
JAVASCRIPT

----

java scripti is a multi-platfrom language. It will run on any modern web browser , it is useful to create a native applications on smartphones and tablets it can create desktop softwares for Mac and Windows computers and it can run on backend servers as well , no matter where you want to your application to run you can code it in JS

We use tool called Plunker

http://plnkr.co/

First Program :(Hello World)

```
console.log('hello World incorrect syntax');
```

Agenda:
- Storing Information in Variables
- Variable Errors
- Starting Our BlackJack Application

Storing Information in Variable :

```
let productName = "Hammer";
let productId = "H123";


console.log(productName,productId);
```

Variable Errors

spelling Mistake
```
let productName = "Hammer";
console.log(productNme);
```

```
script.js:6 Uncaught ReferenceError: productNme is not defined
    at VM442 script.js:6
(anonymous) @ script.js:6
```

ReservedWords can’t be used as variable name

```
let class = "Hammer";
```

console.log(class);

Uncaught SyntaxError: Unexpected token class

Black Jack application Basic code:

```
//
//Blackjack
//by Krishna Dutt Sharma
//

let card1 = "Ace of Spades",
    card2 = "Ten of Hearts";

console.log("Welcome to Blackjack!");

console.log("You are dealt:");
console.log(" " + card1);
console.log(" " + card2);
```

let value =0/0;
console.log(value, typeof(value));
NaN "number"

Undefined and null

Undefined	Null
• JavaScript will initialize variables to undefined • We don't assign undefined to variables	Our source code should set variables to null if needed

Arrays:

let values = [1,2,3,44];

values.splice(1,1); // removes element from starting from first argument as index value and second of elements to be removed

console.log(values);

```
values.shift();  // removes first element
values.pop(); // removes last element

values .push(10) //adds new element to the end of the array
```

splice also insert elements

```
let values = [1,2,3,44];
values.splice(0,3,11,22,33);
console.log(values);

output: [11,22,33,44]
```

Array Methods

Method	Description
concat()	Joins two or more arrays, and returns a copy of the joined arrays
copyWithin()	Copies array elements within the array, to and from specified positions
entries()	Returns a key/value pair Array Iteration Object
every()	Checks if every element in an array pass a test
fill()	Fill the elements in an array with a static value
filter()	Creates a new array with every element in an array that pass a test
find()	Returns the value of the first element in an array that pass a test
findIndex()	Returns the index of the first element in an array that pass a test
forEach()	Calls a function for each array element
from()	Creates an array from an object
includes()	Check if an array contains the specified element
indexOf()	Search the array for an element and returns its position
isArray()	Checks whether an object is an array
join()	Joins all elements of an array into a string
keys()	Returns a Array Iteration Object, containing the keys of the original array
lastIndexOf()	Search the array for an element, starting at the end, and returns its position
map()	Creates a new array with the result of calling a function for each array element
pop()	Removes the last element of an array, and returns that element
push()	Adds new elements to the end of an array, and returns the new length
reduce()	Reduce the values of an array to a single value (going left-to-right)
reduceRight()	Reduce the values of an array to a single value (going right-to-left)
reverse()	Reverses the order of the elements in an array
shift()	Removes the first element of an array, and returns that element
slice()	Selects a part of an array, and returns the new array
some()	Checks if any of the elements in an array pass a test
sort()	Sorts the elements of an array
splice()	Adds/Removes elements from an array
toString()	Converts an array to a string, and returns the result
unshift()	Adds new elements to the beginning of an array, and returns the new length

valueOf()   Returns the primitive value of an array

Black Jack application Using Array:

```
//
//Blackjack
//by Krishna Dutt Sharma
//

let deck = [
  "Ace of Spades",
  "Two of Spades",
  "Three of Spades"
  ];

  let playerCards = [deck[0], deck[2]];

  console.log("Welcome to Blackjack!");

  console.log("You are dealt:")
  console.log(" " + playerCards[0]);
  console.log(" " + playerCards[1]);
```

Program Flow

Flasy	Truthy
False 0 “” or ‘’ (empty strings) null undefined NaN	Everything Not falsy

