ECMAScript

ECMAScript, also known as JavaScript, is a programming language adopted by the European Computer Manufacturer's Association as a standard for performing computations in Web applications. ECMAScript is the official client-side scripting language of VoiceXML. ECMAScript is a limited programming model for simple data manipulation.

# 1 Variables and Data Structures

## 1.1 Let

<body>

<style>

#box {

display: flex;

justify-content: space-around;

}

#box > div {

width: 5rem;

height: 5rem;

background-color: blueviolet;

}

</style>

<h1>Let</h1>

<div id="box"></div>

<script type="text/javascript">

var div

var box = document.getElementById('box')

for (let i = 0; i < 5; i++) {

div = document.createElement('div')

div.onclick = function () {

alert('this is box # ' + i)

}

box.appendChild(div)

}

</script>

</body>

## 1.2 Template String

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8" />
5. <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6. <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7. <title>Template String</title>
8. <link rel="stylesheet" href="style.css" />
9. </head>
10. <body>
11. <div class="login">
12. <h1>Using template String</h1>
13. <label for="fname">First Name:</label>
14. <input type="text" id="fname" name="fname" />
15. <label for="price">Price:</label>
16. <input type="number" name="price" id="price" />
17. <button onclick="createEmail()">Print Email</button>
18. </div>
19. <script>
20. function createEmail() {
21. const shipping = 5.95
22. var fname, price
23. fname = document.getElementById('fname').value
24. price = document.getElementById('price').value
25. if (!fname) alert('please enter the value for the first name')
26. if (!price) alert('please enter the value for the price value')
27. let result = document.createElement('div')
28. if (price && fname) {
29. result.classList.add('result')
30. const email = ` <p>
31. Hi ${fname}! Thanks!
32. Total: $${price}
33. Shipping: $${shipping}
34. Grand Total: $${price + shipping}
35. </p>
36. `
37. result.innerHTML = email
38. let form = document.querySelector('div')
39. form.lastChild.remove()
40. form.append(result)
41. }
42. }
43. </script>
44. </body>
45. </html>

## 1.3 Searching String

const planet = "Earth";

console.log(planet.startsWith("ear")); // return false as its case sensitive.

console.log(planet.endsWith("rth")); // true

console.log(planet.includes("art")) // true

console.log(planet.search("rth"));// 2

## 1.4 Symbol

const id = Symbol()

const courseInfo = {

title: 'javaScript',

topics: ['string', 'arrays', 'objects'],

id: 'js-course',

}

courseInfo[id] = 41283

console.log(courseInfo)

1.5 Map

Why would you want to use a map? Well, you might want to use something other than a string as a key. Unlike objects, maps iterate its elements in their insertion order. So unlike objects that can come back in any order, the key value pairs can come back in any order, map objects are always going to deliver those back to us in the order in which they were added. So, maps are a useful data type in JavaScript that are fairly new to the language, but you can use them for all sorts of interesting things.

let course = new Map()

course.set('react', { description: 'ui' })

course.set('jest', { description: 'testing' })

console.log(course)

/\* wrong \*/ console.log(course.react) // will return undefined as is not in object.

//right

console.log(course.get('react'))

let details = new Map([

[new Date(), 'today'],

[2, { javascript: ['js', 'node', 'react'] }],

['item', [1, 2]],

])

console.log(details)

details.forEach((item) => {

console.log(item)

})

## 1.5 Set()

let books = new Set()

books.add('Pride and Prejudice')

books.add('War and Peace').add('Oliver Twist')

console.log('Books', books)

books.add('War and Peace')

// set will not add the same value

console.log('How many books inside the set?', books.size)

books.delete('Oliver Twist')

// you can't use map with set

let i = 1

books.forEach((item) => {

console.log(i + '-', item)

i++

})

console.log('has Oliver Twist', books.has('Oliver Twist'))

Result

Books Set(3) {'Pride and Prejudice', 'War and Peace', 'Oliver Twist'}

How many books inside the set? 3

1- Pride and Prejudice

2- War and Peace

has Oliver Twist false

# 2 Array and Arrays method

## 2.1 Spread Operator

let cats = ['Biscuit', 'Jungle']

let dogs = ['stella', 'camper']

//without spread Operator

let animals = ['smoky', 'Miro', 'Swimmy', cats, dogs]

console.log('without spread Operator', animals)

//with spread operator

animals = ['smoky', 'Miro', 'Swimmy', ...cats, ...dogs]

console.log('with spread operator', animals)

Result

without spread Operator

(5)

 ['smoky', 'Miro', 'Swimmy', Array(2), Array(2)]

0: "smoky"1: "Miro"2: "Swimmy"3: (2) ['Biscuit', 'Jungle']4: (2) ['stella', 'camper']

with spread operator (7)

