As with the other project, things to look out for in solving the questions are:

- Make sure to name functions and arguments as stipulated in the question, but never be afraid to create extra functions of your own, e.g. to break up the code into conceptual sub-parts, or avoid redundancy in your code
- Commenting of code is one thing that you will be marked on; get some practice writing comments in your code, focusing on:
  - Describing key variables when they are first defined (but not things like index variables in `for` loops)
  - Describing what "chunks" of code do (i.e. not every line, but chunks of code that perform a particular operation, such as `# find the maximum value in the list` or `# count the number of vowels`
  - Describing what every function does, including what its arguments are, and what it returns.

Overview

In this project, we build upon the elevations, paths, and ponies introduced in Project 1.

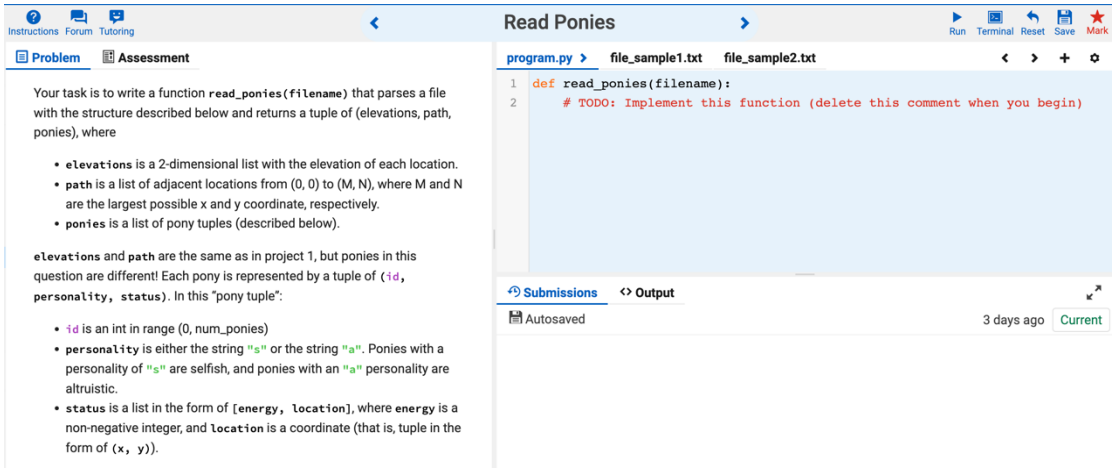
In question 1, you will interact with text files to read information about elevations, paths, and ponies. In question 2 and question 3, you will determine how ponies with different personalities climb along given paths.

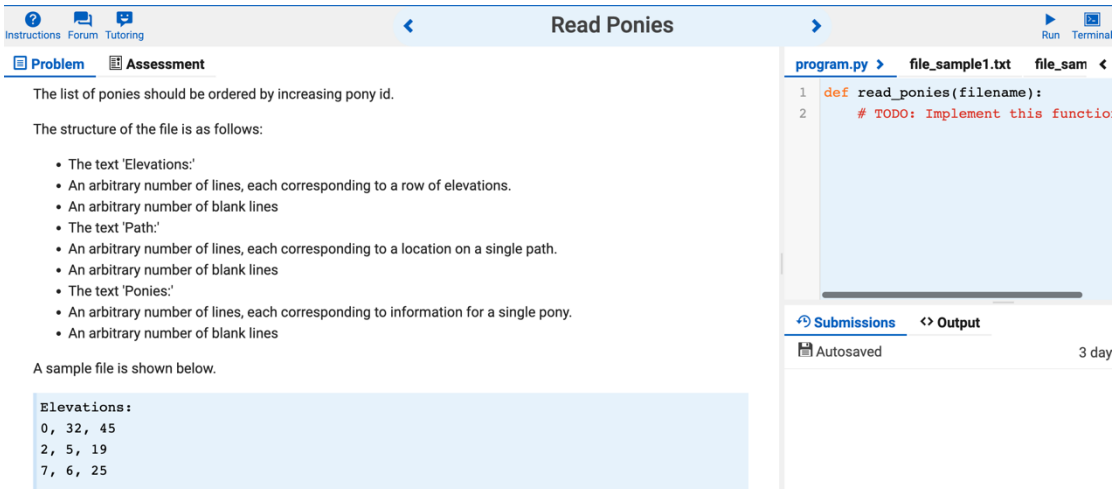
Remember that this is an assessment task, and the work must be your own. It is illegal for others to do this work on your behalf.

