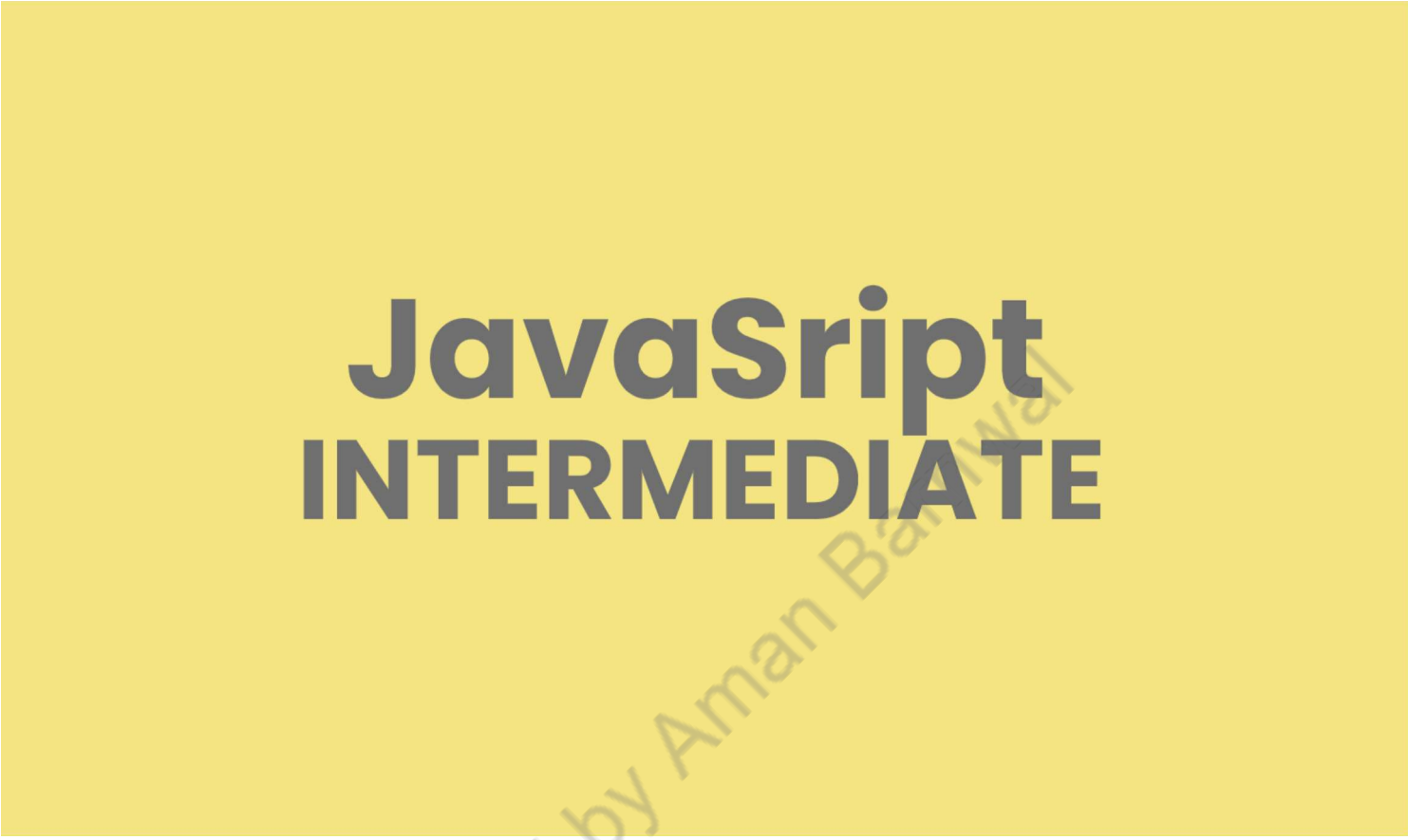
JavaScript Intermediate Part- 1



This document covers these topics modern JavaScript, ES6, let, const, function default parameter, template string, arrow function, spread operator, concatenate multiple arrays, class, constructor, object from class, inheritance, extended class, super,destructing object, truthy vs falsy, null vs undefined, double vs triple equal, map, filter, find, scope, block scope, closure, encapsulation, array slice, splice, join.

1 MODERN JAVASCRIPT, ES6, ES2015, ECMA SCRIPT 2015

1.1 LET, CONST, ARRAY DECLARED WITH CONST, OBJECT DECLARED WITH CONST

JAVASCRIPT

```
const name = 'Jane Smith' // if value will not change;


const numbers = [12, 45]
```

```
numbers[1] = 88 // can change value
numbers.push(15) // can add new value
console.log(numbers) // [ 12, 88, 15 ]

//const numbers = [50, 60, 34]; // can't change the variable with new array

const person = {
  name: 'Will Smith',
  phone: 000 - 000 - 0000,
} // can add, change property value but can't change the variable with new object

// if value may change use let, let is scope variable
let userName = 'Ed Sheeran'
userName = 'Adam Levine'

let sum = 0
for (var i = 0; i < 10; i++) {
  // var will be accessible outside the loop
  sum = sum + i
}
console.log(i) // 10

// use let instead
let sum = 0
for (let i = 0; i < 10; i++) {
  sum = sum + i
}
```

