Ranked Election

Your task is to analyze a ranked election. Here's how the election works: each ballot consists of an ordered list of candidates based on a voter's preference. The ballots are processed in rounds. During each round, the first choice on each ballot is tallied up. If a candidate has the most votes AND surpasses a certain percentage of the votes, they win the election. If no candidate reaches the necessary percentage, then the candidate with the fewest votes is eliminated from all ballots. Once again the first choice on each ballot is tallied (remember, this is AFTER removing the last place finisher of the previous round). This process continues until a candidate reaches the necessary percentage. Some extreme cases to think about:

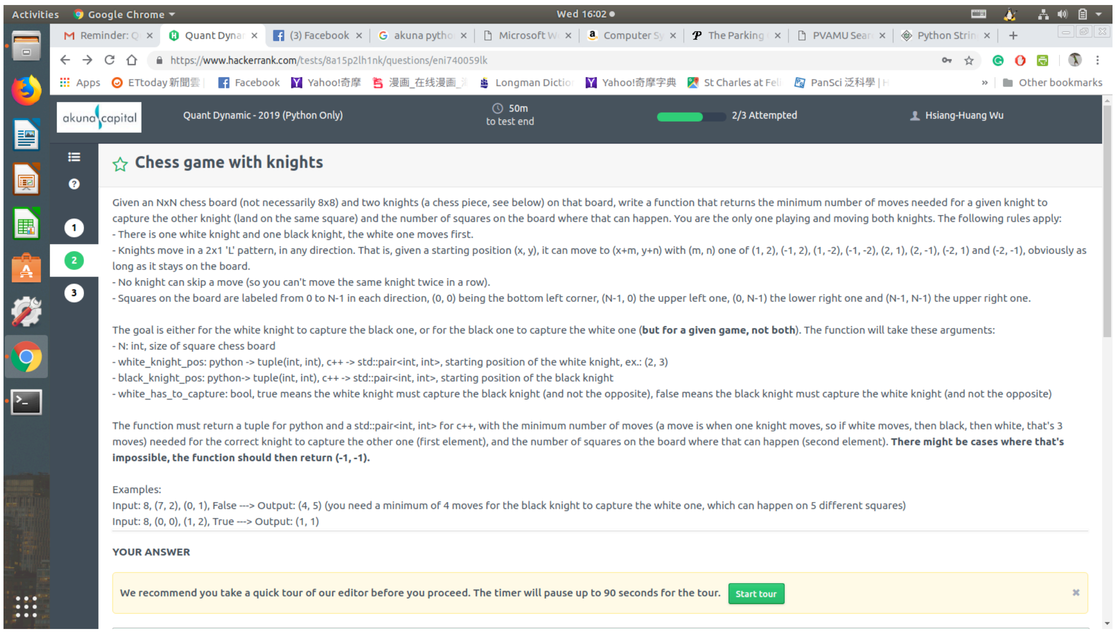
- If multiple candidates receive the same number of votes AND reach the necessary percentage in the SAME ROUND, the election is thrown out and nobody wins.
- If multiple candidates tie for last place, they are ALL removed from the ballots for the next round. This means you can be left with no candidates, in which case, the election is thrown out and nobody wins.
- Voters are not required to order all candidates on their ballot (i.e. even if there are 10 candidates, a ballot can consist of 3 ordered candidates)

To analyze the election, you must determine which candidate will win for a certain percentage given all the ballots. The expected output is a list of percentage intervals and the winning candidate for each interval. We are only concerned with whole number percentage points (i.e. 10.5% - 20% is not a valid interval).

Input Format:
num_candidates -> int
num_ballots -> int
ballots -> 2D int array with num_ballots rows and num_candidates columns *
* Candidates are labeled from 0 to num_candidates - 1. If a voter orders fewer than num_candidates, the blanks will be filled with -1.

