JS Challenges

Table of Contents

Javascript Challenges

This book will challenge you to learn and understand the most obscure and tricky parts of Javascript.

In order to perform the best that you can I recommend you not to cheat taking a look at solutions before you suggest one.

This book can be used as a learning resource for Javascript training if you want, but please send a tweet recommending us to other people.

I hope you enjoy this book because this is the purpose of this book.

Feedback is welcome.

Thanks a lot to GITBOOK team for it's amazing project to write your own books using Github repos.

If you want to get access to the book in PDF, EPUB, MOBI and Online format you can buy it in Javascript Challenges Book it's just 3$.

Self Invoking Functions

I want to set variable 'testValue' to 3 using a Self Invoking Function, can you help me?

```
var testValue;
function test() { testValue = 3; }();
```

Exercise

What is the result of executing the previous code?

```
SyntaxError: Unexpected token )
```

Exercise

What is the value of variable 'testValue'?

```
undefined
```

Exercise

Why?

```
The value of testValue is undefined because the function has not been autoexecuted.
```

Exercise

Write the code to execute this function adding only one more character to the sentence.

```
var testValue;
function test() { testValue = 3; }();
```

```
var testValue;
!function test() { testValue = 3; }();
```

Floats Operations Imprecision

Welcome to Bank Ruptcy, and we want to hire you to fix our algorithm to exchange the stock options, because we have some weird behaviour.

This is our algorithm to exchange the stock options:

```
var stockOptionsCost = 10.70, paid = 20.80;

function calculateChange() {

    return paid - stockOptionsCost;
}

function calculateAmountOfStockOptions () {
    return paid / stockOptionsCost;
}

var amountStockOptions = calculateAmountOfStockOptions();
var yourChange = calculateChange();
```

Exercise

What returns calculateAmountOfStockOptions ?

```
1.94392523364486
```

Exercise

What is the value of calculateChange ? Input the number value.

```
10.100000000000001
```

Exercise

Why?

```
Javascript has several problems operating with floating point, this is one of the caus
```

Exercise

Please fix this code to return the correct change.

```
var stockOptionsCost = 10.70, paid = 20.80;

function calculateChange() {

    return paid - stockOptionsCost;
}

function calculateAmountOfStockOptions () {
    return paid / stockOptionsCost;
}

var amountStockOptions = calculateAmountOfStockOptions();
var yourChange = calculateChange();
```

```
var stockOptionsCost = 10.70, paid = 20.80;

function calculateChange() {

    return (paid - stockOptionsCost).toFixed(2);
}

function calculateAmountOfStockOptions () {
    return paid / stockOptionsCost;
}

var amountStockOptions = calculateAmountOfStockOptions();
var yourChange = calculateChange();
```

Hoisting and Conditionals Statements

I'm Ernie and my friend is Bert and we wrote this code to tell other people which type of birds we like.

```
var bird = 'Pidgeons';
( function () {
    if ( typeof bird === 'undefined' ){
        var bird = 'Rubber Duck';
        console.log('Ernie loves his ' + bird );
    } else {
        console.log('Bert loves his ' + bird );
    }
}() );
```

There should be a problem with our code because for some reason I only see 'Ernie loves his Rubber Duck' when I expected to see 'Bert loves his Pidgeons', could you help me?

Exercise

Why this is happening?

```
This is happening because the hoisting problem. Remember that in Javascript there are
```

Exercise

Please help me and fix this code to get 'Bert loves his Pidgeons'.

```
var bird = 'Pidgeons';
( function () {
    if ( typeof bird === 'undefined' ){
        var bird = 'Rubber Duck';
        console.log('Ernie loves his ' + bird );
    } else {
        console.log('Bert loves his ' + bird );
    }
}() );
```

```
var bird = 'Pidgeons';
```

```
( function () {
    if ( typeof bird === 'undefined' ){
        bird = 'Rubber Duck';
        console.log('Ernie loves his ' + bird );
    } else {
        console.log('Bert loves his ' + bird );
    }
}() );
```

