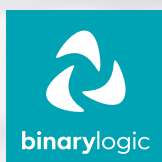


**Foundations
of Programming**

Software Engineering

SAMPLER

**Mc
Graw
Hill**





**Foundations
of Programming**

Software Engineering

Foundations of Programming: Software Engineering

Printed and distributed by McGraw Hill in association with Binary Logic SA.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopying, recording, or otherwise—without prior written permission from the publishers. No part of this work may be used or reproduced in any manner for the purpose of training artificial intelligence technologies or systems.

Disclaimer: McGraw Hill is an independent entity from Microsoft® Corporation and is not affiliated with Microsoft Corporation in any manner. Any Microsoft trademarks referenced herein are owned by Microsoft and are used solely for editorial purposes. This work is in no way authorized, prepared, approved, or endorsed by, or affiliated with, Microsoft.

Please note: This book contains links to websites that are not maintained by the publishers. Although we make every effort to ensure these links are accurate, up-to-date, and appropriate, the publishers cannot take responsibility for the content, persistence, or accuracy of any external or third-party websites referred to in this book, nor do they guarantee that any content on such websites is or will remain accurate or appropriate.

Trademark notice: Product or corporate names mentioned herein may be trademarks or registered trademarks and are used only for identification and explanation without intent to infringe. The publishers disclaim any affiliation, sponsorship, or endorsement by the respective trademark owners.

Windows is a registered trademark of Microsoft Corporation. “MIT” and “MIT App Inventor” are trademarks of the Massachusetts Institute of Technology. “Python” and the Python logos are registered trademarks of the Python Software Foundation. Jupyter is a registered trademark of Project Jupyter. The above companies or organizations do not sponsor, authorize, or endorse this book, nor is this book affiliated with them in any way.

Cover Credit: © dolgachov/123rf

Copyright © 2026 Binary Logic SA

MHID: 1265466068

ISBN: 9781265466060

mheducation.com binarylogic.net



Contents

1. Software Engineering 5

Lesson 1	Principles of Software Engineering	7
	Exercises	19
Lesson 2	Programming Languages and Languages Processors	22
	Exercises	32
Lesson 3	Software Development Tools	34
	Exercises	45

2. Prototyping 49

Lesson 1	Analysis	51
	Exercises	68
Lesson 2	Interaction Between the User and the Computer.....	71
	Exercises	78
Lesson 3	Creating a Prototype	80
	Exercises	93

3. Developing Applications with App Inventor 97

Lesson 1	Introduction to MIT App Inventor	99
	Exercises	124
Lesson 2	Adding More Elements to the App	125
	Exercises	142
Lesson 3	Programming the Mobile App	143
	Exercises	174

4. Software Accessibility and Digital Inclusion 177

Lesson 1	Testing and Deploying Applications	179
	Exercises	184
Lesson 2	Digital Inclusion.....	186
	Exercises	194
Lesson 3	Accessibility Features in an Application	196
	Exercises	219



Software Engineering

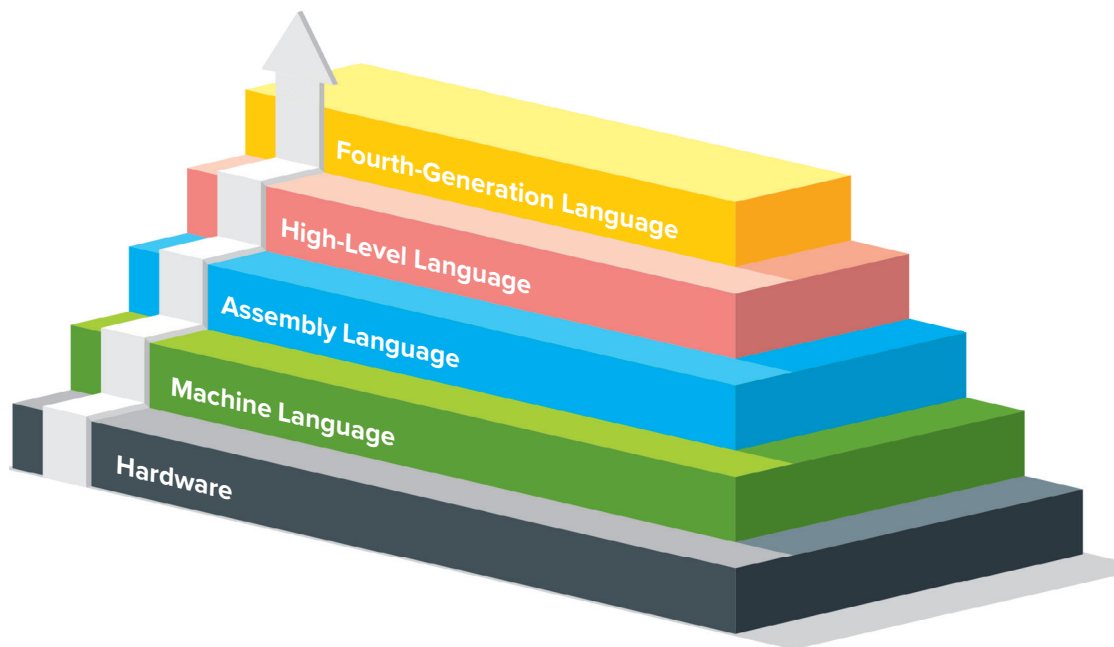
LESSON 2

Programming Languages and Languages Processors

A Brief History of the Development Programming Languages

Many things have changed since the creation of the first computer to the present day. Computer components and technologies have evolved greatly, as have advanced processing capabilities. Despite this, the concepts of computer operation formulated by von Neumann in 1945 still apply.

Programming languages were invented for the purpose of human-machine communication.



Machine language

For a computer to perform any required function, it must receive instructions in the form of binary numbers, consisting of 0s and 1s. This form of communication, known as "**machine language**", is not easily understood by humans. Machine languages are challenging for programmers to use and implement because they require an in-depth understanding of computer components and architecture. Additionally, each Central Processing Unit (CPU) has its own unique machine language, further complicating their application.

A machine language program is a sequence of binary bits, which are the instructions issued to the processor to carry out basic operations.

Assembly language

Between machine language and **high-level programming languages**, there is an intermediate language called **assembly language**, also called symbolic programming language.

Assembly language is similar to machine language but is somewhat easier to program as it allows the programmer to replace numbers (0, 1) with symbols.

Human comprehensible assembly language commands are converted into corresponding sequences of 0s and 1s for the computer to understand and execute.

For example, in assembly language, the word ADD, followed by two numbers, is used for addition. This is easy to understand and memorize for humans but must be translated into a series of bits in the computer to carry out the required operation. This translation process is carried out by a special program called the assembler.

Assembly language commands are made up of symbolic segments that correspond to machine language commands.

Challenges of assembly language

- The use of assembly language makes it easier to program simple operations of unintelligible binary sequences, but it is nevertheless considered a low-level language.
- The assembly language used varies depending on the architecture of each computer.
- Assembly language does not provide commands to perform more complex functions than simple additions, multiplications, and comparisons, forcing the programmer to write long and complex programs that are difficult to understand and debug.
- A program cannot be transferred from one computer to another of different architecture.

