IT5001 Practical Exam

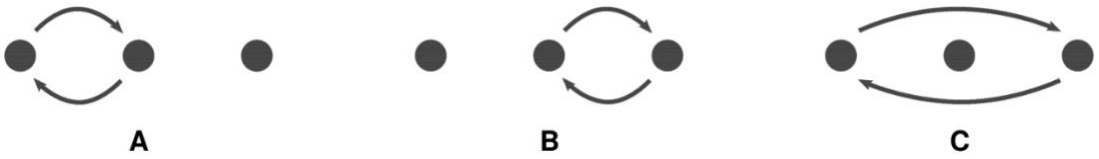
Instructions:

- Your code will be graded and run on **Python 3.10**.
- For this whole paper, you ***cannot import*** any packages or functions and you also cannot use any decorators. However, you are allowed to write extra functions to structure your code better. Do remember to copy them over to the specific part(s) they are supporting.
- No marks will be given *if your code cannot run*, namely if it causes any syntax errors or program crashes.
    - We will just grade what you have submitted and we will **not** fix your code.
    - Remove (or comment out) any lines of code that you do not want us to grade. You should **either delete all test cases or comment them out (e.g. by adding # before each line)** in your code. Submitting more (uncommented) code than needed will result in a **penalty** as your code is "unnecessarily long" in each part.
- Working code that can produce correct answers in public test cases will only give you *partial* marks. Your code must be good and efficient and pass ALL public and hidden test cases for you to obtain full marks. Marks will be deducted if it is *unnecessarily long, hard-coded, in a poor programming style, or includes irrelevant code*.
- The code for each part should be submitted **separately**. Each part is also "independent': if you want to reuse some function(s) from another part, you will need to copy them (and their supporting functions, if any) into the part you are working on. However, any redundant functions for the part (albeit from a different part) will be deemed as “unnecessarily long code".
- You must use the same function name as specified in the question. You must also use the **same function signature and input parameters** without adding any new parameters.
- **You are not allowed to mutate the input arguments.**
- You should `return` your output instead of `print()`.
- Reminder: Any kind of plagiarism such as copying code with modifications will be caught. This includes adding comments, changing variable names, changing the order of lines/functions, adding useless statements, etc.
- **You are not allowed to use any generative AI tools such as ChatGPT in the assessment.**
- Before the PE ends, remember to
    - Save your three parts in three files in your laptop
    - Submit your code to Examplify, and make sure that they are the same as your saved copies
    - **Do not modify any more in Examplify and your saved copies before and after PE ends.**

Part 1 Cup and Ball Trick(10 + 10 + 10 = 30 marks)

We all saw the Cup and Ball game before. A ball was placed under one of the three opaque cups. Someone will keep swapping the cups many times and you have to guess where the ball is finally after the swapping.

Let the three cups be at three positions 1, 2 and 3. There are also three possible swapping actions. Namely, Actions A, B and C. Action A will swap the cups at positions 1 and 2. Action B will swap the positions 2 and 3 and Action C will swap position 1 and 3. These actions are illustrated in following picture:



Task 1 Find the Position after One Move

Given an initial position `pos` of the ball and a move `m`, write the function `next_pos(pos,m)` to return the final position after one move. Here is some example output

```
>>> print(next_pos(1,'A'))
2
>>> print(next_pos(1,'C'))
3
>>> print(next_pos(2,'B'))
3
>>> print(next_pos(2,'C'))
2
```

Task 2 Find the Position after Many Moves (Iterative Version)

Write an ***iterative*** `final_pos_i(pos,seq)` to return the final position after a sequence of moves `seq` by the beginning position pos. The input `seq` is a string that will not contain any character other than A, B or C. Your function must use for-loops or while-loops and cannot be recursive. You can reuse next_pos as you have defined in Task 1, but you cannot reuse final_pos_r as defined in Task 3. Here are some sample output:

