**CS1101 – Programming Fundamentals Assignment 4**

**Student Name:** Merhawit Kahsay Gidey  
**Course:** CS1101 – Programming Fundamentals  
**Instructor:** Henry hu  
**Date:** Jul 13, 2025

**Part 1**

This assignment involved creating a Python function that calculates the hypotenuse of a right-angled triangle using the lengths of the other two legs. The development followed the **incremental development** methodology, which focuses on building and testing functionality in small, manageable steps. This approach helps in isolating errors and validating logic progressively.

To implement a Python function using the **Pythagorean Theorem**:

Hypotenuse =

Where a and b are the legs of a right triangle, and the result is the hypotenuse.

**Step 1: Function Skeleton**

def hypotenuse(height, width):

pass

**What it does:**

* Defines a new function named hypotenuse that takes two parameters: height and width.
* The pass statement acts as a placeholder, allowing the function to be syntactically correct but not perform any operations yet.

**Purpose:**

* To set up the basic structure of the function and confirm that it can be called without error.

**Step 2: Dummy Return Value**

def hypotenuse(height, width):  
 return 1

**What it does:**

* Returns the value 1 regardless of the input values.

**Purpose:**

* To test if the function is working as expected in terms of input/output.
* Ensures that when called with arguments (e.g., hypotenuse(3, 4)), the function executes and returns a result without issues.

**Step 3: Square Each Leg and Print**

def hypotenuse(height, width):  
 height\_squared = height \*\* 2  
 width\_squared = width \*\* 2  
 print("Height squared:", height\_squared)  
 print("Width squared:", width\_squared)  
 return 1

**What it does:**

* Computes the square of height and stores it in height\_squared.
* Computes the square of width and stores it in width\_squared.
* Prints the intermediate squared values to the console for inspection.
* Still returns 1 as a placeholder.

**Purpose:**

* To validate that the squaring operations are performed correctly.
* Useful for debugging and understanding intermediate values before completing the full formula.

**Step 4: Add Sum of Squares**

def hypotenuse(height, width):  
 height\_squared = height \*\* 2  
 width\_squared = width \*\* 2  
 sum\_squares = height\_squared + width\_squared  
 print("Sum of squares:", sum\_squares)  
 return 1

**What it does:**

* Adds the squared values of height and width and stores the result in sum\_squares.
* Prints the result of the sum to check if the addition is correct.
* Still returns a dummy value 1.

**Purpose:**

* To ensure the summation of the two squared legs is correctly computed.
* Serves as a check before taking the square root in the next step.

**Step 5: Final Hypotenuse Calculation**

def hypotenuse(height, width):  
 height\_squared = height \*\* 2  
 width\_squared = width \*\* 2  
 sum\_squares = height\_squared + width\_squared  
 result = sum\_squares \*\* 0.5  
 return result

**What it does:**

* Takes the square root of the sum of the squared legs using exponentiation \*\* 0.5.
* Stores the final hypotenuse in the variable result.
* Returns the calculated hypotenuse value.

**Purpose:**

* Completes the Pythagorean theorem calculation.
* Produces the correct result for any given right triangle side lengths.

**Final Optimized Version**

def hypotenuse(height, width):  
 result = ((height \*\* 2) + (width \*\* 2)) \*\* 0.5  
 return result  
  
#Calling with Integer Arguments  
print(hypotenuse(3, 4)) # Output: 5.0

#Calling with Floating - Point Arguments  
print(hypotenuse(6.5, 2.3)) # Output: approx. 6.893  
  
#Calling with Variables as Arguments  
leg\_a = 9  
leg\_b = 12  
print(hypotenuse(leg\_a, leg\_b)) # Output: 15.0

**What it does:**

* Combines all logic into a single line for a cleaner and more efficient implementation.
* Directly returns the result of the hypotenuse calculation.

**Purpose:**

* Optimizes readability and efficiency once all logic is verified to be working correctly.

**Part 2**

As a freelance software developer, I am building a portfolio that showcases custom software functions. This assignment demonstrates my ability to develop a useful budgeting tool using **incremental development**, solid programming practices, and documentation. The program helps users estimate their monthly expenses based on a fixed budget.

## ****Step 1: Define the Problem****

Users need help distributing their monthly budget across key spending categories: **Food, Entertainment, Travel, and Other**. The solution must:

* Take a user’s monthly budget as input
* Calculate recommended allocations based on percentages
* Display the calculated values in a user-friendly format

## ****Step 2: First Version – Core Computation Function****

We start with the base function calculate\_expenses(budget) which computes the breakdown:

def calculate\_expenses(budget):  
 return {  
 "Food": budget \* 0.15,  
 "Entertainment": budget \* 0.10,  
 "Travel": budget \* 0.20,  
 "Other": budget \* 0.55  
 }

Test:

>>> calculate\_expenses(1000)  
{'Food': 150.0, 'Entertainment': 100.0, 'Travel': 200.0, 'Other': 550.0}

## ****Step 3: Display Function for Results****

To improve usability, we create a display\_expenses() function to format and present the data clearly.

def display\_expenses(budget, expenses):  
 print(f"\nBased on your budget of ${budget:.2f}, your estimated expenses are:")  
 for category, amount in expenses.items():  
 print(f" {category}: ${amount:.2f}")

Test:

budget = 1000  
display\_expenses(budget, calculate\_expenses(budget))

Output:

Based on your budget of $1000.00, your estimated expenses are:  
 Food: $150.00  
 Entertainment: $100.00  
 Travel: $200.00  
 Other: $550.00

## ****Step 4: Add Input Function with Validation****

We now need to handle user input safely. The get\_budget\_input() function ensures the input is valid.

def get\_budget\_input():  
 while True:  
 try:  
 budget = float(input("Enter your total budget for a month: "))  
 if budget <= 0:  
 print("Budget must be a positive number.")  
 else:  
 return budget  
 except ValueError:  
 print("Invalid input. Please enter a number.")

## ****Step 5: Integrate and Test Full Program****

Combine all components into a complete script:

def calculate\_expenses(budget):  
 return {  
 "Food": budget \* 0.15,  
 "Entertainment": budget \* 0.10,  
 "Travel": budget \* 0.20,  
 "Other": budget \* 0.55  
 }

def display\_expenses(budget, expenses):  
 print(f"\nBased on your budget of ${budget:.2f}, your estimated expenses are:")  
 for category, amount in expenses.items():  
 print(f" {category}: ${amount:.2f}")

def get\_budget\_input():  
 while True:  
 try:  
 budget = float(input("Enter your total budget for a month: "))  
 if budget <= 0:  
 print("Budget must be a positive number.")  
 else:  
 return budget  
 except ValueError:  
 print("Invalid input. Please enter a number.")

# Main program  
if \_\_name\_\_ == "\_\_main\_\_":  
 budget = get\_budget\_input()  
 expenses = calculate\_expenses(budget)  
 display\_expenses(budget, expenses)

Overall, this assignment highlighted the importance of **incremental development, clear structure, and testing** in software programming. Both projects serve as strong portfolio pieces demonstrating my ability to design custom solutions, work methodically, and write readable, maintainable Python code. These are critical skills as I continue to grow as a freelance developer.

Reference

Downey, A. (2015). *Think Python: How to Think Like a Computer Scientist* (2nd ed.). Green Tree Press. Retrieved from <https://greenteapress.com/thinkpython2/thinkpython2.pdf>