5.1 Create a program that first prints the even numbers 0 to 100 in an ascending order, and then the same numbers in descending order. Each number is printed on its own row, so the program's printout looks like this:

0

2

4

6

...

98

100

100

98

...

4

2

0

Programming tips:

* Review the subsection "Using the range Function with the for Loop" from the section "The for Loop: a Count-Controlled Loop" of the book Starting out with Python (in the 3rd edition Section 4.3, in the 2nd edition Section 5.3).
* When necessary (when performing the repetition that counts the numbers downwards), the starting value can be greater than the ending value and the step does not need to be positive.

5.2 Create a program that first reads 5 numbers a user has entered and then prints, in the order of entry, all the numbers that are greater than zero. An example of how the program operates:

Give 5 numbers:

Next number: *0*

Next number: *1*

Next number: *-2*

Next number: *3*

Next number: *-4*

The numbers you entered that were greater than zero were:

1

3

Programming tips:

* First, the numbers entered by the user must be saved to a list so that they can be reviewed again.
* It is the easiest to go through the list's elements using a for statement. See the first example of a for-statement in the list section of the book. While going throug the elements, check whether the element is greater than zero and, if it is, print it. The task can also be implemented using a while structure, but the for structure is the one intended for going throug a list this way.

5.3 Create a program that first reads 5 numbers a user has entered and then prints all the entered numbers in reverse order. An example of how the program operates:

Give 5 numbers:

Next number: *2*

Next number: *7*

Next number: *3*

Next number: *-8*

Next number: *6*

The numbers you entered, in reverse order:

6

-8

3

7

2

Programming tips:

* When you started to print numbers from a list in the previous program, the for command processed the elements saved to the list directly (at least if you implemented it in the simplest way). If you copy your program code [to Python Tutor](http://pythontutor.com/visualize.html?#heapPrimitives=true&py=3) and look at execution of the program, this is shown so that the for command that performs the printing goes through the **elements** of the list.
* In this task, the list must be gone through in reverse order at the printing phase. This cannot be done with a simple for command, but rather, the user must use list indexes, ie. the numbers that are, in the image drawn by Python Tutor, inside the yellow box that shows the list. These indexes are the order numbers for the list elements, which can be used to access the members in the list using the principle listname[index].
* To implement printing, write a for command, which goes through the list's **index numbers** in the order the user wants the printing to happen. Then, inside this for command, enter a print command using a for loop variable to print the correct value from the list.

5.4 Create a program that first reads 5 numbers a user has entered and then prints all the entered numbers in reverse order. An example of how the program operates:

Give 5 numbers:

Next number: *2*

Next number: *7*

Next number: *3*

Next number: *-8*

Next number: *6*

The numbers you entered, in reverse order:

6

-8

3

7

2

Programming tips:

* When you started to print numbers from a list in the previous program, the for command processed the elements saved to the list directly (at least if you implemented it in the simplest way). If you copy your program code [to Python Tutor](http://pythontutor.com/visualize.html?#heapPrimitives=true&py=3) and look at execution of the program, this is shown so that the for command that performs the printing goes through the **elements** of the list.
* In this task, the list must be gone through in reverse order at the printing phase. This cannot be done with a simple for command, but rather, the user must use list indexes, ie. the numbers that are, in the image drawn by Python Tutor, inside the yellow box that shows the list. These indexes are the order numbers for the list elements, which can be used to access the members in the list using the principle listname[index].
* To implement printing, write a for command, which goes through the list's **index numbers** in the order the user wants the printing to happen. Then, inside this for command, enter a print command using a for loop variable to print the correct value from the list.

5.7 Mutable and immutable datastructures

The parameters of functions

Look at this program code:

def more(i):

i += 1

def main():

number = 42

more(number)

print(number)

main()

What does it print?

42

43

44

List as a parameter of a function

Look at this program code:

def more(l):

l.append('more')

def main():

list = []

more(list)

print(list)

main()

What does it print?

An empty list

A list that contains the string more once

A list that contains the string more twice

Two ways to sort a list

You can sort a list in two different ways:

list = [1, 4, 7, 2, 5, 3]

new\_list = sorted(list) # function sorted returns a sorted copy of the original list

list.sort() # method sort sorts the list it was called for

if list == new\_list:

print("Yep yep")

else:

print("Oh no")

Is the result of the previous operations different or the same, i.e. what is printed?

