2021

Web Coding JavaScript

Learn about JavaScript, a programming language that helps create dynamic changes in a webpage.

Orb education

☐	1. Introduction to JavaScript
☐	2. JavaScript Outputs
☐	3. Variables
☐	4. Strings
☐	5. Inputs
☐	6. Data Types
☐	7. Functions
☐	8. The 'if … else' Clause
☐	9. Logical Operators
☐	10. Data Validation
☐	11. While Loops
☐	12. For Loops
☐	13. Noughts and Crosses
☐	14. Arrays
☐	15. Connect 4
☐	16. Reversi

JavaScript makes things happen on a webpage. The JavaScript programming language has a wide range of uses, including:

- changing text or switching images on a page after a user action;
- performing calculations and displaying the results;
- validating data entered into a form such as email addresses and telephone numbers;
- building games and other programs.

Where is the JavaScript?

JavaScript can be found in all sorts of places. In the past, scripts were often placed amongst the HTML. Web designers now try and separate as much code from the actual content of the webpage as possible. In fact, JavaScript is often delivered to the web browser in a separate, external file. This makes the HTML less complicated and helps search engines work out what the page is about. Another benefit of this solution is that the JavaScript file can be stored (or cached) on the user's machine so that other pages will load faster if they use the same scripts.

In the example below, a JavaScript function has been added to the head section of a webpage. This particular JavaScript is executed when the user clicks a button on the screen. We will place JavaScript in various locations as we are learning but remind you that external script files are the more desirable solution in the long run.

JavaScript in the Head Section

Start script tag

JavaScript function

End script tag

The JavaScript function runs when a button on the page is clicked (don't worry about how all this works for the moment)

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Section D</title>
<script>
  function ChangeText() {
  document.getElementById("words").innerHTML = "Then change them to these";
  }
</script>
</head>

<body>
<h1>My Web Page</h1>

<p id="words">We will start with these words</p>
<button type="button" onClick="ChangeText()">Try Me</button>
</body>
</html>
```

Note: *It is anticipated that you have a basic understanding of HTML coding before continuing. ORB Education offers a package titled 'Web Coding – HTML & CSS' which covers this content. Otherwise, there is plenty of help online.*

Introduction to JavaScript (page 2)

Some points about JavaScript

1. JavaScript is a true programming language. Mistakes can cause errors in the webpage although most modern browsers will simply ignore the whole script if they find issues with the coding. If nothing appears to happen on the page then it probably means your script is being ignored.

2. JavaScript can be executed at different times e.g. when the page is first displayed, when a button is clicked, when the mouse is placed over an object or when a form is submitted.

3. JavaScript is very fussy; you have to get the code exactly right or it will not work. This can be frustrating at times, especially with the frequent use of the squiggly brackets { and }.

4. Visitors to your website might have JavaScript disabled. This means that they won't be able to use anything that relies on it.

5. The JavaScript interpreter (reader) in your browser ignores spaces and line breaks. You could write long, complicated scripts on a single line but they would be very difficult to read and understand for the human programmer. Website building applications will often condense JavaScript into a compact form before serving it up, so JavaScript files downloaded from other sites may be difficult to decipher.

Task 1 – Alert Boxes

a. Let's get started with the coding. Open a simple text editor on your computer. This may be Notepad (Windows), TextEdit (Mac) or any other simple text editing application. We are using Notepad++ as it colours the code for us. The colours don't make any difference to how the code works; it's just easier to understand for the human.

Note: *Don't use Word Processing applications such as Microsoft Word as they hide too much secret code behind the scenes. You may use Web Design applications such as Dreamweaver, although the bells and whistles can be a distraction when coding; better to keep things simple. Notepad++ for Windows is really one of the best apps for these tasks. There are similar applications for the Mac such as 'Brackets'.*

b. Type the code on the right. If you are using Notepad++, select HTML from the *Language* menu to colour your code as shown.

c. Save the file as **01. Template.htm**. If there is an option to do so in your application, select the file type *HTML*, *htm* or *Hyper Text Markup Language* from the list of choices. Remember the location of your saved file.

This file can be used as a template whenever you need to start afresh. It is ready to accept JavaScript and HTML content.

Note: *You might still see the full attribute <script LANGUAGE="JavaScript"> used. This is no longer necessary as JavaScript is the default scripting language in today's browsers.*

01. Template.htm

