Unit 11
Programming

Computer Concepts 2016
ENHANCED EDITION

Unit Contents
- Section A: Program Development
- Section B: Programming Tools
- Section C: Procedural Programming
- Section D: Object-Oriented Code
- Section E: Declarative Programming

Section A: Program Development
- Programming Basics
- Program Planning
- Program Coding
- Program Testing and Documentation

Programming Basics
- Computer programming encompasses a broad set of activities that include planning, coding, testing, and documenting.
- A related activity, software engineering, is a development process that uses mathematical, engineering, and management techniques to reduce the cost and complexity of a computer program while increasing its reliability and modifiability.
- The instructions that make up a computer program are referred to as code because program instructions for first-generation computers were entered in binary codes.

Programmers typically specialize in either application programming or system development.
- Application programmers create productivity applications such as Microsoft Office.
- Systems programmers specialize in developing system software such as operating systems, device drivers, security modules, and communications software.
Program Planning

- In the context of programming, a **problem statement** defines certain elements that must be manipulated to achieve a result or goal.
- A good problem statement for a computer program has three characteristics:
  - It specifies any assumptions that define the scope of the problem.
  - It clearly specifies the known information.
  - It specifies when the problem has been solved.

Program Planning

- Several software development methodologies exist to help program designers and coders plan, execute, and test software.
- Methodologies can be classified as **predictive** or **agile**.
  - A **predictive methodology** requires extensive planning and documentation up front; it's used to construct buildings and assemble cars—tasks that are well defined and predictable.
  - An **agile methodology** focuses on flexible development and specifications that evolve as the project progresses.

Program Coding

- The core of a computer program is a sequence of instructions.
- A **keyword**, or command, is a word with a predefined meaning.
- Keywords differ depending on the programming language; there is a basic vocabulary that covers most necessary tasks.

**Figure 11-5: Keywords for the Python Programming Language**

- **input**: Collect information from the program's users.
- **print**: Display information on the screen.
- **while**: Begin a series of commands that will be repeated in a loop.
- **break**: Terminate a loop.

Program Coding

- **if**: Execute one or more instructions only if a specified condition is true.
- **else**: Add more options to extend the if command.
- **def**: Define a series of instructions that become a unit called a function.
- **return**: Transfer data from a function to some other part of the program.
- **class**: Define an object as a set of attributes and methods.
11 Program Coding

- Keywords can be combined with specific parameters, which provide more detailed instructions for the computer to carry out.
- These parameters include variables and constants.
  - A variable represents a value that can change.
  - A constant is a factor that remains the same throughout a program.

11 Program Coding

- The set of rules that specify the sequence of keywords, parameters, and punctuation in a program instruction is referred to as syntax.

11 Program Coding

- You may be able to use a text editor, program editor, or graphical user interface to code computer programs.
  - A text editor is any word processor that can be used for basic text editing tasks, such as writing email, creating documents, and coding computer programs.
  - A program editor is a type of text editor specially designed for entering code for computer programs.

11 Program Coding

- Programs that don’t work correctly might crash, run forever, or provide inaccurate results; when a program isn’t working, it’s usually the result of a runtime, logic, or syntax error.
  - A runtime error occurs when a program runs instructions that the computer can’t execute.
  - A logic error is a type of runtime error in the logic or design of the program.
  - A syntax error occurs when an instruction does not follow the syntax rules of the programming language.

11 Program Testing and Documentation

- Programs need to meet performance, usability, and security standards.
  - Performance – programmers need to carry out real-world tests to ensure that programs don’t take too long to load.
  - Usability – programs should be easy to learn and use and be efficient.
  - Security – program specifications are formulated so programmers remain aware of security throughout the software development life cycle.
Program Testing and Documentation

- Techniques associated with **defensive programming** include:
  - **Source code walkthroughs.** Open source software goes through extensive public scrutiny that can identify security holes, but proprietary software can also benefit from a walkthrough with other in-house programmers.
  - **Simplification.** Complex code is more difficult to debug than simpler code. Simplifying complex sections of code can sometimes reduce a program’s vulnerability to attacks.
  - **Filtering input.** It is dangerous to assume that users will enter valid input. Attackers have become experts at concocting input that causes buffer overflows and runs rogue HTML scripts. Programmers should use a tight set of filters on all input fields.

Section B: Programming Tools

- **Language Evolution**
- **Compilers and Interpreters**
- **Paradigms and Languages**
- **Toolsets**

Language Evolution

- When applied to programming languages, **abstraction** inserts a buffer between programmers and the chip-level details of instruction sets and binary data representation.
- For programming languages, abstraction automates hardware-level details, such as how to move data from memory to the processor.

- A **low-level language** has a low level of abstraction because it includes commands specific to a particular CPU or microprocessor family.
- A **high-level language** uses command words and grammar based on human languages to provide a level of abstraction that hides the underlying low-level language.

