BEGINNER'S STEP-BY-STEP CODING COURSE

JavaScript

#

SCRATCH

*

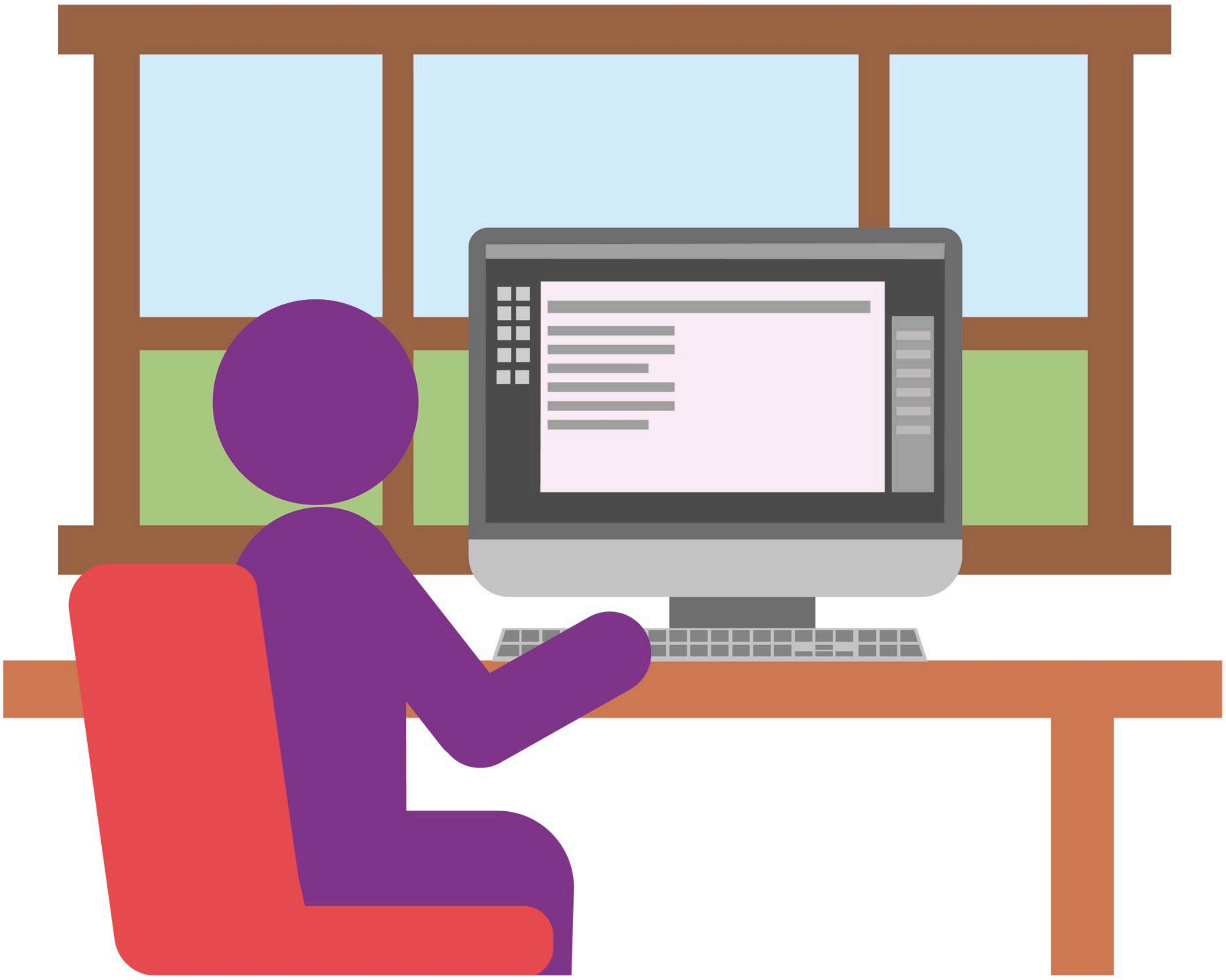
PYTHON

:

1011010101011
1111011110110
1010101111110
0010101011100
1010101010011
1111011110110
1010101111110

HTML

LEARN COMPUTER PROGRAMMING THE EASY WAY



BEGINNER'S STEP-BY-STEP CODING COURSE

DK

LEARN COMPUTER PROGRAMMING THE EASY WAY

DK DELHI

Senior editor	Suefa Lee
Project art editor	Sanjay Chauhan
Project editor	Tina Jindal
Art editors	Rabia Ahmad, Simar Dhamija, Sonakshi Singh
Assistant editor	Sonali Jindal
Jacket designer	Tanya Mehrotra
DTP designer	Jaypal Chauhan
Pre-production manager	Balwant Singh
Senior managing editor	Rohan Sinha
Managing art editor	Sudakshina Basu

DK LONDON

Project editor	Miezan van Zyl
US editor	Kayla Dugger
Project art editor	Francis Wong
Managing editor	Angeles Gavira Guerrero
Managing art editor	Michael Duffy
Associate publishing director	Liz Wheeler
Publishing director	Jonathan Metcalf
Art director	Karen Self
Jacket design development manager	Sophia MTT
Producer (pre-production)	Gillian Reid
Senior producer	Meskerem Berhane

First American Edition, 2020
Published in the United States by DK Publishing
1450 Broadway, Suite 801, New York, NY 10018

A catalog record for this book
is available from the Library of Congress.
ISBN: 978–1–4654–8221–1

CONTRIBUTORS

Clif Kussmaul is Principal Consultant at Green Mango Associates, LLC, where he designs and implements research projects, faculty development workshops, and classroom activities. Formerly, he taught for 20 years at college level and worked full and part time in software development and consulting. Craig was a Fulbright Specialist at Ashesi University and a Fulbright-Nehru Scholar at the University of Kerala. He has received multiple grants from the US National Science Foundation, Google, and other sources to support his work with Process Oriented Guided Inquiry Learning (POGIL), Free and Open Source Software (FOSS), and other topics in computer science education.

Sean McManus writes and cowrites inspiring coding books, including *Mission Python, Scratch Programming in Easy Steps, Cool Scratch Projects in Easy Steps,* and *Raspberry Pi For Dummies*.

Craig Steele is a specialist in computer science education who helps people develop digital skills in a fun and creative environment. He runs Digital Skills Education and is a founder of CoderDojo in Scotland, which runs free coding clubs for young people. Craig has run digital workshops with the Raspberry Pi Foundation, Glasgow Science Centre, Glasgow School of Art, and the BBC micro:bit project.

Dr. Claire Quigley studied Computing Science at Glasgow University, where she obtained a BSc and PhD. She has worked in the Computer Laboratory at Cambridge University and at Glasgow Science Centre. She is currently STEM Coordinator with Glasgow Life, and lectures part time at the Royal Conservatoire of Scotland, working with BEd Music students. Claire has been involved in running CoderDojo Scotland since its initial session in 2012.

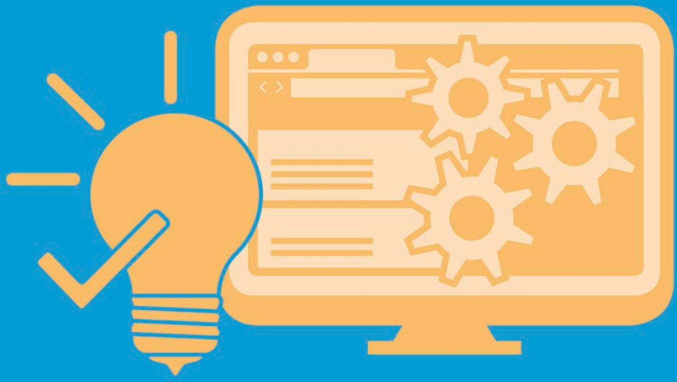
Dr. Tammy Pirmann is a computer science professor at the College of Computing and Informatics at Drexel University in Philadelphia, Pennsylvania. She is an award-winning educator, recognized for her focus on equity in computer science education and for promoting guided inquiry in secondary computing education. She was the co-chair of the Computer Science Teachers Association's Standards Committee and an advisor on the K12 CS Framework.

