

Singapore Management University  
School of Computing and Information Systems  
2024/2025 Semester 1  
IS628: Computational Thinking with Python  
Assignment 2 (due date: **3 November 2024, 11:59PM**)

**Maximum marks = 30.**

**Submission Instructions:**

You are expected to submit as following:

- (1) The skeleton code `a2.ipynb` is provided; read the comments, and fill in the missing parts;
- (2) Run the feedback section in `a2.ipynb` located after each question;
- (3) Check `q1.txt`, `q2.txt`, and `q3.txt` in the same folder.
- (4) Zip your finalized `a2.ipynb`, `q1.txt`, `q2.txt`, and `q3.txt` then upload the zip file to the eLearn Assignment folder **Assignment 2**.

*\* Submissions in wrong format may not be run through auto-grader.*

Multiple submissions are permitted up to the due time, but only the last submission will be saved and graded. Please note that the Originality Check has been enabled.

**[Question 1] (5 marks)**

You are approached by world-renowned treasure hunter Outiana Jones to assist in his latest treasure hunting adventure. He has received a series of messages which encodes the coordinates to the next location of each treasure, and he seeks your assistance in developing a Python code to quickly extract the final location of the treasure.

The **code must be written using regular expression**, as doing otherwise could trigger a deadly trap that would endanger his life (and your grades :D).

**Objective:**

You are given a series of messages in the form of a dictionary, *d*. This dictionary contains as keys the coordinate in XX.XX+YY.YY format, with the message to decrypt as the value. Your goal is to design **ONE regular expression pattern** that extracts the next coordinate from the message to find the next coordinate. Only when that pattern does not return any matches will we stop and conclude that that is the final location of the treasure.

You **must** store this regular expression pattern within *self.pattern*.

**Assumptions:**

- The coordinate format is **strictly** in XX.XX and YY.YY format
  - o Coordinates should only contain numerical digits in X and Y
  - o Valid examples are 12.34, 04.40, 11.43
  - o Invalid examples are 1.34, 11.3, 54.544, 233.555
  - o You will ALWAYS start with 00.00 and 00.00, and thus, 00.00 is also an invalid coordinate you should match (since you will go into an infinite loop :D)
  - o Furthermore, since we start at 00.00, it is meaningless to search for the treasure near to the starting point, thus ALL coordinates which contain 00.XX or 00.YY are invalid coordinates too! (E.g., 00.45, 00.11 are invalid coordinates)
- The first coordinate found will always be the x-coordinate, and the second coordinate found will always be the y-coordinate
- Coordinates need not be separated from other text by a space, e.g., “hello11.23!” hides a valid coordinate 11.23
- All inputs used in open and hidden tests are valid inputs

**Example:**

Refer to the below for a sample dictionary, and the expected result from the regular expression.

Key	Value	Next Coordinate
00.00+00.00	“The next treasure is at 12.34 and 23.45”	12.34, 23.45

12.34+23.45	"The fake treasure is in 222.11 and 11.11 and 14.56"	11.11, 14.56
11.11+14.56	"Congrats!"	Nothing is matched

*Hint: Notice the fake coordinates hidden in the text :D.*

**Grading Criteria:**

- 1 mark will be given for passing each of the two open test cases (Total 2 marks)
- 1 mark will be given for passing each of the two hidden test cases (Total 2 marks)
- 1 mark will be given for providing a solution not exceeding 50 characters. (Refer to the solution template for how this is calculated)

## [Question 2] (15 marks)

After heading to the treasure location following the coordinates you have found for him, you and Outiana Jones come across a gigantic puzzle hindering your progress towards the secret treasure.

“This... this is a crossword puzzle!” Outiana Jones exclaimed. “Hurry, help me to solve this!”

### Objective:

You are given a multi-line string, *s*, forming a 2-dimensional crossword puzzle, and a list, *l*, consisting of multiple strings. Each string is either: (a) A single word that you should attempt to search for in the crossword puzzle, OR (b) A line of instructions for you to find one such word in the crossword puzzle.

There are two parts to this problem:

**Part 1:** Given the list of strings, write a **for loop** that processes each string in *l* to come up with **ONE regular expression pattern** based on the two cases above, while catering to the possible arrangements in the crossword puzzle (listed in Part 2) for each of the strings (*i.e.*,  $len(self.patterns) == len(l)$ ). Refer to the assumptions, examples, and grading criteria below for more information.

