JavaScript

1. it is a object oriented programming language
2. it is a scripting language.
3. it is also a functional programming language.
4. it is used for adding the dynamic behaviour(e.g clicking a button in a website)
5. it is a loosely typed language i.e variable type can be changed
    - there is no type checking
    - datatypes are inferred
    - datatypes are dynamically assigned
6. it does not support pointers

Browser components


Browser Architecture

1. **Network C**:

- is responsibe for sending the request and recieving response

2. **Layout Engine C**:

- is responsible to convert html and css to JS objects,
- is also known as a rendering engine .

3. **JavaScript Engine**:

- is responsible for executing the JS code
- the most important component of any browser

4. **User Interface Component**

- is responsible for displaying output to the user

5. **Web Storage (Data Store)**

- is also known as data store
- is responsible for storing the data
- e.g. history,session,cookies
- e.g of browser:
- google chrome
- rendering engine: blink
- javascript engine : **V8**

- mozilla
- rendering engine: Gecko
- javascript engine : SpiderMonkey
- apple Safari
- rendering engine: webkit
- javascript engine : NitroJS
- Opera
- rendering engine: blink
- javascript engine : -
- MS IE/Edge
- rendering engine: EdgeHTML
- javascript engine : Chakra

JS Fundamentals

- **Rules**
  - the semicolon(😉 is optional,when one statement is written on one line

```
        console.log('this is a JS tag in head scetion')
        console.error('this is an error')
```

  - semicolon is required when multiple statements are written on one line

```
          console.log('this is a JS tag in head scetion');
console.error('this is an error')
```

- conventions - use camel case when declaring function,variable or classes - i.e. start the name with lower case and upper case for first letter of meaningful word - e.g.
  - firstName
  - printPersonInfo()

0. **Pre-Defined Object**

- 1.**console**
- object that represents the browser console (in browser/more tools/dev tools/console)

- 2.**methods**
  - log(): used to print debugging message on the browser's console
  - error() : used to print an error on the browser console
  - info() : use to print info message on the browser console
  - warn() : used to print warning mwessage on the browser console
- 3.**window**
  - object that represents the browser's window (UI)
  - this is a default object used whn calling a method
  - methods
    - alert() :
    - prompt() :
    - confirm():

4. **pop ups**

- alert() :
  - used to show a message to the user on web browser window
  - e.g.

```
//window.alert('this is a alert')
alert('this is an alert')
```

- prompt() :
  - used to take input from user
  - e.g.

```
const userName = window.prompt('Enter your Name')
console.log(userName)
//document.write(userName)
```

- confirm():
  - used t get input in terms of boolean answer to a question
  - e.g.

```
const answer = window.confirm('do you want a break?')
 if(answer)
 {
     console.log('let take a break of 10 min')
 }
 else
```

```
        {
            console.log('continue ')
        }
```

0.1 **Pre-Defined Values**

- 1.**undefined**
- 2.**Nan**
- Not a Number
- NaN has data type as number
- does not represent a valid number

```

```

- 3.**Infinity**
- has a datatype a number
- e.g

```
console.log(`10 / 0 = ${10 / 0}`)
```

1.1 **Variables**

- is a placeholder to store a value in memory
- it is a mutable value
- to declare a variable use "**let**" keyword
- variable must be declared without a data type
- e.g

```
        let FirstName = "Steve"
        let Salary = 10.60
        let age = 40
```

1.2 **Constant**

- is a placeholder whose value *cannot* be changed
- it is immutable value(readonly)
- to declare a constant use "**const**" keyword
- e.g

```

```

```
const pi = 3.14
// cannot update the value
// pi = 100
```

- > ALWAYS PREFER CONSTANT THAN A VARIABLE

2. **Data Types**

- all data type in JS will be inferred
- the datatype will be decided by JS by inspecting the current value in the variable
- **types**

    1. **number**
    - represents whole number and floating point/decimal numbers
    - e.g.

```
//number
 let num = 100
 console.log('num ='+ num)
 console.log('data type of num = ' + typeof(num))

 // number
 let salary = 10.50
 console.log('datatype of salary =' + typeof(salary));
```

    2. **string**

    - collection of characters
    - string can be declared using
    - single ('..')
    - double ("..")
    - backspace (`) for multiple lines
    - e.g.

```
 // string
 let firstName = 'steve'
 console.log('firstName = ' + firstName)
 console.log('datatype of firstName ' + typeof(firstName))
```

```
//

 let address = `address line 1,
                address line 2,
                address line 3 `
 console.log('address = ' + address)
 console.log('datatype of address =' + typeof(address))
```

