Javascript

JS is a programming language. casesensitive

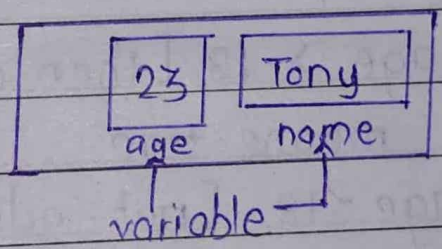
Using Console : Uses REPL

REPL : Read-evalute-print-loop

Variables

A variable is the name given to the memory location.

- Vary : Means it can changed (it's value can be changed)



(we can also change type of data type in JS)

Data types

(Information type) (operator: type of)

(i) Primitive types (primary)

* Number : 23
* Boolean : true or false [yes or no]
* String : text [eg: name or any]
* Undefined : not defined
* Null : a = null ;
* Bigint :
* Symbol :

Page No. :
Date :

[1] Numbers in JS

- positive & negative numbers (-5, 5)
- Integers (45, -50)
- Floating numbers - with decimals (1.1)

[2] Boolean in JS

To define any true or false value ie. yes or no values.

eg:] age = 7
if age > 18 [then adult]
means true
if age < 18 [not-adult]
means false
age = 7 (false)

[3] Strings in JS : Strings are text / sequence or set of characters, written in double or single quotes.

eg : name = 'Tony "stark"'
name = "'Tony' Stark"
char = 'a'
name = "Stark"
num = '123'
space = " "
emp = ""

Page No. :
Date :

4. Undefined : which is not defined (JS doesn't knows value)

5. Null : Represence of absence value.
(null is a keyword in JS)
a = null [this will print null not undefined]

Page No. :
Date :

Operations in JS

a + b ← operand

1) • Arithmetic ↑ operator

(i) Addition : a=2 , b=2 ⇒ a+b=4

(ii) Subtraction : a-b=0

(iii) Multiplication : a * b = 4

(iv) Division : a / b = 1

(v) Modulo :

(Modulo) (Remainder operator)

a=5, b=2

a % b = 1

(vi) Exponentiation (power operator)

$a^{**n} = 2**3 = 2^3 = 8$

NaN in JS

Global property representing value as Not a Number.

: NaN is a type of number in JS.

But is does not represents a valid value of the number.

eg: 0/0 , NaN +1 , NaN * 2,

NaN/NaN

Operator Precedence

( )	Bracket
**	powder operator (1↑, L←R)
*, /, %	Left to right (L→R)
+, -	(L→R)

(i) 3/1 + 2 ** 2
⇒ 3/1 + 4 ⇒ 3+4 ⇒ 7

(ii) 4 + 1 * 6 / 2
⇒ 4 + 6/2 ⇒ 4+3 ⇒ 7

2) Assignment operators

(i) equals to =

(ii) + equal to
⇒ a = a+1 ⇒ a+=1;

(iii) - equal to
⇒ a = a - 4 ⇒ a-=4;

(iv) * equal to
⇒ a = a * 2 ⇒ a *= 2;

(v) % equal to
⇒ a = a % 3 ⇒ a %= 3;

(vi) / equal to
⇒ a = a / 3 ⇒ a/=3;

Page No. :
Date :

3} Binary Operators : Needs two operands
a+b, a-b, a*b

4} Unary Operators : Needs one operand
a+=1, b-=2, a++, b--, --a

post {
- a-- decrement ⇒ a-=1
- b++ increment ⇒ b+=1

(uses & assigns then changes)

pre {
- --a decrement ⇒ a-=1
- ++b increment ⇒ b+=1

(changes then uses & assigns)

eg: ① age = 10
new age = ++a
⇒ (i) age value update age = 11
then assigns new age = 11

② age = 10
new age = a++
⇒ (i) old value stored
new age = 10 then increments
age = 11 (& assigns new age = 11)

5] Comparison Operators
(6] Conditional Statements)
6] Logical Operators

Date :

eg:→ ① What is value of each variable in each line of code ?

```
let num = 5;          → num = 5
newnum = num++;       → num = 6, newnum = 5
newnum = ++num;       num = 7, newnum = 7
```

* Identifier Rules
(Identifiers - Variables)
- Names must begin with letters, $, _
- Names can contain letters, digits, (-) underscores & dollar signs. (no space)
- Names are case sensitive (y & Y diff)
- Reserved (JS keywords) words can't be used as names.

eg :- 1] _point 4] old_price
2] point123 5] $price
3] price_ 6] Price

Invalid : 1price, 1 one price, int, var let

* Way of writing identifiers.

