Javascript:

1) What are the primitive data types in Javascript?

=> Javascript primitive data types are data type that refers to a single value.
It mainly includes Number, String, Boolean, undefined Null, Symbol. Used to add unique property key to an object that won't collide

2) Explain the difference between Null and Undefined in Javascript.

=> undefined

* undefined means a variable has been declared but has yet not been assigned a value.
* It is an ECMAScript1 (ES1) feature
* It does not have any syntax But you can assign undefined to any variable but we do not do that.
* It is a global property

Null

* Null is an assignment value. It can be assigned to a variable as a representation of no value.
* It is a Primitive value in JS.
* Its syntax is:-
null
* It is not a global property

3) How do you check the data type of a variable in Javascript

We use typeof operator to find the data type of a Javascript variable.

For example: var name = 'John'
=> typeof ('John') => returns "string"

4) Explain the concept of truthy and falsy values in Javascript. Provide examples.

undefined, Nan, if block is not run. if the variable value is falsy and the code within the if block is run. if the variable value is anything else, such as number that is not zero, a non-empty string an array, or an object, it truthy and a code in the if block is run.

5) What is the difference between == and === operators in JavaScript, and how do they work to data type?

In javascript, the '==' and '===' operators are used for equality comparision, but they behave differently when it comes to data type.

'==' (Equality Operator)

* This operator perform type coercion, meaning it convert operands to the same type before making comparison.
* This can lead to unexpected results, as different data type may be considered, equal after conversion.
* For example, "5" == 5 would evaluate to 'true' because the string "5" is coerced to a number before comparison
  ⇒ console.log ("5" == 5) // true.

'===' (Strict Equality)

* This operator, does not perform type coercion.
* It checks both the value & the data type of the operands, ensuring that both are identical for the comparision to be true.
* If the type are different, '===' return 'false' without attempting to convert the values.
  ⇒ console.log ("5" === 5) // false

6) How do you convert string to a number in JavaScript?

We can convert a string to a number using various methods:

⇒ parseInt() (or) parseFloat() functions.

- parseInt() ⇒ convert a string to an integer.
- parseFloat() ⇒ convert a string to a floating-point no.

* var str = "456"
  var num = parseInt (str);
  (or)
  var floatNum = parseFloat ("123.45");

⇒ Unary '+' Operator:

* var str = "456";
  var num = +str;

⇒ Number() constructor:

var str = "789";
var num = Number (str);

7) Explain the difference between ++x and x++ increment operator in javascript.

=> Prefix increment operator (++x):

* The value of 'x' is incremented before its current value is used in the expression.
* The new value of x is returned after the increment operation.

Example let x = 7;
let y = ++x; // Now, x is 8 & y is 8.

=> Postfix increment operator (x++):

* In the case of this operator, the current value of 'x' is used in expression before it is incremented.
* The incrementation happens after the current value is used.
* The original value of 'x' is returned befor the increment operation.

Example let x = 7;
let y = x++; // Now, x is 7 & y is 8.

8) How does Javascript handle NaN (Not a Number) values and how can you check if a value is NaN? NaN is a special value representing the result of an operation that should return a number but does not. It often a result of invalid ⓜ undefined mathematical operations.

To check if a value is 'NaN', we can use the 'isNan()' function. It not only checks if a value is 'NaN' but also checks if the value is not a valid no. It returns true for values that are not number, as well as for actual Nan.

Example: let myNum = 10/'xyz'; // This result in NaN
console.log (isNan(myNum)); // This will log true.

To specifically check if a value is 'NaN' and not some other non-number, we can use 'Number.isNan()'

Example let myNum = 10/'xyz';
console.log (Number.isNan (myNum)); // returns true

let anotherNum = "hello";
console.log (Number.isNan(anotherNum); // returns false

9) Explain the concept of type coercion in JS with examples.

The process of converting values from one data type to another, either implicitly (or) explicitly.

=> Explicit Type Coercion:

This occurs when a developer intentionally converts a value from one type to another using functions (or) operators.

Example: 'Number()', 'String()' (or) 'Boolean()' function

* String to number

```
var stringNumber = "42"
var number = Number(stringNumber);
console.log(number);
```

* Number to string

```
var num = 123;
var str = String(num);
console.log(str);
```

=> Implicit Type Coercion:

This occurs when JS automatically converts value from one type to another during certain operation.

Example:

```
var num = 5
var str = "10"
var result = num + str
console.log(result); // O/P: "510"

var num = 5;
var bool = true;
var result = num + bool;
console.log(result); // O/P: 6
```

10) What is the purpose of the undefined data type, and when might it explicitly assigned to a variable?

'undefined' data type and 'undefined' value serve the purpose of representing the absence of a value (or) the uninitialized state of a variable. When a variable is declared but not assigned a value, its default value is 'undefined'.

```
let myVariable; // Declaration w/o assignment
console.log(myVariable); O/P undefined
```

We might explicitly assign it in case where you want to explicitly indicate that a variable has no meaningful value (or) hasn't been initialized.

```
let myVariable = undefined; (explicitly assigned)
console.log(myVariable); O/P undefined
```

11) How do you create and use template literals (String interpolation) in JS?

There are features introduced in ECMAScript6 (ES6) that allows for easy string interpolation and multiline strings in JS. They are created using backticks (` `).

Creating template literals:

```
const firstName = 'John'
const lastName = 'Doe'
const fullName = `${firstName} ${lastName}`;
console.log(fullName);
```

String interpolation:

```
const num1 = 5
const num2 = 7
const sum = `${num1} + ${num2} equals ${num1+num2}`;
console.log(sum); // O/P: 5+10 equals 15.
```

~~Modules Scopes~~

12) What is hoisting?

Hoisting is a behaviour in JavaScript where variable and function declarations are moved to the top of their containing scope during the compilation phase.

```
=> console.log(x); // O/P: undefined
   var x = 5;
   console.log(x) // O/P: 5

-> sayHi();
   function sayHi(){
     console.log("Hello, World!");
   }
```

Unlike var, let & const declarations are also hoisted but are not initialized with undefined. Accessing them before the declarations results in a "ReferenceError"

```
console.log(y);
let y = 20;

console.log(sayHi);
var sayHi = function(){
  console.log("Hi!");
};
```

13) What is IIFE?

Immediately Invoked Function Expression. It is a JS design pattern that involves defining & invoking a function immediately after creation of function

```
(function(a,b){
  var result = a+b;
  console.log(result);
})(5,7);
```

14) What is meant by Default parameter passing.

Default parameter passing refers to the ability to assign default values to parameters in a function, so that if a value is not provided when the function is called, the default value is used instead.

```
function greet (name="Guest")
{
    console.log ("Hello, $name!");
}
greet ();          // o/p Hello, Guest!
greet ("John");    // o/p Hello, John!
```

15) What is default return value in JavaScript?

If a func execution doesn't end at a return statement, (or) if the return keyword doesn't have an expression after it, then the return value is undefined.

16) How to pass unlimited no of parameter to a function.

The rest parameter syntax allows a function to accept an indefinite no of arguments as an array,

```
function f (a, b, ... theArgs) {
}
```