1.2 FUNCTION DEFAULT PARAMETER FOR NOT PROVIDED VALUES

JAVASCRIPT

```
function add(num1, num2) {
  // if num2 is not provided
  // way 1:
  // if (num2 == undefined) {
  //     num2 = 0;
  // }
```

```
  // way 2:
  num2 = num2 || 20
  return num1 + num2
}

// way 3: ES6 convention
function add(num1, num2 = 0) {
  // = 0 is default parameter value
  return num1 + num2
}

const total = add(15, 17)
console.log(total) //32
const total2 = add(15)
console.log(total2) // 15
const total3 = add(15, 1)
console.log(total3) // 16
```

1.3 TEMPLATE STRING, MULTIPLE LINE STRING

JAVASCRIPT

```
const firstName = 'Justin'
const lastName = 'TimerLake'
const fullName = firstName + ' ' + lastName + ' is a good boy'
console.log(fullName)
// Justin TimerLake is a good boy

// Create ES6 template instead:
const fullName2 = `${firstName} ${lastName} is a nice person.`
console.log(fullName2)
// Justin TimerLake is a nice person.

const fullName3 = `${firstName} ${20 + 50 + 30} is a nice person.`
console.log(fullName3)

// Before
const multiLine = 'I love you\n' + 'I miss you\n' + 'I need you'
console.log(multiLine)
```

```
// ES6:
const multiLine2 = `I love you
I miss you
I need you`
console.log(multiLine2)

/* I love you
I miss you
I need you */
```

1.4 ARROW FUNCTION, MULTIPLE PARAMETER, FUNCTION BODY

JAVASCRIPT

```
function doubleIt(num) {
  return num * 2
}
console.log(doubleIt(2)) // 4

const doubleIt2 = function myFun(num) {
  return num * 3
}
console.log(doubleIt2(2)) // 4

// Arrow Function:
// if one parameter
const doubleIt3 = (num) => num * 2
console.log(doubleIt3(5)) //25

// if more than one parameter
const add = (x, y) => x + y
console.log(add(4, 3)) // 7

// if no parameter
const noParameter = () => 5
console.log(noParameter()) // 5

const doMath = (x, y) => {
```

```
  const sum = x + y
  const diff = x - y
  const result = sum * diff
  return result
}

console.log(doMath(7, 5)) // 24
```

1.5 SPREAD OPERATOR, CONCATENATE MULTIPLE ARRAYS, ARRAY MAX

JAVASCRIPT

```
const ages = [12, 14, 16, 13, 17]
const ages2 = [15, 16, 12]
const ages3 = [25, 36, 22, 29]

// const allAges = ages.concat(ages2).concat(ages3);
const allAges = ages.concat(ages2).concat([5]).concat(ages3)
console.log(allAges)
// [ 12, 14, 16, 13, 17, 15, 16, 12, 5, 25, 36, 22, 29 ]

const allAges2 = [ages, ages2, 5, ages3] // array inside array
console.log(allAges2)
// [ [ 12, 14, 16, 13, 17 ], [ 15, 16, 12 ], 5, [ 25, 36, 22, 29 ] ]

// spread operator
const allAges3 = [...ages, ...ages2, 5, ...ages3] // new array
console.log(allAges3)
// [ 12, 14, 16, 13, 17, 15, 16, 12, 5, 25, 36, 22, 29 ]

const business = 650
const minister = 450
const sochib = 250

const maximum = Math.max(business, minister, sochib)
console.log(maximum) // 650

// if array
const amount = [650, 450, 250]
```

```
const maxAmount = Math.max(...amount)
console.log(maxAmount) // 650
```

1.6 CLASS, CONSTRUCTOR, CREATE OBJECT FROM CLASS

JAVASCRIPT

```
class Student {
  constructor(sId, sName) {
    this.id = sId //property that can change, pass as parameter
    this.name = sName //property that can change, pass as parameter
    this.school = 'E-School' //shared property
  }
}

const student1 = new Student(12, 'shuvo')
const student2 = new Student(22, 'Mahiya')
console.log(student1, student2)
// Student { id: 12, name: 'shuvo', school: 'E-School' } Student { id: 22, name:

// access attribute:
console.log(student1.name, student2.name) // shuvo Mahiya

// new
const student3 = new Student(22, 'Bappi')
console.log(student3)
// Student { id: 22, name: 'Bappi', school: 'E-School' }
```


