Programming Logic and Design

Chapter 1

An Overview of Computers and Programming

Objectives

In this chapter, you will learn about:

- Computer systems
- Simple program logic
- The steps involved in the program development cycle
- Pseudocode statements and flowchart symbols
- Programming and user environments
- The evolution of programming models

Understanding Computer Systems

- **Computer system**
  - Combination of all the components required to process and store data using a computer
- **Hardware**
  - Equipment associated with a computer
- **Software**
  - Computer instructions
  - Tells the hardware what to do
  - **Programs**
    - Instructions written by programmers

Understanding Computer Systems

- **Software**
  - Computer instructions
  - Tells the hardware what to do
  - **Programs**
    - Instructions written by programmers
  - **Application software** such as word processing, spreadsheets, payroll and inventory, even games
  - **System software** such as operating systems like Windows, Linux, or UNIX

Understanding Computer Systems (continued)

- Computer hardware and software accomplish three major operations
  - **Input**
    - **Data items** such as text, numbers, images, and sound
  - **Processing**
    - Calculations and comparisons performed by the **central processing unit** (**CPU**)
  - **Output**
    - Resulting **information** that is sent to a printer, a monitor, or **storage devices** after processing
    - A **Cloud** based device is accessed through the Internet

Understanding Computer Systems (continued)

- **Programming language**
  - Used to write computer instructions called **program code**
  - Writing instructions is called **coding the program**
  - Considered high-level language or source code
  - Examples
    - Visual Basic, C#, C++, or Java
- **Syntax**
  - Rules governing word usage and punctuation in a language
  - Mistakes in a language's usage are **syntax errors**
  - Examples
    - Mistyping a word

Understanding Computer Systems (continued)

- **Computer memory**
  - Computer's temporary, internal storage – **random access memory** (**RAM**)
  - **Volatile** memory – lost when the power is off
- Permanent storage devices
  - **Nonvolatile** memory
- **Compiler** or **interpreter**
  - Translates source code into **machine language** (**binary language**) statements called **object code** that computers can understand and run
  - Checks for syntax errors
- Program **executes** or **runs**
  - Input will be accepted, some processing will occur, and results will be output

