Nand
Date
Page

// → Comments
Console.log("hello World");
// variable
var fFrstName = "Harshit"; or fFrstName = "Harshit";
// use a variable
Console.log(fFrstName);
// Change value
fFrstName = "prince";
Console.log(fFrstName);
# Naming of variable · rules. (only use $ · or underscore (_)
1value (Invalid) $value (valid) fFrst Name (Invalide)
value1 (valide) _fFrstname (valid.) fFrst$Name (valide)
→ var first_Name = "harshit"; // Snake Case writing
→ var fFrstName = "harshit"; // Camel Case writting
→ Convention
// start with small letter and use Camelcase.
let Keyword.
let fFrstName = "harshit"
let fFrstName = "harshit";
let fFrstName = "hero";
Console.log(fFrstName);
↓ or error
let fFrstName = "harshit";
fFrstName = "prince";
Console.log(fFrstName);
→ prince
main diffrence let & var Ps block Scope vs functPon
Scope (Covered latter Pn thPs VPdeo)

"use strict"; → error dedecte

// declearie Constant.
Const pP = 3.14 ;
// pP = 3.15 ; //error pP is Constant not change
Console.log(pP)
// String Indexing.
let firstName = "princekumar";
// p r i n c e
// 0 1 2 3 4 5
Console.log(firstName[0]) ;
Console.log(firstName.length); include spaces in length.
print last Indexing - Consol.log(firstName[firstName.length-2]);
# method for string.
// trim()
// toUppercase()
// tolowercase()
// slice()
let firstName = " harshit@. " ;
Console.log(firstName.length);
let newString = firstName.trim(); // "harshit" new variable
Console.log(newString.)
Console.log(newString.length)
→ //old variable
firstName = firstName.trim()
Console.log(firstName).
→ Console.log(firstName.toUppercase())
// String is immutable.
firstName = firstname.toUppercase()
print(firstname.
small to tolowercase
# slice let newString = firstname.slice(1,3) ;
slice(1); → (1 to end print) print(newString)

Nand
Date
Page

#1 type of operator
datatype - string, number, boolean, BPgint, undefined, null, Sm

```
let age = 22;
let name = "harshil";
Console.log(typeof age)
// Convert number to string
age = age + " "
Console.log(typeof(age));
Console.log(string(age));
// Convert string to number
let mystr = +"34"
Console.log(typeof mystr);
```

String Concatenation.

```
let string1 = "prince";
let string2 = "kumar";
let name = string1 + " " + string2;
print(name);
let str1 = "17";
let str2 = "8";
let num = number(str1) + number(str2);
print(num);
```

template string

```
let age = 22;
let name = "prince";
//let aboutme = "my name is" + name + "and my age is" age;
print(aboutme);
```

use template string

```
let aboutme = `my name is ${firstname} and my age is
${age}`;
print(aboutme)
```

Nand
Date
Page

undefine

```
let name;  , var name
Console.log(typeof name);
not for const.
name = "prince";
print(typeof name)
let var1 = null;
print(var1);
var1 = "prince"
print(var1, typeof var1);
Console.log(typeof null); bug, error
```

BigInt.

```
let Mynumber = 123;
// Console.log(Mynumber);
Console.log(Number.MAX_SAFE_INTEGER);
let MyNumber = BigInt(12--------);
    Mynumber = BigInt(12.);
    Mynumber = 12n;
Console.log(Mynumber + mynumber);
```

→ booleans & Comparison operator.

```
let num1 = 5;
let num2 = 7;
Console.log(num1 > num2); False.
```

== Vs ===

```
let num1 = "7"
let num2 = 7;
Console.log(num1 == num2); only check value not datatype
Console.log(num1 === num2); value + datatype check.
Console.log(num1 != num2) → False.
Console.log(num1 == num2) → True.
```

Nand
Date
Page

truthy & Falsy values.
If else condition.

```
let age = 17;
if (age >= 18){
Consol.log("play ddlc");}
else{
 Consol.log("play mario");}
let num = 13;
if (num = 13);
 if (num%2 == 0){
 Console.log("even");}
  else{ Consol.log("odd");}
```

→ truth & Falsy value

Falsy value - false, "", null, Undefined, 0

```
let Name = "prince";  , name = "", null, 0, False → else
if (firstname){                                    execute.
Conede.log(name);}      name = 1, -1 → if executed.
else{
 Console.log("name is empty");
```

→ truthy value = "abc", 1, -1

ternary operator.

```
let age = 15;
 let drink;
if (age >= 5){
  drink = "Coffee";}
else{ drink = "milk";}
Console.log(drink);
```

use ternary op. or conditional operator.

```
let age = 8;
let drink = age >= 5 ? "coffee" : "milk";  console.log(drink);
```

and or operators.

```
let firstname = "prince";
let age = 22;
if (firstname[0] === "H" && age > 18){
    Console.log("name.");}
else{
    Console.log("Inside else");}
```

