### **Module 2: Control Structures**

#### Week 3

### **Learning Objectives**

By the end of this module, you will:

- Implement decision-making logic using conditional statements
- Use different types of loops effectively
- Control loop execution with break and continue
- Write nested control structures
- Build complex program flow logic

#### **Conditional Statements**

### The Need for Decision Making

Programs need to make decisions based on data:

```
javascript

let score = 85;

// Without conditionals - static behavior

console.log("Your grade is B");

// With conditionals - dynamic behavior

if (score >= 90) {

console.log("Your grade is A");
} else if (score >= 80) {

console.log("Your grade is B");
} else {

console.log("Your grade is C or below");
}
```

#### if Statement

### **Basic Syntax:**

```
if (condition) {
    // Execute this code if condition is true
}
```

### **Practical Example:**

```
javascript

let studentAge = 20;

let canEnrollInAdvancedCourse = false;

if (studentAge >= 18) {
    canEnrollInAdvancedCourse = true;
    console.log("Student is eligible for advanced courses");
}
```

### **Truthy and Falsy Values:**

```
javascript

// Falsy values: false, 0, "", null, undefined, NaN

if ("") {
    console.log("This won't run"); // Empty string is falsy
}

// Truthy values: Everything else

if ("Hello") {
    console.log("This will run"); // Non-empty string is truthy
}
```

### if-else Statement

## **Basic Syntax:**

```
javascript
```

```
if (condition) {
    // Execute if condition is true
} else {
    // Execute if condition is false
}
```

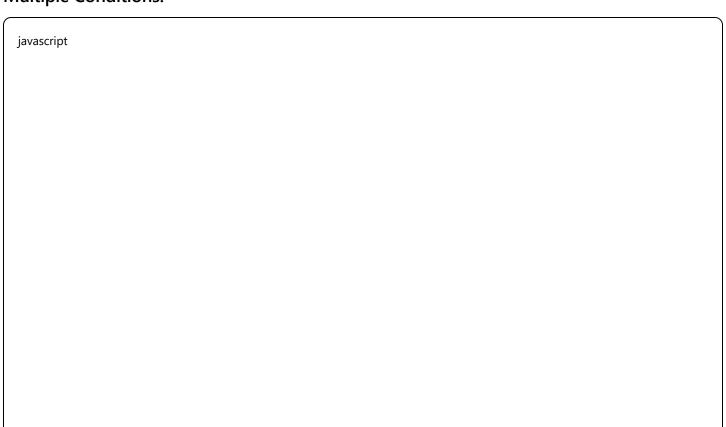
# **Example - Authentication Check:**

```
javascript

function checkAccess(user) {
    if (user.isLoggedIn) {
        console.log('Welcome back, ${user.name}!');
        showDashboard();
    } else {
        console.log("Please log in to continue");
        showLoginForm();
    }
}
```

# if-else if-else Chain

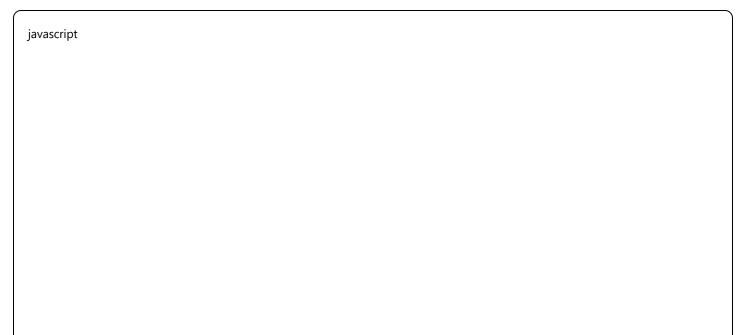
# **Multiple Conditions:**



```
function getLetterGrade(percentage) {
  let grade;
  if (percentage >= 90) {
     grade = 'A';
  } else if (percentage >= 80) {
     grade = 'B';
  } else if (percentage > = 70) {
     grade = 'C';
  } else if (percentage > = 60) {
     grade = 'D';
  } else {
     grade = 'F';
  return grade;
// Usage
console.log(getLetterGrade(95)); // 'A'
console.log(getLetterGrade(73)); // 'C'
console.log(getLetterGrade(45)); // 'F'
```

