JavaScript For Web

Week 1, Lecture 3 - JavaScript Fundamentals: Variables and Operators

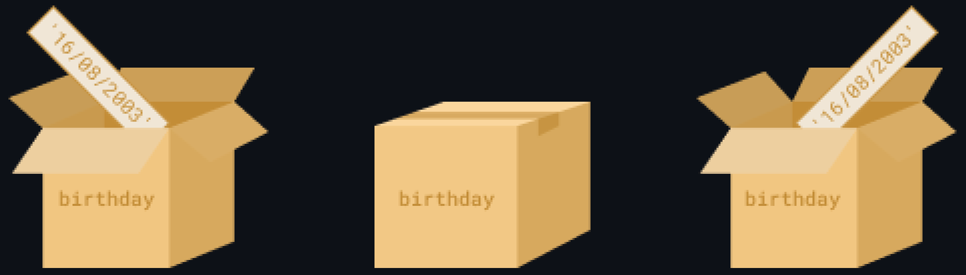
Instructor: Jason Xu

Today's Overview

- JavaScript Fundamentals: variables and operators.
- Tutorial: Assignment 1, Exercise 2.

JavaScript Variables Review

- A variable is a "named storage" for data. You can think of variables as simply "storage containers" for data in your code:



- To create a variable in JavaScript, use the `let` keyword:

```
let birthday;
```

- we can put data into it by using the assignment operator `=`:

```
let birthday;
birthday = '16/08/2003'; // store the string '16/08/2003' in the variable named birthday
```

JavaScript Variables Review: Change the value

- We can change the variable value as many time as we want:

```
let message
console.log(message)

message = 'Hello!'
console.log(message) // value changed

message = 'World!' // value changed
console.log(message)
```

- Here is the console output:

```
undefined
Hello!
World!
```

JavaScript Variable Review: `let` and `const`

- To declare a variable using `let`, we can change the variable value **as many time as we want**:

```
let message = 'Hello!'
message = 'World!' // value changed
```

- To declare a constant(unchanging) variable, use `const`:

```
const birthday = '16/08/2003'
```

- Variables declared using `const` are called “constants”. They cannot be reassigned. An attempt to do so would cause an error:

```
const birthday = '16/08/2003'

birthday = '25/09/2004' // error, can't reassign the constant!
```

- Here is the output error:

```
Uncaught TypeError: Assignment to constant variable.
```

- Use `let` or `const` ONLY when declare a variable, never use `var`.

Rules for naming variables.

There are two limitations on variable names in JavaScript:

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

Example of valid names:

```
let userName
let test123

let $ = 1 // declared a variable with the name "$"
let _ = 2 // and now a variable with the name "_"
```

Example of incorrect variable names:

```
let 1a // cannot start with a digit
let my-name // hyphens '-' aren't allowed in the name
```

When the name contains multiple words, camelCase is commonly used. That is: 1) start with lower case letter, 2) words go one after another, each word except first starting with a capital letter. For example: `myVeryLongName`.

Rules for naming variables continued.

Case matters(sensitive).

- Variables named `apple` and `APPLE` are two different variables.

Non-latin letters are allowed, but not recommanded:

```
let 我 = '...'
```

- My suggestion: avoid non-latin letters all the time.

Reserved names

- 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.
- The code below gives a syntax error:

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

JavaScript Variables: Uppercase constants

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.

For instance, let’s make constants for colors in so-called “web” (hexadecimal) format:

```
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 //#FF7F00
```

Benefits:

- `COLOR_ORANGE` is much easier to remember than .`"#FF7F00"`
- It is much easier to mistype `"#FF7F00"` than `COLOR_ORANGE`.
- When reading the code, `COLOR_ORANGE` is much more meaningful than `#FF7F00`.

JavaScript Variables: Uppercase constant continued

When should we use capitals for a constant and when should we name it normally?