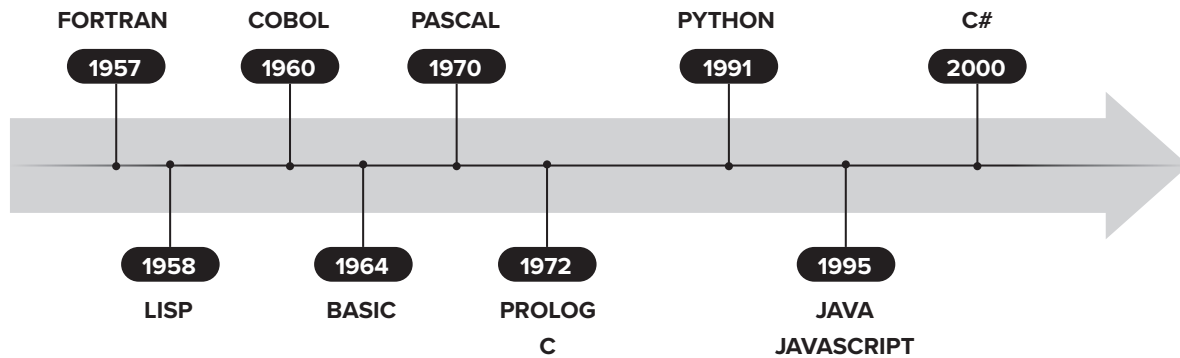
The following table presents a program with an addition written in a high-level programming language and its equivalent in assembly and machine language for a computer with a 6502 8-bit CPU. The high-level language program can be used on most computers, while the assembly and machine translations will work only on a computer with the same CPU architecture.

Calculating an addition		
High-level language	Assembly language	Machine language
sum = 0	LDA #0	10101001 00000000
	STA sum	10000101 00000000
	LDA sum	10100101 00000000
	CLC	00011000 00000000
sum = sum + 5	ADC #5	01101001 00000101
	STA sum	10000101 00000000
print (sum)	LDA sum	10100101 00000000
	JSR print	00100000 11100001

High-level programming languages

The shortcomings of machine language and assembly language led to a concerted effort to achieve better human-machine communication, which resulted in the emergence of the first high-level programming language in the 1950s.

High-level programming languages use programming commands which resemble human language. The resulting programs must be translated into machine language by the computer itself, using a special program called the translator. Compilers and interpreters are types of translators used for different types of programming languages.



The evolution of high-level programming languages

The developer chooses the programming language that allows the application to be easily developed in a given environment to implement a software solution, but at the same time, the developer also chooses the language based on their personal knowledge, skills, and preferences.

Each programming language has a unique set of reserved words (words that the language understands) and a syntax that the programmer uses to write instructions.

Basic information of programming languages			
Programming language	Developer	Etymology	Properties
FORTRAN	IBM	FORMula TRANslation	Suitable for solving mathematical and scientific problems but not suitable for managing data files, for example.
LISP	MIT	LISt Processor	A language for artificial intelligence.
COBOL	CODASYL	Common Business-Oriented Language	Suitable for developing commercial applications and general management applications.
BASIC	Dartmouth College	Beginner's All Purpose Symbolic Instruction Code	A multi-domain programming language.

PASCAL	Professor Nicholas Wirth	Named after the mathematician Blaise Pascal	It is famous for introducing structured programming techniques. It adopts program design in a systematic and accurate manner.
C	Dennis Ritchie and Bell Labs	The C language is named after a prior language named B	It is used for UNIX operating system development and is suitable for different operating systems.
JAVA	Sun Microsystems	Named after a type of coffee (Java)	It is an Object-Oriented programming language used to develop applications that can run on a wide range of computers or different operating systems.

Features of high-level programming languages

High-level programming languages have several advantages over assembly language:

- They use logic and programming formulas that are understandable and close to human language.
- They are independent of the type of computer and can be used on any device with or without minor modifications.
- Developers can learn high-level programming languages more easily and quickly.
- Software debugging and maintenance are much easier.

In general, high-level programming languages reduce the time and cost of software development significantly compared to low-level programming languages.

Fourth-generation programming languages

Among high-level programming languages, you note that there are so-called fourth-generation programming languages, which are usually abbreviated as 4GL. Fourth-generation programming languages are closer to human language than other high-level languages and are accessible to people without formal training as programmers because they require less coding.

Fourth-generation languages are more programmer-friendly and enhance programming efficiency by using English-like words and phrases, as well as icons, symbolic representations, and graphical interfaces when needed. The key to achieving efficiency with 4GL is compatibility between the tool and the field of application.

Computer users in fourth-generation languages can make changes to the program in order to meet a new need and have the ability to solve small problems by themselves. Multiple joint operations can be performed using a single command entered by the programmer.

Scripting languages are a type of programming language typically interpreted rather than compiled. They are used to automate repetitive tasks, simplify complex operations, and enable rapid prototyping of software systems. Some common examples of scripting languages include JavaScript, Ruby, PHP, and Perl. They often feature rich libraries and focus on productivity, making them ideal for tasks requiring quick development and iteration. However, they may not be as efficient or scalable as compiled languages and may not be suitable for performance-critical or resource-intensive applications.

For data operations, a user can create queries and reports using SQL for statistics and scientific projects, and a mathematician or researcher can use software, such as SPSS, MATLAB, and LabVIEW, to analyze this data.

Classifications of Programming Languages

There are many classifications of programming languages. Languages can be classified in terms of the type of commands used, such as procedural programming languages and object-oriented programming languages.

Procedural programming uses a set of instructions to tell the computer what to do step by step. Examples of procedural programming languages are COBOL, Fortran, and C.

In object-oriented programming, the program is divided into units called objects. Examples of object-oriented programming languages are C#, C++, Java, and Python.

Programming languages can also be classified according to what they are used for:

1. General programming languages: In theory, any general programming language can be used to solve any problem, but in practice, each language is designed to solve a specific type of problem. These languages are divided as follows:
 - Science-oriented languages such as Fortran.
 - Business-oriented languages such as COBOL.
 - Multi-domain languages such as BASIC and Pascal.
 - Operating system programming languages such as C.
 - Artificial intelligence languages such as PROLOG.
 - Specialized database management languages such as SQL.
2. Specialized languages, such as LISP, are used for a specific type of application such as robotics or integrated circuits.

How computers understand programming languages

Any program written in any programming language is converted into machine language that can be understood and executed by a computer, through the use of special translation programs.

There are two ways to run programs written in a high-level language. The most common is to compile the program with a **compiler**, but in some programming languages, an **interpreter** is used instead.

Let's go over how to implement these two different methods.

Compiler

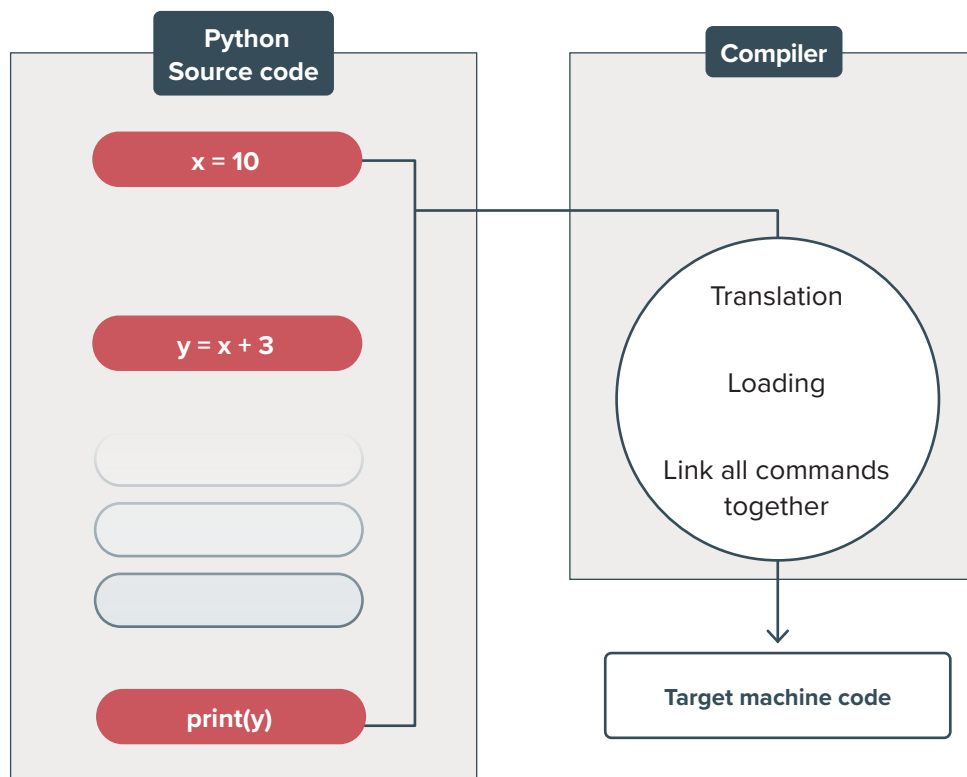
A compiler is a computer program that converts an entire block of code written in a high-level programming language into machine language, which is understood by the computer's processor.

