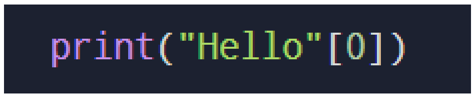
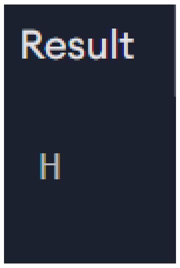
DAY 2: Data Types and How manipulate strings

We're going to explore some of the most important and some of the most basic data types, such as Strings, Integers, Floats, and Booleans. We already learn about strings And we always know that strings, when we create them, we have to create them with these double quotes around.
Now, because this is a string of characters we can actually pull out each character individually. So we could, for example, instead of just writing "hello", we can add some square brackets.

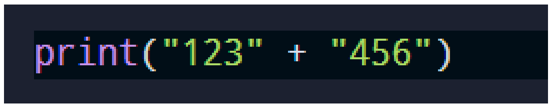


```
print("Hello"[0])
```

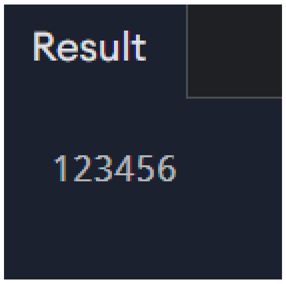


```
Result

H
```

And it's really important to remember that programmers always start counting from zero because we work with binary zeros and ones. This method of pulling out a character is called subscripting.
You can also use instead of positive number negatives numbers but you need to count backwards from -1, so if you want the letter H you need to write (-5)
Now, it's important to remember, though, that just because I can write a number like, "1, 2, 3," as long as it's capped inside these double quotes, then this is not treated as a number by the computer. It's treated just as any other piece of text.



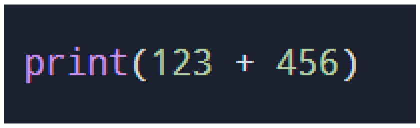
```
print("123" + "456")
```



```
Result

123456
```

Integers

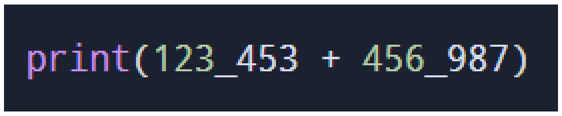
The computer is going to concatenate, if we want to do a math operation. So one of the most common that you'll see is called an integer. So this is programming lingo for just whole numbers. Numbers without any decimal places. And in order to create an integer or declare an integer data type, all you have to do is just write the number without anything else. Like this:



```
print(123 + 456)
```

Large Integers

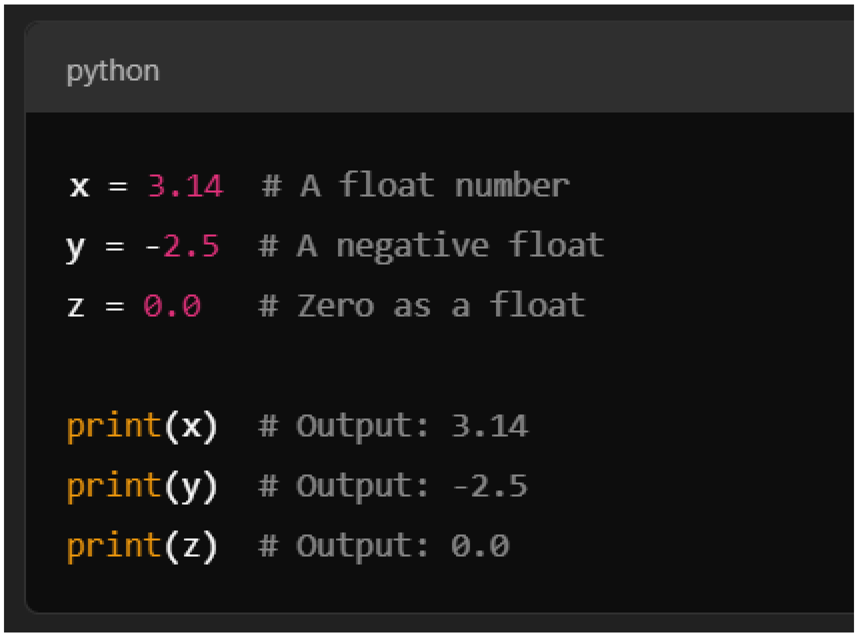
Commonly, when we write large numbers, we like to put commas in between the thousands. In Python we can replace those commas simply with underscores, and it will be interpreted by the computer as if you had written this. So the computer actually removes those underscores and ignores them.



```
print(123_453 + 456_987)
```

