Conditions and Loops

Solve the following exercises and upload your solutions to Moodle until the specified due date. Make sure to use the ***exact filenames*** that are specified for each individual exercise. Use the provided **unit tests** to check your scripts before submission (**see the slides Handing in Assignments on Moodle**). Unless explicitly stated otherwise, you can assume correct user input and correct arguments. You are *not allowed* to use any concepts and modules that have not yet been presented in the lecture.

Important Information!

Please try to *exactly match the output* given in the examples (naturally, the input can be different). We are running automated tests to aid in the correction and grading process, and deviations from the specified output lead to a significant organizational overhead, which we cannot handle in the majority of the cases due to the high number of submissions.

For example, if the exercise has an output of
`Diamond size: X`
(where `X` is some user input), do not write
`Enter diamond size: X`
(additional `Enter` and lower case `diamond`) or
`Diamond size:X`
(missing space after the colon).

Note that in general, after a : there is always a fixed white space. Feel free to copy the output text from the assignment sheet, and then change it according to the exercise task.

Exercise 1 – Submission: `a2_ex1.py` 25 Points

Write a program that calculates and outputs the bonus payment of an employee in your company *PyComp*. To do this, read the duration of the employment and the department the employee works in. The employment years must be greater than 0 and the departments are numbered from 10 to 99.

- For a negative number or 0 (employment), or an invalid department number < 10 or > 99, print an error message (`"Invalid input"`) and terminate the program by calling `exit(0)` You can assume that the user will only enter digits.
- For an employment less than 2 years, the bonus is 200 €.
- For an employment between (and including) 2 and 3 years the bonus is 300 €.
- For an employment of more than three years there is a base bonus of 400 €. Furthermore, there is an additional bonus depending on the department.
  - If the last digit of the department is between (and including) 1 and 5, the base bonus is increased by 10%.
  - If the last digit of the department is between (and including) 6 and 9, the base bonus is increased by 20%.

In case of valid inputs (i.e., no errors as described above), print both the bonus (float with 2 decimal places; see the example input and output below for how it must look like). Do not perform unnecessary checks, i.e., do not check for values that have already been excluded by previous conditions.

Example input and output:[1]

```
Enter employment years (> 0): 2
Enter department (10-99): 8
Invalid input
```

Example input and output:

```
Enter employment years (> 0): 0
Enter department (10-99): 10
Invalid input
```

Example input and output:

```
Enter employment years (> 0): 7
Enter department (10-99): 53
Bonus for 7 years of employment in department 53: 440.00
```

Example input and output:

```
Enter employment years (> 0): 3
Enter department (10-99): 77
Bonus for 3 years of employment in department 77: 300.00
```

Exercise 2 – Submission: `a2_ex2.py` **25 Points**

Write a program that repeatedly reads positive integer numbers from the console using a while loop (see the example input and output below for how it must look like) and multiplies them. You do not need to store all values, only the current input is relevant. The loop should exit in two situations:

- When the user inputs an `"x"` instead of a positive integer number. In this case, the program should output the product of all numbers entered.
- Should the product become greater than the value 1000, the program should exit the loop immediately (without the user needing to type `"x"`). Then a message along with the result should be printed: `"Result has exceeded the value 1000: XXXX"` Where `"XXXX"` is the computed product.

Ensure that your program works for an empty sequence of numbers (i.e., the user immediately entered `"x"`), in which case you must print `"Empty sequence"`. You do not need to check the input, you can assume that they are positive integers.

Example input and output:

```
Enter a value (or 'x' to stop): x
Empty sequence
```

[1] Green colored text indicates user input from the console.

Example input and output:

```
Enter a value (or 'x' to stop): 4
Enter a value (or 'x' to stop): x
Result: 4
```

Example input and output:

```
Enter a value (or 'x' to stop): 4
Enter a value (or 'x' to stop): 5
Enter a value (or 'x' to stop): 3
Enter a value (or 'x' to stop): 2
Enter a value (or 'x' to stop): x
Result: 120
```

Example input and output:

```
Enter a value (or 'x' to stop): 9
Enter a value (or 'x' to stop): 7
Enter a value (or 'x' to stop): 7
Enter a value (or 'x' to stop): 4
Result has exceeded the value 1000: 1764
```

Exercise 3 – Submission: `a2_ex3.py` 25 Points

Using the built-in `range(start, stop, step)`, write a program that reads these three integer numbers and iterates through the value range. Your program must do the following (see the example input and output below for how it must look like):

- Compute the sum of all even values.
- Compute the sum of all uneven values multiplied with the respective index ($value \times index$ for every iteration and the sum of that).
- For the first and last 5 values, print them to the console in the format that you see below. If the user-specified range results in less than 5 values, just print those. If the range is more than 5 but less than 10 (so that you cannot print 5 first and 5 last values), just print all steps.
- After the iteration, print both the even-value sum as well as the sum of odd values times their indices. If the range does not contain any values (an empty range) or if there are no even/odd values, the respective sum must be set to 0. Important: the upper boundary (stop) should be included in the iteration.

Example input and output:

```
Start: 1
Stop: 10
Step: 3
Index: 0, Value: 1
Index: 1, Value: 4
Index: 2, Value: 7
Index: 3, Value: 10
Sum of even values: 14
Sum of odd multiplied values: 14
```

Example input and output:

```
Start: 0
Stop: 100
Step: 1
Index: 0, Value: 0
Index: 1, Value: 1
Index: 2, Value: 2
Index: 3, Value: 3
Index: 4, Value: 4
Index: 96, Value: 96
Index: 97, Value: 97
Index: 98, Value: 98
Index: 99, Value: 99
Index: 100, Value: 100
Sum of even values: 2550
Sum of odd multiplied values: 166650
```

Example input and output:

```
Start: 59
Stop: 20
Step: 1
Sum of even values: 0
Sum of odd multiplied values: 0
```

Example input and output:

```
Start: 9
Stop: 40
Step: 5
Index: 0, Value: 9
Index: 1, Value: 14
Index: 2, Value: 19
Index: 3, Value: 24
Index: 4, Value: 29
Index: 5, Value: 34
Index: 6, Value: 39
Sum of even values: 72
Sum of odd multiplied values: 388
```

Exercise 4 – Submission: `a2_ex4.py` 25 Points

Write a program that prints a diamond over a user-specified size (number of rows/lines - integer). If this number is less than 2, then print `"Invalid size"`. Otherwise print a diamond over the entered number of lines consisting of the character `*` (see the example input and output below for how it must look like). In order to draw this, the size needs to be an uneven number. If the user enters an even number, just increment it by one. This means that for example a size of 10 results in the same output than a size 11. Examples for such "diamonds" for sizes 5 and 7:

```
  *
 * *
*   *
 * *
  *
```

```
   *
  * *
 *   *
*     *
 *   *
  * *
   *
```

Before you start programming, try to find a general pattern depending on the given size: How many spaces must be printed in respective lines before a star `*` has to occur?

Example input and output:

```
Diamond size: 1
Invalid size
```

Example input and output:

```
Diamond size: 7
   *
  * *
 *   *
*     *
 *   *
  * *
   *
```

Example input and output:

```
Diamond size: 12
      *
     * *
    *   *
   *     *
  *       *
 *         *
*           *
 *         *
  *       *
   *     *
    *   *
     * *
      *
```