More Practice with Functions and Programs

Learning Objectives: By the end of this lab you should:

- be more comfortable using functions
- be able to define and use \texttt{main()}
- be able to use optional arguments
- recognize the usefulness of using a “divide and conquer” approach to programming

Prerequisites: Before starting this lab, you should be able to:

- compose programs using good programming habits, that have headers, and that make use of comments, docstrings, variables, basic arithmetic, data types, and basic I/O statements
- use the \texttt{divmod()} function
- define and call functions

Task 1: Counting Digits (pair task)

How many digits are there in $10^1$, $10^2$, and $10^3$? It’s fairly easy to answer 2, 3, and 4, respectively, and we can quickly convince ourselves that, in general, $10^n$ is 1 followed by \( n \) zeros and, thus, $10^n$ has \( n + 1 \) digits. But what about $3^{100}$? How many digits are there in $3^{100}$?\(^1\) In this task, you’ll write a program that will answer this question. Counting digits might seem pointless to you, but counting is frequently used in programs (e.g., for bioinformatics applications).

Write a program that counts the number of digits in \( j^k \) where \( j \) and \( k \) are integer values specified by the user. Your program should consist of at least two functions:

- \texttt{calculate()}: Calculates the value of \( j^k \) and the number of digits in \( j^k \).
- \texttt{main()}: Prompts for the base \( j \) and exponent \( k \) and displays the results of the calculations.

Don’t forget to include \texttt{docstrings} in each function and to end your program with a call to \texttt{main()}. Your output should be similar to the following:

Enter the base \texttt{j}: \texttt{3}

Enter the exponent \texttt{k}: \texttt{100}

\texttt{3 ** 100 = 515377520732011331036461129765621272702107522001}

This number has 48 digits.

\(^1\)Actually, there’s a way to figure this out fairly easily: use \texttt{int(k \log_{10}(j)) + 1}, where \( j = 3 \) and \( k = 100 \), but we don’t know how to calculate logarithms yet.
Hint: You will need to use the functions `str()` and `len()`. The `str()` function converts its argument to a string, and the `len()` function returns the length (i.e., the number of characters) of a string. For example,

```python
>>> str(123)
'123'
>>> x = 197.43
>>> str(x)
'197.43'
>>> len('Go')
2
>>> s = 'Cougs!
>>> len(s)
6
```

Save your program to a folder called Lab3 under CS111, and prove to your TA that it works for credit.

---

**Task 2a: Points on a Line (pair task)**

Recall that the equation for a line is given by:

\[ y = mx + b \]

where \( m \) is the slope and \( b \) is the \( y \)-intercept.

Consider a line passing through two points \((x_1, y_1)\) and \((x_2, y_2)\). The slope \( m \) is given by the “rise” over the “run,” i.e.,

\[ m = \frac{y_2 - y_1}{x_2 - x_1}, \]

and the intercept \( b \) is given by either

\[ b = y_1 - mx_1 \]

or

\[ b = y_2 - mx_2. \]

Write a function called `calc_slope_intercept()` that takes four arguments, two values for point \((x_1, y_1)\) and two for point \((x_2, y_2)\), and returns the slope \( m \) and intercept \( b \) for the line passing through the two points. It should work as follows in the IDLE Shell window:

```python
>>> # Points (0, 2) and (1, 4).
>>> calc_slope_intercept(0, 2, 1, 4)
(2.0, 2.0)
>>> # Points (0, 0) and (1, 4).
>>> calc_slope_intercept(0, 0, 1, 4)
(4.0, 0.0)
>>> m, b = calc_slope_intercept(0, 0, 1, 4)
>>> print("slope: ", m, ", intercept: ", b, sep="")
slope: 4.0, intercept: 0.0
```
Make sure this function works properly before moving on to the next task.

**Task 2b:** (pair task)

Write a function called `line()` that takes three arguments, an $x$ value and the slope $m$ and $y$-intercept $b$, and returns the value of $y$ for the given point $x$, i.e., it returns $mx + b$. It should work as follows in the IDLE Shell window:

```python
>>> line(0, 3, 4)  # Arguments x=0, m=3, b=4
4
>>> line(1, 3, 4)
7
>>> line(2, 3, 4)
10
```

Make sure this function works properly before moving on to the next task.

**Task 2c:** (pair task)

Now use the functions you wrote in the previous two tasks to write a program that first prompts the user for the values associated with the two points $(x_1, y_1)$ and $(x_2, y_2)$, then prompts the user for an arbitrary value of $x$, and finally prints the value of $y$ for the given $x$ and line defined by the two points.

