# Javascript

# Topics

Basic Java Syntax

# What is JavaScript?

JavaScript is the programming language of the web

All modern web browsers use JavaScript

Node.js has enabled JavaScript programming outside of web browsers

JavaScript is completely different from the Java programming language

# What is JavaScript?

The core JavaScript language defines a minimal API for working with numbers, text, arrays, sets, maps, and so on, but does not include any input or output functionality

Input and output (as well as more sophisticated features, such as networking, storage, and graphics) are the responsibility of the “host environment” within which JavaScript is embedded

# What is JavaScript?

The original host environment for JavaScript was only a web browser

Node is another host environment

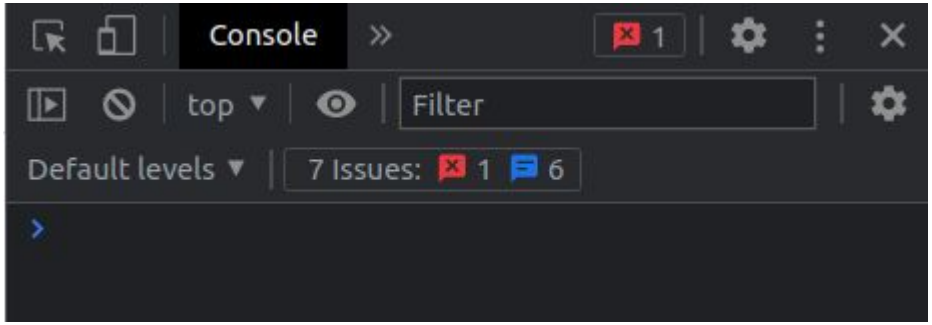
# What is JavaScript?

The **ECMA-262 specification** contains detailed and formalized information about JavaScript. It defines the language

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

# Ways to Use Javascript

On your web browser press **`F12`** to open a developer tool and then select the **`Console`** tab



# Ways to Use Javascript

Create an HTML file and use the **<script>** tag in the **<head>** or **<body>** tag to put your Javascript code

```
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <title>Document</title>
   <script>
     console.log("Hello World");
   </script>
 </head>
 <body></body>
</html>
```

# Ways to Use Javascript

Create an HTML file and use the **<script>** tag in the **<head>** or **<body>** tag to put your Javascript code

```
<html lang="en">
 <head>
   <meta charset="UTF-8" />
   <title>Document</title>
 </head>
 <body>
   <script>
     console.log("Hello World");
   </script>
 </body>
</html>
```

# Ways to Use Javascript

Use separate javascript file with file extension of **`.js`** and attach the **`Javascript`** file to the **`HTML`** file as shown

```
<html lang="en">
 <head>
    <meta charset="UTF-8" />
    <title>Document</title>
 </head>
 <body>
    <script src="example.js">
    </script>
 </body>
</html>
```

# Exercise

Using external `.js` file create a program which print `"Hello World"` on the Console window

# Statements

Statements are syntax constructs and commands that perform actions.

Semicolons at the end of statements is not required but recommended

```
alert('Hello');
alert('World');
```

# Comments

One-line comments `//`

```
// Hello world example
alert('Hello');
```

Multiline comments `/* */`

```
/*
  Hello world alert statement
*/
alert('Hello');
```

# Variables

To create a variable use the **`let`** keyword

```
// define the variable and assign the value
let message = 'Hello!';

alert(message); // Hello!
```

# Constants

To declare a constants use **`const`**

```
const myBirthday = '18.04.1982';


// error, can't reassign the constant!
myBirthday = '01.01.2001';
```

# Exercises

Declare two variables: `x` and `y`

Assign the value `"3"` to `x`

Copy the value from `x` to `y`

Show the value of `y` using `alert` or `console.log` (must output `“3”`)

# Data Types

There are eight basic data types

**`number`** for numbers of any kind: integer or floating-point, integers are limited by **`±(2^53-1)`**

**`bigint`** is for integer numbers of arbitrary length

**`string`** for strings

A string may have zero or more characters, there's no separate single-character type

