## Good Practices

Why inline JavaScript is bad? <https://stackoverflow.com/a/19002892> and <https://stackoverflow.com/a/19002885>

## Redirection

Redirect to another page:

<https://stackoverflow.com/a/20769622>

## Images

Position text on image: <https://www.w3schools.com/howto/howto_css_image_text.asp>

# Overall Syntax

## Identifiers

An identifier is a name you choose for variables, parameters, functions, classes, etc.

Rules: Like C++

## Statements

A statement is a code that declares a variable or instructs the JavaScript engine to do a task. A simple statement is terminated by a semicolon ;. For example:

let formatted = true;

if (formatted) {

    console.log('The code is easy to read');

}

Rules: Like C++. Except that the semicolon is optional.

## Blocks

A block is a sequence of zero or more simple statements. A block is delimited by a pair of curly brackets {}. For example:

if (window.localStorage) {

    console.log('The local storage is supported');

}

Rules: Like C++

## Comments

### Single-line comments

// This is a single-line comment

### Block comments

/\* This is a block comment

that can span multiple lines \*/

## Expressions

An expression is a piece of code that evaluates to a value.

Rules: Like C++.

## Keywords & Reserved Words

The following table shows the JavaScript reserved words defined in ECMA-262:

|  |  |  |
| --- | --- | --- |
| break | case | catch |
| continue | debugger | default |
| else | export | extends |
| function | if | import |
| new | return | super |
| throw | try | null |
| void | while | with |
| class | delete | finally |
| in | switch | typeof |
| yield | const | do |
| for | instanceof | this |
| var |  |  |

In addition to the reserved keywords, ECMA-252 also defines a list of **future reserved words** that cannot be used as identifiers or property names:

|  |  |  |
| --- | --- | --- |
| enum | implements | let |
| protected | private | public |
| await | interface | package |
| implements | public |  |

# Variables

## Declaration

To declare a variable, you use the var keyword followed by the variable name. For example:

var message;

By default, the message variable has a special value undefined if you have not assigned a value to it.

Starting in ES6, you can use the let keyword to declare a variable like this:

let message;

**Notes:**

* JavaScript is a **dynamically typed language**. This means that you don’t need to specify the variable’s type in the declaration like other static-typed languages such as C++, Java or C#.
* It’s **a good practice to use the let keyword** to declare a variable. Later, you’ll learn the differences between var and let keywords. And you should not worry about it for now.

## Initialization

After the variable is declared, it can be initialized with a value.

To initialize a variable, you specify the variable name, followed by an equals sign = and a value. For example, the following declares the message variable and initializes it with a literal string "Hello":

let message;

message = "Hello";

To declare and initialize a variable at the same time. For example:

let message = "Hello";

Tips:

* JavaScript allows you to declare two or more variables using a single statement. To separate two variable declarations, you use a comma , like this:

let message = "Hello",

    counter = 100;

* You can assign a value of a different type to a variable. Although, it is not recommended. For example:

let message = 'Hello';

message = 100;

## Undefined vs. Undeclared Variables

An undefined variable is a variable that has been declared but has not been initialized with a value. For example:

let message;

console.log(message);

Output:

undefined

In contrast, an undeclared variable is a variable that has not been declared. Hence, accessing it causes a ReferenceError. For example:

console.log(counter);

Output:

console.log(counter);

            ^

ReferenceError: counter is not defined

# Constants

A constant **holds a value that doesn’t change**. To declare a constant, you use the const keyword. When defining a constant, you need to initialize it with a value. For example:

const workday = 5;

Once you define a constant, you cannot change its value.

