Python Lab 2.2: Functions - Repetition - Validation

> The purpose of this practice is to help you apply the concepts discussed up to **week 2**:
>
> - obtain user input
> - define functions that accept parameters and return values
> - call functions
> - repeat code
> - validate user input
> - handle exceptions

In `lab2_2.py` in the text editor at top-right, write a program which will:

1. Define an add() function:
    - The function should accept one parameter called `x`, and return the `total`
    - total is initialized to 0
    - The main part of the function should repeat the steps below `x` times:
        - ask the user for a floating point number, using the prompt: `Type a number to add:`
        - validate that the user types a number - if not, display `Wrong input` and quit
        - add the number to the total
        - increment the count variable to keep track of how many times the loop executes (if you use a while loop)
2. The main program should:
    - Ask the user how many numbers they want to add and assign user input to a variable called `numbers`
    - Validate that the user types an integer number - if not, display `Wrong input` and quit
    - Call the add function by passing `numbers` as an argument. Assign the value returned from the function to the sum variable
    - Display the `sum`

Some Technical Details: A Basic Python main()

In some Python scripts, you may see a function definition and a conditional statement that looks like the example below:

```
def main():
    print("Hello World!")

if __name__ == "__main__":
    main()
```

In this code, there is a function called main() that prints the phrase Hello World! when the Python interpreter executes it. There is also a conditional (or if) statement that checks the value of **name** and compares it to the

string "**main**". When the if statement evaluates to True, the Python interpreter executes main().

In this program write your code to accept user input in def main():

{% next %}

{% spoiler "Hint 1: Modify the Function to accept a parameter `x` " %}

```
def add(x):
    total = 0
    count = 0
    # repeat the steps below "x" times:
        # ask the user for a number
        # validate that the user types a number - if not display 'Wrong input' and
quit
        # add the number to the total
        # increment the count variable

    return total
```

{% endspoiler %}

{% spoiler "Hint 2 : Validate user input" %}

use a try/except block after reading the score which will include the conversion to a floating point number... and exit the program if the input is invalid.

```
num = input("Type a number to add: ")
# validate that the user types a number - if not display 'Wrong input' and quit
try:
    num = float(num)
except:
    print('Wrong input')
    quit()
```

{% endspoiler %}

You can solve this problem using a while loop with a count variable!

{% spoiler "Hint 3a : repeat the code x times - using a while loop" %}

```
def add(x):
    total = 0
    count = 0
    while count < x:
        # ask the user for a number
        num = input("Type a number to add: ")
        # validate that the user types a number - if not display 'Wrong input' and
quit
```

```
        try:
            num = float(num)
        except:
            print('Wrong input')
            quit()
        # add the number to the total
        total = total + num
        # increment the count
        count = count + 1
    return total
```

{% endspoiler %}

You can solve this problem using a for loop with a range!

{% spoiler "Hint 3b : repeat the code x times - using a for loop" %}

```
def add(x):
    total = 0
    for count in range (x) :
        # ask the user for a number
        num = input("Type a number to add:")
        # validate that the user types a number - if not display 'Wrong input' and
quit
        try:
            num = float(num)
        except:
            print('Wrong input')
            quit()
        # add the number to the total
        total = total + num
        # increment the count
    return total
```

{% endspoiler %}

{% spoiler "Solution: Main program - Call the add function with validated user input..." %}

```
# This is the main program function
def main():
    try:
        times = int(input ("How many numbers do you want to add? "))
    except:
        print("Wrong input")
        quit()

    sum=(add(times))
    print(sum)
```

```
#DO NOT MODIFY THIS CODE
if __name__ == '__main__':
    main()
```

{% endspoiler %}

{% next %}

Execute your program

Remember in order to execute your code you type in the terminal:

```
python lab2_2.py
```

Use the following test data to make sure your program produces correct resutls.

TEST 1:

How many numbers to you want to add: 3

Type a number to add:5

Type a number to add:5

Type a number to add:5

15.0

TEST 2:

How many numbers to you want to add: a

Wrong input

TEST 3:

Enter your score: 3

Type a number to add:5

Type a number to add:a

Wrong input

{% next %}

Check Your Code

Execute the below to evaluate the correctness of your code using `check50`, but be sure to test it yourself using:

- [x] 0 as numbers to add
- [x] -2 as numbers to add
- [x] 3 as how many numbers to add, and 5,5,5 as data
- [x] a as numbers to add
- [x] 2 as numbers to add, and a as data 🎉

```
check50 mkotsovoulou/ods6001a/main/labs/lab2_2
```

Execute the below to evaluate the style of your code using `style50`.

```
style50 lab2_2.py
```

{% next %}

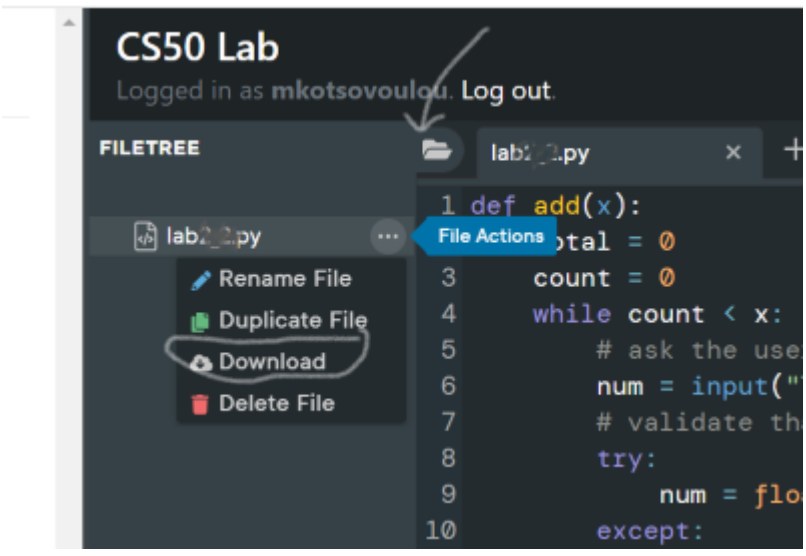
Submit your code

Execute the command below, logging in with your `GitHub username` and `Personal Access Token` when prompted. For security, you'll see asterisks (`*`) instead of the actual characters in your token.

If you do not have generated a Personal Access ToKen follow the instructions: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

```
submit50 mkotsovoulou/ods6001a/main/labs/lab2_2
```

You can re-submit your solution as many times as you want. When you are happy with your solution, download the code and upload it to Canvas.



Done!