IT5001 Practical Exam

Instructions:

- If you failed to submit your screen recording or failed to keep your proctoring camera, your PE will result in **ZERO mark** no matter what you submitted onto Coursemology.
- This PE consists of **THREE** parts and each part should be submitted in a separated file. Namely `part1.py`, `part2.py` and `part3.py`. Each file is independent, e.g. you cannot expect that the answer in `part2.py` calls a function in `part1.y` that is not in `part2.y`. If you want to reuse some function from another file, you need to copy them into that file. For example, you just need to copy the function you want to call in `part1.py` into `part2.py`.
- No mark will be given *if your code cannot run*, namely any syntax errors or crashes.
  - Your tutors will just grade what you have submitted and they will **not** fix your code.
  - Comment out any part that you do not want.
- Workable code that can produce correct answers in the given test cases will only give you *partial* marks. Only good and efficient code that can pass ALL test cases will give you full marks. Marks will be deducted if your code is unnecessarily long, hard-coded, or in poor programming style, including irrelevant code or test code.
- You must use the same function names as those in the skeleton files given. Also, you must NOT change the file names.
- You ***cannot import*** any additional packages or functions.
- Your code should be efficient and able to complete *each* example function call in this paper within 2 seconds.
- In all parts, you should **return** values instead of **printing** your output. In another word, you should not need any "`print()`" in this entire PE for submission.
- You should remove all your test cases before submission. If you submit more code than required, it will result in **penalty** because you code is "unnecessarily long".
- Reminder: Any type of plagiarism like copying code and then modify will be caught, that include adding comments, changing variable names, changing the order of the lines/functions, adding useless statements, etc.

Constraints for Part 1 and Part 2

You cannot use the following Python ***built-it*** string/list functions in Part 1 and Part 2:

- `encode(), decode(), replace(), partition(), rpartition(), split(), rsplit(), translate(), map(), count(), find(), join(), rfind(), rsplit(), rstrip(), strip(), sort(), sorted(), pop()`

Or any sort functions and list comprehensions. Also, your code cannot be too long. Namely, the function in *each* task must be fewer than 400 characters and 15 lines of code including all sub-functions that called by it.

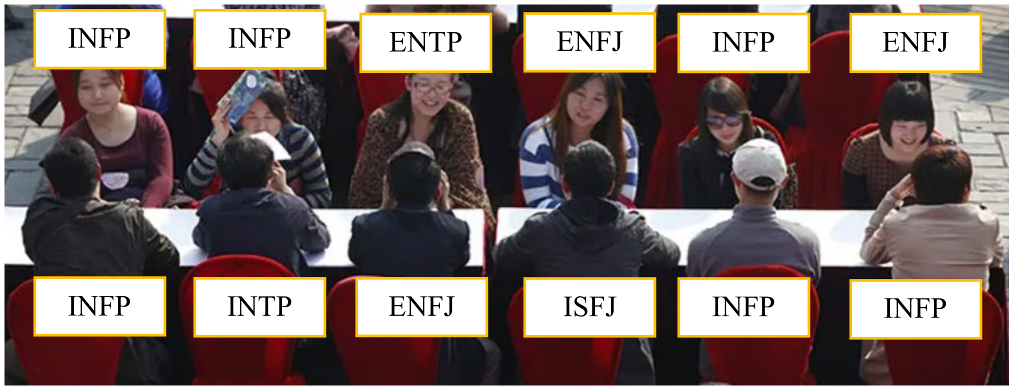
Part 1 Speed Dating Matches

Before we start "dating", let's talk about the 16 types of personality profiles, namely,

INTJ, INTP, ENTJ, ENTP, INFJ, INFP, ENFJ, ENFP, ISTJ, ISFJ, ESTJ, ESFJ, ISTP, ISFP, ESTP and ESFP.

Make it short, each personality profile is made up of four letters. And we said two persons with two personality profiles are compatible if they have more or equal to 3 letters are the same

We are organizing a group match making for **n** guys and **n** gals. The guys will sit on one side of the table and the gals will be on the other side. Each of them already took a test and here are their personality profiles:



The above data will be given as two lists:

```
f1 = ['INFP','INFP','ENTP','ENFJ','INFP','ENFJ']
m1 = ['INFP','INTP','ENFJ','ISFJ','INFP','INFP']
```

From *some* research, it said that if a couple with the exactly same personality profile, they will be a good match!

In this first stage, they are not allowed to move around. According to the above example of seating positions and personality profiles, you can know that there are two pairs of good matches and four pairs of bad matches. In this Part, you can assume the two lists have the same length always.

Task 1 (15 marks)

Write an ***iterative*** version of the function `goodMatchI(fpp,mpp)` to compute the results of good or bad matches. Your function should return a string with the upper case letter `'C'` or `'X'` only, in which, `'C'` stands for being capable and `'X'` otherwise. In this task, you cannot use any recursion. Here is some sample outputs.

```
>>> f1 = ['INFP','INFP','ENTP','ENFJ','INFP','ENFJ']
>>> m1 = ['INFP','INTP','ENFJ','ISFJ','INFP','INFP']
>>> print(goodMatchI(m1,f1))
CXXXCX
```

(The green color in the box above is just for you to see it clearly. You do not need to return a string with green color.)