Comparision (====)

```
let score=10
if(score === 10 )
{
score = score +10;
}
```

console.log(score);

Black Jack application Using Loops:

```
//
//black jack
// krishnadutt sharma
//

let suits = ['Hearts','Clubs','Diamonds','Spades'];
let values = ['Ace','King','Queen','jack',
'Ten','Nine','Eight','Seven','Six',
'Five','Four','Three','Two'];

let deck = [];

for(let suitIdx=0;suitIdx< suits.length; suitIdx++){
  for(let valueIdx =0; valueIdx < values.length ; valueIdx++){
    deck.push(values[valueIdx]+ ' of ' + suits[suitIdx]);
  }
}

for(let i=0;i< deck.length; i++){
  console.log(deck[i]);
}


  let playerCards = [deck[0], deck[2]];

  console.log("Welcome to Blackjack!");

  console.log("You are dealt:")
  console.log(" " + playerCards[0]);
  console.log(" " + playerCards[1]);
```

Functions:

Function Definition

```
function showMessage(){
console.log('in a function ');
}
```

Function calling:

```
showMessage();
```

Black Jack application Using Loops:

```
//
//black jack
// krishnadutt sharma
//

let suits = ['Hearts','Clubs','Diamonds','Spades'];
let values = ['Ace','King','Queen','jack',
'Ten','Nine','Eight','Seven','Six',
'Five','Four','Three','Two'];

function createDeck(){
let deck = [];

for(let suitIdx=0;suitIdx< suits.length; suitIdx++){
  for(let valueIdx =0; valueIdx < values.length ; valueIdx++){
    deck.push(values[valueIdx]+ ' of ' + suits[suitIdx]);
  }
}
  return deck;
}

function getNextCard(){
  return deck.shift();
}

let deck = createDeck();

for(let i=0;i< deck.length; i++){
  console.log(deck[i]);
}


  let playerCards = [getNextCard(),getNextCard()];

  console.log("Welcome to Blackjack!");

  console.log("You are dealt:")
  console.log(" " + playerCards[0]);
  console.log(" " + playerCards[1]);
```

Objects:

```
Let cards = [
 {
suit : “Hearts”,
value : “Queen”
 },
 {
suit: “Clubs”,
value : “king”
 }
]

conosle.log(cards[0].suit);
```

Black Jack application Using Objects:

```
//
//black jack
// krishnadutt sharma
//

let suits = ['Hearts','Clubs','Diamonds','Spades'];
let values = ['Ace','King','Queen','jack',
'Ten','Nine','Eight','Seven','Six',
'Five','Four','Three','Two'];

function createDeck(){
let deck = [];

for(let suitIdx=0;suitIdx< suits.length; suitIdx++){
  for(let valueIdx =0; valueIdx < values.length ; valueIdx++){
    let card = {
      suit : suits[suitIdx],
      value: values[valueIdx]
    };
    deck.push(card);
  }
}
  return deck;
}

function getCardString(card){
  return card.value + ' of ' + card.suit;

}
function getNextCard(){
  return deck.shift();
}
```

```
let deck = createDeck();


  let playerCards = [getNextCard(),getNextCard()];

  console.log("Welcome to Blackjack!");

  console.log("You are dealt:")
  console.log(" " + getCardString(playerCards[0]));
  console.log(" " + getCardString(playerCards[1]));
```

Programming for WebPages:

Till now we are relying on console , now lets create a interactive web page for our Blackjack Game

DOM
Document Object Model: Defines how the data of a web page is organized and manipulated
in simple Document refers to web pages, an HTML file.
And Model refers to data that we use in any program and ofcourse we store our data most of the times in an Object , so that's the Object Model.

Most JavaScript Devleopers use the word DOM instead of webpages .

There are different ways to access the HTML elements in javascript file.
-> One of the common ways is to give each element the ID

```
<!DOCTYPE html>
<html>

  <head>
    <link rel="stylesheet" href="style.css">

  </head>

  <body>
    <h1 id ="title" >Hello Plunker!</h1>
    <p id = "paragraph"> This is some text ..</p>
    <button id= "ok-button">click Me</button>

    <script src="script.js"></script>
  </body>

</html>
```

We placed script tag write before the body closes
So when we work with javascript in script.js we want to make sure that all our Ids are setup. So by placing the script tag underneath all the other elements that we need access to we can be sure that JS knows about them.

