Basics of JS 2

Prepared by Alex Sobolewski

Hello!

My name is Alex Sobolewski

I have been working in IT as a programmer since 2013. My specialization is JavaScript and Java based solutions. I am strong adherent of Clean Code and good design architecture. Currently i am working as Full Stack Engineer. In this course i will guide you through the basics of JS.

1.
Comparison operators

Comparison operators

What are comparison operators?

Comparison operators - operators that are used to compare values.

There are following types:

- == equals(by value or ref)
- === strict equals(by type and value)
- != not equals(by value or ref)
- !== not equals(by type and value)
- > greater
- >= greater or equal
- < lesser
- <= lesser or equal

Comparison operators
Comparison examples

There are following types:

- ‘string’ == ‘string’
- ‘345’ == 345
- ‘Kasia’ != ‘Kasia’
- 444 !== ‘444’
- 5 > 3
- 11 >= 11
- 10 < 14
- 14 <= 14

Comparison operators

Comparison operators pitfalls

The most common pitfall is about equality by value and equality value and type. Consider this:

```
let num = 123;
num == ‘123’ and num == 123;
num === ‘123’ and num === 123;
```

```
let name = undefined;
name == ‘undefined’
name === ‘undefined’
```

Strict equality is considered better way, because it avoids ambiguity in the equality.

Comparison operators

Comparison operators pitfalls

Another often problem - it is about comparison strings.
Consider this:

‘a’ > ‘b’ // false
‘c’ > ‘a’ // true

Letters are compared by internal codes. **DO NOT compare strings in such way!**

Task 1

1) Create 8 examples with each comparison operator, so that the result will be true
2) Create 8 examples with each comparison operator, so that the result will be false

Comparison operators

Summary

- In order to compare values we use predefined comparison operators
- There are 8 comparison operators
- **Remember about strict equality!**
- **Remember about comparison with undefined!**
- **Remember about comparison between strings!**

2.
Logical operators

Logical operators

What are logical operators?

Logical operators - operators that are used to reflect logic actions.

There are following types:

- || or
- && and
- ! not

Logical operators

Logical operators - examples

- 123 === 321 || 5 === 5 || 1 == 1
- ‘A’ === ‘A’ && ‘a’ !== ‘b’ && ‘c’ != ‘c’
- !(‘a’ == ‘a’) && !(‘c’ === ‘c’)

Logical operators
OR, AND, negation

As logical expressions are evaluated left to right, they are tested for possible "short-circuit" evaluation using the following rules:

false && (anything) is short-circuit evaluated to false
true || (anything) is short-circuit evaluated to true

If JS finds that obvious evaluation the rest of expression isn’t executed!

Logical operators

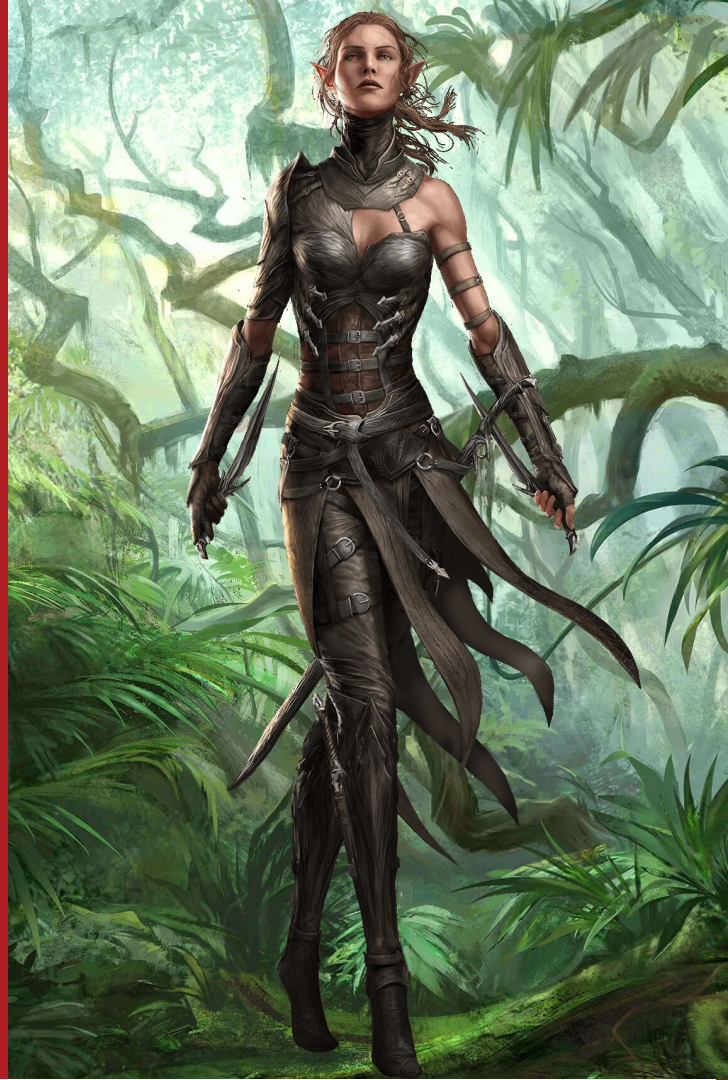
Logical operators pitfalls

The most common problem is about using not-operators. Humans work bad with not-operators.

