- **Variable :**
  Refers to the value store in it.
  Ex: name : “Dev Nakum”
  console.log(name); → Dev Nakum
  can hold → number, string, objects

  3 types of variables → var, let, const

  - var:
    - it is outdated
    - reassigned value to variable declared with var
    - value can be changed
  - let:
    - value can be changed
    - block-scope
  - const:
    - value cannot be changed

- prompt() → Built in JavaScript functionality that helps you to get input from a user through the browser

- Number(“100”) converts the string 100 to number 100.

- Conditionals are used in statements to compare variable’s value and data-types
  - Return true or false

- **functions**: block of code designed to perform a particular task and is executed when “something” calls it.

```
function funName(){
   // function body
}
```

  → template literals provides an easy way to interpolate variables and expressions into strings

```
// regular functions
function myFunction(name){
   console.log(`Hii ${name}`)
}

// arrow functions  -> explicit
const myFunction = (name)=>{
   console.log(`Hii ${name}`)
}

// arrow functions  -> implicit
const sum = (a,b) => a+b
```

1. **Array**:

```
const numArray = [1,2,3,4,5,6];
console.log(numArray.slice(2,5))    // [ firstIndex,lastIndex )
```

2. **Objects:**

It is a type of variable, quite similar to array but they have something called key-value pairs

```
const personName = {
   fname:"Dev",
   lname:"Nakum"
}

console.log(personName.fname)   //dot notaion
console.log(personName['fname'])    //bracket notaion
```

3. **Loops**

```
const num = [1,2,3,4,5,6,7,8,9,10];

for (let i = 0; i < num.length; i++) {
   const element = num[i];
   console.log(element)
}

num.forEach(ele => {
   console.log(ele)
});


// forof loop → it gives iterator
for (const itr of num) {
   console.log(itr)
}


// forin loop → it gives index
const str = 'Hello from other world';
for (const key in str) {
   console.log(key)
```

```
}

while (condition) {
   //body
}


do {
   //body
} while (condition);
```

4. **Higher order functions**

map: loops and returns an array, manipulate the data

```
let result = num.map(number=>number*2)
console.log(result)
```

filter: loop and returns an array with matching condition

```
let filterResult = num.filter(number=>number>2)
console.log(filterResult)
```

reduce: takes in a functions as an argument, loops and gives you back the accumulator

```
const result = num.reduce((prev,next)=>prev+next);
console.log(result)
```

5. **DOM Manipulations**

```
let title = document.getElementById("title");
console.log(title.innerHTML);

title.innerHTML = "okk";
console.log(title.innerHTML);

title.style.color = "red";
title.style.display = "inline-block";
title.style.backgroundColor = "black"
```

6. **OOJS**

4 Pillars

- Encapsulation → Reduce complexity + increase reusability

- Abstraction → Reduce complexity + isolate impact of changes
- Inheritance → Eliminate redundant code
- Polymorphism → Rename switch case statements

7. Class & Objects

```
class RailwayForm{
   constructor (name,trainNo) {
      this.name = name;
      this.trainNo = trainNo;
   }

   submit(){
      alert(`${this.name} your form is submitted for train no ${this.trainNo}`);
   }

   cancel(){
      alert(`${this.name} your form is canceled for train no ${this.trainNo}`);
   }
}

const dev = new RailwayForm("Dev",123123);
dev.submit();

const kishan =  new RailwayForm("Kishan",234234);
kishan.submit();

// cancel the form
dev.cancel();
kishan.cancel();


// Inheritances
class Animal{
   constructor(name,color){
      this.name = name;
      this.color = color
   }

   run(){
      console.log(`${this.name} is running!!`);
   }
```

```
    shout(){
        console.log(`${this.name} is shouting!!`);
    }
}


class Monkey extends Animal{
    eatBanana(){
        console.log(`${this.name} is eating banana!!`);
    }
}

const ani = new Animal("Tiger","white")
const m = new Monkey("Chimpanji","Black");

ani.run();
m.run();
m.shout();
m.eatBanana();
```

super → call methods of parent class

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

    logedIn(){
        console.log(`${this.name} is logedIn`);
    }

    logedOut(){
        console.log(`${this.name} is logedOut`);
    }

    requestLeaves(leaves){
        console.log(`${this.name} your ${leaves} days leaves has been approved`);
    }
}

class Programmerr extends Employee{
    requestLeaves(leaves){
        super.requestLeaves(leaves);
```

