Document for Y2 project: Nutrients and Calories Calculator

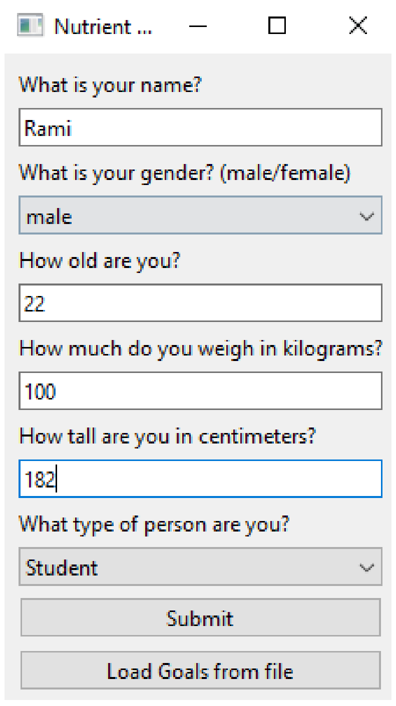
Rami Lahtinen, 916929, Bioinformaatioteknologia, 26.4.2024

1. General Description

This program allows the user to monitor their nutrients and energy intake by. It first evaluates the user by asking them a short series of questions and calculating a nutrient goal of sorts for the user. Then the user will be taken into a monitoring window showing them the goal both as text and graphically. The user can then input eaten foods which then update the display. I have completed the project on the ‘medium’ level as planned: in addition to all the ‘easy’ level requirements, the program has a working GUI and tests for some parts of the program.

2. User instructions

The program is started from the “app.py” file (formerly “main.py”) by pressing run from any IDE. The user will then be asked a series of questions and after which they are encouraged to press a button titled “Submit”. Alternatively the user may load their goals from a text file which also leads them to the monitoring window. In the monitoring window, the user is able to log foods and export their goals to a text file if they so choose.



What is your name?

Rami

What is your gender? (male/female)

male

How old are you?

22

How much do you weigh in kilograms?

100

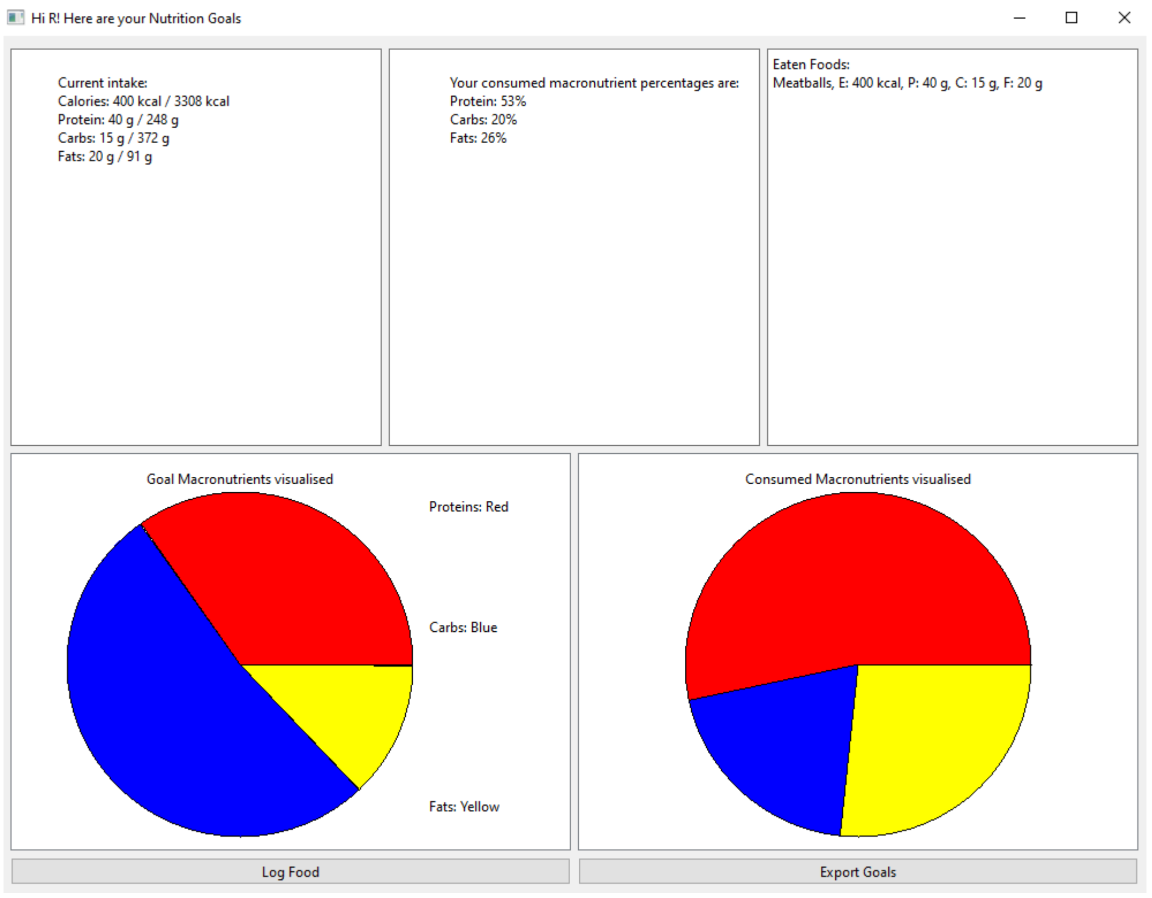
How tall are you in centimeters?

