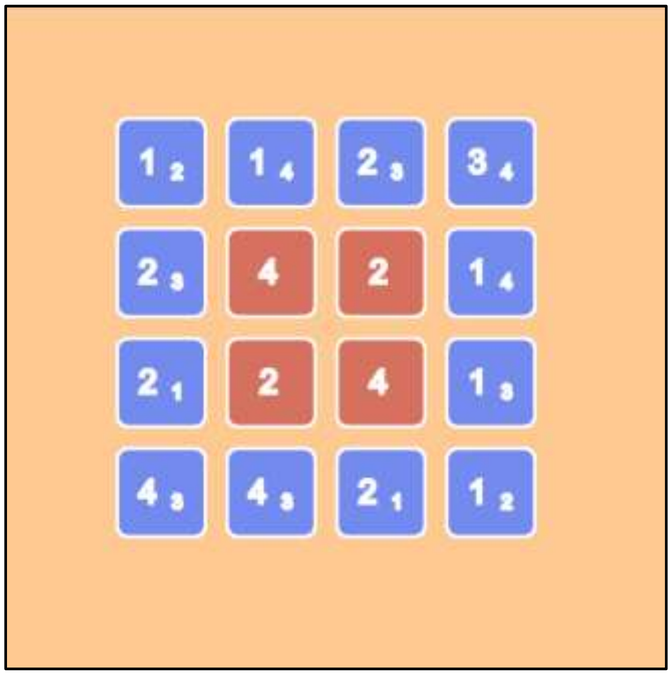
Cross Set Documentation

Cross Set is a game where you are given a NxN field with numbers. For each square there are 2 possible numbers to put in (except from the special squares). The target of the game is to have all possible numbers on each row and column without them repeating. In this exercise we are going to create that game using JavaScript and the p5 library (helps us draw in canvas using JavaScript) At the end, the game should look like this:



1. What are we going to use?

1.1. Web Storm

Powerful IDE for JavaScript. If you don't have it, you can download it from here:

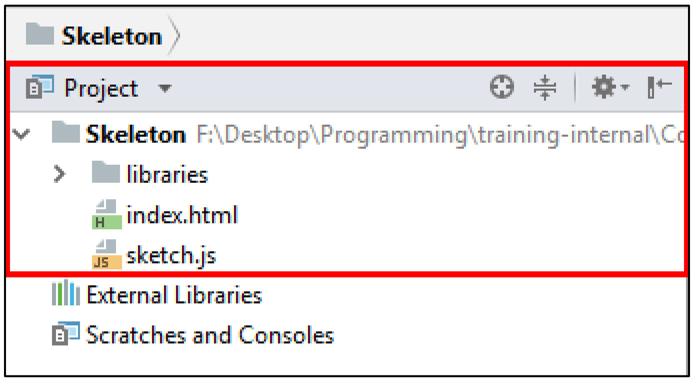
https://www.jetbrains.com/webstorm/download/#section=windows.

1.2. P5 Library

To draw easily we are going to use the p5 library. Unzip the zip-file given to you. It contains all the files you are going to need. You may want to read the reference for p5 to help you follow along. https://p5js.org/reference/

2. Setting up the project

Create a project containing all the files from the archive

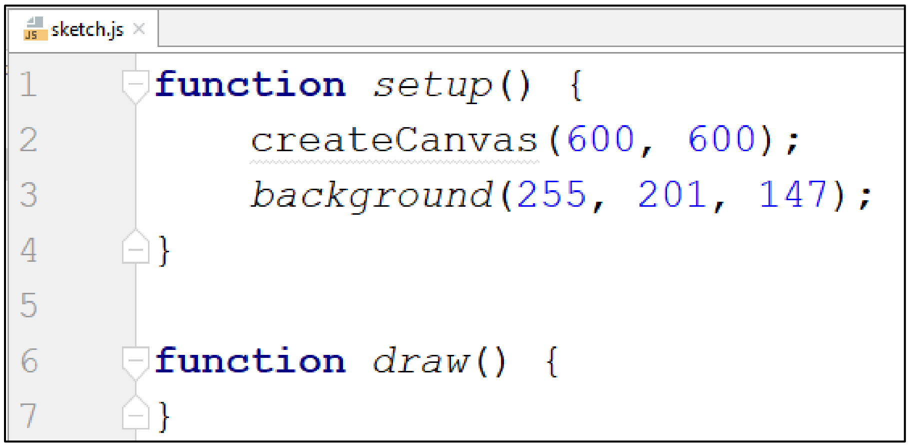


- **Index.html** is the file we are going to run in the browser (just by double-clicking)
- **Sketch.js** is the file, where our main logic is going to be
- **Libraries folder** contains all the additional functions we are going to use for drawing