Interpreter

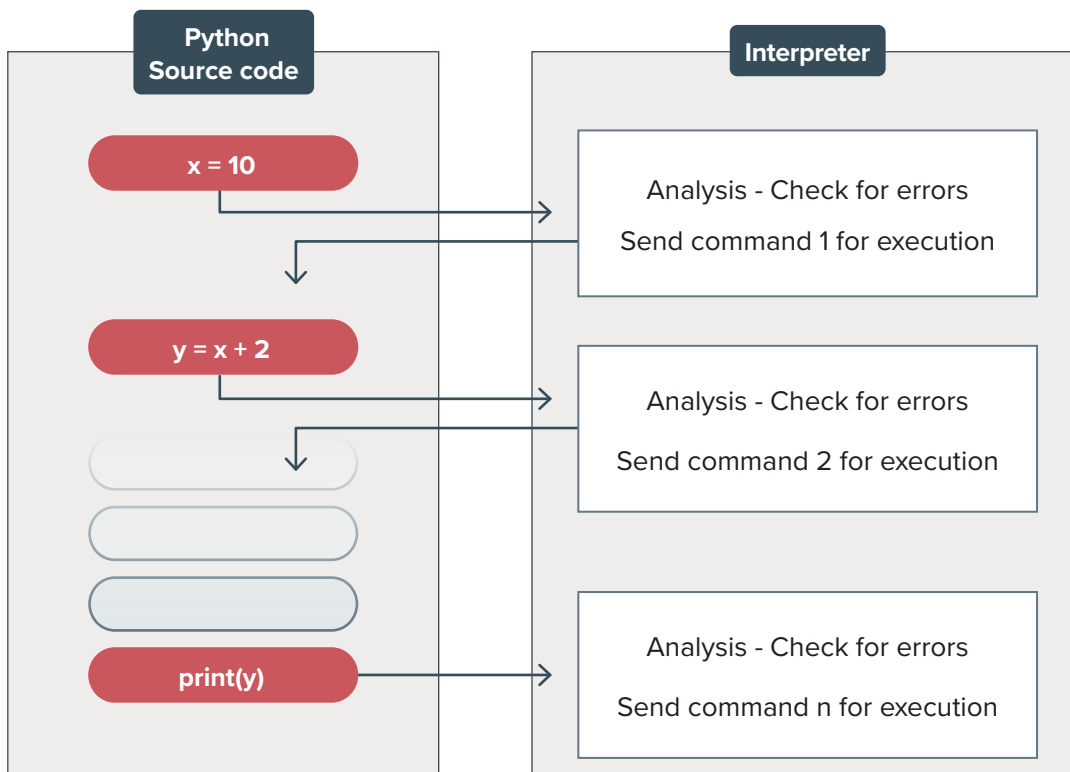
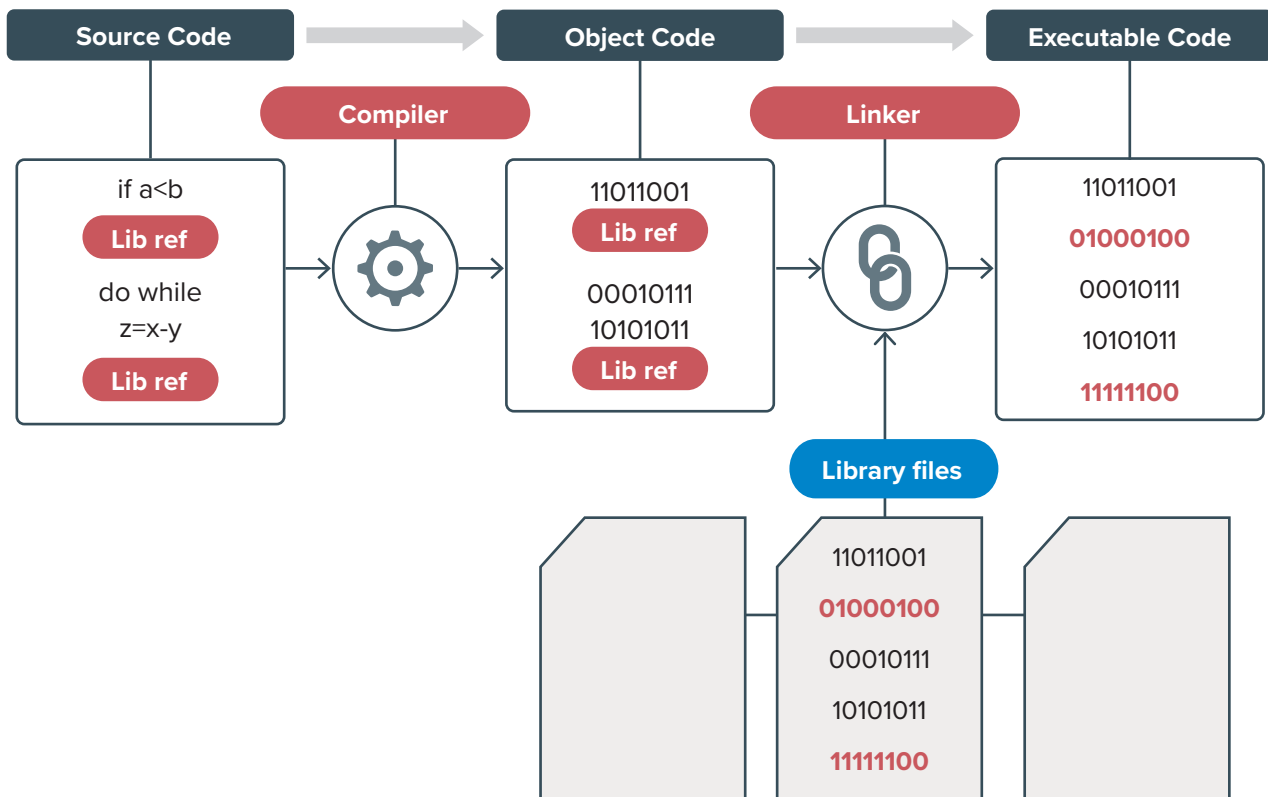
An interpreter is a computer program that converts each line of code from a block of code written in a high-level language into machine language and sends it for execution directly before moving on to the next line of code.

Program translation and linking process:

- The compiler accepts a program written in a high-level language as the input file (the source code), and produces an equivalent machine language program called object code.
- The compiler cannot compile statements that refer to standard libraries or resources outside of the source code, so the process will require an additional step of linking and converting those statements.
- Another program called the **Linker** or Loader handles the linking process and links the object code file with the standard library files, and produces the executable code, which is the final program that the computer executes.



The source code is the program written in a high-level programming language.



Compilers and interpreters perform the same task, which is to convert the program written in the high-level programming language into machine language, but in two different ways.