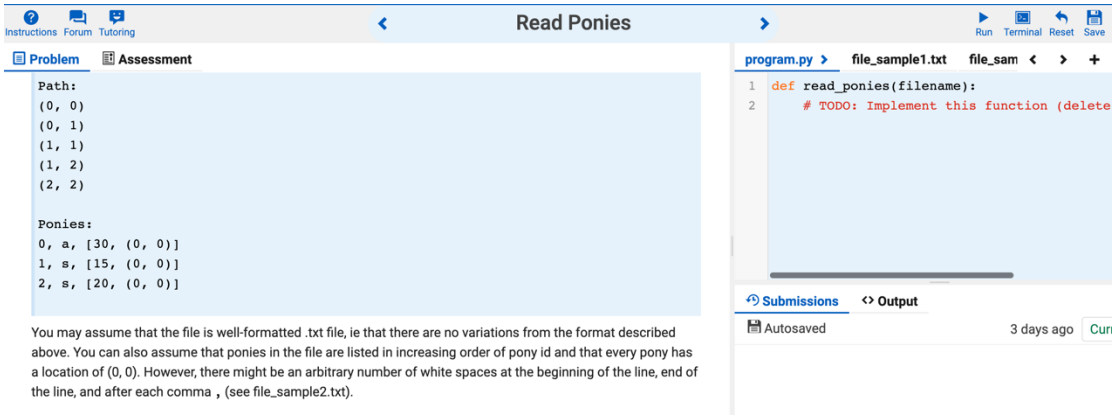
Updates

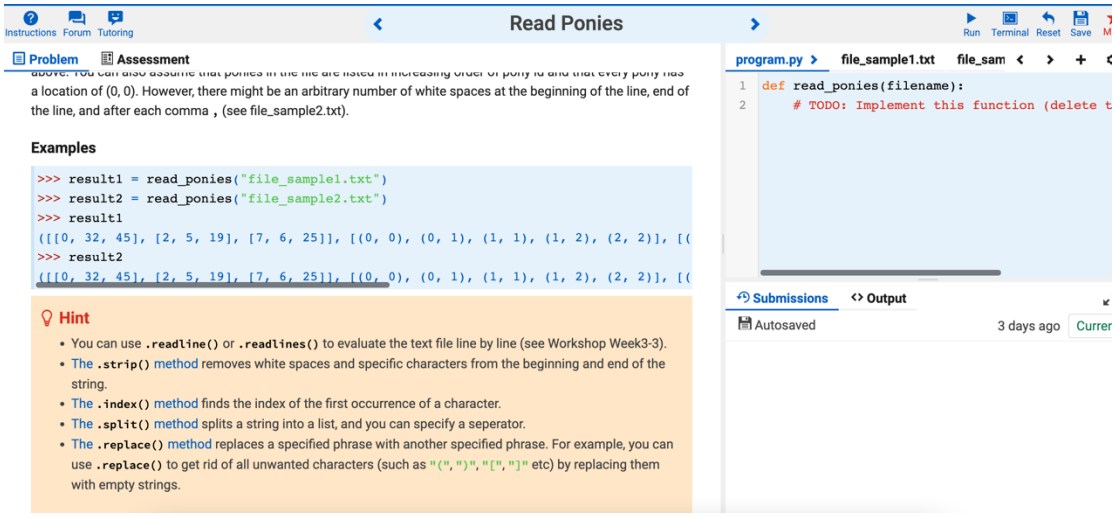
- 8 Feb on Q2: The output of the first example is updated.
- 10 Feb on Q1: You can assume the files to read are all .txt files.
- 12 Feb on Q1: Added some hints.

