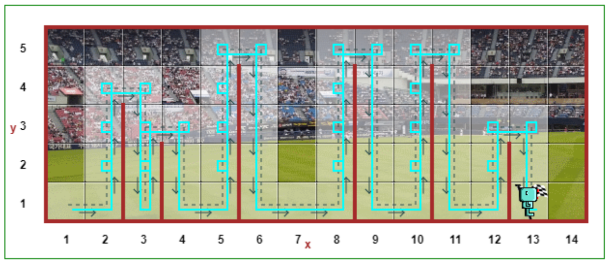
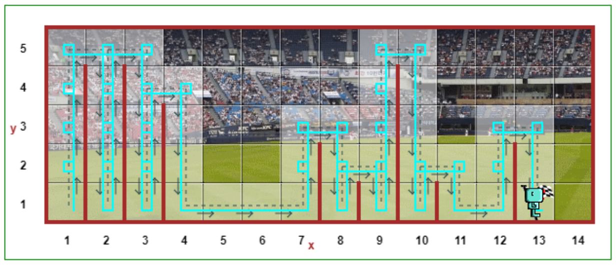
CS521 - Homework 5 - Alexander Bean

1. Reeborg's World

Hurdle 4 Challenge



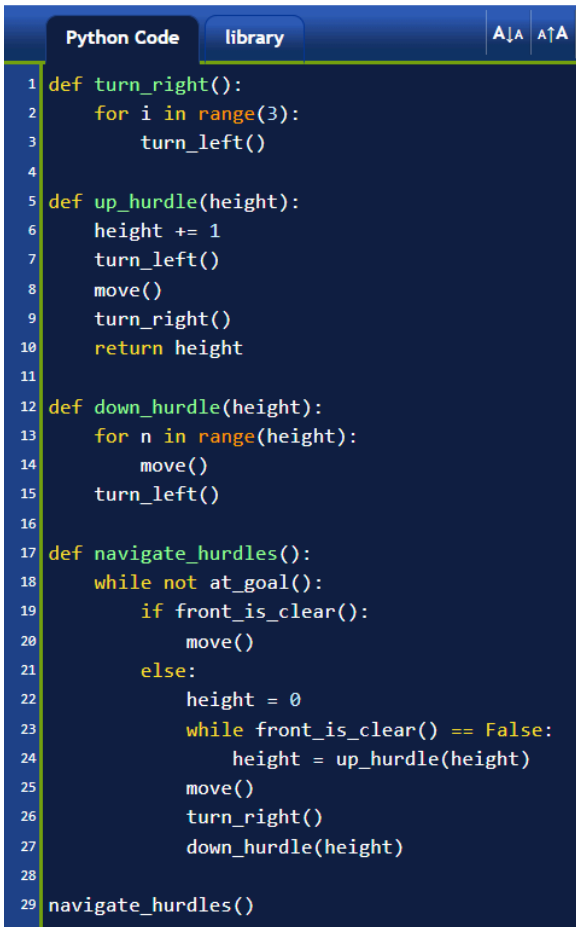


Since the hurdles are in variable locations with variable heights, I started off by altering the *navigate_hurdles()* function we created in class. I kept the if condition the same: If the front is clear, then Reeborg can move one step forward. However, I altered the else statement to make multiple new steps and call two new functions. The new functions are *up_hurdle()* and *down_hurdle()*. First, a height variable will be established to track how tall the hurdle is.

This variable will be added to when Reeborg climbs up and will be used when Reeborg needs to climb down. I added another while loop to iterate as long as the front is NOT clear. Then, I call the *up_hurdle()* function and pass in the height variable.

The *up_hurdle()* function and pass in the height variable as an argument. This function increases the height variable by 1 and makes Reeborg move one step upward and then face right. The function returns the new height value to track how many steps Reeborg has taken up the hurdle. Once Reeborg's front is clear, he will move, turn right, and face downwards. The *down_hurdle()* function will then be called and the height variable is passed in.

