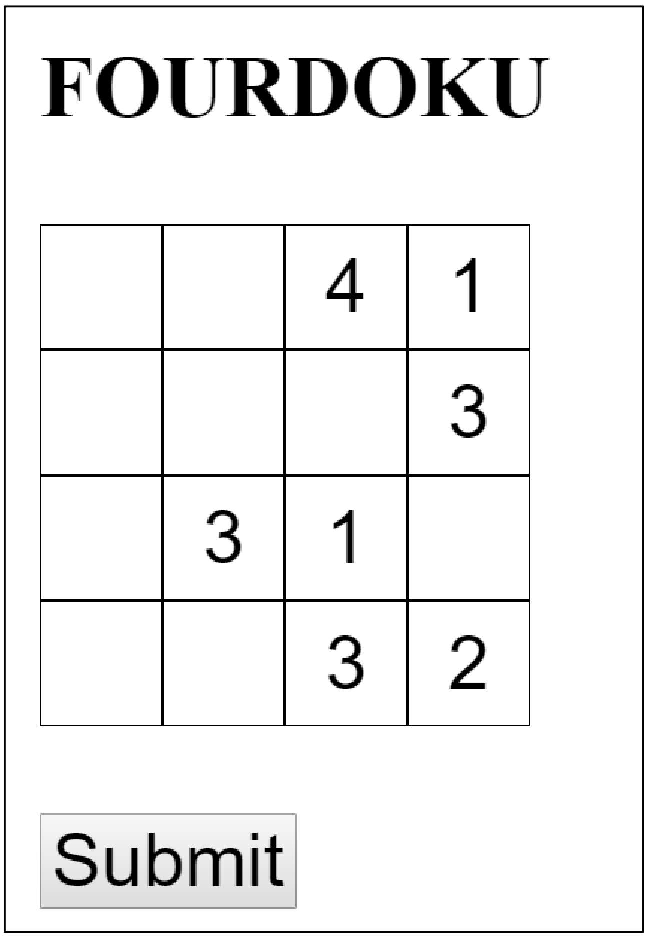
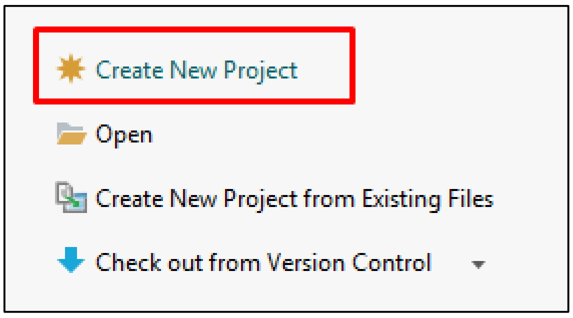
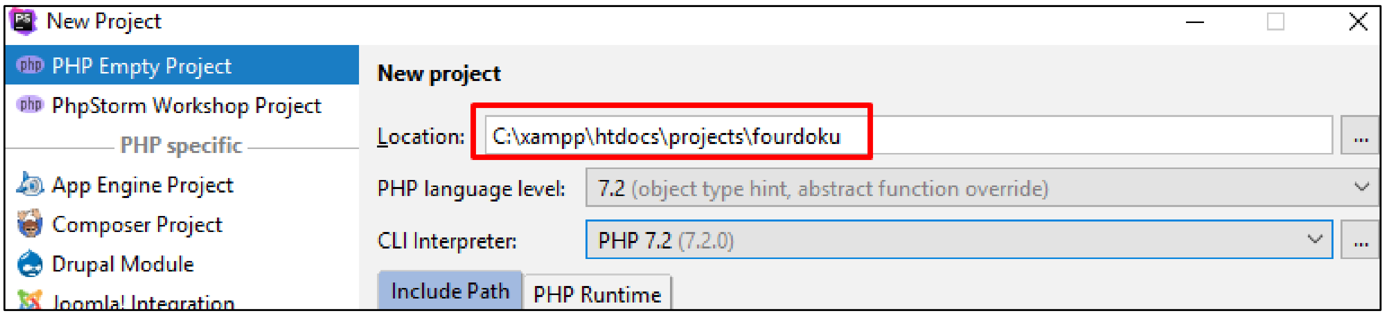
Lab: PHP Fourdoku

This document defines a complete walkthrough of creating a **Fourdoku /4 sudoku/** application. Make sure you have installed XAMPP and added PHP root folder to the path environment variable.



1. Create Project


Create New Project
Open
Create New Project from Existing Files
Check out from Version Control



Location: C:\xampp\htdocs\projects\fourdoku

PHP language level: 7.2 (object type hint, abstract function override)

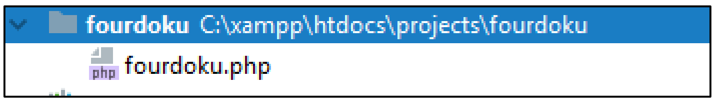
CLI Interpreter: PHP 7.2 (7.2.0)

Create a new project and place it in htdocs folder to run it later via Apache server, installed with XAMPP.

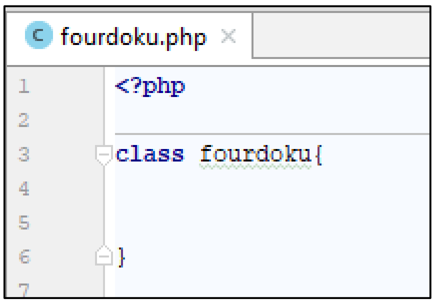
We need to specify the **php executable**, which most probably resides in **c:/xampp/php**

2. Create fourdoku.php

Now, lets create new PHP file, which will contain our fourdoku logic.



In this file we are going to define our class fourdoku:

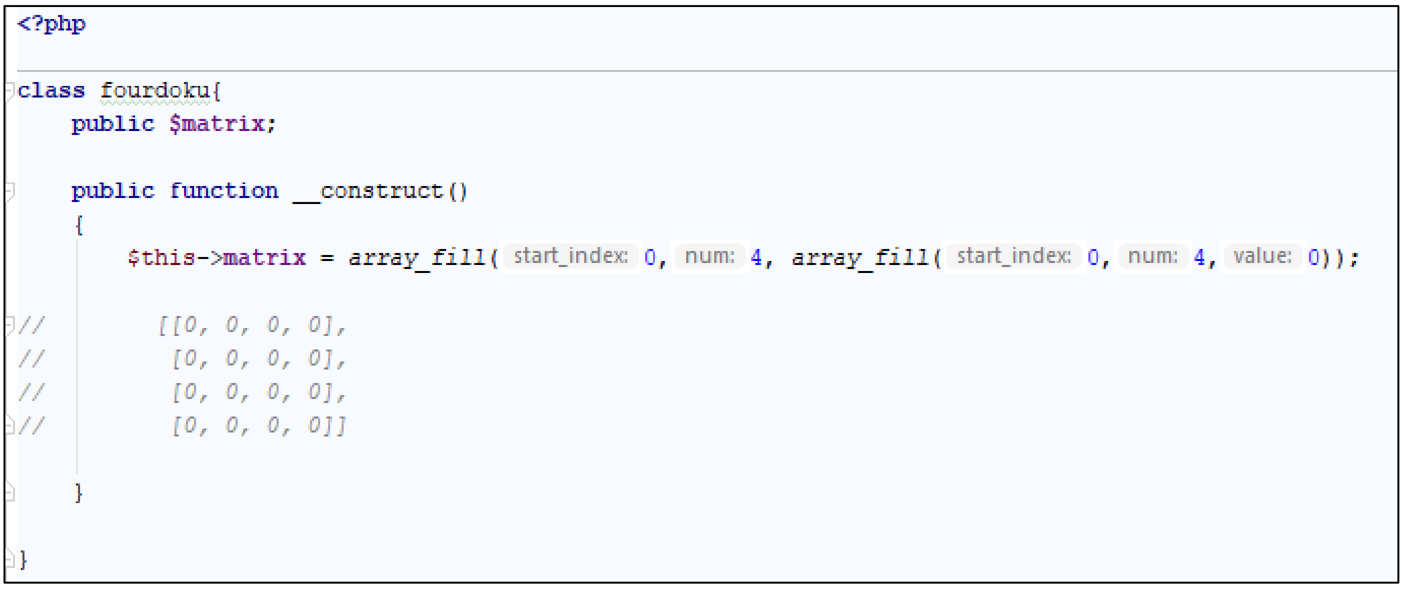


```
<?php

class fourdoku{


}
```

Soon you`ll be able to create more complex classes.