### **Nested Conditionals**

# **Complex Decision Trees:**



### **Refactored with Logical Operators:**

```
javascript

function canEnrollInCourseSimplified(student) {
   if (student.age >= 18 &&
       student.hasPrerequisites &&
       student.gpa >= 2.0 &&
       !student.hasHolds) {
       return "Enrollment approved";
   } else {
       return "Enrollment denied";
   }
}
```

#### switch Statement

### When to Use switch:

- Multiple discrete values to check
- Cleaner than long if-else chains

• Better performance for many conditions

## **Basic Syntax:**

```
javascript

switch (expression) {
    case value1:
        // code
        break;
    case value2:
        // code
        break;
    default:
        // code
}
```

# switch Examples

# Day Type Checker:

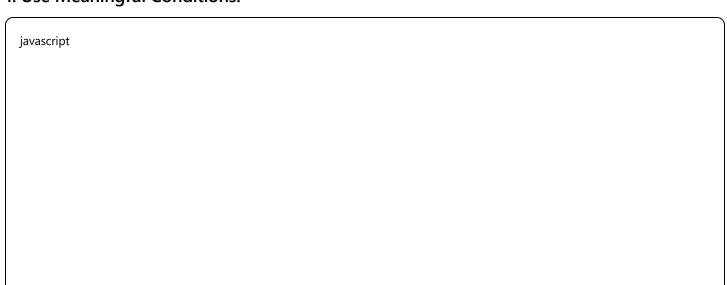
```
javascript
function getDayType(day) {
  switch (day.toLowerCase()) {
    case 'monday':
    case 'tuesday':
    case 'wednesday':
    case 'thursday':
    case 'friday':
       return 'Weekday';
    case 'saturday':
    case 'sunday':
       return 'Weekend';
    default:
       return 'Invalid day';
console.log(getDayType('Monday')); // 'Weekday'
console.log(getDayType('Saturday')); // 'Weekend'
```

### **Calculator Function:**

```
javascript
function calculator(a, b, operation) {
  let result;
  switch (operation) {
     case '+':
        result = a + b;
       break:
     case '-':
       result = a - b;
       break;
     case '*':
       result = a * b;
       break:
     case '/':
        result = b !== 0 ? a / b : 'Division by zero!';
        break;
     default:
       result = 'Invalid operation';
  return result;
```

## **Best Practices for Conditionals**

## 1. Use Meaningful Conditions:



```
// Bad
if (user.age >= 18 && user.hasJob && user.creditScore > 600) {
    // approve loan
}

// Good
const isAdult = user.age >= 18;
const hasIncome = user.hasJob;
const hasGoodCredit = user.creditScore > 600;

if (isAdult && hasIncome && hasGoodCredit) {
    // approve loan
}
```

### 2. Handle Edge Cases:

```
javascript

function gradeStudent(score) {
    if (typeof score !== 'number' || score > 0 || score > 100) {
        return 'Invalid score';
    }

    if (score >= 90) return 'A';
    if (score >= 80) return 'B';
    if (score >= 70) return 'C';
    if (score >= 60) return 'D';
    return 'F';
}
```

## **Loops Introduction**

## Why Use Loops?

Loops allow you to:

- Execute code repeatedly
- Process collections of data
- Avoid code duplication
- Implement algorithms efficiently

## Types of Loops in JavaScript:

- (for) loop when you know iteration count
- (while) loop when condition-based
- (do-while) loop execute at least once
- (for...in) loop iterate over object properties
- (for...of) loop iterate over iterable values

