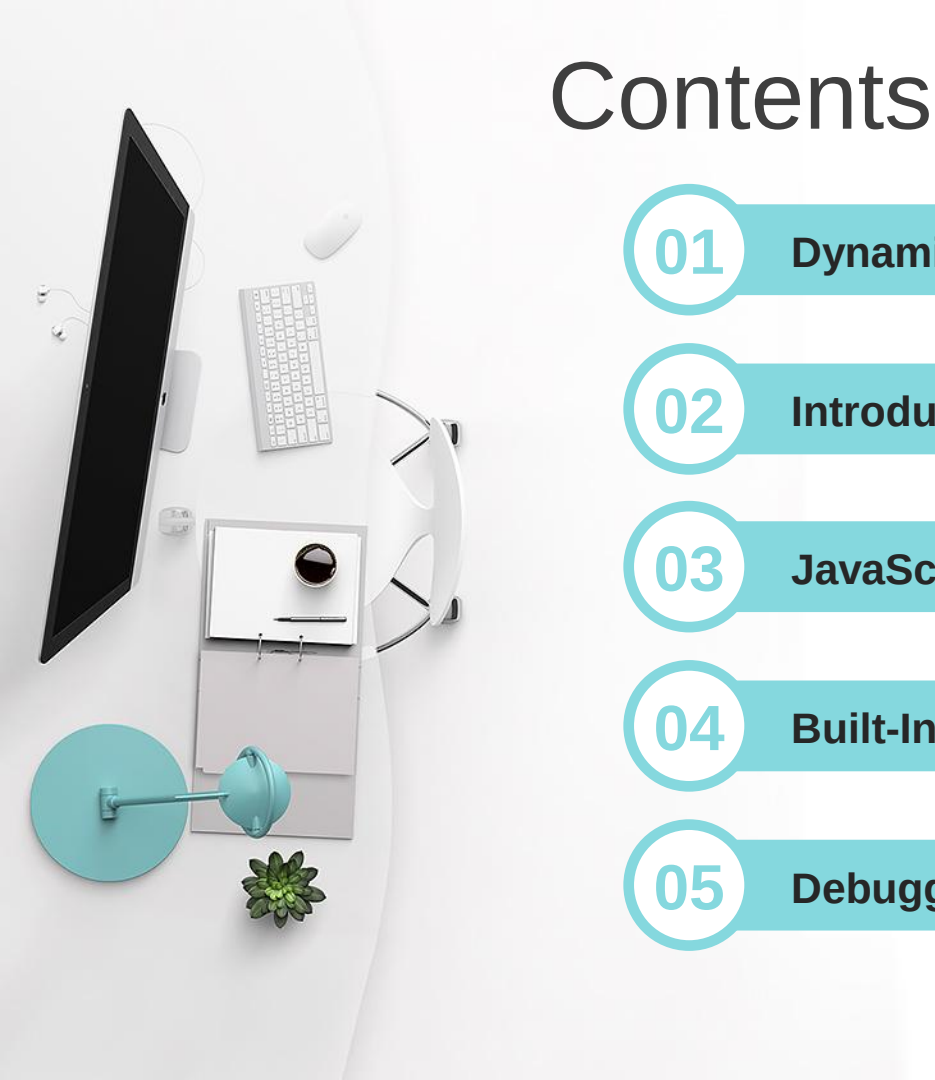
Front-end Development

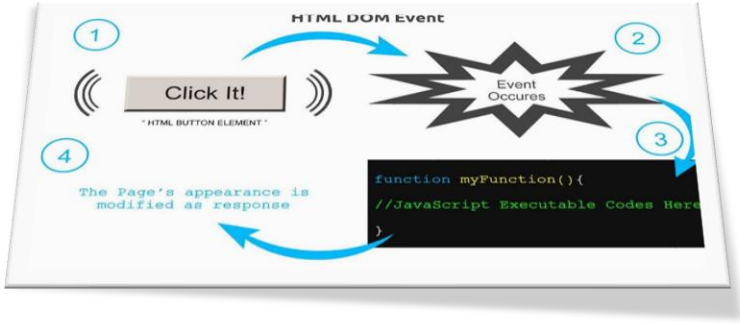
Lecturer: Ung Văn Giàu
Email: giau.ung@eiu.edu.vn

