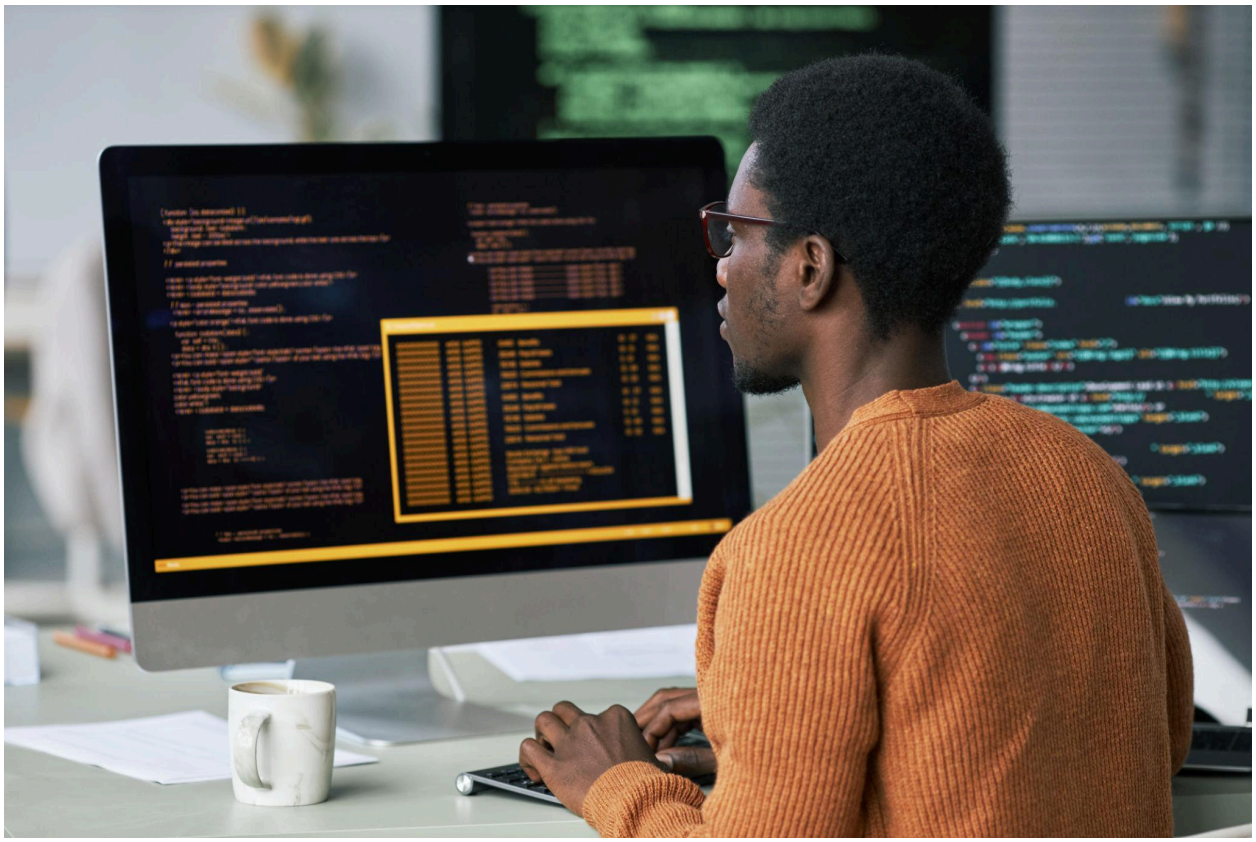
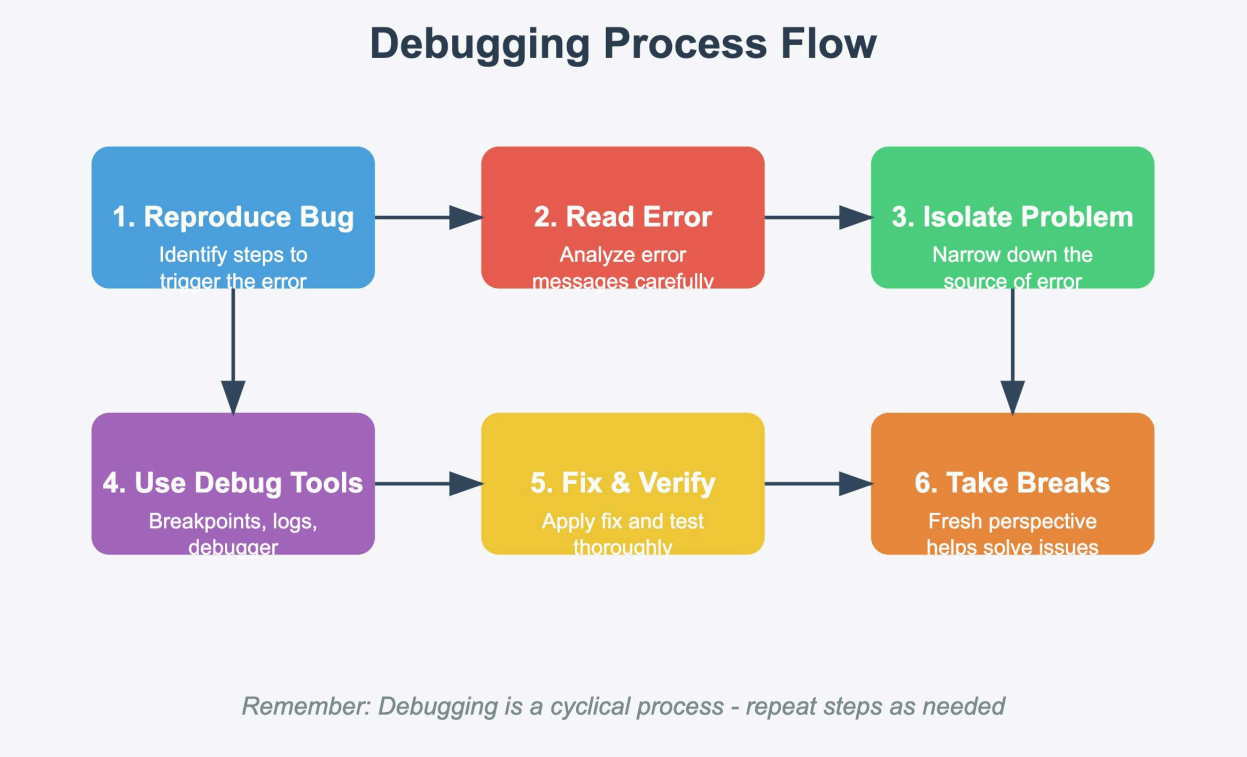