1

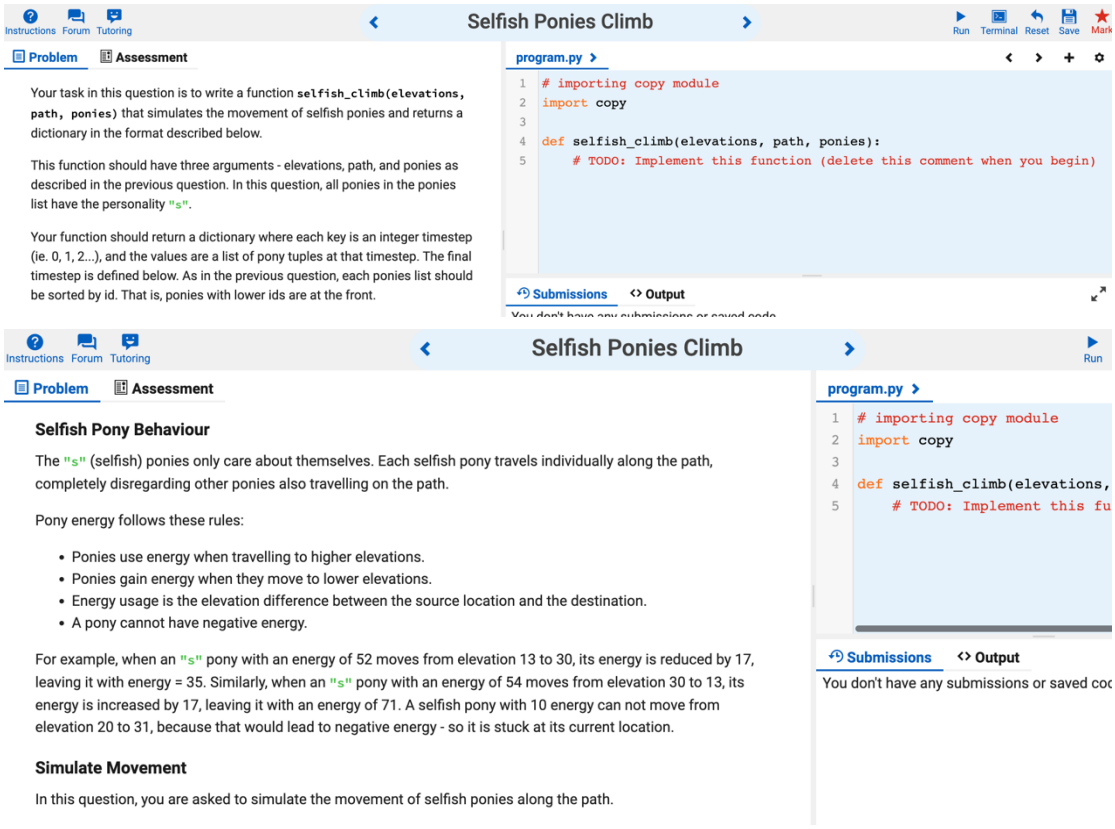

Instructions
Forum
Tutoring
Read Ponies
Run
Terminal
Reset
Save
Mark
Problem
Assessment
Your task is to write a function read_ponies(filename) that parses a file with the structure described below and returns a tuple of (elevations, path, ponies), where
elevations is a 2-dimensional list with the elevation of each location.
path is a list of adjacent locations from (0, 0) to (M, N), where M and N are the largest possible x and y coordinate, respectively.
ponies is a list of pony tuples (described below).
elevations and path are the same as in project 1, but ponies in this question are different! Each pony is represented by a tuple of (id, personality, status). In this "pony tuple":
id is an int in range (0, num_ponies)
personality is either the string "s" or the string "a". Ponies with a personality of "s" are selfish, and ponies with an "a" personality are altruistic.
status is a list in the form of [energy, location], where energy is a non-negative integer, and location is a coordinate (that is, tuple in the form of (x, y)).
program.py
file_sample1.txt
file_sample2.txt
1 def read_ponies(filename):
2 # TODO: Implement this function (delete this comment when you begin)
Submissions
Output
Autosaved
3 days ago
Current