Contents

Introduction to JavaScript Development



Contents



1. Dynamic HTML

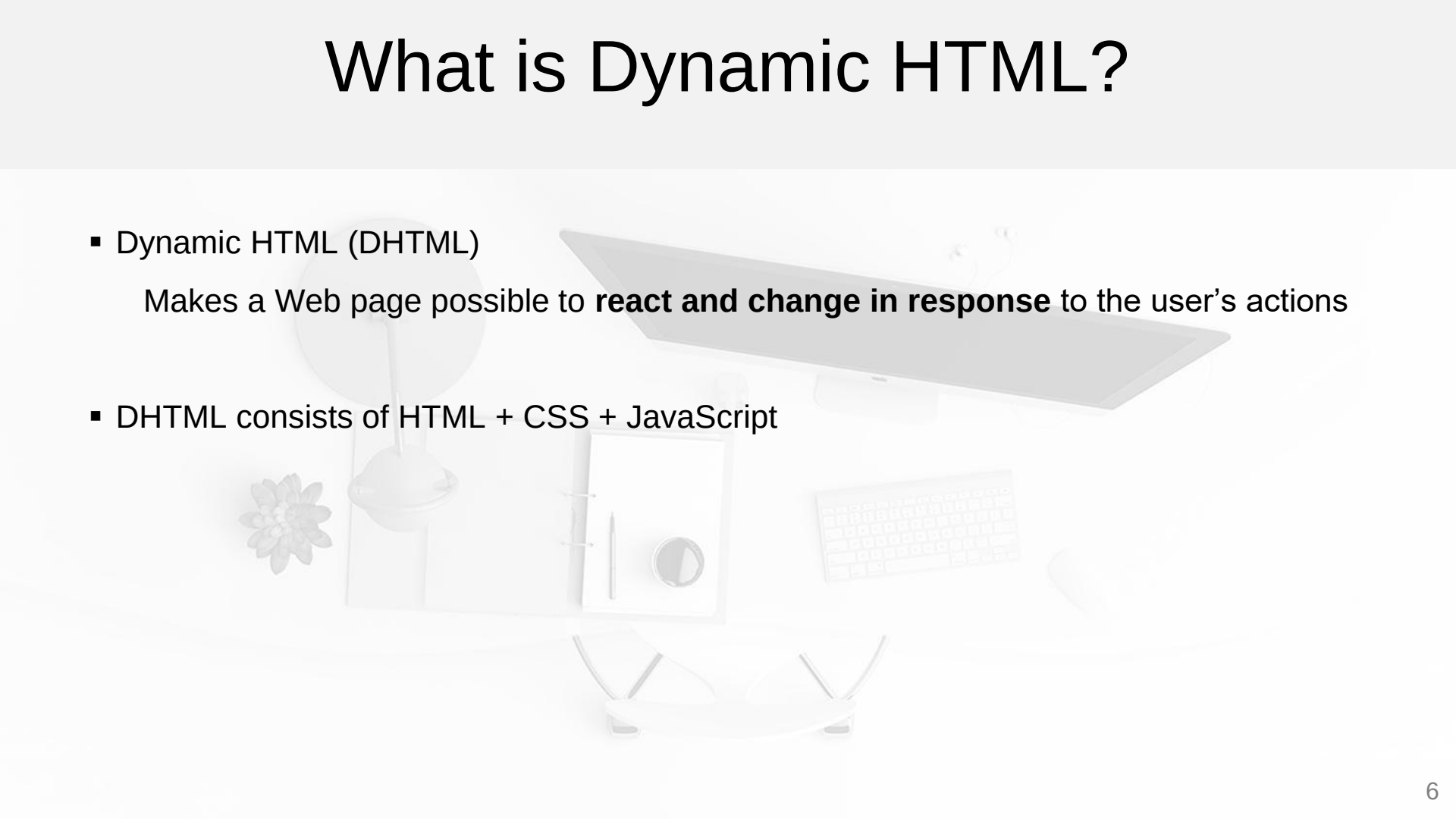
Dynamic Behavior at the Client Side

What is Dynamic HTML?

- Dynamic HTML (DHTML)

  Makes a Web page possible to **react and change in response** to the user's actions

- DHTML consists of HTML + CSS + JavaScript



DTHML = HTML + CSS + JavaScript

- **HTML** defines Web sites **content** through **semantic** tags (headings, paragraphs, lists, etc.)

- **CSS** defines **‘rules’** or **‘styles’** for **presenting** every aspect of an HTML document
  - Font (family, size, color, weight, etc.)
  - Background (color, image, position, repeat)
  - Position and layout (of any object on the page)

- **JavaScript** defines dynamic behavior

  Programming logic for interaction with the user, to handle events, etc.

2. JavaScript

Dynamic Behavior in a Web Page

JavaScript

- JavaScript is a **front-end scripting language** developed by Netscape for dynamic content
  - Lightweight, but with limited capabilities
  - Can be used as object-oriented language
  - Embedded in your HTML page
  - Interpreted by the Web browser
- Client-side, mobile and desktop technology
- Simple and flexible
- Powerful to manipulate the DOM

JavaScript Advantages

JavaScript allows interactivity such as:

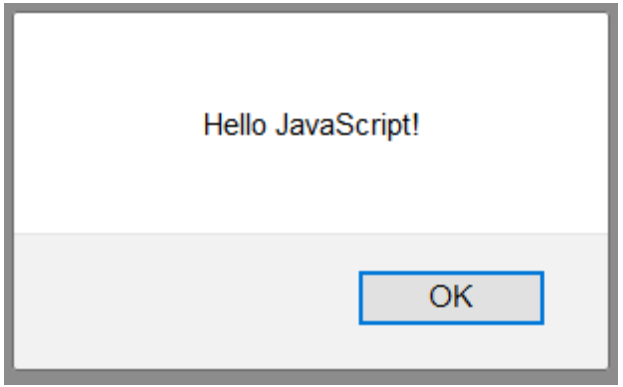
- Implementing form validation
- React to user actions, e.g., handle keys
- Changing an image when moving mouse over it
- Sections of a page appearing and disappearing
- Content loading and changing dynamically
- Performing complex calculations
- Custom HTML controls, e.g., scrollable table
- Implementing AJAX functionality

What Can JavaScript Do?

- Can handle events
- Can read and write HTML elements and modify the DOM tree
- Can validate form data
- Can access / modify browser cookies
- Can detect the user's browser and OS
- Can be used as object-oriented language
- Can handle exceptions
- Can perform asynchronous server calls (AJAX)

The First Script

```
<!DOCTYPE html>
<html>
    <body>
        <script type="text/javascript">
            alert('Hello JavaScript!');
        </script>
    </body>
</html>
```



Using JavaScript Code

The JavaScript code can be placed in:

- **<script>** tag in the **head**
- **<script>** tag in the **body** - not recommended
- External files, **linked** via **<script>** tag the head
  - Files usually have **.js** extension
  - Highly **recommended**
  - The .js files get **cached** by the browser

```
<script src="./js/scripts.js" type="text/javascript">
    <!-- code placed here will not be executed! -->
</script>
```

JavaScript – When is Executed?