The *down_hurdle()* function is a loop that moves Reeborg an amount of steps equal to the height variable. Meaning if Reeborg had to go upwards 3 times, the height variable would be set to 3, and Reeborg would take 3 steps downwards and reach the ground. Once there, Reeborg turns left (facing right).



```python
def turn_right():
    for i in range(3):
        turn_left()

def up_hurdle(height):
    height += 1
    turn_left()
    move()
    turn_right()
    return height

def down_hurdle(height):
    for n in range(height):
        move()
    turn_left()

def navigate_hurdles():
    while not at_goal():
        if front_is_clear():
            move()
        else:
            height = 0
            while front_is_clear() == False:
                height = up_hurdle(height)
            move()
            turn_right()
            down_hurdle(height)

navigate_hurdles()
```

With this program, Reeborg is able to identify when he reaches a hurdle and determine how high he needs to jump to successfully jump the hurdle. When the program is run, Reeborg successfully traverses through the randomized Hurdle 4 maps and makes it to the goal.

Maze Challenge

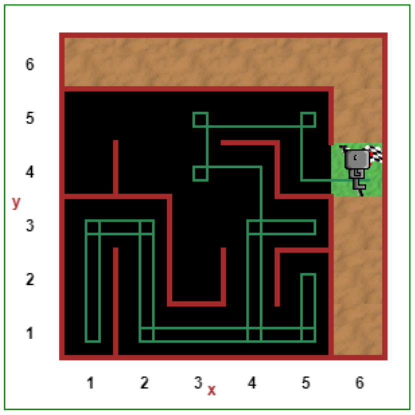
To solve this, I created an if-elif-else statement to have Reeborg follow the right side of the maze until reaching the exit. By following a specific side of a maze (right or left), you will always be able to find your way to the exit.

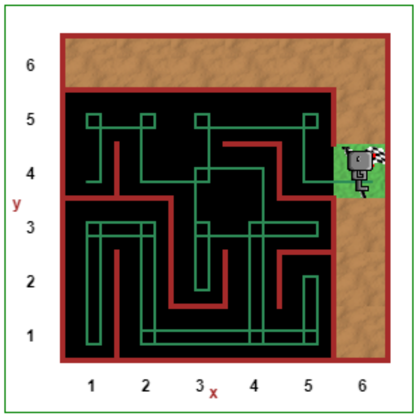
I used the *front_is_clear()* and *right_is_clear()* World functions to have Reeborg identify and follow the right side. The only other function used is the *turn_right*() function used in class.

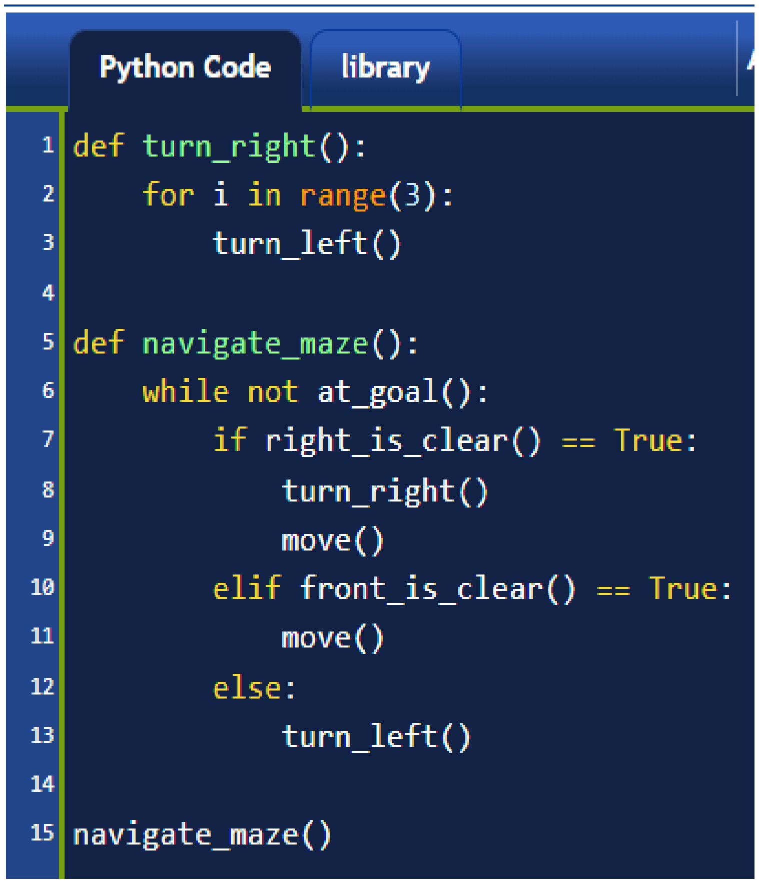
If the right side is clear, Reeborg will turn right and move. If the right is not clear but the front is clear, Reeborg will move forward. If the right side nor front are clear, Reeborg will turn left.