1.7 INHERITANCE, EXTENDS CLASS, SUPER, CLASS METHOD

JAVASCRIPT

```
class Parent {
  constructor() {
    this.fatherName = 'Schwarzenegger'
  }
}
```

```
class Child extends Parent {
  constructor(name) {
    // call Parent class constructor
    super()
    this.name = name
  }

  // create function, not required to specify function keyword
  getFullName() {
    return this.name + ' ' + this.fatherName
  }
}

const baby = new Child('Arnold')
const baby2 = new Child('Tom')
console.log(baby) // Child { fatherName: 'Schwarzenegger', name: 'Arnold' }
console.log(baby2) // Child { fatherName: 'Schwarzenegger', name: 'Tom' }

console.log(baby.getFullName()) // Arnold Schwarzenegger
console.log(baby2.getFullName()) // Tom Schwarzenegger

// inheritance, encapsulation, polymorphism
```


1.8 DESTRUCTURING, OBJECT, ARRAY, DESTRUCTURING COMPLEX OBJECT

JAVASCRIPT

```
const person = {
  name: 'Jack William',
  age: 17,
  address: 'Brooklyn',
  sibling: 'Ema Watson',
  phone: '000 - 000 - 0000',
  job: 'Facebook Analyst',
  friends: ['Tom Hancks', 'Tom Cruise', 'Will Smith'],
}

// way 1:
```

```
console.log(person.sibling) // Ema Watson
console.log(person.sibling) // Ema Watson
console.log(person.sibling) // Ema Watson
console.log(person.sibling) // Ema Watson

// way 2:
const sibling1 = person.sibling
const phoneNumber = person.phone
console.log(sibling1, phoneNumber) // Ema Watson 000 - 000 - 0000
console.log(sibling1, phoneNumber) // Ema Watson 000 - 000 - 0000
console.log(sibling1, phoneNumber) // Ema Watson 000 - 000 - 0000
console.log(sibling1, phoneNumber) // Ema Watson 000 - 000 - 0000

// way 3: efficient and convenient
// one variable
const { phone } = person

// same as above
// const {
//     phone
// } = {
//     name: 'Jack William',
//     age: 17,
//     address: 'Brooklyn',
//     sibling: "Ema Watson",
//     phone: "000 - 000 - 0000",
//     job: "Facebook Analyst",
//     friends: ['Tom Hancks', 'Tom Cruise', "Will Smith"]
// };

console.log(phone) // 000 - 000 - 0000

// more than one variable
const { name, sibling, job } = person

console.log(name, sibling, job) // Jack William Ema Watson Facebook Analyst

// if any property is not in object
const {
  address,
```

```
  salary, // is not in person object, will give undefined
} = person

console.log(address, salary) // Brooklyn undefined

// array destructuring
const friends2 = [
  'Sakib Khan',
  'Arman Khan',
  'Aamir Khan',
  'Salman Khan',
  'Sharukh Khan',
]
const [first, nextFriend, ...restFriends] = friends2
console.log(first) // Sakib Kha
console.log(restFriends) // [ 'Aamir Khan', 'Salman Khan', 'Sharukh Khan' ]

// accessing another object property from an object using deconstructing
const complexObject = {
  name: 'abc',
  info: {
    city: 'New Rochelle',
    state: 'NY',
  },
}

const { state } = complexObject.info
console.log(state) // NY
```


Additional Resources- 1

Additional Resources- 2

2 INTERMEDIATE JAVASCRIPT, INTERVIEW QUESTIONS

2.1 TRUTHY AND FALSY VALUES

JAVASCRIPT

```javascript
// Falsy: 0, "", undefined, null, NaN, let name=false
// Truthy: '0', " ", [], let name='false'

// 0 will return false, for other value true
const age = 0
// const age = 4;
// const age = 6;

if (age) {
  console.log('condition is true')
} else {
  console.log('condition is false')
}

// empty string will return false, other string will be true
let name = ''
// const name = "Solaiman";

if (name) {
  console.log('condition is true')
} else {
  console.log('condition is false')
}

// if variable value is not defined, false will return
let name
console.log(name) // undefined
if (name) {
  console.log('condition is true')
} else {
  console.log('condition is false')
}

// if variable value is null, false will return
let name = null

if (name) {
  console.log('condition is true')
```

```
} else {
  console.log('condition is false')
}

// if variable value is NaN, false will return
let name = NaN

if (name) {
  console.log('condition is true')
} else {
  console.log('condition is false')
}
```

2.2 NULL VS UNDEFINED, DIFFERENT WAYS YOU WILL GET UNDEFINED

JAVASCRIPT

```
// Common cases that return undefined
// 1.
let name
console.log(name) //undefined

// 2.
function add(num1, num2) {
  console.log(num1 + num2) // if you're not returning anything
}
console.log(add(13, 82)) // undefined

// 3.
function add(num1, num2) {
  console.log(num1 + num2)
  return // if typed return, bot not specify what to return
}
console.log(add(13, 82)) // undefined

// 4.
function add(num1, num2) {
  console.log(num1, num2)
}
```

```
console.log(add(13)) // undefined - if less parameter value is passed than requir
// set default value can resolve the issue

// 5.
const car = {
  make: 'Tesla',
  currentSpeed: '30 mph',
  color: 'blue',
  fuelType: 'Electric',
}
console.log(car.mpg) // undefined - if tried to access object property which is n

// 6.
let fun = undefined // if assigned undefined as variable value
console.log(fun)

// 7.
let ageArray = [23, 25, 18]
console.log(ageArray[10]) // if tried to access the element which is not exist

// The value null represents the intentional absence of any object value.
// It is one of JavaScript's primitive values and is treated as falsy for boolean

// Exercise
function doSomething(x, y) {
  console.log(y)
}
console.log(doSomething(32)) // undefined
```


