# CS-119 Lab #6

## Expected Learning Objectives

- Creating applications in Python.
- User defined methods
- More experience with Python repetition (loop) structures.
- More experience with arrays and lists.

## Overview

This lab provides an opportunity to apply the concepts and principles presented in PLD chapter 6. Primarily, we will be focusing on creating user defined methods. The idea here is to build a “tool box” of reusable code that you may reuse for other Python programming exercises and perhaps even in the work place.

In real world software development, it is a common practice to implement libraries of reusable code needed for a larger project in a small “test fixture” environment similar to what we will be creating in some of these exercises. Code can be more easily tested/debugged in a small scale environment; different algorithms can be coded and tested to find the one with the best performance, etc. Once everything is tested and working correctly, it is then integrated into a larger scale project and reused in future projects.

Very often, interns and entry level software developers are tasked with coding and/or testing libraries such as these.

## Exercise 1 Guided

This first exercise will give you a guided walk through of creating a simple library of user defined methods. You will be allowed to (re)use any/all of these methods in this library in future lab exercises so completing it and having everything working correctly now will give you an advantage later on.

### Procedure

1. Create a Python file called Methods.py.

2. Create a user defined method called get_float(prompt) that accepts a prompt as a parameter, uses that prompt to prompt the user for an input, and stays in a loop until a float value is entered. You are given the code for this one. You are also given a “sneak peak” to chapter 10 where we use exception handling to validate input and recover from runtime errors. Watch indentation closely. It gets tricky with both a while loop and try/except blocks.

```
#!/user/bin/env python3

# Cuyamaca College CS-119
# Your name here!
# Lab 6 exercise 1 Methods

# Step # 2 create a user defined method to get a float value
# Sneak peak to chapter 10 exception handling
# we use a try/except block to validate our input
# we stay in a loop until we get a float value
def get_float_val(prompt):
    is_num = False
    str_val = input(prompt) # priming read
    while is_num == False:
        try:
            value = float(str_val)
            is_num = True
        except:
            print(str_val + " is not a float!")
            str_val = input(prompt)
    # end while

    return value
# end get_float_val()
```

3. Create a Python file called MethodTest.py.

4. Import your Methods file. Note how you can create a shortened “alias” name for your import file. In this case, we ‘ll call it “m”. You will also create a main() method for testing the methods you create and code to invoke the main() method. You are given the code for this:

```
#!/user/bin/env python3

# Cuyamaca College CS-119
# Your name here!
# Lab 6 exercise 1 Method test fixture

# import the methods file. Make sure this matches the name of your file EXACTLY!
import Methods as m

# define a main() method
def main():
    # test the get_float_val() method
    float_val = m.get_float_val("Enter a floating point number: ")
    print("Float value " + str(float_val) + " was entered")

    # add your code to test the various other methods




# if this is the main module then execute main()
if __name__ == "__main__":
    main()
```

5. Test your get_float_val() method. Test with both invalid and valid values. Be sure to ***save*** both files ***before*** testing!

```
====== RESTART: C:/Cuyamaca/2019FallCS119/LabsDev/Lab06/MethodsTest.py ======
Enter a floating point number: 3rd
3rd is not a float!
Enter a floating point number: xyz
xyz is not a float!
Enter a floating point number: 3.4.5
3.4.5 is not a float!
Enter a floating point number: 6.75
Float value 6.75 was entered
```

6. In your Methods file, create a user defined method called get_int_val(prompt) that accepts a prompt as a parameter, uses that prompt to prompt the user for an input, and stays in a loop until an integer value is entered. You get to code this one. Hint: The code will be very similar to the get_float_val() method.

7. Add code to the main() method to test your get_int_val() method. The code will be very similar to what you did to test the get_float_val() method.

8. Test your get_int_val() method. Be sure to test with both invalid and valid values.

9. Create a user defined method called is_float(value) that simply tests if an input value is a float. This method will accept a string input value as a parameter and return a True/False (Boolean) result. You are given the code for this one. It also uses exception handling to determine if the value is a float.

```
# User defined method that simply tests if an input value is a float.
def is_float(value):
    try:
        num = float(value)
        return True # if we make it here, the conversion was successful
    except:
        return False # conversion to float failed so return false
# end is_float()
```

10. Add code to the main() method to test your is_float() method. You are given the code:

```
# test is_float()
str_val = input("Enter a float: ")
result = m.is_float(str_val)
if result == True:
    print( str_val + " is a float.")
else:
    print( str_val + " is NOT a float.")
```

11. Be sure to save your work and then test the is_float() method.

```
Enter a float: 3rd
3rd is NOT a float.
```

```
Enter a float: 7.9
7.9 is a float.
```

12. Create a user defined method called is_integer(value) that simply tests if an input value is an integer. This method will accept a string input value as a parameter and return a True/False (Boolean) result. You get to code this one but the code will be very similar to is_float()