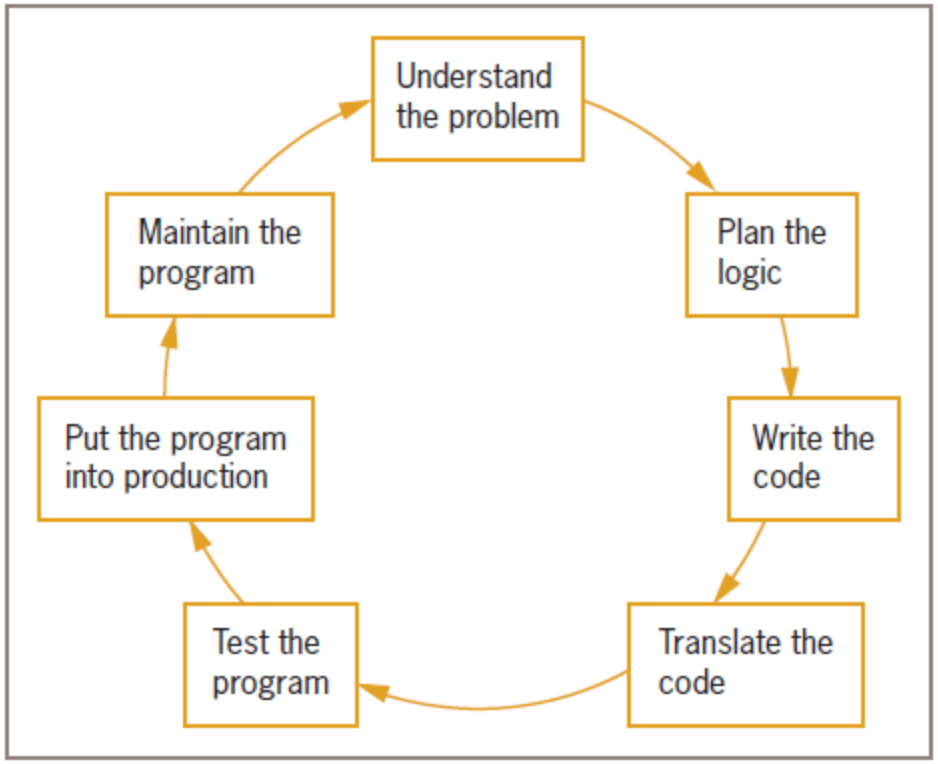
Understanding Simple Program Logic

- Programs with syntax errors cannot execute
- There are other errors as well
  - **Logical errors**
    - Errors in program logic that produce incorrect output
- **Logic** of the computer program
  - Sequence of specific instructions in specific order
- **Variable**
  - Named memory location that stores a given value that can vary

Understanding the Program Development Cycle

- **Program development cycle**
  - Understand the problem
  - Plan the logic
  - Code the program
  - Use software (a compiler or interpreter) to translate the program into machine language
  - Test the program
  - Put the program into production
  - Maintain the program

Understanding the Program Development Cycle (continued)



The program development cycle

Understanding the Problem

- Often considered one of the most difficult aspects of programming, if not the most difficult
- Key is to fully understand what the end users want and need from the program
- **Users** or **end users**
  - People for whom a program is written
- **Documentation**
  - Supporting paperwork for a program

Planning the Logic

- Planning how you will design the program
- An **algorithm** is the sequence of steps or rules you follow to solve a problem
- Most common planning tools
  - Flowcharts
  - Pseudocode
  - **IPO charts** (input, processing, and output)
- **Desk-checking**
  - Walking through a program's logic on paper before you actually write the program

