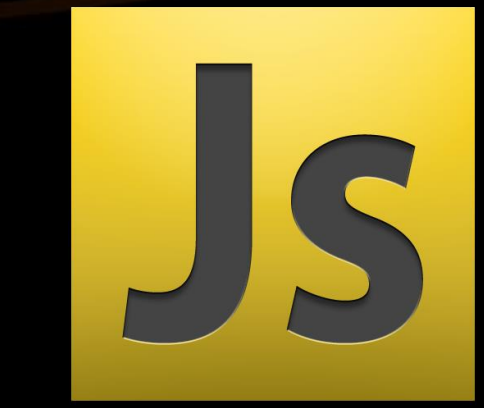

# JavaScript Syntax

## Data Types, Variables, Operators, Expressions

**Au Mau Duong**
**Technical Trainers**

# Table of Contents






# Data Types in JavaScript

# What Is a Data Type?

- A data type:
  - Is a domain of values of similar characteristics
  - Defines the type of information stored in the computer memory (in a variable)
- Examples:
  - Positive integers: **1**, **2**, **3**, …
  - Alphabetical characters: **a**, **b**, **c**, …
  - Dates from the calendar: **1-Nov-2014**, **3-Sep-2006**, …

# JavaScript Data Types

- JavaScript is a typeless language
  - The variable types are not explicitly defined
  - The type of a variable can be changed at runtime
- Variables in JS are declared with the keyword **var**

DON'T DO THAT!

```
var count = 5; // variable holds an integer value
count = 'hello'; // the same variable now holds a string
var name = 'Svetlin Nakov'; // variable holds a string
var mark = 5.25; // mark holds a floating-point number
```

# Integer Numbers

- Integer types represent whole numbers
- In JavaScript integer numbers are in the range from **-9007199254740992** to **9007199254740992**
- The underlying type is a 64-bit floating-point number (IEEE-754)

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

# Floating-Point Numbers

- Floating-point types represent real numbers, e.g. `3.75`
- In JavaScript the floating-point numbers are 64-bit
  - Stored in the IEEE-754 format
  - Have range from `-1.79e+308` to `1.79e+308`
  - Have precision of 15-16 digits
  - The smallest positive number is `5.0e-324`
- Can behave abnormally in the calculations
  - E.g. **0.1 + 0.2 = 0.30000000000000004**

# Floating-Point Numbers – Example

```
var PI = Math.PI, // 3.141592653589793
    minValue = Number.MIN_VALUE, // 5e-324
    maxValue = Number.MAX_VALUE, // 1.79e+308
    div0 = PI / 0, // Infinity
    divMinus0 = -PI / 0, // -Infinity
    unknown = div0 / divMinus0, // NaN
    a = 0.1,
    b = 0.2,
    sum = 0.3,
    equal = (a+b == sum); // false!!!
console.log('a+b = '+ (a+b) + ', sum = ' + sum + ',
  sum == a+b? is ' + equal);
```

# Numbers in JavaScript

- All numbers in JavaScript are stored internally as double-precision floating-point numbers (64-bit)
  - According to the IEEE-754 standard
  - Can be wrapped as objects of type Number

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

# Numbers Conversion

- Convert floating-point to integer number

```
var valueDouble = 8.75;
var valueInt = Math.floor(valueDouble); // 8
```

- Convert to integer number with rounding (up to half values)

```
var valueDouble = 8.75;
var valueInt = Math.round(valueDouble); // 9
```

- Convert to integer number with rounding  (full integer values)

```
var valueDouble = 8.75;
var valueInt = Math.floor(valueDouble); // 8
valueDouble = 8.75;
valueInt = Math.ceil(valueDouble); // 9
```

# Number Parsing/Conversion

- Convert string to integer

```
var str = '1234';
var i = Number(str) + 1; // 1235
```

- Convert string to float

```
var str = '1234.5';
var i = Number(str) + 1; // 1235.5
```

# The Boolean Data Type

- The Boolean data type:
  - Has two possible values: **`true`** and **`false`**
  - Is useful in logical expressions
- Example of Boolean variables:


```
var a = 1;
var b = 2;

var greaterAB = (a > b);
console.log(greaterAB); // false

var equalA1 = (a == 1);
console.log(equalA1); // true
```

# The String Data Type

- The `string` data type represents a sequence of characters
- Strings are enclosed in quotes:
  - Both ' and " work correctly
    - Best practices suggest using single quotes

```
var s = 'Welcome to JavaScript';
```

- Strings can be concatenated (joined together)
  - Using the + operator

```
var name = 'Soft' + ' ' + 'Uni';
```