Dr. Martin Goodfellow is a Lecturer in the Computer and Information Sciences department at the University of Strathclyde. He has also developed educational computer science content and workshops for other organizations worldwide, including Google, Oracle, CoderDojo Scotland, Glasgow Life, Makeblock, and the BBC.

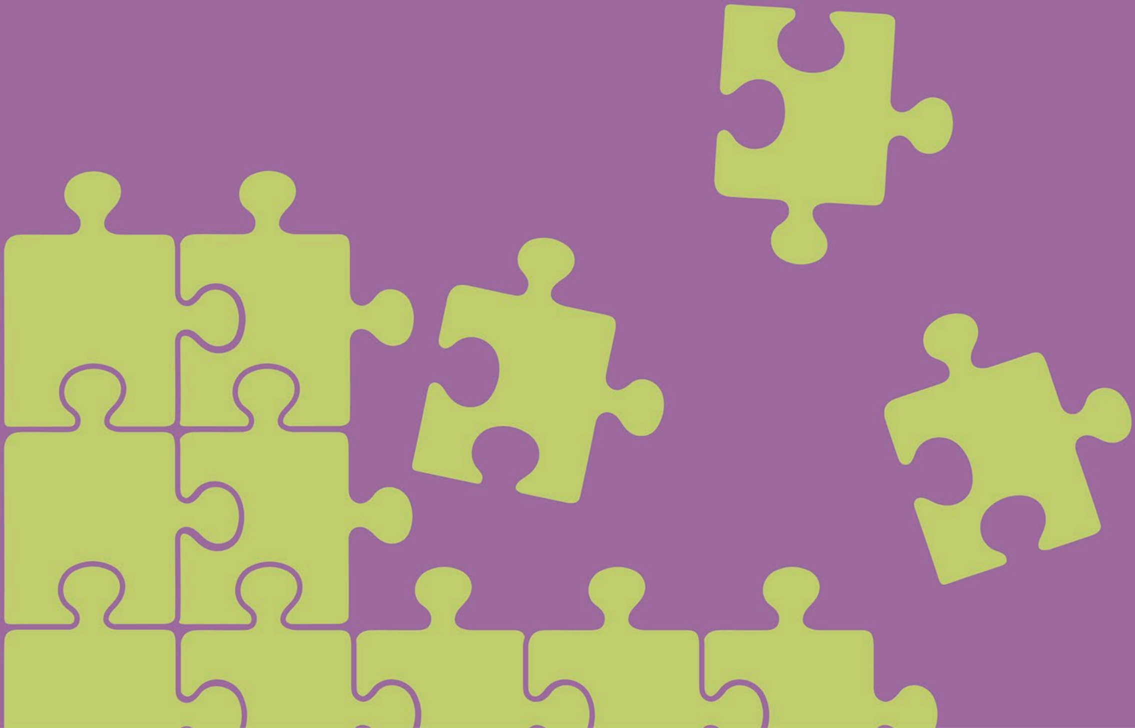
Jonathan Hogg is an audiovisual artist who has spent the last decade constructing works out of combinations of software, electronics, sound, light, wood, plastic, and metal. He often works with young people, running creative and technical workshops. Prior to art, Jonathan designed and developed software in the London finance industry. He began his career researching and teaching Computing at the University of Glasgow. The only constant in all of this has been Python.

David Krowitz learned to program in the early 1980s on a Commodore VIC-20 plugged into a portable black-and-white TV set. He has been studying and practicing computer programming ever since. Nowadays, Dave spends his time building microservice architecture for businesses while expounding his love for object-oriented design patterns and software architecture. See dotpusher.com for more info.

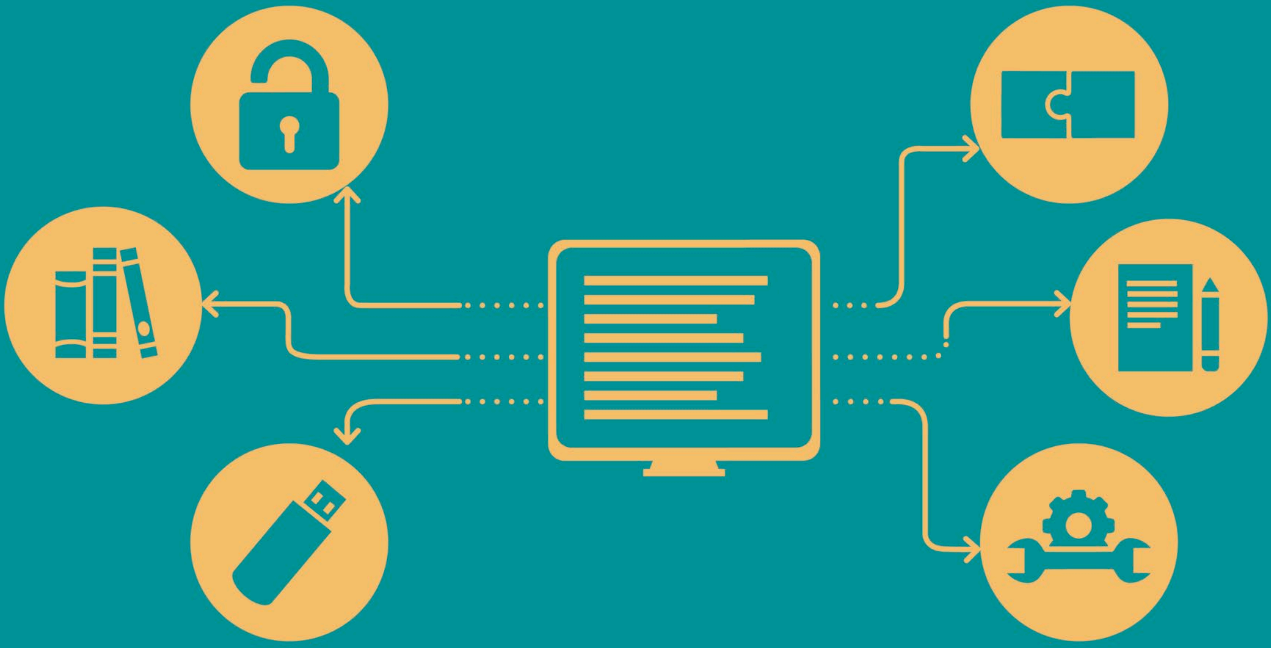
CONTENTS



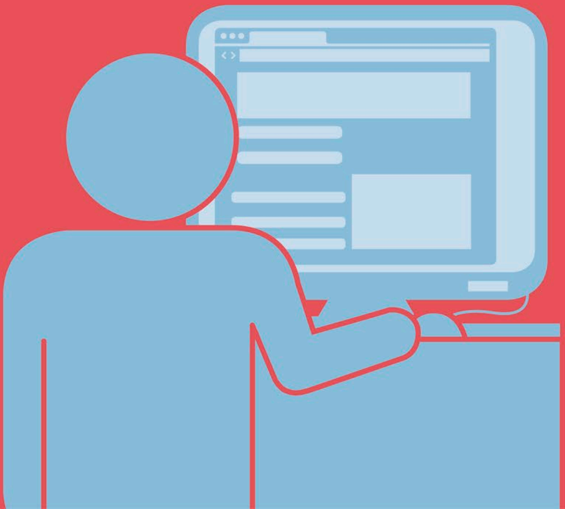
SCRATCH



PYTHON



WEB TECHNOLOGIES



88.0

Foreword

If you've ever asked a teenager for help with your computer, you probably have felt the crushing weight of self-doubt as you realize you understood less than half of what they just said. That same "helpful teen" would most likely scoff at the idea of a book—made out of honest-to-goodness, old-fashioned paper—on the subject of learning to code. "Just Google it. There are loads of tutorials on YouTube!" they might say.