Float

A float in Python is just a number that has a decimal point. It's used for things like 3.14 or -2.5.



```python
x = 3.14  # A float number
y = -2.5  # A negative float
z = 0.0   # Zero as a float

print(x)  # Output: 3.14
print(y)  # Output: -2.5
print(z)  # Output: 0.0
```

Boolean

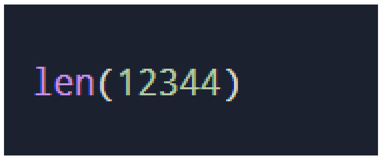
A **Boolean** in Python is a data type that can only have one of two values:

- `True`
- `False`

Now note how these values always begin with a capital T or a capital F, and they don't have any quotation marks around them or anything.
So this is actually a data type which is going to be used a lot in your programs to test if something is true or if something is false, and for your program to respond accordingly.

Type Error, Type Checking and Type Conversion

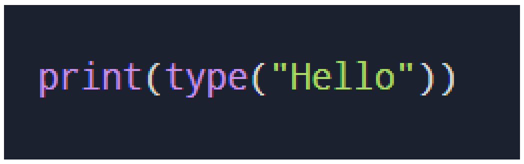
If we use the len option like this:



```python
len(12344)
```

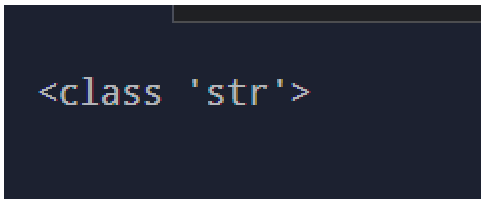
This len() function doesn't like working with integers, and by forcing this through we end up with an error and our code breaks and it gives us this thing, a TypeError.Python is going to give you a sign that the function is wrong which is going to help you to identify.

And the other helpful thing is, when you hover over any function in PyCharm, it gives you the definition of it, and in between the brackets, you can see the data type that that function expects. If we want to identify out data type, theres an option:



```
print(type("Hello"))
```

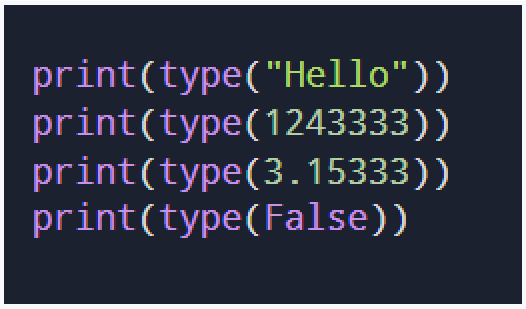
In order to see the output area we write the print code and is going to show it like this:



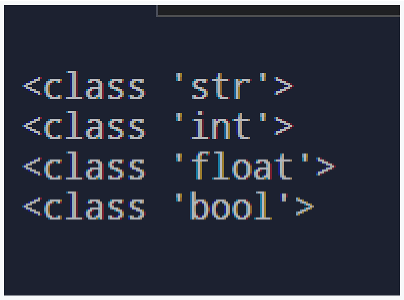
```
<class 'str'>
```

str=strings

So we can do this with various different data types. This is something called TypeChecking, we did it with four more

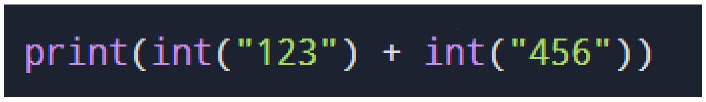


```
print(type("Hello"))
print(type(1243333))
print(type(3.15333))
print(type(False))
```



```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
```

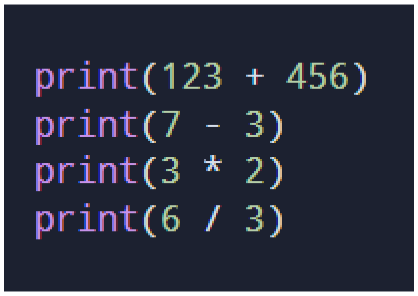
What if we want to convert a piece of data into a different data type?
Well, then we would need to learn about something called type conversion, also known as type casting in Python. For example if we want to convert a string into a integer we can do this:



```
print(int("123") + int("456"))
```

