Basics of function, array and objects

Outline

- Function
- Array
- Object

Function

Function

- A function is a JavaScript procedure—a set of statements that performs a task or calculates a value
- To use a function, you must define it somewhere in the scope from which you wish to call it

Defining Functions

A function definition (or function declaration, or function statement)

```
function square(num) {
    return num * num;
}
```

Passing Parameters to Functions

- Primitive parameters are passed to functions by value
- Object is passed to functions by reference

Function Expressions

JavaScript allows to assign a function to a variable and then use that variable as a function. It is called function expression

```
var add = function sum(val1, val2) {
    return val1 + val2;
};

var result1 = add(10,20);
var result2 = sum(10,20); // not valid
```

Function Expressions (Ex.)

```
const square = function sq(num) { return num * num }
var res = square(4)  // 16
```

```
const factorial = function fac(n){
    return n < 2 ? 1 : n * fac(n - 1)
}
res = factorial(4)  // 24
```

Anonymous Function

- An anonymous function is a function without a name.

```
(function () {
  //code
});
```

- If the anonymous function is not put inside the (), it will give a syntax error.
- The () makes the anonymous function an expression that returns a function object.
- An anonymous function is not accessible after its initial creation. Therefore, it is often needed to assign it to a variable.

Anonymous Function (Ex.1)

```
let show = function () {
    console.log('Anonymous function');
};

show();
// "Anonymous function"
```

Array

Array

- Arrays are generally described as "list-like objects"
- Array are basically single objects that contain multiple values stored in a list
- Array can hold values of mixed types.
- The size of an array is dynamic and auto-growing
- Data can be stored at non-contiguous locations in the array
- It cannot use strings as element indexes (as in an associative array) but must use integers

Creating and manipulating an Array

```
let shopping = ['bread', 'milk', 'noodles'];

console.log(shopping);

console.log(shopping[0]);

shopping[0] = 'cheese';
```

Creating an Array using **new**

```
let shopping = new Array('bread', 'milk', 'noodles');

console.log(shopping);

console.log(shopping[0]);

shopping[0] = 'cheese';
```

Create a new array from elements **(of)**

```
let arr = Array.of(1, "hello", {sub: "AT"})

console.log(arr[0]);  // 1

console.log(arr[1]);  //  "hello"

console.log(arr[2]);  // Object { sub: "AT" }

arr[33] = 4;

console.log(arr.length);    // 34
```

Check for an Array (**isArray**)

```
Array.isArray([1, 2, 3]);          // true

Array.isArray({num: 123}); // false

Array.isArray('foobar');           // false

Array.isArray(undefined);          // false
```

Looping array elements

```
let sequence = [1, 1, 2, 3, 5, 8, 13];

for (let x = 0; x < sequence.length; x++) {

  console.log(sequence[x]);

}
```

Merge two or more arrays (**concat**)

```
const array1 = ['a', 'b', 'c'];

const array2 = ['d', 'e', 'f'];

const array3 = array1.concat(array2);

console.log(array3);

//  ["a", "b", "c", "d", "e", "f"]
```

Add / remove item to / from **end** of an Array

```
let fruits = ['Apple', 'Banana']

let newLength = fruits.push('Orange') // 3

// ["Apple", "Banana", "Orange"]

let last = fruits.pop()              // "Orange"

// ["Apple", "Banana"]
```

Add / remove item to / from **begining** of an Array

```
let fruits = ['Apple', 'Banana']

let first = fruits.shift()                  // "Apple"


//  ["Banana"]

let newLength = fruits.unshift('Berry') // 2

// ["Berry", "Banana"]
```

Find the **index** of an item

```
let fruits = ['Apple', 'Banana'];

let pos = fruits.indexOf('Banana');

Console.log(pos);
```

Output: 1

Copy sub-array elements (**slice**)

```
const animals = ['ant', 'bison', 'camel', 'duck',
'elephant'];

console.log(animals.slice(2));   // ["camel",
"duck", "elephant"]

console.log(animals.slice(2, 4)); // ["camel",
"duck"]

console.log(animals.slice(1, 5)); // ["bison",
"camel", "duck", "elephant"]
```

Converting Array Elements to String (**join**)

```
const elements = ['Fire', 'Air', 'Water'];

console.log(elements.join());

//  "Fire,Air,Water"

console.log(elements.join(''));

//  "FireAirWater"

console.log(elements.join('-'));

//  "Fire-Air-Water"
```

Converting String to an Array (**split**)

```
const str = 'This is a test string';

const words = str.split(' ');

console.log(words[3]); // "test"

const chars = str.split('');

console.log(chars[8]);  //  "a"

const strCopy = str.split();

console.log(strCopy);   // Array ["This is a test string"]

const limitedWords = str.split(" ", 3);

console.log(limitedWords); // [‘This’, ‘is’, ‘a’]
```

Check for an item belongs to an Array (**includes**)

```
const num = [1, 2, 3];

console.log(num.includes(2));   //  true

const pets = ['cat', 'dog', 'bat'];

console.log(pets.includes('cat'));     // true

console.log(pets.includes('at'));      // false
```

