|  |  |  |  |
| --- | --- | --- | --- |
| SL NO | KEYWORDS | DESCRIPTION | EXAMPLE |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 | Data Types | * + **primetive and object data type, Primetive data type are defined by javascript**, **primetive (number, string, boolean, undefined, null), object data type (array, object, function)** |  |
| 10 |  |  |  |
| 11 |  |  |  |
| 12 |  |  |  |
| 13 | Null & Undefined |  |  |
| 14 |  |  |  |
| 15 |  |  |  |
| 16 |  |  |  |
| 17 |  |  |  |
| 18 |  |  |  |
| 19 |  |  |  |
| 20 |  |  |  |
| 21 |  |  |  |
| 22 |  |  |  |
| 23 |  |  |  |
| 24 |  |  |  |
| 25 |  |  |  |
| 26 |  |  |  |
| 27 |  |  |  |
| 28 |  |  |  |
| 29 |  |  |  |
| 30 |  |  |  |
| 31 |  |  |  |
| 32 |  |  |  |
| 33 |  |  |  |
| 34 |  |  |  |
| 35 |  |  |  |
| 36 |  |  |  |
| 37 |  |  |  |
| 38 |  |  |  |
| 39 |  |  |  |
| 40 |  |  |  |
| 41 |  |  |  |
| 42 |  |  |  |
| 43 |  |  |  |
| 44 |  |  |  |
| 45 |  | Array এর শেষ ইনডেক্স হচ্ছে array এর লেন্থ এর চেয়ে ১ কম অর্থাত arr[array.length-1] | var arr = [4, 5, 1, 6, 8, 9, 40, 50]  var sum = 0  for (var i = 0; i < arr.length; i++) {  sum += arr[i]  }  console.log(sum)  for (var i = 0; i < arr.length; i++) {  if (arr[i] % 2 === 0) {  console.log(arr[i])  } } |
| 46 |  |  | var arr = [1, 2, 3, 4, 5, 6, 7, 8]  // Insert 9 to index 3  // arr**[3] = 9 এইভাবে ৩ নম্বর ইনডেক্সে ভেলু ৪ এর পরিবর্তে ৯ বসবে।**  // **arr.push(9) পুশ মেথডের মাধ্যমে ভেলূ এ্যারের শেষে গিয়ে জমা হবে।**  // **arr.unshift(9) আনশিফট মেথডের মাধ্যমে ভেলু এ্যারের প্রথমে গিয়ে জমা হবে।**  // arr.splice(3, 0, 9, 10) এখানে ৩ মানে ৩ ইনডেক্স এর এর জায়গায় ৯ ও ১০ বসবে, ০ মানে কোন ইনডেক্স মুছবে না, স্প্লাইস মেথডের মাধ্যমে এ্যারের এলিমেন্ট মুছা যায়, আপডেট করা যায়, নতুন এলিমেন্ট যোগ করা যায়। এটি খুব গুরুত্বপূর্ণ।  // **arr[3] = undefined এভাবে ৩ ইনডেক্সটি আনডিফাইন্ড হবে।**  // **arr.pop() পপ মেথডের মাধ্যমে এ্যারের শেষ ইনডেক্সটি মুছে যাবে।**  **// arr.shift() শিফট মেথডের মাধ্যমে এ্যারের প্রথম হতে ইনডেক্স মুছে যাবে।**  // arr.splice(3, 1) এখানে ৩ মানে ৩ ইনডেক্সটি মুছবে ১ মানে ১টি ইনডেক্স মুছবে।  arr.splice(3, 1, 44) এখানে ৩ মানে ৩ ইনডেক্সটি মুছবে, ১ মানে ১টি ইনডেক্স মুছবে ৪৪ মানে ৩ ইনডেক্স এর জায়গায় ৪৪ বসবে। |
| 47 | Search Data from an Array |  | var arr = [4, 7, 2, 6, 8, 4, 55, 9, 89, 27, 55]  var find = 55  var isFound = false  for (var i = 0; i < arr.length; i++) {  if (arr[i] === find) {  console.log('Data Found at Index ' + i)  isFound = true  break;  }  }  if (!isFound) {  console.log('Data Not Found')  } |
| 48 | Multidimensional Array |  | var arr = [  [78, 71, 80, 90],  [80, 75, 90, 86],  [71, 75, 79, 80]  ]  // console.log(arr[0][0])  // console.log(arr[1][1])  **for (var i = 0; i < arr.length; i++) {**    **for (var j = 0; j < arr[i].length; j++) {**  **console.log('Element ' + i + ': ' + arr[i][j])**  **}**  **}** |
| 49 | * + Reverse Array Elements Completely |  | var arr = [1, 2, 3, 4, 5, 6, 7, 8]  for (var i = 0; i < (arr.length / 2); i++) {  var temp = arr[i]  arr[i] = arr[arr.length - 1 - i]  arr[arr.length - 1 - i] = temp  } এভাবে এ্যারে কে রিভার্জ করা যায়  console.log(arr.reverse()) reverse মেথডের মাধ্যমেও রিভার্জ করা যায় |
| **50** | **Array Methods** |  | var arr = [4, 5, 9, 6]  var arr2 = [5, 10, 45, 23]  console.log(arr.join(' | ')) এভাবে এ্যারের এলিমেন্টগুলোকে , বা যেকোন সিম্বল দিয়ে  সেপারেট করা যায়।  arr.fill(true) fill মেথডের মাধ্যমে এ্যারের এলিমেন্টগুলোকে কোন সংখ্যা বা  কোন সিম্বল বা টেক্স দিয়ে ফিল করা যায়।  var arr3 = arr.concat(arr2) এভাবে একাধিক এ্যারে কে কনকেটানেট করা যায়  console.log(arr3)  console.log(Array.isArray(arr)) Array.isArray(arr)) এর মাধ্যমে এ্যারে কে  যাচাই করা যায়।এ্যারে হলে ট্রু রিটার্ণ করে না হলে ফলস রিটার্ণ করে।  // var arr2 = Array.from(arr) ডপ্লিকেট নতুন এ্যারে তৈরী করে (7.48মিনিট)  // console.log(arr2)  // var arr2 = arr  var a = [1, 2]  var b = Array.from(a)  b[0] = 5  console.log(a)  console.log(b) |
| 51 |  |  |  |