But not everyone is high-bandwidth, multiscreen ready. Plus, when you are carefully stepping through the creation of your first lines of code, a physical page with your fingertip planted firmly on the next step can act as a valuable lifeline to the tangible world.

If you're reading this as a teenager yourself, congratulations on discovering life beyond YouTube! You're about to find out that the creators of this guide are exactly the kind of industry-defining professionals whose content channels, blogs, and social media posts you'd end up on if you did decide to Google "learning to code."

As a lifelong gamer and computing enthusiast, I've been reporting on technology for almost a quarter of a century. In that time, I've witnessed seismic changes in the way we interact with the world. AI, big data, automation, e-commerce—all now intrinsic parts of our daily routines, even if we aren't always aware of them.

Technology is no longer a niche topic. In fact, today, every industry could be considered a tech industry, which leads to a simple choice: get with it or get left behind.

Starting at the absolute beginning, this guide will introduce the jargon and tools you'll need to get programming in the most popular and versatile software languages. The pages are also peppered with interesting facts about coding and careers, together with step-by-step projects to get you going. Even if you decide not to become the next Mark Zuckerberg, the skills you'll learn will be a great asset when talking to technology professionals and will also help develop your own logic and problem-solving abilities.

It was an ancient Greek philosopher who first noted the irony "the only constant in life is change," and this has never been more true than in the world of computing. Maybe you're looking for a different career or want to learn a new skill to support a hobby or passion project. Or perhaps you just want to be able to talk to your tech-obsessed teenager in a language that will impress them!

For those curious about coding, this guide is full of straightforward information in easily digestible bites, written by some of the leading educators and experts in their field. There is jargon, but it's jargon you'll understand as you get to it. Is learning about coding essential? No. Will it help you understand and feel more comfortable in the world we now live in? I think so. Could it lead to a new and amazing career direction? Definitely, if that's what you want.

There is still a desperate shortage of technology professionals in the workforce. Opportunities exist, but they are not going to come looking for you unless you speak at least a bit of their language.

[illegible]

Kate Russell
Technology reporter,
author, and gamer

About this book

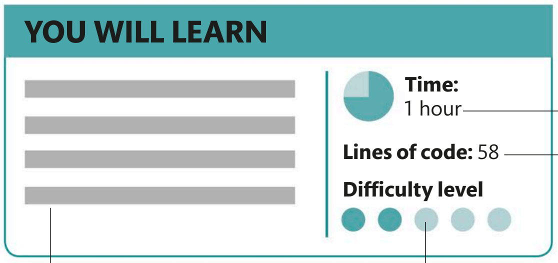
How this book works

Divided into three chapters, this book teaches the fundamentals of five programming languages: Scratch, Python, HTML, CSS, and JavaScript; the last three are grouped under Web Technologies. The book defines the basic concepts of each programming language and builds on them with the help of detailed projects that you can try on your own.

Concepts
Each chapter contains the basic programming concepts of the language. These are explained with the help of practical code examples that you can try out to understand the concept better.

Illustrations help you understand and learn concepts

Projects
The projects in this book teach you how to create games, planners, apps, and websites. Each project starts with a brief overview of what you will learn in the project, how to plan the project, and what you will need to create it. Simple step-by-step instructions guide you through the project and explain every aspect of the code, with the help of detailed annotations.



Indicates the estimated time it will take to create a project

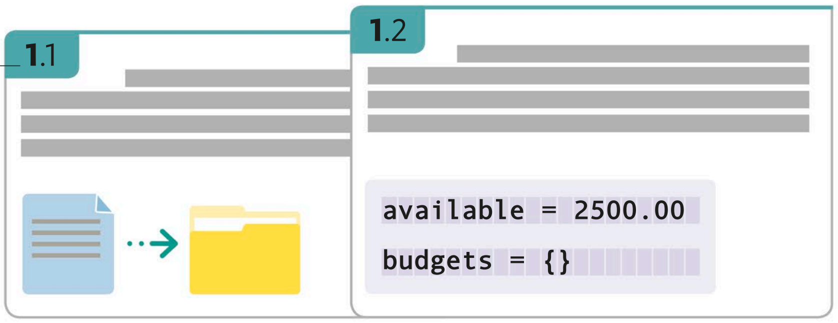
Indicates the estimated lines of code in a project. This may vary depending on the code editor being used

YOU WILL LEARN

This box highlights the concepts being used in a project

Indicates the difficulty level of a project, with one being the easiest

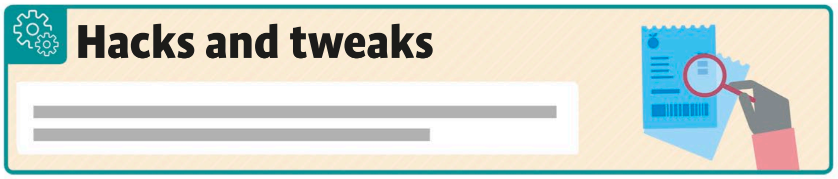
Projects are broken down into smaller sections with clear steps to make learning easier



```
available = 2500.00
budgets = {}
```

STEP-BY-STEP

Hacks and tweaks
The “Hacks and tweaks” section at the end of each project provides tips on how to tweak existing bits of code, or add new functionalities to it.



Coding elements in the book

Icons, color-coded windows with grids, and flowcharts that explain the program structure help you work your way through the projects.

Icons

The "Save" icon will remind you to save the program at a particular point in the project. The "HTML," "CSS," and "JS" icons indicate which web file you need to write the code in.

SAVE HTML CSS JS

Python code windows

Python uses two different windows—the shell window and the editor window—for writing code. To differentiate between the two, this book uses different colors. This will help you know which window you should type the code in.

```
>>> input = 2
>>> score = input * 3
>>> print(score)
6
```

SHELL WINDOW

These chevrons appear only in the shell window. Type in the code at the >>> prompt

Each block of the grid represents a single space in the code

```
def reset_game():
    global score, charms
    score = 0
    charms = 0
```

EDITOR WINDOW

Every indent (spaces at the start of a line) equals four empty grid blocks. All subsequent indents will be in multiples of four

Web languages code window

The code for all the web languages is written in green-colored windows in this book. A special visual element, a turnover arrow, is used to indicate code being split over two lines. This element is not part of the actual code and has only been introduced in the book to help explain the flow of code in a block.

```
...<ul id="topMenu" class="navbar-nav mr-auto">
        <li class="nav-item">
            <a class="nav-link" href=
            "index.html">Home</a>
        </li>
```

CODE WINDOW FOR WEB LANGUAGES

In this book, ellipses are used at the start of a line of code to indicate an extended indent, usually more than eight grid blocks

Gray code indicates an existing line of code in the program. It is used to identify the line below or above which the new code must be added

The placement of the arrow indicates if a space needs to be added before it. In instances where there will be no space, no empty grid blocks are left between the arrow and the code

DK website for code

The resource pack for the projects in this book (except the "Hacks and tweaks" sections and the projects created in Scratch) have been hosted on ***www.dk.com/coding-course***. This includes code in its original format (.py, .html, .css, .js) and images for all the games and websites.

DK

www.dk.com/coding-course

Go to this URL to download the Coding Course Resource Pack

INTRODUCTION

What is programming?

Computers and electronic devices need software (or programs) to tell them what to do. Programming, or coding, is the art of writing these instructions. Though some people are professional programmers, coding can also be a hobby.

