**What is javaScript**

* Popular web **Programming Language**
* Scripting language
* Lightweight
* **Cross-platform**
* Object-oriented syntax
* Run-on **browser**

**History**

In September 1995, a Netscape [programmer](https://www.springboard.com/blog/programmer-portfolio/) named [**Brandan Eich**](https://en.wikipedia.org/wiki/Brendan_Eich) developed a new scripting language in just 10 days. It was originally named Mocha, but quickly became known as LiveScript and, later, JavaScript.



**Before javaScript**

* HTML 5
* CSS 3
* Bootstrap 5
* Github

**Extension for VS Code**

* Auto Close Tag
* Auto Rename Tag
* Beautify
* Live Server
* Auto-Save on Window Change
* Auto import
* Path autocomplete
* ES7+ React/Redux/React-Native snippets
* Prettier-Code formatter
* Material Icon Theme
* Bracket Pair Colorize
* ESLint( For JS)
* JavaScript(ES6) code snippets
* Themes ( **Night owl** , dracula, monokai . . . )

**Dev fonts Download**

[Dev Fonts Download Link](https://drive.google.com/file/d/1Nq_WhC34hg5xJt949gCtQGmbPGEcMxSt/view?fbclid=IwAR2ZCDy6W5AeEwmb7vR5ezHfDi3OX-RFqSJbntRJ5HoWv-WNAxkzLzEV8Bw)

**Environment Setup**

* Install VS Code and configure
* Install Node JS
* Install git

**Start**

* Script Tag
* Internal
* External
* Inline

**Dev Tools**

* Console log
* Use of console
* Code run on terminal

**Basic BOM Function**

* Alert
* Confirm
* Prompt

**Data Types**

* String
* Number ( int , float )
* Boolean
* Array
* Object
* Undefined
* Null

**Statement & rules**

* In a programming language, instructions ( lines of code ) are called statements.
* put a semicolon after a complete statement
* also, you can avoid semicolon in JS
* two or more words are joined by using concatenation ( + )

**JavaScript Variables**

* Var
* Let
* Const
* **Rules for variables**  
  - Names can contain letters, digits, underscores, and dollar signs.   
  - Names must begin with a letter  
  - Names can also begin with $ and \_   
  - Names are case sensitive   
  - Reserved words cannot be used as names

**Template literal Syntax**

* Template literals are literals delimited with backticks (`), allowing embedded expressions called *substitutions*.
* There are 2 types of templates literal   
  - untag template literal   
  - Tag template literal
* Interpolation variables and expression - ${ var / ex }

**Comments**

* Make a comment by using //
* Line comment
* Multiline comments
* Doc block for documentation

**Operators**

* Arithmetic   
   +, -, \*, / , %, --, ++
* Assignment   
  =
* Comparison   
   == === < > <= >= != !== ?
* Logical   
   && - AND || - OR ! - NOT
* String   
   + +=
* Bitwise   
   & ~ ^ | << >> >>> <<<
* Special ( type operator )  
  (?:) , delete - in - instanceof - typeof - new - void - yield

**Conditional Statement**

* if
* else
* else if
* switch case
* Ternary operator ( condition ? true return : false return )
* nullish coalescing operator ( return the value ?? return something if the value null or undefined )
* undefined, null, and empty value of a variable
* parseInt, Number, parseFloat, + for the string to number conversion
* Truthy & Falsy Values ( undefined, null, empty, 0, NaN, false )
* == === concept

**Loops**

* For loading some statement a number of times
* **loop structure**   
  -> initial value   
  -> condition / end step   
  -> operators ( ++, -- )
* **Loop Statement**   
  -> for   
  -> while / do while
* **For Loop**  
  for( let i = 0 ; i <= 10 ; i++ ){  
     
