CW.2 group project

The chefs: Take home pay calculator

James McIntyre – 26198711

Adam Driscoll – 26061392

Joel Moore – 26226723

Hannan Ali – 26096013

Matthew Potter - 26389045

Table of Contents

[Abstract: 1](#_Toc241895966)

[Analysis: 1](#_Toc116272530)

[Introduction: 1](#_Toc1081555836)

[TEAM 1](#_Toc254221652)

[Project Problem 1](#_Toc1707969774)

[Design 1](#_Toc839100377)

[Designs (GUI) 1](#_Toc491073939)

[Design One 1](#_Toc187675301)

[Designs (Program) 1](#_Toc269678504)

[Implementation Summary 1](#_Toc697119496)

[References 1](#_Toc2063758186)

# Abstract:

As a group of five, we collectively have chosen to make a “Take Home Pay Calculator” system for our coursework 2 project. We will be building the application from the ground up and splitting the primary tasks five ways. The primary goal of the project is to create the best and most efficient possible program whilst writing a report that explains each step of the process is a concise manner. Every time a task is completed by a member, each of the other members will be responsible for peer reviewing and editing so that the whole group can be satisfied with the work done.

# Analysis:

In this “Analysis” section of the report, we will be outlining the most intricate details of the project as a whole. This will include explaining each members specific roles and responsibilities, the estimated timeline for the project and an overview of the application itself.

## Introduction:

The primary aim of the project is to create an app that can efficiently calculate the amount of money a given user would be taking for themselves post tax (following English law as of July 2023). This goal, however, can be split into many sub-tasks that will need to be split between the various members of the team.

Initially decomposing the goal reveals a few secondary aims that will be essential to the overall success of the group. The first of these is the programming, which is arguably the most challenging aspect of the coursework. We will allocate specific members of the group to undergo the breakdown and execution involved in programming the application which we have chosen. These chosen members will work together to ensure the building of an efficient and easy to use system.

The secondary aim is to create a report / writeup of everything that the five members of the group have done during their time working on the project. The writing team will have scheduled meetings with the programming team in order to understand what the writeup needs to include in relation to the programming.

The report itself will contain a full breakdown of every step taken to guide the project to success. Examples of this include, individually analysing member specific roles, detailing the characteristics of the program as well as how each characteristic poses different challenges to the programming team. Once the app has been finalised, the write-up team will create a full summary of how each stage went and how each member contributed.

## TEAM

In this section I will explain what each member of the group will be explicitly doing, this separation of tasks will help us complete the project as efficiently as possible.

Adam’s designated tasks for this project are linked very closely to the report section. He will be working with James on this. Together, they will split the report between them. By dividing the workload between them, this will speed up progress and allow peer review to be conducted between them, ensuring work is to a good standard and no errors are present. He will also be communicating frequently with the coding team to determine what progress they have made and identify if there have been any challenges that have posed a significant problem. Working with James, he will review the code for the project and help to pinpoint errors in the program. This extra aid will be crucial in making sure the code functions correctly and the desired outcome is achieved.

James will be performing a very similar role to Adam. This will include splitting the writeup between himself and Adam, expediting the overall progress. He will peer review Adam’s work, which is a crucial part of guaranteeing the writeup is free of mistakes. Feedback will be given to Adam, and he will use this to improve his work to secure a high

level of standard. Just as Adam will peer review the coding team’s program, James will also perform this. By doing this, he will make sure that the code functions properly and is both readable and understandable. Finally, James will be in charge of formatting all of the writeup into an official report. This will contain a bibliography, table of contents, and sufficient citations to avoid plagiarism.

Matthew, as the team leader, will be the designated person in charge of overseeing the whole project. He will especially be the one responsible for managing the coding section of the project. He will undertake the task of creating the basic GUI for the program. This includes creating a database full of people’s usernames, passwords, loans, debts, and annual incomes. By inputting their username and password, this allows a user to see all of their information. This means the computer can also access this to perform calculations.

Matthew will also be the sole person in charge of the user validation sector of the code. This will enhance the user experience and prevent the program from breaking due to a mistake from the user.

Hannan, along with Joel, is in charge of creating the structure for the actual calculations. This includes calculating how much take-home pay someone will get and how much they contribute to national insurance. He will also create the inputs that allow the user to put their name and password in, which then outputs their information. This is done using methods, one of which returns the national insurance contribution for this specific individual, the other returns the income they receive after all deductions (student loan, national insurance, etc.) have been made. He will also be performing some user validation checks, ensuring the user has inputted a valid value and that it is between the basic parameters implemented into the code.

