Module 3: Functions

Weeks 4-5

Learning Objectives

By the end of this module, you will:

- Create reusable code blocks with functions
- Understand different ways to declare functions
- Master parameters, arguments, and return values
- Work with scope, closures, and the call stack
- Use higher-order functions and callbacks
- Apply functional programming concepts

Why Functions Matter

Problems Without Functions:

javascript

```javascript
// Calculating areas - repetitive code
let length1 = 10, width1 = 5;
let area1 = length1 * width1;
console.log("Area 1:", area1);

let length2 = 8, width2 = 3;
let area2 = length2 * width2;
console.log("Area 2:", area2);

let length3 = 12, width3 = 7;
let area3 = length3 * width3;
console.log("Area 3:", area3);
```

Solution With Functions:

javascript

```javascript
function calculateArea(length, width) {
    return length * width;
}

console.log("Area 1:", calculateArea(10, 5));
console.log("Area 2:", calculateArea(8, 3));
console.log("Area 3:", calculateArea(12, 7));
```

Function Declaration

Basic Syntax:

javascript

```javascript
function functionName(parameters) {
    // function body
    return value; // optional
}
```

Example - Student Grader:

javascript

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

// Function calls
console.log(calculateGrade(95)); // 'A'
console.log(calculateGrade(73)); // 'C'
console.log(calculateGrade(58)); // 'F'
```

Function Parameters and Arguments

Parameters vs Arguments:

javascript

```javascript
// Parameters are placeholders in function definition
function greetStudent(name, course) {  // name, course are parameters
    return `Welcome ${name} to ${course}!`;
}

// Arguments are actual values passed to function
let message = greetStudent("Alice", "JavaScript"); // "Alice", "JavaScript" are arguments
```

Multiple Parameters:

javascript

```javascript
function calculateFinalGrade(homework, midterm, final, participation) {
    const homeworkWeight = 0.3;
    const midtermWeight = 0.3;
    const finalWeight = 0.3;
    const participationWeight = 0.1;

    return (homework * homeworkWeight) +
           (midterm * midtermWeight) +
           (final * finalWeight) +
           (participation * participationWeight);
}

let finalGrade = calculateFinalGrade(85, 78, 92, 95);
console.log("Final Grade:", finalGrade.toFixed(1)); // 84.5
```

Default Parameters (ES6)

Setting Default Values:

javascript

```javascript
function greetStudent(name = "Student", course = "Programming") {
    return `Hello ${name}, welcome to ${course}!`;
}

console.log(greetStudent()); // "Hello Student, welcome to Programming!"
console.log(greetStudent("Alice")); // "Hello Alice, welcome to Programming!"
console.log(greetStudent("Bob", "JavaScript")); // "Hello Bob, welcome to JavaScript!"
```

Complex Default Values:

javascript

```javascript
function createStudent(name, gpa = 0.0, year = getCurrentYear()) {
    return {
        name: name,
        gpa: gpa,
        year: year,
        id: generateStudentId()
    };
}

function getCurrentYear() {
    return new Date().getFullYear();
}

function generateStudentId() {
    return Math.floor(Math.random() * 10000);
}
```

Rest Parameters

Handling Variable Arguments:

javascript

```javascript
function calculateAverage(...scores) {
    if (scores.length === 0) return 0;

    const sum = scores.reduce((total, score) => total + score, 0);
    return sum / scores.length;
}

// Can accept any number of arguments
console.log(calculateAverage(85, 92, 78)); // 85
console.log(calculateAverage(95, 87, 92, 88, 90)); // 90.4
console.log(calculateAverage()); // 0
```

Mixed Parameters:

javascript

```javascript
function gradeReport(studentName, ...testScores) {
    const average = calculateAverage(...testScores);
    const letterGrade = calculateGrade(average);

    return {
        student: studentName,
        scores: testScores,
        average: average.toFixed(1),
        grade: letterGrade
    };
}

let report = gradeReport("Alice", 95, 87, 92, 88);
console.log(report);
// { student: "Alice", scores: [95, 87, 92, 88], average: "90.5", grade: "A" }
```

Return Statements

Returning Values:

javascript