2.3 DOUBLE EQUAL (==) VS TRIPLE EQUAL (===), IMPLICIT CONVERSION

JAVASCRIPT

```
// == will check value only but not the type, so this will return true
let first = 2
let second = '2'
// let first = 1;
// let second = true;
```

```
// let first = 0;
// let second = false;

if (first == second) {
  console.log('Condition is true')
} else {
  console.log('Condition is false')
}

// === will check the both value and type also, so this will return false
let first = 2
let second = '2'
// let first = 1;
// let second = true;
// let first = 0;
// let second = false;

if (first === second) {
  console.log('Condition is true')
} else {
  console.log('Condition is false')
}
```


2.4 MAP, FILTER, FIND, SMART WAY TO RUN FOR LOOP

JAVASCRIPT

```
const numbers = [3, 4, 5, 6, 7, 9]
const output = []

for (let i = 0; i < numbers.length; i++) {
  const element = numbers[i]
  const result = element * element
  output.push(result)
}
console.log(output) // [ 9, 16, 25, 36, 49, 81 ]

// using map
function square(element) {
```

```
  return element * element
}

// arrow function, above function can be written as
const square2 = (element) => element * element
const square3 = (x) => x * x

//numbers.map(square);
numbers.map(function (element, index, array) {
  // map can take three parameter
  console.log(element, index, array)
  /*  3 0 [ 3, 4, 5, 6, 7, 9 ]
    4 1 [ 3, 4, 5, 6, 7, 9 ]
    5 2 [ 3, 4, 5, 6, 7, 9 ]
    6 3 [ 3, 4, 5, 6, 7, 9 ]
    7 4 [ 3, 4, 5, 6, 7, 9 ]
    9 5 [ 3, 4, 5, 6, 7, 9 ] */
})

const result = numbers.map(function (element) {
  return element * element
})
console.log(result) // [ 9, 16, 25, 36, 49, 81 ]

// map and arrow function
const result2 = numbers.map((x) => x * x)
console.log(result2) // [ 9, 16, 25, 36, 49, 81 ]

// filter
const numbers2 = [3, 4, 5, 6, 7, 9]
const bigger = numbers2.filter((x) => x > 5) // will return matching element base
console.log(bigger) // [ 6, 7, 9 ]

// find
const numbers3 = [3, 4, 5, 6, 7, 9]
const isThere = numbers3.find((x) => x > 5) // will return first matching element
console.log(isThere) // 6

// forEach, reduce
```

2.5 APPLY MAP, FILTER, FIND ON AN ARRAY OF OBJECTS

JAVASCRIPT

```
const students = [
  {
    id: 31,
    name: 'Jeff Bezos',
  },
  {
    id: 31,
    name: 'Elon Musk',
  },
  {
    id: 41,
    name: 'Bill Gates',
  },
  {
    id: 71,
    name: 'Tim Cook',
  },
]

const names = students.map((s) => s.name)
console.log(names) // [ 'Jeff Bezos', 'Elon Musk', 'Bill Gates', 'Tim Cook' ]

const ids = students.map((s) => s.id)
console.log(ids) // [ 31, 31, 41, 71 ]

const bigger = students.filter((s) => s.id > 40)
console.log(bigger) // [ { id: 41, name: 'Bill Gates' }, { id: 71, name: 'Tim Coo

const biggerOne = students.find((s) => s.id > 40)
console.log(biggerOne) // { id: 41, name: 'Bill Gates' }
```


2.6 SCOPE, BLOCK SCOPE, ACCESS OUTER SCOPE VARIABLE

JAVASCRIPT

```javascript
let bonus = 20 //global scope, accessible any where

function sum(first, second) {
  let result = first + second + bonus
  return result //local scope, not accessible outside function
}

const output = sum(3, 7)
console.log(output) // 30

//console.log(result); // return error
console.log(bonus) // 20

function sum2(first, second) {
  let result = first + second
  if (result > 9) {
    // block scope and hoisting
    // if let and const is used in block scope then hoisting will not happen
    // but var will hoist, so accessible outside block scope
    let mood = 'Happy'
    // const mood ="Happy"
    mood = 'Fishy'
    mood = 'Funky'
    mood = 'Cranky'
    console.log(mood)
  }
  //console.log(mood); // throw error because it outside the block scope if let o
  return result
}
const output2 = sum2(6, 7)
console.log(output2) // Cranky 13

// Difference when var and let used and try to access variable before variable de
//console.log(day); //Error: undefined
var day = 'Friday'
console.log(day)

//console.log(dayLet); // Error: Cannot access 'dayLet' before initialization
```

```
let dayLet = 'Saturday'
console.log(dayLet)
```

