


CS1010E Practical Exam 2

Instructions:

- In Exemplify, in order to view this exam paper in full screen and to copy & paste some of its content to another file, you need to click on the “setting” clog-like button  at the right side, then click on the “Print” button to display the document using PDF reader. Then, you can copy-&paste the code template (in the last page) from this printed document to your respective .py files.
- You **cannot import** any additional packages or functions, or you will get zero mark.
- **You cannot use any global variable.**
- For Part 2 and Part 3, please feel free to use any built-in function.
- This PE consists of **THREE** parts and each part should be submitted separately in Exemplify and Coursemology. It’s better for you to organize your files into `part1.py`, `part2.py` and `part3.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 should save an **exact** copy of your Exemplify 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. If we found out that there are differences between the two copies, the copy on Exemplify will be marked instead of the Coursemology copy.
- You are allowed to write extra functions to structure your code better. However, remember to submit them together mentioned in the point above.
- 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 1 second.
- 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”.
- 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, or restructuring your code, etc.

Part 1 Encryption (30 marks)

The basic of encrypting a string is to offset each alphabet by a number. For example to offset the alphabet 'd' by 3 you will get the new alphabet 'g' because

'd' → 'e' → 'f' → 'g'

Given a *message* string *s1*, a *number* string *s2* and a multiplier *m*, you are going to offset each alphabet by the corresponding number multiplied by *m*. For example

```
>>> print(encode_I('bye', '128', 2))
dcu
>>> print(encode_I('dcu', '128', -2))
bye
```

In the first example, the three offsets of the three alphabets are $1 \times 2 = 2$, $2 \times 2 = 4$ and $8 \times 2 = 16$. Therefore,

- The letter 'b' is offset by 2 and it becomes 'd',
- The letter 'y' is offset by 4 and becomes 'c' because 'y' → 'z' → 'a' → 'b' → 'c'. It will wrap around back to 'a' after 'z'.
- And finally, the letter 'e' becomes 'u'.

Note that, if the multiplier is negative, it will “decode” the message in the second example.

You can assume that

- The message string *s1* will only contain lower case alphabets only or nothing
- The number string *s2* will contain the digits 0 to 9 only or nothing
- The length of the message string and the number string will be the same
- The multiplier *m* can be any integer, including 0 or negative numbers.

Task 1 Iterative Encoding (10 marks)

Write an **iterative** version of the function `encode_I(s1, s2, m)` to **return** the encoded **string** according to the above rules.

```
>>> print(encode_I('spyxfamily', '222222222', 1))
urazhcokna
>>> print(encode_I('krmxhaeaba', '9170109981', -2))
spyxfamily
```

In this task,

- You must write your function with for-loops or while-loops only
- Your function cannot be recursive in this task.
- You cannot use list comprehension, `map()`, `translate()`, `reduce()`.

Task 2 Iterative Encoding (10 marks)

Write a **recursive** version of the function `encode_R(s1, s2, m)` with the same functionality in Part 1 Task 1. However, you cannot use any loops, `map()` or list comprehension in this task, e.g no “while” nor “for”. You have to use recursion technique here.

Task 3 Ultimate One-liner Encoding (10 marks)

Write a one-liner function `encode_U(s1, s2, m)` with the same functionality in Part 1 Task 1 in the form:

```
def encode_U(s1,s2,m):  
    return #your one line code here.
```

Your function has to be independent, e.g. cannot use other function(s) written by you like “return encode_R(s1,s2,m)”. You cannot use recursion in this task.

Preparations

- Please copy and paste the code `preparemaps()` provided and run it to create the two text files `'map1.txt'` and `'map2.txt'`.) Of course, this function will only generate two map examples. Your function should be able to handle other map files
- A function `readmap(filename)` is provided for you to read in a text file and return a 2D array. It is not necessary to use this function if you have other way to solve this part. However, please include this function in your submission if you are using it.

A king has many sons, a.k.a. princes, and he is going to give each of them an island in the kingdom. Each of the princes is represented by an ASCII character except the letter 'W' and the period '.'. E.g. if the king has 3 sons represented by 'A', 'B' and 'C', the map will be like the following and given in a text file (map1.txt):

```

WWWWWWWWWWWWWWWWW.WWWWWWWW
WWWWWWWWW.WWWWWWWW..WWW
WWWW...WWWB.WWW
WWWWWW.WWWWWWWW..WWW
WWWWWW...WWW
WW.....WWW
WWWW...A.WWWWWWWW.WW
WWWW...WWW
WWWWWWWWWW.C.WWW
WWWW.WWWWWWWW..WWW
WW...WWW
WWWWWWWWWWWWWWWWWW