Next, we are going to add our property matrix, this will be an array of arrays, think of it as a table. It will have 4 rows with 4 columns. Here we are going to store the numbers of the fourdoku.

We need to add a __constructor(). This magic function is used when we instantiate the class. Every time we create new class, the constructor is called. So, we are going to fill our matrix with 0 (zeroes).



```
<?php

class fourdoku{
    public $matrix;

    public function __construct()
    {
        $this->matrix = array_fill( start_index: 0, num: 4, array_fill( start_index: 0, num: 4, value: 0));

//          [[0, 0, 0, 0],
//           [0, 0, 0, 0],
//           [0, 0, 0, 0],
//           [0, 0, 0, 0]]

    }

}
```

You can see in the picture how our matrix looks in the beginning.

3. Create index.php

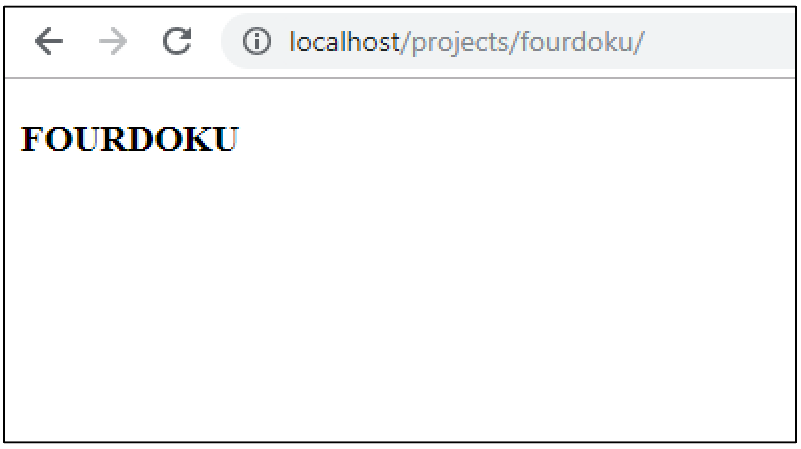
Now let's create our index page, where the fourdoku will be visualized.

Create new php file index.php and place the following code there:

```
<?php
require('./fourdoku.php');
$fourdoku = new fourdoku();
```

```
?>

<!DOCTYPE HTML>
<html>

<head>
    <title>Problem Solving PHP</title>
</head>

<body>
<h3>FOURDOKU</h3>
</body>

</html>
```

Now we can start XAMPP, start our apache server and navigate to the page to see what it looks like.



FOURDOKU

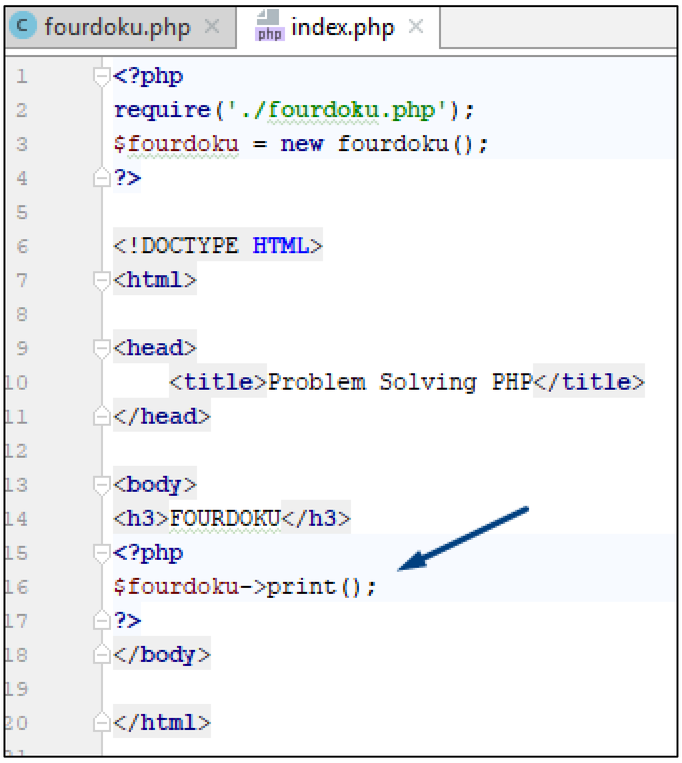
It looks kind of empty, but why, we are creating new fourdoku() in our code. Yes, but we are not printing anything, there is a matrix, but it isn`t visualized. So lets create a function print() which is going to correct this mistake.

4. Function print();

Add another public function in our class. We are going to loop trough the rows of the matrix and print each column value:

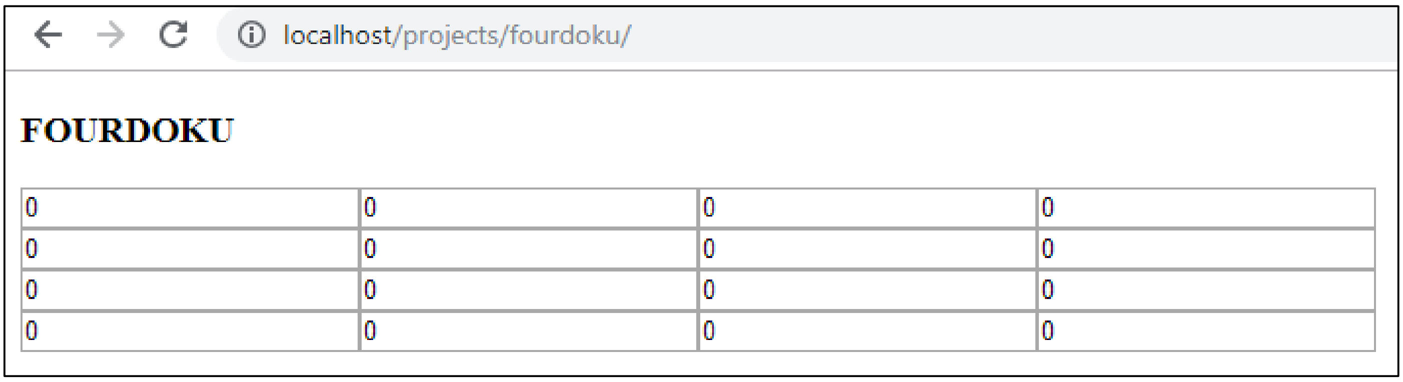
```
<?php

class fourdoku{
    public $matrix;

    public function __construct()
    {
        $this->matrix = array_fill( start_index: 0, num: 4, array_fill( start_index: 0, num: 4, value: 0));
    }

    public function print(){

        for ($row = 0; $row < 4; $row++) {
            for ($column = 0; $column < 4; $column++) {
                    echo '<input name="' . $row . $column . '"value="' . $this->matrix[$row][$column] . '">';
            }

            echo '<br>';
        }

    }
}
```

Add in the body of our index.php another php block and call the function.



```
<?php
require('./fourdoku.php');
$fourdoku = new fourdoku();
?>

<!DOCTYPE HTML>
<html>

<head>
    <title>Problem Solving PHP</title>
</head>

<body>
<h3>FOURDOKU</h3>
<?php
$fourdoku->print();
?>
</body>

</html>
```

Let`s see what it looks like:



