① toString() : reads and returns all array elements in a string format. [, , ...]
② join() : reads and returns all array elements with custom delimeter.

~~Custod~~ Note:
Custom delimeter = Our own seperator

③ slice() : reads elements betⁿ specified index
④ filter() : Returns all elements that match given condition
⑤ find() : returns only first element that match given conditions
⑥ map() : it is an iterator for presenting elements.

It is a design pattern
It can read elements from a collection in sequential order
It does not require any condition, initialization and counter.

PAGE No.
DATE / /

Reading values by using loops and external iterators

For()
while()
do.. while
} Loops

For.. in // iterators of properties
For.. of // iterators of values
} External iterators

PAGE No.
DATE / /

Object type:

- The concept of object in computer programming was introduced in early 1960's by 'Alan Kay'

- To keep related ~~~~ data and logic under one reference
- Object store data in properties and defines logic using functions
- Object is a key and value collection

- Where key is string type
- Values can be any type

Syntax:

```
let object = {
    "key" : value ,
    "key" : value ,
    "key" : ~~value~~ function (){}
}
```

~~Ex:~~

* The properties of object can be accessed within object by using "this" keyword.

* Outside object you can access with reference of object name

PAGE No.
DATE / /

Inside : object {
this.key
}

Outside object.key;

- Object acts as a reusable template with sample data and logic which you can implement and customize according to requirement.

- Hence object is also known as "Pseudo Class".

- If object is representing a format of data then it is known as JSON (JavaScript Object Notation)

Ex. let product = {
"name" : " "
"price" : 0,
"stock" : false,
"qty" : 0,
"cities" : [],
"rating" : { Rate: 0, Count: 0 },
"total" : function () {
return this.qty * this.price;
}

```
print: function (){
        document.write(`Name: ${this.Name}
<br> Price = ${this.price}<br> Stock = ${this.stock}
<br> Quantity = ${this.qty}<br> Cities =
${this.cities.toString()}<br> Rating =
${this.rating.Rate}<br> Rating has been given by
${this.rating.count} people`);
    }
}
```

* Array of objects

- It is collection of objects.

Syntax:

```
[
    {    },
    {    }
]
```

PAGE No.
DATE / /

* Ajax calls in JS with Fetch method

- Ajax is asynchronous JS and XML
- It allows partial post back. It can post only specific portion of page
- It uses "xmlHttpRequest" object
- Java script provides fetch() which uses asynchronous request for fetching data from API

Syntax:

```
fetch("url").then(function(){get
{get data}).then(function(){convert to
JSON});
```

Distributed Computing

- Two applications running on two different machines can share information between them
- Two applications running in two different processes of same machine can share information
- There are various distributed computing technologies
  1. CORBA
  2. DCOM
  3. RMI
  4. EJB
  5. Web Services
  6. Remoting

There are 3 specification's

a) SOAP
b) REST
c) JSON

a) SOAP

Consumer ⇒ XML Request ⟺ XML Responce ⇐ Provider

b) REST

Consumer ⇒ Query Request ⟺ XML Responce ⇐ Provider

c) JSON

Consumer ⇒ JSON Request ⟺ JSON Responce ⇐ Provider

PAGE No.
DATE / /

Map: [From ES 5]

What is Map type?

- Map is similar to object with key and value collection

+++ What is difference beth Map and object

	Object	Map
①	key and value collection	key and value collection
②	key is only string type ex "id" : 1 //valid 1 : "II" //Invalid	key can be any type ex "id": 1 //valid 1 : "IV" //valid
③	You need explicit iterators to read key and values ex. for--in, for--of	Provides implicite iterators to read key and values ex. keys(), values(), entries()
④	Slow in accessing	Faster than object
⑤	Size of keys is unknown	Allows to access size of keys

Map methods:

set() : Adds new value
get() : Fetch value by using key reference
keys() : Returns all keys
values() : Returns all values
entries() : Returns both keys and values
delete() : Delete specific key
clear() : Deletes all keys

