

```
int Add(int x, int y)
{
```

# Week 6: Functions & AI-Assisted Code Generation

```
y = f(x);
```

C# Programming Fundamentals

```
return x
}
```

# Today's Agenda

- Functions: What and Why?
- Function Declaration & Invocation
- Parameters and Return Values
- Scope and Lifetime
- AI-Assisted Function Generation
- Hands-on Coding Session
- Student Tasks
- Debugging Common Issues

# Functions: What and Why?

## What are functions?

- Named blocks of code
- Reusable pieces of logic
- Building blocks of programs

## Why use functions?

- Code organization
- Reusability
- Maintainability
- Readability
- Testing isolation

```
// A simple function
void SayHello()
{
    Console.WriteLine("Hello, World!");
}
```

# Function Declaration

## Anatomy of a function:

```
// Return type, name, parameters
returnType FunctionName(parameters)
{
    // Function body
    // Code to execute

    return value; // If non-void
}
```

## Examples:

```
// No parameters, no return value
void DisplayMenu()
{
    Console.WriteLine("1. New Game");
    Console.WriteLine("2. Load Game");
    Console.WriteLine("3. Settings");
    Console.WriteLine("4. Exit");
}

// With return value
int AddNumbers(int a, int b)
{
    return a + b;
}
```

# Function Invocation

## Calling functions:

```
// Call function with no parameters
DisplayMenu();

// Call function with parameters
int sum = AddNumbers(5, 10);
Console.WriteLine($"Sum: {sum}");

// Call and use result directly
Console.WriteLine($"Result: {AddNumbers(7, 3)}");
```

## When functions are called:

- Execution jumps to the function
- Parameters are passed
- Function code executes
- Return value is sent back (if any)
- Execution continues after the call

# Parameters and Arguments

## Parameters:

- Defined in function declaration
- Act as placeholders

## Arguments:

- Actual values passed
- Matched to parameters

```
// Parameters: firstName, lastName
void Greet(string firstName, string lastName)
{
    Console.WriteLine($"Hello, {firstName} {lastName}!");
}

// Arguments: "John", "Doe"
Greet("John", "Doe");
```

# Parameter Types

## Value Parameters:

- Copy of the value is passed
- Changes inside function don't affect original

## Reference Parameters (ref):

- Reference to the variable is passed
- Changes affect the original variable

```
// Value parameter
void DoubleValue(int x)
{
    x = x * 2; // Only changes local copy
}

// Reference parameter
void DoubleRef(ref int x)
{
    x = x * 2; // Changes original variable
}

int num = 10;
DoubleValue(num); // num is still 10
DoubleRef(ref num); // num is now 20
```

# Output Parameters (out)

## Output Parameters:

- Used to return multiple values
- Must be assigned in the function
- Don't need to be initialized before passing

```
// Function with out parameter
bool TryParse(string input, out int result)
{
    result = 0; // Must assign value

    if (int.TryParse(input, out int parsed))
    {
        result = parsed;
        return true;
    }
    return false;
}

// Using out parameter
int number;
if (TryParse("123", out number))
{
    Console.WriteLine($"Parsed: {number}");
}
```



# Optional Parameters

## Optional Parameters:

- Have default values
- Can be omitted when calling

```
// Function with optional parameters
void DisplayMessage(string message,
                    ConsoleColor color = ConsoleColor.White
                    bool addTimestamp = false)
{
    if (addTimestamp)
    {
        message = $"[{DateTime.Now}] {message}";
    }

    Console.ForegroundColor = color;
    Console.WriteLine(message);
    Console.ResetColor();
}

// Various ways to call
DisplayMessage("Hello");
DisplayMessage("Warning", ConsoleColor.Yellow);
DisplayMessage("Error", ConsoleColor.Red, true);
```

# Named Arguments

## Named Arguments:

- Specify parameter by name
- Can be in any order
- Improve readability
- Useful with optional parameters

```
// Function with multiple parameters
void FormatText(string text, bool bold,
               bool italic, bool underline)
{
    // Implementation here
}

// Using named arguments for clarity
FormatText(
    text: "Hello",
    bold: true,
    italic: false,
    underline: true
);

// Can reorder named arguments
FormatText(
    underline: true,
    text: "Hello",
    italic: false,
    bold: true
);
```

# Return Values

## Return Values:

- Specified by return type
- Function exits when return is reached
- void functions don't return a value

```
// Return a single value
double CalculateArea(double radius)
{
    return Math.PI * radius * radius;
}

// Early return for validation
bool IsValidAge(int age)
{
    if (age < 0 || age > 120)
    {
        return false;
    }

    // More validation logic could go here

    return true;
}
```

# Returning Multiple Values

## Methods to return multiple values:

1. Using out parameters (as seen before)
2. Using tuples
3. Using custom types/classes

```
// Return multiple values with tuple
(int min, int max, double average) AnalyzeNumbers(int[] nu
{
    int min = numbers.Min();
    int max = numbers.Max();
    double avg = numbers.Average();

    return (min, max, avg);
}

// Using the returned tuple
var stats = AnalyzeNumbers(new[] { 4, 7, 2, 9, 5 });
Console.WriteLine($"Min: {stats.min}");
Console.WriteLine($"Max: {stats.max}");
Console.WriteLine($"Average: {stats.average}");
```