  }
* **While & do while**   
  let i = 0;  
    
  while( i <= 10 ){  
     
   (++/--)  
  }

**User Defined Function**

* To avoid repeating the same code
* Create a complex functionality for use
* Declare one-time use many times
* The application will be scalable and clean
* **Declare a function**   
  to declare a function use the **function** key and then put the name of the function   
    
  **function** functionName () {   
    
    **function output**  
  }
* **Invoke a function**When we call a function then it is called **function invoke.** After declaring a function now it’s time to use this function. Just call this function like this **functionName**();  
    
    
  **Arguments & Parameter   
  functionName**(argument1, argument2, . . . . );  
  **function** functionName (parameter1, parameter2, . . . . ) {   
    
    **function output**  
  }
* **function Expression   
  let** functionName = **function** (parameter1, parameter2, . . . . ) {   
    
    **function output**  
  }

* **Arrow function**   
  **let** functionName = (parameter1, parameter2, . . . . ) => {   
    
    **function output**  
  }

**Arrow function Details**

* arrow function syntax   
  let **function\_name** = ( **param1, param2**, . . . ) => {  
   **output / functionality**   
  }
* **Single** **line arrow** function   
  let **func\_name** = ( **p1, p2**, . . . ) => *return* **output**
* **single parameter arrow** function   
  let **func\_name** = ( **p1** ) => **output / functionality**let **func\_name** = **p1**  => **output / functionality**

**Constructor Function**

* The **leader** of the function
* Many functions & variables live under the constructor function
* Just call the leader then you will call all the sub function
* syntax of **constructor function**   
  function **FunctionName**(){  
   **output / functionality**}
* constructor function **expression**let **FunctionName =** function(){  
   **output / functionality**}
* Sub functions in constructor function   
  let **FunctionName =** function(){  
   this.**variables1** = ‘ value1 ’;  
   this.**variables2** = ‘ value2 ’;  
   this.**variables3** = ‘ value 3’;  
    
   this.**fucntion1\_name** = function(){  
   **output / functionality** }  
    
   this.**fucntion2\_name** = function(){  
   **output / functionality** }}
* Call the **constructor function / instance**   
  let **lead\_name** = new **FunctionName**;
* Call the sub functions and variables from **constructor function after instance**let **lead\_name** = new **FunctionName**;   
    
  **lead\_name.variable1;  
  lead\_name.variable2;  
    
  lead\_name.fucntion1();  
  lead\_name.fucntion2();**

**Project Work**

* Create a **Utility** **Constructor** function
* Complete result system with **Result constructor**

**Array ( Data Structure )**

* Used more than one value or can be more than one value
* The best way to decorate data for future
* Any type of data can be stored in array
* **declare a array**const **array\_var** = [v1, v2, v3 . . . . ];
* **Create some array data structure**   
  -> 10 flowers of Bangladesh   
  -> 10 Rivers of Bangladesh   
  -> 10 fish of Bangladesh   
  -> 10 Birds of Bangladesh   
  -> 10 vegetable of Bangladesh
* **Get value from an array   
  array\_var**[ index\_number ]
* Get **array** length   
  **array\_var**.length
* Get all **array** value by **for loop   
  for**( let i = 0; i < **array\_var**.length; i++ ){  
   *return* array\_var[ **i** ];  
  }
* Get all array data by **forEach** & **Map ( iteration )  
  array\_var.**forEach( function(**data**){  
   return **data;**   
  })  
    
  **array\_var.**map((**data**) => {  
   return **data;**   
  })
* Create an array by using **Array** Constructor   
  new **Array**(item1, item2, . . . )
* **Array to string conversion**-> **toString**  
  -> **join**  
  -> **split**
* **Array Add & Remove**-> push  
  -> pop   
  -> shift  
  -> unshift   
  -> slice  
  -> splice   
    
    
    
    
    
