**Lab for Chapter 2 – from Kaefer pp. 36-37**

2.1 Write Python code that uses the input built-in function to ask the user to enter a whole number between 1 and 100. The input function always returns a string value, so use the int built-in function to convert the value entered to an integer data type and square the number that the user entered using the exponentiation operator. Print a message to the user stating the value that they entered and the square of the value that they entered.

1. Code here:
2. value\_entered = input("Please enter an integer between 1 and 100: ")
3. value\_entered\_int = int(value\_entered)
4. value\_squared = value\_entered\_int\*\*2
5. print("The value squared= ", value\_squared)

1. If you use AI to write code for this, what must you be careful of (go beyond “checking the code is correct” discussing comments, style, context, etc.)?

If using AI its important to understand what kind of functionality you are requesting and that the AI generated code aligns with what the intended functionality calls for.

Context is very important.

1. List and explain all the expressions in the code (include discussion of variables and operators):

**Variable**: value\_entered This variable stores the string value entered by the user

**Function**: input()

This function prompts the user to enter a value. The string inside the parentheses is the prompt message displayed to the user.

The input() function returns the user input as a string.

**Function**: int()This function converts the string value stored in value\_entered to an integer.

The = symbol is used to assign the result of the int() function (which is an integer) to the variable value\_entered\_int.

**Variable**: value\_entered\_int This variable stores the integer value converted from the user input.

**Operator**: \*\*This is the exponentiation operator in Python, used to raise value\_entered\_int to the power of 2 (square’s the value).

**Variable**: value\_squared This variable stores the result of squaring value\_entered\_int.

**Function**: print()This function outputs the specified message to the console.

**Concatenation**: The string "The value squared= " is concatenated with the value of value\_squared. In Python, print() can handle multiple arguments, separated by commas, and it automatically inserts spaces between them in the output.

**Variable**: value\_squared This variable's value is displayed as part of the output message.

1. Explain why the input must be converted to an integer data type:

Converting the input to an integer data type is necessary for performing mathematical operations. Without this conversion, there will be errors or unexpected behavior when trying to use the input in calculations.

1. List example “invalid inputs” and explain how the program would respond to them:

"hello", "abc", "!@#". When the input() function retrieves a non-numeric string, the int() function will fail to convert it to an integer.

"3.14", "2.718". If a floating-point number is entered as a string, int() will also raise a ValueError because it cannot convert a string representing a decimal number directly to an integer.

"101", "1000". The program does not check if the integer value is within the specified range (1 to 100) after converting it.

1. Show one alternative way to print the result message and explain the pros and cons of this approach (hint: there are at least two very different ways):

print("The value squared = {}".format(value\_squared))

Pros: **Compatibility** **& Flexibility**-The format() method works with Python 2.7 and later, making it a good option in order to support older versions of Python. The format() method allows for extensive formatting options and can handle multiple placeholders, which can be useful in more complicated formatting situations.

Cons: **Verbosity** **& Performance**- The format() method can be more verbose and less readable compared to f-strings, especially with multiple placeholders. It may be slightly less efficient than f-strings, though in most practical cases, the difference is negligible.

extra credit: Explain if a function is an expression:

2.2 Write Python code that uses the input built-in function to ask the user to enter the year they were born as a four-digit number. The input function always returns a string value, so use the int built-in function to convert the year value entered to an integer data type and subtract the year entered from the current year (e.g., 2019). Print a message to the user stating the value that they entered and their calculated age.

1. Code here:

birth\_year = input("Please enter your four digit birth year: ")

birth\_year = int(birth\_year)

current\_year = 2024 #This should be changed. We should not hare-code the year

age = current\_year - birth\_year #This doesnt take into accound the number of months. Need to fix

print("You entered: ", birth\_year)

print("your age is: " , age)

1. Why might the calculated age not be correct and what can be done to address this:

The current year is hard-coded as 2024. This means that the code will not automatically update as time progresses. If the code is run in a different year, the age calculation will be incorrect.

To automatically get the current year, you can use Python's datetime module. This avoids hard-coding the year and ensures the program always uses the correct current year.

1. If birthyear is a variable used to store the converted input, what’s wrong with the expression "Your age is " + birthyear:

It becomes a problem because it attempts to concatenate a string ("Your age is ") with a variable (birthyear) that is an integer. In Python, you cannot directly concatenate strings and integers using the + operator, as it will result in a TypeError

1. Explain why the names of the variables are good or not good and explain why good naming is important:

They are good because the names clearly indicates that it holds the year of birth, age and current year. Having good variable names make the code easier to read and understand. They provide a clear indication of what each variable represents, reducing the need for comments or explanations. Also, when collaborating with a team, clear and descriptive variable names ensure that all team members can understand and work with the code effectively.

2.3 Write Python code that uses the input built-in function to ask the user to enter a decimal formatted number between 1 and 100. The input function always returns a string value, so use the float built-in function to convert the value entered to a float data type and square the number that the user entered using the exponentiation operator. Print a message to the user stating the value that they entered and the square of the value that they entered.