Joel is performing similar tasks to Hannan. However, he is mostly in charge of creating the very basic structure and logical processes for the code. He will be creating pseudocode, and this will act as a plan in order to later help the other programmers understand what code needs to be implemented, making the whole process easier, less demanding and more efficient on a widescale level. Like Hannan, Joel is assisting with creating the user validation checks for the code. This will require the user to input valid values, and to make sure they have actually inputted values. By doing this, the code will run as expected and reduces the chances of the program failing because of the user.

## Project Problem

The project problem is the primary job that the programming team will have to work through. It is a “Take Home Pay Calculator”. We will be building a full console that allows a user to input a number of variables such as income per annum and which national insurance plan they use. These inputs will then allow the system we make to calculate the amount of money them will be taking for personal use per any given timeframe. This is the basic requirement of the system that if we can accomplish, the project will be deemed a success, however there are a couple specific benchmarks the program must pass, as well as extra features we will be adding.

The benchmarks that have been set for us to complete are:

* Must be accurate to the UK Tax rulings est. July 2023 (gov.uk)
* Must work for any given salary input between £0 to £10 million

These shouldn’t be too hard for us to follow as both of them can be met with simple integer changes within the code.

The extra features that we will be adding have been decided specifically by the coding team. There are two primary ones which will be taking up the majority of coding time.

These are student finance integration and an accounts system.

Going into detail about the first, we considered that many potential users may have other factors that will impact their take home salary. We thought about the various possible causes of impact as decided that the main one we should tackle is student finance. This would seriously alter the final result of the program, so it needs to be executed carefully. The program will need to take the users given salary and work out which repayment bracket that puts them in. This will then determine what percentage of their money is to be repaid for student loan debt.

The second feature is the system that allows users to save their data under a dedicated account associated username. The way this will work is by allowing the user to choose their own username and password. This information will then be stored in a database that the program will access every time a new user opens the app. This system will allow individuals to have secure accounts with all their data integrated into them, saving them time and improving the usability of the app.

# Design

The design (front and back-end) of the app will be determined by the programming group. They understand how different aspects should be appear and work to achieve optimal functionality. There will be a few designs for the overall program, each one with a varying level of abstraction. This will give us a general plan of how we expect our project to go. The first design will be quite basic, then we will slowly implement stuff until we have created a more solid base.

We will break the whole project down into three main components. These will be the interface, the data stores and the actual program logic.

For the interface, we will create a very basic design on MS Paint. This will be composed of simple boxes that will demonstrate how we want our interface to be laid out. This will be communicated to Matthew, who is creating the GUI.

He will use this as a springboard to create more complicated designs. There will be two types of designs that we use before implementing the design we decide on. These types are low-fidelity and high-fidelity. We will create two separate design ideas, each with two ideas using the different kinds of fidelity. The key difference between the two types is the level of details in the design.

In terms of program logic, this will be first planned using pseudocode to plan out how the code should be laid out. This basic plan will help the coding team to actually begin coding the logic for the program. Then, the high-fidelity version will be the actual code that the programming team will create. This will include the actual code, variables, functions, classes and comments. It will be professionally presented.

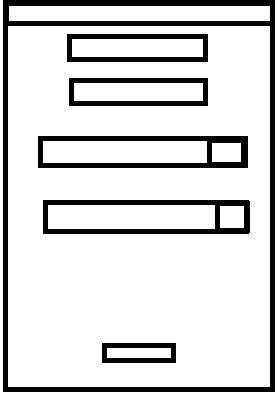
For data stores, we will not need to create various versions of fidelity for these as the data stores will be fairly easy to create. These will include dictionaries to hold the various classes available, such as National Insurance plans. This will also include data stores to hold the different thresholds for the income.

## Designs (GUI)

The first kind of abstraction we will be using in our designs is a “low fidelity” type. These will show possible variations of how the GUI could look. Low Fidelity designs show a very high level of abstraction. These designs will show a basic wireframe presentation that will help us to make designs at a lesser level of abstraction. Upon completing the first design, we will create a new one at the highest fidelity and lowest level of abstraction. Colour and detail will be fully integrated into these so that we can get the best possible vision of how the idea could turn out.

