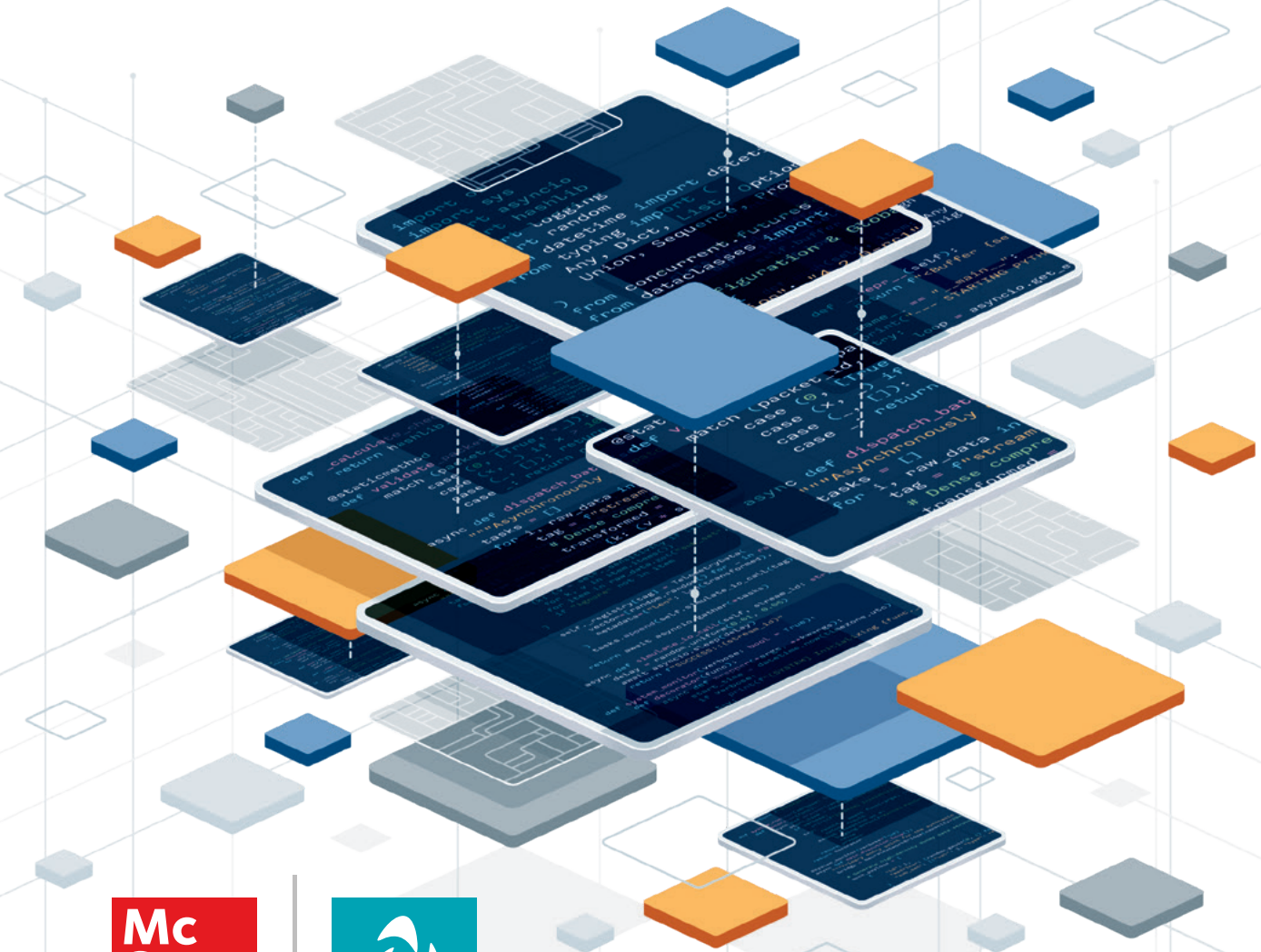
Introduction to

Computer Programming

Software Development in the Age of AI



Mc Graw Hill

binarylogic

Introduction to

Computer Programming

Software Development in the Age of AI

Introduction to Computer Programming
Software Development in the Age of AI

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

 MHID: 1-2669-2838-3 ISBN: 978-1-2669-2838-3

Printed in the United States of America.

1 2 3 4 5 6 7 8 9 LWI 31 30 29 28 27 26

mheducation.com **binarylogic.net**

Contents

Preparing Students for **Software Development in the Age of AI:**

From Python Fundamentals to Career-Ready Skills

Software development is changing rapidly, with AI now supporting every stage of the process, from planning and design to coding, testing, debugging, and deployment. Students therefore need to learn not only programming fundamentals, but also how to work effectively and responsibly with AI tools. This textbook uses Python as a practical entry point into software development because of its widespread use in data science, research, robotics, automation, and AI applications.

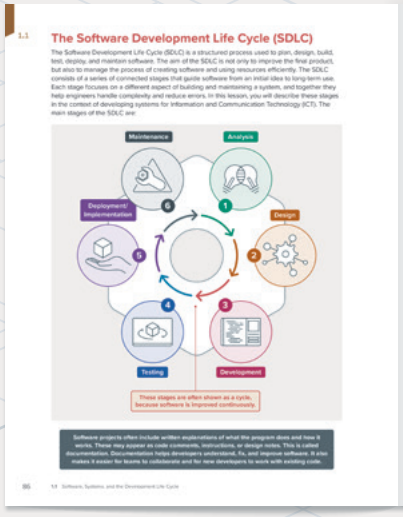
Throughout the course, students build strong foundations in programming, computational thinking, problem-solving, and software engineering while exploring how AI is reshaping the developer's role. Designed for CTE pathways, the material connects classroom learning with real-world technical skills and future career opportunities in software development, AI, data, automation, and other fast-growing computing fields.

Key Features

A structured introduction to programming fundamentals and software development concepts, designed by expert educators.

Step-by-step Guidance

Each unit guides students step by step from core building blocks to more advanced concepts, establishing a solid foundation in programming and software design.



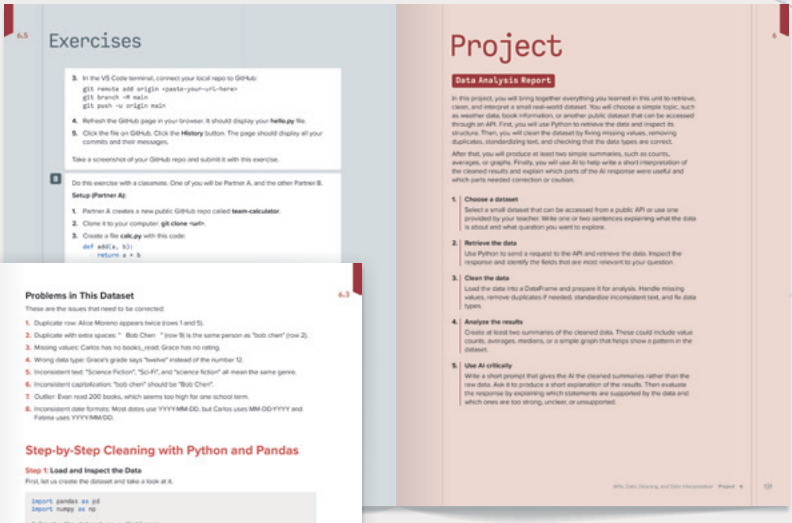
Career-ready Skill Development

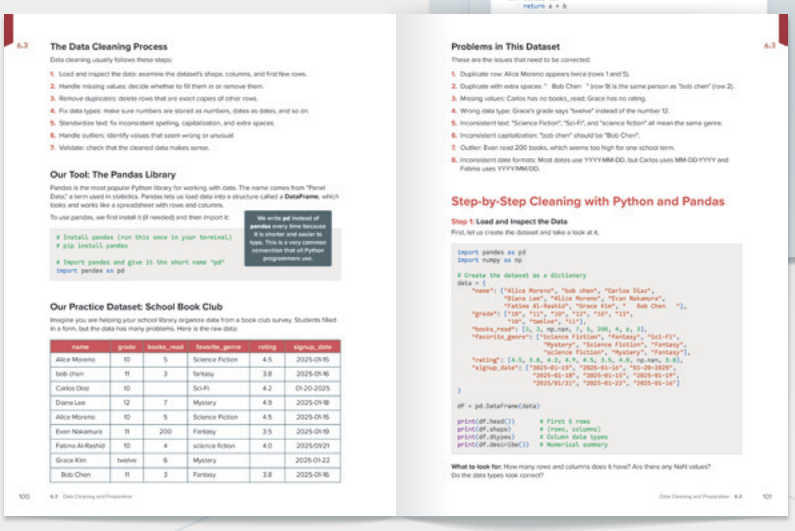
Students learn to think like developers: planning, writing, reviewing, and iterating on code as part of a structured development workflow.

