Back-end Web Development

February 18th, 2020

Dorset College Dublin

JavaScript

- **Functions**
  - Parameters and Arguments
  - Return
  - Function declaration
  - Function expression
- **History of JS**
- **ES6 syntax**

What we covered last week....

- **Arrow Functions**
  - Refactoring
  - Steps
  - Activities
- **Functions as Values**
- **Function Hoisting**

JavaScript

What we will cover today....

- **Functions**
  - Optional parameters
  - Default parameters
  - Variable-Length Argument lists
  - Rest parameters
  - Rest operator

- **Arrays**
  - Creating an array via array literal and constructor
  - Array holes
  - Adding and deleting array elements
  - Iterating arrays
  - Length
  - Const in arrays
  - Reading and writing array elements
  - Sparse arrays

Functions continued

Optional Parameters

- If a function is called with **missing arguments**(less than declared), the missing values are set to **undefined**
- It is often useful to write functions so that some **arguments are optional** and may be **omitted** when the function is invoked
- If you’re calling a function using the same values for some parameters, you can use optional Parameters to **avoid repeating yourself**

Optional Parameters (pre-ES6)

```
function connect(hostname, port, method){
    if (hostname === undefined) hostname = "localhost";
    if (port === undefined) port = 80;
    if (method === undefined) method = "HTTP";
}
```

Optional Parameters - ‘Pretty Version’ (pre-ES6)

```
function connect(hostname, port, method){
    hostname = hostname || "localhost";
    port = port || 80;
    method = method || "HTTP";
}
```

- The **OR operator** || returns the left side if the left argument is **truthy**
- Otherwise it checks if the right argument is truthy and returns it
- We can use the shortcut because **undefined is falsy:** undefined evaluates to false

Default Parameters (ES6)

```
function multiply(a, b = 1){
    return a * b;
}

multiply(5, 2);
multiply(5, 1);
multiply(5);
```

Activity

Refactor this function to use ES6 default parameters.

```
function connect(hostname, port, method){
    if (hostname === undefined) hostname = "localhost";
    if (port === undefined) port = 80;
    if (method === undefined) method = "HTTP";
}
```

Solution

```
function connect(hostname = "localhost", port = 80, method = "HTTP"){
    //function body
}
```

Variable-Length Argument Lists

- When a function is invoked with **more argument** values than there are **parameter names**, there is no way to directly refer to the unnamed values.

- **Arguments object** provides a solution to this problem.

- Within the body of a function, **arguments identifier** refers to the Arguments object.

- Arguments object is an **array-like object** that allows the argument values passed to the function to be retrieved by a **number** rather than a name.

- Does not apply to arrow functions.

Variable-Length Argument Lists

```
function sumOfArguments(){
    let sum = 0;
    for(let i = 0; i < arguments.length; i++){
        sum += arguments[i];
    }
    return sum
}
//Prints 15
console.log(sumOfArguments(1, 2, 3, 4, 5));
```

Rest Parameters (ES6)

- Introduced to ES6 to **clean up** variable-length argument list work.
- There 3 main differences between rest parameters and the arguments object:
  - Rest parameters are the only ones that haven't been given a **separate name**
  - Rest parameters are **real arrays**
  - Arguments object contains **all arguments passed** to the function
  - The arguments object is **not** a real array
  - Arguments object has **additional functionality** specific to itself

Rest Parameters

```
function pizzaBuilder(base, ...toppings){
    console.log('Number of toppings: ' + toppings.length);
    console.log(`You ordered a ${base} based pizza with the
    following toppings: ${toppings}`);
}

pizzaBuilder('thin', 'mushroom', 'pepperoni', 'peppers');
```

Rest (or Spread) Operator ...

```
let arr = [1,2,3];

example(...arr);

function example(var1, var2, var3){
    console.log(var1);
    console.log(var2);
    console.log(var3);
}
```

Allows an array to be **expanded** in places where zero or more arguments (in function calls) or elements (in array literals) are expected.