3. First steps

3.1. Drawing the canvas

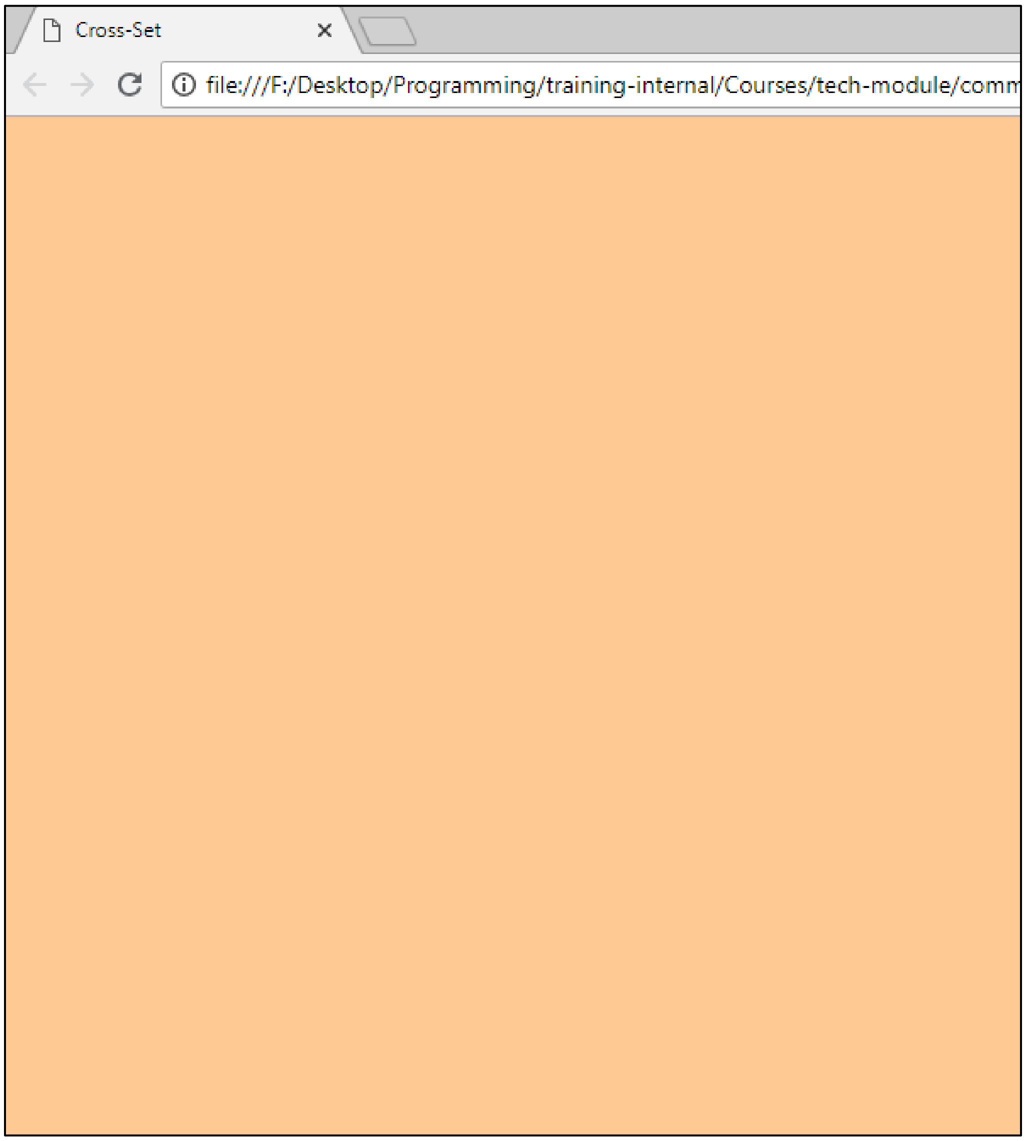
The first thing we are going to do, is to draw our canvas. Open the sketch.js file and write the following code:



```
function setup() {
    createCanvas(600, 600);
    background(255, 201, 147);
}

function draw() {
}
```

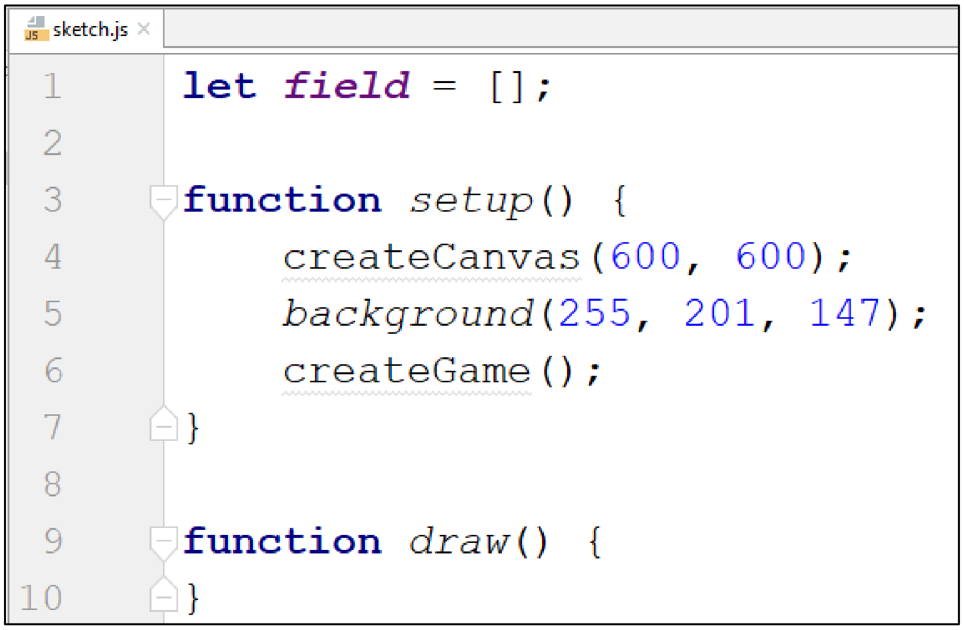
- **createCanvas** – build-in function in the p5 library. It allows us to draw a canvas with specific width and height
- **background** – build-in function that receives 3 values (r, g, b) to set up the color of the canvas

If you now run the index.html file you should see this:


Cross-Set
file:///F:/Desktop/Programming/training-internal/Courses/tech-module/comm

3.2 Creating the field

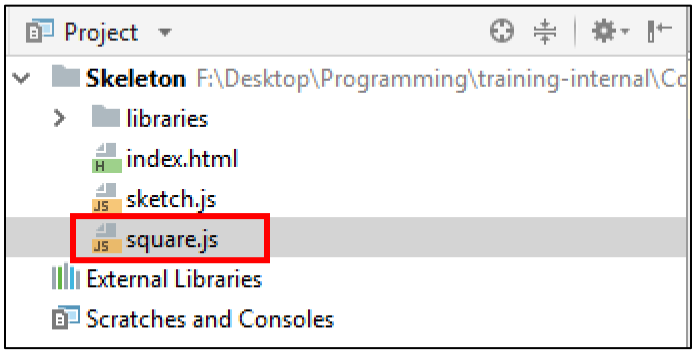
On the top of the sketch.js file create an empty array that will hold all of the squares:



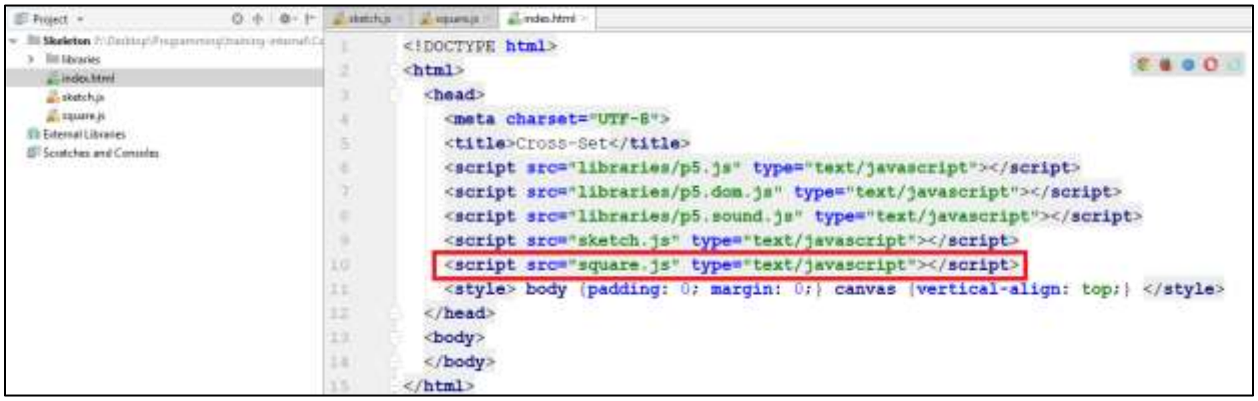
```
let field = [];

function setup() {
      createCanvas(600, 600);
      background(255, 201, 147);
      createGame();
}

function draw() {
}
```

- We also call a function called **createGame**. We don't have it yet, but we will write it in a bit. It is going to create all the squares of the field.