1. Code here:
2. value\_entered = input("Please enter a floating point between 1 and 100: ")
3. value\_entered = float(value\_entered)
4. value\_squared = value\_entered\*\*2
5. print("The value squared= ", round(value\_squared,2))

1. Research alternative ways to square the inputted value that do not use the exponentiation operator and explain why you may want to use such alternatives:

You can square a number by multiplying it by itself:

value\_squared = value\_entered \* value\_entered

It’s straightforward and easy to understand.

The math module provides a pow() function that can be used for exponentiation:

import math

value\_squared = math.pow(value\_entered, 2)

Using math.pow() makes it explicit that you’re performing an exponentiation operation.

It is also **consistent.** If you’re already using math for other operations, it keeps the code consistent.

1. Add or adjust comments to your code. Explain why you added these and how they are important/useful:

For :  
**import math**:

**Comment**: # Import the math module for mathematical functionsThis helps anyone reading the code understand the purpose of the import and makes it easier to maintain or update the code.

**Comment**: # Square the value by multiplying it by itself This comment explains the operation being performed (squaring) and why it’s being done (multiplying the number by itself). It helps understand the logic behind this particular implementation.

2.4 Modify the code in Exercise 2.3 to round the values reported to the user to two decimal places (use the round built-in function).

1. Before you write the code, look up some documentation on the round() function. What data type is the return value and why is it this data type? How do you know this? Do you think another data type might be better?

The return value of the round() function is a float.

**The reason for this data yype** is the round() function returns a floating-point number to accommodate the precision of decimal places. Even if you round to zero decimal places, the result is still a float, not an integer.

The function maintains consistency in data types across different usages. Since floating-point arithmetic and operations are involved, returning a float avoids potential issues with type conversion or precision loss.

The Python documentation specifies that the return value is of type float. I verified this by checking in my Python session.

Code here: print("The value squared= ", round(value\_squared,2))

2.5 Write Python code that uses the input built-in function to ask the user to enter a sentence of their choosing. Use the len built-in function to determine how many characters were in the string entered and report this information back to the user.

Code here:

sentence = input("Enter a sentence: ")

String\_length = len(sentence)

print("You entered: " , sentence, "It has length" ,String\_length)

1. Describe a situation in which it would be useful to have the length of a string

When collecting user input, like a password or username, you might want to enforce certain length requirements. For example, a password should be at least 8 characters long.

2.6 Write Python code that uses the input built-in function to ask the user to enter a weight in pounds. The input function always returns a string value, so use the float built-in function to convert the value entered to a float data type and determine the equivalent weight in kilograms (you can use the conversion factor that 1 pound = 0.453592 kilograms). Print a message to the user stating the weight in pounds that they entered and the equivalent weight in kilograms.

1. Write the program using one line of code. Code here:

print("You weigh: ", round(float(input("Please enter your weight in pounds: ")) \* 0.453592, 1), "kilos")

1. Discuss the pros and cons of implementing it in one line:

**Pro’s**: The code is compact, reducing the number of lines and potentially making it quicker to write.

**Con’s**: The code can become hard to read and understand, especially for those who are not familiar with the logic or the chaining of operations.

2.7 Write Python code that uses the input built-in function to ask the user to enter a temperature in the Fahrenheit temperature scale. The input function always returns a string value, so use the float built-in function to convert the value entered to a float data type and determine the equivalent temperature in the Celsius temperature scale (use the conversion factor °C = (°F – 32) × (5/9)). Print a message to the user stating the temperature in Fahrenheit that they entered and the equivalent temperature in Celsius. You can verify that your code executes properly by entering in 32°F (equivalent is 0°C) and 212°F (equivalent is 100°C).

1. Code here:
2. degreesF = input("Enter a temperature in Fahrenheit: ")
3. degreesC = (float(degreesF) - 32) \* (5/9)
4. print("This converts to " , degreesC, "Celsius")

1. Why do you want to verify your code using the input 32°F? What is it called when you do this kind of verification?

32°F is a special case because it is the freezing point of water in Fahrenheit. When converted to Celsius, it should result in exactly 0°C. This makes it an ideal test case to check if the conversion formula is correctly implemented.

This type of verification is often referred to as **"validation"** or **"testing with known values."** It's a form of **"unit testing"** where you use specific inputs with known outputs to ensure the correctness of your code.

1. Are there other verifications that should be done? Explain:

Yes, in the case for edge cases. Test with temperatures at the extreme ends of the scale, such as very high and very low temperatures, to ensure the program handles these cases correctly.

1. Re-write the program as a function and test the function:

def FtoC(temperatureF)

degreesC = (float(degreesF) - 32) \* (5/9)

return(degreesC)

degreesF = input("Enter a temperature in Fahrenheit: ")

print("This converts to " , FtoC(degreesF, "Celsius")

1. Explain the pros and cons of creating a function to do the conversion:

**Pro**: Functions allow you to reuse the same code across multiple places in your program or even in different programs without duplicating code. This promotes DRY (Don't Repeat Yourself) principles. An **Example** is If you need to convert temperatures in multiple locations or scripts, you can call FtoC() each time instead of rewriting the conversion logic.

**Con**: For very simple scripts or one-off calculations, creating a function might introduce unnecessary overhead. In such cases, the additional lines of code might seem like extra work without significant benefit.

An example of this is if you only need to convert temperature once in a small script, using a function might be overkill.