Intro to JS

UA Resource Development Unit

2021

AGENDA

1. Connection
2. Structure
3. Data Types
4. Operators
5. Loops

ECMA-262

https://www.ecma-international.org/publications-and-standards/standards/ecma-262/

JavaScript

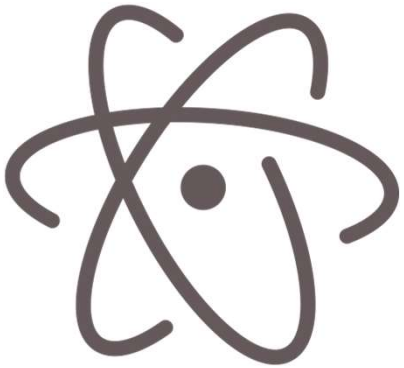
- **JavaScript** is a programming language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else.

https://developer.mozilla.org/en/
http://kangax.github.io/compat-table/es6/
http://learn.javascript.ru/

JavaScript Editors: What to Look For

- Strong ES2015+ support
  - Autocompletion
  - Parse ES6 imports
  - Report unused imports
  - Automated refactoring
- Framework intelligence
- Built-in terminal

Editors and configuration



- Atom

- WebStorm

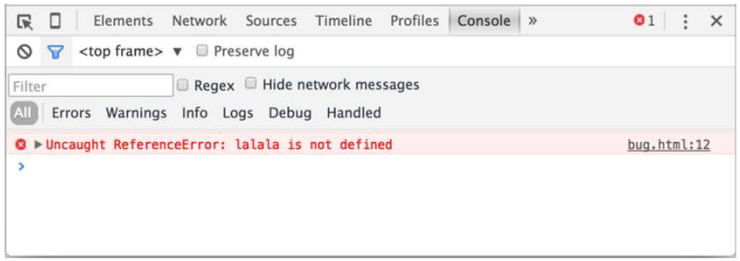
- VSCode

- Brackets

Developer console

Google Chrome

Press F12 or, if you're on Mac, then Cmd+Opt+J.



Firefox, Edge and others

Most other browsers use F12 to open developer tools.

<script> tag

JavaScript programs can be inserted in any part of an HTML document with the help of the `<script>` tag.

```
<script>
  document.getElementById("demo").innerHTML = "My
  First JavaScript";
</script>
```

In tag

```
<a href="delete.js" onclick="return confirm("Delete");">
        delete
    </a>
```

External scripts

```
<script src="/js/script1.js"></script>

<script src="/js/script2.js"></script>
```

External JavaScript Advantages

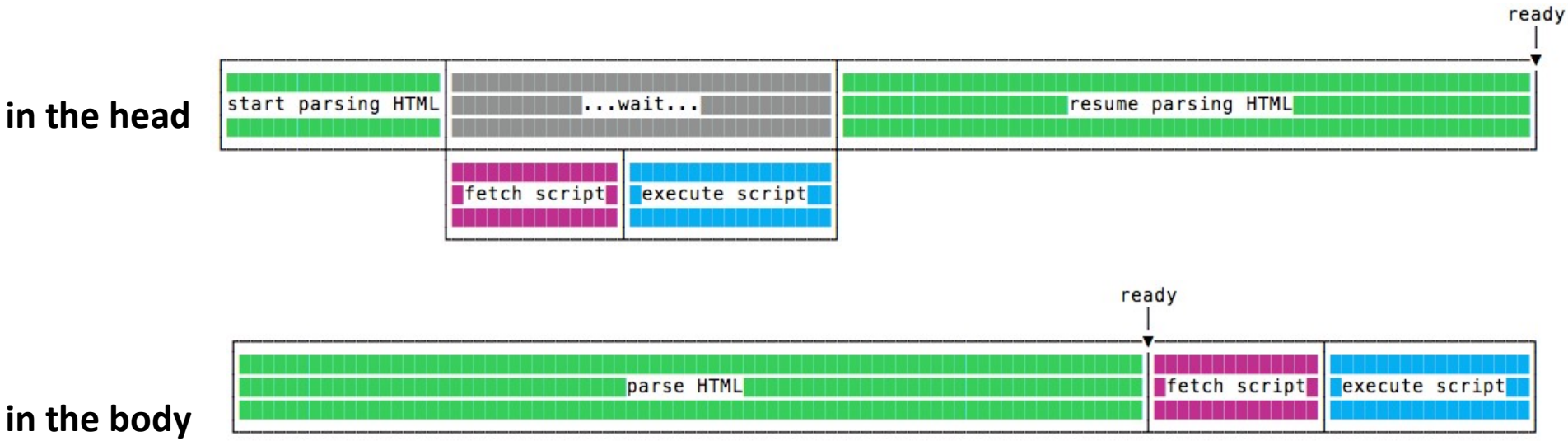
It separates HTML and code

It makes HTML and JavaScript easier to read and maintain

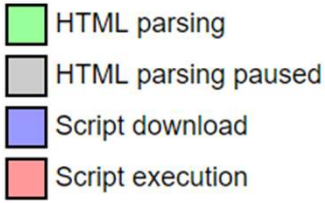
Cached JavaScript files can speed up page loads

<script>

<script> without any attributes does. The HTML file will be parsed until the script file is hit, at that point parsing will stop and a request will be made to fetch the file (if it's external). The script will then be executed before parsing is resumed

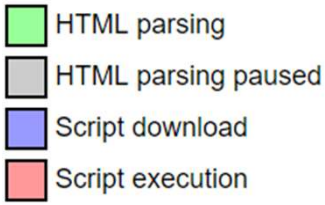


<script async>



async downloads the file during HTML parsing and will pause the HTML parser to execute it when it has finished downloading.