```
function add(a, b) {
  return a + b; // Returns the sum
}

function isEven(number) {
  return number % 2 === 0; // Returns boolean
}

function getStudentInfo(student) {
  return { // Returns object
    name: student.name,
    gpa: student.gpa,
    status: student.gpa >= 3.0 ? "Good Standing" : "Academic Warning"
  };
}
```

Early Returns:

javascript

```
function validateEmail(email) {
  // Early return for invalid input
  if (!email || typeof email !== 'string') {
    return false;
  }

  // Early return for empty string
  if (email.trim() === '') {
    return false;
  }

  // Main validation logic
  return email.includes('@') && email.includes('.');
}
```

Functions Without Return:

javascript

```javascript
function printGradeReport(student) {
    console.log(`Student: ${student.name}`);
    console.log(`GPA: ${student.gpa}`);
    console.log(`Status: ${student.gpa >= 3.0 ? "Good" : "Warning"}`);
    // No return statement - returns undefined
}

let result = printGradeReport({name: "Alice", gpa: 3.5});
console.log(result); // undefined
```

Function Expressions

Anonymous Functions:

javascript

```javascript
// Function expression
const calculateArea = function(length, width) {
    return length * width;
};

// Named function expression
const calculatePerimeter = function perimeter(length, width) {
    return 2 * (length + width);
};

console.log(calculateArea(5, 3)); // 15
console.log(calculatePerimeter(5, 3)); // 16
```

Differences from Declarations:

javascript

```
// Function declarations are hoisted
console.log(declared()); // Works! Outputs: "I'm declared"

function declared() {
    return "I'm declared";
}

// Function expressions are NOT hoisted
console.log(expressed()); // Error! Cannot access before initialization

const expressed = function() {
    return "I'm an expression";
};
```

Arrow Functions (ES6)

Basic Syntax:

javascript

```javascript
// Traditional function expression
const multiply = function(a, b) {
    return a * b;
};

// Arrow function equivalent
const multiplyArrow = (a, b) => {
    return a * b;
};

// Shortened arrow function (implicit return)
const multiplyShort = (a, b) => a * b;

// Single parameter (parentheses optional)
const square = x => x * x;

// No parameters (parentheses required)
const getCurrentYear = () => new Date().getFullYear();
```

Practical Examples:

```javascript
// Array processing with arrow functions
const students = [
  {name: "Alice", gpa: 3.8},
  {name: "Bob", gpa: 3.2},
  {name: "Charlie", gpa: 3.9}
];

// Find high achievers
const highAchievers = students.filter(student => student.gpa >= 3.5);

// Get student names
const names = students.map(student => student.name);

// Calculate average GPA
const averageGPA = students.reduce((sum, student) => sum + student.gpa, 0) / students.length;
```

Scope in JavaScript

Global Scope:

```javascript
// Global variable
let universityName = "Tech University";

function displayUniversity() {
  console.log(universityName); // Can access global variable
}

displayUniversity(); // "Tech University"
```

Function Scope:

```javascript
```

```
function calculateGrades() {
  // Function-scoped variables
  let totalScore = 0;
  let testCount = 0;

  function addTest(score) {
    totalScore += score; // Can access outer function variables
    testCount++;
  }

  addTest(85);
  addTest(92);

  return totalScore / testCount;
}

// console.log(totalScore); // Error! totalScore not accessible outside function
```

Block Scope (let and const):

```javascript
function processStudents() {
  const students = ["Alice", "Bob", "Charlie"];

  for (let i = 0; i < students.length; i++) {
    let studentName = students[i]; // Block-scoped
    console.log(`Processing ${studentName}`);
  }

  // console.log(i); // Error! i not accessible outside block
  // console.log(studentName); // Error! studentName not accessible outside block
}
```

Closures

What is a Closure?

A closure gives you access to an outer function's scope from an inner function.

Simple Closure Example:

```javascript
function createGreeter(greeting) {
    return function(name) {
        return `${greeting}, ${name}!`;
    };
}

const morningGreeter = createGreeter("Good morning");
const eveningGreeter = createGreeter("Good evening");

console.log(morningGreeter("Alice")); // "Good morning, Alice!"
console.log(eveningGreeter("Bob")); // "Good evening, Bob!"
```

Practical Closure - Counter:

```javascript
function createCounter() {
    let count = 0;

    return function() {
        count++;
        return count;
    };
}

const counter1 = createCounter();
const counter2 = createCounter();

console.log(counter1()); // 1
console.log(counter1()); // 2
console.log(counter2()); // 1 (independent counter)
console.log(counter1()); // 3
```

Real-World Example - Grade Tracker:

```javascript
```

```
function createGradeTracker(studentName) {
  let grades = [];

  return {
    addGrade: function(grade) {
      grades.push(grade);
    },

    getAverage: function() {
      if (grades.length === 0) return 0;
      const sum = grades.reduce((total, grade) => total + grade, 0);
      return sum / grades.length;
    },

    getGrades: function() {
      return [...grades]; // Return copy to prevent external modification
    },

    getStudentName: function() {
      return studentName;
    }
  };
}

const aliceTracker = createGradeTracker("Alice");
aliceTracker.addGrade(95);
aliceTracker.addGrade(87);
aliceTracker.addGrade(92);

console.log(`${aliceTracker.getStudentName()}'s average: ${aliceTracker.getAverage()}`);
// "Alice's average: 91.33333333333333"
```

Higher-Order Functions

Functions as Arguments:

javascript

```
function processStudents(students, processor) {
    const results = [];
    for (let student of students) {
        results.push(processor(student));
    }
    return results;
}

// Different processors
const getStudentName = student => student.name;
const getStudentGPA = student => student.gpa;
const getStudentStatus = student => ({
    name: student.name,
    status: student.gpa >= 3.0 ? "Good Standing" : "Academic Warning"
});

const students = [
    {name: "Alice", gpa: 3.8},
    {name: "Bob", gpa: 2.5},
    {name: "Charlie", gpa: 3.2}
];

console.log(processStudents(students, getStudentName));
// ["Alice", "Bob", "Charlie"]

console.log(processStudents(students, getStudentStatus));
// [{name: "Alice", status: "Good Standing"}, {name: "Bob", status: "Academic Warning"}, ...]
```

Functions Returning Functions:

javascript

```
function createValidator(validationType) {
  switch (validationType) {
    case 'email':
      return function(email) {
        return email.includes('@') && email.includes('.');
      };

    case 'gpa':
      return function(gpa) {
        return typeof gpa === 'number' && gpa >= 0 && gpa <= 4.0;
      };

    case 'age':
      return function(age) {
        return typeof age === 'number' && age >= 16 && age <= 100;
      };

    default:
      return function() { return true; };
  }
}

const emailValidator = createValidator('email');
const gpaValidator = createValidator('gpa');

console.log(emailValidator('alice@university.edu')); // true
console.log(emailValidator('invalid-email')); // false
console.log(gpaValidator(3.5)); // true
console.log(gpaValidator(5.0)); // false
```

Callbacks

Understanding Callbacks:

javascript

```
function fetchStudentData(studentId, callback) {
  // Simulate asynchronous operation
  setTimeout(() => {
    const student = {
      id: studentId,
      name: "Alice Johnson",
      gpa: 3.8,
      courses: ["JavaScript", "Data Structures"]
    };

    callback(student); // Call the callback function with data
  }, 1000);
}

// Using the callback
fetchStudentData(123, function(student) {
  console.log(`Loaded data for ${student.name}`);
  console.log(`GPA: ${student.gpa}`);
});
```

Error Handling with Callbacks:

javascript