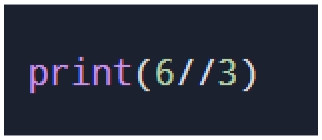
The **int** stands for integer, Now, one of the dangerous things about this is of course sometimes you can't convert things into a different data type.
For example, if I tried to convert "ABC" into a number, that doesn't really make sense. What would ABC be as a number? It would give you what is called a “Value Error”. But we can use this method to convert things into all of the four major primitive types that we've learned about. So we can convert to an integer, we can convert into a float, we can convert into a string, and we can even convert into a bool.

Mathematical Operations in Python



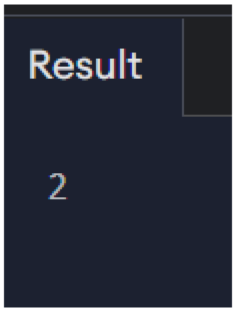
```
print(123 + 456)
print(7 - 3)
print(3 * 2)
print(6 / 3)
```

Now, one thing to notice here is that whenever you're dividing things, you actually always end up with a floating point number. (2.0 , 2.345) So you can see even though 6/3 we're still getting 2.0. Now, sometimes you might not want that. So there is another Python division operator and it is simply done with two forward slashes (//). Like this:



```
print(6//3)
```

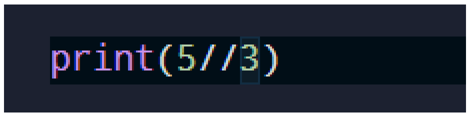
Is going to give us a round number as the result:



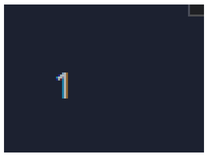
Result

```
2
```

We need to be careful with this divider because what happens if we divide 5/3? Generally it will be 1.66666

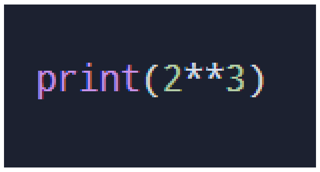


```
print(5//3)
```

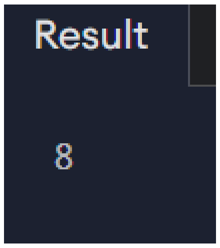


```
1
```

Here it will cut all the decimals and give us 1.Now, the last one that's really useful is two asterisk signs (**), and this gives you access to the exponent. Or when you want to raise a number to a power.



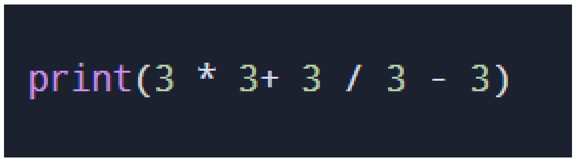
```
print(2**3)
```



```
8
```

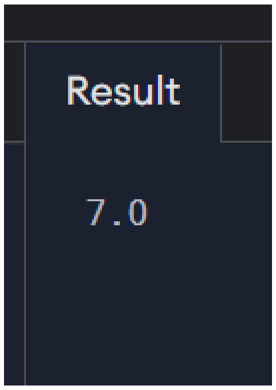
Is 2 x 2 x 2 instead of 2 x 3. Now, one of the things that you have to be careful about when you are doing these mathematical operations is when you have more than one operation on the same line of code. Then there's a certain level of priority. So the things that happen first are the things inside brackets ( ), then it's our exponents, then it's our multiplication * and division /. And finally, the lowest priority is our addition + and subtraction - . Always left to right

In this mathematical operation, what do you think will be printed?



```
print(3 * 3+ 3 / 3 - 3)
```

