**IS1110 Tutorial Portfolio Project**

# Title: Personal Budget Tracker

# Project Overview:

Create a Python programme that simulates a personal budget management system. This project will be built incrementally over the semester, with each week adding new functionality based on the topics covered in class. The system will track weekly income and expenses, calculate remaining money, analyse spending patterns, and provide personalised feedback to help users understand their financial habits and make better budgeting decisions.

# Why This Matters for BIS Students:

This project teaches you to solve real business problems using programming. You'll learn to collect data, make calculations, and present results clearly - all essential skills for working with business systems.

# How It Works:

Your programme will ask users questions, store their answers, do calculations, and show results on screen. The system starts as a simple weekly calculator and gradually evolves into a comprehensive budget management tool, with new features added each week as you learn new programming concepts.

# Weekly Deliverables & Requirements:

You will start working on each project assignment during the tutorial session and submit your work by the end of the week to Canvas. Tutors will walk you through the solutions at the following tutorial session.

# System Description:

Your Personal Budget Tracker works like a digital financial assistant that guides users through a weekly budgeting process. When a user starts the programme, they begin by entering their weekly income, followed by their various expense categories such as rent/accommodation, groceries, transport, and entertainment costs. The system then processes this data by calculating the total expenses, subtracting this from the income to determine money remaining, and analysing the financial health by comparing income versus spending. Based on these calculations, the system determines whether the user has a surplus, deficit, or balanced budget, and provides personalised feedback and recommendations. The system then displays a formatted report showing the income, itemised expenses with percentages, remaining money, and financial status with advice.



As the system evolves over the weeks, it will expand to handle multiple weeks of data through two tracks:

**Core Track (Assignments 1-7):** The system tracks summary statistics across multiple weeks, calculating running totals and averages for income and expenses. Users can add new weeks of data through a menu-driven interface, see how each week compares to their averages, and view professionally formatted summary reports.

**Advanced Track (Optional Assignment 8):** For students with prior programming experience, the system can be extended using lists to store individual week data, enabling features like viewing all previous weeks, finding best/worst weeks, and generating comprehensive comparison reports.

The system maintains a simple menu-driven interface throughout, allowing users to add new budget data, view summaries, and exit the programme cleanly. This creates a complete cycle from data collection through analysis to actionable financial insights, mimicking how professional budgeting applications work in the business world.

Assignment 1: Problem Solving

**Deliverable:** Inputs/processes/outputs & Flowchart

**Task:** Analyse the System Description above, identify all inputs, processes and outputs, and create a flowchart showing the overall system logic



**Requirements:**

* Identify all inputs, processes and outputs for the complete system
* Create a flowchart using standard flowchart symbols

**Hint:** Think about the order of steps when calculating a budget manually

**Submission:** Hand-drawn (screenshot) or digital flowchart with indication of inputs/processes/outputs (see instructions on Canvas on how to create a flowchart with [draw.io](https://www.drawio.com/)). (flowchart.drawio/pdf/png)

Assignment 2: Variables, Expressions & Statements

**Deliverable:** Basic Budget Calculator (Version 1.0)

**Topic Focus:** Using variables and calculations properly

**Task:** Create a simple calculator that tracks weekly income and expenses

**Requirements:**

* Use meaningful variable names following Python naming conventions (e.g., weekly\_income, not x)
* Prompt user for: weekly income, rent, groceries, transport, entertainment costs
* Calculate total expenses and money left over
* Display results showing income vs expenses
* Include header comment and inline comments explaining calculations

**Hints:**

* Use input() to get user data, remember it returns strings
* Convert strings to numbers using int() or float()
* money\_left = income - total\_expenses

**Sample Output:**

Weekly Income: €200

Total Expenses: €150

Money Left: €50

Note: In later weeks, you will add percentages and professional formatting to make this look much more polished.

**Submission:** Python file (budget\_tracker\_v1.py)

Assignment 3: Conditional Execution

**Deliverable:** Budget Status Checker (Version 2.0)

**Topic Focus:** Using if/elif/else statements to make decisions

**Task:** Extend calculator to provide financial health analysis and recommendations, i.e., add logic to tell users if they are overspending or doing well

**Requirements:**

* + Add input validation for monetary amounts (e.g., no negative amounts, only numeric values)
  + Add conditional logic to determine budget status (surplus/deficit/balanced)
  + Give different messages for different situations
  + Check if total expenses exceed income and provide appropriate warnings
  + Provide personalised financial advice based on spending patterns
  + Handle edge cases gracefully

**Hints:**

* + Use if statements:

if money\_left < 0:

print("You're overspending!")

elif money\_left > 50:

print("Good job saving!")