To keep Reeborg moving all the way to the goal, I placed the if-elif-else statement within a while loop. The condition is the same as Hurdle 4, and the sentinel is the at_goal() condition being False.

When executed, Reeborg will keep moving along the right wall of the maze until finding the goal.



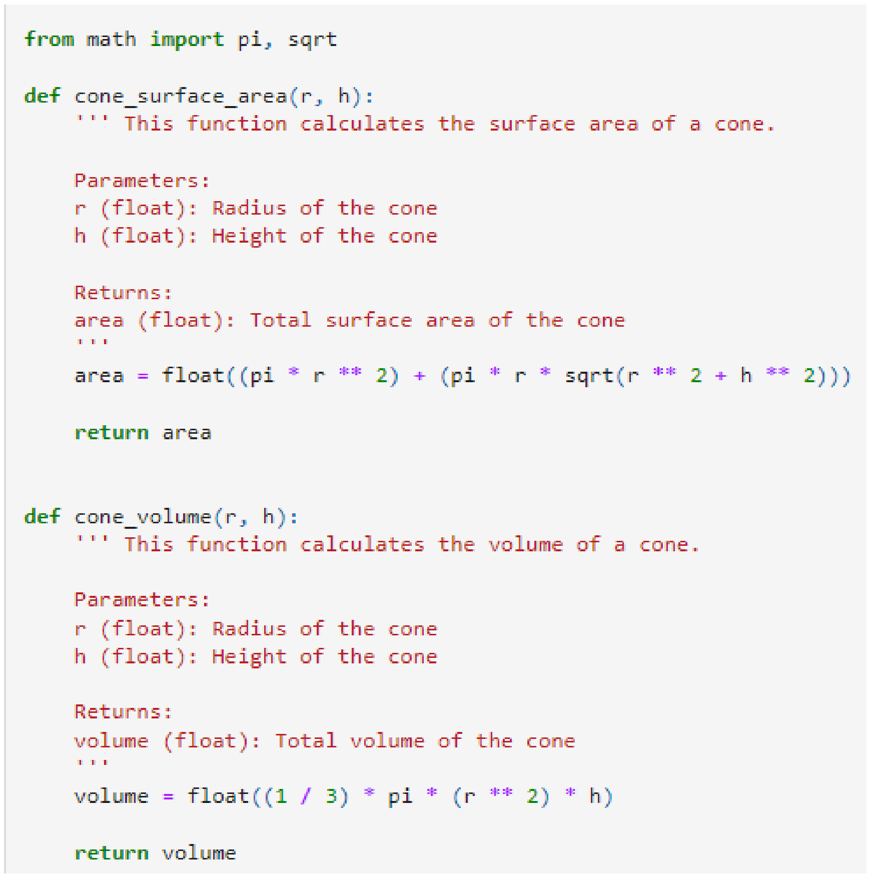




```
def turn_right():
    for i in range(3):
        turn_left()

def navigate_maze():
    while not at_goal():
        if right_is_clear() == True:
            turn_right()
            move()
        elif front_is_clear() == True:
            move()
        else:
            turn_left()

navigate_maze()
```

2. Functions with User Input Processing

Part 1:



