SweatScript

Introduction

The problem that our language helps to solve is the lack of fully user-controlled transparent health data collection software. Our language will empower people to track health events throughout time to better understand their fitness levels, hydration, and sleep. This language would quantify important metrics that are not usually tracked by average people and display those metrics in an easily digestible way through graphs. Graphs of metrics over time spans of weeks or months would 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 granular control over their data entry and would make the process of the transformation of their data into graphs more transparent than something like buttons on an app. 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. Finally, an Apple watch or Garmin watch might not be actually accurate, so having a programming language would help make accurate data entry easier.

Design Principles

Ideas that will guide our design center is around ease of 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 and syntax errors could stop them from visualizing their health data. If it is slow to use, this will waste people's time as users are already investing a small amount of time to use the language. So, we want our commands to be easy to remember and fast to enter, either by a keyboard or voice recognition software.

Examples

```
date 04052023 up 0758 h2o 0918 squash 20 mins h2o 1500 sleep 2200

date 07032023 up 0809 h2o 0912 h2o 1100 berg 60 mins avghr 130
    h2o 1800 sleep 2300

date 03092023 up 0509 run 72 mins h2o 0812 h2o 1000 erg 60 mins
    avghr 130 h2o 1800 sleep 2300

date 04112023 up 700 h2o 1200 sleep 2330

date 11122023 h20 0812 h20 1024 h20 1354 h20 1620 h20 2124 h20 2300

date 11162023 h2o 0700 h2o 0800 h2o 1000 h2o 1030 h2o 1100 h2o 1500 h2o 2100
```

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. Programs can be executed with the command:

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

Language Concepts

For our program, the key primitives will be the date, duration, time, activity, and activity modifiers. Date as a primitive is very important in marking the current day the user will be tracking their date. Time will be used to mark the current time for which the user has filled their water, thereby allowing the program itself to track how often the user fills up their water during the day. Duration 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..). Activity modifiers will be used to provide additional information about the activities such as average heart rate.

The combining forms will combine the duration or time or activity modifier with the specific activity. By combining the activity with the time/duration/date/modifier we are affixing meanings to these quantitative records allowing for certain tracking so users can extrapolate trends from the graphs that we will then produce.

Syntax

Here is the BNF form of our language:

```
<exp> ::= <day> <activity> | <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> | <up> | <sleep> |
    <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 <time>
```

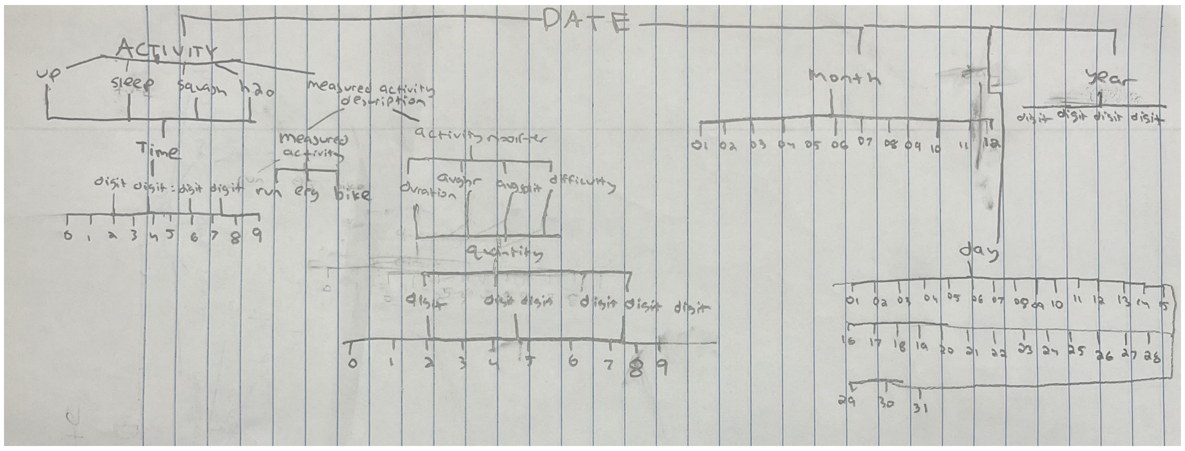
Semantics

i)

