Introduction to JavaScript: Part 2

Introduction to Internet and Web

부산대학교 정보·의생명공학대학
정보컴퓨터공학부

Contents

JAVASCRIPT DATA TYPES

JavaScript Data Type

❖ Primitive Data Type

- Number, String, Boolean, Null, Undefined, …

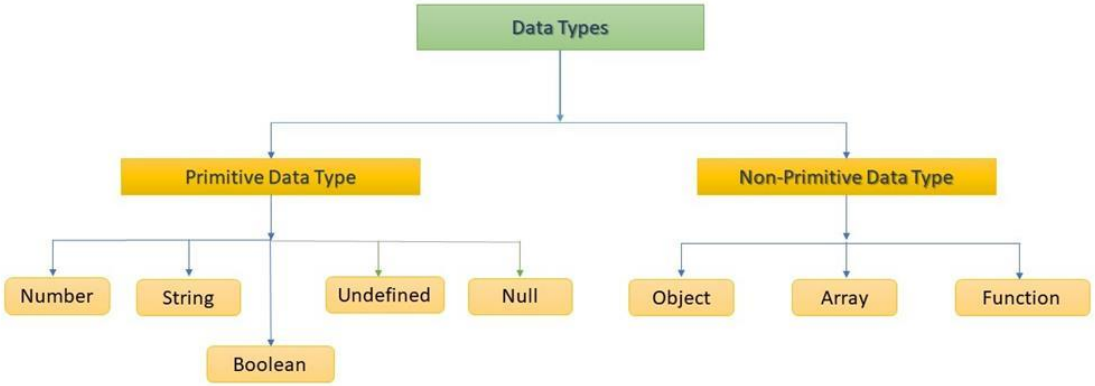
❖ Non-Primitive Data Type

- Object, Array, Function, Date, …

```
typeof "John"                  // Returns "string"
typeof 3.14                    // Returns "number"
typeof NaN                     // Returns "number"
typeof false                   // Returns "boolean"
typeof [1,2,3,4]               // Returns "object"
typeof {name:'John', age:34}   // Returns "object"
typeof new Date()              // Returns "object"
typeof function () {}          // Returns "function"
typeof myCar                   // Returns "undefined" *
typeof null                    // Returns "object"
```

❖ JavaScript has dynamic types

- The same variable can be used to hold different data types



JAVASCRIPT OBJECT

JavaScript Object

- **You have already learned that JavaScript variables are containers for data values.**
- **This code assigns a single value ("Fiat") to a variable named car:**
  - *var car = "Fiat";*
- **But objects can obtain many values.**
  - *var car = {type: "Fiat", model: "500", color:"white"};*
  - The values are written as name:value pairs
  - The name:values pairs in JavaScript objects are called properties

Example - JavaScript Object Properties

❖ **Member operator**

- *Objejectname.propertyname*
- *Objectname[propertyname]*

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
// Create an object:
const person = {
  firstName: "John",
  lastName : "Doe",
  id     :  5566
};
// Display some data from the
object:
document.getElementById("demo").in
nerHTML =
person.firstname + " " +
person["lastName"];

</script>
</body>
</html>
```

JavaScript Object Methods

❖ **Objects can also have methods.**

- Methods are actions that can be performed on objects.
- Methods are stored in properties as function definitions.

Object	Properties	Methods
	car.name = Fiat	car.start()
	car.model = 500	car.drive()
	car.weight = 850kg	car.brake()
	car.color = white	car.stop()

Example - JavaScript Object Methods

- **In a function definition, this refers to the "owner" of the function**
  - In the example, this is the person object that "owns" the fullname function.

- **document object**
  - https://developer.mozilla.org/en-US/docs/Web/API/Document
  - getElementByID() method

- **Element object**
  - https://developer.mozilla.org/en-US/docs/Web/API/Element
  - innerHTML property

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

<script>
// Create an object:
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName: function() {
    return this.firstName + " " +
this.lastName;
  }
};
// Display data from the object:
document.getElementById("demo").in
nerHTML = person.fullName();
</script>
</body>
</html>
```

JavaScript Objects are Mutable

❖ **Objects are mutable: They are addressed by reference, not by value.**

❖ **Example**

- The object x is not a copy of person. It is person. Both x and person are the same object.
- Any changes to x will also change person, because x and person are the same object.

```
const person = {
  firstName:"John",
  lastName:"Doe",
  age:50, eyeColor:"blue"
}

const x = person;
x.age = 10;      // Will change both x.age and person.age
```

JavaScript Property Creation/Deletion

❖ Deleting Properties

- deletes both the value of the property and the property itself
- should not be used on predefined JavaScript object properties. It can crash your application

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};

delete person.age;
```

❖ Create a single object, using a object literal

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

```
const person = {};
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

❖ Define a object constructor, and then create of the constructed type

JavaScript Object Creation

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
// Constructor function for Person objects
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}

// Create two Person objects
const myFather = new Person("John", "Doe", 50, "blue");
const myMother = new Person("Sally", "Rally", 48, "green");