Change Paragraph Text

HTML :

```
< p id = “text-area”> </p>
```

JavaScipt:

```
let paragraph = document.getElementById(‘text-area’);
paragarph.innerText = “This is paragraph text..”;
```

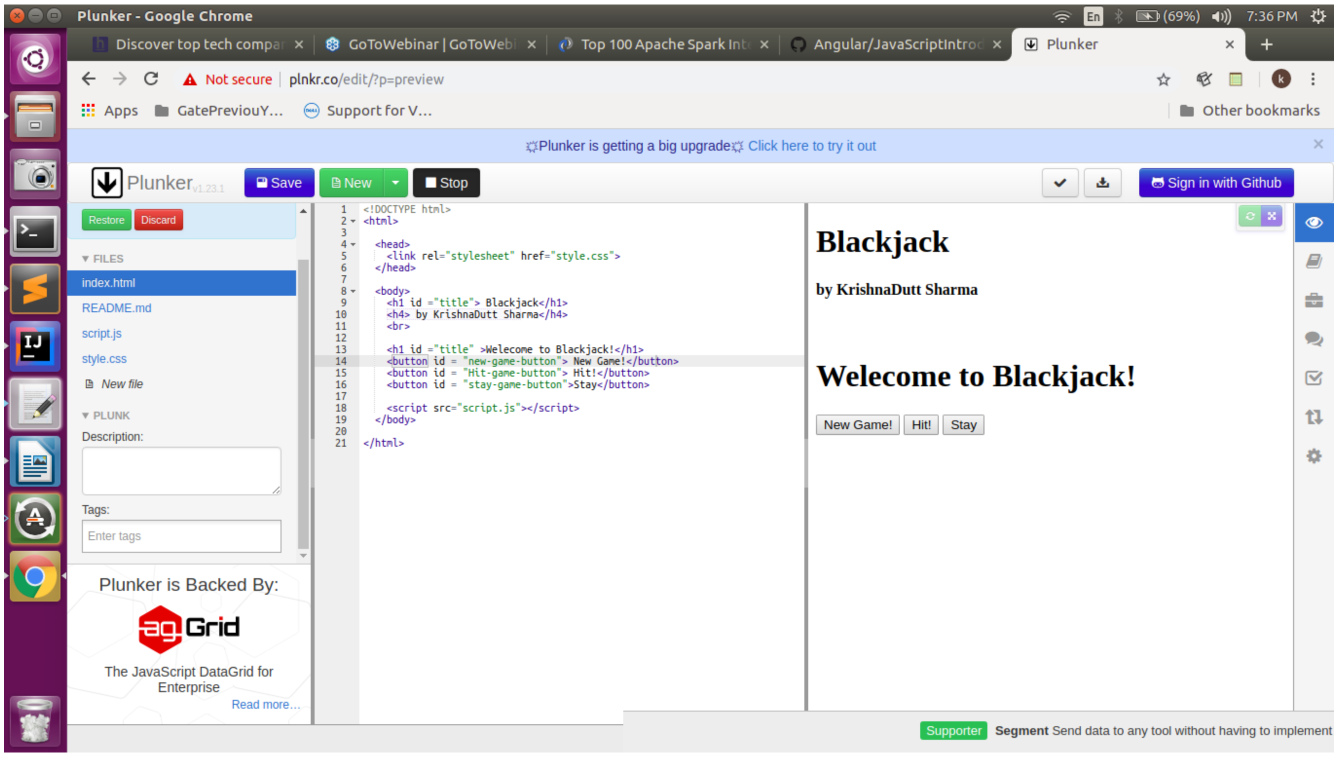
document is special object that refers to Dom in the web page.

