JAVASCRIPT

JavaScript is a scripting language that allows you to implement complex features on web pages.

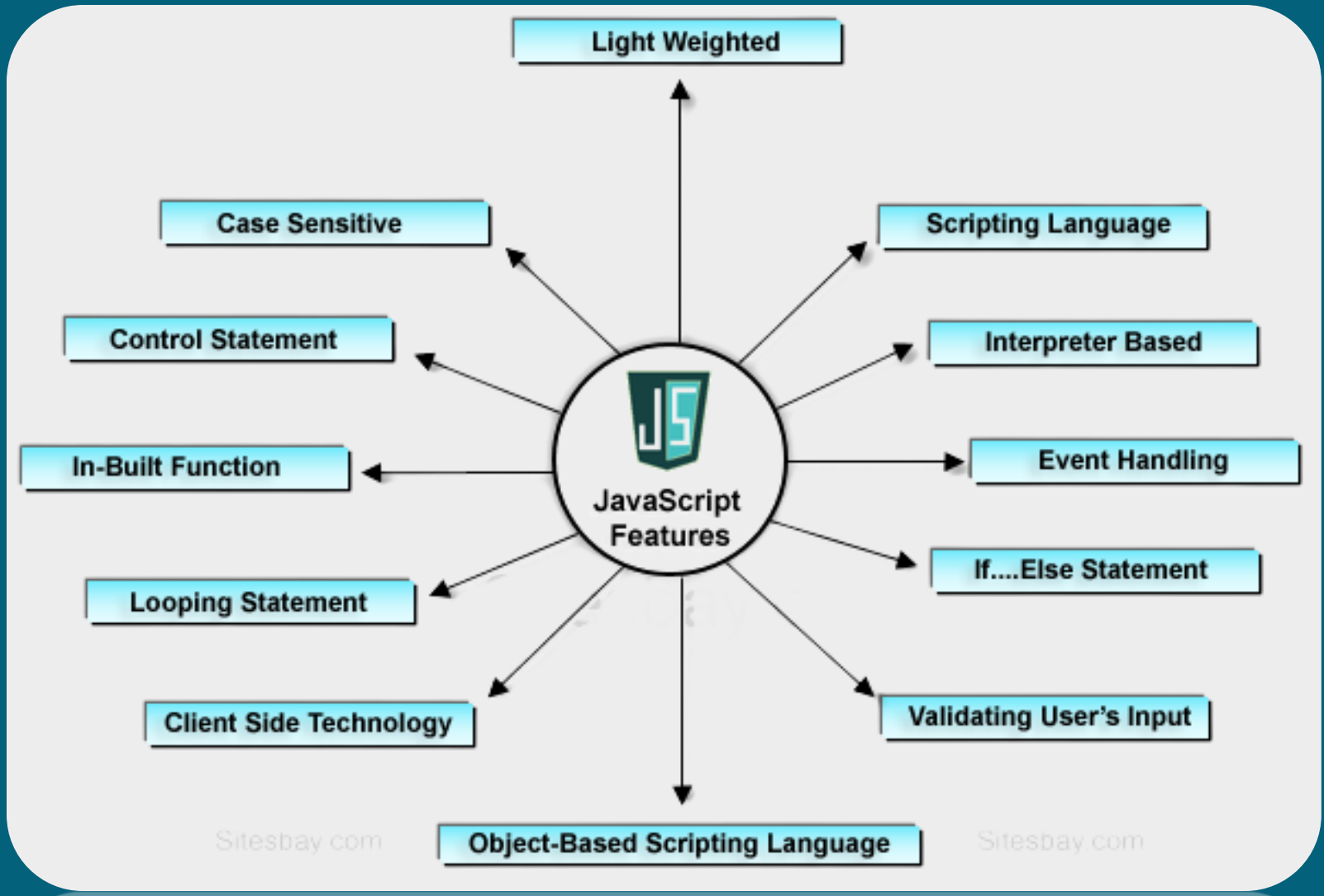
A Brief History of Javascript

- JavaScript was created at Netscape Communications by Brendan Eich in 1995. After its release, more and more browsers started adding JavaScript support.
- In 2008, the creation of Google's open-source Chrome V8, a high-performance JavaScript engine, provided a crucial turning point for JavaScript. The subsequent proliferation of fast JavaScript engines made it possible for developers to build sophisticated browser-based applications with performance that competed with desktop and mobile applications.
- Soon after, Ryan Dahl released an open-source, cross-platform environment called Node.js. It provided a way to run JavaScript code from outside a browser. It freed JavaScript from the browser's confines and led directly to JavaScript's current popularity. Today, you can use JavaScript to write all kinds of applications, including browser, server, mobile, and desktop applications. Most major online companies today, including Facebook, Twitter, Netflix, and Google, all use JavaScript in their products.
- Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.

Why Javascript

- ✓ It's the most popular programming language.
- ✓ It Works in The Browser
- ✓ Easy to Learn
- ✓ Versatile Programming Language
- ✓ Big Community Support
- ✓ To built interactive web pages
- ✓ It is central to all websites
- ✓ Great Career Opportunities

Features of Javascript



Agenda

- Variables and Data Type
- Scope
- Operators
- Conditional statements
- Loops
- Array and its methods Function
- OOP in Javascript
- Asynchronous JavaScript
- JS Events
- DOM- 3 project
- Working with api
- JSON