Consider this:
!((123 !== ‘123’) || (5 != ‘5’) && (x === ‘name’))

Task 2

1) Log 3 statements with true
2) Log 3 statements with false
3) Mix at least to different operators



Logical operators
Summary

- Logical operators are the only way to reflect logic actions
- There are only 3 types of logical operators
- Take care about negation operator - humans work bad with not-operators :)

3.
If-else statements, ternary operator

If-else statements, ternary operator

Statement vs expression

Statement is a part of code that performs some actions, like conditional statements. Statement DOES NOT return anything.

Expressions are part of code that produces or directly return a value, so expressions can be placed wherever a value is needed.

If-else statements, ternary operator

What is if-else statement?

If-else statement - statement, evaluates some condition and based on true or false value decides, which logical path to take.

```
if (x === 1 || a !== 3) {
        ...logical path 1
} else {
        ...logical path 2
}
```

If-else statements, ternary operator

What is if-else-if statement?

If-else-if statement - statement, evaluates some condition and based on true or false value decides, which logical path to take. However, there can be more than just 2 paths.

```
if (x === 1 || a !== 3) {
        ...logical path 1
} else if (z === 'koko') {
        ...logical path 2
} else if (x === 5 && a === 'hoho') {
        ...logical path 3
}
```

If-else statements, ternary operator

What is ternary operator?

Ternary operator - statement, that simplificates two-path if-else statement.

```
if (x === 1 || a !== 3) {
        ...logical path 1
} else {
        ...logical path 2
}
```

=

```
X === 1 || a !== 3 ? ...path 1 : ...path 2
```

If-else statements, ternary operator

If-else statements pitfalls

The most common pitfall is about breaking if-else-if statements into separate statements. Consider this:

```
if (a === ‘a’) {
        ...do smt
} else if (b === ‘b’) {
        ...do smt
} else {
        ...do smt
}
```

```
if (a === ‘a’) {
        ...do smt
}

if (b === ‘b’) {
        ...do smt
}
```

```
if (a !== ‘a’ && b !== ‘b’) {
        ...do smt
}
```

Task 3

1) Declare two variables.
   Create if-else-if statement, which checks those variables. Each of logical path should log different messages. Make them show every path.
2) Rewrite it to ternary shape.

If-else statements, ternary operator

Summary

- If-else statements are used for choosing certain logical path of the application.
- Be aware of breaking if-else-if in separate statements
- Ternary operator is the same as if-else, so if you need something short - use ternary.

4.
Switch statement

Switch statement

What is switch statement?

Switch statement - alternative solution to many if-else-if-else statements. Each case should be ended by **break** statement.

```
let name = ‘Cassandra’;
switch (name) {
        case ‘cassandra’: ...code; break;
        case ‘Cassandra’: ...code; break;
        default: ...code; break;
}
```

Switch statement

Switch statement pitfalls

The most common problem with switch is to forget about break.
Consider this:

```
switch (name)  {
        case ‘Kasia’: console.log(‘Kasia’);
        case ‘Magda’: console.log(‘Magda’);break
        default: console.log(‘It is a man’);
}
```

Switch statement

Switch statement pitfalls

However, it can be useful, if we want to merge some logical paths.
Consider this:

```
switch (transaction)  {
        case ‘Card’: console.log(‘card charged’);
        case ‘Bank’: console.log(‘bank account charged’); break
        default: console.log(‘transaction failed’);
}
```

Task 4

1) Declare a variable.
   Write switch statement, which has 3 logic paths and default path. First logic part should be chosen.
2) Change variable so that the second logic path is chosen.
3) Change variable so that default path chosen.
4) Change the statement, so few logical paths are merged.

Switch statement

Summary

- Switch statement could be very convenient, comparing it to if-else-if
- We can exploit logical path merge case
- However, remember that you can be trapped by merging, if you are not careful with breaks.

5.
Arrays

Arrays

What is an array?

