JavaScript

1. var let const
   var a; //declare
   var a=12; //declare and initialize
   - in window it will be added
   - function scoped
   - we can reassign & redeclare again no error will come

   let a;
   let a=12;
   - we cannnot redeclare again
   - ✓ let b=12;
   - ✓ let b=23;

   const a=12;

2. Scope (global, block, functional):
   - Global: var a=12; // can be used anywhere in code
   - Block: { // can be used within the block
     var a=12;
     }
   - Functional: function hello() { // can be used within the function
     let a=12;
     }

3. Temporal Dead Zone:
   - Js knows that the variable exists but cannot give value
   - Var has no tdz
   - Let has tdz
   - console.log(a);
     let a=12;
     - ➢ ReferenceError

4. Hoisting:

   Eg: var a=12;

   - var a= undefined;
   - a=12;
   - moving declarations on the top
   - when we create a variable, it breaks into 2 parts: declaration part will go on the top (var a=undefined) and the initialization part will be at the down (a=12;)
   - var can be hoisted and value is undefined
   - let can be hoisted and value cannot be set
   - const can be hoisted and value cannot be set

EXAMPLES:

- console.log(name);
  var name=”Suhas”;
  - undefined
- console.log(age);
  let age=21;
  - ReferenceError (because of tdz)
- var x=1;
  {
  var x=2;
  }
  Console.log(x);
  - 2
- let a=10;
  {
  let a=20;
  console.log(“Inside:”, a);
  }
  console.log(“Outside:”, a);
  - Inside: 20
  - Outside: 10
- If(true) {

  Var a=10;

  Let b=20;

  }

  Console.log(a);

  Console.log(b);

  - 10
  - ReferenceError

5. Data Types:
   - Primitive types:
     - when I copy a value, I get a real copy
     - string, number, Boolean, null, undefined, symbol, bigint
   - Reference types:
     - When I copy a value, I don’t get real copy of value but I get reference from parent
     - Arrays [], objects {}, functions ()

6. Strings:
   - ‘ ‘, “ “, ` `
7. Number:
   - 123

8. Boolean:
    - true, false
9. Null:
    - let a = ” ”;
10. Undefined:
    - let a;
11. Symbol:
    - Unique immutable value
12. BigInt:
    - Big integer values
    - MAX_SAFE_INTEGER
13. Arrays:
    - let a= [1,2,3,4];
14. Objects:
    - let a= {id: “1”, name: “Suhas”};
15. Functions:
    - function a() { }

16. typeof (quirks)
    - typeof 12 = number
    - typeof “Suhas” = string
    - typeof [] = object
    - typeof {} = object
    - typeof null = object
    - type of NaN = number

17. Type Coercion (== vs ===)

18. Truthy values and Falsy values:
    - Other than below values all are = true
    - 0 false “” null undefined NaN document.all = false

EXAMPLES:

- true + false => Output: 1
- null + 1 => Output: 1
- 5 + “5” => Output: ‘55’
- !!undefined => Output: false

19. Operators:
    - Arithmetic: + - * / % **
    - Comparison: = == === < > <= >= != !== !!
    - Logical: ! && ||
        i. &&
            1. T T = T
            2. T F = F

3. F T = F
4. F F = F

ii. ||

1. T T = T
2. T F = T
3. F T = T
4. F F = F

- Assignment: = += -= *= /= %=
- Unary: + - ! typeof ++ --
- Ternary: ? :

20. Instanceof:
    - Eg: let a= []; //typeof [] = object
    - a instanceof Array => Output: false
    - a instance of Object => Output: true

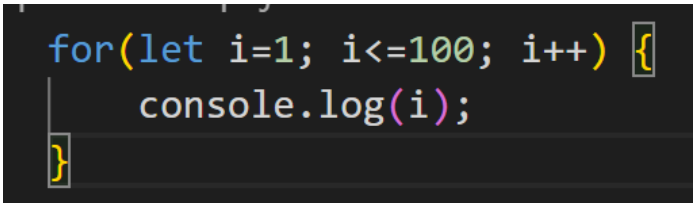
EXAMPLES:

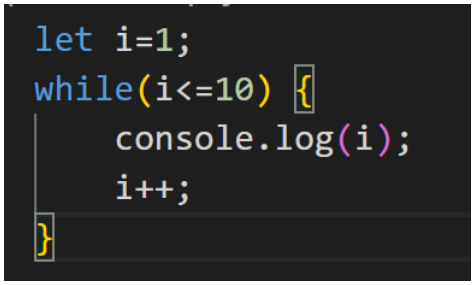
- eg:
  - ✓ a=2;
  - ✓ a++
  - ✓ console.log(a); //3
- eg:
  - ✓ a=2;
  - ✓ ++a;
  - ✓ Console.log(a); //3

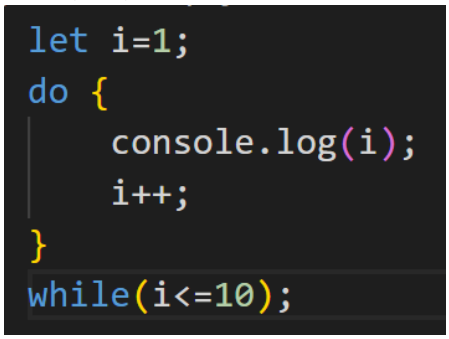
21. Control Flow:
    - If, else, else if
    - Switch case
    - Early return pattern

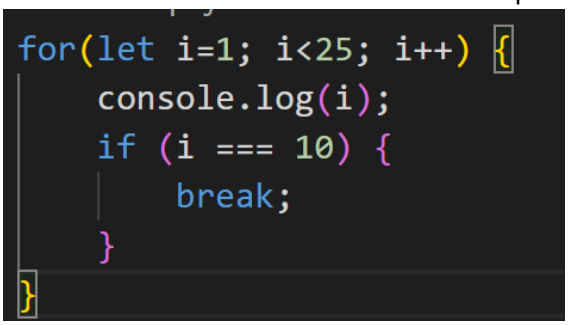
```
function getVal(val) {
  if (val < 25) return "D";
  else if (val < 50) return "C";
  else if (val < 75) return "B";
  else return "A";
}

getVal(76);
```

22. Loops:
    - for, while, do-while
    - break, continue
    - for-of, forEach for arrays
    - for-in, Object.entries for objects

23. For loop:
    - for(start,; end; change) { }
    - for(initialization; condition; iteration) { }
    - 
    - // prints 1 to 100

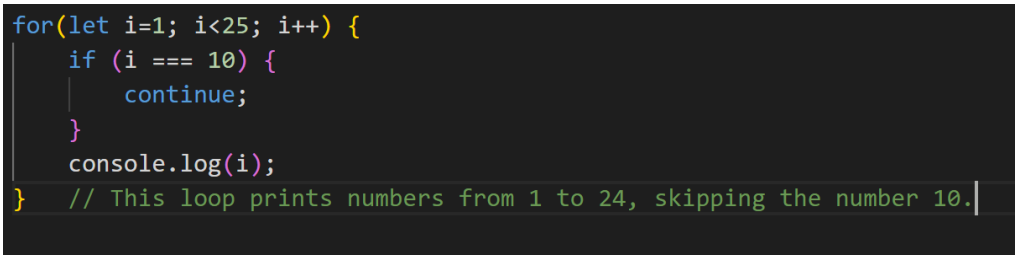
24. While loop:
    - used untill a specific condition is true
    - Pattern:
        i. start
        ii. while(end) {
        iii. change
        iv. };
        v. 
        vi. // 1 2 3 4 5 6 7 8 9 10

25. do-while loop:
    - used to execute the code once before checking the condition is true and will repeat the loop as long as the condition is true
    - Pattern:
        i. start
        ii. do {
        iii. change // executes first once
        iv. }
        v. while(end);
        vi. 
        vii. // 1 2 3 4 5 6 7 8 9 10

26. Break:
    - Breaks the statement and comes out of loop
    - 

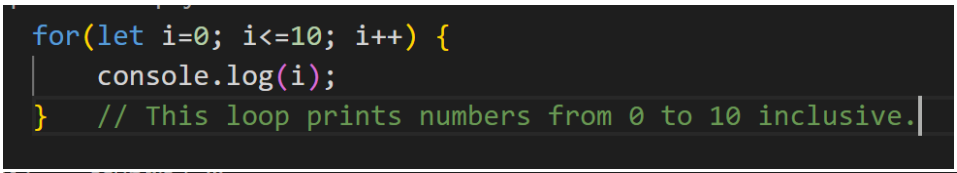
```
for(let i=1; i<25; i++) {
    console.log(i);
    if (i === 10) {
        break;
    }
}
```