I think it will be 7, lets run it

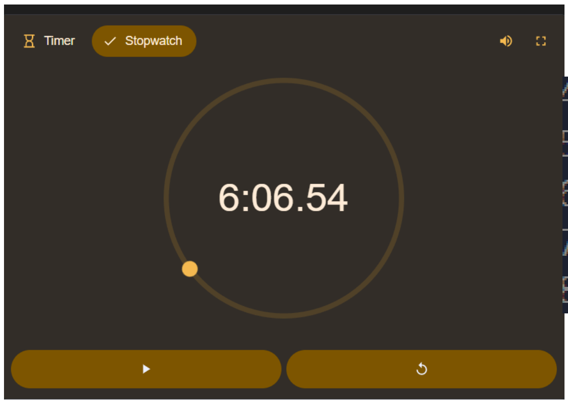


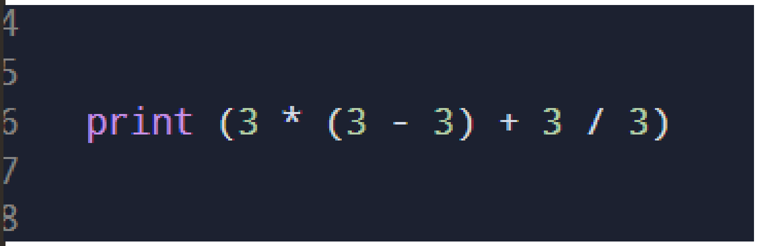
```
7.0
```

We can also guide ourselves by the PEMDAS rule (parentheses, exponents, multiplication, division, addition, and subtraction)

Exercise:

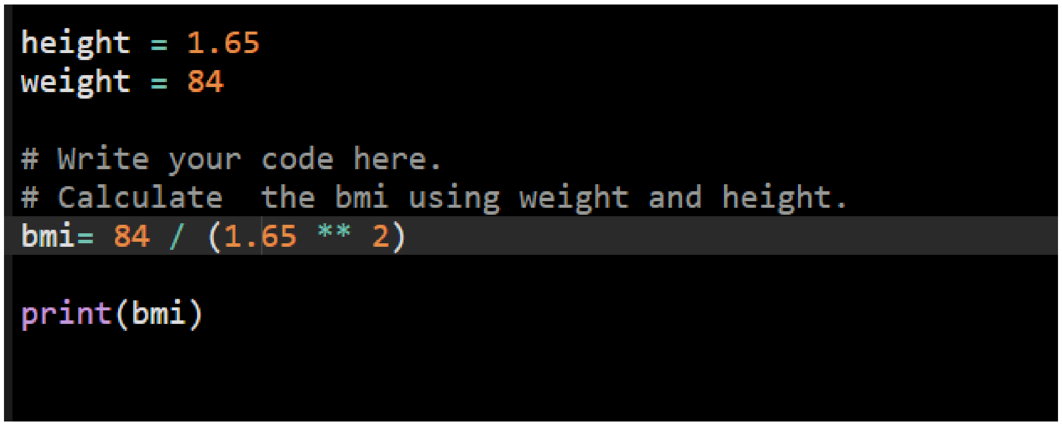
"How can you change this code so that instead of getting 7, we get 3?"





```
print (3 * (3 - 3) + 3 / 3)
```

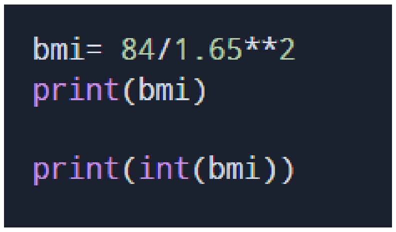
BMI CALCULATOR



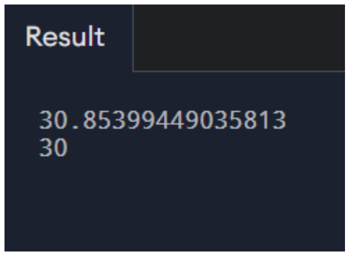
```
height = 1.65
weight = 84

# Write your code here.
# Calculate  the bmi using weight and height.
bmi= 84 / (1.65 ** 2)

print(bmi)
```

Number manipulations and F strings in Python

The output of the BMI calculator usually is a large number after the decimal place, this is not ideal for the user. So instead of simply just printing the original BMI, we can convert it first into an integer, and then we try printing it. **(int)**



```
bmi= 84/1.65**2
print(bmi)

print(int(bmi))
```



