Introduction to modules and functions

Why is this important?

So far in the course, we've been writing all of our code in one file. This is fine for small programs, but as your programs get larger and more complex it becomes harder to manage. This is where modularization comes in. Modularization is the process of breaking up your code into smaller, more manageable pieces.

Functions make your code more modular as you break your code up into smaller pieces that perform a specific task.

Breaking your code into functions makes your code easier to read, more understandable, and debug.

What are we going to do?

We're going to modify a few of our old programs to make them a bit more modular.

- First, we're going to modify our `heads_or_tails_functions.py` program to make it more modular.

Steps

1. Let's take a look at `heads_or_tails_functions.py` and think of some of the functions we can create here.

- Our file here is doing a few things:
  - It's getting the user's input
  - It's generating a random number (simulating a flip of a coin)
  - It's comparing the user's input to the random number (simulating a flip of a coin)
- Let's say we wanted to create a few functions here

2. Let's create a function to get the user's input.

- Below we are defining a function called `get_user_guess` that will get the user's input, and return it from the function

```python
import random

def get_user_guess():
    user_guess = input("Guess the coin flip! Enter heads or tails (h/t): ")
    return user_guess

# other code removed for brevity
```

- Let's take a look at what we have here.

- `def` is the keyword we use to define a function. `get_user_guess` is the name of the function. `()` is where we put the arguments that we want to pass to the function (we'll get to that later). `:` is the end of the function definition.
- `user_guess = input("Guess the coin flip! Enter heads or tails (h/t): ")` is the code that we want to run when we call the function.s
- `return user_guess` is the value that we want to return from the function. In this case, we want to return the user's input.

- Notes:
  - We haven't used this function yet. We've only defined it. In the next step, we'll call the function.
  - You don't need a return in all functions. You can have a function that just does something and doesn't return anything.

3. Let's call the function we just created.

- let's use our function. You can see that the heads or tails is a black box. We don't need to know what's going on inside of it we just need to use it.

```
import random

def get_user_guess():
    user_guess = input("Guess the coin flip! Enter heads or tails (h/t): ")
    return user_guess

# heads will be 0 and tails will be 1
random_number = random.randint(0, 1)

user_guess = get_user_guess()

# other code removed for brevity
```

- You can see that when we call `get_user_function` we don't need to pass any arguments to it, but we need to use `()` to call the function.
  - This is going to execute the lines inside of the function definition.

4. Let's create a function to generate a coin flip.

- we're going to change file to create a function that is going to generate a coin flip for us.

```
import random

def get_coin_filp():
    random_number = random.randint(0, 1)
    if random_number == 0:
        print("The coin flip was: heads")
        return "h"
    else:
        print("The coin flip was: tails")
        return "t"

def get_user_guess():
    user_guess = input("Guess the coin flip! Enter heads or tails (h/t):
")
    return user_guess

# heads will be 0 and tails will be 1
random_number = random.randint(0, 1)

user_guess = get_user_guess()

# other code removed for brevity
```

- Let's take a look at what we have here.
  - `def` is the keyword we use to define a function. `get_coin_filp` is the name of the function. `()` is where we put the arguments that we want to pass to the function (we'll get to that later). `:` is the end of the function definition.
  - `random_number = random.randint(0, 1)` gets `0` or `1` randomly.
  - `if random_number == 0:` is the code that we want to run if the random number is 0.
    - `return "h"` is the value that we want to return from the function. In this case, we want to return the string "h".
  - `else:` is the code that we want to run if the random number is not 0.
    - `return "t"` is the value that we want to return from the function. In this case, we want to return the string "t".
- This is a bit different than the code we already have here. We're going to return "h" or "l".

5. Let's change our code to use the function we just created.

- Let's use our function to generate a coin flip.

```
# function definitions removed for brevity

user_guess = get_user_guess()

random_flip = get_coin_filp()

if (random_flip == user_guess):
    print("you guessed correct!")
else:
    print("you guessed wrong!")
```

- You can see that when we call `get_coin_filp` we don't need to pass any arguments to it, but we need to use `()` to call the function.
  - This is going to execute the lines inside of the function definition.

6. We're going to add one more piece to our program to make it work when we import our code to another file OR when we run it directly.

- We're going to do this by putting our code in an indented block under `if __name__ == "__main__":`

```
# functions removed for brevity

# our code is going to run the function below this line.
if __name__ == "__main__":
    user_guess = get_user_guess()

    random_flip = get_coin_filp()

    if (random_flip == user_guess):
        print("you guessed correct!")
    else:
        print("you guessed wrong!")
```

- This will execute your code when the file runs as a script, but not when it’s imported as a module.
  - We haven't talked about importing modules yet, but we will soon.
    - This is going to be when we define our function

Conclusion

You can see in this example how we can use functions in our code to make it more modular. We can break our code up into smaller pieces that perform a specific task. This makes our code easier to read, more understandable, and debug.