NYU Tandon School of Engineering

CS-UY 1114 Fall 2022

Homework 06

Due: 11:59pm, Thursday, November 3rd, 2022

Submission instructions

1. You should submit your homework on **Gradescope**.
2. For this assignment you should turn in 2 separate `.py` files named according to the following pattern: `hw6_q1.py` and `hw6_q2.py`. You do *not* need to submit `touchTypes.py`.
3. Each Python file you submit should contain a header comment block as follows:

```
"""
Author: [Your name here]
Assignment / Part: HW6 - Q1 (depending on the file name)
Date due: 2022-10-20, 11:59pm
I pledge that I have completed this assignment without
collaborating with anyone else, in conformance with the
NYU School of Engineering Policies and Procedures on
Academic Misconduct.
"""
```

No late submissions will be accepted.

REMINDER: *Do not use any Python structures that we have not learned in class.*

The use of `eval()` and `break` are no longer permitted in this class.

For this specific assignment, you may use everything we have learned up to, **and including**, user-defined functions. Please reach out to us if you're at all unsure about any instruction or whether a Python structure is or is not allowed.

Do **not** use, for example, file i/o, exception handling, dictionaries, lists, tuples, and/or object-oriented programming.

Problems

1. **Good Vibrations (`hw6_q1.py`)**
2. **Le Grand Jour (`hw6_q2.py`)**

Problem 1: *Good Vibrations*

Note: For this question, you need to make sure that this file, **touchTypes.py** (linked here), exists in the same directory as your `hw6_q1.py` file. Just like with the `arpeggiator` module, you don't need to worry about how it works internally. The only thing you need to do is include the following lines at the top of your `hw6_q1.py` file, and everything should work:

```
from touchTypes import TouchType, SwipeDirection

SINGLE_TOUCH = TouchType.SINGLE_TOUCH
DOUBLE_TAP = TouchType.DOUBLE_TAP
SWIPE = TouchType.SWIPE
HOLD = TouchType.HOLD
UP = SwipeDirection.UP
DOWN = SwipeDirection.DOWN
LEFT = SwipeDirection.LEFT
RIGHT = SwipeDirection.RIGHT
NO_DIRECTION = SwipeDirection.NO_DIR
```

These nine constants will be used throughout this problem.

Depending on the way a user interacts with a smartphone screen (i.e. its direction, strength, and duration), the smartphone will give different types of **haptic feedback**.

Assume that there are four types of touches:

- **Single touch**: Represented by the `SINGLE` constant in our file.
- **Double tap**: Represented by the `DOUBLE` constant in our file.
- **Swipe**: Represented by the `SWIPE` constant in our file.
- **Hold**: Represented by the `HOLD` constant in our file.

If the user swipes, we can represent each cardinal direction using the constants `UP`, `DOWN`, `LEFT`, and `DOWN`. If the user **didn't** swipe, this direction automatically defaults to the constant `NO_DIRECTION`.

Duration and strength only matter in the case of a **hold**. In all other touch types, you can assume that the duration and strength will always both be `0.1`. Duration can be any positive integer, and strength can be any float from `0.0` (not inclusive) and `1.0` (inclusive). Duration and strength only matter in the case of a **hold**. In all other touch types, you can assume that the duration and strength will always both be `0.1`.

We will do this by creating three functions: `give_haptic_feedback()`, `register_touch()`, and our "driver" `get_touch()` function.

`give_haptic_feedback()` will accept one positive numerical parameter called `touch_ratio`. Don't worry about what this `touch_ratio` value is or means just yet. Just know that:

- If the touch ratio is anywhere between `0.0` and `0.5` (non-inclusive on both ends), our function will print the message `"Vibrating once..."`.
- If the touch ratio is anywhere between `0.5` and `2.0` (inclusive on both ends), our function will print the message `"Vibrating twice..."`.
- If the touch ratio is anything higher than `2.0`, our function will print the message `"Vibrating thrice..."`.