Variable

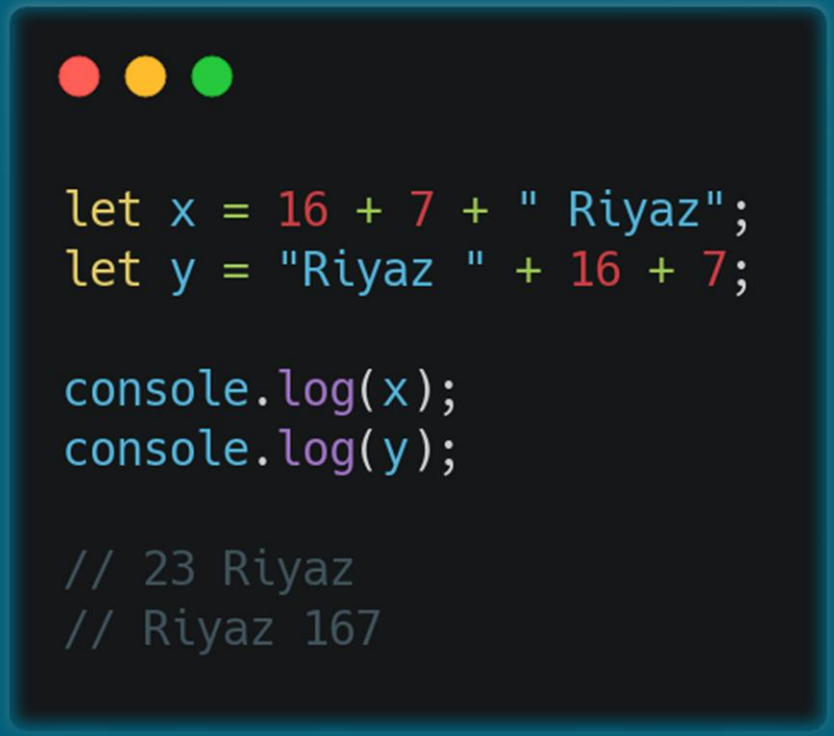
- **Varibales**: Variables are containers for storing data values.
- To declare a variable you have to follow 6 rules
  1. don't use reserve keyword
  2. variable name should be one word
  3. quote is not allowed
  4. Try to use camel case
  5. don't starting with number
  6. case sensitive
- 3 ways to declare varibale in js
  1. var
  2. let
  3. const

```
var myName = "Riyaz";
let number = 17;
const pi = 3.14716;

console.log(myName);
console.log(number);
console.log(pi);
```

Data type

- JavaScript variables can hold different data types: numbers, strings, objects , array and more.

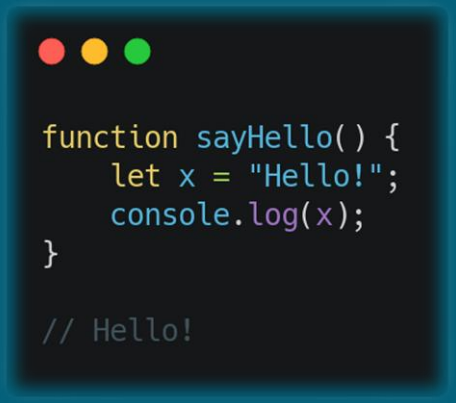
- JavaScript evaluates expressions from left to right.

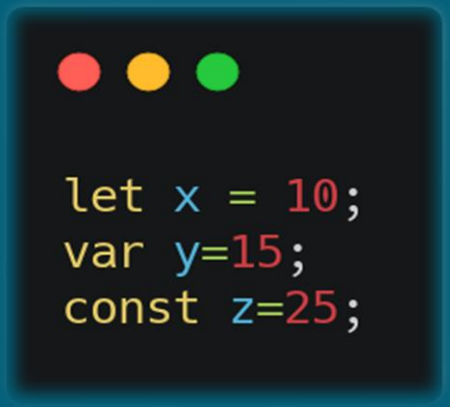


```
let x = 16 + 7 + " Riyaz";
let y = "Riyaz " + 16 + 7;

console.log(x);
console.log(y);

// 23 Riyaz
// Riyaz 167
```

Scope

- Scope determines the accessibility (visibility) of variables.
- JavaScript has 3 types of scope:
  1. Block scope → A variable declared inside a block, becomes block scope.
  2. Function scope → A variable declared inside a function, becomes function scope.
  3. Global scope → A variable declared outside a function, becomes global scope.
- Before ES6 (2015), JavaScript had only Global Scope and Function Scope





Operators

- There are different types of JavaScript operators:
  - Arithmetic Operators (+ , - , * , / , % , ++, -- , **)
  - Assignment Operators (= , += , -= , *= , /= , %= , **=)
  - Comparison Operators (== , === , != , !== , < , > , <= , >= , ? :)
  - Logical Operators (&& , || , !)
  - Conditional Operators ? :

Operators

There are different types of JavaScript operators:

- Arithmetic Operators (+ , - , * , / , % , ++, -- , **)
- Assignment Operators (= , += , -= , *= , /= , %= , **=)

Arithmetic Operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Remainder	x % y
++	Increment (increments by 1)	++x or x++
--	Decrement (decrements by 1)	--x or x--
**	Exponentiation (Power)	x ** y

Assignment Operators

Operator	Name	Example
=	Assignment operator	a = 7; // 7
+=	Addition assignment	a += 5; // a = a + 5
-=	Subtraction Assignment	a -= 2; // a = a - 2
*=	Multiplication Assignment	a *= 3; // a = a * 3
/=	Division Assignment	a /= 2; // a = a / 2
%=	Remainder Assignment	a %= 2; // a = a % 2
**=	Exponentiation Assignment	a **= 2; // a = a**2

Operators

- Comparison Operators (== , === , != , !== , < , > , <= , >= , ? :)
- Logical Operators (&& , || , !)
- Conditional Operators ? :

Comparison Operators

Operator	Description	Example
`==`	**Equal to:** returns `true` if the operands are equal	`x == y`
`!=`	**Not equal to:** returns `true` if the operands are not equal	`x != y`
`===`	**Strict equal to:** `true` if the operands are equal and of the same type	`x === y`
`!==`	**Strict not equal to:** `true` if the operands are equal but of different type or not equal at all	`x !== y`
`>`	**Greater than:** `true` if left operand is greater than the right operand	`x > y`
`>=`	**Greater than or equal to:** `true` if left operand is greater than or equal to the right operand	`x >= y`
`<`	**Less than:** `true` if the left operand is less than the right operand	`x < y`
`<=`	**Less than or equal to:** `true` if the left operand is less than or equal to the right operand	`x <= y`