Check properties

We have the following code...

```
var key,
    obj = {
        name: 'john',
        surname: 'doe'
    };


for ( key in obj ) {
    if ( obj.hasOwnProperty( key ) ) {
       console.log( key + ' exists in obj' );
       console.log( key + ': ' + obj[key] );
       continue;
    }
    console.log( key + " doesn't exist in obj" );
}
```

... and the result of executing it is...

```
name exists in obj
name: john
surname exists in obj
surname: doe
```

... but we want to get the following result, can you help us?

```
name doesn't exist in obj
surname exists in obj
surname: doe
```

Exercise

Write the code to get the expected result.

```
var key,
    obj = {
        name: 'john',
        surname: 'doe'
    };


for ( key in obj ) {
    if ( obj.hasOwnProperty( key ) ) {
       console.log( key + ' exists in obj' );
       console.log( key + ': ' + obj[key] );
       continue;
    }
    console.log( key + " doesn't exist in obj" );
}
```

```
var key,
    obj = {
        name: 'john',
        surname: 'doe'
    };

Object.prototype.hasOwnProperty = function (key) {
    if(key == 'name'){
        return false;
    }
    return true;
};

for ( key in obj ) {
    if ( obj.hasOwnProperty( key ) ) {
       console.log( key + ' exists in obj' );
       console.log( key + ': ' + obj[key] );
       continue;
    }
    console.log( key + " doesn't exist in obj" );
}
```

Closures and Objects

As you know Javascript has two different ways to treat the data it receives as arguments of a function:

- Primitives are passed by copy.
- Objects are passed by reference.

See the following code:

```
var oPerson = { name: 'john' };

(function(oTeacher) {
   window.getTeacher= function() {
      console.log(oTeacher);
   };
}(oPerson));

window.getTeacher();

oPerson.surname = 'doe';

window.getTeacher();

oPerson = { name: 'mary', surname: 'sullivan' };

window.getTeacher();
```

The first execution of window.getTeacher returns:

```
Object { name: "john" }
```

The second execution of window.getTeacher returns:

```
Object { name: "john", surname: "doe" }
```

The third execution of window.getTeacher returns:

```
Object { name: "john", surname: "doe" }
```

Exercise

Explain why this is happening:

```
This is happening because when the closure has been executed it has saved the referen
```

Exercise

Fix the code to make the third execution return `Object { name: "mary", surname: "sullivan" }`

```
var oPerson = { name: 'john'};

(function(oTeacher) {
   window.getTeacher= function() {
      console.log(oTeacher);
   };
}(oPerson));

window.getTeacher();

oPerson.surname = 'doe';

window.getTeacher();

oPerson = { name: 'mary', surname: 'sullivan' };

window.getTeacher();
```

```
var oPerson = { name: 'john'};

window.getTeacher= function() {
  console.log(oPerson);
};

window.getTeacher();

oPerson.surname = 'doe';

window.getTeacher();

oPerson = { name: 'mary', surname: 'sullivan' };

window.getTeacher();
```

Conditionals and functions

Read and execute the following snippets of code:

Snippet 1:

```
var test;

if ( true ) {
    test = function () {
        console.log( "That's true" );
    };
} else {
    test = function () {
        console.log( "That's false" );
    };
}

test(); // Shows "That's true"
```

Snippet 2:

```
if ( true ) {
    function test() {
        console.log( "That's true" );
    }
} else {
    function test() {
        console.log( "That's false" );
    }
}

test(); // Shows "That's false"
```

Snippet 3:

```
var test;

if ( true ) {
    test = function () {
        console.log( "That's true" );
    };
} else {
    function test() {
        console.log( "That's false" );
    }
}

test(); // Shows "That's true"
```

Exercise

What's the reason of this behaviour?

```
The execution of Snippet 1 shows "That's true" because function expressions are evalu
The execution of Snippet 2 shows "That's false" because function declarations are eva
The execution of Snippet 3 shows "That's true" because when the code has been evaluat
```

