Simon Socolow and Rafay Syed

SweatScript Specification

Introduction
SweatScript is one of the few fully user-controlled transparent health data collection software. SweatScript empowers people to track important health metrics to better understand their fitness levels, hydration, and sleep. For the most part, people keep track of important metrics—like water consumption and sleep patterns—in their heads which is not a reliable way of tracking data. SweatScript quantifies important metrics in an easily digestible way through graphs. Graphs that span days, weeks, or even months allow users to extrapolate important trends that can help readjust how they go about everyday life.

This problem should have its own programming language because this would allow users more control over their data entry. In doing so, users are forced to be more honest about these important health metrics as any false input would alter the graphs and the user would have a possible warped perception of their health. A platform-independent language could allow users to not be restricted to certain proprietary ecosystems like Apple or Garmin to track their health. Also, without having to own a physical health tracking device to use this software, it is more accessible.

Design Principles
SweatScript is intended to be easily accessible and easy to use. As this language is designed to be used throughout the day by simple entry of commands as health events occur, the speed and simplicity of the language is of high importance. If the language is difficult to use, people won't use it. If it is slow to use, this will waste people's time as users are already investing a small amount of time to input their health information.

Therefore, our language often deals in abbreviations and simplified commands. For example, when entering the date, the user would enter a number. For example, December 3, 2023 would be inputted as 12032023. Although not a visually pleasing date, when entering the date through their phone, the user doesn't want to switch between keyboards to find a hyphen or forward slash. Thus, inputting 12032023 is much easier to the user than entering 12/03/2023.

Examples
A program can consist of one or more days. Examples can be found in example-1.ss, example-2.ss, and example-3.ss in the code directory. Here are shortened versions of those programs.

```
date 04112023 up 0700 h2o 37 h2o 46 h2o 48 sleep 2330
date 04122024 up 0800 run 34 mins avghr 46 berg 37 mins avghr 135
    h2o 36 squash 37 mins sleep 2320

date 04052023 up 0758 h2o 18 squash 20 mins h2o 15 sleep 2200
date 04062023 up 0809 h2o 12 h2o 11 berg 60 mins avghr 130 h2o 18 sleep 2300

date 12052023 up 0730 h2o 10 berg 60 mins h2o 30 h2o 20 sleep 2330
date 12062023 up 0800 h2o 20 erg 45 mins avghr 145 h2o 30 h2o 20 sleep 2345
date 12072023 up 0830 h2o 20 h2o 30 run 20 mins h2o 10 h2o 20 sleep 2330
```

Each example program (here assuming source code in 'tracker.ss' and output file name 'sweatscript.html') should produce an HTML file containing graphs that is in the same directory. The HTML file can then be viewed in a web browser, where three graphs should be shown. Programs can be executed with the command (assuming plotly package is installed):

```
dotnet run tracker.ss > sweatscript.html
```

Plotly can be installed with

```
dotnet add package Plotly.NET --version 4.2.0
```

Language Concepts
For SweatScript, the key primitives will be the date, duration, activity, and an activity modifier. Date as a primitive is very important in marking the current day the user will be tracking their date. Duration will be used for two instances. One refers to water consumption where duration will mark the amount of water the user has filled up in their water bottle. The consumption will be measured in ounces. The other instance will be used for the duration of an activity, allowing the program to chart how active the user is during the day. Activity will be used to indicate to the program what certain task the user is performing (waking up, filling up water, going on a run, etc..). An activity modifier will be used to provide additional information about an activity. In this case, the activity modifier will be the average heart rate of a specific heart rate.

The combining forms will combine the duration or activity modifier with the specific activity. By combining the activity with the duration/modifier we are affixing quantitative values to these specific activities, so users can extrapolate trends from the then produced graphs.

Furthermore a singular space needs to be placed between each activity in order to be parsed and processed precisely. For example, one could not write

```
run   3 mins
```

Instead they would have to input

```
run 3 mins
```

Formal Syntax
Here is the BNF form of our language:

```
<exp> ::= <day> <up> <activity> <sleep> | <exp>\n<exp>
<day> ::= date <month><day><year>
<month> ::= 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12
<day> ::= 01| 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12
    | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
    26 | 27 | 28 | 29 | 30 | 31
<year> ::= <digit><digit><digit><digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<activity> ::= <activity> <activity> |
    <h2o> | <measured activity description>
<measured activity description> ::= <measured activity>
    <activity modifier>
<activity modifier> ::= <activity modifier> <activity modifier> |
    <duration> | <avghr>
<measured activity>  ::= run | erg | berg | squash
<quantity> ::= <digit> | <quantity> <quantity>
<time> ::= <quantity>
<duration> ::= <quantity> mins
<avghr> ::= avghr <quantity>
<sleep> ::= sleep <time>
<up> ::= up <time>
<h2o> ::= h2o <quantity>
```

Semantics
type Date is a string and represents the date of the current day in question.
type Time is represented as an int and will be used to represent the time an user wakes up and goes to sleep.
type Up is represented as type Time and represents the time a user wakes up.
type Sleep is represented as type Time and represents the time a user goes to sleep.

type H2o is an int that represents the amount of water (in ounces) a user fills up.
type H2oList is a list of type H2o's which represents the number of instances a user has filled up their water through the day. H2oList tracks the differing amounts of water the user fills up through the day.
type ActivityModifiers is a record of the duration of an activity (duration) and the average heart rate (avgHR) of the activity in question. The duration is an int and the average heart rate is also an int. If an activity, like h2o only uses the duration (which in this case represents ounces and not minutes), the average heart rate is set to -1. AvgHR is also set to -1 if the user doesn't input their avgHR for the exercise activity they just completed.
type Activity is a record of the name of the activity and the activity modifier. The name is a string which can be "run", "squash", "berg", "erg" or "h2o". The modifier is of type ActivityModifiers.
type Day is a record of the date, wakeTime, bedTime, and the activities. date is of type Date. wakeTime and bedtime are ints. activities is an Activity List.
type History is a list of Day's. History keeps track of all the days the user inputs through the week/month.

Remaining Work

One further enhancement we would like to see is to implement more activities. Right now we only have 4 exercises. With the way our parser is constructed, it should be easy to expand to other cardio activities like swimming or jump rope. Furthermore we would like to implement more activity modifiers that are more activity-specific. For example, for erging, one additional modifier could be split time or distance.

One feature that we did not implement was using the h2o type to represent the time the user filled up their water rather than the amount of water they filled up. We made this switch because it made more logical sense in our language to track the cumulative ounces of water a user drinks per day so they can see how how their water consumption changes on a macro level. They can compare the valleys and peaks of their hydration across a week or even a month. Ideally in the future we can implement our initial plan and allow the user to have a more micro perspective and see how their water consumption changes throughout the day. IE see how their water use changes between the morning and the night.