WEB TECHNOLOGIES

Java Script

CAP 756

Faculty: Dr. Mirza Shuja

JavaScript Methods and this Keyword

- In JavaScript, objects can also contain functions. For example,

```
// object containing method
const person = {
    name: 'John',
    greet: function() { console.log('hello'); }
};
```

- **Accessing Object Methods**
- You can access an object method using a dot notation. The syntax is:
- **objectName.methodKey()**
- You can access property by calling an objectName and a key. You can access a method by calling an objectName and a key for that method along with (). For example,
- **// accessing method and property**

```
const person = {
    name: 'John',
    greet: function() { console.log('hello'); }
};
// accessing property
person.name; // John
// accessing method
person.greet(); // hello
```

- **JavaScript Built-In Methods**
- In JavaScript, there are many built-in methods. For example,
- **let number = '23.32';**
- **let result = parseInt(number);**
- **console.log(result); // 23**
- **Adding a Method to a JavaScript Object**

```
// creating an object
let student = { };
// adding a property
student.name = 'John';
// adding a method
student.greet = function() {
    console.log('hello');
}
// accessing a method
student.greet(); // hello
```

- **JavaScript this Keyword**
- To access a property of an object from within a method of the same object, you need to use the this keyword. Let's consider an example.

```
const person = {
    name: 'John',
    age: 30,
    // accessing name property by using this.name
    greet: function() { console.log('The name is' + ' ' + this.name); }
};
person.greet();
```

JavaScript Constructor Function

- In JavaScript, a constructor function is used to create objects. For example,
- // constructor function
- function Person () {
- this.name = 'John',
- this.age = 23
- }
- // create an object
- const person = new Person();

- **Create Multiple Objects with Constructor Function**
- In JavaScript, you can create multiple objects from a constructor function. For example,
- **// constructor function**

```
function Person () {
    this.name = 'John',
    this.age = 23,
     this.greet = function () {
        console.log('hello');
    }
}
// create objects
const person1 = new Person();
const person2 = new Person();
// access properties
console.log(person1.name);  // John
console.log(person2.name);  // John
```

JavaScript Constructor Function Parameters

- You can also create a constructor function with parameters. For example,

```
// constructor function
function Person (person_name, person_age, person_gender) {
  // assigning  parameter values to the calling object
   this.name = person_name,
   this.age = person_age,
   this.gender = person_gender,
   this.greet = function () {
      return ('Hi' + ' ' + this.name);
   }
}
// creating objects
const person1 = new Person('John', 23, 'male');
const person2 = new Person('Sam', 25, 'female');
// accessing properties
console.log(person1.name); // "John"
console.log(person2.name); // "Sam"
```

- In the last example, we have passed arguments to the constructor function during the creation of the object.
- **const person1 = new Person('John', 23, 'male');**
- **const person2 = new Person('Sam', 25, 'male’);**
- This allows each object to have different properties. As shown above,

- console.log(person1.name); gives John

- console.log(person2.name); gives Sam

JavaScript Arrays

- An array is an object that can store multiple values at once. For example,
- const words = ['hello', 'world', 'welcome’];
- **Create an Array**
- You can create an array using two ways:

1. Using an array literal

- The easiest way to create an array is by using an array literal []. For example,
- const array1 = ["eat", "sleep"];

2. Using the new keyword

- You can also create an array using JavaScript's new keyword.
- const array2 = new Array("eat", "sleep");
- In both of the above examples, we have created an array having two elements.

- Here are more examples of arrays:

```
// empty array
const myList = [ ];

// array of numbers
const numberArray = [ 2, 4, 6, 8];

// array of strings
const stringArray = [ 'eat', 'work', 'sleep'];

// array with mixed data types
const newData = ['work', 'exercise', 1, true];
```

