



Topic – Javascript-2

© Copyright 2021 Accolite. All Rights Reserved



About Me

Javascript-2

Pranoy Ghosh

Associate Technical Delivery Manager

Pranoy has 8+ years of experience in the IT industry across the Telecom & Insurance domain

Expertise : ASP.NET,C#, NodeJs
JavaScript (Angular, AngularJs)
Database (MSSQLServer, Couchbase)
Extras: RASA NLP, SignalR



Agenda

- 01 JS Advanced
- 02 Scope
- 03 IIFE
- 04 Closures
- 05 Callbacks
- 06 Object Oriented JS
- 07 Call,Apply,Bind
- 08 Errors
- 09 Questions





Agenda

- 01 JS Advanced
- 02 Scope
- 03 IIFE
- 04 Closures
- 05 Callbacks
- 06 Object Oriented JS
- 07 Call,Apply,Bind
- 08 Errors
- 09 Questions





- What is JavaScript- Advanced?
 - It is the language of Web.
 - Used along with HTML , CSS.
 - Controls behaviour of DOM elements.
 - Makes dynamic / interactive web-pages on the client side.
 - **Advanced JS supports OOPS concepts.**
 - **Easy to understand and maintain.**
 - **One of the most powerful language in current world.**
 - **Has integrated every feature available like other famous languages like C++/JAVA etc.**
 - **Can independently run an entire application.**
 - **JavaScript is asynchronous and single-threaded.**



Agenda

- 01 JS Advanced
- 02 Scope
- 03 IIFE
- 04 Closures
- 05 Callbacks
- 06 Object Oriented JS
- 07 Call,Apply,Bind
- 08 Errors
- 09 Questions





What is a SCOPE?

Scope determines the accessibility (visibility) of variables.

In JavaScript there are two types of scope:

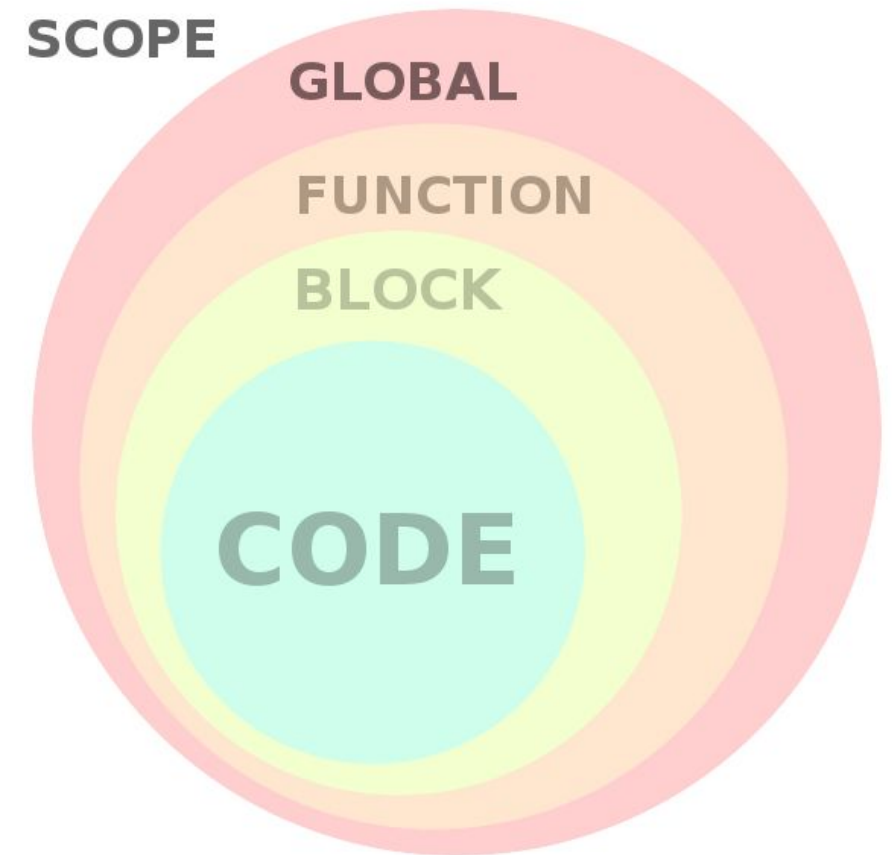
1. LOCAL
2. GLOBAL

JavaScript has function scope: Each function creates a new scope.