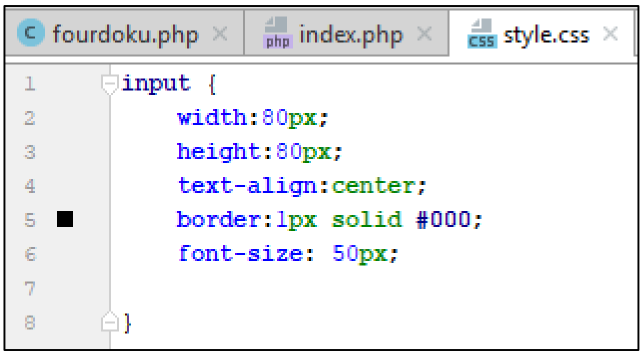
FOURDOKU

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

5. Add Style

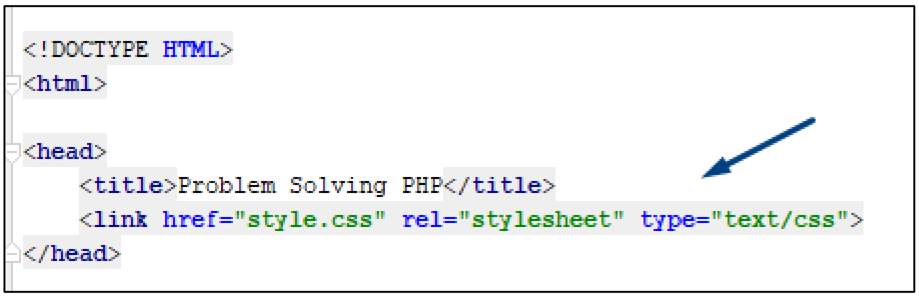
Almost right, but let's add some CSS to the picture and see what happens.

Create a style.css file in the directory of the project and add the following code:



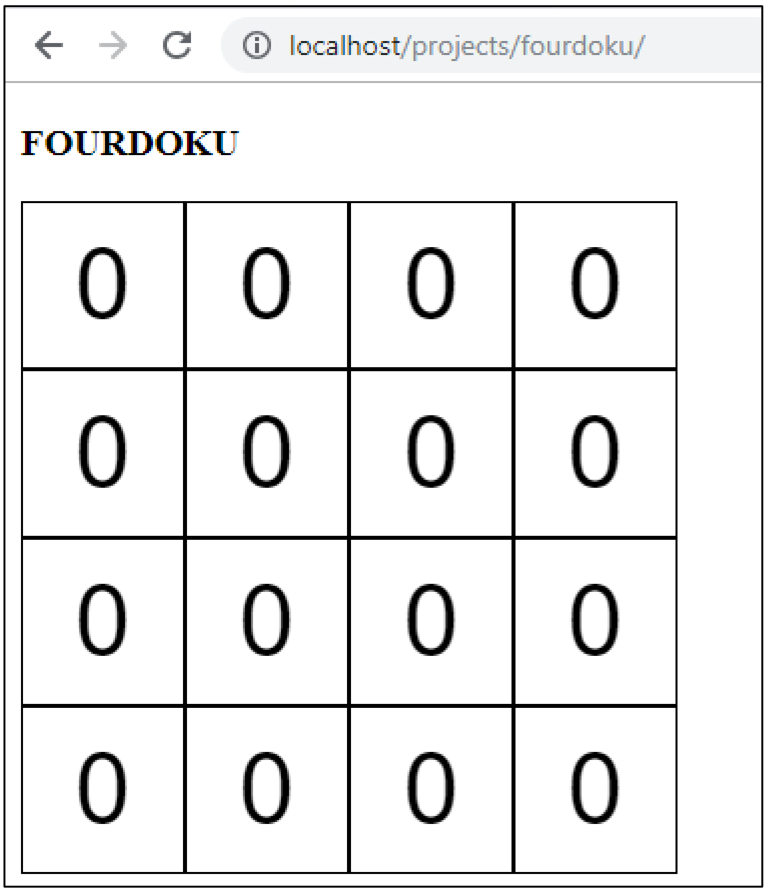
```
input {
    width:80px;
    height:80px;
    text-align:center;
    border:1px solid #000;
    font-size: 50px;

}
```

Now link the file in the html document in index.php in the head section under the title:



```
<!DOCTYPE HTML>
<html>

<head>
    <title>Problem Solving PHP</title>
    <link href="style.css" rel="stylesheet" type="text/css">
</head>
```

Let's check what have we done:



localhost/projects/fourdoku/

FOURDOKU

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Much better, now it looks like fourdoku. Later we are going to add much more html and css styling, but be patient.

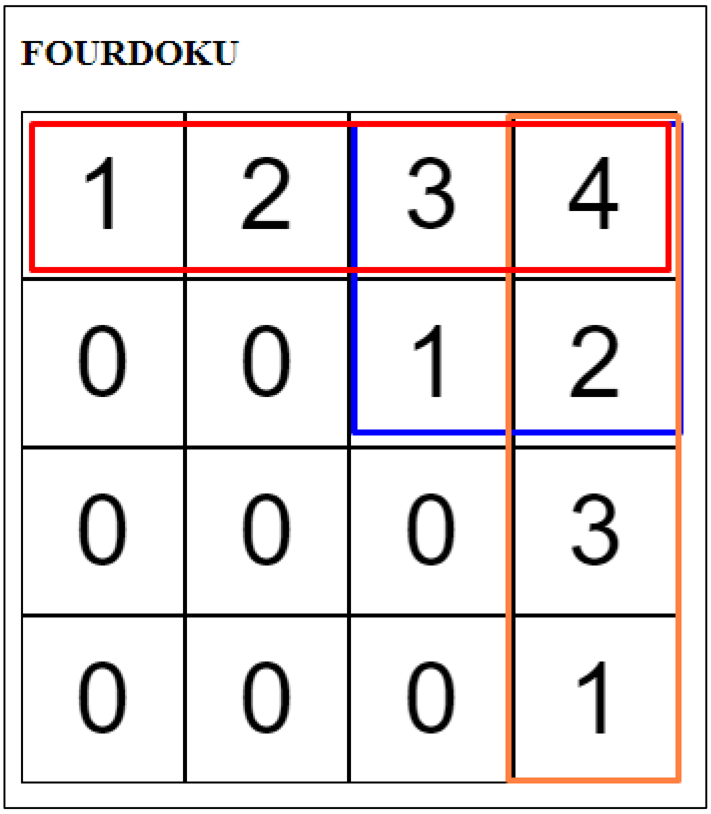
It`s time to start creating our logic for generating fourdoku.

6. generate();

Let`s go back in foudoku.php and create a generate() function:

```
public function generate(){

}
```

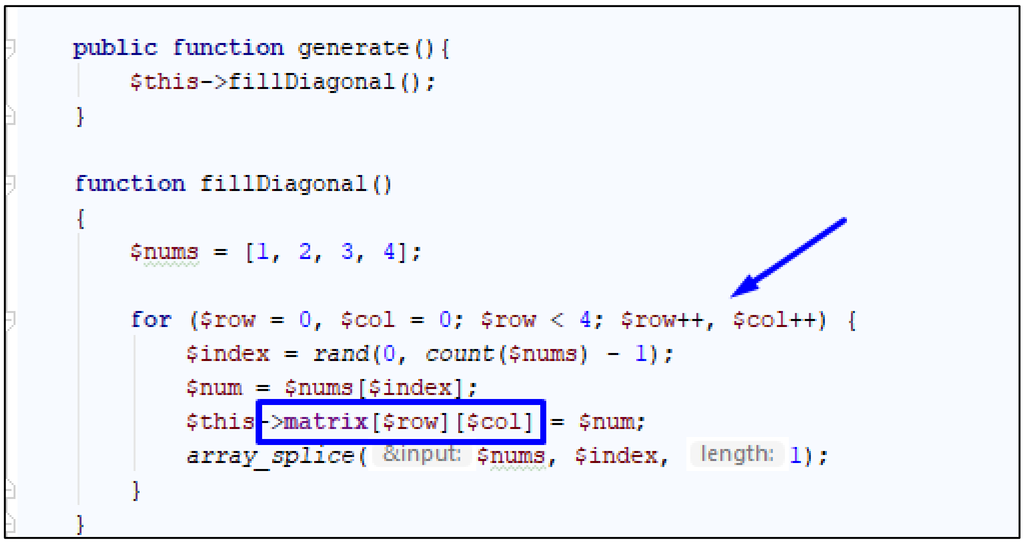
So how our fourdoku will be correct? It can't be random, each row, column and square should have the digits from 1 to 4 only once, as shown in the picture below;



