**Syntax, Functions and Statements**

**Operators, Parameters, Return Value, Arrow Functions**

1. **What is JavaScript?**

▪ JavaScript (JS) is a high-level programming language

▪ One of the core technologies of the World Wide Web

▪ Enables interactive web pages and applications

▪ Can be executed on the server and on the client

▪ Features:

▪ C-like syntax (curly-brackets, identifiers, operator)

▪ Multi-paradigm (imperative, functional, OOP)

▪ Dynamic typing

1. **Dynamic programming language**

▪ Operations otherwise done at compile-time can be done at

run-time

▪ It is possible to change the type of a variable or add new

properties or methods to an object while the program

is running

▪ In static programming languages, such changes are normally

not possible

1. **What is Node.js?**

▪ Server-side JavaScript runtime

▪ Chrome V8 JavaScript engine

▪ NPM package manager

▪ Install node packages

1. **Data Types**

▪ Seven data types that are primitives

▪ String - used to represent textual data

▪ Number - a numeric data type

▪ Boolean - a logical data type

▪ Undefined - automatically assigned to variables

▪ Null - represents the intentional absence of any object value

▪ BigInt - represent integers with arbitrary precision

▪ Symbol - unique and immutable primitive value

▪ Reference types – Object

1. **Identifiers**

An identifier is a sequence of characters in the code that

identifies a variable, function, or property

▪ In JavaScript, identifiers are case-sensitive and can contain

Unicode letters, $, \_, and digits (0-9), but may not start with

a digit.

1. **Variables**

Used to store data values

▪ Variables that are assigned a non-primitive value are

given a reference to that value

▪ Undefined - a variable that has been declared with a

keyword, but not given a value

▪ Undeclared - a variable that hasn't been declared at all

let, const and var are used to declare variables

▪ let - allows reassignment

▪ const - once assigned it cannot be modified

▪ var - defines a variable in the function scope

regardless of block scope

You will see var used in old examples

▪ Using var to declare variables is a legacy technique

▪ Since ES2015 keywords let and const are available

▪ var introduces function scope hoisting

▪ Will be discussed later in the lesson

▪ There is no good reason to ever use var

1. **Scopes**

▪ Global scope – Any variable that’s NOT inside any

function or block (a pair of curly braces);

▪ Functional scope – Variable declared inside a

function is inside the local scope;

▪ Block scope – let and const declares block

scoped variables

1. **Dynamic Typing**

Variables in JavaScript are not directly associated

with any particular value type.

▪ Any variable can be assigned (and re-assigned)

values of all types

▪ NOTE: The use of dynamic typing is considered a

bad practice!

1. **Functions**

▪ Function - named list of instructions (statements

and expressions)

▪ Can take parameters and return result

▪ Function names and parameters use camel case

▪ The { stays at the same line

function printStars(count) {

console.log("\*".repeat(count));

}

Invoke the function:

printStars(10);

**Declaring Functions:**

Function declaration:

function walk() {

console.log("walking");

}

Function expression:

const walk = function () {

console.log("walking");

}

Arrow Function:

const walk = () => {

console.log("walking");

}

**Parameters and Returned Value:**

You can receive parameters with no value

▪ The unused parameters are ignored:

function foo(a,b,c){

console.log(a);

console.log(b);

console.log(c); //undefined

}

foo(1,2)

function foo(a,b,c){

console.log(a);

console.log(b);

console.log(c);

}

foo(1,2,3,6,7)

Functions can yield a value with the return operator:

function identity(param){

return param;

}

console.log(identity(5)) // 5

**Object Methods and Standard Library**

▪ Any object may have methods

▪ Functions that operate from the context of the object

▪ Accessed as a property using the dot-notation

let myString = 'Hello, JavaScript!';

console.log(myString.toLowerCase());

// hello, javascript!

JavaScript has a large standard library

▪ Math, Number, Date, RegExp, JSON and more

▪ For more information, visit MDN

**Default Function Parameter Values**

▪ Functions can have default parameter values

function printStars(count = 5) {

console.log("\*".repeat(count));

}

printStars(); // \*\*\*\*\*

printStars(2); // \*\*

printStars(3, 5, 8); // \*\*\*

1. **Operators and Statements**

Arithmetic operators - take numerical values (either

literals or variables) as their operands

▪ Return a single numerical value

▪ Addition (+)

▪ Subtraction (-)

▪ Multiplication (\*)

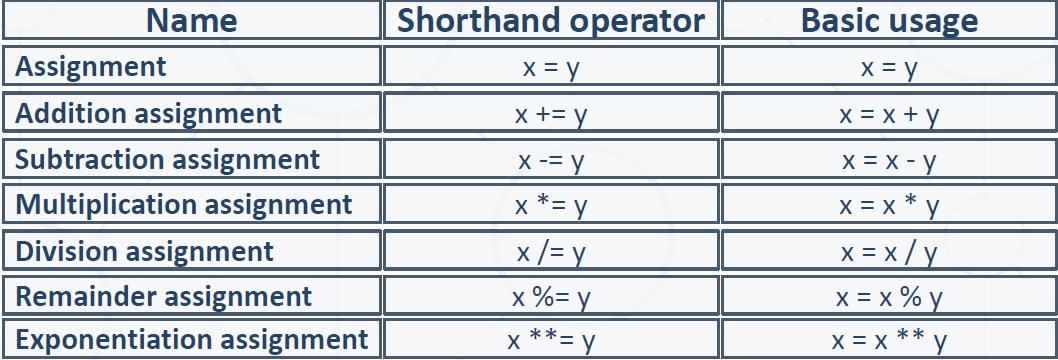
▪ Division (/)

▪ Remainder (%)

▪ Exponentiation (\*\*)

**Assignment operators** - assign a value to its left

operand based on the value of the right operand



**Comparison operators**

A screenshot of a calculator

AI-generated content may be incorrect.