```
        console.log(`also one extra leave has been approved`);
    }
}

const pr = new Programmerr("Dev");
pr.logedIn();
pr.requestLeaves(3);
pr.logedOut();
```

static methods → This methods are used to implements functions that belongs to a as a whole class and not to any particular objects

```
class Animal{
    get animal_name(){
        return this.name;
    }

    set animal_name(name){
        this.name = Animal.capitalize(name);
    }

    walk(){
        console.log(`${this.name} is walking`);
    }

    static capitalize(name){
        return name.charAt(0).toUpperCase() + name.substr(1);
    }
}


const ani = new Animal();
ani.animal_name = "lion"
ani.walk()

console.log(ani.capitalize("lion"));
console.log(ani.animal_name);
```

→ **not possible to create instance**
→ **Lion**

8. async – await, Promises & callback

callback is a function that is passed as an argument to another function.

```
let data = [
```

```
    {fname:"Karan",lname:"Aggrewal"},
    {fname:"Yash",lname:"Ramani"},
]

// function - display the data into body
const getData = () => {
    setTimeout(() => {
        let result = "";
        data.forEach((it,idx)=>{
            result += `<li>${it.fname} ${it.lname}</li>`
        })

        document.body.innerHTML = result;
    }, 1000);
}


// function - add data into main data object use callback
const createData = (newData,cb) => {
    setTimeout(() => {
        data.push(newData);
        cb();
    }, 2000);
}

createData({fname:"Mayan",lname:"Patel"},getData);
```

Promises → 2 arguments – reject and resolve

```
let data = [
    {fname:"Karan",lname:"Aggrewal"},
    {fname:"Yash",lname:"Ramani"},
]

// function - display the data into body
const getData = () => {
    setTimeout(() => {
        let result = "";
        data.forEach((it,idx)=>{
            result += `<li>${it.fname} ${it.lname}</li>`
        })
```

```
        document.body.innerHTML = result;
    }, 500);
}


// function - add data into main data object use promise
const createData = (newData) => {
    return new Promise(((resolve,reject)=>{
        setTimeout(() => {
            data.push(newData);

            let error = false;
            if(!error){
                resolve();
            }
            else{
                reject("Something went wrong!!");
            }
        }, 1000);

    }))
}

createData({fname:"Mayan",lname:"Patel"})
    .then(getData)
    .catch((err)=>{
        console.log(err);
    });
```

async – await

async always returns promise
The **await** keyword is used inside an async function to pause the execution of the function until a Promise is resolved or rejected.

```
// function - display the data into body
const getData = () => {
    setTimeout(() => {
        let result = "";
        data.forEach((it,idx)=>{
            result += `<li>${it.fname} ${it.lname}</li>`
        })
```

```
        document.body.innerHTML = result;
    }, 500);
}


// function - add data into main data object use promise
const createData = (newData) => {
    return new Promise(((resolve,reject)=>{
        setTimeout(() => {
            data.push(newData);

            let error = false;
            if(!error){
                resolve();
            }
            else{
                reject("Something went wrong!!");
            }
        }, 1000);

    }))
}

const start = async () => {
    await createData({fname:"Mayan",lname:"Patel"});
    getData();
}
start();
```

9. caching

- store the data temporarily in memory or storage for quick retrieval when needed, instead of repeatedly fetching the same data from the original source