- JavaScript code is executed during the **page loading** or when the **browser fires an event**
  - All statements are executed at page loading
  - Some statements just define functions that can be called later
  - No compile time checks

- **Function calls** or **code can be attached** as “event handlers” **via tag attributes**

  Executed when the event is fired by the browser

```
<img src="./imgs/flower.jpg" onclick="alert('clicked!')" />
```

Calling a JavaScript Function from Event Handler

Example

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            function test(message) {
                alert(message);
            }
        </script>
    </head>

    <body>
        <img src="./images/clickme.jpeg" onclick="test('Clicked!')" />
    </body>
</html>
```

JavaScript - When is Executed?

- Using external script files:

```
<!DOCTYPE html>
<html>
    <head>
        <script src="./js/sample.js" type="text/javascript"></script>
    </head>
    <body>
        <button onclick="sample()">Call JavaScript function
            from sample.js</button>
    </body>
</html>
```

- External JavaScript file:

```
function sample() {
    alert('Hello from sample.js!');
}
```

3. JavaScript Syntax

JavaScript Syntax

The JavaScript syntax is similar to C#, C++, Java

- **Operators** (+, *, =, !=, &&, ++,…)
- **Variables** (typeless)
- **Conditional statements** (if, else)
- **Loops** (for, while)
- **Arrays** (my_array[]) and **associative arrays** (my_array[‘abc’])
- **Functions** (can return value)

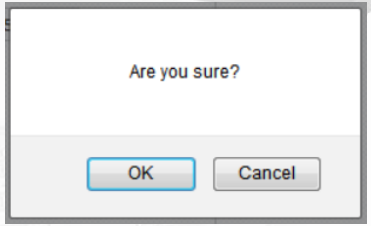
Note:

A **semicolon** at the end of a line indicates where a statement ends; it is only absolutely required when you need to separate statements on a single line.

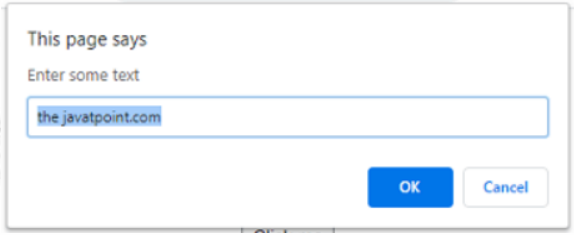
Standard Pop-up Boxes

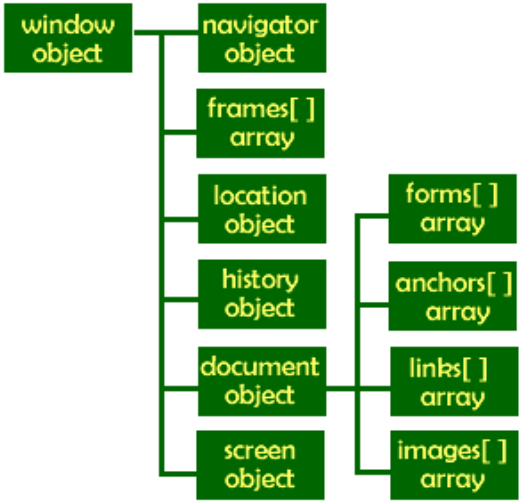
- **Alert box** with text and **[OK]** button
  - Just a message shown in a dialog box
  - **alert**(“Some text here”);

- **Confirmation box**
  - Contains text, **[OK]** button and **[Cancel]** button
  - **confirm**(“Are you sure?”);



- **Prompt box**
  - Contains text, input field with default value
  - **prompt** (“enter amount”, 10);





4. Built-In Browser Objects

Built-in Browser Objects

The browser provides some read-only data via:

- **window**
  - The top node of the DOM tree
  - Represents the browser's window
- **document**

  Holds information the current loaded document
- **screen**

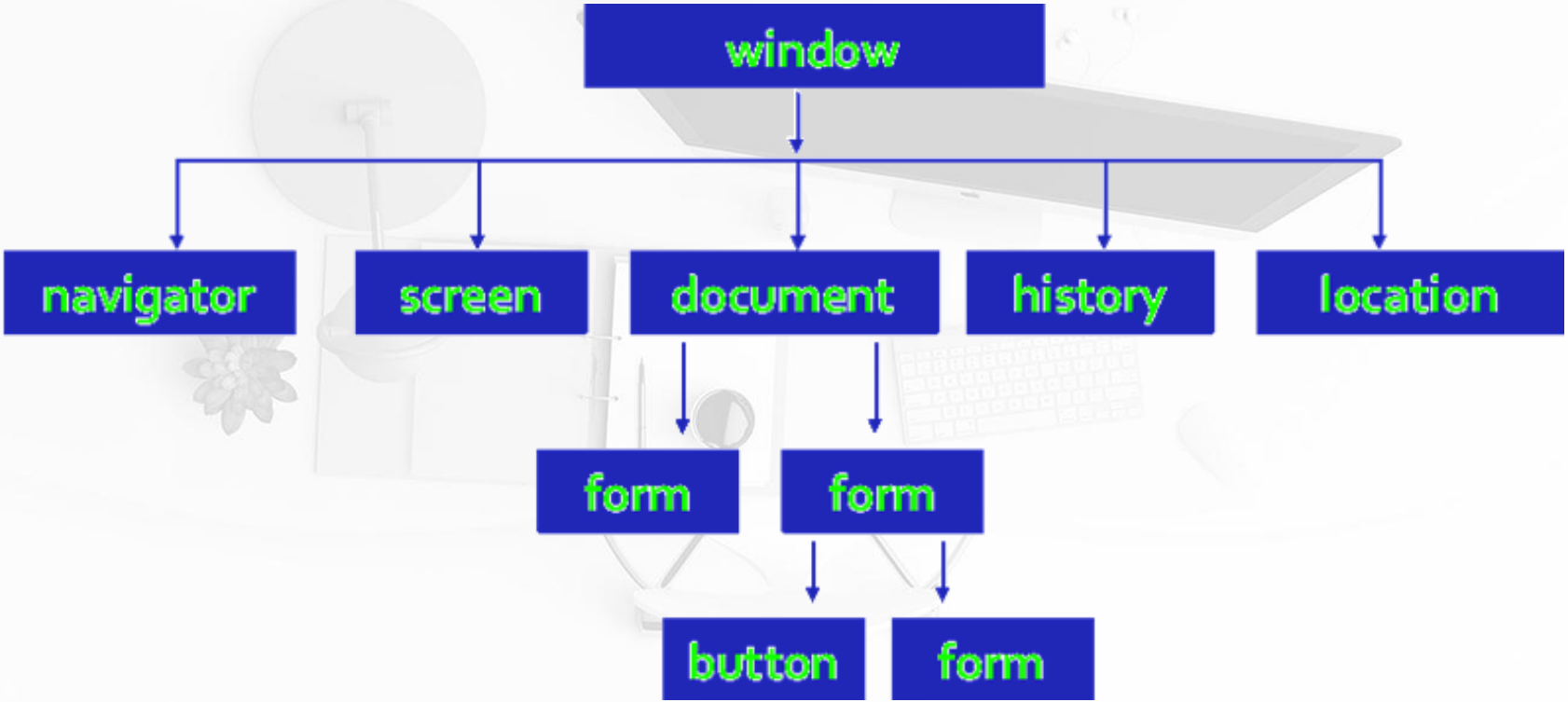
  Holds the user's display properties
- **browser**

  Holds information about the browser
- **location**

  Contains information about the current URL

DOM Hierarchy

Example



The Math Object

The Math object provides some **mathematical functions**

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            for (i = 1; i <= 20; i++) {
                var x = Math.random();
                x = 10 * x + 1;
                x = Math.floor(x);
                document.write("Random number (" + i + ") in range "
                    + "1..10 --> " + x + "<br/>");
            }
        </script>
    </head>
    <body>
    </body>
</html>
```