<script defer>



defer downloads the file during HTML parsing and will only execute it after the parser has completed. defer scripts are also guarenteed to execute in the order that they appear in the document.

When should use it

If the script is **modular** and does **not rely** on any scripts then use **async**.

If the script **relies** upon or is relied upon by another script then use **defer**.

If the script is **small** and is **relied** upon by an **async** script then use an inline script with **no attributes** placed above the async scripts.

defer/async

```
<script src="1.js" async></script>
<script src="2.js" async></script>
```

```
<script src="1.js" defer></script>
<script src="2.js" defer></script>
```

```
<script src="1.js" async></script>
<script src="2.js" defer></script>
```

"use strict"

```
"use strict";
var v = "Hello!";

function strict() {
'use strict';
  function nested() {
  return "And so am I!";
  }
return "Hi! I'm a strict mode function! " + nested();
}

function notStrict() { return "I'm not strict."; }
```

Keywords and reserved words

Names that have a special meaning, such as var, while, and for may not be used as variable names. These are called keywords. There are also a number of words which are 'reserved for use' in future versions of JavaScript.

break case catch continue debugger default delete

do else false finally for function if implements

in instanceof interface let new null package private

protected public return static switch throw true

try typeof var void while with yield this

Expressions and statements

A fragment of code that produces a value is called an expression. If an **expression** corresponds to a sentence fragment, a **statement**, in JavaScript, corresponds to a full sentence in a human language. A program is simply a list of statements.

```
// expression
2-1
true
// statement
var a = 5 + 8;
!false;
```

definition

A variable is a named container for a ***value***

The ***name*** that refers to a variable is sometime called *an* ***identifier***

Declaration, Initialization and Assignment

DECLARATION

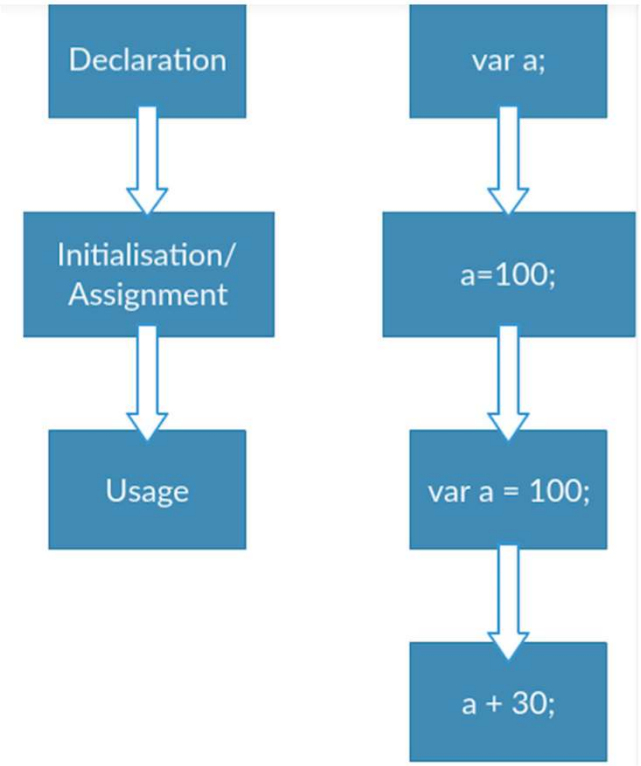
Registers a variable in the corresponding scope

INITIALIZATION

Allocates memory for the variable

ASSIGNMENT

Assigns a specified value to the variable



Declaration Types

var Declares a variable, optionally initializing it to a value

let Declares a block-scoped, local variable, optionally initializing it to a value

const Declares a block-scoped, read-only named constant

var

```
var x; // Declaration and initialization
x = "Hello World"; // Assignment
var y = "Hello World"; // Or all in one
```

Variables declared with var are available in the scope of the enclosing function. If there is no enclosing function, they are available globally.

```
function sayHello(){
  var hello = "Hello World";
  return hello;
}
console.log(hello);
```

This will cause an error `ReferenceError: hello is not defined`, as the variable `hello` is only available within the function `sayHello`.

let

```
let x; // Declaration and initialization
x = "Hello World"; // Assignment

// Or all in one
let y = "Hello World";
```

let is the descendant of var in modern JavaScript. Its scope is not only limited to the enclosing function, but also to its enclosing block statement. A block statement is everything inside { and }, (e.g. an if condition or loop). The benefit of let is it reduces the possibility of errors, as variables are only available within a smaller scope.

```
var name = "Peter";
if(name === "Peter"){
  let hello = "Hello Peter";
} else {
  let hello = "Hi";
}
console.log(hello);
```

