# Lab: Lists Advanced

Problems for in-class lab for the Python Fundamentals Course @SoftUni.
Submit your solutions in the SoftUni judge system at https://judge.softuni.org/Contests/1730.

## 1. No Vowels

Using comprehension, write a program that receives a **text** and **removes** all its **vowels**, **case insensitive**. Print the new text **string after removing the vowels**. The vowels that should be considered are **'a', 'o', 'u', 'e', 'i'**.

### Examples

| Input | Output |
|---|---|
| Python | Pythn |
| ILovePython | LvPythn |

## 2. Trains

You will receive a number representing the number of **wagons** a train has. Create a **list** (train) with the given length containing **only zeros**. Until you receive the command **"End"**, you will receive some of the following commands:

- **"add {people}"** – you should add the number of people in the last wagon
- **"insert {index} {people}"** - you should add the number of people at the given wagon
- **"leave {index} {people}"** - you should remove the number of people from the wagon. There will be no case in which the people will be more than the count in the wagon.

There will be no case in which the index is invalid!

After you receive the **"End"** command print the train.

### Example

| Input | Output |
|---|---|
| 3<br>add 20<br>insert 0 15<br>leave 0 5<br>End | [10, 0, 20] |
| 5<br>add 10<br>add 20<br>insert 0 16<br>insert 1 44<br>leave 1 12<br>insert 2 100<br>insert 4 61<br>leave 4 1<br>add 15<br>End | [16, 32, 100, 0, 105] |

# 3. To-do List

You will be receiving **to-do notes** until you get the command **"End"**. The notes will be in the format: **"{importance}-{note}"**.

Return a list containing all **to-do notes** sorted by **importance in ascending order.**

The importance value will always be an integer between **1** and **10 (inclusive)**.

## Hint

- Use the **pop()** and **insert()** methods.

## Example

| Input | Output |
|---|---|
| 2-Walk the dog<br>1-Drink coffee<br>6-Dinner<br>5-Work<br>End | ['Drink coffee', 'Walk the dog', 'Work', 'Dinner'] |
| 3-C<br>2-A<br>1-B<br>6-V<br>End | ['B', 'A', 'C', 'V'] |

## Hints

Start by and creating an empty list:

02-to-do-list.py

```
1  notes = [0] * 10
```

Create a while loop that reads the command and splits it if it is not "End". Then, remove the zero from the list by index (priority) and insert the note in its place:

```
3  while True:
4      command = input()
5      if command == "End":
6          break
7      tokens = command.split("-")
8      priority = int(tokens[0]) - 1
9      note = tokens[1]
10     notes.pop(priority)
11     notes.insert(priority, note)
```

Finally, filter only the nonzero elements from the notes list and print the result:

```
13  result = [element for element in notes if element != 0]
14  print(result)
```

## 4. Palindrome Strings

On the **first line,** you will receive words separated by a **single space**. On the **second line,** you will receive a **palindrome**. First, you should print a list containing **all the found palindromes in the sequence**. Then, you should print the number of **occurrences** of the given palindrome in the format: **"Found palindrome {number} times"**.

### Example

| Input | Output |
|---|---|
| wow father mom wow shirt stats<br>wow | ['wow', 'mom', 'wow', 'stats']<br>Found palindrome 2 times |
| hey how you doin? lol<br>mom | ['lol']<br>Found palindrome 0 times |

### Hints

First, read all the **strings** and the **searched** palindrome, and we create an **empty list** for the found **palindromes**:

```
1  strings = input().split(" ")
2  searched_palindrome = input()
3  palindromes = []
```

Then, we create a loop that checks if each word is a palindrome:

```
5  for word in strings:
6      if word == "".join(reversed(word)):
7          palindromes.append(word)
```

- We use the **join()** method with the **reversed()** method because **reversed()** returns an **iterator**, not a **string**, so we should **make it into one**.

Finally, we print the result:

```
9  print(f"{palindromes}")
10 print(f"Found palindrome {palindromes.count(searched_palindrome)} times")
```

## 5. Sorting Names

Write a program that reads a **single string** with **names** separated by comma and space **", "**. Sort the names by **their length** in **descending order**. If 2 or more names have the **same length**, sort them in **ascending order** (**alphabetically**). **Print** the resulting list.

### Example

| Input | Output |
|---|---|
| Ali, Marry, Kim, Teddy, Monika, John | ["Monika", "Marry", "Teddy", "John", "Ali", "Kim"] |
| Lilly, Tim, Kate, Tom, Alex | ['Lilly', 'Alex', 'Kate', 'Tim', 'Tom'] |

## Hints

Read the string and split it:

```
1    names_list = input().split(", ")
```

Use a sorted() function to sort the names by their length first, and then - alphabetically:

```
3    sorted_list = sorted(names_list, key=lambda x: (-len(x), x))
```

Print the final result:

```
5    print(sorted_list)
```

# 6. Even Numbers

Write a program that reads **a single string** with **numbers** separated by comma and space **", "**. Print the **indices** of all **even numbers**.

## Example

| Input | Output |
|---|---|
| 3, 2, 1, 5, 8 | [1, 4] |
| 2, 4, 6, 9, 10 | [0, 1, 2, 4] |

## Hints

Read the string, split it, and convert the list of strings into a list of numbers using map function:

```
1    number_list = list(map(int, input().split(", ")))
```

Use a map function to iterate over the list to find all the even numbers, and add their indices:

```
3    found_indices_or_no = map(
4        lambda x: x if number_list[x] % 2 == 0 else 'no',
5        range(len(number_list))
6    )
```

Use the filter function to filter only the indices and print the result:

```
8    even_indices = list(filter(lambda a: a != 'no', found_indices_or_no))
9    print(even_indices)
```

# 7. The Office

You will receive two lines of input:

- a list of **employees' happiness** as a string of numbers separated by a single space
- a happiness improvement **factor** (single number).

Your task is to find out if the employees are generally happy in their office.

First, **multiply** each employee's happiness by the factor.

Then, **print one** of the following lines:

- If **half or more** of the employees have happiness **greater than or equal to the average**:
  **`"Score: {happy_count}/{total_count}. Employees are happy!"`**
- Otherwise:
  **`"Score: {happy_count}/{total_count}. Employees are not happy!"`**

## Example

| Input | Output | Comment |
|---|---|---|
| 1 2 3 4 2 1<br>3 | Score: 2/6. Employees are not happy! | After the mapping:<br>3 6 9 12 6 3<br>After the filtration:<br>9 12<br>2/6 people are happy, so the overall happiness is bad |
| 2 3 2 1 3 3<br>4 | Score: 3/6. Employees are happy! | Half of the people are happy, so the overall happiness is good |

## Hints

First, **read** the input:

05-the-office.py

```
employees = input().split(" ")
happiness_factor = int(input())
```

Then, use the **`map()`** function to **multiply** each element with **the factor:**

```
employees = list(map(lambda x: int(x) * happiness_factor, employees))
```

- Since all the elements in the employees' list are **strings**, we **parse** them to **integers** before multiplying them.
- Do not forget that the map function returns a **map object**, so we must **cast it to a list**.

Now, it is time to filter the elements that are greater than the average:

```
filtered = list(filter(lambda x: x >= (sum(employees) / len(employees)), employees))
```

- We find the average by **summing** the elements and **divide** the result by its **length**

Finally, we print the result:

```
if len(filtered) >= len(employees) / 2:
    print(f"Score: {len(filtered)}/{len(employees)}. Employees are happy!")
else:
    print(f"Score: {len(filtered)}/{len(employees)}. Employees are not happy!")
```