r/6thForm

# > CompChallenge 2020*

*According to the ISO8601 week at least

---

## Challenge A-1 | Chain

📃 A **3-chain** is a line of three **consecutive** symbols on a 2D grid - horizontally, vertically, or diagonally.

For example, the following $3 \times 3$ grids have a **3-chain**:

| | | |
|---|---|---|
| ! | % | ( |
| * | * | * |
| ) | % | ! |

| | | |
|---|---|---|
| B | A | A |
| E | R | A |
| P | E | A |

| | | |
|---|---|---|
| 3 | 6 | 9 |
| 5 | 9 | 4 |
| 9 | 1 | 1 |

💡 Write a function, `Chain`, that takes a $3 \times 3$ array as input, representing a grid of symbols.

The function should determine whether at least one **3-chain** is present on the grid.
If a **3-chain** exists, return `True`. Otherwise, return `False`.

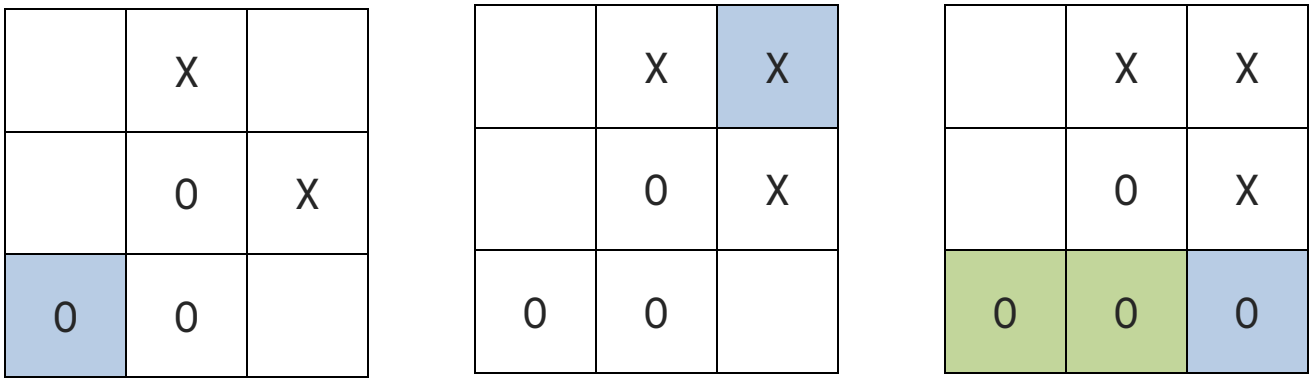
⚙ To help test your solution, input grids are provided in the `input_a1.txt` file.
Feel free to alter or add to the existing testing data, which includes these example cases:

| | | |
|---|---|---|
| `[["£", "*", "£"],`<br>`["^", "£", "£"],`<br>`["$", "%", "£"]]` | `[["/", "#", "/"],`<br>`["-", "#", "\"],`<br>`["-", "/", "\"]]` | `[["<", "=", ">"],`<br>`["<", "+", ">"],`<br>`["<", "?", ">"]]` |
| True | False | True |

## Challenge A-2 | TicTacToe

📃 The game **tic-tac-toe** begins with an empty $3 \times 3$ grid. Player 1 and 2 are assigned the symbols X and O respectively. Then, they take it in turns to place a copy of their symbol on one of the empty tiles.

- Either player may make the first move, this is decided beforehand.
- If either player gets a **3-chain** of their symbol, the game immediately ends – that player has won.
- If no empty tiles are remaining and no player has a **3-chain** – the game ends in a draw.




*Better luck next time, cross!*

More in-depth information about the game can be found here.

💡 Write a second function, TicTacToe**,** that takes a $3 \times 3$ array as input, representing a grid of symbols.

- Empty tiles are represented by a **–**.
- Player tiles are represented by either an **X** or **0**.
- No other symbols should be accepted.

If the grid represents an **obtainable tic-tac-toe state**, return `True`. Otherwise, return `False`.
You may use Challenge A to assist you.

⚙ To help test your solution, input grids are provided in the `input_a2.txt` file.
Feel free to alter or add to the existing testing data, which includes these example cases:

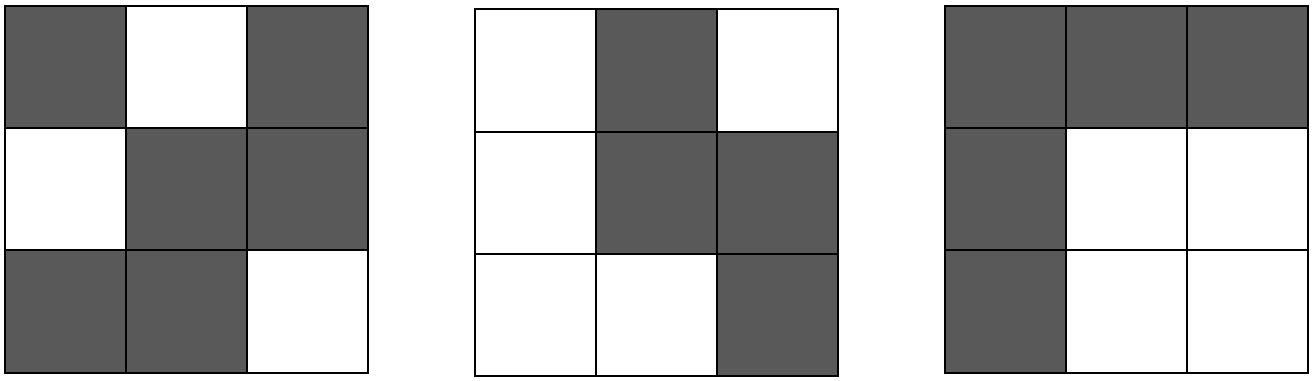
| `[["X", "-", "0"], ["X", "-", "0"], ["X", "-", "0"]]` | `[["X", "0", "X"], ["0", "0", "0"], ["X", "0", "X"]]` | `[["X", "-", "0"], ["0", "X", "X"], ["X", "-", "-"]]` |
|---|---|---|
| `False` | `True` | `False` |

`[35 Points - Section A]`

## Challenge B-1 | MaxChain

A $k$-chain is a line of $k$ consecutive symbols on a 2D grid – horizontally, vertically, or diagonally.

