

1. Difference between Var , const, Let

Var: is Function scoped it can used inside the function without any restrictions of braces { } for example

```
Function new(a){  
    if(a>2){  
        Var x =2  
    }  
    console.log(x)  
}
```

Here x will be printed no matter the braces { } of if statement because var is function scope

It can be re declared in the same scope i.e

Var x = 2

Var x = 3

No errors will be thrown

Let: is Block scoped it can only be used in the braces its defined eg globally or inside any braces unlike var that can be used any where inside a function without braces { } restriction eg

```
Function new(a){  
    if(a>2){  
        Let x = 2  
    }  
    console.log(x)  
}
```

This will not work because let is inside braces { } of if statement

It cannot be redeclared inside the same scope unlike Var

Const: it is same as let but it cannot be reassigned a value after its being assigned

Eg

```
const a = 2
```

```
a = 3  X
```

2. Primitive and Non-Primitive Datatypes

Primitive:

Primitive means low level values not composed of other parts

They are simple single value datatypes

- Is not an object
- Has no methods or properties
- Is immutable
- Is not passed by reference

There are 7 Primitive datatypes

- Number
- String
- Boolean
- Null
- Undefined
- Symbol
- BigInt

Why is num , string etc are primitive yet methods can be applied to them because methods are not being directly applied on them instad javascript temporarily wraps them into an object using a wrapper class when performing a method and after the method is performed they are turned back so methods are are not directly performed on the numbers etc

Undefined is the absence of the value eg if a variable is not assigned a value then then that variables value is undefined

Null is a primitive datatype yet if we do `typeof(null)` the answer would be object that's because of a bug in previous versions of javascript and resolving the bug could cause unexpected problems so null is a primitive datatype but javascript due to bug counts it as object

Non-Primitive Datatypes:

Non primitive datatype can hold multiple values and are composed of other primitive datatypes

Instead of storing value itself they store the reference and when doing comparison they are compared by reference

Eg `arr1 = [1,2,3]` , `arr2 = [1,2,3]`

`arr1 === arr2` is false both have same values but difference reference

- Object
- Array
- Date
- Function
- RegExp
- Map
- Set

3. ECMAScript:

- It is an standard specification that defines how javascript should work
- Javascript is an implementation of ECMAScript
- Current specification is called ECMA-262
- Javascript != ECMAScript

- Javascript includes ECMAScript and environment specific stuff like dom in browsers and fs in node js
- Ecma script is a standard not a language
- Javascript is a real world language
- Javascript engines like chrome , node , safari etc implement ECMAScript standard to keep javascript work consistently on all engines

4. Prototypes:

In javascript every object has an internal link to other object called prototype this forms a prototype chain

```
function Person(name) {
  this.name = name;
}
Person.prototype.greet = function () {
  console.log("Hello, I'm " + this.name);
};
```

```
const alice = new Person("Alice");
alice.greet(); // Hello, I'm Alice
```

What's Happening Here?

- Person is a constructor function.
- When you use new Person(...), JS:
 - Creates a new object
 - Sets its prototype to Person.prototype
 - Runs the constructor with this bound to the new object
- greet is available to all Person instances via prototype.

Primitive Type	Object Wrapper	Prototype Used For Methods
number	Number	Number.prototype.toFixed()
string	String	String.prototype.toUpperCase()
boolean	Boolean	Boolean.prototype.toString()

5. Null and Undefined:



- Absence of value means undefined eg
Let a;
console.log(a);
- Intentional Absence of value means Null eg
Let a = null;
console.log(a);
a = 2;
console.log(a)

6. Accessing objects:

- Dot notation eg let obj = {item: "one"}
console.log(obj.item)
- Bracket Notation eg let obj = {item: "one"}
console.log(obj[item])

Both give same output

Why Bracket Notation is Better:

- Property name is stored in a Variable eg
let key = "item";
console.log(obj[key]); // "one"
- Property name is not a valid identifier eg
let obj2 = {
 "some weird key!": 123
};
console.log(obj2["some weird key!"]); //  works
// obj2.some weird key!  syntax error

7. Alert, Prompt and TypeCasting:

Let a = prompt("hey whats your name")

Prompts the user to ask name and store it inside the variable a

Let a = prompt("give age")

Number.parseInt(a)

```
If (a > 0){  
    alert("this is a valid age")  
}
```

Prompts the user to input age typeCast the string age to number inside var a then checks condition for valid age and give alert

Prompt asks input , alert shows info