Language Evolution

- **First-generation** languages are the first machine languages programmers used.
- **Second-generation** languages added a level of abstraction to machine languages by substituting abbreviated command words for binary numbers.
- **Third-generation** languages were conceived in the 1950s and used easy-to-remember command words, such as PRINT and INPUT.
Language Evolution

- An assembly language is classified as a low-level language because it is machine specific
- An assembler typically reads a program written in an assembly language, which has two parts: the op code and the operand
  - An op code, which is short for operation code, is a command word for an operation such as add, compare, or jump
  - The operand for an instruction specifies the data for the operation

- Fourth-generation languages are considered “high-level” languages, and more closely resemble human languages
- The computer language Prolog, based on a declarative programming paradigm, is identified as a fifth-generation language—though some experts disagree with this classification

Compilers and Interpreters

- The human-readable version of a program created in a high-level language by a programmer is called source code
- Source code must first be translated into machine language using a compiler or interpreter
  - A compiler converts all the statements in a program in a single batch, and the resulting collection of instructions, called object code, is placed in a new file
  - An interpreter converts and executes one statement at a time while the program is running; once executed, the interpreter converts and executes the next statement
The phrase **programming paradigm** refers to a way of conceptualizing and structuring the tasks a computer performs.

A programmer uses a programming language that supports the paradigm.

Other programming languages—referred to as **multi-paradigm languages**—support more than one paradigm.
Section C: Procedural Programming

- Algorithms
- Pseudocode and Flowcharts
- Flow Control
- Procedural Applications

Unit 11: Programming

Algorithms

- The traditional approach to programming uses a procedural paradigm (sometimes called an imperative paradigm) to conceptualize the solution to a problem as a sequence of steps.
- A programming language that supports the procedural paradigm is called a procedural language; these languages are well suited to problems that can easily be solved with a linear, step-by-step algorithm.

Algorithms

- An algorithm is a set of steps for carrying out a task that can be written down and implemented.
- For example, the algorithm for making macaroni and cheese is a set of steps that includes boiling water, cooking the macaroni in the water, and adding the cheese sauce.
- Algorithms are usually written in a format that is not specific to a particular programming language.

Algorithms

- Steps for designing an algorithm:
  - Record the steps required to solve the problem manually.
  - Specify how to manipulate the information needed to calculate and solve the problem.
  - Specify how the computer decides what to display as the solution.

Pseudocode and Flowcharts

- You can express an algorithm in several different ways, including structured English, pseudocode, and flowcharts.
  - Structured English is a subset of the English language with a limited selection of sentence structures that reflect processing activities.
  - Pseudocode is a notational system for algorithms that is less formal than a programming language.
  - A flowchart is a graphical representation of the way a computer should progress from one instruction to the next as it performs a task.
Flow Control

- The key to a computer's ability to adjust to so many situations is the programmer's ability to control the flow of a program.
- Flow control refers to the sequence in which a computer executes program instructions.
- Programmers assign a sequential execution for computers to follow when performing program instructions.

A sequence control structure changes the order in which instructions are carried out by directing the computer to execute an instruction elsewhere in the program.

In the following simple program, a goto command tells the computer to jump directly to the instruction labeled "Widget":

```
print("This is the first line.")
goto Widget
print("This is the next line.")
Widget: print("All done!")
```

A function is a section of code that is part of a program but is not included in the main sequential execution path; a sequence control structure directs the computer to the statements contained in a function—when the statements have been executed, the computer returns to the main program.

The program asks the user to enter a number:
1. The program takes the number and jumps to the checkIn function.
2. In the checkIn function, the program makes sure the number is between 1 and 10.
3. A valid number between 1 and 10 is sent to the next function, where the number is maximized and printed.
Flow Control

- A selection control structure tells a computer what to do based on whether a condition is true or false; a simple example of a selection control structure is the if else command.

Flow Control

- A repetition control structure directs the computer to repeat one or more instructions until a certain condition is met.
- The selection of code that repeats is usually referred to as a loop or an iteration.

Procedural Applications

- Procedural languages encourage programmers to approach problems by breaking the solution down into a series of steps; the earliest programming languages were procedural.
- The procedural approach is best used for problems that can be solved by following a step-by-step algorithm.
- Programs using the procedural approach tend to run quickly and use system resources efficiently.
- The procedural paradigm is quite flexible and powerful, which allows programmers to apply it to many types of problems.

Objects and Classes

- The object-oriented (OO) paradigm is based on objects and classes that can be defined and manipulated by program code.
- It is based on the idea that the solution for a problem can be visualized in terms of objects that interact with each other.
- Rather than envisioning a list of steps, programmers envision a program as data objects that essentially network with each other to exchange data.
In the context of the OO paradigm, an **object** is a unit of data that represents an abstract or real-world entity, such as a person, place, or thing. Whereas an object is a single instance of an entity, a **class** is a template for a group of objects with similar characteristics.

**Objects and Classes**

A **class attribute** defines the characteristics of a set of objects. Each class attribute generally has a name, scope, and data type; its scope can be defined as **public** or **private**.

- A public attribute is available for use by any routine in the program.
- A private attribute can be accessed only from the routine in which it is defined.