~~* Date type~~

Example

```
<script>
var collection = new Map();
collection.set("TV", "Samsung TV");
collection.set(1, "lenovo Laptop");
for (var key of collection.entries()) {
    document.write(key + "<br>");
}
document.write("Total keys: "
    collection.size
    OR
    : collection.keys.length );
</script>
```

PAGE No.
DATE / /

* Date type

- Java script date values are defined by using "Date()" constructor.
- It loads current date and time in memory.

```
var now = new Date();
```

- You can configure any specific date and time by using date value in constructor

```
var now = new Date("yyyy-mm-dd");
```

- You can access date and time values by using following methods

getHours() : Returns hour no. in 24 hr format
getMinutes() : Returns minutes no. [0-59]
getSeconds() : Returns seconds no. [0-59]
getMiliSeconds() : Returns miliseconds no. [0-999]
getDate() : Returns date no. [1-31]
getDay() : Returns weekday no [0 = Sunday]
getMonth() : Returns math no. [0 = January]
getFullYear() : Retruns year number [2023]
toLocaleDateString() : Returns complete date

toLocaleTimeString() : Returns complete time

Ex.

```
<script>
    let mfd = new Date("2023-07-27");

    document.write(`
        Manufactured Date : ${mfd.getDate()}
    <br> Manufactured Month : ${mfd.getMonth()}
    <br> Manufactured Weekday : ${mfd.getDay()}
    <br> Manufactured Year : ${mfd.getFullYear()}
    <br> Manufactured full date :
        ${mfd.toLocaleDateString()}
    `);
</script>
```

O/P

27

6

2023

27/07/2023

PAGE No.
DATE / /

- Javascript provides following methods for setting new date and time.

① ~~st~~ set Hours()
② set Minutes()
③ set Seconds()
④ set MiliSeconds()
⑤ set Date()
⑥ set Month()
⑦ set Year()

Note:

We can not set week day because week day is setted by Date so we don't have method for setting week day as "setDay()"

Note:

You can access date from HTML date picker and convert into Date format by using "Date()".

Syntax:

```
var departure = new Date(document.getElementById("txtDate").value);
```

Note :

JavaScript timer events :

- setInterval ()
- clearInterval()
- setTimeout ()
- clearTimeout()

- setInterval is used to perform specific task repeatedly at given time interval

Syntax :

setInterval (functionName, timeInterval);

(timeInterval in milisecs)

Ex :

setInterval (bodyload , 1000);

Note :

For real clock

100 mili secs = 1 sec

But for CPU clock

1000 mili secs = 1 sec.

PAGE No.
DATE / /

Regular Expression Type

- Regular expression is group of meta characters and quantifiers enclosed in "/ /"

Syntax:

Datatype ~~let~~ variable = /expression/;

ex:

let regExp = /\+91[0-9]{10}/;

- Regular expression is verified by using "match()" method

Syntax:

let mobile = "+919876543210";

if(mobile.match(regExp)){

code

}

PAGE No.
DATE / /

Math Object

- Provides set of properties and methods to handle mathematical operations

- Math.PI
- Math.sqrt()
- Math.sin()
- Math.tan()
- Math.pow()
- Math.round()
- Math.random() etc.

PAGE No.
DATE / /

JavaScript Operators

- Operator is an object that evaluates / ~~return~~ returns a value
- Operator comprises of operands that store data.

x + y - x and y are operands
+ is operator.

- Based on how many operands an operator can handle, the operators are ~~called~~ classified into following types.

a) Unary Operator [x++ , --y]
b) Binary Operator [x+y , x*y]
c) Ternary Operator [? :]

<condition> ? if_true : if_false

- Operators are again classified into different groups
Based on type of value they return

a) Arithematic Operators : number
b) Conditional Operators : boolean
c) Logical Operators : boolean ~~Binary~~
d) Bitwise Operators : Binary
e) Special Operators : Vary in functionality

a) Arithematic operators

①	+	Addition
②	-	Subtraction
③	*	Multiplication
④	/	Division
⑤	%	Modulus
⑥	**	Exponent
⑦	++	Increment
⑧	--	Decrement

