JavaScript For Web

Week 2, Lecture 1 - JavaScript Fundamentals

Instructor: Jason Xu

Today's Overview

Week 1 Review

- Variables
- Operators
- Data Types
- Conditionals

JavaScript Conditional continued

- `switch` statement

Tutorial: No assignment today

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` or `const` keyword, we can put a data value into it by using the assignment operator `=`:

```
let birthday = "16/08/2003" // store the string "16/08/2003" in the variable named birthday
const COLOR_RED = "#F00"    // store the string "#F00" in the constant variable named COLOR_RED
```

- When declaring a variable using `let`, we can change the variable value as many time as we want:

```
birthday = "16/08/2004"
birthday = "Aug-16-2004"
```

- When declaring using `const`, we **cannot** change the variable value. An attempt to do would cause an error:

```
COLOR_RED = "#F01" // error, can't reassign the constant!
```

- Never use `var`.

JavaScript Data Types Review

There are eight basic data types in JavaScript

- Primitive data types: `number`, `bigint`, `boolean`, `string`, `null`, `undefined`, `symbol`
- Non-primitive data type: `object`
- Prmitive data types are **immutable**.
- The `typeof` operator allows us to see which type is stored in a variable.

```
typeof 0 // "number"

typeof true // "boolean"

typeof "foo" // "string"

typeof null // "object"

typeof undefined // "undefined"
```

JavaScript Data Types Review

Five commonly used primitive data types

- `number`: Integer or floating-point limited by ±(2^53 - 1).

```
//special number value
const MAX = Number.MAX_SAFE_INTEGER // The maximum safe integer (2^53 - 1)
const MIN = Number.MIN_SAFE_INTEGER // The minimum safe integer -(2^53 - 1)
const INF = Infinity // The special value that's greater than any number.
console.log(typeof(NaN)) // NaN is the result of an incorrect or an undefined mathematical operation
```

- `boolean`: `true` and `false`
- `string`: A string in JavaScript must be surrounded by quotes. There are three different types of quotes:

```
let username = "Jason" //double quotes
let userId = '56498725' //single quotes
let phrase = `Hello, my name is ${username} and my ID is ${userId}.` //backticks
```

- `null` and `undefined`: Both represents "nothing", "unknown", "empty". But there are **two different data types**. We use `null` to assign an "empty" or "unknown" value to a variable, while `undefined` is reserved as a default initial value for unassigned things.

JavaScript Operators Review

- An **operator** is a symbol that produces a result based on one(unary) or two(binary) values (or variables).
- An **operand** – is what operators are applied to. For instance: in `5 * 2` there are two operands: `5` and `2`.
- Arithmetic(Math) operators: `+`, `-`, `*`, `/`, `%`, `**`
- Assignment operators: `=`

We often need to apply an operator to a variable and store the new result in that same variable. For example:

```
let num = 2
num = num + 5 // now n = 7
num = num * 2 // now n = 14
```

This notation can be shortened using the operators `+=`, `*=`, `-=`, `/=`, etc:

```
let num = 2
num += 5 // now n = 7 (same as num = num + 5 )
num *= 2 // now n = 14 (same as num = num * 2)
```

JavaScript Operators Review continued

- Increment/Decrement operators, increases/decreases a variable value by 1: `++`, `--`

```
let counter = 1
counter++ // now counter is 2, works the same as counter = counter + 1
counter-- // now counter is 1, works the same as counter = counter - 1
console.log(counter) // 1
```

- The operators `++` and `--` can be placed either before or after a variable.

```
++counter // now counter is 2, works the same as counter = counter + 1
--counter // now counter is 1, works the same as counter = counter - 1
```

- The `++counter` increments `counter` and returns the new value, `2`. So, the output is `2`.

```
let counter = 1, result = ++counter

console.log(counter) // 2
console.log(result)  // 2
```

- The `counter++` also increments `counter` but returns the old value (prior to increment). So, the output `1`.

```
let counter = 1, result = counter++

console.log(counter) // 2
console.log(result)  // 1
```

JavaScript Operators Review continued

- String concatenation operator: usually, the plus operator `+` sums numbers. But, if `+` is applied to strings, it concatenates them:

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

- String concatenation operator: 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( true + "1" ) // "true1"
```

- Numeric conversion operator: The need to convert strings to numbers arises very often. We can use unary `+` to do this:

```
let apples = "2"
let oranges = "3"

console.log( apples + oranges ) // "23", the binary plus concatenates strings
console.log( +apples + +oranges ) // 5, both values converted to numbers before the binary plus
```

JavaScript Operators Review continued

- Other arithmetic operators(other than `+`) work only with number and always convert their operands to numbers.