The Date Object

The Date object provides **date / calendar functions**

```
<!DOCTYPE html>
<html>
    <body>
        <p id="timeField"></p>

        <script type="text/javascript">
            var now = new Date();
            var result = "It is now " + now;
            document.getElementById("timeField").innerText = result;
        </script>
    </body>
</html>
```

The Date Object

Make something happen (once) after a fixed delay

```
var timer = setTimeout(functionName, milliSeconds);
```

- 5 seconds after this statement executes, this function is called

```
var timer = setTimeout(bang, 5000);
```

- Cancels the timer

```
clearTimeout(timer);
```

The Date Object

Make something happen repeatedly at fixed intervals

```
var timer = setInterval(functionName, milliSeconds);
```

- This function is called continuously per 1 second

```
var timer = setInterval(clock, 1000);
```

- Stop the timer

```
clearInterval(timer);
```

Timer

Example

```
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <p id="clock"></p>
        <script type="text/javascript">
            function timerFunc() {
                var now = new Date();
                var hour = now.getHours();
                var min = now.getMinutes();
                var sec = now.getSeconds();
                document.getElementById("clock").innerHTML =
                "" + hour + ":" + min + ":" + sec;
            }
            setInterval(timerFunc, 1000);
        </script>
    </body>
</html>
```

5. Debugging JavaScript

Debugging JavaScript

- Modern browsers have **JavaScript console** where errors in scripts are reported

  Errors may differ across browsers

- Several tools to debug JavaScript:
  - Firefox Developer Tools
  - Chrome DevTools

JavaScript Console Object

- The **console** object exists only if there is a debugging tool that supports it

  Used to write log messages at runtime

- **Methods** of the **console** object:
  - debug(message)
  - info(message)
  - **log**(message)
  - warn(message)
  - error(message)

Data Types and Variables

Contents

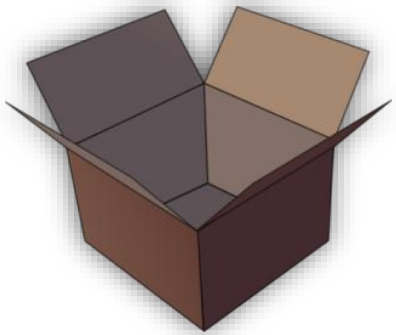
1. Data Types in JavaScript

JavaScript Data Types

JavaScript is **weakly typed** language

- allows most operations on values without regards to their types
- **values have types, variables don't**
- variables can hold any type of value
- All variables are declared with the keywords **var**, **let** or **const**

```
var count = 5; // variable holds an integer value
count = 'hello'; // the same variable now holds a string

var name = 'Telerik Academy'; // variable holds a string

let mark = 5.25 // mark holds a floating-point number
mark = true; // mark now holds a boolean value

const MAX_COUNT = 250; // name is a constant variable that holds a string
MAX_COUNT = 0; // error, cannot assign to a constant variable
```



2. Introducing Variables

What Is a Variable?

- A variable is a:
  - **Placeholder of information** that can usually be changed at run-time
  - A piece of computer memory holding some value