```

Task 1 Cropping Map

Write a function `crop_map(filename,minr,maxr,minc,maxc)` to crop a map and **return** a list of strings from the map file `filename`. The function should return the portion of the map from rows `minr` to `maxr-1`, and from columns `minc` to `maxc-1`. If the 4 dimensional parameters are 'out of range', your program should not crash but only output the portion of the map within that range. You can assume `maxr >= minr` and `maxc >= minc` always.

```

>>> pprint(crop_map("map1.txt",5,9,3,25))
['.....WWWWWWWWWWWWWWWW',
 'W...A..WWWWWWWWWWWW..WW',
 'WW.....WWWWWWWW..WWWW',
 'WWWW..WWWWWWWWWW.C.WWWW']
>>> pprint(crop_map("map1.txt",7,10,-10,40))
['WWWWW.....WWWWWWWW..WWWWW',
 'WWWWWWWW..WWWWWWWWWW.C.WWWWW',
 'WWWW..WWWWWWWWWWWW..WWWWW']
>>> pprint(crop_map("map1.txt",3,3,4,4))
[]
>>> pprint(crop_map("map0.txt",5,9,2,2))
[]
>>> pprint(crop_map("map1.txt",-10,100,3,4))
['W', 'W', 'W', 'W', 'W', '.', 'W', 'W', 'W', 'W', '.', '.']

```

Note that you should not have any empty string in your cropped output if the width of the cropped area is 0.

Task 2 Computing Island Area

Write a function `island_area(filename,prince)` to read in a map file named 'filename' and **return** the **area** of that prince represented by `prince` in the unit of km² as an **integer**. Also, the variable `prince` is the name of the prince and it is always a string of one visible character except 'W' or '.'. (By visible we mean that they are not something like '\n' or space ' '). If the prince is NOT in the map, then return 0. Here is some example output (with the right figure as the file "map2.txt"):

```

>>> print(island_area("map1.txt","A"))
29
>>> print(island_area("map1.txt","B"))
11
>>> print(island_area("map1.txt","C"))
9
>>> print(island_area("map2.txt","A"))
21

```

map2:

```

WWWWWWWWWWWWWWWWWWWW
WWWWW.WWWWWWWWW..WWWWW
WWWWWWWWWWWWWWWW.B..WWWW
WWWWW.WW.WWWWWWW..WWWWW
WWWWW..WWWWWWWWWW..WWWWW
W.....WWWWWWWWWWWWWWWW
W...A..WWWWWWWWWWWW..WWWW
WW.....WWWWWWWWWW..WWWWW
WWWWWWWWWWWWWWWWWWWW

```

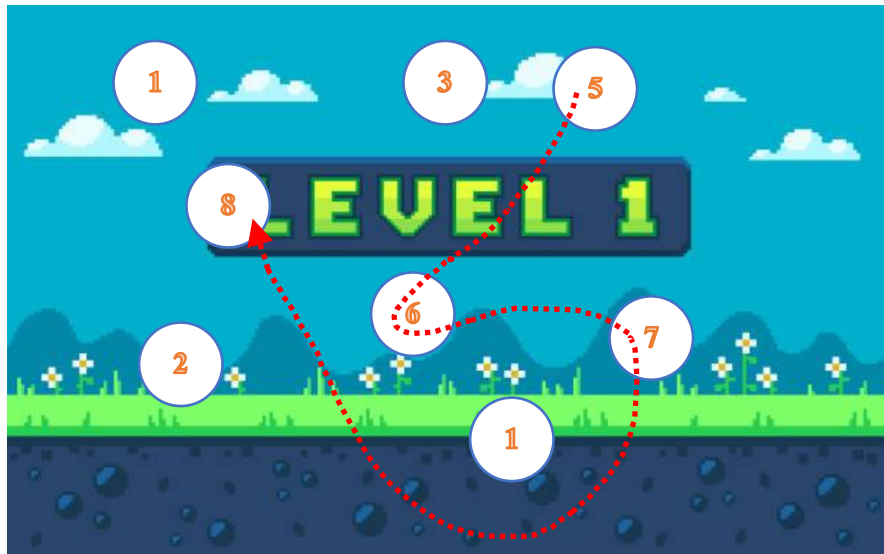
Task 3 Prince Map

Write a function `prince_map(filename,prince)` to crop the *smallest rectangle* of the map portion that contains the island of that prince only. The output of the function should be a list of strings and each string is one row of the cropped map. If the prince is NOT in the map, return an empty list. Here is some example output:

```
>>> pprint(prince_map('map1.txt','A'))
['WW...WWW',
 'WWW.WWWW',
 'WWW.....',
 '.....WW',
 'W...A..WW',
 'WW.....WW',
 'WWW..WWW']
>>> pprint(prince_map('map1.txt','B'))
['W.WWW', '...WW', 'W.B..', '...WW']
>>> pprint(prince_map('map1.txt','C'))
['...', '.C.', '...']
```