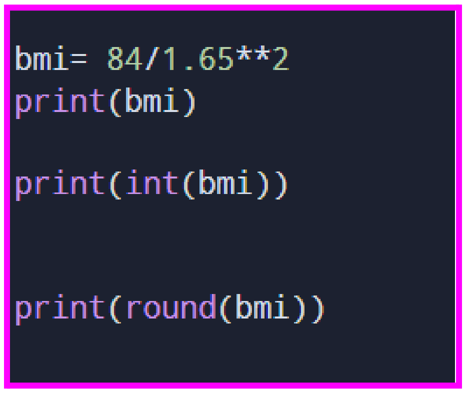
Result

```
30.853994490358l3
30
```

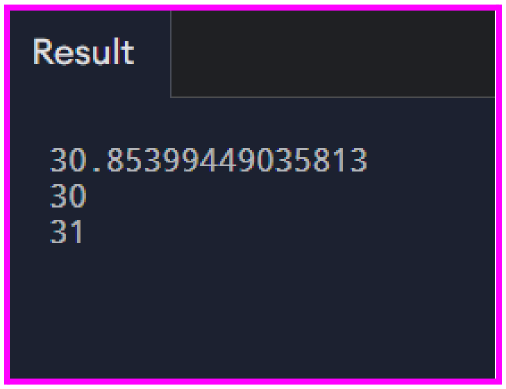
Instead of the first one it will look like the second one, and this is a programming term for basically just removing all of the remaining decimal places, flooring it to the lowest whole number.

round()

We want the function in the traditional mathematical sense, where when it's 0.5, it rounds up to the next whole number, and when it's below that, it rounds up to the lower whole number. Lets see it on code:



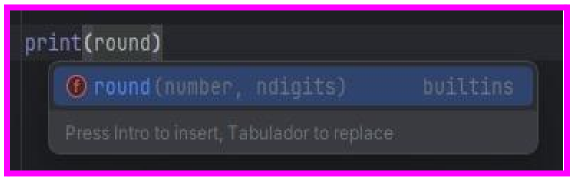
```
bmi= 84/1.65**2
print(bmi)

print(int(bmi))


print(round(bmi))
```



Result

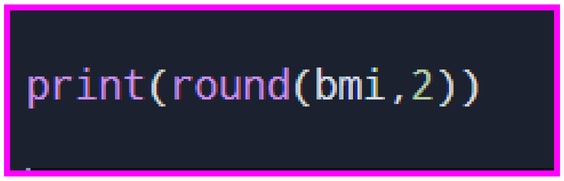
```
30.85399449035813
30
31
```

If it is 3.9 will round into 4, whereas 3.3 will round into 3. Also python suggest that you can round it into two decimal places, lets see it:

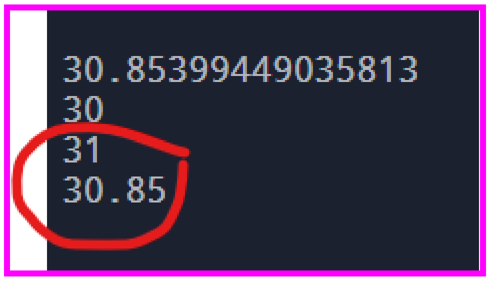


```
print(round)
```

round(number, ndigits)   builtins
Press Intro to insert, Tabulador to replace



```
print(round(bmi,2))
```

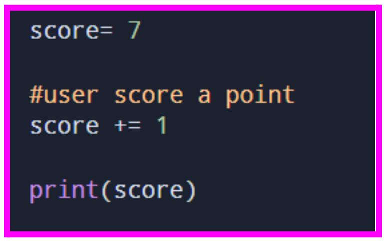


```
30.85399449035813
30
31
30.85
```

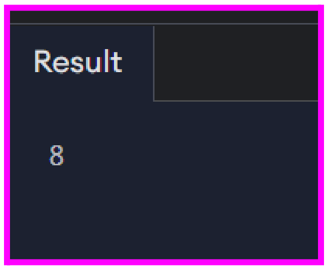
If you're working with money operations this can be really helpful.