`register_touch()` will accept four parameters: `touch_type`, `direction`, `duration`, `strength`.

- `touch_type` is a `TouchType` value that is represented by four of the nine constants described above (i.e. `SINGLE`, etc.).
- `direction` is a `SwipeDirection` value represented by the other five constants described above (i.e. `UP`, etc.). Remember that these constants are already included in your file, so *you don't need to define them*.
- You can assume that `duration` and `strength` will always be numerical values.

With these parameters, `register_touch()` will first check for the type of touch:

- If `touch_type` is equal to `SINGLE`, simply print the message `"Registering single touch..."`.
- If `touch_type` is equal to `DOUBLE`, simply print the message `"Registering double tap..."`.
- If `touch_type` is equal to `SWIPE`, print the message `"Registering single touch..."` and:
    - Print `"Exiting app..."` if `direction` is equal to `UP`.
    - Print `"Changing page..."` if `direction` is equal to `LEFT` or equal to `RIGHT`.
    - Print `"Scrolling up..."` if `direction` is equal to `DOWN`. You can assume that if `touch_type` is not equal to `SWIPE`, `direction` will be equal to `NO_DIRECTION`. You don't need to do anything else in this case.
- If `touch_type` is equal to `HOLD`, print the message `"Registering hold..."`, calculate the touch ratio (`strength / duration`), and invoke `give_haptic_feedback()` using the touch ratio as an argument.

The function that will drive the whole program will be called `get_touch()`. This function should:

- **Always** ask what the touch type and...
- How strong it was.
- It should only ask for direction **if the touch type was *swipe***, and
- Only ask for touch duration **if the touch type was *hold***. Duration and strength only matter in the case of a **hold**. In all other touch types, you can assume that the duration and strength will always both be `0.1`.

You can assume that the user will always enter valid strings and numbers for these four values.

Here are a few sample executions. Your format MUST look the same, so consider copying-and-pasting:

```python
def main():
  get_touch()

main()
```

Running this script could result in any of the following executions:

```
What type of touch did the user perform? [single/double/swipe/hold] single
How strong was the user's touch? [0.0 to 1.0] 0.5
Registering single touch...
```

```
What type of touch did the user perform? [single/double/swipe/hold] double
How strong was the user's touch? [0.0 to 1.0] 0.3
Registering double tap...
```

```
What type of touch did the user perform? [single/double/swipe/hold] swipe
In what direction did the user swipe? up
How strong was the user's touch? [0.0 to 1.0] 0.7
Registering swipe...
Exiting app...
```

```
What type of touch did the user perform? [single/double/swipe/hold] hold
For how long did the user hold the touch? 1
How strong was the user's touch? [0.0 to 1.0] 0.25
Registering hold...
Vibrating once...
```

```
What type of touch did the user perform? [single/double/swipe/hold] hold
For how long did the user hold the touch? 2
How strong was the user's touch? [0.0 to 1.0] 1
Registering hold...
Vibrating twice...
```

Oh, and, please re-read submission instruction number 2.

Problem 2: *Le Grand Jour*

> In metric, one milliliter of water occupies one cubic centimeter, weighs one gram, and requires one calorie of energy to heat up by one degree centigrade—which is 1 percent of the difference between its freezing point and its boiling point. An amount of hydrogen weighing the same amount has exactly one mole of atoms in it.
>
> Whereas in the American system, the answer to *"How much energy does it take to boil a room-temperature gallon of water?"* is *"Go f*** yourself,"* because you can't directly relate any of those quantities.

— Josh Bazell

Note: The format of the output in this problem must *perfectly match* the examples'. Consider copying-and-pasting.

Background

The metric system was developed in the 1790s as part of the reforms introduced during the **French Revolution**, which provided an opportunity for the French to reform their inconsistent, unwieldy, and archaic system of many local weights and measures. It is now used as the official system of measurement in all but **three countries** around the world, either fully or to some extent.

While metric weights and lengths were readily adopted by the rest of the world and continue to be used, the French also introduced the concept of ***decimal time and calendarization*** into their new government, but were both abolished at the end of the revolution.