1. Camel case [camelCase] myFullName;
   ↳ Java & c++
2. snake_case my-full-name python
3. PascalCase MyFullName

Typescript : static type
JS is dynamic type (can be changed)
: Typescript is strict version of JS in which we can't change data type of any another variable once defined.
: Designed by MS.

String Index / String Indices
index means position.
In JS index starts from '0'.

"	N	T	O	N	Y		S	T	A	R	K	"
0	1	2	3	4	5		6	7	8	9	10	

"	T	O	N	Y		S	T	A	R	K	"
0	1	2	3	4	5	6	7	8	9		

: We can access individual character of String.

Page No. :
Date :

eg :] let name = "Tony"

T	O	N	Y
0	1	2	3

: name [0] : name [2]
'T' 'N'

: name [1] : name [3]
'O' 'Y'

↦ : for length
name.length / "Tony".length
4 4

: type of name.length
number

: name [name.length - 1];
'y'

: name [name.length - 2];
'n'

: "Tony" [0] : "Tony" [2]
'T' 'n'

: "Tony" [1] : "Tony" [3]
'o' y

:→ Concatenation : adding strings

① "Tony" + " " + "Stark"
= "Tony Stark"

② "Tony" + 1 = "Tony1"

Page No. :
Date :

* Console.log

: Proper way to print text on window.

- for string eg : console.log ("Hello");
  o/p : Hello
- for num eg : num = 100;
  console.log ("num");
  o/p : 100
- for more eg : num = 50;
  console.log ("num", "Hi", num);
  o/p : num Hi 50

calculation eg : n = 1;
n1 = 2
console.log (n1+2, n+n1);
~~o/p~~ : ~~12 n1~~
o/p : 12 3

: The JS is not connected to our browser so if we want something which should print on browser then we use console.log for it.
: This is the way to connect JS with browser console.

* Link JS

<script src = "JS filename"></script>

- Comments in JS

  command + forward slash

  ⌘ + /

o/p : // this is a comment

Template literals

```
let a = 10;
let b = 20;
```

- `console.log ("The total price is :", a+b, "Rupees.");`

  o/p : The total price is : 30 Rupees.

- `string1 = "The total price is :" + (a+b) + "Rs";`
  `console.log (string1)`

o/p : The total price is : 30 Rs.

`let string1 = ` The total price is: ${a+b} Rs `;`

${ } ← for expressions embedded. o/p

` ` ← Backtick

`console.log (` The total price ${a+b} Rs `);`

Page No. :
Date :

5] Comparison Operators

>	greater than
>=	greater than equal to
<	Less than
<=	less than equal to
==	equals to equals to (values comp
!=	not equal to
===	(also work for type of)

eg: age = 10;
if age > 18 //then true else false
false

- 5 != 4;
  true
- 5 == 5;
  true
- 5 == 'not';
  true
- 5 === 'not';
  false
- 0 == '';
  true
- 0 === '';
  false
- 0 == false
  true
- 0 === false
  false

- comparison for non-numbers
  It is possible because of unicodes.

'a' > 'A'
true

} - letters are not compared.
unicodes are compared

Date :

- Statements

1 Conditional Statements

(i) if-else

syntax

```
if ( condition )
 {
    do this;
 }
else
 {
    or this;
 }
```

} if Statement (more than 1 if statements can be also used)

(ii) else if

```
if ( condition1 )
 { do this;
 }
else if (condition 2)
 { do this;
 }
else if (condition 3 )
 { do this;
 }
```

if means each if's statement will be checked.

'else if' is only check true statements. (if 1st statement gets true then next is not checked in else if)

(iii) else

```
{ do this;
}
```

→ for applying else the all above statements should be false

Page No. :
Date :

(iv) Nested if-else

• Nested is writing if-else inside if-else

Syntax :

```
if ( cond)
 {
    if ( cond) { - - }
    else (cond) { - - }
 }
else
 { - - - }
```

6 Logical Operators

(i) Logical AND (&&)

(exp1) && (exp2)

both should true then true.

