SELECT cust_name
FROM customer_information
WHERE purchases > 100

**Fifth generation**

Fifth-generation languages are built on third- and fourth-generation languages. They are sometimes called non-procedural languages. **Procedural languages** require the programmer or user to write out the sequence of steps to solve a problem. **Non-procedural languages** only require the programmer to write the goals that have to be achieved and any constraints or parameters that are required. The computer solves the problem — it works out the required steps.

For this to work, the computer must have a knowledge base to work from, so its problem-solving abilities are limited to what has already been programmed. This type of language is used in artificial intelligence. Prolog is an example of a fifth-generation language. Here are two commands in Prolog:

```
?- fruit(apple).
?- fruit(x).
```

The first asks, Is an apple a fruit? The second asks, What things are fruits? The computer could respond:

```
Yes
X = apple
```

**EXERCISE 9.1**

Match each statement to the correct generation of programming languages.

**Statements**

a. A low-level language that is also called assembly language
b. A high-level language that uses prompts and assistants to make programming easy
c. A high-level language used in artificial intelligence that works out problem-solving steps for you
d. A low-level language, also called machine language, that uses binary
e. A high-level language that uses English-like statements

**Generations**

1. First
2. Second
3. Third
4. Fourth
5. Fifth

**ACTIVITY 9.1**

Copy and complete the table below. There may be more than one advantage or disadvantage for each generation.

| Table 9.2 Table for Activity 9.1 |
|-------------------------------|-----------------|-----------------|
| **Advantages**                | **Disadvantages** |
| First-generation languages    | Does not require translation | Requires translation |
| Second-generation languages   |                             | Limited to existing knowledge base |
| Third-generation languages    |                             | |
| Fourth-generation languages   |                             | |
| Fifth-generation languages    |                             | |

Now do Exercise 9.1.
Implementing a program

Chapter 7 introduced the steps for problem solving. Step 5 was to write a program in a programming language and step 6 was to test and debug the program. Steps 5 and 6 can be taken together and expressed as five stages that ensure smooth progress:

- **Stage 1**: Create the source code.
- **Stage 2**: Interpret and compile.
- **Stage 3**: Link.
- **Stage 4**: Test and execute.
- **Stage 5**: Maintain the program.

**Stage 1: Create the source code**

Step 1 is when you write the program using a programming language. The code you write using a high-level language or assembly language is known as source code. Chapters 10 to 12 explain how to write simple programs using the programming language Pascal.

**Stage 2: Interpret and compile**

The computer does not understand high-level source code. It requires an interpreter or compiler to translate the source code into object code, which is a machine language it can understand. Translators are discussed in Chapter 5.

Interpreters translate code one line at a time. Interpreters have to run on the same machine that runs the program, because the program is interpreted every time it is run. Compilers translate code in one go and create a stand-alone program. This program can be run on many different machines, which makes compilers more efficient than interpreters.

If there are errors in a program, it will not translate properly. If there is a syntax error (page 88), the translator will stop translating and may return an error message.

![Diagram](image)

*Figure 9.1 Assemblers, compilers and interpreters translate assembly language and high-level languages into machine code*
Stage 3: Link
When you write a long or complex program, you will probably write it in pieces or sections. Linking puts the sections together to create one piece of code.

Stage 4: Test and execute
Before you run, or execute, a program you should test it on paper to see if it works properly. Chapter 8 discusses some methods for testing a program. If there are errors in a program, it will not run properly.

Stage 5: Maintain the program
Once you have a working program that fulfils its purpose, try to maintain it. Perhaps it needs some extra features or could be restructured so it is easier to use. Go back to the program code and modify it.

Programmers often keep versions of the programs they write. Each version is given a number and the programmer will usually write release notes to accompany it. The release notes explain what is new or different about that version. If there is a new version of an operating system, such as a new version of Windows, application programmers may have to create versions of their applications that are compatible with the new version of the operating system. Now do Exercise 9.2.

Testing and debugging
Programs are tested to pick up errors. Three kinds of error can prevent a program from working: logic errors, syntax errors and run-time errors. Errors can be detected when the program is being translated, when you check it on paper, and when you execute it. Debugging is the process of fixing errors when you discover them.