Similar to or.

nested if else.

```
let WinningNumber = 19;
let userGuess = +prompt("Guess a number");
if(userGuess === winningnumber){
    Console.log("right Guess");}
else{
  if(userguess < winningnumber){
    Console.log("too low");}
  else{
    Console.log("too high");}
```

else if

```
let temp = 50;
if (temp < 0){
    Console.log("low");}
else if (temp < 15){
    Console.log("medium");}
else{
    Console.log("high");}
```

Switch statement.

```
let day = 2;
switch(day){ case 0: Console.log("Sunday");
    break;   case 1: Console.log("Monday");
             case 2: Console.log("tuesday");
  default:
  Console.log("invalid");
  break;}
```

Nand
Date
Page

While loops

```
let i = 0;
while(i<=9){
Console.log(i);
  i++;}
Console.log(`current value of i is ${i}`);
```

1st n natural number sum.

```
let num = 10; let i=0
let total = 0;
while(i<10){
  total = total + i;
    i++;}
Console.log(total);
```

Forloop.

```
for(let i = 0; i<=9 ; i++){
  Console.log(i);}
Console.log("value of i is", i); error.
```

when `for(var i=0 ; i<=9 ; i++){ }`

```
    Console.log(i); i=0 output not error.
```

Sum of n number sum

```
let total = 0;
let num = 10
for(let i=1 ; i<num; i++){
  total = total + i;}
Console.log(total);
```

break & Continue.

```
for(let i= 1 ; i<= 10 ; i++){
  if(i==4){
    break;}
  Console.log(i);}
```

o/p - 1, 2, 3.

```
if(i==4){
  Continue;}
Console.log(i);}
```

1, 2, 3, 5, 6, 7, 8, 9, 10.

Other is premitive data type. one value store.

// do while loop.

```
let i = 10;
do{ console.log(i);
    i++; }
while(i<=9);
console.log("value of i is", i);
```

Intro to arrays. - reference type. → object hai.
→ ordered collection of item. array is mutible.

```
let fruits = ["apple", "mango", "grapes"];
    Console.log(fruits[2]);
let num = [1, 2, 3, 4];
Console.log(num);
let fruits = ["apple", "mango", "grapes"];
  Console.log(fruits);
  fruits[1] = "banana";
Console.log(fruits);
Console.log(type of fruits); → objects.
Console.log(Array.is Array(fruits)); → true.
let obj = {}; // object literal.
Console.log(Array.is Array(obj)); → false.
```

push & pop in array

```
let fruits = ["apple", "mango", "grapes"];
Console.log(fruits);
fruits.push("banana"); → add last in array.
Console.log(fruits)
```

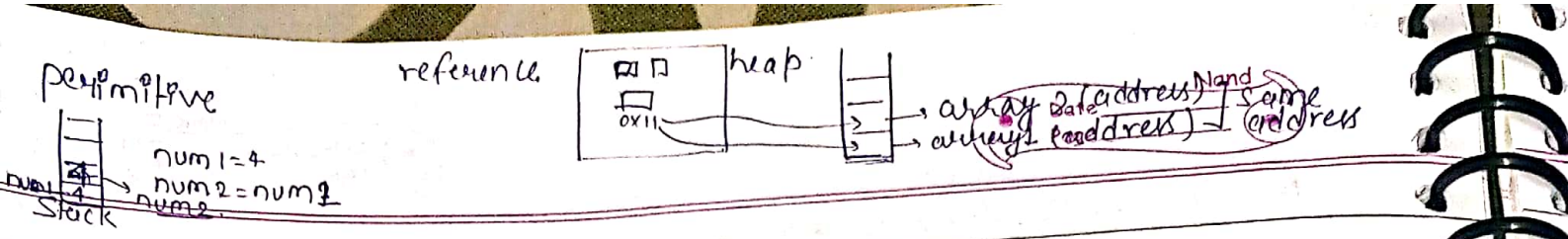
→ refrence type ko original type ko change kar skte hai but permitive type (string, number etc) ko change nahi kar skte hai;

→ fruits.pop(); Console.log(fruits);

```
let poppedfruits = fruits.pop();
Console.log(popedfrupts);
# unshift: → add starting on array.
Fruits.unshift("banana");
Console.log(fruits);
# shift: (remove from first)
  let removedfruits = fruit.shift();
  Console.log(frupts);
** push, pop is fast than shift, unshift.

  # primitive v/s refrence data type
→ primitive type.
let num1 = 6;
let num2 = num1;
Console.log(num1);  6
Console.log(num2);  6
num1++;
Console.log(num1);  7
Console.log(num2);  6.

→ refrence type. → array.
let array1 = ["item1", "item2"];
let array2 = array1;
Console.log(array1);   it1, it2
Console.log(array2);   it1, it2
array1.push("item3");
Console.log(array1);   it1, it2, it3
Console.log(array2);   it1, it2, it3.
```