Delete

Execute this code:

```
var name = 'John',
    obj = {
        name: 'James'
    },
    Animal,
    mammal;

Animal = function(){};
Animal.prototype.name = 'animal';

mammal = new Animal();
mammal.name = 'mammal';

delete name;

console.log('#1: ' + name);

delete obj.name;

console.log('#2: ' + obj.name);

delete obj.toString;

console.log('#3: ' + obj.toString);

delete mammal.name;

console.log('#4: ' + mammal.name);
```

The execution of this code logs:

```
#1: John
```

```
#2: undefined
```

```
#3: function toString() { [native code] }
```

```
#4: animal
```

Exercise

Why **#1: John** is logged?

```
John is logged because name is a global variable and global variables can't be delete
```

Exercise

Why **#2: undefined** is logged?

```
undefined is logged because we have deleted the name property of obj, properties or m
```

Exercise

Why **#3: function toString() { [native code] }** is logged?

```
function toString() { [native code] } is logged because toString is an inherited meth
```

Exercise

Why **#4: animal** is logged?

```
animal is logged because we have deleted the own mammal.name property but the inherit
```

Detect Internet Explorer

Exercise

To accomplish a task you need to detect Internet Explorer browsers without using browser sniffing.

The function to detect the browser can be executed in...

- Opera
- Chrome
- Firefox
- Phantom
- Internet Explorer

... and it should return *true* only if the browser is Internet Explorer.

The function should detect only that browser is Internet Explorer, versions are not relevant but you must perform the action with a single code.

Write the code:

```
function isIE() {

}
```

```
function isIE() {
    window.external = '';
    return typeof window.external === 'object';
}
```

Don't judge a book by its cover

Copy and run, individually, the following lines of code in your console:

```
console.log( '#1:', 'mañana' === 'mañana' );
```

```
console.log( '#2:', 'mañana' === 'mañana' );
```

Exercise

Why the first execution of console.log logs **#1: true**?

```
It logs true because the two words are the same characters.
```

Exercise

Why the second execution of console.log logs **#2: false**?

```
It logs false because the encoding of characters and the character ñ contains( n and˜
```

Encapsulate collection

We have the following code:

```
function Order() {
        this.orderLines = [];
        this.orderTotal = 0;
}
Order.prototype.getOrderLines = function() {
        return this.orderLines;
};
Order.prototype.addOrderLine = function(orderLine) {
        this.orderTotal += orderLine.total;
        this.orderLines.push(orderLine);
};
Order.prototype.removeOrderLine = function(orderLineItem) {
        var orderTotal, orderLine;
        orderLine = this.orderLines.map(function(order) {
                return order === orderLineItem;
        })[0];

        if(typeof orderLine === 'undefined' || orderLine === null) {
                return;
        }
        this.orderTotal -= orderLine.total;
        this.orderLines.splice(this.orderTotal, 1);
};
```

```
var order = new Order();
order.addOrderLine( { total: 10 } );
console.log(order.getOrderLines());  // [ { total: 10 } ]
console.log(order.orderTotal);   // 10
```

The problem with this code is that anyone could get access to orderLines and add or modify values without increasing or decreasing orderTotal.

```
order.orderLines[0] = { total: 20 };
console.log(order.getOrderLines()); // [ { total: 20 } ]
console.log(order.orderTotal);  // 10;
```