JavaScript:

```
let okButton = document.getElementById(‘ok-button’);

okbutton.addEventListener(‘click’,function(){
//code goes here
});
```

Removing and Adding Elements to DOM

```
let paragarph = document.getElementById(‘text-area’);

paragraph.style.display = ‘none’;  // remove it

paragraph.style.display = ‘block’;  //add it back
```



```
<!DOCTYPE html>
<html>

  <head>
    <link rel="stylesheet" href="style.css">
  </head>

  <body>
    <h1 id ="title"> Blackjack</h1>
    <h4> by KrishnaDutt Sharma</h4>
    <br>

    <h1 id ="title" >Welcome to Blackjack!</h1>
    <button id = "new-game-button"> New Game!</button>
    <button id = "Hit-game-button"> Hit!</button>
    <button id = "stay-game-button">Stay</button>

    <script src="script.js"></script>
  </body>

</html>
```

Blackjack

by KrishnaDutt Sharma

Welcome to Blackjack!

index.html

```
<!DOCTYPE html>
<html>

  <head>
    <link rel="stylesheet" href="style.css">
  </head>

  <body>
    <h1 id ="title"> Blackjack</h1>
    <h4> by KrishnaDutt Sharma</h4>
    <br>

    <h1 id ="title" >Welcome to Blackjack!</h1>
    <button id = "new-game-button"> New Game!</button>
    <button id = "Hit-game-button"> Hit!</button>
    <button id = "stay-game-button">Stay</button>

    <script src="script.js"></script>
  </body>

</html>
```

Points to Remember:

- **Set up a Web Page**
  -Remember to move <script> to bottom
- **ChangeText**
  - myParagraph.innerText = "..."
- **Handle a Button Click**
  -myButton.addEventListner(...)
- **Remove and Add Elements**
  -element.sytle.display = "none"
  -element.sytle.display = "block"

