**If-else selection**

1. Write a program which asks the user for an integer number. The program should print “The war is over!!” if the number is exactly 1945, otherwise do nothing.
2. Write a program which asks the user for an integer number. If the number is less than zero, the program should print out the number multiplied by -1. Otherwise, the program prints out the number as is. For example, if -9 is inputted then the output should be 9. If 150 is inputted, then the output should be 150.
3. Write program that asks the user for 2 numbers and an operation. If the operation is *add*, *multiply* or *subtract*, the program should calculate and print out the result of the operation with the given numbers. If the user types anything else, the program should print out nothing. For example, if the numbers are 4 and 5 and the operation is add, then the output should be as follows:  
   number 1: 4  
   number 2: 5  
   operation: add  
   4 + 5 = 9
4. Write a program which asks for the hourly wage, hours worked and the day of the week. The program should then print out the daily wages, which equal hourly wage multiplied by hours worked, except on Sundays when the hourly wage is double. The output could look as follows:  
   hourly wage: 8.5  
   hours worked: 3  
   day of the week: Monday  
   daily wages: 25.5 pounds  
   hourly wage: 12.5  
   hours worked: 10  
   day of the week: Sunday  
   daily wages: 250.0 pounds
5. Write a program which asks for tomorrow’s weather forecast and then suggests weather-appropriate clothing. The suggestion should change if the temperature is over 20, 10 or 5 degrees and if there is rain on the way. The output could look as follows:  
   what is the weather forecast for tomorrow?  
   temperature: 21  
   will it rain (yes / no): no  
   wear jeans and a T  
   what is the weather forecast for tomorrow?  
   temperature: 10  
   will it rain (yes / no): no  
   wear jeans and a T  
   a jumper is recommended  
   what is the weather forecast for tomorrow?  
   temperature: 3  
   will it rain (yes / no): yes  
   wear jeans and a T  
   a jumper is recommended  
   take a warm coat  
   take an umbrella

**While loops iteration**

1. Go to if\_looping.py and include a score for the user where they accumulate points as they play each game.
2. This program has some syntactic issues, fix them so that the output is correct:  
   print(“Let’s do this”)  
   num = int(input(“Enter a number:”)  
   while num =0:  
    print(num)  
    print(“DONE!!!”)  
   Output:  
   Enter a number:5  
   5  
   4  
   3  
   2  
   1  
   DONE!!!
3. In the game of Lucky Sevens, the player rolls a pair of dice. If the dots add up to 7, the player wins £4, otherwise the player loses £1. You need to write a program that should take as input the amount of money that the player wants to put into the pot and play the game until the pot is empty. At that point, the program should print the number of rolls it took to break the player, as well as the maximum amount of money in the pot.
4. Make use of a nested while loop to achieve the following output of building a pyramid of numbers:  
   0 1 2 3 4  
   0 1 2 3  
   0 1 2  
   0 1  
   0  
   **Hint:** look at the stars code from slides\_examples3.

**Strings**

1. Given a list of integers, we can decide that 2 items are neighbours if they differ by 1. For example, the list 2, 3, 5, 7, 9 would show that 2 and 3 are neighbours as they only differ by 1. If the list was 2, 3, 4, 5, 9 then 2 and 3, 3 and 4, 4 and 5 are neighbours.
   1. Just think about / scribble down / code the following - Write a function that looks for the longest series of neighbours within the list and returns its length. For example, the above-mentioned list the longest list would be 2, 3, 4, 5 and the length would be 3.
   2. Now, use ChatGPT and ask for a solution (copy and paste the above problem to the prompt).
   3. Once you have completed reading ChatGPT’s code solution / explanation, please complete the survey using this QR code:  
      **A qr code with a person's face

      Description automatically generated**

**For loops**

1. Write a function called anagrams. The function must take 2 strings as arguments. The function returns True if the strings are anagrams of each other. An anagram means that both words contain the same letters. For example, tame and meta are anagrams.
2. Create a matrix of your own (numbers). Create 3 functions that sum\_rows(matrix), sum\_cols(matrix) and change\_value(matrix, row\_nbr, col\_nbr, new\_value).

**Lists**

You have been asked to build a student management system where students are stored (using a list). Each student has a student number and 2 assessment grades. For example:

[[12345, 67, 81], [54321, 77, 45], [56789, 90, 59]]

Develop a fully functioning application that provides the user with a menu displaying the following:

1. Add a student
2. Change an assessment grade for a student
3. Remove a student
4. Generate a report of all students and their grades
5. Generate a report showing student grades highest to lowest
6. Exit

Please make your choice: \_\_\_\_\_

Let’s look at each menu option and what is required to code this. The list can be called student\_list[].

1. Start with main() and include a menu with options that reflect the functions. Declare student\_list in main(). Use match / case.
2. Add a student -> use prompts so that a user can add a student number, module code and 2 grades (if no grades are available, they can default to zero). The student number must be generated automatically. Type in the student’s name and surname and the student id must be a combination of these characters. For example, if we type Jack Smith then the id must be js(followed by a random number up to 999). So, it could be js846. Assume there is only one first name and a last name unless you would like to improve on your coding skills. This function should also ask the user if there is a grade to be entered. If the user chooses yes, call a function enter\_grade(). This function should ask for the grades and return the grades to the add\_student(). Finally, pass the id and both grades back to main() and update the student\_list [].
3. Change an assessment grade – use a prompt so that a user can enter a student id, find the student to change the grades. Call the enter\_grade() – already created to enter the grades and update the grades for that student. Keep it simple, don’t worry about which grade, etc for now, unless you want to.
4. Remove a student – use a prompt to ask which student to remove, the function must find the student using the student id and remove the student – include a “are you sure you want to remove student xxx? As part of the coding.
5. The function must display all students with their grades.
6. **Optional:** The function must display students and their grades from highest to lowest, this must be where the 2 grades are averaged showing the averaged grade from highest to lowest.