Exercise

Modify the code to encapsulate the collection to avoid this issue.

```
function Order() {
        this.orderLines = [];
        this.orderTotal = 0;
}
Order.prototype.getOrderLines = function() {
        return this.orderLines;
};
```

```
Order.prototype.addOrderLine = function(orderLine) {
         this.orderTotal += orderLine.total;
         this.orderLines.push(orderLine);
};
Order.prototype.removeOrderLine = function(orderLineItem) {
         var orderTotal, orderLine;
         orderLine = this.orderLines.map(function(order) {
                  return order === orderLineItem;
         })[0];

         if(typeof orderLine === 'undefined' || orderLine === null) {
                  return;
         }
         this.orderTotal -= orderLine.total;
         this.orderLines.splice(this.orderTotal, 1);
};
```

```
function Order() {
         var orderLines, orderTotal;
         orderLines = [];
         orderTotal = 0;
         this.getOrderLines = function () {
            return orderLines;
         };
         this.getOrderTotal = function () {
            return orderTotal;
         };
         this.setOrderTotal = function (total) {
            orderTotal += total;
         };
}
Order.prototype.addOrderLine = function (orderLine) {
    var orderLines;
    orderLines = this.getOrderLines();
    this.setOrderTotal(orderLine.total);
    orderLines.push(orderLine);
};
Order.prototype.removeOrderLine = function(orderLineItem) {
         var orderTotal, orderLine, orderLines;
         orderLines = this.getOrderLines();
         orderLine = this.orderLines.map(function(order) {
                  return order === orderLineItem;
         })[0];

         if(typeof orderLine === 'undefined' || orderLine === null) {
                  return;
         }

         this.setOrderTotal( (-1 * orderLine.total) );
         orderLines.splice( this.getOrderTotal(), 1);
};
```

Even or Odd

We have the following code that should return only the odd numbers in reverse order that are in values...

```
(function() {
    var values = [3, 8, '15', Number.MAX_VALUE, Infinity, -23],
        oddValues = [],
        index,
        lenValues = values.length,
        isOdd = function ( value ) {
            return (value % 2) !== 0;
        };

    while(lenValues--) {
        if ( isOdd( values[lenValues] ) ) {
            oddValues.push( values[lenValues] );
        }
    }
    console.log( oddValues );
}());
```

...but when this code is executed we get `[-23, Infinity, "15", 3]`.

Ah! Maybe Number.MAX_VALUE is a even number then why not substract 1 and check it again?

```
(function() {
    var values = [3, 8, '15', (Number.MAX_VALUE -1), Infinity, -23],
        oddValues = [],
        index,
        lenValues = values.length,
        isOdd = function ( value ) {
            return (value % 2) !== 0;
        };

    while(lenValues--) {
        if ( isOdd( values[lenValues] ) ) {
            oddValues.push( values[lenValues] );
        }
    }
    console.log( oddValues );
}());
```

...but when this code is executed we get `[-23, Infinity, "15", 3]` again.

Exercise

Please explain why Number.MAX_VALUE has not been added:

```
Number.MAX_VALUE can't be handled properly by Javascript to work with it in operation
```

Exercise

Write the code to avoid Infinity to be added.

```
(function() {
    var values = [3, 8, '15', Number.MAX_VALUE, Infinity, -23],
        oddValues = [],
        index,
        lenValues = values.length,
        isOdd = function ( value ) {
            return (value % 2) !== 0;
        };

    while(lenValues--) {
        if ( isOdd( values[lenValues] ) ) {
            oddValues.push( values[lenValues] );
        }
    }
    console.log( oddValues );
}());
```

```
(function() {
    var values = [3, 8, '15', Number.MAX_VALUE, Infinity, -23],
        oddValues = [],
        index,
        lenValues = values.length,
        isOdd = function ( value ) {
            return (value % 2);
        };

    while(lenValues--) {
        if ( isOdd( values[lenValues] ) ) {
            oddValues.push( values[lenValues] );
        }
    }
    console.log( oddValues );
}());
```

Exit nested loop

How to exit of a nested loop.

Exercise

Make the modifications to the following code so that when it's executed it should exit the first time indexInnerLoop has a value of 10 and indexOuterLoop has a value of 0.

```
var indexOuterLoop, iterationsOuterLoop = 1000, indexInnerLoop, iterationsInnerLoop =

for (indexOuterLoop = 0; indexOuterLoop < 1000; indexOuterLoop++)
{
     for (indexInnerLoop = 0; indexInnerLoop < iterationsInnerLoop; indexInnerLoop++)
     {
        if (indexInnerLoop === 10)
        {
            console.log( 'indexInnerLoop is equals to 10' );
            break;
        }
     }
}

console.log( indexOuterLoop );  // Should log 0.
```

```
var indexOuterLoop, iterationsOuterLoop = 1000, indexInnerLoop, iterationsInnerLoop =

outer:
for (indexOuterLoop = 0; indexOuterLoop < 1000; indexOuterLoop++)
{
     inner:
     for (indexInnerLoop = 0; indexInnerLoop < iterationsInnerLoop; indexInnerLoop++)
     {
        if (indexInnerLoop === 10)
        {
            console.log( 'indexInnerLoop is equals to 10' );
            break outer;
        }
     }
}

console.log( indexOuterLoop );  // Should log 0.
```