The following example attempts to change the value of the workday constant to 4 and causes an Uncaught TypeError error:

workday = 2;

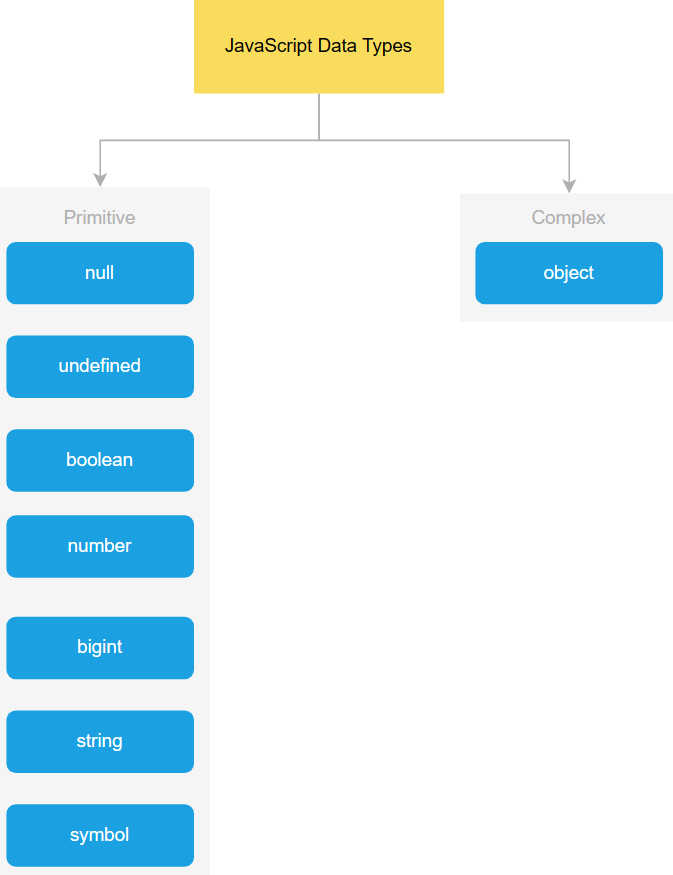
Output:

Error:

Uncaught TypeError: Assignment to constant variable.

Code language: JavaScript (javascript)

# Data Type



## Primitive Types

Tip: To get the current type of the value that the variable stores, you use the typeof operator:

### The undefined type

The undefined type has only one value undefined. By default, **when a variable is declared but not initialized**, it's assigned the value of undefined.

For example:

let counter;

console.log(counter);               // undefined

console.log(typeof counter);        // undefined

### The null type

The null type has only one value null. For example:

let obj = null;

console.log(typeof obj);     // object

**Note**: JavaScript defines that null is equal to undefined:

console.log(null == undefined);     // true

**You might not know:**

The typeof null returns object is a known bug in JavaScript. A proposal to fix this was proposed but rejected. The reason was the that fix would break a lot of existing sites.

### The number type

JavaScript uses the number type to represent both integer and floating-point numbers.

For example:

let num = 100;      // integer

let price= 12.5;    // float

**Notes**:

* JavaScript automatically converts a floating-point number into an integer number if the number appears to be a whole number. The reason is that Javascript always wants to use less memory since a floating-point value uses twice as much memory as an integer value. For example:

let price = 200.00;         // interpreted as an integer 200

**Tips**:

* To get the range of the number type, you use Number.MIN\_VALUE and Number.MAX\_VALUE. For example:

console.log(Number.MAX\_VALUE);  // 1.7976931348623157e+308

console.log(Number.MIN\_VALUE);  // 5e-324

* Also, you can use Infinity and -Infinity to represent the infinite number. For example:

console.log(Number.MAX\_VALUE + Number.MAX\_VALUE);       // Infinity

console.log(-Number.MAX\_VALUE - Number.MAX\_VALUE);      // -Infinity

#### NaN

NaN stands for Not a Number. It is a special numeric value that indicates an invalid number. For example, the division of a string by a number returns NaN:.

console.log('a'/2);         // NaN

The NaN has two special characteristics:

* Any operation with NaN returns NaN:

console.log(NaN/2); // NaN

* The NaN does not equal any value, including itself.