Append each of these patterns into the variable *patterns* to be used in Part 2.

To achieve the full marks for this part, you also need to:

1. Ensure that  $len(self.patterns) == len(l)$ , that is, you write ONE regular expression for each line in *l* such that that one expression checks all possible matches for that line in *s*.
2. Use a regular expression (stored in *self.line\_pattern*) to process the line of instructions (Case (b)).

*Hint: Here, a regular expression pattern is basically a string, so you can use what you have learnt to design one pattern based on the given information!*

**Part 2:** Given *patterns* defined in Part 1, write another **for loop** to process each pattern and search for the word in the crossword puzzle. Here, the word can appear in the following form:

- A match from left to right
- A match diagonally downwards from either left to right or right to left
- A match from top to bottom

You should **return** the word which was matched in the crossword puzzle.

**Assumptions:**

- The crossword puzzle contains ONLY lower-case letters from a – z.
- Each string in the list will be in either of the below format:
  - A single word, e.g., snake, rope
  - A sentence of the following form:
    - “A length Y word with  $X_1$  in the  $Y_1$  position,  $X_2$  in the  $Y_2$  position, ...,  $X_n$  in the  $Y_n$  position”
    - Y represents the length of the word,  $(X_i, Y_i)$  represents the character and the index position (i.e., starting from 0) of that character
    - There can be a variable number of such word and positions
    - E.g., A length 5 word with a in the 1 position, b in the 3 position
    - E.g., A length 3 word with x in the 2 position
- Only a single match from the list can be found in the crossword puzzle
  - A match can be found in the following directions:
    - Left to right
    - Top to bottom
    - Diagonally downwards from left to right or right to left
- All inputs used in open and hidden tests are valid inputs

**Examples:**

Refer to the below crossword puzzle for string  $s$  -

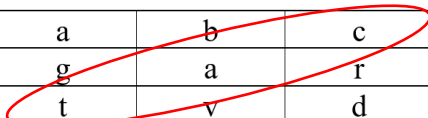
a	b	c
g	a	r
t	v	d

And the below list of strings for  $l$  –

- bat
- a length 3 word with g in the 0 position, a in the 1 position, t in the 2 position
- a length 3 word with a in the 1 position, t in the 2 position

The only match here is “cat”, matching the third string “a length 3 word with a in the 1 position, t in the 2 position”.

a	b	c
g	a	r
t	v	d



**Grading Criteria:**

For part 1 (6 marks),

- (a) 2 marks will be given for a solution that creates a list labelled *pattern* containing regular expression patterns that will be used in part 2 to give **EXACTLY** one match with the provided part 2 solution (2 marks total)
- (b) 2 marks will be given for a solution that processes each line in *l* using a regular expression to find the values satisfying the sentence format (2 marks total)
- (c) 2 marks will be given for a solution that creates **ONE** regular expression for each string in *l* that is used in part 2, i.e., `len(self.patterns) == len(l)` (2 marks total)

For part 2 (5 marks),

- (a) 2 marks will be given for a solution that makes use of the results in *pattern* to search for the word in *s* (2 marks total)
- (b) 2 marks will be given for using the below solution for part 2 (2 marks total)

```
for pattern in self.pattern:
    p = re.compile(pattern)
    m = p.findall(s)
    if len(m) == 1:
        return XXXX # replace this with what you want to return
```

*Hint: Yes, I just gave you a possible solution for part 2 :D. If you did part 1 correctly, this would be all you need for part 2!*

- (c) 1 mark will be given for returning the correctly matched word in your solution (1 mark total)

For the remainder of the marks (4 marks),

- 1 mark will be given each for any solution given which passes each of the 2 open test cases (2 marks total)
- 1 mark will be given each for any solution given which passes each of the 2 hidden test cases (2 marks total)

### [Question 3] (10 marks)

After solving the crossword puzzle, you and Outiana Jones enter an intricate tunnel system. In front of you is an array of equipment for you to pick up and use. Beside each equipment displays a description of the weapon, the amount of damage it deals, and what it is effective against.

Suddenly, a loud “Stage 1” is heard, and a trapdoor in the far end opens, where monsters begin to rush towards both of you. Above the monster displays its type and health. You immediately connect the dots on what you need to do in this stage...