182

What type of person are you?

Student

If the user presses “Load Goals from file”, they are prompted for their username and and a file path to the text files containing their goals.



Hi R! Here are your Nutrition Goals

Current intake:
Calories: 400 kcal / 3308 kcal
Protein: 40 g / 248 g
Carbs: 15 g / 372 g
Fats: 20 g / 91 g

Your consumed macronutrient percentages are:
Protein: 53%
Carbs: 20%
Fats: 26%

Eaten Foods:
Meatballs, E: 400 kcal, P: 40 g, C: 15 g, F: 20 g

Goal Macronutrients visualised
Proteins: Red
Carbs: Blue
Fats: Yellow
Consumed Macronutrients visualised
Log Food
Export Goals

Note: before inputting the first food item, the text in the center widget will instead show `"No data yet, input something!")` and the right pie chart will not show.

3. External libraries

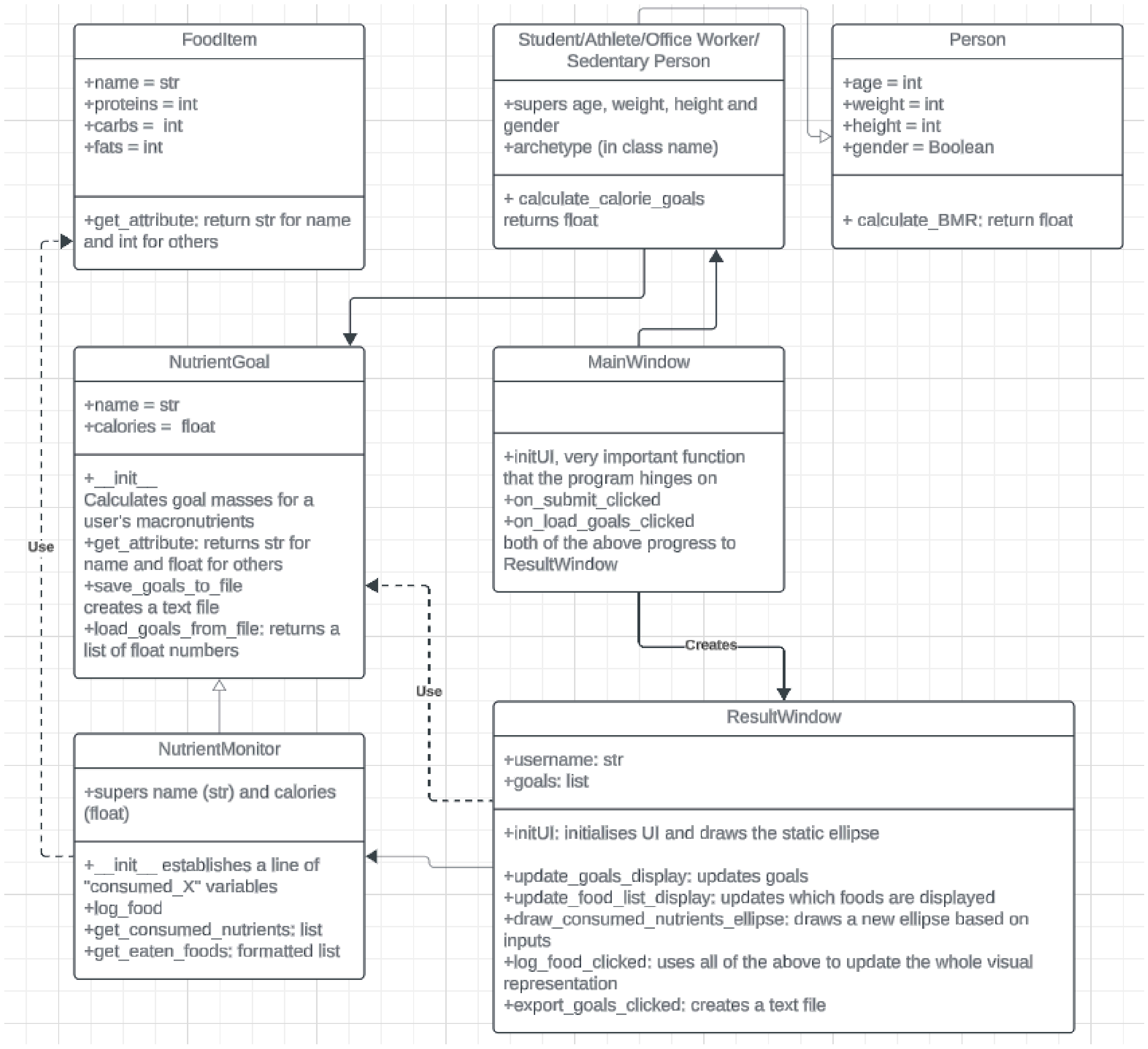
In this program, in addition to Unittest, only PyQt6 and its sub-libraries are used. Most notable of these are QtCore, QtWidgets and maybe QtGui.