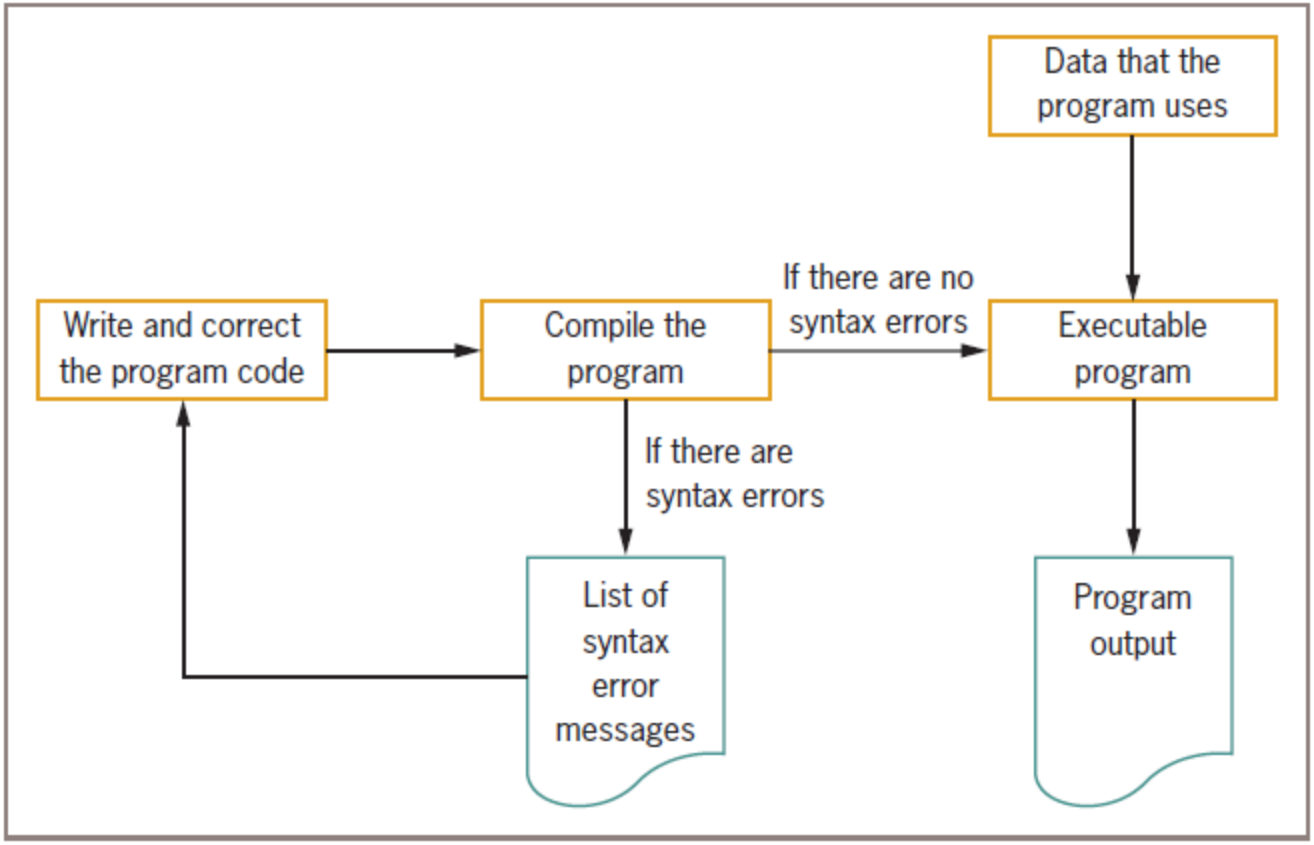
Coding the Program

- Hundreds of programming languages available
  - Choose based on features
  - Similar in their basic capabilities
- Coding is easier than the planning step
- Experienced programmers can successfully combine logic planning and program coding in one step

Using Software to Translate the Program into Machine Language

- Translator program
  - Compiler or interpreter
  - Changes the programmer's English-like **high-level programming language** into the **low-level machine language**
- Find **Syntax Errors**
  - Misuse of a language's grammar rules
  - Programmer corrects listed syntax errors
  - Might need to recompile the code several times

Using Software to Translate the Program into Machine Language (continued)



Creating an executable program

Testing the Program

- Find all the Logical Errors
  - Results when a syntactically correct statement, but the wrong one for the current context, is used
- Test
  - Execute the program with some sample data to see whether the results are logically correct
- **Debugging** is the process of finding and correcting program errors
- Programs should be tested with many sets of data

Putting the Program into Production

- Roll out the program to the end users
- Process depends on program's purpose
  - May take several months
- **Conversion**
  - The entire set of actions an organization must take to switch over to using a new program or set of programs

Maintaining the Program

- Making sure the program runs for years to come and adapts to meet new needs of the end users
- **Maintenance**
  - Making changes after the program is put into production
- Common first programming job
  - Maintaining previously written programs
- Make changes to existing programs
  - Repeat the development cycle

Using Pseudocode Statements and Flowchart Symbols

- To planning our logic we will use pseudocode and flowcharts to map out the program before we write it in code
- **Pseudocode**
  - English-like representation of the logical steps it takes to solve a problem
- **Flowchart**
  - Pictorial representation of the logical steps it takes to solve a problem

Writing Pseudocode

- Pseudocode representation of a number-doubling problem

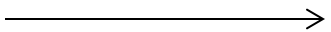
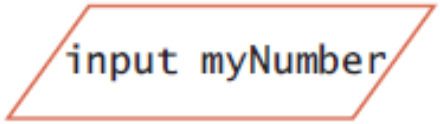
```
1. input myNumber
2. set myAnswer = myNumber * 2
3. output myAnswer
```

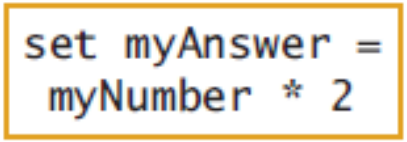
Pseudocode Guidelines

- Each step performs a single action
- The overall flow generally begins with input, then moves to processing, then to output
- If you have statements within a section of code, the statements are indented and the section is indicated with a start and stop
  - Example:

```
IF (value > 10)
    calculate value * 4
END IF
```

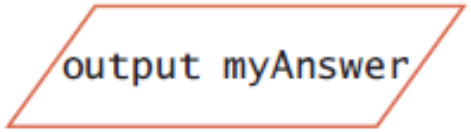
Drawing Flowcharts

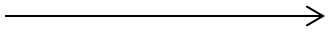
- Create a flowchart
  - Draw geometric shapes that contain the individual statements
  - Connect shapes with arrows 
- **Input symbol**
  - Indicates input operation
  - Parallelogram 