Arrays

An array is an **ordered collection** of values where each value is called an element. Each element has a numeric position in the array known as index.

- JS arrays are **untyped:** can be of any type (string, numerical, boolean)
- Array elements can be **objects** or **other arrays**
- JS arrays are always **zero-based:** index of the first element is 0

Creating Arrays via an array literal

```
let empty = []; // no element array

let primes = [2,3,5,7,11]; // array with 5 numerical elements

let misc = [1.1,true,'a']; // various data type array

let arr = [1,2,3,]; // an array with a trailing comma (ignored)

let arr2 = [{x:1, y:2}, {x:5, y:5}]; // an array containing 2 objects

let arr3 = [1, , 3]; // an array with 3 elements, middle one is undefined

let arr4 = [,,,,]; // an array with 4 undefined elements, trailing comma ignored
```

Creating Arrays using Array constructor (don’t)

```
let a = new Array(); // emprty array, avoid

let a123 = new Array(1,2,3); // array containing [1,2,3], avoid

let a10 = new Array(10); // array containing 10 empty values, avoid

let a100 = Array(100); // the new keyword is optional, avoid
```

Reading and Writing Array Elements

- Arrays are a specialized kind of object, which is a **map from indices** (natural numbers, starting at zero to arbitrary (not assigned to a specific value) values)

- The square brackets used to **access** array elements work the same way as the square brackets used to access **object** properties

- All indexes are property names, but only property names that are integers are indexes

Reading and Writing Array Elements

```
let donuts = ['chocolate', 'red velver', 'custard'];

console.log(donuts[8]); // undefined
```

Array indexes are a special type of object property name that do not have an **“out of bounds”** error.

If you query a nonexistent property of any object, you won’t get an error.

Sparse Arrays

```
let arr = [1,2, , 4,5, , 7];

console.log(arr.length); // 7
```

- A sparse array is one where the elements **do not** have a **continuous indexes** starting at 0.
- Normally the length property of an array specifies the number of elements in the array.
- If an array is sparse, the value of length property is **greater** than the number of elements

Holes

```
let arr = [1,2, , 4,5, , 7];
```

- Holes are indices inside an Array that have **no associated element**
- Bellow indices 2 and 5 are holes
- In ES6 holes are treated as **undefined**
- Holes are treated **inconsistently** in JS and should be avoided

Adding and Deleting Array Elements

```
let arr = [];
arr[0] = 'index 1';
arr[1] = 'index 2';
console.log(arr); // ['first', 'second']

let students = [];
students.push('Merrion');
students.push('Andrew', 'Ross');
console.log(students);

let donuts = ['chocolate', 'red velvet', 'custard'];
delete donuts[1];
console.log(donuts);
console.log(donuts.length);
// [ 'chocolate', <1 empty item>, 'custard' ]
```

Iterating Arrays using loops

```
let donuts = ['chocolate', 'red velvet', , , 'custard'];

for(let i = 0; i < donuts.length; i++){
    if(donuts[i]){ //skip elements that are undefined
        console.log(donuts[i]);
    }
}
```

length

```
let a = [1,2,3];
console.log(a.length); // 3
let b = [1, , 3]; // an array with a hole
console.log(b.length); // 3
```

The length property **tracks the highest index** in an array and not the number of elements.

length

```
let a = [1,2,3];
a.length = 5;
console.log(a); // 1,2,3, ,
```

If you **manually increase** the length of an array, it will just add more holes.

length

```
let a = [1,2,3];
a.length = 5;
console.log(a); // 1,2,3, ,

a.length = 0;
console.log(a); // []
```

If you set the length to **zero** then it will **empty** all the elements inside the array.

const in Arrays

```
const menuItems = ['pizza', 'chips', 'burger'];
menuItems.push('soda');
menuItems = [];
console.log(menuItems);
```

- You cannot change the value of a constant through **re-assignment**
- It’s possible to **add items** into the array
- Assigning a new array to the variable will throw an error

Lets Code!

Continue on with your lab on CodeAcademy.

https://www.codecademy.com/learn/introduction-to-javascript