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

More details:

<https://www.javascripttutorial.net/javascript-number/>

<https://www.javascripttutorial.net/es-next/javascript-numeric-separator/>

<https://www.javascripttutorial.net/es6/octal-and-binary-literals/>

### The string type

A string is a sequence of zero or more characters. A string literal begins and ends with either a single quote (') or a double quote ("). For examples:

let greeting = 'Hi';

let message  = "Bye";

If you want to single quote or double quotes in a literal string, you need to use the backslash to escape it. For example:

let message = 'I\'m also a valid string';   // use \ to escape the single quote (')

**Notes**:

* JavaScript strings are immutable. This means that it **cannot be modified once created**.

let s = 'JavaScript';

s[0] = 'j';

console.log(s) // still 'JavaScript'

* However, you **can** **create a new string from an existing string**. For example:

let str = 'JavaScript';

str = str + ' String'; // Javascript String

Behind the scene, the JavaScript engine creates a new string that holds the new string 'JavaScript String' and destroys the original strings 'JavaScript' and ' String'.

More details: <https://www.javascripttutorial.net/javascript-string/>

### The boolean type

The boolean type has two literal values: true and false. For example:

let completed = false;

console.log(typeof completed);      // boolean

**Tips**:

* JavaScript allows to **convert values of other types into boolean values** of true or false using the Boolean() function. The following table shows the conversion rules:

|  |  |  |
| --- | --- | --- |
| **Type** | **true** | **false** |
| string | non-empty string | empty string |
| number | non-zero number and Infinity | 0, NaN |
| object | non-null object | null |
| undefined |  | undefined |

For example:

console.log(Boolean('Hi'));     // true

console.log(Boolean(''));       // false

console.log(Boolean(20));       // true

console.log(Boolean(Infinity));     // true

console.log(Boolean(0));         // false

console.log(Boolean({foo: 100}));   // true on non-empty object

console.log(Boolean(null));     // false

More details: <https://www.javascripttutorial.net/javascript-boolean-type/>

### The symbol type

JavaScript added a primitive type in ES6: the symbol. Different from other primitive types, the symbol type does not have a literal form.

To create a symbol, you call the Symbol() function which **creates a new unique value every time you call it**:

let s1 = Symbol();

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

### The bigint type

The bigint type represents the **whole numbers that are larger than 253 – 1**. To form a bigint literal number, you append the letter n at the end of the number:

let pageView = 9007199254740991n;

console.log(typeof(pageView)); // 'bigint'

## Complex Types

### The object type

An object is a collection of properties, where each property is defined as **a key-value pair**.

The following example defines an empty object using the object literal syntax:

let emptyObject = {};

The following example defines the person object with two properties: firstName and lastName.

let person = {

    firstName: 'John',

    lastName: 'Doe'

};

To access a object’s property, you can use:

* The dot notation (.):

console.log(person.firstName);

console.log(person.lastName);

If you reference a property that does not exist, you’ll get an undefined value. For example:

console.log(person.age); // undefined

* The array-like notation ([]):

console.log(person['lastName']);

console.log(person['firstName']);

**Notes**:

* The property name of an object can be **any string**. You can use quotes around the property name if it's not a valid identifier. For example, if the person object has a property first-name, you must place it in the quotes such as "first-name".
* The property value of an object **can hold another object**. For example, the following address property holds an object that has street, city, state, and country properties:

let contact = {

    firstName: 'John',

    lastName: 'Doe',

    phone: '(408)-555-9999',

    address: {

        street: 'North 1st street',

        city: 'San Jose',

        state: 'CA',

        country: 'USA'

    }

}

More details: <https://www.javascripttutorial.net/javascript-objects/>

# Array

<https://www.javascripttutorial.net/javascript-array/>

# Primitive vs. Reference Values

<https://www.javascripttutorial.net/javascript-primitive-vs-reference-values/>

[JavaScript Tutorial](https://www.javascripttutorial.net/)