| 52 | Object Literal vs Constructor | অবজেক্ট কে ২ভাবে ডিকলার করা যায়। লিটারাল উপায়ে এবং কনসট্রাক্টর উপায়ে। | var point = { x: 10, y: 20, z: 15 } অবজেক্ট লিটারাল  point['y'] = 100 এখানে y কি এর ভেলূ ১০০ দিয়ে রিপ্লেস হয়ে যাবে। |
| 53 |  | অবজেক্টের প্রপারটি ২ভাবে এক্সেস নেওয়া যায়। **ডট নোটেশন** ও **এ্যারে নোটেশন।**  ডট নোটেশন ও এ্যারে নোটেশন ব্যবহার করে অবজেক্টের বাহির হতে প্রপারটি ডিকলার করা যায় ও আপডেট করা যায়। | var point = { x: 10, y: 20, z: 15 }  console.log(**point.x**) ডট নোটেশন  console.log(**point['x']**) এ্যারে নোটেশন  point.a = 16 ডট নোটেশন ডিকলারেশন  point['C'] = 100 এ্যারে নোটেশন ডিকলারেশন |
| 54 |  |  |  |
| 55 | Remove Object Properties | Delete কিওয়ার্ড ব্যবহার করে অবজেক্ট এর কি ও ভেলু মুছা যায় | var point = { x: 10, y: 20, z: 15 }  point.a = 16 /নতুন ১টি কি ও ভেলু এসাইন করা হয়েছে  **delete point.a** এভাবে কোন অবজেক্টের কি ভেলূ মুছা যায়  console.log(point) |
| 56 | Compare Two Objects |  | var obj1 = { a: 10, b: 20 }  var obj2 = { a: 10, b: 2 }    console.log(JSON.stringify(obj1) === JSON.stringify(obj2))  এখানে JSON.stringify মেথড দিয়ে অবজেক্টকে ট্রিং এ রুপান্তর করা হয়েছে।এভাবে ২টি অবজেক্টের মধ্যে তুলনা করা যায়।২টি অবজেক্ট এক হলে সত্য রিটার্ণ করবে না হয় মিথ্যা |
| 57 | **Iterate Object Properties** |  | var obj = { x: 40, y: 60, z: 75 }  // console.log('x' in obj) এখানে true রিটার্ণ করবে  // console.log('p' in obj) এখানে false রিটার্ণ করবে    **for (var i in obj) {**  **console.log(i)**  **console.log(i + ': ' + obj[i]) এভাবে অবজেক্টের কি ও ভেলু ইটারেট করা যায়**  **}** |
| 58 | Object Methods |  | var obj = { x: 40, y: 60, z: 75 }  console.log(Object.keys(obj)) Object.keys মেথড অবজেক্ট এর কিগুলো নিয়ে আসে console.log(Object.values(obj)) Object. values মেথড অবজেক্ট এর ভেলূ নিয়ে আসে  console.log(Object.entries(obj))  // **var obj2 = obj এভাবে ১টি অবজেক্ট হতে আর ১টি অবজেক্ট তৈরী করলে ১টি কে পরিবর্তন করলে অন্যটিও পরিবর্তন হয়ে যায়।**  // obj2.x = 70  // obj2.y = 100  **var obj2 = Object.assign({}, obj) obj অবজেক্ট হতে obj2 নামে নতুন অবজেক্ট তৈরী করা হয়েছে।**  **obj2.x = 100 এখন শুধু obj2 তে পরিবর্তন আসবে।** |
| 59 |  |  |  |
| 60 | How to Define a Function | জাভাস্ক্রিপ্টে ৩ ভাবে ফাংশন ডিফাইন করা যায় |  |
| 61 | How to Invoke a Function | Invoke মানে হল ফাংশন ব্যবহার করা, কল করা |  |
| 62 | Arguments and Parameter |  |  |
| 63 | Argument Object in Javascript Function |  | Repeat again |
| 64 | How to Return Anything from a Function |  |  |
| 65 |  |  |  |
| 66 | Inner Functions |  | function something(greet, name) {  function getFirstName() {  if (name) {  return name.split(' ')[0]  } else {  return ' '  }  }  var message = greet + ' ' + getFirstName()  console.log(message)  }  something('Good Morning', 'Shegufa Taranjum') // Good Morning Shegufa |
| 67 | Function Scoping |  |  |
| 68 | Functional |  |  |
| 69 | Pure Function | // What is Pure Function?  // `\* It Returns the same result if given the same arguments`  // `\* It does not cause any observable side effects` | function sqr(n) {  return n\*n  }  console.log(sqr(3))  var n = 10  function change() {  n = 100  }  change()  console.log(n)  var point = {  x: 45,  y: 30  }  function printPoint(point) {  point.x = 100  point.y = 200  console.log(point)  }  printPoint(point)  console.log(point) |
| 70 | First Class Functions |  | // First Class Function  function add(a, b) {  return a+b  }  // 1. A Function can be Stored in a Variable  var sum = add  console.log(sum(4, 5))  console.log(typeof sum)  // 2. A Function can be Stored in an Array  var arr = []  arr.push(add)  console.log(arr)  console.log(arr[0](5, 3))  // 3. A Function can be Stored in an Object  var obj = {  sum: add  }  console.log(obj)  console.log(obj.sum(7, 9))  // 4. We can Create Function as Need  setTimeout(function () {  console.log('I have created...')  }, 100)  // 5. We can Pass Function as an Arguments  // 6. We can return Functions from Another Function |
| 71 | Higher Order Functions |  | Do again |
| 72 | Javascript Closure |  |  |