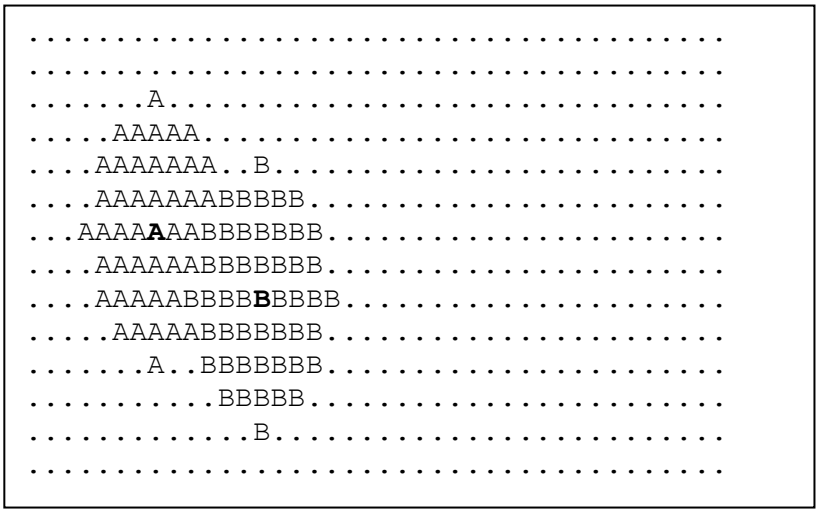
```
>>> print(final_pos_i(1,'AAA'))
2
>>> print(final_pos_i(1,'AC'))
2
>>> print(final_pos_i(2,'ABBBAAACCACACB'))
1
>>> print(final_pos_i(3,'ABBBAAACCACACB'))
2
```

Task 3 Find the Position after Many Moves (Recursive Version)

Write a ***recursive*** version `final_pos_r(pos,seq)` with the same functionality of final_pos_i in Part 1 Task 2. You cannot use any loops or list comprehension in this task. You can also reuse next_pos as you have defined in Task 1 but you cannot reuse final_pos_i as you have defined in Task 2.

Part 2 Splastoon!! (15 + 10 + 10 + 5= 40 marks)

We create a game and we let players shoot paint on a wall. Each shot will paint a circle on the wall and cover up the previous paint. The wall is represented by a 2D list with `r` row and `c` columns. Below is an example after Player A shoots the wall at Row 6 and Column 7 (row and column numbers start at 0) before Player B shoots the wall at Row 8 and Column 13. The bold characters are the centers of the shots. At this time, the radius is set to be 4 and it means any pixel of the wall with a distance less than or equal to 4 will be painted by that player. The radius of the paint can be varied from game to game.



Please note that you may want to finish reading Tasks 1 and 2 before you start because they may be solved in any order.

Task 1 Splash!

Write a function `splash(no_r,no_c,radius,actions)` to return a 2D list such that `no_r` and `no_c` are the numbers of rows and columns of the wall respectively. The input `radius` is the radius of the circle made by the paint and `actions` is a tuple that contains many triplets `(a, b, c)` such that `a` is the player character. The two numbers `b, c` are the row and column numbers of the center of the paint

The function should return a 2D list that contains like the followings. Example output here:

```
>>> pprint(splash(9,8,3,(('A',2,3),('B',5,6))))
[['.', 'A', 'A', 'A', 'A', 'A', '.', '.'],
 ['.', 'A', 'A', 'A', 'A', 'A', '.', '.'],
 ['A', 'A', 'A', 'A', 'A', 'A', 'B', '.'],
 ['.', 'A', 'A', 'A', 'B', 'B', 'B', 'B'],
 ['.', 'A', 'A', 'A', 'B', 'B', 'B', 'B'],
 ['.', '.', '.', 'B', 'B', 'B', 'B', 'B'],
 ['.', '.', '.', '.', 'B', 'B', 'B', 'B'],
 ['.', '.', '.', '.', 'B', 'B', 'B', 'B'],
 ['.', '.', '.', '.', '.', '.', 'B', '.']]

>>> tight_print(splash(16,20,4,(('A',6,7),('B',9,12),('C',10,6))))
.....
.....
.....A.....
.....AAAAA.....
....AAAAAAA.....
....AAAAAAA.B.....
...AAACAAABBBBB.....
....CCCCCBBBBBBB....
...CCCCCCCBBBBBB....
...CCCCCCCBBBBBBB...
..CCCCCCCCCBBBBB....
...CCCCCCCBBBBBB....
...CCCCCCCBBBBB.....
....CCCCC...B.....
.....C.....
.....
```