Array - some amount of same or different values, which are encapsulated in one logical entity.

```
let names =[‘Cassandra’, ‘Kasia’, ‘Amely’];
let promoCodes = [ 123, 555, ‘swieta-18’]
let namesArray = [[‘Kasia’, ‘Ewa’], [‘Elison’, ‘Amy’], [‘Karina’, ‘Zuhra’], ‘Zosia’, 777];
```

Arrays

Other ways to declare an array

There is really plenty of array declaration approaches.
Consider this:

```
let names = [];
let names = [‘Kasia’, ‘Alejna’];
let names = new Array();
let names = new Array(‘Kasia’, ‘Alejna’);
```

Arrays
Index and .length

In order to access an item in an array we use **[index]** expression.
let names = [‘Kasia’, ‘Ewa’];
names[0]; // ’Kasia’

.length - shows amount of element in the array
names.length // 2

Arrays
push(value) and pop()

.push(value) is used in order to push an item into the array:
let names = [];
names.push('Kasia');
names[0] // 'Kasia'

.pop() is used in order to take the last item from the array and return it.
names.pop();
Names.length; // 0

Arrays
shift() and unshift()

.shift() is used in order to take the first element and return it:
let names = [‘Bob’, ‘Jack’];
console.log(names.shift()); // ‘Bob’ returned and removed from the array.

.unshift() is used in order to add some element at the beginning of the array:
names.unshift(‘Jackson’, ‘Bobson’);
console.log(names); // ‘Jackson’, ‘Bobson’, ‘Bob’

Arrays
slice() and indexOf()

.slice(start, end) is used in order to return shallow copy of a portion of an array. It starts from **start** index and ends at **end** index. The end index is not included.

```
let names = [‘Kasia’, ‘Ewa’, ‘Magda’, ‘Ada’, ‘Julia’, ‘Alejna’];
names.slice(2, 4); // ‘Magda’, ‘Ada’
```

.indexOf(value) returns an index of an item in the array:

```
let names = [‘Kasia’, ‘Ewa’];
names.indexOf(‘Ewa’) // 1
names[names.indexOf(‘Kasia’)]; // ‘Kasia’
```

Arrays

Arrays pitfalls

The most common problem with arrays is about understanding, that an array - it is an object. Objects are passed by reference. You should remember that objects will be compared not by its content, but initial reference. However, its items can point to the same values.

Task 6

1) Create an array with next values Ania, Ewelina, Karina, Elina.
2) add Andromeda, Natasha
3) get index of Ewelina, store it in separate variable, now add 2 to this index and store
4) slice it from stored index to Natasha’s index to new array
5) unshift new name Shepard
6) compare index 3 from initial array to index 1 from new. Should be true.

Arrays
Summary

- [index] - index expression is crucial for working with arrays.
- .length, ,.push() - they shall be remembered and used asap
- .slice() - is quite common, we should know about it, if there is no external libs available
- You can declare an array as you want
- Remember about the fact, that arrays are passed by reference

6.
Loops

Loops

What is a loop?

Loop - special functionality, which provides convenient way for working with arrays.

There are some types of loops:

- For loop
- Do-while loop
- While-do loop
- Functional loops(described later in the course)

Loops
For loop

For loop - traditional loop for iterating over an array.

```
let names = [...];
for (let i = 1; i <= 10; i++) {
        if(names[i] === ‘Zosia’) {
                continue;                    continue = skip iteration logic
        }
        ...do smt;
        if(names[i] === Kasia’) {
                break;                       break = break current for loop
        }
}
```

Loops
For loop pitfalls

There are few tricks about break. Consider this:

```
let names = [‘Andromeda’, ‘Cassandra’, ‘Cassiopeia’];

for(let i = 0; i < names.length; i++) {
        if(names[i] === ‘Andromeda’) {
                ...do smt;
                break; // otherwise unnecessary iteration are made
        }
}
```

Loops
Do-while loop

Do-while do something if condition is true. There is no explicit iteration. Do-while loop is not quite often too in modern web programming.

```
do {
        ..do smt;
} while (x > 5);
```

Pay attention to the fact, that first expression is executed and only then condition is checked.

Loops
While-do loop

While-do do something if condition is true. Same as do-while.
While-do loop is not quite often too in modern web programming.

```
while (x < 5) {
        ...do smt;
}
```

Pay attention to the fact, that first the condition is checked and only then expression is executed.

Loops
Do-while, while-do pitfalls

The most common problem about do-while and while-do is about first iteration:

```
do {
        ...do smt;
}  while( x > 5) // it will do first and only then checks the condition

while (x > 5) do {
        ...do smt; // if x is 5 or greater - this statement will never be executed
}
```

Loops
Do-while, while-do pitfalls

The second most common problem about do-while and while-do is about infinity:

```
do {
        ...do smt;
}  while(true) // it will be working endlessly
```

Task 7

1) Declare an array with numbers from 1 to 35. Use for for it
2) Now iterate over the array and for numbers from 1 to 15 log current number and index. Put console.log('Iteration', i) at the very end of the for body
3) For numbers 16 to 25 do nothing with continue.
4) For numbers from 25 to 30 check if it can be divided without rest and log those which can.
5) Stop iteration if the value equals to 31

Loops
Summary

- For loop is most common, though it gives its place to functional approach
- Do while and while do are used seldom in modern web
- Remember about correct breaking of iteration process
- Remember about do while first iteration

7.
The end