Reversing array items in place (**reverse**)

```
const array1 = ['one', 'two', 'three'];

const reversed = array1.reverse();

console.log(reversed);     // ["three", "two",
"one"]

console.log(array1);     // ["three", "two",
"one"]
```

Sorting array items in place (**sort**)

Converts the elements into strings, then comparing their sequences of UTF-16 code units values:

```
const months = ['March', 'Jan', 'Feb', 'Dec'];
months.sort();
console.log(months);      //   ["Dec",   "Feb",   "Jan",
"March"]

const array1 = [1, 30, 4, 21, 1000];
array1.sort();
console.log(array1);    // [1, 1000, 21, 30, 4]
```

Object

Objects

- An Object is a non-primitive data type in JavaScript.
- It is like any other variable, the only difference is that an object holds multiple values in terms of properties and methods.
- Properties can hold values of primitive data types and methods or functions.
- Objects are sometimes called associative arrays, since each property is associated with a string value that can be used to access it

Object without Class

- In other programming languages like Java or C#, you need a class to create an object of it.
- In JavaScript, an object is a standalone entity because there is no class in JavaScript.
- However, you can achieve class like functionality using functions.

Object Creation

In JavaScript, an object can be created in two ways:

1. Object literal
2. Object constructor

Object Literal

- The object literal is a simple way of creating an object using { } brackets.
- You can include key-value pair in { }, where key would be property or method name and value will be value of property of any data type or a function.
- Use comma (,) to separate multiple key-value pairs.
- **Syntax:**

```
var <object-name> = {
        key1: value1,
        key2: value2, ...
        keyN: valueN
};
```

Object Literal : Examples

```
var emptyObject = {}; // object with no properties or methods

var person = { firstName: "John" }; // object with single property

// object with single method
var message = {
                showMessage: function (val) {
                            alert(val);
                }
            };
// object with properties & method
var person = {
                firstName: "James",
                lastName: "Bond",
                age: 15,
                getFullName: function () {
                        return this.firstName + ' ' + this.lastName
                }
            };
```

Object Literal

- You must specify key-value pair in object for properties or methods.
- Only property or method name without value is not valid. The following syntax is invalid.

Example: Wrong Syntax

```
var person = { firstName };

var person = { firstName: };
```

Object Constructor

- The second way to create an object is with Object Constructor using new keyword.
- You can attach properties and methods using dot notation.
- Optionally, you can also create properties using [ ] brackets and specifying property name as string.

Object Constructor : Example

```
var person = new Object();

// Attach properties and methods to person object
person.firstName = "James";
person["lastName"] = "Bond";
person.age = 25;
person.getFullName = function () {
        return this.firstName + ' ' + this.lastName;
    };
```

Accessing Object Properties & Methods

- You can get or set values of an object's properties using dot notation or bracket.
- However, you can call an object's method only using dot notation.

```
var person = {
                firstName: "James",
                lastName: "Bond",
                age: 25,
                getFullName: function () {
                    return this.firstName + ' ' + this.lastName
                }
            };

person.firstName; // returns James
person.lastName; // returns Bond

person["firstName"];// returns James
person["lastName"];// returns Bond

person.getFullName();
```

- An object's methods can be called using **()** operator
- e.g. person.getFullName().
- Without **()**, it will return function definition.

Undefined Property or Method

- JavaScript will return 'undefined' if you try to access properties or call methods that do not exist.
- If you are not sure whether an object has a particular property or not, then use hasOwnProperty() method before accessing properties.

```
var person = new Object();

person.firstName; // returns undefined

if(person.hasOwnProperty("firstName")){
            person.firstName;
    }
```

Access Object Keys

- Use **for..in** loop to get the list of all properties and methods of an object.

```
var person = new Object();

person.firstName = "James";
person["lastName"] = "Bond";
person.age = 25;
person.getFullName = function () {
        return this.firstName + ' ' + this.lastName;
    };

for(var key in person){
        alert(key);
    };
```

OR

```
Object.keys(person);
```

Pass by Reference (1)

- Object in JavaScript passes by reference from one function to another.

```
function changeFirstName(per)
{
    per.firstName = "Steve";
}

var person = { firstName : "Bill" };

changeFirstName(person)

person.firstName; // returns Steve
```

Pass by Reference (2)

- If, two objects point to the same object then the change made in one object will reflect in another object.

```
var person = { firstName : "John" };

var anotherPerson = person;

anotherPerson.firstName = "Bill";

person.firstName; //returns Bill
```

Nested Objects

- You can assign another object as a property of an object.

```
var person = {
    firstName: "James",
    lastName: "Bond",
    age: 25,
    address: {
        id: 1,
        country:"UK"
    }
};

person.address.country; // returns "UK"
```

Points to Remember

1. JavaScript object is a standalone entity that holds multiple values in terms of properties and methods.

2. Object property stores a literal value and method represents function.

3. An object can be created using object literal or object constructor syntax.

4. Object properties and methods can be accessed using dot notation or [ ] bracket.

5. An object is passed by reference from one function to another.

6. An object can include another object as a property.

References

- https://www.tutorialsteacher.com/javascript/javascript-object