Variable Lifetime

Global variables live until the page is discarded, like when you navigate to another page or close the window.

Local variables have short lives. They are created when the function is called and deleted when the function is finished.





How to declare variables ?

- With the keyword [var](#). \Rightarrow `var x = 42;` (function scoped)
- By simply assigning it a value. \Rightarrow `x = 42;`
- With the keyword [let](#). \Rightarrow `let y = 13;` (block scoped)

Evaluating variables

```
> var globalVar=new Date();
   console.log(globalVar);
Sun Jan 10 2021 12:54:07 GMT+0530 (India Standard Time)
< undefined
```

```
> function getDate(){
   var todaysDate=new Date();
   console.log("From Function:"+todaysDate);
   console.log("From Global:"+globalVar);
}
< undefined
```

```
> getDate();
From Function:Sun Jan 10 2021 12:54:17 GMT+0530 (India Standard Time)
From Global:Sun Jan 10 2021 12:54:07 GMT+0530 (India Standard Time)
```

```
console.log("From Function:"+todaysDate);
```

```
► Uncaught ReferenceError: todaysDate is not defined
   at <anonymous>:1:30
```

```
console.log("From Global:"+globalVar);
```

```
From Global:Sun Jan 10 2021 12:54:07 GMT+0530 (India Standard Time)
undefined
```




Agenda

- 01 JS Advanced
- 02 Scope
- 03 IIFE
- 04 Closures
- 05 Callbacks
- 06 Object Oriented JS
- 07 Call,Apply,Bind
- 08 Errors
- 09 Questions



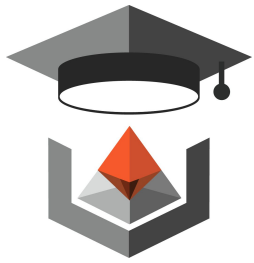


Immediately Invoked Function Expression (IIFE) is one of the most popular design patterns in JavaScript.

As you know that a function in JavaScript creates the local scope. So, you can define variables and function inside a function which cannot be access outside of that function. However, sometime you accidentally pollute the global variables or functions by unknowingly giving same name to variables & functions as global variable & function names. For example, there are multiple .js files in your application written by multiple developers over a period of time. Single JavaScript file includes many functions and so these multiple .js files will result in large number of functions. There is a good chance of having same name of function exists in different .js files written by multiple developer and if these files included in a single web page then it will pollute the global scope by having two or more function or variables with the same name.

The parenthesis () plays important role in IIFE pattern. In JavaScript, parenthesis cannot contain statements; it can only contain an expression.

```
(var foo = 10 > 9); // syntax error
(var foo = "foo", bar = "bar"); // syntax error
(10 > 9); // valid
(alert("Hi")); // valid
```



EXAMPLES:

```
var myIIFE = function () {  
    //write your js code here  
};
```

```
(function () {  
    //write your js code here  
});
```

```
(function () {  
    //write your js code here  
})();
```

```
(function () {  
    var userName = "Steve";  
  
    function display(name)  
    {  
        alert("MyScript2.js: " + name);  
    }  
  
    display(userName);  
})();
```

Without parameter

```
var userName = "Bill";  
  
(function (name) {  
  
    function display(name)  
    {  
        alert("MyScript2.js: " + name);  
    }  
  
    display(name);  
})(userName);
```

With parameter



Advantages of IIFE:

- Do not create unnecessary global variables and functions
- Functions and variables defined in IIFE do not conflict with other functions & variables even if they have same name.
- Organize JavaScript code.
- Make JavaScript code maintainable.

```
function a(){ /*body*/  
a()//call exactly after declaration  
  
is equivalent to  
(function a(){ /*body*/})();
```



Agenda

- 01 JS Advanced
- 02 Scope
- 03 IIFE
- 04 **Closures**
- 05 Callbacks
- 06 Object Oriented JS
- 07 Call,Apply,Bind
- 08 Errors
- 09 Questions





Closures

Closures:

JavaScript variables can belong to the local or global scope.
Global variables can be made local (private) with closures.