Before doing that, we are going to create a separate js file. It is going to contain the square class (since the field is made of many squares). Create a new js file called **square.js.**

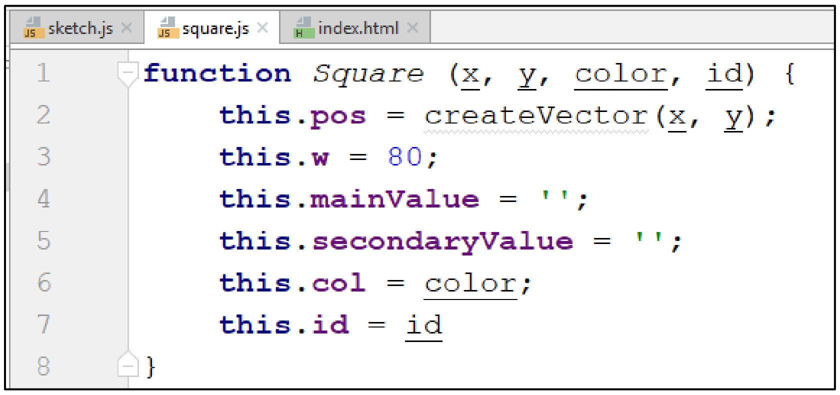


Link the new file in the **index.html** file. We must do that, so the html knows that it must use it.



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Cross-Set</title>
    <script src="libraries/p5.js" type="text/javascript"></script>
    <script src="libraries/p5.dom.js" type="text/javascript"></script>
    <script src="libraries/p5.sound.js" type="text/javascript"></script>
    <script src="sketch.js" type="text/javascript"></script>
    <script src="square.js" type="text/javascript"></script>
    <style> body {padding: 0; margin: 0;} canvas {vertical-align: top;} </style>
  </head>
  <body>
  </body>
</html>
```

Now return to the square.js file and write the following code:

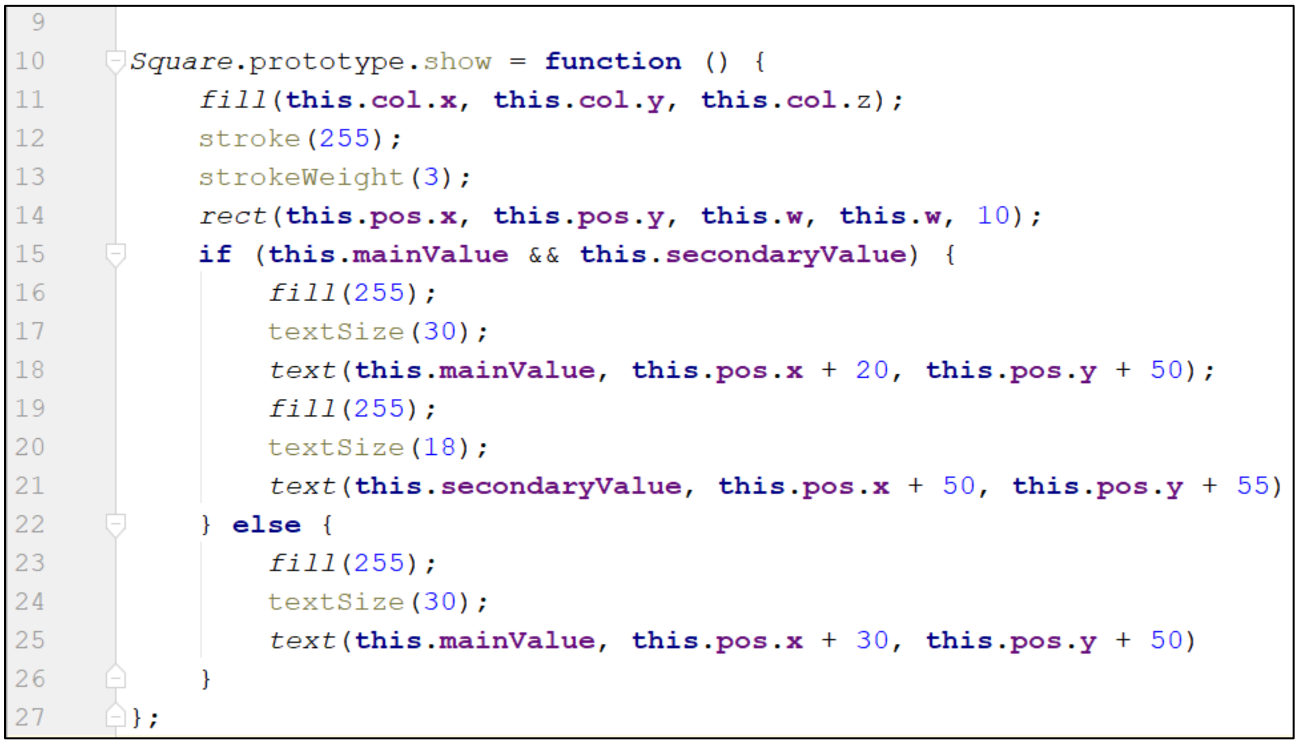


sketch.js × square.js × index.html ×