// Display age
document.getElementById("demo").innerHTML =
"My father is " + myFather.age + ". My mother is " + myMother.age + ".";
</script>

</body>
</html>
```

JAVASCRIPT ARRAY

JavaScript Arrays

- **JavaScript arrays are used to store multiple values in a single variable.**
  - var *array_name* = [*item1*, *item2*, ...];
- **An array can hold many values under a single name, and you can access the values by referring to an index number.**
  - *var name = cars[0];*
  - *cars[0] = "Opel";*

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

JavaScript Arrays

Saab,Volvo,BMW

JavaScript Array Methods

- A number of methods are provided for Number
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array
- The *pop*() method removes the last element from an array, and returns the removed element.
- The *push*() method adds a new element to an array (at the end), and returns the new length.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.pop();       // the value of x is "Mango"
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.push("Kiwi");   //  the value of x is 5
```

JavaScript Array Methods

- The *shift*() method removes the first item of an array, and returns the removed item.
- The *unshift*() adds new items to the beginning of an array, and returns the new length.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.shift();    // the value of x is "Banana"
```

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Adds a new element "Lemon" to fruits
```

JavaScript Array Methods

❖ Changing Elements

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi";          // Changes the first element of fruits to
```

❖ Deleting Elements

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<p>Deleting elements leaves undefined holes in an array.
</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML =
"The first fruit is: " + fruits[0];
delete fruits[0];
document.getElementById("demo2").innerHTML =
"The first fruit is: " + fruits[0];
</script>

</body>
</html>
```

JavaScript Array Methods

Deleting elements leaves undefined holes in an array.

The first fruit is: Banana

The first fruit is: undefined

JavaScript Array Methods

❖ *splice()* method

- The first parameter defines the position where new elements should be added (spliced in).
- The second parameter defines how many elements should be removed.
- The rest of the parameters define the new elements to be added.

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>splice()</h2>

<p>The splice() method adds new elements to an array.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = "Original
Array:<br>" + fruits;
function myFunction() {
  fruits.splice(2, 0, "Lemon", "Kiwi");
  document.getElementById("demo2").innerHTML = "New Array:
<br>" + fruits;
}
</script>

</body>
</html>
```

JavaScript Array Methods

splice()

The splice() method adds new elements to an array.

Try it

Original Array:
Banana,Orange,Apple,Mango

New Array:
Banana,Orange,Lemon,Kiwi,Apple,Mango

JavaScript Array Methods

❖ Using *splice*() to remove array elements

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>splice()</h2>

<p>The splice() methods can be used to remove array
elements.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;
function myFunction() {
  fruits.splice(0, 1);
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

JavaScript Array Methods

splice()

The splice() methods can be used to remove array elements.

Try it

Orange,Apple,Mango

JAVASCRIPT PRIMITIVE DATA TYPE

String, Number, Boolean

JavaScript String

❖ A JavaScript string is zero or more characters written inside quotes.

- You can use single or double quotes:
- You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
let answer1 = "It's alright";
let answer2 = "He is called 'Johnny'";
let answer3 = 'He is called "Johnny"';
```

❖ Escape Character

- The backslash (\) escape character turns special characters into string characters

```
let text = "We are the so-called "Vikings" from the north.";
let text = "We are the so-called \"Vikings\" from the north.";
```

JavaScript String

❖ **Primitive values, like "John Doe", cannot have properties or methods (because they are not objects).**

❖ **But with JavaScript, methods and properties are also available to primitive values**

- Because JavaScript treats primitive values as objects when executing methods and properties.
- String Ojbect Documentation (link)

```
let txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
let length = txt.length;
```

❖ **JavaScript String as Object**

- The new keyword complicates the code and slows down execution speed.
- (x == y)
- (x === y)

```
let x = "John";
let y = new String("John");
```

```
let x = new String("John");
let y = new String("John");
```

JavaScript String Methods

❖ Search Methods

- indexOf(), lastIndexOf(), startsWith(), endsWith()
- search(), match(), includes()

```
let str = "Please locate where 'locate' occurs!";
str.lastIndexOf("locate");                  //21
str.lastIndexOf("John");                    //-1
str.indexOf("locate", 15);                  // 21
str.lastIndexOf("locate", 15);              // 7
str.search("locate");                       // 7
```

❖ Extract String Parts

- slice(), substring(), substr()

```
let str = "Apple, Banana, Kiwi";
let part1 = str.slice(7, 13);
let part2 = str.substr(7, 6);
let part3 = str.substring(7, 13);
```

JavaScript Number

- Unlike many other programming languages, JavaScript does not define different types of numbers
  - integers, short, long, floating-point etc.
- JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard.
- NaN is a JavaScript reserved word indicating that a number is not a legal number
- Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number.
- JavaScript Numbers as Objects

```
let x = 123;
let y = new Number(123);
```

```
let x = 123e5;     // 12300000
let y = 123e-5;    // 0.00123
let z = 0xFF;
```

JavaScript Number Methods

❖ **A number of methods are provided for Number**

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number

❖ ***toString()***

- Returns a number as a string

```
var x = 123;
x.toString();            // returns 123 from variable x
(123).toString();        // returns 123 from literal 123
(100 + 23).toString();   // returns 123 from expression 100 + 23
```

❖ ***parseInt()***

- Parses its argument and returns an integer
- Spaces are allowed. Only the first number is returned:

```
parseInt("10");          // returns 10
parseInt("10.33");       // returns 10
parseInt("10 20 30");    // returns 10
parseInt("10 years");    // returns 10
parseInt("years 10");    // returns NaN
```

JavaScript Boolean

❖ A JavaScript Boolean represents one of two values: true or false.