2.7 CLOSURE, ENCAPSULATION, PRIVATE VARIABLE

JAVASCRIPT

```
// Closure:
/* A closure is the combination of a function bundled together (enclosed) with re
to its surrounding state (the lexical environment). In other words, a closure giv
access to an outer function’s scope from an inner function. In JavaScript, closur
created every time a function is created, at function creation time. */

function stopWatch() {
  let count = 0
  return function () {
    count++
    return count
  }
}

const clock1 = stopWatch()
console.log(clock1()) // 1
console.log(clock1()) // 2
console.log(clock1()) // 3
console.log(clock1()) // 4

const clock2 = stopWatch()
console.log(clock2()) // 1
console.log(clock2()) // 2
console.log(clock1()) // 5
console.log(clock2()) // 3
```


2.8 ARRAY SLICE, SPLICE, ARRAY JOIN ELEMENTS

JAVASCRIPT

```
const nums = [1, 2, 3, 4, 5, 6, 7, 8]

// slice
const part = nums.slice(2, 5) // start index, excluding end index, select based o
console.log('slice: ', part) // [3, 4, 5]
console.log('array after slice: ', nums) // original array will remain same

// splice
const removed = nums.splice(2, 3) // here 2 is index start, and 3 is count of ele
console.log('splice: ', removed) // [3, 4, 5]
console.log(' array after splice: ', nums) // [ 1, 2, 6, 7, 8 ]

const nums2 = [1, 2, 3, 4, 5, 6, 7, 8]
const removedPPlusInject = nums2.splice(2, 3, 77, 88) // here 2 is index start, a
console.log('splice: ', removedPPlusInject) // [3, 4, 5]
console.log(' array after splice: ', nums2) // [ 1, 2, 77, 88, 6, 7, 8 ]

// join
const nums3 = [1, 2, 3, 4, 5, 6, 7, 8]
// const together = nums3.join(""); // when join array element together
// const together = nums3.join(" ");
// const together = nums3.join("ami");
const together = nums3.join(',')
console.log(together) // 1,2,3,4,5,6,7,8
```