Yep yep

Oh no

Two lists, the same or different?

In Python the operation is compares the references that are stored. In case of lists it compares whether the references point in the same list.

If we change the comparison in the previous program to be list is new\_list, i.e.:

list = [1, 4, 7, 2, 5, 3]

new\_list = sorted(list)

list.sort()

if list is new\_list:

print("Yep yep")

else:

print("Oh no")

What is printed then?

Yep yep

Oh no

5.8 Create a function named input\_to\_list that has 1 parameter: number of integers it wants to read. The function then asks for that many number from the user, reads the numbers from the user, saves them in a list and returns the list.

Also create a main program that

1. asks the user for the number of integers to be processed
2. calls the previously-defined function to read the numbers from the user
3. asks the user what number they are searching for
4. prints data on whether the searched numbers are found from the entered numbers and, if so, how many times

Examples of how the program works (rows 2-7 are handled by the function):

How many numbers do you want to process: *5*

Enter 5 numbers:

*7*

*3*

*2*

*7*

*7*

Enter the number to be searched: *7*

7 shows up 3 times among the numbers you have entered.

How many numbers do you want to process: *3*

Enter 3 numbers:

*5*

*6*

*7*

Enter the number to be searched: *3*

3 is not among the numbers you have entered.

Programming tips:

* First implement the function that asks the user for the numbers and returns them to the main program as a list.
* The book presented the list method called append that is useful for storing the values in the list. When performing this task, you should be aware that the list also includes the method count, which counts the amount of times the member given as a parameter shows up. count is used like append, but of course the greatest difference is that append has no return value and, when using count, the most important thing is the number count returns.

5.9 Create the function are\_all\_members\_same, which uses a list as a parameter and returns information on whether all the members contained by the list are the same.

Examples of how the function operates, when tested in the interactive mode of the Python interpreter:

>>> are\_all\_members\_same([42, 42, 42, 43, 42])

False

>>> are\_all\_members\_same([42, 42, 42, 42])

True

Programming tips:

* The return value for this function is a truth value (True or False).
* You can implement the check by either utilizing the list methods or by going through the list and comparing whether the first element is the same as the second, the second one the same as the third etc.
* This task also allows for a bit of philosophizing. Does the set theory define whether all the members of an empty set are the same? They are.

5.10 Create the function is\_the\_list\_in\_order, which uses a list as a parameter and returns the information on whether the numbers contained by the list are in an ascending order, ie. is each number always at least as large as the one preceding it.

Examples of how the function operates when tested in the interactive mode of the Python interpreter:

>>> is\_the\_list\_in\_order([37, 42, 43])

True

>>> is\_the\_list\_in\_order([42, 37, 43])

False

Programming tips:

* What is the type of the function's return value?
* You can check whether the members are in an order of magnitude by going through the list and checking if the first and second member are in the right order, if the second and third member are etc.

5.11 In official Rubik's Cube contests, a participant's effort is evaluated in the following way:

* The contestant may solve the Rubik's Cube 5 times. Each achievement time is measured in seconds.
* The best and the worst time are removed.
* An average of the remaining times is calculated and set as the contestant's official score.

Implement a program that asks for the times of the contestants' performances and prints the result to the hundredth of a second. An example of how the program functions:

Enter the time for performance 1: *5.80*

Enter the time for performance 2: *5.40*

Enter the time for performance 3: *5.17*

Enter the time for performance 4: *5.19*

Enter the time for performance 5: *5.22*

The official competition score is 5.27 seconds.

5.12

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

5.13Let's assume that at some less inhabited area (Teisko?), buses leave for Tampere according to the following schedule:

|  |
| --- |
| 6.30 |
| 10.15 |
| 14.15 |
| 16.20 |
| 17.20 |
| 20.00 |

Design and implement a program that asks the user what time it is and prints the times for the next three buses, based of the entered time.

