Unit - 4 Javascript

Java script :-

1. Java script is a program language used for creating dynamic, interactive webpages.
2. It is client side scripting language it means it runs users web browser rather than on the server.
3. It was released in 1995 by Netscape communication corporation.
4. The most popular frame works and libraries built on top of Java script include. React, Angular.
5. Javascript is used for front-end development which involves creating web pages.
6. It is also used for backend development for processing data.

Need of Java script :-

1. Dynamic interactivity :- Java script makes the web pages dynamically and it makes interactively.
2. Form validation :- Java script is used to validate the form which was filled by user.
3. client side processing :- Java script allows to perform calculations, manipulations and other operations on client side.

4. Cross browser compatibility:- Java script is supported by all web browsers.

→ One code can run in multiple web browsers.

5. Integration with other technologies:- Javascript can integrate with other technologies like HTML, CSS.

Environment setup:-

Environment setup means installing necessary s/w tools on computer.

Steps:-

1. Install Text editor:- We need to install text editor for writing Java script code.

Ex:- Vs code.

2. Install a Web browser:- We need to install a web browser to run our Java script code.

Ex:- Google chrome.

3. Install Node.js:- Node.js is a runtime enviroment that allows you to run Java script code ouside the web browser.

4. Install a version control System:- It means downloading Git.

5. Install a Javascript frameworks or library:-

Depending on project if necessary we need to install frameworks

Identifiers:- Identifiers are the names given to variables, functions and object in Javascript code.

Rules for identifiers:-

1. Identifiers must start with either letter, underscore, dollar sign.
2. Identifiers cannot start with numbers.
3. Java script is case-sensitive (Y and y are different)
4. Keywords cannot used as identifiers.

Types of identifiers:-

There are several types of identifiers to name variables, functions, objects.

1. Variables:- Variables are used to store data values. They are declared using "var", "let", "const" followed by identifier name.

```
var first name = "John";
let age = 30;
const pi = 3.14;
```

2. Functions:- Functions are used to perform actions and return values in Javascript.

→ Declared using "function" keyword.

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

Objects:- Objects are used to group related data and functions in Java Script.

→ Declared using curly braces with key value pairs separated by colons.

Classes:- Classes are new feature in Java Script that allows to create objects using template.

→ Declared using the "class" keyword.

Data types:- Java Script provides different data types to hold different values. (Stored directly in memory)

→ There are 2 types of data types

1. primitive data type:- An primitive datatype there are 5 types
   1. String:- Represents sequence of characters
      - ex:- "hello"
   2. Number:- Represents numerical values
      - ex:- 100
   3. Boolean:- Represents boolean value either
      - true or false.

Undefined:- Represents Undefined value.

Null:- Represents null i.e No value at all.

Non-primitive data types:- There are 3 types in non-primitive data type. (can't store directly)

1. Object:- Object represents a collection of values.

Ex:- Var person = {name: "John", age: 30};

2. Array:- Collection of similar datatype elements

Var numbers = [1, 2, 3];

3. Function:- Represents a reusable blocks of code that performs specific task.

→ primitive data types are simple and stored directly in memory.

→ Non-primitive data types are complex and cannot store directly in memory.

Operators:- Operators are the symbols that are used to perform operations on operands.

→ There are some types of operators in JavaScript.

1. Arithmetic operators:- Arithmetic operators are used to perform mathematical operations on numerical values.

→ The arithmetic operators are

1. addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Modulus (%)
6. Increment (++)
7. Decrement (--)

2. Comparision operators:- Comparision operators are used to compare two values & return boolean value

→ The Comparision operators are

1. Equal to (==)
2. Not equal to (!=)
3. Strict equal to (===)
4. Greater than (>)
5. Strict not equal to (!==)
6. Less than (<)
7. Greater than or equal to (>=)
8. Less than or equal to (<=)

3. Logical operators:- Logical operators are used to perform Logical operations on boolean values.

1. Logical AND : &&
2. Logical OR : ||
3. Logical NOT : !

4. Assignment operators :- Assignment operators are used to assign values to variables.

1. Assignment : =
2. Addition assignment : + =
3. Subtraction assignment : - =
4. Multiplication assignment : * =
5. Division assignment : / =
6. Modulus assignment : % =

5. Bitwise operators :- Bitwise operators are used to perform bitwise operations on binary values.

1. Bitwise AND : &
2. Bitwise OR : |
3. Bitwise XOR : ^
4. Bitwise NOT : ~
5. Left shift : <<
6. Right shift : >>

Types of Statements :-

Statements are used to control the flow of program and execute code.