This will cause an error `ReferenceError: hello is not defined` as `hello` is only available inside the enclosing block – in this case the `if` condition.

const

```
const x = "Hello World";
```

Technically a constant isn't a variable. A const is limited to the scope of the enclosing block, like let.
Constants should be used whenever a value must not change during the applications running time, as you'll be notified by an error when trying to overwrite them.

```
function f() {}
const f = 5; // Uncaught SyntaxError:
Identifier 'f' has already been declared

function foo() {
  const g = 5;
  var g; // Uncaught SyntaxError:
Identifier 'g' has already been declared
}
```

A constant cannot change value through assignment or be re-declared while the script is running. It has to be initialized to a value.

Accidental Global Creation

can be written all of above named declarations in the global context (i.e. outside of any function), but even within a function, if var, let or const will not be written before an assignment, the variable will automatically be global.

```
function sayHello(){
  hello = "Hello World";
  return hello;
}
sayHello();
console.log(hello);
```

! To avoid accidentally declaring global variables you can use strict mode.

Hoisting and the Temporal Dead Zone

A variable declaration will always internally be hoisted (moved) to the top of the current scope.

```
console.log(hello);
var hello;
hello = "I'm a variable";
```

~

```
var hello;
hello = "I'm a variable";
console.log(hello);
```

Variable hoisting

```
console.log(x === undefined);  //true
var x = 3;

var myvar = "my value";
(function() {
   console.log(myvar); // undefined
   var myvar = "local value";
})();
```

Function hoisting

```
/* Function declaration */
foo(); // "bar"
function foo() {
 console.log('bar');
}
/* Function expression */
baz();                    // TypeError: baz is not a function
var baz = function() {
 console.log('bar2');
};
```

Variables

start with a letter, underscore (_), or dollar sign ($);
subsequent characters can also be digits (0-9).
ISO 8859-1 or Unicode letters

Number_hits
Temp99
$credit
and _name
userName

Reserved Words

Some keywords can not be used as variable names:

null true false break do instanceof typeof case else new var catch finally return void continue for switch while debugger function this with default if throw delete in try class enum extends super const export import implements let private public yield interface package protected static

Evaluating variables

A variable declared using the **var** or **let** statement with no assigned value specified has the value of undefined.

An attempt to access an undeclared variable will result in a ReferenceError exception being thrown

```
var a;
let b=5;
x=7;
var y = "Hello JS!";
var z;
z = false;
z = 101;
```

example

// What is i, $, p, and q afterwards?

```
var i, $, p, q;
i = -1;
for (i = 0; i < 10; i += 1) {
   $ = -i;
}
if (true) {
   p = 'FOO';
} else {
   q = 'BAR';
}
```

When the program runs, all variable declarations are moved up ***to the top of the current scope***

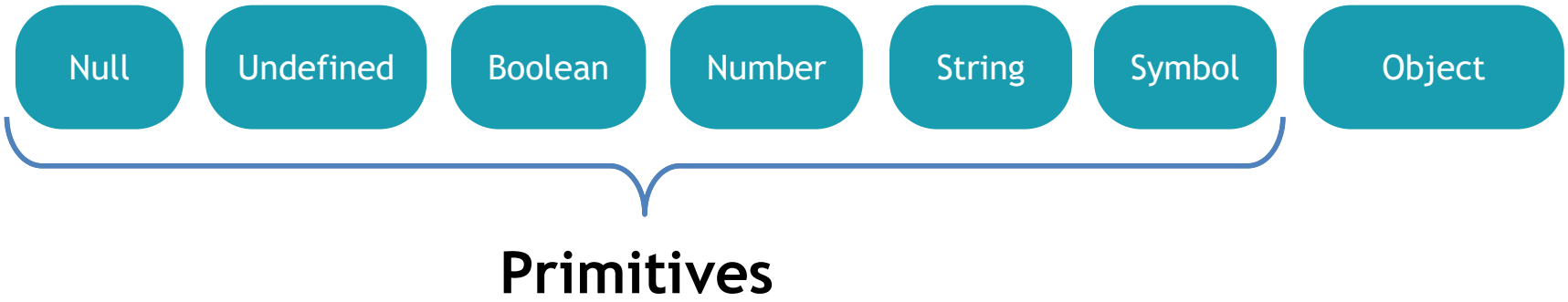
Variable scope

```
if (true) {
  var x = 5;
}
console.log(x);       //x=?
if (true) {
  let y = 5;
}
console.log(y);       //y=?
```