FOURDOKU

1	2	3	4
0	0	1	2
0	0	0	3
0	0	0	1

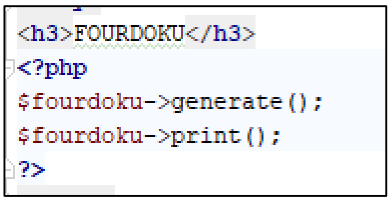
So, to properly create a fourdoku, first we need to fill the diagonal. We are going to fill it randomly. Let`s create another function – fillDiagonal() and call it in our generate() function. In this function we are going to add random number from 1 to 4 in our diagonal.



```
public function generate(){
    $this->fillDiagonal();
}

function fillDiagonal()
{
    $nums = [1, 2, 3, 4];

    for ($row = 0, $col = 0; $row < 4; $row++, $col++) {
        $index = rand(0, count($nums) - 1);
        $num = $nums[$index];
        $this->matrix[$row][$col] = $num;
        array_splice( &input: $nums, $index,  length: 1);
    }
}
```

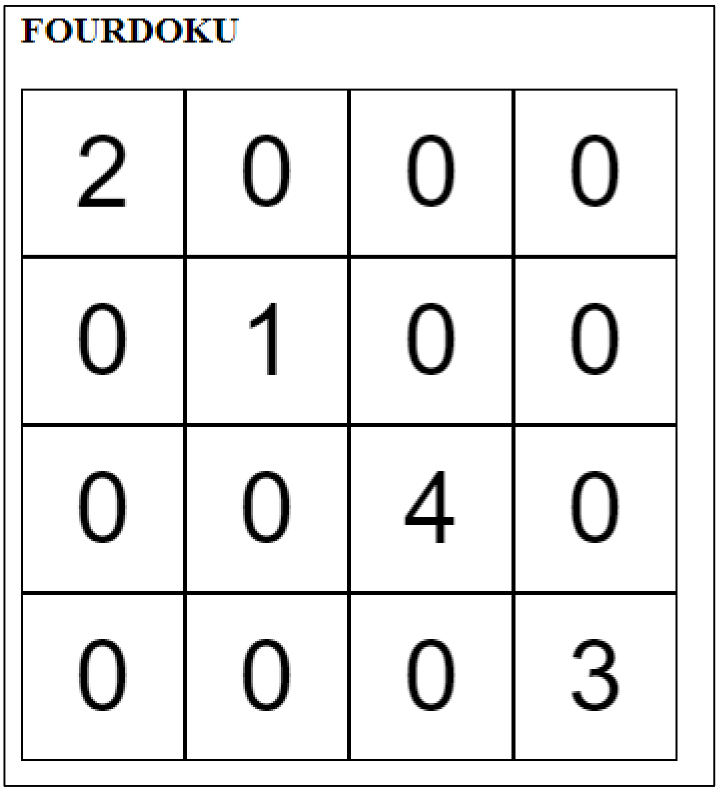
We use rand() to take random number from our array with numbers and place it on 0-0 position in the matrix(table). When we assign the random digit, we remove it from the numbers array and proceed to the next position in the diagonal(1-1) until we fill it.

Don’t forget to call the function in index.php:



```
<h3>FOURDOKU</h3>
<?php
$fourdoku->generate();
$fourdoku->print();
?>
```

Let's see if the code is right and run the index.php:



FOURDOKU

2	0	0	0
0	1	0	0
0	0	4	0
0	0	0	3

Looks right! Every time we refresh the page, the numbers in the diagonal will be different and there are no rules broken.

So now what we do?

We need to add "Candidates" in each cell. We start iterating through each row checking each column that contains 0 and add there the numbers that can be placed.

Let's take a look at position row 0, column 1 – we can put there 3 or 4, so the candidates for this position are 3 and 4.

But wait a moment, if we put candidates now, each position can have not less that 2 digits, so it isn`t right.

That’s why we are going to place one of the two possible candidates right away in position row - 0 col - 1.

We can do this either in the same function, but I prefer to export it in another.

Let's call it placeOne():

```
function placeOne()
{
    $nums = [1, 2, 3, 4];
    while (true) {
        $index = rand(0, count($nums) - 1);
        $num = $nums[$index];

        if ($this->checkVertical($num, col: 1) && $this->checkHorizontal($num, row: 0)) {
            $this->matrix[0][1] = $num;
            break;
        } else {
            array_splice( &input: $nums, $index, length: 1);
        }

    }

}
```

We check for each number if we can place it on the position. We are using two different functions – checkHorizontal() and checkVertical(). Both functions receive the number they are checking and the column or the row to check. Let`s implement them!

```
function checkHorizontal($num, $row)
{
    for ($c = 0; $c < 4; $c++) {
        if ($this->matrix[$row][$c] == $num) {
            return false;
        }

    }

    return true;
}
```

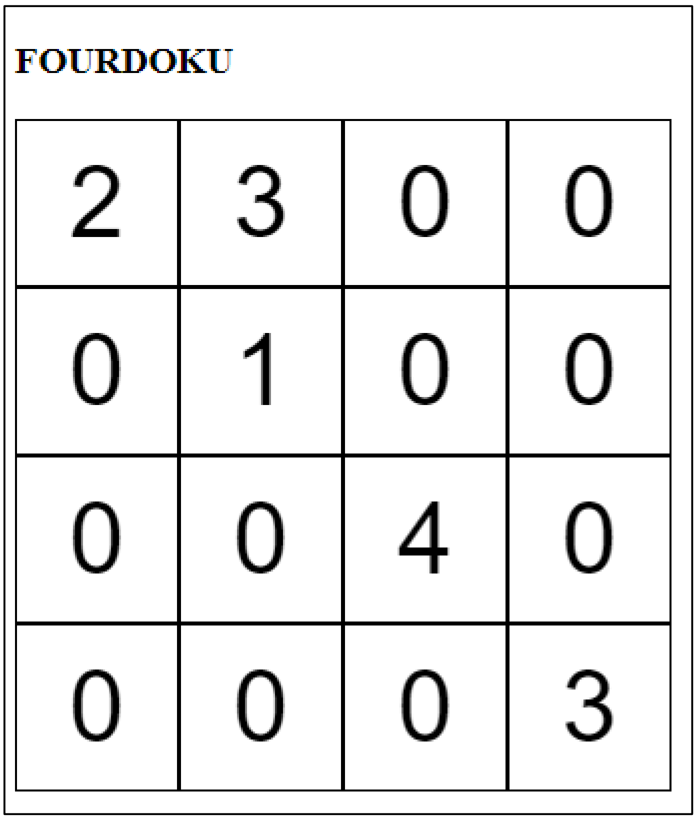
checkHorizontal() iterates through the columns in our matrix (table) and checks whether the row contains such number. If there isn`t such number in the current row, the function returns true.

Try implementing the other function by yourself. The logic is the same but you should iterate through the rows to check vertical.

Now let's call the placeOne() function in the generate function, right after we fillDiagonal();

```
public function generate(){
    $this->fillDiagonal();
    $this->placeOne();
}
```

Check the result. We are placing the digit on position 0, 1 and it looks right.