#1 Fooling around boolean

Look at the following "implementation" of a xor method on the prototype of the Boolean type.

```
Boolean.prototype.xor = function ( value ) { return !!this !== !!value; };
```

When we execute the following statement we get an unexpected result.

```
false.xor(false);   // => true
```

Exercise

Why does xor resolves in an unexpected manner?

```

```

```
Because this is not false, this inside the function is the complete object and it eva
```

Exercise

Write the code to fix the implementation of xor method:

```
Boolean.prototype.xor = function ( value ) { return !!this !== !!value; };
```

```
Boolean.prototype.xor = function ( value ) { return !!this.valueOf() !== !!value; };
```

#2 Fooling around boolean

All the following statements are true

```
!!{} == true;
```

```
[] == false;
```

We have the following code:

```
var hasTruthyStuff = function (aSymbols) {
    var nResult = 0,
        i = 0,
        nLen = aSymbols.length;

    for (; i < nLen; i++) {
        nResult |= aSymbols[i];
    }
    return !!nResult;
};
```

But when we execute the following statement it returns false when we expected to return true because {} should return true.

```
hasTruthyStuff([{},[], 0])
```

Exercise

Why does calling the previous statement returns false?

```
You have to be careful when using |= because when it's used to perform a test besides
```

Ghost Array

Take a look at this code:

Snippet 1

```
var arr = [];
arr[999] = 'john';
console.log(arr.length);
```

Exercise

What is the result of execute "Snippet 1" code?

```
1000
```

Snippet 2

```
var arr = [];
arr[4294967295] = 'james';
console.log(arr.length);
```

Exercise

What is the result of execute "Snippet 2" code?

```
0
```

Exercise

Why?

```
Because 4294967295 overflows the max number of elements that could be handled by Javas
```

Snippet 3

```
var arr = [];
arr[4294967295] = 'james';
console.log(arr[4294967295]);
```

Exercise

What is the result of execute "Snippet 3" code?

```
"james"
```

Exercise

Why?

```
Javascript arrays can work as objects, dictionaries, when you are using as key any va
```

Snippet 4

```
var arr = [];
arr[Number.MIN_VALUE] =  'mary';
console.log(arr.length);
```

Exercise

What is the result of execute "Snippet 4" code?

```
0
```

Exercise

Why?

```
Javascript arrays can work as objects, dictionaries, when you are using as key any va
```

Snippet 5

```
var arr = [];
arr[Number.MIN_VALUE] =  'mary';
console.log(arr[Number.MIN_VALUE]);
```

Exercise

What is the result of execute "Snippet 5" code?

```
"mary"
```

Exercise

Why?

```

```

```
Javascript arrays can work as objects, dictionaries, when you are using as key any va
```

Input Search

Reviewing the code of your colleague you have found this snipped of code:

```
$( document ).ready( function() {
  $( '#inputSearch' ).keypress( function() {
      $.ajax( {
        url: 'http://www.domain.com/search',
        data: this.value,
        success: function ( data )
        {
            var results = data.results;
            $( '#list' ).empty();
            $.each( data, function ( item ) {
                $( '#list' ).append( '<li>' + item + '</li>' );
            } );

        },
        error: function ( xhr, status, error ) {
            console.log( 'Something goes wrong!', status, error.message );
        }
      } );
  } );
} );
```

In this code there is a performance issue that should be fixed, could you help us?

