* Garbage Collector

→ It is used to deallocate the memory.

→ It runs ~~at~~ autometically in background.

→ We have no contol over it, mean we cannot contol the stard / end of it.

* Math Object

Math is a built-in object that has properties and methods for mathematical constants and function.

* Math Properties

The syntax is :

```
Math.property
```

Ex- Math.E // returns Eulers number

Math.PI // returns PI

Math.SQRT2 // returns square root of 2

* Math Methods

The syntax is:

Math.method(number)

Ex.

Math.round(x)	Returns x rounded to its nearest integer
Math.ceil(x)	Returns x rounded up to its nearest integer.
Math.floor(x)	Returns x rounded down to its nearest integer
Math.trunc(x)	Returns integer part of x
Math.pox(x,y)	Returns the value of x to the power of y.
Math.sqrt(x)	returns the square root of x.
Math.abs(x)	Returns the absolute (+ve) value of x.
Math.min()	Returns the lowest value in a list of arguements
Math.max()	Returns the highest value in a list of arguements.

* String

The string object is used to represent and manipulate a sequence of characters.

Creating Strings

It can be created as primitive, from string literals or as objects, using String() constructor.

Ex // primitive

```
const str1 = 'A string primitive';
const str2 = "A string primitive";
const str3 = `A string primitive`;
                // here ` are backtick characters

// using objects
const str4 = new String("A string object");
```

=> We can convert primitive string into object by using . character.

=> A string object can also be converted to its primitive counterpart with the valueOf() method.

* String Methods

=> length — It returns the length of a string

=> slice() — Extracts a part of string and returns the extracted part in a new string

=> substring(start, end) — Same as slice but start & end values are less than 0

=> substr(start, end) — Same as slice but second parameter specifies the length of the extracted part

=> replace() — It replaces a specified value with another value in a string.

=> toUpperCase() — A string is converted into upper case.

=> toLowerCase() — A string is converted into lower case.

=> concat() — It joins two or more strings

=> trim() — It removes whitespace from both ends of a string.

=> trimStart() and trimEnd()

* Template Literals

It uses back-ticks (` `) rather than the quotes ("") to define a string.

Ex

```
let text = `Hello World`;
```

⇒ You can use both single and double qoutes inside a string.

Ex

```
let text = `He's often called "Johnny"`;
```

⇒ Template literals allows multiline strings:

Ex

```
let text =
`This is
the best
news ever`;
```

* Interpolation

Template literals provide an easy way to interpolate variables and expression into string.

This method is called string interpolation.

Syntax

```
${...}
```

* Variable Substitution

Template literals allows variables in strings.

Ex

```
let fname = "Rishabh";
let lname = "Kushwaha";

let text = `Welcome ${fname}, ${lname} !`;
```

Output

Welcome Rishabh Kushwaha!

* Escape Sequence

Special characters can be encoded using escape sequence.

code	Result	Description
\0		null character
\'	'	single quote
\"	"	double quote
\\	\	backslash
\n		newline
\t		tab space

* Date Objects

Date objects are created with the new Date() constructor.

There are 9 ways to create a new date object.

new Date()

new Date(date string)

// const d = new Date("October 13, 2014 11:13:00");

new Date(year, month)

new Date(year, month, day)

new Date(year, month, day, hours)

new Date(year, month, day, hours, minutes)

new Date(year, month, day, hours, minutes, seconds)

new Date(year, month, day, hours, minutes, seconds, ms)

new Date(milliseconds)

* Getter & Setter

⇒ get

It is a function without arguements. The Javascript getter methods are used to access the properties of an object.

=> set

A function with one arguement. The Javascript setter methods are used to change the values of an object.

* Array Object

An array object is used to store multiple values in a single variable.

Ex

```
const cars = ["Saab", "Volvo", "BMW"];
```

* Array Operations
  * Adding new elements
  * Finding elements
  * Removing elements
  * Spilitting elements
  * Combining elements

* Array Creation

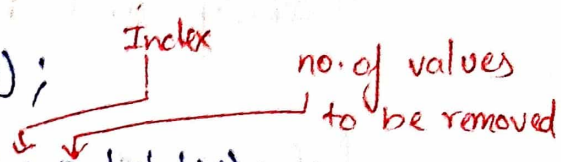
```
let arr1 = [1,2,3,4,5];
```

* Insertion

→ at start -- arr.push(0);

→ at middle -- arr.splice(1,0,'a','b');

→ at end -- arr.unshift(6);



* Searching

We can use;

indexOf() - Searches an array for an element and returns its position.

includes() - checks if an array contains the specified element.