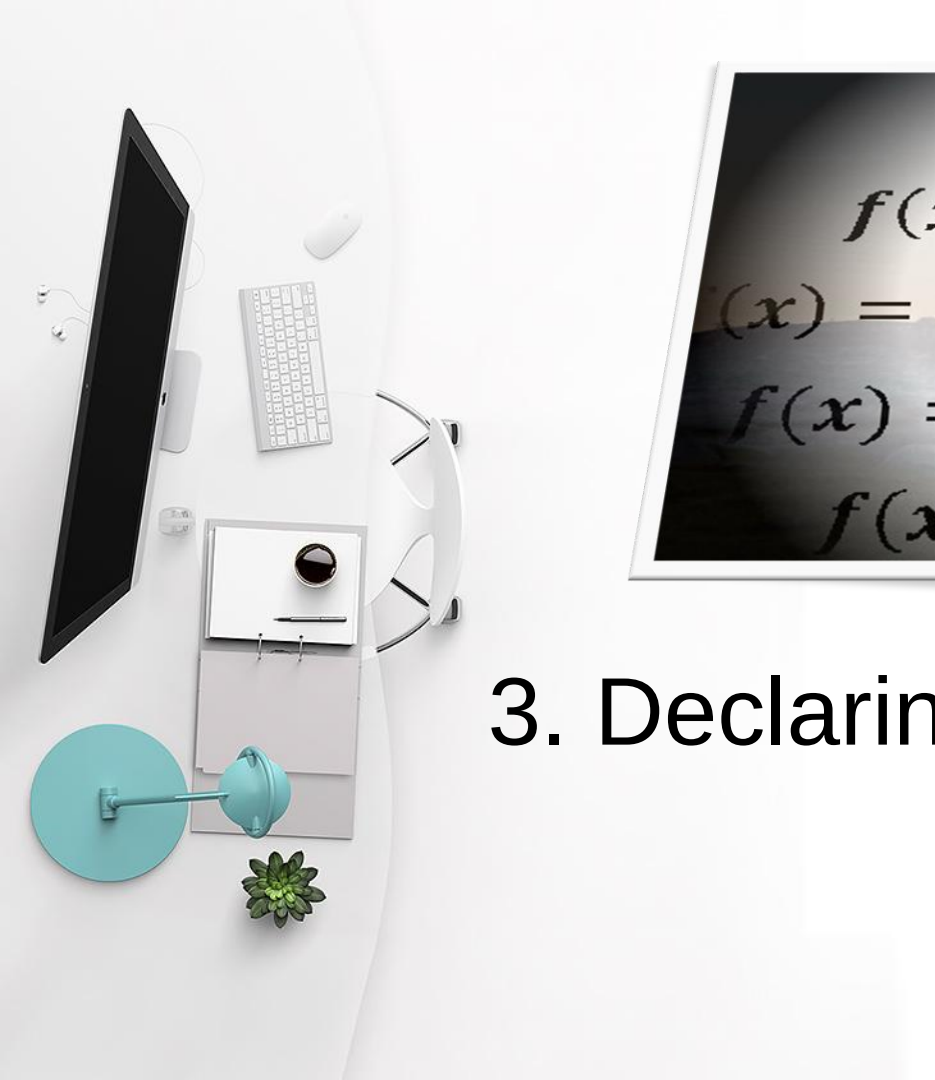
- Variables allow you to:
  - **Store** information
  - **Retrieve** the stored information
  - **Manipulate** the stored information

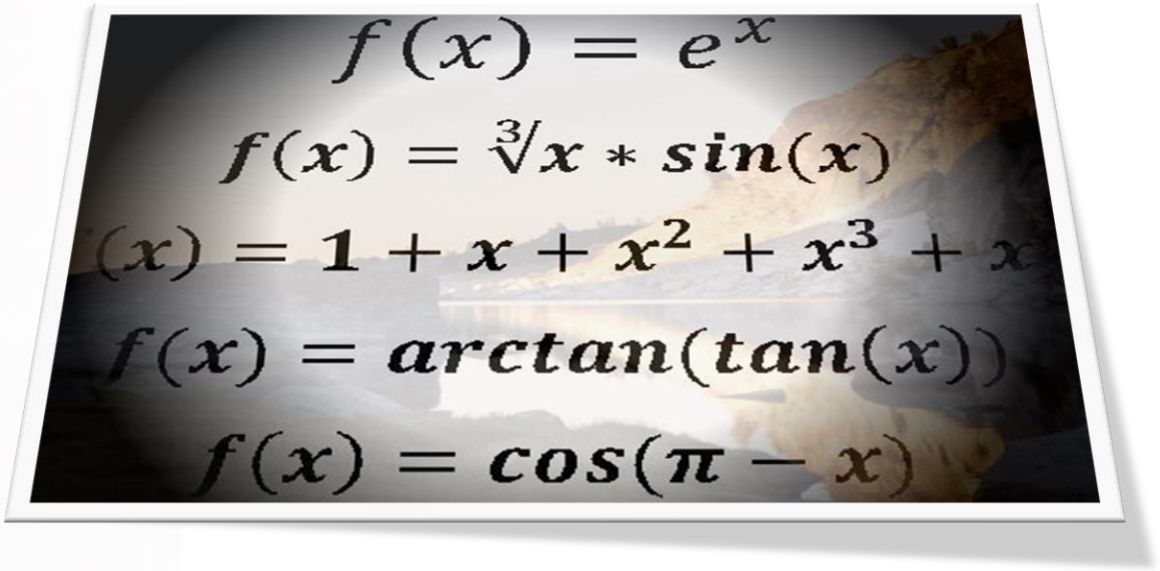
Variable Characteristics

- A variable has:
  - Name
  - Value

- Example: let count = 5;
  - Name: counter
  - Value: 5

    Type of the counter's value: number





3. Declaring and Using Variables

Declaring Variables

- When declaring a variable, we:
  - Specify its **name** (called identifier)
  - May give it an **initial value**

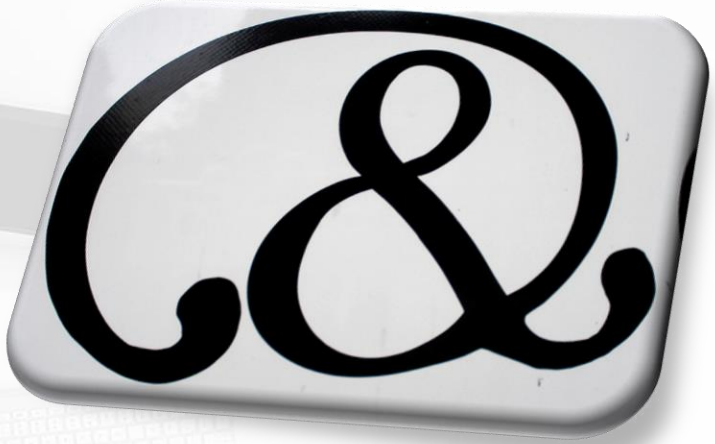
- The **syntax** is the following:

<var | let | const> <identifier> [= <initialization>];

```
let emptyVariable;
var height = 200;
let width = 300;
const depth = 250;
```

Identifiers

- Identifiers may consist of:
  - Letters (**Unicode**)
  - Digits [**0**-**9**]
  - Underscore '_'
  - Dollar '**$**'



- Identifiers
  - Can **begin only** with a **letter**, **$**, or an **underscore**
  - **Cannot** be a JavaScript **keyword**