4. Structure

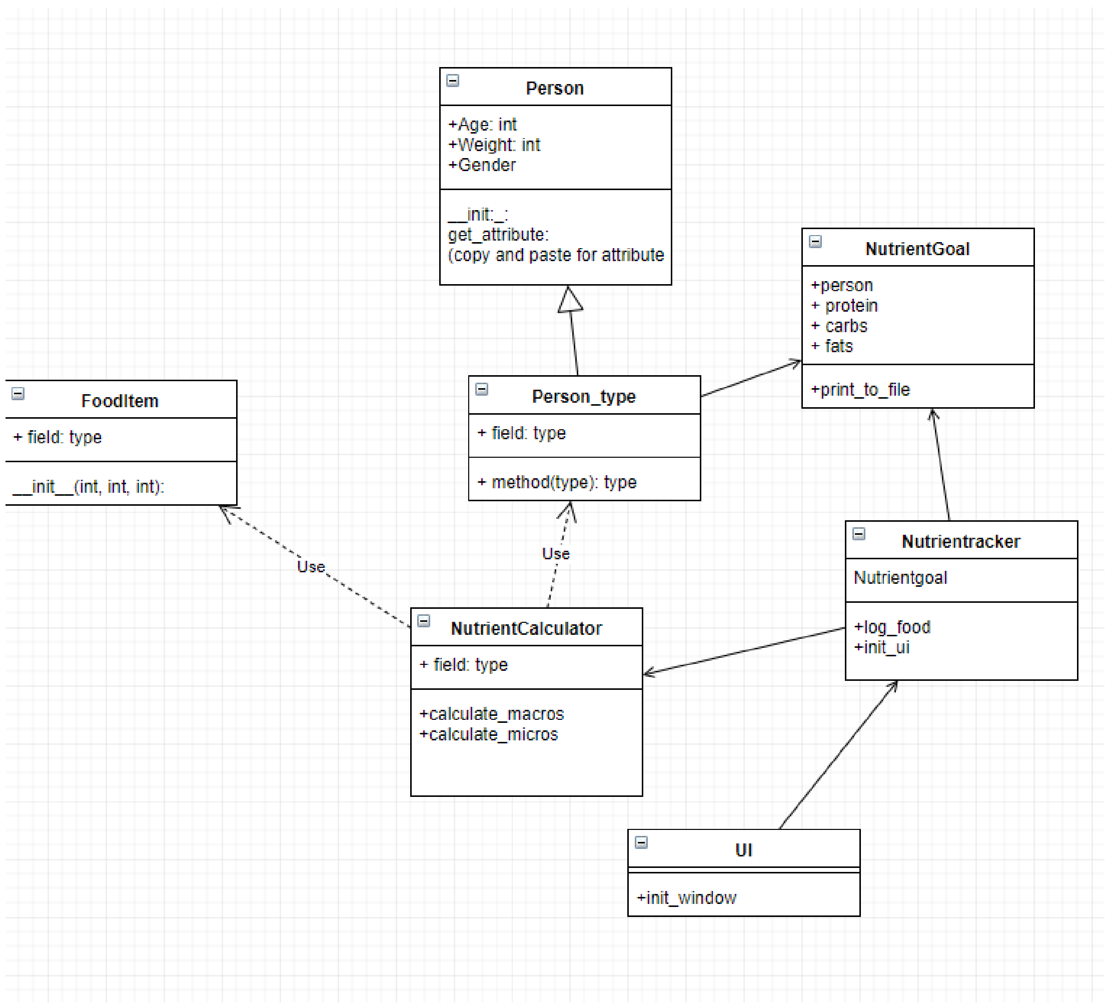
There are many integral parts to this seemingly simple program and they can roughly be divided into front-end and back-end files. The front-end and back-end consist of the elements that are presented to the user and the information processing and calculations respectively. The first problem is to gather information from the user that the rest of the program can then utilise for its calculations. This would be the MainWindow class: it collects substantial information from the user which it passes on to the calculating part of the program. This is where the classes NutrientGoal and Person (with subclasses representing different person archetypes) enter the picture. They calculate both energy and macronutrient goals for the user. Important methods for these operations are calculate_bmr in Person and __init__ in NutrientGoal. The latter both initialises and calculates macronutrient goals for the user based on the BMR calculations. This information is then stored within the NutrientGoal class and passed onto the next part of the program.

When these operations are complete, the program moves to heavily rely upon the ResultWindow class that houses many important functions. Firstly, initUI creates a window where all the other elements are housed. The pie charts are drawn using GraphicsEllipseItem. The pie chart on the left is static and drawn with the user's goal data, while the right pie chart is dynamic and updates according to the user's food inputs. Very important methods inside the class are ones that update the display in some way whether it being a text or a graphical element. These methods are