Instructions
Forum
Tutoring
Read Ponies
Run
Terminal
Problem
Assessment
The list of ponies should be ordered by increasing pony id.
The structure of the file is as follows:
The text 'Elevations:'
An arbitrary number of lines, each corresponding to a row of elevations.
An arbitrary number of blank lines
The text 'Path:'
An arbitrary number of lines, each corresponding to a location on a single path.
An arbitrary number of blank lines
The text 'Ponies:'
An arbitrary number of lines, each corresponding to information for a single pony.
An arbitrary number of blank lines
A sample file is shown below.
Elevations:
0, 32, 45
2, 5, 19
7, 6, 25
program.py
file_sample1.txt
file_sam
1 def read_ponies(filename):
2 # TODO: Implement this functio
Submissions
Output
Autosaved
3 day


Instructions
Forum
Tutoring
Read Ponies
Run
Terminal
Reset
Save
Problem
Assessment
Path:
(0, 0)
(0, 1)
(1, 1)
(1, 2)
(2, 2)
Ponies:
0, a, [30, (0, 0)]
1, s, [15, (0, 0)]
2, s, [20, (0, 0)]
You may assume that the file is well-formatted .txt file, ie that there are no variations from the format described above. You can also assume that ponies in the file are listed in increasing order of pony id and that every pony has a location of (0, 0). However, there might be an arbitrary number of white spaces at the beginning of the line, end of the line, and after each comma , (see file_sample2.txt).
program.py
file_sample1.txt
file_sam
1 def read_ponies(filename):
2 # TODO: Implement this function (delete
Submissions
Output
Autosaved
3 days ago


Instructions
Forum
Tutoring
Read Ponies
Run
Terminal
Reset
Save
Problem
Assessment
a location of (0, 0). However, there might be an arbitrary number of white spaces at the beginning of the line, end of the line, and after each comma , (see file_sample2.txt).
Examples
>>> result1 = read_ponies("file_sample1.txt")
>>> result2 = read_ponies("file_sample2.txt")
>>> result1
([[0, 32, 45], [2, 5, 19], [7, 6, 25]], [(0, 0), (0, 1), (1, 1), (1, 2), (2, 2)], [(
>>> result2
([[0, 32, 45], [2, 5, 19], [7, 6, 25]], [(0, 0), (0, 1), (1, 1), (1, 2), (2, 2)], [(
Hint
You can use .readline() or .readlines() to evaluate the text file line by line (see Workshop Week3-3).
The .strip() method removes white spaces and specific characters from the beginning and end of the string.
The .index() method finds the index of the first occurrence of a character.
The .split() method splits a string into a list, and you can specify a seperator.
The .replace() method replaces a specified phrase with another specified phrase. For example, you can use .replace() to get rid of all unwanted characters (such as "(",")","[","]" etc) by replacing them with empty strings.
program.py
file_sample1.txt
file_sam
def read_ponies(filename):
# TODO: Implement this function (delete t
Submissions
Output
Autosaved
3 days ago
Currer

2


Instructions
Forum
Tutoring
Selfish Ponies Climb
Run
Terminal
Reset
Save
Problem
Assessment
Your task in this question is to write a function selfish_climb(elevations, path, ponies) that simulates the movement of selfish ponies and returns a dictionary in the format described below.
This function should have three arguments - elevations, path, and ponies as described in the previous question. In this question, all ponies in the ponies list have the personality "s".
Your function should return a dictionary where each key is an integer timestep (ie. 0, 1, 2...), and the values are a list of pony tuples at that timestep. The final timestep is defined below. As in the previous question, each ponies list should be sorted by id. That is, ponies with lower ids are at the front.
program.py
# importing copy module
import copy
def selfish_climb(elevations, path, ponies):
# TODO: Implement this function (delete this comment when you begin)
Submissions
Output

