Programming Test

This test consists of 5 questions on various topics and 1 problem. You have to answer any **2 questions**, and the **problem** in a total of 1,5 hour. The problem is **mandatory**.

Notes:

1. You can return .py files, Jupyter notebooks or .txt files
2. Your code should be interpretable, avoid delivering code which contains syntax errors
3. If you can't solve the problem completely, deliver what you have and comment the status
4. SQL should be close enough to execution. Any reasonable SQL standard is fine.
5. You can use any libraries you deem necessary or adapted to the resolution.

Try to get as much done as you can but prioritize your delivery as explained above!

1. Sums (Python) – time estimate 10mn

Write a function `check_sum()`, which, given `li`, a list of integers, and `num`, returns whether any two numbers in `li` add up to `num`. There are no duplicates in `li`.

Provide a solution that is faster than O(n^2) time. Example:

```
>>> check_sum(li=[11, 1, 2, 3, 10], num=21)
True
>>> check_sum(li=[11, 1, 2, 3],     num=21)
False
```

2. Overlaps (Python) – time estimate 15mn

Write a function `check_overlap()`, which given li, a list of intervals like `[2, 4]`, returns whether any two intervals overlap. Boundary overlaps don't count.

Example:

```
>>> check_overlap(li=[[1,5], [8,9], [3,6]])
True
>>> check_overlap(li=[[1,5], [5,6]])
False
```

3. String replace (Python) – time estimate 20mn

Write a function `replace_latest_slice()` which takes a string `query`, and changes all occurrences of `latest_slice::<table>` to `(SELECT * FROM <table> WHERE ds = 'LATEST')`. Use regexps if you want.

Example:

```
>>> replace_latest_slice(query="""SELECT * FROM
latest_slice::sale_order so INNER JOIN latest_slice::res_partner rp ON
so.id = rp.id""")
SELECT * FROM (SELECT * FROM sale_order WHERE ds = 'LATEST') so INNER
JOIN (SELECT * FROM res_partner WHERE ds = 'LATEST') rp ON so.id =
rp.id
```

4. SQL – time estimate 20mn

Given two tables:

- `deliveries(id INT, day DATE, driver_id INT)`
- `drivers(id INT, name VARCHAR, city VARCHAR)`

Write an SQL query that

A. counts for each driver, 2017 September deliveries that happened in Dubai (stats).

B. counts how many drivers made more than 1,000 overall deliveries in Dubai (top drivers).

C. counts how many days driver `John Smith` made deliveries in 2017 September (active days).

D. counts deliveries on `2017-09-01` that were assigned to a non-existent driver (errors).

5.Occurences – time estimate 15mn

Define a function percent(word, text) that calculates how often a given word occurs in a text, and expresses the result as a percentage.

(note: this question is open, on purpose, provide your assumptions via comments in the code)

A. Problem – time estimate 45mn

We can define 2 special types of strings (both consisting of lowercase English letters) :
two_string : string of length 2 (example: "ab","zz")
three_string: string of length 3 (example: "abc","ghw").

You can create new strings by placing one or several two_strings and three_strings side by side in a line and without placing two same strings next to one another.
From the example above you can create for example "abzz", "ababc", "zzabcghwab" etc.

Given a string S, think of all the ways in which that string S could have been built with the above method and enumerate all the distinct two_strings and three_strings you would need for all these different ways of building S.
For example given a string abcdef there are two ways to build (ab, cd, ef) (abc, def) so the list of distinct strings is ['ab', 'cd', 'ef', 'abc', 'def'].

Write a function split_string() which, given a string S of length N, returns the list of all distinct two_strings and three_strings that could be used for building S, and an empty list if there is no solution.

Constraints:
N<100
Adjacent two same strings are forbidden

Example :

```
>>> split_string(S="abcdef")
['ab', 'abc', 'cd', 'def', 'ef']

>>> split_string(S="abcdefg")
['ab', 'abc', 'cd', 'cde', 'de', 'efg', 'fg']

>>> split_string(S="ababcabc")
['aba', 'abc', 'bc', 'bca']

>>> split_string(S="ccccacccc")
['cac', 'ccc']
```