**Inheritance**

In OO jargon, **inheritance** refers to passing certain characteristics from one class to other classes. The process of producing new classes with inherited attributes creates a class hierarchy that includes **superclass** and **subclasses**.

- A superclass is any class from which attributes can be inherited.
- A subclass (or derived class) is any class that inherits attributes from a superclass.

**Methods and Messages**

In an OO program, the objects interact; programmers specify how they interact by creating methods. A method is a segment of code that defines an action; the names of methods end in a set of parenthesis, such as `compare()` or `getArea()`.

- The code that is contained in a method may be a series of steps similar to code segments in procedural programs.
- **Polymorphism**, sometimes called overloading, is the ability to redefine a method in a subclass; it provides OO programmers with easy extensibility and can help simplify program control structures.

A method is activated by a **message**, which is included as a line of program code that is sometimes referred to as a call.

In the OO world, objects often interact to solve a problem by sending and receiving messages.
Methods and Messages

For classes and methods to fit together they must be placed within the structure of a Java program, which contains class definitions, defines methods, initiates the comparison, and outputs results.

The computer begins executing a Java program by locating a standard method called main(), which contains code to send messages to objects by calling methods.

Methods and Messages

OO Program Structure

OO Applications

In 1983, OO features were added to the C programming language, and C++ emerged as a popular tool for programming games and applications.

Java was originally planned as a programming language for consumer electronics, but it evolved into an OO programming platform for developing Web applications.

Most of today’s popular programming languages, such as Java, C++, Swift, Python, and C#, include OO features.

Section E: Declarative Programming

The declarative paradigm describes aspects of a problem that lead to a solution.

Programmers using declarative languages write code that declares, or states, facts pertaining to a program.

The Declarative Paradigm

The Declarative Paradigm

Procedural paradigm:
- Programs detail how to solve a problem
- Very efficient for number-crunching tasks

Object-oriented paradigm:
- Programs define objects, classes, and methods
- Efficient for problems that involve real-world objects

Declarative paradigm:
- Programs describe the problem
- Efficient for processing words and language
11 The Declarative Paradigm

➢ The programming language Prolog uses a collection of facts and rules to describe a problem
➢ In the context of a Prolog program, a fact is a statement that provides the computer with basic information for solving a problem; a rule is a general statement about the relationship between facts

11 Prolog Facts

➢ Prolog programming is easy to use; the punctuation mainly consists of periods, commas, and parentheses, so programmers don’t have to track levels and levels of curly brackets
➢ The words in the parentheses are called arguments, which represent one of the main subjects that a fact describes

11 Prolog Facts

➢ Prolog programming is easy to use; the punctuation mainly consists of periods, commas, and parentheses, so programmers don’t have to track levels and levels of curly brackets
➢ The words in the parentheses are called arguments, which represent one of the main subjects that a fact describes

11 Prolog Facts

➢ Each fact in a Prolog program is similar to a record in a database, but you can query a Prolog program’s database by asking a question, called a goal
➢ As an example, the following facts can easily be queried by entering goals:

```prolog
shapeof(pizza, round).
priceof(pizza1, 10).
priceof(pizza2, 12).
shapeof(pizza1, square).
priceof(pizza2, 14).
shapeof(pizza2, round).
```

11 Prolog Rules

➢ With just facts and goals, Prolog would be nothing more than a database
➢ The addition of rules gives programmers a set of tools to manipulate the facts
➢ Unlike other programming languages, the order or sequence of rules in a Prolog program is usually not critical to making sure the program works
Prolog Rules

The head of a rule defines an outcome or fact. In this case, the fact is that if \( \text{PizzaX} \) is a better deal than \( \text{PizzaY} \), then

\[
\text{betterdeal} \left( \text{PizzaX}, \text{PizzaY} \right) :-
\]

\[
\text{squareinchprice} \left( \text{PizzaX}, \text{AmountX} \right),
\]

\[
\text{squareinchprice} \left( \text{PizzaY}, \text{AmountY} \right),
\]

\[
\text{AmountX} < \text{AmountY}.
\]

Interactive Input

- In order for programmers to collect input from the user, they can use \textit{read} and \textit{write} statements.
- Read and write predicates collect user input.
- Prolog uses the \textit{write} predicate to display a prompt for input.
- The \textit{read} predicate gathers input entered by the user, and then creates a fact.

Declarative Logic

- Programmers need to determine how many conditions will apply to a program before starting to code facts and rules.
- A decision table is a tabular method for visualizing and specifying rules based on multiple factors.
- The decision table lays out the logic for the factors and actions and allows the programmer to see the possible outcomes.

Declarative Applications

- As a general rule, declarative programming languages are most suitable for problems that pertain to words and concepts rather than to numbers.
- Declarative languages offer a highly effective programming environment for problems that involve words, concepts, and complex logic.
- One of the disadvantages of declarative languages is that they are not commonly used for production applications—today’s emphasis on the OO paradigm has pushed declarative languages out of the mainstream, both in education and in the job market.
Unit 11 Complete

Computer Concepts 2016