| 73 | Callback Function |  | function sample(a, b, cb) {  var c = a + b  var d = a - b  var result = cb(c, d)  return result  }  function sum(a, b) {  return a + b  }  var result = sample(5, 8, sum)  console.log(result)  var result2 = sample(5, 8, function (c, d) {  return c - d  })  console.log(result2)  var result3 = sample(5, 8, function (c, d) {  return c \* d  })  console.log(result3) |
| 74 | of ForEach Function |  | var arr = [1, 2, 3, 4, 5]  var sum = 0  arr.forEach(function (value, index, arr) {  //console.log(value, index, arr)  sum += value  })  console.log(sum)  function forEach(arr, cb) {  for (var i = 0; i < arr.length; i++) {  cb(arr[i], i, arr)  }  }  var sum = 0  forEach(arr, function (value, index, arr) {  console.log(value, index, arr)  sum += value  })  console.log(sum)  forEach(arr, function (value, index, arr) {  arr[index] = value + 5  })  console.log(arr) |
| 75 | Map Function |  |  |
| 76 | Filter Function |  | var arr = [4, 8, 1, 3, 5, 6, 4, 3, 9]  var filteredArr = arr.filter(function (value) {  return value > 4  })  console.log(filteredArr)  function myFilter(arr, cb) {  var newArr = []  for (var i = 0; i < arr.length; i++) {  if (cb(arr[i], i, arr)) {  newArr.push(arr[i])  }  }  return newArr  }  console.log(myFilter(arr, function (value) {  return value % 2 === 1  }))  console.log(myFilter(arr, function (value) {  return value > 4  })) |
| 77 | Reduce Function |  |  |
| 78 | Find and Find Index Function |  | var arr = [7, 4, 8, 6, 9, 2, 1, 70, 3]  var result = arr.find(function (value) {  return value === 9  })  console.log(result)  var result = arr.findIndex(function (value) {  return value === 9  })  console.log(result)  function myFind(arr, cb) {  for (var i = 0; i < arr.length; i++) {  if (cb(arr[i])) {  // return arr[i]  return i  }  }  }  var result = myFind(arr, function(value) {  return value === 9  })  console.log(result) |
| 79 | Sort, Some and Every Function | Do again and again | var persons = [{ name: 'A', age: 24 },  { name: 'B', age: 19 },  { name: 'C', age: 26 },  { name: 'D', age: 21 } ]  var arr = [4, 8, 1, 6, 7, 9, 0, 4, 3, 5, 6, 8, 2, 1, 7, 1]  arr.sort(function (a, b) {  if (a > b) {  return 1  } else if (a < b) {  return -1  } else {  return 0  }  })  console.log(arr)  persons.sort(function (a, b) {  if (a.age > b.age) {  return 1  } else if (a.age < b.age) {  return -1  } else {  return 0  }  })  console.log(persons)  var res1 = arr.every(function (value) {  return value % 2 === 0  })  console.log(res1)  var res2 = arr.every(function (value) {  return value >= 0  })  console.log(res2)  var res3 = arr.some(function (value) {  return value % 2 === 1  })  console.log(res3)  var res4 = arr.some(function (value) {  return value < 0  })  console.log(res4) |
| 80 | Return a Function from Another Function |  |  |
| 81 | Recursive Function | নিজেই নিজেকে কল করে। |  |
| 82 | What is Currying in Javascript |  |  |
| 83 | Function Composition |  |  |
| 84 |  |  |  |
| 85 |  |  |  |
| 86 |  |  |  |
| 87 |  |  |  |
| 88 |  |  |  |
| 89 |  |  |  |
| 90 |  |  |  |
| 91 |  |  |  |
| 92 |  |  |  |
| 93 |  |  |  |
| 94 |  |  |  |
| 95 |  |  |  |
| 96 |  |  |  |
| 97 |  |  |  |
| 98 |  |  |  |
| 99 |  |  |  |
| 100 |  |  |  |
| 101 |  |  |  |
| 102 | Four Main Pillars of OOP |  |  |
| 103 | Abstraction |  |  |
| 104 | Encapsulation |  |  |
| 105 | Inheritance |  |  |
| 106 | Polymorphism |  |  |
| 107 |  |  |  |
| 108 | Object Literal with Methods |  |  |
| 109 |  |  |  |
| 110 | Class Using Factory pattern |  |  |
| 111 | Class Using Constructor Pattern |  |  |
| 112 |  |  |  |
| 113 | Factory Pattern vs Constructor Pattern |  |  |
| 114 | Constructor Property of a Constructor Function |  |  |
| 115 |  |  |  |
| 116 | Bind, Call, Apply: Call Your Function with Your Custom "This" Value |  |  |
| 117 | Pass By Value vs Pass by Reference - Mutable vs Immutable |  |  |
| 118 | Abstraction |  |  |