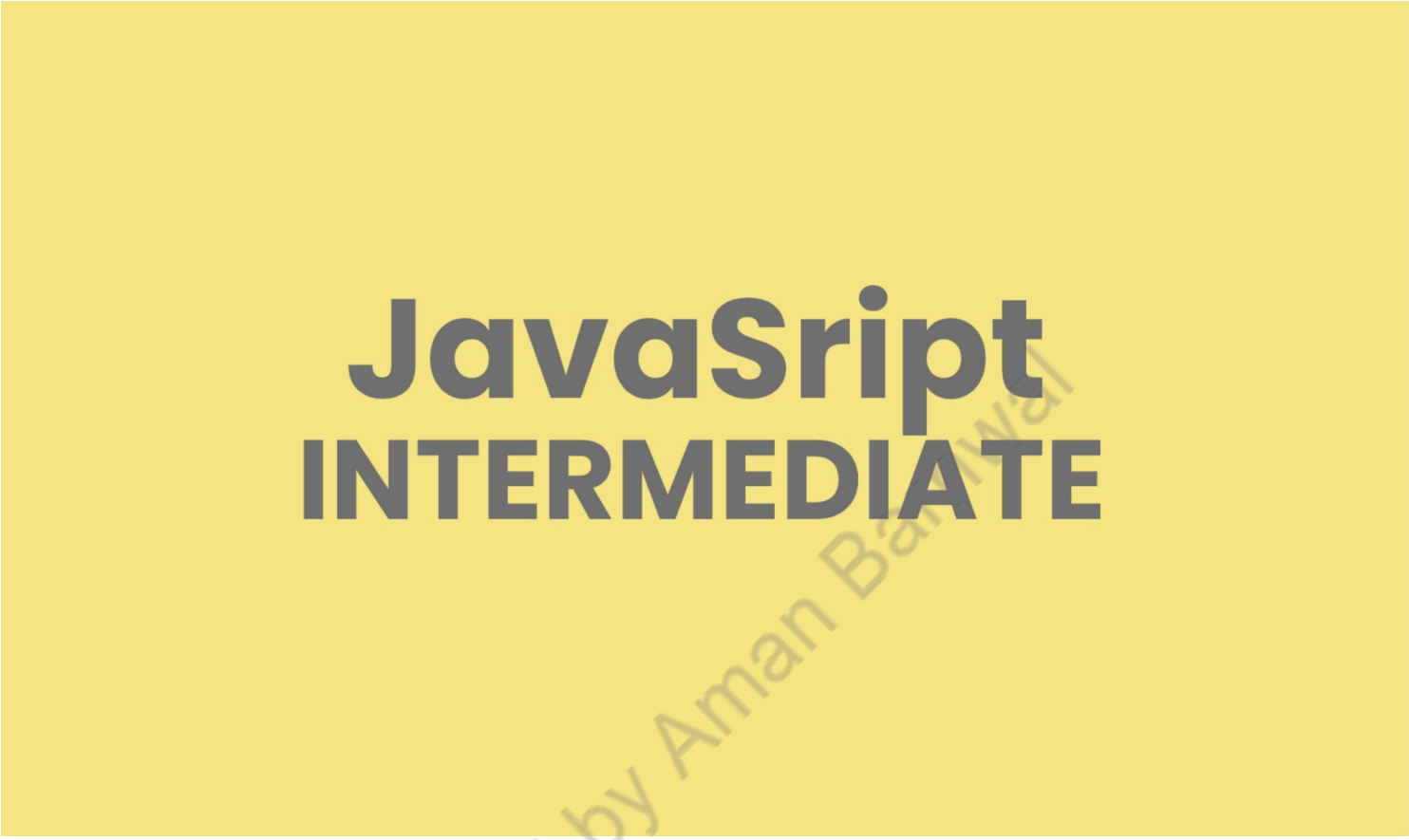

2.9 SUMMARY AND OVERVIEW

JAVASCRIPT

```
const nums = [1, 2, 3, 4, 5, 6, 7, 8, 9]

for (let i = 0; i < nums.length; i++) {
  if (nums[i] > 3) {
    // debugger
    break
  }
  console.log(nums[i]) // 1 2 3
```

```
}

const nums2 = [1, -2, 3, 4, -5, 6, 7, -8, 9]

for (let i = 0; i < nums2.length; i++) {
  if (nums2[i] < 0) {
    // debugger
    continue
  }
  console.log(nums2[i]) // 1 3 4 6 7 9
}
```

JavaScript Intermediate Part- 2



This document covers these topics api, json, json parse, json stringify, load data, get data, display data on UI, http request status code, network tab, send data to server, object method, bind, call, apply, window, global scope, new keyword, class, this keyword, async, await, setTimeout, setInterval, event loop and datetime

1 API JSON, SERVER, DATA LOAD, DYNAMIC WEBSITE, HTTP

1.2 JSON, JSON STRUCTURE, PARSE, STRINGIFY, JSON PROPERTIES

HTML

```
<body>
  <h1>JSON</h1>
  <h2>JavaScript Object Notation</h2>
  <script>
    const user = {
```

```
      // JS object
      id: 245,
      name: 'Masud',
      age: 26,
      sibling: {
        name: 'Riya',
        favoriteFood: 'pizza',
        age: 22,
      },
      friendAge: [25, 23, 26],
      friends: ['Kamal', 'Jamal', 'Riaj'],
    }

    // when sending
    const userJSON = JSON.stringify(user) // convert JS object to JSON
    console.log(userJSON) // {"id":245,"name":"Masud"} // JSON

    // when receiving
    const userFromJSON = JSON.parse(userJSON) // JSON to JS Object
    console.log(userFromJSON) // {id: 245, name: "Masud"}
  </script>
</body>
```


1.3 LOAD DATA, JSON PLACEHOLDER, GET DATA, DISPLAY DATA ON UI

HTML

```
<body>
  <h1>JSON</h1>
  <h2>JavaScript Object Notation</h2>
  <ul id="user-container"></ul>
  <script>
    // https://jsonplaceholder.typicode.com/
    // fetch('https://jsonplaceholder.typicode.com/todos/1')
    //     .then(response => response.json())
    //     .then(json => console.log(json))
    fetch('https://jsonplaceholder.typicode.com/users')
      .then((response) => response.json())
      .then((json) => displayUser(json))
```

```
    function displayUser(users) {
      // console.log("users", users);
      const userNames = users.map((user) => user.username)
      // console.log(userNames);

      const ul = document.getElementById('user-container')
      for (let i = 0; i < userNames.length; i++) {
        const user = userNames[i]
        const li = document.createElement('li')
        li.innerText = user
        ul.appendChild(li)
      }
    }
  </script>
</body>
```

1.4 HTTP REQUEST, STATUS CODE, NETWORK TAB, BAD API

HTML

```
<body>
  <h1>JSON</h1>
  <h2>JavaScript Object Notation</h2>
  <ul id="user-container"></ul>
  <script>
    // https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

    fetch('https://jsonplaceholder.typicode.com/people')
      .then((response) => response.json())
      .then((json) => displayUser(json))
      .catch((error) => console.log(error))

    function displayUser(users) {
      // console.log("users", users);
      const userNames = users.map((user) => user.username)
      // console.log(userNames);

      const ul = document.getElementById('user-container')
```

```
      for (let i = 0; i < userNames.length; i++) {
        const user = userNames[i]
        const li = document.createElement('li')
        li.innerText = user
        ul.appendChild(li)
      }
    }
  </script>
</body>
```