Instructions
Forum
Tutoring
Selfish Ponies Climb
Run
Problem
Assessment
Selfish Pony Behaviour
The "s" (selfish) ponies only care about themselves. Each selfish pony travels individually along the path, completely disregarding other ponies also travelling on the path.
Pony energy follows these rules:
Ponies use energy when travelling to higher elevations.
Ponies gain energy when they move to lower elevations.
Energy usage is the elevation difference between the source location and the destination.
A pony cannot have negative energy.
For example, when an "s" pony with an energy of 52 moves from elevation 13 to 30, its energy is reduced by 17, leaving it with energy = 35. Similarly, when an "s" pony with an energy of 54 moves from elevation 30 to 13, its energy is increased by 17, leaving it with an energy of 71. A selfish pony with 10 energy can not move from elevation 20 to 31, because that would lead to negative energy - so it is stuck at its current location.
Simulate Movement
In this question, you are asked to simulate the movement of selfish ponies along the path.
program.py
# importing copy module
import copy
def selfish_climb(elevations,
TODO: Implement this fu
Submissions
Output
You don't have any submissions or saved coc

Instructions Forum Tutoring

Selfish Ponies Climb

Problem Assessment

Simulate Movement

In this question, you are asked to simulate the movement of selfish ponies along the path.

Movement follows these rules:

- At a given time `t`, the pony with `id < t` can move, where t is a valid integer less than or equal to the value of the final timestep. For example, at `t=0`, all ponies with `id < 0` consider a move, and at `t=1`, all ponies with `id < 1` consider a move.
- A pony may or may not move after a move is considered, depending on (1) the energy difference and (2) if the pony has reached the destination.
- At each timestep, ponies move in order of ascending id, ie pony 0 would move before pony 1.

Note that you can assume all ponies have unique ids, and the ids of ponies in the **`ponies`** list are continuous integers in ascending order. The smallest id is 0, and the largest id is **`len(ponies) - 1`**.

Final Timestep

There must be enough timesteps so that every pony considers at least one move. The final timestep is the first timestep when both of the following conditions are met:

- each pony has considered at least one move
- each pony either has insufficient energy to move or has arrived at the final destination.

program.py

```
# importing copy mo
import copy

def selfish_climb(e
    # TODO: Impleme
```

Submissions Output

You don't have any submissions

Instructions Forum Tutoring

Selfish Ponies Climb

Problem Assessment

Run Terminal Rese

- each pony has considered at least one move
- each pony either has insufficient energy to move or has arrived at the final destination.

Hints

- Say the final timestep is **`t_final`**, then if your program continues to run after **`t = t_final`**, no pony will change their **`status`**.

Examples

```
>>> result1 = selfish_climb([[0, 32, 45], [2, 5, 19], [7, 6, 25]], [(0, 0), (1, 0),
>>> result1
{0: [(0, 's', [7, (0, 0)]), (1, 's', [2, (0, 0)]), (2, 's', [25, (0, 0)])], 1: [(0,
>>> result2 = selfish_climb([[0, 10, 20], [0, 0, 20], [0, 0, 0]], [(0, 0), (0, 1), (
>>> result2
{0: [(0, 's', [0, (0, 0)]), (1, 's', [0, (0, 0)])], 1: [(0, 's', [0, (0, 0)]), (1, '
>>> result3 = selfish_climb([[0, 0], [0, 0]], [(0, 0), (1, 0), (1, 1)], [(0, 's', [0
>>> result3
{0: [(0, 's', [0, (0, 0)])], 1: [(0, 's', [0, (1, 0)])], 2: [(0, 's', [0, (1, 1)])]}
>>> result4 = selfish_climb([[100, 90, 80, 0], [30, 20, 40, 100], [31, 45, 0, 400]],
>>> result4
{0: [(0, 's', [0, (0, 0)]), (1, 's', [10, (0, 0)])], 1: [(0, 's', [10, (0, 1)]), (1,
```

program.py

```
# importing copy module
import copy

def selfish_climb(elevations, path, po
    # TODO: Implement this function (c
```

Submissions Output

You don't have any submissions or saved code.