# Strings are Unicode

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

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

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

var leafJapanese = '葉'; // Pronounced as "ha"
alert(leafJapanese);
```

# Object Type

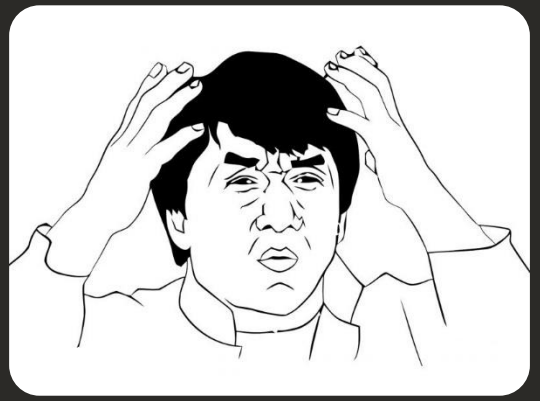
- Objects in JavaScript hold key-value pairs:

```
var obj = { name : "SoftUni", age : 2 };
console.log(obj); // Object {name: "SoftUni", age: 2}
obj['site'] = "http://www.softuni.bg";
obj.age = 10;
obj['name'] = "Software University";
console.log(obj); // Object {name: "Software University",
age: 10, site: "http://www.softuni.bg"}
delete obj.name;
delete obj.site;
console.log(obj); // Object {age: 10}
```

# Type Conversions

- All JS types can be compared due to automatic type conversion

```
5 == "5.0" // true
5 === "5.0" // false (=== checks type + value)
1 == true // true
[] == false // true
'' == false // true
"" == 0 // true
5 + false + true === 6 // true
```



# Problem: Sum Two Numbers

- Write a JS function to sum two numbers given as array of strings

```
let num1 = Number(4)
let num2 = Number(6)
let sum = num1 + num2
// sum = 10
```

Test this function in the judge system

# Problem: Calculate Expression

- Write a JavaScript program to print the value of the following expression: $[(30 + 25) * 1/3 * (35 - 14 - 12)]^2$
- Sample solution:

```
let val = (30 + 25) / 3 * (35 - 14 - 12)
let valSquare = val * val
console.log(valSquare)
```

# Undefined and Null Values

## What is '`undefined`' in JavaScript?

# Undefined and Null Values

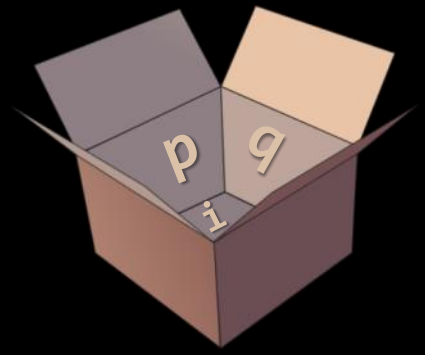
- In JS there is a special value **`undefined`**
  - It means the variable has not been defined (no such variable exist in the current context)
- Undefined is different than **`null`**
  - Null means that an object exists and is empty (has no value)

```
var x = 5;
x = undefined;
alert(x); // undefined

x = null;
alert(x); // null
```

# Checking the Type of a Variable

- The variable type can be checked at runtime:

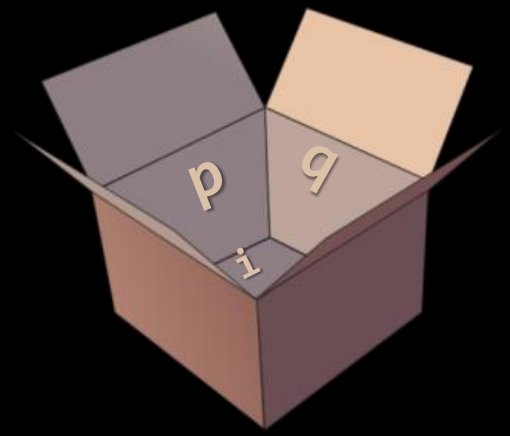
```
var x = 5;
console.log(typeof(x)); // number
console.log(x); // 5

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

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

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




# Declaring and Using Variables

# What Is a Variable?

- A variable is a:
  - Placeholder of information that can be changed at run-time
  - A piece of computer memory holding some value
- Variables allow you to:
  - Store information
  - Retrieve the stored information
  - Change the stored information