FOURDOKU

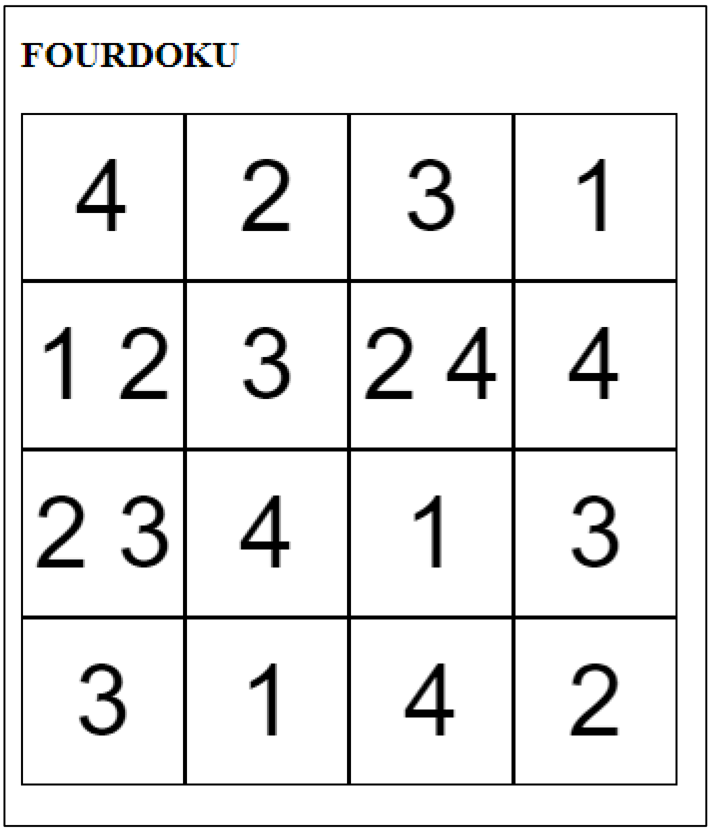
2	3	0	0
0	1	0	0
0	0	4	0
0	0	0	3

7. Place Candidates

Now we can place the candidates in the other cells. Create the function placeCandidates() and implement the logic

```
function placeCandidates(){
    $nums = [1, 2, 3, 4];

    for ($row = 0; $row < 4; $row++) {
        for ($col = 0; $col < 4; $col++) {
            if ($this->matrix[$row][$col] === 0) {
                $this->matrix[$row][$col] = ' ';

                foreach ($nums as $num) {

                    if ($this->checkVertical($num, $col) && $this->checkHorizontal($num, $row)) {
                        $this->matrix[$row][$col] .= ' ' . $num;
                        $this->matrix[$row][$col] = trim($this->matrix[$row][$col]);
                    }

                }
            }
        }
    }
}
```

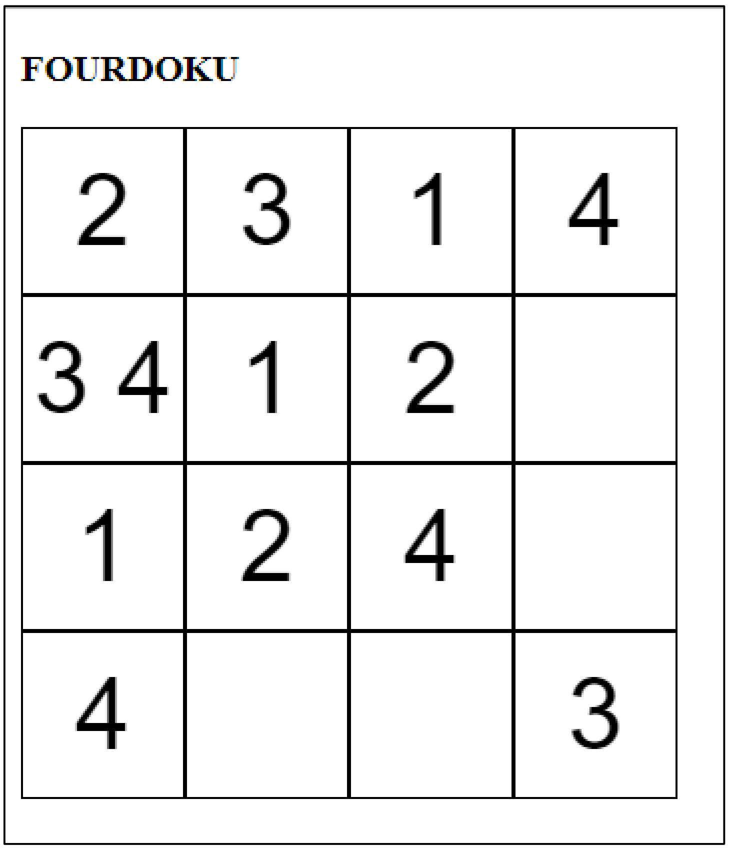
Let`s see, we go through each cell and check if it`s 0, if it is we start checking each number, if we can place it there. If so, we concatenate it. Lets call the function after placeOne() and check the result:



FOURDOKU

4	2	3	1
1 2	3	2 4	4
2 3	4	1	3
3	1	4	2

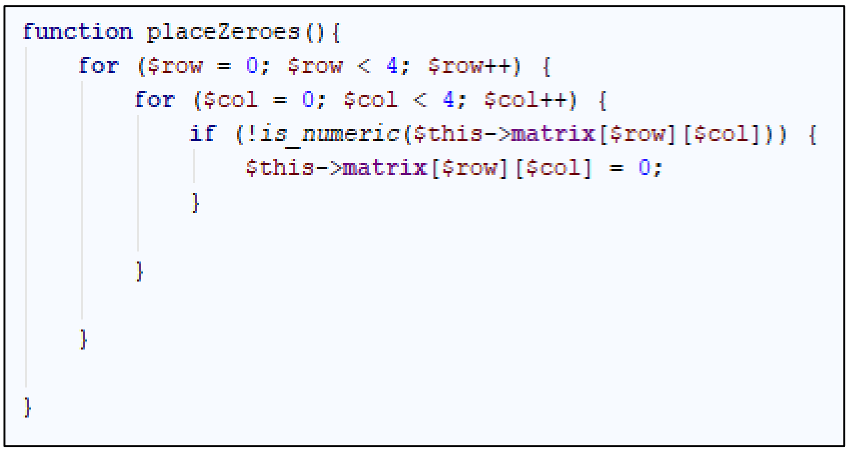
Looks nice, but let's refresh few times and see if we can find a bug.



FOURDOKU

2	3	1	4
3 4	1	2	
1	2	4	
4			3

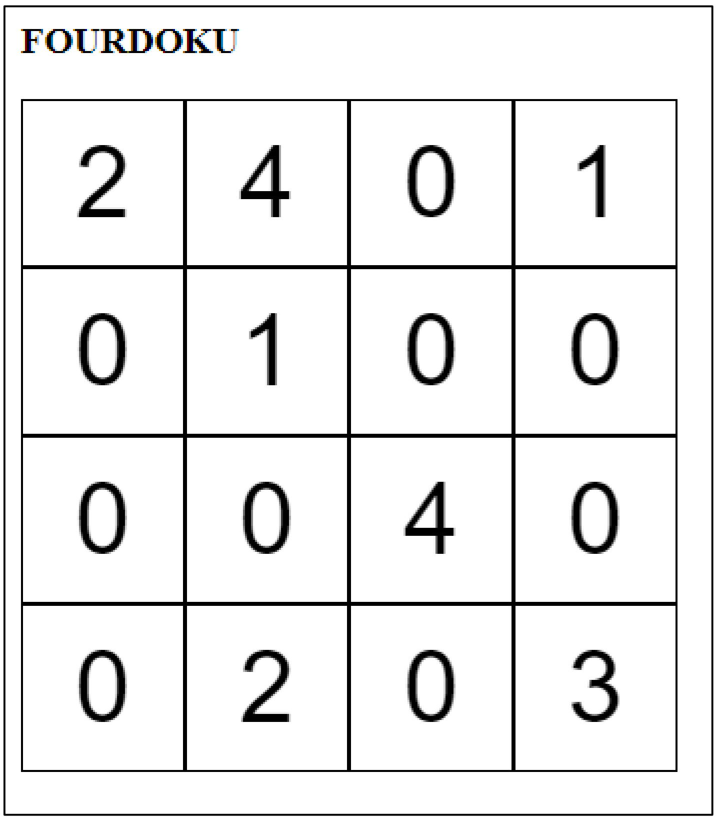
Let`s try understanding better what is happening. Let's create method placeZeroes() which function goes and replace each cell that contains more or less than 1 number with 0.



```
function placeZeroes(){
    for ($row = 0; $row < 4; $row++) {
        for ($col = 0; $col < 4; $col++) {
            if (!is_numeric($this->matrix[$row][$col])) {
                $this->matrix[$row][$col] = 0;
            }

        }

    }

}
```

Let`s call the function right after placeCandidates().