| 119 | Hide Private Properties |  |  |
| 120 | Use Getter Setter |  |  |
| 121 | Inheritance |  |  |
| 122 | Prototype and Prototypical Inheritance in Javascript (getPrototypeOf(),prototype, getOwnPropertyDescriptor(),defineProperty(), Property Descriptor ) | জাভাস্ক্রিপ্টে Prototype হচেছ প্যারেন্ট ক্লাশ। প্যারেন্ট ক্লাশ ব্যতিত আর কিছুই নয়।জাভাস্ক্রিপ্টে প্রতিটি অবজেক্টের প্যারেন্ট ক্লাশ আছে এবং ঐ প্যারেন্ট ক্লাশই হলো Prototype। |  |
| 123 | Multi Level Inheritance |  |  |
| 124 | Property Descriptor |  | var person = {  name: 'Mohammed Ibrahim'  }  Object.defineProperty(person, 'name', {  enumerable: false, // enumerable যদি false থাকে পার্সন অবজেক্টের name এর ভেলু ইটারেট করা যাবে না ।  writable: false, // writable যদি false থাকে পার্সন অবজেক্টের name এর ভেলু নতুন করে আর এসাইন করা যাবে না।  configurable: false configurable যদি false থাকে পার্সন অবজেক্ট কে কনফিগার করা যাবে না বা delete কিওয়ার্ড ব্যবহার করে মুছা যাবে না।  }) |
| 125 | Constructor Prototype Object of Any Constructor Function | কোন অবজেক্টের এর প্রটুটাইপ দেখতে চাইলে Object.getPrototypeOf ব্যবহার করে সহজে দেখা যায় | Function Person(name){  This.name = name  }  Var p1 = new Person('Md Ibrahim')  Console.log(Object.getPrototypeOf(p1)) //এভাবে p1 অবজেক্ট এর প্রটুটাইপ দেখা যাবে।  Console.log(Person.prototype) /এভাবেও প্রটুটাইপ দেখা যায়। |
| 126 | Instance vs Prototype Members |  | function Shape(color) {  this.color = color  }  Shape.prototype.common = function () {  console.log('I am Common Method')  } |
| 127 |  |  |  |
| 128 |  |  |  |
| 129 |  |  |  |
| 130 | Our First Own Prototypical Inheritance |  |  |
| 131 | Reset Constructor Property After Inheritance |  |  |
| 132 | Calling super with Call method |  |  |
| 133 | Creating Extends Function to Reduce Repeating Code |  |  |
| 134 | Method Overriding in Javascript |  |  |
| 135 | Polymorphism in Javascript |  |  |
| 136 | When to Use Inheritance |  |  |
| 137 | Inheritance and Composition Mixing Together |  |  |
| 138 | What is ES6 |  |  |
| 139 | Setup Your Environment for ES6 | Git repository হতে wevpack কোলন করা, nmp install কমান্ড রান করা, npm audit fix কমান্ড রান করা, npm start কমান্ড রান করা |  |
| 140 | How to Use Template String |  | var age = 23  var name = 'Ibrahim'  console.log('My name is ' + name + ' and I am ' + age + ' years old') // es5  console.log(`My name is ${name} and I am ${age} years old. I am ${age < 18 ? 'not ' : ''}adult`) //es6 ‍age এর মান ১৮ এর কম হলে not বসবে।  console.log(name.padStart(15, '\*')) //name এর লেন্থ ১৫ কেরেক্টর না হওয়া পর্যন্ত Ibrahim এর আসে \* বসবে  console.log(name.padEnd(15, 'a')) / name এর লেন্থ ১৫ কেরেক্টর না হওয়া পর্যন্ত Ibrahim এর পরে a বসবে  console.log('S'.repeat(10)) / S কেরেক্টরটি ১০ বার রিপিট হবে |
| 141 | Let vs Const vs Var: What is the Difference |  | var vs let vs const  if (true) {  let a = 10 // ‍a কে let কিওয়ার্ড দ্বারা ডিক্লার করার কারণে এটি {} ব্যাকেট এর বাইরে থেকে এক্সেস করা যাবে না। var কিওয়ার্ড দ্বারা ডিক্লার করলে এক্সেসেবল।  }  console.log(a) // এখানে error দেখাবে  for (let i = 0; i < 5; i++) {  }  console.log(i) // এখানে error দেখাবে  {  let iAmLet = 'I am Let'  }  console.log(iAmLet) // // এখানে error দেখাবে |
| 142 | New Arrow Function Syntax in ES6 |  | function add(a, b) { //es5  return a + b  }  let add = (a, b) => {  return a + b  }  // es6 এ {} এর মধ্যে সিংগল লাইন রিটার্ণ করলে return কিওয়া্র্ড ও {} ব্যাকেট ব্যবহার করার দরকার নেই। যেমন নিচের ফাংশনটি একই কাজ করবে।    let add = (a, b) => a + b  console.log(add(45, 50))  let sqr = x => x \* x // প্যারামিটার ১টি থাকলে () ব্যাকেট ব্যবহার করার দরকার নেই। |
| 143 | Arrow Functions and This: What is the Benefit | Do again and again |  |
| 144 | How to Use Default Parameter in ES6 | Default প্যারামিটার হিসেবে কখনো null পাঠানো যাবে না। আবার দেখুন। | function greet(name = 'HM Nayem', msg = 'Hello') {  // console.log(name.length)  // console.log(`${msg}! ${name}`)  // }  // greet(null, 'Hello') |
