Day 8 - Objects

Scope

A variable can be declared at different scopes. Variable scopes can be:

- Global
- Local

Variable can be declared globally or locally. Anything declared without let, var, or const is scoped at global level.

Window Global Object

A global object in JS is an always defined object that provides variables and functions, and is available anywhere. In a web browser, the global object is the window object. The global object can be accessed using the *this* operator in the global scope.

Global scope

A globally declared variable can be accessed everywhere in the same file, but, the term global is relative. It can be global to the file or global relative to some block of codes.

```
// scope.js
let a = 'JavaScript'
let b = 10 // Global scope; can be accessed anywhere in the file.
function letsLearnScope() {
  console.log(a, b) // JavaScript 10, accessible
  if (true) {
    let a = 'Python'
    let b = 100
    console.log(a, b) // Python 100
  }
  console.log(a, b)
}
letsLearnScope()
console.log(a, b) // JavaScript 10, accessible.
```

Local Scope

A variable declared as local can be accessed only in certain block code.

- Block Scope
- Function Scope

```
//scope.js
let a = 'JavaScript'
let b = 10
// Function scope
function letsLearnScope() {
  console.log(a, b) // JavaScript 10, accessible
  let value = false
// block scope
  if(true) {
    // we can access from the function and outside the function but
    // variables declared inside the if will not be accessed outside the if block
    let a = 'Python'
    let b = 20
    let c = 30
    let d = 40
    value = !value
    console.log(a, b, c, value) // Python 20 30 true
  }
  // We can't access the c variable because its scope is only the if block.
  console.log(a, b, c, value) // JavaScript 10 true
}
letsLearnScope() // JavaScript 10, accessible
```

A variable declared with *var* only scopes to function, while a variable declared with *let* or *const* is block scope (function block, if block, loop block, etc.). Block is code between two curly brackets ({}).

```
//scope.js
function letsLearnScope() {
  var gravity = 9.81
  console.log(gravity)

}
// console.log(gravity), Uncaught ReferenceError: gravity is not defined

if (true){
  var gravity = 9.81
  console.log(gravity) // 9.81
}
console.log(gravity)  // 9.81

for(var i = 0; i < 3; i++){
  console.log(i) // 0, 1, 2
}
console.log(i) // 3
```

In ES6 and above there is let and const, so we don’t necessarily *need* to use var. Using let scopes our variable to the block and won’t infect other parts of our code.

```
//scope.js
function letsLearnScope() {
  // you can use let or const, but gravity is constant I prefer to use const
  const gravity = 9.81
  console.log(gravity)
```

```
}
// console.log(gravity), Uncaught ReferenceError: gravity is not defined

if (true){
  const  gravity = 9.81
  console.log(gravity) // 9.81
}
// console.log(gravity), Uncaught ReferenceError: gravity is not defined

for(let i = 0; i < 3; i++){
  console.log(i) // 0, 1, 2
}
// console.log(i), Uncaught ReferenceError: i is not defined
```

The scope of *let* and *const* are the same, and the only difference is reassigning. We should use *let* for any value that can change and *const* for any constant value, array, object, arrow function and function expression.

Object

Everything can be an object, objects have properties, and properties have values, so an object is a key value pair. The order of keys is not reserved, or there is no order. We can create an object literal using two curly brackets.

Creating an empty object

```
const person = {}
```

Creating an object with values

Below, we add to the person object the following properties: firstName, lastName, age, location, skills and isMarried. Property values or keys could be a string, number, boolean, an object, null, undefined or a function.

```
const rectangle = {
  length: 20,
  width: 20
}
console.log(rectangle) // {length: 20, width: 20}

const person = {
  firstName: 'Chris',
  lastName: 'Guerrero',
  age: 250,
  country: 'US',
  city: 'Phoenix',
  skills: [
```

```
    'HTML',
    'CSS',
    'JavaScript',
    'React',
    'Node',
    'MongoDB',
    'Python',
    'D3.js'
  ],
  isMarried: false
}
console.log(person)
```