```
1  <!doctype html>
2  <html>
3
4  <head>
5  <meta charset="utf-8">
6  <title>JavaScript Template</title>
7  <script>
8
9  </script>
10 </head>
11
12 <body>
13 <h1>JavaScript Template</h1>
14
15 </body>
16
17 </html>
```

Strings are pieces of text. Strings are usually enclosed in double quotes. The following are all examples of strings:

"This is a string" ***"And another"*** ***"2"***

Strings are used in association with variables e.g.

	Variable name	***Value***
var string_1 = "This is a string"	*string_1*	*This is a string*
var string_2 = "And another"	*string_2*	*And another*
var string_3 = "2"	*string_3*	*2 (as text)*

Task 1 – Concatenation

a. Save a copy of your template as **04. Strings 1.htm**. Add the code on the right and view the result. The words "This is a string" should be displayed below your heading.

```
<body>
<h1>JavaScript Strings</h1>
<p id="Answer"></p>
<script>
var string_1 = "This is a string"
document.getElementById("Answer").innerHTML = string_1;
</script>
</body>
```

b. We can use the '+' sign to join two strings together. This is called *concatenation*. Concatenation is different to adding numeric values together.

Adjust the script so that it now looks like ours on the right.

Write down the text displayed **exactly** as it appears in your browser.

```
<script>
var string_1 = "This is a string"
var string_2 = "And another"
var string_3 = "2"
var joinString = string_2 + string_3;
document.getElementById("Answer").innerHTML = joinString;
</script>
```

c. Change the line so that it now reads:

var joinString = string_2 + " " + string_3;

How have we changed the presentation of the text?

Task 2 – Do you get it?

Analyse the script shown and write down the text that you think it will display.

After making your prediction, type the code into a copy of the file saved as **04. Strings 2.htm** and see if you were correct.

```
<script>
var valueX = "2";
var valueY = "4";
var total = valueX + valueY;
document.getElementById("Answer").innerHTML = total;
</script>
```

Strings (page 2)

Task 3 – Length Property

A string is made up of a certain number of characters. For example, the string "house" is made up of five characters and the string "265" is made up of three. The length is one of the *properties* of a string.

To find the length of a string in JavaScript, use the *syntax*: string.length; *e.g.: myString.length;*

Note: *'Syntax' is the structure of statements in a computer language. It is like the rules of grammar applied to computing.*

In the code on the right, we have placed the word "house" in a string named *string_1*. We have then created a second variable called *result* which holds the calculated length of the string Finally, this value is displayed in the *Answer* paragraph on our page. Notice that we don't need the quote marks when using variable names.

```
<p id="Answer"></p>
<script>

var string_1 = "house";
var result = string_1.length;

document.getElementById("Answer").innerHTML = result;

</script>
```

a. Create this code in a file saved as **04. Strings 3.htm**.

b. Edit the code so that it instead displays the length of the word "multiples" and save the file.

Extension

In a copy of the file named **04. Strings 3E.htm**, create code that produces the output shown on the right, correct for whatever word is entered into the string.

Length Method

The word multiples has 9 characters.

Concatenate pieces of text and variables, remembering to place the text in quotes but not the variables. Part of the code is shown below.

```
var output = "The word " + string_1 +
```

Once you have cracked that challenge, add some single quotes around the word when it is displayed. Test your code with different words.

The word 'multiples' has 9 characters.

Task 4 – toUpperCase and toLowerCase Methods

A *method* is a section of programming code that performs a specific task on an *object* such as a string. There are lots of built-in methods in JavaScript that you can make use of. We will try the string methods *toUpperCase* and *toLowerCase*.

Methods in JavaScript use the *syntax*: string.method(); e.g.: string_1.toUpperCase();

Create a webpage named **04. Strings 4.htm** that displays the output shown below. It should take some text from a variable and display it. It should then change the text into upper and lower case and display each of these strings. The snippet of code below will help.