To be able to concentrate on the essential matter (presenting the schedule as a list), simplify the presentation of the time by saving the time as one integer where the minutes and the hours are expressed in the same number, ie. 6.30 as 630 and 10.15 as 1015. Times presented this way can easily be compared with each other (ie. does a certain time come before another time) using the normal comparison operators. Examples of how the program functions:

Enter the time (as an integer): 1000

The next buses leave:

1015

1415

1620

Enter the time (as an integer): 1800

The next buses leave:

2000

630

1015

Programming tips:

* The first thing you should do is, of course, saving the bus schedule to a Python data structure (list).
* After asking a time from the user, find the part of the data structure (index) that contains the time the next bus leaves. In other words, go through all the elements of the list and compare the times on the list to the time entered by the user. When the program finds a time from the list that is greater or as great as the time entered by the user, it is the time the next bus leaves.
* When the right element has been found, print the next three bus times from there onwards, unless you are at the end of the table, in which case print only until the end of the table and then print more times from the beginning of the table, so that a total of three times are printed.
* Implement a variable that calculates how many bus times have already been printed (or, alternatively, a counter with a value starting at 3, which then goes one down every time a bus time is printed).
* Then, develop a mechanism that continues printing from the start of the table when required. You can do this using an if structure, for instance, but a cleverer way is utilizing the remainder operator (% in Python).

5.14Let's implement a noughts and crosses game. We begin by creating a version that only allows adding marks (noughts or crosses) on the game board, but does not check when the game ends. The implementation of the program thus always ends after nine turns, independently of whether either player manages to fill a row with their mark earlier.

Use the attached [template code file](https://plus.tuni.fi/graderB/static/tie02101-s2020/static/code/ristinolla.py) to implement a program that prints a noughts-and-crosses grid, asks where the player wants to place their mark, places the player's mark on this grid if the square is empty, and prints the grid again if something was placed on board.

The board is a 3x3 grid, where an empty square is marked by a "." symbol. Each square contains a coordinate whose x and y values are between 0 and 2. The coordinate (0,0) is on the left upper corner of the grid. The coordinate values increase when going rightwards and downwards. For instance, the square coordinate of the lower right corner is (2,2) and the one on the right side of the centre row is (2,1).

The user enters the coordinates of the square they choose, separated by one space. The first coordinate is the x coordinate and the second one is the y coordinate. The mark of the player whose turn it is is placed the square given by the player if the square is empty. If there is already a mark in the square, print the error message "Error: a mark has already been placed on this square." The turn changes to the other player when a mark is placed on the board successfully. We have already implemented the reviewing of errors for the user's entries, and the ending of the game when the grid is filled, to the file template.

An example of how the program operates:

...

...

...

Player X, give coordinates: *1 1*

...

.X.

...

Player O, give coordinates:*1 1*

Error: a mark has already been placed on this square.

Player O, give coordinates: *2 5*

Error: coordinates must be between 0 and 2.

Player O, give coordinates: *two*

Error: enter two integers, separated with spaces.

Player O, give coordinates: *2 1*

...

.XO

...

Player X, give coordinates:

Programming tips:

* If it is hard to visualize the indexes, draw a data structure on the paper.
* Remember to implement a division to functions in your program.
* Which condition must be fulfilled for the player to locate their mark and thus perform their turn?
* Remember to change the player's turn after a player has located their mark on the board. You can change the turn by increasing the variable found from the code template turn by one.
* The string join method can be useful for printing the board.

5.15 Implement a program that checks when the game ends and declares the winner.

The program must check the horizontal and vertical rows and both slanted rows.

If the players are in a draw, print: "Draw!"

An example of how the program operates:

...

...

...

Player X, give coordinates: 1 1

...

.X.

...

Player O, give coordinates: 0 0

O..

.X.

...

Player X, give coordinates: 0 1

O..

XX.

...

Player O, give coordinates: 2 2

O..

XX.

..O

Player X, give coordinates: 2 1

O..

XXX

..O

The game ended, the winner is X

Programming tips:

* Remember the division into functions.
* When checking whether an user won, check only the row and column that has last been marked. Slanted rows and columns can be checked every time.
* When should you check the winning condition? Would you, for instance, need to check the b section of the example run after each user entry?
* The game ends when all the marks in a row are the same. Which mark does not meet this condition, however?
* There are only a few victory conditions, so you should check them directly by indexing the screens. Python can perform the comparison a == b == c.
* When a victory condition is fulfilled, the winner of the game is the player who has last added a mark to the board.