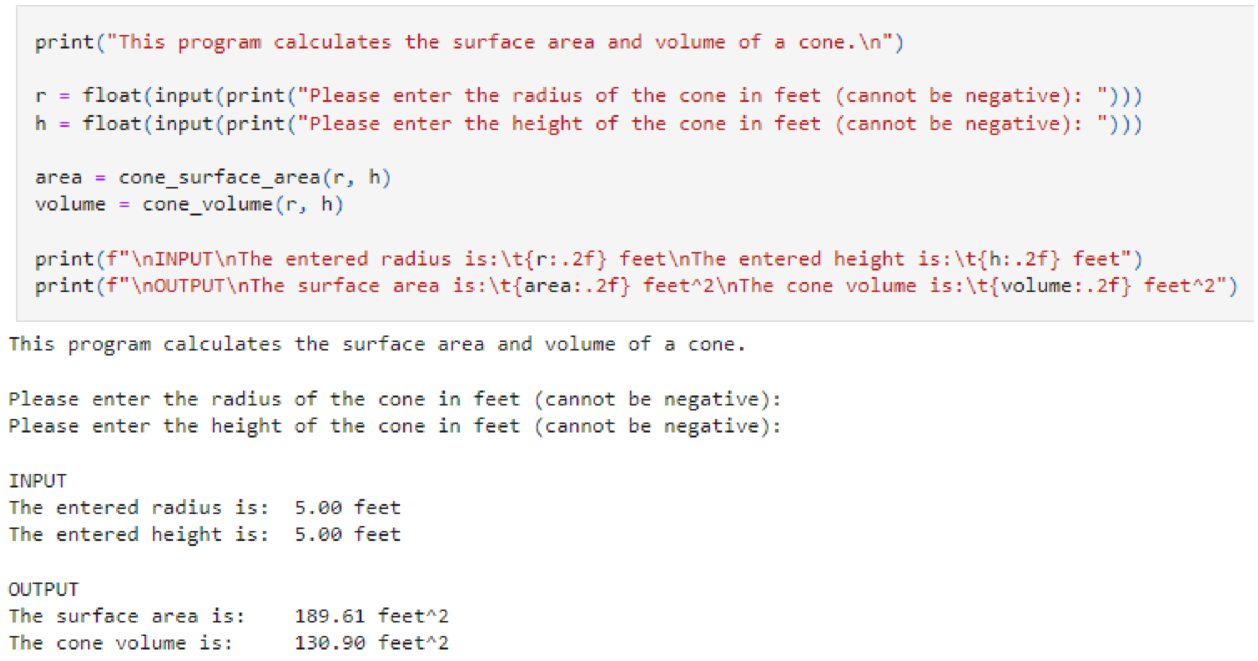
```
from math import pi, sqrt

def cone_surface_area(r, h):
    ''' This function calculates the surface area of a cone.

    Parameters:
    r (float): Radius of the cone
    h (float): Height of the cone

    Returns:
    area (float): Total surface area of the cone
    '''
    area = float((pi * r ** 2) + (pi * r * sqrt(r ** 2 + h ** 2)))

    return area


def cone_volume(r, h):
    ''' This function calculates the volume of a cone.

    Parameters:
    r (float): Radius of the cone
    h (float): Height of the cone

    Returns:
    volume (float): Total volume of the cone
    '''
    volume = float((1 / 3) * pi * (r ** 2) * h)

    return volume
```

I imported pi and sqrt() from the math module to use in the calculations. I specified to only import those from the math module, as I know those are the only vars/functions I need and there is no need to import the whole module.

Following the instructions, I defined two functions: cone_surface_area(r, h) and *cone_volume*(r, h). I added docstrings with a description of what the function does, what the parameters are, and what the function returns.

Each function has a calculation which uses two variables entered in the parameters–r for radius and h for height. This will need to be defined before the function is called within a program. While PEMDAS will handle the calculations correctly, I added redundant parentheses to make the calculations easily readable.