    - // 1 2 3 4 5 6 7 8 9 10

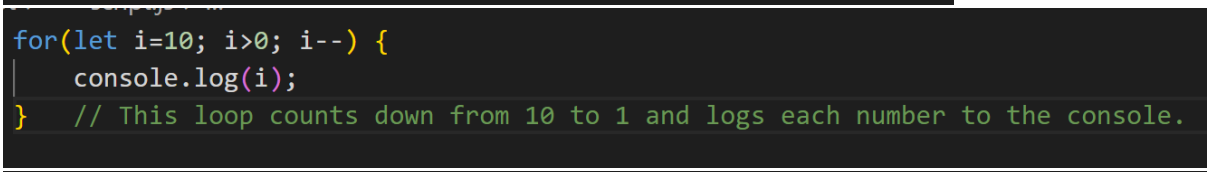
27. Continue:
    - skips the current/present iteration in loop
    - 

```
for(let i=1; i<25; i++) {
    if (i === 10) {
        continue;
    }
    console.log(i);
}   // This loop prints numbers from 1 to 24, skipping the number 10.
```

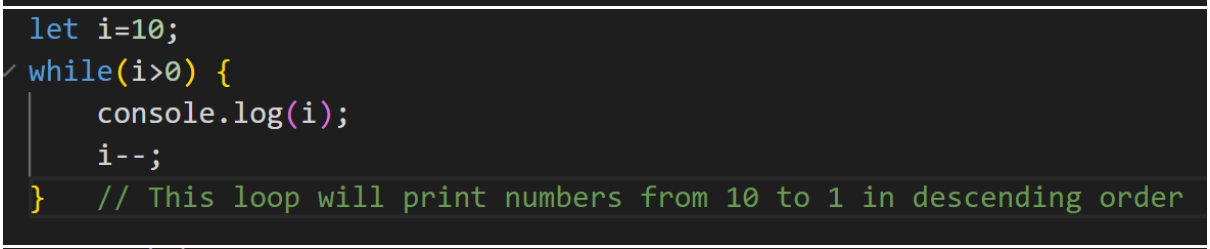
EXAMPLES:

- 

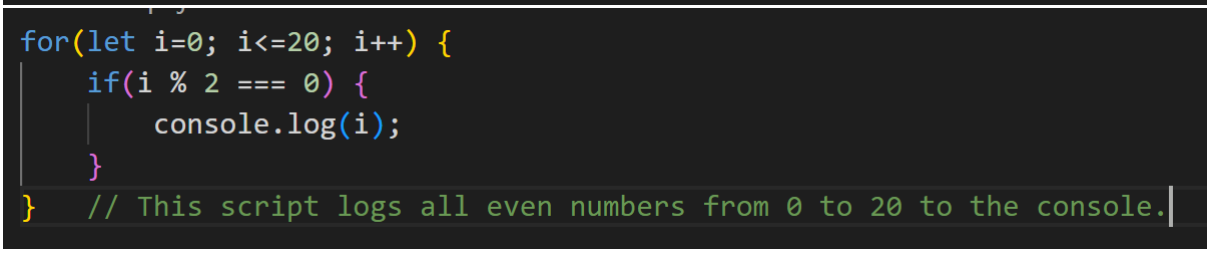
```
for(let i=0; i<=10; i++) {
    console.log(i);
}   // This loop prints numbers from 0 to 10 inclusive.
```

- 

```
for(let i=10; i>0; i--) {
    console.log(i);
}   // This loop counts down from 10 to 1 and logs each number to the console.
```

- 

```
let i=10;
while(i>0) {
    console.log(i);
    i--;
}   // This loop will print numbers from 10 to 1 in descending order
```

- 

```
for(let i=0; i<=20; i++) {
    if(i % 2 === 0) {
        console.log(i);
    }
}   // This script logs all even numbers from 0 to 20 to the console.
```

- ```
  let i=1;
  while(i<=15) {
      if(i%2===1) {
          console.log(i);
      }
      i++;
  }   // This code prints all odd numbers from 1 to 15
  ```
- ```
  for(let i=1; i<=10; i++) {
      console.log(`5 * ${i} = ${5 * i}`);
  }   // This loop prints the multiplication table of 5 from 1 to 10
  ```
- ```
  let sum = 0;
  for(let i=1; i<=100; i++) {
      sum += i;
  }
  console.log(sum);
  // calculates the sum of numbers from 1 to 100
  ```
- ```
  for(let i=1; i<=50; i++) {
      if(i%3 === 0) {
          console.log(i);
      }
  }   // This script logs all numbers from 1 to 50 that are divisible by 3.
  ```
- ```
  let val= prompt("Enter number:");
  for(let i=1; i<=val; i++) {
      if(i%2 === 0) {
          console.log(`${i} is even`);
      } else {
          console.log(`${i} is odd`);
      }
  }   //ask a number from user and check whether the number is even or odd
  ```
- ```
  for(let i=1; i<=100; i++) {
      if(i%3===0 && i%5===0) {
          console.log(i);
      }
  }   // prints all numbers from 1 to 100 that are divisible by both 3 and 5.
  ```

-
```
for(let i=1; i<=100; i++) {
    if(i%7 === 0) {
        break;
    }
    console.log(i);
}   // This will print numbers from 1 to 100 and stop when it finds the number divisible by 7
```
-
```
for(let i=1; i<=20; i++) {
    if(i%3 === 0) {
        continue;
    }
    console.log(i);
}   // This code logs numbers from 1 to 20, skipping multiples of 3.
```
-
```
let count=0;
for(let i=1; i<=100; i++) {
    if(i%2 === 1) {
        count++;
        console.log(i);
    }
    if(count === 5) {
        break;
    }
}   //prints first 5 odd numbers from 1 to 100
```