Assignment Operator

essentially, what this allows us to do is to accumulate the results of our calculations. For example if youre keeping track of the user scores and he scores a point



```python
score= 7

#user score a point
score += 1

print(score)
```

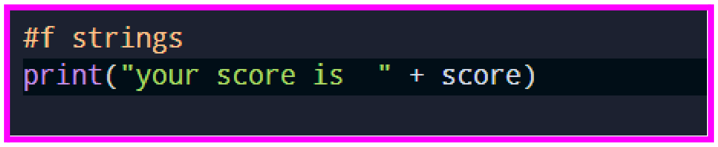


```
Result

8
```

and instead of saying score now equals the previous value of score + 1, you can simply use this shorthand += or for the contrary -=.

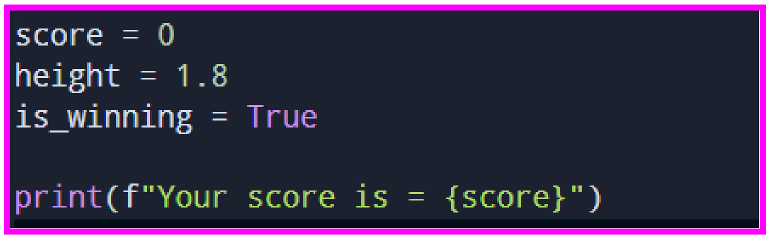
F-Strings

So far, up to this point, if we wanted to print something like, "Your score is..." and then we wanted it to print the score, we have to write +, because these are different data types; this is a string and this is an integer, we get a TypeError.

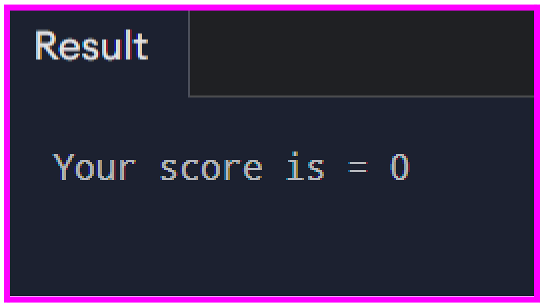


```python
#f strings
print("your score is  " + score)
```

We've had to convert so it can be a match, we know how to do it but usually it is really annoying so we want a faster way to do it, we want **f-string** to help us. And what an f-string allows us to do is in front of a string like this one, we type the character f, and it's really important that it goes in front of the double quotes or single quotes, you start adding values to the string , like this:

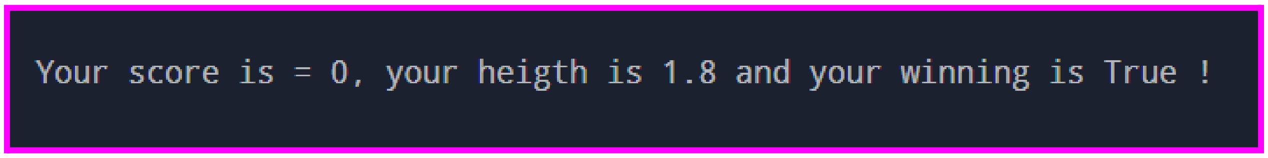


```python
score = 0
height = 1.8
is_winning = True

print(f"Your score is = {score}")
```

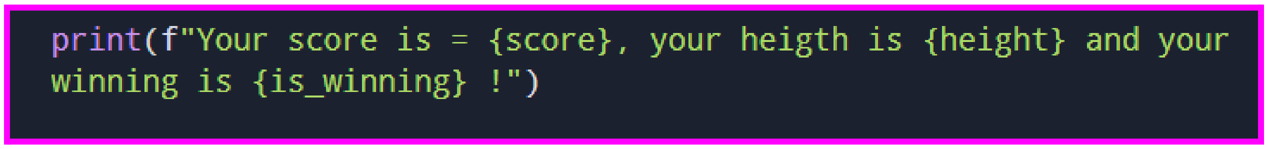


```
Result

Your score is = 0
```

And it does all of the converting and all of the stuff behind the scenes,
and you don't have to worry about any of this. Make sure you put the values between curls.