 ['smoky', 'Miro', 'Swimmy', 'Biscuit', 'Jungle', 'stella', 'camper']

## 2.2 Destruction Array

let [s1, s2, s3, s4, s5] = [

'Spokane',

' Boston',

'Los Angeles',

'Seattle',

'Portland',

]

console.log(s1, s2, s3)

## 2.3 Includes

let cities = [

'spokane',

' Boston',

' Los Angeles ',

' Seattle',

'Portland',

].map((city) => city.toLowerCase().replace(' ', '')) // only delete the first space.

console.log(cities.includes('los angeles'))

console.log(cities)

# 3.Objects

## 3.1 Object Literals

function skier(name, sound) {

return {

name,

sound,

powderYell: function () {

let yell = this.sound.toUpperCase()

console.log(`${yell}! ${yell}!`)

},

}

}

console.log(skier('Sendy', 'Yeah'))

skier('Ali', 'woo').powderYell()

## 3.2 Object spread

const daytime = {

breakfast: 'oatmeal',

lunch: 'peanut butter and jelly',

}

const nighttime = 'mac and cheese'

const backpackingMeals = {

...daytime,

nighttime,

}

console.log(backpackingMeals)

## 3.4 Destructing

const vacation = {

destination: 'Chile',

travellers: 2,

activity: 'skiing',

cost: 'so much',

}

function marketing({ destination, activity }) {

return `Come to ${destination} and do some ${activity}`

}

console.log(marketing(vacation))

## 3.5 for of

let topics = new Map()

topics.set('html', '/topic/html'),

topics.set('CSS', 'topic/css'),

topics.set('JavaScript', 'topic/javascript'),

console.log(topics)

for (let topic of topics.keys()) console.log(topic)

for (let topic of topics.values()) console.log(topic)

for (let topic of topics.entries()) console.log(topic)

## 3.6 Class

class Vehicle {

constructor(description, wheels) {

this.description = description

this.wheels = wheels

}

DescribeYourself() {

return ` I am a ${this.description}

with ${this.wheels} wheels.`

}

}

let coolSkiVan = new Vehicle('cool ski van', 4)

console.log(coolSkiVan.DescribeYourself())

## 3.7 inheritance with JavaScript class

class Vehicle {

constructor(description, wheels) {

this.description = description

this.wheels = wheels

}

DescribeYourself() {

console.log(` I am a ${this.description}

with ${this.wheels} wheels.`)

}

}

class SemiTruck extends Vehicle {

constructor() {

super('semi truck', 18)

}

}

let groceryStoreSemi = new SemiTruck()

groceryStoreSemi.DescribeYourself()

## 3.8 Get and Set

let attendance = {

\_list: [],

set addName(name) {

this.\_list.push(name)

},

get list() {

return this.\_list.join(', ')

},

}

attendance.addName = 'joanne starr'

attendance.addName = 'Bill Benkelman'

attendance.addName = 'Charlie Charlson'

console.log(attendance.list)

# 4. Function

## 4.1 Repeat

let yell = 'woo!'

let party = yell.repeat(10)

console.log(party)

let cat = {

meow(times) {

console.log('meow '.repeat(times))

},

}

cat.meow(4)

## 4.2 default parameters

function add(x = 3, y = 4) {

console.log(x + y)

}

add(1, 2) // 3

add() // 7

## 4.3 Arrow function

let list = ['apple', 'banana', 'strawberry']

list.map((item) => console.log(item))

## 4.5 This in arrow function

let person = {

first: 'Angie',

hobbies: ['bike', 'hike', 'ski'],

printHobbies: function () {

// first this refer to the outside the function

this.hobbies.forEach((hobby) => {

// this refer to the inside the function

let string = `${this.first} like to ${this.hobbies}`

consel.log(string);

})

},

}

## 4.6 Generator

function\*

The function\* declaration (function keyword followed by an asterisk) defines a generator function, which returns a Generator object.

function\* generator(i) {

yield i

yield i + 10

}

const gen = generator(10)

console.log(gen.next().value)

// expected output: 10

console.log(gen.next().value)

// expected output: 20

6. Asynchronous JavaScript

The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

A callback used to initialize the promise. This callback is passed two arguments:

a resolve callback used to resolve the promise with a value or the result of another promise,

and a reject callback used to reject the promise with a provided reason or error.

const delay = (seconds) =>

new Promise((resolve, reject) => {

if (typeof seconds !== 'number') {

reject(new Error('seconds must be numbers'))

}

setTimeout(resolve, seconds \* 1000)

})

console.log('zero seconds')

delay(2).then(() => console.log('2 sec'))