# Function Scope

## Variable Scope:

- Local variables only exist inside the function
- Can't access local variables from outside
- Parameters are local to the function

```
void ExampleFunction()
{
    int localVar = 10; // Local variable
    Console.WriteLine(localVar); // Works fine
}

// Console.WriteLine(localVar); // ERROR!
// localVar doesn't exist here

int globalVar = 20;

void AnotherFunction()
{
    // Can access global variable
    Console.WriteLine(globalVar);




    // Local variable with same name hides global
    int globalVar = 30;




    // Refers to local version
    Console.WriteLine(globalVar);
}
```

# AI-Assisted Code Generation

## AI tools can help with:

- Generating function templates
- Converting pseudocode to real code
- Suggesting parameter types
- Documentation generation
- Code optimization
- Debugging assistance

**Benefits & Limitations:**  Saves time on boilerplate  
 Offers different approaches  Learns from your patterns

 May not understand full context  Could suggest inefficient code  Might not follow best practices

**Remember: AI is a tool, not a replacement for understanding!**

# AI Exploration: Function Generation

Let's ask an AI to generate a function to:

- Calculate the grade (A, B, C, D, F) based on a student's score
- Validate input is between 0-100
- Return both the grade and a boolean indicating if they passed

AI Prompt:

```
"Write a C# function that takes a student's numeric score (0-100),  
validates the input, and returns both a letter grade (A, B, C, D, F)  
and a boolean indicating if they passed (60 or above)."
```

# AI-Generated Function

```
/// <summary>
/// Calculates a letter grade and pass status based on a numeric score.
/// </summary>
/// <param name="score">Numeric score between 0 and 100</param>
/// <returns>A tuple containing the letter grade and whether the student passed</returns>
public (string Grade, bool Passed) CalculateGrade(int score)
{
    // Validate input
    if (score < 0 || score > 100)
    {
        throw new ArgumentOutOfRangeException(nameof(score),
            "Score must be between 0 and 100");
    }

    // Determine letter grade
    string grade;
    if (score ≥ 90) grade = "A";
    else if (score ≥ 80) grade = "B";
    else if (score ≥ 70) grade = "C";
    else if (score ≥ 60) grade = "D";
    else grade = "F";

    // Determine if passing (60 or above)
    bool passed = score ≥ 60;
```



# Analyzing AI-Generated Code

## Strengths:

- Good input validation
- Clear documentation with XML comments
- Uses tuples for multiple return values
- Logical grade calculation
- Good variable naming

## Potential Improvements:

- Add constants for grade thresholds
- Consider using switch expression (C# 8+)
- Add unit tests for boundary conditions
- Consider culture-specific grading systems
- Maybe add more detailed feedback?

**Key Takeaway:** AI gives you a starting point, but you need to understand and improve it.

# Guided Coding Session

Let's build a more comprehensive grading system:

1. Calculate weighted grades from multiple assignments
2. Generate student performance reports
3. Save and load results from files

## First step: Create a function to calculate weighted average

```
/// <summary>
/// Calculates weighted average from scores and their weights
/// </summary>
public static double CalculateWeightedAverage(
    double[] scores, double[] weights)
{
    // Input validation
    if (scores == null || weights == null)
        throw new ArgumentNullException();

    if (scores.Length != weights.Length)
        throw new ArgumentException("Arrays must be same length");

    if (scores.Length == 0)
```

# Student Coding Task

Now it's your turn! Using what we've learned, complete the following tasks:

1. Write a function called `GenerateGradeReport` that:
  - Takes a student name, ID and an array of scores
  - Calculates the average score
  - Determines the letter grade
  - Returns a formatted string with all this information
2. Write a function called `SaveGradeReport` that:
  - Takes a report string and a filename
  - Saves the report to a text file
  - Returns a boolean indicating success or failure

Use AI to help you draft the functions, then analyze and improve the generated code.

# Debugging & Discussion

## Common Function Issues:

### 1. Scope Problems:

- Trying to access variables outside their scope
- Shadowing variables with same name

### 2. Parameter Errors:

- Incorrect parameter types
- Wrong number of arguments
- Incorrect order of arguments

### 3. Return Value Issues:

- Forgetting to return a value
- Returning the wrong type

## Function Debug Example:

```
// Bug: Not all code paths return a value
string GetLetterGrade(int score)
{
    if (score ≥ 90)
        return "A";
    else if (score ≥ 80)
        return "B";
    else if (score ≥ 70)
        return "C";
    else if (score ≥ 60)
        return "D";
    // Missing else branch!
}
```

```
// Fix: Ensure all paths return a value
string GetLetterGrade(int score)
{
    if (score ≥ 90)
        return "A";
    else if (score ≥ 80)
        return "B";
```

# Wrap-Up & Next Steps

## Today we learned:

- Function declaration and invocation
- Parameters and return values
- Optional parameters, named arguments
- Function scope
- Using AI to generate function code

## Practice on your own:

- Create functions for common tasks
- Convert repetitive code into functions
- Use AI to help draft functions, then refine

## Coming Next Week: Arrays & Lists

We'll explore:

- Working with collections of data
- Array initialization and access
- List operations and methods
- Common algorithms for data processing
- Choosing between arrays and lists

**Be ready to build on your function knowledge to process collections of data!**

# Thank You!

Questions?