AN INTRODUCTION TO JAVASCRIPT

ANDREAS DRANIDIS

AN INTRODUCTION TO JAVASCRIPT

- JavaScript was initially created to “make web pages alive”.
- The programs in this language are called scripts. They can be written right in a web page’s HTML and run automatically as the page loads.
- Scripts are provided and executed as plain text. They don’t need special preparation or compilation to run.
- When JavaScript was created, it initially had another name: “LiveScript”. But Java was very popular at that time, so it was decided that positioning a new language as a “younger brother” of Java would help. But as it evolved, JavaScript became a fully independent language with its own specification called ECMAScript, and now it has no relation to Java at all.

JAVASCRIPT ENGINE

- Today, JavaScript can execute not only in the browser, but also on the server, or actually on any device that has a special program called the JavaScript engine.
- The browser has an embedded engine sometimes called a “JavaScript virtual machine”.
- Different engines have different “codenames”. For example:
  - V8 – in Chrome, Opera and Edge.
  - SpiderMonkey – in Firefox.
  - …There are other codenames like “Chakra” for IE, “JavaScriptCore”, “Nitro” and “SquirrelFish” for Safari, etc.

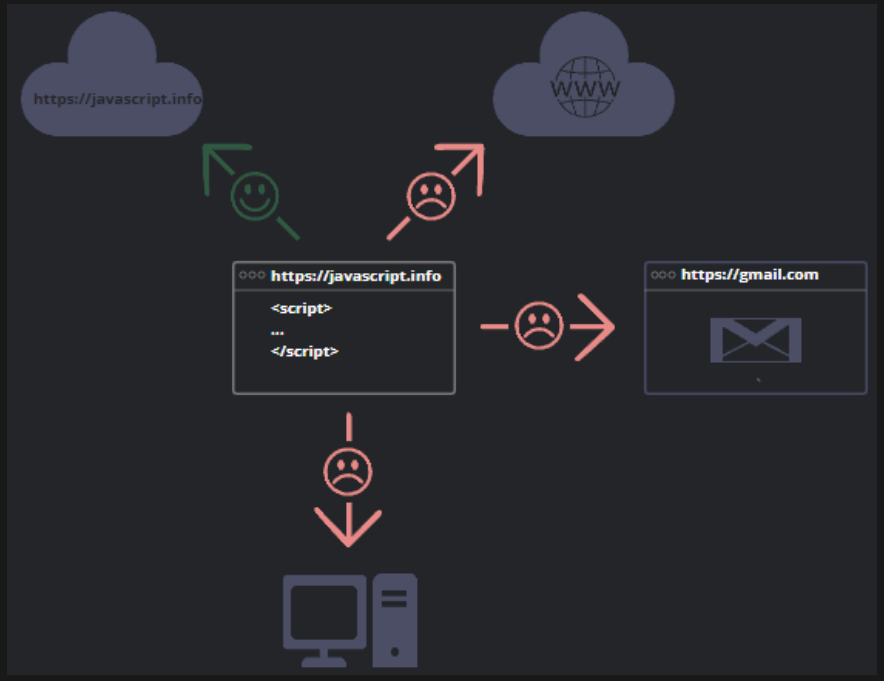
WHAT CAN IN-BROWSER JAVASCRIPT DO?

- Modern JavaScript is a “safe” programming language. It does not provide low-level access to memory or CPU, because it was initially created for browsers which do not require it.
- JavaScript’s capabilities greatly depend on the environment it’s running in. For instance, Node.js supports functions that allow JavaScript to read/write arbitrary files, perform network requests, etc.
- In-browser JavaScript can do everything related to webpage manipulation, interaction with the user, and the webserver.
- For instance, in-browser JavaScript is able to:
  - Add new HTML to the page, change the existing content, modify styles.
  - React to user actions, run on mouse clicks, pointer movements, key presses.
  - Send requests over the network to remote servers, download and upload files (so-called AJAX and COMET technologies).
  - Get and set cookies, ask questions to the visitor, show messages.
  - Remember the data on the client-side (“local storage”).

WHAT CAN’T IN-BROWSER JAVASCRIPT DO?

WHAT CAN'T IN-BROWSER JAVASCRIPT DO?