Interpreted and Compiled Programming Languages

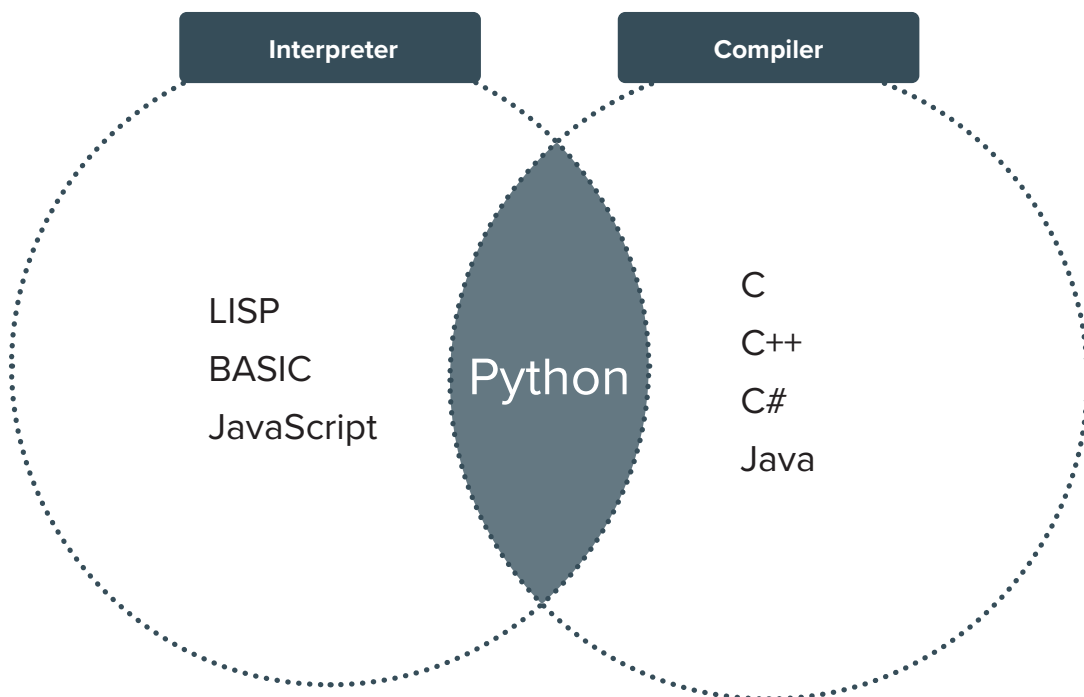
Most modern programming languages use compilers to produce optimized code quickly, but there are some languages that still use interpreters when there is a need to create a simple program for which speed is not the primary concern.

Compiled languages

The C, C++, C#, and Java programming languages use a compiler to build fast, reliable programs. The executable code is created for each type of computer hardware, which means that developers have to know what the end user's computer hardware is.

Interpreted languages

Legacy JavaScript, LISP, and BASIC use interpreters, which means that programs run slowly but their source code can run on any computer that has a particular programming language's interpreter. For example, a web application written in JavaScript can run on a Windows computer or an Android tablet with a web browser that integrates the JavaScript interpreter.



Python is both an interpreted and a compiled language. The Python application compiles each line of code so that it can be read by the interpreter on the hardware in use. The syntax used by the programmer doesn't change because the Python application converts it into the correct form for the interpreter used on that hardware.

Compiler vs Interpreter comparison

	Compiler	Interpreter
Main function	Converts the entire source code written in a high-level programming language into machine language, and produces an executable program .	Converts the block of code into machine language so that it translates and then executes the block of code, moving to the next block while the program is running.
Input	The compiler takes the entire source code as input.	The interpreter takes one of the source code instructions as input each time.
Output	The compiler generates and stores an object code file as output.	The interpreter does not generate an object code file.
Memory	Requires more memory due to object code generation.	Less memory is required.
Implementation process	The compilation process takes place for the entire source code before execution begins.	The interpretation process for each code statement takes place in parallel with the execution process.
Error checking	The compiler displays all language errors and warnings when compiling the program. You cannot run the program until all errors are corrected.	The interpreter reads one line of code and displays any errors in it. This error must be corrected before moving on to the next line.
Link files	Needs a program to associate the object file with standard library files to create the executable.	Does not need the link process, and does not create an executable file.
Speed	Availability of .exe file makes execution faster.	The execution process is slower because the executable file is not available. The program is interpreted again on each execution.
Dependence on hardware and operating systems	The executable file generated by the compiler depends on the target hardware. It cannot run on different CPU architectures or different operating systems.	The interpreter is a hardware and operating system that is independent. For example, a Python interpreter can run on Windows and Linux with the same source code and give the same results.

Dealing with Software Errors

Compilers and interpreters operate differently when they face errors and bugs in the source code.

Compiler:

1. Program creation.
2. The compiler will analyze and process all lines of code and make sure that they are correct.
3. If there is an error, an error message will appear.
4. If there is no error, the compiler will convert the source code into machine language.
Multiple code files will be associated with a single executable program (known as an EXE file).

Interpreter:

1. Program creation.
2. The interpreter reads one line of code and displays any syntax error. This error must be corrected before moving on to the next line.
3. All the lines of the source code are executed line by line during program execution by the interpreter.

Correction of errors during the debugging process

The source code in its first version may often contain many errors, which are divided into three types:

- Logical errors: Errors in the logic of the program.
- Runtime errors: Errors that occur during the execution of the program.
- Syntax errors: Errors in the syntax of the code.

Logical errors and runtime errors only occur when the program is executed, while syntax errors occur during compilation. The program is executed only if the source program contains no syntax errors.

Debugging syntax errors:

- In the first step, the compiler or interpreter detects syntax errors and presents messages indicating the error and its location. Some of them can specify the cause of the error.
- The next step is to correct errors in the program.
- Finally, the corrected program compiles correctly, without any error messages.

EXERCISES

- 1** What are the shortcomings of assembly language?
- 2** Analyze the role of high-level programming languages in software development by identifying three key advantages. How do these advantages influence software efficiency, scalability, and developer productivity?
- 3** For each project below, identify the most appropriate programming language classification (science-oriented, business-oriented, multi-domain, operating system programming, artificial intelligence, specialized database management, or specialized languages) and provide reasoning based on the characteristics of the language and its usage.
 - 1.** A scientific research team is creating a simulation to model complex chemical reactions. The language must be able to perform heavy mathematical computations and solve scientific equations efficiently. Which classification of programming language is most suited to this project?
 - 2.** A team of database administrators is building a new database management system for a company that handles millions of customer records. The language needs to support efficient querying and management of large data sets. Which classification of programming language is most appropriate here?
- 4** Choose the correct option to complete each sentence.
 - 1.** _____ accepts the source program as input and produces an equivalent machine language program called _____.
 - A.** The interpreter, object code
 - B.** The compiler, object code
 - C.** The source code, runtime errors
 - D.** The compiler, syntax errors

2. The _____ used by the interpreter is less than that used by the compiler.
- A. object code
 - B. link
 - C. memory
 - D. syntax errors
3. Using _____ is an advantage in terms of real-time debugging, but program execution is slower.
- A. the compiler
 - B. syntax errors
 - C. interpreted languages
 - D. object code
4. The compiler cannot convert statements that refer to _____, so it needs to concatenate and convert those statements.
- A. syntax errors
 - B. standard libraries
 - C. memory
 - D. runtime errors
5. The executable can be created if there are no _____ in the source program.
- A. runtime errors
 - B. syntax errors
 - C. memory issues
 - D. object code
6. Errors that occur during program execution are called _____.
- A. runtime errors
 - B. syntax errors
 - C. memory leaks
 - D. standard libraries

LESSON 3

Software Development Tools

Software Development Tools and Programs

Developers use a wide range of tools to develop software applications, each of which has its advantages and disadvantages. The programming process requires developers to be flexible and creative to take full advantage of the capabilities of different **software development tools** to deliver high-quality work for their clients.

Software development tools and programs are used to assist the software development team in various tasks, including creating, modifying, and maintaining programs, as well as debugging and implementing software tasks and development processes. There are also many specialized programs that provide or support specific tasks in the stages of the software development cycle.