13. Add code to the main() method to test your is_integer() method. You get to code this.

14. Be sure to save and test your is_integer() method. Test with both invalid and valid data.

15. In lab 5, you wrote code to search an array for a value. This is actually a quite common programming task. Do you want to code this over and over every time you need to search an array? Let's code a search method we can reuse! In your Methods file, create a user defined method called find_string() to find a string value in a Python list. This method will accept 2 parameters. The first will be a string list to search and the second parameter will be the value to find. It will return the subscript where the value was found or -1 if the value was not found. You are given the code for this one.

```
# Step 15 find_string() method
def find_string(the_list, find_val):
    list_len = len(the_list) # get the list size
    found = False
    i = 0
    idx = -1
    while i < list_len and found == False:
        # convert both strings to upper case for the comparison
        if find_val.upper() == the_list[i].upper():
            idx = i
            found = True
        # end if
        i += 1
    # end while

    return idx
# end find_string
```

16. In the main() method, add some code to test your find_string() method. You are given the code.

```
# test find_string() method
# Create a list of random stuff and let's vary the letter case to
# test our string comparison
my_list = ["goat", "ZEBRA", "HorsE", "foX", "dog", "Cat"]
value = input("Enter a value to find: ")

# call our method
idx = m.find_string(my_list, value)
# Did we find what we were looking for?
if idx >= 0:
    print("Found " + value + " at index " + str(idx))
else:
    print( value + " NOT FOUND!")
```

17. It's time to test. Test with both valid and invalid input values to make sure the method returns the correct index or -1 if not found. Since we are comparing the string values in find_string() using upper case, the letter case doesn't matter on the input. What does matter is that the word is spelled correctly!

```
Enter a value to find: horse
Found horse at index 2
```

```
Enter a value to find: house
house NOT FOUND!
```

18. In your Methods file, create a user defined method called find_int() to find an integer value in a Python list. This method will accept 2 parameters. The first will be an integer list to search and the second parameter will be the value to find. It will return the subscript where the value was found or -1 if the value was not found. You get to code this one. Hint: The code from step 15 is quite similar. There is also code from Lab 5 that will help if you need it. **Write a flowchart or pseudocode for this method**.

19. In the main() method, add some code to test your find_int() method. You get to code this although the code will be very similar to the code in step 16. Use a hard-coded integer list (at least 6 values) and prompt the user for the value to find. Display the subscript of the found value and a “Not found” message if the value was not found.

20. Test your find_int() method with both valid and invalid values.

## Exercise 2

Coyote Tours has hired you to develop an application that will provide the user the cost of a sightseeing tour. The user will enter a tour code number and number of persons. Your application will look up the destination, and cost per person for that destination and compute the total price. Tour codes, destination and cost per person will be stored in parallel arrays.

total_price = cost_per_person * number_of_people

You will import the Methods module and use the find_int() method like you did in exercise 1.

You are given the following table of tour codes. Use your creative talents to fill in the destinations and costs.

| Tour Code | Destination | Cost per Person |
|---|---|---|
| 1001 | | |
| 1002 | | |
| 1003 | | |
| 1004 | | |
| 1005 | | |

If the user types in an invalid tour code, display “NOT FOUND” for the destination and 0 for the cost.

### Steps

1. Document your design with either pseudocode or a flowchart. If you are doing pseudocode, you may add it to the pseudocde file you created in exercise 1.

2. Create a new Python file Lab6Lookup. Make sure you are saving it to your Lab6 folder on your USB drive if you are using the college lab computers.

3. Code the application. Create a main() method and write your application code in it like you did for exercise 1.

4. Test your application and verify lookups and computations are correct.

## Grading Criteria:

| Deliverable | Points | Breakdown |
|---|---|---|
| Exercise 1 flowchart or pseudocode | 5 | Completed for the find_int() method, logic “makes sense”, neat, easy to follow. Code follows logic. If flowchart, proper use of symbols, flowlines, etc. |
| Exercise 1 Code | 10 | Complete, “makes sense”, arrays, loops and selection structures properly implemented. Proper use/naming of variables and constants. Appropriate use of comments |
| Exercise 1 run | 10 | Loads, compiles, runs, produces correct results. |
| Exercise 2 flowchart or pseudocode | 5 | Completed for the findItem() method, logic “makes sense”, neat, easy to follow. Code follows logic. If flowchart, proper use of symbols, flowlines, etc. |
| Exercise 2 Code | 10 | Complete, “makes sense”, arrays, loops and selection structures properly implemented. Proper use/naming of variables and constants. Appropriate use of comments |
| Exercise 2 load/run | 10 | Loads, compiles, runs, produces correct results |
| **Lab Total** | **50** | |