#### Objective:

In this problem, your objective is to use Classes to design the above scenario. An initial code has already been set-up for you. You would need to fill in the blanks to ensure that all the classes are working following their descriptions. In this scenario, we have the following Classes:

1. The Weapon class represents the weapons available for the player to use. This class requires the following attributes and methods:

- a. Attributes:

self.damage	An <b>int</b> variable that indicates the damage this weapon deals.
self.range	An <b>int</b> variable that indicates the range of this weapon.
self.durability	An <b>int</b> variable with a default value, 10, that indicates the remaining durability of this weapon.
self.is_broken	A <b>Boolean</b> variable that indicates whether the weapon is broken.

- b. Methods:

attack(self, monster)	<p>Attacks a Monster instance.</p> <ol style="list-style-type: none"> <li>1. This first reduces the monster’s health by self.damage.</li> <li>2. Then, reduce the weapon’s durability by 1.</li> <li>3. Then, call the Monster class’ attack method to reduce the durability of this weapon.</li> <li>3. Finally, call check_broken to check if the weapon is broken or not.</li> </ol> <p><i>(This method should return None)</i></p>
check_broken(self)	<p>Checks if the weapon is broken (self.durability &lt;= 0), and set is_broken to the necessary Boolean.</p> <p><i>(This method should return None)</i></p>

2. The Monster class consists of a base class, the Monster class, which is already designed for you. This class has the following:

Attributes:

self.damage	An <b>int</b> variable that indicates the damage this monster deals.
self.health	An <b>int</b> variable with a default value, 10, that indicates the remaining health of this monster.
self.is_alive	A <b>Boolean</b> variable that indicates whether the monster is alive or not.

Methods:

check_alive(self)	Checks if the monster is dead ( $\text{self.health} \leq 0$ ), and set <code>is_alive</code> to the necessary Boolean.  <i>(This method returns <code>self.is_alive</code>)</i>
-------------------	---

Your goal is to implement the two possible types of Monsters using this base class, represented by RangeMonster and MeleeMonster. You **must** use `super()` to initialize the two classes.

- a. A RangeMonster additionally has the following:

Attributes:

self.range	An <b>int</b> variable that shows the range of this monster.
------------	--

Methods:

attack(self, weapon)	Attempts to counterattack a Weapon instance whenever the monster is attacked. 1. Check if the monster is still alive, by calling <code>check_alive</code> . 2. A monster can only counterattack if it is still alive, and the monster's range is greater than or equal to the Weapon's range. 3. If it is able to counterattack, the Weapon's durability is reduced by the Monster's attack.  <i>(This method should return None)</i>
----------------------	--

- b. A MeleeMonster additionally has the following:

Attributes:

self.range	An <b>int</b> variable that shows the range of this monster. <i>Melee monsters always have a range of 1.</i>
------------	---



**Methods:**

attack(self, weapon)	<p>Attempts to counterattack a Weapon instance whenever the monster is attacked.</p> <ol style="list-style-type: none"> <li>1. Check if the monster is still alive, by calling <code>check_alive</code>.</li> <li>2. A monster can only counterattack if it is still alive, and the monster's range is greater than or equal to the Weapon's range.</li> <li>3. If it is able to counterattack, the Weapon's durability is reduced by 2 x of the Monster's attack. (Melee monsters deal twice the damage in this attack!)</li> </ol> <p><i>(This method should return None)</i></p>
----------------------	---

**Assumptions:**

- All self.damage, self.range, self.health, self.durability have starting values more than 0.
- The types of each attributes are given in the objective (You may assume that all test cases contains valid inputs).
- You are required to use `check_alive` and `check_broken` when indicated in the attack methods.
- You are required to use `super()` to call the Monster class initializer to initialize the RangeMonster and MeleeMonster class.

**Examples:**

- Refer to the feedback cells within the ipynb file to see example sequences of execution of the code.

**Grading Criteria:**

- 1 mark will be given for passing each of the three open test cases (Total 3 marks)
- 1 mark will be given for passing each of the three hidden test cases (Total 3 marks)
- 2 marks will be given for providing a solution that uses `super()` to initialize the RangeMonster and MeleeMonster class using the Monster class initializer. (Total 2 marks)
- 2 marks will be given for using the `check_alive` and `check_broken` methods when indicated in the Objective. (Total 2 marks)