```
var string_1 = "This Is Some Text";

var output1 = string_1.toUpperCase();
var output2 = string_1.toLowerCase();
```

toUpperCase / toLowerCase

This Is Some Text

THIS IS SOME TEXT

this is some text

Although JavaScript can be used for all sorts of things in a webpage, we will spend some time looking at the validation of inputs as this enables us to test lots of ideas without building complicated pages. Forms are used to collect information from the user e.g. names, addresses and emails. These forms are found all over the internet but they have been a source of a huge number of problems because devious people can use them to gain access to online databases. It is therefore necessary to place strict controls over the data that is allowed through. This is an example of *data validation*.

As far as the validation of form data is concerned, you may refuse information for any of the following reasons:

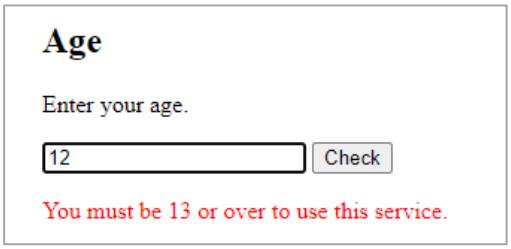
- There are too many or too few characters (e.g. for usernames and passwords);
- A number that is either too high or too low has been entered (e.g. for age checks);
- The inclusion of unwanted characters (e.g. you may not want brackets in a telephone number field);
- An email address may clearly be fake;
- There may be computing code included in the text that is designed to infiltrate the website and database.

Task 1 – Age Restriction

An easy place to start with data validation is to check the size of a number.

Create a file named **10. Data Validation 1.htm** and check that the age entered into a box is 13 or over. Display a warning if it is less than 13. Place the function in your JavaScript file.

You might also improve the display a little. We have simply made the output paragraph red. If you understand CSS you can make further improvements.



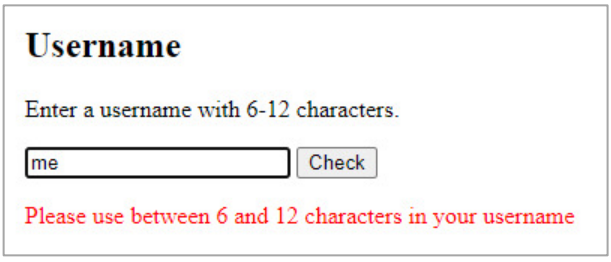

```
<p style="color:red" id="outputText"></p>
```

Task 2 – Username Length

Copy your solution and name the new file **10. Data Validation 2.htm**. This page should check that the username entered is between 6 and 12 characters in length.

Remember that the length property (myString.length) tells you the number of characters in a string.

Use the *focus()* method to return the cursor to the box if the username is not allowed.




```
document.getElementById('username').focus();
```

Extension

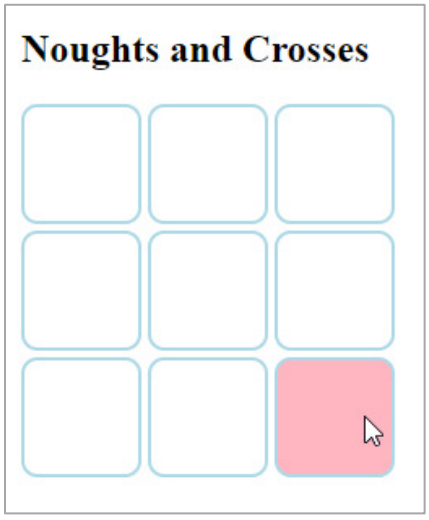
Using a file named **10. Data Validation 2E.htm**, create a single page that checks both the age and the username. There should be only one *Check* button and one function. You may make use of an *else if* statement.

We should have enough understanding now to create some more interesting programs. We'll start will a very simple *Noughts and Crosses* game.

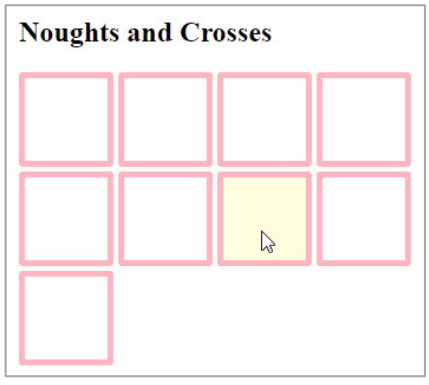
Task 1 – HTML and CSS

This course is about learning JavaScript rather than HTML and CSS, so we will give you the basic design for the board.

a. Copy the code on the right into a new page named **13. Noughts and Crosses 1.htm**.



b. Experiment with different designs, changing any of the values <u>after</u> the colons in the style section. Note that the width setting determines how many boxes fit across the *board* div before starting a new line. This value might need to be adjusted if you change your design.