# Variable Characteristics

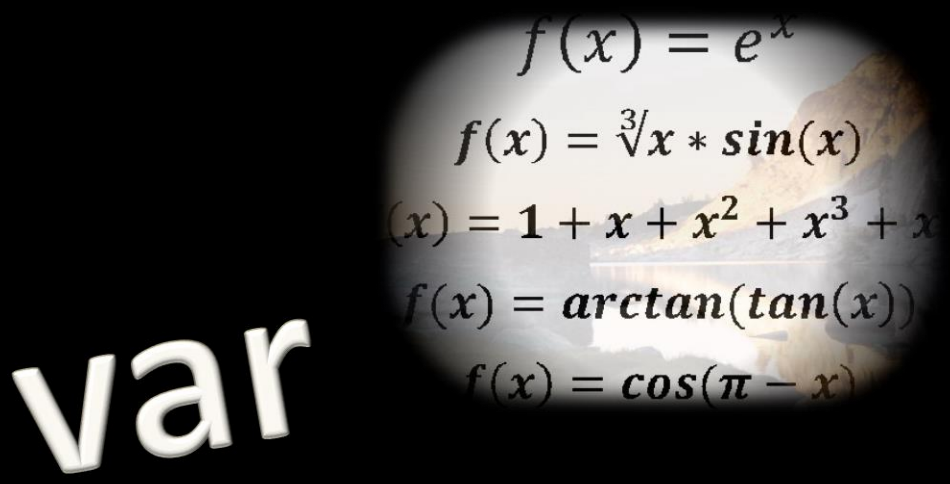
- A variable has:
  - Name
  - Type (of stored data)
  - Value
- Example:

```
var counter = 5;
```

  - Name: **counter**
  - Type: **number**
  - Value: **5**

# Declaring Variables

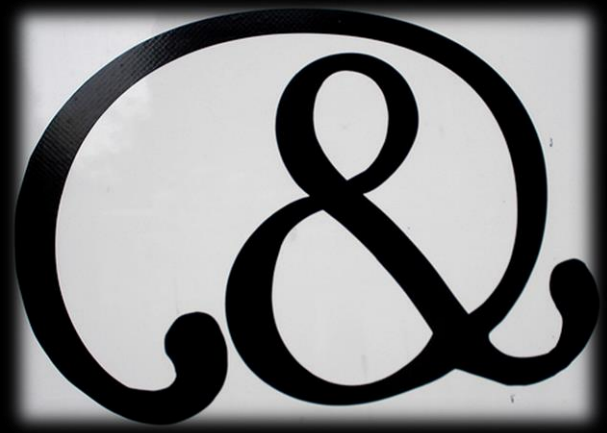
- When declaring a variable we:
  - Specify its name (called identifier)
  - The type is inferred by the value
  - Give it an initial value
- Example:




```
var height = 200;
var str = "Hello";
var obj = { name : 'Peter', age : 19 };
```

# Identifiers

- Identifiers may consist of:
  - Letters (Unicode), digits [**0**-**9**], underscore '**_**', dollar '**$**'
  - Cannot start with a digit
  - Cannot be a JavaScript keyword
- Identifiers in JavaScript are case-sensitive
- Identifiers should have a descriptive name
  - Only Latin letters
- Variable names: use **`camelCase`**
- Function names :use **`camelCase`**



# Identifiers – Examples

- Examples of correct identifiers:

```
var New = 2; // Here N is capital, so it's not a JS keyword
var _2Pac = 2; // This identifier begins with _

var поздрав = 'Hello'; // Unicode symbols used
// The following is more appropriate:
var greeting = 'Hello';

var n = 100; // Undescriptive
var numberOfClients = 100; // Descriptive

// Overdescriptive identifier:
var numberOfPrivateClientOfTheFirm = 100;
```

- Examples of incorrect identifiers:

```
var new = 5; // new is a keyword
var 2Pac = 2; // Cannot begin with a digit
```

# Assigning Values

- The = operator is used to assign a value to a variable:

```
// Assign a value to a variable
var firstValue = 5;

// Using an already declared variable:
var 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:
var thirdValue = firstValue = 3; // Avoid this!
```

# Local and Global Variables

- Local variables
  - Declared **with** the keyword **`var`**

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

- Global variables
  - Declared **without** the keyword **`var`**
  - Stored as properties of the **`window`** object

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

  - Using global variables is very bad practice!

# Variables in JavaScript

- A variable in JavaScript can be:
  - unresolvable

```
console.log(asfd); // ReferenceError
```

  - undefined

```
var p = undefined; console.log(p); // undefined
```

  - null

```
var p = null; console.log(p); // null
```

  - local

```
var localVar = 5; console.log(localVar); // 5
```

  - global

```
globalVar = 5; console.log(globalVar); // 5
```

# Unresolvable Variables and Undefined

- Unresolvable variables in JavaScript are different than undefined

```
console.log(msg); // ReferenceError: msg is not defined
```

```
var greeting = 'hello'; // A local variable
console.log(greeting); // hello
msg = greeting; // msg is a global variable with value 'hello'
console.log(msg); // hello
msg = undefined; // Different than "delete msg"
console.log(msg); // undefined
console.log(greeting); // hello
delete msg; // Delete a global variable
console.log(msg); // ReferenceError: msg is not defined
```

# Unresolvable Variables – Examples

- In this code **secondVar** is unresolvable:

```
var firstVar = 10;
console.log(firstVar); // 10
console.log(secondVar); // ReferenceError: secondVar is not defined
```