Colon. (copy)

```
let array1 = ["item1", "item2"];
let array2 = ["item1", "item2"];
array1.push("item3")  or
Console.log(array1 === array2);
Console.log(array1);
Console.log(array2);
```

** M-2. `let array2 = array1.slice(0);` for colone.

```
array1.push("item3");
Console.log(array1 === array2);
Console.log(array1);
Console.log(array2);
```

M3. `let array2 = [].Concat(array1);`

```
array.push("item3);
```

M4- Spread operator.

```
let array2 = [...array1];
array1.push("item3");
Console.log(array1 === array2);
```

* merging array.

```
let array2 = array1.slice(0).Concat(["item3","item4"]);
```

* `let onemorearray = ["p", "q"];`

```
let array2 = [...array1, ...onemorearray];
Console.log(array2);
```

Nand
Date
Page

for loop on array.

```
let fruits = ["apple", "mango", "grapes", "banana"];
  for(let i=0 ; i<=9 ; i++){
    Console.log(i); }
Console.log(fruits.length);
for(let i=0 ; i<fruits.length ; i++){
    Console.log(i);
    Console.log(fruits[i]); }
    Console.log(fruits.touppercase());
    fruits2.push(fruits.touppercase());
```

- use const for creating array.

```
Const pi = 3.14.
Const fruits = ["apple", "mango"]; //0x11
  fruits = ["grapes", "pineapple"] error.
  fruits.push("banana");
  Console.log(fruits);
```

→ jab refrence type use karo to const use karo.

```
* Const fruits = ["apple", "mango", "grapes"];
  Const fruits = [];
  let i=0;
  while(i<fruits.length){
    fruits2.push(fruits[i].touppercase());
    i++ }
  Console.log(fruits2);
```

→ i, k, j

```
→ for(let fruits of fruits){
    Console.log(fruit); }
→ for(let index in fruits){
    Console.log(fruits[index]); }.
```

Nand
Date
Page

Array destructuring

```
Const myarray = ["value1", "value2", "value3", "value4"]
let myvar1 = myarray[0];
let myvar1 = myarray[1];
Console.log(myvar1), (myvar2);
```

in javascript. , normal variable.

```
let [myvar1, myvar2] = myarray;
    [myvar1, , myvar2] = myarray;
Console.log("value of myvar1", myvar1, myvar2);
```

→ let mynewarray = myArray.slice(2);
Console.log(mynewarray); → 2 to 4 index print.

→ let [myvar1, myvar2, ...mynewarray] = myarray;
Console.log(mynewarray);

☆ Objects (refrence type)

→ Arrays are good but not sufficient
→ For real world data
→ Object store key value pairs
→ Object don't have index.
→ Creat a objects.

```
Const person = {name: "prince", age: 22};
Console.log(person), typeof person.
Console.log(person.name);
→ Const person = {name: "p", hobbies: ["g", "s", "l"]};
Console.log(person.hobbies);
person.gender = "male"; → add in object.
Console.log(person);
Console.log[person["name"]];
Console.log(person["age"]);
Const key = "email"; person[key] = "harshitv@gmail.com";
```

loop in object:

```
for(let key in person){
  Console.log(person[key]); // only print value.
  Console.log(key); // only print key.
  Console.log(`${key}: ${person[key]}`); // key & value print.
  Console.log(key, ":", person[key]);
→ Console.log(typeof(Object.keys(person)));
  Console.log(Object.keys(person));
  Const val = array.isArray((Object.keys(person)));
  Console.log(val); // true
→ for(let key of Object.keys(person)){
    Console.log(person[key]); }
// Computed properties.
  Const key1 = "objectkey1";
  Const key2 = "objkey2";
  Const value1 = "myvalue1";
  Const value2 = "myvalue2";
  Object creation.
  Const obj = {
  [key1]: value1,
  [key2]: value2, }
m-2
const obj = {};
obj[key1] = value1;
obj[key2] = value2;
Console.log(obj);
# Spread operator.
  Const array1 = [1, 2, 3];
  Const array2 = [5, 6, 7];
Const newarray = [...array1, ...array2]; // spread eg.
  Console.log(mynewarray); 1, 2, 3, 5, 6, 7 in newarray
```

Stud

Const newarray = [..."abc"];
newarray = [...array1, 8, 6, 5];
// spread operator on objects
Const obj1 = {
key1 : "v1",
key2 : "v2";
key1 : "v3"; }

obj2 = { key1 : "v11",
key4 : "v6",
key5 : "v1" }

→ repeated key according to last spread (obj1)

Const newobject = {...obj2, ...obj1};

Const newobj = {..."abc"};

~~Destructuring object~~

object destructing.

Const band = { bandName : "led zepplin",
famoussong : "stairway to heaven"};

let bandName = band.bandname;
Const famoussong = band.famoussong;
bandName = "queen";
Console.log(bandname, famoussong);

Short

Const { bandname, famousesong } = band;
Console.log(bandname);
Console.log(band);