Writing your own functions

Writing your own functions unlocks the true power of programming: they're reusable, more readable, and easier to debug than arbitrary blocks of statements. The key challenge in writing your own functions is in identifying what is always the same versus what will vary (i.e., the parameters).

While writing your own functions initially can be challenging, the following steps are a fool proof way to correctly implement your functions every time. We'll use the following running example as we walk through the tutorial steps and will conclude with a summary checklist you can follow moving forward.

Example Problem:
Write a function, **`header`**, that takes a string *text* and a single character *surround* as parameters and displays the following:

Example call	Displays

`header("Hello, World!", "*")`	`**********`
	`* Hello, World! *`
	`**********`

`header("Python Rocks", "!")`	`!!!!!!!!!!!!!!`
	`! Python Rocks !`
	`!!!!!!!!!!!!!!`

`header("Coders 4 EVER", "+")`	`++++++++++++++++`
	`+ Coders 4 EVER +`
	`++++++++++++++++`

Steps to Implement:

1. **Understand what to implement**
   Begin with an example. Write the lines of code to output one of the example test cases. You may find it useful to make variables for values that are used multiple times:

```
string = "Hello, World!"          # text to display
print("*" * (len(string) + 4) )   # display top line
print("*", string, "*")           # display middle line
print("*" * (len(string) + 4) )   # display bottom line
```

2. **Test the code**
   Run the code to ensure the output matches the target example.

3. **Make variables for repeated values**

   Are there any values in your example code solution that are repeated? If so, replace them with variables. In our running example, the string "`Hello, World!`" and the "`*`" character are used many times. Since we already created a variable for "`Hello, World!`" in the previous step, we only need to create one additional variable now:

```
string  = "Hello, World!"        # text to display
char    = "*"                    # surround character
print(char * (len(string) + 4) ) # display top line
print(char, string, char)        # display middle line
print(char * (len(string) + 4) ) # display bottom line
```

4. **Identify the parameters**

   Of the variables you created, which will stay the same & which will vary when you want to call the function? The variables that you expect to change will become the ***parameters***. In this example, we can see from the 3 example calls that both variables, the text to display and the surround character, will change & become parameters.

5. **Write the function signature**

   Now we're ready to define the first line of the function. At this stage of your learning, most of these details will be given as part of the question. For our example:

   > *Write a function, **`header`**, that takes a string **text** and a single character **surround** as parameters*

   We can use these details to write the following function signature:

```
def header(text, surround):
```

6. **Substitute in parameters**

   Now we can finish implementing the function's body by changing the variables over to parameters & indenting. In our earlier code, we had created two variables that would become parameters: `string` and `char`. Now our goal is to identify which variables should become which parameters and align the names to match. In this example, `string` will become the parameter `text` and `char` will become the parameter `surround`:

```
def header(text, surround):
    print(surround * (len(text) + 4) )    # display top line
    print(surround, text, surround)       # display middle line
    print(surround * (len(text) + 4) )    # display bottom line
```

7. **Test the function**

   The next step is to test that our function works for our initial test case by calling it from main (or below the function definitions). In this step we should use the exact same example as step 1, and we expect to see the same output:

```
header(“Hello, World!”, “*”)
```

   where the output should be:

```
**********
* Hello, World! *
**********
```

8. **Test the function some more**

   Now we get to test the generalizability & reusability of our function -- does it work correctly with parameters *other* than our original test case? Our output should exactly match the example output for each test cases. If it doesn’t, we should refine our implementation until it does.

   Our final program with all 3 test cases:

```
def header(text, surround):
      print(surround * (len(text) + 4) )     # display top line
      print(surround, text, surround)        # display middle line
      print(surround * (len(text) + 4) )     # display bottom line

def main():                                  # execution begins here
      header(“Hello, World!”, “*”)           # original test case
      header(“Python Rocks”, “!”)            # test case #2
      header(“Coders 4 EVER”, “+”)           # test case #3
```

   which outputs:

```
**********
* Hello, World! *
**********
!!!!!!!!!!!!!!!
! Python Rocks !
!!!!!!!!!!!!!!!
+++++++++++++++
+ Coders 4 EVER +
+++++++++++++++
```

   How many test cases you need will depend in part on how complicated your function is. This function is relatively simple, with no loops or ifs. You would want to add at least 2-3 more test cases for every additional loop or if in a function to cover all possible paths through the code.

Writing Your Own Functions Checklist

- ❑ **Understand what to implement:** begin with an example test case.
- ❑ **Test the code:** run the code to ensure the output matches the target example test case.
- ❑ **Make variables for repeated values:** any values in your code that repeat, replace with variables.
- ❑ **Identify the parameters:** the variables that you expect to change when you call the function will become the parameters.
- ❑ **Write the function signature,** or define the first line of the function.
- ❑ **Substitute in parameters:** finish implementing the function's body by changing the variables over to parameters by identifying which variables should become which parameters and aligning the names to match. Don't forget to indent!
- ❑ **Test the function** by calling it from main (or below the function definitions) with the same example test case from the first step.
- ❑ **Test the function some more** by using parameters *other* than the original test case. If the output doesn't exactly match the example output for each test case, refine the implementation.