# for Loop

### **Basic Syntax:**

```
javascript

for (initialization; condition; increment) {

// code to execute
}
```

### **Example - Counting:**

```
javascript

// Count from 1 to 5

for (let i = 1; i <= 5; i++) {
    console.log('Count: ${i}');
}

// Output:
// Count: 1
// Count: 2
// Count: 3
// Count: 4
// Count: 5</pre>
```

### **Example - Array Processing:**

```
javascript
```

```
const students = ['Alice', 'Bob', 'Charlie', 'Diana'];

for (let i = 0; i < students.length; i++) {
    console.log(`${i + 1}. ${students[i]}`);
}

// Output:
// 1. Alice
// 2. Bob
// 3. Charlie
// 4. Diana</pre>
```

# while Loop

## **Basic Syntax:**

```
javascript

while (condition) {

// code to execute

// must update condition to avoid infinite loop
}
```

## **Example - Input Validation:**

javascript

```
let password = "";
let attempts = 0;
const maxAttempts = 3;

while (password !== "secret123" && attempts < maxAttempts) {
    password = prompt("Enter password:");
    attempts++;

    if (password !== "secret123") {
        console.log("Wrong password. ${maxAttempts - attempts} attempts left.");
    }
}

if (password === "secret123") {
    console.log("Access granted!");
} else {
    console.log("Access denied. Too many attempts.");
}</pre>
```

# do-while Loop

## **Basic Syntax:**

```
javascript

do {
// code to execute
} while (condition);
```

### **Key Difference:**

• Executes at least once, even if condition is false

## Example - Menu System:

javascript			

```
let choice;
do {
  console.log("\n=== Student Management System ===");
  console.log("1. Add Student");
  console.log("2. View Students");
  console.log("3. Exit");
  choice = prompt("Enter your choice (1-3):");
  switch (choice) {
    case '1':
       console.log("Adding student...");
       break:
     case '2':
       console.log("Viewing students...");
       break;
     case '3':
       console.log("Goodbye!");
       break:
     default:
       console.log("Invalid choice. Please try again.");
} while (choice !== '3');
```

## **Loop Control Statements**

#### break Statement:

```
javascript

// Find first even number

for (let i = 1; i <= 10; i++) {
    if (i % 2 === 0) {
        console.log(`First even number: ${i}`);
        break; // Exit the loop
    }
}

// Output: First even number: 2</pre>
```

#### continue Statement:

```
javascript

// Print only odd numbers from 1 to 10

for (let i = 1; i <= 10; i++) {
    if (i % 2 === 0) {
        continue; // Skip even numbers
    }

    console.log(`Odd number: ${i}`);
}

// Output: Odd number: 1, 3, 5, 7, 9</pre>
```

## **Nested Loops**

### **Example - Multiplication Table:**

#### Pattern Generation:

```
javascript
```

```
// Create a number pyramid
function createNumberPyramid(height) {
  for (let i = 1; i <= height; i++) {
    let spaces = " ".repeat(height - i);
    let numbers = "";
    // Ascending numbers
     for (let j = 1; j <= i; j++) {
       numbers += j;
    // Descending numbers
    for (let j = i - 1; j > = 1; j--) {
       numbers += j;
     console.log(spaces + numbers);
createNumberPyramid(4);
// Output:
// 1
// 121
// 12321
// 1234321
```

# for...in Loop

# **Iterating Over Object Properties:**

javascript

```
const student = {
  name: "Alice Johnson",
  age: 20,
  major: "Computer Science",
  gpa: 3.85,
  isEnrolled: true
};
console.log("Student Information:");
for (let property in student) {
  console.log(`${property}: ${student[property]}`);
// Output:
// name: Alice Johnson
// age: 20
// major: Computer Science
// gpa: 3.85
// isEnrolled: true
```

### With Arrays (not recommended):

```
javascript

const courses = ["JavaScript", "Python", "Java"];

// This works but gives indices, not values

for (let index in courses) {
    console.log(`${index}: ${courses[index]}`);
}

// Output: 0: JavaScript, 1: Python, 2: Java
```

