[Lesson 3]

Roi Yehoshua 2018

[DAN.IT EDUCATION]

What we learnt last time?

- JavaScript basic data types
- How primitive types in Javascript work when combined together and with each other
- Type casting when working with different types

[Our targets for today]

- JavaScript general operators
- Logic operators
- If-else construction
- Switch statement

[DAN.IT EDUCATION]

Operators

→ An **operand** – is what operators are applied to
→ For example, in multiplication 5 * 2 there are two operands: the left operand is 5,
→ and the right operand is 2.
→ An operator is unary if it has a single operand
  → For example, the unary negation - reverses the sign of the number:

```
let x = 1;

x = -x;
alert(x); // -1, unary negation was applied
```

→ An operator is binary if it has two operands.
→ The same minus exists in the binary form as well:

```
let x = 1, y = 3;
alert(y - x); // 2, binary minus subtracts values
```

[DAN.IT] EDUCATION

String Concatenation

→ Usually the plus operator + sums numbers
→ But if the binary + is applied to strings, it concatenates them:

```
let s = "my" + "string";  alert(s); // mystring
```

→ If any of the operands is a string, then the other one is converted to a string too:

```
alert('1' +  2); // "12"
alert(2 +  '1'); // "21"
```

→ However, operations run from left to right. If there are two numbers followed by a string, the numbers will be added before being converted to a string:

```
alert(2 + 2 + '1'); // "41" and not "221"
```

[DAN.IT] EDUCATION

Integer Division and Remainder %

→ The division operator a / b produces the exact quotient of its operands

→ Integer division can be achieved by applying **parseInt()** on the quotient

```
alert(5 / 2) // 2.5
alert(parseInt(5 / 2)) // 2
```

→ The result of a % b is the remainder of the integer division of a by b

→ For instance:

```
alert(5 % 2); // 1  is a  remainder  of  5  divided  by 2
alert(8 % 3); // 2  is a  remainder  of  8  divided  by 3
alert(6 % 3); // 0  is a  remainder  of  6  divided  by 3
```

Exponentiation **

→ The exponentiation operator ** is a recent addition to the language (ES6)
→ For a natural number b, the result of a ** b is a multiplied by itself b times

```
alert(2  ** 2); // 4   (2 * 2)
alert(2  ** 3); // 8   (2 * 2 * 2)
alert(2 ** 4); // 16 (2 * 2 * 2 * 2)
```

→ The operator works for non-integer numbers of a and b as well, for instance:

```
alert(4 ** (1 / 2)); // 2 (power of 1/2 is the same as a square root)
alert(8 ** (1 / 3)); // 2 (power of 1/3 is the same as a cubic root)
```

Operators Precedence

→ Operator precedence determines the way in which operators are parsed with respect to each other

→ Operators with higher precedence become the operands of operators with lower precedence

→ Parentheses override any precedence

Level	Operators	Description	Associativity
15	() [] . new	Function Call Array Subscript Object Property Access Memory Allocation	Left to Right
14	++ -- + - ! ~ delete typeof void	Increment / Decrement Unary plus / minus Logical negation / bitwise complement Deallocation Find type of variable	Right to Left
13	* / %	Multiplication Division Modulo	Left to Right
12	+ -	Addition / Subtraction	Left to Right
11	>> <<	Bitwise Right Shift Bitwise Left Shift	Left to Right
10	< <= > >=	Relational Less Than / Less than Equal To Relational Greater / Greater than Equal To	Left to Right
9	== != === !==	Equality Inequality Identity Operator Non Identity Operator	Left to Right
8	&	Bitwise AND	Left to Right
7	^	Bitwise XOR	Left to Right
6	|	Bitwise OR	Left to Right
5	&&	Logical AND	Left to Right
4	||	Logical OR	Left to Right
3	?:	Conditional Operator	Right to Left
2	= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operators	Right to Left
1	,	Comma Operator	Left to Right

[DAN.IT] EDUCATION

Assignment =

- → An assignment = is also an operator
- → It is listed in the precedence table with the very low priority of 3
- → That's why when we assign a variable, like x = 2 * 2 + 1, then the calculations are done first, and afterwards the = is evaluated, storing the result in x
- → Every operator returns a value, including the assignment operator
- → The call x = value writes the value into x *and then returns it*

```
let a = 1;  let b = 2;
let c = 3 - (a = b + 1);

alert(a); // 3
alert(c); // 0
```

- → The result of (a = b + 1) is the value which is assigned to a (that is 3). It is then used to subtract from 3.