28. Functions:
    - function hello() { }
    - hello();
    -
```
function hello() {
    console.log("Hi");
}
hello();    // This is a simple function that prints "Hi" to the console.
```

```
function hello() {
    //function statements
}

let func= function() {
    //function expressions
}

let arrowFunc = () => {
    //arrow function
}
```

```
function dance(val) {
    console.log(`${val} is dancing!`);
}

dance("Suhas");
dance("Lion");
```

```
function add(a,b) { //a, b are passed as parameters
    console.log(a + b);
}

add(5, 10); //5 and 10 are passed as arguments to the function
```

29. Default, Rest, Spread parameters:
    - Default parameter:
        i.
```
function add(a=0, b=0) {
    console.log(a + b);
}

add();  // Outputs: 0
```

- Rest:
  - ```
    function abcd(...args) {
        console.log(args);
    }

    abcd(1,2, 3, 4, 5, 6, 7, 8, 9, 10); //arguments
    ```

30. Return:
    - Returns the value from where it came there only
    - ```
      function abcd() {
          return 12;
      }

      console.log(abcd());     // Output: 12
      ```

31. First Class Functions:
    - Treat functions as values
    - ```
      function abcd(val) {
          val();
      }

      abcd(function() {
          console.log("Hello");
      });
      ```

32. High Order Functions (HOF):
    - Function where parameter accepts function
    - ```
      function abcd(val) {    //HOF
          val();
      }

      abcd(function() {
          console.log("Hello");
      });
      ```

33. Pure functions
    - The Function where outside value does not change

```
let a=10;

function abcd() {       //outside a value does not change the value of function
    console.log("Hello");
}

abcd(); //Output: Hello
```

34. Impure functions
    - The Function where the outside value changes

```
let a=10;

function abcd() {
    a++;              //outside a value changes the value of function
    console.log(a);
}

abcd(); //Output: 11
```

35. Closures
    - is a function that returns another function will return a function which has always has access to parent scope function variables

```
function abcd() {
    let a=10;
    return function() {
        console.log(a);
    }
}

abcd()();   // Output: 10
```

36. Lexical Scoping

```
function abcd() {
    let a=10;   // a value can be accessed within abcd efgh
    function efgh() {
        let b=23;   // b value can be accessed within efgh ijkl
        function ijkl() {
            let c=45;   // c value can be accessed within ijkl
        }
    }
}
```

37. IIFE (Immediately invoked function expression)

```
(function () {
  console.log("Hello");
})();
```

EXAMPLES:

```
//difference between function declaration and function expression in terms of hoisting
// eg:1
abcd(); // This will work because of hoisting

function abcd() {
    console.log("Hello");
}   // Function declaration is hoisted
// Output: Hello

//eg:2
xyz();  // This will throw an error because function expressions are not hoisted

let xyz = function() {
    console.log("Hi");
}   // Function expression is not hoisted
// Output: Uncaught ReferenceError: Cannot access 'xyz' before initialization
```

```
//convert this function to arrow function
function add(a, b) {
    return a + b;
}

let add = (a, b) => {
    return a + b
};
```

```
//identify parameter and arguments
function welcome(name) {    // 'name' is the parameter
    console.log(name);
}

welcome("Suhas");   // "Suhas" is the argument
```

```
function sayHi(name="Suhas") {  //default parameter
    console.log("Hi", name);
}

sayHi(); //Output: Hi Suhas
```

```
function getScore(...val) { // Rest parameter to collect all arguments into an array
    console.log(val);
}

getScore(12, 15, 30, 45, 50);   // Output: [12, 15, 30, 45, 50]
```

```
function checkAge(age) {
    if (age < 18) return "Minor";
    return "Adult";
}

console.log(checkAge(20));  // Output: Adult
```

38. Arrays

```
//Array creation
let arr = [];
let arr1= new Array();

//Array access
let arr2= new Array(1,2,3,4,5,6)
arr2[1];   //Output: 2

//Array modification
arr2[3]=30; //Output [1,2,3,30,5,6]
```

- Methods:
  - Push:
    - Adds new elements to the end of an array, and returns the new length
  - Pop:
    - Removes the last element of an array, and returns that element
  - Unshift:
    - Adds new elements to the beginning of an array, and returns the new length
  - Shift:
    - Removes the first element of an array, and returns that element
  - Splice:
    - Adds or Removes array elements from original array
  - Slice:
    - Selects a part of an array, and returns the new array

- Reverse:
  - Reverses the order of the elements in an array
- Sort:
  - Sorts the elements of an array
  - It accepts function
  - arr.sort((a,b) => a-b (ASCENDING ORDER)
  - arr.sort((a,b) => b-a (DESCENDING ORDER)
- forEach:
  - Calls a function for each array element
  - It accepts function and no return statement
  - ```
    let arr= [1,2,3,4,5,6,7];

    arr.forEach(function(val) {
        console.log(val);
    })  //Output: 1,2,3,4,5,6,7
    ```
- Map:
  - Creates a new array with the result of calling a function for each array element
  - It accepts function
  - ```
    let arr= [1,2,3,4,5,6,7];

    let newArray = arr.map(function(val) {
        return val*2;
    })

    console.log(newArray);  //Output: [2, 4, 6, 8, 10, 12, 14]
    ```
- Filter:
  - Creates a new array with every element in an array that pass a test
  - It accepts function
  - ```
    let arr= [1,2,3,4,5,6,7];

    let newArray = arr.filter(function(val) {
        return val>4;   //test
    })

    console.log(newArray);  //Output: [5, 6, 7]
    ```
    ```

- Reduce:
  - Reduce the values of an array to a single value (going left-to-right)
  - It accepts function
  - ```
    let arr= [1,2,3,4,5,6,7];

    let newArray = arr.reduce(function(accumulator, val) {
        return accumulator + val;
    })

    console.log(newArray);  //Output: 28
    ```
- Find:
  - Returns the value of the first element in an array that pass a test
- Some:
  - Checks if any of the elements in an array pass a test
  - Gives true or false values
- Every:
  - Checks if every element in an array pass a test
  - Gives true or false values

39. Destructring

- ```
  let arr = [1,2,3,4,5,6];

  let [a,b,c] = arr;
  //Output: a=1 b=2 c=3
  ```

40. Spread Operator

- ```
  let arr = [1,2,3,4,5,6,7,8,9];
  let arr1 = [...arr];
  //Output: [1,2,3,4,5,6,7,8,9]
  ```
  ```