- JavaScript can easily communicate over the net to the server where the current page came from. But its ability to receive data from other sites/domains is crippled. Though possible, it requires explicit agreement (expressed in HTTP headers) from the remote side. Once again, that's a safety limitation.
- Such limits do not exist if JavaScript is used outside of the browser, for example on a server. Modern browsers also allow plugin/extensions which may ask for extended permissions.



HELLO, WORLD!

---

inside the script tags

```
<script>
  alert('Hello, world!');
</script>
```

linking to external local js file

```
<script src="/path/to/script.js"></script>
```

linking to external remote js file

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.11/lodash.js"></script>
```

src is prioritised from code inside

```
<script src="file.js">
  alert('Hello, world!');
</script>
```

STATEMENTS

Statements are syntax constructs and commands that perform actions.

We can have as many statements in our code as we want. Statements can be separated with a semicolon.

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

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

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

IN MOST CASES, A NEWLINE IMPLIES A SEMICOLON. BUT “IN MOST CASES” DOES NOT MEAN “ALWAYS”! IT IS RECOMMENDED TO PUT SEMICOLONS BETWEEN STATEMENTS EVEN IF THEY ARE SEPARATED BY NEWLINES.

```
alert("Hello")

[1, 2].forEach(alert);
```

COMMENTS

One-line comments start with two forward slash characters //.

```
// This comment occupies a line of its own
alert('Hello');

alert('World'); // This comment follows the statement
```

Multiline comments start with a forward slash and an asterisk /* and end with an asterisk and a forward slash */.

```
/* An example with two messages.
This is a multiline comment.
*/
alert('Hello');
alert('World');
```

TIP: Use Ctrl + / *to comment easily*

"USE STRICT"

- For a long time, JavaScript evolved without compatibility issues. New features were added to the language while old functionality didn’t change.
- That had the benefit of never breaking existing code. But the downside was that any mistake or an imperfect decision made by JavaScript’s creators got stuck in the language forever.
- This was the case until 2009 when ECMAScript 5 (ES5) appeared. It added new features to the language and modified some of the existing ones. To keep the old code working, most such modifications are off by default. You need to explicitly enable them with a special directive: "use strict".

```
"use strict";

// this code works the modern way
...
```

Modern JavaScript supports “classes” and “modules” – advanced language structures (we’ll surely get to them), that enable use strict automatically. So we don’t need to add the "use strict" directive, if we use them.

VARIABLES

To create a variable in JavaScript, use the let keyword.

```
let message;

message = 'Hello'; // store the string 'Hello' in the variable named message
```

```
let user = 'John', age = 25, message = 'Hello';

let user = 'John';
let age = 25;
let message = 'Hello';

let user = 'John'
  , age = 25
  , message = 'Hello';
```

```
let hello = 'Hello world!';
let message;
message = hello;

// now two variables hold the same data
alert(hello); // Hello world!
alert(message); // Hello world!
```

```
let message = "This";
let message = "That"; // SyntaxError: 'message' has already been declared
```

VARIABLE NAMING

There are two limitations on variable names in JavaScript:

1. The name must contain only letters, digits, or the symbols $ and _.
2. The first character must not be a digit.

When the name contains multiple words, camelCase is commonly used. That is: words go one after another, each word except first starting with a capital letter: `myVeryLongName`.

Variables named `apple` and `AppLE` are two different variables.

There is a list of reserved words, which cannot be used as variable names because they are used by the language itself. For example: let, class, return, and function are reserved.

```
let let = 5; // can't name a variable "let", error!
let return = 5; // also can't name it "return", error!
```

CONSTANTS

To declare a constant (unchanging) variable, use const instead of let:

```
const myBirthday = '18.04.1982';

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

There is a widespread practice to use constants as aliases for difficult-to-remember values that are known prior to execution. Such constants are named using capital letters and underscores.

```
const COLOR_RED = "#F00";
const COLOR_GREEN = "#0F0";
const COLOR_BLUE = "#00F";
const COLOR_ORANGE = "#FF7F00";

// ...when we need to pick a color
let color = COLOR_ORANGE;
alert(color); // #FF7F00
```

```
const pageLoadTime = /* time taken by a webpage to load */;
```

The value of pageLoadTime is not known prior to the page load, so it's named normally. But it's still a constant because it doesn't change after assignment. In other words, capital-named constants are only used as aliases for "hard-coded" values.