- Variables / functions names: use **camelCase**

Identifiers

- Identifiers
  - Should have a **descriptive name**
  - It is recommended to **use only Latin letters**
  - Should be **neither too long nor too short**

- Names in JavaScript are **case-sensitive**

  Small letters are considered different than the capital letters

Identifiers

Examples

- Examples of **correct** identifiers:

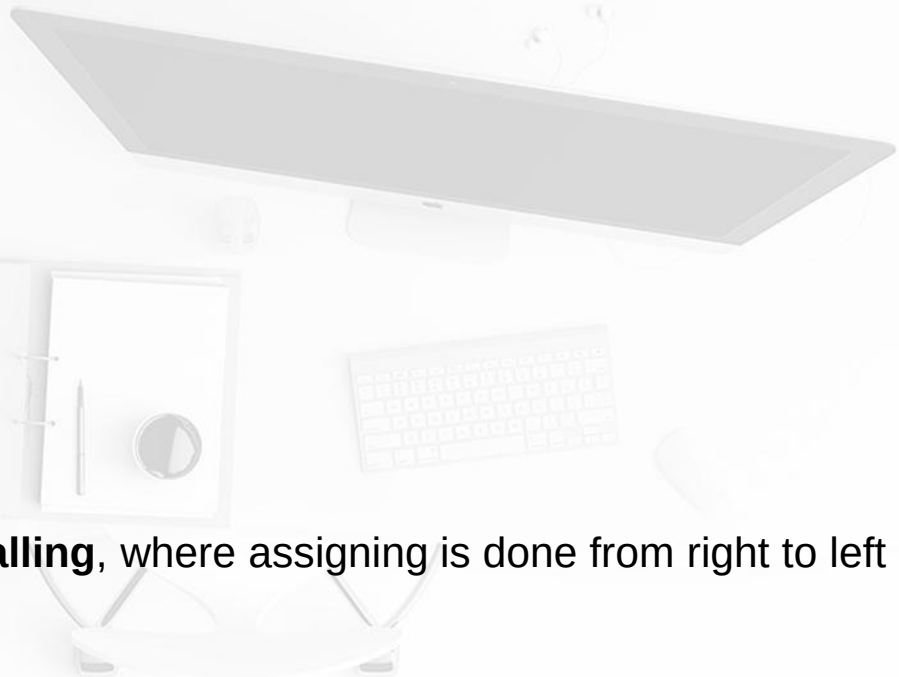
```
let New = 2; // Here N is capital
let _2Pac; // This identifier begins with _
let поздрав = 'Hello'; // Unicode symbols used
// The following is more appropriate:
let greeting = 'Hello';
let n = 100; // Undescriptive
let numberOfClients = 100; // Descriptive
// Overdescriptive identifier:
let numberOfPrivateClientOfTheFirm = 100;
```

- Examples of **incorrect** identifiers:

```
let new;     // new is a keyword
let 2Pac;     // Cannot begin with a digit
```

Assigning Values To Variables

Assigning Values



- Assigning values to variables

  Is achieved by the **= operator**

- The = operator has
  - Variable **identifier** on the **left**
  - **Value** on the **right**

    Can be of **any value type**
  - Could be used in a **cascade calling**, where assigning is done from right to left

- Variables declared with the **const** keyword **cannot be reassigned** after their initial assignment

Assigning Values

Examples

Assigning values example:

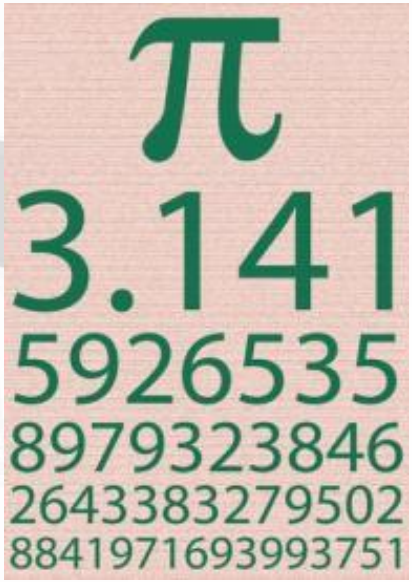
```
let firstValue = 5;
let secondValue;
let thirdValue;

// Using an already declared variable:
secondValue = firstValue;

// The following cascade calling assigns
// 3 to firstValue and then firstValue
// to thirdValue, so both variables have
// the value 3 as a result:

thirdValue = firstValue = 3; // Avoid this!
```

Initializing Variables

- Initialization
  - Assignment of initial value
  - Must be done before the variable is used!

- Several ways of initializing a variable:
  - By using a **literal expression**
  - By **referring** to an already **initialized variable**

- **Uninitialized** variables are **undefined**



Initialization

Examples

Example of some initializations:

```
// This is how we use a literal expression:
let heightInMeters = 1.74;

// Here we use an already initialized variable:
let greeting = 'Hello World!';
let message = greeting;

// Use a result from an expression
const parsedNumber = parseInt('1239') + 1;
```

Local and Global Variables

- **Local** variables - declared with the keywords **var**, **let** or **const**
  - **var** - the variable lives in the scope of the current **function** or in the **global scope**
  - **let** - the variables lives in the **current (block) scope**, and cannot redeclare
  - **const** - like let, but cannot be reassigned

```
let a = 5; // a is local in the current scope
a = 'alabala'; // the same a is referenced here
```

- **Note:**
  - Duplicate variable declarations using **var** will not trigger an error
  - Variables declared by **let** have their scope in the block for which they are declared

Local and Global Variables

Example

```
function varTest() {
    var x = 1;
    {
        var x = 2;  // same variable!
        console.log(x);  // 2
    }
    console.log(x);  // 2
}

function letTest() {
    let x = 1;
    {
        let x = 2;  // different variable
        console.log(x);  // 2
    }
    console.log(x);  // 1
}

// If you use var to declare a variable
var myName = 'Chris';
var myName = 'Bob'; // You can do it

// If you use let to declare a variable
let myName = 'Chris';
let myName = 'Bob'; // You can't do it
```

