The dangers of cut & paste

Example: Let’s look at a program which asks a user for two positive integers divisible by 5 and then computes the following formula

\[
\text{result} = \text{firstValue} / 5 + \text{secondValue}
\]
Approach 1: Using cut and paste

We cut and paste the code block that asks the user for input.

But it is full of bugs!!!!

Can you find them all?
Approach 2: Using a function

With a function:

- only need to fix bugs once
- abstracts the details so we don’t need to worry about them in main()
- makes main() easier to read
Exercise: Write your own len() function for strings

Analysis:

text = “test” # Input from user

characterCount = 0 # Initial Value

<table>
<thead>
<tr>
<th>Iteration</th>
<th>i</th>
<th>characterCount = characterCount + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0 + 1 = 1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1 + 1 = 2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2 + 1 = 3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3 + 1 = 4</td>
</tr>
</tbody>
</table>
Stack

Terminology: A **stack** is a list where the last item added is the first to be removed.

Analogies:

Pancake stack where we eat the top-most pancake first.

Paper stack where we read the top-most paper first.
Function stack

When we call a function, we create a new context. When the function completes, we return to the previous context.

Analogies:

- Russian Matryoshka dolls
- A dream within a dream (like Inception)

This process is called the function stack, or call stack, or runtime stack
**Function scope**

Local variables are variables that only exist within a function. When the function completes, these variables are destroyed (although their values may stick around).

Global variables are variables that can be seen by the entire program. Because global variables can cause subtle bugs, it is best practice to avoid global variables.

Scope refers to the lines of code where a variable exists. For example, the scope of a local variable in Python3 is the function it belongs to.
Heap

The **heap** contains all the values in our program

A variable lets us refer to raw data that is on the heap

    When you say value = 6, the value 6 is created on the heap

The heap is like a common workspace for your program

    Your program “checks out” values from the heap and then returns values when they aren’t needed anymore (like a library book)