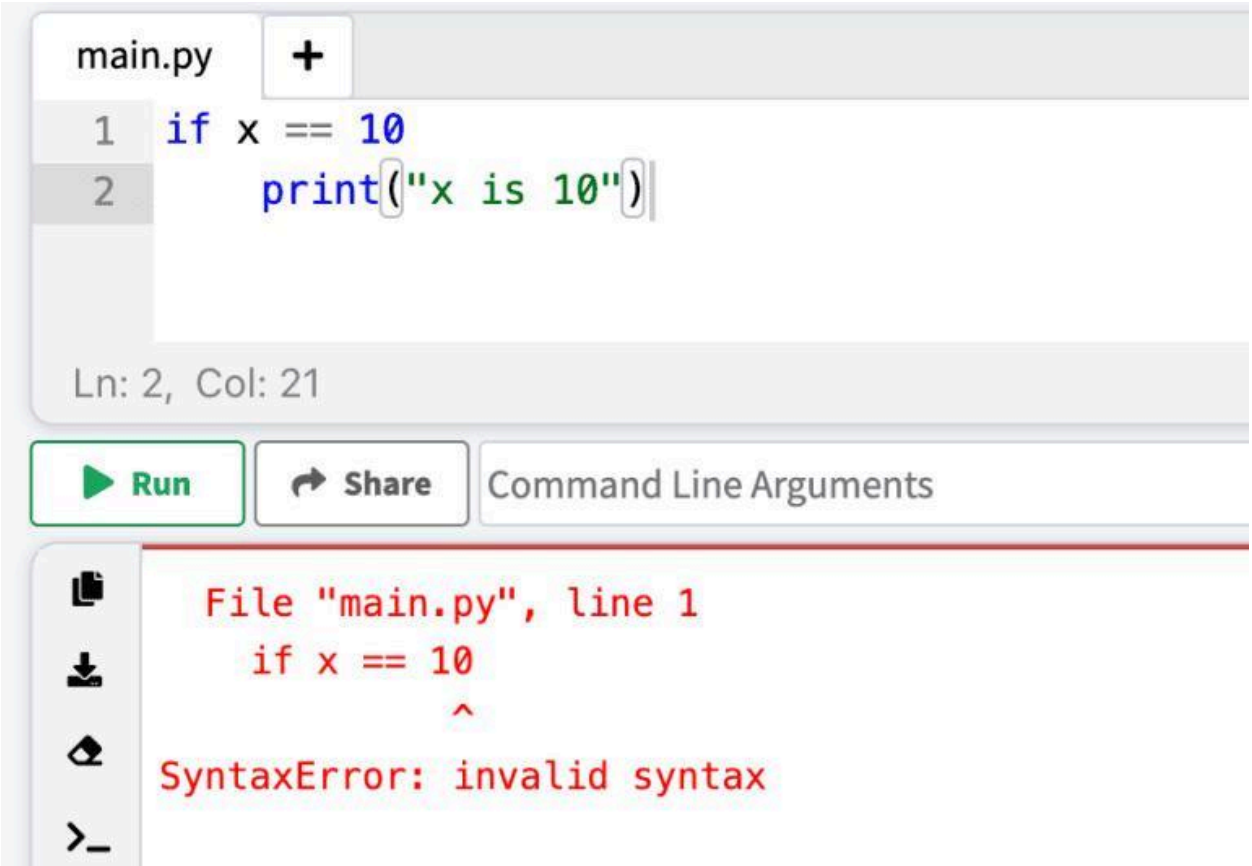
Tutorial: How to Debug a Computer Program