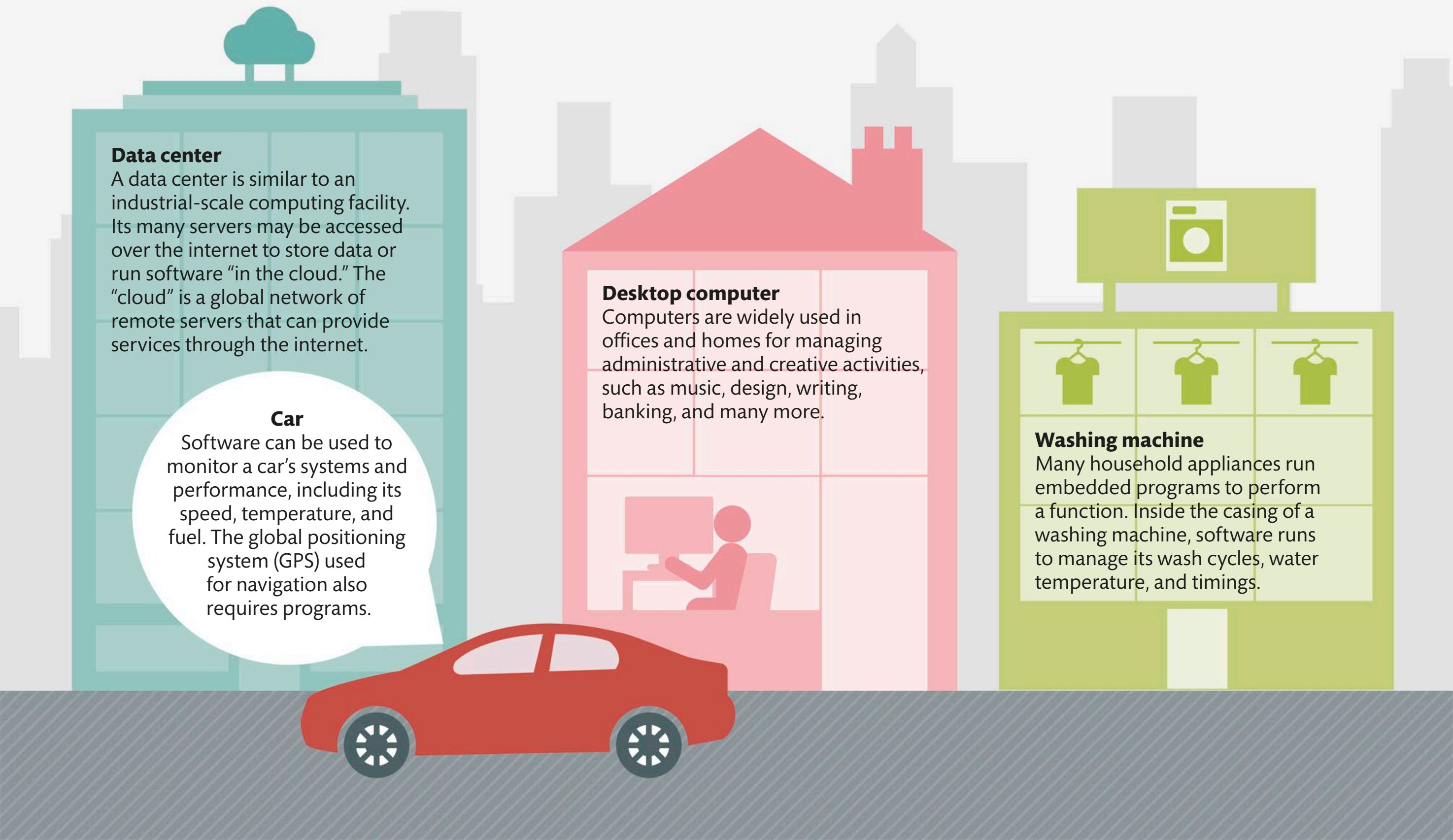
Computer programs are everywhere

Programming is not just about conventional computer systems anymore. The world has become increasingly digital, and almost everything runs on software. Programs are now incorporated into devices such as cell phones and tablets, labor-saving equipment around the home, and even in transportation systems.

BECOMING A CODER

After learning the basics of programming, these tips can be used to develop coding skills further.

- **Practice:** Write and experiment with code.
- **Read code:** A lot can be learned by studying other people's programs.
- **Learn multiple languages:** Learning the different ideas and concepts of other languages can help programmers choose the most suitable language for each project.
- **Publish projects:** Putting work online and getting feedback on it from other coders helps you write better code.



Data center
A data center is similar to an industrial-scale computing facility. Its many servers may be accessed over the internet to store data or run software "in the cloud." The "cloud" is a global network of remote servers that can provide services through the internet.

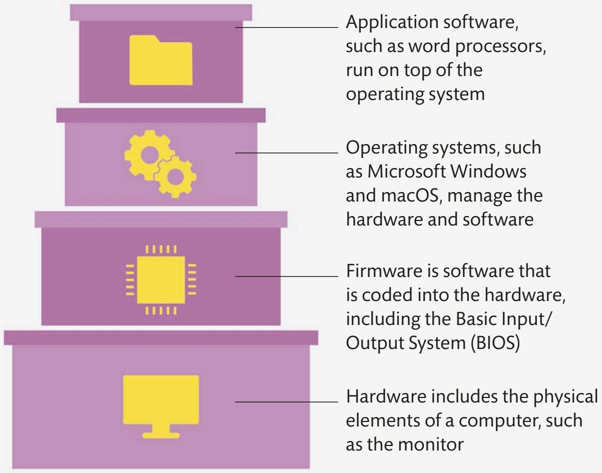
Car
Software can be used to monitor a car's systems and performance, including its speed, temperature, and fuel. The global positioning system (GPS) used for navigation also requires programs.

Desktop computer
Computers are widely used in offices and homes for managing administrative and creative activities, such as music, design, writing, banking, and many more.

Washing machine
Many household appliances run embedded programs to perform a function. Inside the casing of a washing machine, software runs to manage its wash cycles, water temperature, and timings.

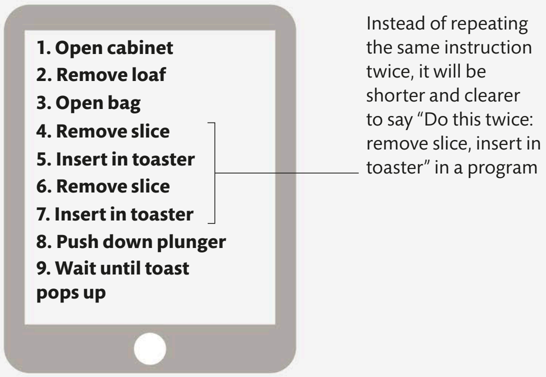
What is a computer program?

A program is a set of instructions that a computer follows to perform a task. Programs can be extremely complex, and there can be several different layers of programs working together. Microsoft Windows, for example, is made up of millions of lines of instructions.