```
<!DOCTYPE html>
<html>

<head>
<meta charset="utf-8">

<title>Noughts and Crosses</title>

<style>

.board {
    width: 264px;
}

.box {
    width: 40px;
    height: 40px;
    display: inline-block;
    border: 2px solid lightblue;
    border-radius: 10px;
    color: gray;
    text-align: center;
    padding: 20px;
    font-size: 30px;
    margin: 2px;
    float: left;
}

.box:hover {
    background: lightpink;
}

</style>
</head>

<body>

<h2>Noughts and Crosses</h2>

<div class="board">
  <div id="00" class="box"></div>
  <div id="01" class="box"></div>
  <div id="02" class="box"></div>
  <div id="10" class="box"></div>
  <div id="11" class="box"></div>
  <div id="12" class="box"></div>
  <div id="20" class="box"></div>
  <div id="21" class="box"></div>
  <div id="22" class="box"></div>
</div>

</body>

</html>
```

Noughts and Crosses (Page 2)

Task 2 – Onclick Events

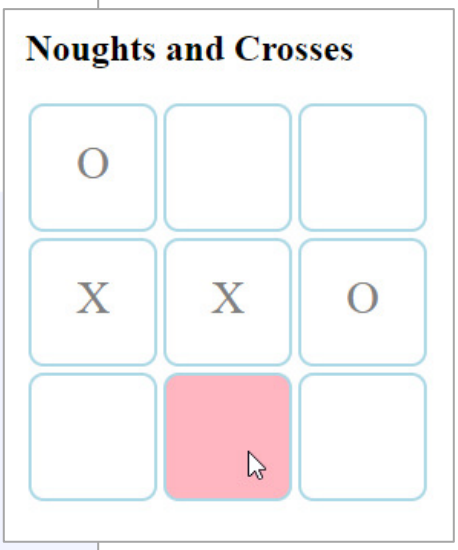
We can now add onclick events to each box, calling a function we have named *clickBox*.

a. Add the onclick event to each box as below. The *this* keyword identifies the object to the function (in this case a div).

b. Create the *clickBox* function. This can be placed beneath the page in the HTML; you can move it into a separate JavaScript file later if you wish (as with the style sheet).

c. Adjust the style of your 0s and Xs if you wish, remembering that if the boxes don't arrange themselves in a 3x3 grid then you might have to change the width of the *board* div.

```
    <div id="20" class="box" onclick="clickBox(this)"></div>
    <div id="21" class="box" onclick="clickBox(this)"></div>
    <div id="22" class="box" onclick="clickBox(this)"></div>
</div>

</body>

</html>



<script>

var currentPlayer = "O";

function clickBox(box) {

 box.innerHTML = currentPlayer;

 if(currentPlayer == "O") {
 currentPlayer = "X"
 }
 else {
 currentPlayer = "O";
 }
}

</script>
```



Questions

1. What is the starting value of the *currentPlayer* variable?

2. What two values does this variable switch between?

   Note: *The currentPlayer variable is declared outside the function. This makes it a global variable, remembered at all times. If we declared the variable inside the function then the value would never switch; it would start as a "O" after every click. You can test this.*

3. What happens if you click on the same box more than once? How could you stop this from happening?

   Try adding the *if statement* shown to the start of your function. It basically stops the function processing if the selected box isn't empty.

```
if(box.innerHTML != "") {
return;
}
```

We can build on the ideas from the *Noughts and Crosses* task to develop a simple *Connect 4* game. Note that we are really entering territory here where other, more complex methods should be used for the programming. However, we are just practicing basic skills so will stick to the ideas that we have learned so far.

Task 1 – Coloured Circles

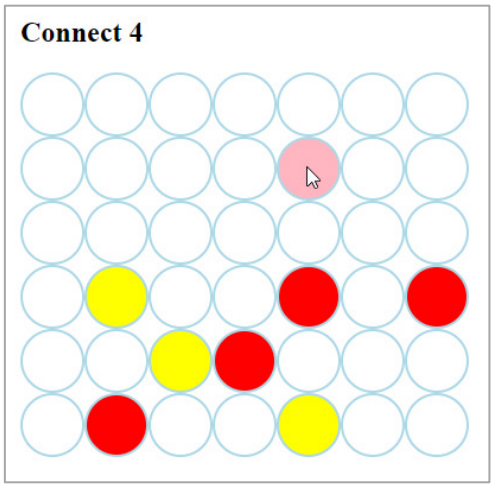
a. Working in a copy of your *Noughts and Crosses* solution (named **15. Connect 4.htm**) build a 7x6 grid. By increasing the *border-radius* in the style section, the shapes can be changed to circles.

b. The circles need to be given IDs. To make things easier, our bottom row will have the IDs 11, 12, 13 etc. The second row up will be 21, 22, 23 etc. The first circles created in the HTML will be the top row, with IDs 61, 62, 63 etc.