NOTE - Predicate Functions

Predicate functions are functions that return a single TRUE or FALSE.

* Removing

→ at start -- arr.shift()

→ at middle -- arr.splice(index, no. of elements you want to delete)

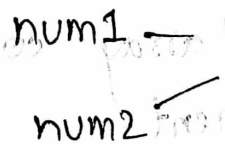
→ end — arr.pop()

* Emptying An Array

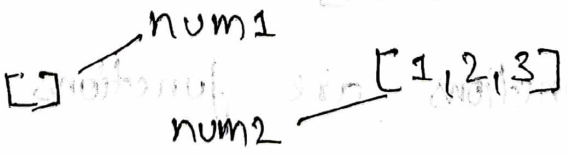
```
arr1 = [];
// array will be empty because of garbage collector

let num1 = [1,2,3];
let num2 = num1;

num1 = [];
// but it's not a good way delete an array
// because it's an object so address will be
// copied, not the copy is created.
```



So when we assign num1 = []



Here num1 will point to an empty array.

⇒ The best way to delete an array is using the below code.

```
arr.length = 0;
```

⇒ There is another way to delete an array using the splice().

```
num1.splice(0, num1.length);
```

* Combining And Slicing An Array

⇒ Combine

```
let arr1 = [1,2,3];
let arr2 = [3,4,5];
let combine = arr1.concat(arr2);
```

⇒ Slice

```
combine.slice(starting index, ending index);
// ending index means it will take value
// to index ending -1.
```

Combining an array using Spread operator(...)

```
let combine = [...arr1, ...arr2];
```

Creating a copy using spread operator (...)

```
let another = [...combine];
```

* Iterating / Traversing an Array

We will use **for of** loop because it works on iterables only.

```
for(let value of arr1) {
        console.log(value);
}
```

=> forEach()

The forEach() method calls a function for each element in an array.

```
arr1.forEach(function (number) {
        console.log(number);
});
```

Converting forEach() into arrow (=>) function

```
arr1.forEach(number => console.log(number));
```

* Joining Arrays

```
const joined = arr1.join(',');
```

* Splitting an Array

```
let mes = 'This is my name';
let parts = mes.split(' ');
```

* Sorting an Array

```
let numb = [3,7,0,9,6];
numb.sort()
```

We can also sort any array in reverse order after sorting in an unsorted array.

```
[numb.reverse()]
```

reverse() is used to reverse an array, it overwrites the original array.

⇒ Above functions cannot sort an object, it can be done using predicate function.

* Filtering an Array

⇒ The filter() method creates a new array filled with elements that pass a test provided by a function.

⇒ It doesn't change the original array.

Syntax

```
array.filter(function(currentValue, index, arr),
thisValue)
```

→ // Here → refers to optional values

Ex

```
let num = [5,7,-6,-3,0];
let filtered = num.filter(function(value){
    return value >= 0;
});
```

Using ⇒ function

```
let filtered = num.filter(value => value >=0);
```

* Mapping in Array

→ map() creates a new array from calling a function for each array element.

→ It calls a function once for each element in an array.

→ It does not change the original array.

Ex

```
let num = [7, 8, 9, 1];
let items = num.map(function(value) {
    return 'student_no ' + value;
});
```

Using => function

```
let items = num.map(value => 'student_no ' + value);
```

* Mapping With Objects

```
let num = [7, 8, -7, -10];
let filtered = num.filter(value => value >= 0);
let items = filtered.map(function(number) {
    return {value: number};
});
```

Using => function

```
let items = filtered.map(number => {value: number});
```

* Chaining

In chaining we replace a variable with its original code.

Ex

```
let items = num
    .filter(value => value >= 0)
    .map(number => {value: number});
// In the above code we replaced filtered
// variable with its original code.
```