Output Format:
2D int array of the form [[pct1, pct2, candidate], [pct3, pct4, candidate]...] *
* The intervals are inclusive on both sides. You must cover the entire percentage range from 0 to 100. If an interval exists where no candidate can win, put -1 in the candidate slot. You should minimize the number of intervals where possible.

Example:
num_candidates = 5
num_ballots = 5
ballots = [[1, 4, 3, -1, -1], [2, 1, -1, -1, -1], [1, 3, 2, 0, -1], [3, 2, 4, 0, 1], [2, 3, 4, 0, -1]]
Expected output: [[0, 40, -1], [41, 100, 2]]
For any whole number percent below 41, no candidate can win. For any whole number percent above 40, only candidate 2 can win.



Chess game with knights

Given an NxN chess board (not necessarily 8x8) and two knights (a chess piece, see below) on that board, write a function that returns the minimum number of moves needed for a given knight to capture the other knight (land on the same square) and the number of squares on the board where that can happen. You are the only one playing and moving both knights. The following rules apply:
- There is one white knight and one black knight, the white one moves first.
- Knights move in a 2x1 'L' pattern, in any direction. That is, given a starting position (x, y), it can move to (x+m, y+n) with (m, n) one of (1, 2), (-1, 2), (1, -2), (-1, -2), (2, 1), (2, -1), (-2, 1) and (-2, -1), obviously as long as it stays on the board.
- No knight can skip a move (so you can't move the same knight twice in a row).
- Squares on the board are labeled from 0 to N-1 in each direction, (0, 0) being the bottom left corner, (N-1, 0) the upper left one, (0, N-1) the lower right one and (N-1, N-1) the upper right one.

The goal is either for the white knight to capture the black one, or for the black one to capture the white one (**but for a given game, not both**). The function will take these arguments:
- N: int, size of square chess board
- white_knight_pos: python -> tuple(int, int), c++ -> std::pair<int, int>, starting position of the white knight, ex.: (2, 3)
- black_knight_pos: python-> tuple(int, int), c++ -> std::pair<int, int>, starting position of the black knight
- white_has_to_capture: bool, true means the white knight must capture the black knight (and not the opposite), false means the black knight must capture the white knight (and not the opposite)

The function must return a tuple for python and a std::pair<int, int> for c++, with the minimum number of moves (a move is when one knight moves, so if white moves, then black, then white, that's 3 moves) needed for the correct knight to capture the other one (first element), and the number of squares on the board where that can happen (second element). **There might be cases where that's impossible, the function should then return (-1, -1).**

Examples:
Input: 8, (7, 2), (0, 1), False ---> Output: (4, 5) (you need a minimum of 4 moves for the black knight to capture the white one, which can happen on 5 different squares)
Input: 8, (0, 0), (1, 2), True ---> Output: (1, 1)

YOUR ANSWER

We recommend you take a quick tour of our editor before you proceed. The timer will pause up to 90 seconds for the tour. Start tour

Word Graph

There are two inputs to this question:

1) A list of words
2) A list of tuples that represent directed edges in a graph.

Imagine that there exists a graph consisting of 26 nodes wherein each node corresponds to one letter of the English language. You are given a list of words as well as a list of lists. Each of the sublists in the latter input represents a directed edge between two letters i.e. ('a', 'b') represents an edge going from a to b. Your task is to determine whether or not each word given in the input can be spelled out by tracing the edges in this graph, starting at any node. The output should be a list of 1's and 0's (1 = True, 0 = False), in order of the input words.

For example, let's say you get ['a', 'b', 'ab', 'ba'] and [('a', b')]. The output should be [1, 1, 1, 0].