**`boolean`** for `true`/`false`

# Data Types

There are eight basic data types

**`null`** for unknown values – a standalone type that has a single value `null`

**`undefined`** for unassigned values – a standalone type that has a single value `undefined`

**`object`** for more complex data structures

**`symbol`** for unique identifiers

# Number

The number type represents both **integer** and **floating point** numbers.

```
let n = 123;
n = 12.345;
```

Besides regular numbers, there are so-called “special numeric values” which also belong to this data type: **`Infinity`**, **`-Infinity`** and **`NaN`**

# BigInt

In JavaScript, the "**number**" type cannot represent integer values larger than **$(2^{53}-1)$** (that's 9007199254740991), or less than **$-(2^{53}-1)$** for negatives

A BigInt value is created by appending **n** to the end of an integer

```
// the "n" at the end means it's a BigInt
const bigInt = 12345678901234567890123456789012345678901234567890n;
```

# String

There are 3 types of quotes

Double quotes: **"Hello"**

Single quotes: **'Hello'**

Backticks: **`Hello`**

```
let str = "Hello";
let str2 = 'Single quotes are ok too';
let phrase = `can embed another ${str}`;
```

# Boolean

The boolean type has only two values: **`true`** and **`false`**

```
let isGreater = 4 > 1;


alert( isGreater ); // true (the comparison result is "yes")
```

# The “`null`” value

It’s a special value which represents “**nothing**”, “**empty**” or “**value unknown**”

```
let age = null;
```

# The “undefined” value

The meaning of **undefined** is “value is not assigned”.

If a variable is declared, but not assigned, then its value is **undefined**

```
let age;

alert(age); // shows "undefined"
```

# Objects and Symbols

Objects are used to store collections of data and more complex entities

The symbol type is used to create unique identifiers for objects

# The `typeof` operator

The `typeof` operator returns the type of the argument

It supports two forms of syntax:

As an operator: **`typeof x`**

As a function: **`typeof(x)`**

# The `typeof` operator

```
typeof undefined // "undefined"

typeof 0 // "number"

typeof 10n // "bigint"

typeof true // "boolean"

typeof "foo" // "string"
```

# The `typeof` operator

```
typeof Symbol("id") // "symbol"

typeof Math // "object"

typeof null // "object"

typeof alert // "function"
```

# Exercise

What is the output of the following

```
let name = "Abebe";

alert( `hello ${1}` ); // ?

alert( `hello ${"name"}` ); // ?

alert( `hello ${name}` ); // ?
```

# Interaction: `alert`, `prompt`, `confirm`

**`alert`**

It shows a message and waits for the user to press "**`OK`**"

```
alert("Hello");
```

# Interaction: `alert`, `prompt`, `confirm`

**`prompt`**

The function **`prompt`** accepts two arguments, the syntax is shown below

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

**`title`** The text to show the visitor

**`default`** An optional second parameter, the initial value for the input field

# Interaction: `alert`, `prompt`, `confirm`

**`prompt`**

```
let age = prompt('How old are you?', 100);


alert(`You are ${age} years old!`); // You are 100 years old!
```

# Interaction: `alert`, `prompt`, `confirm`

**`confirm`**

The function **`confirm`** shows a modal window with a question and two buttons: **`OK`** and **`Cancel`**

The result is **`true`** if **`OK`** is pressed and **`false`** otherwise

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

# Interaction: `alert`, `prompt`, `confirm`

**`confirm`**

```
let isBoss = confirm("Are you the boss?");


alert( isBoss ); // true if OK is pressed
```

# Exercise

Create a web-page that asks for a name and outputs it

# Type Conversions

Most of the time, operators and functions automatically convert the values given to them to the right type

- For example, alert automatically converts any value to a string to show it
- Mathematical operations convert values to numbers

There are also cases when we need to explicitly convert a value to the expected type

# String Conversion

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

We can also call the `String(value)` function to convert a value to a string

```
let value = true;
alert(typeof value); // boolean

value = String(value); // now value is a string "true"
alert(typeof value); // string
```

# Numeric Conversion

Numeric conversion happens in mathematical functions and expressions automatically

For example, when division `/` is applied to non-numbers

```
alert( "6" / "2" ); // 3, strings are converted to numbers
```

# Numeric Conversion

```
let str = "123";
alert(typeof str); // string

let num = Number(str); // becomes a number 123

alert(typeof num); // number
```