DATA TYPES

Programming languages that allow such things, such as JavaScript, are called “dynamically typed”, meaning that there exist data types, but variables are not bound to any of them.

```
// no error
let message = "hello";
message = 123456;
```

- Number
- BigInt
- String
- Boolean (logical type)
- The “null” value
- The “undefined” value
- Objects and Symbols

```
typeof undefined // "undefined"
typeof 0 // "number"
typeof 10n // "bigint"
typeof true // "boolean"
typeof "foo" // "string"
typeof Symbol("id") // "symbol"
typeof Math // "object"
```

INTERACTION: ALERT, PROMPT, CONFIRM

alert

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

prompt

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

let age = prompt('How old are you?', 100);
alert(`You are ${age} years old!`); // You are 100 years old!
```

confirm

```
result = confirm(question);

let isBoss = confirm("Are you the boss?");
alert( isBoss ); // true if OK is pressed
```

TYPE CONVERSIONS

String Conversion

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

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

Numeric Conversion

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

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

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

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

Boolean Conversion

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

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

BASIC OPERATORS, MATHS

Terms: “unary”, “binary”, “operand”

operand

is what operators are applied to. For instance, in the multiplication of 5 * 2 there are two operands: the left operand is 5 and the right operand is 2. Sometimes, people call these “arguments” instead of “operands”.

unary

An operator is unary if it has a single operand. For example, the unary negation - reverses the sign of a number

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

binary

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

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

MATHS

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

MATHS

String concatenation with binary +

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

alert( '1' + 2 ); // "12"
alert( 2 + '1' ); // "21"
alert(2 + 2 + '1' ); // "41" and not "221"
alert('1' + 2 + 2); // "122" and not "14"

alert( 6 - '2' ); // 4, converts '2' to a number
alert( '6' / '2' ); // 3, converts both operands to numbers
```

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
```

The need to convert strings to numbers arises very often. For example, if we are getting values from HTML form fields, they are usually strings. What if we want to sum them?

MATHS

Assignment

```
let x = 2 * 2 + 1;

alert( x ); // 5
```

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
```

```
('b'+'a'+ +'🍌'+'a').toUpperCase();
```

MATHS

Chaining assignments

```
let a, b, c;

a = b = c = 2 + 2;

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

Modify-in-place

```
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
```

Increment/decrement

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

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

COMPARISONS

- Greater/less than: a > b, a < b.
- Greater/less than or equals: a >= b, a <= b.
- Equals: a == b, please note the double equality sign == means the equality test, while a single one a = b means an assignment.
- Not equals: In maths the notation is ≠, but in JavaScript it's written as a != b.

```
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
```

FUN FACT

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

COMPARISONS


```
alert( '2' > 1 ); // true, string '2' becomes a number 2
alert( '01' == 1 ); // true, string '01' becomes a number 1
alert( true == 1 ); // true
alert( false == 0 ); // true

let a = 0;
alert( Boolean(a) ); // false

let b = "0";
alert( Boolean(b) ); // true

alert(a == b); // true!
```

COMPARISONS

Strict equality

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

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

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

CONDITIONAL BRANCHING: IF, '?'

The “if” statement

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

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

if (year == 2015) {
  alert( "That's correct!" );
  alert( "You're so smart!" );
}

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

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

The “else” clause

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

CONDITIONAL BRANCHING: IF, '?'

Several conditions: “else if”

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

Conditional operator ‘?’

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

if (age > 18) {
  accessAllowed = true;
} else {
  accessAllowed = false;
}

alert(accessAllowed);

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

LOGICAL OPERATORS

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

LOGICAL OPERATORS

|| (OR)

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

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

let hour = 12;
let isWeekend = true;

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

OR "||" finds the first truthy value

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

let firstName = "";
let lastName = "";
let nickName = "SuperCoder";

alert( firstName || lastName || nickName || "Anonymous"); // SuperCoder
```

LOGICAL OPERATORS

&& (AND)

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

let hour = 12;
let minute = 30;

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

AND “&&” finds the first falsy value

```
result = value1 && value2 && value3;

alert( 1 && 2 && null && 3 ); // null
alert( 1 && 2 && 3 ); // 3, the last one
```

Precedence of AND && is higher than OR ||

```
a && b || c && d
(a && b) || (c && d)
```

LOGICAL OPERATORS

! (NOT)

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

alert( !!"non-empty string" ); // true
alert( !!null ); // false

alert( Boolean("non-empty string") ); // true
alert( Boolean(null) ); // false
```

The precedence of NOT ! is the highest of all logical operators, so it always executes first, before && or ||.

LOGICAL OPERATORS

Nullish coalescing operator '??'