YOUR ANSWER

We recommend you take a quick tour of our editor before you proceed. The timer will pause up to 90 seconds for the tour. Start tour

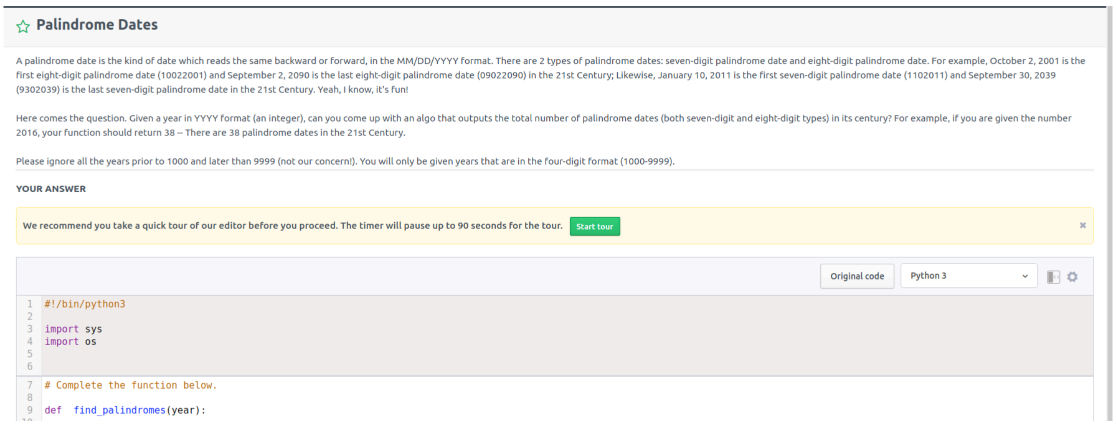
Draft saved 07:37 pm Original code Python 3

```
#!/bin/python3

import sys
import os


# Complete the function below.

```



Palindrome Dates

A palindrome date is the kind of date which reads the same backward or forward, in the MM/DD/YYYY format. There are 2 types of palindrome dates: seven-digit palindrome date and eight-digit palindrome date. For example, October 2, 2001 is the first eight-digit palindrome date (10022001) and September 2, 2090 is the last eight-digit palindrome date (09022090) in the 21st Century; Likewise, January 10, 2011 is the first seven-digit palindrome date (1102011) and September 30, 2039 (9302039) is the last seven-digit palindrome date in the 21st Century. Yeah, I know, it's fun!

Here comes the question. Given a year in YYYY format (an integer), can you come up with an algo that outputs the total number of palindrome dates (both seven-digit and eight-digit types) in its century? For example, if you are given the number 2016, your function should return 38 -- There are 38 palindrome dates in the 21st Century.

Please ignore all the years prior to 1000 and later than 9999 (not our concern!). You will only be given years that are in the four-digit format (1000-9999).

YOUR ANSWER

We recommend you take a quick tour of our editor before you proceed. The timer will pause up to 90 seconds for the tour.

```python
#!/bin/python3

import sys
import os


# Complete the function below.

def  find_palindromes(year):
```

Number Base Arithmetic

Complete the function `number_base_arithmetic`. This function takes in two strings of which the first is a numeric base between 2 and 36. The second input string is a number expressed in that base: if the base is <= 10, then every character must be between '0' and the character for the base - 1 (e.g. '0' .. '8' are valid for base 9). If the base is between 11 and 36, then every character must either be between '0' .. '9' or between 'a' .. 'z' (inclusive, and only lower case letters). In the latter case, the value for the character is either 0 .. 9, or 10 + the distance between the character and 'a' -- for example, if the base is 18, then characters must be between '0' .. '9', or 'a' .. 'h', and in the latter case, the value of the character is 10 if the character is 'a'; 11 if it is 'b'; 17 if it is 'h.

Your program should check whether the base is valid. If it is not, your program should return:
`base invalid`.

If the base is valid, your program should use the base to read the next string as a number in that base (using the character rules above) and convert the number to base 10.

While converting the number, if your program detects that the number is not valid in the given base, your program should return:
`number invalid`.

If the number is valid, your program should simply return the number in base 10, for example
`633`

In every case, your program should return a string.