| 145 | Rest and Spread Operator in ES6 | Do again | function sum(...rest) { //৩টি ডট ফাংশনে ব্যবহার করলে এটিকে বলে রেস্ট অপারেটর।  return rest.reduce((a,b) => a + b)  }  //৩টি ডট এ্যারে বা অবজেক্ট এ ব্যবহার করলে এটিকে বলে Spread অপারেটর।  console.log(sum(1, 2, 3 , 4 , 5))  let a = [1, 2, 3]  console.log(...a) // Spread অপারেটর ব্যবহার করা হয় এ্যারের উপাদানগুলোকে ছড়িয়ে দেওয়ার জন্য  let obj = {  a: 10,  b: 20,  c: 30  }    let obj2 = {  ...obj  } |
| 146 | Enhance Object in ES6 |  | let a = 10, b = 20  let obj = {  a: a,  b: b,  print: function() {  console.log(this)  }  }  //es6 অবজেক্টের কি ও ভেলূ যদি একই হয় তাহলে a: a এভাবে দেওয়ার দরকার নেই। ‍a এভাবে দিলেই চলবে এবং অবজেক্ট এর মধ্যে : ও function কিওয়ার্ড ব্যবহার করার দরকার নেই।  let obj = {  a,  b,  print() {  console.log(this)  }  }  obj.print()  পরবর্তীতে দেখবো 06 Enhanced object literals by [Code For Food](https://www.youtube.com/channel/UC3PPNB_VsG7-NO42mgdrbbA) |
| 147 | **Destructuring in ES6** |  | var person = {  name: 'HM Nayem',  email: 'hasan.m.nayem@gmail.com',  address: {  city: 'Dhaka',  country: 'Bangladesh'  }  }  let { name, email, address: {city, country} } = person  console.log(name, email, city, country)  let arr = [1, 2, 3, 4, 5]  let [first, second, , ,last] = arr  console.log(first, second, last) |
| 148 | Object fromEntries Method in ES6 | Object.entries()  Object.fromEntries() | var obj = { a: 10, b: 20 }  console.log(Object.entries(obj)) // Object.entries উপরের obj অবজেক্টটি ১টি এ্যারেতে রুপান্তর হবে।  var objArr = [  ['a', 10],  ['b', 20]  ]  console.log(Object.fromEntries(objArr)) // Object. fromEntries উপরের objArr এ্যারেটি ১টি অবজেক্ট এ রুপান্তর হবে। |
| 149 | What are Symbols in ES6 |  |  |
| 150 | New Environment Setup with Webpack 4 + babel 7 |  | Npm init, npm i -D webpack webpack-cli webpack-dev-server, (-D for devdepandancy, webpack-sev-server for a server), npm i -D @babel/core @babel/cli @babel/node @babel/polyfill @babel/preset-env @babel/register babel-loader babel-register html-webpack-plugin, (@babel/core for babel main library, @babel/cli for cli interface, @babel/node for node interface, @babel/preset-env এখানেই বলা থাকে বেবেল কিভাবে কাজ করবে, @babel/register এফিসারটির কাজ হচ্ছে ইএস৬ এর সাথে নোডের সাথে সঠিকভাবে কাজ করানো, ), index.html ফাইল তৈরী, src ডাইরেক্টরী তৈরী, .babelrc ফাইল তৈরী, webpack.config.js ফাইল তৈরী, package.json ফাইলে "start": "webpack-dev-server" লাইন সংযুক্ত করণ, |
| 151 | What is Iterator: Iterator Basic Idea |  |  |
| 152 | Create Iterator with ES6 Symbol.Iterator Function |  |  |
| 153 | How does For of Loop Work |  | const arr = [10, 22, 35]  for (let v of arr) {  console.log(v)  }  for (let v of 'HM Nayem') {  console.log(v)  }  let obj = {  a: 10,  b: 20  }  for (let v of obj) {  console.log(v) // ইরর দেখাবে  }  **For of loop সরাসরি ভেলু নিয়ে কাজ করে। অবজেক্ট এর উপর ফর অব লোপ কাজ করে না। আমরা ইচ্ছা করলেই অবজেক্ট এর সাথে Symbol.Iterator জুড়ে দিয়ে কোন অবজেক্টকে ইটারেবল বানাতে পারি।** |
| 154 | What is Generator Functions in ES6 |  | **Function কিওয়ার্ড এর পর \* সাইন বসালে তাকে জেনারেটর বলে** |
| 155 | How to Create Custom Iterable Object |  |  |
| 156 | Understanding Sets in ES6 |  | // let set = new Set([1, 2, 3])  // let set = new Set([1, 2, 3])  // set.add(5)  // set.add(6)  // set.add(1)  // set.add(2)  // console.log(set)  // console.log(set.size)  // console.log(set.has(5))  // console.log(set)  // console.log(set.keys())  // console.log(set.values())  // let keyIterate = set.values()  // console.log(keyIterate.next())  // console.log(keyIterate.next()) |
