midnight as seen on a clock. (This means that the return value should be in the range 0 to 59, inclusive.)

Step 6: Complete the bodies of functions `time_to_utc` and `time_from_utc`

Complete functions `time_to_utc` and `time_from_utc`. The header and docstrings are in the starter code. Use those examples to determine the appropriate formula. **We have intentionally left out tests involving time zones that are not on the hour: you need to figure out your own test cases for these.**

Function name: (Parameter types) -> Return type	**Description**

`time_to_utc: (number, float) -> float`	The first parameter is a UTC offset specifying a time zone and the second parameter is a time in that time zone. Return the equivalent UTC+0 time. Be sure to call `to_24_hour_clock` to convert the time to a time on a 24 hour clock before returning.
`time_from_utc:' (number, float) -> float`	The first parameter is a UTC offset specifying a time zone and the second parameter is a time in time zone UTC+0. Return the equivalent time in the time zone specified by the UTC offset. Be sure to call `to_24_hour_clock` to convert the time to a time on a 24 hour clock before returning.

Step 7: Submit your work

Go to the Assignment 1 page. Select the `My submission` tab and click `Create submission`. Click on `Assignment 1 automarking`, upload your .py file, and click on `Submit`. It should be marked within a few minutes.

You can resubmit `a1.py` many times. Your maximum score will count. Notice that this means that you can submit a lot of times before the due date if you start early. As you saw in Step 2, you can even submit before you've finished all of the functions in order to get feedback on what you have done so far.

Fun stuff: a Graphical User Interface!

This will not work correctly until you have finished the rest of the assignment!

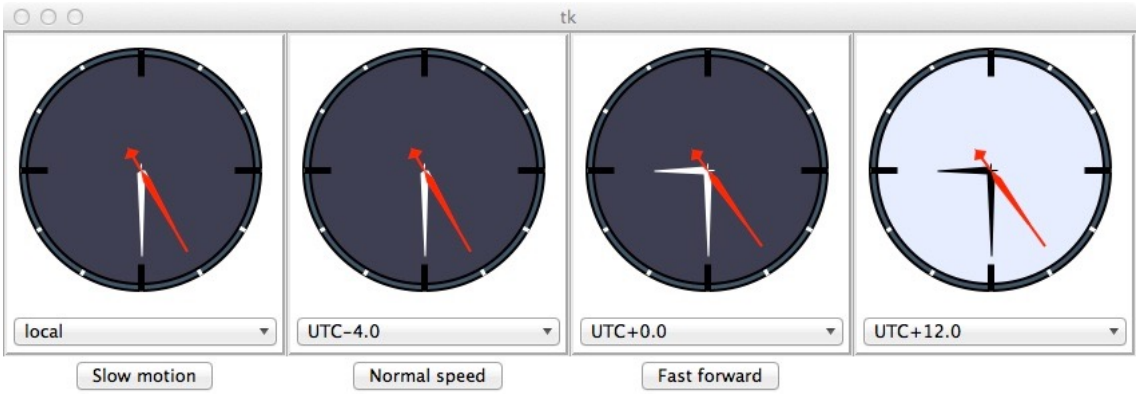
We provide a graphical user interface (GUI) that shows four clocks and allows you to choose time zones to display. Download this file and save it in the same directory as your `a1.py` file:

a1_gui.py

Open `a1_gui.py` in IDLE, select `Run -> Run Module`, and see the clocks in action!

You are not expected to understand the code in `a1_gui.py`*.*

Here's a screenshot; notice that clocks can have a light or dark background depending on whether they indicate a time before or after noon. We've also provided buttons for speeding up and slowing down the clocks in case you want to see what happens around midnight and so on.



How to submit

When you're ready to submit, you can upload files for each part of the assignment on the "My submission" tab.