[DAN.IT] EDUCATION

Assignment =

→ It is possible to chain assignments:

```
let a, b, c;
a = b = c = 2 + 2;

alert(a); // 4
alert(b); // 4
alert(c); // 4
```

→ Chained assignments evaluate from right to left

→ First the rightmost expression 2 + 2 is evaluated then assigned to the variables on the left: c, b and a

→ At the end, all variables share a single value

[DAN.IT] EDUCATION

Increment/Decrement

→ Increasing or decreasing a number by one is among the most common numerical operations

→ So, there are special operators for that:

→ **Increment** ++ increases a variable by 1:

```
let counter = 2;
counter++; // works the same as counter = counter + 1, but is shorter
alert(counter); // 3
```

→ **Decrement** -- decreases a variable by 1:

```
let counter = 2;
counter--; // works the same as counter = counter - 1, but is shorter
alert(counter); // 1
```

→ Increment/decrement can be applied only to a variable

→ An attempt to use it on a value like 5++ will give an error

[DAN.IT] EDUCATION

Increment/Decrement

→ Operators ++ and -- can be placed both after and before the variable

→ When the operator goes after the variable, it is called a “postfix form”: counter++.

→ When it goes before the variable, it is called a “prefix form”: ++counter

→ Both of these records do the same: increase counter by 1

→ Is there any difference? Yes, but we can only see it if we use the returned value of ++/--

→ The prefix form returns the new value, while the postfix form returns the old value (prior to increment/decrement)

→ To see the difference, here’s the example:

```
let counter = 1;
let a = ++counter; // prefix increment

alert(a); // 2
```

```
let counter = 1;
let a = counter++; // postfix increment

alert(a); // 1
```

Modify-in-place

→ We often need to apply an operator to a variable and store the new result in it
→ For example:

```
let n = 2;
n = n + 5;
n = n * 2;
```

→ This notation can be shortened using operators += and *=:

```
let n = 2;
n += 5; // now n = 7 (same as n = n + 5)
n *= 2; // now n = 14 (same as n = n * 2)
alert(n); // 14
```

Short “modify-and-assign” operators exist for all arithmetic and bitwise operators: /=, -= etc.

Such operators have the same precedence as a normal assignment, so they run after most other calculations

Comparisons

→ Many comparison operators we know from maths:

→ Greater/less than: a > b, a < b.

→ Greater/less than or equals: a >= b, a <= b.

→ Equality check is written as a == b

→ Please note the double equation sign ==

→ A single symbol a = b would mean an assignment

→ Not equals: In maths the notation is ≠, in JavaScript it's written as an assignment with an exclamation sign before it: a != b

→ Just as all other operators, a comparison returns a value

→ The value is of the boolean type:

→ true – means "yes" or "correct"

→ false – means "no" or "wrong"

Comparisons

→ For example:

```
alert(2 >  1);   // true (correct)
alert(2 == 1); // false (wrong)
alert(2 != 1); // true (correct)
```

→ A comparison result can be assigned to a variable, just like any value:

```
let result = 5 > 4; // assign the result of the comparison
alert(result); // true
```

String Comparison

→ To see which string is greater than the other, the so-called “dictionary” or “lexicographical” order is used

→ In other words, strings are compared letter-by-letter.

→ For example:

```
alert('Z' > 'A'); // true
alert('Glow' > 'Glee'); // true
alert('Bee' > 'Be'); // true
```

→ Note that case matters. A capital letter "A" is not equal to the lowercase "a".

→ Which one is greater? Actually, the lowercase "a" is. Why? Because the lowercase character has a greater index in the internal encoding table (Unicode)

→ You can find the table here: https://www.rapidtables.com/code/text/unicode-characters.html

[DAN.IT] EDUCATION

Comparison of Different Types

→ When comparing values that belong to different types, they are converted to numbers:

```
alert('2' > 1); // true, string '2' becomes a number 2
alert('01' == 1); // true, string '01' becomes a number 1
```

→ For boolean values, true becomes 1 and false becomes 0:

```
alert(true == 1); // true
alert(false == 0); // true
```

→ An empty string converts to 0

→ A non-numeric string converts to NaN which is always false

Strict Equality

→ A regular equality check == has a problem: it cannot differ 0 or empty string from false

```
alert(false == 0); // true
alert('' == false); // true
```

→ That's because operands of different types are converted to a number by the equality operator

→ An empty string, just like false, becomes a zero.