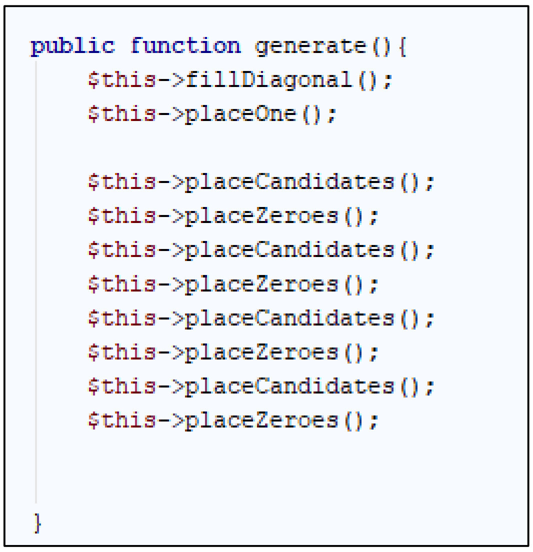
So now when the number in the cell isn`t valid number it is changed to 0.



FOURDOKU

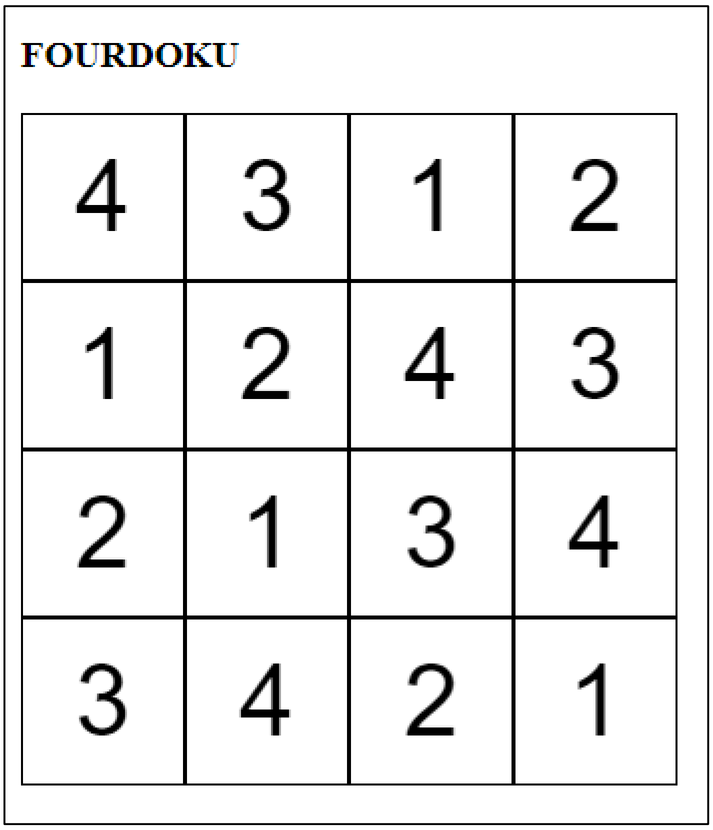
2	4	0	1
0	1	0	0
0	0	4	0
0	2	0	3

Ok, so we are fine, because we managed to place some numbers. Let's try and repeat the process one more time. Just call the functions several times.



```
public function generate(){
    $this->fillDiagonal();
    $this->placeOne();

    $this->placeCandidates();
    $this->placeZeroes();
    $this->placeCandidates();
    $this->placeZeroes();
    $this->placeCandidates();
    $this->placeZeroes();
    $this->placeCandidates();
    $this->placeZeroes();


}
```

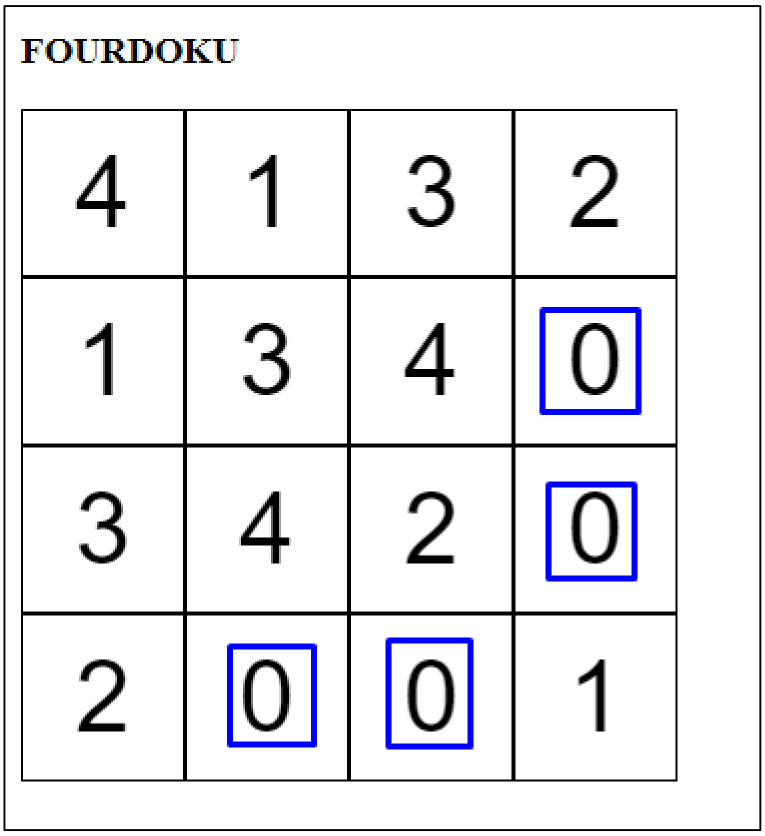
Yes, sometimes we have a solved fourdoku:



FOURDOKU

4	3	1	2
1	2	4	3
2	1	3	4
3	4	2	1

But sometimes there is such result:

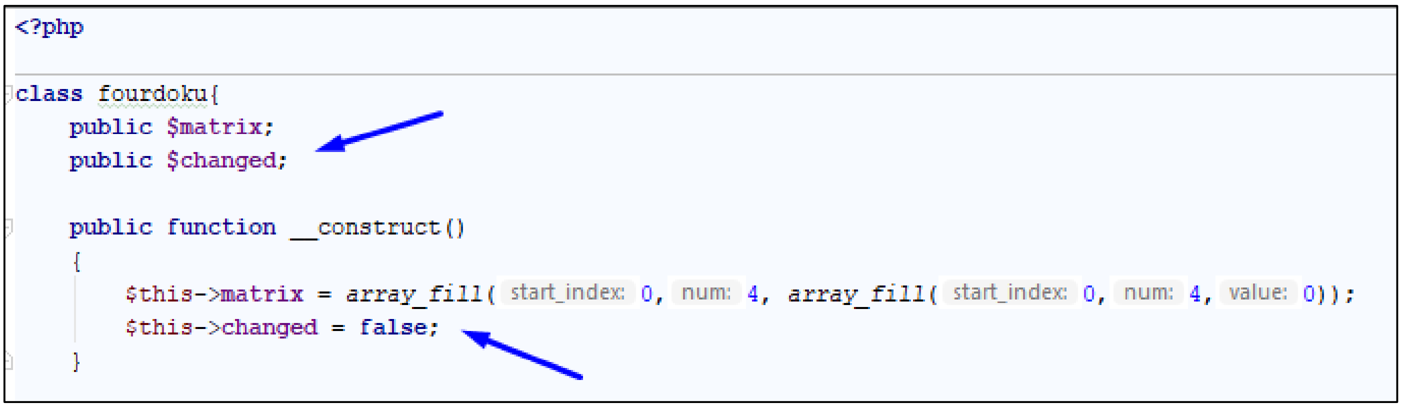


4	1	3	2
1	3	4	0
3	4	2	0
2	0	0	1

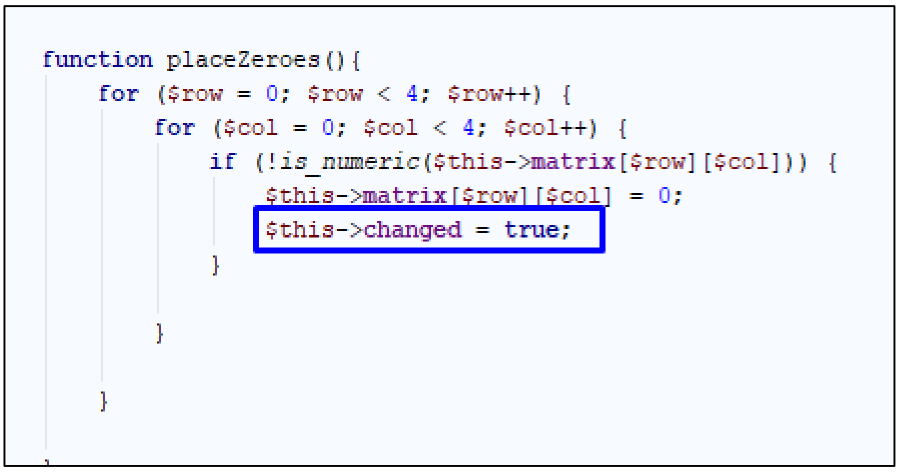
This case is special, because we ended in a situation where there is no solution. What should we do now?

The better way is to learn what backtracking is, simply like going back and trying the last step differently. But it is kind of advanced matter, so we are going to take a more newbie approach. We are going to reset our solution if it needed more than 10 tries to solve the fourdoku.

First, we are going to need a flag if there were some changes made. Let`s add another property to our class. Call it $changed, which will be a Boolean value. Also add it in the constructor and set it to false.

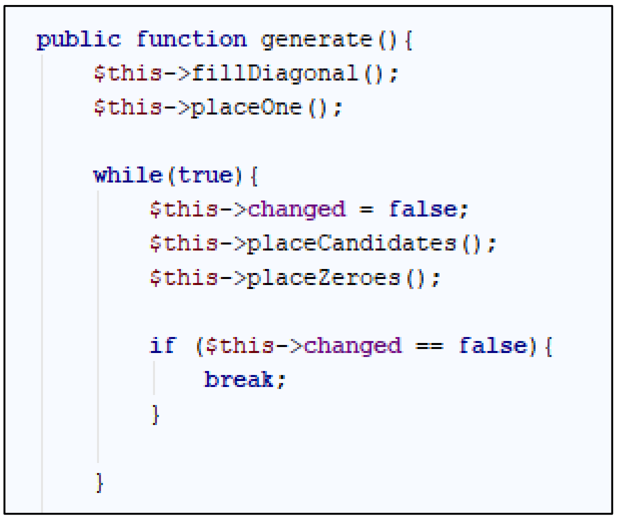

```php
<?php

class fourdoku{
    public $matrix;
    public $changed;

    public function __construct()
    {
        $this->matrix = array_fill( start_index: 0, num: 4, array_fill( start_index: 0, num: 4, value: 0));
        $this->changed = false;
    }
```

Now we need to change the value to true, if we made any changes to our matrix(table). There are two cases when we change something – when we put 0-es in place of the candidates and when we place candidates. Let's add the flag in our logic:



```
function placeZeroes(){
    for ($row = 0; $row < 4; $row++) {
        for ($col = 0; $col < 4; $col++) {
            if (!is_numeric($this->matrix[$row][$col])) {
                $this->matrix[$row][$col] = 0;
                $this->changed = true;
            }

        }

    }
```

Now lets add a while loop in generate() function where we repeat placeCandidates() and placeZeroes(). Let's break the loop if the flag $changed equals false. But don`t forget to set it to false in the beginning of each iteration:

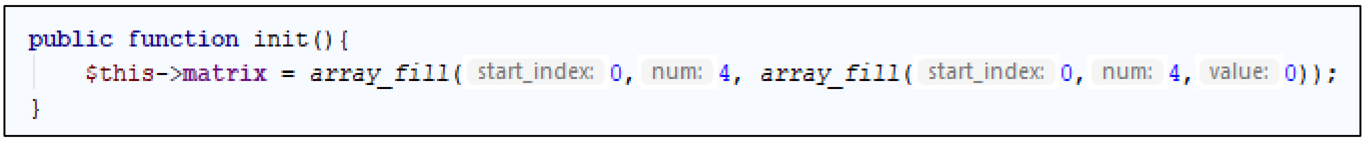


```
public function generate(){
    $this->fillDiagonal();
    $this->placeOne();

    while(true){
        $this->changed = false;
        $this->placeCandidates();
        $this->placeZeroes();

        if ($this->changed == false){
            break;
        }

    }
```

Now, if we refresh the index.php sometimes we are going to fall into infinite loop, sometimes there will be a fourdoku generated for us.

Now we can go newb style and just restart the whole process when we made more than 10 tries to solve the current puzzle.

We can just reuse filling the matrix with 0-es into a function, let`s call in init():



```
public function init(){
    $this->matrix = array_fill( start_index: 0, num: 4, array_fill( start_index: 0, num: 4, value: 0));
}
```

Add a counter of the tries needed to solve the puzzle into our while loop and check if it exceeds 10. If so, call init() and repeat the process. You can add fillDiagonal() and placeOne() functions into init() for convenience() ;P.

```
while(true){
    $this->changed = false;
    $this->placeCandidates();
    $this->placeZeroes();

    if ($this->changed == false){
        break;
    }
    $tries++;

    if ($tries > 5){
        $this->init();
        $this->fillDiagonal();
        $this->placeOne();
        $tries= 0;
    }

}
```

Now, each time, we`ll have a solved fourdoku.

8. Make it preety

Now it`s time to add the game element to the application.

Let`s edit the index.php and add replace it with the following code:

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Problem Solving PHP</title>
    <link href="style.css" rel="stylesheet" type="text/css">
</head>
<body>
<div class="sudoku">
    <div class="inline">
        <h3>FOURDOKU</h3>
        <?php
        require('./fourdoku.php');
        $fourdoku = new fourdoku();

        if (!empty($_POST)) {
            echo "<form>";
            $fourdoku->checkSolution();
            echo "<br>";
            echo '<input type="submit" value="Reset">';
            echo '</form>';



        } else {
            echo "<form method=\"post\">";
            $fourdoku = new fourdoku();
            $fourdoku->generate();
            $fourdoku->print();
            echo "<br>";
            echo "<input type=\"submit\">";
            echo "</form>";
        }
        ?>
    </div>
</div>
</body>
```