```
function Square (x, y, color, id) {
      this.pos = createVector(x, y);
      this.w = 80;
      this.mainValue = '';
      this.secondaryValue = '';
      this.col = color;
      this.id = id
}
```

- We create the **Square class** – the constructor should receive an **x, y, color and id**
- The **X and Y** will be the **coordinates of the square** (we store them in a **vector**). **createVector** is build-in function in p5 to **store two values**
- Each square will have a **width (w)**
- Each square is going to have a **main value** and a **secondary value** (the two numbers to choose from). At the beginning they are **empty** in the beginning. We are going to set them later.
- Each square is also going to have a **color** (depending on whether the square can be clicked or not). It will be a **Vector3** (storing 3 values – R, G, B)
- In order to know on which square we are, we are going to have an **Id** for each of them

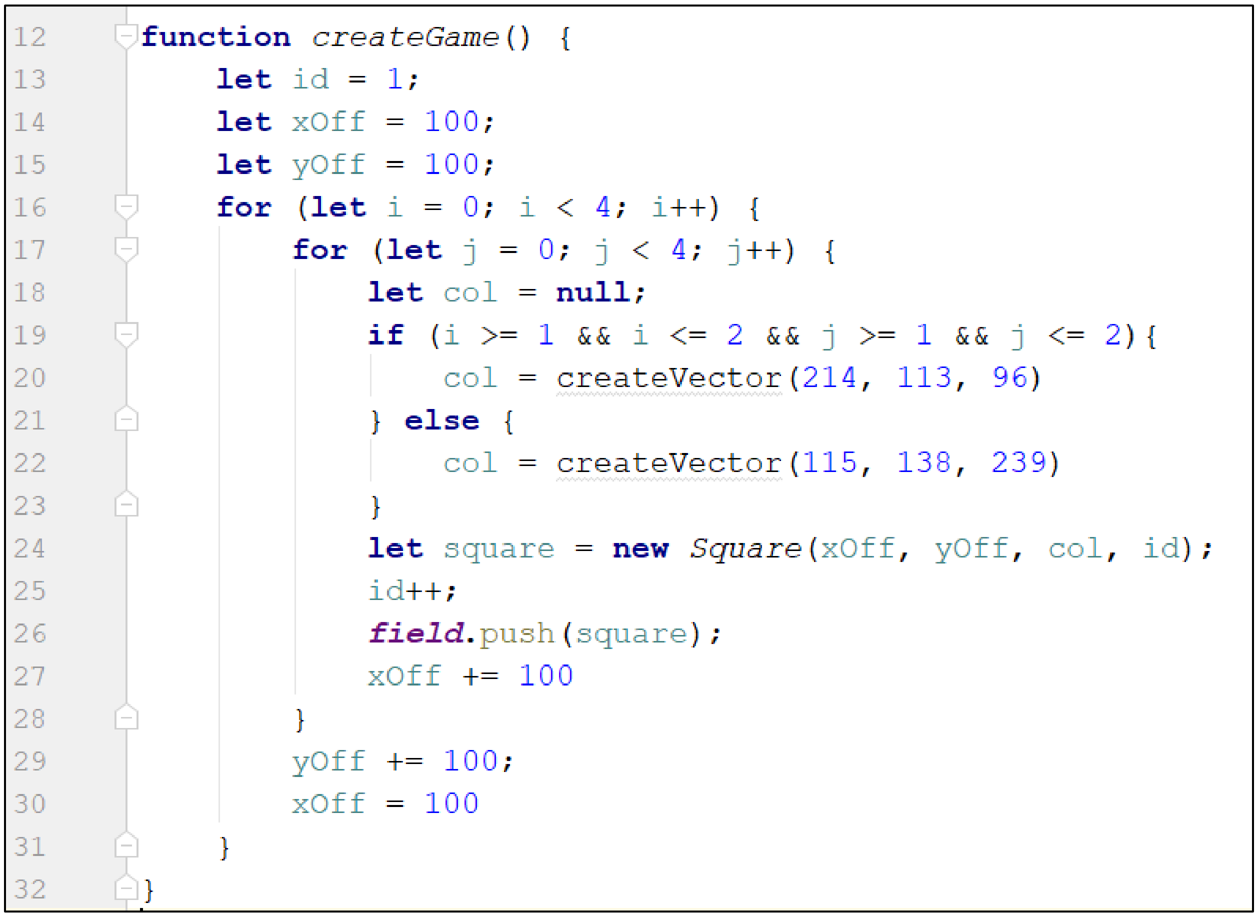
Now we need a function to draw the square. Add the following code:



```

Square.prototype.show = function () {
     fill(this.col.x, this.col.y, this.col.z);
     stroke(255);
     strokeWeight(3);
     rect(this.pos.x, this.pos.y, this.w, this.w, 10);
     if (this.mainValue && this.secondaryValue) {
         fill(255);
         textSize(30);
         text(this.mainValue, this.pos.x + 20, this.pos.y + 50);
         fill(255);
         textSize(18);
         text(this.secondaryValue, this.pos.x + 50, this.pos.y + 55)
     } else {
         fill(255);
         textSize(30);
         text(this.mainValue, this.pos.x + 30, this.pos.y + 50)
     }
};
```

- We create a **show function** attached to the **prototype** of the square. That means, whenever we **type {square}.show()** it will call the function for that **particular instance of the square object**
- **Fill** – build-in function that sets the color to **given RGB values** (we use the color of the square)
- **Stroke** – build-in function that **sets a color for border** (thickness)
- **StrokeWeight** – build-in function that sets the **thickness of the border**
- **Rect** – build-in function that **draws a rectangle (it requires x, y, width, height) and an optional value for angle (how oval the square should be).** Since the square is a square, the width and height we pass in are the w parameter of the square
- We need to check if the square **has both main and secondary value** (if it is **clickable**) to set its color to some blueish color. We also draw the **values in the square**. If it is **not clickable**, we only draw **one value (main) and use reddish color**
- We set the **text size to 30 for the main value** and **18 for the secondary value**.
- **Text** – build-in function that requires a **string (for the text), x , y (we use the position of the square with a little bit of offset both on the X and the Y in order to draw the text near to the center).** For **the secondary value** we draw in a **little bit to the right**

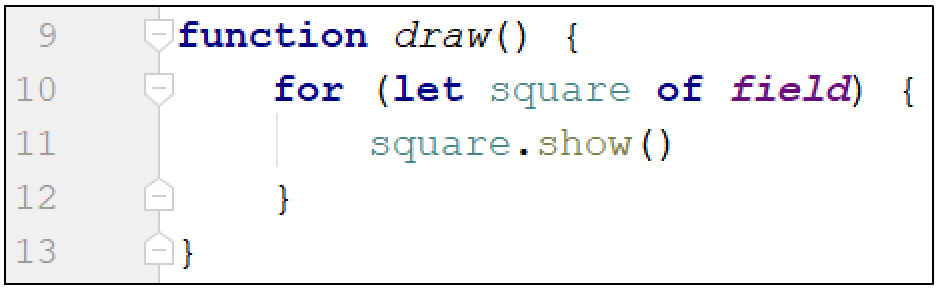
Now go back to the sketch.js function and create the createGame function. Write the following code:



```
function createGame() {
    let id = 1;
    let xOff = 100;
    let yOff = 100;
    for (let i = 0; i < 4; i++) {
        for (let j = 0; j < 4; j++) {
            let col = null;
            if (i >= 1 && i <= 2 && j >= 1 && j <= 2){
                col = createVector(214, 113, 96)
            } else {
                col = createVector(115, 138, 239)
            }
            let square = new Square(xOff, yOff, col, id);
            id++;
            field.push(square);
            xOff += 100
        }
        yOff += 100;
        xOff = 100
    }
}
```

- First, we need an **Id variable** to store the id of the squares we create.
- We make **two nested loops**, each iterating 4 times (we will have 4x4 field). ***<u>Feel free to make it as you like</u>***
- We are also going to have an **xOff** and **yOff** which we will use as **spacing between the squares**. They will **increase each column and the xOff will be reseted to a 100 on each new row**
- In this game, we will create the **4 squares in the middle to NOT be clickable**, so we add an if-statement to check that (for **unclickable we create vector with reddish RGB value**, otherwise we **make it blueish**). ***<u>Feel free to make it as you like</u>***
- We create a square at that **xOff** and **yOff** with **color** and **id**
- We **push the new square in the field** and we **increase the id**

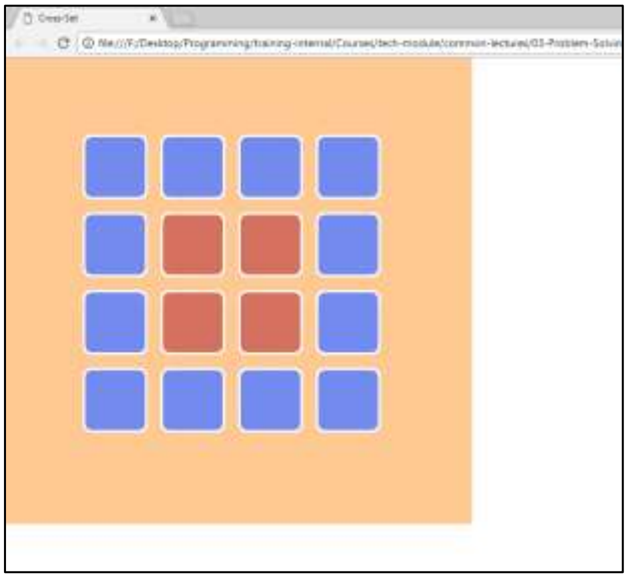
Last step before seeing the game for the first time. We must draw the squares after we have pushed them to the field. Now add the following code in the draw function:



```
function draw() {
    for (let square of field) {
        square.show()
    }
}
```

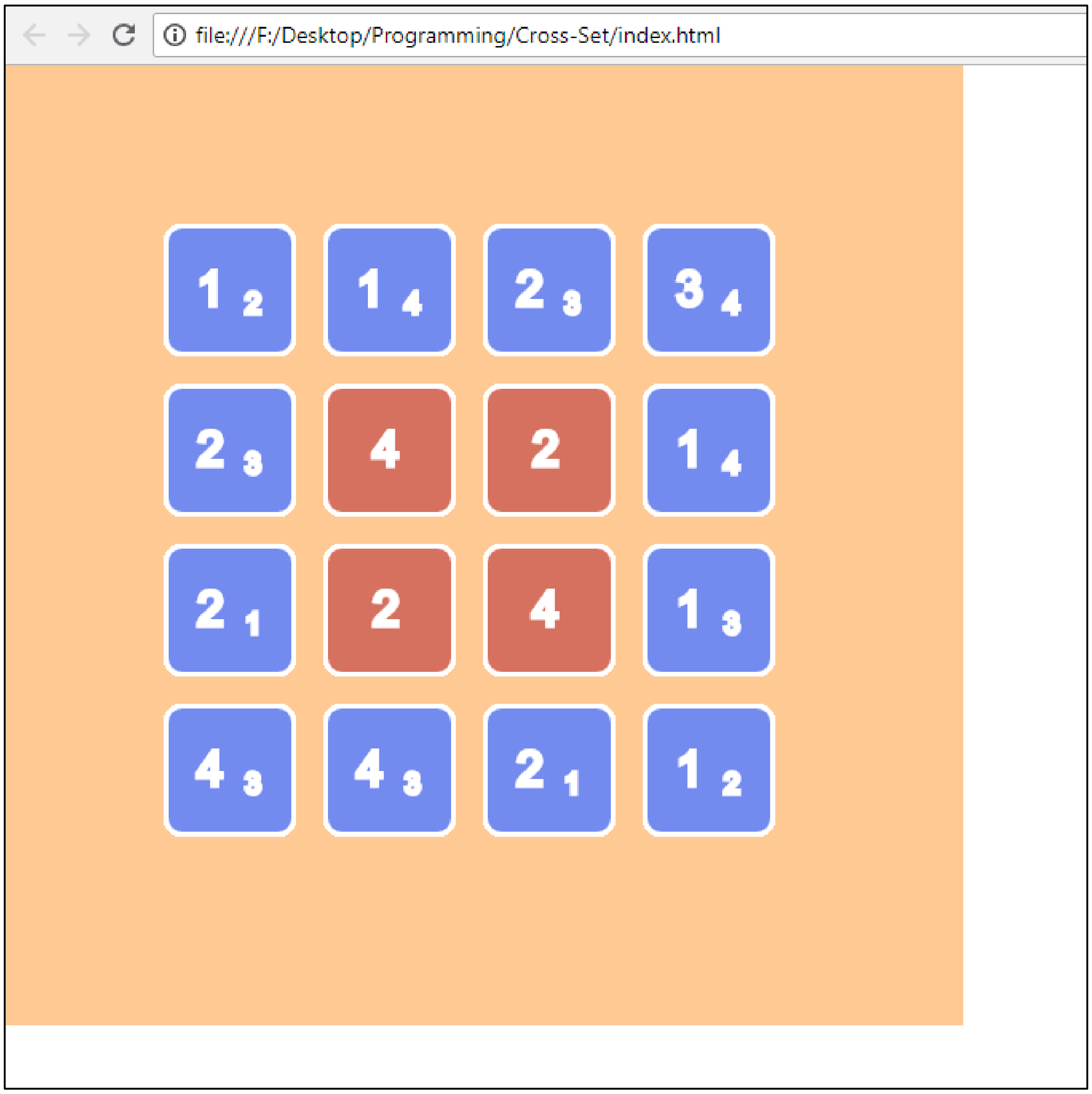
- The **draw function is a while(true)** loop. It repeats **all the time** which allows us to draw each iteration
- We loop through the created field and **call the show function for each square**

Now **run the index.html** and you should see something like this (if you haven't improvised with the field):



4.Game logic

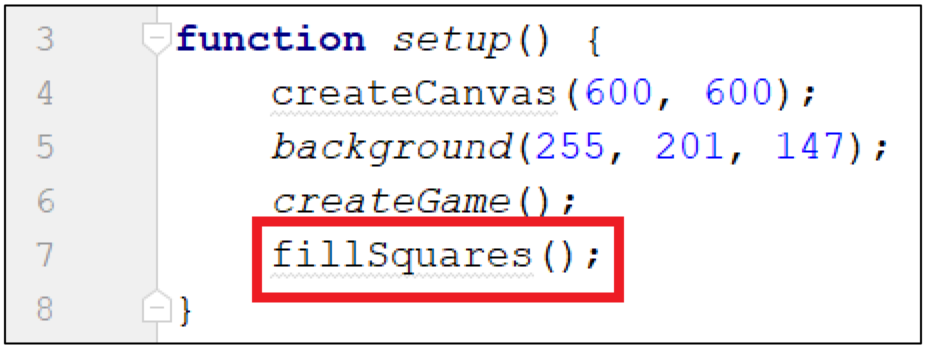
Here is where you can improvise as you want. Create a solution of the game and add secondary values for the clickable squares. I am going to use this:



- I mixed some of the right squares with wrong main values and right secondary values

4.1 Writing the values

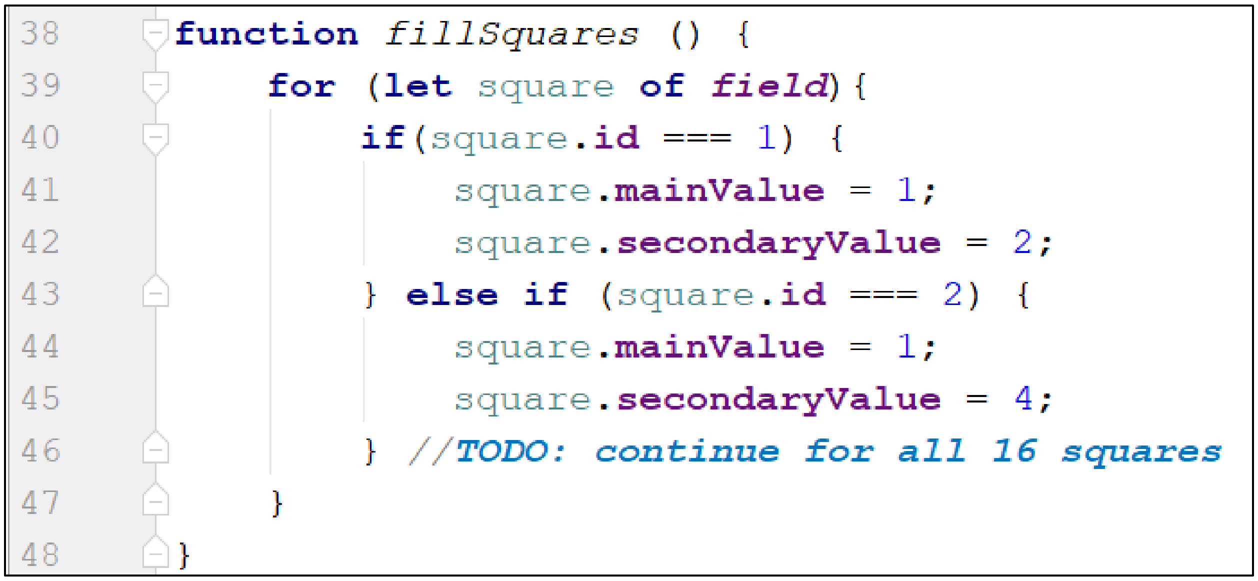
After you have chosen the layout and difficulty of your game let us set the values for each square. Go to the sketch and in the setup function after creating the game write function called fillSquares