Application software, such as word processors, run on top of the operating system
Operating systems, such as Microsoft Windows and macOS, manage the hardware and software
Firmware is software that is coded into the hardware, including the Basic Input/Output System (BIOS)
Hardware includes the physical elements of a computer, such as the monitor

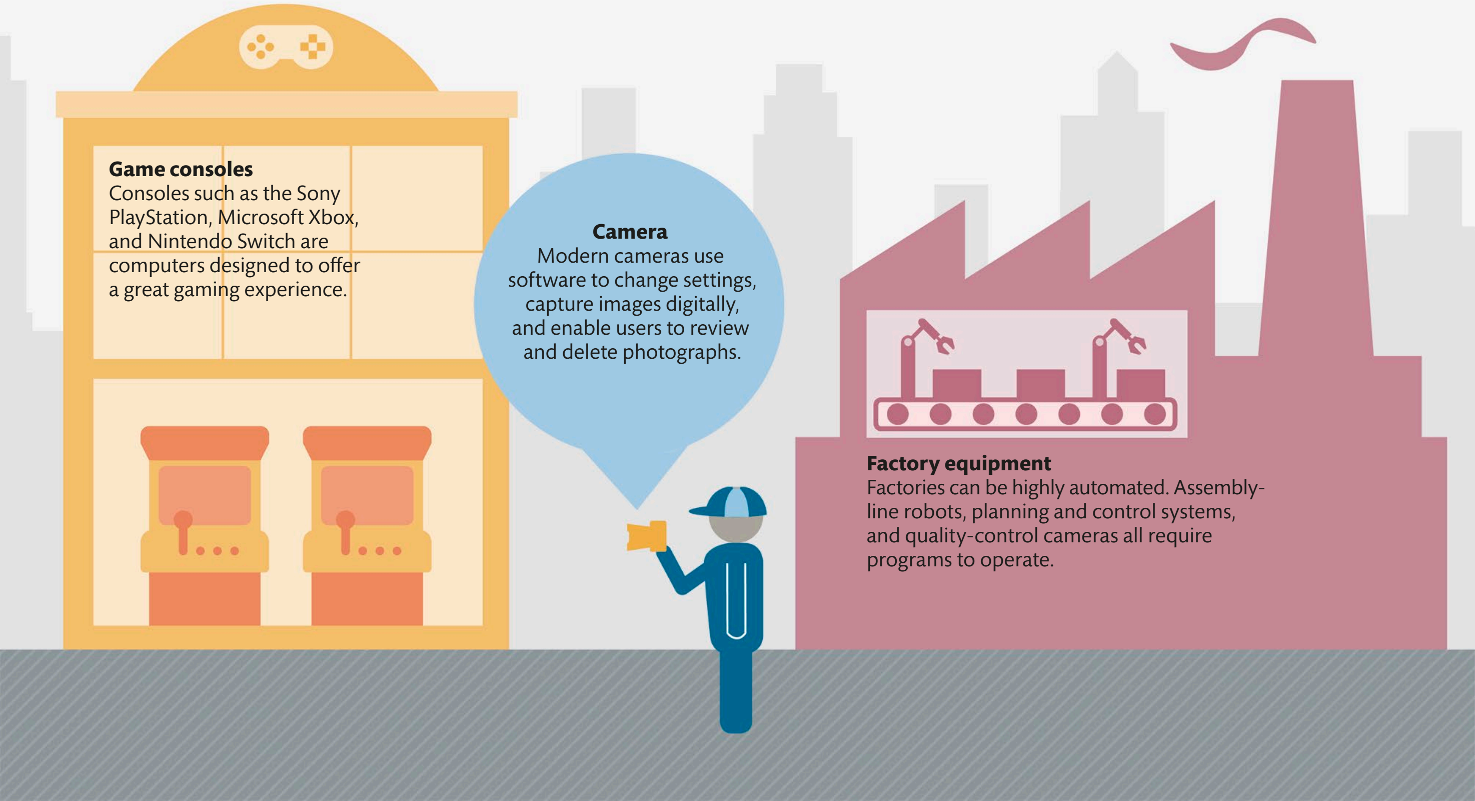
Thinking like a computer

To write a program, it is necessary to understand how a computer processes instructions. This means that tasks need to be broken down into smaller chunks so that the computer can understand the instructions. For example, a robot cannot simply be asked to "make some toast." It is necessary to program precise and detailed instructions for each step.



1. Open cabinet
2. Remove loaf
3. Open bag
4. Remove slice
5. Insert in toaster
6. Remove slice
7. Insert in toaster
8. Push down plunger
9. Wait until toast pops up

Instead of repeating the same instruction twice, it will be shorter and clearer to say "Do this twice: remove slice, insert in toaster" in a program

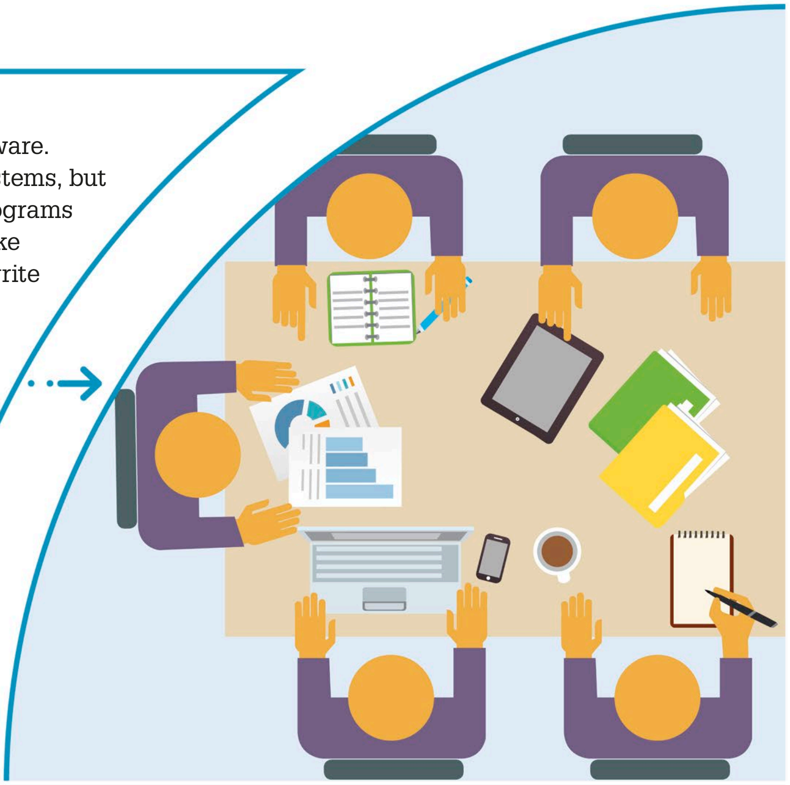


Game consoles
Consoles such as the Sony PlayStation, Microsoft Xbox, and Nintendo Switch are computers designed to offer a great gaming experience.

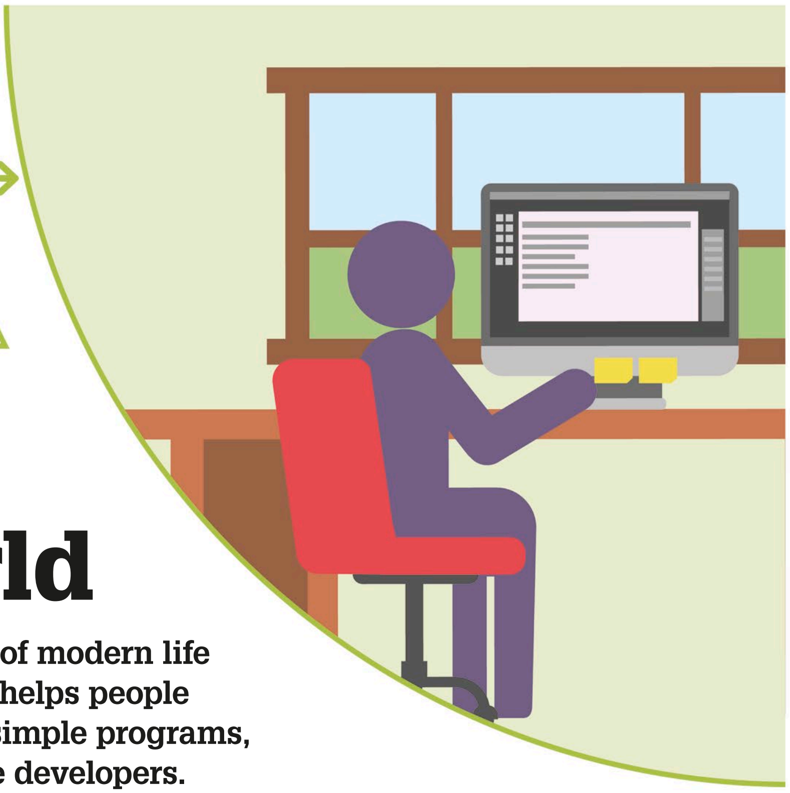
Camera
Modern cameras use software to change settings, capture images digitally, and enable users to review and delete photographs.