EXAMPLES

- ```
  let arr = ["Apple", "Banana", "Kiwi"];
  console.log(arr[1]);
  // Output: Banana
  ```
- ```
  //add mango at end
  arr.push("Mango");
  console.log(arr);
  //Output: ['Apple', 'Banana', 'Kiwi', 'Mango']

  //pineapple at begining
  arr.unshift("Pineapple");
  console.log(arr);
  //Output: ['Pineapple', 'Apple', 'Banana', 'Kiwi', 'Mango']
  ```
- ```
  //merge 2 arrays using spread operator
  let a = [1, 2];
  let b = [3, 4];

  let ans = [...a, ...b];
  console.log(ans);
  ```
- ```
  //add 12 to the start using spread operator
  let a = [1, 2];
  let b = [3, 4];

  let ans = [12, ...a, ...b];
  console.log(ans);
  //Output: [12, 1, 2, 3, 4]
  ```

41. Objects

```
//creation of object
let obj = {
    fn: "Suhas",
    ln: "H",
    age: 22,
    place: "Bangalore"
};

//accessing of objects
obj.age;
```

-
```
const user = {
  name: "Suhas",
  age: 22,
  address: {
    city: "Bangalore",
    pin: "652652",
    state: "Karnataka",
    location: {
      lat: 22.5,
      long: 33.6,
    },
  },
};

user.address.state;
//Output: 'Karnataka'
```

-
- Object Destructuring

```
const user = {
  name: "Suhas",
  age: 22,
  address: {
    city: "Bangalore",
    pin: "652652",
    state: "Karnataka",
    location: {
      lat: 22.5,
      long: 33.6,
    },
  },
};

//Object Destructuring
let {lat, long} = user.address.location;
```

42. For-in loop:
    - For let in

```
const user = {
  name: "Suhas",
  age: 22,
  email: "test@gmail.com"
};

for(let key in user) {
    console.log(key);
}
//Output: name age email
```

```
const user = {
  name: "Suhas",
  age: 22,
  email: "test@gmail.com",
};

for (let key in user) {
  console.log(key, user[key]);
}
//Output: name Suhas age 22 email test@gmail.com
```

43.
    - Accesing objects (2 ways)
      user.age and user[age]

44.
    - Object.keys(user) = all keys will be printed

```
> Object.keys(user);
<- ▶ (3) ['name', 'age', 'email']
```

```
> Object.entries(user);
<- ▼ (3) [Array(2), Array(2), Array(2)] i
    ▶ 0: (2) ['name', 'Suhas']
    ▶ 1: (2) ['age', 22]
    ▶ 2: (2) ['email', 'test@gmail.com']
      length: 3
    ▶ [[Prototype]]: Array(0)
```

45. Spread in Objects

```
let user = {
  name: "Suhas",
  age: 22,
  email: "test@gmail.com",
};

let user1 = {...user};
```

46. Object.assign
    - Used as spread operator

```
let user = {
  name: "Suhas",
  age: 22,
  email: "test@gmail.com",
};

let user1 = Object.assign({}, obj);
```