→ What to do if we'd like to differentiate 0 from false?

→ **A strict equality operator ===** checks the equality without type conversion

→ If a and b are of different types, then a === b immediately returns false without an attempt to convert them

```
alert(0 === false); // false, because the types are different
```

→ There also exists a "strict non-equality" operator !==, as an analogy for !=

[DAN.IT] EDUCATION

Comparison with null and undefined

→ There's a non-intuitive behavior when null or undefined are compared with other values

→ For a strict equality check === these values are different

→ because each of them belongs to a separate type of its own

```
alert(null === undefined); // false
```

→ For a non-strict check == there's a special rule:

→ These two are a "sweet couple": they equal each other (in the sense of ==), but not any other value

```
alert(null == undefined); // true
```

→ For maths and other comparisons < > <= >= values null/undefined are converted to a number: null becomes 0, while undefined becomes NaN

→ NaN is a special numeric value which returns false for all comparisons

[DAN.IT] EDUCATION

Conditions

→ Sometimes we need to perform different actions based on a condition

→ The **if** statement gets a condition, evaluates it and, if the result is true, executes the code

```
let num = prompt('Please enter a number');
if (num % 2 == 0) alert('The number is even');
```

→ If there is more than one statement to be executed if the condition holds, we have to wrap our code block inside curly braces:

```
let num = prompt('Please enter a number');
if (num % 2 == 0) {
  alert('The number is even');
  alert('Have fun');
}
```

→ It is recommended to wrap your code block with curly braces {} every time with if, even if there is only one statement. This improves readability.

[DAN.IT] EDUCATION

Boolean Conversion

→ The if (…) statement evaluates the expression in parentheses and converts it to the boolean type

→ A number 0, an empty string "", null, undefined and NaN become false

→ Other values become true,

→ So, the code under this condition would never execute:

```
if (0) { // 0 is falsy
    ...
}
```

→ And inside this condition – always works:

```
if (x = 5) { // the expression x = 5 has the value of 5 which is truthy
    ...
}
```

→ Always use == inside conditions!

[DAN.IT] EDUCATION

The “else” Clause

→ The if statement may contain an optional “else” block

→ It executes when the condition is wrong

→ For example:

```
let num = prompt('Please enter a number');
if (num % 2 == 0) {
  alert('The number is even');
} else {
  alert('The number is odd');
}
```

Several Conditions: else if

→ Sometimes we'd like to test several variants of a condition. There is an else if clause for that.

→ For example:

```
let num = prompt('Please enter a number');
if (num > 0) {
   alert('The number is positive');
} else if (num < 0) {
   alert('The number is negative');
} else {
   alert('The number is zero');
}
```

Ternary Operator '?'

→ Sometimes we need to assign a variable depending on a condition, e.g.,

```
let accessAllowed;
let age = prompt('How old are you? ');
if (age > 18) {
    accessAllowed = true;
} else {
    accessAllowed = false;
}
alert(accessAllowed);
```

→ The "ternary" or "question mark" operator lets us do that shorter and simpler

```
let result = condition ? value1 : value2
```

→ The condition is evaluated, if it's truthy then value1 is returned, otherwise – value2.

→ For example:

```
let accessAllowed = age > 18 ? true : false;
```

→ In this example, we could also have written

```
let accessAllowed = age > 18;
```

Logical Operators

→ There are three logical operators in JavaScript: || (OR), && (AND), ! (NOT).

→ Although they are called “logical”, they can be applied to values of any type, not only boolean

→ The “OR” operator is represented with two vertical line symbols

→ If any of its arguments are true, then it returns true, otherwise it returns false

```
let hour = 9;

if (hour < 10 || hour > 18) {
    alert('The office is closed.');
}
```

→ If an operand is not boolean, then it’s converted to boolean for the evaluation:

```
if (1 || 0) { // works just like if(true || false)
    alert('truthy!');
}
```

[DAN.IT] EDUCATION

Short-Circuit Evaluation

→ OR evaluates and tests its operands from left to right

→ It returns the first truthy value or the last value if none were found

```
alert(1 || 0); // 1 (1 is truthy)
alert(true || 'no matter what'); // (true is truthy)  alert(null || 1); // 1
(1 is the first truthy value)  alert(null || 0 || 1); // 1 (the first truthy
value)
alert(undefined || null || 0); // 0 (all falsy, returns the last value)
```