Classification of software development tools

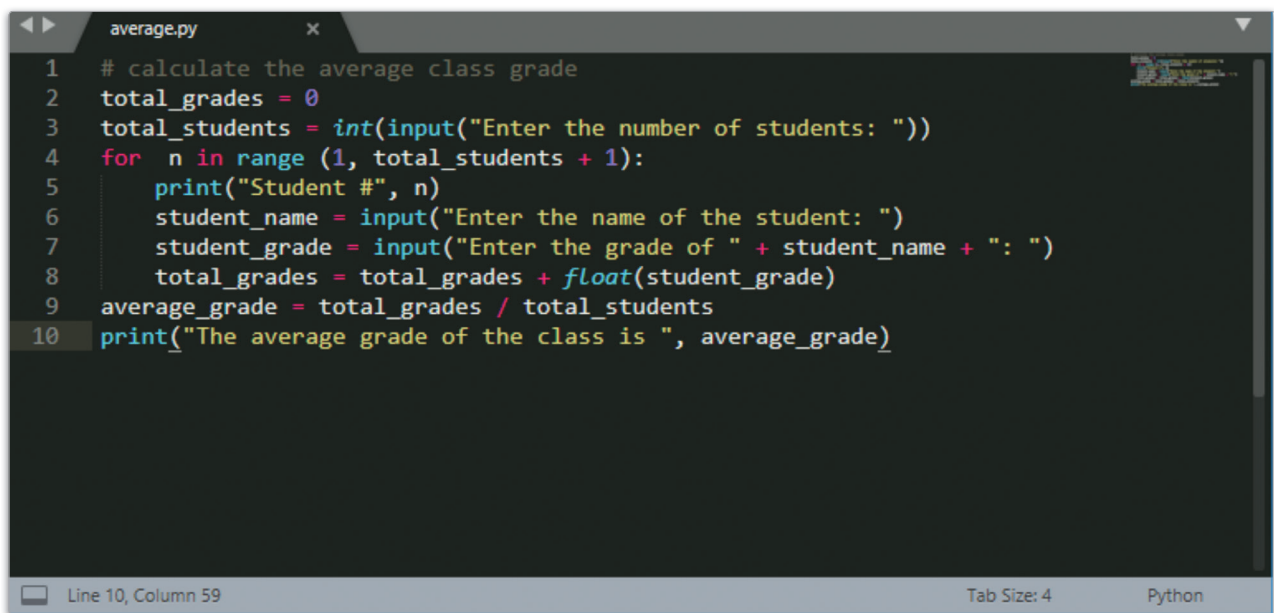
Software development tools	Description
Code Editors	Used to write and make changes to code.
Compilers and Linkers	Translate programs into executable machine language.
Debuggers	Help us correct errors in the software.
Project Builders	Make sure that all the necessary files will be compiled and linked to one final program.
Code Management Tools	Ensure that program files are not accidentally replaced when multiple programmers are working on the same program concurrently.
Integrated Development Environment (IDE)	Provides programmers with an integrated software environment that includes a text editor, compiler, linker, and debugger.
Profilers	Usually give us a good idea of the program's needs and handling of processor time and memory resources while running.
Network Analyzers	Necessary when writing software for networking applications in particular.
Database Explorer and Analyzer	Allows dealing with databases and analyzing the performance of specific database queries.

Code Editors

A **code editor** allows us to create and edit several connected programming language files, and usually it can handle many different languages like HTML, CSS, JavaScript, PHP, Ruby, Python, C, etc. Code editors use indents and different colors to format the code into code sections. This makes them much more suitable for writing code than ordinary word processors and text editors like Microsoft Word or Notepad.

Features of code editors

- Error-checking
- Auto-completing and code suggestions
- Code snippets
- Syntax highlighting
- Facilitate navigation of code files and resources
- Adding more functionality via extensions

A screenshot of a code editor window with a dark theme. The title bar shows 'average.py' and a close button. The code is written in Python and is syntax-highlighted. Line numbers 1 through 10 are visible on the left. The code calculates the average grade of a class. The status bar at the bottom indicates 'Line 10, Column 59', 'Tab Size: 4', and 'Python'.

```
1 # calculate the average class grade
2 total_grades = 0
3 total_students = int(input("Enter the number of students: "))
4 for n in range(1, total_students + 1):
5     print("Student #", n)
6     student_name = input("Enter the name of the student: ")
7     student_grade = input("Enter the grade of " + student_name + ": ")
8     total_grades = total_grades + float(student_grade)
9 average_grade = total_grades / total_students
10 print("The average grade of the class is ", average_grade)
```

There are many code editors that can be chosen by the programmer according to their preferences. The only criterion for choosing an editor is the efficiency of that editor for the required task. Some examples of code editors are:

- | | |
|----------------------|-------------|
| • Sublime Text | • Coda 2 |
| • Atom | • Notepad++ |
| • Visual Studio Code | • Vim |
| • Espresso | • BBedit |
| • Python IDLE | • Ultraedit |

Advantages and challenges of using code editors

Advantages

- They can rival the Integrated Development Environment (IDE) Editor for standard programming tasks when appropriate extensions to support different programming languages are used.
- They are smaller and faster to load than IDEs.
- Their streamlined interfaces make it easy to focus on our code.

Challenges

- They lack a lot of editing features that only IDEs provide, such as smart editing.
- Users may need to configure the code editor with the appropriate extensions before use.

Integrated Development Environments

Integrated development environments (IDEs) are usually presented with their own built-in applications, which include a number of software development tools such as an interpreter for use during the program creation phase, and a compiler for finalizing and publishing the program.

Modern integrated development environments are not limited to providing just a compiler for the programming language, but rather contain all the necessary programs and tools to help write and implement code, and most importantly, to diagnose and correct programs. Among the most important tools included in integrated programming environments are:

- File Explorer
- Code Editor
- Interpreter
- Compiler
- Linker
- Debugger
- Output Viewer

IDEs must include an editor dedicated to facilitating the creation of graphical objects, such as forms, menus, and dialog boxes, in order to provide the developer with the appropriate tools to create the code blocks related to these objects.

Features of IDEs

- Smart completion of code in the code editor.
- Integration with code management tools for version control.
- Advanced testing tools for debugging and validation
- Automatic linking of source code libraries.
- Tools for automating code creation and deployment.

All these tools and services are accessible through a unified user interface.

Examples of IDEs

In the past, most IDEs supported only one programming language and were usually created by the software companies or organizations that created that specific language.

Today, most software development projects integrate different technologies and programming languages, which requires IDE development environments that can support a wide range of languages.

For example, Microsoft Visual Studio supports C, C++, C#, VB.Net, Python, Ruby, Node.js, JavaScript, HTML/CSS, etc.

Other popular IDE tools include NetBeans, Eclipse, Atom-IDE, Xcode, Android Studio, IntelliJ IDEA, that and PyCharm.

Xcode is used to develop mobile application software for iOS devices. For Android devices, Android Studio is used.

Advantages and challenges of using Integrated Development Environments (IDE)

Advantages

- Provide intelligent code completion and analysis tools for faster programming with less errors.
- They provide powerful code browsing and discovery tools and make it easy to access any part of the program, no matter how large the project.
- They offer multiple ways to debug and test code without leaving the editor.
- They support many programming languages natively and provide many code navigation and code analysis tools to facilitate work and productivity on large projects.

Challenges

- The interfaces are packed with a lot of features that can make them complicated and difficult to use.
- They require a certain amount of training to use them correctly.
- Excessive functions often lead to slow performance.

Cloud software environments

Besides traditional software development environments, there are web-based cloud development environments, such as Amazon Cloud9.