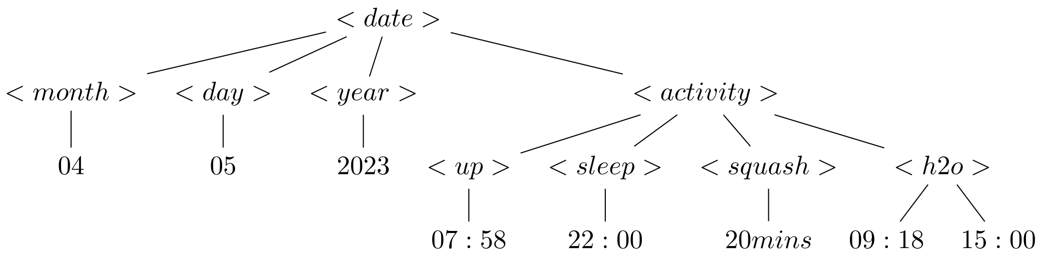
Currently supported data types: date, time, activity, and activity modifier. The date defines the day that is the parent of all the activities which have occurred throughout that day. Activities are events during the day like: up and sleep for sleep tracking, h2o for hydration tracking, and run, erg, berg, squash for cardio tracking. Time uses the 24-hour time format that represents when an activity occurred. Activity modifiers are optional attributes of the activity they modify, such as duration and avghr (average heart rate). For example, a time primitive would be "1438", a date primitive would be "04222023", an activity primitive would be "run", and an activity modifier would be "avghr 140" or "80 mins".

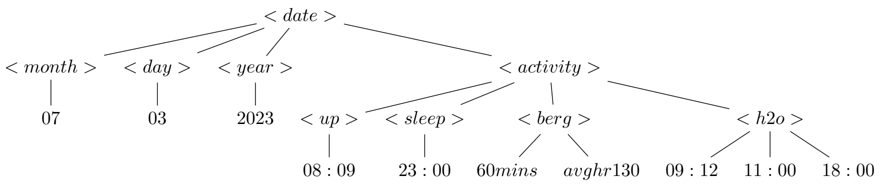
ii)

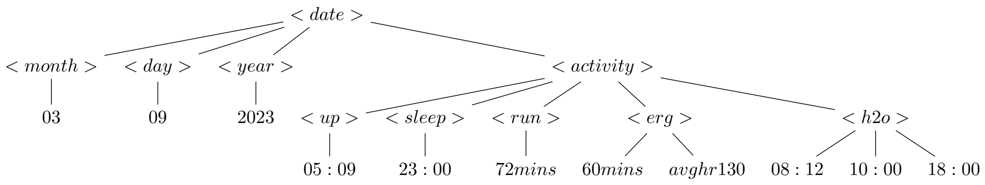
The actions of our language are: combining quantitative properties to a certain activity and groups of activities to certain days, and using the formed data structure to produce graphs representing hydration, activity, and sleep for each day. For example, our activity could be "erg". We could then combine the duration and the average heart rate. So our combined form could look like "erg 60 mins avghr 145." We do this with key word constants that are parsed and the properties are affixed to that activity in question.

iii) 

iv)

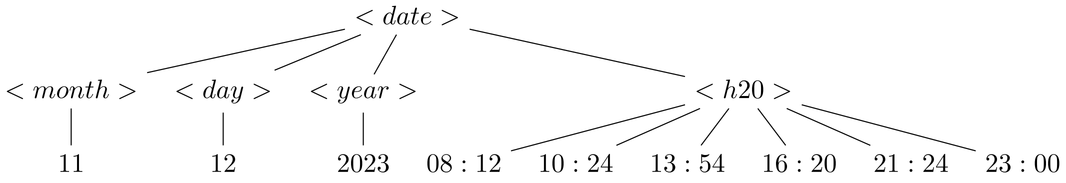






v)

Programs in our language read in input from the user provided they provide the correct keyword usage that will allow us to create the proper AST's for the program.
The output for evaluating a program would be a graphical representation of the input the user provides. For example, if the user provides certain times in which they fill up their water bottle, our program could then output a line graph where each point is a time in the day they filled up their water bottle.



There is no precedence or associativity

Remaining Work

We have not yet implemented up, sleep, and activities besides h2o. Also, in our minimal working version we produced one svg that showed hydration over the course of a single day. In the future, we hope to make 3 bar graphs (with dates on the x-axis) showing: hydration, sleep, exercise all in one html document, using a framework called plotly. We could also include an svg of the hydration over the most recent date.