**Lab 5**This lab is designed to practice skills in Chapter 5 on functions. It has several parts. It may be easier to write each part as a separate python program. After each part is completed, copy / paste it into one menu-driven loop. The menu-driven loop program is provided as startup code.

The file **lab5\_4ed\_startup\_code.py** is a running python program. It is a “shell” program containing a menu-driven loop. It will eventually contain all parts. When finished, rename lab5\_4ed\_startup\_code.py to DDHH\_L5\_lastname.py. Other startup files are provided which you have to download.

**Part 0** is an introduction to functions. Functions are very versatile and are used in many different ways. This part describes 10 function "design patterns". Each design pattern is a different way to use functions. The generic name for these functions are: f1, f2, f3,... f10, f10a, f10b, f10c.

These are the 10 function design patterns to try:

1. define and call 1 function f1 with no parameters, no return
2. define and call 1 function f2 with 1 parameter, no return.
3. define and call 1 function f3 with 2 parameters, no return.
4. define and call 1 function f4 with 2 parameters, 1 return.
5. define and call 1 function f5 with 2 parameters, 2 returns.
6. define and call 1 function f6 with no parameters, 1 return.
7. define and call 1 function f7 with no parameters, 2 returns.
8. define and call 1 function f8 with 3 keyword parameters, 1 return.
9. define and call 3 functions: f9, f9a, f9b. f9 calls f9a and f9b. All 0 parameters, 0 return.
10. define and call 4 function f10, f10a, f10b, f10c. f10 calls f10a, f10a, f10b, f10c. All 0 parameters, 0 return.

For each, you write code to demonstrate the design pattern. For example, for design pattern 1:  
“define and call 1 function f1 with no parameters, no return”

You can write:

def f1\_greeting():

print('Hello and welcome to my world.')

f1\_greeting()

The generic name: f1, has been extended to indicate the purpose of the function.

You can write any code that fits the design pattern. It should “make sense” for the pattern. Below are ideas for each pattern. You can use any of the ideas provided or make your own. Be sure your code fits the design pattern. Extend the generic name to indicate how the function should be used. Your testing should display some results to the screen to demonstrate the function was tested.

Ideas for the 10 function design patterns:

1. f1\_greeting(): print a short greeting.
2. f2\_hello(***name***): print hello to ***name***.
3. f3\_taunt(***boy***, ***girl***): print: There is ***boy*** and ***girl***, sitting in a tree; K-I-S-S-I-N-G.
4. f4\_average(a, b): compute (a+b)/2, test with: print(‘The average score is’, f4\_average(76, 88)).
5. f5\_min\_max(a, b): return the minimum and maximum values.
6. f6\_my\_name(): return your full name.
7. f7\_first\_last: return your last name and first name.
8. f8\_combine\_into\_100s(huns=3, tens=2, ones=1), returns 321 by default.  
   See page 233-235 (4ed) for Keyword Arguments.
9. f9\_place(), f9a\_city(), f9b\_state(). f9\_place() calls f9a\_city() and f9b\_state(); displays: “Austin, TX”.
10. f10\_face(), f10a\_eye(), f10b\_nose(), f10c\_mouth(). f10\_face() calls f10a\_eye(), f10a\_eye(), f10b\_nose(), f10c\_mouth(); displays:  
      
    (o) (o)  
     <=>  
    \\-----//  
      
    Here, f10\_face() calls the other functions and provides extra spacing to complete the picture. f10a\_eye() displays: (o), f10b\_nose() displays: <=>, and f10c\_mouth() displays: \\-----//.

Having practiced several ways to use functions, the next parts should be a little easier.

**Parts 1 to 5:**

Attached are several startup files. They will be combined into one file featuring a menu driven loop.  
 **lab5\_4ed\_startup\_code.py** a bare minimum menu driven loop (startup code used to hold all parts)  
 **launch.py** launch a rocket (startup code for part 1)  
 **tip\_table.py** compute a table of restaurant payment options (startup code for part 2)  
 **scope.py** follow data into and out of several functions (startup code for part 3)  
 **<sort 3 numbers>** no startup code, see instructions below

**Part 1** is a launch option. It works, but is not written very well. The programmer did not use functions, resulting in a lot of redundant code. Your job is to gather the common code together into one or more functions. Then eliminate any unneeded extra code. Use parameters to make your functions flexible.

**Part 2** is a tax and tip table option. Reduce program size by calling one function several times to create a table. You will have to fix (debug) the initial function provided which has problems (incorrect results).

**Part 3** is a data passing exercise to practice passing data into and out of multiple nested functions. Each function gets some data from the “outside” (the caller). Inside the called function this becomes a copy of a variable, now called a "local" variable. You write several functions for this part; each function is small.

**Part 4** uses a group of simple functions working together to do a more difficult task. The overall goal is to sort a list of three numbers. To do this, use three functions. Each function is defined with an IPO (Input, Processing, and Output) chart.

**more\_than(a, b)** : Inputs: integers a and b; Processing: check if a is greater than b; Output: True if a > b, False otherwise. Example: more\_than(27, 53) returns: False

**swap(a, b):** Inputs: integers a and b; Processing: swap a and b; Output: b and a. Example: swap(27, 53) returns: 53, 27

**sort3(a, b, c)** Inputs: integers a, b, and c; Processing: sort a, b and c into order from smallest to largest by calling the functions more\_than() and swap() as needed At most, you may need to swap three times. Outputs: a, b and c in sorted order. Example: sort(34, 12, 78) returns: 12, 34, 78

Make sure that your sort3 function works in all possible cases. For example, it sorts: 1, 2, 3; 3, 2, 1; 1, 3, 2; 2, 3, 1; 2, 1, 3; 1, 1, 1; 1, 2, 2; 2, 1, 1; etc. into 1,2,3. Take no chances!

Make a loop to test your sort() function 10 times. Use randint() random numbers. You will have to import random to get the function randint(). You do not have to write randint. Call randint to generate three random numbers (a, b and c) between 1 and 100. With these variables, call sort3(a, b, c) and return min, mid, max. Display the random numbers, followed by the same numbers, but sorted.

Example output:  
1) random numbers 54, 98, 9 sort to: 9, 54, 98  
2) random numbers 23, 12, 34 sort to: 12, 23, 34

3) random numbers 77, 81, 88 sort to: 77, 81, 88  
…  
10) random numbers 67,60, 2 sort to: 2, 60, 67

You can develop this to be called in the function main. When complete the function main becomes the function sort in the main menu-driven loop.

**Summary:** The individual programs created for individual parts 0, 1, 2, 3, and 4 become the functions intro(), launch(), tip\_table(), scope() and sort(). The single resulting program will be one large, all-in-one program.

**Part 5, Extra Credit:** Turtle graphics: Create two functions: one draws a letter A, another draws a letter C. These functions should input three variables: location\_x, location\_y, and height. Draw the letters A, C, C by calling the letter\_A(x, y, h) function once, and the letter\_C(x, y, h) function twice. Provide values for x, y, h so that the “ACC” appears in an appropriate location. Draw all letters in an area not to exceed a 400 x 400 pixel box. A size of about 100 x 100 is about right for each letter.