* + Make sure to handle the case where money\_left == 0

**Sample Output:**

Money Left: €-20

Status: OVERSPENDING! You need to cut expenses by €20

**Submission:** Python file (budget\_tracker\_v2.py)

Assignment 4: Functions

**Deliverable:** Modular Budget System (Version 3.0)

**Topic Focus:** Breaking code into separate functions

**Task:** Refactor code into reusable functions for better code organisation

**Requirements:**

* + Create functions for: getting user input, calculating expenses, showing results
  + Each function should do one clear job
  + Add simple docstrings (1-2 lines explaining what each function does)
  + Use parameters and return values effectively
  + Use a main() function to run everything

**Hints:**

* def get\_positive\_float(prompt) # Validates input and returns positive float
* def get\_user\_input(): # gets all budget input from user with validation
* def calculate\_budget(income, rent, food, transport, entertainment): # calculates total expenses and money remaining
* def analyse\_status(money\_left): # determines budget status and provides recommendations
* def show\_results(income, rent, food, transport, entertainment, total\_expenses, money\_left): # displays formatted budget report
* def main(): # main programme function

**Submission:** Python file (budget\_tracker\_v3.py)

Assignment 5: Loops and Iterations

**Deliverable:** Multi-Week Budget Tracker (Version 4.0)

**Topic Focus:** Using while loops and for loops

**Task:** Extend your budget system to track data across multiple weeks using a menu-driven interface. Since we're not using lists yet, you'll track summary statistics (running totals and averages) rather than storing individual week details.

**Requirements:**

* + Add a menu system with options:
    - Add This Week's Budget
    - Show Overall Summary
    - Exit
  + Use a while loop to keep the programme running until the user chooses to exit
  + Track across all weeks entered:
    - Total number of weeks entered
    - Total income across all weeks
    - Total expenses across all weeks
    - Running averages (average weekly income and expenses)
  + Use loops to validate input (keep asking until valid)
  + After each week is entered, show how it compares to the running averages
  + Continue using functions to organize your code (building on Assignment 4)

**Hints:**

week\_count = 0

total\_income = 0

total\_expenses = 0

while True: # keeps going until user chooses to exit

# show menu and get choice

if choice == 1: # Add week

week\_count += 1

total\_income += this\_week\_income

total\_expenses += this\_week\_expenses

# calculate and show averages

**Sample Output:**

=== Week 3 Added ===

This week's income: €200

This week's expenses: €150

Average weekly income so far: €210

Average weekly expenses so far: €140

**Submission:** Python file (budget\_tracker\_v4.py)

Assignment 6: Strings

**Deliverable:** Improved Budget Categories (Version 5.0)

**Topic Focus:** Working with text data using string methods

**Task:** Enhance the weekly budget entry process by allowing users to add optional short descriptions to expense categories. These descriptions help users remember what each expense was for (e.g., 'Tesco weekly shop' for groceries).

**Requirements:**

* + When entering a new week's data, let users add optional short descriptions for each expense category (rent, groceries, transport, entertainment)
  + Use string methods for data cleaning:
    - .strip() to remove extra spaces
    - .title() or .capitalize() for proper capitalization
    - Note that the methods can be chained (.strip().title())
  + Continue using functions to organize your code
  + Display the current week's expenses with their cleaned descriptions immediately after entry
  + Continue tracking the summary statistics from Assignment 5

**Hints:**

# When getting expense input

rent\_desc = input("Rent description (optional): ").strip().title()

groceries\_desc = input("Groceries description (optional): ").strip().title()

# Display after week entry

print(f"Rent: €{rent:.2f} - {rent\_desc}")

print(f"Groceries: €{groceries:.2f} - {groceries\_desc}")

Submission: Python file (budget\_tracker\_v5.py)

