Essential Programming Concepts

16/03/2025

Table of Contents

Syntax Error

A mistake in the code's structure, like missing semicolons or incorrect keywords.

Example

```
Console.WriteLine("Hello World") // ❌ Missing semicolon
```

Logical Error

A mistake in the program’s logic that **produces incorrect results without crashing**.

```
int a = 5, b = 10;
int sum = a * b; // ❌ Should be sum = a + b
```

Bug

An error in the program causing unexpected behavior or crashes.

Example

Incorrect conditions in loops leading to infinite execution.

```
for(i=1; i<3; i--)
{
    Console.WriteLine(“I am here”);
}
```

Debug

The process of finding and fixing errors (bugs) in your code.

Example

```
for(i=1; i<3; i--)
{
    Console.WriteLine("I am here");
}
```

Debugging

A **systematic approach** to troubleshooting issues in your code.

Example

```
for(i=1; i<3; i--)
{
    Console.WriteLine("I am here");
}
```

Breakpoint

A marker in your code that tells the **debugger to pause execution** at a specific line.

Use Case: Helps inspect variable values and identify logic errors.

Dry Running

Manually stepping through code **without running it** to predict outputs.

```
for (int i = 1; i <= 3; i++)
{
    Console.WriteLine(i);
}
```

Dry Run Process:

1. `i = 1` → Print `1`
2. `i = 2` → Print `2`
3. `i = 3` → Print `3`
4. `i = 4` → Exit loop

Unit Testing

Writing small, automated tests to check if **functions return expected results**.

```
public int Add(int a, int b)
{
    var result = a + b;
    return result;
}

Assert.AreEqual(5, Add(2, 3)); // ✅ Test passes
```

Edge Case

An unusual or extreme input value that might cause **unexpected behavior**.

Example

Handling **division by zero** in calculations.

Edge Case

Defensive Coding

Writing code that **anticipates errors** and handles them gracefully.

Example

Checking for **null values** before accessing them.

Exception Handling

Using `try-catch` blocks to handle errors without **crashing the program**.

Example

```
try
{
   int result = 10 / 0;
}
catch (DivideByZeroException)
{
   Console.WriteLine("Cannot divide by zero!");
}
```

Git

Git is a **version control system** that helps developers track changes in their code, collaborate with others, and revert to previous versions if needed.

Key Features

- Tracks changes in your project.
- Allows multiple developers to work on the same code without conflicts.
- Enables branching, merging, and rollback to previous versions.

Basic Git Commands

```
git init         # Initialize a new repository
git clone URL    # Clone an existing repository
git status       # Check file changes
git add .        # Stage all changes
git commit -m "Message"  # Commit changes
git push origin main  # Push changes to remote repository
```

GitHub

GitHub is a cloud-based platform for hosting Git repositories, making it easy to collaborate, share, and manage projects.

Why Use GitHub?
Stores your code online for backup and collaboration.
Allows multiple developers to work on a project.
Supports pull requests, issue tracking, and project management.

Basic GitHub Workflow

1. Create a repository on GitHub.
2. Clone it using git clone.
3. Make changes, commit, and push to GitHub using:

```
git add .
git commit -m "Updated project"
git push origin main
```

4. Collaborate using pull requests and branches.

Question & Answer

The End.