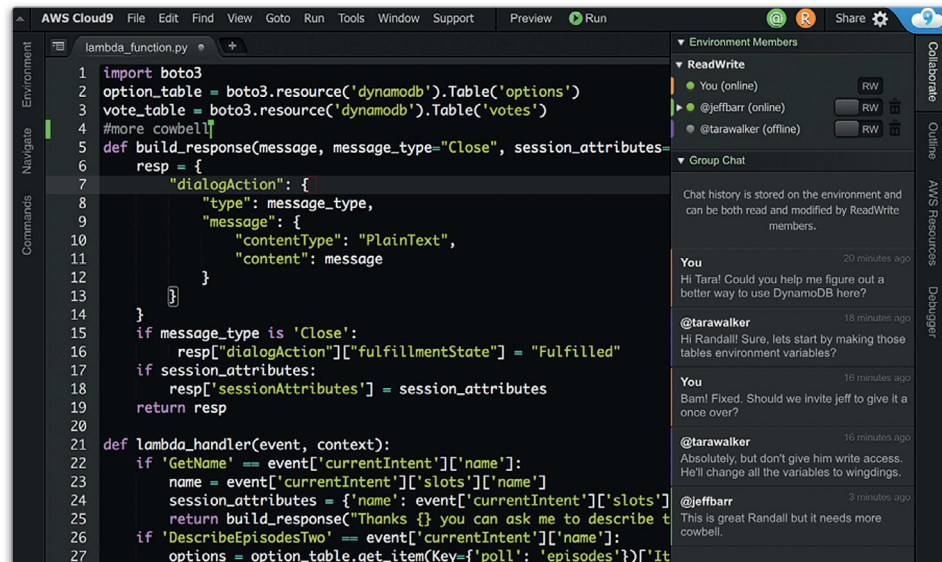
Cloud software environments provide the ability to work on your project from any computer, anywhere in the world, as your software development project data resides in the cloud.

One of the main drawbacks of these environments is the need to connect to the Internet to access data and do work.

Advantages of using cloud software development environments

- Access to software development tools from anywhere in the world.
- Possibility of using any device with a web browser.
- There are no requirements to download and install the software environment.
- Can facilitate collaboration between remote developers.

Programmers spend most of their programming time in testing and debugging, so integration of the code editor with the compiler and debugger is very convenient. This is the main feature of the IDE.



Specialized Tools for Specific Stages of Software Development

Creating professional software solutions requires working in a team and using a variety of tools that are not limited to the programming stage only but extend to the process as a whole.

There are many tools that can be used during the SDLC of a software product, and it can be difficult to list all the software and other essentials needed to develop business software, but a selection of these tools are described below.

Prototype creation

A software prototype is usually an organization chart, an image, or a set of images that represent the functional elements of an application, or it may be a website used to map out applications or the structure and functionality of the website.

Examples of tools used:

- Pencil
- Balsamiq Mockups
- Adobe Xd

Version control management—source code

The source code is subject to many modifications during the development process, and it may be necessary to undo certain steps in the program or reuse code that has been changed or deleted. When working in a team of programmers, two or more may need to work on the same files at the same time and make changes to the same code.

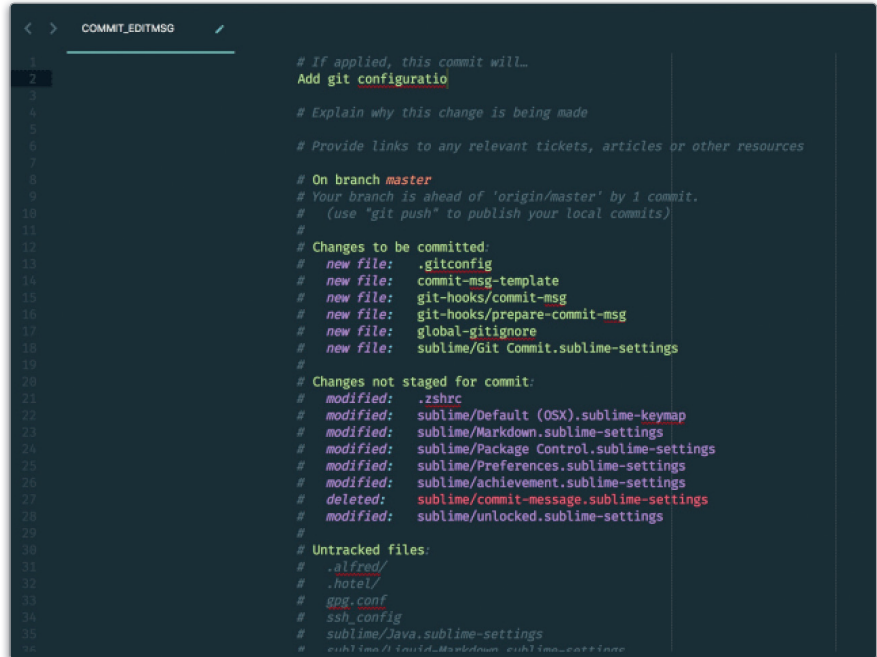
The tool we can use to control this process is called "version control management" or "code management". This tool enables the following:

1. Different team members can access the source code simultaneously without creating work conflicts.
2. Previous versions of code files can be kept for reference when some problems occur.

Version control uses a repository to record all changes made and creates a working copy of the project's code files, sometimes called a checkout copy, when a programmer wants to work on the code. All changes to the code are approved by the version control management software when changes to the code are saved to the repository.

Examples of tools used:

- Git
- Subversion
- Mercurial
- Azure DevOps
- DiffMerge



```
COMMIT_EDITMSG
1
2 Add git configuratio
3
4 # Explain why this change is being made
5
6 # Provide links to any relevant tickets, articles or other resources
7
8 # On branch master
9 # Your branch is ahead of 'origin/master' by 1 commit.
10 # (use "git push" to publish your local commits)
11
12 # Changes to be committed:
13 #   new file:   .gitconfig
14 #   new file:   commit-msg-template
15 #   new file:   git-hooks/commit-msg
16 #   new file:   git-hooks/prepare-commit-msg
17 #   new file:   global-gitignore
18 #   new file:   sublime/Git Commit.sublime-settings
19 #
20 # Changes not staged for commit:
21 #   modified:   .zshrc
22 #   modified:   sublime/Default (OSX).sublime-keymap
23 #   modified:   sublime/Markdown.sublime-settings
24 #   modified:   sublime/Package Control.sublime-settings
25 #   modified:   sublime/Preferences.sublime-settings
26 #   modified:   sublime/achievement.sublime-settings
27 #   deleted:    sublime/commit-message.sublime-settings
28 #   modified:   sublime/unlocked.sublime-settings
29
30 # Untracked files:
31 #   .alfred/
32 #   .hotel/
33 #   .pgp.conf
34 #   ssh_config
35 #   sublime/Java.sublime-settings
36 #   sublime/Markdown.sublime-settings
```

Code deployment

Until a few years ago, it was easy to deploy an application since the compiled output of the program was placed on a disk ready to use.

With the advent of the Internet, it became necessary to "publish" applications via the Internet, as installable software through application stores or directly as web applications, and accordingly special programs and tools appeared for publishing code on the web.

Examples of tools used:

- TeamCity
- Google Cloud Deployment Manager
- GitLab
- Jenkins
- AWS CodeDeploy
- Azure DevOps

Testing

Testing is not just debugging the code, but also includes testing the operation of the program and the effectiveness of its use by a large number of users as well as performing security and other tests.

Examples of tools used:

- Apache JMeter
- Ghostlab
- Selenium
- Telerik Test Studio
- Azure DevOps
- IronWASP
- Zed Attack Proxy
- Wapiti

Information

"Branching" is a very useful feature of version control. It is the ability to copy all project code as a new parallel project to allow testing or making changes to create an updated or new version of the application. Parts of the new code can later be ported over to the original project to be used in the original application as well.

Project management, collaboration, and issue tracking

As we have already learned, having a successful product requires keeping track of the entire process and sharing knowledge with the entire team, especially when the team is expanding. This is where the project management process becomes especially important.