1. Expression Statement :- Expression Statement are used to evaluate the expressions and update values.

Ex:-
```
var x = 5;
x = x + 1;
```

2. Conditional statements:- Conditional statements are used to control flow of program based on conditions.

→ There are 2 types.

1. if statement:- Executes block of code if condition is true.

```
if (x>5) {
    console.log("x is greater than 5");
}
```

2. if-else statement:- Executes one block of code if condition is true & executes another block of code if condition is false.

```
if (x>5) {
    console.log("x is greater than 5");
}
else {
    console.log("x is lesser than 5");
}
```

3. Switch statement:- Executes block of code based on value of the expression.

```
Switch (day of week) {
    case "Monday":
    console.log("Today is Monday");
    break;
```

```
case "Tuesday":
    Console.log("Today is Tuesday");
    break;
default:
    console.log("Today is not Monday or Tuesday");
}
```

3 Looping Statements:- Looping Statements are used to execute a blocks of code repeatedly.

1. For loop:- Executes a block of code specific number of times.

```
for(var i=0; i<10; i++)
{
    console.log(i);
}
```

2. While loop:- Executes a blocks of code until the condition is false.

```
while (x<10)
{
    console.log(x);
    x++;
}
```

3. Do-while loop:- Executes the block of code atleast once and then condition is checked.

```
do {
    console.log(x);
    x++;
} while (x<10);
```

4 Jump statements:- Jump statements are used to transfer the control to another part of program

1. Break statement :- It terminates the current loop.

ex:-
```
for (var i=0; i<10; i++)
{
    if (i==5)
    {
        break;
    }
    console.log(i);
}
```

2. Continue statement :- It skips the current iteration and continues next iteration.

ex:-
```
for (var i=0; i<10; i++)
{
    if (i==5)
    {
        continue;
    }
    console.log(i);
}
```

3. Return Statement :- Terminates the execution of a function and returns a value.

Ex:
```
function add (x,y)
{ return x+y;
}
```

Non-Conditional Statements :- Non-Conditional statements are the statements that do not involve any decision making based on condition.

1. Variable declaration Statement :- A Variable declaration statement is used to declare a variable.

Ex:-
```
Var x;
Var y=5;
```

2. Assignment Statement :- An assignment statement is used to assign value for variable.

```
x=5;
```

3 Function declaration Statement :- A function declaration statement is used to declare a function.

Ex:
```
function add (x,y)
{ return x+y;
}
```

Object declaration Statement:- Object declaration Statement is used to declare a object.

Ex:-
```
Var person {
    name : "xxx";
    age : 30;
    gender : xx;
};
```

Array declaration Statement:- Array declaration Statement is used to declare a array.

Ex:- fruits = ["apple", "banana", "Grapes"];

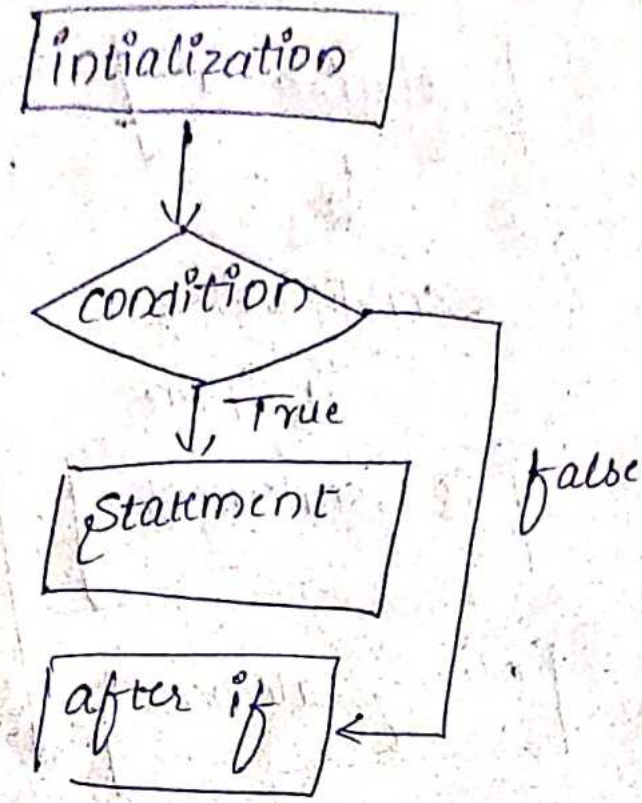
Types of Conditional Statements:-

→ Conditional Statements are used to execute a blocks of code based on condition.