**Sample Output:**

Enter rent amount: 800

Rent description (optional): monthly apartment rent

Enter groceries: 120

Groceries description (optional): tesco weekly shop

=== Week 2 Budget Details ===

Rent: €800.00 - Monthly Apartment Rent

Groceries: €120.00 - Tesco Weekly Shop

Transport: €30.00 -

Entertainment: €50.00 - Cinema And Dinner

**Submission:** Python file (budget\_tracker\_v5.py)

Assignment 7: Output Formatting

**Deliverable:** Professional Budget Reports (Version 6.0)

**Topic Focus:** Create professionally formatted console reports using string formatting

**Task:** Make your budget reports look neat, organized, and professional with proper alignment and formatting.

**Requirements:**

* + Implement consistent currency formatting (€ symbol, 2 decimal places everywhere)
  + Add headers and boders to make reports visually clear
  + Use functions to organize different parts of your output:
    - Separate functions for weekly reports and summary reports
    - Functions for formatting different types of output
  + Create aligned columns for category names and amounts
  + Add report headers with the current date (use from datetime import date at the top of the programme file)
  + Show summary statistics with percentages:
    - What percentage of average income goes to each expense category
    - Percentage of income remaining as savings
  + Format both the weekly detail view AND the overall summary view

**Hints:**

# Import current date

from datetime import date

today = date.today()

print(f"Date: {today}") # Displays: 2025-09-29

# Align text in columns using f-string format specifiers

print(f"{'Category':<15} {'Amount':>10}")

# <15 = left-align in 15 characters (good for labels)

# >10 = right-align in 10 characters (good for numbers)

# Create border lines

print("-" \* 40) # Repeats dash 40 times: ----------------------------------------

# Format currency with thousands separator

print(f"Income: €{income:,.2f}")

# , = thousands separator (1,234.50)

# .2f = 2 decimal places

# Calculate and display percentages

percent = (expense / income) \* 100 # Don't forget to multiply by 100!

print(f"{percent:.1f}%") # .1f = 1 decimal place

# Combine alignment + formatting

print(f"{'Rent':<15} €{rent:>9,.2f} {rent\_percent:>11.1f}%")

# Produces: Rent € 650.00 76.5%

**Sample Output:**

╔════════════════════════════════════════╗

║ BUDGET SUMMARY REPORT ║

║ Date: 2025-09-29 ║

╚════════════════════════════════════════╝

Total Weeks Tracked: 4

─────────────────────────────────────────

Category Average % of Income

─────────────────────────────────────────

Income € 850.00 100.0%

Rent € 650.00 76.5%

Groceries € 80.00 9.4%

Transport € 30.00 3.5%

Entertainment € 40.00 4.7%

─────────────────────────────────────────

Total Expenses € 800.00 94.1%

Remaining € 50.00 5.9%

─────────────────────────────────────────

Status: ✓ Balanced Budget - You're on track!

**Submission:** Python file (budget\_tracker\_v6.py)

Assignment 8: Lists [OPTIONAL – ADVANCED]

**Note:** This assignment is **optional** and designed for students who already have programming experience or want an additional challenge. It is NOT required to complete the portfolio project. Assignments 1-7 and 9 form a complete, working budget tracker without lists.

**Deliverable:** Complete Budget System (Version 7.0)

**Topic Focus:** Using lists to store and work with multiple pieces of data

**Task:** Store all budget data in lists to enable week-by-week analysis and comparison features.

**Requirements:**

* + Use lists to store income and expense data for each individual week
  + Add ability to find highest and lowest spending weeks
  + Calculate average weekly spending (using list data, not running totals)
  + Let users view all stored weeks
  + Display a formatted report showing all stored weeks with totals, averages, and identification of your best and worst spending weeks

**Hints:**

weekly\_incomes = [] # empty list to start

weekly\_expenses = []

# When adding a week

weekly\_incomes.append(this\_week\_income)

weekly\_expenses.append(this\_week\_expenses)

# For analysis

max(weekly\_expenses) # finds highest value

min(weekly\_expenses) # finds lowest value

sum(weekly\_expenses) / len(weekly\_expenses) # calculates average

**Main menu:**

1. Add This Week's Budget
2. Show All Weeks Summary
3. Find Best/Worst Week
4. Show Complete Report
5. Exit

**Submission:** Python file (budget\_tracker\_v7.py)

Assignment 9: Integration & Testing (NO VERSION CHANGE)

**Deliverable:** Final Budget Management System + Testing

**Topic Focus:** Putting everything together and making sure it works

**Task:** Complete your system (using Version 6.0 from Assignment 7, or Version 7.0 if you completed the optional lists assignment) and test it thoroughly.