Finishing Our BlackJack Game

script.js

```
//
//Black jack
//krishnadutt sharma
//


//Card Variables
let suits = ['Hearts','Clubs','Diamonds','Spades'];
let values = ['Ace','King','Queen','jack',
'Ten','Nine','Eight','Seven','Six',
'Five','Four','Three','Two'];


//DOM Variables
let textArea = document.getElementById('text-area');
let newGameButton = document.getElementById('new-game-button');
let hitButton = document.getElementById('hit-game-button');
let stayButton = document.getElementById('stay-game-button');


//Game Variables
let gameStarted= false,
  gameOver = false,
  playerWon= false,
  dealerCards = [],
  playerCards = [],
  dealerScore=0,
  playerScore=0,
  deck=[];
```

```
hitButton.style.display= 'none';
stayButton.style.display= 'none';


newGameButton.addEventListener('click',function(){
  gameStarted=true;
  gameOver=false;
  playerWon=false;

  deck=createDeck();
  shuffleDeck(deck);
  dealerCards=[getNextCard(),getNextCard()];
  dealerCards=[getNextCard(),getNextCard()];

  newGameButton.style.display= 'none';
  hitButton.style.display = 'inline';
  stayButton.style.display = 'inline';
  showStatus();
});


hitButton.addEventListener('click',function(){
  playerCards.push(getNextCard());
  checkForEndOfGame();
  showStatus();
});

stayButton.addEventListener('click',function(){
  gameOver = true;
  checkForEndOfGame();
  showStatus();
});

function createDeck(){
let deck = [];

for(let suitIdx=0;suitIdx< suits.length; suitIdx++){
  for(let valueIdx =0; valueIdx < values.length ; valueIdx++){
    let card = {
      suit : suits[suitIdx],
      value: values[valueIdx]
    };
    deck.push(card);
  }
}
  return deck;
}
```

```
function getCardString(card){
  return card.value + ' of ' + card.suit;

}
function getNextCard(){
  return deck.shift();
}

function getCardNumericValue(card){
  switch(card.value){
    case 'Ace' :
      return 1;
      case 'Two':
        return 2;
    case 'Three':
      return 3;
    case 'Four':
      return 4;
    case 'Five':
      return 5;
    case 'Six':
      return 6;
    case 'Seven':
      return 7;
    case 'Eight':
      return 8;
    case 'Nine':
      return 9;
      default:
      return 10;
  }
}
function getScore(cardArray){
  let score =0;
  let hasAce = false;
  for(let i=0; i<cardArray.length ; i++){
    let card = cardArray[i];
    score += getCardNumericValue(card);
    if(card.value === 'Ace'){
      hasAce = true;
    }
  }
  if(hasAce && score + 10 <= 21){
    return score +10;
  }
  return score;
}
```

```
function updateScores(){
  dealerScore = getScore(dealerCards);
  playerScore = getScore(playerCards);
}

function showStatus(){
  if(!gameStarted){
    textArea.innerText= "Welcome to Blackjack";
    return ;
  }
let dealerCardString = '';
for(let i=0; i< dealerCards.length; i++){

dealerCardString += getCardString(dealerCards[i]) + '\n';
}

let playerCardString = '';
for(let i=0; i< playerCards.length; i++){

playerCardString += getCardString(playerCards[i]) + '\n';
}

updateScores();

textArea.innerText =
'Dealer has : \n' +
dealerCardString +
'(score: ' + dealerScore + ') \n\n' +

'Player has : \n' +
playerCardString +
'(score: ' + dealerScore + ') \n\n';

if(gameOver){
  if(playerWon){
    textArea.innerHtml += "YOU WIN!";
  }
  else{
    textArea.innerText += "DEALER WINS";
  }

  newGameButton.style.display = 'inline';
  hitButton.style.display = 'none';
  stayButton.style.display= 'none';
}
}

function shuffleDeck(deck){
  for(let i=0;i<deck.length; i++){
    let swapIdx = Math.trunc(Math.random() * deck.length);
    let tmp = deck[swapIdx];
    deck[swapIdx]= deck[i];
    deck[i]= tmp;
  }
}


function checkForEndOfGame(){
  updateScores();
  if(gameOver){
    //let dealer take cards
    while(dealerScore < playerScore && playerScore <=21 && dealerScore <=21){
      dealerCards.push(getNextCard());
      updateScores();
    }
  if(playerScore >21){
    playerWon = false;
    gameOver = true;
  }
  else if(dealerScore > 21)
  {
    palyerWon = true;
    gameOver = true;
  }
  else if(gameOver){
    if(playerScore > dealerScore){
      playerWon = true;
    }
    else {
      playerWon = false;
    }
    newGameButton.style.display= 'inline';
    hitButton.style.display='none';
    stayButton.style.display= 'none';
  }
}
}
```

Advanced JS

Javascript is the language web browsers.

- Desktop applications can be built with “**Electron**”

- Smartphone and Tablet applications can be built with **Cordova**
- Server applications can be built with **NodeJS**

Agenda:

- Intoduction and setup
- JS language Features
- Operators
- Functions and scope
- Objects and Array
- Classes and Modules
- Programming the BOM and DOM
- Promises and Error handling
- Data Access Using Http
- Forms
- Security
- Building for Production

Javascript was created by Brendan Eich in 1995 then in 1997 ECMA standard ,ECMAScript

Tooling : visual Studio
clone https://github.com/wbkd/webpack-starter.git
Make sure you install node then npm package manager
cmd : npm install

npm run dev: this is the command we use to run our code in development environment.
For this to work we need to keep "dev" in scripts block
like below:

```
"scripts": {
"build": "webpack --config webpack/webpack.config.prod.js --colors",
"dev": "webpack-dev-server --open --config webpack/webpack.config.dev.js"
}
```

Below code in webpack.config.dev.js

```
{
test: /\.(js)$/,
include: Path.resolve(__dirname, '../src'),
loader: 'babel-loader'
},
```

`babel-loader :` It converts all of javascript file down to an earlier version of JavaScript, Version 5

```
so we commented out that in our code , we don't to convert to JS5.
```

JavaScript Language Features

- **Constants:**
  const carId=42; // constant needs to be initialized and cannot be changed.
- **Let and Var for Variable Declarations**

Let:

```
console.log (carId) ; // error! This make sense becuase programs are executed from top   to bottom .
let carId=42;
```

```
if(true){
let foo=9;
}
console.log(foo); // error , because let has block scoping where as var don't
```