```
function setup() {
        createCanvas(600, 600);
        background(255, 201, 147);
        createGame();
        fillSquares();
}
```

- We do that in the setup function, because we only want the **squares to be filled once**.
- The **setup function** is **called only once at startup**

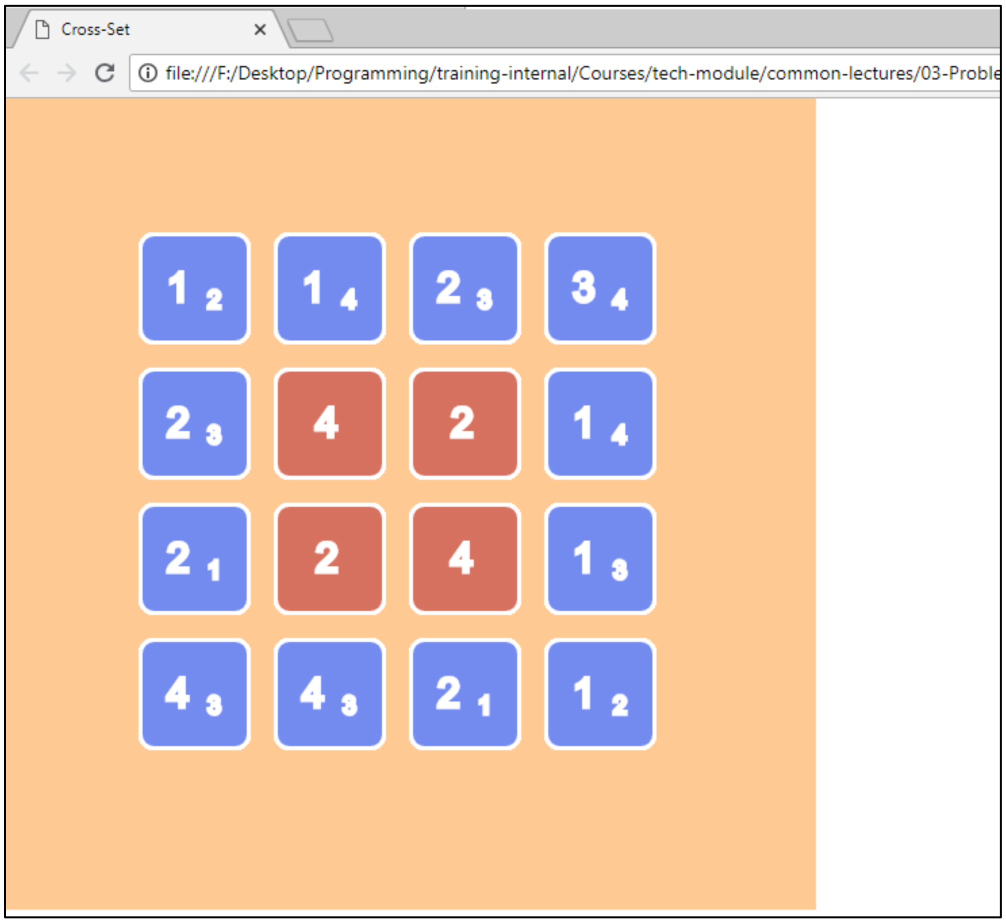
Now at the end add the **fillSquares** function with the following code:



```
function fillSquares () {
    for (let square of field){
        if(square.id === 1) {
            square.mainValue = 1;
            square.secondaryValue = 2;
        } else if (square.id === 2) {
            square.mainValue = 1;
            square.secondaryValue = 4;
        } //TODO: continue for all 16 squares
    }
}
```

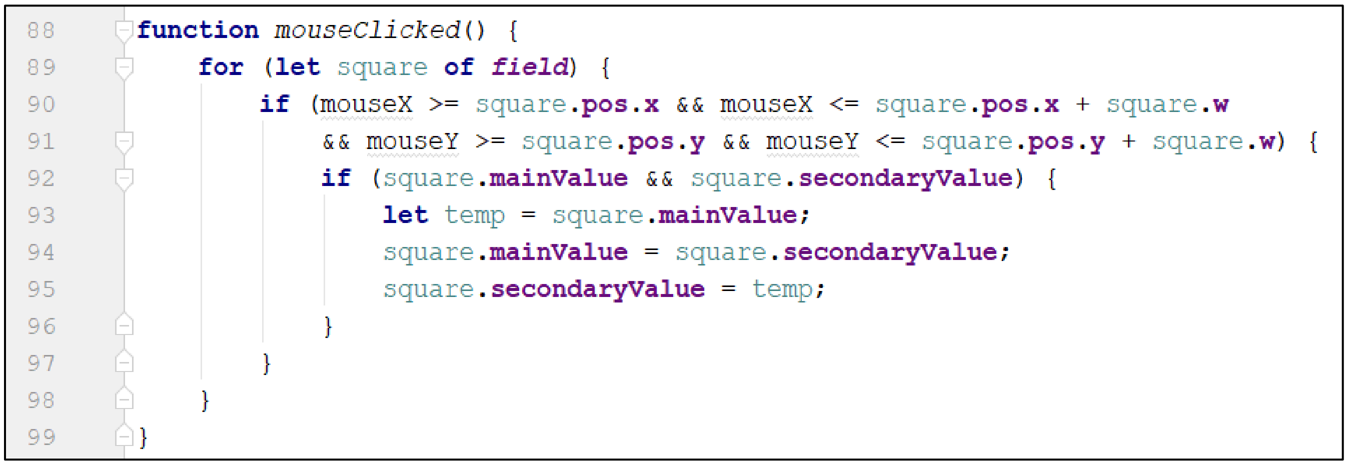
- When filling the values, we check for their **id to know where we are at** and **set a main and secondary value**
- When we arrive at a **NOT clickable square**, we only **set the main value** (it **does not have secondary** value)

Now refresh the browser and you should see your game:


Cross-Set
file:///F:/Desktop/Programming/training-internal/Courses/tech-module/common-lectures/03-Proble
1 2
1 4
2 3
3 4
2 3
4
2
1 4
2 1
2
4
1 3
4 3
4 3
2 1
1 2

4.2 Making them clickable

Here we are going to use the build-in function in p5 called mouseClicked. When the mouse is clicked that function is called. When that happens we take the position of the mouse (mouseX and mouseY) and check for intersection with the squares:



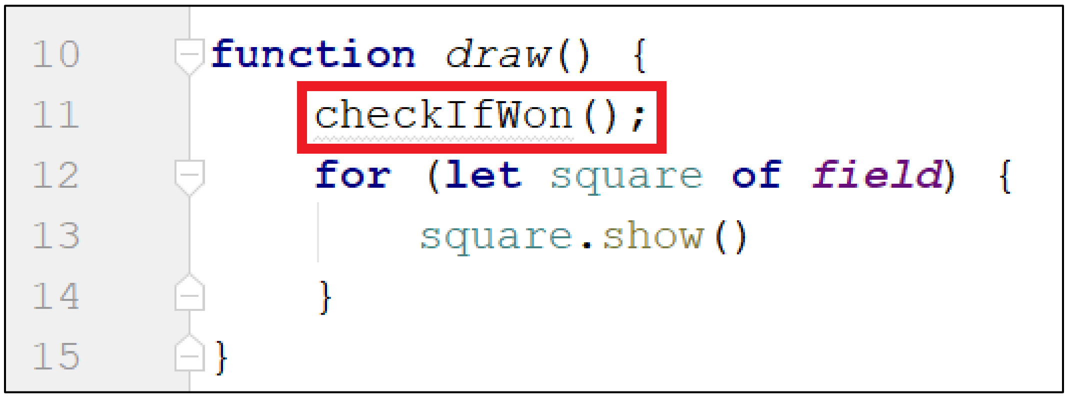
```
function mouseClicked() {
    for (let square of field) {
        if (mouseX >= square.pos.x && mouseX <= square.pos.x + square.w
            && mouseY >= square.pos.y && mouseY <= square.pos.y + square.w) {
            if (square.mainValue && square.secondaryValue) {
                let temp = square.mainValue;
                square.mainValue = square.secondaryValue;
                square.secondaryValue = temp;
            }
        }
    }
}
```

- We loop through the squares and check if the **mouseX** and **mouseY** (the position) **is within the square** we are at.