# Numeric conversion Rules

| Values | Becomes |
|---|---|
| `undefined` | `NaN` |
| `null` | `0` |
| `true` and `false` | `1` and `0` |
| `string` | Whitespaces from the start and end are removed. If the remaining string is empty, the result is `0`. Otherwise, the number is “read” from the string. An error gives `NaN` |

# Numeric Conversion

```
alert( Number("    123    ") ); // 123
alert( Number("123z") ); // NaN (error reading a number at "z")
alert( Number(true) );    // 1
alert( Number(false) );   // 0
```

# Boolean Conversion

It happens in logical operations but can also be performed explicitly with a call to `Boolean(value)`

**The conversion rule**

> Values that are intuitively "`empty`", like `0`, an `empty string`, `null`, `undefined`, and `NaN`, become `false`
>
> `Other values` become `true`

# Boolean Conversion

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

alert( Boolean("hello") ); // true
alert( Boolean("") ); // false
```

# Math Operations

The following math operations are supported:

| | |
|---|---|
| `Addition +` | `Division /` |
| `Subtraction -` | `Remainder %` |
| `Multiplication *` | `Exponentiation **` |

# String concatenation with binary +

If the binary + is applied to strings, it merges (concatenates) them

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

# Numeric conversion, unary +

The plus **+** exists in two forms: the **binary** form and the **unary** form

The unary plus or, in other words, the plus operator + applied to a single value, doesn't do anything to numbers

But if the operand is not a number, the unary plus converts it into a number

# Numeric conversion, unary +

```
// No effect on numbers
let x = 1;
alert( +x ); // 1

let y = -2;
alert( +y ); // -2

// Converts non-numbers
alert( +true ); // 1
alert( +"" );   // 0
```

# Increment/Decrement

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

```
let counter = 2;
counter++;          // same as counter = counter + 1
alert( counter ); // 3
```

# Increment/Decrement

Decrement **--** increases a variable by **1**

```
let counter = 2;
counter--;          // same as counter = counter - 1
alert( counter ); // 1
```

# Exercise

What are the final values of all variables **a**, **b**, **c** and **d**

```
let a = 1, b = 1;


let c = ++a; // ?
let d = b++; // ?
```

# Exercise

What are the values of **a** and **x**?

```
let a = 2;


let x = 1 + (a *= 2);
```

# Exercise

What are results of these expressions?

```
"" + 1 + 0
"" - 1 + 0
true + false
6 / "3"
"2" * "3"
4 + 5 + "px"
"$" + 4 + 5
```

```
"4" - 2
"4px" - 2
"   -9   " + 5
"   -9   " - 5
null + 1
undefined + 1
" \t \n" - 2
```

# Exercise

The output in the example below is 12 (for default prompt values)

Why?

The result should be 3

Fix it

```
let a = prompt("First number?", 1);
let b = prompt("Second number?", 2);

alert(a + b); // 12
```

# Comparisons

Greater/less than: `a > b`, `a < b`

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

Equals: `a == b`

# Comparisons

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

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

# String comparison

JavaScript uses the so-called “dictionary” or “lexicographical” order to check whether a string is greater than another

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

# Strict equality

A regular equality check **==** has a problem

It cannot differentiate **0** from **false,** for example

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

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

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

# Strict equality

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

# Exercise

What will be the result for these expressions?

```
5 > 4
"apple" > "pineapple"
"2" > "12"
undefined == null
undefined === null
null == "\n0\n"
null === +"\n0\n"
```

# The “if” statement

```
let year = prompt('In which year was ECMAScript-2015 specification published?', '');

if (year == 2015) alert( 'You are right!' );
```

# The “else” clause

```
let year = prompt('In which year was ECMAScript-2015
specification published?', '');

if (year == 2015) {
 alert("You guessed it right!");
} else {
 alert("How can you be so wrong?"); // any value except 2015
}
```

# Several conditions: “`else if`”