Part 3 Clear the Numbers Game (30 marks)

In a video game, the screen will show a lot of integers on the screen. A player needs to clear all the numbers as fast as possible. In each **turn**, a player can use a mouse pointer to link up a **straight** of consecutive integers in ascending order. For example, given the following numbers, the player can clear 5, 6, 7, 8 in one turn.



The next turn we can clear 1, 2 and 3.



Then with one more turn to clear the last remaining 1. Therefore, the player uses **3 turns** to finish this game. Of course, the player may use more turns to clear all the number. However, in order to win the game, can you use the **minimum number of turns** to clear the game and win?

Write a function `min_no_of_turns(L)` to **return** the minimal number of turns to finish the game as an integer. The input `L` is a **tuple** of collection of the integers provided in the beginning of the game.

```
>>> print(min_no_of_turns ((1, 8, 3, 6, 5, 7, 2, 1)))
3
>>> print(min_no_of_turns ((1, 8, 3, 6, 5, 7, 2, 1, 4)))
2
>>> tup3 = (6,5,4,3,2,1,11,12,13,14,15,16,6,5,16,16)
>>> print(min_no_of_turns (tup3))
5
```

Note that:

- The input tuple \mathbb{L} will be containing positive integers or nothing only.
- The positive integers can be any integer $x > 0$.

In order to gain the full mark for this question, you need to compute large tuple (length > 100000) in less than 1 second.

--- End of Paper ---s

Appendices: Function Template

```
#Part 1
def encode_I(s1,s2,m):
    pass
def encode_R(s1,s2,m):
    pass
def encode_U(s1,s2,m):
    pass

#Part 2
def island_area(filename,prince):
    pass
def crop_map(filename,minr,maxr,minc,maxc):
    pass
def prince_map(filename,prince):
    pass

#Part 3
def min_no_of_turns(L):
    pass

#Given code for Part 2 There is no need for you to submit these code unless you
are using it

def preparemaps(): # run this to generate two text file, map1.txt and map2.txt
    m1 = ['XXXXXXXXXXXXXXXXXXXXX.XXXXXXXXXX',
          'XXXXXXXXXX.XXXXXXXXXX...XXXXXXX',
          'XXXXXX...XXXXXXXXXXXXX.B..XXXXX',
          'XXXXXXXXXX.XXXXXXXXXXXXXX.XXXXXXX',
          'XXXXXXXXXX.....XXXXXXXXXXXXX.XXXXXX',
          'XXXX.....XXXXXXXXXXXXXXXXXXXXX',
          'XXXXX...A..XXXXXXXXXXXXXXXXX..XXXX',
          'XXXXXX.....XXXXXXXXXXXXX...XXXXXX',
          'XXXXXXXXXX..XXXXXXXXXXXXX.C.XXXXXX',
          'XXXXXX..XXXXXXXXXXXXXXXXXX...XXXXXX',
          'XX....XXXXXXXXXXXXXXXXXXXXX',
          'XXX.XXXXXXXXXXXXXXXXXXXXXX']
    map1file = open('map1.txt','w')
    for r in m1:
        map1file.write(r+'\n')
    m2 = ['XXXXXXXXXXXXXXXXXXXXX.XXXXXXXXXX',
          'XXXXXX.XXXXXXXXXXXXXX.XXXXXXX',
          'XXXXXXXXXXXXXXXXXXXXX.B..XXXXX',
          'XXXXXX.XX.XXXXXXXXXX...XXXXXX',
          'XXXXXX...XXXXXXXXXXXXX..XXXXX',
          'X.....XXXXXXXXXXXXXXXXXXXXX',
          'X...A..XXXXXXXXXXXXXXXXX..XXXX',
          'XX.....XXXXXXXXXXXXX...XXXXXX',
          'XXXXXXXXXXXXXXXXXXXXX']
    map2file = open('map2.txt','w')
    for r in m2:
        map2file.write(r+'\n')
```

```
#This will read in a file and covert it into a 2D array
#You do not need to use it if you have another way to solve Part 2
#However, if you use this code, please remember to include in your submission
def readmap(filename):
    f = open(filename)
    m = []
    for line in f:
        m.append(list(line.rstrip('\n')))
    return m
```