Curriculum aligns with the latest industry standards, preparing students for real-world coding roles and future careers.

Hands-on Practice

Every unit includes tasks and projects that develop essential programming competencies through practical, real-world scenarios.





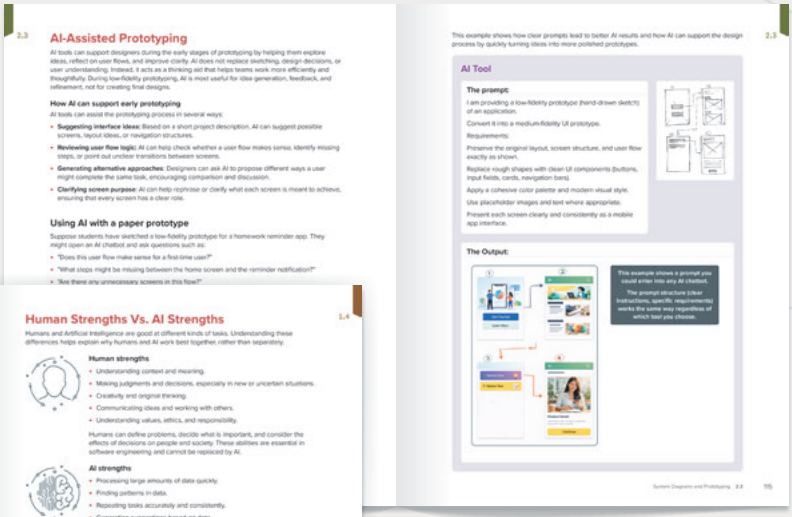
Problem-solving Skill Development

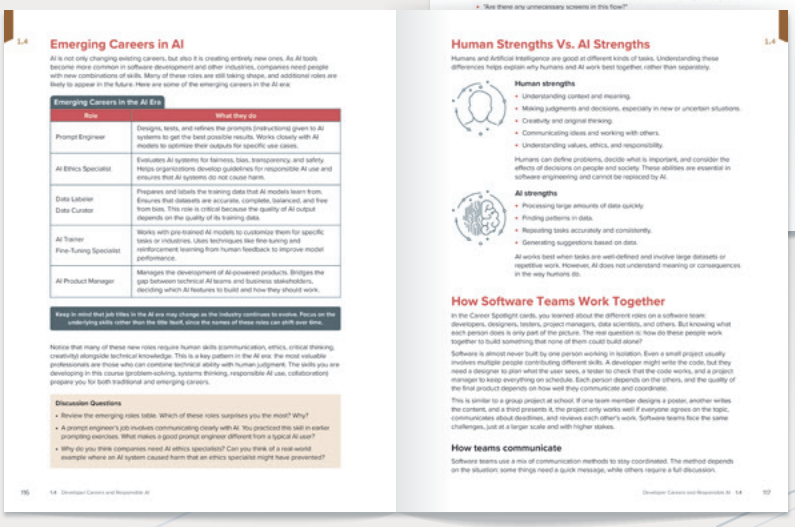
Computational thinking is woven throughout: students learn to break down problems systematically before writing a single line of code.

Clear, measurable goals guide students from writing their first program to understanding how AI tools fit into every phase of development.

AI-infused Processes

Students discover how AI technologies support the full development pipeline, from requirements and design to writing and testing code.





An Ethical Approach

Each unit addresses the ethical dimensions of AI in software development, including code ownership, over-reliance, and responsible use, equipping students for today's technology-driven workplace.

Planning and Prototyping

2

2

Introduction

This unit introduces the planning and design phase of software development, focusing on how to think through and improve digital systems before any code is written. You will learn how to define problems, write requirements, and use diagrams to understand how a system works. You will also create wireframes and user flows to plan how users interact with an app, and evaluate whether designs are clear, usable, and well structured.

The unit also introduces accessibility, helping you design interfaces that a wider range of people can use, and explores how AI tools can support the creation of higher-fidelity prototypes without replacing your own design decisions.

Learning Objectives

In this unit, you will:

- Analyze a software problem and define the scope of a project.
- Explain how the project scope can be translated into specific tasks.
- Break a project into smaller, manageable tasks that guide development.
- Identify and document system requirements through user stories and acceptance criteria.
- Create basic system diagrams that represent workflows and interactions.
- Design a low-fidelity prototype to illustrate how users interact with a system.
- Plan simple test cases to verify whether system requirements are met.
- Identify potential technical and ethical risks during software development.
- Design a simple UX prototype using wireframes, user flows, UX improvements, and accessibility notes.
- Use AI tools responsibly to support planning and prototyping tasks.

LESSON 1

Defining the Problem and Project Scope

Before you begin designing or building software, it is important to clearly understand what problem the software is meant to solve. Many software projects fail not because of poor coding, but because the problem was not defined clearly from the beginning.

It is also important to consider the people who will use the software. Different users have different needs, experiences, and levels of access to technology. A solution that works well for one group of users may not work effectively for another. By considering diverse users during the design process, developers can create software that is more usable, inclusive, and effective.

Clearly defining the project scope is one way to support this process. The scope of a project helps software engineers decide what problem they are solving, what features will be included, and what is outside the boundaries of the project. This lesson focuses on understanding project scope and how it guides the planning of a software system.

What Is a Project?

In software development, a **project** is a planned effort to create or improve a software product within a given time frame and with defined resources. Projects are considered unique because they are carried out to produce a specific product, service, or result.

Projects have a defined start and end date, and they are typically constrained by a budget, a scope, and a timeline. They often involve multiple stakeholders, resources, and complex interactions between them. Project management is the discipline of planning, organizing, and managing resources to achieve project goals and objectives.

Examples of software projects are developing a new application, adding major features to an existing system, or redesigning a software platform. Projects can be small or large, simple or complex, and can be found in almost every industry and field of work.

Project characteristics

- A fixed timeline that specifies the start and end date of the project
- A defined scope of work and clear objectives
- Independent, with a set budget and resources
- Involves a series of sequential and interrelated activities
- Faces risks and challenges that must be guarded against

Once we understand what a project is and what makes it unique, the next step is planning how the project will be carried out. This is where project planning becomes essential.

2.1

Project planning

Project planning is the process of organizing and preparing a project before the work begins. It defines the project's goals, scope, tasks, and the steps needed to complete it successfully within a certain time.

A good project plan includes the resources required, the budget, and the timeline for completing different activities. It also explains the roles and responsibilities of team members and how they will communicate during the project.

Project planning also helps teams identify possible risks or problems in advance and develop ways to manage them. With careful planning, team members can work more efficiently, stay organized, and successfully achieve the project goals.