French revolutionary dates and times functioned as follows:

- There were **twelve months**, each divided into **three ten-day weeks** called *décades*.
    - For this problem you can assume that the Gregorian month will always have 30 days.
- Each day in the was divided into 10 hours.
- Each hour was divided into 100 minutes.
- Each minute was divided into 100 seconds (for this problem, you can assume that 1 decimal second is the same length as a regular second).

While making programs dealing with times and dates is **notoriously difficult**, we will create a simplified date-and-time converter that will take a conventional date and time (say, today and right now) and will convert it into its French revolutionary date-time equivalent.

Part 1: Converting to decimal time

Write a function called `get_decimal_time()` that will accept three integer parameters, each representing a conventional hour, minute, and second, respectively. You can assume that this function will always receive positive arguments during invocation.

It will then use this information to determine its decimal equivalent, which it will return in a `"HOUR:MIN:SEC"` format.

Recall that French revolutionary days each have 10 hours, each with 100 minutes, each with 100 seconds. For example:

```python
decimal_time = get_decimal_time(16, 7, 46)  # i.e. roughly 4:07pm in military time
print(decimal_time)

decimal_time = get_decimal_time(7, 47, 2)  # i.e. roughly 7:47am
print(decimal_time)
```

Output:

```
5:80:66
2:80:22
```

Hint: `//` and `%`.

Part 2: Converting to revolutionary dates

Write a function called `get_decimal_date()` that will accept three integer parameters, each representing a Gregorian month number (i.e. `1` through `12`), a date of the month (assume `1` through `30`), and a **Common Era** year, respectively.

Your function will then use this information to convert this date to its French revolutionary equivalent, and return it as a string of `"[Day] [month] [year], Décade [décade]"`.

The French revolutionary months are roughly equivalent to the following:

Gregorian	French Revolutionary
January	Nivôse
February	Pluviôse
March	Ventôse
April	Germinal
May	Floréal
June	Prairial
July	Messidor
August	Thermidor
September	Fructidor
October	Vendémiaire
November	Brumaire
December	Frimaire

Figure 3: Gregorian approximations of French revolutionary months.

Since months in this system have only three 10-day weeks, you can easily figure out the *décade* by checking in which of the weeks the current date is.

Finally, the revolutionary year is the **difference between the Gregorian year and `1792`**, the year the calendar was implemented.

For example:

```python
revolutionary_date = get_decimal_date(3, 22, 2022)  # i.e. March 22nd, 2022

print(revolutionary_date)
```

Output:

```
22 Ventôse Year 230, Décade 3
```

Note: Since we're making a ton of assumptions to make the math easier, your program won't give you the exact equivalent date, which is totally fine. You can go **here** for exact equivalents).

Part 3: Putting it all together

Your last function will be called `get_french_datetime()`, and it will accept a single **string** parameter containing a Gregorian date and time of the following format:

```
"HR:MIN:SEC MONTH/DAY/YEAR"
```

Your function must then isolate each piece of information from this string, and pass the relevant information to `get_decimal_time()` and `get_decimal_date()` to get their respective decimal equivalent.

Your function must return a string with two lines: the first giving you the decimal time, and the second giving you the decimal date.

For example:

```
gregorian_datetime = "16:07:46 03/22/2022"
french_datetime = get_french_datetime(gregorian_datetime)

print(french_datetime)
```

Output:

```
5:80:66
22 Ventôse Year 230, Décade 3
```

Note that you **may not assume that the location of the `:` characters will always be the same**. For example, the following two strings are also valid input:

```
"02:50:20 02/12/2022"

"2:50:20 2/12/2022"
```

Hint: **`find()`** and **`int()`**.

There's no need for you to write a `main()` function here, but if you would like one so that you can test your code, here's a simple one:

```
def main():
    gregorian_datetime = "16:07:46 03/22/2022"
    french_datetime = get_french_datetime(gregorian_datetime)
    print(french_datetime)
```