→ The evaluation of operands stops when a truthy value is reached:

```
let x;
true || (x = 1);
alert(x); // undefined, because (x = 1) not evaluated
```

→ This process is called “a short-circuit evaluation”, because it goes as short as possible from left to right

[DAN.IT] EDUCATION

AND Operator

→ The AND operator is represented with two ampersands &&

→ AND returns true if both operands are truthy and false otherwise:

```
let hour = 12;
let minute = 30;

if (hour == 12 && minute == 30) {
  alert('Time is 12:30');
}
```

→ Just as for OR, any value is allowed as an operand of AND:

```
if (1 && 0) { // evaluated as true && false
    alert("won't work, because the result is falsy");
}
```

AND Operator

→ AND returns the first falsy value or the last value if none were found

```
// if the first operand is truthy, AND returns the second operand:
alert(1 && 0); // 0
alert(1 &&    5); // 5

// if the first operand is falsy, AND returns it.
// The second operand is ignored
alert(null && 5); // null
alert(0 && "no matter what"); // 0
```

→ AND && executes before OR ||

```
alert(5 || 1 && 0); // 5
```

NOT Operator

→ The boolean NOT operator is represented with an exclamation sign !

→ The operator accepts a single argument and does the following:

→ Converts the operand to boolean type: true/false

→ Returns an inverse value

```
alert(!true); // false
alert(!0); // true
```

→ NOT has higher precedence than the AND and OR operators

```
alert(!5 || 1); // 1
alert(!(5 || 1)); // false
```

The switch statement

- → A switch statement can replace multiple if checks
- → It gives a more descriptive way to compare a value with multiple variants
- → The switch has one or more case blocks and an optional default
- → It looks like this:

```
                        switch(x) {
    case 'value1':    // if (x === 'value1')
    ...
    [break]

    case 'value2':    // if (x === 'value2')
    ...
    [break]

    default:
    ...
    [break]
}
```

- → The value of x is checked for a strict equality to the value from the first case (value1) then to the second (value2) and so on
- → If the equality is found, switch starts to execute the code starting from the corresponding case, until the nearest break (or until the end of switch)
- → If no case is matched then the default code is executed (if it exists)

[DAN.IT] EDUCATION

The switch statement - example

```
let a = 2 + 2;

switch (a) {
                        case 3:
        alert('Too small');
        break;
                        case 4:
        alert('Exactly!');  break;
                        case 5:
        alert('Too large');
        break;
    default:
        alert("I don't know such values");
}
```

→ Here the switch starts to compare *a* from the first case variant that is 3
→ The match fails
→ Then 4. That's a match, so the execution starts from case 4 until the nearest break.

The switch statement

→ If there is no break then the execution continues with the next case without any checks

```
let a = 2 + 2;

switch (a) {
        case 3:
                alert('Too small');  case 4:
                alert('Exactly!');  case 5:
                alert('Too big');  default:
                alert("I don't know such values");
}
```

→ In the example above we'll see sequential execution of three alerts:

- → alert('Exactly!');
- → alert('Too big');
- → alert("I don't know such values");

The switch statement

→ Any expression can be a switch/case argument

→ For example:

```
let a = "1";  let b = 0;

switch (+a) {
        case b + 1:
                alert("this runs, because +a is 1, exactly equals b+1");  break;

        default:
                alert("this doesn't run");
}
```

Grouping of Cases

→ Several variants of case which share the same code can be grouped.

→ For example, if we want the same code to run for case 3 and case 5:

```
let a = 2 + 2;

switch (a) {
    case 4:
        alert('Right!');
        break;

    case 3:          // (*) grouped two cases
    case 5:
        alert('Wrong!');
        alert("Why don't you take a math class?");
        break;

    default:
        alert('The result is strange. Really.');
}
```

[DAN.IT EDUCATION]

Control questions

1. What is operator and what is operand?
2. What is the difference between concatenation and addition?
3. How comparison of different data types works?
4. How does if-else construction work?
5. How does switch operator work?

Materials

Core materials:
https://learn.javascript.ru/operators
https://learn.javascript.ru/comparison
https://learn.javascript.ru/ifelse
https://learn.javascript.ru/logical-ops
https://learn.javascript.ru/switch

Additional materials:
https://learn.javascript.ru/bitwise-operators
https://dorey.github.io/JavaScript-Equality-Table/

Video materials:
https://www.youtube.com/watch?v=FeRO2x-FXzc
https://youtu.be/S2Y2i6g7IZE

[DAN.IT] EDUCATION