# for...of Loop

## **Iterating Over Array Values:**

javascript			

```
const courses = ["JavaScript", "Python", "Java", "C++"];

for (let course of courses) {
    console.log(`Course: ${course}`);
}

// Output:
// Course: JavaScript
// Course: Python
// Course: Java
// Course: C++
```

## With Strings:

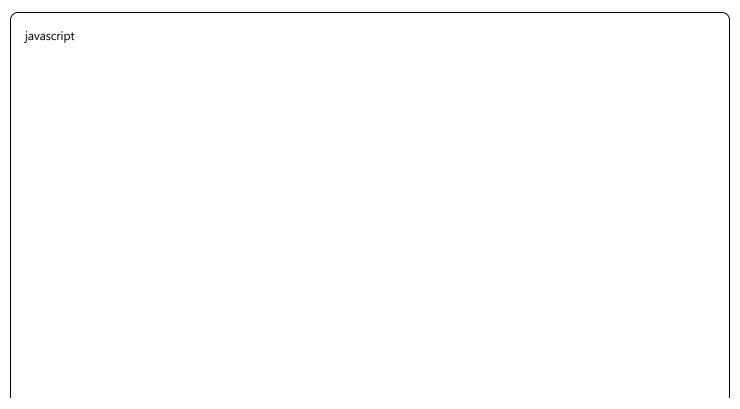
```
javascript

const message = "Hello";

for (let char of message) {
    console.log(char);
}

// Output: H, e, l, l, o
```

# **Practical Example: Prime Number Finder**



```
function findPrimes(max) {
  const primes = [];
  for (let num = 2; num <= max; num++) {
     let isPrime = true;
     // Check if num is divisible by any number from 2 to sqrt(num)
     for (let i = 2; i <= Math.sqrt(num); i++) {
       if (num % i === 0) {
          isPrime = false;
          break; // Exit inner loop early
     if (isPrime) {
       primes.push(num);
  return primes;
// Find all prime numbers up to 30
console.log("Prime numbers up to 30:");
console.log(findPrimes(30));
// Output: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

# **Complex Example: Grade Calculator**

javascript

```
function calculateGrades() {
  const students = [
    { name: "Alice", scores: [95, 87, 92, 88] },
     { name: "Bob", scores: [78, 85, 90, 82] },
     { name: "Charlie", scores: [92, 96, 88, 94] },
     { name: "Diana", scores: [85, 89, 91, 87] }
  ];
  console.log("=== Grade Report ===");
  for (let i = 0; i < students.length; <math>i++) {
     const student = students[i];
    let total = 0:
    // Calculate total score
     for (let j = 0; j < student.scores.length; <math>j++) {
       total += student.scores[j];
     const average = total / student.scores.length;
     let letterGrade;
     // Determine letter grade
     if (average \geq = 90) {
       letterGrade = 'A':
    } else if (average > = 80) {
       letterGrade = 'B';
    } else if (average > = 70) {
       letterGrade = 'C';
    } else if (average > = 60) {
       letterGrade = 'D';
    } else {
       letterGrade = 'F';
     console.log(`${student.name}: Average ${average.toFixed(1)} (${letterGrade})`);
     // Show individual scores
     let scoreDetails = " Scores: ";
     for (let score of student.scores) {
       scoreDetails += score + " ";
     console.log(scoreDetails.trim());
```

```
}
calculateGrades();
```

# **Loop Performance and Optimization**

# **Optimize Loop Conditions:**

```
javascript

// Inefficient - recalculates length each iteration
for (let i = 0; i < students.length; i++) {
    // process student
}

// Efficient - cache length
const studentCount = students.length;
for (let i = 0; i < studentCount; i++) {
    // process student
}

// Or use for...of for arrays
for (let student of students) {
    // process student
}</pre>
```

# **Avoid Infinite Loops:**

javascript		