Examples:

```
> number_base_arithmetic('1', '1010')
base invalid
> number_base_arithmetic('8', '19')
number invalid
> number_base_arithmetic('2', '1010')
10
> number_base_arithmetic('11', 'a1')
111
> number_base_arithmetic('12', 'c33')
number invalid
```

Your task is to analyze a ranked election. Here's how the election works: each ballot consists of an ordered list of candidates based on a voter's preference. The ballots are processed in rounds. During each round, the first choice on each ballot is tallied up. If a candidate has the most votes AND surpasses a certain percentage of the votes, they win the election. If no candidate reaches the necessary percentage, then the candidate with the fewest votes is eliminated from all ballots. Once again the first choice on each ballot is tallied (remember, this is AFTER removing the last place finisher of the previous round). This process continues until a candidate reaches the necessary percentage. Some extreme cases to think about:

- If multiple candidates receive the same number of votes AND reach the necessary percentage in the SAME ROUND, the election is thrown out and nobody wins.
- If multiple candidates tie for last place, they are ALL removed from the ballots for the next round. This means you can be left with no candidates, in which case, the election is thrown out and nobody wins.
- Voters are not required to order all candidates on their ballot (i.e. even if there are 10 candidates, a ballot can consist of 3 ordered candidates)

To analyze the election, you must determine which candidate will win for a certain percentage given all the ballots. The expected output is a list of percentage intervals and the winning candidate for each interval. We are only concerned with whole number percentage points (i.e. 10.5% - 20% is not a valid interval).

Input Format:
num_candidates -> int
num_ballots -> int
ballots -> 2D int array with num_ballots rows and num_candidates columns *
* Candidates are labeled from 0 to num_candidates - 1. If a voter orders fewer than num_candidates, the blanks will be filled with -1.

Output Format:
2D int array of the form [[pct1, pct2, candidate], [pct3, pct4, candidate]...] *
* The intervals are inclusive on both sides. You must cover the entire percentage range from 0 to 100. If an interval exists where no candidate can win, put -1 in the candidate slot. You should minimize the number of intervals where possible.

Sum of unique fibonacci numbers

Given positive integers x and n, determine if x can be expressed as a sum of n unique fibonacci numbers. Fibonacci numbers are from a series 1, 1, 2, 3, 5, 8, 13, 21, ... in which starting from the third number, each number is the sum of the previous two numbers.

	Input Format Arguments: x: int, target sum n: the number of integers Constraints: x <= 10^9 n <= 5	**Output Format** A boolean value indicating whether this can be achieved.

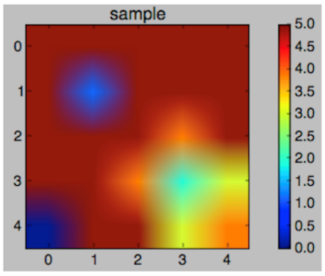
	Sample Input
	x = 6, n = 2
	x = 5, n = 3
	x = 10059560, n = 4
		Sample Output
		True
		False
		True

☆ Minima

Find the 3 most extreme local minima of a 2D surface.

Input Format	Output Format
The surface is given as a square matrix (n by n). A minima is a value that is smaller than all its neighbors. Arguments: n: int, the order of the array m: a 2D array (list of lists) of floats Constraints: n < 50	An array (list) of floats, sorted ascending, of the minima values. Return a max of 3 values. If none are found, return an empty array (list).

Sample Input



```
n = 5
m= [[5., 5., 5., 5., 5.],
    [5., 1., 5., 5., 5.],
    [5., 5., 5., 4., 5.],
    [5., 5., 4., 2., 3.],
    [0., 5., 5., 3., 4.]]
```

Sample Output

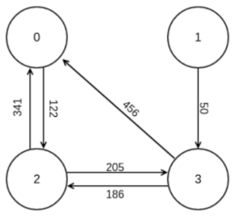
```
[0., 1., 2.]
```

☆ One-to-All Time on a Sparse Directional Network

Suppose you have a network of devices. Each device is directionally connected to a subset of other devices and each communication link has some fixed delay.