Var

```
console.log (carId) ; // undefined(not an error)
var carId=42;
```

```
if(true){
var foo =9;
}
console.log(foo); //9
```

- Rest Parameters
  Rest Parameter is a moder feature of JS that allows a function to store multiple arguments in a single array.
  -> Rest parameter must be last parameter in the argument list
  -> that is why the name rest of parameters (restParameters)

```
Function sendCars(... allCarIds){
allCarIds.forEach(id => console.log(id));
}
sendCars(100,200,255);
// 100 200 255
```

```
Funtion sendCars(day , ... allCarIds){
allCarIds.forEach(id => console.log(id));
}
sendCars('Monday',100,200,300);

//100 200 300
```

- Destructuring Arrays

```
Let carIds =[1,2,5]
let [car1,car2,car3] = carIds;
console.log(car1,car2,car3);

//1 2 5
```

```
Let carIds =[1,2,5]
let car1, remainingCars;
let [car1, ... remainingCars] = carIds;
console.log(car1,remainingCars);

//1 [2 5]
```

```
Let carIds =[1,2,5]
let car1, remainingCars;
let [, ... remainingCars] = carIds; // first element will be skipped , if you have two commas(,) two
elements will be skipped.
console.log(remainingCars);

// [2 5]
```

- **Destructuring Objects**

```
Let car = { id: 5000, style: 'convertible'};
let {id,style} = car; //if you are destrucing objects make sure to use curly braces

console.log(id,style);

//5000 convertible
```

```
Let car = { id: 5000, style: 'convertible'};
let id,style;
 {id,style}= car; //we get compiler error here because illegal syntax
({id,style}=car); // this is legal syntax
console.log(id,style);

//5000 convertible
```

- **Spread Syntax**
  This is opposite of restparamters

```
Funcion startCars(car1,car2,car3){
console.log(car1,car2, car3);
}
let carIds = [100,300,500];
startCars(...carIds);
//100,300,500
```

```
Funcion startCars(car1,car2,car3){
console.log(car1,car2, car3);
}
let carIds = “abc”;
startCars(...carIds);
//a b c
```

- **typeof()**

```
typeof(1);  //number
typeof(true); //boolean
typeof(‘hello’) //string
typeof(function(){}); //function
typeof({}); //object
typeof(null); //object
typeof(undefined); //undefined
typeof(NaN); //number
```

- **Common Type Conversions**

```
//convert to string
foo.toString();

//converting string to integer
Number.parseInt(‘55’);  //55 as a number

//convert string to number
Number.parseFloat(‘55.99’); //55.99 as a number
```

OPERATORS

- Equality Operators
- Unary
- logical
- Relational
- Conditional
- Assignment
- Operator Precedence

- **Equality Operators**

```
if(var1 == var2)  {}  // double equal to
// the code block will execute only when var1 and var2 are equal, JS will attempt to convert this
variables to same type
//lets say var1 = number and var2 = string , JS will convert var1 to string.
```

```
//if we use triple equal (=== )
// it will not do type convesrsion
if(var1 === var2){}
```

Functions and scope

IIFE:
Immediately Invoked Function Expression

Function Expression is nothing but taking function and doing something with it.

let app = (function(){
console.log('in function ');
return {};
})();

Closures
Normally when a function executes it runs through all its code and then completes. All of its varilables go out of scope,all of its functions go out of scopes. But sometimes we want that functions and its variables and nested functions to hang around and that's what a closure is.

```
let app = (function(){
let carId=123;
let getId=function(){
return carId;
};
return {
getId: getId
};
})();
console.log(app.getId());
```

Call

```
let o = {
carId : 123,
getId: function(){
return this.carId;
}
};
```

```
let newCar = { carId : 456 } ;

console.log(o.getId.call(newCar));
```

Apply

```
let o = {
carId : 123,
getId: function(prefix){
return prefix+this.carId;
}
};

let newCar = { carId : 456 } ;

console.log(o.getId.apply(newCar,['ID: ']));
```

Bind

```
let o = {
carId : 123,
getId: function(){
return this.carId;
}
};

let newCar = { carId : 456 } ;

let newFn = o.getId.bind(newCar);
console.log( newFn());
```

Note:
call and apply will not create a copy of a function , it will just call an existing function , changing this , but if you want a brand new function , which is a copy of an existing function , you use bind and pass it the value for this function

Arrow Functions

Traditionally we define functions in JS using a keyword - 'function '
 but in the ECMAScript 2015 vesion of JS we now have arrow functions, which have a more concise and modern syntax for function declarations .