❖ You can use the Boolean() function to find out if an expression (or a variable) is true:

```
Boolean(10 > 9)
```

❖ JavaScript Booleans as Objects

```
let x = false;
let y = new Boolean(false);

// typeof x returns boolean
// typeof y returns object
```

JAVASCRIPT CONDITIONS/SWITCH

JavaScript Conditions

❖ **Conditional statements are used to perform different actions based on different conditions.**

- Use **if** to specify a block of code to be executed, if a specified condition is true
- Use **else** to specify a block of code to be executed, if the same condition is false
- Use **else if** to specify a new condition to test, if the first condition is false
- Use **switch** to specify many alternative blocks of code to be executed

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2
is true
} else {
  //  block of code to be executed if the condition1 is false and condition2
is false
}
```

JavaScript Conditions

```html
<!DOCTYPE html>
<html>
<body>

<p>Click the button to get a time-based greeting:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var greeting;
  var time = new Date().getHours();
  if (time < 10) {
    greeting = "Good morning";
  } else if (time < 20) {
    greeting = "Good day";
  } else {
    greeting = "Good evening";
  }
  document.getElementById("demo").innerHTML = greeting;
}
</script>

</body>
</html>
```

Click the button to get a time-based greeting:

Try it

Good day

JavaScript Switch

- Use the switch statement to select one of many code blocks to be executed
- When JavaScript reaches a break keyword, it breaks out of the switch block
- The default keyword specifies the code to run if there is no case match
- Switching details
  - If multiple case matches a case value, the first case is selected.
  - If no matching cases are found, the program continues to the default label.
  - If no default label is found, the program continues to the statement(s) after the swtich

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

JavaScript Switch

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript switch</h2>

<p id="demo"></p>

<script>
var text;
switch (new Date().getDay()) {
  case 4:
  case 5:
    text = "Soon it is Weekend";
    break;
  case 0:
  case 6:
    text = "It is Weekend";
    break;
  default:
    text = "Looking forward to the Weekend";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JavaScript switch

Soon it is Weekend

JAVASCRIPT FOR/WHILE

JavaScript Loops

❖ JavaScript Loops

- Loops are handy, if you want to run the same code over and over again, each time with a different value.

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

```
var i;
for (i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
```

❖ Different Kinds of Loops

- For – loops through a block of code a number of times
- For/in – loops through the properties of an object
- For/of – loops through the values of an iterable object
- While – loops through a block of code while a specified condition is true
- Do/while – also loops through a block of code while a specified condition is true

JavaScript For Loop

❖ Syntax

- Statement1 is executed (one time) before the execution of the code block
- Statement2 defines the condition for executing the code block
- Statement3 is executed (every time) after the code block has been executed

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For Loop</h2>

<p id="demo"></p>

<script>
var text = "";
var i;
for (i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JavaScript For Loop

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4

JavaScript For/Of Loop

❖ **The JavaScript for/of statement loops through values of an iterable objects**

- It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For/Of Loop</h2>

<p>The for/of statement loops through the values of an iterable
object.</p>

<script>
var cars = ['BMW', 'Volvo', 'Mini'];
var x;

for (x of cars) {
  document.write(x + "<br >");
}
</script>

</body>
</html>
```

JavaScript For/Of Loop

The for/of statement loops through the values of an iterable object.

BMW
Volvo
Mini

JavaScript For/in Loop

❖ **The JavaScript for/in statement loops through the properties of an object**

- for/in statement can also loop over the properties of an Array

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For/In Loop</h2>

<p>The for/in statement loops through the properties of an
object.</p>

<p id="demo"></p>

<script>
var txt = "";
var person = {fname:"John", lname:"Doe", age:25};
var x;
for (x in person) {
  txt += person[x] + " ";
}
document.getElementById("demo").innerHTML = txt;
</script>

</body>
</html>
```

JavaScript For/In Loop

The for/in statement loops through the properties of an object.

John Doe 25

JavaScript While Loop

❖ While

- loops through a block of code as long as a specified condition is true.

```
while (condition) {
  // code block to be executed
}
```

```
while (i < 10) {
  text += "The number is " + i;
  i++;
}
```

❖ Do/while

- This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

```
do {
  // code block to be executed
}
while (condition);
```

```
var i = 11;

do {
  text += "<br>The number is " + i;
  i++;
}
while (i < 10);
```

요 약

- JavaScript Data Types
- JavaScript Objects
- JavaScript String/Number/Array
- JavaScript Conditions/Switch
- JavaScript For/While