```
let year = prompt('In which year was ECMAScript-2015
specification published?', '');

if (year < 2015) {
 alert("Too early...");
} else if (year > 2015) {
 alert("Too late");
} else {
 alert("Exactly!");
}
```

# Conditional operator ‘? . . . :’

Also called ternary operator

The syntax is shown below

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

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

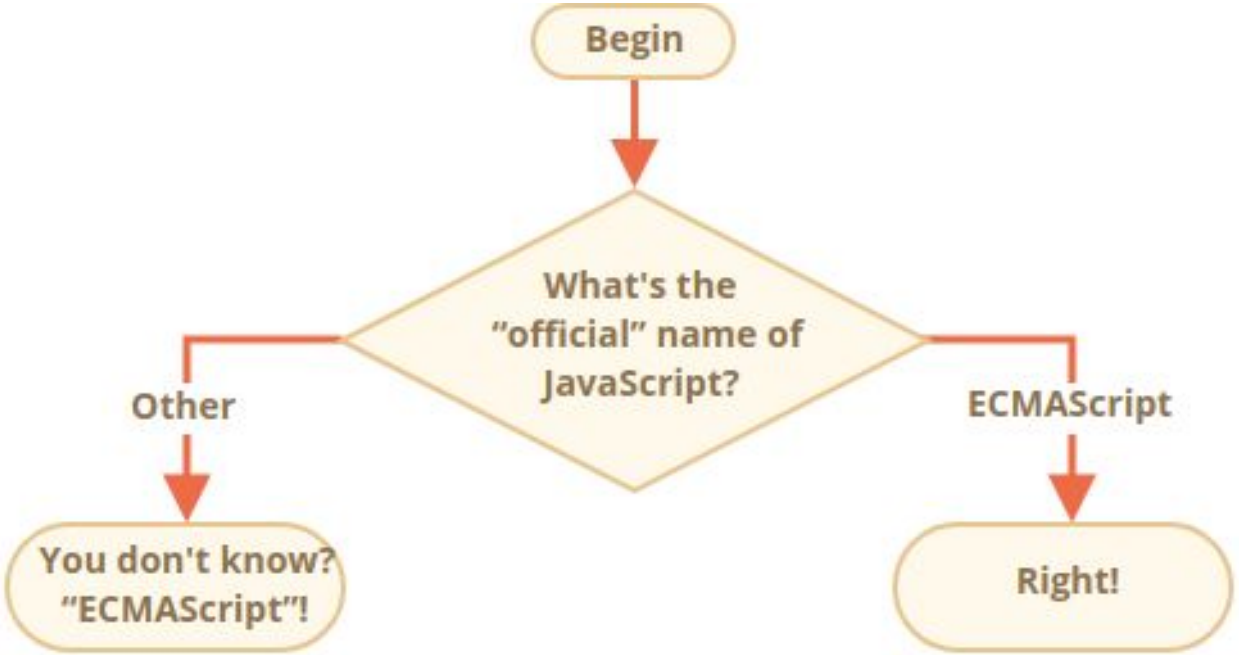
# Exercise

Will alert be shown?

```
if ("0") {
 alert("Hello");
}
```

# Exercise

Convert the following conditional flow chart into `if ... else` construct and ternary operator



# Exercise

Using `if..else`, write the code which gets a number via `prompt` and then shows in `alert`:

`1`, if the value is greater than zero,

`-1`, if less than zero,

`0`, if equals zero.

# Exercise

Rewrite the following **`if...else`** using the conditional operator **`?...:`**

```
let result;

if (a + b < 4) {
 result = "Below";
} else {
 result = "Over";
}
```

# Logical Operators

There are four logical operators

- `||` (OR),
- `&&` (AND),
- `!` (NOT),
- `??` (Nullish Coalescing).

# || (OR)

```
alert(true || true); // true
alert(false || true); // true
alert(true || false); // true
alert(false || false); // false
```

# OR "||" finds the first `truthy` value

The OR `||` operator does the following:

Evaluates operands from left to right

For each operand, converts it to boolean. If the result is `true`, stops and returns the original value of that operand

If all operands have been evaluated (i.e. all were `false`), returns the last operand

```
result = value1 || value2 || value3;
```

# OR "||" finds the first `truthy` value

```
alert( 1 || 0 ); // 1 (1 is truthy)

alert( null || 1 ); // 1 (1 is the first truthy value)
alert( null || 0 || 1 ); // 1 (the first truthy value)