Zero Parameter	Let getId = () => 123; console.log(getId()); //123
One parameter	Let getId = prefix => prefix + 123; console.log(getId(‘ID: ‘)); //ID: 123

Two Parameter	Let getId = (prefix,suffix) => {
	return prefix + 123 + suffix;
	}
	console.log(getId(‘ID: ‘, ‘!’)); //ID: 123!

```
Let getId = _ => 123;   // many programmers uses _(underscore) instead of ‘()’ ,here _ is a variable
console.log(getId());      //123
```

Note:

- Arrow functions do not have their own “this” value .
- “this” refers to the enclosing context.

Default Parameters

Earlier version of JS there is no way to take a parameter and give it a default value, we can do that from ES2015 version of JS

```
let trackCar = function (carId, city = 'NY'){
console.log(`Tracking ${carId} in ${city}. `); // This is an example of
interpolation
};

console.log(trackCar(123));

//Tracking 123 in NY.

console.log(trackCar(123,'Chicago'));
//Tracking 123 in Chicago
```

Points to remember:

IIFE’s are used to organize code into modules.
Closures: how function and its context can be kept around in memory and resused, even after a function completes execution.
Call, apply and bind are used to change the **this** value of the function .
-> call is used to call a function,changing the this value.
->apply is same as call , except you can pass it an array of arguments
-> bind actually makes a copy of the function , resetting the this keyword.

Objects and Arrays

Constructor Function

Prototypes

All JavaScript objects inherit properties and methods from a prototype:

- `Date` objects inherit from `Date.prototype`
- `Array` objects inherit from `Array.prototype`
- `Person` objects inherit from `Person.prototype`

The `Object.prototype` is on the top of the prototype inheritance chain:

`Date` objects, `Array` objects, and `Person` objects inherit from `Object.prototype`.

```
function Car(id){
this.carId = id;
}
Car.prototype.start = function (){
console.log('start: ' + this.carId);
};

let car = new Car(123);
car.start(); //123
```

Expanding Objects Using Prototypes:

One of the use of prototype is to extend the given functionality of the objects.

```
String.prototype

String.prototype.hello = function (){
return this.toString() + ‘ Hello ‘;
};
```

console.log(‘foo’.hello()); //foo Hello

JSON – JavaScript object Notation

The purpose of JSON is to transmit JavaScript object over wire. Most Often sending or receiving JS object to some API such as RESTful API on the web.

```
Let car = {
id : 123;
style: ‘convertible’
};
```

```
console.log( JSON.stringify(car));
//{“id”: 123, “sytle”: “convertible”}
```

```
Let jsonIn=
`
[
{“carId” : 123},
{“carId”: 456},
{“carId”: 789}
]
`;

let carIds = JSON.parse(jsonIn);
console.log(carIds);
```

Array Iteration

```
let carIds = [
{ carId: 123, style: ‘sedan’ },
{ carId: 456, style: ‘convertible’ },
{ carId: 789, style: ‘sedan’ }
];

carIds.forEach( car => console.log(car));
carIds.forEach((car,index) => console.log(car,index));
```

Filtering arrays

```
let carIds = [
{ carId: 123, style: ‘sedan’ },
{ carId: 456, style: ‘convertible’ },
{ carId: 789, style: ‘sedan’ }
];

let convertibles = carIds.filter(
car => car.style === ‘convertible’
);
carIds.forEach( convertibles):
```

Every function

```
let carIds = [
{ carId: 123, style: 'sedan' },
{ carId: 456, style: 'convertible' },
{ carId: 789, style: 'sedan' }
];
let results = carIds.every(
car => car.carId > 0
);
console.log( results);
```

