CS1010E Practical Exam II

Instructions:

- If you failed to keep your proctoring camera, your PE will result in **ZERO mark** no matter what you submitted onto Exemplify or Coursemology.
- You ***cannot import*** any additional packages or functions, or you will get zero mark.
- This PE consists of **FOUR** parts and each part should be submitted seperately in Examplify and Cousremology. It's better for you to organize your files into `part1.py`, `part2.py`, `part3.py` and `part4.py` and each file is independent, e.g. you cannot expect that the answer in `part2.py` calls a function in `part1.py` that is not in `part2.py`. If you want to reuse some function(s) from another file/part, you need to copy them (as well as their supporting functions) into that file/part. For example, you just need to copy the function you want to call in `part1.py` into `part2.py`.
- You are allowed to write extra functions to structure your code better. However, remember to submit them together mentioned in the point above.
- You should not declare any global variables.
- No mark will be given *if your code cannot run*, namely any syntax errors or crashes.
    - We will just grade what you have submitted and we 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 template given.
- 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 either delete all test cases before submission or comment them out by adding # before the test cases. If you submit more code than required, it will result in **penalty** because your code is "unnecessarily long".
- You should save an **exact** copy of your submission in your computer for submitting in Coursemology again ***after*** the PE. Any differences between Exemplify and Coursemology submissions will be severely penalized, except for indentation differences.
- Reminder: Any type of plagiarism like copying code with modifications will be caught, that include adding comments, changing variable names, changing the order of the lines/functions, adding useless statements, etc.

Summary of this Paper

There are four parts in this PE

- Part 1 (25 marks)
- Part 2 (20 marks)
- Part 3 (25 marks)
- Part 4 (30 marks)

Constraints for Part 1

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

- `split, rsplit, join, map, filter.`

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. You cannot use any `list`, `tuple`, `dict` or `set` and their functions in these tasks.

Part 1 Permutation Cipher (10+10+5 = 25marks)

Permutation Cipher is a form of encryption/decryption technique for information security. For example, we want to encrypt the following message

`'PYTHON CODING IS VERY USEFUL'`

One way is to use permutation cipher. Given an integer n>0, we group the above message into groups of n characters. For example, let n = 4, the above message will be divided into the following strings,

`'PYTH', 'ON C', 'ODIN', 'G IS', ' VER', 'Y US', 'EFUL'`

And then, we *reverse* each set of the strings and they become

`'HTYP','C NO','NIDO','SI G','REV ','SU Y','LUFE'`

Finally, merging them together will be our decrypted message,

`'PYTHON CODING IS VERY USEFUL' → 'HTYPC NONIDOSI GREV SU YLUFE'`

Note that if we apply the same 'encryption' to the encrypted message, the message will be decrypted.

If the length of message is not a multiple of n, we just reverse whatever at the end of the message. E.g. if n = 4,

`'PYTHON' → 'HTYPNO'`

Task 1 Iterative Permutation Cipher (10 marks)

Write an ***iterative*** version of the function `per_cipher_i(s,n)` to encrypt a string `s` with an interval n as mentioned above. In this task, you cannot use any recursion. Here is some sample output.

```
>>> print(per_cipher_i('12345678910',3))
32165498701
>>> print(per_cipher_i(per_cipher_i('12345678910',3),3))
12345678910
>>> print(per_cipher_i('PE Part 1 is supposed to be easy',7))
traP EPs si 1 desoppu eb ot ysae
```

Task 2 Recursive Permutation Cipher (10 marks)

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

Task 3 One-liner Permutation Cipher (5 marks)

Write a one liner version of the function `per_cipher_o(s,n)` with the same functionality in Part 1 Task 1. A one-liner function is a function with only one line of function body without any colon or semi-colon, and less than 80 characters in the body.

Part 2 Pizza Map Property Agents (20 marks)

Preparation

This part is not graded and not part of the submission. In order to create two map files for you to test your code. Please run the function `create_map_file()` in your directory. This function will create two files, namely `'pmap1.txt'` and `'pmap2.txt'` for our sample input. These two files are for your testing only and you DO NOT need to submit these two files nor the `create_map_file()` function. We give you this is just because we cannot distribute extra files by Examplify.

Task 1 Property Realty Agent (20 marks)

Remember our Pizza map assignment? We will continue on that.

But don't worry, we won't need you to redo the assignment. We will give you the final map you produced in the assignment in a text file. For example, you will find one map with the file name `'pmap1.txt'` that contains the map on the right.

```
0000011111
0000X11111
0000111111
000X111111
0001111111
0222111111
2222211111
2222222111
2222222211
2222222222
2222222222
```

Just a recap that each entry contains the shop number with the shortest distance to that location. And an `'X'` represent that the location has more than 1 shop that has the same shortest distance.

Because of the love of pizzas in the town, people want to buy the houses with the locations that can buy more types of pizzas form different shops. And there is one tradition in the pizza town that any household may ask his neighbor to buy pizzas for them. If your house is in an `'X'` location *or* your house has a neighbor that has a different shortest distance pizza shop from your house, then your house is very ***buyable***! By "neighbor", it means the eight locations that is next to the current home.

On the other hand, if your house is not at the `'X'` location and all of your neighbors have the same shortest distance pizza shops as yours, your house is ***unbuyable***. So the buyable map of the above file `'pmap1.txt'` will be this second map on the right. You can see that the ***unbuyable*** position is replaced with a `'.'` character.

```
...001....
...0X1....
..0011....
..0X1.....
00011.....
022211....
22.22111..
....22211.
......2211
.......222
..........
```

Write a function `buyable_map(filename)` that will take in a string that is the filename of the pizza map text file. Read that file and return a single string that is the buyable map of that town. Your output single string will include newlines characters to represent a 2D map. So the above output buyable map of `'pmap2.txt'` is actually a string shown as below:

```
'...001....\n...0X1....\n..0011....\n..0X1.....\n00011.....\n022211....\n22.22111
..\n....22211.\n......2211\n.......222\n..........\n'
```

Note that, you should only return the string and there should be no "`print()`" function call is involved.