Factory equipment
Factories can be highly automated. Assembly-line robots, planning and control systems, and quality-control cameras all require programs to operate.

Managers and office workers
Many businesses create and use specialized software. Software engineers develop complex software systems, but managers and office workers often write short programs to solve problems or automate tasks that might take hours or days by hand. For example, they might write code to query databases, format information, analyze data, control equipment, customize websites, or add features to word processors and spreadsheets. Some programming languages are specifically designed for these purposes (see pp.344–347).

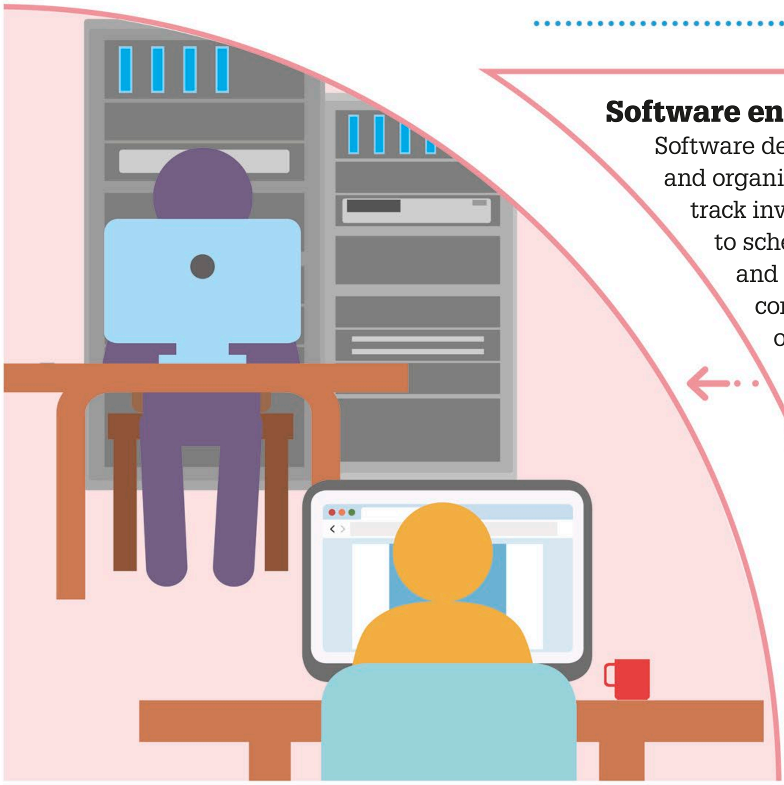


Artists and hobbyists
Coding can support many forms of creativity. Artists can create software to make music or visual art that changes as people interact with it. Hobbyists might create software for games and interactive stories, to direct simple robots, to control lighting, or to do tasks around the house.



Coding in the real world

Coding is used in nearly every aspect of modern life and work. Basic knowledge of coding helps people use software more effectively, create simple programs, and communicate with other software developers.



Software engineers and web developers

Software developers work for many different businesses and organizations. Businesses require software to track inventory and place orders with suppliers, to schedule employees and track work hours, and to send mailings to customers. Insurance companies use software to set pricing based on the number and cost of events and to review and approve policies. Websites often combine existing programs with custom coding for special features. Software engineers play key roles in developing systems that suit a client's needs.

Scientists and researchers

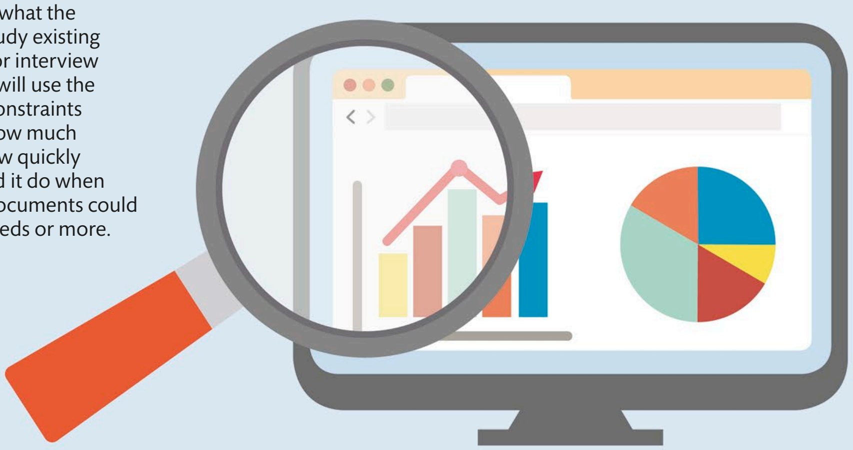
Code can also be used to create experiments, analyze data, and create medical reports. For example, brain scientists might use software to display shapes or words to a patient, to record brain activity, and to analyze the data to learn what parts of the brain are most active.

25,000,000
WORKERS ACROSS THE WORLD **ARE SOFTWARE DEVELOPERS**

Working as a software developer

Writing a simple program might seem straightforward, but it can be surprisingly difficult. Developing large, reliable software systems is even more complex and requires teams of software developers with a variety of specialized skills and roles.

Analyze
In this phase, developers decide what the software must do. They might study existing systems, design new processes, or interview people to understand how they will use the system. This also defines other constraints or requirements. For example, how much data must the system handle, how quickly must it respond, and what should it do when problems occur? The resulting documents could range from a few pages to hundreds or more.

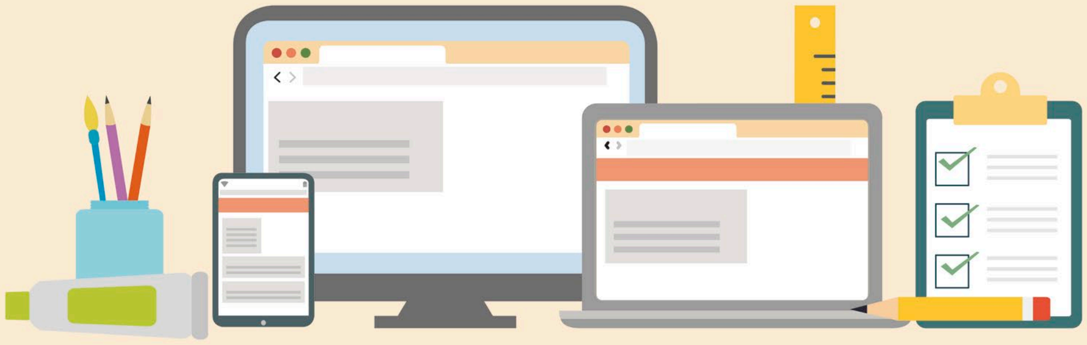


Test
Developers check whether the software works correctly and fix any problems they find during the testing phase. This is often the longest and least predictable phase and a common reason for delays and extra costs. There are many types of tests—unit tests check if individual functions are correct, functional tests check individual components, integration tests check if components work together, and system tests check the entire system.