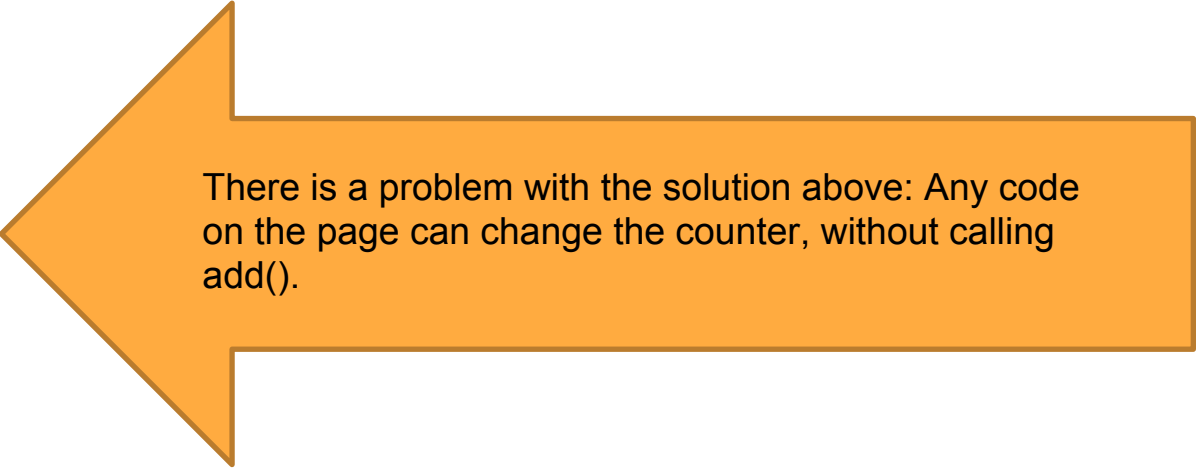
A closure is a function having access to the parent scope, even after the parent function has closed.

```
// Initiate counter
var counter = 0;

// Function to increment counter
function add() {
    counter += 1;
}

// Call add() 3 times
add();
add();
add();

// The counter should now be 3
```



There is a problem with the solution above: Any code on the page can change the counter, without calling add().



Closures

```
// Initiate counter
var counter = 0;

// Function to increment counter
function add() {
  var counter = 0;
  counter += 1;
}

// Call add() 3 times
add();
add();
add();
```

```
// Function to increment counter
function add() {
  var counter = 0;
  counter += 1;
  return counter;
}

// Call add() 3 times
add();
add();
add();
```

It did not work because we reset the local counter every time we call the function.



Closures

```
var add = (function () {  
    var counter = 0;  
    return function () {counter += 1; return counter}  
})();
```

```
add();  
add();  
add();
```

The variable **add** is assigned to the return value of a self-invoking function.

The self-invoking function only runs once. It sets the counter to zero (0), and returns a function expression.

This way add becomes a function. The "**wonderful**" part is that it can access the counter in the parent scope.

This is called a JavaScript closure. It makes it possible for a function to have "**private**" variables.

The counter is protected by the scope of the anonymous function, and can only be changed using the add function.

One important characteristic of closure is it keep the state of outer variable between the multiple calls



Advantages of using Closures

As we all know variables which we create inside function have local scope and only accessible inside the function not outside the function.

Problem 1: Also variable defined inside the function created when we call function and destroyed when function close. We can define global variables which created when program starts till the end of program and accessible anywhere in the program.

Problem 2: If we define the global variable these can be changed anywhere in program.

Solution :

Data Encapsulation

we can overcome above problems by using closures.

1. By using a closure we can have private variables that are available even after a function task is finished.
2. With a function closure we can store data in a separate scope, and share it only where necessary.



Disadvantages of using Closures

Issue 1: As long as the closure are active , the memory can't be garbage collected.

example : If we are using closure in ten places then unless all the ten process complete it hold the memory which cause memory leak.

-How to fix this?

@ If there come a point in you program where you are done using closure then you need to set closure to null.

Issue 2: Creating a function inside a function leads to duplicity in memory and cause slowing down the application.

-How to fix?

@ Use closures only when you need privacy otherwise use module pattern to create new objects with shared methods.



Agenda

- 01 JS Advanced
- 02 Scope
- 03 IIFE
- 04 Closures
- 05 **Callbacks**
- 06 Object Oriented JS
- 07 Call,Apply,Bind
- 08 Errors
- 09 Questions





Callbacks

Callbacks

"I will call back later!"