1. **Defining the project scope:** Defining the requirements of the project, the desired outcomes, and the stakeholders involved.

2. **Developing a project plan:** Creating a detailed project plan that outlines the tasks, resources, and timelines necessary to complete the project.

3. **Identifying project risks:** Identifying potential risks that may arise during the project and developing strategies to mitigate them.

4. **Defining project roles and responsibilities:** Gathering the team members involved in the project and assigning their roles and responsibilities.

5. **Setting project milestones:** Defining specific points in the project where progress can be measured and evaluated.

6. **Monitoring and controlling the project:** Tracking project progress, identifying issues that arise, and taking corrective action as necessary.

These steps do not always happen in a strict order, but together they help teams stay organized and focused.

Discussion Questions

- Think about a project you have done at school (for example, a science fair or group presentation). What was the final goal of the project?
- Why do projects need a clear start and end date?

Project planning is important for several reasons, including:

- **Clarity of objectives**
  Project planning clearly defines the project's objectives, including the expected outcomes, timelines, and budgets. This clarity ensures that all stakeholders are on the same page and can work together toward a common goal.
- **Assigning resources**
  Project planning helps to identify the resources required to complete the project, including personnel, equipment, and materials. This information allows for resources to be assigned effectively, which can help to minimize costs and improve efficiency.
- **Risk management**
  Project planning helps to identify potential risks and develop strategies to mitigate them. By addressing potential issues before they arise, project managers can reduce the likelihood of delays, cost overruns, and other project failures.

The lack of a project management plan leads to lost time and poor performance. In order to avoid such situations, the project must be organized and managed in the most effective and efficient manner. Effective project planning is essential for ensuring that a project is completed on time, within budget, and to the satisfaction of stakeholders. It helps to minimize risks and ensure that project goals are achieved.

Project Scope Planning

Project scope planning is the process of defining the goals, tasks, and deliverables of a project so that everyone understands what work will be done and what is included in the project. Two major elements of scope planning are determining the scope of the project and breaking the project into smaller parts (decomposition).

Determining the scope of the project

It is important to determine the main tasks that must be carried out to implement the project. For example, if the project involves hiring new staff members, the individual steps that need to be implemented should be determined, that is:

- Advertise the role
- Select candidates
- Interview the candidates
- Submit a job offer

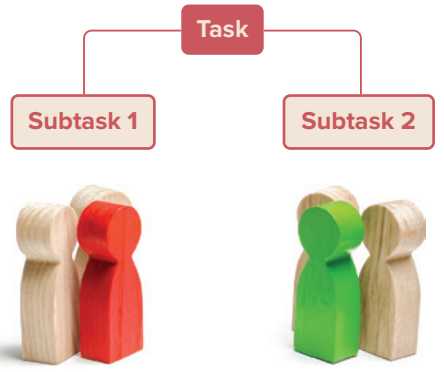
Breaking the project into smaller tasks

Teamwork involves dividing the larger team into smaller groups, with each group responsible for completing its assigned subtask.

In software projects, this might include tasks such as designing screens, writing requirements, testing features, or collecting user feedback.

For example, the engineering design process for building a house includes the following tasks:

- Creating a draft of the architecture
- Preparing construction documents



From Scope to Action: Organizing the Work

Software projects involve many tasks that must be completed in the right order and by the right people. Even when a team understands what needs to be built, the project can fail if the work is not organized properly. Teams must decide who will complete each task, when the tasks should be finished, and how different tasks depend on one another.

To manage this complexity, professional software teams use planning tools that help organize work, track progress, and coordinate team members. In particular, three tools that are often considered essential in software projects are: task breakdowns, simple timelines, and project boards.

Task Breakdowns: From big goals to small steps

A task breakdown takes a large project goal and divides it into specific, actionable tasks. Each task should be small enough that one person can finish it in a reasonable amount of time (usually a few hours to a few days, not weeks). The characteristics of these actionable tasks are described below.

Characteristic	What it means	Example
Specific	It is clear exactly what needs to be done. Anyone reading the task knows what the outcome looks like.	"Sketch the login screen wireframe" (not "work on design").
Assignable	One person (or a specific pair) can take ownership of it.	"John will sketch the login screen wireframe."
Sized appropriately	It can be completed in a few hours to a few days, not weeks.	"Sketch the login screen" is sized appropriately. "Design the entire app" is too big.
Testable	It is clear when the task is complete. There is a clear finish line.	"The task is complete when the wireframe is sketched, shows all required fields, and has been reviewed by a colleague."

Example: Assignment reminder app

Suppose your software development team is building a homework assignment reminder app where students can add assignments, set due dates, and receive reminders. The project scope can be broken into smaller tasks.

Feature 1: Add assignments

- Sketch the "add assignment" screen (Designer, 2 hours)
- Create the input form for subject, description, and due date (Developer, 4 hours)
- Save assignment data to the database (Developer, 3 hours)
- Test the form with valid and invalid inputs (Tester, 2 hours)

Feature 2: View assignments

- Design the assignment list screen (Designer, 2 hours)
- Program the screen that displays saved assignments (Developer, 3 hours)
- Test that assignments appear correctly (Tester, 2 hours)

Feature 3: Receive reminders

- Design the notification layout (Designer, 1 hour)
- Program the reminder logic (Developer, 5 hours)
- Test reminder timing (Tester, 3 hours)

Each feature is divided into clear tasks with a responsible team member. This makes it easier to track progress and identify problems early.

Discussion Questions

- Review the task breakdown example above. The longest single task is "Program the reminder logic," at 5 hours. If the developer gets stuck, how would the team know? What should the developer do?
- Why is it important that each task has a specific owner? What happens in group projects when a task belongs to "everyone"?

Simple Timelines: Putting tasks in order

Once you have a list of tasks, the next question is: What order should they happen in? Some tasks can happen at the same time (the designer can sketch screens while the developer sets up the database). Other tasks depend on each other (the tester cannot test a feature until the developer has built it). A timeline organizes tasks in the order they need to happen and shows when each one should start and finish.

Example: Two-week timeline for the assignment reminder app

Below is a simple chart showing the planned timeline for all the tasks in the project.

	Week 1					Week 2				
Task	**Mon**	**Tue**	**Wed**	**Thu**	**Fri**	**Mon**	**Tue**	**Wed**	**Thu**	**Fri**
Sketch all screens	design	design								
Set up the database	development									
Code: add assignment		development	development	development						
Code: view assignments				development	development					
Test features 1 and 2					testing	testing				
Code: reminders						development	development	development		
Test reminders								testing	testing	
Fix bugs and polish									bug fixing	bug fixing
Final review and demo										review

design development testing bug fixing review

The timeline illustrates some important features that should be considered when designing timelines for team projects.

Parallel work: The designer sketches the screens on Monday and Tuesday while the developer sets up the database on Monday. These tasks do not depend on each other, so they can happen at the same time.

Dependencies: The developer cannot start coding the "add assignment" feature until the designer has finished sketching that screen (Tuesday). The tester cannot test features 1 and 2 until the developer has finished building them (Friday of week 1). These are called dependencies: tasks that must be completed before another task can begin.

Buffer time: The last two days include bug fixing and a final review. Experienced teams know that something always takes longer than expected, so they build in extra time at the end. This is called buffer time or a contingency.

Discussion Questions

- In the timeline above, what would happen if the designer was sick on Monday and Tuesday of week 1? Which other tasks would be delayed? Which could still continue?
- Why is testing placed immediately after development in the timeline, not at the very end after everything is built? What is the advantage of testing each feature as it is completed?
- The timeline includes a two-day buffer. Some students might say: "That is wasted time; we should use it to add more features." Do you agree or disagree? Why?

Project Boards: Seeing the big picture at a glance

A task breakdown tells you what needs to be done. A timeline tells you when. A project board tells you the current status of every task. Project boards give the whole team a shared view of where the project stands: what is finished, what is being worked on, and what has not started yet.