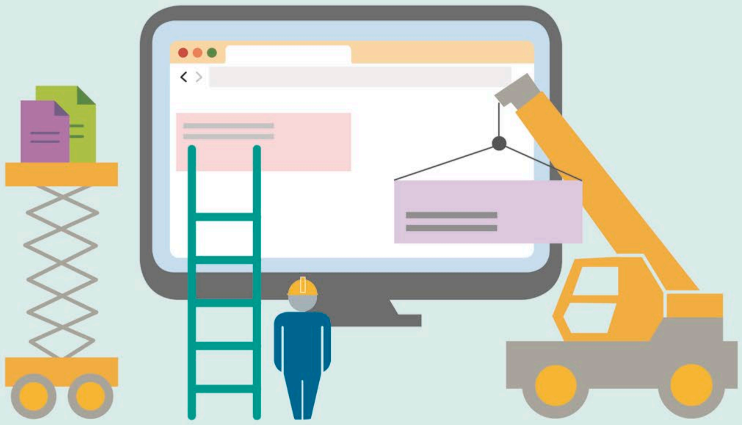
Overview

Software development involves four phases: analyze, design and plan, build, and test. These phases, however, can be structured in a variety of ways. A waterfall model steps through each phase once, which seems simple, but often leads to problems. An iterative model cycles through the phases several times, building part of the system in each cycle. An agile model cycles through each phase many times, adding different features in each cycle.

Design and plan
In this phase, developers decide how the software will work and how it will be created. This can include deciding on which language to use, sketching user interfaces, designing databases, subdividing it into pieces, and specifying the files and even the individual functions to be created. Developers also need to estimate the time, effort, materials, and cost to create the system and put together a schedule for who will do which tasks at what time.



Build
In the build phase, developers create the software, including user interfaces, databases, code, and documentation for users and programmers. This means that coding is just one part of one phase of software development, and in some ways the easiest and most predictable. As each piece is built, developers might inspect or review the code to see how well it is written and then integrate it into the larger system.

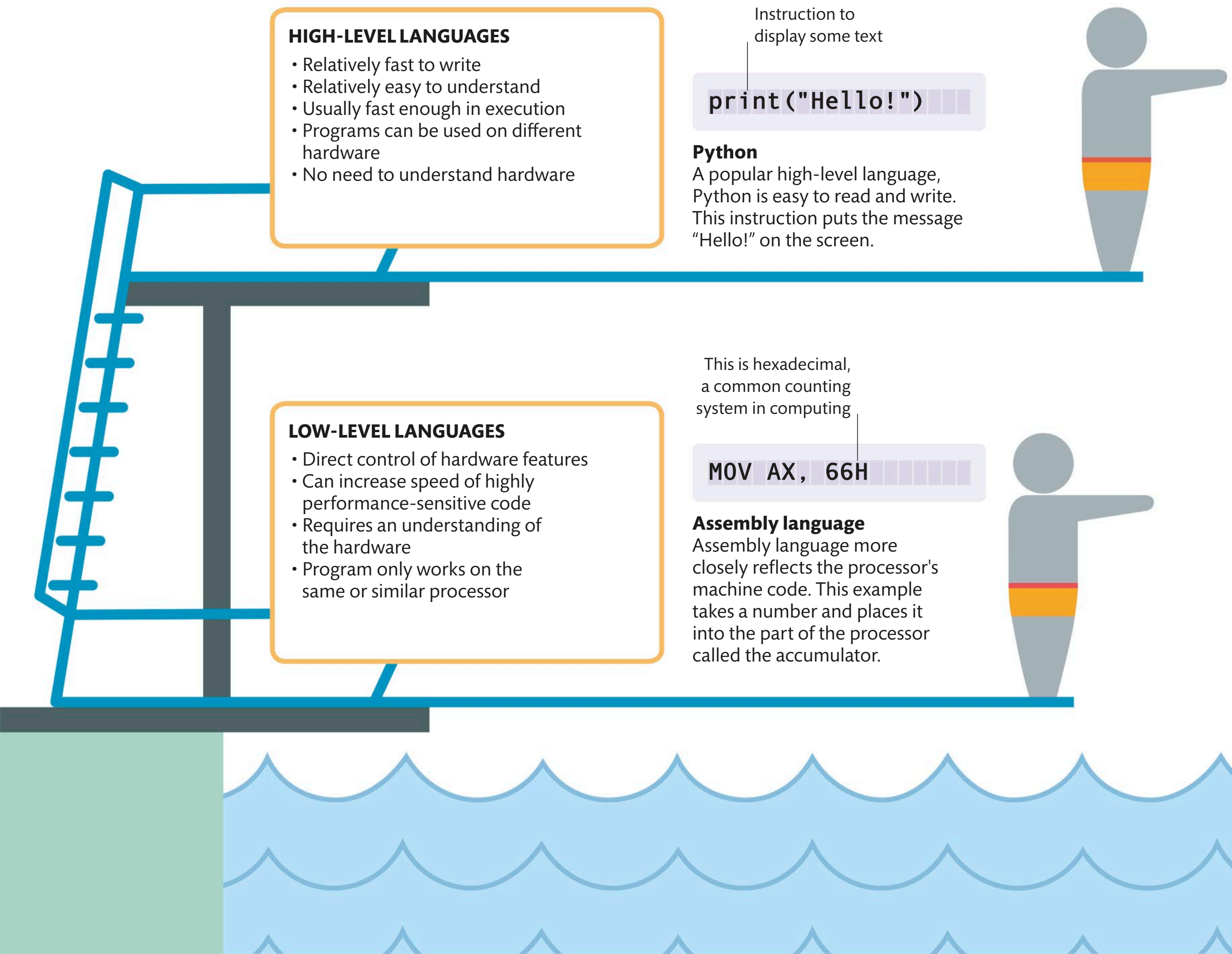


Programming languages

A programming language is a set of words and symbols that allows people to write instructions for a computer. There is sometimes a compromise between how easy the language is to use and how powerful it is.

High- and low-level languages

High-level programming languages are designed to be easy to use without needing a detailed understanding of the computer hardware. They often use words that are similar to human language and manage some aspects of the computer automatically. Often, the same program can run on different hardware if it is written in a high-level language. By contrast, low-level languages give programmers granular control over the computer but also require a deeper understanding of how it works. Programs written in a low-level language might not work on other hardware.



HIGH-LEVEL LANGUAGES

- Relatively fast to write
- Relatively easy to understand
- Usually fast enough in execution
- Programs can be used on different hardware
- No need to understand hardware

Instruction to display some text

```
print("Hello!")
```

Python
A popular high-level language, Python is easy to read and write. This instruction puts the message "Hello!" on the screen.

LOW-LEVEL LANGUAGES

- Direct control of hardware features
- Can increase speed of highly performance-sensitive code
- Requires an understanding of the hardware
- Program only works on the same or similar processor

This is hexadecimal, a common counting system in computing

```
MOV AX, 66H
```

Assembly language
Assembly language more closely reflects the processor's machine code. This example takes a number and places it into the part of the processor called the accumulator.

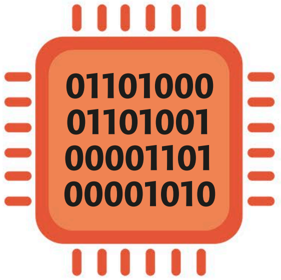
Machine code

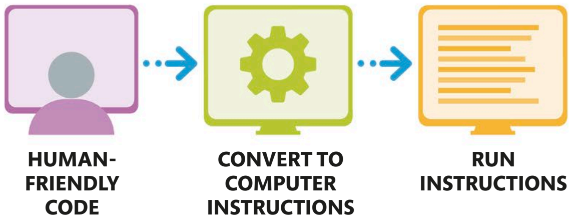
Low-level code that represents how the computer hardware and processor understand instructions is called machine code. It is a collection of binary digits—1s and 0s—that the processor reads and interprets. Machine code instructions are comprised of an opcode and one or more operands. The opcode tells the computer what to do and the operand tells the computer what data to use.

How the computer understands a programming language

Ultimately, all programs end up as machine code. Most programs are written in more human-friendly languages and need to be translated into raw bits so that they can be executed by a processor. An interpreter translates and executes the instructions as the program is running, while compilers translate the program in one go before it runs.

Microprocessor
The microprocessor is the "brain" of a computer and controls most of the machine's operations. It carries out commands and runs the machine code instructions.