A callback is a function passed as an argument to another function

This technique allows a function to call another function

A callback function can run after another function has finished

```
function add(x,y)
{
  console.log("called add function");
  return x+y;
}
function display(z) {
  console.log(z);
}
var sumValue=add(3,4);
display(sumValue);
called add function
7
```

JavaScript functions are executed in the sequence they are called. Not in the sequence they are defined.

```
function add(x,y)
{
  console.log("called add function");
  return x+y;
}
function display(z) {
  console.log(z);
}
display(sumValue);
var sumValue = add(3, 4);
7
called add function
```




Callbacks

Callbacks are used to synchronously execute JS functions. Majorly used for time consuming transactions like reading file, reading db, API calls etc.

```
function add(x,y,callback)
{
    console.log("called add function");
    let sum=x+y;
    callback(sum);
}
function display(z) {
    console.log(z);
}
add(3, 4,display);
called add function
7
```

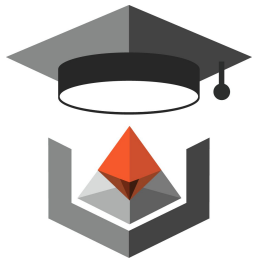
1. Defined 2 functions add and display.
2. Passed an extra param in add function – callback
3. Called the callback method inside the function add.
4. This will hold the execution of display function unless “let sum=x+y” is executed and evaluated.



Agenda

- 01 ■ JS Advanced
- 02 ■ Scope
- 03 ■ IIFE
- 04 ■ Closures
- 05 ■ Callbacks
- 06 ■ **Object Oriented JS**
- 07 ■ Call,Apply,Bind
- 08 ■ Errors
- 09 ■ Questions

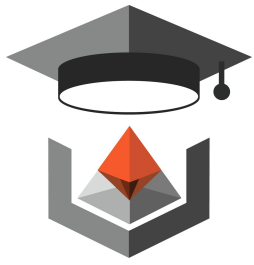




OOPS in JS

OOPS means it has to have 4 specific features:

- Object**
- Classes**
- Encapsulation**
- Inheritance**



Object in JavaScript :

```
//Defining object
let person = {
  first_name: 'Mukul',
  last_name: 'Latiyan',

  //method
  getFunction : function(){
    return (`The name of the person is
      ${person.first_name} ${person.last_name}`)
  },
  //object within object
  phone_number : {
    mobile: '12345',
    landline: '6789'
  }
}

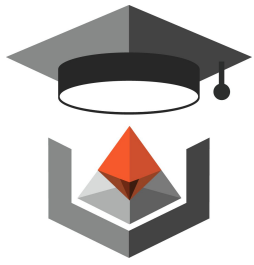
console.log(person.getFunction());
console.log(person.phone_number.landline);
```

```
//using a constructor
function person(first_name,last_name){
  this.first_name = first_name;
  this.last_name = last_name;
}

//creating new instances of person object
let person1 = new person('Mukul','Latiyan');
let person2 = new person('Rahul','Avasthi');

console.log(person1.first_name);
console.log(`${person2.first_name} ${person2.last_name}`);
```

Using Constructor



Class in JavaScript :

```
// Defining class in a Traditional Way.
function Vehicle(name,maker,engine){
    this.name = name,
    this.maker = maker,
    this.engine = engine
};

Vehicle.prototype.getDetails = function(){
    console.log('The name of the bike is ' + this.name);
}

let bike1 = new Vehicle('Hayabusa','Suzuki','1340cc');
let bike2 = new Vehicle('Ninja','Kawasaki','998cc');

console.log(bike1.name);
console.log(bike2.maker);
console.log(bike1.getDetails());
```

```
// Defining class using es6
class Vehicle {
    constructor(name, maker, engine) {
        this.name = name;
        this.maker = maker;
        this.engine = engine;
    }
    getDetails(){
        return `The name of the bike is ${this.name}.`
    }
}

// Making object with the help of the constructor
let bike1 = new Vehicle('Hayabusa', 'Suzuki', '1340cc');
let bike2 = new Vehicle('Ninja', 'Kawasaki', '998cc');

console.log(bike1.name);    // Hayabusa
console.log(bike2.maker);   // Kawasaki
console.log(bike1.getDetails());
```



