JavaScript : Module ①

__/__/____

① Printing :

→ In JS "Printing" typically refers to displaying output to the user or developer.

(a) Basic Printing

- prints normal messages to the console & commonly used for developers for debugging or showing output.

```
ex : console.log ("Hello!");          // Hello!
```

(b) Without Newline

- Prints directly to standard output without adding a new line.

```
ex : process.stdout.write ("A");
     process.stdout.write ("B");      // AB
```

(c) Error Message

- Prints message in red & used for errors or exception.

```
ex : console.error ("Error :", new Error ("Invalid ID"));
                    // Error : Error : Invalid ID
```

(d) Warning Message

- Prints message in yellow/orange & used for warning about bad practices or upcoming issues.

```
ex : console.warn ("password is weak");
```

(e) Info message

- Prints in blue (Node.JS) or normal (browser) & used for informational messages.

ex: console.info ("user logged in");

(F) Tabular format

- Displays array / object in a table format

```
ex: console.table ([
    { name: "Alice", age: 25, city: "Mumbai" },
    { name: "Bob", age: 20, city: "Delhi" },
]);
```

(index)	name	age	city
0	Alice	25	Mumbai
1	Bob	20	Delhi

② Data Types:

(A) Primitive - are immutable + stored directly in memory by value.

(a) Numbers

```
let age = 25;                    // Integer
let price = 99.9;                // Decimal
let negative = -42;              // Negative Number
let notANumber = NaN;            // "Not A Number"
let infinity = Infinity;         // Special Numeric Value
```

③

(b) Strings

```
let John = "Doe" ;                    // Double quotes
let greeting = "Hello";
let templateLiteral = `Hello, ${name}`;   // Template literal
```

(c) Booleans

```
let isActive = true ;                 // true
let isComplete = false ;              // false
```

(d) Undefined

```
let unitialized ;                     // variable declared but not assigned
let x ;                               // undefined
let obj = {name: "John"};
console.log (obj.age) ;               // undefined (property x)
```

(e) Null

```
let emptyValue = null ;               // Explicitly set to "no value"
```

(f) Symbols (ES6+) [unique]

```
let id = Symbol ("id");
let anotherId = Symbol ("id");
console.log ( id === anotherID ) ;    // false
```

(g) BigInt

```
let bigNumber = 1234567891011213141516n ;
```

(B) Non-Primitive - Mutable + Stored in memory by reference.

(a) Objects

```
let A = { name : "A", age : 28, city : "Kolkata" };
```

(b) Array

```
let A = ["red", "yellow"]
```

(c) Functions

```
Function greet () {
    return "Hello !"; }
```

③ Type checking :

```
console.log (typeof 42) ;                    // number
console.log (typeof NaN);                    // number
console.log (typeof ("Hello"));              // string
console.log (typeof true);                   // boolean
console.log (typeof null);                   // Object
console.log (typeof {});                     // Object
console.log (typeof [];                      // Object
console.log (typeof Function () {});         // Function
```

this is a bug in JS Kept for backward compatibility.

* `Array.isArray(colors)` // true

__/__/____

④ Dynamic Typing :

- Same variable value changes type dynamically depending on the assigned value

```
let value ;                                    // undefined
console.log (typeof value);
value = 42 ;                                   // assigned Number
console.log (typeof value);
value = "Hello"                                // assigned String

value = true ;
                                               // assigned boolean
value = { name : 'Alex' } ;
                                               // assigned object
value = [1, 2, 3] ;
                                               // assigned array
value = function () { return "Hi"; };
console.log (typeof value);                    // Function
```

OUTPUT

① undefined
② number
③ string
④ boolean
⑤ object
⑥ object
⑦ function

Sequence of variable (value)

__/__/____

⑤ Coersion Typing :

→ Type coersion means JS automatically converts one DT to another type when needed & it happens at runtime.

(i) Implicit - JS does it automatically.

(ii) Explicit - You do it manually using fnc like number(), string(), boolean()

(i) Implicit

(a) String + Number → String

- when one operand is a string, JS converts the other to string & does concatenation.

```
console.log ("6" + 2) ;          // "62"
```

(b) Number - String → Number

```
console.log ("10" - 5);          // 5
```

(c) Boolean → Number

```
console.log (true+1) ;           // 2
```

(d) Null & Undefined

```
console.log (null + 1);          // (null→0)
consule.log (undefined + 1);     // (undefined → NaN)
```

(e) comparison

```
consule.log ("5" == 5);          // T  (string "5" → number 5)
console.log ("5" === 5);         // F  (No, coersion)
```

** checks type

__/__/____

(ii) Explicit

(a) Numbers

```
console.log(Number("123"));   // 123
console.log(Number(true));    // 1
```

(b) String

```
console.log(String(123));     // "123"
console.log(String(true));    // "true"
```

(c) Boolean

```
console.log(Boolean(1));      // true
console.log(Boolean(""));     // false
console.log(Boolean("hi"));   // true
```

(6) Var vs Let vs Const :

(A) Var

- Scope : Function-Scoped (or global if declared outside a function)
- Hoisting : Yes, the variable is hoisted & initialized as undefined.
- Re-declaration } → Allowed
  Re-assigned

```
var x = 10;    var x = 20;    x = 30;
console.log(x);          // 30
```

```
function testVar() { var y = 5; if (true) {
  var y=15;  console.log(y);  // 15   }} testVar();
```

(B) let

- Scoped : Blocked-scoped ({... })
- Hoisting : Yes, but not initialized
- Re-declaration - ✗
  Re-assigned - ✓

```
let a = 10;   a = 20;   console.log(a); // 20

if (true) {
    let b = 50;   console.log(b); // 50
}

console.log(b); // ✗ Error
```

(C) Const

- Scope : Block-Scoped
- Hoisting : Yes, but not initialized
- Re-declaration & Reassigned : ✗

```
const pi = 3.14;
// pi = 3.1415;   // Error ✗

const obj = { name: "Priya" };
obj.name = "Ravi";   // ✓
console.log(obj);   // {name: "Ravi"}
```

__/__/__

⑦ Operators + Precedence :

(A) Arithmetic

⑤	+	Addition / concat	5 + 3	8
	-	Subtraction	5 - 3	2
L-R ④	*	Multiplication	5 * 3	15
	/	Division	6 / 3	2
	%	Modulus (remainder)	7 % 3	1
R-L ③	**	Exponentiation	2 ** 3	8

(B) Assignment

	=	Assign value	x = 5
	+=	Add & assign	x += 3 → x = x+3
R-L ⑪	-=	Subtract & assign	x -= 2
	*=	Multiple & assign	x *= 3
	/=	Divide & assign	x /= 2
	%=	Modulus assign	x %= 5

(C) Comparison

⑦	==	Equal (coerces types)
	===	Strict Equal (no coersion)
	!=	Not equal
	!==	Strict not equal
L-R ⑥	>	greater than
	<	less than
	>=	greater or equal
	<=	less or equal

__/__/____

(D) Logical ⑧ L-R

```
let isLoggedIn = true;
let isAdmin = False;

console.log (isLoggedIn && isAdmin) ;   // F   AND
console.log (isLoggedIn || isAdmin);    // T   OR
console.log ( ! isLoggedIn) ;           // F   NOT
```

(E) Increment & Decrement

```
let count = 5;
count ++;      // 6
++count;       // 7
count --;      // 6
-- count;      // 5
```

(e) Ternary

R-L ⑩

```
let age = 18;
let status = (age > 18) ? "Adult" : "Minor";
console.log (status);      // Adult
```

① ( ) L-R

② ++ , -- , + , - , ! R-L

unary