Logical Operators

Operator	Description	Example
`&&`	**Logical AND:** `true` if both the operands are `true`, else returns `false`	`x && y`
`||`	**Logical OR:** `true` if either of the operands is `true`; returns false if both are `false`	`x || y`
`!`	**Logical NOT:** `true` if the operand is `false` and vice-versa.	`!x`

```
function getFee(isMember:any) {
return (isMember ? '$2.00' : '$10.00');
}

console.log(getFee(true));
// expected output: "$2.00"

console.log(getFee(false));
// expected output: "$10.00"
```

Conditional Operators

Conditional statements

- when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.
- In JavaScript we have 4 conditional statements
  1. if
  2. Else
  3. Else if
  4. switch

Conditional statements

If Else

```
let age = 19;

if (age <= 0) {
    console.log("Please Enter a valid age!")
}
else if (age >= 18) {
    console.log("You are adult!");
}
else {
    console.log("You are child!")
}
```

Conditional statements

Switch

```
let fruits = "Banana"

switch (fruits) {

    case "Banana":
        console.log("Enter fruit is Banana");
        break;

    case "Apple":
        console.log("Enter fruit is Apple");
        break;
}

// Enter fruit is Banana
```

Conditional statements

- 1- Positive or negative
- 2- Even or Odd
- 3- Find max between 2 numbers
- 4- Find max among 3 numbers
- 5- Check vowel or consonant
- 6- input week number and console total day.
- 7- input month number and console number of days in that month.
- 8- Number of day in a date
- 9- Result Sheet
- 10- calculate electricity bill from given condition
- For first 50 units tk. 0.50/unit
- For next 100 units tk. 0.75/unit
- For next 100 units tk. 1.20/unit
- For unit above 250 tk. 1.50/unit
- An additional Meter charge is 55 tk added to the bill.

Loops

- JavaScript supports different kinds of loops:
  1. for loop
  2. for in loop
  3. for of loop
  4. forEach loop
  5. while loop
  6. do while loop

Loops

- **For Loop :**

```
for (initialization ; condition
   ; increment or decrement)
{
   // statements inside the
   body of  loop
}
```

- **For in Loop:**

```
for (key in objects/array/)
{
   // statements inside the
   body of  loop
}
```

```
//for loop

for (let i = 0; i < 15; i++) {
    console.log(i);
}

// 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

let employees = ["Shuvo", "Poyal", "Shahid"];

// for in loop

for (let emp in employees) {
    console.log(employees[emp]);
}

// Shuvo Poyal Shahid
```

Loops

- **For of Loop:**

```
for (variable of iterable)
 {
   // code block to be
   executed
}
```

- **forEach Loop:**

```
forEach (variable of
   iterable)
 {
   // code block to be
   executed
}
```

```
// for of loop

let userInfo = [
    { name: "Riyaz", email: "mdfdf@gmail.com", phone: "+88011*****" },
    { name: "Riyaz", email: "mdfdf@gmail.com", phone: "+88011*****" }
]

for (const iterator of userInfo) {
    console.log(iterator)
}

//forEach loop

let arr = ["Tahoid", "Sojol", "Shuvo"];
arr.forEach((name) => {
    console.log(name);
});
```

Loops

- **While Loop:**

```
while(condition)
 {
    // code block to be
    executed
}
```

- **do while loop:**

```
do
{
    // code block to be
    executed
 } while (conditional);
```

```
// while loop

let i = 1;
while (i < 10) {
    console.log(i);
    i++;
}

//do while loop

do {
    console.log(i);
    i++;
} while (i < 10)
```

Some Important Math Function

- Math.PI Returns the value of PI
- Math.round(x) Returns x rounded to its nearest integer
- Math.ceil(x) Returns x rounded up to its nearest integer
- Math.floor(x) Returns x rounded down to its nearest integer
- Math.trunc(x) Returns the integer part of x
- Math.pow(x, y) Returns the value of x to the power of y
- Math.sqrt(x) Returns the square root of x.
- Math.abs(x) Returns the absolute (positive) value of x:
- Math.sign(x) Returns 1 if x is positive Returns -1 is x is negative.
- Math.random() Returns a random number between 0 and 1
- Math.min() and Math.max() can be used to find the lowest or highest value in a list of arguments:

The Backtick Symbol (`)

- In JavaScript (JS), the backtick symbol (`) is used to create template literals or template strings. Template literals are an enhanced way of creating strings that allow for easy embedding of variables and multiline strings.
- Inside the template literal, variables can be embedded using the ${variable} syntax
- Template literals also support multiline strings without the need for escaping newlines.
- Using backticks and template literals can make string manipulation and variable interpolation in JavaScript more convenient and readable.

The Backtick Symbol (`) Example

```
const multilineString = `
This is a
multiline string
without the need for escaping newlines.
`;

console.log(multilineString);

let name = "Riyaz"
let citizen = "Bangladesh"

console.log(`My Name is ${name} and I am citizen of ${citizen}`);
```

Loop Example

- Loop Example numbers, odd numbers, even numbers
- console all numbers from an array which are less then 50
- for and while loop in a reverse way
- practice problem:

- Write a program to calculate the average marks of math, biology, chemistry, physics and bangla for a student

input:
    5 subject marks mentioned above
output:
    Console the result in 2 decimal places
sample input:
    75.25,65,80,35.45,99.5
output
    71.04

Array

- Array is a special type of variable that stores multiple value in one single variable.

```
let employees = ["Riyaz" , "Shuvo" , "Shahid" , "Taohid"];
```

- To find the array length:

```
//to find the array length
console.log(employees.length);
```

Array

- **indexOf** : by using indexOf we can find the element index number.
- **Push()** : The push() method adds a new element to an array (at the end).
- **Pop()** : The pop() method removes the last element from an array.
- **Shift()** : The shift() method removes the first array element.
- **unshift()**: The unshift() method adds the first array element.

```
let employees = ["Riyaz", "Shuvo", "Shahid", "Taohid"];
let employeeIndex = employees.indexOf("Shuvo");
console.log(employeeIndex);
//1
```

```
employees.push("Ismail")
console.log(employees);
//[ 'Riyaz', 'Shuvo', 'Shahid', 'Taohid', 'Ismail' ]
```

```
employees.pop()
console.log(employees);
//[ 'Riyaz', 'Shuvo', 'Shahid', 'Taohid' ]
```

```
employees.shift()
console.log(employees);
//[ 'Riyaz', 'Shuvo', 'Shahid', 'Taohid' ]
```

```
employees.unshift("Poyal");
console.log(employees);
//[ 'Poyal', 'Riyaz', 'Shuvo', 'Shahid', 'Taohid' ]
```