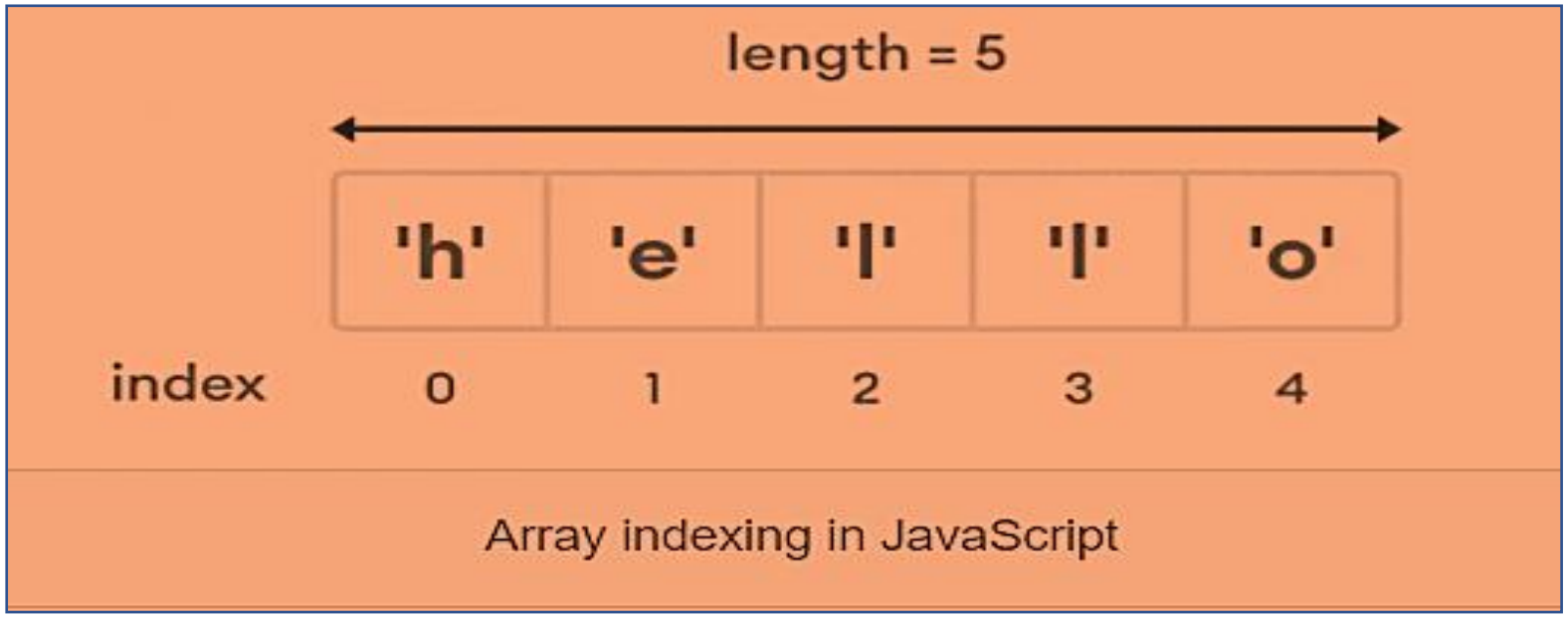
- You can also store arrays, functions and other objects inside an array. For example,

```
const newData = [
    {'task1': 'exercise'},
    [1, 2 ,3],
    function hello() { console.log('hello')}
];
```

Access Elements of an Array

- You can access elements of an array using indices **(0, 1, 2 …)**. For example,

```
const myArray = ['h', 'e', 'l', 'l', 'o'];
// first element
console.log(myArray[0]);  // "h"
// second element
console.log(myArray[1]); // "e"
```



- **Add an Element to an Array**
- You can use the built-in method **push()** and **unshift()** to add elements to an array.
- The **push()** method adds an element at the end of the array. For example,

```
let dailyActivities = ['eat', 'sleep'];
// add an element at the end
dailyActivities.push('exercise');
console.log(dailyActivities); //  ['eat', 'sleep', 'exercise’]
```

- The unshift() method adds an element at the beginning of the array. For example,

```
let dailyActivities = ['eat', 'sleep'];
//add an element at the start
dailyActivities.unshift('work');
console.log(dailyActivities); // ['work', 'eat', 'sleep']
```

- **Change the Elements of an Array**
- You can also add elements or change the elements by accessing the index value.

```
let dailyActivities = [ 'eat', 'sleep'];
// this will add the new element 'exercise' at the 2 index
dailyActivities[2] = 'exercise';
console.log(dailyActivities); // ['eat', 'sleep', 'exercise’]
```

- Suppose, an array has two elements. If you try to add an element at index 3 (fourth element), the third element will be undefined. For example,

```
let dailyActivities = [ 'eat', 'sleep'];
// this will add the new element 'exercise' at the 3 index
dailyActivities[3] = 'exercise';
console.log(dailyActivities); // ["eat", "sleep", undefined, "exercise"]
```

- **Remove an Element from an Array**
- You can use the **pop()** method to remove the last element from an array. The **pop()** method also returns the returned value. For example,

```
let dailyActivities = ['work', 'eat', 'sleep', 'exercise'];
// remove the last element
dailyActivities.pop();
console.log(dailyActivities); // ['work', 'eat', 'sleep’]

// remove the last element from ['work', 'eat', 'sleep']
const removedElement = dailyActivities.pop();

//get removed element
console.log(removedElement); // 'sleep'
console.log(dailyActivities);  // ['work', 'eat']
```

- If you need to remove the first element, you can use the **shift()** method.
- The **shift()** method removes the first element and also returns the removed element. For example,

```
let dailyActivities = ['work', 'eat', 'sleep'];
// remove the first element
dailyActivities.shift();
console.log(dailyActivities); // ['eat', 'sleep']
```

- **Array length**
- You can find the length of an element (the number of elements in an array) using the length property. For example,

```
const dailyActivities = [ 'eat', 'sleep'];


// this gives the total number of elements in an array
console.log(dailyActivities.length); // 2
```

- **Working of JavaScript Arrays**
- In JavaScript, an array is an **object**. And, the indices of arrays are **objects** keys.
- Since arrays are objects, the array elements are **stored by reference**. Hence, when an array value is copied, any change in the copied array will also reflect in the **original array**. For example,

```
let arr = ['h', 'e'];
let arr1 = arr;
arr1.push('l');
console.log(arr); // ["h", "e", "l"]
console.log(arr1); // ["h", "e", "l"]
```