Task 1: Primitives and Complex Types

Primitive Data Types

- Stored by value (the variable directly holds the value).
- Immutable.
- Examples: string, number, boolean, undefined, null, symbol, bigint.

```javascript
let x = 10;
let y = x;
y = 20;
console.log(x); // 10
console.log(y); // 20
```

Complex Data Types

- Stored by reference (variable points to memory address).
- Mutable.
- Examples: object, array, function, date, map, set.

```javascript
let obj1 = { name: "Alice" };
let obj2 = obj1;
obj2.name = "Bob";
console.log(obj1.name); // "Bob"
```

Comparison

Feature	Primitive	Complex
Storage	By value	By reference
Mutability	Immutable	Mutable
Examples	number, string, boolean	object, array, function
Use cases	Simple values	Structured data

Task 2: Practical Applications

Convert string to number

```javascript
let str1 = "42";
let str2 = "42.56";
console.log(parseInt(str1)); // 42
console.log(parseFloat(str2)); // 42.56
```

Check if a variable is an object

```javascript
function isObject(variable) {
return typeof variable === "object" && variable !== null;
}
console.log(isObject({})); // true
console.log(isObject([])); // true
console.log(isObject("Hello")); // false
console.log(isObject(null)); // false
```

Access and modify array

```javascript
let fruits = ["apple", "banana", "cherry"];
console.log(fruits[1]); // "banana"
fruits[1] = "blueberry";
console.log(fruits); // ["apple", "blueberry", "cherry"]
```

Add and remove object properties

```javascript
let student = { id: 1, name: "Alice" };
student.age = 20;
console.log(student); // { id: 1, name: "Alice", age: 20 }
```

```
delete student.name;
console.log(student); // { id: 1, age: 20 }
```

Task 3: Deep and Shallow Copy

Original Dataset

```javascript
const studentRecords = [
{
id: 1,
name: "Alice",
courses: ["Math", "English"],
grades: { Math: 90, English: 85 },
},
{
id: 2,
name: "Bob",
courses: ["History", "Biology"],
grades: { History: 78, Biology: 92 },
},
];
```

Shallow Copy

```javascript
const shallowCopy = [...studentRecords];
shallowCopy[0].courses.push("Science");

console.log(studentRecords[0].courses);
// ["Math", "English", "Science"]
```

Change affects both arrays because nested objects are referenced.

Deep Copy

```javascript
const deepCopy = JSON.parse(JSON.stringify(studentRecords));
deepCopy[0].grades.Math = 100;

console.log(studentRecords[0].grades.Math); // 90
console.log(deepCopy[0].grades.Math); // 100
```

Use Case Scenario

If you update grades in a shallow copy, the original dataset may also change.
In student management systems, deep copy is safer.

Reflection

- Shallow copy: fast, less memory, but risky for nested data.
- Deep copy: safe for nested data, but slower and uses more memory.
- Shallow copy works for simple objects, deep copy is better for complex datasets.