- If we are, we just use a simple **swapping algorithm** to **switch the main and secondary** value (only **if the square is clickable/it has both main and secondary value**)

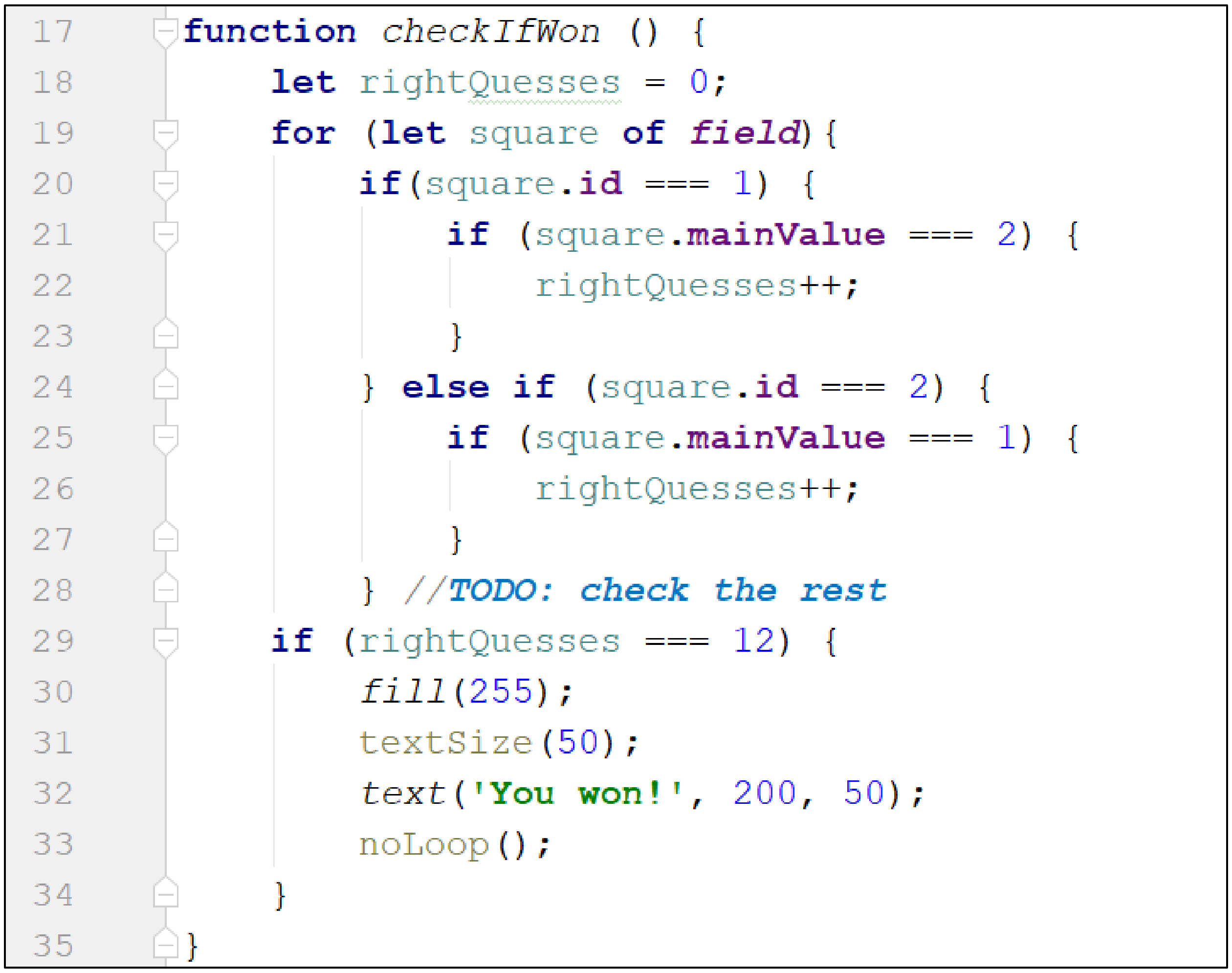
Refresh and now you should be able to play the game!

5. End of game

The final step is to check each frame if the main values are correct (the player wins). Add a function in the draw loop and implement it:



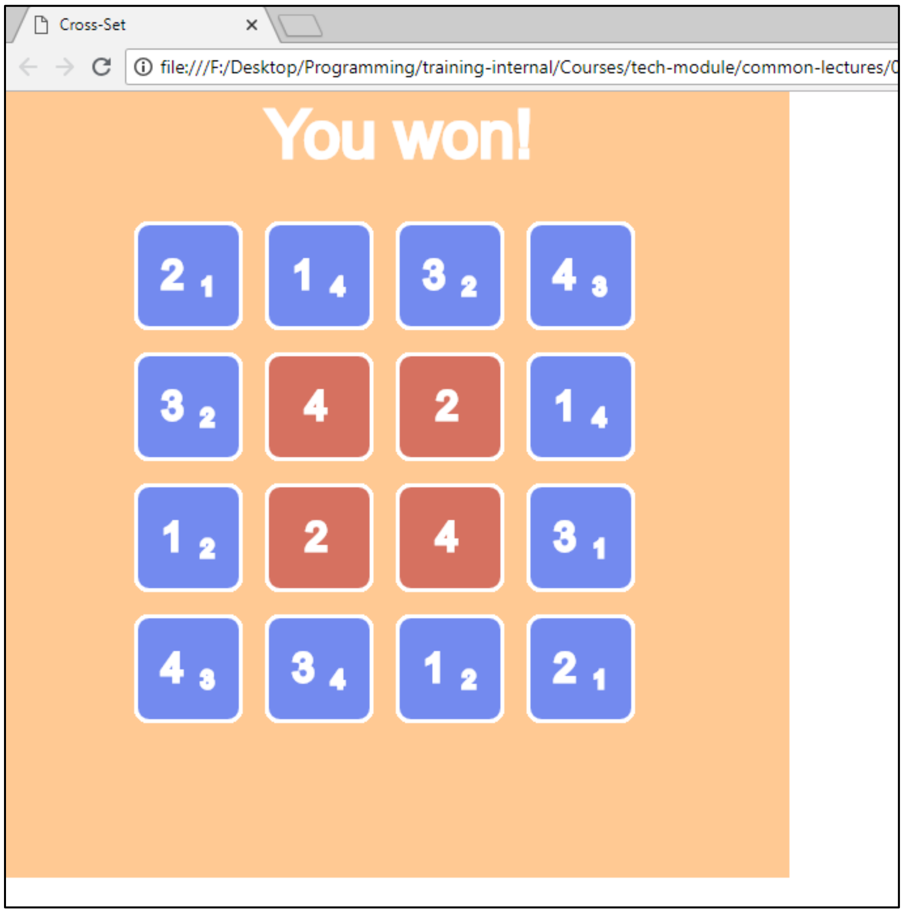
```
function draw() {
    checkIfWon();
    for (let square of field) {
        square.show()
    }
}
```



```
function checkIfWon () {
    let rightQuesses = 0;
    for (let square of field){
        if(square.id === 1) {
            if (square.mainValue === 2) {
                rightQuesses++;
            }
        } else if (square.id === 2) {
            if (square.mainValue === 1) {
                rightQuesses++;
            }
        } //TODO: check the rest
    if (rightQuesses === 12) {
        fill(255);
        textSize(50);
        text('You won!', 200, 50);
        noLoop();
    }
}
```

- Create variable to **store** how many of the **squares are correct** (except from the fixed)
- Each time you have a **right value**, **increase the counter**
- At the end check if the counter is **as high as it can be** (depending on the **size of your field** and the **count of the non-clickable** squares)
- If it is, **write a text** saying that the **player has won** and type **noLoop()**
- **noLoop will stop the draw loop** (end the game)

Congrats! Now you have a logical game.


Cross-Set
file:///F:/Desktop/Programming/training-internal/Courses/tech-module/common-lectures/C
You won!
2 1
1 4
3 2
4 3
3 2
4
2
1 4
1 2
2
4
3 1
4 3
3 4
1 2
2 1

Upgrades

- Try creating more levels of difficulty
- Try creating bigger fields with more complicated solves