1. if Statement:- if Statement is used to execute a block of code if condition is true.

Syntax:-
```
if(Condition)
{
    // Statements
}
```

flowchart :-

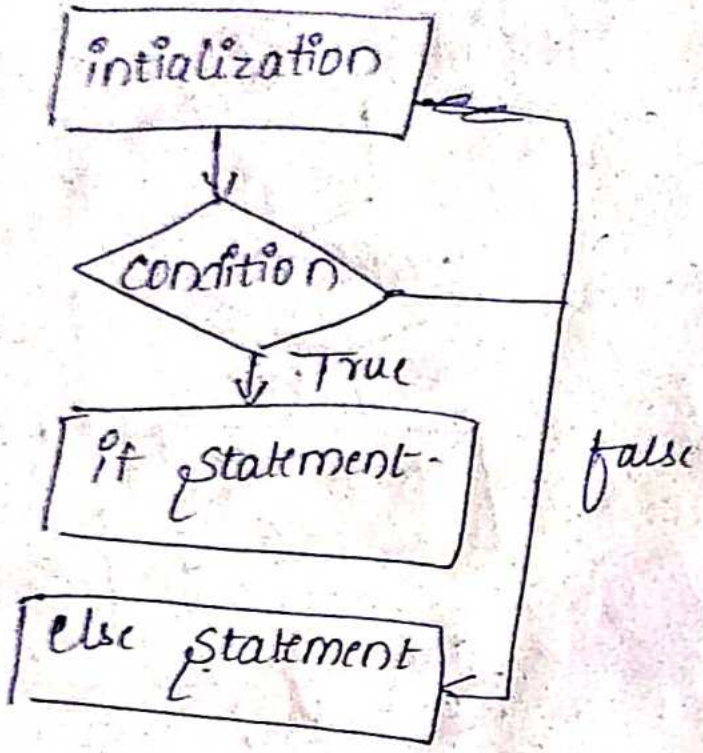


Ex:-
```
<script>
var a = 20;
if (a>10)
{
  console.log("a is greater than 10");
}
</script>
```

if-else statement :- if the condition is true the statement inside if condition is executed if the condition is false inside else condition statement is executed.

Syntax:-
```
if (condition)
{ // statement
}
else {
  // statement }
```

Flow chart :-



Ex:-

```
<script>
Var x = 20;
if (x==20)
{
  console.log ("x is equal to 20");
}
else {
  console.log ("x is not equal to 20");
  }
</script>
```

else if statement :- The statements are executed which condition is true.

Syntax:-

```
if (condition)
{
  // statement
}
else if (condition)
{
  // statements
}
else {
  // statements
}
```

ex:-

```
<script>
var a=20;
if(a==10);
{
console.log("a is equal to
      10");
}
else if (a==15);
{
  console.log("a is equal to
      15");
}
else {
  console.log("a is not both
      10 & 15");
}
```

If & Switch statements:-

(Refer conditional statements)

Types of loops:- loops are used to execute the block of code repeatedly until condition is false.