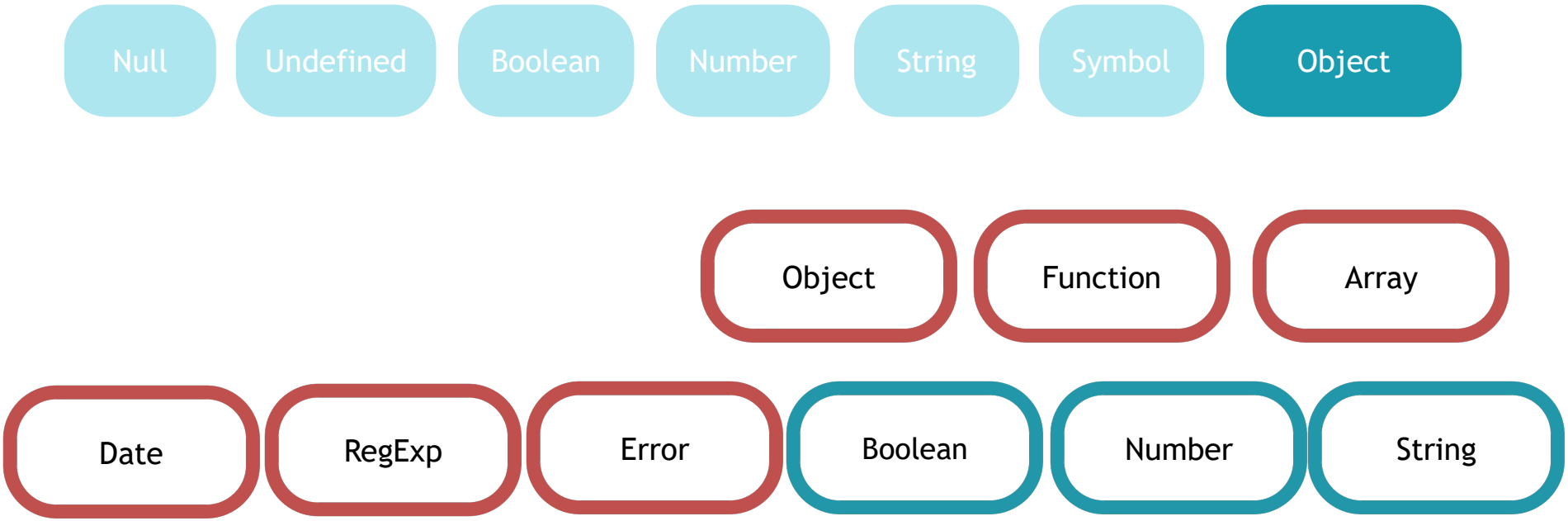
6 types of *values* in JavaScript (ES5)

- Null
- Undefined
- Boolean
- Number
- String
- Object

7 types of *values* in JavaScript (ES6)



7 types of *values* in JavaScript (ES6)



Since there are ***wrapper objects*** for the 3 types of primitive values,
a Boolean/Number/String primitive value can ***behave like an object***

Primitive value

Null	null				
Undefined	undefined				
Boolean	true	false			
String	“”	“\n”	‘Hello’	“אבג”	
Number	01101	.45	-Infinity	NaN	5e-14

number

```
var n = 536;
n = 3.1415;
n = 0.1 + 0.2;   //!=0.3
```

`Infinity` represents the mathematical Infinity `∞`. It is a special value that's greater than any number.

NaN

- Division of zero by zero
- Dividing an infinity by an infinity
- Multiplication of an infinity by a zero
- Any operation in which NaN is an operand
- Converting a non-numeric string or undefined into a number

NaN is unordered

```
NaN < 1;     // false
NaN > 1;     // false
NaN == NaN; // false
// But we can still check for NaN:
isNaN(NaN); // true
isNaN(true); // false
isNaN(false); // false
```

To Number

parseInt(*string*, *radix*);

```
parseInt(" 0xF", 16);
parseInt(" F", 16);
parseInt("17", 8);
parseInt(021, 8);
parseInt("015", 10);
parseInt(15.99, 10);
parseInt("1111", 2);
parseInt("15*3", 10);
parseInt("12", 13);
```

Number(object)

```
Number('12.3')
Number('')
Number('0x11')
Number('0b11')
Number('0o11')
```

parseFloat(*value*)

```
parseFloat(3.14);
parseFloat('3.14');
parseFloat('314e-2');
parseFloat('0.0314E+2');
parseFloat('3.14more');
```

toString

```
var n = 255;
alert( n.toString(16) );

2.toString(); // SyntaxError

2..toString(); // the second point is correctly recognized
2 .toString(); // note the space left to the dot
(2).toString(); // 2 is evaluated first
```

boolean

type consisting of the primitive values true and false

```
var amIAlwaysRight = true;
var areYouAlwaysRight = false;
```

undefined

primitive value used when a variable has not been assigned a value

```
var foo;
console.log(foo);
console.log(window.bar);
console.log(bar);
```