Getting values from an object

We can access values of an object using two methods:

- using . followed by key name if the key-name is one word
- using square bracket and a quote

```
const person = {
  firstName: 'Chris',
  lastName: 'Guerrero',
  age: 250,
  country: 'US',
  city: 'Phoenix',
  skills: [
    'HTML',
    'CSS',
    'JavaScript',
    'React',
    'Node',
    'MongoDB',
    'Python',
    'D3.js'
  ],
  getFullName: function() {
    return `${this.firstName}${this.lastName}`
  },
  'phone number': '+3584545454545'
}

// accessing values using .
console.log(person.firstName)
console.log(person.lastName)
console.log(person.age)
console.log(person.location) // undefined

// value can be accessed using square bracket and key name
console.log(person['firstName'])
console.log(person['lastName'])
console.log(person['age'])
console.log(person['age'])
console.log(person['location']) // undefined
```

```
// for instance to access the phone number we only use the square bracket method
console.log(person['phone number'])
```

Creating object methods

Above, the getFullName property is a function inside the person object, which we call an object method. The *this* keyword refers to the object itself. We can use the word *this* to access the values of different properties of the object. We can't use an arrow function as an object method because the word *this* refers to the window inside an arrow function instead of the object itself.

```
const rectangle = {
  length: 10
  width: 5
  getPerimeter: function() {
    return ((2 * this.length) + (2 * this.width))
  }
}
```

Setting new key for an object

An object is a mutable data structure and we can modify the content of an object after it gets created.

```
const person = {
  firstName: 'Chris',
  lastName: 'Guerrero',
  age: 250,
  country: 'US',
  city: 'Phoenix',
  skills: [
    'HTML',
    'CSS',
    'JavaScript',
    'React',
    'Node',
    'MongoDB',
    'Python',
    'D3.js'
  ],
  getFullName: function() {
    return `${this.firstName}${this.lastName}`
  },
  'phone number': '+3584545454545'
}

person.nationality = 'Dominican'
person.title = 'student'
person.skills.push('Express')

person.getPersonInfo = function() {
  let skillsWithoutLastSkill = this.skills
```

```
  .splice(0, this.skills.length - 1)
  .join(', ')
  let lastSkill = this.skills.splice(this.skills.length - 1)[0]

  let skills = `${skillsWithoutLastSkill}, and ${lastSkill}`
  let fullName = this.getFullName()
  let statement = `${fullName} is a ${this.title}.\nHe lives in ${this.country}.\nHe teaches ${skills}.`
  return statement
}
console.log(person.getPersonInfo())
// Chris Guerrero is a student.
// He lives in US.
// He teaches HTML, CSS, JavaScript, React, Node, MongoDB, Python, D3.js, and Express.
```

Object Methods

There's different methods to manipulate an object. Here are some of the available methods.

Object.assign: To copy an object without modifying the original object.

```
//Object methods: Object.assign, Object.keys, Object.values, Object.entries
//hasOwnProperty

const copyPerson = Object.assign({}, person)
console.log(copyPerson)
```

Getting object keys using Object.keys()

Object.keys: To get the keys or properties of an object as an array

```
const keys = Object.keys(copyPerson)
console.log(keys) // ['firstName', 'age', 'country','city', 'skills','title', 'getPersonInfo']
const age = Object.keys(copyPerson.age)
```

Getting object values using Object.values()

Object.values: To get values of an object as an array

```
const values = Object.values(copyPerson)
console.log(values)
```

Getting object keys and values using Object.entries()

Object.entries: To get the keys and values in an array

```
const entries = Object.entries(copyPerson)
console.log(entries)
```

Checking properties using hasOwnProperty()

hasOwnProperty: To check if a specific key or property exist in an object

```
console.log(copyPerson.hasOwnProperty('name'))
console.log(copyPerson.hasOwnProperty('score'))
```