update_goals_display, update_food_list_display and draw_consumed_nutrients ellipse and they complete the functions that their name names imply. The ResultWindow class is by far the largest file in the repository because it houses so many integral functions in order for the GUI to work.

Below is a class diagram for the program:


FoodItem
+name = str
+proteins = int
+carbs = int
+fats = int
+get_attribute: return str for name and int for others
Student/Athlete/Office Worker/ Sedentary Person
+supers age, weight, height and gender
+archetype (in class name)
+ calculate_calorie_goals returns float
Person
+age = int
+weight = int
+height = int
+gender = Boolean
+ calculate_BMR: return float
NutrientGoal
+name = str
+calories = float
+__init__
Calculates goal masses for a user's macronutrients
+get_attribute: returns str for name and float for others
+save_goals_to_file
creates a text file
+load_goals_from_file: returns a list of float numbers
MainWindow
+initUI, very important function that the program hinges on
+on_submit_clicked
+on_load_goals_clicked
both of the above progress to ResultWindow
Use
Use
Creates
NutrientMonitor
+supers name (str) and calories (float)
+__init__ establishes a line of "consumed_X" variables
+log_food
+get_consumed_nutrients: list
+get_eaten_foods: formatted list
ResultWindow
+username: str
+goals: list
+initUI: initialises UI and draws the static ellipse
+update_goals_display: updates goals
+update_food_list_display: updates which foods are displayed
+draw_consumed_nutrients_ellipse: draws a new ellipse based on inputs
+log_food_clicked: uses all of the above to update the whole visual representation
+export_goals_clicked: creates a text file

