Part A – (Theory Qs)

Ans1 – Output will be Undefined in case of var but in case of const and let error comes that cannot access b or c before initialization.

This is because, when you declare a variable with var, JavaScript moves the declaration to the top of the scope, and it is initialized as undefined at first. But in const and let they are not initialized immediately .

Ans2- Output – 53 (from int to string), 2 (string to int), 10 (from string to integer), 2(Boolean to integer), 1(null to Integer)

Ans3 – var no = Number("123”)

var string = String(456);

var bool = Boolean("true");

var string = [1, 2, 3].toString();

Ans4 - Stack: Stores primitive types (e.g., numbers, strings, booleans). Fixed size and fast.

Heap: Stores objects, arrays, and functions. Dynamic size.

Ans5 –

Part A - A is stored with value 10 in memory

B is also stored in memory with value 10 passed not as a reference.

When B change to 20 a not affected

So output will be = 10 20

Part B - Obj1 object is stored in memory

B is also stored in memory with obj1 passed as a reference.

When obj2 name change to Jane obj1 is also affected

So output will be = Jane Jane

Ans 6 –function abc(str) {

return {

length: str.length,

firstChar: str[0],

lastChar: str[str.length - 1],

```
    upper: str.toUpperCase()
  };
}
```

Ans 7 – Random - var random = Math.floor(Math.random() * (50 - 10 + 1)) + 10;

Round - var rounded = Math.round(4.7);

Max check –

```
let a=[3, 7, 2, 9, 1];
let max=a[0];
for(let i=1;i<a.length;i++){
   max=Math.max(a[i],max);
}
console.log(max);
```

Ans 8 -

```
let students = [
  { name: "Alice", age: 20, grade: 85 },
  { name: "Bob", age: 22, grade: 75 }
];
function addStudent(student) {
  students.push(student);
}
function findStudentByName(name) {
  return students.find(s => s.name === name);
}
function getTopStudents() {
  return students.filter(s => s.grade > 80);
}
addStudent({ name: "Charlie", age: 21, grade: 90 });
```

Ans 9 - Array destructuring lets you unpack values from an array into individual variables.

Object destructuring lets you extract properties into variables by matching keys.

Ans 10 -

function greet() {} - Function Declaration

const greet = function() {}; - Function Expression

const greet = () => {}; -> Arrow Function

We use declarations for hoisting, arrow for short callbacks, and expressions for flexibility.

Ans 11 –

```
function mapArray(arr, callback) {
  return arr.map(callback);
}
```

Ans 12 –

Hoisting is JavaScript's default behavior of moving declarations to the top of their scope before code execution. Example var

Output will be - undefined

[Function: getName]

undefined

Ans 14 –

```
function createCounter() {
  let count = 0;  // This is a private variable
  return {
    increment: function () {
      count++;
    },
    decrement: function () {
      count--;
    },
    getValue: function () {
      return count;
    } };  }
```

Ans 15 –

```
function ageCal(dob) {
  const dob= new Date(dob);
  const currentDate = new Date();
  let years = currentDate.getFullYear() - dob.getFullYear();
  let months = currentDate.getMonth() - dob.getMonth();
  let days = currentDate.getDate() - dob.getDate();
  if (days < 0) {
    months--;
    const dayPrev = new Date(currentDate.getFullYear(), currentDate.getMonth(),0)
.getDate();
    days += dayPrev;
  }
  if (months < 0) {
    years--;
    months += 12
  }
  return {
    years: years,
    months: months,
    days: days
  };
}
```

Ans 17 –

```
function User(jsonStr) {
  try {
    const user = JSON.parse(jsonStr);
    if (!user.name || !user.email || typeof user.age !== "number") {
      return { success: false, message: "Missing required fields." };
```

```
    }
    return { success: true, data: user };
  } catch {
    return { success: false, message: "Invalid JSON." };
  }
}
```

Ans 19 –

```
function processArray(arr, ...callbacks) {
  return callbacks.reduce((acc, fn) => fn(acc), arr);
}
const result = processArray([1, 2, 3],
  arr => arr.map(x => x * 2),
  arr => arr.filter(x => x > 3)
);
```

Ans 20 –

Global Context - (contains a = 1, outer)

outer() Context - (contains a = 2, inner)

inner() Context - (contains a = 3)

First outer call then inside outer inner call so first 3 comes as a output in inner then 2 print as a ouput in outer fn and last print 1 as a global variable a

Output - 3

2

1