Lab 6 – The Function of Functions

The Way of the Program - Fall 2023

Part 1: Building a Town

1. Download and unzip **lab06_files.zip** from our class web page (under Labs). Open the file **startcode.py** in IDLE and run `house()`, which creates a new GraphWin window and draws a pink house in it 200 pixels wide and 100 pixels high, with the bottom left corner at (50, 300). We will use this code as our starting point to develop a program to draw an entire "town" containing many houses of different shapes, colors, and sizes.

2. The `house` function does two things: (1) it creates a GraphWin, and (2) it draws a pink house at (50, 300). We will start by breaking this function into two separate functions called **`town`** and **`drawHouse`** that together do exactly the same thing as `house`. The `town` function is already written for you, but you need to complete the `drawHouse` function, which takes six parameters: *win*, *x1*, *y1*, *houseWidth*, *houseHeight*, and *houseColor*. The *win* parameter will be a GraphWin object in which to draw a house of the specified width, height, and color at location (*x1*, *y1*). To complete `drawHouse`, cut-and-paste the code that appears below the dashed line in `house` directly into `drawHouse`. The `town` function has no parameters. It simply creates a GraphWin object and then call `drawHouse` with the appropriate values to draw the house.

3. Your `drawHouse` function can now be used as a building-block to draw houses of arbitrary sizes and locations. Add two more houses to the scene by adding two more lines of code to `town`, one that draws a white house 60 pixels wide and 80 pixels high at location (275, 225), and one that draws a salmon-colored house 50 pixels wide and 30 pixels high at location (150, 100).

4. Modify the `town` function so that it uses a for-loop to draw 30 houses of random sizes and colors at locations specified by the user repeatedly clicking the mouse in the window. Choosing house widths and heights from a random range of 20-50 pixels tends to give good results. For the colors, try picking randomly from a list of earth-tone colors such as "white", "pink", "salmon", "bisque", "coral", "tan", and "wheat".

5. Now change the width of the GraphWin from 400 to 800 pixels, to make more room for houses. Also modify `town` so that all houses are drawn automatically instead of using mouse clicks. All houses should be drawn along the bottom edge of the window. To do this, just make each house's *y* coordinate be 300 (the height of the window). The *x* coordinate can be chosen randomly from the range 0-800. Your program should now draw an entire "street" of randomly arranged houses along the bottom of the window.

6. Next, define a new function called **`drawStreet`** that draws an entire horizontal "street" of houses, all with the same *y* coordinate value for the house base. You should move most of your code from `town` directly into your new `drawStreet` function. This function should take three parameters: *win*, *y*, and *numHouses*, which specifies the number of houses to draw along the horizontal "street". Rewrite `town` so that it uses `drawStreet` to draw one street at *y* coordinate 300 and another one at 225. Each street should contain 30 houses each.

7. Finally, modify `town` so that it uses a for-loop to draw several streets at regular intervals from the top of the window to the bottom. One approach is to first divide the total pixel height of the window by the number of desired streets, which will give the amount to increment the *y* value by on each loop cycle. Drawing the streets starting from the top of the window and going down will give better results than starting from the bottom and going up, because the houses in the "foreground" at the bottom will appear in front of the houses in the "background" at the top. What number of streets seems to give the best-looking results overall?

Part 2: Returning Values From Functions

For these exercises, you can use the autotester by typing `test( function_name )` *at the Python prompt.*

8. Another important use of functions is to return values to other functions, which may use the values as part of some bigger computation. For example, the function below computes and returns the factorial value *n*!

```
def factorial(n):
    product = 1
    for num in range(1, n+1):
        product = product * num
    return product
```

The mathematical function choose(*n*, *k*) is defined as *n*! / (*k*! × (*n* – *k*)!). For example, choose(10, 3) equals 10! / (3! × 7!), or 120, and choose(6, 4) equals 6! / (4! × 2!), or 15. Write a function called **choose(*n*, *k*)** that takes parameters *n* and *k* as input and uses the above `factorial` function as a "helper" to compute the value of choose(*n*, *k*). Your function should use a `return` statement to return its answer, rather than printing it out.

9. Write a function called **addup(*values*)** that takes a list of values as an input parameter, and returns the sum of all of the values in the list. For example, calling `addup([1,2,3])` should return 6. Make sure that typing `2*addup(range(10))` at the Python prompt gives 90. If you get an error message that refers to "NoneType", you probably used a `print` statement instead of `return` in your function.

10. The *mean* of a list of numbers is simply the average of all of the numbers in the list, which is just the sum of the numbers divided by the length of the list. Define a function **mean(*values*)** that takes a list of values as an input parameter and uses your `addup` function as a "helper" to compute and return the mean value. For example, `mean([1,1,2,3])` should return 1.75, and `mean(range(10))` should return 4.5.

11. Write a function called **squares(*values*)** that takes a list of values as an input parameter and returns a new list containing the squares of the original values. For example, typing `squares([1,2,3,4])` should return the list [1, 4, 9, 16], and typing `mean(squares([1,2,3,4]))` should return 7.5.

12. Using your functions `addup`, `mean`, and `squares` as helpers, write a function called **standardDev(*values*)** that takes a single list containing two or more values as an input parameter and returns the *standard deviation* of the values. The standard deviation of the values $v_1, v_2, \ldots, v_N$ is defined to be:

$$\sqrt{\frac{(v_1 - m)^2 + (v_2 - m)^2 + \ldots + (v_N - m)^2}{N - 1}}$$

where *m* is the mean of the values, and *N* is the total number of values in the list.* Hint: first compute the mean value *m* by calling `mean(values)`. Then use a for-loop to build a new list of the *differences* between each v_i value and the mean *m*. You can then create a list of the *squares* of the differences by calling `squares(differences)`. Then add up these squares by calling `addup(squares(differences))`. For example, the standard deviation of the list [1,2,3,4] is a little over 1.29.

*Technically, this version of the definition is known as the *corrected sample* standard deviation.

13. Write a function called **flip(*s*)** that takes a string *s* as an input parameter and constructs and returns a new string that is the reversal of the input string. For example, `flip("apple")` should return (but not print) the string "elppa". Typing `flip("ah")*2` should return the string "haha". Then write another function called **palindrome(*s*)** that returns the string "yes" or "no" depending on whether *s* equals flip(*s*). For example, `palindrome("lever")` should return "no", whereas `palindrome("level")` should return "yes".