For reference, the plan-stage diagram is presented below:



The class structure has been altered somewhat from the project plan. The reason for this is necessity. The proposed NutrientCalculator and NutrientTracker have been combined into the NutrientMonitor class and the NutrientGoal taking some of the intended functionality as well. Meanwhile the vague "UI" class has formed into the MainWindow and ResultWindow classes. This time I needed the GUI to include two separate classes with different functionalities, so two classes were written.

5. Algorithms

The energy need for the user is calculated based on the information they input into program using the formula from the Institute of Medicine are as follows:

Males:

$E = 662 - (9,53 * A) + PA * ((15,91 * W) + (539,6 * H))$

Females:

$E = 354 - (6,91 * A) + PA * ((9,36 * W) + (726 * H))$

These values are first found by the selected Archetype Class (student, athlete etc.) and are then used in NutrientGoal to calculate goal mass intakes for each of the macronutrients. The energy of proteins is found by multiplying the above formula's solution by 0.3, the energy of carbohydrates by using 0.45 and the energy of fats by 0.25. Then the masses of each macronutrients are found. Because proteins and carbohydrates contain 4 kcal/g and fats contain 9 kcal/g, the masses are found as follows:

$$m(prot)(g) \; = \; E\,kcal/g \; * \; 0.3/4\,kcal/g$$
$$m(carb)(g) \; = \; E\,kcal/g \; * \; 0.45/4\,kcal/g$$
$$m(fats)(g) \; = \; E\,kcal/g \; * \; 0.25/9\,kcal/g$$

There are also calculations associated with drawing both ellipses. The first ellipse is static and is drawn by given parameters. First the macronutrients are summed together to create a variable representing a 'total' of the nutrients. Then portions of the ellipse are calculated by dividing all the macronutrients by the total sum (the individual macronutrient needs are calculated above). The angle starts at 0 and each of the section angles the following formula holds true.

$\beta_1 = \alpha * 16 + \beta_0 * 16$, where α is the current start angle and β_0 is the section angle for the current macronutrient. Both of the variables are multiplied by 16 because PyQt6 operates in sixteenth degrees. After this operation is complete, the new start angle is calculated by $\alpha_1 = \alpha_0 + \beta_1$, where α_0 is the previous start angle and β_1 the calculated section of the pie chart. After these are complete, the ellipse is drawn. The dynamic pie chart is drawn in exactly the same manner, only that the angles are obtained from the consumed nutrients rather than goal nutrients. The previous ellipse is first removed from the scene if it exists. This way of drawing the pie charts without using forbidden libraries is probably the most sensible, since it was graciously tipped to me by my project assistant. I also briefly explored other ways of drawing the pie charts (other GraphicsItem subclasses) but they didn't seem anywhere as compelling as the recommended path. Of course matplotlib would have been by far the most efficient solution with dozens of lines of code being saved, but I fully understand that it is forbidden for learning's sake.

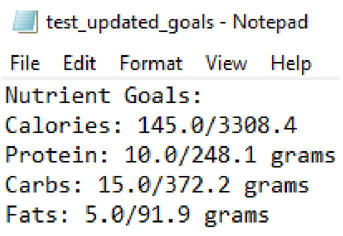
6. Data Structures

In the project there are of course many different data structures that are instrumental to the program being functional. The most prominent are lists. The NutrientGoal class returns a list of goals that is then used very many times later in the program, especially when drawing the ellipses. That part utilises one float number at a time to calculate the portions of the ellipse, which is perfect for that situation.