Suppose you want to send a One-to-All broadcast message. In this case, some origin node attempts to send a message to every other node. It sends the message to each of it's outbound neighbors, and each of those nodes forwards the message on to their outbound neighbors, and so on. Each edge traversal incurs a time-penalty equal to the edges weight.

For example, you may have some topology like this:



One can define such a network using an adjacency matrix where each row defines the outbound edges, and each column, the inbound, using Null/None to refer to the absence of an edge, and an Integer value indicating the edge penalty.

For example, the above network can be represented as:

Device ID	0	1	2	3
0	None	None	122	None
1	None	None	None	50
2	341	None	None	205

One can define such a network using an adjacency matrix where each row defines the outbound edges, and each column, the inbound, using Null/None to refer to the absence of an edge, and an Integer value indicating the edge penalty.

For example, the above network can be represented as:

Device ID	0	1	2	3
0	None	None	122	None
1	None	None	None	50
2	341	None	None	205
3	456	None	186	None

Write a function called "*broadcast_delivery_time*" that takes the ID of the origin device (list index of node), and an adjacency matrix as a list of lists (row-wise IE: input[0] == ROW 1), and outputs the minimum guaranteed time of delivery, or Null if the message *cannot* be delivered. That is, the time required for the message to have been received by every other device, or Null if the message *cannot* be delivered to every other device.

IE your function signature should look something like the following:

```
func broadcast_delivery_time(origin_id (int), adj_matrix (list of lists[int])) -> int or Null/None
```

☆ Detect colinearity

Given n distinct lattice points (i.e., coordinates are integers) on the xy plane, determine if there are three points that lie on a straight line. There are at least 3 points in all test cases.

Input:
The first line is the number of points n. This number is guaranteed to be at least 3 in all test cases
The second line is the dimension of the points, which is fixed to be 2 in this problem.
The following n lines consists of two integers, separated by space.

Output:
If there are three points lying on the same line, return 1. Else return 0

Example 1:
Input:
3
2
-1 -1
0 0
1 1
Output:
1

Example 2:
Input:
3
2
-1 -1
0 0
1 2
Output:
0

☆ Minima

Find the 3 most extreme local minima of a 2D surface.

Input Format

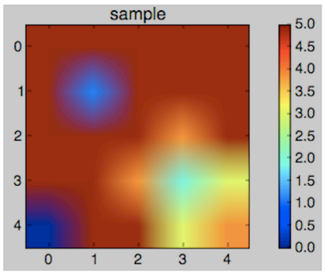
The surface is given as a square matrix (n by n).
A minima is a value that is smaller than all its neighbors.

Arguments:
    n: int, the order of the array
    m: a 2D array (list of lists) of floats
Constraints:
    n < 50

Output Format

An array (list) of floats, sorted ascending, of the minima values.
Return a max of 3 values. If none are found, return an empty array (list).

Sample Input



```
n = 5
m= [[5., 5., 5., 5., 5.],
    [5., 1., 5., 5., 5.],
    [5., 5., 5., 4., 5.],
    [5., 5., 4., 2., 3.],
    [0., 5., 5., 3., 4.]]
```

Sample Output

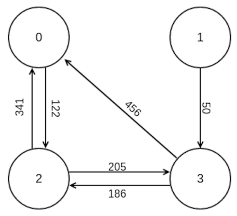
[0., 1., 2.]

☆ One-to-All Time on a Sparse Directional Network

Suppose you have a network of devices. Each device is directionally connected to a subset of other devices and each communication link has some fixed delay.

Suppose you want to send a One-to-All broadcast message. In this case, some origin node attempts to send a message to every other node. It sends the message to each of it's outbound neighbors, and each of those nodes forwards the message on to their outbound neighbors, and so on. Each edge traversal incurs a time-penalty equal to the edges weight.