Array

- **Splice()**: The splice() method changes the contents of an array by removing or replacing existing elements and/or adding new elements
- **syntax**:

splice(index) ⟶ remove all element starting from index

splice(index , n) ⟶ Remove n elements from index number

splice(start , n , item1) ⟶ Remove n elements from index number and insert item1

splice(start, n, item1, item2, itemN) ⟶ Remove n elements from index number and insert item1, item2 , itemn

```
let months = ["January", "February", "March", "April", "Monday", "Tuesday", "May"];
let days = months.splice(3);
console.log(days); //[ 'April', 'Monday', 'Tuesday', 'May' ]
console.log(months); // [ 'January', 'February', 'March' ]
let month = months.splice(1, 2);
console.log(month);//[ 'February', 'March' ]
console.log(months);//[ 'January' ]
let newMonths = months.concat(month, days);
console.log(newMonths)
//["January", "February", "March", "April", "Monday", "Tuesday", "May"]
```

Array

- **slice()**: The slice() method slices out a piece of an array into a new array.
- **toString()**: The toString() method returns an array as a comma separated string.

```
let arr = [1, 15, 17, 19, 5]
console.log(arr.slice(2, 4));
//17,19
```

```
let arr = [1, 15, 17, 19, 5]
console.log(arr); //[ 1, 15, 17, 19, 5 ]
console.log(typeof (arr));//object
let st = arr.toString();
console.log(st);//1,15,17,19,5
console.log(typeof (st))//string
```

Array

- **Map()**: map() creates a new array from calling a function for every array element. map() calls a function once for each element in an array. map() does not execute the function for empty elements. map() does not change the original array.

```
let userInfo = [
    { name: "Riyaz", email: "mdfdf@gmail.com", phone: "+88011*****" },
    { name: "Riyaz", email: "mdfdf@gmail.com", phone: "+88011*****" }
]
userInfo.map((data, index) => console.log(data , index))
// { name: 'Riyaz', email: 'mdfdf@gmail.com', phone: '+88011*****' } 0
// { name: 'Riyaz', email: 'mdfdf@gmail.com', phone: '+88011*****' } 1
```

Array (map vs. forEach)

- Return Value: map returns a new array with transformed elements, while forEach returns undefined.
- If you need to create a new array with transformed values, use map. If you simply need to iterate over an array and perform an action on each element without creating a new array, use forEach.

Array

- **Filter()**: The filter() method creates a new array filled with elements that pass a test provided by a function. The filter() method does not execute the function for empty elements. The filter() method does not change the original array.

```
let ages = [11, 15, 19, 21, 23, 29];
let adult = ages.filter((age, index) => age > 18);
console.log(adult);
//[ 19, 21, 23, 29 ]
```

Let’s Solve some Problems on Arrays

1. find the sum of all element of an array
2. find the max number of an array
3. find the min number of an array
4. remove duplicate item from an array
5. find the adult form given array using filter
6. find the cheapest phone from an array of phone objects

```js
//1- find the sum of given array

let arr = [1, 2, 3, 4, 5];
let sum = 0;
for (let i = 0; i < arr.length; i++) {
    const element = arr[i];
    sum += element;
}
console.log(sum);//15

// 2- find the max number of an array

let max = arr[0]
for (let i = 0; i < arr.length; i++) {
    const element = arr[i];
    if (element > max) {
        max = element;
    }
}
console.log(max)//5

// 3- find the min number of an array

let min = arr[0]
for (let i = 0; i < arr.length; i++) {
    const element = arr[i];
    if (element < min) {
        min = element;
    }
}
console.log(min)//1
```

```js
//4- remove duplicate item from an array

let arr = [1, 3, 5, 7, 9, 1, 11, 15, 5];
let unique = [];

for (let i = 0; i < arr.length; i++) {
    const element = arr[i];
    if (unique.indexOf(element) == -1) {
        unique.push(element);
    }
}
console.log(unique);//[1,  3,  5, 7,9, 11, 15]

//6- find the cheapest phone from an array of phone objects

let phones = [
    { name: "Samsung A10", price: 15000, camera: 15, storage: 32 },
    { name: "Walton A55", price: 13000, camera: 15, storage: 32 },
    { name: "HTC Desiar10", price: 25000, camera: 15, storage: 32 },
    { name: "Huwei A30", price: 17000, camera: 15, storage: 32 },
]
let cheapest = phones[0]
for (const ph of phones) {
    if (cheapest.price > ph.price) {
        cheapest = ph;
    }
}
console.log(cheapest);
```

