Exercises

Just to give you some context, sometimes we make some not very smart mistakes (we are only human…). Tools like SonarLint are a valuable help in these situations. SonarLint statically analyzes code to quickly find issues.

For all the exercises proposed you should take some time to analyze the code WITHOUT using the tool. As an exceptionally talented programmer, you should be able to catch some of these issues without using it.

Then you can view the code on *VS Code* and *SonarLint* will outline the issues for you and give tips for fixing them! As easy as that!

You should take into account that, in some cases, fixing one issue reveals another set of brand new issues for you to enjoy! Fun right?

For additional information on the issues go here.

Exercise 1

file: *ex1.js*

solution: *sol1.js* (don't do it, any doubts ask us first!!)

The main goal of this exercise is for you to understand how the tool works and the different issues it identifies. In order to do that, four simple functions are presented to you. You should start fixing them in order.

Have fun 😃

Exercise 1.a.

In this function, it is expected that you fix 3 (+1 bonus) different issues.

```
// return the op between two POSITIVE numbers
// if the numbers are not >= 0, returns -1
// if op is not SUM, SUB, PROD or DIV, returns -2
function exA(a, b, op, op) {
  let res;
  if (a < 0 || b < 0)
    return - 1;
  switch (op) {
    case 'SUM':
      res = a + b;
      break;
    case 'SUB':
      res = a - b;
    case 'PROD':
      res = a * b;
      break;
```

```
    // sonarLint does not flag this, ESlint would...
    // lets pretend it does and fix it anyways :)
    case 'SUM':
      res = a + b;
      break;
    case 'DIV':
      res = a / b;
      break;
  }
  return res;
}
```

Exercise 1.b.

In this function, it is expected that you fix 4 different issues.

```
// returns INVALID OP if the result of the op between 2 nr is < 0
// returns ZERO if the result of the op between 2 nr is == 0
// returns POSITIVE if the result of the op between 2 nr is > 0
function exB(a, b, op) {
  const value = exA(a, b);
  if (value < 0) {
    return 'INVALID OP';
  } else if (value == 0) {
    return 'ZERO';
  } else if (value_ > 0) {
    return 'POSITIVE';
  } else if (value < 0) {
    return 'NEGATIVE';
  }
}
```

Exercise 1.c.

In this function, it is expected that you fix 5 different issues.

```
// returns true if the result of the op between two positive numbers is >=
0, false otherwise
// yes, there are better ways to do this, just do the exercise ;)
function exC(a, b, op) {
  const value = exB(a, b, op);
  if (vallue != 'INAVLID OP') {
    const c = true;
  } else {
    const b = false;
  }
```

```
  return c;
}
```

Exercise 1.d.

In this function, it is expected that you fix 2 different issues.

```
function exD(a, b) {
  const ops = ["SUB", "PROD", "DIV"];
  let i = 0;
  let max = exA(a,b, "SUM");
  while (i < 3) {
    const opRes = exA(a,b, ops[i])
    if (opRes > max)
      max = opRes
  }
  return max
}
```

Exercise 2

file: *ex2.js*

solution: *sol2.js* (nooo!)

This exercise presents a more “real” application of this tool. It is a very simple program that implements the tic tac toe game. As in the previous exercise, we advise you to fix the issues in order.

We are expecting you to encounter around 15/16 issues (it will depend on your resolution).

Have fuuuun! When you FINISH the exercise you can play the game in the index.html file 😉

```
const statusDisplay = document.querySelector('.game--status');

const gameActive = true;
let currentPlayer = 'X';
let gameState = ['', '', '', '', '', '', '', '', ''];

const winningMessage = () => `Player ${currentPlayer} has won!`;
const drawMessage = () => 'Game ended in a draw!';
const currentPlayerTurn = () => `It's ${currentPlayer}'s turn`;

statusDisplay.innerHTML = currentPlayerTurn();

const winningConditions = [
  [0, 1, 2],
  [3, 4, 5],
  [6, 7, 8],
  [0, 3, 6],
```

```
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6],
  ];

  function handleCellPlayed(clickedCell, clickedCellIndex) {
    gameState[clickedCellIndex] = currentPlayer;
    clickedCell.innerHTML = currentPlayer;
  }

  function handlePlayerChange() {
    currentPlayer = currentPlayer === 'X' ? 'O' : 'X';
    nextPlayer = currentPlayer;
    statusDisplay.innerHTML = currentPlayerTurn();
  }

  function handleResultValidation() {
    let roundWon = false;

    for (let i = 0; i <= 7; i++) {
      const winCondition = winningConditions[i];
      const a = gameState[winCondition[0]];
      let b = gameState[winCondition[0]];
      b = gameState[winCondition[1]];
      const c = gameState[winCondition[2]];
      const val = false;


      if (a ==='' || b === '' || c === '') {
        continue;
      }
      if (a === b && b === c) {
        roundWon = true;
        break;
      }
    }

    if (roundWon) {
      statusDisplay.innerHTML = winningMessage();
      gameActive = false;
      return;
    }

    const roundDraw = !gameState.includes('');
    if (roundDraw) {
      statusDisplay.innerHTML = drawMessage();
      gameActive = false;
      return;
    }

    handlePlayerChange();
  }
```

```
function handleCellClick(clickedCellEvent) {
  const clickedCell = clickedCellEvent.target;
  const clickedCellIndex = parseInt(clickedCell.getAttribute('data-cell-
index'));

  if (gameState[clickedCellIndex] != '' || !gameActive) {
    return;
  }

  handleCellPlayed(clickedCell, clickedCellIndex);
  handleResultValidation();
}

function handleRestartGame() {
  gameActive = true;
  currentPlayer = 'X';
  gameState = ['', '', '', '', '', '', '', '', ''];
  statusDisplay.innerHTML = currentPlayerTurn();
  document.querySelectorAll('.cell').forEach((cell) => cell.innerHTML =
'');
}

function setGameActive(value, oldValue) {
  gameActive = value;
}

document.querySelectorAll('.cell').forEach((cell) =>
cell.addEventListener('click', handleCellClick));
document.querySelector('.game--restart').addEventListener('click',
handleRestartGame);
```