Problem Assessment

{0: [(0, 's', [7, (0, 0)]), (1, 's', [2, (0, 0)]), (2, 's', [25, (0, 0)])], 1: [(0,
>>> result2 = selfish_climb([[0, 10, 20], [0, 0, 20], [0, 0, 0]], [(0, 0), (0, 1), (
>>> result2
{0: [(0, 's', [0, (0, 0)]), (1, 's', [0, (0, 0)])], 1: [(0, 's', [0, (0, 0)]), (1, '
>>> result3 = selfish_climb([[0, 0], [0, 0]], [(0, 0), (1, 0), (1, 1)], [(0, 's', [0
>>> result3
{0: [(0, 's', [0, (0, 0)])], 1: [(0, 's', [0, (1, 0)])], 2: [(0, 's', [0, (1, 1)])]}
>>> result4 = selfish_climb([[100, 90, 80, 0], [30, 20, 40, 100], [31, 45, 0, 400]],
>>> result4
{0: [(0, 's', [0, (0, 0)]), (1, 's', [10, (0, 0)])], 1: [(0, 's', [10, (0, 1)]), (1,

Hints

- Lists are mutable. Make sure you understand what happens in the following example:

```
lst = [1, 2]
container = (1, lst)
lst_ptr = container[1]
lst_ptr[0] = 10
print(container)
```

- To create a copy of a list, you can use **`.copy()`** or **`.deepcopy()`**. You can check them out here (we helped you import the copy module already, but you are not required to use it).