3. **boolean**
   - represent only true or false values
   - e.g

```
//boolean
  let canVote = true
  console.log('canVote = '  + canVote + typeof(canVote))
```

4. **undefined**
   - in JS undefined is both : datatype as well as pre-defined value
   - represents a variable without an initial value
   - e.g.

```
  //undefined

  let myvar
  console.log('myvar = ' + myvar) //undefined
  console.log('datatype of myvar = ' + typeof(myvar)) //undefined
```

5. **object**
   -

3. **Statements**

- the smallest unit that executes
- types - assignment s - declaration s - comment s

4. **Operations**

- mathetical Operator
- addition(+) - the plus operator works as - mathe addition when both param are numbers

```
```javascript
console.log(10 + 20)
```
```

```
- string concatination operator when one operand is a string
then params get changed to string
```javascript
      //test1test2
      console.log(`test1` + `test2`)
      //1050
      console.log(`10` + `"50"`)
```
```

- substraction (-)
- division (/) -
- multiplication (*) - mathematical multiplication of two operands - the answer is always a **number** - e.g

```
 //200
 console.log(10*20)
// NaN
console.log(`test1`*`test2`)

//400
console.log(`10` * 40)
//400
console.log(`10`*`40`)
```

- modulus (%)
- relational/ comparison Operator
    - double equals to (==)
        - also known as value equaity operator
        - checks ONLY values of operands
        - e.g

```
//true
console.log(50 == 50)
// '50' is a string
console.log(50 == '50')
```

    - triple equals to (===)
    - also known as identity equality operator
    - checks both the value as well as the data type of the operands
    - always prefer === over ==
    - e.g.

```
//true
```

```
console.log(50 === 50)
//false as datatype is also checked
//'50' is a string  and 50 is a number
// both need to be of same datatype for triple equal to be equal with
same value
console.log('50' === 50)
```

  - not equals to (!=) : only value
  - not equal equal to (!==) : value with datatype
  - less than (<)
  - less than equals to (<=)
  - greater than (>)
  - greater than equals to (>=)

- logical Operator
  - and (&&)
  - OR (||)

5. **predefined Function**

- 1. typeof() - used to get the datatype of a variable
- e.g

```
let myvar
console.log('myvar = ' + myvar)
console.log('datatype of myvar = ' + typeof(myvar))
```

6. **Type Conversion**

- 1.string to number
  - parseInt
  - convert string to integer number(no decimal value)
  - e.g.

```
//10
console.log(parseInt('10'))
```

  - parseFloat
  - convert string to integer number(with decimal value)
  - e.g.

```
//10.60
console.log(parseFloat('10.60'))
```

- 2.anything to string
- any datatype can be converted into string by concatinating with plus operator
    - e.g.

```
//'10'
console.log(10 + '')
```

7. **Function**

- a block of code which can be reused
- i.e a reusable block of a code having a name
- types
    - Empty function
        - function with no code in the body
        - e.g

```
//empty function
function fun1() {}
```

    - parameterless function
        - which does not accept any parameter
        - e.g

```
//para-less function declaration
function fun1()
{
    console.log('inside function')
}
//function call
fun1()
```

    - parameterized function
        - which accepts at least one parameter

- e.g

```
 function fun2(param)
 {
     console.log('inside the function 2')
     console.log(`param = ${param}, type of param =
${typeof(param)}`)
 }
  //function call
 fun2(10)
 fun2(10.60)
 fun2("test")
 fun2("test + 10")
 fun2('10' + 10)
 fun2(true )
```

- **variable length argument function**
  - a function which can accept varible length of arguments
  - every function in JS recieves hidden parameter named arguments
  - which contains a list of all the argumetneds passed while function call
  - e.g.

```
   //variable length argument function
    function add()
    {
        console.log('inside add')
        console.log(arguments)
        let sum = 0
        for(let index = 0;index < arguments.length;index++)
        {
         sum += arguments[index]
        }
        console.log(`addition = ${sum}`)
    }
    //actual parameter or arguments
    add(10,20)
    add(10,20,30,40,50,60)
```

- **parameters:**
  - the parameter(s) of a function can not have static datatype(s)
  - the number and type of parameters will be controlled by the caler instead of the function
  - a caller can pass
    - same number of param
    - less number of param thqan expacted
      - missing argument will be treated as undefined