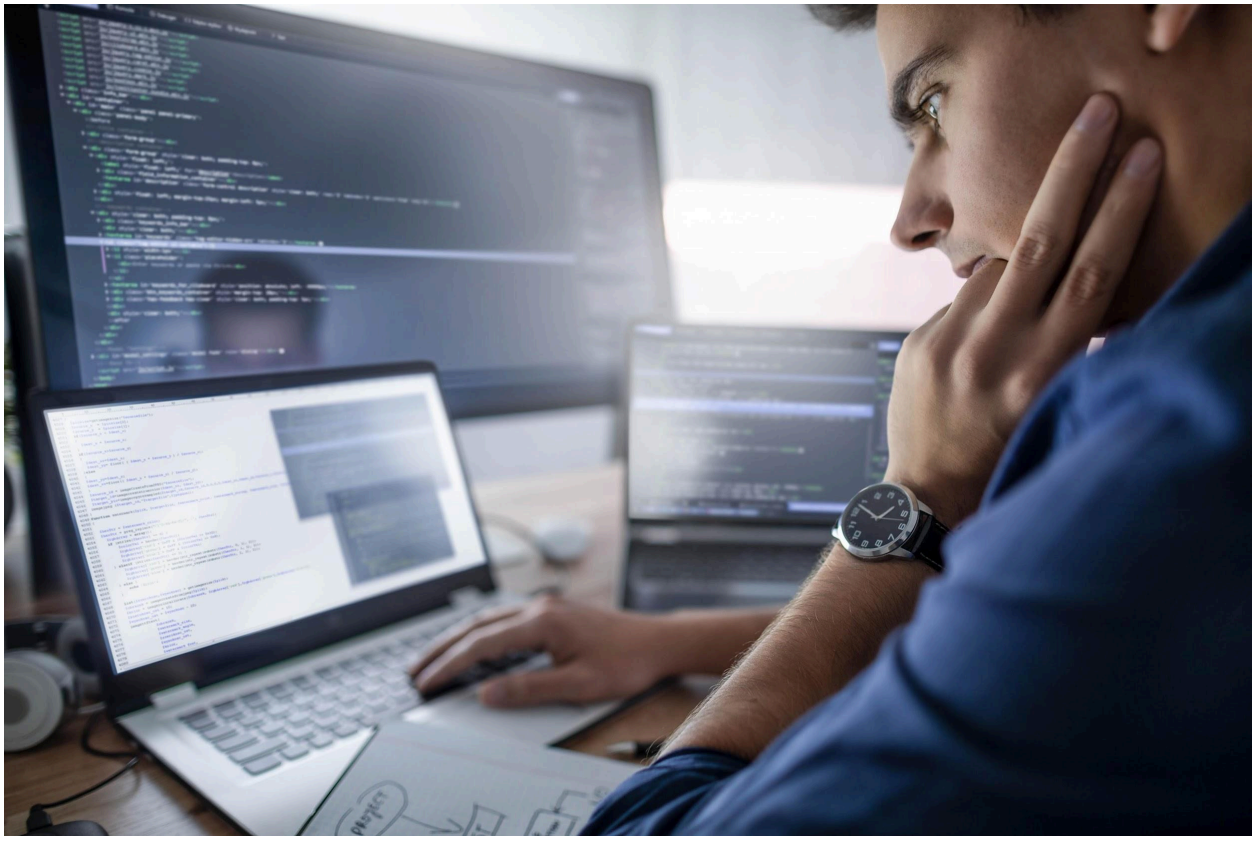
Understanding Syntax Errors and Logical Errors