The most common type of project board is a Kanban board. It uses three columns: To Do, In Progress, and Done. Tasks are written on cards that move from left to right as work progresses. When a task starts, its card moves from "To Do" to "In Progress." When it is finished, the card moves to "Done."

Example: Two-week timeline for the assignment reminder app

For smaller projects, a simple table or chart showing all the tasks can be used, but for more complex tasks we can use dedicated project management software to create a project board.

To Do	In Progress	Done
Code: reminder logic Owner: Dev	Code: view assignments Owner: Dev	Sketch all screens Owner: Designer
Design: reminder notification Owner: Designer	Test: assignment feature Owner: Tester	Set up the database Owner: Dev
Test: reminders Owner: Tester		Code: add assignment Owner: Dev

By looking at the project board, team members can quickly see that three tasks are waiting to begin, two tasks are actively being worked on, and three tasks are complete. The project manager can quickly see if too many tasks are in "In Progress" (which might mean people are starting things without finishing them) or if "To Do" is empty (which might mean the team needs to prepare for the next phase of the project).

Putting it all together: From scope to organized plan

The three tools we have studied above can be used sequentially during a project and complement each other to ensure the project has clearly defined tasks and a clear schedule, and that the project stays on track from start to finish.

Tool	What you create
Task breakdown	A list of all tasks with owners and time estimates
Timeline	A schedule showing task order, dependencies, and deadlines
Project board	A live view of task status that is updated throughout the project

2.1

Using AI in Project Planning

The project planning and definition techniques we have described so far have been developed over the last few decades to help developers solve problems and plan complex projects effectively and efficiently. Recently, new AI tools have appeared that help to improve this process even further, in particular the process of problem definition.

AI tools can be used to ask questions about a problem, suggest alternative ways to describe it, or help identify missing details. For example, when a problem statement is too broad or vague, AI can help narrow it down by asking who the users are, what their main needs are, and what constraints exist.

How AI can help define a problem

AI can support problem definition in several ways:

- **Clarifying vague ideas:** If a problem statement is unclear, AI can help rephrase it in simpler or more specific terms.
- **Identifying missing information:** When used correctly, AI can point out details that may be missing, such as user types, environments, or limitations.
- **Suggesting different perspectives:** AI can help students think about the problem from the user's point of view, not just the developer's.
- **Improving wording and structure:** AI can help turn a general idea into a well-structured problem statement with clear goals.

AI is most useful during the early thinking stage of a project, when ideas are still being shaped.

Suppose that a team of students wants to create a software solution, and they start with this problem statement: "We want to build an app for students."

This statement is very general. It does not clearly explain what the app is for, what problem it solves, or how it helps users.

Using an AI chatbot, the students can type their problem statement and ask for help making it clearer. For example, they might enter: "Help me improve this problem statement: We want to build an app for students. Ask me questions to understand what I mean." The AI might respond with questions such as:

- What kind of students are we focusing on?
- What difficulty are they facing?
- In what situation will they use the app?

After thinking about these questions and reviewing the AI's suggestions, the students might refine their problem statement to: "We want to build a mobile app that helps high school students who have difficulty managing their time keep track of homework deadlines and receive reminders."

In this improved version, the problem is clearer. It identifies the users, the main problem, and the purpose of the software. The AI helped guide the thinking process, but the students made the final decisions. This shows how AI can support problem definition without replacing human judgment.

Human responsibility remains essential

Although AI tools can support thinking and improve clarity, humans remain fully responsible for defining the problem a software project aims to solve. AI does not understand real-world situations, user emotions, or the consequences of decisions in the way people do.

When using AI to refine problem statements, students and software engineers must decide:

- which AI suggestions are helpful
- which ideas should be changed or rejected
- whether the problem being defined is realistic and appropriate

AI can generate many possible versions of a problem statement, but it cannot decide which one best reflects real user needs or ethical considerations. Only humans can make those judgments.

Human responsibility also includes thinking about the impact of a problem statement. A poorly defined problem can lead to software that is confusing, unfair, or even harmful. For example, if a problem ignores certain users or oversimplifies an important issue, the solution may not work well for everyone.

Using AI responsibly means treating it as a partner, not an authority. Developers should question AI responses, reflect on whether they make sense, and revise them based on their own understanding and values. Clear thinking, careful evaluation, and accountability remain human responsibilities at every stage of the project.

In short, AI can help improve ideas, but people are responsible for decisions, outcomes, and consequences. This responsibility is a key part of software engineering and cannot be automated.

Always think of AI as a helper, not a decision-maker.

Discussion Questions

- Why might an unclear problem statement lead to a poor software solution?
- How can AI help students think about a problem from different perspectives?
- Why is it still important for humans to review and improve AI suggestions?

> **Example: Reviewing an AI suggestion**
>
> A student asks an AI assistant for help defining a problem for a study app. The AI suggests: "Build an app that tracks student performance and predicts grades."
>
> The student realizes this could require private data and might lead to unfair predictions that have negative consequences for some students. Instead, the student revises the idea to: "Build an app that helps students organize assignments and set study reminders." It is always important for humans to review AI suggestions carefully to ensure they don't have negative consequences.

2.1

Using AI Tools Responsibly

As you work through this book, you will use AI tools as partners to help you learn and practice software development. AI chatbots are text-based tools where you type a question or instruction (called a prompt) and the AI generates a response. These tools can help you explore ideas, receive feedback, and improve your work as you develop software projects.

A variety of AI chatbots are available, including options from companies such as OpenAI, Google, Anthropic, and others. Your teacher may recommend a specific tool for classroom use. If you are working independently, most general-purpose AI chatbots will work for the activities in this book.

How to access a chatbot:

1. Open the chatbot in a web browser (most chatbots offer a free version that works in any browser).
2. Create an account if your teacher directs you to do so.
3. Type your prompt in the text box and press Enter or the Send button.

You will use AI tools for a variety of tasks throughout this course of study, including:

- Refining problem statements
- Suggesting and critiquing requirements and user stories
- Getting feedback on prototypes and user flows
- Generating task breakdowns and test cases
- Generating Python code and code examples
- Testing, debugging, and improving code
- Explaining programming concepts and helping you understand errors

Remember that AI is a tool to support your learning, not to replace your thinking. You should always read AI responses carefully, evaluate their accuracy, and decide what to keep, change, or improve.

AI tools change frequently. New tools appear, and existing ones are updated with new features. The specific tool you use is less important than the skills you develop: writing clear prompts, evaluating AI-generated results, and using AI responsibly to support your learning.

In later units, these same AI tools will also be used inside the coding environment to help generate Python code, debug errors, suggest tests, and improve documentation.

Exercises

Answer each set of questions in your notebook.

1

Explain what project scope means in a software project. Why is it important to clearly define what is included in a project and what is not included?

2

A team of students plans to develop a mobile application that helps high school students organize homework tasks, set reminders for deadlines, and track their progress across different subjects.

Based on this project idea, identify the following elements:

- Project goal: Clearly describe the main objective of the application.
- Resources: List two technical or human resources that the team may need to develop the application.
- Potential risk: Describe one possible risk that could affect the success of the project and explain why it could be a problem.

3

Choose one of the following project ideas:

A. A mobile app that helps students find study partners

B. A website for a school library where students can search for and reserve books

C. A tool that helps teachers create and share quizzes

- Identify at least three main features of the project.
- For each feature, write 3 to 4 specific tasks. Each task must be specific, assignable, sized correctly, and testable.
- Create a table to organize your tasks. Your table should include the following columns: Task, Owner (role), Estimated time, Depends on, How do you know it is done?
- Then complete the table using the tasks you created for each feature.

4

Using the task breakdown you just created, organize your tasks into a timeline.