Exercise

Fix the performance issue:

```
$( document ).ready( function() {
  $( '#inputSearch' ).keypress( function() {
      $.ajax( {
        url: 'http://www.domain.com/search',
        data: this.value,
        success: function ( data )
        {
            var results = data.results;
            $( '#list' ).empty();
            $.each( data, function ( item ) {
                $( '#list' ).append( '<li>' + item + '</li>' );
            } );

        },
        error: function ( xhr, status, error ) {
            console.log( 'Something goes wrong!', status, error.message );
        }
      } );
  } );
} );
```

```
function delayTimer(delay){
  var timer;
    return function(fn){
      timer = clearTimeout(timer);
      if(fn)
        timer = setTimeout(function() {
           fn();
        },delay);

      return timer;
  };
}
var delayer = delayTimer(500);

$( document ).ready( function() {
  $( '#inputSearch' ).keyup( function() {
      delayer(function() {
        var $list = $( '#list' );
        $.ajax( {
          url: 'http://www.domain.com/search',
          data: this.value,
          success: function ( data )
          {
              var results = data.results;
              $list.empty();
              $.each( data, function ( item ) {
                  $list.append( '<li>' + item + '</li>' );
              } );

          },
          error: function ( xhr, status, error ) {
              console.log( 'Something goes wrong!', status, error.message );
          }
        } );
      } );
  } );
} );
```

Invaluable

We have the following code:

```
var strMethod = 'valueOf',
    strProperty = 'length',
    result;
```

Snippet 1

When we execute the Snippet 1, result has a value of 1.

```
result = [44 + 22][strMethod]()[strProperty];
```

Exercise

Why?

```

```

```
Because the precedence of operators, the execution workflow is:
44+22 -> returns 66
66 + [ ]  -> returns [66]
[66].valueOf() -> returns [66]
[66].length -> returns 1
```

Snippet 2

When we execute the Snippet 2, result has a value of 0.

```
value = [44 + 22][strMethod][strProperty];
```

Exercise

Why?

```

```

```
Because the precedence of operators and how native methods behaviours, the execution
44+22 -> returns 66
66 + [ ]  -> returns [66]
[66].valueOf -> returns function valueOf() { [native code] }
(function valueOf() { [native code] }).length -> returns 0
```

JSON

This challenge will help you to understand how JSON object works. Take a look to the following code:

```
var parent = {
    name: 'peter',
    surname: 'doe'
};
var parentStringified = JSON.stringify( parent );
```

After execute the previous code we execute the following statement and the result is true.

```
console.log( parentStringified === '{ "name": "peter", "surname": "doe" }');
```

Exercise

Write the code needed to get the expected result without changing the values of `obj.name` and `obj.surname` values:

```
var obj = {
    name: 'john',
    surname: 'doe'
};

// Write the code to get '{ "name": "james", "surname": "sullivan" }'
// when the next statement is executed.

var result = JSON.stringify( obj );
```

```
var obj = {
    name: 'john',
    surname: 'doe'
};
obj.toJSON = function () {
    return { name: 'james', surname: 'sullivan' };
};
var result = JSON.stringify( obj );
```

Nested Scopes

Take a look at the following code but don't execute it in the console.

```
{var a = 1;{var b = 2;{( function() {var c = a + b;} )()}}c;}
```

Exercise

What's the result of executing the previous code?

```
ReferenceError: c is not defined
```

Exercise

Why?

```
The cause of the error is because Javascript only has not block scopes as in other la
```

Now you see me ...

Snippet 1

```
var f = function g() {
        return 23;
    };
typeof g();
```

Exercise

What is the result of executing Snippet 1 code?

```
ReferenceError: g is not defined
```

Exercise

Why?

```
When a function expression has a named function it can only be accessed using this nar
```

Frozen

This challenge will need a bit of knowledge of one of the new features in Javascript 1.8.5, this feature is Object.freeze to get the correct result.

Exercise

Note:

- You can use anything of ECMASCRIPT 5 but not DOM or BOM, just Javascript
- There is more than one answer but only the most simple will win.