```
function validateAndProcessGrade(grade, successCallback, errorCallback) {
    if (typeof grade !== 'number') {
        errorCallback(new Error('Grade must be a number'));
        return;
    }

    if (grade < 0 || grade > 100) {
        errorCallback(new Error('Grade must be between 0 and 100'));
        return;
    }

    // Process valid grade
    const letterGrade = grade >= 90 ? 'A' :
                       grade >= 80 ? 'B' :
                       grade >= 70 ? 'C' :
                       grade >= 60 ? 'D' : 'F';

    successCallback(letterGrade);
}

// Usage
validateAndProcessGrade(85,
    function(letterGrade) {
        console.log(`Grade: ${letterGrade}`);
    },
    function(error) {
        console.error(`Error: ${error.message}`);
    }
);
```

IIFE (Immediately Invoked Function Expressions)

Creating Private Scope:

javascript

```
// Basic IIFE
(function() {
    let privateVariable = "This is private";
    console.log("IIFE executed immediately");
})();

// IIFE with parameters
(function(name) {
    console.log(`Hello, ${name}!`);
})("Alice");
```

Module Pattern with IIFE:

javascript

```javascript
const StudentModule = (function() {
  // Private variables
  let students = [];
  let nextId = 1;

  // Private functions
  function generateId() {
    return nextId++;
  }

  function validateStudent(student) {
    return student.name && student.gpa !== undefined;
  }

  // Public API
  return {
    addStudent: function(name, gpa) {
      if (!name || gpa === undefined) {
        throw new Error("Name and GPA are required");
      }

      const student = {
        id: generateId(),
        name: name,
        gpa: gpa,
        enrollmentDate: new Date()
      };

      students.push(student);
      return student;
    },

    getStudent: function(id) {
      return students.find(student => student.id === id);
    },

    getStudentCount: function() {
      return students.length;
    },

    getTopStudents: function(count = 5) {
      return students
        .sort((a, b) => b.gpa - a.gpa)
```

```
            .slice(0, count);
        }
    };
})();

// Usage
StudentModule.addStudent("Alice", 3.8);
StudentModule.addStudent("Bob", 3.2);
console.log(StudentModule.getStudentCount()); // 2
console.log(StudentModule.getTopStudents(1)); // [{id: 1, name: "Alice", gpa: 3.8, ...}]
```

Function Composition

Combining Functions:

```javascript
// Individual functions
const addTax = price => price * 1.08;
const addShipping = price => price + 5.99;
const formatCurrency = amount => `$${amount.toFixed(2)}`;

// Function composition
function calculateTotal(basePrice) {
    return formatCurrency(addShipping(addTax(basePrice)));
}

// Or using a compose function
function compose(...functions) {
    return function(value) {
        return functions.reduceRight((acc, fn) => fn(acc), value);
    };
}

const calculateTotalComposed = compose(formatCurrency, addShipping, addTax);

console.log(calculateTotal(100)); // "$113.99"
console.log(calculateTotalComposed(100)); // "$113.99"
```

Recursion

Understanding Recursive Functions:

javascript

```javascript
// Factorial calculation
function factorial(n) {
  // Base case
  if (n <= 1) {
    return 1;
  }

  // Recursive case
  return n * factorial(n - 1);
}

console.log(factorial(5)); // 120 (5 * 4 * 3 * 2 * 1)
```

Fibonacci Sequence:

javascript

```javascript
function fibonacci(n) {
  if (n <= 1) {
    return n;
  }

  return fibonacci(n - 1) + fibonacci(n - 2);
}

// Generate first 10 Fibonacci numbers
for (let i = 0; i < 10; i++) {
  console.log(`F(${i}) = ${fibonacci(i)}`);
}
// F(0) = 0, F(1) = 1, F(2) = 1, F(3) = 2, F(4) = 3, F(5) = 5, ...
```

Tree Traversal Example:

javascript

```
// Nested course structure
const curriculum = {
  name: "Computer Science",
  courses: [
    {
      name: "Programming Fundamentals",
      modules: [
        { name: "Variables and Data Types",
```