```
<div class="board">
  <div id="61" class="box" onclick="clickBox(this)"></div>
  <div id="62" class="box" onclick="clickBox(this)"></div>
  <div id="63" class="box" onclick="clickBox(this)"></div>
  <div id="64" class="box" onclick="clickBox(this)"></div>
  <div id="65" class="box" onclick="clickBox(this)"></div>
  <div id="66" class="box" onclick="clickBox(this)"></div>
```

c. Use code such as that below to change the background colour of the clicked object.

```
if(currentPlayer == "R") {
box.style.backgroundColor = "red";
currentPlayer = "Y";
}
else {
```



d. In the *Noughts and Crosses* game, we placed text in the box when it was clicked. To find out whether a line had been formed we then checked through the text in each box in the grid. It is possible to use a similar solution for our Connect 4 game. However, it will make life a lot easier if the function actually knows which circle is being clicked. We will therefore amend our solution so that the ID is passed to the function after a click. Change the *onclick* events for the circles so that they are as below (copying into a text editor then using the *Find and Replace* function will make light work of this).

```
<div id="61" class="box" onclick="clickBox(this.id)"></div>
<div id="62" class="box" onclick="clickBox(this.id)"></div>
```

As we are now working with IDs rather than objects, we must use *getElementById* in order to colour the clicked circle. We have also changed our variable name from *box* to *boxID* to reflect its new contents.

```
function clickBox(boxID) {

 if(currentPlayer == "R") {
 document.getElementById(boxID).style.backgroundColor = "red";
```

We will finish our series of games with a real challenge. Reversi (sometimes called Othello) is a strategy game where players try and dominate the board with as many of their own coloured discs as possible. If you haven't played the game before, search for 'play reversi online' and learn about out how the game works.

This project is relatively complicated but we suggest you have a go at building a solution yourself first. Programming challenges are interesting to think about and simply copying someone else's program is like looking at the solution to a puzzle. Having said this, if you have thrashed away at the problem solving for a while, it's good to get a helping hand. We therefore do provide a walkthrough for a solution below. Try to build this yourself first and then use our solution if required.

Task 1 – Setting up the Board

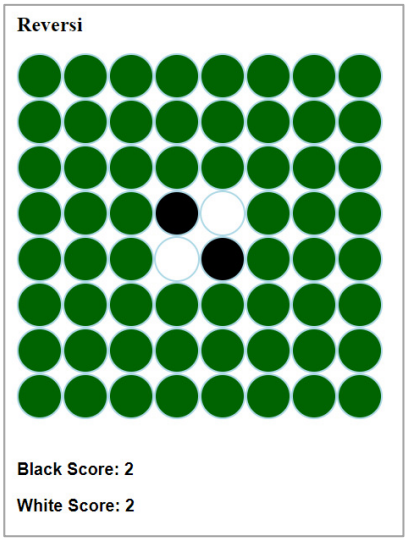
a. Working in a copy of your *Connect 4* solution (named **16. Reversi 1.htm**) build an 8x8 grid. The IDs for the boxes will be 81-88, 71-78 etc.

b. We have simply set the discs with a dark green background colour in the CSS. You may redesign the board to create a better solution.

c. Set up the initial state of the board with four of the discs coloured at the start of the JavaScript. You should also place these starting values in the array that holds the information about the current state of play.