Once calculated, I converted the value to a float to be organized and formatted to two decimal places. Then the value is returned from the function.



```
print("This program calculates the surface area and volume of a cone.\n")

r = float(input(print("Please enter the radius of the cone in feet (cannot be negative): ")))
h = float(input(print("Please enter the height of the cone in feet (cannot be negative): ")))

area = cone_surface_area(r, h)
volume = cone_volume(r, h)

print(f"\nINPUT\nThe entered radius is:\t{r:.2f} feet\nThe entered height is:\t{h:.2f} feet")
print(f"\nOUTPUT\nThe surface area is:\t{area:.2f} feet^2\nThe cone volume is:\t{volume:.2f} feet^2")
```

```
This program calculates the surface area and volume of a cone.

Please enter the radius of the cone in feet (cannot be negative):
Please enter the height of the cone in feet (cannot be negative):

INPUT
The entered radius is:  5.00 feet
The entered height is:  5.00 feet

OUTPUT
The surface area is:    189.61 feet^2
The cone volume is:     130.90 feet^2
```

To get the radius and height, I asked for inputs from the user. I converted the entered values to floats and assigned them to variables for the calculations.

Then, I called the *cone_surface_area()* and *cone_volume*() functions and entered the radius and height as arguments in the parameters. Now, the functions have numbers to use in the calculations to determine surface area and volume.

Once the values are returned, we now need to format the output to 2 decimal places and return to the user. However, when attempting to implement the requested 2 decimal formatting, I ran into a formatting issue. When using the round() function, I found that it would not properly round some numbers to 2 decimal places. For integer values, it would only round to one decimal place and not properly work with trailing zeroes. (For example with the output above is entered, an input of 5 and an output of 130.9-. When using round(5, 2), it makes 5.0, and using round(130.9, 2), it does not change).

To resolve this, I used the format() function and used a format specifier {:.2f} to fix the decimal notation to 2 decimal places. I have used this practice in my undergrad and at my current job and found it to be the most applicable here. As a result, the values will always display 2 decimal places and produce the expected result.

Part 2:

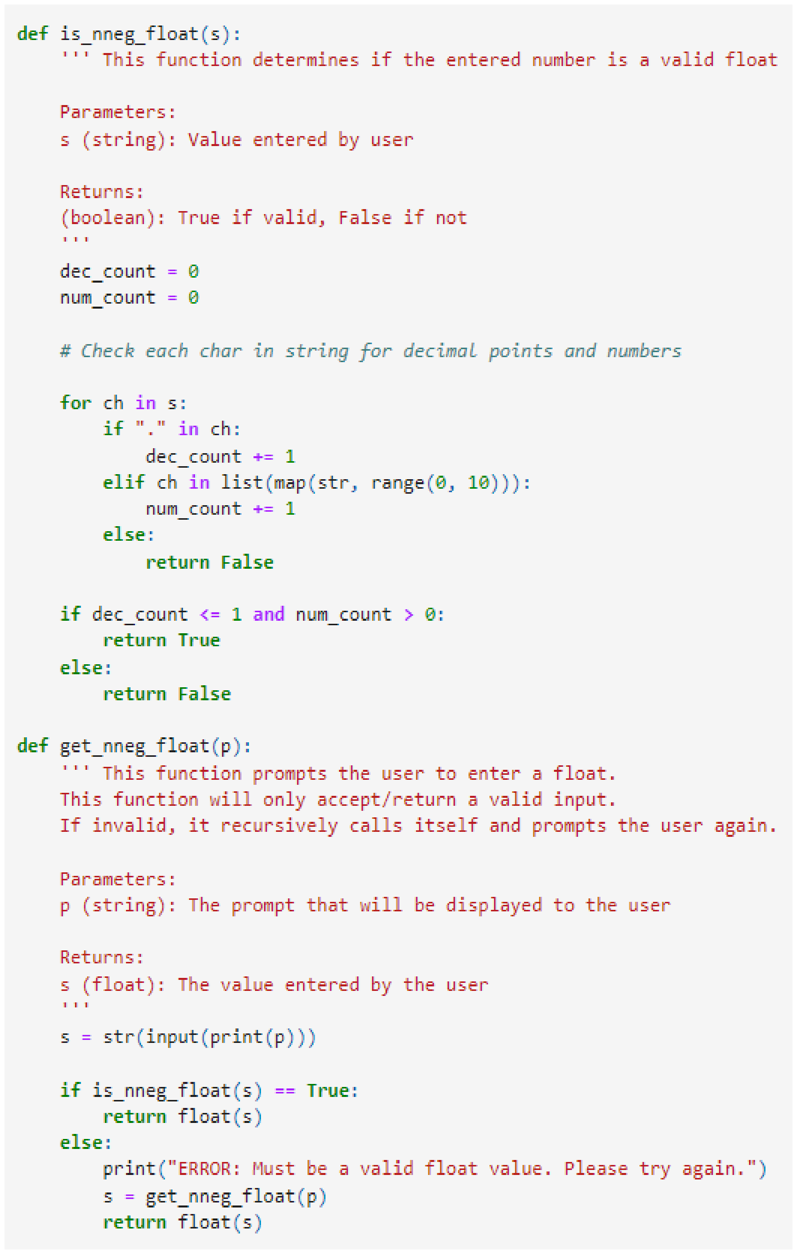
For *is_nneg_float*(s): I created *dec_count* and *num_count* to track if the entered string meets the criteria (no more than 1 decimal place and more than 1 number).