For example, you may have some topology like this:



One can define such a network using an adjacency matrix where each row defines the outbound edges, and each column, the inbound, using Null/None to refer to the absence of an edge, and an Integer value indicating the edge penalty.

For example, the above network can be represented as:

Device ID	0	1	2	3
0	None	None	122	None
1	None	None	None	50
2	341	None	None	205

One can define such a network using an adjacency matrix where each row defines the outbound edges, and each column, the inbound, using Null/None to refer to the absence of an edge, and an Integer value indicating the edge penalty.

For example, the above network can be represented as:

Device ID	0	1	2	3
0	None	None	122	None
1	None	None	None	50
2	341	None	None	205
3	456	None	186	None

Write a function called "*broadcast_delivery_time*" that takes the ID of the origin device (list index of node), and an adjacency matrix as a list of lists (row-wise IE: input[0] == ROW 1), and outputs the minimum guaranteed time of delivery, or Null if the message *cannot* be delivered. That is, the time required for the message to have been received by every other device, or Null if the message *cannot* be delivered to every other device.

IE your function signature should look something like the following:

```
func broadcast_delivery_time(origin_id (int), adj_matrix (list of lists[int])) -> int or Null/None
```

☆ Detect colinearity

Given n distinct lattice points (i.e., coordinates are integers) on the xy plane, determine if there are three points that lie on a straight line. There are at least 3 points in all test cases.

Input:
The first line is the number of points n. This number is guaranteed to be at least 3 in all test cases
The second line is the dimension of the points, which is fixed to be 2 in this problem.
The following n lines consists of two integers, separated by space.

Output:
If there are three points lying on the same line, return 1. Else return 0

Example 1:
Input:
3
2
-1 -1
0 0
1 1
Output:
1

Example 2:
Input:
3
2
-1 -1
0 0
1 2
Output:
0

3-Card Poker

Imagine a game, similar to many poker games, in which each player is dealt 3 cards, and the player with the higher value hand wins the draw. We play this game with a 40-card deck: 4 cards of each value 0-9, and for simplicity, unsuited.

Hand rank is as follows (highest to lowest):
Three of a kind: All three cards have the same value.
One pair: Two cards of the same value.
High card: The highest value card in your hand.

Cards are valued (highest to lowest) 9, 8, ... 0.

If two players have the same ranked hand, then the rank made of the highest value wins, for example, a pair of 9's beats a pair of 2's, and a high card 9 beats a high card 2. If, however, both players have a pair of 8's, then the player with the higher value third card wins. For the high card case, if the second cards also tie, then the third card is considered, and if this card is also tied, then the hand results in a draw.

Consider the following five dealt hands:

P1	Rank	P2	Rank	Winner
742	HC	653	HC	P1
882	1P	742	HC	P1
752	HC	742	HC	P1
884	1P	882	1P	P1

Flipping signs

Given a string made of '+' and '-' signs of length L, the only allowed operation is to flip K consecutive signs at the same time.

Input:
String: length L, made of only '+' and '-'
Integer: K

Output:
Integer: the minimum number of times needed to flip all signs to '+'. if not possible, output -1.

Constraints:
L >= K
K >= 2

Packing Melons

Before you are two assembly lines, one with boxes and one with watermelons, both of varying size. Your desire is to put as many watermelons into boxes as possible. You can pick where you start taking watermelons from, but once you start, all melons going past you must be placed, or you must stop. We want you to calculate how many watermelons you can place.

As input, you are given two lists, one of box sizes and one of watermelon sizes. A watermelon will fit into a box with a number greater than or equal to the melon's. Only one melon can go into a box. You can hold onto the melon and skip a box to place it in a later one. Calculate how many watermelons can be placed.

Flipping signs

Given a string made of '+' and '-' signs of length L, the only allowed operation is to flip K consecutive signs at the same time.

Input:
String: length L, made of only '+' and '-'
Integer: K

Output:
Integer: the minimum number of times needed to flip all signs to '+'. if not possible, output -1.

Constraints:
L >= K
K >= 2