Local and Global Variables

- **Global** variables
  - Declared **without** any **keyword**
  - Bad practice - **never do this**!

```
a = undefined;
a = 5; // the same as window.a = 5;
```

Numbers in JavaScript

Numbers in JavaScript

- All numbers in JavaScript are stored internally as double-precision floating-point numbers

- According to the IEEE-754 standard

  Can be wrapped as objects of type **Number**

- Example:

```
let value = 5;
value = 3.14159;
value = new Number(100); // Number { 100 }
value = value + 1; // 101
let biggestNum = Number.MAX_VALUE;
```

Numbers Conversion

- Convert **floating-point** to **integer** number

```
let valueDouble = 8.75;
let valueInt = valueDouble | 0; // 8
```

- Convert to **integer** number with **rounding**

```
let valueDouble = 8.75;
let roundedInt = (valueDouble + 0.5) | 0; // 9
```

- Convert **string** to **integer**

```
let str = '1234';
let i = str | 0 + 1; // 1235
```

Integer numbers

What are Integer numbers?

- Integer numbers in JavaScript:
  - Represent **whole numbers**
  - Have range of values, depending on the size of memory used

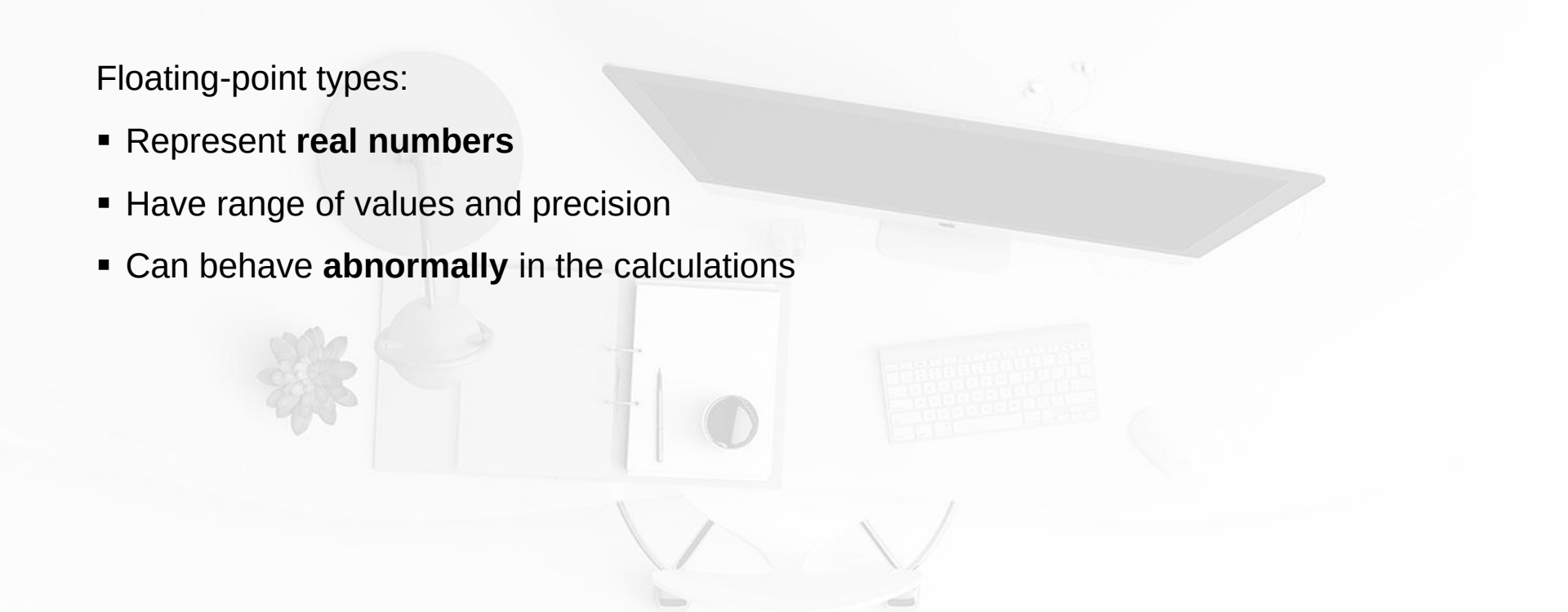
- Integer values can hold numbers from **-9007199254740992** to **9007199254740992**

  Their underlying type is a floating-point number (IEEE-754)

```
let studentsCount = 5;
let maxInteger = 9007199254740992;
let minInteger = -9007199254740992;
let a = 5, b = 3;
let sum = a + b; // 8
let div = a / 0; // Infinity
```

Floating-Point numbers

What are Floating-Point numbers?



Floating-point types:

- Represent **real numbers**
- Have range of values and precision
- Can behave **abnormally** in the calculations

Floating-Point numbers

- Floating-point size depend on the platform
  - The browser and the OS
- 32-bit OS and browser have 32 bits for number, while 64-bit have 64 bits
  - It is good idea to use up to 32-bit numbers
    - Will always work on all platforms

Floating-Point Types

Example

The floating-point type can hold numbers from 5e-324 to 1.79e+308

```
let PI = Math.PI; // 3.141592653589793
let minValue = Number.MIN_VALUE; // 5e-324
let maxValue = Number.MAX_VALUE; // 1.79e+308
let div0 = PI / 0; // Infinity
let divMinus0 = -PI / 0; // -Infinity
let unknown = div0 / divMinus0; // NaN
```

