Computer Science CS134 (Spring 2021)
Duane Bailey & Molly Feldman
Laboratory 2
Computing the Day of the Week.

Objective. To develop a non-trivial script.

This week we'd like you to write a simple script to print out the current day of the week in Williamstown. Here's what would happen if you ran day.py on a Monday (the $ represents the prompt in your shell):

$ python3 day.py
It's Monday!

An Approach. All computers keep track of time. On Unix machines time is represented by the number of seconds after the end of the 1960s (the "Epoch"). The first moments of Thursday, January 1, 1970 had a time less than 1, the first minute had time values less than 60, and the first hour had times less than 3600. The value that represents the current time is a much bigger number.

In Python, we can access this value—understood to be the current time in Universal Coordinated Time, or UTC, in England—through the time() function, which is found in the time module. In interactive Python, you can use it in the following way:

$ python3
>>> from time import time
>>> time()
1612800680.9091752

The result of calling time is a float value, with the fractional component representing the time after the second. This value is, on most machines, accurate to the millimonth of a second.

In this lab, we'll be using only the integer part of this value, which might be accomplished by the following code:

>>> now = int(time())
>>> now
1612800680

We're taking the current time and storing just its integer part in a variable, now.

If $a$ and $b$ are integers then when we compute $a // b$ the result is always an integer. We can compute the remainder, $r$, of this integer division with the modulo ($\%$) operation:

$r = a \% b$

This is precisely the amount needed to bring $(a/b) \times b$ back up to $a$. When $a$ and $b$ are integers, the result of $a \% b$ is one of the $b$ possible integer remainders: 0, 1, ..., $b - 1$. This is very convenient if we want to take a large randomly selected integer and reduce it one of a very small range of values.

Now let's apply this to our time value. Suppose, for example, we compute the remainder when a time, now, is divided by 60. The result is a value between 0 and 59 representing the number of seconds past the current minute. During the first minute of 1970, this value ranged from 0 to 59. During the second
minute, these remainders again took on the values from 0 to 59. If we subtract this remainder from the current time, the result is an exact multiple of 60: the top of the minute. This suggests that when we divide a time value—using integer division (\(\div\))—the result is the number of minutes since the Epoch.

Continuing the example (1612800680), we see it must have been captured at

\[
\text{>>> secs = now} \% 60 \\
\text{>>> secs} \\
20 \\
\text{>>> mins = now} \div 60 \\
\text{>>> mins} \\
26880011
\]

20 seconds past the minute. With care, you can determine the precise time of day in England. (Using similar logic, can you demonstrate that 26880011 corresponds to 4:11pm in London—which is 11:11am in Williamstown?)

You might now see that it’s possible to compute the number of whole days that have elapsed since the Epoch. The remainder is, of course, the portion of the day that can be accounted for by the hours, minutes, and seconds since midnight. Once we know, for example, that 18666 days have passed, it’s then possible to figure out how many whole weeks have passed (2666), and the remainder (4) is the number of whole days have passed after “the beginning of the week.” Since January 1, 1970 fell on a Thursday, this method treats Thursday as “weekday 0”, Friday as “weekday 1”, etc. Since the remainder is 4, it suggests the weekday associated with now is Monday.

Because we would like to have the logical start of the week (“weekday 0”) happen on Sunday instead of Thursday, we can adjust the time value, measured as the number of seconds from Thursday, January 1, 1970, to be, instead, the number of seconds from Sunday, January 4, 1970. If we make this correction, then the final remainder, as computed above, would be 1 instead of 4. Again, that suggests it’s a Monday (“weekday 1” in this adjusted system).

This Week’s Tasks. Clone the starter repository for this week’s lab into your cs134 directory as you did last week. Open your terminal application, navigate to your cs134 directory, and clone the repository:

\[
\text{cd ~/cs134} \\
\text{git clone https://evolene.cs.williams.edu/cs134-labs/22xyz3/lab02.git ~/cs134/lab02}
\]

Your CS username replaces 22xyz3. The clone command will follow this pattern each week.

We would like you to write several functions that will, eventually, allow us to determine today’s day-of-week. Please write your code as part of a script, day.py, which we have included as part of the starter code in the lab02 directory.

1. Write a function, UTCDay(timeval). This takes an floating point number, timeval, that represents the UTC time in England, and returns the number of the day of the week. As we have discussed, the day is ideally represented by a small integer (0-6), where Sunday is 0, Monday is 1, etc. (If you wish, you may assume that timeval encodes a time on or after Sunday, January 4, 1970.) Here are some examples of how you might call this function interactively:

\footnote{A point-of-view typical in Unix and the US.}
>>> from day import UTCDay
>>> UTCDay(1612800680)  # the UTC date, discussed above
1
>>> UTCDay(345600)  # 00:00:00, Monday, January 5, 1970
1
>>> UTCDay(345599)  # 23:59:59, Sunday, January 4, 1970
0

Here's how you might find the current day of the week in England (UTC+0 time):

>>> from time import time
>>> UTCDay(time())
...