String

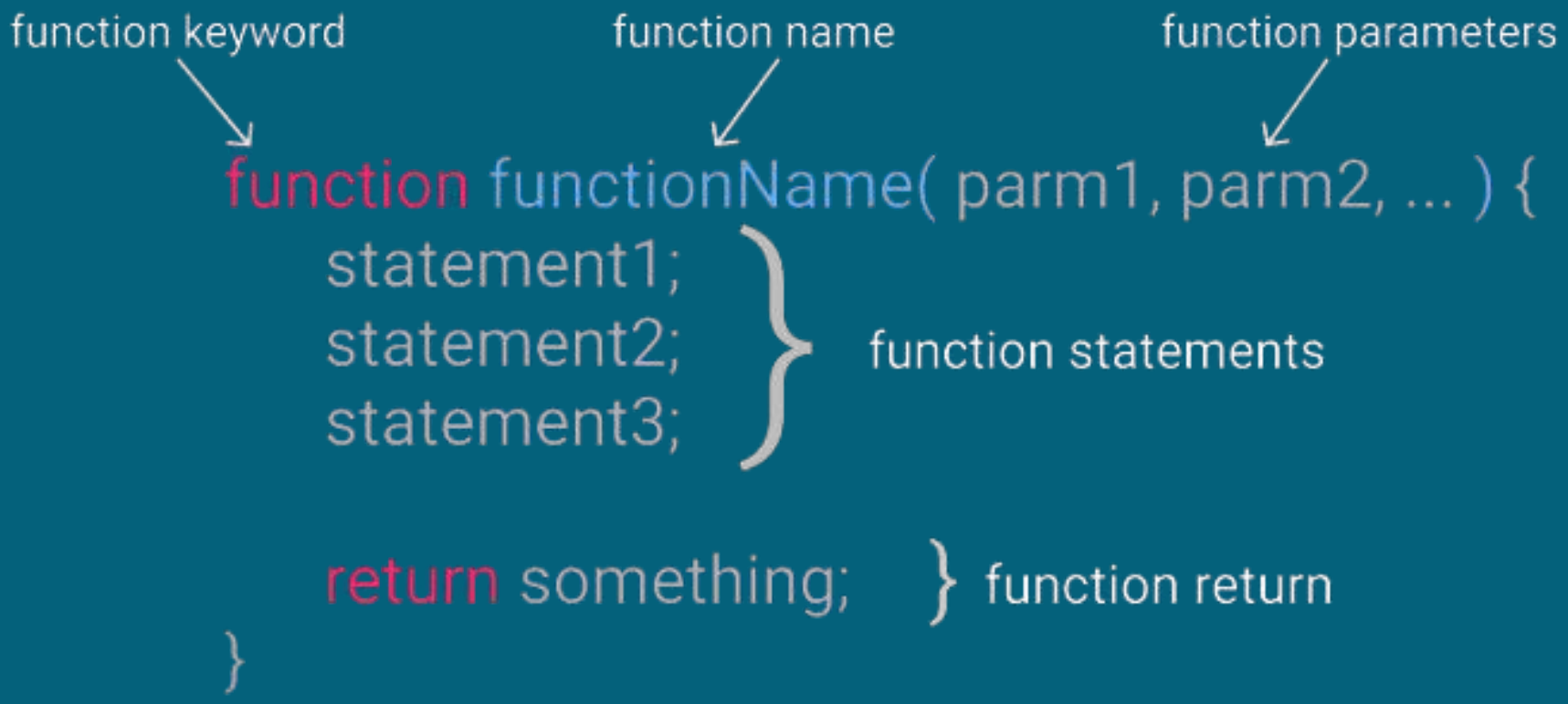
- In JavaScript, a string is a sequence of characters enclosed in single quotes (''), double quotes ("") or backticks (``).
- Strings in JavaScript have various built-in properties and methods that can be used to manipulate and work with them. Here are a few commonly used string methods:
- length: Returns the number of characters in a string.
- toUpperCase(): Converts the string to uppercase.
- toLowerCase(): Converts the string to lowercase.
- concat(): Concatenates two or more strings.
- slice(): Extracts a portion of a string.
- indexOf(): Returns the index of the first occurrence of a specified substring.
- replace(): Replaces a specified value or regular expression with a new value.

ES6

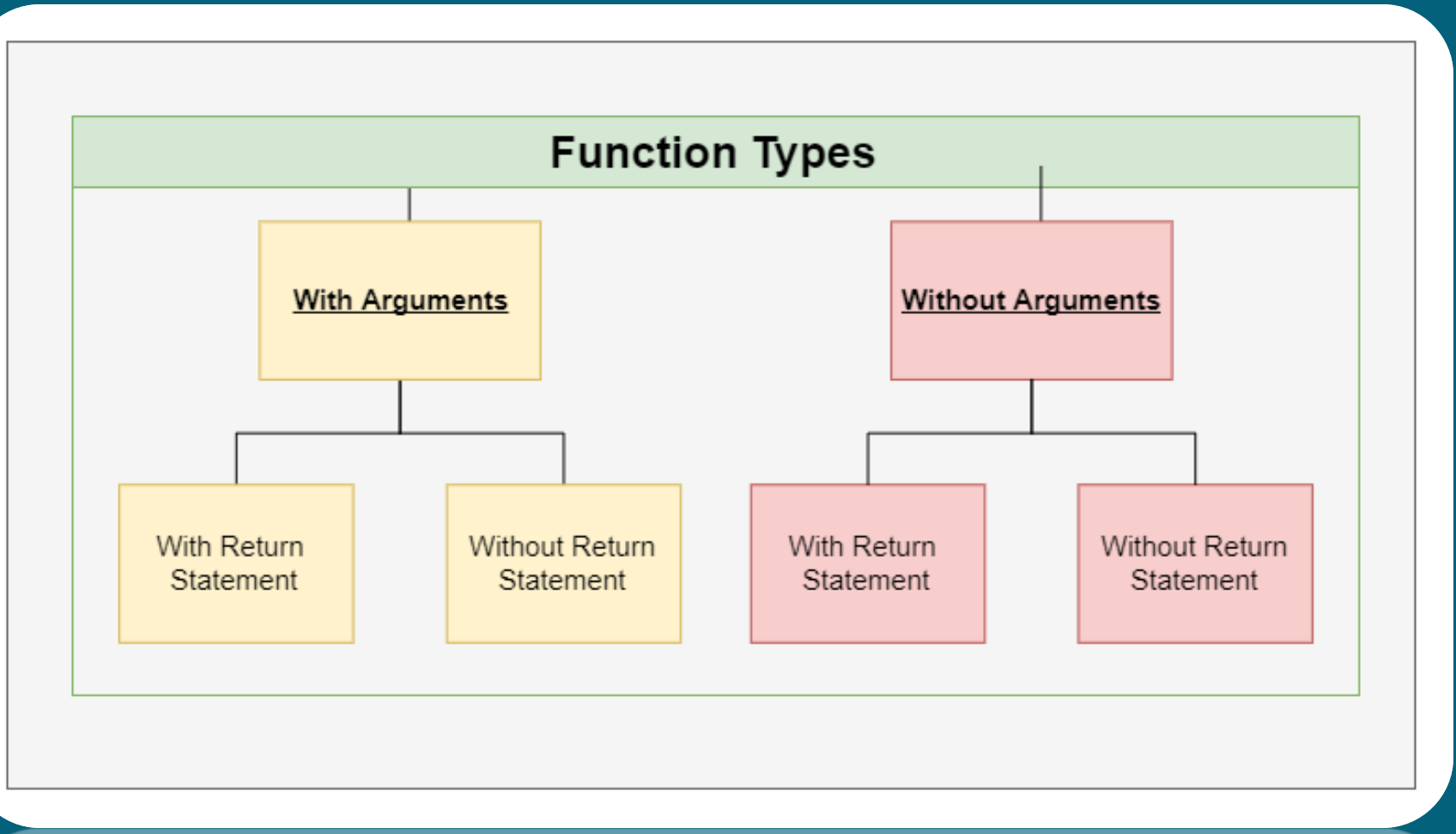
- The let keyword
- The const keyword
- Arrow Functions
- The ... Operator
- For/of
- Map
- Objects
- Classes
- Promises Default Parameters Function
- Rest Parameter
- String.includes()
- String.startsWith()
- String.endsWith()
- Array.from()
- Array keys()
- Array find()
- Array findIndex()
- New Math Methods
- Object entries