// 0 (all falsy, returns the last value)
alert( undefined || null || 0 );
```

# && (AND)

```
alert( true && true );   // true
alert( false && true );  // false
alert( true && false );  // false
alert( false && false ); // false
```

# AND "`&&`" finds the first `falsy` value

The AND `&&` operator does the following:

Evaluates operands from left to right

For each operand, converts it to a boolean. If the result is **`false`**, stops and returns the original value of that operand

If all operands have been evaluated (i.e. all were **`truthy`**), returns the last operand

# AND “&&” finds the first `falsy` value

```
// 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
```

# ! (NOT)

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

# Exercise

What is  the output of the following code?

```
alert( null || 2 || undefined );
```

# Exercise

What is  the output of the following code?

```
alert( alert(1) || 2 || alert(3) );
```

# Exercise

What is  the output of the following code?

```
alert( 1 && null && 2 );
```

# Exercise

What is  the output of the following code?

```
alert( alert(1) && alert(2) );
```

# Nullish coalescing operator '??'

We'll say that an expression is "**defined**" when it's **neither** `null` nor `undefined`

The result of `a ?? b` is:

if `a` is defined, then `a`

if `a` isn't defined, then `b`

In other words, `??` returns the first argument if it's not **`null/undefined`**. Otherwise, the second one

# Nullish coalescing operator '??'

```
let user;
alert(user ?? "Anonymous"); // Anonymous (user not defined)
```

```
let user = "Bekele";
alert(user ?? "Anonymous"); // Bekele (user defined)
```

# Loops: `while` and `for`

Loops are a way to repeat the same code multiple times

# The “while” loop

```
while (condition) {
 // code
 // so-called "loop body"
}
```

```
let i = 0;
while (i < 3) { // shows 0, then 1, then 2
 alert( i );
 i++;
}
```

# The “do...while” loop

```
do {
 // loop body
} while (condition);
```

```
let i = 0;
do {
 alert( i );
 i++;
} while (i < 3);
```

# The “for” loop

```
for (begin; condition; step) {
 // ... loop body ...
}
```

```
for (let i = 0; i < 3; i++) { // shows 0, then 1, then 2
 alert(i);
}
```

# The “for” loop

Any part of for can be skipped

```
let i = 0; // we have i already declared and assigned

for (; i < 3; i++) { // no need for "begin"
 alert( i ); // 0, 1, 2
}
```

# The “for” loop

Any part of for can be skipped

```
let i = 0;

for (; i < 3;) {
 alert( i++ );
}
```

# The “for” loop

Any part of for can be skipped

```
for (;;) {
 // repeats without limits
}
```

# Breaking the loop

Normally, a loop exits when its condition becomes falsy

But we can force the exit at any time using the special break directive

```
let sum = 0;
while (true) {
 let value = +prompt("Enter a number", '');
 if (!value) break;
 sum += value;
}
alert( 'Sum: ' + sum );
```

# Continue to the next iteration

```
for (let i = 0; i < 10; i++) {

 // if true, skip the remaining part of the body
 if (i % 2 == 0) continue;

 alert(i); // 1, then 3, 5, 7, 9
}
```

# Exercise

What is the last value alerted by the following code? Why?

```
let i = 3;


while (i) {
 alert( i-- );
}
```

# Exercise

Replace "`for`" with "`while`"

```
for (let i = 0; i < 3; i++) {
 alert( `number ${i}!` );
}
```

# Exercise

An integer number greater than 1 is called a prime if it cannot be divided without a remainder by anything except 1 and itself

In other words, n > 1 is a prime if it can't be evenly divided by anything except 1 and n

> For example, 5 is a prime, because it cannot be divided without a remainder by 2, 3 and 4

Write the code which outputs prime numbers in the interval from 2 to n.

For n = 10 the result will be 2,3,5,7

# The "switch" statement

A **switch** statement can replace multiple **if** checks

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

 case 'value2':
    ...
    [break]

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

# The "switch" statement

```
let a = 2 + 2;