| 157 | Understanding Map in ES6 |  | Set এর মতো map কেও আমরা ২ভাবে এসাইন করতে পারি।  // let map = new Map([  // ['a', 10],  // ['b', 20],  // ['c', 30]  // ])  // map.set('d', 40) // map এর মধ্যে নতুন কি ভেলু এড করা হয়েছে।  // map.set({name: 'HM Nayem'}, 45)  // map.delete('c') // map এর কোন কি ভেলূ মুছার জন্য শুধু কি প্রভাইড করলেই  হয়।  // console.log(map.size) // এভাবে map এর সাইজ দেখা যাবে।  // console.log(map)  // console.log(map.get('b')) //map এর কোন কি ভেলূ পাওয়ার জন্য  // console.log(map.values()) // শুধু ভেলূগুলো নিয়ে আসবে।  // console.log(map.keys()) // শুধু কিগুলো নিয়ে আসবে।  // console.log(map.entries()) //কি ভেলূ আকারে নিয়ে আসবে  // for (let [k, v] of map) { //প্রথমটি কি, ২য়টি ভেলু  // console.log(v, k)  // }  // map.forEach((v, k) => { //প্যারামিটার হিসেবে প্রথমটি ভেলূ, ২য়টি কি  // console.log(k, v)  // }) |
| 158 | What is Weak Set |  | done |
| 159 | What is Weak Map |  | done |
| 160 | New Class Syntax in ES6 |  | Done, do again |
| 161 | ES6 Class Properties |  |  |
| 162 | ES6 Static Method |  |  |
| 163 | "this" Property and It's Value in ES6 Class |  |  |
| 164 | Hide Private Data with Symbol |  |  |
| 165 | Hide Private Properties with Weak Map |  |  |
| 166 | ES6 Getter and Setter |  |  |
| 167 | Inheritance in ES6 |  |  |
| 168 | Method Overriding in ES6 |  |  |
| 169 | ES6 Module System Explained |  |  |
| 170 | Errors in Javascript: How to Handle |  | Done |
| 171 | Error Handling with If Else Condition Check [Logical Errors] |  | Done |
| 172 | Error Handling with Try Catch Block [Runtime Errors] |  | Done |
| 173 | How to Throw an Error in Javascript |  | done |
| 174 | Optional Finally Block in Javascript |  | Done |
| 175 | Optional Catch Binding in ESNext |  | Done |
| 176 | How to Create a Custom Error in ES6 |  | Done |
| 177 | Understanding Asynchronous Programming in Javascript |  | Done |
| 178 | Asynchronous Programming is Action (Simulate Using Java & Javascript) |  | Done |
| 179 | How to Store Result Returned from Asynchronous Function |  | মনে রাখা গুরুত্বপূর্ণ  Asynchronous Function এর রেজাল্ট কখনো কোন ভেরিয়েবলে ষ্টোর করা যায় না। function sayMyName(name) {  let result  setTimeout(() => {  console.log('I have done...')  console.log(name)  }, 3000)  return result  }  let output = sayMyName('HM Nayem')  console.log(output) **// এটি আনডিফাইন দেখাবে**  **আবার দেখুন।** |
| 180 | Why does Javascript Act Like Asynchronous |  | Javascript is a single threaded language thats mean javascript process one task at a time  জাভাস্ক্রিপ্ট একটা সময়ে মাত্র একটা টাক্স কম্প্লিট করে থাকে। javascript uses two data structure to handle variables and function calls.জাভাস্কিপ্ট দুই ধরণের ডাটা টাক্সার ব্যবহার করে থাকে।১টি ভেরিয়েবল হেন্ডেল করার জন্য আর ১টি ফাংশন হেন্ডেল করার জন্য।    Java is a multi threaded language |
| 181 | How does Javascript Handle Asynchronous Programming |  | Done |
| 182 | NodeJS vs Browser: Does Javascript Act Same [V8 Engine] |  | জাভাস্ক্রিপ্ট বা নোড জেএস যে ইঞ্জিনটি ব্যবহার করছে তা গোগলের প্রভাইড করা V8 Engine  What is v8 Engine ?  V8 is Google's open source high-performance JavaScript and WebAssembly engine, written in C++ |
| 183 | How to Handle Asynchronous Operations in Javascript |  | Async Iterator, For Await Of Loop, Async Generator এখানে কোন আলোচনা নেই। পরবর্তী চাপটারে আলোচনা হবে। |
| 184 | What is Ajax (Asynchronous Javascript and XML) |  | <https://jsonplaceholder.typicode.com/>  AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. |
| 185 | How to Handle XMLHttpRequest Using Callback |  | // const xhr = new XMLHttpRequest()  // xhr.open('get', 'https://jsonplaceholder.typicode.com/users')  // xhr.onreadystatechange = function (e) {  // if (xhr.readyState === 4) {  // if (xhr.status === 200) {  // let users = JSON.parse(xhr.response)  // console.log(users)  // } else {  // console.log(xhr.status)  // }  // }  // }  // xhr.send()  function getRequest(url, callback) {  const xhr = new XMLHttpRequest()  xhr.open('get', url)  xhr.onreadystatechange = function (e) {  if (xhr.readyState === 4) {  if (xhr.status === 200) {  let response = JSON.parse(xhr.response)  callback(null, response)  } else {  callback(xhr.status, null)  }  }  }  xhr.send()  }  getRequest('https://jsonplaceholder.typicode.com/users', (err, res) => {  if (err) {  console.log(err)  } else {  console.log(res)  res.forEach(r => alert(r.name))  }  })  getRequest('https://jsonplaceholder.typicode.com/posts', (err, res) => {  if (err) {  console.log(err)  } else {  res.forEach(r => console.log(r.title))  }  }) |