Encapsulation in JavaScript :

```
//encapsulation example
class person{
  constructor(name,id){
    this.name = name;
    this.id = id;
  }
  add_Address(add){
    this.add = add;
  }
  getDetails(){
    console.log(`Name is ${this.name},Address is: ${this.add}`);
  }
}

let person1 = new person('Mukul',21);
person1.add_Address('Delhi');
person1.getDetails();
```

```
// Abstraction example
function person(fname,lname){
  let firstname = fname;
  let lastname = lname;

  let getDetails_noaccess = function(){
    return `First name is: ${firstname} Last
      name is: ${lastname}`;
  }

  this.getDetails_access = function(){
    return `First name is: ${firstname}, Last
      name is: ${lastname}`;
  }
}

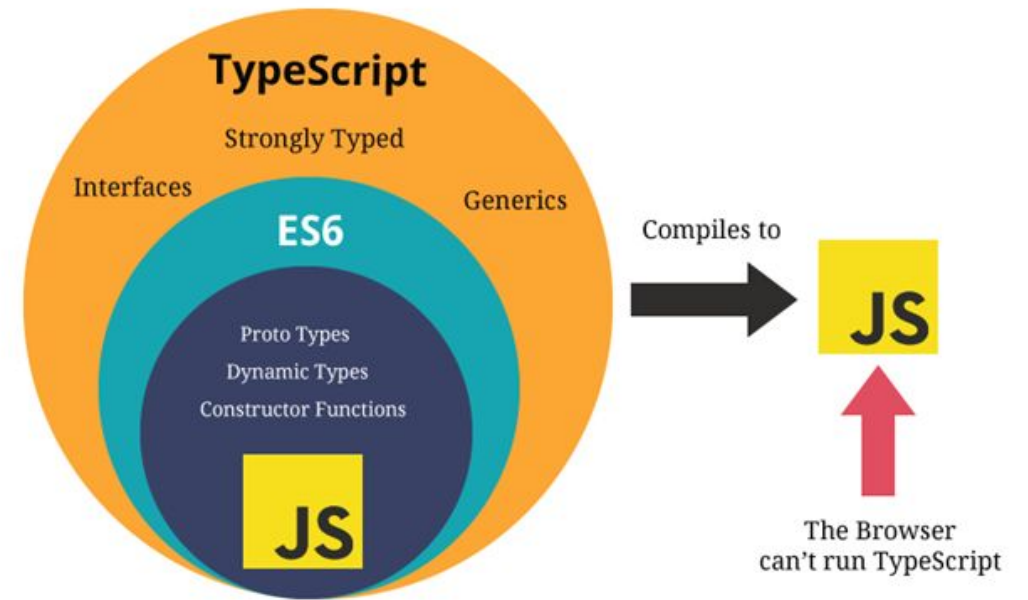
let person1 = new person('Mukul','Latiyan');
console.log(person1.firstname);
console.log(person1.getDetails_noaccess);
console.log(person1.getDetails_access());
```




OOPS in JS

Inheritance in JavaScript :

```
//Inheritance example
class person{
  constructor(name){
    this.name = name;
  }
  //method to return the string
  toString(){
    return `Name of person: ${this.name}`;
  }
}
class student extends person{
  constructor(name,id){
    //super keyword to for calling above class constructor
    super(name);
    this.id = id;
  }
  toString(){
    return `${super.toString()},Student ID: ${this.id}`;
  }
}
let student1 = new student('Mukul',22);
console.log(student1.toString());
```





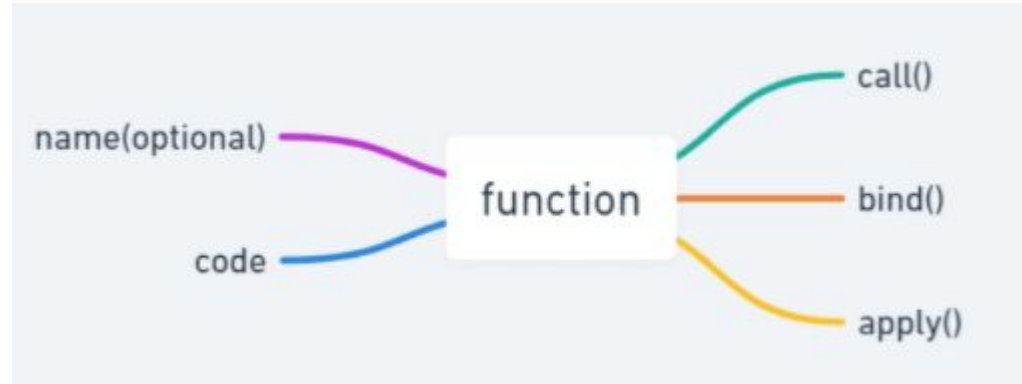
Agenda

- 01 JS Advanced
- 02 Scope
- 03 IIFE
- 04 Closures
- 05 Callbacks
- 06 Object Oriented JS
- 07 **Call,Apply,Bind**
- 08 Errors
- 09 Questions