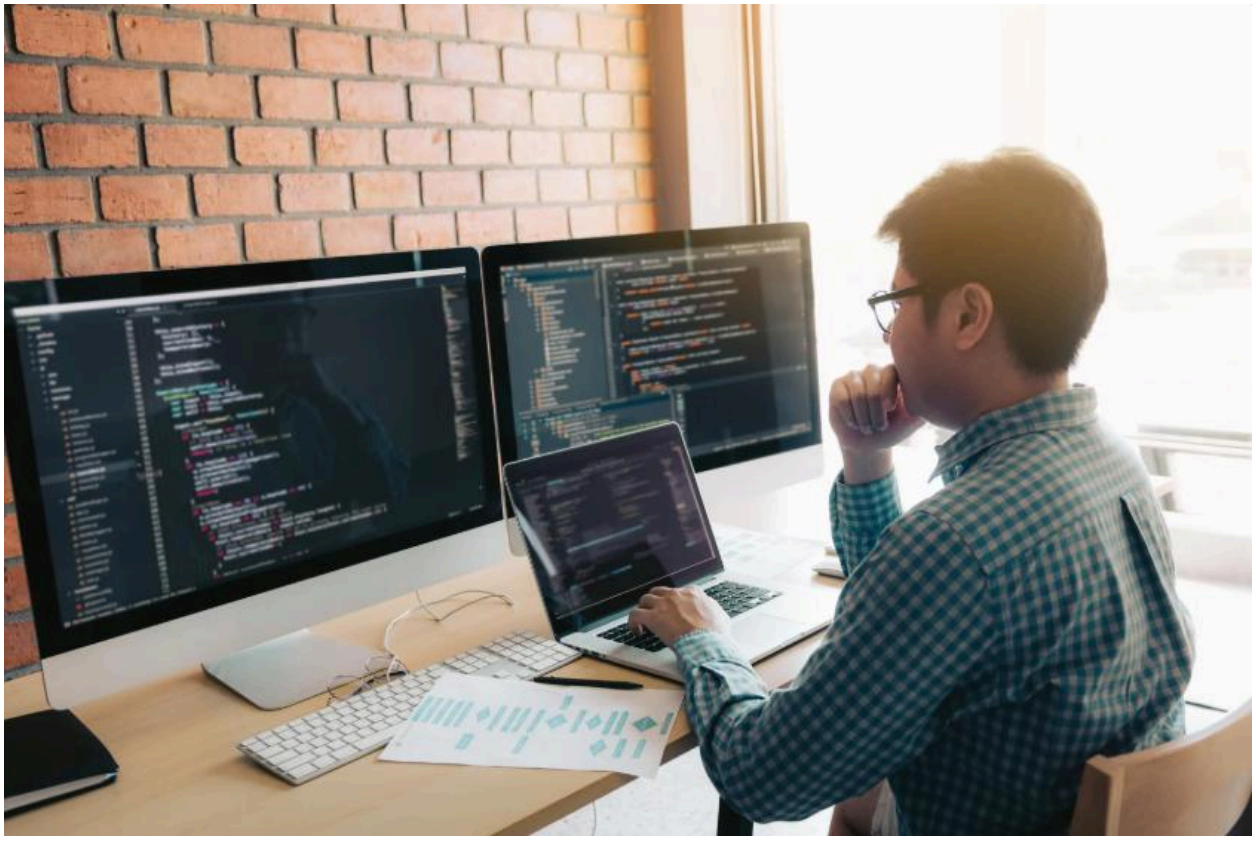




Remember: Debugging is a cyclical process - repeat steps as needed







Learning Objectives

By the end of this tutorial, you will be able to:

- Explain what debugging is.
- Identify syntax errors.
- Identify logical errors.
- Follow a systematic debugging process.
- Use debugging tools effectively.
- Improve the quality of your programs.

What is Debugging?

Debugging is the process of finding, identifying, and fixing errors (called **bugs**) in a computer program.

Programmers rarely write perfect code on the first try. Debugging is a normal part of software development and helps ensure that a program works as intended.

Types of Programming Errors

There are three common types of errors:

1. Syntax Errors
2. Runtime Errors
3. Logical Errors

What is a Syntax Error?

A **syntax error** happens when the code does not follow the rules (grammar) of the programming language.

The compiler or interpreter usually catches syntax errors before the program runs.

Example (Missing Semicolon in Java)

