CSCE 1035 Lab 07 Fall 2021

1. zyBooks Labs

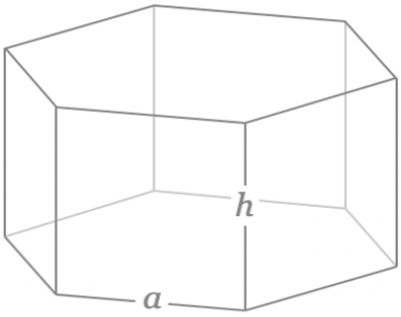
Please follow the link on Canvas to complete the following zyBooks labs:

- 6.18 *LAB: Miles to track laps*
- 6.19 *LAB: Max magnitude*
- 6.24 *LAB: Convert to binary - functions*
- 6.25 *LAB: Swapping variables*
- *6.27 LAB: Even/odd values in a list*
- *6.29 LAB: Warm up: Text analyzer & modifier*

2. Function Arguments and Return Values

Functions can accept 0 or more arguments and can return exactly 1 item, which may be `None`, a single variable, or a container that holds several items. To show how this works and get you some practice, you will write a complete Python 3 program that calculates the surface area of a hexagonal prism using a few different types of functions.

First, the surface area of a hexagonal prism can be described as follows:



$$A = \underbrace{6\,a\,h}_{\text{Surface area of the 6 rectangular faces.}} + \underbrace{3\sqrt{3}\,a^2}_{\text{Surface area of the 2 hexagonal faces.}}$$

This means that the total surface area of the prism is the sum of the area of the hexagonal faces (at the top and bottom) and the area of the six rectangular sides.

Write a small, but complete Python3 program called **Lab7A.py** that computes and prints out the surface area of a hexagonal prism as follows:

a. Create a user-defined function called `calc_hex_area()` that accepts one argument, the length of the base edge of the hexagonal prism in feet. This function will simply have one line that returns the surface area of the 2 hexagonal faces using the formula above.

b. Create a user-defined function called `calc_side_area()` that accepts two arguments, the length of the base edge and the height of the hexagonal prism in feet. This function will simply have one line that returns the surface area of the 6 rectangular faces using the formula above.

c. Create a user-defined function called `display_surface_area()` that accepts two arguments, the surface area of the hexagonal faces and the surface area of the rectangular sides returned in `calc_hex_area()` and `calc_side_area()` functions, respectively. This function will simply have one line that prints the sum of the two area arguments passed to the function with no `return` statement. Be sure to format the result to 3 decimal places.

d. Prompt the user for and read in the length of the base edge, `a`, and height, `h`, of a hexagonal prism as floating-point numbers in feet.

e. Call the `calc_hex_area()` function, passing in the length of the base edge entered by the user, and assigning its return value to the variable called `hex_area`. Then, print the surface area of the hexagonal faces, formatting the result to 3 decimal places.

f. Call the `calc_side_area()` function, passing in the length of the base edge and the height of the hexagonal prism entered by the user, and assigning its return value to a variable called `side_area`. Then, print the surface area of the rectangular sides, formatting the result to 3 decimal places

g. Call the `display_surface_area()` function, passing in the surface areas of the hexagonal faces and rectangular sides. Note that no assignment is needed since this function does not return any item.

For example, the output might look like this (input shown in **bold**):

```
$ python3 Lab7A.py
Enter height of hexagonal prism in feet: 6.5
Enter length of the base edge in feet  : 4.2
Surface area of hexagonal faces is 91.660 square feet
Surface area of rectangular sides is 163.800 square feet
The total surface area is 255.460 square feet
```

Note that you will submit this file to Canvas.

3. Working with Functions and Lists

In this lab component, you will explore working with functions and lists.

a. Create a user-defined function that accepts no arguments and returns a list.

- Create an empty list.
- Prompt for and read in the number of integer values in the list.
- Use a loop and the `range()` function to iterate through the number of values in the list. Inside the loop:
  - Prompt for and read in each integral value and then append the item to the end of the list.
- Return the list.

b. Create another user-defined function that accepts one argument, the list of integers, and returns the second highest number in a list of integers *without modifying the original list*.

   There are several ways to do this. The first is the more complicated way of manually stepping through each element in the list, keeping track of the both the highest and second highest integers in the list. However, we want to take advantage of some Python 3 capabilities and let Python do the work! Other ways would be to sort the list or remove the largest integer, but since lists are mutable, this modifies the list, which is not acceptable. Let's explore a couple different ways to copy a list.

   - Make a copy of the list using the assignment operator =, such as `new_list = old_list`.
   - Use the list `sort()` method to sort the list. Remember the difference between a method and a function – a method is associated with an object and accessed using the dot operator.
   - Print the original list (i.e., the list that was passed to the function to see if it was updated by the `sort()` method.
   - Return the second largest element in the now sorted copy of the list. Remember that you can access the largest item at the end of the list using -1.

   Before we go on to the next part of this lab component, was the original list modified when you sorted your new list? What happened is that the assignment operator = made a *shallow copy* of the original list, meaning that the objects were the same so that if one was modified, so was the other. Instead, we want a *deep copy* that can be accomplished using the `copy()` method that creates a separate instance of the list.

   - Now, instead of making a copy using the assignment operator =, use the `copy()` method to make a deep copy of the original list.

c. Create another user-defined function that accepts one argument, the list of integers, and modifies the list so that all the integers in the list are zero or positive (i.e., not negative) without returning any item.

   - Loop through the list, but instead since we want to access and modify individual elements in the list, we should iterate from 0 to the length of the list (using the `range()` function). Inside the loop:
     - If the element in the list is less than 0, modify that element in the list to make it positive using the `abs()` function in the `math` module.
     - Note that there is no `return` statement in this function.

d. Once the three functions are defined, you will call your function to create your list, without passing any arguments, assigning the result to a list.

e. Print your list.

f. In a `print()` statement, call your function that returns the second highest integer, passing in your list as an argument, with an appropriate message.

g. Call your function to make all elements in the list zero or positive, passing in your list as an argument.

h. Finally, print your now positive list.

For example, the output might look like this (input shown in **bold**):

```
$ python3 Lab7B.py
Enter number of integers in list: 5
Enter item #1: 4
Enter item #2: -3
Enter item #3: 8
Enter item #4: 7
Enter item #5: -1
My original list: [4, -3, 8, 7, -1]
[4, -3, 8, 7, -1]
Second largest integer in list is 7
My now positive list is [4, 3, 8, 7, 1]
```

Note that you will submit this file to Canvas.

Now that you have completed this lab, it's time to turn in your results. Once you've moved the files to your windows machine (using **WinSCP**), you may use the browser to submit them to Canvas for the **Lab 07** dropbox.

You should submit the following files:

- **Lab7A.py**
- **Lab7B.py**
- **(Note that the zyBooks labs are submitted separately through Canvas.)**

Ask your TA to check your results before submission.

Now that you've finished the lab, use any additional time to practice writing simple programs out of the textbook, lectures, or even ones you come up with on your own to gain some more experience.