Then there are, of course, many primitive data structures such as integers, floats, or strings. There is also a dictionary in the later parts of the program, when assigning correct macronutrients to their proportionate values. This is especially useful when recalling information about the macronutrient in question. There are also strings, of course, spread throughout the program. They are mainly used in labelling objects, such as making clear which colour represents which macronutrient. I have used both immutable (e.g. strings) and mutable (e.g. lists) data structures in my project.

7. Files and formats

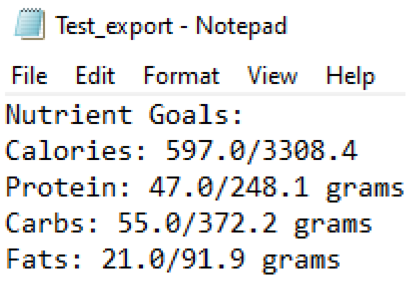
My program handles exclusively text files and not anything else. It can both read from and create text files in the appropriate part of the program. The program reads goals row by row from a dictionary-like file:



```
Nutrient Goals:
Calories: 145.0/3308.4
Protein: 10.0/248.1 grams
Carbs: 15.0/372.2 grams
Fats: 5.0/91.9 grams
```

It skips the first row and starts assigning the second element of each row to the corresponding macronutrient. These are then stored as integers as passed further in the program.

When the program creates a text file, it looks somewhat similar:



```
Nutrient Goals:
Calories: 597.0/3308.4
Protein: 47.0/248.1 grams
Carbs: 55.0/372.2 grams
Fats: 21.0/91.9 grams
```

8. Testing

The testing of the program is more or less according to the plan presented in the early stages, though that was intentionally left vague. The program has tests for valid inputs for the NutrientGoal and NutrientMonitor for valid goal and food logging data. It also tests MainWindow's labels and opens the window (although that isn't tested) and it checks that invalid user inputs are not accepted further. For ResultWindow, there are a few simple tests to ensure that the init function is finished properly.

The program was tested both manually by inputting data into the running program like a user would normally do. Then, of course, there exist three separate files for testing different parts of the program. Currently, the program clears all tests that can be done in the testing files except one that it intentionally fails.

9. Known deficiencies and problems

In normal running of the program, I have tried to make sure that false (negative or invalid) inputs are not accepted when collecting data from the user or the food-logging process. However, I know that large inputs in the weight, height and age categories are accepted. This would mean that if a user inputs their age as 1000, the program would give them a negative calorie and macronutrient goal. Also, the program forgets the inputs a user has put in if it is closed. That is why the program is able to read consumed nutrients from goal files as well. I have to admit that with my current knowledge I am unable to establish a user base for the program so that it would remember users even beyond exiting.

I also guess that the text-file operations house something that would cause them to break. I haven't had sufficient time to thoroughly test them so there might be something that I haven't noticed. Currently they work as intended, however I know they are not as robust as I'd like them to be.

There are a few parts of the program that I would like to improve further. If this was for work, I would try to make the GUI even easier to understand with one or two more functionalities. I would also implement a password and memory beyond termination for the program had I more time for the project.

I would also implement better file handling for the program so that they would house more information and also be able to absorb more information from the user.

10. Three best and three worst

Let me start from the weakest parts of the program. Like I mentioned, there is probably something wrong with the text file operations that are below the surface. They just feel “weak” to me and can probably be hacked very easily.

Next is very clearly the testing. I know the tests are very simple and test for fairly superficial things like title validity. I have to admit that unit tests are among my weakest areas when it comes to Python programming; I can very easily follow my own program from a debugging standpoint but when the testing “mock” layer is added, I am out of the water somewhat.

Among the weaker parts of the program is the visual presentation, it does hold all the necessary information but does so in a very rudimentary way. If I had more time I would like to tweak fonts, text placement, pictures and so on.

Even though I said that the visual representation is one of the weakest areas in the program, it is simultaneously the strongest. It is very simple to understand. There are titles that tell the user what they should be expecting from any given area or widget in the window.

Next good part is the ease of use. I can claim that every user will understand what is going on after they press ‘run’ in the IDE and that I am very proud of.

Finally, the third best part is the updating GUI. I am proud of the fact that the display changes according to the user’s inputs, it feels like the user is actively participating in the running of the program.