1.5 SEND DATA TO THE SERVER, HTTP POST METHOD, GET VS POST AND SEND DATA TO SERVER, HTTP POST, JSON STRINGIFY

HTML

```
<body>
  <h1>JSON</h1>
  <h2>JavaScript Object Notation</h2>
  <ul id="user-container"></ul>
  <input type="text" name="" id="title" placeholder="title" />
  <br />
  <input
    type="text"
    name=""
    id="body-content"
    placeholder="post main section"
  />
  <br />
  <button id="submit">Submit</button>
  <script>
    //.... const postInfo = {
    //.........  title: 'fooooooo',
    //.........  body: 'barrrrrrr',
    //.........  userId: 1
    //.....  }

    document.getElementById('submit').addEventListener('click', function () {
      const title = document.getElementById('title').value
      const bodyContent = document.getElementById('body-content').value
```

```
    // console.log(title, bodyContent)
    const post = {
      title: title,
      body: bodyContent,
    }
    nowPostToServer(post)
  })

  function nowPostToServer(postInfo) {
    fetch('https://jsonplaceholder.typicode.com/posts', {
      method: 'POST',
      body: JSON.stringify(postInfo),
      headers: {
        'Content-type': 'application/json; charset=UTF-8',
      },
    })
      .then((response) => response.json())
      .then((data) => console.log(data))
      .catch((error) => alert('Please try again later'))
  }

  function displayUser(users) {
    // console.log("users", users);
    const userNames = users.map((user) => user.username)
    // console.log(userNames);

    const ul = document.getElementById('user-container')
    for (let i = 0; i < userNames.length; i++) {
      const user = userNames[i]
      const li = document.createElement('li')
      li.innerText = user
      ul.appendChild(li)
    }
  }
</script>
</body>
```


2 OBJECT MASTERING, INTERVIEW QUESTION

2.1 OBJECT METHOD PROPERTY REVIEW

JAVASCRIPT

```
const person = {
  firstName: 'Rahim',
  lastName: 'Udding',
  salary: 15000,
  getFullName: function () {
    // this can be used to access any property of an object
    console.log(this.firstName, this.lastName)
  },
  chargeBill: function (amount) {
    this.salary = this.salary - amount
    return this.salary
  },
}

console.log(person)
// if access from outside object, if access inside object use this
console.log(person.firstName) // Rahim // access object property

// call function inside a object
person.chargeBill(150)
console.log(person.salary)
```


2.2 OBJECT USE BIND TO BORROW METHOD FROM ANOTHER OBJECT

JAVASCRIPT

```
// BINDING

// object - 1
const normalPerson = {
  firstName: 'Albert',
  lastName: 'Einstein',
  salary: 15000,
  getFullName: function () {
    // this can be used to access any property of an object
```

```
    console.log(this.firstName, this.lastName)
  },
  chargeBill: function (amount) {
    console.log(this)
    this.salary = this.salary - amount
    return this.salary
  },
}

// object - 2
const heroPerson = {
  firstName: 'Blaise',
  lastName: 'Pascal',
  salary: 25000,
}

// object - 3
const friendlyPerson = {
  firstName: 'Enrico',
  lastName: 'Fermi',
  salary: 35000,
}

// normalPerson.chargeBill();

// bind normalPerson with heroPerson
const heroBillCharge = normalPerson.chargeBill.bind(heroPerson)

heroBillCharge(2000)
heroBillCharge(3000)
console.log(heroPerson.salary) // 20000

console.log(normalPerson.salary) // 15000

// bind normalPerson with friendlyPerson
const friendlyChargeBill = normalPerson.chargeBill.bind(friendlyPerson)
friendlyChargeBill(1500)
console.log(friendlyPerson.salary)
```

2.3 DIFFERENCE BETWEEN BIND, CALL AND APPLY

JAVASCRIPT

```
// CALL

// object - 1
const manager = {
  firstName: 'Albert',
  lastName: 'Einstein',
  salary: 80000,
  getFullName: function () {
    // this can be used to access any property of an object
    console.log(this.firstName, this.lastName)
  },
  chargeBill: function (amount, tips, tax) {
    console.log(this)
    this.salary = this.salary - amount - tips - tax
    return this.salary
  },
}

// object - 2
const assistantManager = {
  firstName: 'Blaise',
  lastName: 'Pascal',
  salary: 70000,
}

// object - 3
const employee = {
  firstName: 'Enrico',
  lastName: 'Fermi',
  salary: 35000,
}

manager.chargeBill.call(assistantManager, 900, 100, 10)
manager.chargeBill.call(assistantManager, 3000, 300, 30)
console.log(assistantManager.salary) // 65660
```

```
manager.chargeBill.call(employee, 5000, 500, 50)
console.log(employee.salary) // 29450

// APPLY - in apply array needs to be pass as function arguments
manager.chargeBill.apply(assistantManager, [900, 100, 10])
manager.chargeBill.apply(assistantManager, [3000, 300, 30])
console.log(assistantManager.salary) // 65660

manager.chargeBill.apply(employee, [5000, 500, 50])
console.log(employee.salary) // 29450
```