1. For loop:- The for loop is used when we know the no: of times to execute block of code.

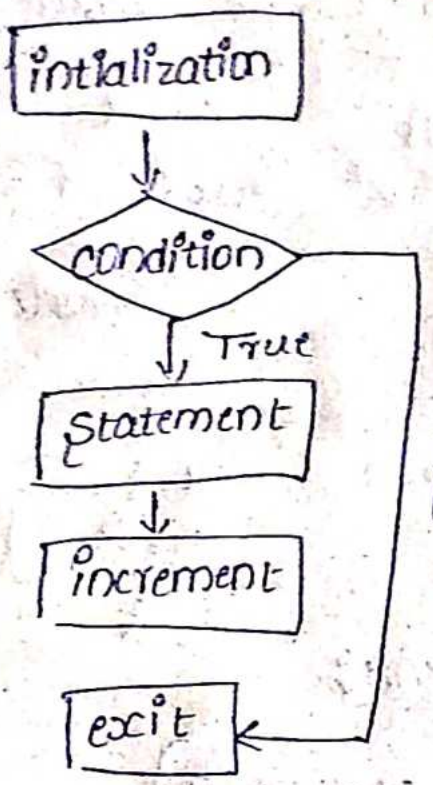
Syntax:-

```
for (intialization; condition; increment)
{
  // statement
}
```

Intialization:- Set the starting value.

condition:- Specifing the condition

increment:- increases the loop counter after each iteration.



Ex:-

```
for(i=0; i<10; i++
{
console.log(i);
}
```

While loop:- While loop is executed the block of code until the condition is false.

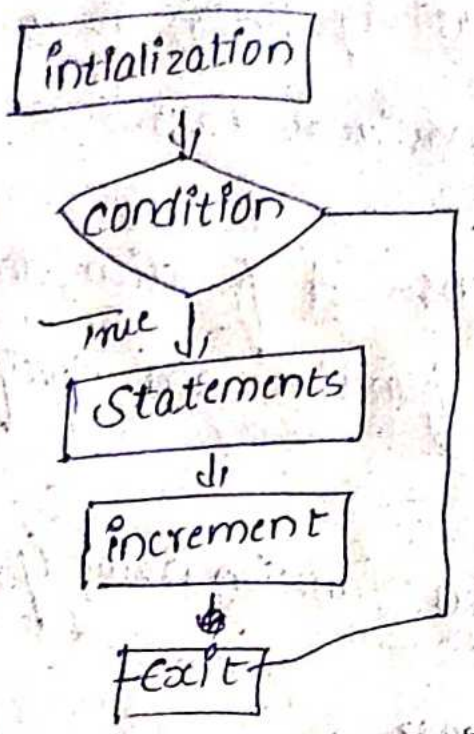
Syntax:-

```
while(condition)
{
  // statement //
}
```

Ex:-

```
var i=1;
while(i<5)
{
  console.log(i);
  i++;
}
```

flowchart:-



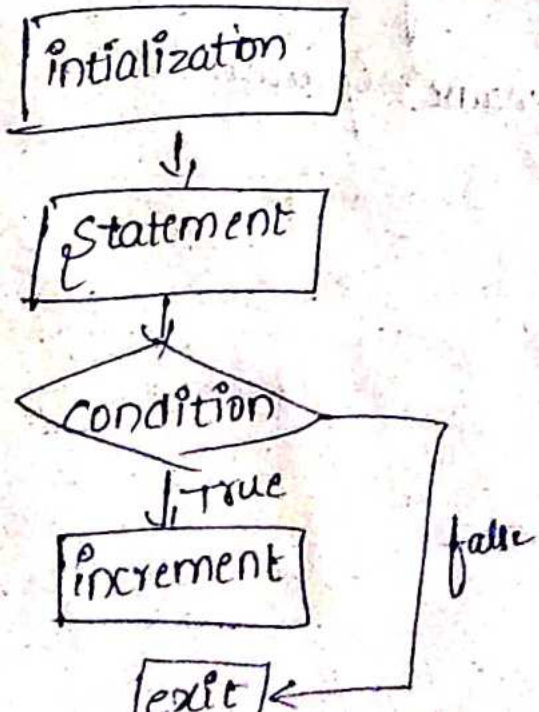
do while:- It is similar to while loop.

→ But it executes the block of code atleast once even the condition is false.

Syntax:-

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

flow chart:-



ex:-

```
var i=1;
do {
console.log(i);
i++;
}
while (i<5);
```

Types of functions:- Function is a block of code designed to perform particular tasks.

1. Function declaration:- A function declaration specifies the name of the function and specified with set of parameters.