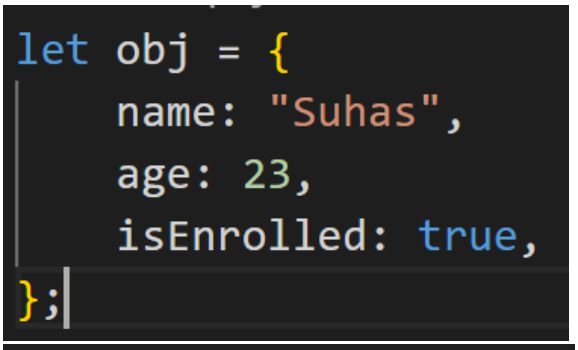
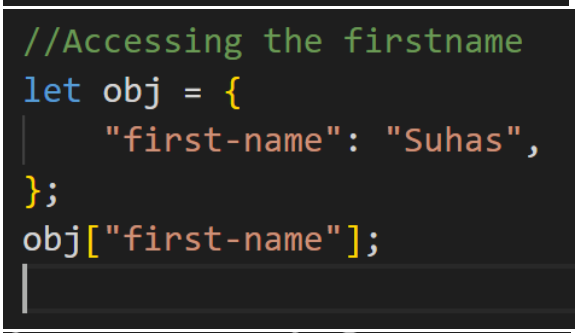
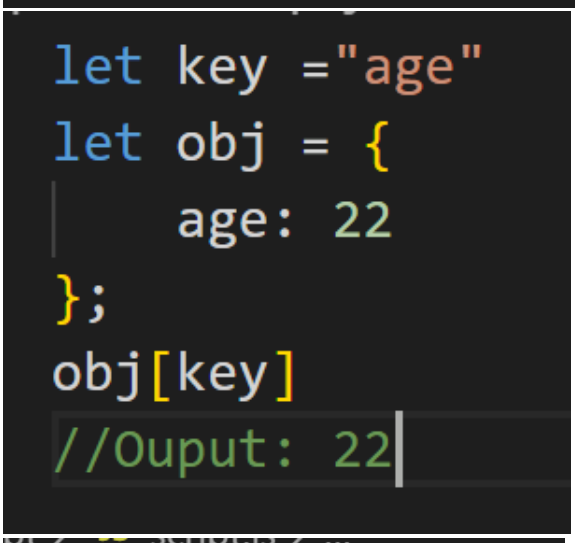
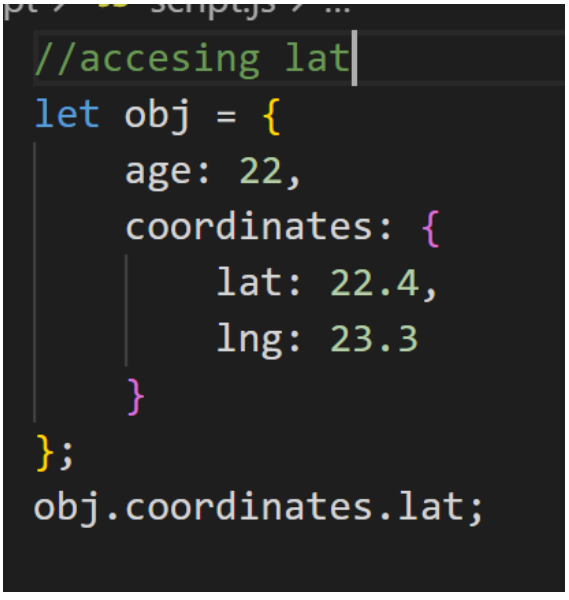
```
let user = {
  name: "Suhas",
  age: 22,
  email: "test@gmail.com",
};

let user1 = Object.assign({}, user); //displays all key and values
let user2 = Object.assign({place: "Bangalore", user});  //adds new key and value
```

    - Deep Cloning
        i. When there is nesting in objects, we cannot use spread to copy or clone
        ii. We can use JSON.stringify and JSON.parse

```
let user = {
  name: "Suhas",
  age: 22,
  email: "test@gmail.com",
  address: {
    state: "KA",
    pin: 357395,
    place: "Bangalore"
  }
};

let user1 = JSON.parse(JSON.stringify(user));
```

        iii.

EXAMPLES

- 
```
let obj = {
    name: "Suhas",
    age: 23,
    isEnrolled: true,
};
```
- 
```
//Accessing the firstname
let obj = {
    "first-name": "Suhas",
};
obj["first-name"];
```
- 
```
let key ="age"
let obj = {
    age: 22
};
obj[key]
//Ouput: 22
```
- 
```
//accesing lat
let obj = {
    age: 22,
    coordinates: {
        lat: 22.4,
        lng: 23.3
    }
};
obj.coordinates.lat;
```

- ```
  //destructure city and lat
  let obj = {
      age: 22,
      city: "Bangalore",
      coordinates: {
          lat: 22.4,
          lng: 23.3
      }
  };
  let {city} = obj.city;
  let {lat} = obj.coordinates.lat;
  ```
- ```
  //destructure key "first-name" as a variable called firstName
  const user = {
      "first-name": "SUhas",
  };

  let {"first-name": firstName} = user;
  ```
- ```
  //use for-in to log all keys
  const course = {
      title: "JS",
      duration: "4 weeks"
  };

  for(let key in course) {
      console.log(key);
  }
  ```
- ```
  //use Object.entries() to print all key value pairs as:
  // title: JS
  // duration: 4 weeks
  const course = {
      title: "JS",
      duration: "4 weeks"
  };

  Object.entries(course).forEach(function(val) {
      console.log(val[0] + ": " + val[1]);
  });
  // Output:
  // title: JS
  // duration: 4 weeks
  ```

```
//use a variable to dynamically assign a property
const key = "role";
let obj = {
    name: "Suhas",
    [key]: "admin"
};
//Output: {name: 'Suhas', role: 'admin'}
```