| 186 | Callback Not Always Mean Asynchronous |  |  |
| 187 | Callback is Not Always a Good Solution |  |  |
| 188 | What is Promise, Take a Look at It |  |  |
| 189 | Implement Your Own Version of Fetch API |  |  |
| 190 | Functionalities Came with Promise |  |  |
| 191 | Async Function with Await Keyword |  |  |
| 192 | Async Iterator in Action |  |  |
| 193 | Async Iterator and For Await Of Loop |  |  |
| 194 | Create Async Iterator Using Async Generators |  |  |
| 195 | Introduction to DOM - Document Object Model |  |  |
| 196 | Understanding The Window Object in DOM |  |  |
| 197 | Understanding The Document Object in DOM |  |  |
| 198 | Basic Selectors in DOM Manipulation |  |  |
| 199 | Query Selectors in DOM Manipulation |  |  |
| 200 | What is The Difference Between Two Selectors (getElement vs Query Selector) |  |  |
| 201 | How to Traverse DOM Elements |  |  |
| 202 | How to Loop Throw HTML Collections |  | let listItem = document.getElementsByTagName('li')  // let listItems = Array.from(listItem) //পর্যায়ক্রমে ৩টি লাইন একই কাজ করে  // let listItems = Array.prototype.slice.apply(listItem)  let listItems = [...listItem]  listItems.forEach((li, ind) => {  let text = li.innerHTML  li.innerHTML = `(${ind + 1}) ${text}`  }) |
| 203 | How to Create DOM Elements |  |  |
| 204 | Insert Created Element Using Adjacent Element | getElementById('id'), getElementsByTagName, querySelectorAll, parentElement, children, previousElementSibling, nextElementSibling, firstElementChild, lastElementChild, Array.from(listItem), Array.prototype.slice.apply(listItem), [...listItem], document.createAttribute('title'), setAttribute, attributes, getAttributeNames(),getAttributeNode('class'), getAttribute('id'), addEventListener, appendChild, |  |
| 205 | Update and Remove any DOM Element |  |  |
| 206 | How to Clone Node Include It's Child Elements |  |  |
| 207 | How to Deal with Attributes in DOM Manipulation |  |  |
| 208 | How to Work with Style in DOM |  | // let title = document.getElementById('title')  // title.style.color = '#f00'  // title.style.fontSize = '4rem'  // title.style.background = '#000'  let styleObj = {  background: 'black',  fontSize: '30px',  color: '#efefef'  }  let list = document.getElementById('list');  [...list.children].forEach(li => Object.assign(li.style, styleObj))  Object.assign(list.style, styleObj) |
| 209 | Introducing Events in DOM Manipulation |  | let btn = document.getElementById('btn')  // btn.onclick = function (e) {  // console.log(e)  // }  let list = document.getElementById('list')  btn.addEventListener('click', function (e) {  let item = list.lastElementChild.cloneNode(true)  item.innerHTML = 'Newly Created Item'  list.appendChild(item)  });  // Event Delegation Problem  // [...list.children].forEach(li => {  // li.onclick = function (e) {  // e.target.remove()  // }  // })  list.addEventListener('click', function (e) {  if (this.contains(e.target)) {  e.target.remove()  }  })  let box = document.getElementById('box')  box.addEventListener('mousemove', function (e) {  document.getElementById('x-value').innerHTML = e.offsetX  document.getElementById('y-value').innerHTML = e.offsetY  if (e.offsetY === 50 && e.offsetX) {  alert('50 50')  }  }) |
| 210 | Understanding Event Delegation in DOM Manipulation |  |  |
| 211 | Input Elements and Event Handling |  |  |
| 212 | Check Box Element and Event Handling |  |  |
| 213 | List Item and Input Element Event Handling |  |  |
| 214 | How to Handle Form Data in DOM |  |  |
| 215 | How to Validate Form Data in DOM |  |  |
| 216 | AJAX: How to Send Get Request and Display Data |  |  |
| 217 | AJAX: How to Send Post Request and Display Data |  |  |