Syntax:-
```
function function name (parameters)
{
  //code
}
```

2. Function expression:- Function expression is used to express function as expression rather than statement

Syntax:-
```
var function name = function(parameters)
{
  //code
}
```

3. Arrow functions:- These are similar to function expressions but have simpler syntax.

Syntax:-
```
var function name = (parameters)
{
  //code
}
```

4. Anonymous functions:- Anonymous functions are the functions without a name these are used as callback functions.

Ex:- Syntax:- var add numbers = function(a,b)

```
{ return(a+b);
}
```

5. Named functions:- A named function is a function that can be called with that name.

Syntax:-

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

6. Recursive functions:- A recursive function is a function that calls itself.

→ Recursive functions used to solve problem that can be broke down into smaller sub problems.

Ex:-

```
function factorial(n)
{ if(n==0)
  {
    return 1;
  }
else {
    return n*factorial(n-1);
  }
}
```

Declaring & Invoking functions:-

Declaring function:-

1. To declare a function we use "function" keyword followed by a function name & parenthesis.
2. Inside parenthesis we pass an arguments for function.
3. We use curly braces to define body of function

Example:

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

Invoking a function:-

1. To invoke (or) call a function we use function name followed by parenthesis.
2. In parenthesis we give a argument.

Ex:-

```
var result = add numbers(2,3);
console.log(result);
```

→ In both Examples we declare a function using different Syntax but we can still invoke the function in same way using function name, parenthesis

Arrow function:-

1. Arrow functions is a way of writing functions in shorter way.
2. Arrow functions were introduced by ES6 version they make our code more structured & readable.
3. Arrow functions are the anonymous functions i.e function without name.
4. The other name of arrow function is "Lambda" function.

Syntax:- const function name (parameters) {
// statement
}

Eg:- ~~con function add (2,3)~~
function add (a,b) {
return a+b;
}
const add=(a,b) => a+b;

→ it is defined using arrow syntax.

Advantages:-

1. Reduce Space complexity
2. Return statements are optional.
3. Increases readability and structure of code

Limitations:-

1. Do not have the prototype property
2. Cannot be used with new keyword
3. Cannot debug code

Function parameters:-

1. Functions take parameters which are input to function when it is called.
2. parameters are taken to perform some action for function.
3. Syntax for declaring function parameters is keep parameters inside parenthesis after function name & Separated with commas.

```
function function name (para1, para2)
{ // body
}
```

Ex:-

```
function add numbers (num1, num2)
{
    return num1 + num2;
}
console.log(add numbers (5,7));
```

Nested functions:-

1. Functions within another functions are called nested functions.
2. These nested functions have an access to variables and parameters of the outer function.
3. A function can have more than 1 inner functions.

Syntax:-

```
function outerfunction()
{
 function innerfunction()
  {
    // statement
  }
 innerfunction();
}
 outerfunction();
```

ex:-

```
function outerfunction()
 {
  function innerfunction()
  {
   console.log("innerfunction");
  }
  innerfunction();
 }
  outerfunction();
```

Built in functions:-

1. Javascript have many built in functions.
2. The other name of built in functions are "global functions.
3. These functions are used without need to define them first.

1. console.log():- This function is used to print the output of the program.
2. parseInt():- This function converts the string into an integer.
3. parseFloat():- This function converts the string into float.
4. alert():- this function creates an popup window.
5. confirm():- This function creates an pop up window with a message & two buttons.
6. prompt():- This function creates an pop up window with a message & input field.
7. setTimeout():- This function executes a function after sometime.
8. setInterval():- This function executes repeatedly at a specific interval.

Variable Scope functions:-

1. Scope of variables refers to the accessibility of a particular variable within program.

2. Javascript variables have different scopes they are:

1. Global scope:-

→ Any variable declared outside a function is said to have global scope.

→ A variable that can be accessed anywhere in the program is known as variable with global scope.

→ Global scoped variables can be defined using any of the 3 keywords let, const, var.

2 Local scope:-

→ Any variable that you declare inside a function is said to have local scope.

→ We can access a local variable within a function.

→ if we try to access any variable defined inside function from outside or another function it throws an error.

3. Block scope:-

→ This scope restricts the variable that is declared inside a specific block, from access by the outside of blocks.

→ The block scope does not works with Var keyword.

→ To access the variables we need to create an object.

Function level scope:-

→ The variables in this scope level are restricted to access only in inside the function where it is declared.

→ It can be accessed anywhere inside of that function, not accessible at outside function.

Working with classes:-

→ Classes are blueprint or template of object.

→ Javascript classes can be used to create new objects.