: true && true
o/p true

: true && false
o/p false

: false && false
o/p false

: false && true
o/p false

eg: (3>2) && (1>1)
false

(ii) Logical OR (||)

(exp1) || (exp2)
one should true at least.
: true || true
o/p : true
: true || false
o/p : true
: false || false
o/p : false
: false || true
o/p : true

(iii) Logical NOT (!)
: !true
o/p false
: !false
o/p true

Truthy & Falsy
Everthing in JS is true or false means boolean value.
It doesn't change its original value but when it is written in boolean context then it is taken as true or false.
eg:

- Falsy values :
  false, 0, -0, "" (empty string), null, undefined, NaN (Not a Number), 0n (BigInt value)

- Truthy values : Everything else.

eg:

```
let num = 0;

if (num)
  { console.log ("The num is not 0");
  }
else
  { console.log ("Num is zero");
  }
```

o/p Num is zero

2] Switch Statements

: cases means no. of conditions.

: Used when we have some fixed values that we need to compare.

: Break is used for breaking flow. If we don't used then next conditions will all executed.

: default is same as else.

Page No. :
Date :

Syntax:

```
let color = "red";

switch (color) {
    case "pink":
    console.log ("Wrong turn");
    break;
  case "red":
     console.log ("Stop");
     break;
  case "yellow":
     console.log ("Go");
     break;
  default :
     console.log ("Choose other color");
     }
}
```

: o/p Stop

Alert & Prompt

★ alert ("Danger");

: Similar to pop-up window.

: for removing alert we need to press ok.

Page No. :
Date :

- Some different consoles o/p
  console.error ("This is an error");
  : This will display as error in window.
  console.warn ("This is a warning");
  : This will display as warning in window.

★ Prompt
: for some input taking from user.
(similar popup)

```
let name1 = prompt("Enter first name");
let name2 = prompt("Enter 2nd name");
console.log("name1, + " " + , name2);
```

* Strings : Strings are immutable in JS
  - No changes can be made to strings.
  - Whenever we try to apply any method to strings then it makes new string but the original string remains same.

eg:
```
let str = "Hello   ";
let msg = str.trim();
msg;
"Hello"
str;
"Hello   "
```

Page No. :
Date :

String Methods

Methods are actions that can be performed on objects.

Here are some methods will only applicable on string.

(console.log() similar method)

Format:

StringName.method()

(: if there is () parenthesis infront of any noun it is considered as a method.)

(i) trim() : To remove whitespaces of the edges (ie. start or end).

eg: let string = " Hello ";
console.log(string.trim());
o/p 'Hello'

[: This creates new string it doesn't replaces the original string]

(ii) toLowerCase()

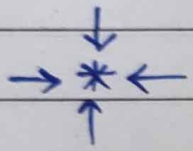
String.toLowerCase()

"hello"

(iii) toUpperCase()

String.toUpperCase()

"HELLO"



String Methods with Arguments

Argument is some value which is passed to method.

: arguments are passed for making changes

stringName.method(arg)

(iv) indexOf(arg)

```
let str = "pink";
str.indexOf("p");      // 0

let str = "colorIsPink"
str.indexOf("Is");     // 5
str.indexOf("z");      // -1
```

: This method is used for index searching.

Page No. :
Date :

Methods Chaining

To join/ apply more than one method i.e. called method chaining.

```
let msg = " red "
msg.trim().toUpperCase()

"RED"
```

(v) Slice (arg)

:-This cuts & returs the part of original string & makes new string.

```
let str = "ILoveCoding"
```

starting index ⟷ ending index (exclusive)

```
str.slice(1,5)    // "Love"
```

→ : This cuts from 1st index & goes to 4th & the last 5th place is not considered.

⟶ starting index

```
str.slice(5)    // "Coding"
```

→ : This automatically takes end index as length(ending)

```
str.slice(-2) = str.slice(length-2)
    : str.slice(11-2) = str.slice(9)
      = ge
```

(vi) replace (arg)

let str = "ILoveCoding";

str.replace ("love", "do"); // IdoCoding

: Replaces the original to new value.

: only one value (same if) replaced at a time.

: also for one alphabet

str.replace ("o", "x"); // ILxveCoding.