**Logical operators:**

▪ && (logical AND) - returns the leftmost "false" value

or the last truthy value, if all are true

|| (logical OR) - returns the leftmost "true" value or

the last falsy value, if all are false

▪ ! (logical NOT) - Returns false if its single operand

can be converted to true; otherwise, returns true

**typeof operator**

The typeof operator returns a string indicating the

type of an operand

**Special cases:**

console.log(typeof NaN); // number

console.log(NaN === NaN); // false

console.log(typeof null); // object(legacy reasons)

console.log(new Array() == false); // true

console.log(0.1 + 0.2); // 0.30000000000000004

console.log((0.2 \* 10 + 0.1 \* 10) / 10); // 0.3

const variable = []; // empty array

console.log(variable == false); // evaluates true

if (variable) { console.log('True!') };

1. **Truthy and falsy values:**

"truthy" - a value that coerces to true when evaluated in a

boolean context

▪ The following values are "falsy" - false, null, undefined, NaN, 0, 0n

and ""

1. **Loops**

▪ The for / while loops work as in C++, C# and Java

▪ Classical for-loop

for (let i = 0; i <= 5; i++) { console.log(i); }

// 0 1 2 3 4 5

JavaScript supports two more variants of the for-loop:

▪ for-of – used with arrays and iterators

▪ for-in – used with objects and associative arrays

▪ Both will be reviewed in upcoming lessons

1. **Mixing HTML & JavaScript**

<!DOCTYPE html>

<html>

<body>

<h1>JavaScript in the HTML page</h1>

<script>

for (let i=1; i<=10; i++) {

document.write(`<p>${i}</p>`);

}

</script>

</body>

</html>

47

<form>

num1: <input type="text" name="num1" /> <br>

num2: <input type="text" name="num2" /> <br>

sum: <input type="text" name="sum" /> <br>

<input type="button" value="Sum" onclick="calcSum()" />

</form>

function calcSum() {

let num1 = document.getElementsByName('num1')[0].value;

let num2 = document.getElementsByName('num2')[0].value;

let sum = Number(num1) + Number(num2);

document.getElementsByName('sum')[0].value = sum;

}

<!DOCTYPE html>

<html>

<head>

<script src="numbers.js">

</script>

</head>

<body>

<input type="submit"

onclick="printRandNum()" />

</body>

</html>

function printRandNum() {

let num = Math.round(

Math.random() \* 100);

document.body.innerHTML +=

`<div>${num}</div>`;

}

1. **Strict Mode**

▪ Strict mode limits certain "sloppy" language features

▪ Silent errors will throw Exception instead

Enabled by default in modules

1. **Language specifics**

First-class functions – a function can be passed as an

argument to other functions

▪ Can be returned by another function and can be

assigned as a value to a variableA screenshot of a computer code

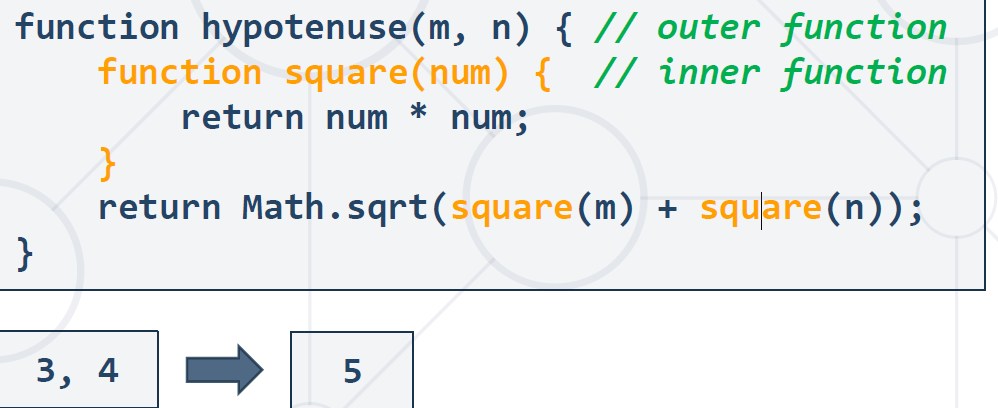
AI-generated content may be incorrect.

Nested functions

Functions can be nested - hold other functions

▪ Inner functions have access to variables from

their parent



1. **Hoisting**

* Variable and function declarations are put into

memory during the compile phase, but stay

exactly where you typed them in your code

* ▪Only declarations are hoisted