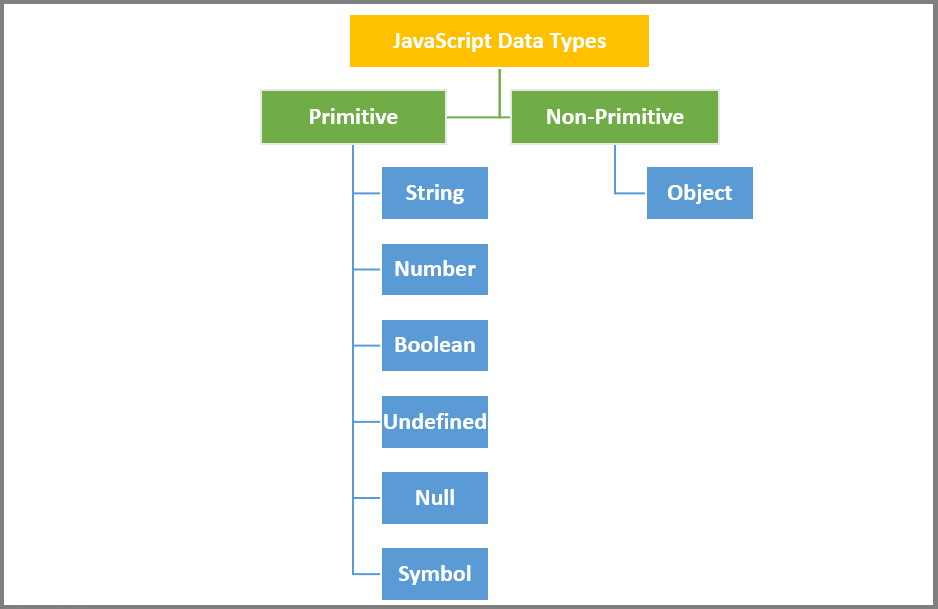
**Q1. Datatypes in JavaScript?**



**Q2. Let, Const, Var in JavaScript?**

var declarations are globally scoped or function scoped while let and const are block scoped.

|  |
| --- |
| function exampleFunction() {  if (true) {  var x = 10; // Function-scoped variable  }  console.log(x); // x is accessible here  }  exampleFunction();  function exampleFunction() {  if (true) {  let y = 20; // Block-scoped variable  const z = 30; // Block-scoped constant  }  console.log(y); // Error: y is not defined  console.log(z); // Error: z is not defined  }  exampleFunction(); |

var variables can be updated and re-declared within its scope; let variables can be updated but not re-declared; const variables can neither be updated nor re-declared.

|  |
| --- |
| function exampleVarScope() {  var x = 10;  console.log(x); // 10  var x = 20; // Re-declaring the variable  console.log(x); // 20  x = 30; // Updating the variable  console.log(x); // 30  }  function exampleLetScope() {  let y = 10;  console.log(y); // 10  // Attempting to re-declare will result in an error  // let y = 20; // Error: Identifier 'y' has already been declared  y = 30; // Updating the variable  console.log(y); // 30  }  function exampleConstScope() {  const z = 10;  console.log(z); // 10  // Attempting to re-declare will result in an error  // const z = 20; // Error: Identifier 'z' has already been declared  // Attempting to update will result in an error  // z = 30; // Error: Assignment to a constant variable  } |

They are all hoisted to the top of their scope. But while var variables are initialized with undefined, let and const variables are not initialized.

|  |
| --- |
| function hoistingExampleVar() {  console.log(x); // Outputs: undefined  var x = 10;  console.log(x); // Outputs: 10  }  function hoistingExampleLetConst() {  // This would result in a ReferenceError: Cannot access 'y' before initialization  // console.log(y);  let y = 20;  console.log(y); // Outputs: 20  // This would result in a ReferenceError: Cannot access 'z' before initialization  // console.log(z);  const z = 30;  console.log(z); // Outputs: 30  } |

While var and let can be declared without being initialized, const must be initialized during declaration.

|  |
| --- |
| // Valid: Declaring a variable with 'var' without initialization  var x;  console.log(x); // Outputs: undefined  // Valid: Declaring a variable with 'let' without initialization  let y;  console.log(y); // Outputs: undefined  // Invalid: Declaring a 'const' variable without initialization  const z; // Error: Missing initializer in const declaration |

**Q3. Arrow Function?**

An arrow function is a concise way to define functions in JavaScript. It was introduced in ECMAScript 6 (ES6) and provides a shorter syntax for writing functions, especially for small, simple functions. Arrow functions are often used for inline, one-liner functions and are particularly useful for preserving the value of this from the surrounding context. Here's the basic syntax of an arrow function:

|  |
| --- |
| const add = (a, b) => a + b;  console.log(add(5, 3)); // Outputs: 8 |

**Q4. Immediately-Invoked Function Expression**

IIFE is a function expression that is immediately executed after its definition. It's often used to create a private scope.

|  |
| --- |
| (function () {  // This code runs immediately  })(); |

**Q5. Loose equality(==) and strict equality(===).**

|  |
| --- |
| 5 == "5" // true, because "5" is converted to a number before comparison  false == 0 // true, because false is converted to 0  null == undefined // true, as they are loosely equal  5 === "5" // false, because the data types are different (number vs. string)  false === 0 // false, because the data types are different (boolean vs. number)  null === undefined // false, as they are not of the same type |

**Q6. Loops in JavaScript?**

for Loop

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

while loop

|  |
| --- |
| let i = 0;  while (i < 5) {  console.log(i);  i++;  } |

do while loop

|  |
| --- |
| let i = 0;  do {  console.log(i);  i++;  } while (i < 5); |

for in loop

|  |
| --- |
| const person = { name: "Alice", age: 30 };  for (const key in person) {  console.log(key, person[key]);  } |

for of loop

|  |
| --- |
| const colors = ["red", "green", "blue"];  for (const color of colors) {  console.log(color);  } |

forEach loop

|  |
| --- |
| const numbers = [1, 2, 3];  numbers.forEach((number) => {  console.log(number);  }); |

**Q7. Map function in javascript?**

|  |
| --- |
| const numbers = [1, 2, 3, 4, 5];  const doubledNumbers = numbers.map(function (number) {  return number \* 2;  });  console.log(doubledNumbers); // Outputs: [2, 4, 6, 8, 10] |

**Q8. Reduce in javascript?**

|  |
| --- |
| const numbers = [1, 2, 3, 4, 5];  const sum = numbers.reduce(function (accumulator, currentValue) {  return accumulator + currentValue;  }, 0);  console.log(sum); // Outputs: 15 |

**Q9. Filter in Javascript?**

|  |
| --- |
| const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9];  const evenNumbers = numbers.filter(function (number) {  return number % 2 === 0;  });  console.log(evenNumbers); // Outputs: [2, 4, 6, 8] |

Q10. Object In javascript?

|  |
| --- |
| const person = {  name: "John",  age: 30,  isStudent: false  };  **Accessing Object Properties:**  console.log(person.name); // John  console.log(person['age']); // 30  **Adding and Modifying Properties:**  person.city = "New York"; // Adding a new property  person.age = 31; // Modifying an existing property |