: for more than one replacement:

→ Str.replace ('o', 'x').replace ('o', 'x'); // ILxveCoding

(vii) repeat

: Repeats to the no. of arguments given.

if str = "go";

str.repeat (5);

"gogogogogo"

Array (data structure)

linear collection of things.
: Data types of different type also can be stored in one array.

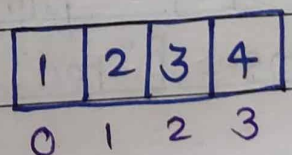
Syntax:

```
name = ["abcd", 123, 1.23]
```

eg:

```
samedata = ["pink", "red", "blue"];
diffdata = ["pink", 1, 1.236];
num = [1, 2, 3, 4];
```

visualization :




```
num[0];  // 1
num[3];  // 4
num.length;  // 4
typeof num;  // 'object'
samedata[1]  // 'red'
```

for empty array:

```
num = [];
num.length  // 0
```

: for accessing index of index of array

eg:

```
nams = ["red", "white", "pink"];
nams[1][2]  // 'i'
nams[2][3]  // 'k'
name[0].length  // 3
```

* Arrays are mutable

: While applying any method on arrays then the array gets edited. Means new array is not created. The original array changes its value.

```
nams = ["red", "white", "pink"];
nams[2] = "purple";
nams = ["red", "purple", "pink"];
```

: We also can store any element at any index.

```
nams[6] = "green";
```

```
nams = ["red", "purple", "pink", empty x2,
        "green"];
nams.length - // 6
```

this makes an empty space

Page No. :
Date :

Array Methods

(i) Push : add to end

```
names = ["anu", "tanu"];
names.push("manu");
names = ["anu", "tanu", "manu"];
```

(ii) Pop : delets from end

```
names.pop();
names = ["anu", "tanu"]
```

(iii) Unshift :- add to start

```
names.unshift("manu");
names = ["manu", "anu", "tanu"];
```

(iv) shift : delets from start

```
names.shift();
names = ["anu", "tanu"];
```

Array methods with arguments

```
names = ["anu", "tanu", "manu"];
```

(i) includes : returns true if presence of item else false. [includes (arg);]

eg: names.indudes ("anu");

true

(ii) indexOf() : returns index no.
[indexOf(arg)];

: names.indexOf("anu");
0

: names.indexOf("manu");
2

: names.indexOf("shradha");
-1

(iii) Concatination [concat()]
[concat(arg)]

: Joins two arrays (doesn't change original array instead creates new array)

colors1 = ["red", "pink"];
colors2 = ["green", "black"];

let colors = colors1.concat(colors2);

colors = ["red", "pink", "green", "black"];

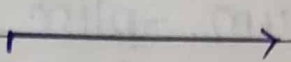
(iv) reverse();
: changes original array & reverses items
colors1.reverse();
colors1 = ["pink", "red"];

(v) Slice (arg)

: copies the original part of array.

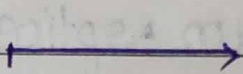
nums = [1, 2, 3, 4, 5];

eg:



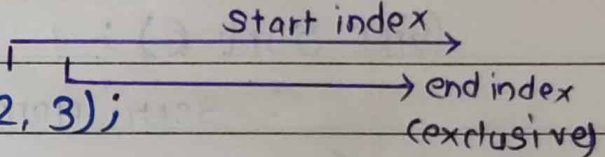
(1) nums.slice();

nums1 = [1, 2, 3, 4, 5];



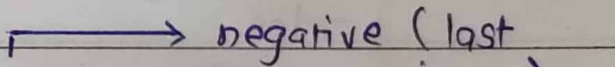
(2) nums.slice(2);

nums2 = [3, 4, 5];



(3) nums.slice(2, 3);

nums3 = [3];



(4) nums.slice(-2);

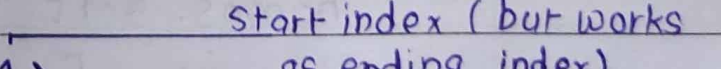
nums4 = [4, 5];

* (vi) Splice

: removes/replaces/add items in place, changes original array.

splice(start, deleteCount, item0...itemN)

num = [1, 2, 3, 9, 8, 9, 10];



(i) num.splice(4)

num = [1, 2, 3, 9, 8]

0 1 2 3 4

start index (delete) ← → end index (exclusive)

(ii) num.splice (0, 1);
num = [2, 3, 9, 8];

start index, delete count → new items

(iii) num.splice (2, 0, "red", "grey");
num = [2, 3, "red", "grey", 9, 8];

(iv) num.splice (2, 2, 5, 6);
num = [2, 3, 5, 6, 9, 8];

(vii) sort ();

: sort items if string is there it will be (ascending or descending order) or alphabetical order.