Function

- A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is executed when "something" invokes it (calls it).

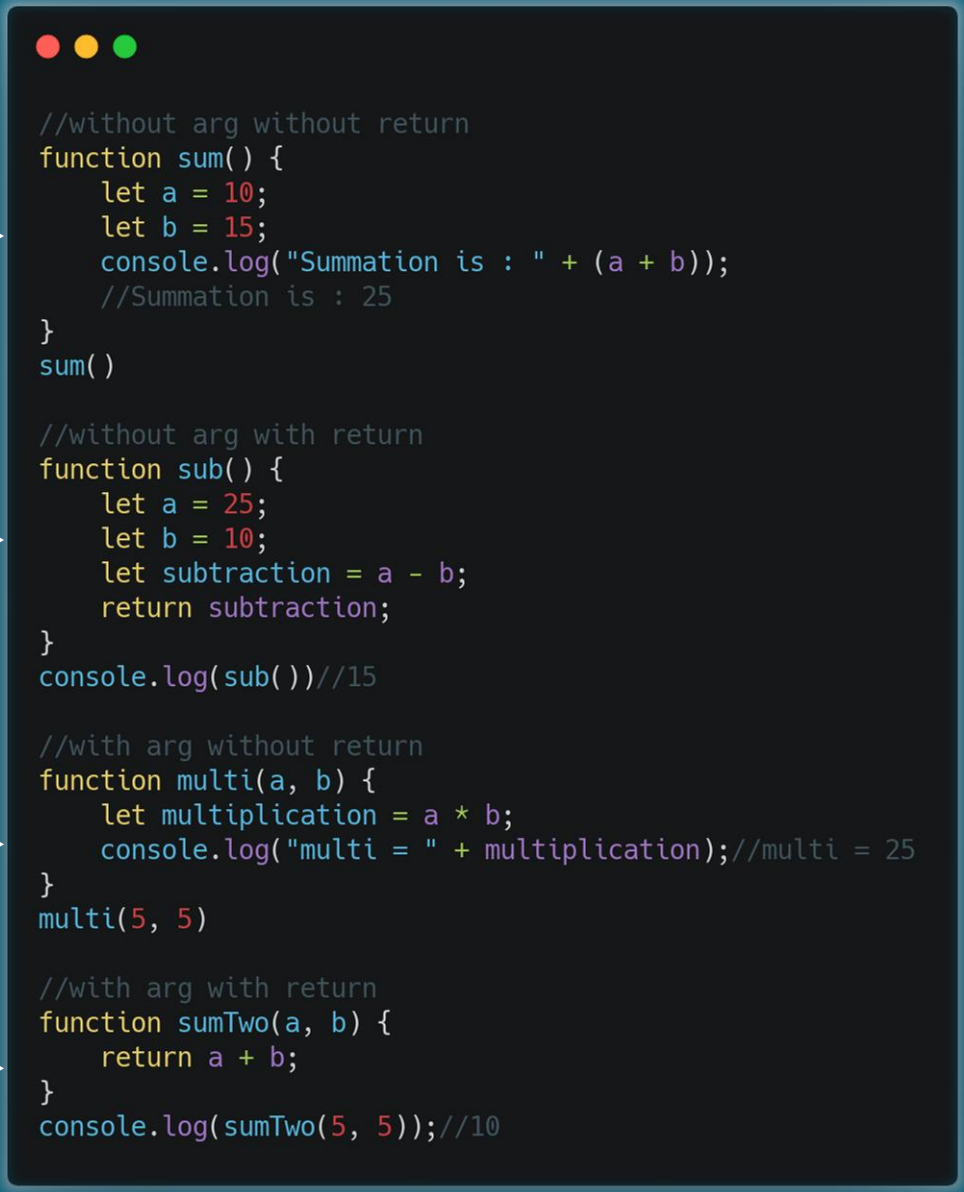


Function


Function Types
With Arguments
Without Arguments
With Return Statement
Without Return Statement
With Return Statement
Without Return Statement

Function

- No arguments without return value
- No arguments with return value
- Arguments without return value
- Arguments with return value



```
//without arg without return
function sum() {
    let a = 10;
    let b = 15;
    console.log("Summation is : " + (a + b));
    //Summation is : 25
}
sum()

//without arg with return
function sub() {
    let a = 25;
    let b = 10;
    let subtraction = a - b;
    return subtraction;
}
console.log(sub())//15

//with arg without return
function multi(a, b) {
    let multiplication = a * b;
    console.log("multi = " + multiplication);//multi = 25
}
multi(5, 5)

//with arg with return
function sumTwo(a, b) {
    return a + b;
}
console.log(sumTwo(5, 5));//10
```