→ The new version of Javascript is using classes instead of functions.

→ In prior they were used functions to call objects.

Syntax:-

```
class class name
{
    constructor() {...}
}
```

→ The constructor method is used to the initialize the class name is used first.

Example:-

```
class person {
  constructor (name, age)
  {
    this.name = name;
    this.age = age;
  }
  greet()
  {
    console.log("Hello, my name is (this.name) and
    i'm (this.age) years old");
  }
}

const john = new person ('John', 30);
john.greet()
```

Creating & inheriting classes:-

1. The Javascript inheritance is a mechanism that allow us to create new classes on basis of already existing classes.
2. It provides flexibility to child class to reuse the methods and variables of parent class.
3. It is used extend keyword to create child class on basis of parent class.
4. It acquires all the properties of parent class to child class.

Important points:-

1. It maintains an IS A relationship.
2. extend keyword is used in class expressions or class declarations.
3. Using extend keyword we can aquire all properties and behaviours.
4. We can also use prototype method to acheive inheritance.

```
Class Animal
{
 constructor (name)
 { this.name = name;
 }
  Speak ()
  {
   console.log (this.name) make a noise;
  }
}
Class Dog extends Animal
 {
  constructor (name)
  {
   super (name);
  }
  Speak()
  {
   console.log ('this.name) barks;
  }
 }
  const dog = new Dog ('Fido');
   dog.speak();
```

Inbuilt events & handlers:-

1. The change in the state of object called event.

2. When Javascript code is included in HTML, JS react over these events & allows the execution.

3. This process of reacting over the events called event handling.

→ Some of the events are

1. Click-Event :- This event occurs when user click on an element, such as button or link.

→ The event handler for this event is "onclick" event handler.

Ex:-
```
<button onclick = "myfunction()">click me</button>
<script>
function myfunction()
{
  alert("Hello world");
}
</script>
```

2. Submit event :- This event occurs when an user Submits a form.

→ The event handler for this event is "onsubmit" event handler.

Ex:-
```
<button onclick = "myfunction">click me</button>
<script>
function myfunction()
{
  alert("Hello world");
```

```
}
</script>
```

3. Mouseover Event :- This event occurs when user moves mouse on element.

→ The Event handler for this event is "onmouseover" event handler.

Ex:-
```
<div onmouseover="myfunction()"> hover </div>
<script>
function myfunction()
{
  alert("mouse over");
}
</script>
```

4. Keydown event :- The event occurs when an user presses key on keyboard.

→ The event handler for this event is "on keydown"

Ex:-
```
<input type="text" onkeydown="myfunction()">
<script>
function myfunction()
{
  alert("key down");
}
</script>
```

5. Load Event :- This event occurs when a whole page or image has finished loading.

→ The event handler for this event is "onload"

Eg:-

```
<img src="my image.jpg" onload="my function
<script>
function my function()
{
alert("image loaded");
}
</script>
```

1. Working with objects:-

→ Objects are fundamental data structure that allows you to store and manipulate data.

→ Objects are collections of properties where each property has key and value.

Creating of object:-

```
let person={
name: "aaa";
age: 30;
gender: "male";
}
```

Accessing of objects:-

1. Dot notation:- We use dot to access the object

```
console.log(person.name);
```

2. Bracket notation:- We use brackets to access the object

```
console.log(person['age']);
```

Types of objects:-

1. Built-in-objects:- These are objects that are built into Javascript such as object, array, string & number.
2. custom objects:- These are the objects created ourself by using classes & constructors.
3. Document objects:- These are the objects that represent elements in HTML document such as document, window.
4. Host objects:- These are the objects that are provided by environment.

Creating objects:-

We have some types for creating objects.

1. Object literal Syntax:-

→ It is the simplest way to create an object.

→ We define an object using curly braces {} & Specify key-value pairs separated by commas.

Ex:-
```
const person =
{ name = "John";
  age = 30;
  hobbies: ["reading", "running"].
};
```

2. Constructor function:-

→ We can create object using constructor function.

→ This involves creating function that serves blueprint for the object.

→ We use "New" keyword to create object.