2.4 WINDOW, GLOBAL VARIABLE, GLOBAL SCOPE

HTML

```
<body>
  <script>
    // Browser ->devloper tool -> console -> type window
    // document === window.document // true
    // console === window.console // true
    // document.getElementById

    var name = 'Alex'

    function add(num1, num2) {
      var result = num1 + num2
      // result = num1 + num2; // without var the variable will be implicitly glo
      // result = num1 + num2;
      // console.log('result inside', result); // local variable
      console.log('name inside', name)

      function double(num) {
        return num * 2
      }
      var total = double(result)
      return total
    }

    // console.log('result outside', result); // throw error because result is no
```

```
    console.log('name outside', name)
    // console.log(result); // if result set as implicitly global then it can't b
    var sum = add(13, 21)
    console.log(sum)
    console.log(result)
    // console.log(result);

    // browser console -> name -> will display because it's global
    // browser console -> double -> throw error because it's local
    // browser console -> add -> will display because it's global
  </script>
</body>
```

2.5 NEW KEYWORD, CLASS AND OBJECT DIFFERENCE

JAVASCRIPT

```
class Person {
  constructor(firstName, lastName, salary) {
    this.firstName = firstName
    this.lastName = lastName
    this.salary = salary
  }
}

// new keyword is used to create object from class
const heroPerson = new Person('Blaise', 'Pascal', 25000)
console.log(heroPerson)
// Person { firstName: 'Blaise', lastName: 'Pascal', salary: 25000 }

const friendlyPerson = new Person('Enrico', 'Fermi', 35000)
console.log(friendlyPerson)
// Person { firstName: 'Enrico', lastName: 'Fermi', salary: 35000 }

// before ES6 function is used to create class, function name start with capital
function Person2(firstName, lastName, salary) {
  this.firstName = firstName
  this.lastName = lastName
```

```
  this.salary = salary
}

const oldPerson = new Person2('Grand', 'Papa', 12000)
console.log(oldPerson)
// Person2 { firstName: 'Grand', lastName: 'Papa', salary: 12000 }
```

2.6 HOW TO UNDERSTAND THE THIS KEYWORD

HTML

```
<body>
  <h1>This is not confused anymore</h1>
  <p onclick="this.innerText = 'Thank you for clicking me'">
    Click for surprise
  </p>

  <script>
    const myObj = {
      name: 'Niels Bohr',
      getFullName: function () {
        console.log(this)
        return 'Mr. ' + this.name
      },
    }

    // myObj.getFullName();

    const anotherObj = {
      name: 'Sarah Boysen',
    }

    anotherObj.getFullName = myObj.getFullName
    console.log(anotherObj.getFullName)
    myObj.getFullName() // left of .function() is the object this will refer
    anotherObj.getFullName() // left of .function() is the object this will refer

    function add(a, b) {
```

```
    console.log(this) // this will return window object
    return a + b
  }

  add(12, 13) // nothing before function(), means this will refer window

  anotherObj.sum = add
  anotherObj.sum()

  setTimeout(function () {
    console.log(this)
  }, 1000)
</script>
</body>
```

2.7 ASYNC AWAIT HOW TO USE IT FOR ASYNC CALL

HTML

```
<body>
  <ul id="myList"></ul>

  <script>
    // 26.8 ASYNC AWAIT HOW TO USE IT FOR ASYNC CALL
    /* async function greet(name) {
        return "Hello " + name;
    }

    const greeting = greet("Mofiz");
    console.log(greeting); // will return promise
    // access using then
    greeting.then(res => console.log(res));

    // arrow function with async
    // const abc = async () =>
    */

    async function loadData() {
```

```
    const response = await fetch('https://jsonplaceholder.typicode.com/users')
    const data = await response.json()
    // console.log(data);
    displayData(data)
    return data
  }

  loadData()

  // function loadData() {
  //     fetch('https://jsonplaceholder.typicode.com/users')
  //         .then(response => response.json())
  //         .then(data => {
  //             displayData(data);
  //         })
  // }
  // loadData();

  function displayData(data) {
    // console.log(data);
    const parentNode = document.getElementById('myList')
    for (let i = 0; i < data.length; i++) {
      const user = data[i]
      const item = document.createElement('li')

      item.innerText = user.name
      parentNode.append(item)
    }
  }
</script>
</body>
```


2.8 ASYNCHRONOUS JAVASCRIPT SETTIMEOUT, SETINTERVAL

JAVASCRIPT

```
function doSomething() {
  console.log(3333)
```

```
}

console.log(2222)
// doSomething();
// SetTimeOut run only once
setTimeout(doSomething, 3000)

setTimeout(function () {
  console.log('Lazy and waiting')
}, 5000)

// arrow function
setTimeout(() => {
  console.log('Lazy and waiting....')
}, 5000)

console.log(4444)

// setInterval run repeatedly
setInterval(() => {
  console.log('Set interval')
}, 1000)
```

2.10 JAVASCRIPT DATETIME TIMEZONE AND OTHERS

JAVASCRIPT

```
// browser console
let start = new Date()
let sum = 0
for (let i = 0; i < 100000000; i++) {
  sum++
}
let end = new Date()
console.log('Time elapsed', end - start, sum)
```