Think about this a bit before starting and, perhaps, jot down how your program will flow from `main()` to `calc_slope_intercept()` and from `main()` to `line()`. Don’t forget to use docstrings for each function. Altogether your program should have three functions and end with a call to `main()`.

Your output should look similar to the following:

```plaintext
Enter x and y for first point [comma-delimited]: 0, 2
Enter x and y for second point [comma-delimited]: 1, 4
Enter any x: 5
Corresponding value of y: 12.0
```

Save this program in your Lab3 folder, and show it to your TA to get credit.

---

**Task #3:** Using an Optional Argument (Keyword) (pair task)

Functions often have default values for parameters. For example, the built-in `print()` function has a default value of a single space for `sep`. Whenever you use the `print()` function, it will use a space between variables unless you set `sep` to something other than a single space. To create an optional argument, simply define on optional argument and set its default value.

Write a function called `go()` that takes a single optional argument called `name`. The default value for `name` should be `'Cougs'`. In the IDLE Shell window, the output would look like the following:
Run this function in your Shell window, but also save it as a function in your Editor window so you won’t forget how to do it. Show it to your TA for credit.

---

**Task 4a: Summing Digits (pair task)**

Write a function called `sum_digits()` that takes a single argument, a positive integer up to five digits long (i.e., between 0 and 99999), and returns its sum. For example, for 5378 the sum is \(5 + 3 + 7 + 8 = 23\). Your output should look like the following:

```python
>>> sum_digits(5378)
23
>>> sum_digits(1)
1
>>> sum_digits(11)
2
>>> sum_digits(111)
3
>>> sum_digits(11113)
7
```

Make sure this function works properly before going on to the next task.

**Task 4b: (pair task)**

If you sum the digits of a number and the result is a multiple of 3, then the original number is a multiple of 3. For large sums, you may have to think a bit before deciding whether a sum is a multiple of 3. To make life easier, you can sum the digits in the sum and see whether it results in a multiple of 3. You can continue to do this until it becomes obvious that the number is a multiple of 3—or not.

Write a program that will prompt the user for an integer up to 5 digits long, sum the digits in this number, and display the results. It should then sum the digits in the sum and display the results. Your program should consist of a `main()` function and the `sum_digits()` function. (Hint: Recall how you used `divmod()` to find the number of dollars, quarters, dimes, nickels, and pennies to make change in your last lab.) The output should be similar to the following:

```text
Enter an integer up to 5 digits long: 99999
Sum of digits in 99999: 45
Sum of digits in 45: 9

Enter an integer up to 5 digits long: 1234
Sum of digits in 1234: 10
Sum of digits in 10: 1
```
Save your program in your Lab3 folder, and show it to your TA to get credit for it.

---

**Task 5: Mad Libs—Short Version (pair task)**

In the word game Mad Libs, people are asked to provide parts of speech such as nouns, verbs, adverbs, or adjectives. These words are used to fill in the blanks of a template or to replace the same parts of speech in a sentence. In this task you’re going to write a program to demonstrate how the game works for a single sentence. Consider this sentence from Ready Player One by Ernest Cline:

A lich was an undead creature, usually an incredibly powerful wizard or king who had employed dark magic to bind his intellect to his own reanimated corpse, thus achieving a perverted form of immortality.

Write a program that will do the following:

- Prompt the user for one verb, four nouns, and three adjectives.
- Print the sentence above but with words replaced as follows:
  
  A [noun] was an undead [noun], usually an incredibly powerful [noun] or king who had employed [adjective] magic to [verb] his [noun] to his own [adjective] corpse, thus achieving a [adjective] form of immortality.

Your program should have three functions and use function calls as appropriate. It should end with a call to `main()`. Note that the `main()` function doesn’t require a `return` statement because the `main()` call doesn’t have a body or a parameter. Include the following functions:

- `get_sent_part()`: Prompts user for part of speech; called multiple times from `main()`
- `build_sentence()`: Uses string concatenation, i.e., combining string using the plus sign, as appropriate to build the sentence from the user’s words and the sentence words
- `main()`: Calls all the functions as appropriate and prints the Mad Lib sentence. Don’t forget to assign names to function calls that require returned values.

If you finish your lab tasks early, please consider helping others or else work on the programming assignment that’s due on Friday. Also, note that your first exam is next week.