Ex:-
```
function person (name, age, hobbies)
{ this.name = name;
  this.age = age;
  this.hobbies = hobbies;
}
```

const person = new person ("John", 30, ["reading", "running"]);

3. Object create method:- This method allows you to create new object with specified prototype.

Ex:-

```
const personProto = {
    greet() {
        console.log("Hello" my name is " + name);
    }
};
const person = object.creat(person proto);
person.name = "John";
person.age = 30;
person.hobbies = "[reading", "running"];
```

Combining & cloning objects using Spread operator:-

The Spread operator is used to spread the properties of one object into another object & also create new object with existing object.

1. Combing object with Spread operator:-

We can combining object by spreading properties into new object.

→ useful when we want to merge two or more objects into single object.

Ex:-
```
const obj1 = {a:1, b:2};
const obj2 = {c:3, d:4};
const combined obj = {obj1, obj2};
console.log("combined obj");
```

O/P :- {a:1, b:2, c:3, d:4}

Cloning object with Spread operator:-

→ We can clone an object by spreading its properties into new object.

→ This creates new object with properties of existing object, but they both are not equal.

Ex:-
```
const obj1 = {a:1, b:2};
const cloned obj = {obj1};
console.log(cloned obj);
```

O/P: {a:1, b:2}

Destructing object:- Destructing object means it is an expression which allows us to access the data from objects like arrays, objects & assign it to new variables

```
const person = {name = "John", age = 30,
                hobbies = ['reading', 'running']};
const {name, age, hobbies} = person;
console.log(name);
```

→ We can also use destructing to assign default values to variables.

```
const person = {name: 'John', age: 30};
const {name, age, hobbies=[]} = person;
console.log(hobbies);
```

Browser & Document Object model:-

→ The browser & document object model are two important Api's that allow code to interact with web browser and document displayed in it.

→ The browser object model provides to access browser's window.

→ The browser object model is different in different browsers.

Bom methods:-

Windows.alert():- displays an alert box with a message and an ok button.

Windows.prompt():- displays a dialog box for user input.

Windows.open():- Opens a new browser window (or) tab

Windows.close():- closes the current browser window.

Document object model:-

→ The document object model provides to access html document displayed on browser.

→ The DOM represents the document as tree like structure of nodes.

→ We can also manipulate the structure & content of document.

DOM methods:-

1. document.getElementID():- Returns the element with specific id.
2. document.createElement():- Creates a new HTML element.
3. element.appendchild():- adds a new child node to element.
4. element.innerHTML:- Sets the HTML content of an element.

Creating Arrays:- Arrays are type of object that can store multiple values in single variable.

Ways to create Arrays:-

1. Literal notation:- We can create arrays by using literal notation by enclosing comma separated

values in square brackets []

Ex:- const fruits = ['apple', 'fruit'];

3. Constructor notation:- We can also create an array using constructor notation & passing values as arguments

Ex:- const numbers = new Array (1, 2, 3, 4);

3 Empty array:- We can also create an empty array by using literal notation with no elements in brackets and constructor notation without arguments

const array = [];

const num = new Array();

→ We can access elements using bracket notation and index of element. Starts with 0

const fruits = ['apple', 'grape', 'banana'];

console.log (fruits[0]);

console.log (fruits[2]);

Destructing Arrays:- Destructing Arrays is a way to extract individual values from arrays or objects & assigning them to variables

```
const numbers = [1, 2, 3];
const [a, b, c] = numbers;
console.log(a);
console.log(b);
console.log(c);
```

1
2
3

→ We can also used to assign default values to variables:

```
const numbers = [1, 2];
const [a=0, b=0, c=0] = numbers;
console.log(a);
console.log(b);
console.log(c);
```

1
2
0

Accessing arrays:-

→ We can access arrays by using indices of array.

→ The index of array starts from 0 and next index is 1 & so on.

Ex:-
```
const fruits = ['apple', 'banana', 'orange'];
console.log(fruits.index(0));
console.log(fruits.index(1));
```

→ We can also use array methods to access & manipulate arrays

Array methods:- Arrays have some built in methods to perform some tasks like adding, ormoving & sorting elements in array.