| 218 | Conclusion of DOM Manipulation |  |  |
| 219 | What is Library |  |  |
| 220 | Introduction to JQuery |  |  |
| 221 | Introduction to Lodash |  |  |
| 222 | Introduction to Underscore |  |  |
| 223 | Introduction to Math JS |  |  |
| 224 | Introduction to Moment JS | Date time নিয়ে কাজ করার জন্য Moment JS ১১টি অসাম লাইব্রেরী |  |
| 225 | Introduction to Collect JS |  |  |
| 226 | Introduction to Axios JS |  |  |
| 227 | Conclusion of this course |  |  |
| 228 | How to Be A Frontend Developer |  |  |
| 229 | How to Be A Backend Developer |  |  |
| 230 | How to Be A Full Stack Developer |  |  |
| 231 | How to Be A Mobile Apps Developer |  |  |
| 232 | How to Be A Desktop Apps Developer |  |  |
| 233 | Introduction to our Project |  |  |
| 234 | Setup and Resources |  |  |
| 235 | User Permission for Location |  |  |
| 236 | Set Weather Information to DOM |  |  |
| 237 | Handle User Input |  |  |
| 238 | Setup Backend Server |  |  |
| 239 | MongoDB Database and Mongoose Setup |  |  |
| 240 | Create History Model |  |  |
| 241 | Create Hisotry Router |  |  |
| 242 | History Request and Update DOM |  |  |
| 243 | Using Callback to Post New History |  |  |
| 244 | Deploying Apps to Heroku |  |  |
| 245 |  |  |  |
| 246 |  |  |  |
| 247 |  |  |  |
| 248 |  |  |  |
| 249 |  |  |  |
| 250 |  |  |  |
| 251 |  |  |  |
| 252 |  |  |  |
| 253 |  |  |  |
| 254 |  |  |  |
| 255 |  |  |  |
| 256 |  |  |  |
| 257 |  |  |  |
| 258 |  |  |  |
| 259 |  |  |  |
| 260 |  |  |  |
| 261 |  |  |  |
| 262 |  |  |  |
| 263 |  |  |  |
| 264 |  |  |  |
| 265 |  |  |  |
| 266 |  |  |  |
| 267 |  |  |  |
| 268 |  |  |  |
| 269 |  |  |  |
| 270 |  |  |  |
| 271 |  |  |  |
| 272 |  |  |  |
| 273 |  |  |  |
| 274 |  |  |  |
| 275 |  |  |  |
| 276 |  |  |  |
| 277 |  |  |  |
| 278 |  |  |  |
| 279 |  |  |  |
| 280 |  |  |  |

**es6 by code for food**

|  |  |  |  |
| --- | --- | --- | --- |
| 01 Const and let |  |  |  |
| 2 |  |  |  |
| 03 Default parameters |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 |  |  |  |
| 10 |  |  |  |
| 11 Set | সেট কে আমরা কল্পনা করতে পারি অবজেক্ট হিসেবে যার শুধুমাত্র কি আছে কোন ভেলু নেই। সেটের মধ্যে সেইম মাল্টিপল ভেলূ এসাইন করা যায় না।অর্থাত ইউনিক ভেলু এসাইন করা যায়।  Const s = new Set(); // সেট এসাইন করা হয়েছে।  s.add('new'); // ভেলু এড করা হয়েছে।  s.add('new2');  s.add('new');  console.log(s.has('new3')); //ভেলু আছে কিনা চেক করা হয়েছে। থাকলে ট্রু, না থাকলে ফল্স রিটার্ণ করবে।  s.delete('new2'); // ভেলু মুছা হয়েছে।  console.log(s.size); // কয়টি ভেলু আছে তা দেখাবে।  console.log(s.clear()) // সব ভেলু ক্লিয়ার হবে।  const b = new Set(['test1', 'test2']);  b.add('test3');  b.add('test4');  for (const k of b){  console.log(k);  }  Const arr = [...b] // এ্যারেতে কনভার্ট হবে।  Conlose.log(arr); |  |  |
| 12 Array includes | Const arr = [4, 5, 6, 7, 8, 'test', 'boom'];  Console.log(arr.indexOf('test')); // থাকলে ইনডেক্স রিটার্ণ করবে না  থাকলে -1 রিটার্ণ করবে।  Console.log(arr.includes('test')); //es7 থাকলে ট্রু রিটার্ণ করবে না  থাকলে ফলস রিটার্ণ করবে। |  |  |
| 13 Exponentiation Operator | Console.log(Math.pow(2, 4)); // 2 দি পাওয়ার 4 এর জন্য মেথ এর  pow ফাংশনটি ব্যবহার করতে হতো।  console.log(2 \*\* 4); কিন্তু বর্তমানে এভাবে ব্যবহার করা যায় |  |  |
| 14 String Padding | String padding মানে হলো সামনে পেছনে এক্সট্রা কিছু কেরেক্টার এড করা।  Console.log('test'.paddStart(8, 'ab'));  //(es2017 es8) test এর সামনে ‍ab দিয়ে 4 কেরেক্টর পূর্ণ হবে।অর্থাত ababtest এভাবে 8 কেরেক্টরের স্ট্রিং হবে।কোন সিম্বল বা কেরেক্টর না দিলে স্পেস দিয়ে পূর্ণ হবে।  Console.log('te', paddEnd(8, '\*')); // te\*\*\*\*\*\*  স্ট্রিং এর শেষে স্পেস বা কেরেক্টর যুক্ত করতে চাইলে paddEnd ব্যবহার করতে হবে। |  |  |
| 15 Object methods |  |  |  |
| 16 Async function |  |  |  |
| 19 Object fromEntries |  |  |  |
| 17 Array flat |  |  |  |
| 18 Optional catch binding |  |  |  |
| 20 String trim |  |  |  |
| 21 |  |  |  |
| 22 |  |  |  |