Syntax errors
Code has to be written according to syntax rules, which dictate how your code structure statements, much like the rules of English grammar. A compiler cannot understand code when there is a syntax error. A good indicator that there is a syntax error in a program is when a translator cannot translate the program properly. An example of incorrect syntax is an IF statement without an accompanying THEN statement.

Logic errors
Logic errors occur when there is a mistake in the flow or sequence of a program. An example is a calculation that uses a variable before the user has entered the variable into the program. The program may run, but output may be inaccurate. Logic errors can be picked up by doing tests on paper and creating flowcharts that follow the logical flow of the program.

Run-time errors
Run-time errors occur when the program is being carried out. They may be the result of external factors, such as lack of computer memory or getting stuck in a loop. Run-time errors can occur when values lead to illegal operations, such as dividing by zero.
Types of testing

Test your program before you execute it. You can do this in two ways: testing on paper and computer testing. Dry-run tests and trace tables are methods of testing on paper. Dry-run testing is a method of checking that a program does not contain any logic errors. If the program uses loops, you can use trace tables to trace the logic of a program through multiple values. Both these methods are described and demonstrated in Chapter 8.

Computer testing is where you test a program on a computer by running it with as many different types and combinations of data as possible. In an IT environment, this task would go to a program tester. The tester runs the program using obvious test data and obscure test data to ensure the program will run in all circumstances and conditions.

For example, if a program requires a user to input numeric data, a tester must ensure that all kinds of numeric data will work, such as identical numbers, positive and negative numbers, real numbers, and values that are higher and lower than each other. This can help to avoid run-time errors when the program is used in a live environment.

Validation and verification

When a program is implemented in real time, it may be free of all kinds of errors and run perfectly. But a user may enter data that is faulty or invalid and causes the program to run incorrectly. To avoid this, programmers put data validation and verification checks in the program.

Validation is where a program checks that the data a person has entered is valid, but it cannot check the accuracy of the data entered. For example, if a user is required to enter an eight-digit password, the program can check that the user has entered eight digits, not more or less. Here are some other examples of data validation:

- Use data ranges to ensure a value falls in a specified range.
- Check that the correct format is used, such as a date or time format.
- Use a spell checker.
- Check that a field has not been left blank.
- Check that the correct data type is used, such as numeric values and not symbols.

Verification checks that the data entered is exactly correct. This is a time-consuming process and tends to be used only when data accuracy is crucial.

Now do Exercise 9.3.

Documenting programs

A programmer is not the only person to use a program. Many programs are written by more than one programmer. A program is not much use to a person if they do not know what it is or what it is used for. Documentation says what a program does and how it is used.
Internal documentation

Internal documentation is also called technical documentation. If you are looking at a long and complex program that you did not write, internal documentation can help you see how it works. There are several forms of internal documentation:

- A programmer can insert comments into the code. Comments are sentences intended for the user. They are coded so a computer does not read them. Comments are useful for explaining a piece of code or naming the program. A good programmer will structure a program so it is easy to read and does not require many comments.

- Indentation groups sets of statements to reveal the structure and flow of a program.

- White spaces are spaces in a program created by indentation and by writing statements on separate lines instead of on one line. White spaces help to make programs clear and easy to follow.

- Identifiers are used for different items of data in a program. Good identifiers clearly indicate the nature of the data. Chapter 10 has more about identifiers, comments and program structure.

Programmers may write and update their own technical guides. These may have useful indexes of the mnemonics used in coding plus details of how the program was written. Technical guides may contain diagrams, flowcharts, and algorithms to help explain the program. Other programmers can refer to technical guides to help them modify and maintain programs.

External documentation

External documentation is often called user documentation or user manuals. User manuals provide installation instructions, operation instructions and any other technical specifications that a user needs to run or install a program. Most general-purpose applications have a built-in help function, which is a digital copy of the user documentation.

If you have installed a program on a computer you may have seen a document called "readme" or similar in the folder that contains the program. The readme document has instructions on how to install the program and may have a few instructions on getting started.

Many popular general-purpose applications, such as Word, have instructions built into the program – as you install the program, a wizard guides you through the steps. When you create a new version of a program you have written, create a new version of the documentation.