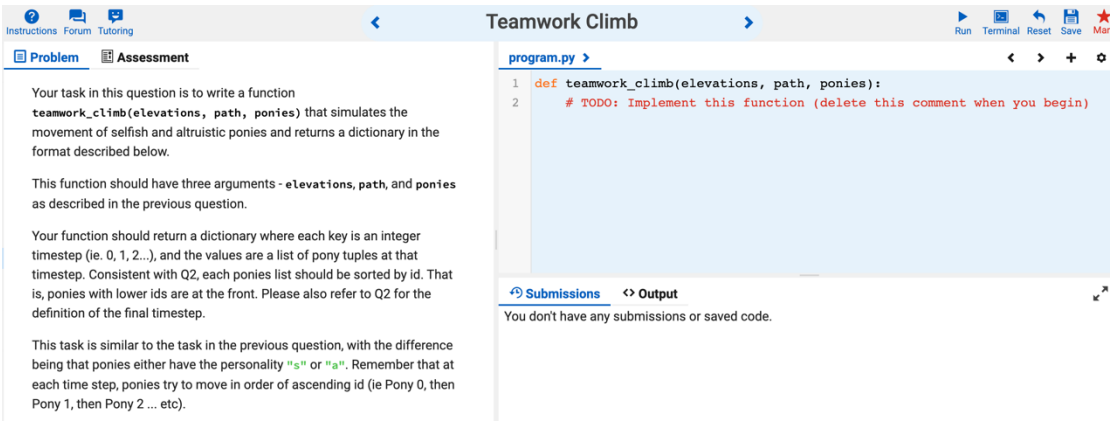
program.py

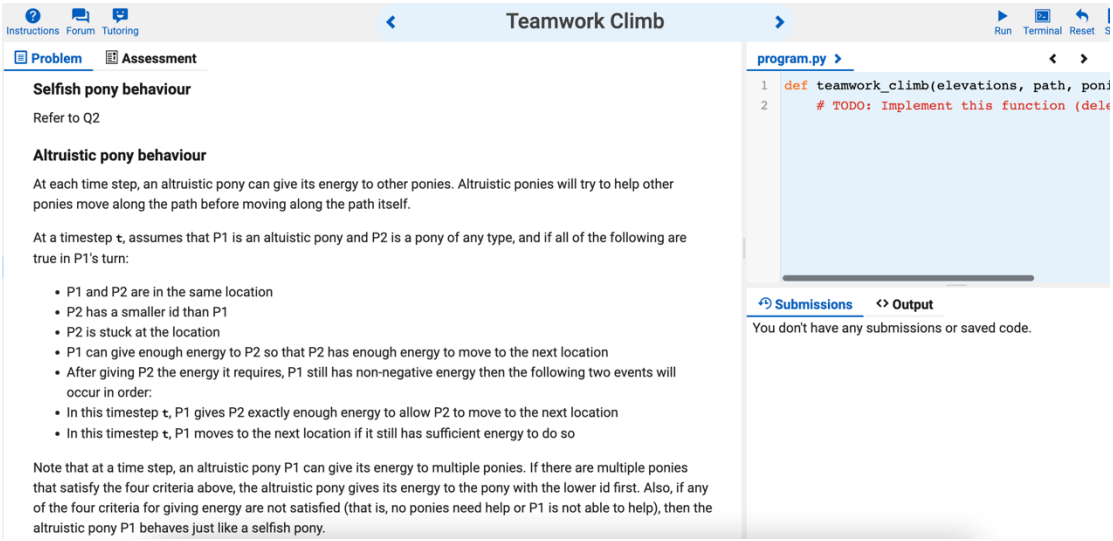
```
# importing copy moc
import copy

def selfish_climb(el
    # TODO: Implemen
```

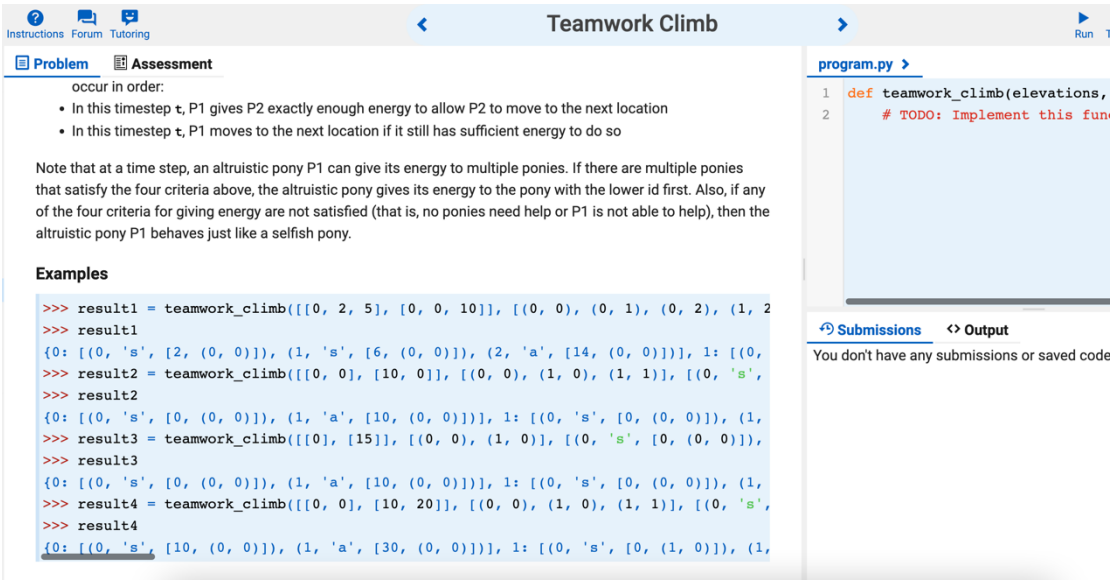
Submissions Output

You don't have any submissions

3


Instructions
Forum
Tutoring
Teamwork Climb
Run
Terminal
Reset
Save
Problem
Assessment
Your task in this question is to write a function teamwork_climb(elevations, path, ponies) that simulates the movement of selfish and altruistic ponies and returns a dictionary in the format described below.
This function should have three arguments - elevations, path, and ponies as described in the previous question.
Your function should return a dictionary where each key is an integer timestep (ie. 0, 1, 2...), and the values are a list of pony tuples at that timestep. Consistent with Q2, each ponies list should be sorted by id. That is, ponies with lower ids are at the front. Please also refer to Q2 for the definition of the final timestep.
This task is similar to the task in the previous question, with the difference being that ponies either have the personality "s" or "a". Remember that at each time step, ponies try to move in order of ascending id (ie Pony 0, then Pony 1, then Pony 2 ... etc).
program.py
1 def teamwork_climb(elevations, path, ponies):
2 # TODO: Implement this function (delete this comment when you begin)
Submissions
Output
You don't have any submissions or saved code.