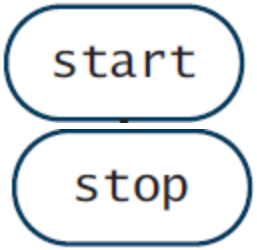
- **Processing symbol**
  - Contains processing statements such as arithmetic
  - Rectangle 

Drawing Flowcharts (continued)

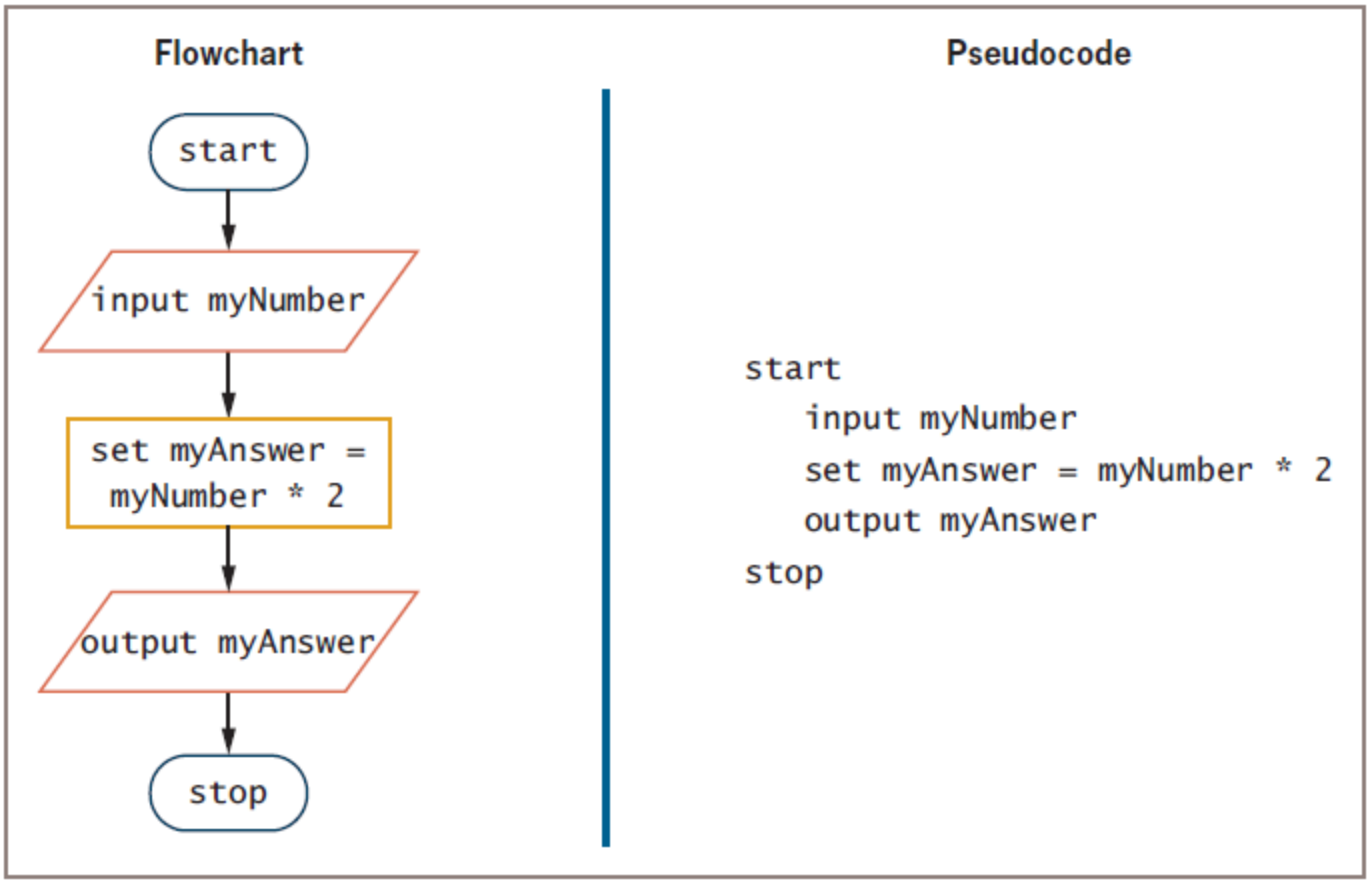
- **Output symbol**
  - Represents output statements
  - Parallelogram
- **Flowlines**
  - Arrows that connect steps
- **Terminal symbols**
  - Start/stop symbols
  - Shaped like a racetrack
  - Also called lozenges