Step 1:

Decide how long your project will take (1 week, 2 weeks, or 3 weeks). Divide that time into days.

Step 2:

On paper or in a spreadsheet, create a grid with tasks as rows and days as columns (like the timeline example in the lesson). Color in the days when each task will be worked on.

Exercises

Step 3:

Check for these common problems:

- Are there any days where one person has two tasks at the same time? (Overload)
- Is there a task that starts before the task it depends on is finished? (Dependency error)
- Is there buffer time at the end? (If not, add some)
- Are there any days where nobody has anything to do? (Gap)

Step 4:

Fix any problems you found. Then write 2 to 3 sentences explaining one dependency in your timeline and why the order matters.

5

Consider the following problem statement:

"We want to build an app for students."

- Explain why this statement is too vague.
- Then rewrite it to make it clearer and more specific.

LESSON 2

Requirements and User Stories

After defining the project scope, the next step is to clearly describe what the software must do and who it is for. These descriptions are called requirements. Requirements are clear statements that describe what the system must do and the limits or standards it must meet. Clear requirements help software teams understand what needs to be built before any design or coding begins.

Well-defined requirements reduce confusion, prevent misunderstandings, and help ensure that the final software meets user needs.

The Analysis Phase of the SDLC

As we have seen, the Software Development Life Cycle (SDLC) consists of six main stages: Analysis, Design, Development, Testing, Deployment/Implementation, and Maintenance. This lesson focuses on the Analysis phase, in which teams identify user needs and define system requirements. In the analysis phase, the project team gathers and examines information about the system that needs to be built, including any requirements provided by the customer. These requirements are typically divided into two main categories:

1. Functional requirements
2. Non-functional requirements

A **functional requirement** specifies an action or task the system must perform. Common areas covered by functional requirements include:

- Business rules and administrative functions
- Transaction corrections, adjustments, and cancellations
- Authentication and authorization levels
- External interfaces
- Certification requirements
- Reporting requirements

Examples of functional requirements are:

1. The system must send a confirmation email whenever an order is placed.
2. The system must allow users to verify their accounts using their phone number.
3. The system must allow blog visitors to sign up for the newsletter by leaving their email.

Functional = features (actions the system performs)

Non-functional = qualities (how well it performs)

2.2 Non-functional requirements

Non-functional requirements are a set of rules that describe how the software should work, perform, or operate beyond its basic functional requirements. Some of the more typical non-functional requirements include:

- **Performance**: Requirements related to the speed, responsiveness, and scalability of the software system, such as response time, throughput, and resource usage.
- **Security**: Requirements for protecting sensitive data, such as user authentication, encryption, and access control.
- **Usability**: Requirements for ease of use and user experience, such as navigation, user interface design, and accessibility.
- **Reliability**: Requirements related to the software system's availability and stability, such as error handling, failover, and recovery.
- **Compatibility**: Requirements related to the compatibility of the software system with other systems, platforms, or devices, such as browser compatibility, mobile compatibility, and systems working together.

Examples of non-functional requirements are:

1. The system must recover unsaved data when a sudden power outage occurs.
2. The system must respond to user actions within two seconds when up to 10,000 users are connected at the same time.

Now that we know about the different types of requirements, we can examine how these requirements can be gathered and the methods used to collect them.

Requirements Gathering

An important part of the analysis phase of the SDLC involves finding out what users want from a proposed system or examining an existing system to find out how it works and what might be improved. There are several methods for collecting this data.

- Questionnaires
- Interviews
- Observations
- Examination of existing documentation

Questionnaires

Questionnaires are often collected anonymously, which can encourage users to provide more honest and credible responses. They are generally quicker to administer than interviews, making them a practical option when information needs to be gathered from many users. In addition, responses can be easily processed and analyzed using electronic forms and specialized software.

However, questionnaires may sometimes produce inaccurate responses, particularly if the questions are unclear or if respondents are not fully engaged with the process. They are also less suitable for gathering detailed or descriptive information, as they typically rely on structured or short-answer responses rather than in-depth explanations.

Interviews

Interviews usually take more time than questionnaires, so they are more suitable when there are only a small number of system users. People from different levels of the organization who will use the new system should be interviewed, as this helps the analyst understand how the current system works and what users expect from the new one. During an interview, the interviewer can explain questions immediately if needed and adjust them to suit each person, and participants often take the process more seriously.

But, some people may feel nervous during interviews, which can affect how accurately they respond. Interviews can also be expensive and time-consuming because they require visiting people at their workplace and arranging time for each session.

Observation

Observation allows the analyst to identify the processes involved in the system directly as they occur. By watching users perform their tasks, the analyst can gain detailed and accurate information about how the current system operates, details that may be difficult to obtain through questionnaires or interviews. In addition, this method may be less expensive than interviews because users can continue carrying out their work while being observed, without the need for formal meetings or interruptions.

However, effective observation requires the analyst to have a good understanding of both the existing system and the proposed system in order to interpret what is being observed correctly. Another limitation is that individuals may change their behavior when they know they are being observed, which can affect the accuracy of the information collected.

Examination of existing documentation

Examining documentation can save time, particularly when previous system analysis documents are available. These materials often provide a clear view of how data moves through the system and help the analyst understand the structure and operation of existing processes. Documents can also reveal the scale of the system by showing information such as order volumes, transaction records, and invoice numbers. In addition, they provide a clear picture of the current input and output formats used by the system.

On the other hand, this method relies heavily on the quality and accuracy of the organization's documentation. If records are incomplete, outdated, or inaccurate, the information obtained may be unreliable. Furthermore, collecting and analyzing large numbers of documents can be time-consuming and may require considerable effort from those responsible for the analysis.

Comparison of user requirement collection methods

Methods	Advantages	Disadvantages
Questionnaires	Users are more likely to be honest when they answer anonymous questionnaires.	Questionnaires may be unclear or misunderstood by the user.
	Information can be collected from many users quickly.	Not all useful or necessary information can be collected by questionnaires.
Interviews	Questions can be adjusted for specific users depending on their position or other criteria.	Users may not give honest answers since there is no anonymity.
	Participants are more likely to take the interview or the focus group seriously.	Interviews are time-consuming for both analysts and users, who must take time away from their jobs. Only a limited number of people can be interviewed because it is an expensive process.
Observation	The analyst can get a real understanding of the existing system, while the user can continue working.	Users may not work as they do normally because observation may be intimidating.
Examination of existing documentation	Users are not directly involved in the process, which minimizes disruption to their daily tasks and workload.	The method may overlook undocumented processes or informal practices that users rely on, leading to incomplete understanding of their actual needs.

It is important to note that the criteria for choosing the method of data collection may differ according to the nature of the organization's work, the number of people targeted in the data collection process, etc. Usually, more than one method is used to obtain accurate and realistic data.

Once user requirements have been collected, analyzed, and compared using different methods, they are documented in a clear and structured way. These documented requirements become the foundation for the next stages of the Software Development Life Cycle (SDLC).

Designers use requirements to create system diagrams and interfaces, developers rely on them to guide implementation, and testers use them to verify that the system works as required. Moving forward without clear requirements can lead to misunderstandings, unnecessary extra work, and software that does not meet user needs.

User Stories

Once requirements are documented, they are often expressed in a more user-focused way using user stories. User stories describe what a user wants and why, helping software teams keep the user's needs at the center of development.

A strong user story includes:

- A real user (who)
- A clear action (what)
- A reason/purpose (why)

The following are simple examples of user stories:

- As a student, I want to receive reminders for homework deadlines so that I can manage my time better.
- As a teacher, I want to view student submissions online so that I can give feedback more efficiently.
- As a user, I want to reset my password so that I can regain access if I forget it.