: The numbers are not sorted by ascending or descending orders. because sorting is done by string precedence.

: So, mostly we use sort for either strings or characters.

eg: names = ["abc", "xyz", "bmw"];
name = ["d", 'c', 'a', 't'];

names.sort();
names = ["abc", "bmw", "xyz"];
name.sort();
name = ['a', 'c', 'd', t];

Page No. :
Date :

* Array references → address in memory

[1] === [1]
false
[1] == [1]

: If there is a string & we are comparing the string then they are compared on the basis of their values. [ie. "name" == "name" is true]

: If there is an array then the arrays are compared on basis of their references means address where they are stored.

: so if we want to get equal value then we have to store variable of array in same. because every variables stores the address of array, not the value of array.

```
let arr = ['a', 'b', 'c'];
let arrcpy = arr;
    arrcpy == arr;
    true
    arrcpy === arr;
    true
```

} true because here the addresses are compared

arr 'this reference variable always stores address of array. (variable)

Page No. :
Date :

* Constant Arrays

As this is depend upon the address value so we can't change the overall array but the array can be edited after making it constant.

eg: const arr = [1, 2, 3];
arr.push(4);
arr = [1, 2, 3, 4];
let arr = [1, 2, 3, 4] ; ← This can't be done
(means creating new ~~array~~ can't be done)

* Nested arrays

: array of arrays / multidimensional array

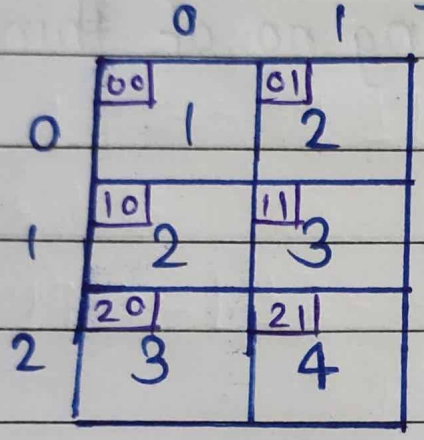
```
nums = [[1,2], [2,3] [3,4]];

nums[0];        // [1,2]
nums[0][1]      // 2
nums.length     // 3
nums[1].length  // 2

nums = [[1,2], [2,3], [3,4]];
```

Page No. :
Date :

: Nested array visualization



nums[row][col]

ie. nums[0][0]

= 1

Page No. :
Date :

Loops

: for repeating no. of things

1. for loop

Syntax:

```
for (initialisation; condition; updation)
    {
      // do --
    }
```

eg:

```
for (let i=0; i<=5 ; i++)
    {
      console.log (i);
    }
```

o/p

```
0
1
2
3
4
5
```

Dry Run :→

In case of dry run the first initialisation is checked (i=0) then(ending) condition ie i<=5 is checked after than one we go into the program & execution is done ie. [console.log (i)] after that updation condition ie. i++ [i=1] then ending condition (i<=5) then into program & so on.

Page No. :
Date :

for odd numbers :

```
(i=1; i<=20; i+=2)
```

for even :

```
(i=2; i<=20; i=i+2)
```

for reverse (odd) :

```
(i=20; i>=1; i-=2)
```

for reverse (even):

```
(i=20; i>=2; i-=2)
```

: Infinite loop : Crashes the site:

eg: `for (i=1; i>=5; i++)`

`for (let i=1; ; i++)`

* Multiplication table:

```
let n= prompt("Write no.");
n = parseInt(n);
for (let i=n; i<=n*10; i+=n)
  {
    console.log(i);
  }
```

As prompt converts text into string. So, with help of 'parseInt' the integer value gets generated from string.