- There are constants that are known prior to execution (like a hexadecimal value for red)

```
const COLOR_RED = "#F00"
```

- There are constants that are calculated in run-time, during the execution, but do not change after their initial assignment.

```
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 this case, we still use camelCase.

Therefore, uppercase constants are only used for **"hard-coded" values** that are known prior to execution.

JavaScript Variables: Name things right

A variable name should have a clean, obvious meaning, describing the data that it stores.

Some good-to-follow rules are:

- Use human-readable names like `userName` or `shoppingCart`.
- Stay away from abbreviations or short names like `a`, `b`, `c`, unless you really know what you're doing.
- Make names maximally descriptive and concise. Examples of bad names are `data` and `value`. Such names say nothing. It's only okay to use them if the context of the code makes it exceptionally obvious which data or value the variable is referencing.
- Agree on terms within your team and in your own mind. If a site visitor is called a "user" then we should name related variables `currentUser` or `newUser` instead of `currentVisitor` or `newManInTown`.

JavaScript Variables: `use strict`

In the old JavaScript, it was technically possible to create a variable by a mere assignment of the value without using `let`. This still works now if we don't put use strict in our scripts to maintain compatibility with old scripts.

```
// note: no "use strict" in this example

num = 5 // the variable "num" is created if it didn't exist
```

- This is a bad practice and would cause an error in strict mode:

```
"use strict"

num = 5 // error: num is not defined
```

- Therefore, no matter `"use strict"` is stated, we should always use `let` or `const` to declare a variable.

JavaScript Variable: Value and Data types

- A value in JavaScript is always of a certain type. For example, a string or a number(we talked about in previous slides).
- There are eight basic data types in JavaScript. We will talked about all of these in detail soon. For now, we should be familiar with these three basic data type: **Number**, **String** and **Boolean**.

Number: The number type represents both integer and floating point numbers.

```
let n = 123
n = 12.345
```

String: A string in JavaScript must be surrounded by quotes.

```
let str = "Hello"
let str2 = 'Single quotes are ok too'
```

- Both single and double quotes works for declaring a string

Boolean: The boolean type has only two values: `true` and `false` (They are case-sesitive).

```
let nameFieldChecked = true // yes, name field is checked
let ageFieldChecked = false // no, age field is not checked
```

JavaScript Operators

JavaScript Operators: What are operators

An operator is a symbol that produces a result based on one or two values (or variables).

An 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”.

Types of operators:

- Assignment operators

```
let x = 1
```

- Arithmetic, Math
- String concatencation
- Type conversion
- Comparison
- etc.

JavaScript Operators: Maths

- Addition `+`,
- Subtraction `-`,
- Multiplication `*`,
- Division `/`,
- Remainder(Modulus/Mod) `%`,
- Exponentiation `**`.

The first four are straightforward, while `%` and `**` need a few words about them.

JavaScript Math Operators: Remainder/Modulus/Mod %

The remainder operator %, despite its appearance, is not related to percents.

The result of a % b is the remainder of the integer division of a by b.

For instance:

```
console.log( 5 % 2 ) // 1, the remainder of 5 divided by 2
console.log( 8 % 3 ) // 2, the remainder of 8 divided by 3
console.log( 8 % 4 ) // 0, the remainder of 8 divided by 4
```

JavaScript Math Operators: Exponentiation `**`

The exponentiation operator `a ** b` raises a to the power of b.
In school maths, we write that as a .

For instance:

```
console.log( 2 ** 2 ) // 2² = 4
console.log( 2 ** 3 ) // 2³ = 8
console.log( 2 ** 4 ) // 2⁴ = 16
```

Just like in maths, the exponentiation operator is defined for non-integer numbers as well.

For example, a square root ($\sqrt{n}$) is an exponentiation by ½:

```
console.log( 4 ** (1/2) ) // 2 (power of 1/2 is the same as a square root)
console.log( 8 ** (1/3) ) // 2 (power of 1/3 is the same as a cubic root)
```

String concatenation with `+`

Let’s meet the features of JavaScript operators that are beyond school arithmetics.

- Usually, the plus operator `+` sums numbers. But, if `+` is applied to strings, it merges (concatenates) them:

```
let s = "my" + "string"
console.log(s) // "mystring"
```

- if any of the operands is a string, then the other one is converted to a string too:

```
console.log( '1' + 2 ) // "12"
console.log( 2 + '1' ) // "21"
```

Summary

- What are operators, operands and operations?
- How to use arithmetic operators in JavaScript?
- What is string concatenation?
- Next: more operators and data types.