```
// BAD - infinite loop
let i = 0;
while (i < 10) {
    console.log(i);
    // Forgot to increment i!
}

// GOOD - proper increment
let i = 0;
while (i < 10) {
    console.log(i);
    i++; // Always update loop variable
}</pre>
```

# **Common Loop Patterns**

#### 1. Accumulator Pattern:

```
javascript

function sumArray(numbers) {
    let sum = 0; // accumulator

    for (let number of numbers) {
        sum += number; // accumulate
    }

    return sum;
}
```

#### 2. Counter Pattern:

javascript			

```
function countPassingGrades(scores) {
    let passingCount = 0; // counter

for (let score of scores) {
    if (score >= 60) {
        passingCount++; // increment counter
    }
    }
    return passingCount;
}
```

#### 3. Search Pattern:

```
javascript

function findStudent(students, targetName) {
    for (let student of students) {
        if (student.name === targetName) {
            return student; // found
        }
    }
    return null; // not found
}
```

# **Error Handling in Loops**

## **Defensive Programming:**

	anning.			
, javascript				

```
function processGrades(studentData) {
   if (lArray.isArray(studentData)) {
      console.error("Invalid input: expected array");
      return;
   }

   for (let i = 0; i < studentData.length; i++) {
      const student = studentData[i];

      // Validate student object
      if (lstudent || lstudent.name || lArray.isArray(student.scores)) {
            console.warn("Skipping invalid student at index ${i}");
            continue;
      }

      // Process valid student
      const average = calculateAverage(student.scores);
      console.log("${student.name}: ${average}");
    }
}</pre>
```

## **Assignment 2: Multiplication Tables and Patterns**

## Requirements:

#### Part 1: Multiplication Table Generator

Create a program that:

- 1. Generates multiplication tables from 1 to N
- 2. Allows user to specify the range
- 3. Formats output in a neat table

#### Part 2: Number Pattern Generator

Create functions that generate:

- 1. Number triangles
- 2. Star patterns
- 3. Prime number sequences

4. Fibonacci sequences

#### Part 3: Grade Processing System

#### Build a system that:

- 1. Processes multiple students' grades
- 2. Calculates statistics (average, highest, lowest)
- 3. Generates grade distribution reports
- 4. Identifies students needing help

### **Example Code Structure:**

```
javascript

// Multiplication table generator

function generateMultiplicationTable(size) {

// Your implementation here
}

// Pattern generators

function createNumberTriangle(height) {

// Your implementation here
}

function findPrimesInRange(start, end) {

// Your implementation here
}

// Grade processing

function processStudentGrades(students) {

// Your implementation here
}
```

## **Best Practices Summary**

## 1. Choose the Right Loop:

- (for): When you know iteration count
- (while): When condition-based
- (for...of): For array values

• (for...in): For object properties

### 2. Avoid Deep Nesting:

## 3. Use Meaningful Variable Names:

```
javascript

// Bad

for (let i = 0; i < arr.length; i++) {
    if (arr[i] > x) {
        // what does this do?
    }
}

// Good

for (let studentIndex = 0; studentIndex < students.length; studentIndex++) {
    if (students[studentIndex].gpa > minimumGPA) {
        // clear purpose
    }
}
```

#### **Next Module Preview**

#### Module 3: Functions

- Function declaration and expressions
- Parameters, arguments, and return values
- Scope and closures
- Higher-order functions
- Arrow functions (ES6)

### Preparation:

- Master control flow concepts
- Practice nested structures
- Understand loop optimization
- Think about code organization

#### **Questions for Review**

- 1. When would you use a while loop instead of a for loop?
- 2. How can you avoid infinite loops?
- 3. What's the difference between (for...in) and (for...of)?
- 4. When should you use (break) vs (continue)?
- 5. How can nested loops be optimized?

#### **Practice Exercises:**

- Create a number guessing game
- Build a simple calculator with menu
- Generate various mathematical sequences
- Process and analyze data sets