SECTION 1 – Real-Time Function Logic

1. Payroll System:

Create calculateSalary(basicSalary, bonusPercentage)

- Calculate bonus
- Deduct 5% tax
- Return final salary
- Print full salary breakdown

```
function calculateSalary(basicSalary, bonusPercentage) {
    let bonus = (basicSalary * bonusPercentage) / 100;
    let grossSalary = basicSalary + bonus;
    let tax = grossSalary * 0.05;
    let finalSalary = grossSalary - tax;

    console.log("----- Salary Breakdown -----");
    console.log("Basic Salary      :", basicSalary);
    console.log("Bonus Percentage  :", bonusPercentage + "%");
    console.log("Bonus Amount      :", bonus);
    console.log("Gross Salary      :", grossSalary);
    console.log("Tax (5%)          :", tax);
    console.log("Final Salary      :", finalSalary);

    return finalSalary;
}
// Example usage:
calculateSalary(20000, 20);
```

2. Student Result System:

Create generateResult(name, marksArray)

- Calculate total
- Calculate average
- Decide Grade (A/B/C/Fail)

- Return result object

```
function generateResult(name, marksArray) {
    let total = 0;

  for (let mark of marksArray) {
        total += mark;
    }

   let average = total / marksArray.length;
    let grade;

    if (average >= 75) {
        grade = "A";
    } else if (average >= 60) {
        grade = "B";
    } else if (average >= 40) {
        grade = "C";
    } else {
        grade = "Fail";
    }

    return {
        studentName: name,
        totalMarks: total,
        averageMarks: average,
        grade: grade
    };
}

let result = generateResult("Ramya", [80, 70, 75, 85, 90]);
console.log(result);
```

SECTION 2-Scope & Hoisting(Debugging)

3. Debug This Code:

```
function demo(){
if(true){
var a = 10;
let b = 20;
}
console.log(a);
console.log(b);
}
```

Questions:

- What will happen?

- Why?
- Fix it properly.

Given Code:

```
function demo(){
   if(true){
      var a = 10;
      let b = 20;
   }
   console.log(a);
   console.log(b);
}
```

output:ReferenceError: b is not defined

Explanation:

- var is function-scoped, so a is accessible outside the if block.
- let is block-scoped, so b is not accessible outside the if block, resulting in a ReferenceError.

Why?

Var a is function -scopped

let b is block -scopped

```
function demo(){
   let a;
   let b;

   if(true){
      a = 10;
      b = 20;
   }

   console.log(a);
   console.log(b);
}
```

output:

10

20

4. Hoisting Analysis:

```
console.log(x);
var x = 100;
console.log(y);
let y = 200;
```

Predict output and explain.

Predicted output:

undefined

ReferenceError: Cannot access 'y' before initialization

Explanation: Var x is hoisted -declaration moves to the top,intialization stays in place.

SECTION 3– Callback & Higher Order (Industry Simulation)

5. Order Processing System:

Create processOrder(orderId, callback)

- Print "Order Processed"
- Call generateInvoice(orderId)

```
function generateInvoice(orderId) {
    console.log("Invoice generated for Order ID:", orderId);
}

function processOrder(orderId, callback) {
    console.log("Order Processed");
    callback(orderId);
}
```

Function call:

```
processOrder(101, generateInvoice);
```

output:

Order Processed

Invoice generated for Order ID: 101

6. Bank Transaction System:

Create transaction(amount, type, callback)

- If deposit → add
- If withdraw → subtract
- Call sendSMS()

```
let balance = 5000;

function sendSMS(message) {
    console.log("SMS Sent:", message);
}

function transaction(amount, type, callback) {
    if (type === "deposit") {
        balance += amount;
```

```
        console.log("Deposit Successful");
    } else if (type === "withdraw") {
        balance -= amount;
        console.log("Withdrawal Successful");
    } else {
        console.log("Invalid Transaction Type");
        return;
    }

    console.log("Current Balance:", balance);
    callback("Transaction of " + amount + " completed. Balance: " + balance);
}
```

SECTION 4 – Currying (E-Commerce)

7. Dynamic Price Builder:

Create priceBuilder(basePrice)(discount)(tax)