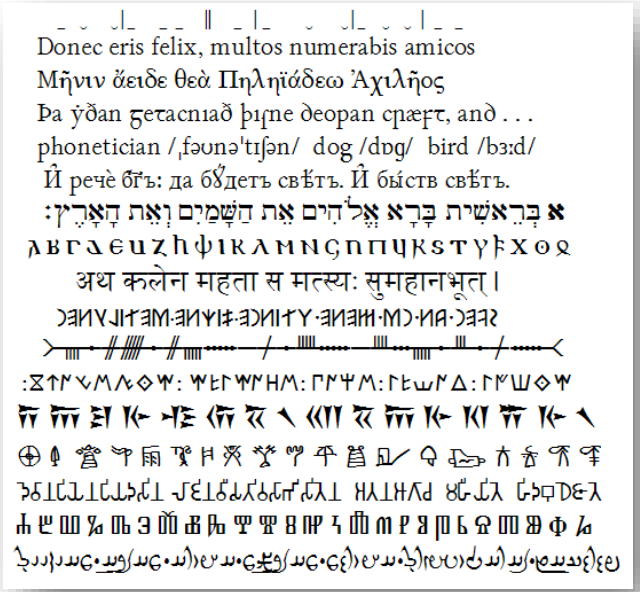
Abnormalities in the Floating-Point Calculations

- Sometimes abnormalities can be observed when using floating-point numbers

  Comparing floating-point numbers can not be performed directly with the **equals operators** (== and ===)

- Example:

```
let a = 0.1;
let b = 0.2;
let sum = 0.3;
let equal = (a+b === sum); // false!!!
console.log('a+b = '+ (a + b) + ', sum = ' +
    sum + ', sum == a+b is ' + equal);
```



Donec eris felix, multos numerabis amicos
Μῆνιν ἄειδε θεὰ Πηληϊάδεω Ἀχιλῆος
Þa ẏðan ᵹeꞇacniað þiꞅne ꝺeopan cꞃæꝼꞇ, anꝺ . . .
phonetician /ˌfəʊnəˈtɪʃən/ dog /dɒɡ/ bird /bɜːd/
И҆ речѐ б҃гъ: да бꙋ́детъ свѣ́тъ. И҆ бы́ств свѣ́тъ.
א בְּרֵאשִׁית בָּרָא אֱלֹהִים אֵת הַשָּׁמַיִם וְאֵת הָאָרֶץ׃
अथ कलेन महता स मत्स्यः सुमहानभूत् ।

String Type

H	e	l	l	o	,		J	S	!

The String Data Type

- Represents a **sequence of characters**
- Strings are enclosed in quotes:
  - **Both ‘ and “** work correctly
  - **ES6** also includes ` **(ticks)** for string **interpolation**
- Strings can be concatenated

  Using the **+ operator**

```
let s = 'Welcome to JavaScript';
let name = 'John' + ' ' + 'Doe';
let greeting = `${s}, ${name}`;

console.log(greeting); // Welcome to JavaScript, John Doe
```

Saying Hello

Example

Concatenating the two names of a person to obtain his full name:

```
let firstName = 'Ivan';
let lastName = 'Ivanov';
console.log('Hello, ' + firstName + '!');

let fullName = firstName + ' ' + lastName;
console.log('Your full name is ' + fullName);
```

Strings are Unicode

Strings are stored as Unicode

- Unicode supports all commonly used alphabets in the world
- E.g., Cyrillic, Chinese, Arabic, Greek, etc. scripts

```
let asSalamuAlaykum = 'السلام عليكم';
alert(asSalamuAlaykum);

let кирилица = 'Това е на кирилица!';
alert(кирилица);

let leafJapanese = '葉';
alert(leafJapanese);
```

Parsing String to Number

- Strings can be parsed to numbers

  Floating-point and rounded (integer)

- The trivial way to parse string to a number is using the functions **parseInt** and **parseFloat**:

```
let numberString = '123'
console.log(parseInt(numberString)); // prints 123
let floatString = '12.3';
console.log(parseFloat(floatString)); // prints 12.3
```

- **parseInt** and **parseFloat** exhibit stranger behavior:

  If a non-number string starts with a number, only the number is extracted:

```
let str = '123Hello';
console.log(parseInt(str)); // prints 123
```

Better String to Number Parsing

- parseInt and parseFloat are readable, but slow and show strange behavior
- Better ways to parse string to numbers are as follows:
  - With rounding:

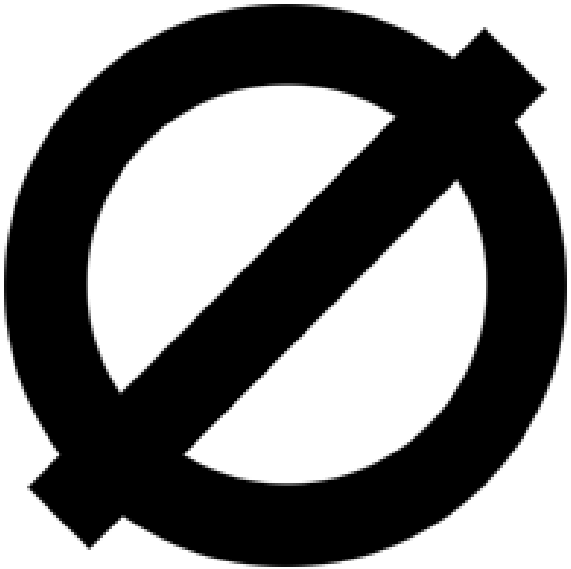
    ‘123.3’ | 0 → returns 123

  - As is:

    Number(‘123.3’) → returns 123.3

    ‘123.3’ * 1 → returns 123.3

    +‘123.3’ → returns 123.3



Undefined and Null Values

Understanding ‘undefined’ in JavaScript

Undefined and Null Values

- JavaScript has a special value **undefined**

  It means the **variable has not been defined** (no such variable in the current context)

- undefined is different than null

  null represents an **empty value**

```
let x;
console.log(x); // undefined

x = 5;
console.log(x); // 5

x = undefined;
console.log(x); // undefined

x = null;
console.log(x); // null
```

Checking a Variable Type

The variable type can be checked at runtime:

```
let x = 5;
console.log(typeof x); // number
console.log(x); // 5

x = new Number(5);
console.log(typeof x); // object
console.log(x); // Number {}

x = null;
console.log(typeof x); // object

x = undefined;
console.log(typeof x); // undefined
```


Summary


Q&A