null

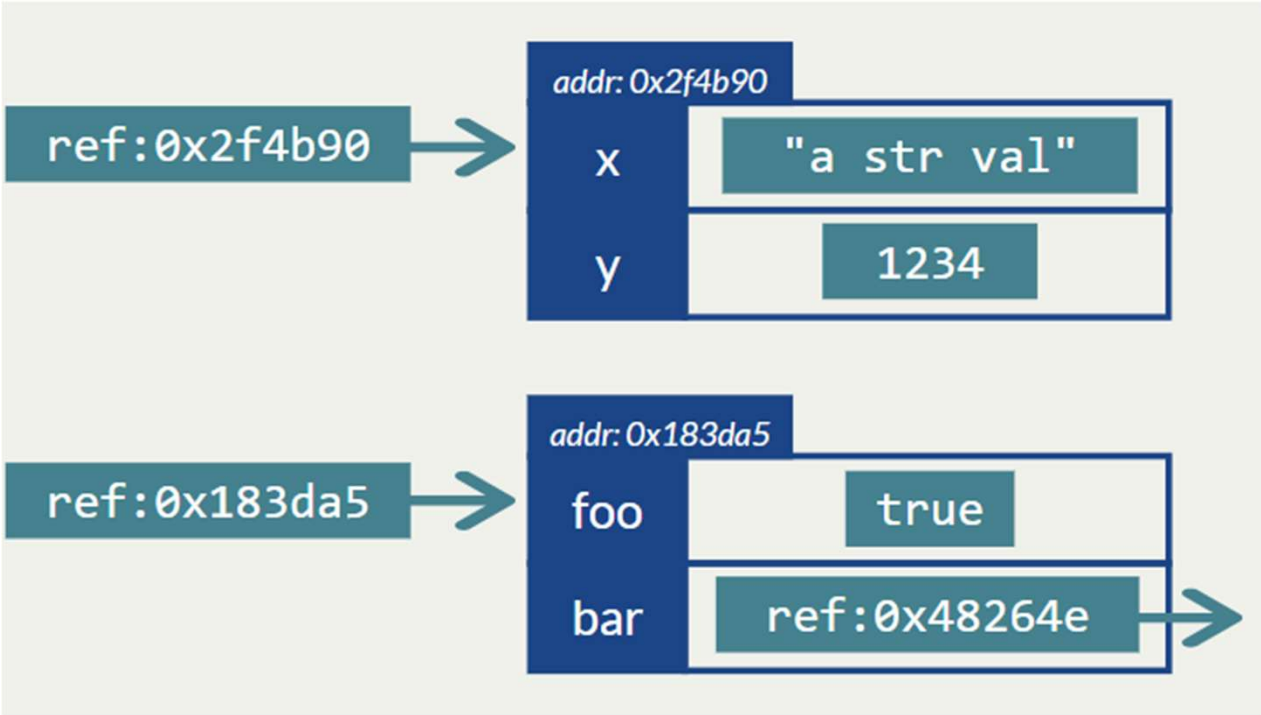
primitive value that represents the intentional absence of any object value

```
var age = null;
```

Symbol

```
let sym = Symbol();
var s=Symbol('name');
console.log(typeof s);
console.log(Symbol('name')==Symbol('name'));
```

object



Any value of this type is ***a reference to some "object"***; sometimes we would simply call such value ***an object***

Data type conversion

Converting to strings
Converting to numbers
Converting to boolean

ToBoolean Conversions

it happens in logical operations, but also can be performed manually with the call of Boolean(value)

Argument Type	Result
Undefined	false
Null	false
Boolean	The result equals the input argument (no conversion).
Number	The result is false if the argument is +0, −0, or NaN; otherwise the result is true.
String	The result is false if the argument is the empty String (its length is zero); otherwise the result is true.
Object	true

ToBoolean

JAVASCRIPT

```
console.log( !!"0" );                    //true
console.log( !!" " );                    // true
console.log( 0 == "\n0\n" );             //true
console.log( !!undefined );              //false
console.log( !!null );                   // false
console.log( !!"" );                     // false
console.log( !!NaN );                    //false
console.log( !!{} );                     // true
console.log( !![] );                     // true
```

ToNumber Conversions

numeric conversion happens in mathematical functions and expressions automatically

Argument Type	Result
Undefined	**NaN**
Null	**+0**
Boolean	The result is **1** if the argument is **true**. The result is **+0** if the argument is **false**.
Number	The result equals the input argument (no conversion).
String	See grammar and note below.
Object	Apply the following steps: 1.Let *primValue* be ToPrimitive(*input argument*, hint Number). 2.Return ToNumber(*primValue*).

ToNumber

JAVASCRIPT

```
let a = +'123';
console.log( +'   \n  123   \n  \n' );      // 123
console.log( +true );                        // 1
console.log( +false );                       // 0
console.log( '\n0 ' == 0 );                  // true
console.log( "\n" == false );                // true
console.log( "1" == true );                  // true
```

ToString Conversions

string conversion happens when we need the string form of a value

Argument Type	Result
Undefined	"undefined"
Null	"null"
Boolean	If the argument is **true**, then the result is "true". If the argument is **false**, then the result is `"false"`.
Number	If *m* is **NaN**, return the String "NaN". If *m* is **+0** or **−0**, return the String "0". If *m* is less than zero, return the String concatenation of the String "-" and ToString(−*m*). If *m* is infinity, return the String "Infinity".
String	Return the input argument (no conversion)
Object	Apply the following steps: 1. Let *primValue* be ToPrimitive(input argument, hint String). 2. Return ToString(*primValue*).