Return final price

Example: priceBuilder(2000)(15)(18)

```
const priceBuilder = basePrice => discount => tax => {
    let discountAmount = (basePrice * discount) / 100;
    let priceAfterDiscount = basePrice - discountAmount;
    let taxAmount = (priceAfterDiscount * tax) / 100;
    let finalPrice = priceAfterDiscount + taxAmount;

    return finalPrice;
};
```

Example usage:

```
let finalAmount = priceBuilder(2000)(15)(18);
console.log(finalAmount);
```

SECTION 5 – IIFE (Security + Encapsulation)

8. Secure Company Module:

- Store private variable companyCode

- Expose getCompanyStatus()

companyCode should not be directly accsible.

```
const companyModule = (function () {
   // private variable
   let companyCode = "COMP-2026";

   // public method
   function getCompanyStatus() {
      return "Company Active | Code: " + companyCode;
   }

   // expose only required methods
   return {
      getCompanyStatus: getCompanyStatus
   };
})();

console.log(companyModule.getCompanyStatus());
```

SECTION 6 – Generator (Advanced Logic)

9. Unique Order ID Generator:

Generate ORD1001, ORD1002, ORD1003, etc.

```
function* orderIdGenerator() {
   let id = 1001;

   while (true) {
      yield "ORD" + id;
      id++;
```

```
    }
}

const orderGen = orderIdGenerator();

console.log(orderGen.next().value);
console.log(orderGen.next().value);
console.log(orderGen.next().value);
```

10. Coupon Spin System:

Yield:

- 10% OFF
- 20% OFF
- 50% OFF
- No Luck
- Jackpot

Simulate multiple spins.

```
function* couponSpin() {
    const coupons = [
        "10% OFF",
        "20% OFF",
        "50% OFF",
        "No Luck",
        "Jackpot"
    ];

    let index = 0;

    while (true) {
        yield coupons[index];
        index = (index + 1) % coupons.length;
```

```
    }
}
```

simulating multiple spins:

```
const spin = couponSpin();

console.log(spin.next().value);
console.log(spin.next().value);
console.log(spin.next().value);
console.log(spin.next().value);
console.log(spin.next().value);
console.log(spin.next().value);
```

output:

10% OFF

20% OFF

50% OFF

No Luck

Jackpot

10% OFF

SECTION 7 – Mini Project (Integrated)

Mini E-Commerce Flow:

- addToCart(product, price)
- calculateTotal()
- applyDiscount() using currying
- generateCoupon() using generator
- processPayment() using callback
- Hide config using IIFE

/

```
let cart = [];
```

```
function addToCart(product, price) {
    cart.push({ product, price });
    console.log(product + " added to cart");
}

function calculateTotal() {
    let total = cart.reduce((sum, item) => sum + item.price, 0);
    console.log("Cart Total:", total);
    return total;
}

const applyDiscount = total => discount => {
    let discountAmount = (total * discount) / 100;
    return total - discountAmount;
};

function* generateCoupon() {
    const coupons = ["10% OFF", "20% OFF", "50% OFF", "NO LUCK", "JACKPOT"];
    let index = 0;

    while (true) {
        yield coupons[index];
        index = (index + 1) % coupons.length;
    }
}

function processPayment(amount, callback) {
    console.log("Processing payment of:", amount);
    callback("Payment Successful");
}
```

```
function paymentStatus(message) {
   console.log(message);
}
```

Concept Questions:

1. Difference between function declaration & expression?

Function Declaration-“Create the function first, then run code”

Function Expression → “Store the function in a variable at runtime”

2. What is higher order function?

Higher Order Function-Takes another function as an argument, or

Returns a function

3. Real-time example of generator?

Generators are used to produce multiple values one by one without stopping the program execution.

They are commonly used in applications like generating order IDs, coupon codes, and pagination systems.

4. Why do we use IIFE?

IIFE (Immediately Invoked Function Expression) is used to execute a function immediately and create a private scope.

Main Reasons:1.Avoid Global Variable Pollution

2.Data Privacy(Encapsulation)

Modern Pattern

3.Excute code immediately

5. Difference between var, let, const?

eature var let const

Feature	Var	Let	Const
Scope	Function	Block	Block
Redeclaration	Yes	No	No
Reassignment	Yes	Yes	No

```
</script>

<body>
```

</html>