1. push():- adds one or more elements to end of array.

ex:- const fruits = ['apple', 'orange'];
fruits.push ('grape');
console.log ('fruits);

o/p:- ['apple', 'orange', 'grape']

2. pop():- Removes the last element from the array.

ex:- const fruits = ['apple', 'orange', 'grape'];
pop.fruits.pop();
console.log (fruits);

o/p:- ['apple', 'orange']

3 shift():- Removes the first element & returns it.

ex:- const fruits = ['apple', 'banana', 'grape'];
fruits.shift();
console.log (fruits);

o/p:- apple ['banana', 'grape']

4. unshift() :- adds one or more elements at the begning of array.

ex:-
```
const fruits = ['apple', 'banana'];
fruits.unshift('grape');
console.log(fruits);
```
o/p:- ['grape', 'apple', 'banana']

5. Splice :- adds or removes elements from specified position.

ex:-
```
const fruits = ['apple', 'banana', 'grape'];
fruits.splice(1, kiwi);
console.log(fruits)
```
o/p:- ['apple', 'kiwi', 'banana', 'grape']

6. concat() :- Returns a new array by combining two or more arrays.

ex:-
```
const fruits1 = ['apple', 'banana'];
const fruits2 = ['grape', 'kiwi'];
a = fruits1.concat(fruits2);
console.log(a)
```
o/p:- ['apple', 'banana', 'grape', 'kiwi']

sort():- Sort the elements of an array.

Ex:- const fruits = ['orange', 'banana', 'apple'];
fruits.sort();
console.log(fruits);

o/p:- ['apple', 'banana', 'orange']

Reverse():- Reverse the elements of an array.

Ex:- const fruits = ['orange', 'apple', 'kiwi'];
fruits.Reverse();
console.log(fruits);

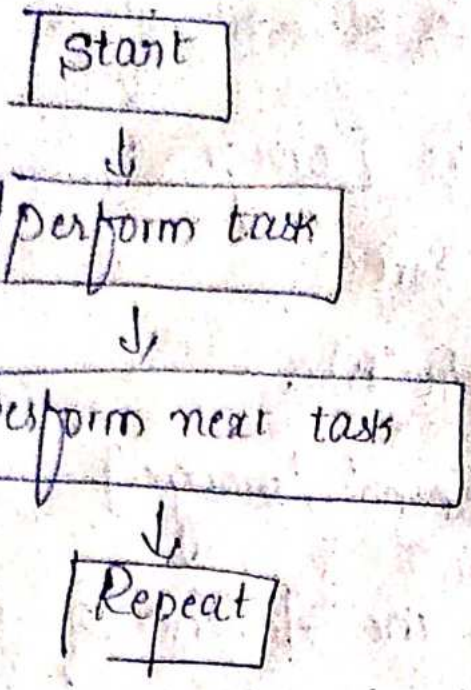
o/p:- ['kiwi', 'apple', 'orange']

Topic:- Introduction to Asynchronous programming:-

→ Asynchronous means that if javascript has to wait for an operation to complete then it will execute the rest of the code while waiting.

→ Java script is single threaded language means it can only execute one task at a time. But asynchronous programming allowing us to perform multiple tasks at a time.

→ Asynchronous programming is achieved using callbacks, promises, async/await



Techniques for writing asynchronous:-

1. promises 2. async/await 3. Callbacks.

Callbacks:- callback is a function which is to be executed after another function has finished execution.

→ Any function that is passed as an argument to another function so that it can be executed in the other function is called as a callback function.

→ callbacks can also be used with asynchronous tasks such as making HTTP request.

Key concepts of callback:-

1. Error handling:- callbacks contain error handling mechanisms to clear the errors.

2. Callback hell:- A common issue in callback is callback hell. it occurs when nested callbacks are taken.

Higher order functions:- functions that take other functions as arguments is called higher functions.

promises:-

1. promises are a way to handle asynchronous operations that return a value (or throw a error.
2. There are alternative ways to using callbacks, and provides a cleaner and more structured way of handling asynchronous code.
3. A promise is an object that represents a value that may not available yet but will be at some point.
4. It has 3 states
   1. Pending:- The intial state before the operation is completed.
   2. Fulfilled:- The operation is completed successfully & promise value is available.
   3. Rejected:- the operation is failed & error was thrown.

→ A promise can be created using promise constructor, which takes a single function as argument

→ This function has 2 parameters resolve & reject

→ the resolve function is called when the operation completes successfully.

→ the reject function is called when the operation fails

→ promises are also used to handle errors