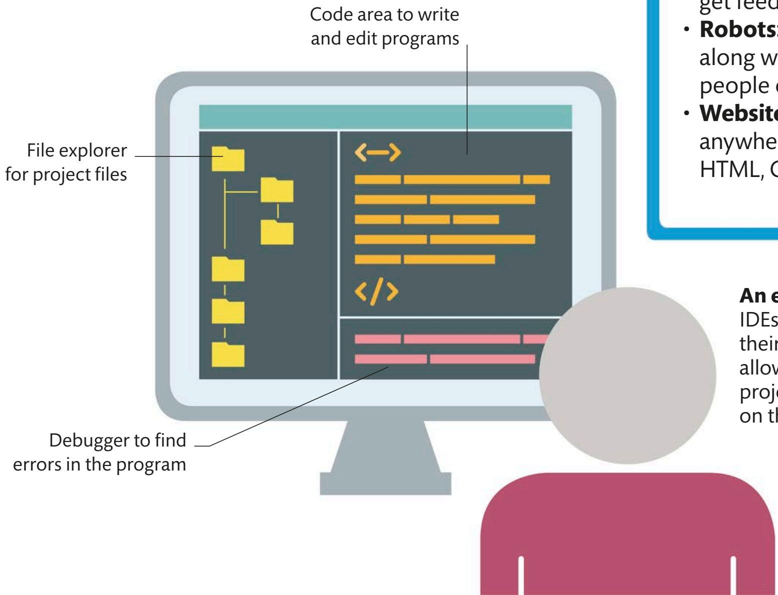
Using an integrated development environment (IDE)

An IDE is a set of tools that helps programmers. It has a code editor for writing programs and may include productivity features, such as autocomplete for instructions and color coding to help readability. Some IDEs also include a debugger to help find errors and a compiler or interpreter to test and run the programs.

APPLICATIONS

Once you have learned how to program, these skills can be used for a wide range of creatively fulfilling and useful projects.

- **Home automation:** To control things such as lights or curtains remotely.
- **Games:** A great way to experiment with coding, games are easy to share and to get feedback on (see pp.80–91, 178–203).
- **Robots:** Using Arduino or Raspberry Pi boards along with kits or electronic components, people can program their own robots.
- **Websites and web apps:** Programs that can run anywhere in a web browser can be created using HTML, CSS, and JavaScript (see pp.210–343).


Code area to write and edit programs
File explorer for project files
Debugger to find errors in the program

An example IDE layout
IDEs sometimes enable users to configure their setup. Here is one configuration that allows the programmer to browse the project files on the left, code and edit on the right, and debug at the bottom.

Types of programming languages

Many different philosophies or paradigms have been used to design programming languages over the years. Because they are not mutually exclusive, programming languages often embody several core ideas. They can also be used in different ways depending on the programmer's preferred approach. Python, for example, can be used for both object-oriented and procedural programming. JavaScript can be used for event-driven and object-oriented programming. The best approach or the best programming language to use often depends on the programmer's preference. Below are some of the ways that programming languages can be defined and classified.

IMPERATIVE PROGRAMMING

These languages require a list of instructions for the computer to carry out. The programmer has to first work out how the task can be completed, then provide step-by-step instructions to the computer. Imperative languages are common and include Python (see pp.94–95), C, C++, and Java.

```
user = input("What's your name? ")
print("Hello", user)
```

INPUT IN PYTHON

Python program to greet a user by name

```
What's your name? Sean
Hello Sean
```

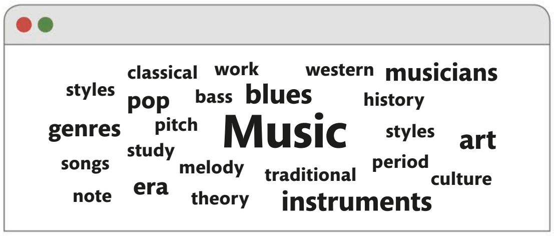
OUTPUT IN PYTHON

DECLARATIVE PROGRAMMING

In declarative programming, programmers tell the computer what result they want without needing to say how it will be achieved. In the Wolfram Language, for example, a word cloud based on the words in Wikipedia's Music page can be created using a single line. Other declarative languages include SQL, which is used for databases.

```
WordCloud[WikipediaData["music"]]
```

INPUT IN WOLFRAM



OUTPUT IN WOLFRAM

EVENT-DRIVEN PROGRAMMING

The event-driven programming concept is one where the program listens for certain things to happen and then starts the appropriate program sequence when they do. For example, a program might react to user actions, sensor input, or messages from other computer systems. JavaScript (see pp.264–265) and Scratch (see pp.28–29), among others, can be used to write event-driven programs.

Creates a web page button

```
<input type="button" value="Click me!" ↵
onClick="showMessage();">
```

Runs the `showMessage()` JavaScript instructions when the button is clicked

This icon has been used in the book to indicate code being split into two lines

CHOOSING A LANGUAGE

Sometimes, programmers' choice of language may be dictated by the hardware they are using, the team they are programming with, or the kind of application they want to create. Often, they will have a choice. Here are some popular languages that can be considered.

PYTHON
A flexible language, it emphasizes ease of understanding in the code.

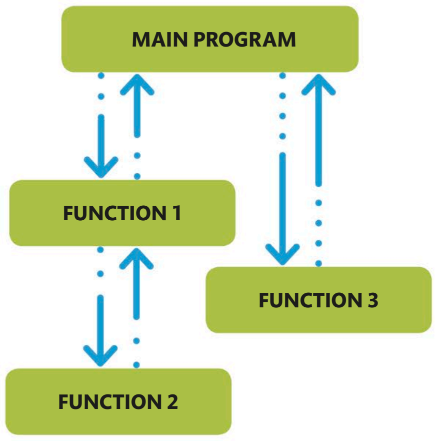
SCRATCH
A great first programming language, Scratch is perfect for simple games.

JAVA
Widely used in financial services, small devices, and Android phones.

JAVASCRIPT
The language used by web pages for interactivity.

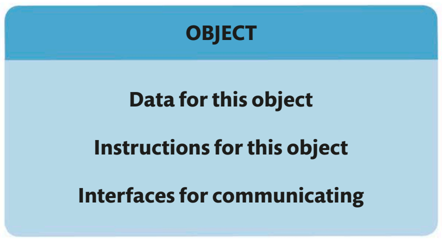
PROCEDURAL PROGRAMMING

This type of programming is based on functions, which contain reusable chunks of programs. Functions can start other functions at any time and can even start themselves again. They make programs easier to develop, test, and manage. Many popular programming languages, such as Java and Python (see pp.94–95), support procedural programming.



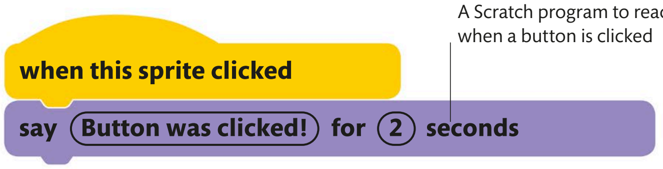
OBJECT-ORIENTED PROGRAMMING

In object-oriented programming, the idea is that data and the instructions related to it are stored together in "objects." Objects can interact with each other to achieve the program's objectives. The aim is to make code more modular so it is easier to manage and is more reusable. Many popular programming languages, such as C++, JavaScript, and Python, support object-oriented programming.



VISUAL PROGRAMMING LANGUAGES

These languages make it easier to develop software using drag-and-drop interfaces so a programmer can create software more quickly and with fewer errors. Visual Basic, for example, includes tools to design user interfaces visually. Scratch (see pp.28–29) is another highly visual language, often used to learn programming.



```
when this sprite clicked
say (Button was clicked!) for (2) seconds
```

A Scratch program to react when a button is clicked