Find method()

```
let carIds = [
{ carId: 123, style: 'sedan' },
{ carId: 456, style: 'convertible' },
{ carId: 789, style: 'sedan' }
];
let results = carIds.find(
car => car.carId > 500
);
console.log( results);
```

Classes and Modules

Constructor :

```
class car{
constructor(id){
this.id = id;
}
}

let car = new Car(123);
console.log(car.id);  //123
```

Methods:

methods are functions that exists on an object

```
class car{
constructor(id){
this.id = id;
}
identify(){
return  ‘ Car Id:  ${this.id}’;
}
}

let car = new Car(123);
console.log(car.identify());  //Car Id: 123
```

Modules

An application may have dozens or hundreads of classes how we gonna organize this
The answer is to use classes with modules and any JS code can be placed in module.

Importig modules
import {Car} from ‘./models/car.js’;

./models/car.js	`export class Car{`
	`constructor(id){`
	`this.id = id;`
	`}`
	`}`

./index.js	`import {Car} from './models/car.js';`
	`let car = new Car(123);`
	`console.log(car.id);`

Programming the BOM and DOM

BOM refers to Browser Object Model and that lets you access functionality in the browser. You can change the URL you are pointing at , get information on the URL, and that kind of thing.
DOM refers to Document Object Model thats we use to change the actual web page.

Window Object

Window is the global object in JS. We can access window from anywhere let’s take a look at some of the properties , methods and events on windows.

Below are the some properties , methods and Events on Window

Properties	Methods	Events

documents
location
console
innerHeight
innerWidth
pageXOffset
pageYOffset
	alert()
	back()
	confirm()
		(not common)

Timers:

```
let intervalId = setInterval( function() {
console.log('1 second passed');
});

//if need to cancel...
clearInterval(intervalId);
```

Location Object – thats let you access the URL the browser is currently pointing at.
The Location object is the part of the BOM.

Properties	Methods	Events

href
hostname
port
pathname
search
	assign()
	reload()
		(not common)

DOM

Properties	Methods	Events

body
forms
links
	createElement()
	createEvent()
	getElementById()
	getElementByClassName()
		onload
		onclick
		onkeypress

Selecting DOM Elements

document.getElementById(‘elementId’);

document.getElementsByClassName(‘className’);

document.getElementByTagName(‘tagName’);

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>JavaScript Fundamentals</title>
</head>
<body>
<h1>JavaScript Fundamentals</h1>
<p id = "first" class = "p1" > First Paragraph </p>
<p id = "name1" class = "p1" > Second Paragraph </p>
```

```
<p class = "p3" > Third Paragraph </p>

</body>
</html>
```

```
let el = docuement.getElementById(‘first’);

console.log(el);
```

```
let els = docuement.getElementByClassName(‘p1’);

console.log(els);
```

```
let els = docuement.getElementByTagName(‘p’);

console.log(el);
```

Modifying DOM Elements

```
let element = document.getElementById('elementId');
element.textContent = 'new text here' ;
element.setAttribute('name','nameValue');
element.classList.add('myClassName');
element.style.color = 'blue';
```

Promises and Error handling

Promises:
Promises are design to work with asynchronous JS , if you are working with timer, if you are working with HTTP calls, you can think promise as a temporary holder for a a value that you’ll retrieve once an asynchronous operations completes

```
let promise = new promise(
function(resolve,reject) {
setTimeout(resolve,100,'someValue');
}
);
promise.then(
value => console.log('fulfilled: ' + value),
error => console.log('rejected: ' + error)
)
```

Data Access Using HTTP

HTTP Request Using XHR

```
let  xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function(){
if(this.readyState == 4 && this.status ==200){
console.log(this.responseText);
}
};

xhttp.open("GET","http://myid.mockapi.io/api/v1/users", true);
xhttp.send();
```

HTTP request using jQuery:

```
import $ from 'jquery';

let promise= $.get("http://myid.mockapi.io/api/v1/users",
promise.then(
data => console.log('success',data),
error => console.log('error: ',error)
)
);
```