ToString

JAVASCRIPT

```
console.log( true + "test" );          // truetest
console.log( "123" + undefined );      //123undefined
console.log( "123" + {});              //123[object Object]
console.log( 123 + 123);               //246
console.log( '123' + 123);             //123123
console.log( [] + 1);                  //1
console.log( [1] + 1);                 //11
console.log( [1, 2] + 1);              //1,21
console.log( '\n' === false);          //false
console.log( '\n' == false);           //true
console.log('Hi' == false);            //false
```

Special values

JAVASCRIPT

```
console.log(null >= 0);          // true
console.log(null >  0);          // false
console.log(null == 0);          // false
console.log(null >  0);          // false
console.log(undefined == 0);     // false
console.log(undefined <0);       // false
console.log(NaN == NaN);         // false
console.log(NaN === NaN);        // false
console.log(undefined == null);  //true
```

typeof

Type	
undefined	**undefined**
null	**object**
true	**boolean**
new Boolean(true)	**object**
5	**number**
new Number(5)	**object**
“foo”	**string**
new String(“foo”)	**object**
[1, 2, 3]	**object**
function foo() {}	**function**

Operators

Arithmetic operators
Assignment operators
Logical operators
Comparison operators
Bitwise operators
Bitwise logical operators
Bitwise shift operators

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (remainder of a division)
++	Increment
--	Decrement

Examples

```
console.log(100 + 4 * 11 / 2 - 1);        //121

var i = 20;
console.log(++i);                          // 21
console.log(i);                            // 21
console.log(i--);                          // 21
console.log(i);                            // 20

var x = 25;
x += 20;                                   //(x = x + 20);
console.log(x);                            //45
console.log(5 / 0);                        // Infinity
console.log(0 / 0);                        // NaN

typeof(Infinity)                           // 'number'
typeof(NaN)                                // 'number'
```

Assignment Operators

Operator	Description
=	Assign
+=	Add and assign. For example, x+=y is the same as x=x+y.
-=	Subtract and assign. For example, x-=y is the same as x=x-y.
*=	Multiply and assign. For example, x*=y is the same as x=x*y.
/=	Divide and assign. For example, x/=y is the same as x=x/y.
%=	Modulus and assign. For example, x%=y is the same as x=x%y.

Comparison Operators

Operator	Description
==	Is equal to
===	Is identical (is equal to and is of the same type)
!=	Is not equal to
!==	Is not identical
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Comparisons

Way to produce boolean values

```
console.log(10 > 20);                    // false
console.log(10 < 20);                    // true
console.log(10 >= 20);                   // false
console.log(10 <= 20);                   // true
console.log(10 === 20);                  // false
console.log(10 !== 20);                  // true
console.log(10 == '10')                  // true
console.log(10 === '10');                // false
console.log(null == undefined)           // true
console.log(null === undefined)          // false
```

Logical/boolean Operators

Operator	Description
&&	and
||	or
!	not

A	B	A&&B	A||B	!A
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

OR

JAVASCRIPT

```
true || false          // true
true || true           // true
Infinity|| true        // Infinity
'\n'|| false           //
'' || 0|| false        // false
'' || 1|| 'hi'         // 1
```

AND

JAVASCRIPT

```
true && false                    // false
1 && 2 && 3                      // 3
'Hi' && true && null && 1        // null
```

NOT

JAVASCRIPT

```
!true          // false
!!true         // true

!!''           // false
!!'false'      // true
!!false        // false
!!{}           // true
```

Multiple

JAVASCRIPT

```
!{} || ![1] && [] || !+true                // false
0 || !!false && !{} || NaN || !null        // true
```

Type conversion

String conversion:

```
var a = true + "test"; // "truetest"
var b = "test" + undefined; // "testundefined"
var c = 123 + ""; // "123"
var d = String(55); // "55"
var e = 123;
e = e.toString(); // "123"
```

Interaction

alert

prompt

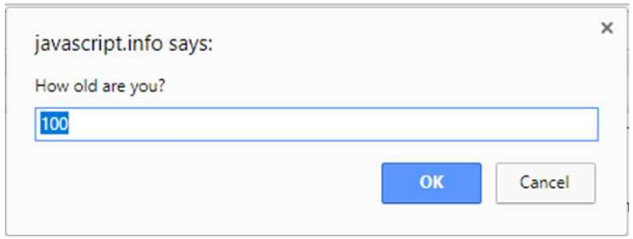
confirm

Syntax:

```
alert(message);
```

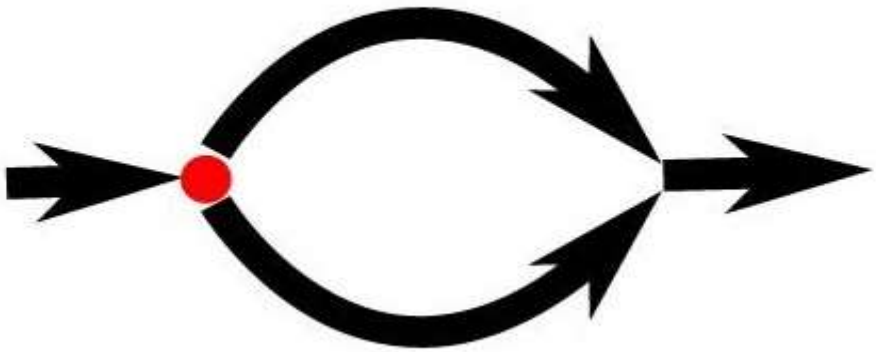
```
result = prompt(title[, default])
```

```
result = confirm(question);
```



Conditional execution

In conditional execution we choose between two different routes based on a boolean value.