Part 3 Sum of Three Numbers (15 + 10 = 25 marks)

This part is similar with one of our assignment problems. Guess which?

Task 1 (15 marks)

Given a tupe `L` of ***unique*** integers that are larger than 0, and another integer `n`. And it means that no two numbers in `L` are the same. Write a function `sum_of_3(L,n)` to return `True` if there exists 3 numbers in `L` with their sum equals to `n`, and return `False` otherwise. Here are some sample outputs.

```
>>> print(sum_of_3((11,14,13,12),39)) # 12+13+14 == 39
True
>>> print(sum_of_3((11,14,13,12),40))
False
```

Note that the numbers in `L` cannot be "reused" for more than one time for a single number `n`. For example, in the second function call above, the combination of `14 + 14 + 12 == 40` is NOT counted as a valid answer.

Also, you should not modify the input tuple `L` if `L` is mutable.

Task 2 (10 marks)

You need to handle tuples with large sizes within 1 second, e.g. `len(L) > 4000`.

```
>>> print(sum_of_3(tuple(range(1,1000)),2500))
True
>>> print(sum_of_3(tuple(range(1,1000)),2998))
False
>>> print(sum_of_3(tuple(range(1,4000)),11994))
True
>>> print(sum_of_3(tuple(range(1,4000)),11995))
False
```

Part 4 Ancient Martian DNA (5 + 15 + 10 = 30 marks)

Before we start, you can only get full mark in each part if your function can handle large numbers.

Long long time ago in Mars, every Martian can encode their DNA into a positive integer. And it is very interesting that when a Martian give birth to a child,

- There is no zero in the DNA
- Each Martian child only needs one parent to be reproduced.
- The child's DNA is the ***digit product of*** his parent's DNA. For example, if a parent's DNA is 262, then his son's DNA will be

$$2 \times 6 \times 2 = 24.$$

- The parent will mutate after giving birth to a child. **His DNA will change to the *smallest integer* that produce the same digit product of his child**. E.g. if the parent's DNA is 262, his own DNA will mutate to 38 after giving birth.
- If the DNA of a Martian contains a zero in his DNA, he will not give birth to any child anymore.
- Don't ask me where did these Martians originated from…

Task 1 Child's DNA (5 marks)

Given a parent's DNA, what is the child's DNA? Write a function `child_DNA(d)` to return the child DNA from the parent DNA `d`. You can assume `d > 10`. Some example output:

```
>>> print(child_DNA(262))
24
>>> print(child_DNA(987161))
3024
```

Task 2 Parent's DNA Mutation (15 marks)

Given a parent Martian's DNA, what will his mutated DNA be after he gave birth?

```
>>> print(parent_mutated_DNA(262))
38
>>> print(parent_mutated_DNA(12131))
6
```

Task 3 Traitor Discovery (10 marks)

Oh dear, there are some traitor aliens mixed into the group of Martians!!!! They may be harmful to the Martian society. However, you may be able to discover the aliens if you check their DNAs because you cannot find a suitable Martian parent. Implement a function `isMartian(d)` to return `True` if there exists a possible parent for the creature with DNA `d`, or `False` otherwise.

```
>>> print(isMartian(3024)) # The possible parent has DNA=9786
True
>>> print(isMartian(16632)) #no possible parent can produce a child DNA=16632
False
```

Appendix

```
# Skeleton Code for Part 1
def per_cipher_i(s,n):
    return
def per_cipher_r(s,n):
    return
def per_cipher_o(s,n):
    return
print(per_cipher_i('12345678910',3))
print(per_cipher_i(per_cipher_i('12345678910',3),3))
print(per_cipher_i('12345678910',4))

print(per_cipher_r('12345678910',4))
print(per_cipher_r(per_cipher_r('12345678910',3),3))
print(per_cipher_r('12345678910',1))

print(per_cipher_o('12345678910',4))
print(per_cipher_o('PE Part 1 is supposed to be easy',7))
print(per_cipher_o(per_cipher_o('This is the Part 1 of the PE',6),6))

########################################################################
# Skeleton Code and Map Preparation for Part 2
def create_map_file():
    pmap1list =  ['0000011111',
     '0000X11111',
     '0000111111',
     '000X111111',
     '0001111111',
     '0222111111',
     '2222211111',
     '2222222111',
     '2222222211',
     '2222222222',
     '2222222222']
    pmap2list = ['000000X22222',
     '00000X222222',
     'XXXXX2222222',
     '111122222222',
     '111122222222',
     '111X22222222',
     '111222222222',
     '111222222222',
     '11X222222222',
     '112222222222']
    f1 = open("pmap1.txt","w")
    for line in pmap1list:
        f1.write(line+'\n')
    f1.close()
    f2 = open("pmap2.txt","w")
    for line in pmap2list:
        f2.write(line+'\n')
    f2.close()
create_map_file()
def buyable_map(filename):
    return
print(buyable_map('pmap1.txt'))
print(buyable_map('pmap2.txt'))
```

```
#########################################################################
# Skeleton Code for Part 3
def sum_of_3(L,n):
    return

print(sum_of_3((11,14,13,12),39)) # 12+13+14 == 39

print(sum_of_3((11,14,13,12),40))

#########################################################################
# Skeleton Code For Part 4
def child_DNA(n):
    return

def parent_mutated_DNA(parent_original_dna):
    return

def isMartian(n):
    return
print(child_DNA(982374972))
```