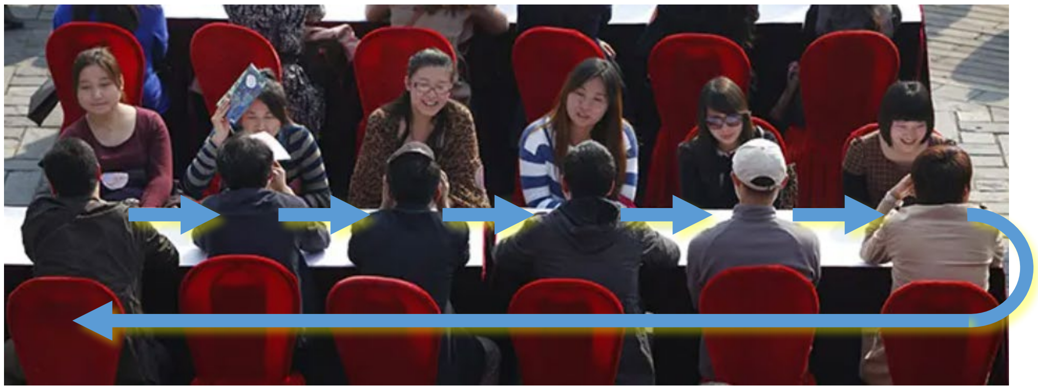
Task 2 (15 marks)

Write a ***recursion*** version of the function `goodMatchR(fpp,mpp)` with the same functionality in Part 1 Task 1. However, you cannot use any loops or list comprehension in this task.

Task 3 (15 marks)

In this part, it doesn't matter if you use iterations or recursions.

For the next stage of speed dating, they are allowed to rotate. In order to be chivalrous, only the guys will move. And they will ONLY move in the following manner: For each ***turn***, every guy will move to the right by one seat except that the rightmost guy will move to the leftmost seat. After 5 min. of conversation, they will do another ***turn*** by this rotation for n times if there are n guys or gals.



Your task is to write a function `maxRotationalGoodMatch(fpp,mpp)` that will return the integer that is the maximum number of good matches. For example, the `m1` and `f1` lists in the above sample run have only 3 good matches without any rotation. However, the maximum number of good matches is 3 after *ONE* rotations. (The right most 'INFP' guy moves to the leftmost position and the rest of the guys moved right by one seat.) Illustrated by the following shifting of profiles:

```
['INFP','INFP','ENTP','ENFJ','INFP','ENFJ']
['INFP','INFP','INTP','ENFJ','ISFJ','INFP',~~'INFP'~~]
```

Here is some sample run

```
>>> print(maxRotationalGoodMatch(f1,m1))
3
```

Part 2 Full Speed Dating (25 marks)

Note that if you need any function(s) from Part 1 in this part, you have to copy the function(s) into part2.py from part1.py again. And please take notes of the forbidden functions in this part in the “constraints” mentioned above. However, the hint is, you should not need any functions from Part 1.

In this part, it is the same setting in Part 1, except that the guys can move around freely now. For each turn, a guy can find a different gal to talk with randomly. However, for each turn, it must be a one gal to one gal manner. (You cannot get two guys to talk to one single girl or one guy talks to more than one girl at a time.) Your task is to write a function `maxAnyCombGoodMatch(fpp,mpp)` to find the maximum number of good matches.

```
>>> print(maxAnyCombGoodMatch(f1,m1))
4
```

Part 3 The Chinese nth Auspicious Numbers

In some culture, some digits are considered to be auspicious (meaning lucky, prosperous or favorable). In China, the number ‘8’ sounds like ‘prosperous’ and the number ‘9’ sounds like ‘everlasting’. They are said to be the lucky numbers.

So starting from 1, the first auspicious number is 8. The second and the third ones are 9 and 88 respectively.

Task 1 (20 marks)

Write a function `nthAuspiciousNum(n)` to return the integer that is the nth auspicious number. Here are some sample runs:

```
>>> print(nthAuspiciousNum(1))
8
>>> print(nthAuspiciousNum(3))
88
>>> print(nthAuspiciousNum(1))
8
>>> print(nthAuspiciousNum(2))
9
>>> print(nthAuspiciousNum(3))
88
>>> print(nthAuspiciousNum(10))
899
>>> print(nthAuspiciousNum(59))
99988
```

Task 2 [Challenge] Need some extra efforts to solve this. (10 marks)

You got this part right if your code can compute the following correct answers within 1 second.

```
>>> print(nthAuspiciousNum(10**20))
898998989999888999898999988898998989988899888988888888888888888889
>>> print(len(str(nthAuspiciousNum(10**1000))))
3321
```

(You may need to press Ctrl-C to stop your execution if it runs too long on your computer.)

- End of the Paper -