Date :

[2] * Nested loop

loops into loops.

```
for (let i=1; i<=3; i++) {
    for (let j=1; j<=3; j++) {
        console.log (ij);
    }
    console.log ("new");
}
```

o/p: 11
12
13
'new'
21
22
23
'new'
31
32
33
'new'

[3] while loop

syntax

```
while (condition) {
    // do --
}
```

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

o/p : 1
2
3
4
5

Here 1st condition gets ✓ & then goes into loop & o/p given. As there is increment then loop goes to next iteration. & upto when the value of i is not equals to that condition the loop goes on. When the condition is false the loop is stopped.

eg:- Favourite movie.

```
~~gtet~~ ~~Prompt~~ let favM = "Thor";
let guess = prompt ("Enter fav movie");

while ((guess != favM) && (guess != "quit"))
{
  guess = prompt ("wrong, try again");
}
if (guess == favM) {
        console.log ("congratulations");
} else { console.log ("you quit"); }
```

Page No. :
Date :

* **Break keyword**

Breaks from loop & goes out of loop means to new iteration.

loops with Arrays

```
let fruits = ["one", "two", "three"];
for (let i=0; i<fruits.length; i++)
  {
    console.log (fruits[i]);
  }
```

o/p: 'one'
'two'
'three'

Loops with Nested Arrays.

```
let cams =[["one","two"],["three","four"]];

for ( let i=0; i<cams.length; i++) {
    for (let j=0; j<cams[0].length; j++)
    {
      console.log (cams[i][j])
    }
}
```

Page No. :
Date :

3 For of loop
: used for mostly collection of items.

```
let fruits = ["mango", "apple", "banana"];

for (fruit of fruits)
    { console.log (fruit) }
```

o/p : 'mango'
'apple'
'banana'

```
let fruit = "pineapple";

for ( char of fruit) {
        console.log (char);
    }
```

or

```
for (char of "pineapple"){
    console.log (char);
    }
```

o/p // 'p' 'i' 'n' 'e' 'a' 'p' 'p' 'l' 'e'

Page No.:
Date :

:→ Nested for of loop

```
let heros =[["batman","ww"],["thor","hulk"]]

for (list of heros) {
        for (hero of list) {
            console.log (hero);
        }
}
```

op: batman
ww
thor
hulk

Object literals

Used to store keyed collections & complex entities ie. values.

: In these the order is not defined it's random

property = (key, value) pair.

So, the object is collection of these properties

[key: value,]

```
let delhi = {
        latitude: "28.704° N"
        longitude: "77.1025 E"
    };
```

```
const student = {
    name: "shradha",
    age: 23,
    marks: 99,
    city: "Delhi",
    gender: "female"
};
```

:- object converts values into strings.

:→ Object literals are made constant which are similar to array constant.

: also can store array:

eg:

```
const item = {
        colors = ["red", "pink"];
        names = ["pencil", "pen"];
        }
```

→: How to access object literal values

(i) student["name"] or
student.name
o/p 'shradha'

(ii) item["colors"]; or
item.colors
o/p ["red", "pink"]

Page No. :
Date :

Adding / Updating values in object literals

```
const student = {
        name: "shradha",
        city: "Delhi",
        age: 23
       }
```

: for adding new value:

```
student.marks = 94.4;
```

: for changing value:

```
Student.city = "Mumbai";
```

Nesting object literals

```
const students = {

        aman: {
                grade: "A",
                age: 30},
        shradha: {
                grade: "A+",
                age: 23}
}
```

: for accessing:

(i) students.aman

o/p { grade: 'A',
age: 30
}

(ii) student.aman.age

o/p 30

: for changing/adding. similar to object literals.

(i) students.aman.age = 40;

Array of objects

```
Const students =[
    { name:"aman",
      grade: 'A',
      age : 30
    },
    {
      name: "shradha",
      grade: 'A+',
      age: 23
    }
]
```

(i) students[0]

o/p : {
name: "aman",
grade: 'A',
age: 30
}

(ii) students[1].grade // 'A+'

(iii) students[1].grade = 'B+';
↳ for updating

(iv) students[0].gender = "male"
↳ for new addition

Math Object

(i) Math.abs (arg)
: same as modulus function
Math.abs (12) or Math.abs (-12)
o/p 12

(ii) Math.pow (arg, arg)
: power function a^b
Math.pow (2,3)
o/p 8

(iii) Math.floor(arg)
: Rounding off the number to less than / smallest integer value.

eg: Math.floor (1.2341)
o/p 1

(iv) Math.ceil(arg)
: Rounding off the number to nearest greater integer.

eg. Math.ceil (1.2341)
o/p 2

(v). Math.random()
: gives any random number