Examples of tools used:

- Microsoft Teams for collaboration and communication.
- Scrum Trello for Agile Planning and Tracking.
- Jira to track specific issues and manage projects.
- MeisterTask for task management.
- Slack for collaboration and communication.
- Basecamp for managing projects and communicating with clients.
- Azure DevOps for Application Life Cycle Management (ALM)

Using Development Tools to Provide Different Solutions

Development teams rely on the tools we described earlier to produce a wide range of IT solutions, many of which we use today to build applications of various kinds, such as:

- Web applications
- Smartphone Applications
- General applications
- Embedded systems

Building a web application

A **web application** is an interactive program that is built using HTML, CSS, and JavaScript web technologies and which stores data on database servers. This application is used by users who perform tasks over the Internet.

Stages of building a web application

1. The ideation stage

Before creating a web application, we must set the goals and main idea of the application.

2. Market research

You must do what is called market analysis to find out:

- Whether the target consumer has a need for this product or service.
- Whether a similar product or service exists.

3. Define web application functionality

You must identify functions that provide solutions to the problems of the target market.

4. Wireframing/prototyping

Wireframing is about designing the layout of your web application, and prototyping takes the organization chart a step further by adding interactivity to test the functionality of the application.

5. Seek validation

At this stage, constructive opinions and feedback are collected from relevant parties and potential users regarding the design.

6. Architect and build database

The data needed by programmers and users, as well as the tool for building the required database, are determined at this stage.

There are many database design tools that are used for different purposes, but the nature of the program and how the software solution is proposed for deployment will determine the choice of a specific tool. Examples of tools used in designing and building databases are:

- MySQL
- SQL Server
- Amazon DynamoDB
- Azure SQL
- MongoDB
- Firebase

7. Building the front-end

The front-end is the visual element of your web application, and it represents the interface between the user and the system. This interface represents what the user encounters and interacts with.

Examples of tools used to build an optimized user interface for the web include:

- jQuery
- Reactjs
- Django
- Vue.js
- Angular

8. Building the back-end

The back-end is used to manage the data in the program. It refers to the databases and servers as well as all other parts that are not visible to the user within the web application.

Building the back-end includes writing the core code that provides the application's functionality, as well as preparing the database, the networks, and verifying the integration between the different subsystems. Security and performance are particularly important. Examples of tools used in building the back-end are:

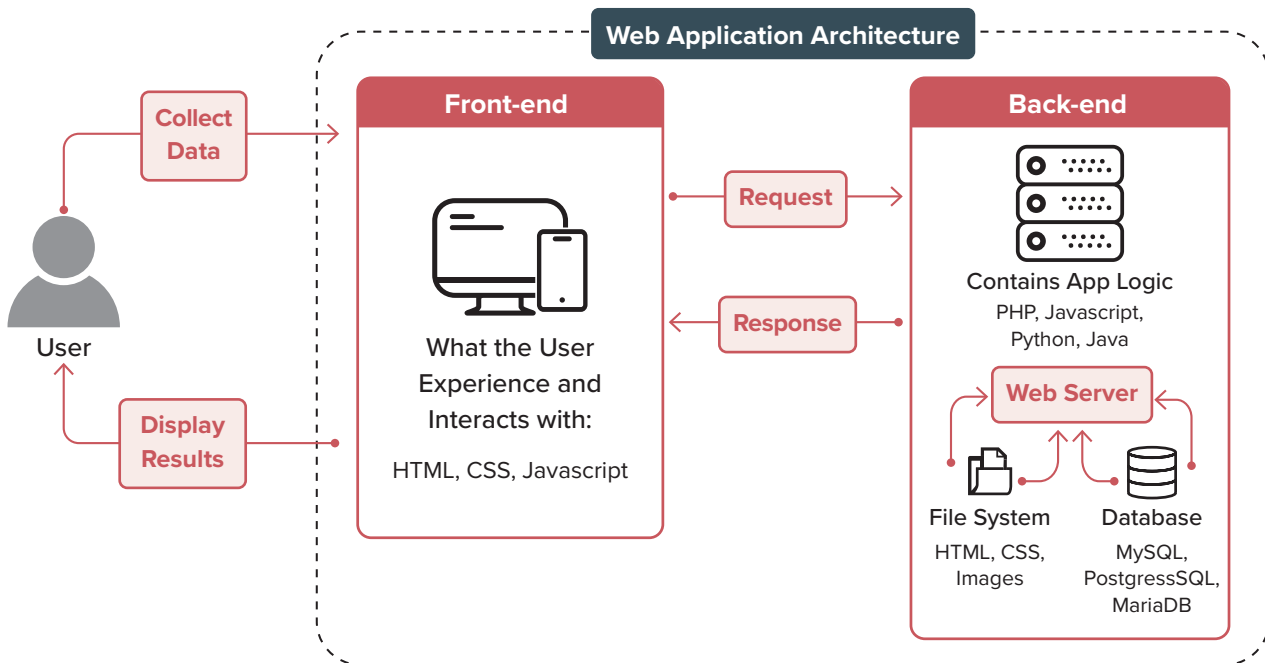
- Express JS
- ASP.NET
- Ruby on Rails
- Flask
- Laravel
- Spring Boot

9. Hosting your web application

To run your web application on a specific server, a web hosting provider is required. The hosting service may be simple and cheap, or it may be a large cloud computing service that allows your cloud infrastructure to grow as the number of application users grows and your needs grow.

Web hosting providers

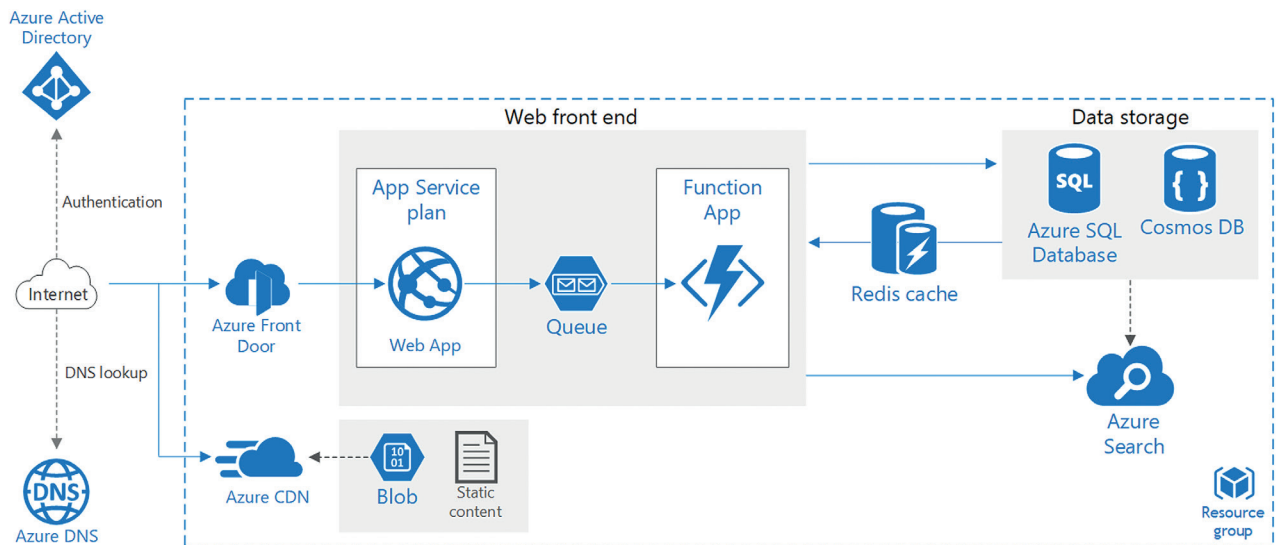
Types	Examples
Hosting Providers	<ul style="list-style-type: none"> • Bluehost • HostGator • GoDaddy • Rackspace
Cloud Service Providers	<ul style="list-style-type: none"> • IBM Cloud • Microsoft Azure • Amazon Web Services • Google Cloud Platform • Alibaba Cloud



The cloud-ready application architecture

It is preferable to develop and deploy cloud-based web applications as a set of cloud services. This process involves building data structures and then creating services, which are combined to form an integrated system.