```
console.log( 6 - "2" ) // 4, converts "2" to a number
console.log( "6" / "2" ) // 3, converts both operands to numbers
```

- Type conversion:

```
let num = Number("0") // convert a string "0" to a number 0 and assign to the variable named num
let str = num.toString() //convert a number 0 to a string "0" and assign to the variable named str
```

- Operator precedence: If an expression has more than one operator, the execution order is defined by their **precedence**, or, in other words, the **default priority order** of operators. Check the operator precedence table on MDN.

JavaScript Operators Review continued

- Comparison operators produce a result of boolean value(`true` or `false`): `>`, `<`, `==`, etc.
- Regular comparison and equality check: `==`, `>=`, `<=`, `!=`
- Strict comparison and equality check: `===`, `>==`, `<==`, `!==`
- A regular check cannot differentiate data types because operands of different types are converted to numbers:

```
console.log( 0 == false )  // true
```

- A **strict check** `===` checks the equality without type conversion.

```
console.log( 0 === false )  // false
```

- Use **strict comparison and equality** check all the time.
- Greater / Less: `>` / `<`
- Greater / Less and equal: `>==` / `<==`
- Equal / Not equal: `===` / `!==`

JavaScript Operators Review continued

- Logical operators: `||`, `&&`, `!`
- The OR operator is represented with two vertical line symbols `||`, there are four possible logical combinations in a binary comparison(at least one is `true` gives a result of true):

```
console.log( true  || true )   // true
console.log( false || true )   // true
console.log( true  || false )  // true
console.log( false || false )  // false
```

- The AND operator is represented with two ampersands `&&`, there are four possible logical combinations in a binary comparison(all should be true to gives a result of true):

```
console.log( true  && true )   // true
console.log( false && true )   // false
console.log( true  && false )  // false
console.log( false && false )  // false
```

- The NOT operator is represented with an exclamation sign `!`:

```
console.log( !true ) // false
```

- The precedence of `!` NOT is the **highest** of all logical operators, so it always executes first, then `&&` AND and `||` OR.

JavaScirpt Conditional Review

- Truthy/Falsy value: a **truthy** value is a value that is considered true when encountered in a boolean context/operation. All values are truthy unless they are defined as **falsy**.
- All values are truthy **except**:
  - `false`
  - `0`, `-0`, `0n` (The number/bigint zero)
  - `""`, ``` `` ```, `''` (empty string)
  - `null`, `undefined`, `NaN` (unknown, empty value)
  - `document.all` (The only falsy build-in object in JavaScript)

```
//The condition is true
if (1) {
  console.log("Hello!")
}

//The condition is false
if (0) {
  console.log("Hello!")
}
```

JavaScript Conditional Review continued

- The `if` statement: `if`, `else if`, `else` keywords and syntax

```
let hour = new Date().getHours()

if (hour < 10) {
  console.log("Good morning")
} else if (hour < 20) {
  console.log("Good day")
} else {
  console.log("Good evening")
}
```

- More complex conditions with logical operators

```
let hour = new Date().getHours(), isWeekend = true

if (hour > 10 && hour < 18 && !isWeekend ) {
  console.log( 'The office is opened.' )
} else {
  console.log( 'The office is closed.')
}
```

- Conditional operator: `?`

```
let greetings = (hour < 18) ? "Good day" : "Good evening"
```

JavaScript Conditional: `switch` statement

- In JavaScript, you can also consider use `switch` statment if you've got a large number choices.
- The syntax is:

```
switch (expression) {
  case choice1:
    // run this code
    break

  case choice2:
    // run this code instead
    break

  case choice3:
    // run this code instead
    break

  // include as many cases as you like

  default:
    // actually, just run this code
    break
}
```

JavaScript Conditional: `switch` statement continued

- For example, we can rewrite our assignment 1, exercise 3 to this:

```
...

const choice = ...

switch (choice) {
    case "sunny":
      para.textContent =
        "It is nice and sunny outside today. Wear shorts! Go to the beach, or the park, and get an ice cream."
      break
    case "rainy":
      para.textContent =
        "Rain is falling outside; take a rain coat and an umbrella, and don't stay out for too long."
      break
    case "snowing":
      para.textContent =
        "The snow is coming down — it is freezing! Best to stay in with a cup of hot chocolate, or go build a snowman."
      break
    case "overcast":
      para.textContent =
        "It isn't raining, but the sky is grey and gloomy; it could turn any minute, so take a rain coat just in case."
      break
    default:
      para.textContent = ""
      break
  }

...
```

My comment: check the syntax **carefully**, don't forget the `break` statement after each `case`.

Thank you.