    - more number of param than expected
        - extra parameter will be discarded
    - every function in JS recieves two hidden parameters
        - **arguments** : used to get all the arguments in an array
        - **this** : used to point the current object
            - **this points to window in a function inside javascript in html**
            - **this point to Object if called outside the html**
    - e.g

```
 function add(number1, number2)
  {

      console.log(`number1 = ${number1}, type =
${typeof(number1)}`)
      console.log(`number2 = ${number2}, type =
${typeof(number2)}`)
      const answer = number1 + number2
      console.log(   ` ${number1 } + ${number2} =   ${answer}`)
  }

  add(10,20)
  add(10)
  add()
  add(10,20,30)
```

- default parameters
- also known as optional parameters as you may not pass the value to these parameters - one or more parameters may be declared with a default value - the default value will be used if the caller has not passed the argument for the optional parameters

```
- e.g
```javascript

function multiply(num1,num2=20)
    {
        const multiplication = num1 * num2
        console.log(`multi = ${multiplication}`)
    }
    multiply(40,100)
    // num2 = 20 here
    multiply(40)


```
- **function alias**
 - given a function another name
 - same function can be called with original function or function alias
```

```
- e.g.
 ```javascript
        function fun1()
        {

        }
        const myfun = fun1
        myfun()

 ```
```

---

8. **Collection**

- collection of values (similar to array of values in C and C++)
- properties
    - 1.**length**
    -
- method
    - 1.**push**
    - used to append a value at the end of the collection
    - e.g

```
      const numbers = [10,20,30]
       numbers.push(40)
```

    - 2.**pop**
    -

9. **html + JS**

JS as Functional programming language

- in JS, function is considered as first class citizen
    - function can be **called by passing another function as an argument**
    - a variable can be created for a function (**function alias**)
    - a function can be considered as a variable
    - one function can be considered as a **return value of another function**

OOP JS

- JS is object oriented programming language
- **alan key** coined OOP terms
- JS created by **brendan eich**
- object can be created by four ways

1. using **Object**

2. using **constructor functions**
3. using **JSON**
4. using keyword **class**

object

- similar to instance of a class in other language
- collection of key-value pairs
- kay also known as property, so also property -value pairs -e.g

```
    const person  = {
     name: 'person1',
     age : 40
 }

  // properties => name,age
// values => person1,40
```

- to access value of a property
- use subscript ([] syntax)
- used when a property is having special character like space
- e.g.

```
const p1 = {name: 'perosn1', email: 'perosn1@test.com', age: 40}
   console.log(`name : ${p1['name']}`)
   console.log(`email : ${p1['email']}`)
   console.log(`age : ${p1['age']}`)
}
```

```
const person = {
 'first name' : 'steve',
 'last name' : 'jobs
}
```

- use dor (.) syntax
- e.g

```
    const p1 = {name: 'perosn1', email: 'perosn1@test.com', age: 40}
     console.log(`name : ${p1.name}`)
```

```
        console.log(`email : ${p1.email}`)
        console.log(`age : ${p1.age}`)
```

3. ***JSON**

- javascript Object Notation
- way to create an object
- JSOn supports
- object
- collection of key-value pairs
- e.g.

```
function fun1()
{
   const person  = {
       name: 'person1',
       age : 40
   }

   console.log(person)


}
```

- **array**
  - collection of objects -e.g

```
      function fun2()
   {
   const persons = [
       {name: 'person1',age : 40 },
       {name: 'person1',age : 40 },
       {name: 'person1',age : 40 }
        ]
  console.log (` person = ${persons} type - ${typeof(persons)}`)
    }
```

creating object using **O**bject

- Object is a root function provided by JS
- everything in JS is a Object

node

module

- any js file having extension .js
- the NodeJs embeds an object named module in every mudule to present the current module
- the module object has following properties
    - id: the identification of the module
    - path: the file which is representing this module
    - exports:
        - contains the list of function/variable/constant which can be exported from the current module

HTTP Request (request) by client

1. **HTTP Method - Operation**

- GET - get data
- POST - insert data
- PUT - update the data
- DELETE - Delete data

2. **path** = URL : of the page

HTTP Request , server side what is done

1. **HTTP Method - Operation**

- GET - select * from ...
- POST - insert into ...
- PUT - update ...
- DELETE - delete ...