① Addition operator : Returns sum of given nos

```
no + no          : no
no + String      : String
no + boolean     : no
String + String  : String
String + no      : String
String + boolean : String
boolean + String : String
boolean + no     : no
boolean + boolean : no
```

Note:

Any operation with undefined will be "NaN" except string

Any operation with null will be number except string

Substraction operator: Returns difference value

~~number~~ ~~string~~ boolean

~~number~~ String - number = NaN
String [in no. format] - no. = number
~~String~~ number - boolean = number
boolean - boolean = number
~~boolean~~
All other operations = NaN

Note
"-" operator uses implicite parsing but only for numeric expression in String format

Multiplication operator: Returns product of given numbers

number * no. != no.
no. * boolean != no.

All other operations = NaN

Division operator: Returns quotient of value

number / number = number
number / boolean = number
[true]

PAGE No.
DATE / /

Modulus operator : Returns remainder value

number % number = number
number % boolean [true] = number

Exponent operator : Returns value of base raised to power

2 ** 3 = 8

Increment and decrement operators :

They increase or decrease the current value with 1 and store the returned value

* Post Increment : Assigns then increments

x = 10
y = x++ x = 11 y = 10

* Post Decrement : Assigns then decrements

x = 10
y = x-- x = 9 y = 10

* Pre Increment : Increments then assigns

x = 10
y = ++x x = 11 y = 11

* Pre decrement :- Decrements then assigns.

x = 10
y = --x x = 9, y = 9

Conditional operators

==	Equal
===	Identicalt equal
!=	Not equal
!==	Not Identical equal
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal

FAQ: Diff. between =, ==, === operators

→ = : To assign a value
== : To compare 2 values of different type
=== : To compare 2 values of same type

Syntax

```
x = 10
y = "10"
x == y    // true
x === y   // false
```

PAGE No.
DATE / /

```
x != y      // false
x !== y     // true
```

All comparison operators return only boolean values

Logical Operators

&&	AND
~~OR~~	
||	OR
!	NOT

Syntax:

① Condition 1 && Condition 2

Expressions will be true only when both conditions evaluate to true

② Condition 1 || Condition 2

Expression will be true if any one condition is true

③ ! Condition

Negation

FAQ: what is difference beth break and jump statements ?

Ans: Both are Jump Statements.

"break" will terminate the block but will stay in script

"return" will terminate the script

Assignment Operators :

+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Mod and assign

Syntax

```
var x = 10 ;
var y = 20;
y += x ;   // y = y + x ;
```

PAGE No.
DATE / /

* Special Operators:

① typeof

- Returns datatype of reference variable

Syntax:

~~type of~~ typeof referenceName

OR

typeof object.property

② instanceof

- Used to ~~check~~ check classname from where given instance is derived [~~is~~ instance is object

Syntax: object instanceof Class

- It is a boolean operator which returns true or false.

Note:

Every Non primitive data type is Considered as an Object

PAGE No.
DATE / /

③ in operator :

- Used to varify existance of ~~object in~~ any property in object

Syntax: property in object

④ of operator

- Used to read all values of collection using iterator [for...of]

Syntax: for(var item of collection)
{
}

⑤ void :

- Discards the written value

Syntax:

Home

Ex.

Home

PAGE No.
DATE / /

⑥ Ternary operator

- It is used in decision making where it defines a condition and statements to execute on true or false

Syntax:

(Condition) ? Statement_if_true
: Statement_if_false

PAGE No.
DATE / /

* Javascript Statements

- A statement is used to control the execution flow

- Javascript statements are classified into following ~~statements~~ types

1) ~~Sett~~ Selection statements
2) Iteration and looping control statements
3) Jump statements
4) Exception handling statements.

1) Selection Statements

- Used in decision making
- Executed according to state and situation
- The selection statement keywords are

if, else, switch, case, default

i) if select :

- It is a decition making statement that executes statements according to condition.