Drawing Flowcharts (continued)

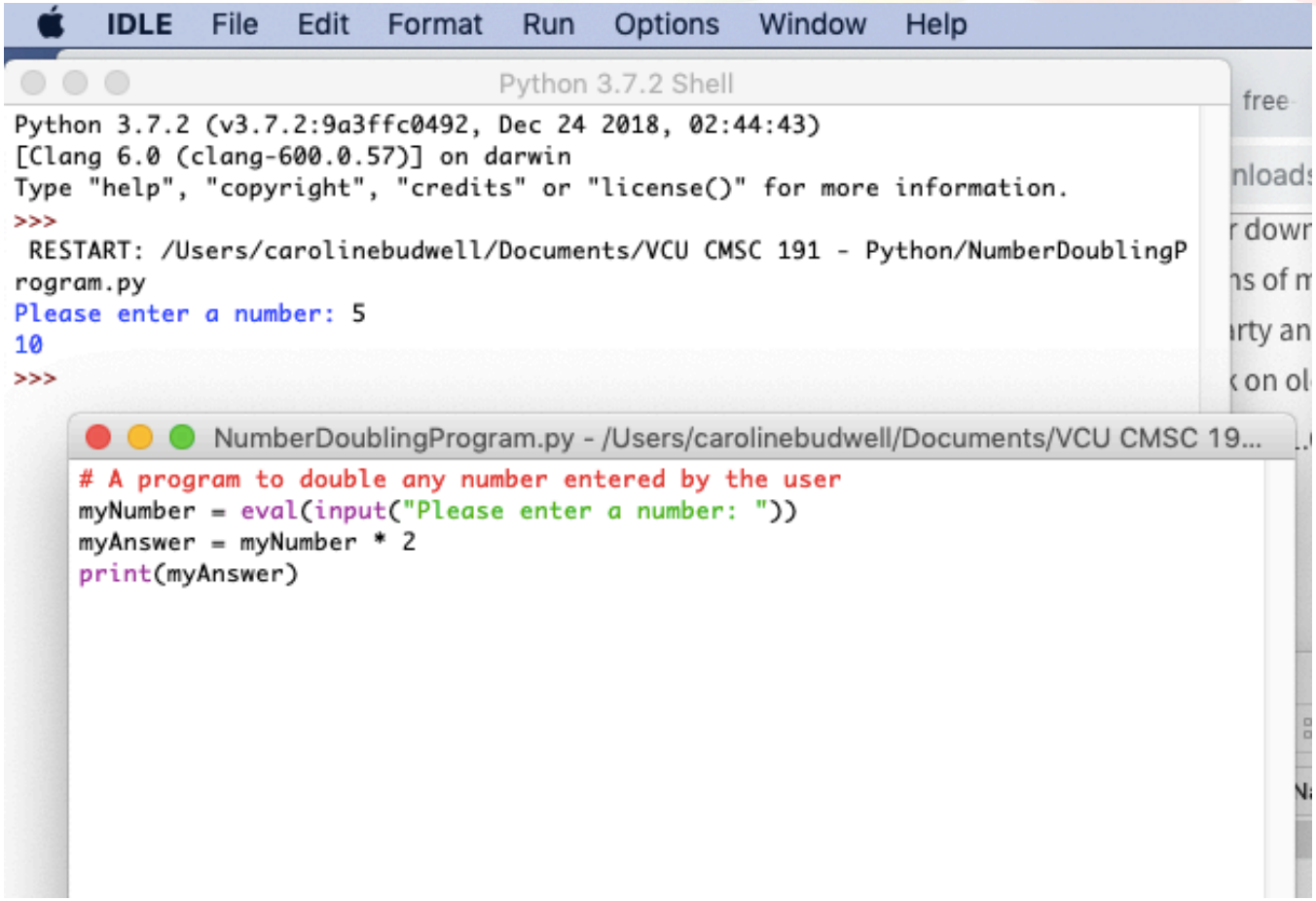


Flowchart and pseudocode of program that doubles a number

Understanding Programming and User Environments

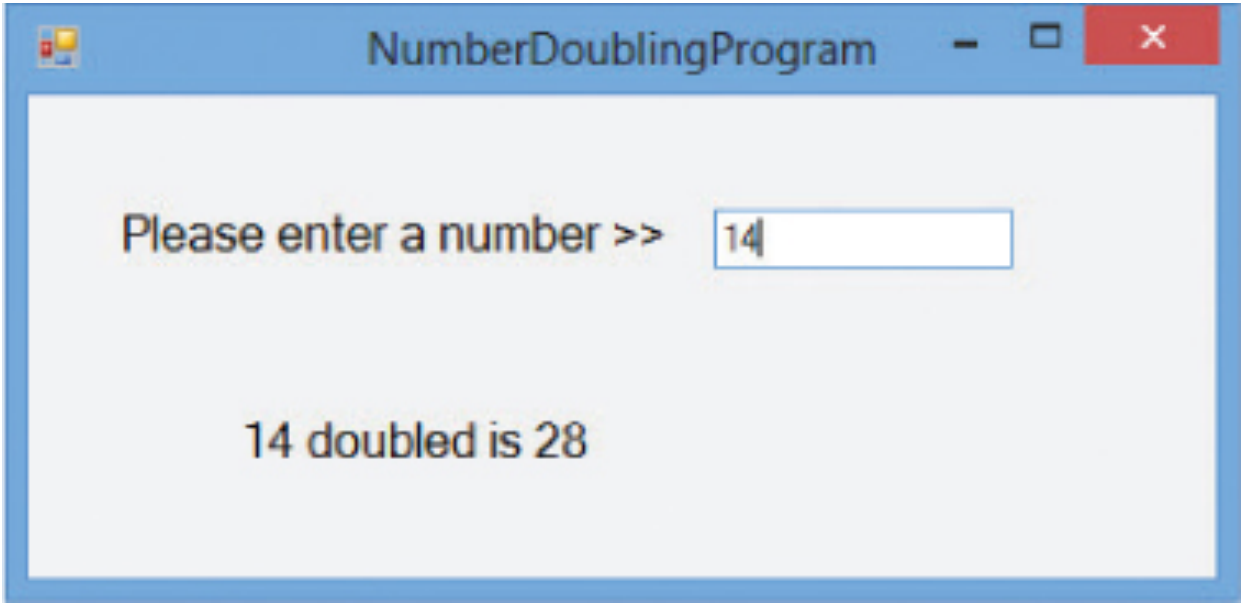
- **Understanding Programming Environments**
  - **Text Editor** is used to create simple text files
  - Integrated Development Environment (IDE) provides an editor, compiler, and other programming tools
    - Microsoft Visual Studio IDE
- **Understanding User Environments**
  - **Command Line** is a location on your computer screen at which you type text entries to communicate with the computer's operating system
  - A **graphical user interface**, or GUI (pronounced gooey), allows users to interact with a program in a graphical environment

Understanding Programming Environments (continued)



A Python number-doubling program in IDLE

Understanding User Environments (continued)



Executing a number-doubling program in a GUI environment

Understanding the Evolution of Programming Models

- People have been writing modern computer programs since the 1940s
- Newer programming languages
  - Look much more like natural language
  - Are easier to use
  - Create self-contained modules or program segments that can be pieced together in a variety of ways

Understanding the Evolution of Programming Models (continued)

- Major models or paradigms used by programmers
  - **Procedural programming**
    - Focuses on the procedures that programmers create
    - Usually all code is located within one file
  - **Object-oriented programming**
    - Focuses on objects, or "things," and describes their features (or attributes) and their behaviors
    - Code can be broken into different files to help protect code from unintended changes

Summary

- Hardware and software accomplish input, processing, and output
- Logic must be developed correctly
- Logical errors are much more difficult to locate than syntax errors
- Use flowcharts, pseudocode, and IPO charts to plan the logic
- Use a text editor or an IDE to enter your program statements