```
let user;

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

user = "John";

alert(user ?? "Anonymous"); // John (user defined)
```

The common use case for ?? is to provide a default value.

```
let firstName = null;
let lastName = null;
let nickName = "Supercoder";

// shows the first defined value:
alert(firstName ?? lastName ?? nickName ?? "Anonymous"); // Supercoder
```

LOGICAL OPERATORS

Nullish coalescing operator '??'

Comparison with ||

- || returns the first truthy value.
- ?? returns the first defined value.

```
let height = 0;

alert(height || 100); // 100
alert(height ?? 100); // 0
```

The precedence of the ?? operator is the same as ||

```
// without parentheses
let area = height ?? 100 * width ?? 50;

// ...works this way (not what we want):
let area = height ?? (100 * width) ?? 50;
```

LOOPS: WHILE AND FOR

The “while” loop

```
let i = 3;
while (i) { // when i becomes 0, the condition becomes falsy, and the loop stops
  alert( i );
  i--;
}
```

The “do…while” loop

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

The “for” loop

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

LOOPS: WHILE AND FOR

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
}

i = 0;

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

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

Breaking the loop

```
let sum = 0;

while (true) {

  let value = +prompt("Enter a number", '');

  if (!value) break; // (*)

  sum += value;

}
alert( 'Sum: ' + sum );
```

LOOPS: WHILE AND FOR

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
}

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

  if (i % 2) {
    alert( i );
  }

}
```

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" );
}
```

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" );
}
```

THE "SWITCH" STATEMENT

Grouping of “case”

```
let a = 3;
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.');
}
```

Type matters

```
let arg = prompt("Enter a value?");
switch (arg) {
  case '0':
  case '1':
    alert( 'One or zero' );
    break;
  case 3:
    alert( 'Never executes!' );
    break;
  default:
    alert( 'An unknown value' );
}
```

FUNCTIONS

Function Declaration

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

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

showMessage();
```

Local variables

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

  alert( message );
}

showMessage(); // Hello, I'm JavaScript!

alert( message ); // <-- Error! The variable is local to the function
```

FUNCTIONS

Outer variables

```
let userName = 'John';

function showMessage() {
  userName = "Bob"; // (1) changed the outer variable

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

alert( userName ); // John before the function call

showMessage();

alert( userName ); // Bob, the value was modified by the function
```

Parameters

```
function showMessage(from, text) {

  from = '*' + from + '*'; // make "from" look nicer

  alert( from + ': ' + text );
}

let from = "Ann";

showMessage(from, "Hello"); // *Ann*: Hello

// the value of "from" is the same, the function modified a local copy
alert( from ); // Ann
```

FUNCTIONS

Default values

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

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

function showMessage(from, text = anotherFunction()) {
  // anotherFunction() only executed if no text given
  // its result becomes the value of text
}
```

Returning a value

```
function checkAge(age) {
  if (age >= 18) {
    return true;
  } else {
    return confirm('Do you have permission from your parents?');
  }
}

let age = prompt('How old are you?', 18);

if ( checkAge(age) ) {
  alert( 'Access granted' );
} else {
  alert( 'Access denied' );
}
```

FUNCTIONS

Never add a newline between return and the value

```
return
 (some + long + expression + or + whatever * f(a) + f(b))

return;
 (some + long + expression + or + whatever * f(a) + f(b))

return (
  some + long + expression
  + or +
  whatever * f(a) + f(b)
  )
```

FUNCTIONS

Naming a function