The for loop iterates through the string and checks each character. If the character is ".", the decimal count will increase by 1.

To determine if the character is a number,I used map() to convert a range of numbers (effectively 0-9) to strings, and then I placed them all in a list. I can then compare the character to a list of numbers as strings. If the character is in the list, then it is a number and the number count increases by one

In the else statement–if the character is not a number or a decimal point, the function returns false since it's not a decimal or number.

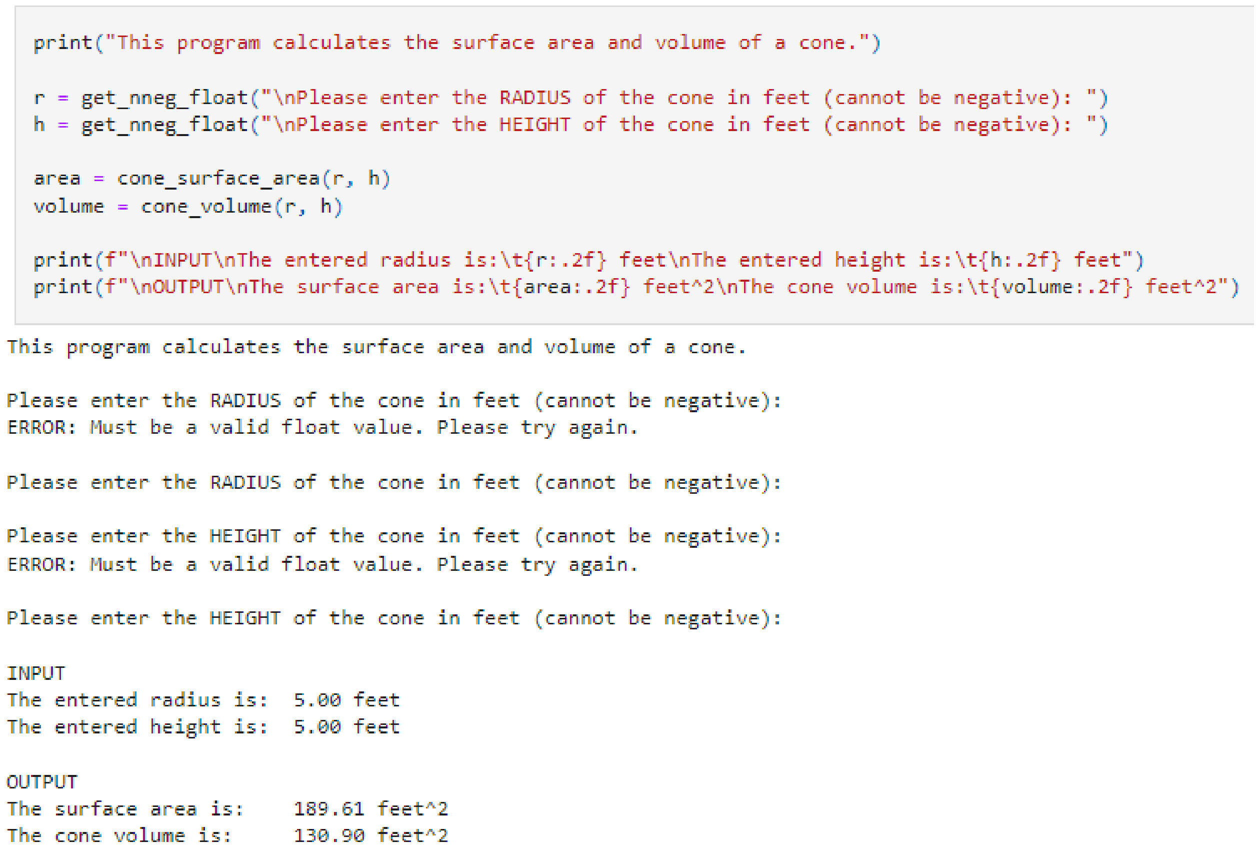
Once the entire string has been checked, it will then see if the decimal count is no greater than 1 and if the number count is greater than 0. If both of those conditions are met, then it is a valid float and returns True. If not, it is NOT a valid float and returns False.



```python
def is_nneg_float(s):
    ''' This function determines if the entered number is a valid float

    Parameters:
    s (string): Value entered by user

    Returns:
    (boolean): True if valid, False if not
    '''
    dec_count = 0
    num_count = 0

    # Check each char in string for decimal points and numbers

    for ch in s:
        if "." in ch:
            dec_count += 1
        elif ch in list(map(str, range(0, 10))):
            num_count += 1
        else:
            return False

    if dec_count <= 1 and num_count > 0:
        return True
    else:
        return False

def get_nneg_float(p):
    ''' This function prompts the user to enter a float.
    This function will only accept/return a valid input.
    If invalid, it recursively calls itself and prompts the user again.

    Parameters:
    p (string): The prompt that will be displayed to the user

    Returns:
    s (float): The value entered by the user
    '''
    s = str(input(print(p)))

    if is_nneg_float(s) == True:
        return float(s)
    else:
        print("ERROR: Must be a valid float value. Please try again.")
        s = get_nneg_float(p)
        return float(s)
```

