JS BASICS (DAY 2) conditions, strings, loops

Some of examples and definitions are from very good JS docs - https://developer.mozilla.org/ .

1.
Comparison and logical operators

Comparison and logical operators
Equality ==

== operator try to converts and compare the operands event if they are not of the same type. If both operands are objects, then JS checks if operands refer to the same object in memory.

Comparison and logical operators
Equality ==

```
1     ==  1           // true
'1'   ==  1           // true
1     == '1'          // true
0     == false        // true

var object1 = {'value': 'key'};
var object2 = {'value': 'key'};

object1 == object2 //false

NaN == NaN  // false - WTF ?? -> check that link
```

Comparison and logical operators
Inequality !=

Do opposite thing as equality (checks if operands are NOT equal).

Comparison and logical operators
Strict equality (===)

The strict equality operator returns true if the operands are strictly equal **with no type conversion**!

```
3 === 3   // true
3 === '3' // false
var object1 = {'value': 'key'};
var object2 = {'value': 'key'};
object1 === object2 //false
```

Comparison and logical operators
Strict inequality (!==)

Do opposite thing as strict equality (checks if operands are NOT equal).

Comparison and logical operators
Relational operators

Sometimes we want to check the relation between two operands. We can use one of these quite obvious operators:

<	less than operator
>	greater than or equal operator
<=	less than or equal operator
>=	greater than or equal operator

“ *Task 0*

Compare minimum 10 variables of different type and console.log the results.

Comparison and logical operators
OR, AND, negation

OR OPERATOR IS ||

AND OPERATOR IS &&

NEGATION OPERATOR IS !

Comparison and logical operators
OR, AND, negation

```
true || false // true

true && false // false

!true // false
```

Comparison and 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 thet rest of expression isn’t executed!

Comparison and logical operators
OR, AND, negation

Consider an example:

```
var a = 1

console.log(a || b) // b is not defined -
                       we should get an
                       error!
```

Comparison and logical operators
OR, AND, negation

OR and AND expressions returns the first truthy value that make the result obvious.

It can be used to conditionally assign values to variables!

```
var a = false || 1         // a is assigned 1
var a = 0 || 2             // a is assigned 2
var a = 0 || false || 3    // a is assigned 3
```

Comparison and logical operators
OR, AND, negation

Consider an example:

```
var array = ‘array’ // this is string!

var result = array && array.length

// array.length should throw an error
```

Task 1

Make a variable that is falsy. Make another variable and assign it first variable OR 1. Console.log result.

2.
Conditional statements and expressions

Conditional statements and expressions

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.

Conditional statements and expressions
Conditional statement

When we want to execute different code depending on some value or expression we can use conditional statement:

```
if (condition1)
    Statement1
```

OR

```
if (condition1){
    statement1
    statement2
}
```

Conditional statements and expressions
Conditional statement

We can combine multiple conditions:

```
if (condition1)
   statement1
else if (condition2)
   statement2
else if (condition3)
   statement3
...
else
   statementN
```

Task 2

“

Make a variable, then make if statement that logs:
1 if variable is equal to 1
2 if variable is equal to “1”
that variable if if variable is equal to 2
0 in other case

Conditional statements and expressions
Switch statement

When we want to execute different code depends on many cases we can use the switch statement:

```
switch(expression) {
    case n:
        code block
    case n:
        code block
        break;
    default:
        code block
}
```

Task 3

“

Make a variable, then make switch statement that logs:
1 if variable is equal to 1
2 if variable is equal to “1”
that variable if if variable is equal to 2
0 in other case

Conditional statements and expressions
Ternary operator (conditional expression)

When we want to use a condition as an expression (we need a value from it) we can’t do if statement because statements don’t returns a value.

We can use a ternary operator, which returns a value because it is an expression:

```
condition ? expr1 : expr2
```

Conditional statements and expressions
Ternary operator (conditional expression)

```
var expressionToCheck =   true;

console.log(expressionToCheck ? ‘This is true!’ : ‘This
is false!’);

var isThatTure = expressionToCheck ? ‘This is true!’ :
‘This is false!’;

console.log(isThatTure);
```

Conditional statements and expressions
Ternary operator (conditional expression)

We can use multiple ternary operators together:

```
var firstCheck = false;
var secondCheck = false;
    access = firstCheck ?
                'Access denied'
                :
                secondCheck ?
                    'Access denied'
                    :
                    'Access granted';
console.log(access); // logs "Access granted";
```

infoShare <academy/>

Task 4

“

Make a variable, then make ternary operator that RETURNS:
1 if variable is equal to 1
2 if variable is equal to “1”
that variable if if variable is equal to 2
0 in other case
Console.log what was returned.

