rem vs em:

rem:

- relative to the root element (html)
- r stands for root
- for example:

```
<div style=”font-size: 20px”>
        <p style=”font-size: 2rem”> hello world</p>
</div>
```

2rem here means 2*16 (16 is the font-size of the root element a.k.a the html docment)

em:

- relative to the parent element
- for example:

```
<div style=”font-size: 16px”>
        <p style=”font-size: 2em”> hello world</p>
</div>
```

2em here means 2*16 (16 is the font-size of the parent element a.k.a the div element)

css position:

position: static;

- default
- positioned according to the normal flow of the page

position: relative;

- element positioned relative to its normal position.
- top, right, bottom, and left properties will cause it to be adjusted away from its normal position
- element leaves a gap in the page where it would normally have been located and other elements will not be adjusted to fill in that gap

position: fixed;

- element positioned relative to the viewport which means it stays in the same place even if the page is scrolled
- top, right, bottom, and left properties are used to position the element
- element does not leave a gap in the page where it would normally have been located (other elements will be adjusted to fill in the gap the fixed positioned element created)

position: absolute;

- positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed)
- if absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling
- element is removed from the normal flow, and can overlap elements.

position: sticky;

- element positioned based on the user's scroll position
- element toggles between relative and fixed

  depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed)

for vs while loop:

for loop:

- better used when the number of iterations is known
- if there is no condition in the loop, it keeps running infinite times

while loop:

- better used when execution depends on a statement being true and stops once it’s proven wrong
- if there is no condition in the loop, it gives an error

splice() and slice():

splice():

- adds and/or removes array elements.
- overwrites the original array
- **arr.splice(index, num, item1, item2);**
    Index: the index to add/remove at
    Num: the number of items to remove starting at the index
    Item1: an item to add to the array in the index
    Item2: an item to add to the array in index+1

slice():

- returns selected elements in an array, as a new array
- does not change the original array
- **arr.slice(start, end)**
    Start: the index to start at
    End: the index to end at (takes the last item before the end index. For example arr.slice(1, 3) will return a new array that has only two items which are arr[1] and arr[2])

object methods:

- methods are actions that can be performed on objects
- method is a property containing a function definition.
- accessing object methods:

    Object.methodName(objectName)

- there are some built-in functions like:

    - **Object.create()** method is used to create a new object and link it to the prototype of an existing object

    - **Object.keys()** creates an array containing the keys of an object

    - **Object.values()** creates an array containing the values of an object

    - **Object.entries()** creates a nested array of the key/value pairs of an object

    - **Object.assign()** is used to copy values from one object to another

    - **Object.freeze()** prevents modification to properties and values of an object, and prevents properties from being added or removed from an object

    - **Object.seal()** prevents new properties from being added to an object, but allows the modification of existing properties

    - **Object.getPrototypeOf()** is used to get the internal hidden [[Prototype]] of an object, also accessible through the **__proto__** property

regular vs arrow function:

regular function:

- syntax:

```
let x = function function_name(parameters){
   // body of the function
};
```

- have its own this
- accepts arguments

arrow function:

- syntax:

```
let x = (parameters) => {
   // body of the function
};
```

- do not have its own this. For example:

```
let user = {
    name: "John",
    arrowfun:() => {
        console.log("hello " + this.name); // no 'this' binding here
    },
    reqularfun(){
        console.log("Welcome " + this.name); // 'this' binding works here
    }
 };
user.arrowfun();
user.reqularfun();

Output:
        hello undefined
        Welcome John
```

- doesn’t accept arguments

objects vs instance oop:

objects:

- object means when memory location is associated with the object (is a run-time entity of the class) by using the new operator

instance:

- Instance refers to the copy of the object at a particular time whereas object refers to the memory address of the class

for example:

```
Class student()
{
   private string firstName;
  public student(string fname)
  {
    firstName=fname;
  }
  Public string GetFirstName()
  {
    return firstName;
  }
}
```

Object example:

Student s1=new student("Martin"); Student s2=new student("Kumar");

The s1,s2 are having object of class Student

Instance:

s1 and s2 are instances of object student the two are unique

it can be called as reference also.

basically the s1 and s2 are variables that are assigned an object