**Requirements:**

* + Make sure all features work together properly
  + Test with different types of input (including bad input)
  + Ensure the programme doesn't crash with unexpected input
  + Add clear instructions for users (consider adding a welcome message)
  + Fix any bugs you find
  + Fix any formatting issues

**Submission:** Python file (budget\_tracker\_final.py) & Complete portfolio project folder with all previous versions and, if applicable, an Appendix (see *Referencing* section below)

# Technical Constraints

* **Allowed constructs:** Variables, expressions, conditionals, functions, loops, strings, lists, input/output
* **Prohibited:** Dictionaries, classes, file I/O, external libraries (except datetime)
* **Platform:** Console-based application only
* **Code standards:** Must follow Python naming conventions (PEP 8)

# Example Console Interaction (Final Version):

=== MY BUDGET TRACKER ===

Main Menu:

1. Add This Week's Budget

2. Show All Weeks Summary

3. Find Best/Worst Week

4. Exit

Choose option: 1

--- Week 3 Budget Entry ---

Weekly income (€): 150.00

Rent/Accommodation (€): 60.00

Food shopping (€): 35.00

Expense description (optional): Weekly groceries at Tesco

Transport (€): 15.00

Expense description (optional): Bus pass

Entertainment (€): 25.00

Expense description (optional): Cinema with friends

=== WEEK 3 BUDGET REPORT ===

Date: 2025-09-15

Income: €150.00

Total Expenses: €135.00

Rent: €60.00 (44%)

Food: €35.00 (26%) - Weekly groceries at Tesco

Transport: €15.00 (11%) - Bus pass

Entertainment: €25.00 (19%) - Cinema with friends

Money Left: €15.00

Status: Good job! You saved €15 this week.

Press Enter to continue...

=== MY BUDGET TRACKER ===

Main Menu:

1. Add This Week's Budget

2. Show All Weeks Summary

3. Find Best/Worst Week

4. Exit

Choose option: 2

=== ALL WEEKS SUMMARY ===

Week 1: Income €140, Spent €130, Saved €10

Week 2: Income €160, Spent €145, Saved €15

Week 3: Income €150, Spent €135, Saved €15

Total across 3 weeks:

Total Income: €450.00

Total Expenses: €410.00

Total Saved: €40.00

Average weekly savings: €13.33

Best week: Week 2 (saved €15)

Worst week: Week 1 (saved €10)

Press Enter to continue...

# Deliverables:

* A flowchart file (with one of the following extensions.pdf, .png or .drawio) documenting the programme’s logic.
* Python script files (e.g., budget\_tracker\_v1.py) that fulfils the project requirements. **Include a header comment** with the file name, your name, student ID, date and a brief description of the programme, as shown in the following example.

# budget\_tracker\_v1.py

# Author: Your name

# ID: Your student ID

# Date: YYYY-MM-DD

# Description: ADD A SHORT DESCRIPTION OF THE PROGRAMME HERE

* Include comments in the code to explain the functionality of different sections.
* Test the programme to ensure it works as expected with different inputs and edge cases.
* Ensure all code adheres to Python naming conventions.
* Provide proper references to any external sources used and disclose any use of AI tools as per the provided instructions.

By completing this project, you will demonstrate your ability to build basic business applications using Python, focusing on user interaction, calculations, output presentation, and code organisation—all while adhering to Python's naming conventions and referencing guidelines.

💡 **Remember:** This project grows with you! Each week builds on what you've learned before. Start simple, ask for help when needed, and by the end you'll have created something really useful.

# Referencing:

* References to any external sources that were used/adapted in the code should be included in the Python file following the guidelines for citing code available on Canvas.
* If AI tools are used for assistance, provide complete transparency including specific prompts, AI responses, critical evaluation, and explanation of adaptations (prompts and AI responses should be copied and pasted into an **Appendix**; additionally, a link to the conversation can be provided, but ensure that it is accessible.)
* If no AI tools were used, state this clearly in the References section.

# Submission & Formatting Guidelines:

* **Each assignment** should be submitted via Canvas in the file format and name specified under each assignment.
* For the **final portfolio submission**, all the assignment files (i.e., Python script, flowchart and appendix, if applicable) should be compressed in a ZIP file submitted to Canvas.
  + The ZIP file should be named IS1110\_Last\_First.zip (e.g.: IS1110\_Seppala\_Selja.zip).
  + The final assignment must be submitted via Canvas using the relevant assignment submission page.

# Feedback Policy:

You may request feedback from the tutors during tutorial hours or book an appointment with the lecturer by sending an email via Canvas Inbox.

# Marking:

The assignments are marked as complete/incomplete and do not count towards your final module mark. They are meant for you to practice developing an entire programme from beginning to end, improve your programming skills with a real-world application, and familiarise yourselves with the Canvas submission process for Semester 2.