

ITSE 1359 – Lab 5 Assignment (Refer to Ch. 5 as needed):

LAB 5: This lab requires you to work on and submit two programs. Read this document carefully and comply with submission and grading criteria. For your first problem (Body Mass Index) use the video provided to help you build it. As for the second program (Order) use the provided pseudocode to help you build it. **Do not deviate from requirements.**

Problem #1 – Body Mass Index (bmi.py):

Problem Definition: Create a Python program that calculates BMI on a list of four people. This program should show name of person, bmi, and classification. Name your file “bmi.py”. Do not forget general and specific comments. **You will need this solution again later, so keep it safe.**

Create 3 parallel lists and name them: names, heights, and weights. You can use any 4 first names of your choice for the names list but use the provided data for heights and weights. For the heights list (inches): 66, 62, 50, 70. For the weights list (pounds): 150, 100, 140, 110. The names list should have four first names entered in lower case.

The formula to find bmi is: $(\text{weight} * 703) / (\text{height} * \text{height})$. You must use and elif structure to determine feedback or classification. Use this scheme: If bmi ≥ 25 then classification is Overweight; Else if bmi ≥ 18.5 then classification is Healthy. Else if bmi ≥ 16 then classification is Underweight. Else classification is Invalid.

This lab assignment must show use of lists, loops, conditionals, and string formatting. Be sure to use tab escape codes to create the columns of output. Be sure to format bmi to 2 decimal places. Make sure you use the for in Loop to traverse the lists and format the data as required. Your program should look like the screenshot provided.

Specific Comments / Pseudocode (BMI):

```
# define & initialize 3 parallel lists for names, heights, and weights
```

```
# print program name and report header line
```

```
# use for in loop to traverse lists:
```

```
    # calc bmi by accessing index value of weights and heights lists
```

```
    # based on bmi, provide feedback on BMI classification
```

```
    # print detail line for name, bmi, and classification
```

Screenshot of output (BMI):

```
BMI Program:

NAME      BMI      CLASSIFICATION
Bob       24.21    Healthy
Betty     18.29    Underweight
Liz       39.37    Overweight
Chris    15.78    Invalid

[Finished in 0.1s]
```

Code Examples (BMI):

How to use a loop to find BMI for those in list?

```
# use for in loop to traverse lists:
for i in range(len(names)):
```

How to use conditional structure to provide feedback? (elif chain)

```
# based on bmi provide feedback
if bmi >= 25:
    classification = "Overweight"
```

How to print detail line? (Numeric formatting not in book)

```
# print detail line for name, bmi, and feedback
print(f"{names[i].title()} \t{bmi:.2f} \t{classification}\n")
```

Correction Needed: The coding style used in my solution above is not compliant with PEP-8. We should **not** create variables called “i”. I just can’t get it out of my system because I learned it that way in C++/Java programming. You have a chance to learn and follow “PYTHON” conventions the right way. So, instead of using “i” use “value”:

```
# use for in loop to traverse lists:
for value in range(len(names)):
    # calc bmi by accessing index value of lists
    bmi = (weights[value] * 703) / (heights[value] * heights[value])
```

VIP – Keep your BMI Solution: Keep your solution to this program in a safe place because you will need it again in another lab assignment.

Problem #2 – Order (order.py):

Problem Definition: Create a Python program that sums the order at a hamburger restaurant. If an item is on the menu, it is added to order, else, feedback is provided saying sorry. This program must use parallel lists, for in loop, and an if / else structure. Call your file order.py.

Coding: Create three parallel lists: menu, price, and order. Use the data provided. Items in the menu list: hamburger, soda, fries, chicken nuggets, coffee, cheeseburger. Items in the price list: 3.00, 1.25, 2.00, 2.90, .90, 3.40. Items in the order list: hamburger, fries, soda, pizza. All the items should be entered in the order provided and in lowercase if string data.

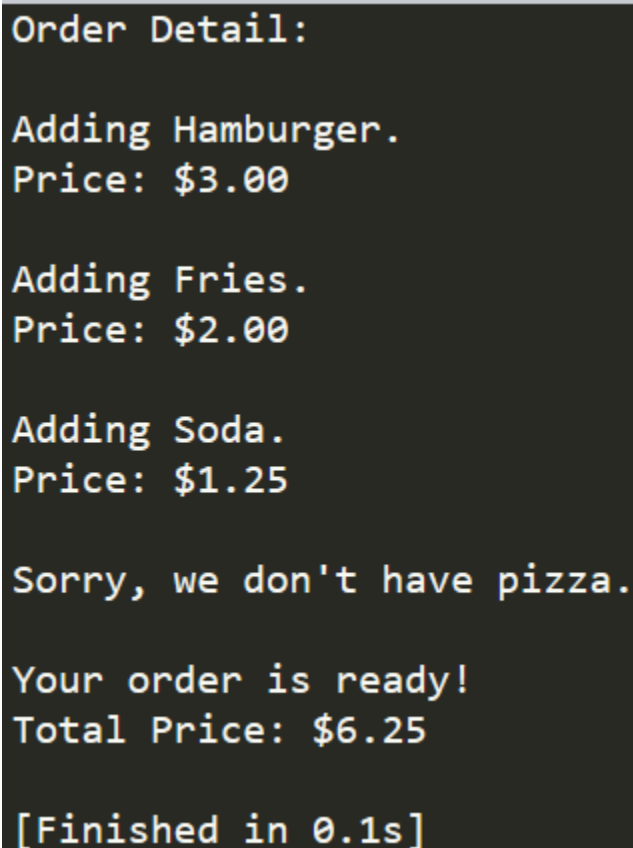
Output: Loop through the order and inside the loop check to see if in menu. If in menu, then print it out. Get the index of the menu item and use that to find the price of that item. Print the price for the item. If there is an item that is not on the menu, state that it is not on menu. Finally, print that the order is ready and the price for the whole order. Money should be formatted for currency. Your program output should look like the screenshot provided.