System.out.println("Hello World")

Correct:

System.out.println("Hello World");

Other Syntax Errors

- Missing parentheses
- Missing quotation marks
- Missing brackets { }
- Misspelled keywords
- Missing commas
- Incorrect punctuation

What is a Logical Error?

A **logical error** occurs when the program runs without crashing but produces the wrong result because the algorithm or logic is incorrect.

Example

Suppose you want to add two numbers:

int total = num1 - num2;

The program compiles successfully, but subtraction is used instead of addition.

Correct version:

int total = num1 + num2;

The code runs, but the original program gives the wrong answer because of a logical error.

What is a Runtime Error?

A **runtime error** occurs while the program is running.

Examples include:

- Dividing by zero
- Accessing an array index that doesn't exist
- Opening a file that cannot be found
- Running out of memory

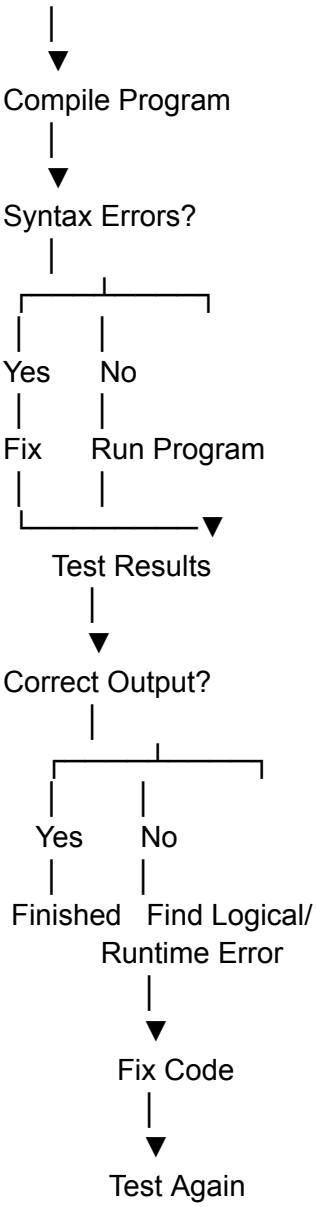
Example:

int answer = 10 / 0;

This causes a runtime error because division by zero is not allowed.