```
</html>
<?php
```

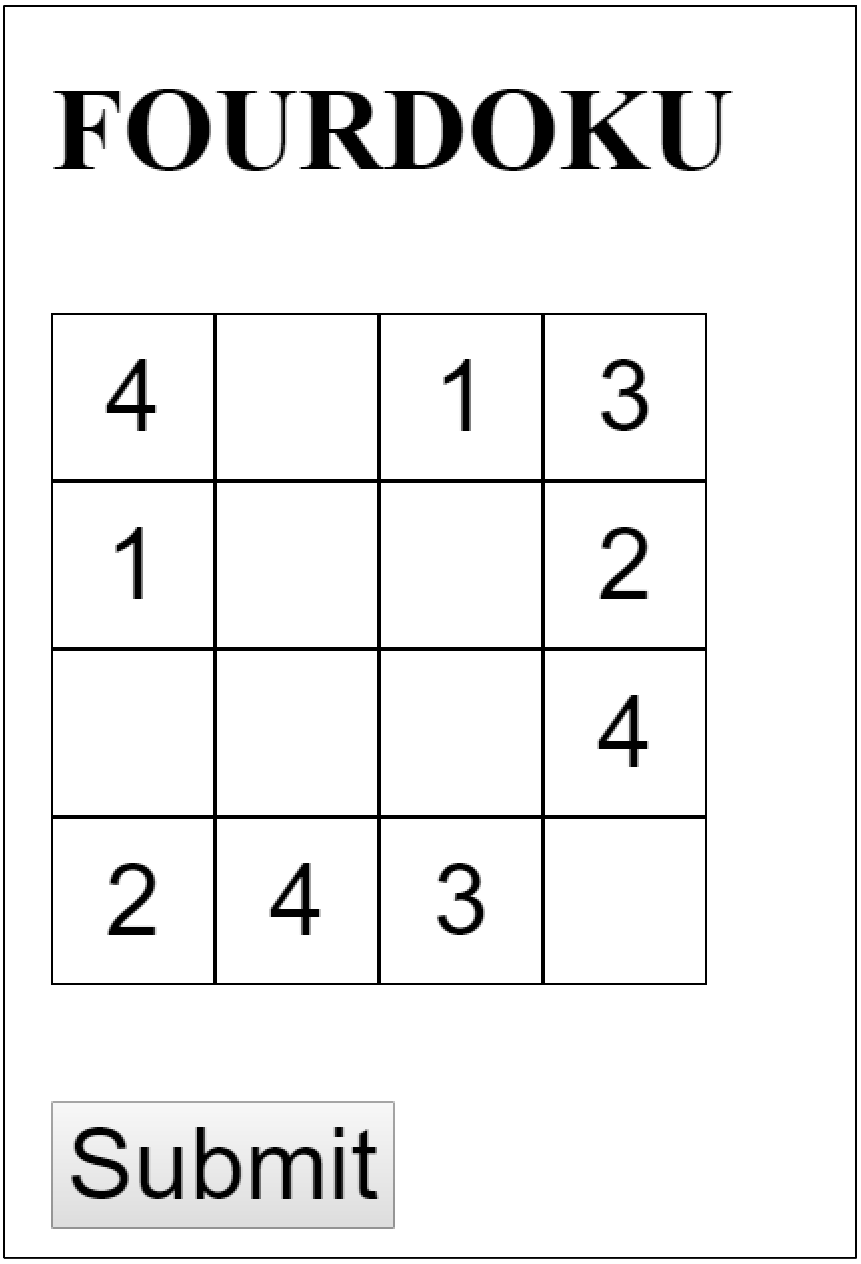
Also, add this code in our css file:

```
.sudoku {
    margin: 0 auto;
    margin-top: 5px;
    font-size: 50px;
}

.inline {
    display: inline-block;
    padding: 15px;
}

.field {
    width:80px;
    height:80px;
    text-align:center;
    border:1px solid #000;
    font-size: 50px;

}
input{
    font-size: 50px;
}

.wrong{
    background: #f5afa9;
    color: crimson;
}

input::-webkit-outer-spin-button,
input::-webkit-inner-spin-button {
    /* display: none; <- Crashes Chrome on hover */
    -webkit-appearance: none;
    margin: 0; /* <-- Apparently some margin are still there even though it's hidden */
}
```

Next we need to edit the print() function:

```
public function print(){
    for ($row = 0; $row < 4; $row++)  {
        for ($column = 0; $column < 4; $column++)  {
            if (rand(0, 1) == 1)  {
                echo '<input type="number" class="field" min="1" max="4" name="' . $row .
$column . '"value="">';
            } else {
                echo '<input type="number" readonly="readonly" class="field" min="1" max="4"
name="' . $row . $column . '"value="' . $this->matrix[$row][$column] . '">';
            }

        }

        echo '<br>';
    }

}
```

This way we will skip some of the cells and the game is on my friend.

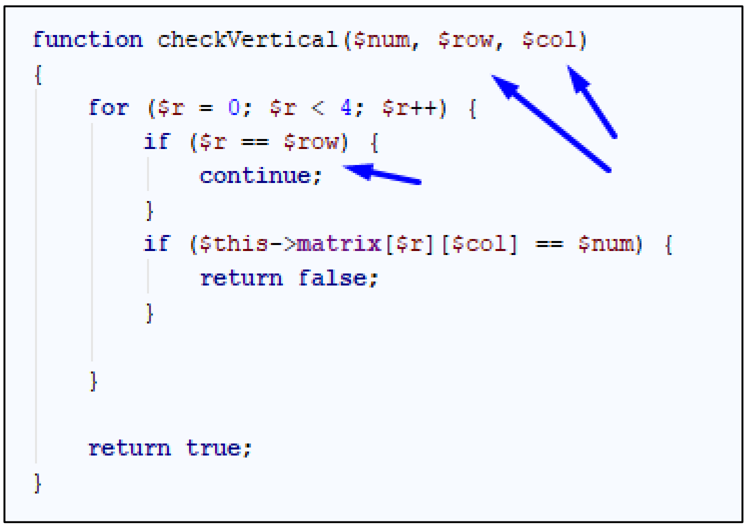


Now we need to implement the solving part. When someone submits a solution, we need to display if its correct, else, to show the wrong cells.

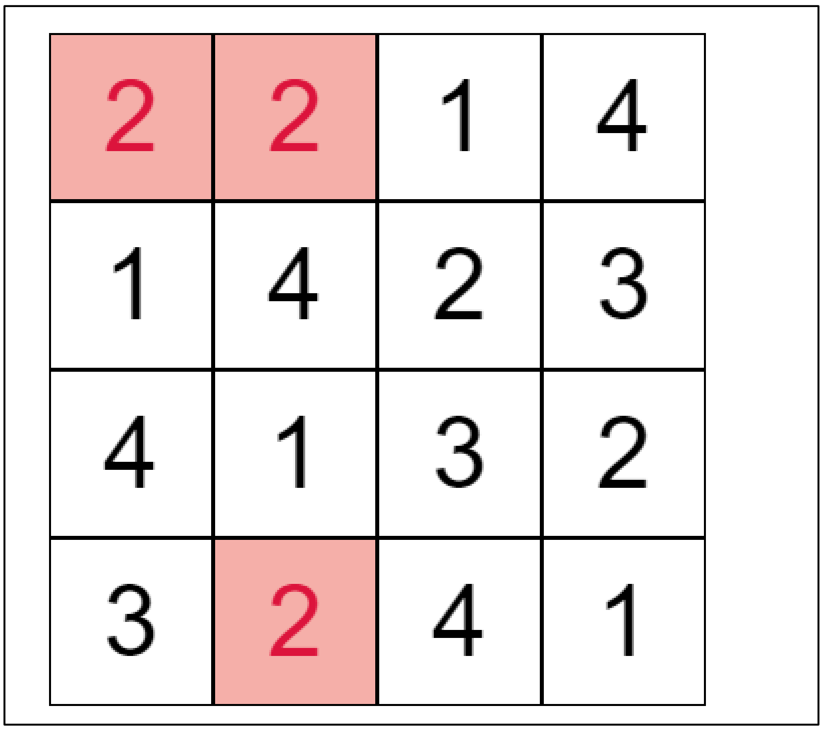
I am giving you the function right away, try understanding what is done here.

```
function checkSolution()
{
    $data = array_values($_POST);
    $i = 0;
    for ($row = 0; $row < 4; $row++) {
        for ($column = 0; $column < 4; $column++) {
            $this->matrix[$row][$column] = $data[$i++];
        }
    }

    for ($row = 0; $row < 4; $row++) {
        for ($column = 0; $column < 4; $column++) {
            $num = $this->matrix[$row][$column];
            if(!$this->checkHorizontal($num,$row, $column)|| !$this->checkVertical($num, $row,
$column)|| $num == 0 || $num == ''){
                $class= 'wrong';
            }
            else{
                    $class = 'right';
                }

            echo '<input class="field '.$class.'" value="' . $num . '">';
        }

        echo '<br>';
    }

}
```

To work properly, you`ll need to add one condition in checkVertical() and checkHorizontal() functions. Try figuring it out. I am showing you only one:



```
function checkVertical($num, $row, $col)
{
    for ($r = 0; $r < 4; $r++) {
        if ($r == $row) {
            continue;
        }
        if ($this->matrix[$r][$col] == $num) {
            return false;
        }

    }

    return true;
}
```



2	2	1	4
1	4	2	3
4	1	3	2
3	2	4	1

Feel free to expand the game and learn more about sudoku and how its solved.

Be prepared for much complicated stuff if you try 9X9 puzzle.

Good Luck!