  **Array methods & Uses**   
  -> concat()   
  -> reverse()  
  -> sort()   
  -> filter()   
  -> reduce()  
  -> every()   
  -> some()   
  -> indexOf()  
  -> lastIndexOf()  
  -> find()   
  -> findIndex()  
  -> from()   
  -> keys()   
  -> includes()   
  -> isArray()  
  -> valueOf()
* **Multidimensional Array**   
  [  
   [item1, item2, . . . . ],  
   [item1, item2, . . . . ],  
   [item1, item2, . . . . ]  
  ]
* **Create a student Data Structure**
* **Create a developers team member data structure**

**Object Data Structure**

* A complete Data structure will made by **array** and **object**
* In array data structure we face some problem like **data key**
* But with array and object data, we can build a complete structure for the future.
* **Declare a object data structure**   
  const **obj\_name** = {  
   name : ‘Asraful Haque’,  
   age : 10,  
   skill : ‘Laravel Developer’  
  }
* **Declare a object data structure with new Object**   
  const **obj\_name** = **new** **Object**({  
   name : ‘Asraful Haque’,  
   age : 10,  
   skill : ‘Laravel Developer’  
  })
* **Get data form an object data structure**   
  -> By dot notation **obj\_name**.property\_name;  
  -> By array notation **obj\_name[‘**property\_name**’];**
* **Create a Complete Array and Object Data structure**const **obj\_name** = [  
   {  
   name : ‘Asraful Haque’,  
   age : 10,  
   skill : ‘Laravel Developer’  
   },  
   {  
   name : ‘Asraful Haque’,  
   age : 10,  
   skill : ‘Laravel Developer’  
   },  
   {  
   name : ‘Asraful Haque’,  
   age : 10,  
   skill : ‘Laravel Developer’  
   }  
  ]
* **Fetch all Student data by loops**-> for   
  -> for in   
  -> for of   
  -> forEach   
  -> map
* **For in and For of loop   
  for (** datain **array )**{*return* **data;**} **for (** dataof **array )**{*return* **data;**}
* Create a complete **Developer array and object data structure** with monthly **income**. And search devs by stack, location, income, age
* Create a complete fifth class students **array and object data structure** and find their result with gpa, grade, cgpa and final result

Day 08

**Date Object**

* For date & time management
* Date object has a constructor
* Declare a Date object   
  let **date** = new **Date**();
* Get date information   
  -> **date**.getDate()   
  -> **date**.getMonth()   
  -> **date**.getFullYear()   
    
    
  -> **date**.getHoures()   
  -> **date**.getMinutes()   
  -> **date**.getSeconds()   
  -> **date**.getMilliseconds()   
    
  -> **date**.getTime()
* **Date formates**   
  -> 2021-12-07 => ISO   
  -> 07/12/2021 => short   
  -> December 7, 2021 => Long
* **Find Today form Date Object**
* **Find current month name from JS object**

**Math Object**

* Math object has no constructor
* It is static / you don't need to create any instance
* **Math Property**   
  - **Math**.PI

- **Math**.E

- **Math**.SQART2

- **Math**.SQUART1\_2

- **Math**.LN2

- **Math**.LN10

- **Math**.LOG2E

- **Math**.LOG10E

* **Math methods**

- **Math**.abs()

- **Math**.ceil()

- **Math**.floor()

- **Math**.round()

- **Math**.max()

- **Math**.min()

- **Math**.sqrt()

- **Math**.pow()

- **Math**.random()

**String Object**

* The string can be an object  
  **new String()**but do not use this, please
* **String property**- constructor  
  - length
* **String Methods**

- concat  
- startWith

- endsWith

- includes

- indexOf

- lastIndexOf

- repeat

- replace

- search

- slice

- split

- substr

- toUppercase / Local

- toLowercase / Local

- toString

- trim

**Number Objects**

* Developer can create a number object by using this  
  **new Number()**but do not use this

**Booleans Objects**

* True / False
* **Declare a Boolean Object**-> new Boolean(true/false);  
  but do not create a Boolean object by new key just use **true** or **false**
* **Truthy & Falsy**Undefined  
  Null  
  0  
  False  
  Empty