Task 2 Specific Position

Write a function `which_paint(no_r,no_c,radius,r,c,actions)` to return the final paint on the position at row r and column c on the wall. Return `'.'` if there is no paint on that position.

```
>>> actions1 = (('A',3,2),('B',5,6),('C',4,7))
>>> print(which_paint(9,8,3,3,4,actions1))
B
```

The wall of the above example should look like this, but you do not need to produce the wall.

```
>>> pprint(splash(9,8,3,actions1))
[['.', '.', 'A', '.', '.', '.', '.', '.'],
 ['A', 'A', 'A', 'A', 'A', '.', '.', 'C'],
 ['A', 'A', 'A', 'A', 'A', 'C', 'C', 'C'],
 ['A', 'A', 'A', 'A', 'B', 'C', 'C', 'C'],
 ['A', 'A', 'A', 'A', 'C', 'C', 'C', 'C'],
 ['A', 'A', 'A', 'B', 'B', 'C', 'C', 'C'],
 ['.', '.', 'A', '.', 'B', 'C', 'C', 'C'],
 ['.', '.', '.', '.', 'B', 'B', 'B', 'C'],
 ['.', '.', '.', '.', '.', '.', 'B', '.']]
```

Task 3 BIG Wall

All of your function `which_paint` () should be able to complete within one second for big numbers. (E.g. 7680 x 4320 is the resolution for a 8K display.) For example:

```
>>> print(which_paint(7680,4320,3000,1600,1700,(('A',1500,1500),('B',2000,3500))))
B
>>> print(which_paint(10000000,20000000,4000000,5000000,5000000,(('A',6000000,6000000),)))
A
```

Task 4 Multiple Choice

Write a function P2T4_`answer()` to return either 'A', 'B', etc. for the following question.

The problem of this part is closest to which of the following assignments in IT5001?

```
A. Snell's Law
B. Drawing a flower
C. nChooseK
D. Matching resistors
E. Find Treasure
F. Finding ancestor
G. Pizza
H. Battle RPG
```

Your function should simply return a character. For example, if your answer is 'A', it should be like this:

```
>>> print(P2T4_answer())
A
```

Part 3 Travel back in time to buy stocks (20 + 10 marks)

Dr. Mad makes a breakthrough in his time travel research! Although he cannot go back in time himself, he is able to send a robot to go back in time. Because of the limitation of the robot, it can only go back in a specific time and

- Buy a stock at that time
- And set when is the time to sell the stock

Dr. Mad gets the past records of a stock and he wants to maximize his profit. So, he wants to buy the stock at a low point and sell it at a high point so that he can get the maximum profit. The variation of the price of that stock is given in a list of integer in dollars. Your job is to come up with a function `max_profit(lst)` to return the maximum profit that Dr. Mad could get.

For example, if the stock prices for 16 days are:

```
[6, 11, 4, 1, 4, 9, 11, 12, 8, 5, 2, 3, 8, 13, 15, 14]
```

The best time to buy it is at the price of $1 and sell it at the price of $15 and make a profit of $14. Of cousre, the buying date of the stock must be before or the same as the day of selling

Task 1 Maximum Profit

Write a function `max_profit(lst)` to return an integer of the maximum profit from the stock list `lst`. You can assume that the input `lst` contains only positive integers with list length > 0. Here is an example:

```
>>> A1 = [6, 11, 4, 1, 4, 9, 11, 12, 8, 5, 2, 3, 8, 13, 15, 14]
>>> print(max_profit(A1))
14
>>> A2 = [6, 11, 18, 1, 4, 3, 8, 13, 14]
>>> print(max_profit(A2))
13
```

Task 2 Large List

Your function should be able to compute large list (e.g. length > 6000) within 1 second.

```
>>> A3 = [i%1007 for i in range(1,10000)]
>>> print(max_profit(A3))
1006
```

Code Template

```
#####################################################################
#Part 1

def next_pos(pos,m):
    return 0


def final_pos_i(pos,seq):
    return 0

def final_pos_r(pos,seq):
    return 0

#####################################################################
#Part 2

#you can use this for your test
def tight_print(m):
    for row in m:
        print(''.join(row))

def splash(no_r,no_c,radius,actions):
    return

def which_paint(no_r,no_c,radius,r,c,actions):
    return


def P2T4_answer()
    return 0



#####################################################################
#Part 3

#you can use this for your test
def max_profit(lst):
    return 0



# end of template
```

-- End of Paper --