Project 2

Important Note: In this project (and thereafter) you should NOT use **`input()`** function in any of your problems unless it is explicitly told in the problem. You will get your inputs as function parameters.

Problem 1 - Recurse with me

Write a program that will find and return the number with the given row (i-th) and column (j-th) index using the following rules.

- `i`, and `j` show the row and column indexes respectively.
- `f(i, j)` function is given as: `f(i,j) = f(i-1, j-1) + f(i-1, j)`
- `f(i=1, j=1) = 1`
- `f(i, j=1) = 3 for i > 1`
- `f(i, j=i) = 2 for i > 1`
- `j <= i`
- Function parameter names will be **row**, and **column**. Make sure to have these for any credit.
- `inputs : `**`row, column`**` ∈ [1:100]`
- `output : integer`

```
>>> problem1(1, 1)
1

>>> problem1(4, 1)
3

>>> problem1(9, 1)
3

>>> problem1(row=5, column=5)
2

>>> problem1(8, 8)
2

>>> problem1(8, 3)
57

>>> problem1(12, 7)
1134
```

Problem 2 - Similarity

Write a program that takes two parameters as strings and returns the number of characters that appear in the same index.

- If they share no characters in the same index, return 0
- One string might be longer than the other, so take necessary precautions.
- Function parameter names do not matter, but function should expect 2 parameters.
- `inputs: s1, s2 = {s1, s2 : printable characters except whitespaces and len(s1), len(s2) ∈ [1, 200] }`
- `outputs: integer`

```
>>> problem2('tarkan', 'gurkan')
4

Explanation: both strings have rkan in common (same index)


>>> problem2('kesit', 'telas')
1

Explanation: both strings have only character e in common in the same index.


>>> problem2('ke@', 'telas')
1

Explanation: both strings have only character e in common in the same index.


>>> problem2('k', 'tekkk')
0

>>> problem2('telas', 'ke')
1

>>> problem2('!telas', 'k')
0
```

Problem 3 - Transfer

Write a function that will transfer money between given source and destination accounts and return the updated accounts. The function should take six named arguments:

- `accounts:` a list of strings that will hold the money on all the accounts
- `source:` the source index which the money will be transferring from
- `destination:` the destination index which the money will be transferring to
- `lira:` the amount that will hold the lira part of the transfer
- `kurus:` the amount that will hold the kurus part of the transfer
- `fee:` `True` or `False`. Should default to `False`. This will be used to apply transfer fees to the transaction or not.

Accounts will be a list of strings that will hold all the accounts of the customers. The strings will be in the form of `"lira.kurus"` and you need to make appropriate conversions to do any operations on these strings. The final accounts list that is returned will also be in the same form. An example accounts list would be `["11.23", "10.43", "100.63", "0.10"]`.

- Index starts from 0.
- The transfer amount should be subtracted from the source, and added to the destination.
  - If the `fee` parameter is set to `True`, the transfer fee should be subtracted from the source.
- The transfer fee should be `0.1` for any amount less than `10.0` liras, for any amount that is greater than `10.0` liras, the transfer fee should be `1%` of the transfer amount.
  - Careful with floating point operations here. (i.e. `1%` of `115` should be `1.15`, and not `1.1500000000000001`). Take the first two digits after the dot. (i.e. `0.1013` will be `0.10`)
- If the source and the destination indexes are the same, return `the accounts` list with no change.
- If the source or the destination are less than 0 or greater than the length of the list return `the accounts` list with no change.
- If the transition is not doable (i.e. account has less money than the requested amount) return `the accounts` list with no change.
  - If the `fee` parameter is set to `True`, and if the transition + the transfer fee is not doable (i.e. account has less money than the requested amount + the transfer fee) return `the accounts` with no change.
- `lira` part of the string should NOT have any leading 0's. (ie. should be "2" instead of "02")
- `kurus` part of the string should have the preceding 0's unless it is actually 0. If it is 0, just keep 0. (ie. should be ".20" instead of ".2", ".40" instead of ".4", and ".0" instead of ".00")
- Function parameter names should be `accounts, source, destination, lira, kurus, fee`.
- `inputs :`
  - `accounts = { x : x is list of strings and len(x) ∈ [1:200]}`
  - `source, destination ∈ ℤ`
  - `lira = { x : x ∈ [0:1E6]}`
  - `kurus = { x : x ∈ [0:99]}`
  - `fee ∈ {True, False}`
- `outputs : accounts = { x : x is list and len(x) ∈ [1:200]}`

```
>>> problem3(accounts=["11.23", "10.43", "100.63", "0.10"], source=0,
destination=0, lira=10, kurus=13)
["11.23", "10.43", "100.63", "0.10"]

>>> problem3(accounts=["11.23", "10.43", "100.63", "0.10"], source=3,
destination=0, lira=10, kurus=13)
["11.23", "10.43", "100.63", "0.10"]

>>> problem3(accounts=["11.23", "10.43", "100.63", "0.10"], source=0,
destination=1, lira=9, kurus=13, fee=True)
["2.0", "19.56", "100.63", "0.10"]

>>> problem3(accounts=["11.23", "10.43", "100.63", "0.10"], source=0,
destination=1, lira=10, kurus=13, fee=True)
["1.0", "20.56", "100.63", "0.10"]
```

Explanation: %1 of 10.13 is 0.1013. Taking the first two digits after dot gives: 0.10.