User stories help teams:

- Focus on real users, not just technical features
- Communicate requirements clearly among team members
- Prioritize features based on user needs

Acceptance criteria

Each user story should include acceptance criteria. Acceptance criteria describe the conditions that must be met for the user story to be considered complete and working correctly. Acceptance criteria help developers and testers understand exactly what "done" means. For example, consider the following user story for a school reminder app.

User story:

As a user, I want to receive reminders for upcoming tasks so that I do not miss important deadlines.

Acceptance criteria:

- The system sends a reminder 24 hours before the deadline.
- The reminder includes the task name and due date.
- The reminder is sent as a notification or email.
- The user can turn reminders on or off.

A user story describes the goal. Acceptance criteria describe the exact conditions for success.

These acceptance criteria help to:

- Reduce misunderstandings about how the feature should work
- Support testing and validation of the system
- Ensure the software behaves as expected

2.2

AI-Assisted Requirements Development

AI tools can be helpful to software teams during the requirements stage, but they do not replace human judgment. Instead, AI acts as a support tool that helps teams think more clearly, spot problems, and improve the quality of their requirements.

During this stage, an AI chatbot can be used as an assistant. Students can paste their project description into the chatbot and ask it to suggest user stories, identify missing requirements, or rewrite vague requirements in clearer language. It can analyze a project description and suggest ideas, but people must always decide which suggestions make sense and which do not.

AI can support the requirement definition process in several ways:

- Suggesting user stories: Using data that the team has collected from and about current or potential users, AI can propose user stories.
- Identifying missing requirements: AI can point out gaps, such as missing user types, unclear system behavior, or forgotten constraints.
- Improving clarity: If a requirement is vague or confusing, AI can help rewrite it in clearer and more specific language.
- Checking consistency: AI can help detect duplicate requirements or requirements that contradict each other.
- Helping teams reflect: By asking follow-up questions, AI encourages teams to think more deeply about user needs and system behavior.

Discussion Questions

- Why might organizations use more than one method to collect data about requirements?
- How do user stories help developers understand user needs and how do acceptance criteria help testers?

Example: Password reset feature

User story: As a user, I want to reset my password so that I can access my account if I forget it.

Acceptance criteria:

- The system sends a password reset email.
- The email contains a secure reset link.
- The user can create a new password.

Using AI responsibly during requirements development

AI tools can be helpful during the requirements stage, but they must be used carefully and thoughtfully. AI should support thinking, not replace it. To use AI effectively, it is important to follow a clear and responsible process.

A simple workflow for using AI in requirements

- **Describe the project goal clearly**
  Open an AI chatbot and start by typing a 1- to 2-sentence description of your project. This gives the AI the context it needs to help you.
  Example: "We want to build a mobile app that helps users organize tasks and manage deadlines."
- **Ask AI to suggest user stories**
  Ask the AI to generate a small number of possible user stories based on the project description. This can help teams think about different types of users and their needs.
- **Ask AI what might be missing**
  Ask questions such as: "What requirements might be missing?" or "What other users or features should we consider?" This step helps uncover gaps or overlooked ideas.
- **Ask AI to improve clarity**
  If a requirement or user story is vague, AI can help rewrite it in clearer and more specific language. This is useful for reducing ambiguity before design or testing begins.
- **Humans review and decide**
  Carefully review all AI suggestions and decide what to keep, change, or remove. Final decisions should always be based on the project goals, real user needs, and ethical considerations.

Some requirements are not written down because they seem obvious. For example, users expect apps to load quickly, protect data, and provide safe content. These are called implicit requirements. AI tools rely only on what is explicitly written, so they will miss these unless you clearly include them.

Guidelines for using AI during requirements development

Recommended practices	Practices to avoid
Use AI as a support tool for brainstorming and refining ideas.	Treat AI output as final or automatically correct.
Ask AI to suggest possible user stories from a project description.	Assume AI fully understands users or real-world context.
Use AI to identify gaps or unclear requirements.	Let AI decide which features are necessary.
Review AI suggestions critically and collaboratively.	Skip human review or discussion.
Compare AI output with actual project goals and user needs.	Rely on AI instead of reasoning and judgment.
Ask AI to rewrite vague requirements in clearer language.	Copy AI-generated text directly into project documentation.

Exercises

Answer each set of questions in your notebook.

1

Explain what software requirements are and why they are important in the early stages of the Software Development Life Cycle (SDLC).

2

Explain the difference between:

- Functional requirements
- Non-functional requirements

Then give one example of each.

3

Your team is building an online bookstore. Write user stories for three different types of users.

> **User story format:** "As a [who], I want to [what] so that [why]."
>
> **Users to consider:** a customer shopping for books, a bookstore manager adding inventory, and a delivery driver tracking orders.

Step 1:

Write one user story for each of the three users.

Step 2:

For one of your user stories, write at least three acceptance criteria that describe exactly what "done" means.

Step 3:

Exchange your user stories with a classmate. Review his or her work and answer: Is the user clear? Is the action specific? Is the reason meaningful? Suggest one improvement.

4

Describe what happens during the analysis phase of the SDLC. Why is this stage important before software design and development begins?

5

A major publishing company wants to improve its employee scheduling system.

Explain which of the following methods might be most useful and why:

- Questionnaires
- Interviews
- Observation

6

Open an AI chatbot and enter this prompt: "I am building a mobile app that helps high school students organize homework and receive deadline reminders. Generate 5 functional requirements and 3 non-functional requirements for this app."

A. For each requirement provided by the AI, evaluate:

- Is it clear and specific enough to be tested?
- Is it realistic for a high school student project?
- Does it belong in the correct category (functional vs. non-functional)?
- Is anything missing that you would add?

B. Create your own final list of requirements. Keep the AI suggestions that are good, fix the ones that need improvement, remove any that are out of scope, and add at least one new requirement the chatbot did not suggest.

C. Was the chatbot's output useful as a starting point? What did you have to change and why?

LESSON 3

System Diagrams and Prototyping

Software systems can be complex, involving many steps, users, and interactions. To understand and explain these systems clearly, software teams use diagrams and simple visual models. These tools make it easier to see how a system works, how users interact with it, and how different parts are connected before any code is written.

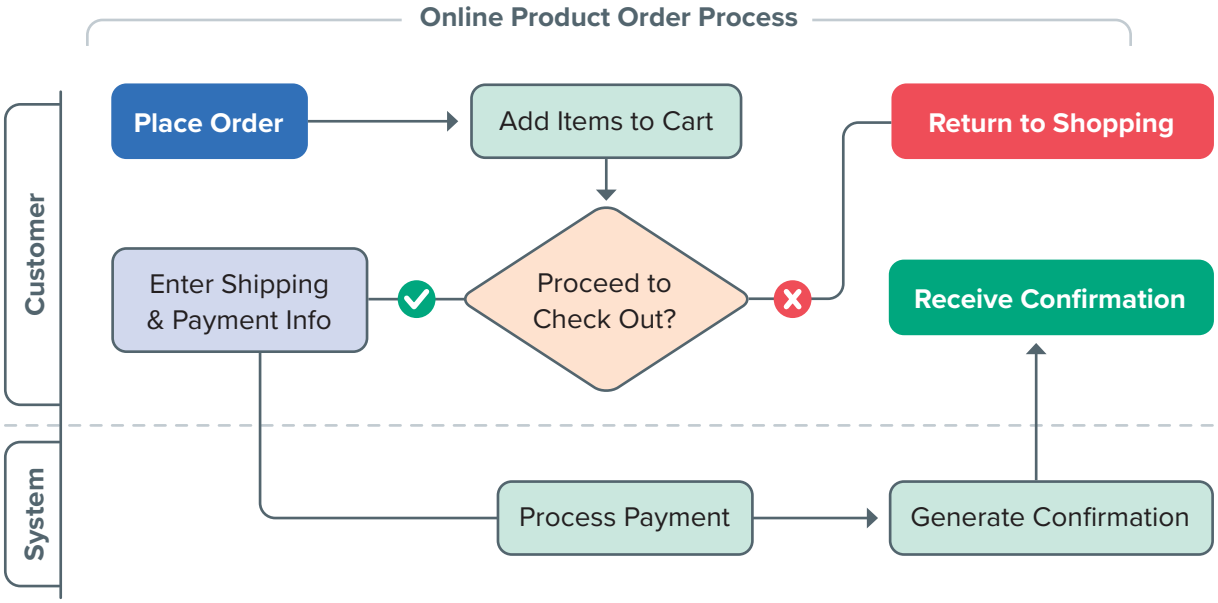
Using Diagrams During Analysis

Diagrams, especially workflow diagrams, and charts are useful tools that can help us in the analysis phase of the SDLC. But, before we start working with workflow diagrams, let's think about what exactly a diagram is. A diagram is a visual representation of information that uses shapes and arrows to show relationships and organization.