- It is used in following forms
  a) Forward Jump
  b) Simple decition

c) Multi level decision
d) Multiple Decisions

a) Forward Jump :

- In this statements are executed only when condition is true
- There will be no alternative

Syntax

```
if (Condition)
{
    statement if true
}
```

b) Simple Decision : [if and else clause]

- In this a set of statements are executed when condition is true and another set of statements are executed when condition is false

Syntax

```
if (Condition)
{
}
else
{
}
```

* It can have only one alternative

c) Multi level Decision

- In this condition comprises of another condition with in context

Syntax:

```
if (condition 1)
{
    if (condition 2)
    {
    }
}
else
{
}
else
{
}
```

d) Multiple Decision:

- In this more than one alternative is provided for a set of statements.

Syntax:

```
if (condition 1)
{
}
else if (condition 2)
{
}
```

```
else if (condition 3)
{
}
else
{
}
```

ii) Switch Selector

- Switch can control flow of execution by selecting only the block of ~~ma~~ that matches given condition.

Syntax:

```
Switch (value / expression)
{
    case : statement jump;
    default : statement jump;
}
```

FAQ: Can we define switch without default?

Ans: Yes, There will be no alternative to execute.

FAQ: Can we define default above or before case?

Ans: Yes.

FAQ: Can we define case without break

Ans: Yes but execution continues to next case and stops when break occured or will stop at end of switch

FAQ: Can we define case with return ~~and~~ as jump statement

=> Yes only difference is:

Break will stop the block and stay in script

Return will stop the block and will come out of script

FAQ: Can we define case value as string type

Ans: Yes

FAQ: How to handle multiple case values for a set of statements?

Ans: a) You can handle multiple cases for one block

b) You can change capitalization for text

PAGE No.
DATE / /

* How to define case: For range of values:

Ans: By using boolean ~~conditions~~ expressions Switch can have only boolean true If multiple conditions are matching then it will execute only the first.

Ex:

```
<script>
    var n=6;
    switch (true)
    {
        case (n>=1 && n<=10) :
        document.write (`Your no. ${n} is
        between 1 to 10`);
        break;
        case (n>10 && n<=20) :
        document.write (`Your no. ${n} is
                between 11 to 20`);
        break;

        default:
        document.write(`Your no. is ${n} is
                -ve or above 20`);
        break;
    }
```

FAQ: Can we define if condition in switch case ?

Ans: Yes

FAQ: Can we define switch inside case ?

Ans: Yes we can
But... it is not recomended because of abnormal behaviour
Instead we should call another function containing the other switch

Note: What is Cascading dropdown ?

Ans: Set of dropdowns which change their options dynamically according to given inputs is called Cascading dropdown

Looping Control and Iteration Statements

- Looping is the process of executing set of statements repeatedly until given condition is satisfied.

- Looping requires

  a) Initialization
  b) Condition
  c) Counter

- Loops are created by using for, while and do-while

FAQ: What is difference between loop and recursion

i) for Loop

- Used when the exact iterations are known to developer and iteration count will not change dynamically.

- ~~It requires~~

Syntax:
```
for (initialization ; Condition ;
    counter )
{
}
```

- Initialization, Condition, Counter, Values are optional in for Declaration.

Syntax:
```
for(;;)
{
}
```

- You ~~th~~ can have counter decrement or counter with step value more than one

ex. i++ , i+=2 , i--

DATE / /

ii) While loop :

- It is used when iterations are unknown and iteration counter may change dynamically
- It will start loop only when condition is true

Syntax:

```
while (condition)
{
    statements ;
    counter;
}
```

Ex:

```
var i = 1;
while (i <= 10)
{
    document.write(i + "<br>");
    i++;
}
```

iii) Do loop :

- It is similar to while loop but ensures that the statements are executed at least once even when condition is false

Syntax:

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

Note: Dont use loops for reading data from collection always use them for a counter to execute.

PAGE No
DATE / /

* Iterators :

∴ Iterator is the design pattern used to read elements from a collection in sequential order

- It doesn't require initialization, condition and counter

- Iterators can be defined with

For .. in To read properties from collection
For .. of To read values from collection

Syntax:

```
For (var property in collection/obj)
{
}
For (var value of collection)
{
}
```