```
for (i=0; i<88; i++) {
filledCircles[i]="N"
}
```

```
document.getElementById(45).style.backgroundColor = "black";
document.getElementById(54).style.backgroundColor = "black";

filledCircles[44] = "W";
filledCircles[55] = "W";
```



d. Add two paragraphs to hold the scores with some initial text. The text will be replaced each time a disc is placed.

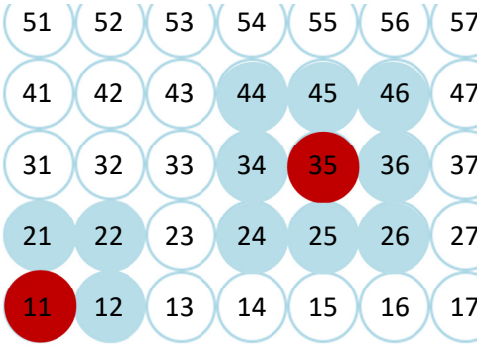
Task 2 – Adjacent Boxes

When deciding if a move is allowed, the first things to check are that:

1. The clicked box is empty;
2. One of the surrounding boxes is of the opposite colour.

Think about how these checks might be achieved. Using a copy of your file named **16. Reversi 2.htm**, have a go at building a solution to this part of the problem. Try and use loops to check the applicable surrounding boxes. You might also need the *toString()* method which is used like this:

myString = myNumber.toString()



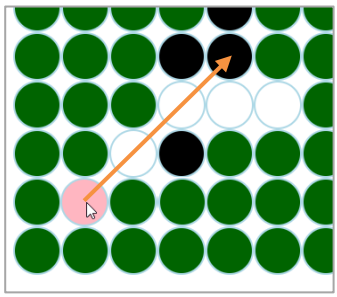
Note that this is confusingly different to the *Number* function which (if you remember) was used like this:

myNumber = Number(myString)

Don't worry if you can't complete this task; you may move on and get help with the next one. It's good to have a go.

Reversi (Page 2)

Task 3 – Allowed Clicks



If you completed the last task you are doing well (especially if you introduced some loops within loops). Unfortunately, the full solution is quite a bit more complicated. Not only must we check that the adjacent box is of the opposite colour, but if it is, we must continue checking in that same direction to find out if one of our own discs is further down the line.

Rather than simply checking the 8 adjacent boxes, we must check lines of boxes in all 8 possible directions.

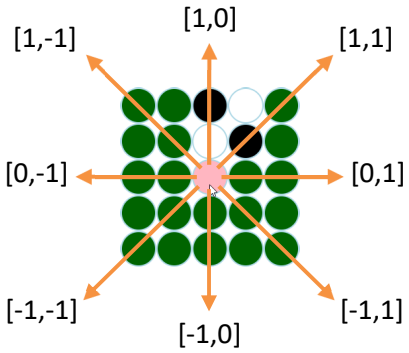
If you think you can do this then please just continue with your own solution. You might scan the instructions below to pick up a few ideas but do your own thing as much as possible.

The rest of us will work through one solution, keeping things as straightforward as possible.

a. Work in a copy of your file from Task 1 named **16. Reversi 3.htm**.

b. We will consider steps in each of the 8 different directions. From our starting position, the instruction [0,1] would move us one step to the right (up 0 rows; across 1 column). If the first step finds a box of the opposite colour then we would continue stepping in the same direction until we find either an empty box, one of our own discs or we hit the edge of the board.

Similarly, each of the other 8 directions can be navigated using the steps shown on the right.



c. Our initial JavaScript created an array called *filledCircles*. This held information about the state of each box ("N", "B" or "W"). We then changed 4 of these to "B" or "W" for the starting discs. Our *clickBox* function will first check that the box being clicked is empty.

d. We will need the row and column of the clicked box as a pair of numbers rather than strings. After slicing up the ID, use the *Number* function to convert the strings to actual numbers.

e. Add a variable called *boxAccepted* that starts with a value of *false* but will be switched to *true* if the click is allowed.

```
function clickBox(boxID) {

 if(filledCircles[boxID] != "N") {
 return;
 }

 var rowNumber = boxID.slice(0,1)
 var colNumber = boxID.slice(1,2)
 var boxAccepted = false;

 rowNumber = Number(rowNumber);
 colNumber = Number(colNumber);
```

Add the code shown on the right to your function (but with the cut lines completed, of course). This code places all the steps in an array then moves through them one at a time. The last line is for error checking. You may remove the comment marks to display information about the cycle as it is processed.

```
var stepsArray = [ [0, 1], [1, 1], [1, 0], [1, -1], [0,

for (i = 0; i < 8; i++) {

 stepsToMove = stepsArray[i];

 rowStep = stepsToMove[0];
 colStep = stepsToMove[1];

 //document.write("rowStep=" + rowStep + ";colStep=" +
}
```