### Design One

This first design shows a basic wireframe for where each of the buttons, titles and dropdown windows should go.



Employment Status

Annual Gross Income

Password input Box

Username Input Box

Sign up Button

This wireframe is generally approved by the group so we will continue to make different fidelity designs based around this layout. As mentioned previously, we will also make more designs to give the team extra options. Here is the high-fidelity improvement on the initial design. This was made in collaboration between the writing and programming team and both sides agree this is a good layout to stick to. For the sake of ensuring the best possible design, we will still create a secondary set of designs in case we end up preferring them. A screenshot of a computer screen

Description automatically generated

## Designs (Program)

The primary logic of the program will revolve around the calculation of the pay from the given inputs by the user. We will start to create pseudocode for this that will give us a good idea of how the actual code should be laid out.

One block of code that we know will need to be included is a section for calculating income after tax (Lewis) so we will write pseudocode for that first.

Function returnIncomeAfterTax:

Is income =< 12570:

return income

Else if income <= 50270:

return 12570 + ((income – 12570) \* 0.8

Else if income <= 125140:

bracket1 = 12570

bracket2 = (50270-12570) \* 0.8

bracket3 = (income – 125140) \*0.55

Return bracket1 + bracket2 + bracket3

This pseudocode sets the framework for the actual code to be written. The final version of the code will be pasted into the implementation summary.

Another important section of code is the making of the GUI. This part of the code will be more complicated that the calculation shown prior so getting a pseudocode base will be essential.

Import library tkinter

Declare Class GUI:

Declare function initialise:

Self title = (“Take Home Pay Calculator”)

self geometry = (X by X)

create label (“Username”)

create label (“Password”)

sign up button = button(“Sign Up”)

One last feature we will display pseudocode for is a visualiser of the tax breakdown. This feature uses a pie-chart to make the information returned by the program easier to read and understand.

Import library matplotlib

Import library tkinter

Define function piechart:

Labels = [(“Take Home Pay”), ("National Insurance"), (“Income Tax”), ( “Student Loan Repayments”)]

Axis set title (“Your Tax Breakdown”)

Canvas.draw()

Window = tk.tk

Button = tk.button(window, command = piechart)

# Implementation Summary

Throughout this project, we have achieved most of our goals. The main goals were as follows:

Create a program that computes an employee’s take home pay after tax has been applied

Must work for any salary input between £0 and £10 million

There were a number of possible enhancements we could have added:

Allow the user to compare the difference in take home pay for two separate salaries

Can be modified to include student loan repayments and national insurance contributions

Can be modified to provide weekly or monthly tax breakdowns

Firstly, we have successfully created a program in Python that will take in an employee’s income and perform the necessary calculations to output how much tax that they owe; this is respective of the salary they earn. For example, someone that earns £12,570 pays no tax, as that is the threshold. The tax someone pays is a percentage on how much over the threshold they earn. If they earn £20,000, they pay 20% of 7430 which is £1486 tax.

Our program is suited to take in any values from £0 to £10 million, therefore these have been met.

Our program is also capable of calculating student loan repayments and national insurance. These are both dependent on the categories the user is part of, and the plan they have. For example, for student loan, there are multiple plans which will affect how much you pay back, therefore the program will take these as input and perform the necessary calculations. This is also true for the national insurance categories.

However, while the main requirements were met and the program works as expected, there were a number of improvements we could have made that may improve the quality of life of the user’s experience. For example, we could have added a function which allows the user to compare the difference between the tax payouts for two salaries, helping the user better with their finances. This may be helpful if they get offered a promotion, they will earn more money but have to pay more back. In addition, a user may find it more helpful to receive their tax payouts in weekly or even monthly increments, rather than it being based off an annual salary.

In conclusion, while we added the main requirements that were asked of us for this project, we believe that we could have added more quality-of-life changes that may better the user’s experience, such as giving the user an option between weekly and monthly breakdowns. This may help them budget their finances more effectively.

# References

(No date) Income Tax Rates. Available at: https://www.gov.uk/income-tax-rates.

Lewis, M.S. (no date) Income Tax Calculator, Income Tax Calculator: Find your take-home pay. Available at: https://www.moneysavingexpert.com/tax-calculator/ (Accessed: 17 December 2024).

Appendices

