- # Day_4

- Store the data for any students:
  - roll, name, marks, phone
  - roll = 111;
  - name = "Sumit"
  - marks = 55
  - phone = "81111111"
- with above approach, there is no single container of data
- array: s1 = [111, "Sumit", 55, ""81111111]
  - but values are not having labels, so it will difficult when we wnat to process the data
- JSON: JavaScript Object Notation
  - JSON stores the data into key and value pairs
  - obj = {key1:value, key2:value, key3:value};
  - s1 = {roll: 111, name: "Sumit", marks: 55, phone: "81111111"}
  - fetching members of object ('.' member access operator):
    - console.log(s1.roll);
    - console.log(s1.name);
    - console.log(s1.marks);
    - console.log(s1.phone);
    - Now a days JSON has become a standard to share data between client and server
- XML: store the data/represent the data
- HTML: display

```
s1 = {roll: 111, name: "Sumit", marks: 55, phone: "81111111"};
console.log(s1);
console.log(s1.roll);
console.log(s1.name);
console.log(s1.marks);
console.log(s1.phone);
s1 = {roll: 111, name: "Sumit", marks: 55, phone: ["81111111", "991292938"]};
console.log(s1.phone);
console.log(s1.phone.length);

s1 = {
    roll: 111,
    name: "Sumit",
    marks: 55,
    phone: ["81111111", "991292938"],
    address: {houseno: "A198", city: "Nagar", pincode: "223213"}
    };

console.log(s1.address.city);

list=[
    {
    roll: 111,
    name: "Sumit",
    marks: 55,
    phone: ["81311111", "99323292938"],
    address: {houseno: "A198", city: "Nagar", pincode: "223213"}
    },
    {
    roll: 112,
    name: "Amit",
    marks: 57,
    phone: ["8111111431", "92391292938"],
    address: {houseno: "A1938", city: "jabalpur", pincode: "223233"}
    },
    {
    roll: 113,
    name: "Adi",
    marks: 58,
    phone: ["822111111", "99122392938"],
    address: {houseno: "A32", city: "Dholakpur", pincode: "223433"}
    }

]

console.log(list);
console.log(list[0].roll, list[0].name, list[0].phone);

for(i=0; i<list.length; i++){
  console.log(list[i].roll, list[i].name);
  console.log(list[i].phone[0]);
  console.log(list[i].phone[1]);
  for(j=0; j<list[i].phone.length; j++){
    console.log(list[i].phone[j]);
  }

}

rollNo = 112;
isFound = false;
for(i=0; i<list.length; i++){
  if(list[i].roll === rollNo){
        console.log(list[i].name);
        isFound = true;
  }

}
    if(!isFound){
        console.log("Not Found"+rollNo);
    }

topper = list[0];
for(i=1; i<list.length; i++){
  if(topper.marks< list[i].marks){
        topper = list[i].marks
  }

}
console.log("Topper: "+topper);
```

- Functions:
  - function is a collection of statements those are grouped together
  - function prevents code duplicacy
  - function provides code reusability, modularity
  - syntax:
    - function func_name(parameters){ //body; //return statement; }

```
    function sum(a, b){
      return a+b;
    }

    console.log(sum(4, 6));

    function max(a, b){
      if(a>b){
        return a;
      }
      return b;
    }

    console.log("Max: "+max(6, 2));
```

- keywords to declare a variable:

  - var, let, const
  - var a = 5;
  - let b = 6;
  - const c = 7;
  - any variable is declared without var, let, const keywords then that variable will always have global scope. Because without var, let, const keywords variable are referred on the window object

- var keyword:

  - variables declared using var keyword will get either local or global scope specifically
  - we can re-declare a variable using var keyword

```
function show(){
  var a = 5;
  console.log("inside show: ", a);
}
 console.log("outside show: ", a);

  var a = 5;
  console.log(a);
  var a = 6;
  console.log(a);
```

```
  console.log(a); //undefined but no error
  var a;
```

- whenever browser loads the JS code so before execution, JS engine reads the complete code and will place/move all the variables declaration(not assignment) and function definations at the top of the code, this process is known as hoisting
- but inside js engine:

```
var a;
console.log(a); //undefined
a=5;
```

```
  function demo(){
    if(true){
      var x = 8;
      console.log("inside if x:", x);
    }
    console.log("outside if x=", x);
  }
  demo();
```

- let keyword:
  - let variables are block scoped
  - we cannot redeclare a variable let keyword
  - Cannot acces let variable before initialization

```
  function demo(){
    if(true){
      let x = 8;
      console.log("inside if x:", x);
    }
    console.log("outside if x=", x); //Error
  }
  demo();
```

```
  let x = 10;
  console.log(x);
  let x = 11;
```

```
console.log(a); //Error
let a = 3;
```

- const keyword:
  - has same properties as let keyword
  - but 1 difference is there, that we can't change its value

```
const x = 5;
x = 2; //Error
```

- For JS, a function is also an object

```
var str = "Hello";

var myfun = function(){
  console.log("Hello world");
}

console.log(myFun);
console.log(typeof myFun);
myFun();
```

- a function without a name is anonymous function
- Callback function:
  - a function which is passed as an argument;

```
    function show(a){
      console.log(a);
    }

    show(function demo(){
```

```
    console.log("inside function");
  }); // a complete function defination is getting passed as the argument
```

- in the above code demo is the callback function

```
  function show(a){
    console.log(a);
  }
  function demo(){
    console.log("inside function");
  }
  show(demo); // a function reference is passed as the argument
```

```
  function show(a){
    console.log(a);
    console.log(a());
  }
  function demo(){
    console.log("inside function");
    return 5;
  }
  show(demo); // a function reference is passed as the argument
```

```
  function show(a){
    console.log(a);
    console.log(a());
  }

  show(function(){
    console.log("inside function");
    return 5;
  });
```

- if in a function call we are passing the entire defination then name of the callback function is not necessary
- we can pass anonymous function as a callback function

```
  var arr = [10, 9, 11, 12, 8, 78, 3, 5, 6];

  var evenarr = [];

  for(i=0; i<arr.length; i++){
    if(a[i]%2 === 0){
      evenarr.push(arr[i]);
```

```
        }
      }
      console.log(evenarr);
```

```
  var arr = [10, 9, 11, 12, 8, 78, 3, 5, 6];

  var evenarr = arr.filter(function(element){
          return element%2 == 0;
  }); // filter function will filter the given array based on the condition and it will return the filtred array
```

- Types of syntax for defining any function:
  - function keyword: function name(parameters){}
  - fat arrow function: (arguments) => {}

```
const sum = (a, b)=>{
  return a+b;
}

const c = sum(4, 6);
console.log(c);
```

```
const sum = (a, b)=>a+b;
```

```
const sum = (a)=>a+5;

const c = sum(10);
console.log(c);
```

```
 var evenarr = arr.filter(element => element%2 == 0);
```