11. Deviations from the plan

The only thing that I can think of that deviates from the plan is that I didn’t implement micronutrients into the program. To be frank, I was so far along the programming process when I remembered that I was supposed to implement them that I just didn’t want to completely overhaul the initialising process. I was afraid that it would lead to problems that I couldn’t recognise very easily.

The timetable for the project is roughly what I envisioned, though I think I have used more than 100 hours at this point. If at the time of checkpoint six I was around 60 hours, then at this point I must be above 100. I was very surprised how long this very document took to write. I fixed many bugs and added some things while writing it when it forced me to look at my project’s functionality.

The order of implementation however was exactly as I planned: first a text-based program with supporting functions for a GUI, then the GUI and its functionality, and finally testing.

12. Timetable

The following is a very rough timetable for the program’s construction during this Spring. I am not able to recall dates but will tell what I was working with and approximately when.

Early Spring: Project Plan, first classes, skeleton main function.

March: A finished text-based program, nearly all integral functions finished, first attempts at a GUI, first few tests, many bug fixes.

April: Greatly expanding GUI, adding dynamic buttons, adding another ellipse, improving file operations greatly, very many bug fixes.

May: Adding a few final functions, last tests, last bug fixes, writing the document.

I would say that the general timetable and order of implementation did not deviate in any significant way from the planned order.

13. Evaluation of the outcome

So, this has been by far the largest programming endeavour that I have partaken in. It has been difficult at times, especially when it comes to diagnosing problems and spotting mistakes in the 600 odd lines of code that I have written. It has, however, sparked my imagination and has shown me that creating something from nothing is both possible and fun!

To quickly summarise my program: it calculates calories and nutrients based on a user’s background information. It can also import goals from files. It shows how much a user needs and how much they have consumed while the program has been running and can then export that information to a text file.

The program is, in my opinion, both fairly robust and easy to use without any prior knowledge. However, it lacks in testing, file operations and remembering users. In the future the program could be made prettier with graphical elements, accept and deliver more information from and with files and it could be fitted with a userbase so that they could be remembered without having to save to a file.

In general, I think the program’s structure is good and I have used inheritance well when it is needed. The only major thing that I’d like to change in that department is the ResultWindow class. It is very bloated and had I known that it would get so large, I would have separated some functions into a different file. All in all I think that the current project could facilitate the implementation of these proposed changes with next to no major tweaks in the structure. This is due to the modularity and descending nature of the structure, where many different branches combine into the final presented view. With an addition of a class or two, I think the aforementioned changes would be possible.

The algorithms in this program are fairly straightforward, so I doubt they could've been done much more efficiently. I think some of the data structures could have used some standardising: somewhere they are integers and somewhere floating numbers.

I am very satisfied with both my learning from and my effort towards this project.

14. References and links

1. Ramalho, Luciano. Fluent Python. Sebastopol, CA: O'Reilly, 2015. Print.
2. Python Data Structures https://docs.python.org/3/tutorial/datastructures.html
3. Dietary Reference Intakes for Energy, Carbohydrate, Fiber, Fat, Fatty Acids, Cholesterol, Protein, and Amino Acids. Washington, DC: National Academies Press, 2005. Print.
4. Unit tests https://docs.python.org/3/library/unittest.html
5. PyQt6 https://www.riverbankcomputing.com/software/pyqt/
6. QtWidgets https://doc.qt.io/qtforpython-6/PySide6/QtWidgets/index.html
7. Course home page https://plus.cs.aalto.fi/y2/2024/
8. QtCore documentation: https://doc.qt.io/qt-6/qtcore-index.html
9. Sys documentation: https://docs.python.org/3/library/sys.html
10. YouTube: https://www.youtube.com/?app=desktop&gl=FI&hl=fi
11. GraphicsEllipseItem: https://doc.qt.io/qt-6/qgraphicsellipseitem.html
12. Stackoverflow (also generally for some tips): https://stackoverflow.com

15. Attachments

All of the needed files are present in the folder that will be turned in at MyCourses. Pictures of the running program are in their respective sections of this document.