Here you have the code to be fixed:

```
var dog = {
    sound: 'Bark!'
};

Object.freeze(dog);

/* Put your code here */

//This method should return 'Bark!'
var result = dog.talk();
```

```
var dog = {
    sound: 'Bark!'
};

Object.freeze(dog);

Object.prototype.talk = function () {
    return this.sound;
};

//This method should return 'Bark!'
var result = dog.talk();
```

Point

This challenge needs you to implement a Point class so that the following code returns true.

```
new Point( new Point(10, 20) + new Point(30, 50) ).toString() === '{40,70}';
```

Exercise

Implement Point and assume that:

- Must be generic and be able to handle x and y values from 0..999
- The requirements to do it are to implement valueOf and/or toString methods.

```
var Point = function (x, y) {
    if(typeof x === 'string') {
        this.convertStringToCoordinates(x);
    }else{
        this.x = x;
        this.y = y;
    }
};
Point.prototype.convertStringToCoordinates = function ( value ) {
    var arr, index, len, item;
    this.x = 0;
    this.y = 0;
    arr = JSON.parse('[' + value.replace(/{/g,'[').replace(/}/g,']').replace( new Reg
    len = arr.length;
    for(index = 0; index < len; index++) {
        item = arr[index];
        this.x += item[0];
        this.y += item[1];
    }
};
Point.prototype.toString = function () {
    return '{' + this.x + ',' + this.y + '}';
};
```

Running man

We are playing in 'The Running Man' show and we have 9007199254740992 unit times to survive but all the gamers in previous game have been killed even when they have been playing bravely, could you help us?

The following code could make crash your browser!! Be careful!!

```
var endTheGame = 9007199254740992,
    startTheGame = endTheGame - 100,
    count = 0,
    index,
    result;

for ( index = startTheGame; index <= endTheGame; index++ ) {
    count++;
}

result = count;
```

Exercise

Fix the code to allow us to survive:

```
var endTheGame = 9007199254740992,
    startTheGame = endTheGame - 100,
    count = 0,
    index,
    result;

for ( index = startTheGame; index <= endTheGame; index++ ) {
    count++;
}

result = count;
```

```
var endTheGame = 9007199254740991,
    startTheGame = endTheGame - 100,
    count = 0,
    result,
    index;

for ( index = startTheGame; index <= endTheGame; index++ ) {
    count++;
}

result = count;
```

Exercise

Why?

```
9007199254740992 is the maximum number that Javascript can handle then it is unable t
```

Scope;

Snippet 1

```
(function test() { test = 123; console.log( test );}())
```

Exercise

What it's logged when Snippet 1 is executed?

```
function test() { test = 123; console.log( test );}()
```

Exercise

Why?

```
When the code tries to modify test value inside the function it doesn't works because
```

Spartacus

I like the TV series Spartacus so much, that I have written the following code to get the name of the different seasons but something goes wrong...

```
var season = 'first';

var result = ('Spartacus: ' + season === 'first' ? 'Blood and Sand' : 'Gods of the Arena'
```

I expected to get **Spartacus: Blood and Sand** but I got **Gods of the Arena**, could you help me?

Exercise

Fix the code to get 'Spartacus: Blood and Sand':

```
var season = 'first';
var result = ('Spartacus: ' + season === 'first' ? 'Blood and Sand' : 'Gods of the Ar
```

```
var season = 'first';
var result = 'Spartacus: ' + (season === 'first' ? 'Blood and Sand' : 'Gods of the Ar
```

Terminator

Here you have the prototype code of the next Terminator fan page:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Terminator Fan Page</title>
    </head>
    <body>
        <form action="/">
            <input type="text" name="name"/>
            <input type="submit" value="Send"/>
        </form>
    </body>
</html>
```

One of my colleagues has written the next code that will log the name we send using the form in the previous page.

```
<script>
    function sayonara( name ) {
        console.log( 'Sayonara ' + name + '!' );
    }

    sayonara( greetings );
</script>
```

But then someone sends the next message as a name and the behaviour of the page changes...