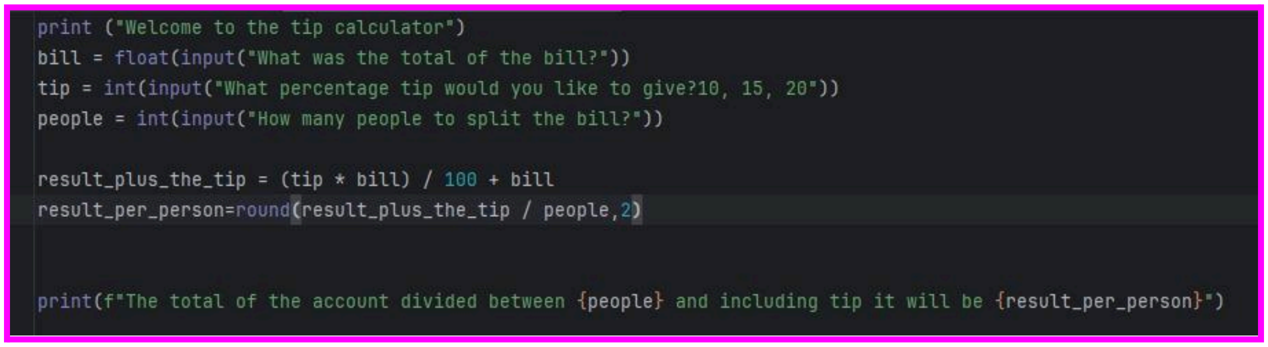
```
Your score is = 0, your heigth is 1.8 and your winning is True !
```



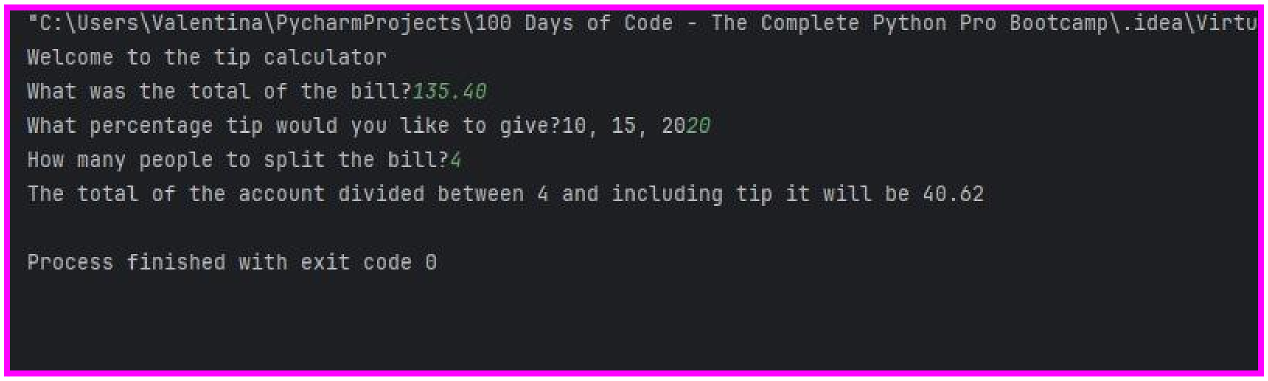
```
print(f"Your score is = {score}, your heigth is {height} and your winning is {is_winning} !")
```

Final project: "Tip Calculator"

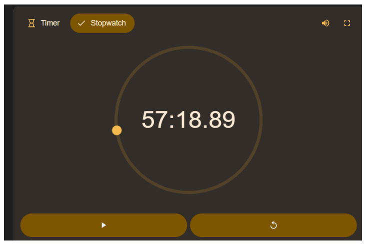
- The final bill need to have two decimals after the point
- The second thing to remember is that these are percentages, in order to calculate a percentage of something, you can multiply a number by the percentage number divided by 100. (150 * 1.12)
- Embarrassing the time I took to do this bullshit



```
print ("Welcome to the tip calculator")
bill = float(input("What was the total of the bill?"))
tip = int(input("What percentage tip would you like to give?10, 15, 20"))
people = int(input("How many people to split the bill?"))

result_plus_the_tip = (tip * bill) / 100 + bill
result_per_person=round(result_plus_the_tip / people,2)


print(f"The total of the account divided between {people} and including tip it will be {result_per_person}")
```



```
"C:\Users\Valentina\PycharmProjects\100 Days of Code - The Complete Python Pro Bootcamp\.idea\Virtu
Welcome to the tip calculator
What was the total of the bill?135.40
What percentage tip would you like to give?10, 15, 2020
How many people to split the bill?4
The total of the account divided between 4 and including tip it will be 40.62

Process finished with exit code 0
```


Timer
Stopwatch
57:18.89