**Type Conversion**

* **String to number**- Number()  
  - parseInt()  
  - parseFloat()  
  - unary + operator
* **Number to string**- String()  
  - toString()  
  - toExponential()  
  - toFixed()  
  - toPrecision()
* **Date to number**- Number()  
  - getTime()
* **Date to String**- Sting()  
  - toString()
* **Boolean to number**- Number()
* **Boolean to String**- String()

Day 09

**JSON Data**

* **JSON** stands for **JavaScript Object Notation**
* **The lightweight** data-interchangeformat
* **Language** independent
* Easy to **understand** and **self-describing**
* **JSON** is a text format for **storing** and **transporting** data
* **JSON** helps to **convert array** and **object** data to a string format for devs-friendly data use.
* **JSON Server** for apps
* **Declare a JSON**

- It looks like an **object**

- Data is **named/Value** pairs

- Data is separated by a **comma**

- **Curly** braces hold the object

- Square brackets hold arrays  
  
{  
 “name1” : “value1”,  
 “name2”: “value2”  
}

* **JSON data types**

- string

- number

- object

- array

- Boolean

- null

* **JSON values cannot be one of the following types**

- function  
- date  
- undefined

* **JSON.parse()**  
  - to convert **JSON** data string to **object**  
  - to convert an **array** string to an **array**
* **JSON.stringify()**- to convert an **object** data to a **JSON string**  
  - to convert an **array** to **JSON string**
* **JSON file**We have to create a JSON file by setting.json   
  **db.json / api.json**

**Errors handling**

* To Handle errors in a **custom way**
* Prevent apps **crashing** for an error
* **Try Catch Finally**

-> Try  
-> Catch   
-> Throw   
-> Finally   
  
**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*    
}

**Local Storage 10**

* Browser storage for temporary data
* Send data to LS   
  -> **localStorage**.setItem( ‘key’, ‘value’ );
* Get Data from LS   
  -> **localStorage**.getItem(‘key’);

**Session Storage**

* Browser storage for temporary data
* Send data to SS   
  -> **sessionStorage**.setItem( ‘key’, ‘value’ );
* Get Data from S   
  -> **sessionStorage**.getItem(‘key’);

**Cookie Storage**

* Cookies are data, stored in small text files, on your computer
* It is used to remember a user from the browser
* **Send data to a cookie   
  document**.**cookie** = “name = data ; expire ; path =/ ”;
* **Get Data from cookie**   
  let **cookie\_data** = **document**.**cookie ;**

**Regular Expression**

* A regular expression is a sequence of characters that forms a search pattern
* A regular expression can be a single character or a more complicated **pattern**
* **Syntax**/ pattern / modifier
* **Modifier**- /i ( case insensitive )  
  - /g ( global Search )  
  - /m ( multiline search for the match )  
  - / ( empty modifier is case sensitive )

* **Methods**- exec ( check data is in an array or not )  
  - test ( return true or false for data check )  
  - match ( check the match is in or not )  
  - search ( search the index number of pattern )  
  - replace ( replace words of a string )
* **Literal character**- all regular character is a literal character
* **Meta Character  
  - ^** ( start with the character )  
  **- $** ( ends with character )  
  **- .** ( any character length will be one )  
  **- \***  ( any character length one to more )  
  **- ?** ( set optional character by using this key )  
  **- [abc]** ( character group )  
  **- [^abc]** ( except those character )  
  **- [A-Z][a-z][0-9]** ( uppercase, lowercase and number )  
  - abc{2} ( quantifier for repeat character )  
  - () ( for creating group )  
  - \w ( alphanumeric word character )  
  - \W ( non-word character )  
  - \d ( digit character )  
  - \D ( non-digit )  
  - \s ( white space )  
  - \S ( non-white space )  
  - \w ( word boundary )  
  - \a(?=b) ( condition 1 )  
  - \a(?!b) “