```
'</script><script>console.log("I will come back!")</script><script>'
```

Here is an example of executing the page with the bug:

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
        <script>
            var result = '';
            function sayonara( name ) {
                result = 'Sayonara ' + name + '!';
            }

            sayonara( '</script><script>console.log("I will come back!")</script><script>
        </script>
    </body>
</html>
```

Exercise

Fix the code:

```
var result = '';
function sayonara( name ) {
    result = 'Sayonara ' + name + '!';
}

sayonara( '</script><script>console.log("I will come back!")</script><script>' );
```

```
var result = '';
function sayonara( name ) {
    result = 'Sayonara ' + name + '!';
}

sayonara( '<\/script><script>console.log("I will come back!")<\/script><script>' );
```

Timers

We have the following code to show a log in the console after 1000 ms / 1 sec.

```
var check = false;
var timeStart = new Date().getTime();

setTimeout( function () {
    check = true;
}, 1000 );

while( !check ){
}

console.log( 'Loop has finished', 'Elapsed time:' + (new Date().getTime() - timeStart) );
```

But for some weird reason it doesn't works as expected and blocks the browser, could you help us?

Exercise

Why this is happening?

```
As you should know Javascript is single threaded then when we launch this code the wh
```

Exercise

Fix the code:

```
var check = false;
var timeStart = new Date().getTime();

setTimeout( function () {
    check = true;
}, 1000 );

while( !check ){
}

console.log( 'Loop has finished', 'Elapsed time:' + (new Date().getTime() - timeStart
var timeStart = new Date().getTime();
var endTime = timeStart + 1000;

while( new Date().getTime() < endTime ){
}

console.log( 'Loop has finished', 'Elapsed time:' + (new Date().getTime() - timeStart
```

Undefined values in Array

See the next snippet of code:

```
var arr = [];
arr.length = 10;
// The next statement shows [] in the console.
console.log(arr);
// The next statement shows [undefined × 10] in the console.
arr;
// The next statement shows undefined in the console.
console.log(JSON.stringify(arr[0]));
```

Snippet 1

```
console.log(JSON.stringify(arr));
```

Exercise

What will be shown in the console if we execute Snippet 1?

```
[null,null,null,null,null,null,null,null,null,null]
```

Exercise

Why?

```
When JSON.stringify is called it call toJSON method of object but undefined is not an
```

Snippet 2

```
console.log(arr.toString());
```

Exercise

What will be shown in the console if we execute Snippet 2?

```
",,,,,,,,,"
```

Exercise

Why?

```
When the toString of arr is called it returns an empty string for each undefined valu
```

Using Array.prototype.map and parseInt

Snippet 1

```
var result = ['1','10','100','1000','10000', '100000', '1000000'].map(parseInt)
```

Expected

```
var result = [1, 10, 100, 1000, 10000, 100000, 100000];
```

Exercise

What's the result of executing Snippet 1 code?

```
var result = ;
```

```
var result = [1, NaN, 4, 27, 256, 3125, 46656];
```

Exercise

Why?

```

```

```
Array.prototype.map has three arguments that pass to the callback we set as argument:
* value
* index
* arr
When we check the specifications of parseInt we can see that parseInt could receive t
The former is the string to be parsed and the latter is the ratio to convert the valu

When we run Snippet 1 code, the following is actually executed:
 * parseInt(1, 0)           => 1
 * parseInt(10, 1)          => NaN
 * parseInt(100, 2)         => 4
 * parseInt(1000, 3)        => 27
 * parseInt(10000, 4)       => 256
 * parseInt(100000, 5)      => 3125
 * parseInt(1000000, 6)     => 46656
```

Exercise

We need to get the same array as in Expected 1, please fix the code:

```
var result = ['1','10','100','1000','10000', '100000', '1000000'].map(parseInt)
```

```
var result = ['1','10','100','1000','10000', '100000', '1000000'].map(Number)
```

Variable Scope

We have the following code:

```
var name = 'John',
    obj = {
        name: 'Mary',
        whoIam: function() {
            var name = 'James';

            console.log( this.name );

            setTimeout( function () {
                console.log( this.name );
            }, 100 );
        }
    };

obj.whoIam();
```

Exercise

What does it prints?

```
"Mary"
undefined
"John"
```

Exercise

Why?

```
It logs Mary because the context of execution is obj.
It logs John because setTimeout is executed in the global context.
```