Debugging Process

Write Code



Step 1: Read the Error Message

Most programming environments display helpful error messages.

Example:

Error:
';' expected

The message often tells you:

- File name
- Line number
- Type of error

Always start here before making changes.

Step 2: Find the Error

Go to the line identified by the compiler.

Check for:

- Missing punctuation
- Incorrect spelling
- Missing braces
- Incorrect variable names
- Missing parentheses

Step 3: Fix One Error at a Time

Don't try to fix everything at once.

Instead:

- Fix the first error.
- Recompile.
- Check for remaining errors.
- Repeat as needed.

Many later errors disappear after fixing the first one.

Step 4: Run the Program

Once the program compiles:

- Test every feature.
- Enter different inputs.
- Try unusual cases.
- Verify the output.

Step 5: Compare Expected vs. Actual Results

Ask yourself:

Expected:

2 + 3 = 5

Actual:

2 + 3 = -1

If the program runs but the answer is wrong, you likely have a logical error.

Step 6: Use Print Statements

Print statements help you understand what your program is doing.

Example:

System.out.println("Value = " + total);

Printing variable values can show where incorrect data appears.

Step 7: Use a Debugger

Most IDEs, such as Android Studio, IntelliJ IDEA, Visual Studio, and Eclipse, include a debugger.

Common debugger features:

- Breakpoints
- Step Into
- Step Over
- Step Out
- Watch Variables
- Call Stack
- Variable Inspection

These tools let you pause execution and inspect the program's state.

Step 8: Test Again

After making changes:

- Recompile.
- Run the program.
- Test multiple scenarios.

Good testing includes:

- Typical input
- Invalid input
- Large values
- Small values
- Empty input (where applicable)

Common Debugging Tips

✔ Read error messages carefully.

✔ Don't ignore compiler warnings.

✔ Keep your code organized and well-indented.

✔ Use meaningful variable names.

✔ Test frequently instead of waiting until the end.

✔ Save working versions of your project.

✔ Ask someone else to review your code if you're stuck.

Example Debugging Session

Original Code

```
int total = number1 - number2;
System.out.println(total);
```

Input:

5
3

Output:

2

Expected output:

8

Problem

The code subtracts instead of adds.

Fixed Code

```
int total = number1 + number2;
System.out.println(total);
```

Now the output is correct.

Summary Table

Error Type	What It Means	Example
Syntax Error	The code breaks the language's grammar rules.	Missing semicolon or brace
Runtime Error	The program crashes or fails while running.	Division by zero
Logical Error	The program runs but gives an incorrect result.	Using subtraction instead of addition

---

Best Practices

- Write small sections of code and test often.
- Read compiler messages carefully.
- Use debugging tools built into your IDE.
- Test with a variety of inputs.
- Keep your code simple and well organized.
- Learn from each bug you fix.

Knowledge Check

1. What is debugging?
2. What is a syntax error?
3. What is a logical error?
4. What is a runtime error?
5. Why should you fix the first compiler error before the others?
6. What are breakpoints used for?
7. Why are print statements helpful?
8. What is the difference between expected output and actual output?
9. Name two common runtime errors.

10. Why is testing with different inputs important?

Conclusion

Debugging is an essential skill for every programmer. By understanding the differences between **syntax errors**, **runtime errors**, and **logical errors**, you can solve problems more efficiently and write more reliable software. Following a step-by-step debugging process—reading error messages, fixing one issue at a time, testing thoroughly, and using debugging tools—will help you become a more effective software developer.