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

# Exercise

Write the code shown using **`if..else`**

```
switch (browser) {
 case 'Edge':
   alert( "You've got the Edge!" );
   break;

 case 'Chrome':
 case 'Firefox':
 case 'Safari':
 case 'Opera':
   alert( 'Okay we support these browsers too '
);
   break;

 default:
   alert( 'We hope that this page looks ok! ' );
}
```

# Exercise

Rewrite the code shown using a single **`switch`** statement

```
let a = +prompt('a?', '');

if (a == 0) {
 alert( 0 );
}
if (a == 1) {
 alert( 1 );
}

if (a == 2 || a == 3) {
 alert( '2,3' );
}
```

# Functions

Functions are the main “building blocks” of the program

They allow the code to be called many times without repetition

> `alert(message)`, `prompt(message, default)` and `confirm(question)` are examples of functions

# Function Declaration

To create a function we can use a `function` declaration

```
function name(parameter1, parameter2, ...parameterN) {
 // body
}
```

```
function showMessage() {
 alert( 'Hello everyone!' );
}
```

# Local variables

A variable declared inside a function is only visible inside that function.

```
function showMessage() {
 let message = "Hello, I'm JavaScript!"; // local variable
 alert( message );
}
showMessage(); // Hello, I'm JavaScript!
alert( message ); // <-- Error!
```

# Outer variables

A function can access an outer variable as well, for example:

```
let userName = 'Mohammed';

function showMessage() {
 let message = 'Hello, ' + userName;
 alert(message);
}

showMessage(); // Hello, Mohammed
```

# Parameters

We can pass arbitrary data to functions using parameters.

```
function showMessage(from, text) { // parameters: from, text
 alert(from + ': ' + text);
}

showMessage('Abebe', 'Hello!'); // Abebe: Hello!
showMessage('Aster', "What's up?"); // Aster: What's up?
```

# Default values

If a function is called, but an argument is not provided, then the corresponding value becomes **`undefined`**

```
function showMessage(from, text = "no text given") {
 alert( from + ": " + text );
}

showMessage("Abebe"); // Ann: no text given
```

# Returning a value

A function can return a value back into the calling code as the result

```
function sum(a, b) {
 return a + b;
}

let result = sum(1, 2);
alert( result ); // 3
```

# Exercise

Will the following function work differently if `else` is removed?

```
function checkAge(age) {
 if (age > 18) {
   return true;
 } else {
   // ...
   return confirm('Did parents allow you?');
 }
}
```

# Exercise

Write a function `min(a,b)` which returns the least of two numbers a and b

# Exercise

Write a function `pow(x,n)` that returns `x` in power `n`. Or, in other words, multiplies `x` by itself `n` times and returns the result.

# Function expressions

```
let sayHi = function() {
 alert( "Hello" );
};
```

# Function expressions

```
function sayHi() {
 alert( "Hello" );
}

alert( sayHi ); // shows the function code
```

# Function expressions

```
function sayHi() {   // (1) create
 alert( "Hello" );
}

let func = sayHi;    // (2) copy

func(); // Hello     // (3) run the copy
sayHi(); // Hello    // this still works too
```

# Callback functions

```
function ask(question, yes, no) {
 if (confirm(question)) yes()
 else no();
}
```

```
function showOk() {
 alert( "You agreed." );
}
function showCancel() {
 alert( "You canceled the execution." );
}
```

```
// functions showOk, showCancel are passed as arguments to ask
ask("Do you agree?", showOk, showCancel);
```

# Arrow functions, the basics

There’s another very simple and concise syntax for creating functions

```
let func = (arg0, arg2, ..., argN) => expression
```

# Arrow functions, the basics

There’s another very simple and concise syntax for creating functions

```
let sum = (a, b) => a + b;


alert( sum(1, 2) ); // 3
```

# Multiline arrow functions

```
let sum = (a, b) => {
 let result = a + b;
 return result; // we need an explicit "return"
};

alert( sum(1, 2) ); // 3
```

# Exercise

Replace Function Expressions shown with arrow functions

```
function ask(question, yes, no) {
 if (confirm(question)) yes();
 else no();
}
```

```
ask(
 "Do you agree?",
 function() { alert("You agreed."); },
 function() { alert("You canceled the execution."); }
);
```

# References


The Modern JavaScript Tutorial

JavaScript: The Definitive Guide, 7th Edition