2. Write a function `localDay(timeval, offset)` that calls `UTCDay` to help compute the current day of the week for a timezone that is offset (a floating point value) hours ahead of UTC. In Williamstown, the offset is -5. That means the actual time is 5 hours, or 18000 seconds, earlier than that reported by `time()`. Here are some well-known locales and their UTC offsets:

<table>
<thead>
<tr>
<th>Location</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chatham Island</td>
<td>12.75</td>
</tr>
<tr>
<td>Auckland</td>
<td>12</td>
</tr>
<tr>
<td>Solomon Islands</td>
<td>11</td>
</tr>
<tr>
<td>Vladivostok</td>
<td>10</td>
</tr>
<tr>
<td>Tokyo</td>
<td>9</td>
</tr>
<tr>
<td>Hong Kong</td>
<td>8</td>
</tr>
<tr>
<td>Jakarta</td>
<td>7</td>
</tr>
<tr>
<td>Rangoon</td>
<td>6.5</td>
</tr>
<tr>
<td>Kathmandu</td>
<td>5.75</td>
</tr>
<tr>
<td>Karachi</td>
<td>5</td>
</tr>
<tr>
<td>Abu Dhabi</td>
<td>4</td>
</tr>
<tr>
<td>Nairobi</td>
<td>3</td>
</tr>
<tr>
<td>Cairo</td>
<td>2</td>
</tr>
<tr>
<td>Cape Verde</td>
<td>-1</td>
</tr>
<tr>
<td>South Georgia Island</td>
<td>-2</td>
</tr>
<tr>
<td>Sao Paulo</td>
<td>-3</td>
</tr>
<tr>
<td>Corner Brook</td>
<td>-3.5</td>
</tr>
<tr>
<td>Santiago</td>
<td>-4</td>
</tr>
<tr>
<td>Kalamazzo</td>
<td>-5</td>
</tr>
<tr>
<td>Easter Island</td>
<td>-6</td>
</tr>
<tr>
<td>Phoenix</td>
<td>-7</td>
</tr>
<tr>
<td>San Francisco</td>
<td>-8</td>
</tr>
<tr>
<td>Ancorage</td>
<td>-9</td>
</tr>
<tr>
<td>Honolulu</td>
<td>-10</td>
</tr>
<tr>
<td>Midway Atoll</td>
<td>-11</td>
</tr>
<tr>
<td>Baker Island</td>
<td>-12</td>
</tr>
</tbody>
</table>

Positive offsets are an indication that a new day begins that much earlier in these locations—all east of England—while negative offsets indicate a later end to the day—all to the west.

>>> from day import localDay
>>> localDay(345000, 0)  # 23:50, Sunday, January 4, 1970, in London
0
>>> localDay(345000, +1)  # 00:50, Monday, January 5, 1970, in Paris
1
>>> from time import time
>>> localDay(time(), -5)  # current time in Williamstown
...

This offset could be a float because some timezones involve corrections that are partial hours

3. Write a function, `dayOfWeek(day)`, that takes an integer between 0 and 6 (inclusive) and returns the name of that day as a string:
>>> from day import dayOfWeek
>>> dayOfWeek(1)
'Monday'
>>> dayOfWeek(6)
'Saturday'

4. You can now use your functions to print the current day in Williamstown, when day.py is run as a script:

    if __name__ == "__main__": # run as a script?
        now = time() # UTC time
        dayNumber = localDay(now, -5) # Eastern day of week number
        dayName = dayOfWeek(dayNumber) # get day name
        print("It's " + dayName + "!") # print it out

This should work when run at any time of the day.

5. Thoroughly test your code before you turn it in. In particular, each of the little tests you have seen in this document should generate the indicated answers. You might think about what kinds of mistakes one could make in writing the above functions, and write tests that prove to yourself you didn't introduce those mistakes. This is an important skill we will develop as the semester progresses.

Submit your work. When you are finished with adding your functions to the script day.py, make sure you add and commit your work.

    $ git add day.py
    $ git commit -m "Lab 2 completed"

You can then push your work (remembering to start the VPN if you're working from off campus):

    $ git push

You can, if you wish, check to make sure your work by going to evolene.cs.williams.edu in your browser, logging in using your CS credentials, and inspecting the current state of your lab02 repository. Your changes should be reflected in what you see.

*