Why do we use diagrams?

With diagrams, we can better explain statistical data, system functions, organization charts, and other processes. The visual representation of such information makes it easier to understand. For example, using shapes and different colors in a diagram makes it easier for the reader to compare data and differentiate each result.

Diagrams are used in a wide range of applications. We can use a diagram to illustrate the organization chart of a company, the flow of the processes for a task to be completed, or how network components are connected and related.



Types of diagrams

There are many kinds of diagrams we can use during the different phases of a software development life cycle. Some diagrams focus on how a system works, and others focus on how a user experiences the system. These diagrams help teams understand what the user sees and does, not just how the system operates.

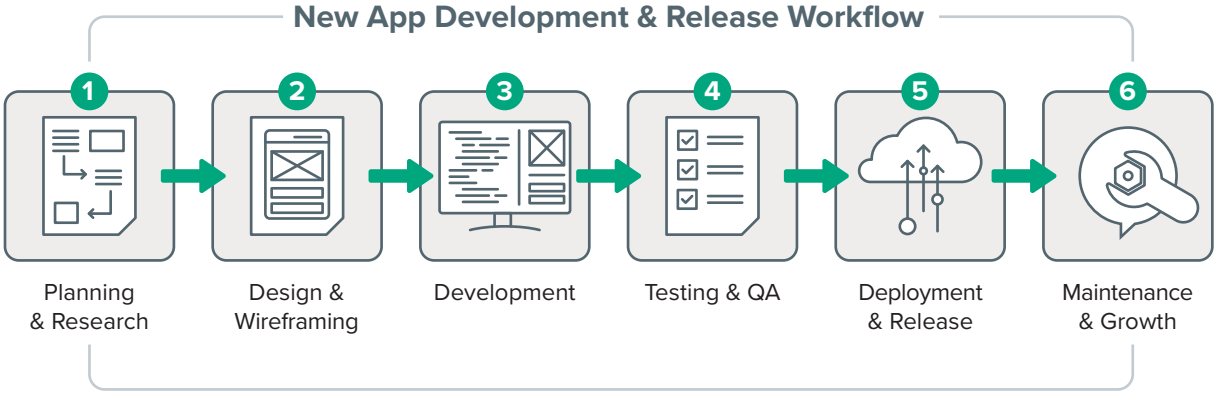
System diagrams: explain processes and structure.

User experience diagrams: explain interactions and journeys.

System-focused diagrams

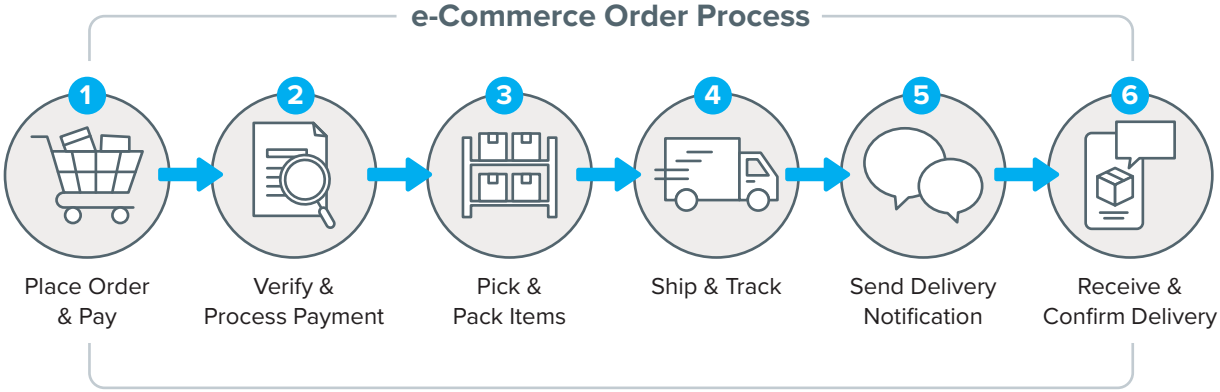
Workflow diagram

A **workflow diagram** is a visual representation that illustrates how tasks, decisions, and people interact within a process. Typically, it consists of a set of symbols representing actions and processes connected by arrows indicating the flow from one to the next. We can use workflow diagrams to illustrate the flow of tasks during each phase of an SDLC.



Flow diagram

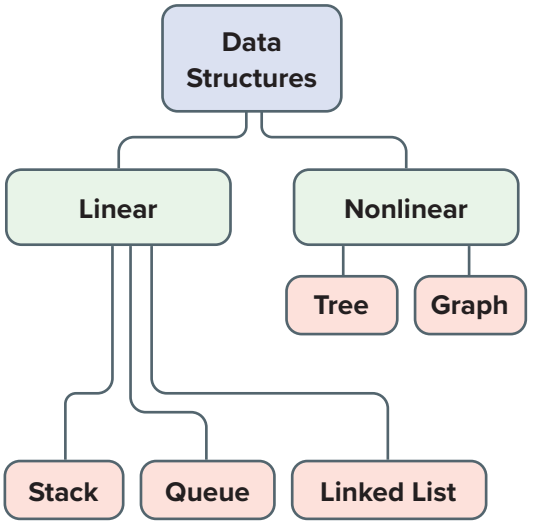
A **flow diagram** shows the step-by-step sequence of actions in a system or process. It is commonly used during the analysis phase of an SDLC to show how tasks move from one step to the next, including decisions and possible outcomes. Flow diagrams help teams spot missing steps, unclear logic, or unnecessary complexity before implementation begins.



Tree diagram

A **tree diagram** is a visual way to represent information that has a hierarchical structure. It starts with a main element at the top, called the root, and branches into smaller related elements called nodes. In software development, tree diagrams help organize and visualize relationships between different parts of a system. They can be used to represent categories, system components, or how information is structured.

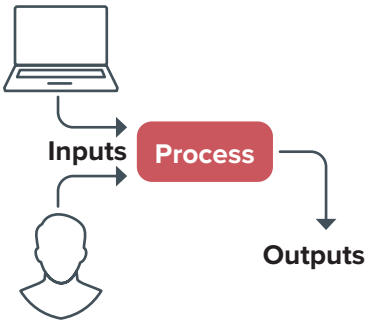
The diagram shown here illustrates how data structures can be grouped into categories. In this lesson, the diagram is used mainly to demonstrate how tree diagrams organize information visually.



Input/Output diagram

An **input/output diagram** is used to show what data enters a system and what results the system produces. Inputs may come from users, devices, or other systems, while outputs are the information or actions generated by the system.

During the analysis phase, input/output diagrams help teams identify required data, expected results, and system boundaries. This ensures that all necessary inputs are considered and that outputs meet user needs before development begins.