Pseudocode / Comments (Order):

```
# define list of menu items
# define list of prices that correspond to menu items
# define list containing a customer order
# initialize total_price to 0.0
# print report header – Order Detail

# use for in loop to traverse customer order
    # inside the loop check to see if order in menu
        # print menu item name
        # assign menu item index to variable
        # use index variable to show price from price list
        # increment total_price
    # else print sorry we don't have that item

# print order is ready
# print total price
```

Screenshot of output (Order):

```
Order Detail:

Adding Hamburger.
Price: $3.00

Adding Fries.
Price: $2.00

Adding Soda.
Price: $1.25

Sorry, we don't have pizza.

Your order is ready!
Total Price: $6.25

[Finished in 0.1s]
```

Code Examples (Order):

What is an accumulator variable?

- An accumulator is used to keep a running total of numbers.
- In a loop, a value is usually added to the current value of the accumulator.
- If it is not properly initialized, it will not contain the correct total.

How do I initialize the `total_price` accumulator variable?

```
# initialize total price
total_price = 0.0
```

So, declare your accumulator variable outside the loop and increment your accumulator inside your loop.

How do I loop through the order?

```
# use loop to get order
for order in customer_order:
    if order in menu_items:
        print(f"Adding {order.title()}")
        order_index = menu_items.index(order)
        print(f"Price: ${menu_prices[order_index]:.2f}\n")
        total_price = total_price + menu_prices[order_index]
    else:
        print(f"Sorry, we don't have {order}.")
```

Lists in Python should be plural:

In the above code you can see that the **menu_items** list holds the list of menu items and the **menu_prices** list holds the prices for each of those items. Lists should be plural. These two lists are “parallel lists”. You can use other names for your lists so long as they comply with coding conventions and have meaningful names. Note that my **customer_order** list is not plural because the order contains “a” customer order, so it’s OK for it to be singular.

Abbreviated Assignment Operator:

This is the full expression:

```
total_price = total_price + menu_prices[order_index]
```

This is the abbreviated version (use the one you like):

```
total_price += menu_prices[order_index]
```

See w3schools for listing of assignment operators: [Python Operators \(w3schools.com\)](https://www.w3schools.com/python/python_operators.asp)

Submit your lab assignment:

Using the Canvas assignment tool, upload your completed work (**2 files**) to the **lab 5 assignment**. Attach the first file (bmi.py) and then the second (order.py) and then submit.

All lab assignments must be submitted using the CANVAS assignments tool. **Lab Assignments will not be accepted any other way.** Make sure you submit your work to the right lab assignment number otherwise you will not get credit.

Grading Criteria:

- ✓ You must use the provided video to help you do the BMI program.
- ✓ Don't forget general comments.
- ✓ Specific comments are optional.
- ✓ Use white space in the right places to make your code easy to read.
- ✓ Comply with **PEP-8** conventions for variable names, file names, etc.
- ✓ Satisfy the problem definition and other grading standards.
- ✓ Your work should not have syntax errors (crashing programs major points off).
- ✓ Your work must be your own.
- ✓ Match your output screen to screenshot provided.
- ✓ Use only the coding techniques ask for in problem statement.
- ✓ **Any deviations from lab specifications will result in points off.** (Read this again!)
- ✓ If you want to vary, do so on your own. In fact, I would love to see your enhanced version – send to me by canvas inbox and let's discuss in zoom office hours.

VIP Videos:

Follow this (step-by-step): [Lab 5 BMI Video](#)

Lectures and other videos: [VIEW MY PYTHON PLAYLIST](#)

Closing:

If you have questions about this lab, send me a message using canvas inbox or attend zoom office hours. See Unit 0 for link for my office hours. Programming tutors are also available and listed in your Canvas class as announcements.

By-the-way, a great way to get ready for your lab assignments (and exams) is do the Try It Yourself problems in your book. Most of the [solutions](#) are on the authors website and I also discuss them in my YouTube video lectures.

Another way to get ready for your lab assignments is to review my lecture notes for each chapter. In Unit 0, find the link for my chapter notes.

Warning: Labs will increase in complexity with each lab assignment. A beginner may need 5 to 15 hours to complete each lab. Please start early because there will be no extensions.

Consider doing this lab over a three-day period:

- ✓ Day 1 – Get confused and run out of time.
- ✓ Day 2 – Research and debug errors.
- ✓ Day 3 – Polish, double-check everything, and submit.

Have an exception free day!

Prof. Benavides