For *get_nneg_float*(p): I passed the prompt argument into the function and the prompt text is used to ask the user to enter a float. The value is assigned to *s* for string. Then, I call the *is_nneg_float* function with the entered string as the argument. If it returns True, then the string is a valid float, is converted to float, and returned. If it's false, an error message is displayed and the *get_nneg_float*(p) is recursively called to prompt the user again for a valid input.

I directly assigned the recursive call to the *s* variable so, when the user enters a valid float, the valid number is assigned and returned. Otherwise, the initial input will be returned even though it is invalid.



```python
print("This program calculates the surface area and volume of a cone.")

r = get_nneg_float("\nPlease enter the RADIUS of the cone in feet (cannot be negative): ")
h = get_nneg_float("\nPlease enter the HEIGHT of the cone in feet (cannot be negative): ")

area = cone_surface_area(r, h)
volume = cone_volume(r, h)

print(f"\nINPUT\nThe entered radius is:\t{r:.2f} feet\nThe entered height is:\t{h:.2f} feet")
print(f"\nOUTPUT\nThe surface area is:\t{area:.2f} feet^2\nThe cone volume is:\t{volume:.2f} feet^2")
```

```
This program calculates the surface area and volume of a cone.

Please enter the RADIUS of the cone in feet (cannot be negative): 
ERROR: Must be a valid float value. Please try again.

Please enter the RADIUS of the cone in feet (cannot be negative): 

Please enter the HEIGHT of the cone in feet (cannot be negative): 
ERROR: Must be a valid float value. Please try again.

Please enter the HEIGHT of the cone in feet (cannot be negative): 

INPUT
The entered radius is:  5.00 feet
The entered height is:  5.00 feet

OUTPUT
The surface area is:    189.61 feet^2
The cone volume is:     130.90 feet^2
```

After creating the new functions, I altered the program to use *get_nneg_float* to retrieve and validate the user input for the radius and height. Compared to the original program, I don't need to convert the value returned by *get_nneg_float*() to a float, as it is already converted within the function.

The prompts are entered as arguments, which are then used by the function to request an input from the user. This way, the *get_nneg_float*() function can be used for any scenario where you may need a float from a user and not strictly for a radius or height. Any prompt can be used to request a value.