The following diagram illustrates how to build a scalable and high-performance web application using Microsoft Azure services. The same concept applies to all cloud computing providers.



The most important points to consider when using cloud application architecture are:

- The design of the application as a set of services.
- The separation of data, security, and performance standards.
- The requirements for communication through networks between application components.
- The scalability of the design.
- System security must be a core part of the application and not something to be planned for later.
- The physical distance from users is the most important consideration when choosing data centers.

Building an application for smartphones

The steps for creating a mobile application are similar to those for a web application but with some special considerations. The mobile application is used on a phone, which typically has a small screen. As the name suggests, the user will use the application "on the go", which means it is important to consider the convenience of the interface. The user should be able to adjust the screen size and access important information in a clear and simple way. It is also important to note that the difference in devices leads to the need for responsive applications.

The two major mobile platforms are iOS and Android, each supporting a different but similar set of technologies. For example, iOS recommends Xcode and Swift for software development, while Android recommends Android Studio and Java.

These environments only allow building a final application that is ready to be published to the specific application store in that environment. However, there are environments that try to solve this problem by supporting application deployment to multiple stores.

With the following tools, a single application can be developed in a way that runs in different software environments:

- Xamarin
- React Native
- Ionic
- Kotlin

Testing mobile applications is a big challenge, and it is difficult for a programmer or even a software development company to have all the mobile devices available in the market to do the testing. This is why online services offer simulations for a wide range of mobile devices, allowing applications to be tested for compatibility across different devices.

Examples of tools used:

- Xamarin Test Cloud
- BrowserStack
- Firebase Test Lab

Building a general-purpose application

General purpose software is a type of application that can be used to perform many tasks, such as traditional office software, including word processors, graphic design tools, Enterprise Resource Planning (ERP) applications, or Customer Relationship Management (CRM) systems.

Despite the focus of new software development technologies on the web and mobile applications, these traditional applications still retain their importance. The development of such applications relies on ready-made and reusable code libraries, especially user interface components and reporting tools.

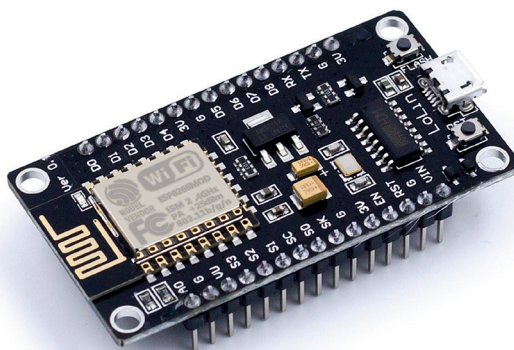
Building an embedded application

An **embedded system** is a special computer with a real-time operating system, often without a user interface. Software on the embedded system handles sensors, actuators, and mechanisms for wired or wireless data exchange. These programs must be reliable, secure, and fast. These applications require real-time operating systems such as RTLinux, Windows 10 IoT, and QNX as well as programming languages that are optimized for data processing, and network connectivity.

Examples of embedded systems are traffic lights, fire alarms, and home security systems.

Embedded systems can be programmed using the following programming languages:

- Assembly language (difficult and unsuitable for practical use).
- C, Embedded C, nesC, and Rust.
- Object-oriented languages such as C#, C++, and Java.



EXERCISES

1 Choose the correct answer.

1. What is the purpose of a code editor?
 - A. Compile programs into machine code.
 - B. Debug software issues.
 - C. Write and modify code.
 - D. Analyze database performance.
2. Which tool translates programs into executable machine language?
 - A. Debuggers
 - B. Code Editors
 - C. Profilers
 - D. Compilers and Linkers
3. Which tool is essential for managing code written by multiple programmers?
 - A. Project Builders
 - B. Network Analyzers
 - C. Code Management Tools
 - D. Database Explorer
4. Which of the following provides an integrated environment with a text editor, compiler, and debugger?
 - A. Code Management Tools
 - B. Integrated Development Environment (IDE)
 - C. Profilers
 - D. Project Builders
5. What is the primary function of a profiler?
 - A. To check for memory and processor usage during program execution.
 - B. To link different program files.
 - C. To manage the collaboration of multiple programmers.
 - D. To detect errors in a software program.

6. Which tool is necessary when writing software for web applications?

- A.** Debugger
- B.** Profiler
- C.** Network Analyzer
- D.** Project Builder

2 Identify the most important points to consider when using cloud application architecture.

3 Explain the meaning of general purpose software and provide some examples.

4 List the basic steps for building a web application.

5 In your notebook, match the following terms with their correct descriptions:

- 1.** Text editor
- 2.** Version control software
- 3.** IDE
- 4.** Front-end
- 5.** Back-end

-
- A.** A type of software used to modify text files.
 - B.** Manages your data, databases, servers, and all components that the user can't observe inside the web application.
 - C.** Enables previous versions of code files to be preserved for reference when problems occur.
 - D.** The visual elements of a web application; the interface between the user and the system.
 - E.** Contains all the software and tools needed to write and implement programs and to diagnose and fix problems.

PROJECT

Planning the Development of an Application

Imagine you need to create a mobile application named "Community Resources Application" aimed at helping elderly users access essential community services.

This application focuses on connecting seniors to local resources, like senior centers and community help lines, in a simple and accessible format.

1. Your job is to find details and images about these places to include in the application. After that, create a plan for building the application using the Software Development Life Cycle (SDLC). In your plan, explain the steps needed to make sure the application is developed in an organized way.
2. Finally, make a presentation to explain the key parts of the project. This includes the application's goals, who will use it, its features, and how it will make a difference in your community by supporting local projects and attractions.

WRAP UP

THIS UNIT COVERED HOW TO:

- > explain the stages of the Software Development Life Cycle (SDLC).
- > compare the advantages and challenges of Waterfall, RAD, and Agile methods.
- > describe different programming languages, their history, and uses.
- > explain how a computer processes programming languages and how errors are handled through compilers or interpreters.
- > identify software tools and their roles in each stage of software development.

KEY TERMS

- Agile Methodology
- Assembly Language
- Code Editor
- Compiler
- Development
- Embedded System
- Evaluation
- Executable Program
- Fourth-Generation Language
- General-Purpose Application
- High-Level Programming Language
- Integrated Development Environment (IDE)
- Interpreter
- Life Cycle
- Linker
- Machine Language
- Maintenance
- Mobile Application
- Rapid Application Development (RAD)
- Software Development Life Cycle (SDLC)
- Software Development Methodologies
- Software Development Tool
- Testing
- Web Application

Foundations of Programming

Software Engineering

Design, Develop, and Innovate in the Digital World

Imagine creating software that connects people, solves problems, and enhances lives. What if you could master the principles of software engineering and use them to design innovative applications? From understanding programming languages to building accessible apps, this course equips you with the tools to excel in the digital age.

Foundations of Programming: This course introduces you to essential principles, programming languages, and development tools. Explore the software prototyping process, focusing on user interaction and creating functional prototypes. Learn to design mobile applications using MIT App Inventor, adding elements, and programming features that bring your ideas to life. Delve into software accessibility and digital inclusion, mastering techniques to test, deploy, and ensure your applications meet the needs of all users.

By the end of this course, you'll have the skills to design, prototype, and develop software applications with confidence. Whether crafting interactive prototypes, programming mobile apps, or incorporating accessibility features, you'll be prepared to make a meaningful impact in the world of technology. Empowered with knowledge and hands-on experience, you'll be ready to innovate and shape the future of digital solutions.