Conditional execution is written with the if keyword in JavaScript. In the simple case, we just want some code to be executed if, and only if, a certain condition holds.

if .. else

Syntax

if (*condition*)
statement1
[else
 statement2]

```
if (cipher_char === from_char) {
  result = result + to_char;
  x++;
} else {
  result = result + clear_char;
}
```

else if

```
if (x > 5) {
  /* do the right thing */
} else if (x > 50) {
  /* do the right thing */
} else {
  /* do the right thing */
}
```

if

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

false

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

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

true

Other values become true, so they are called “truthy”.

```
if (1) {
// 1 is truthy ...
}
```

Example (if)

```
1   if ([expression]) {
2       [statement]
3   }
4
5
6   var a = 10;
7   if (a > 3) {
8       console.log('it is true')
9   }
10
11
12  // Bad style, always use braces
13  if (a > 5) console.log('it is true');
14
15  var user = false;
16  if (user) {
17      console.log('hello dear user');
18  } else {
19      console.log('please signIn');
20  }
```

if. Always use braces!

```
var a;
var b;
var c;

//Indenting is bad
if (a===true)
  alert(a); //Only on IF
  alert(b); //Always but expected?

//Nested indenting is misleading
if (a===true)
  if (b===true)
    alert(a); //Only on if-if
  alert (b); //Always but expected as part of first if?

//Problematic
if (a===true)
  alert(a);
  alert(b); //We're assuming this will happen with the if but it'll happen always
else       //This else is not connected to an if anymore - error
  alert(c);
```

```
//Compound line is misleading
//b will always alert, but suggests it's part of if
if (a===true) alert(a);alert(b);
else alert(c); //Error, else isn't attached

//Obvious
if (a===true) {
  alert(a); //on if
  alert(b); //on if
} else {
  alert(c); //on !if
}
```

if, without if

```
var i = 1,
    j = 5,
    result;

// bad approach, try to avoid it
if (i != j) {
    result = true;
} else {
    result = false;
}

// the same, but easier
result = i != j;
```

else if

Use the else if statement to specify a new condition if the first condition is false.

```
1   if(city === "Lviv"){
2      console.log("Hi Lviv")
3   } else if (city === "Kiev"){
4       console.log("Hi Kiev")
5   } else if(city === "Ternopil") {
6        console.log("Hi Ternopil")
7   } else {
8        console.log("Hi Unknown city")
9   }
```

Conditional (ternary) operator

syntax

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

```
var result = Math.PI > 4 ? “yes" : “no";
```

Example

```
1   var salary = 90000,
2       reaction = '';
3
4   // To define reaction we can use simple if statement
5   if (salary > 50000) {
6       reaction = 'Not bad!';
7   } else {
8       reaction = 'WTF?!!';
9   }
10
11
12  // But I'd rather use this one
13  reaction = a > 50000 ? 'Not bad!' : 'WTF?!!';
```

switch

```
switch (expression) {
  case value1:
    //Statements executed when the result of expression matches value1
    [break;]
  case value2:
    //Statements executed when the result of expression matches value2
    [break;]
  ...
  case valueN:
    //Statements executed when the result of expression matches valueN
    [break;]
  [default:
    //Statements executed when none of the values match the value of the expression
    [break;]]
}
```

Examples

```
switch (city) {
    case "Lviv" :
        console.log("Hi Lviv");
        break;
    case "Kiev" :
        console.log("Hi Kiev");
        break;
    case "Ternopil" :
        console.log("Hi Ternopil");
        break;
    default:
        console.log("Hi Unknown city");
        break;
}
```

```
var Animal = 'Giraffe';
switch (Animal) {
  case 'Cow':
  case 'Giraffe':
  case 'Dog':
  case 'Pig':
    console.log('This animal will go on Noah\'s
Ark.');
    break;
  case 'Dinosaur':
  default:
    console.log('This animal will not.');
}
```

You may put any number of case labels inside the block opened by switch. The program will jump to the label that corresponds to the value that switch was given, or to default if no matching value is found.

for

Syntax

```
for ([initialization]; [condition]; [final-expression])
statement
```

```
for (var i = 0; i < 9; i++)
{
   console.log(i);
// more statements
}
```

