Advance Javascript

->Session storage , local storage and basic of cookie

->session storage:

->Session storage are used data store a particular time. Whenever you have to close a tab then data will deleted.

->Data store in key value pairs.

Temporary data storage
Data security
Better performance

Syntax:

->sessionstorage.setitem(“key”,”value”);
->sessionstorage.getitem(“key”);
->sessionstorage.removeitem(“key”);
->sessionstorage.clear();

->local storage:

->local storage are used to store data permanently , data are stored when you have not data manually delete or browser uninstall.

->data are store in key value pairs.

Permanent storage
Large data storage
Security
Cross tab access

Syntax:

->localstorage.setitem(“key”,”value”);
->localstorage.getitem(“key”);
->localstorage.removeitem(“key”);
->localstorage.clear();

Local storage and session storage are string only storage. Whenever you have to store object and array so you have use json.stringfy() and retrieve this content to use json.parse() it’s use a original format data.

->cookie:

->cookie are data store in text file.it’s size very small 4kb.
->like token,small information etc.
->this data are share between client or server.
->you have to set cookie expiry date.
->when you have to closed browser so cookie is deleted.
->but you have to set expiry date so cookie will store before expiry date.
->the cookie will only sent through https connection. Not http
->cookie has only access by server. not access by javascript.

Syntax:

->Document.cookie = “username=parth;exipre=wed, 20dec 2024 12:00:00;path=”/” ”.

->**caching:**

->caching means data store in temporary time.
->caching is not store to permanently

Speed and Performance
Low server load
Offline access

Example:e-commerce website.

Syntax:

1. First check data is available in cache?
2. Then data is not available in cache after data is fetched from original resource.
3. If data is available in cache after data is return .

Permanently Data store in cache so we have some face issue.
Like security,outdateddata, performance,memoryfull etc.
Ex. news website.

Browser Debugging:

element:This panel is used to inspect and manipulate the HTML and CSS of a webpage.

Console: Used for debugging JavaScript code.

Sourse: This is the primary place for debugging JavaScript with features like breakpoints

Network Panel:network requests, responses, and other resources loaded by the webpage.

Performance Panel: measure JavaScript performance

Application: used to manage like session , local and cookie etc storage.

What is oojs?
-> oop is a programming paradigm that uses objects to represent data and methods to manipulate that data, allowing for better organization and modularity in code.

->es6 model it is a built in class keyword to define a classes.

Use Class keyword to create a class .
Always add a method name constructor().

```
class Introduction{
    constructor(fName,age){
        this.fName=fName;
        this.age=age;
    }
    details(){
        console.log(`hello everyone my name is ${this.fName} and my age is
${this.age}`);
    }
}

const intro = new Parth("ravi",21);
intro.details();
```

Fname and age is class property and details is class method.

Constructor:
It is executed when object is initialized.
If you have to use constructor so you have some benifit reusable code,obj not initialize manually.

If you have not use constructor so js will add a empty constructor method.

->prototype:

->it is use to add new keyword in class.

->with out prototype you have to not add keyword.

->get and set keyword:

->get is use to retrieve an object property.

->set is use to set an object property.

->static keyword:

->it has to not to an object but it has to a class.

->it is directly access by class name not object name.

```
class car{
    static startEngine(){
        return "start engine";
    }
}
console.log(car.startEngine());
```

```
class myCar{
    constructor(brand){
        this.brand=brand;
    }
    static numOfWheels =4;
}
const xuv = new myCar("innova");
console.log(myCar.numOfWheels);
```

It is use a mycar.numOfWheels;

Don’t use xuv.numOfWheels(error).

Difference between let var and const:

Let :

- it is block scope.
- It can’t update and redeclare
- It can’t access without initialization.

Var:

- It is use functional scope.
- It can update and redeclare.in scope.
- It can access without initialization.(undefined)

Const:

- It is use block scope.
- It can’t update and redeclare.
- It can’t declare and access without initialization.

Arrow function:

- **->**it’s syntax is short/
- ->it’s use map,filter

Let sum = (a,b)=> a+b;

It’s not use return keyword.

It’s not use this keyword.

It’s not argument object.

If you have to use constructor so you not use arrow function.

-> import export async await

->export: you can export var,fun from a file so they can use in other file.
->import:it is used to code from other file.

->type of export:

Named export:
export const add = (a, b) => a + b;
export const subtract = (a, b) => a - b;

Default export:
const greet = (name) => `Hello, ${name}!`;
export default greet;

->async:
It is use to declare an asynchronous function.

```
async function firstGetApiData(){
    // sent req
    let responce = await
fetch('https://jsonplaceholder.typicode.com/posts/')
    console.log(responce)

    // json
    let data = await responce.json();
    console.log(data)
}

firstGetApiData();
```

-> await:
It is used only in async function.
It is used to wait a result for function operation.

Difference == , === , != , !==

==: it is check an only value not check data type.
It is call loose equality
console.log(5 == ‘5’) // true

===: it is check data type and value .
console.log(5 === ‘5’) // false
It is called strong equality
5 is number and ‘5’ is string so both data type not matched.

!=: it is check data type.
console.log(5 != ‘5’) // false

!== : it is check data type and it’s value.
console.log(5 !== ‘5’) // true.
5 is number and ‘5’ is string so does not match data type
So return true.

Mixins

```
const CanEat = {
  eat: function() {
    console.log("Eating...");
  }
};
const CanWalk = {
  walk: function() {
    console.log("Walking...");
  }
};
const CanSwim = {
  swim: function() {
    console.log("Swimming...");
  }
};



class Person {
  constructor(name) {
    this.name = name;
  }
```

```
}

Object.assign(Person.prototype, CanEat, CanWalk, CanSwim);

const person = new Person("John");
person.eat();
person.walk();
person.swim();
```