- In this code **p** is undefined (instead of unresolvable):

```
console.log(p); // undefined
var p = undefined;
console.log(p); // undefined
// p is now undefined, it is resolvable
```

# JavaScript Strict Syntax

- It is recommended to enable the "**strict syntax**"
  - Converts global variables usage to runtime errors
  - Disables some of the "bad" JavaScript features

```
"use strict";

var local = 5; // Local variables will work in strict mode
global = 10; // Uncaught ReferenceError: global is not defined

// This code will not be executed, because of the error above
console.log(5 * 5);
```

# Operators in JavaScript

Arithmetic, Logical, Comparison, Assignment, …

# Arithmetic Operators

- Arithmetic operators `+`, `-`, `*`, `/` are the same as in math
- The division operator `/` returns number or `Infinity` or `NaN`
- Remainder operator `%` returns the remainder from division of numbers
  - Even on real (floating-point) numbers
  - E.g. 5.3 % 3 → 2.3
- The operator `++` / `--` increments / decrement a variable
  - Prefix `++` vs. postfix `++`

# Logical Operators

- Logical || operator returns the first "true" value

```
var foo = false || 0 || '' || 4 || 'foo' || true;
console.log(foo); // Logs 4, because its the first true value in the expression
```

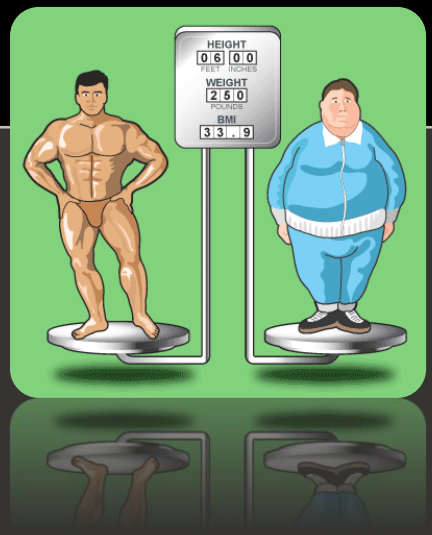
- Logical && operator returns the first "false" value

```
var foo = true && 'foo' && '' && 4 && 'foo' && true;
console.log(foo); // Logs '' an empty string, because its the first false value
```

# Comparison Operators

- Comparison operators are used to compare variables
  - `==`, `<`, `>`, `>=`, `<=`, `!=`, `===`, `!==`
- The `==` means "equal after type conversion"
- The `===` means "equal and of the same type"

```
var a = 5;
var b = 4;
console.log(a >= b); // True
console.log(a != b); // True
console.log(a == b); // False

console.log(0 == ""); // True
console.log(0 === ""); // False
```



# Assignment Operators

- Assignment operators are used to assign a value to a variable
  - =, +=, -=, |=, …
- Assignment operators example:

```
var y = 4;
console.log(y *= 2); // 8
var z = y = 3; // y=3 and z=3
console.log(z += 2); // 5
```

- Variables, with no value assigned, are undefined

```
var foo;
console.log(foo); // Logs undefined
```

# Other Operators (1)

- String concatenation operator **+** is used to concatenate strings
- If the second operand is not a string, it is converted to string automatically
- Member access operator **.** is used to access object members
- Square brackets **[ ]** are used with arrays to access element by index
- Parentheses **( )** are used to override the default operator precedence

```
var output = "The number is : ";
var number = 5;
console.log(output + number);
// The number is : 5
```

# Other Operators (2)

- Conditional operator `?:` has the form

```
b ? x : y
```

(if `b` is `true` then the result is `x` else the result is `y`)

- The `new` operator is used to create new objects
- The `typeof` operator returns the type of the object
- `this` operator references the current context
  - In JavaScript the value of `this` depends on how the function is invoked

# Other Operators (3)

```
var obj = {};
obj.name = "SoftUni";
obj.age = 2;
console.log(obj); // Object {name: "SoftUni", age: 2}

var a = 6;
var b = 4;
console.log(a > b ? "a > b" : "b >= a"); // a>b
var c = b = 3; // b=3; followed by c=3;
console.log(c); // 3
console.log((a+b)/2); // 4.5
console.log(typeof(a)); // number
console.log(typeof([])); // object
```

# Expressions

# Expressions

- Expressions are
  - Sequences of operators, literals and variables that are evaluated to some value
- Examples:

```
var r = (150-20) / 2 + 5; // r=70

// Expression for calculation of circle area
var surface = Math.PI * r * r;

// Expression for calculation of circle perimeter
var perimeter = 2 * Math.PI * r;
```

# Summary

- JavaScript dynamic data types
  - Number, String, Boolean, Undefined, Null
  - Local and Global variables
- Operators (same as in C#, Java and C++)
- Expressions (same as in C#, Java and C++)


# JavaScript Syntax

Questions?