Fun with for

```
1   // we can move initialization
2   var i = 0;
3
4   for (; i < 10; i++) {
5       console.log(i);
6   }
7
8
9   // we can move increment
10  for (; --i;) {
11      console.log(i);
12  }
13
14
15  // finally we can remove everything
16  for (;;) {
17      if (i == 0) {
18          console.log(i);
19          break;
20      }
21  }
```

break and continue

```
for (var i = 0; i < 10; i++) {
    console.log('i=', i);
    if (i % 2 == 0) {
    continue;
 }
    console.log('i2=', i);
    if (i == 9) {
      break;
   }
}
console.log('i3=', i);
```

```
i= 0
i= 1
i2= 1
i= 2
i= 3
i2= 3
i= 4
i= 5
i2= 5
i= 6
i= 7
i2= 7
i= 8
i= 9
i2= 9
i3= 9
```

while

syntax

while (*condition*)
statement

```
var n= 2;
var x = 1;
while (n < 3) {
 n++;
 x += n;
}
```

do...while

syntax

do
  statement
while (condition);

```
var result = '';
var i = 0;
do {
  i += 1;
  result += i + ' ';
} while (i < 5);
```

for .. in

Syntax

```
for (variable in object) {
... }
```

```
var obj = {a: 1, b: 2, c: 3};
for (const prop in obj) {
console.log(`obj.${prop} = ${obj[prop]}`);
}
```

for .. of

Syntax

```
for (variable of iterable) {
statement
}
```

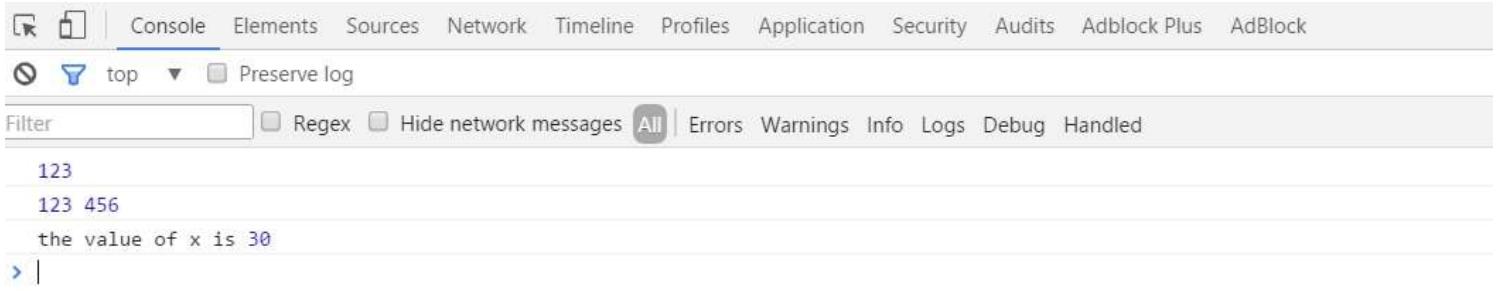
```
let iterable = [10, 20, 30];
for (let value of iterable) {
value += 1;
console.log(value);
}
```

JS output and debugging

For JavaScript values output you can use “console.log” or “alert” function.

```
console.log(123);
console.log(123, 456);
var x = 30;
console.log("the value of x is", x);
alert(123);
alert("the value of x is");
```

You can execute JS directly into console, open dev tools (F12) and go in "Console" tab



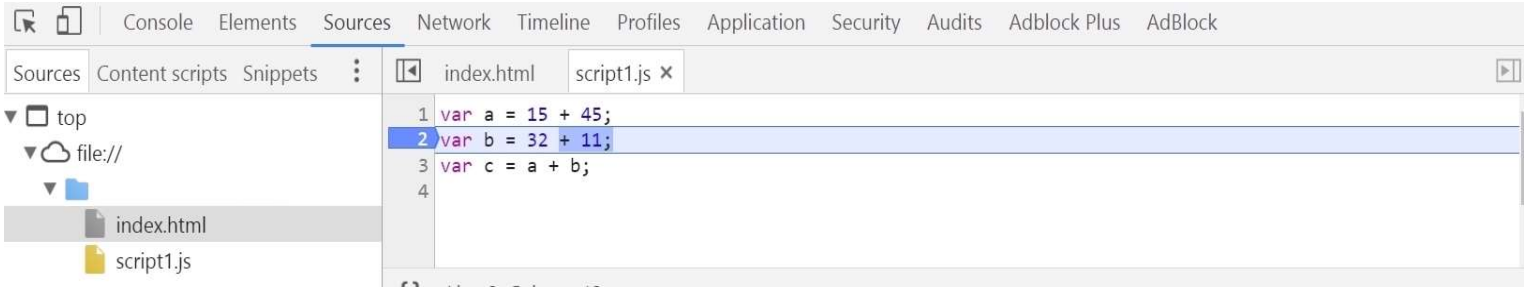
JS output and debugging

You can stop execution of JS in some particular place of code and take a look what is going on there. Open dev tools and go in 'Sources' tab.
Take a look at https://developers.google.com/web/tools/chrome-devtools/javascript/reference
index.html

```
<!Doctype html>
<html>
<head>
<script src="script1.js"></script>
</head>
...
</html>
```

script1.js

```
var a = 15 + 45;
var b = 32 + 11;
var c = a + b;
```



Q&A

UA Frontend Online LAB