```
showMessage(..)      // shows a message
getAge(..)           // returns the age (gets it somehow)
calcSum(..)          // calculates a sum and returns the result
createForm(..)       // creates a form (and usually returns it)
checkPermission(..) // checks a permission, returns true/false
```

One function – one action

FUNCTIONS

Functions == Comments

```
function showPrimes(n) {
  nextPrime: for (let i = 2; i < n; i++) {

    for (let j = 2; j < i; j++) {
      if (i % j == 0) continue nextPrime;
    }

    alert( i ); // a prime
  }
}
```

```
function showPrimes(n) {

  for (let i = 2; i < n; i++) {
    if (!isPrime(i)) continue;

    alert(i);  // a prime
  }
}

function isPrime(n) {
  for (let i = 2; i < n; i++) {
    if ( n % i == 0) return false;
  }
  return true;
}
```

FUNCTION EXPRESSIONS

Function is a value

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

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

alert( sayHi ); // shows the function code

let func = sayHi;    // (2) copy

func(); // Hello     // (3) run the copy (it works)!
sayHi(); // Hello    //     this still works too (why wouldn't it)
```

Please note: there are no parentheses after sayHi. If there were, then func = sayHi() would write the result of the call sayHi() into func, not the function sayHi itself.

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." );
}

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

The arguments showOk and showCancel of ask are called callback functions or just callbacks.

CALLBACK FUNCTIONS

Function Expression vs Function Declaration

A Function Expression is created when the execution reaches it and is usable only from that moment.

A Function Declaration can be called earlier than it is defined.

```
sayHi("John"); // Hello, John

function sayHi(name) {
  alert( `Hello, ${name}` );
}
```

```
sayHi("John"); // error!

let sayHi = function(name) {  // (*) no magic any more
  alert( `Hello, ${name}` );
};
```

CALLBACK FUNCTIONS

In strict mode, when a Function Declaration is within a code block, it's visible everywhere inside that block. But not outside of it.

```
let age = 16; // take 16 as an example

if (age < 18) {
  welcome();               // \   (runs)
                           //  |
  function welcome() {     //  |
    alert("Hello!");       //  |  Function Declaration is available
  }                        //  |  everywhere in the block where it's declared
                           //  |
  welcome();               // /   (runs)

} else {

  function welcome() {
    alert("Greetings!");
  }
}

// Here we're out of curly braces,
// so we can not see Function Declarations made inside of them.

welcome(); // Error: welcome is not defined
```

CALLBACK FUNCTIONS

```
let age = prompt("What is your age?", 18);

let welcome;

if (age < 18) {

  welcome = function() {
    alert("Hello!");
  };

} else {

  welcome = function() {
    alert("Greetings!");
  };

}

welcome(); // ok now
```

```
let age = prompt("What is your age?", 18);

let welcome = (age < 18) ?
  function() { alert("Hello!"); } :
  function() { alert("Greetings!"); };

welcome(); // ok now
```

CALLBACK FUNCTIONS

When to choose Function Declaration versus Function Expression?

- As a rule of thumb, when we need to declare a function, the first to consider is Function Declaration syntax. It gives more freedom in how to organize our code, because we can call such functions before they are declared.
- That's also better for readability, as it's easier to look up function f(…) {…} in the code than let f = function(…) {…};. Function Declarations are more "eye-catching".
- …But if a Function Declaration does not suit us for some reason, or we need a conditional declaration (we've just seen an example), then Function Expression should be used.

ARROW FUNCTIONS, THE BASICS

```
let func = (arg1, arg2, ..., argN) => expression;

let func = function(arg1, arg2, ..., argN) {
  return expression;
};
```

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

/* This arrow function is a shorter form of:

let sum = function(a, b) {
  return a + b;
};
*/

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

```
let double = n => n * 2;
// roughly the same as: let double = function(n) { return n * 2 }

alert( double(3) ); // 6
```

```
let sayHi = () => alert("Hello!");

sayHi();
```

ARROW FUNCTIONS, THE BASICS

```
let age = prompt("What is your age?", 18);

let welcome = (age < 18) ?
  () => alert('Hello!') :
  () => alert("Greetings!");

welcome();
```

```
let sum = (a, b) => {  // the curly brace opens a multiline function
  let result = a + b;
  return result; // if we use curly braces, then we need an explicit "return"
};

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