Call, Apply & Bind



BIND:

The bind() method creates a new function that, when called, has its this keyword set to the provided value.

```
var pokemon = {
  firstname: 'Pika',
  lastname: 'Chu ',
  getPokeName: function() {
    var fullname = this.firstname + ' ' + this.lastname;
    return fullname;
  }
};
```

```
var pokemonName = function() {
  console.log(this.getPokeName() + 'I choose you!');
};
```

```
var logPokemon = pokemonName.bind(pokemon); // creates new object and binds pokemon.
```

```
logPokemon(); // 'Pika Chu I choose you!'
```

```
var pokemon = {
  firstname: 'Pika',
  lastname: 'Chu ',
  getPokeName: function() {
    var fullname = this.firstname + ' ' + this.lastname;
    return fullname;
  }
};
```

```
var pokemonName = function(snack, hobby) {
  console.log(this.getPokeName() + 'I choose you!');
  console.log(this.getPokeName() + ' loves ' + snack + ' and ' + hobby);
};
```

```
var logPokemon = pokemonName.bind(pokemon); // creates new object and binds pokemon.
```

```
logPokemon('sushi', 'algorithms'); // Pika Chu loves sushi and algorithms
```



Call, Apply & Bind

CALL:

The `call()` method calls a function with a given this value and arguments provided individually.

`call()` and `apply()` serve the exact same purpose. The only difference between how they work is that `call()` expects all parameters to be passed in individually, whereas `apply()` expects an array of all of our parameters.

```
var pokemon = {
  firstname: 'Pika',
  lastname: 'Chu ',
  getPokeName: function() {
    var fullname = this.firstname + ' ' + this.lastname;
    return fullname;
  }
};

var pokemonName = function(snack, hobby) {
  console.log(this.getPokeName() + ' loves ' + snack + ' and ' + hobby);
};

pokemonName.call(pokemon, 'sushi', 'algorithms'); // Pika Chu  loves sushi and algorithms
pokemonName.apply(pokemon, ['sushi', 'algorithms']); // Pika Chu  loves sushi and algorithms
```



Agenda

- 01 JS Advanced
- 02 Scope
- 03 IIFE
- 04 Closures
- 05 Callbacks
- 06 Object Oriented JS
- 07 Call,Apply,Bind
- 08 **Errors**
- 09 Questions





Errors

The **try** statement lets you test a block of code for errors.

The **catch** statement lets you handle the error.

The **throw** statement lets you create custom errors.

The **finally** statement lets you execute code, after try and catch, regardless of the result.

```
try {  
    Block of code to try  
}  
catch(err) {  
    Block of code to handle errors  
}  
finally {  
    Block of code to be executed regardless of the try / catch result  
}
```




Excercise

1. Write a JavaScript program along with UI which accept a string as input and then does the following.
 1. Calculates the length of the string and displays on UI
 2. Converts the string to uppercase and lowercase respectively and displays on UI
 3. Separates the vowels and consonants – displays on UI
 4. Gives an error if any number is present in the string.

All these should be executed 1 by 1 synchronously and logged in console.



Agenda

- 01 JS Advanced
- 02 Scope
- 03 IIFE
- 04 Closures
- 05 Callbacks
- 06 Object Oriented JS
- 07 Call,Apply,Bind
- 08 Errors
- 09 Questions





Any Question?

Q & A





Do it yourself

01 Exercise 1

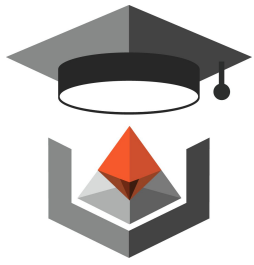
Create an HTML Form with different kind of controls.
Submit button should validate all the controls.

- Input fields should be required
- Radio should be selected
- Dropdown should be selected
- Email should be validated.

Create an object of student with firstname, lastname, age , gender, rollno and dob. Add a field occupation to the object with an input value from user.

Create a Get Button which when clicked with an input of roll no. shows the student details.

Make use of console.log wherever possible.



Thank You



twitter.com/weareaccolite



facebook.com/accolite



linkedin.com/company/accolite