3.
Loops

Loops
For loop

Loops offer a quick and easy way to do something repeatedly.

```
for ([initialExpression]; [condition]; [incrementExpression])
    statement
```

```
var numbers = [1, 2, 3, 4, 5];

for (var i = 0; i < 4; i++)
    console.log(numbers[i])


for (var i = 0; i < numbers.lenght; i++){
    console.log(numbers[i])
}
```

Loops

For loop - where I should put array.length?

```
console.time('in loop')
for (var i = 0; i < numbers.length; i++){
    console.log(numbers[i])
}
console.timeEnd('in loop')

console.time('before loop')
var length = numbers.length
for (var i = 0; i < length; i++){
    console.log(numbers[i])
}
console.timeEnd('before loop')
```

Task 5

Make an array with 10 random numbers (check in Google how to obtain random number)

Loops
while and do while

A **while** statement executes its statements as long as a specified condition evaluates to true.

```
while(condition)
     statement
```

```
var i = 0;
var a = 0;

while(i < 10){
     i++;
     a += 2;
}
```

Loops
while and do...while

The **do...while** statement repeats until a specified condition evaluates to false.

```
do
    statement
while (condition);
```

```
var i = 5;
do {
  i += 1;
  console.log(i);
} while (i < 5);
```

The do loop iterates at least once!

Loops
continue

The **continue** can be used to restart a **while, do-while** or **for** loop.

```
var i = 0;
var n = 0;
while (i < 5) {
  i++;
  if (i == 3) {
    continue;
  }
  n += i;
  console.log(n); // 1, 3, 7, 12
}
```

Loops
break

Use the **break** to terminate a loop or switch.

```
var numbers = [1,2,3,4,5];

for (var i = 0; i < numbers .length; i++) {
  if (numbers[i] ==+ 3) {
    break;
  }
  console.log(numbers[i]);
}
```

Task 6

Make a while loop that console.logs array of random items (from last task).

Break loop when item is > 1.

4.
Managing arrays

Arrays
Declarations

```
var a = [],               // these are the same arrays
    b = new Array(),      // length 0

    c = ['a', 'b'],                // these are the same arrays
    d = new Array('a', 'b'),   // c and d are arrays with 2
strings

    e = [3],                  // different arrays
    f = new Array(3)
;
```

WHY?

Arrays
Length

The **length** property of an array sets or returns the number of elements in that array.

```
var array = new Array(4);

array.length // 3

array.length = 2;

console.log(array);
```

Arrays
Iterating over an array

```
var numbers = [1, 2, 3, 4, 5];

var length = numbers.length;

for (var i = 0; i < length; i++) {
  console.log(numbers[i]);
}
```

Task 7

Make an array with 100 random numbers and console.log its items.

Arrays
pop()

The **pop()** method removes the last element from an array and returns that element. This method changes the length of the array.

```
var a = [1, 2, 3];
a.pop();

console.log(a); // [1, 2]
```

Arrays
push()

The **push()** method adds one or more elements to the end of an array and returns the new length of the array.

```
var numbers = [1, 2, 3];
numbers.push(4);

console.log(numbers); // [1, 2, 3, 4]

numbers.push(5, 6, 7);

console.log(numbers); // [1, 2, 3, 4, 5, 6, 7]
```

Arrays
shift()

The **shift()** method removes the first element from an array and returns that element. This method changes the length of the array.

```
var a = [1, 2, 3];
var b = a.shift();

console.log(a); // [2, 3]
console.log(b); // 1
```

Arrays
unshift()

The **unshift()** method adds one or more elements to the beginning of an array and returns the new length of the array.

```
var a = [1, 2, 3];
a.unshift(4, 5);

console.log(a); // [4, 5, 1, 2, 3]
```

“ **Task 8**

Make an array with 5 first positive integers.
Add 0 as a first element.
Add 6 as last element.
Console.log array.
Undo last two operations.
Console.log array.

Arrays
slice()

The **slice()** method returns **a shallow copy** of a portion of an array into a new array object selected from begin to end **(end not included)**. The original array will not be modified.

```
var a = ['zero', 'one', 'two', 'three'];
var sliced = a.slice(1, 3);

console.log(a);      // ['zero', 'one', 'two', 'three']
console.log(sliced); // ['one', 'two']
```

Arrays
slice()

```
var array = ['zero', 'one', 'two', 'three'];
var copy = a.slice(); // only a shallow copy !
```

A negative index can be used, indicating an offset from the end of the sequence. **slice(2,-1)** extracts the third element through the second-to-last element in the sequence.

```
array.slice(1,-1) // ["one", "two"]
```

If end **is omitted** OR end **is greater than the length** of the array, slice extracts through **the end of the array**.

Task 4

“ *Make an array with random length and random values.*
Make new array that contains middle item & 2 items before & 2 items after middle item from last array.
If there is no middle item do not include it in new array.

Arrays
indexOf()

The indexOf() method returns the first index at which a given element can be found in the array, or -1 if it is not present.

```
arr.indexOf(searchElement[, fromIndex])

var array = [2, 9, 9];
array.indexOf(2);     // 0
array.indexOf(7);     // -1
array.indexOf(9);     // 1
array.indexOf(9, 2);  // 2
array.indexOf(2, -1); // -1
array.indexOf(2, -3); // 0
```

Task 5

Find 9 in [1,2,3,4,5,6,7,8,9,10] array.

JS Koans
What is koan?

It is a story, dialogue, question, or statement, which is used in Zen practice to provoke the "great doubt" and test a student's progress in Zen practice.

“ ***Task 6***

Practice JS Koans.
(AboutExpects.js)