Instructions
Forum
Tutoring
Teamwork Climb
Run
Terminal
Reset
Problem
Assessment
Selfish pony behaviour
Refer to Q2
Altruistic pony behaviour
At each time step, an altruistic pony can give its energy to other ponies. Altruistic ponies will try to help other ponies move along the path before moving along the path itself.
At a timestep t, assumes that P1 is an altuistic pony and P2 is a pony of any type, and if all of the following are true in P1's turn:
P1 and P2 are in the same location
P2 has a smaller id than P1
P2 is stuck at the location
P1 can give enough energy to P2 so that P2 has enough energy to move to the next location
After giving P2 the energy it requires, P1 still has non-negative energy then the following two events will occur in order:
In this timestep t, P1 gives P2 exactly enough energy to allow P2 to move to the next location
In this timestep t, P1 moves to the next location if it still has sufficient energy to do so
Note that at a time step, an altruistic pony P1 can give its energy to multiple ponies. If there are multiple ponies that satisfy the four criteria above, the altruistic pony gives its energy to the pony with the lower id first. Also, if any of the four criteria for giving energy are not satisfied (that is, no ponies need help or P1 is not able to help), then the altruistic pony P1 behaves just like a selfish pony.
program.py
1 def teamwork_climb(elevations, path, poni
2 # TODO: Implement this function (dele
Submissions
Output
You don't have any submissions or saved code.


Instructions
Forum
Tutoring
Teamwork Climb
Run
Problem
Assessment
occur in order:
In this timestep t, P1 gives P2 exactly enough energy to allow P2 to move to the next location
In this timestep t, P1 moves to the next location if it still has sufficient energy to do so
Note that at a time step, an altruistic pony P1 can give its energy to multiple ponies. If there are multiple ponies that satisfy the four criteria above, the altruistic pony gives its energy to the pony with the lower id first. Also, if any of the four criteria for giving energy are not satisfied (that is, no ponies need help or P1 is not able to help), then the altruistic pony P1 behaves just like a selfish pony.
Examples
>>> result1 = teamwork_climb([[0, 2, 5], [0, 0, 10]], [(0, 0), (0, 1), (0, 2), (1, 2
>>> result1
{0: [(0, 's', [2, (0, 0)]), (1, 's', [6, (0, 0)]), (2, 'a', [14, (0, 0)])], 1: [(0,
>>> result2 = teamwork_climb([[0, 0], [10, 0]], [(0, 0), (1, 0), (1, 1)], [(0, 's',
>>> result2
{0: [(0, 's', [0, (0, 0)]), (1, 'a', [10, (0, 0)])], 1: [(0, 's', [0, (0, 0)]), (1,
>>> result3 = teamwork_climb([[0], [15]], [(0, 0), (1, 0)], [(0, 's', [0, (0, 0)]),
>>> result3
{0: [(0, 's', [0, (0, 0)]), (1, 'a', [10, (0, 0)])], 1: [(0, 's', [0, (0, 0)]), (1,
>>> result4 = teamwork_climb([[0, 0], [10, 20]], [(0, 0), (1, 0), (1, 1)], [(0, 's',
>>> result4
{0: [(0, 's', [10, (0, 0)]), (1, 'a', [30, (0, 0)])], 1: [(0, 's', [0, (1, 0)]), (1,
program.py
1 def teamwork_climb(elevations,
2 # TODO: Implement this fun
Submissions
Output
You don't have any submissions or saved code