A **blocking grid** has only two types of squares – blocked or unblocked.
It is impossible to form a $k$-chain using blocked squares. For example, compare the following:



Only one of these grids has a **3-chain**. However, two of them have a **2-chain**, and all three have a **1-chain**.

Write a third function, MaxChain, taking an $n \times n$ array as input that represents a blocking grid.

- Blocked squares are represented by the character **B**.
- Unblocked squares are represented by the character **U**.
- Constraint: $1 \leq n \leq 500$

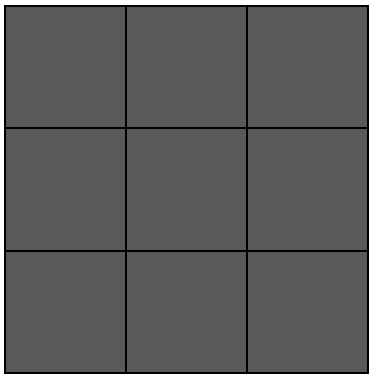
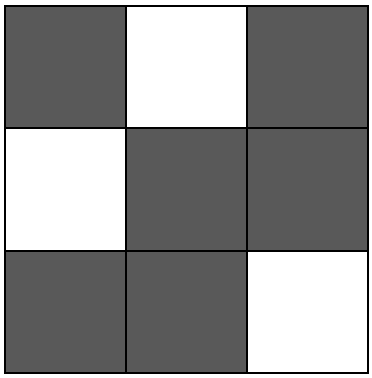
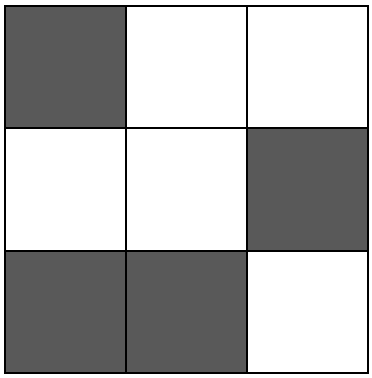
Return the length of the **longest unblocked chain** that can be formed on the grid.
You may use Challenge A or B to assist you.

To help test your solution, input grids are provided in the `input_b1.txt` file.
Feel free to alter or add to the existing testing data, which includes these example cases:

| | | |
|---|---|---|
| `[["B", "U"],`<br>`["U", "B"]]` | `[["U", "B", "U"],`<br>`["U", "U", "B"],`<br>`["B", "U", "B"]]` | `[["U", "B", "U", "B"],`<br>`["B", "U", "U", "B"],`<br>`["U", "B", "U", "U"],`<br>`["B", "U", "U", "B"]]` |
| 2 | 2 | 4 |

## Challenge B-2 | TicTacNo

📰 Sometimes it's easy to find a solution to a problem – but more difficult to find a **good solution**.

Although three of these grids prevent a **3-chain** from being formed, some use less blocked squares.

💡 Write a fourth function, TicTacNo**,** taking a single integer $n$ (with $1 \leq n \leq 1000$) as input.

Return the **smallest number of blocked squares** needed to prevent an $n$-chain on an $n \times n$ grid.
You may use previous challenges to assist you.

⚙ To help test your solution, integers are provided in the `input_b2.txt` file.
Feel free to alter or add to the existing testing data, which includes these example cases:

| 1 | 2 | 3 |
|---|---|---|
| 1 | 3 | 3 |

## Challenge B-3 | ChainBlocker

📰 Other times, finding the best solution to a problem may not be feasible in terms of available resources. Instead, we might aim to find a **reasonably good solution** to work around this.

💡 Write a final function, ChainBlocker, taking two integers $n$ and $k$ as input.

Return a $n \times n$ array representing a blocked grid such that no $k$-chain can be formed.

- Blocked squares should be represented by the character B.
- Unblocked squares should be represented by the character U.
- Constraints: $1 \leq n \leq 200$ and $1 \leq k \leq 200$

While there are many solutions, your aim is to **minimise the number of blocked squares**.

Your solution should return an answer for any given input within a reasonable timeframe.
You may use previous challenges to assist you.

⚙ To help test your solution, integer pairs are provided in the `input_b3.txt` file.
Feel free to alter or add to the existing testing data, which includes these example cases:

| n=2, k=1 | n=3, k=2 | n=4, k=3 |
|---|---|---|
| [[“B”, “B”],<br>[“B”, “B”]] | [[“U”, “B”, “U”],<br>[“B”, “B”, “B”],<br>[“U”, “B”, “U”]] | [[“U”, “U”, “B”, “U”],<br>[“U”, “B”, “B”, “B”],<br>[“B”, “U”, “U”, “B”],<br>[“U”, “U”, “B”, “U”]] |

*There may be multiple blocking solutions of minimum length.*

[65 Points - Section B]