

Learn to Program: The Fundamentals // Assignment 1

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 watched all the week 2 videos and also done the week 2 exercise. However, you can get a good start understanding the assignment. Understanding what a program should do almost always takes a significant amount of time, possibly as much as an hour or two.

Problem Domains

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 nice [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 / 30000.0023333333333333335>>> 7 * (1 / 3000)0.002333333333333333
```

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. You can download it here: [a1.py](#). In IDLE, select `File -> Open` to open it.

Step 2: Complete the code for function `seconds_difference`

Function name: (Parameter types) -> Return type	Description
<code>seconds_difference:</code> <code>(number, number) -> number</code>	The parameters are times in seconds. Return how many <i>seconds</i> later the second time is than the first. Please note: in <code>a1.py</code> , 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 `seconds_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, go to the [Assignments page](#) and submit your code. Click the Feedback button to view the results.

Step 3: Complete the code for function `hours_difference`

Here is some helpful information:

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

Function name: (Parameter types) -> Return type	Description
<code>hours_difference:</code> <code>(number, number) -> float</code>	The parameters are times in seconds. Return how many <i>hours</i> later the second time is than the first. (Please note: in <code>a1.py</code> , 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 code for function `to_float_hours`

Function name: (Parameter types) -> Return type	Description
<code>to_float_hours:</code> <code>(int, int, int) -> float</code>	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 <code>float</code> value. (Please note: in <code>a1.py</code> , 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`.

The three functions, `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 ways to write these three function bodies. You may find operators `%` and `//` to be helpful.

Function `to_24_hour_clock` in the starter code has an example of using `%`.

As an example of the approach you might use, let's 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 % 10023
```

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

Next, we get rid of the tens.

```
>>> 23 % 103
```

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

Function name:

(Parameter types) -> Return type	Description
<code>get_hours:</code> <code>(int) -> int</code>	The parameter is a number of seconds since midnight. Return the number of <i>hours</i> that have elapsed since midnight, as seen on a 24-hour clock. (You should call <code>to_24_hour_clock</code> 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.)
<code>get_minutes:</code> <code>(int) -> int</code>	The parameter is a number of seconds since midnight. Return the number of <i>minutes</i> 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.)
<code>get_seconds:</code> <code>(int) -> int</code>	The parameter is a number of seconds since midnight. Return the number of <i>seconds</i> 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 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 should make sure you handle those cases.

Function name: (Parameter types) -> Return type	Description
<code>time_to_utc:</code> <code>(number, float) -> float</code>	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 <code>to_24_hour_clock</code> to convert the time to a time on a 24 hour clock before returning.
<code>time_from_utc:</code> <code>(number, float) -> float</code>	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 <code>utc_offset</code> . Be sure to call <code>to_24_hour_clock</code> to convert the time to a time on a 24 hour clock before returning.

Step 7: Submit your work

Go to the Assignments page and click the appropriate Submit button. Choose your completed `a1.py` file. It should be marked within a few minutes. You can view your results by clicking on the View button in the Feedback column.

You can resubmit `a1.py` once every hour. 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. You can even submit before you've finished all of the functions, although the feedback will be incomplete.