Let’s Solve some Problems with function

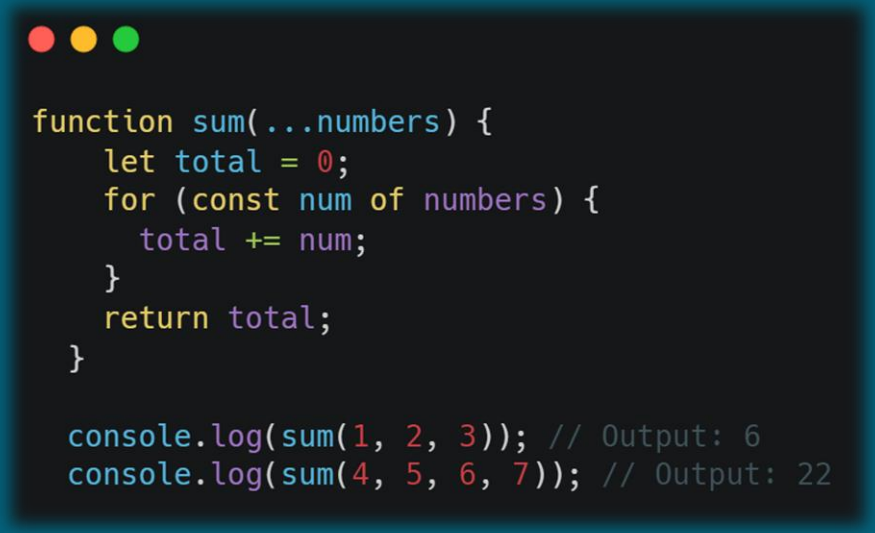
- 1- odd or even
- 2- convert inch to feet mile to kilometer
- 3- leap year
- 4- factorial of a number
- 5- find max of three values.
- 6- convert celsius to fahrenheit

Spread operator(...)

- The spread operator is denoted by three dots (...) and is used to expand an iterable (like an array or a string) into individual elements. It allows you to conveniently copy or combine elements from one array or object to another.
- some use cases of the spread operator:
  - Copying an array: You can create a new array by spreading the elements of an existing array.
  - Concatenating arrays: You can combine multiple arrays into a single array using the spread operator.
  - Passing arguments to functions: The spread operator can be used to pass an array of arguments to a function.

Rest Operator(...)

- The rest operator is also denoted by three dots (...), but it is used within function parameters to represent an indefinite number of arguments as an array. It allows you to pass multiple arguments to a function without explicitly defining them.



Closure

- A closure is created when you define a function inside another function. The inner function has access to its own local variables, parameters, and the variables of the outer (enclosing) function.

```
function outerFunction() {
    var outerVariable = 'I am from the outer function';

    function innerFunction() {
        console.log(outerVariable);
    }

    return innerFunction;
}

var closure = new outerFunction();
closure();

// Output: I am from the outer function
```

```
function stopWatch() {
    let counter = 0;
    return function () {
        counter++;
        return counter;
    }
}

let watch1 = stopWatch();
```

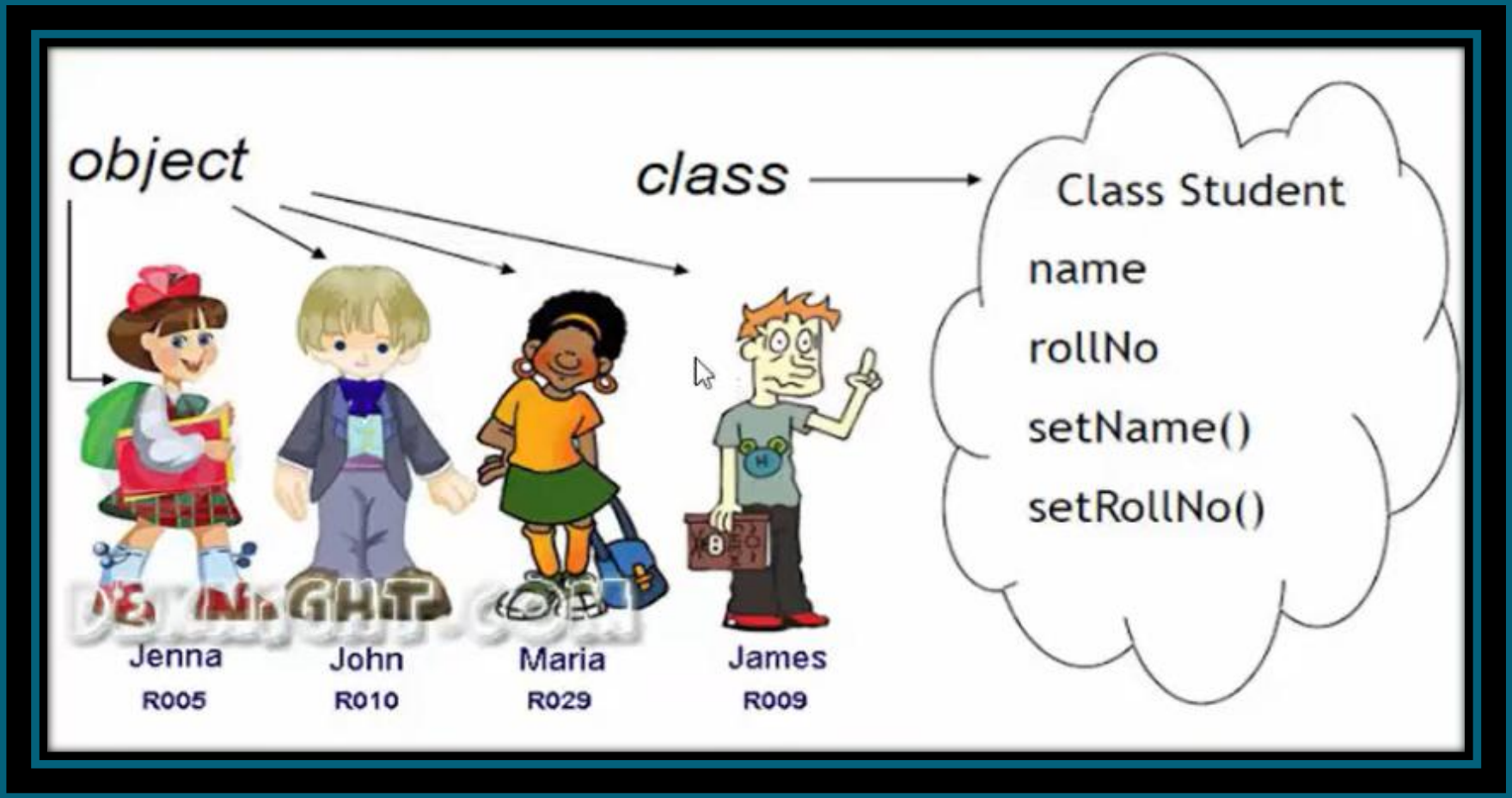
OOP

- O = Object
- O = Oriented
- P= Programming

OOP

Class and Object:

- class is a template from which individual objects can be created.
- Any class type of variable is called an object



```
//class and object

class Student {
    constructor(name, id) {
        this.id = id;
        this.name = name;
    }
    getDetails() {
        return (`Name is : ${this.name} Rool is ${this.id}`)
    }
}
let st1 = new Student("riyaz", "1920");
let st2 = new Student("shuvo", "1921");
console.log(st1.getDetails());
console.log(st2.getDetails());
//Name is : riyaz Rool is 1920
//Name is : shuvo Rool is 1921
```

OPP

Inheritance:

- Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.
- To create a class inheritance, use the extends keyword.

  The super() method refers to the parent class.

  By calling the super() method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods.

OOP

Inheritance:

- This is the real life example of inheritance is child and parents. All the properties of father are inherited by his son.

son I am

Base Class

Inheritance

Dad I am
Derive Class

superClass

subClass

```
// inheritance

class Car {
    constructor(brand) {
        this.carname = brand;
    }
    present() {
        return 'I have a ' + this.carname;
    }
}

class Model extends Car {
    constructor(brand, mod) {
        super(brand);
        this.model = mod;
    }
    show() {
        return this.present() + ', it is a ' + this.model;
    }
}

let myCar = new Model("Ford", "Mustang");

console.log(myCar.show());
//I have a Ford, it is a Mustang
```

OOP

Encapsulation:

- The word encapsulation means sensitive data is hidden from the user.
- Hiding a specific property or a method and making them accessible within the function or class and can't access outside of the class or function.
- The main advantage of encapsulation is to secure the data from other methods .

encapsulation

```
// Encapsulation

function ATM(pin, name, acNumber, withdraw) {
    let acName = name;
    let acNum = acNumber;
    let totalAmount = 50000;
    let remainingAmount = totalAmount - withdraw;
    let showUserInfo = function () {
        console.log("Welcome , " + acName)
        console.log("A/C : " + acNum);
        console.log("Name : " + acName);
        console.log("Total Amount : " + totalAmount);
    }
    if (pin == 1234) {
        showUserInfo()
        console.log("After withdraw you amount is : " + remainingAmount)
    }
    else {
        console.log("Please, Enter your correct Pin Number!");
    }
}
let user1 = new ATM(1234, "Riyaz", 65454, 5000);
// Welcome , Riyaz
// A/C : 65454
// Name : Riyaz
// Total Amount : 50000
// After withdraw you amount is : 45000
```

OPP

Polymorphism:

- The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.
- Note: Method overloading is not supported in javaScript.

OPP
Polymorphism

- Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter, Here one person have different-different behaviors.

In Shopping malls behave like Customer

In Bus behave like Passenger

In School behave like Student

At Home behave like Son

```
// Polymorphism

class Person {
    behave() {
        return ("Normal behave");
    }
}
class ShoppingMall extends Person {
    behave() {
        return (`${super.behave()}\nIn Shopping mall behave like a Customer`);
    }
}
class School extends ShoppingMall {
    behave() {
        return (`${super.behave()}\nIn School behave like a student`);
    }
}

let sh2 = new School()
console.log(sh2.behave())
// Normal behave
// In Shopping mall behave like a Customer
// In School behave like a student
```

OPP

Abstraction:

- Abstraction is the process of hiding the implementation details and showing only the functionality to the users.
- Hiding of data is known as data abstraction. In object oriented programming language this is implemented automatically while writing the code in the form of class and object.
- Abstraction and encapsulation is works together.

Asynchronous JavaScript

Synchronous JavaScript:

- every statement of the code gets executed one by one. So, basically a statement has to wait for the earlier statement to get executed.

Asynchronous JavaScript:

- Asynchronous code allows the program to be executed immediately where the synchronous code will block further execution of the remaining code until it finishes the current one

```
console.log('Hello');
console.log('World');

console.log('hello');
setTimeout(() => {
    console.log('after two seconds')
}, 2000);
console.log('end');

// Hello
// World
// hello
// end
// after two seconds
```

Callback function

- a callback is a function that you pass as a parameter to another function that function will do some work and then call that callback function.
- A callback is a function passed as an argument to another function.
- In the real world, callbacks are most often used with asynchronous functions.

```
//callback function

function Display(cal) {
    console.log(cal);
}

function myCal(x, y, callBack) {
    let sum = x + y;
    callBack(sum);
}

myCal(5, 6, Display)
```

```
setInterval(myFunction, 1000);

function myFunction() {
    let d = new Date();
    console.log(d.getHours() + ":" +
        d.getMinutes() + ":" +
        d.getSeconds())
}
```

JS Event

- HTML events are "things" that happen to HTML elements. When JavaScript is used in HTML pages, JavaScript can "react" on these events.
- Common HTML Events
- Here is a list of some common HTML events:
- Onclick
- Onmousehover
- Onmouseout
- Onkeydown

JS Event - Onclick

In the following example, an **onclick** attribute (with code), is added to a <button> element:

```
<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>
```

In the example above, the JavaScript code changes the content of the element with id="demo".

In the next example, the code changes the content of its own element (using **this.innerHTML**):

```
<button onclick="this.innerHTML = Date()">The time is?</button>
```

For more JS Event Please visit this link

https://www.w3schools.com/jsref/dom_obj_event.asp

JS DOM

- DOM = Document Object Model
- Dom is a system to access html through js. When our browser reads or sees the html file, it prepares the html file as an object in javascript so that we can access it in the future, change it, modify it, so that we can easily modify it in real time.

- Dom Project:
- Project-1
- Project-2
- Project-3