JavaScript

Introduction to JavaScript

JavaScript is a single threaded language that is used to build a strong backend. it supports both synchronous and asynchronous operations. Synchronous means executing instruction sequentially like traditional way. Asynchronous means executing in such a way that executing one instruction doesn't affect the execution of another instruction.

It helps us build a strong backend which help user interact with more effectively and going good.

JavaScript has many traditionally feature like operator, variable, flow, and function with some new concept such as callbacks, promises, new kind of loops and more advanced topics.

Tokens

variable : Variable are the basic entity of a program which holds a value that can be used when needed in the program.

operator : Operators are the special symbol which have an specific meaning or say role to play in program such as '+' is used to add two operands,

'==' is compare two operands,

'&&' performs logical operations on operands.

We have different operator available in JS which can be used to manipulate and operate the operands.

1. Arithmetic operators,
2. Relational operators,
3. Logical operators,
4. Assignment operators
5. Equality Operators.

Constants : Constants are the value that can never be change throughout the program. i.e. string constant, number constant, etc.

1. string constants : 'Hello world', 'Ch'.
2. bool constants : true, false.

3. number constants : 1, 1.43, 1.323E12 .

Keywords : Keywords are the special symbol which have a predifined meaning in our programing, for example

1. function : is used to create a function.
2. if : is used for conditional statement.
3. try- catch : are used for error handling tasks.

Identifier : identifier are the set of words that are as an identity in our program, such if we create a function named add(), then throughout the program if we want to access the functionality of add we got to call add() every time.

Special symbols : Special symbols are the symbol which are assigned an special meaning. for example,

1. ‘,’ : is used to separate
2. ’’ : used to create an string
3. () : to create a function
4. ‘;’ : used as a terminator
5. ‘:’ : used in key- value pair.

Advance Concepts

Callbacks

Callbacks are the function which takes a parameter that is an function itself.

for example

```
// This is an example of callbacks
settimeout(()=>{
console.log('hello world');
}, 2000)

arr = [1,2,34,3]
arr.forEach((val) => {
```

```
  console.log(val*val);
})
```

Callback hell

Callback hell occurs when one callback depends upon the execution of another callback and wait till the deeper hierarchical callback gets executed.

```
graph TD
  Callback1[parent callback] --> |execute|Callback2[child1 callback]
  Callback2 --> |execute|callback3[child2 callbakc]
  callback3 --> |result|Callback2
  Callback2 --> |result| Callback1
  callback3 --> |execute| callback3
```

Code of callback hell

```
function get(){
    setTimeout(()=> console.log('get called'),5000)
    // console.log('get called');
}
let arr = function now(ele,callback){
    setTimeout(()=>{
        console.log("now function called...");
        console.log(ele);
        if(typeof callback == 'function') {
            console.log('calling argument function...')
            callback(ele,get)
        }
    },5000);
}

function secondCall(i,callback){
    setTimeout(()=>{
        console.log(i-3)
        if(typeof callback == 'function'){
            console.log('calling next argument function')
            callback()
        }
    },5000)


}
arr(5,secondCall)
```

Promise

Promise are the advance concepts of JavaScript which helps us handle errors in better ways. We use promise keyword to create a promise that has two parameter one is resolve and other is reject.

for example :

```
let promise  = new Promise((res, rej)=> {
  if(val > 10){
        res("good choice.");
  }
  else{
      rej("greater value required");
  }
});

promise(5).then(mes => console.log(mes)).catch(err=> console.log(err));
```

Promise Chaining

Promise chaining is the concept where result of one promise depends on the executing of another promise.

Flow of promise chaining.

```
graph TD
  Promise1[Parent Promise] --> |execute|Promise2[Child Promise]
  Promise2 --> |execute|Promise3[Child Promise]
  Promise3 --> |execute|Promise3
  Promise3 --> |result|Promise2
  Promise2 --> |result|Promise1
```

For example:

```
let promise= new Promise((resolveOuter)=>{
    return resolveOuter(new Promise(resolveInner =>{
        // console.log("wait for 10s ==> ")
        setTimeout(() => {
            resolveInner("Got it resolved.")
        }, 10);
    }).then(mes => {
        console.log("wait for 2s ==> ")
        return mes;
    }));
});
```

```
// promise.then(message => console.log(message)).catch(err => console.log(err));

let anotherPromise = new Promise(resolve=>{
    return resolve(new Promise(resolveInner =>{
        setTimeout(() => {
            // resolveInner("Resolved.")
            resolveInner(promise.then(message => {
                console.log("Inner Promise : "+message);
                return message;
            }).catch(err => console.log(err)));
        }, 10);
    }).then(res => {
        console.log("wait for 1s ==> ")
        return res;
    }));
});

anotherPromise.then(res => console.log("Outer Promise : "+res)).catch(err=>console("got an err "+err));
```