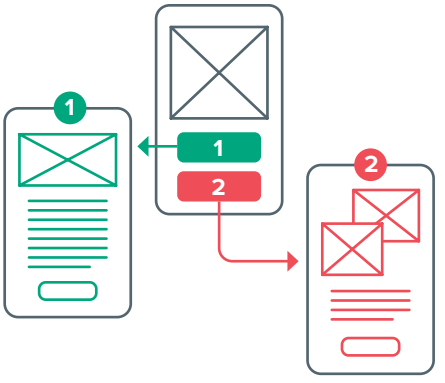


User-focused diagrams

Storyboard

A **storyboard** is a user-focused diagram that shows how a person interacts with a system over time. Each frame represents a screen, action, or moment in the user's journey. In software projects, storyboards are often used after requirements are defined to visualize user stories and system behavior. They help designers and developers share a common understanding of how the system should feel and function from the user's perspective.

For example, a storyboard for a homework reminder app might show a student opening the app, viewing upcoming assignments, and receiving a reminder notification.

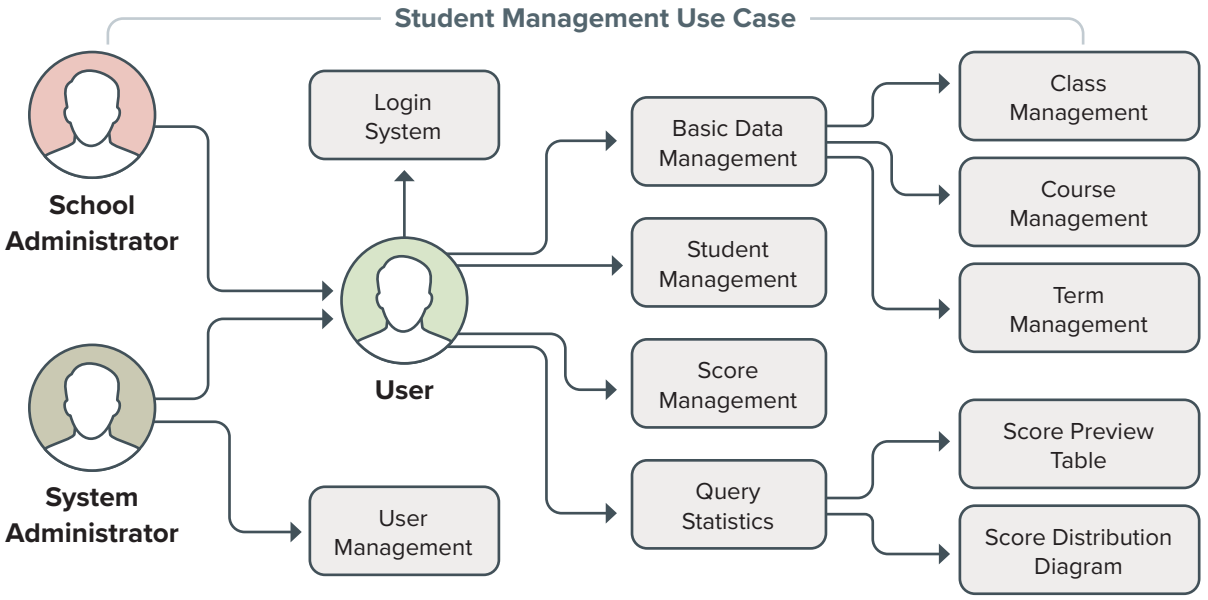


Discussion Questions

- Why might diagrams help people understand a system faster than text?
- Why do developers create diagrams before writing code?

Use case diagram

A **use case diagram** is a type of diagram that represents the different ways a user might interact with a system. Use case diagrams are very useful for representing the requirements of a system gathered during the analysis phase of an SDLC.



The Design Phase

After the analysis phase, the project moves into the design phase. In this stage, the focus shifts from what the system must do to how the system will work and how users will interact with it. During the design phase, system elements such as interfaces, inputs, outputs, and data structures are planned. The goal is not to write code yet, but to decide how the system should be organized and how it should respond to user actions. Some design decisions focus on the technical structure of the system, while others focus on the user experience.

What happens during the design phase

During this phase, teams may plan:

- How users will enter data into the system, such as through text boxes, forms, or drop-down lists.
- How screens, menus, and pages will be laid out for users.
- What outputs the system will produce, such as search results, error messages, or reports.
- How data will be stored, for example in databases or tables.
- How inputs will be validated to prevent incorrect or invalid data.

During the design phase, teams often use wireframes and user flows to plan what the app will look like and how a user will move through it. When you build an app, it is easy to start adding buttons, forms, and pages without a clear plan. The result is often messy, important actions are hard to find, and it is not obvious how to navigate the app. Designing how the app will look and how users will move through it before writing any code helps create a clearer, more organized experience.

Wireframes

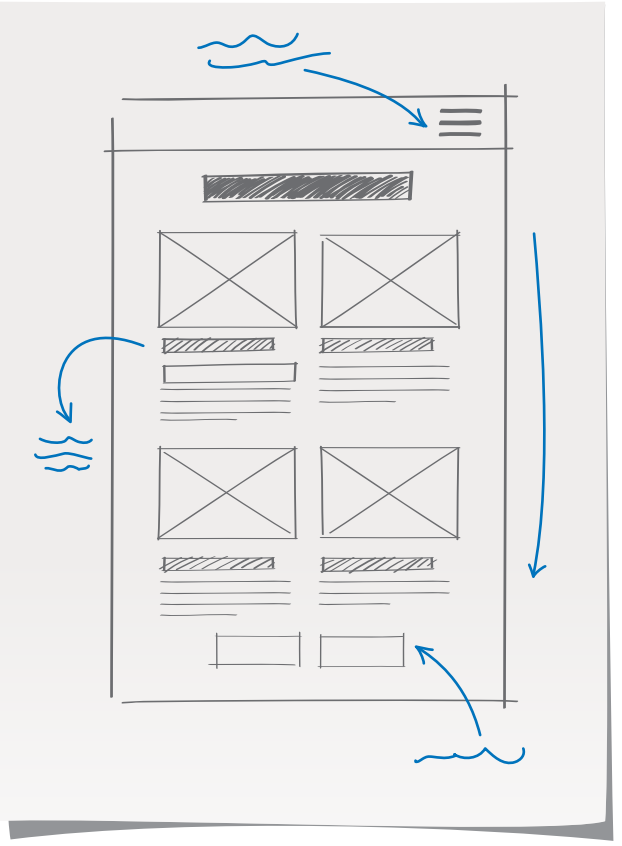
A **wireframe** is a rough sketch of a screen. It ignores colors, fonts, and decoration and focuses on structure: what is on the screen, where it goes, and what the main action is.

On paper, it can be a rectangle for the screen with smaller boxes and labels for headings, input fields, images, and buttons. It does not need to be pretty, but it must be clear.

A useful wireframe lets you answer three questions quickly:

- "What is this screen for?"
- "What should the user do first?"
- "What happens when they are finished?"

For example, a "Create account" wireframe might show a heading at the top, fields for name, email and password, an "Accept terms" checkbox, and a clear "Create account" button at the bottom. If someone glances at the sketch and understands that they should enter their details and press that button, the wireframe is effective.



> **Example: Food ordering app**
>
> Think about a food ordering app. When the screen is well organized, you can tell what to do next: search for food, add items, and tap a clearly labeled "Checkout" button. If the screen is confusing, it is harder to complete tasks efficiently. A wireframe helps you fix that early.

Discussion Questions

- What is one benefit of planning on paper?
- Which sounds easier to fix: a sketch or code?

Wireframes can be changed quickly: you can cross something out and redraw it without spending hours adjusting detailed layouts. Titles are usually near the top, the primary action is more prominent than secondary actions, and related elements are grouped together. You can add short notes in the margins, such as "show error here if password is too short."

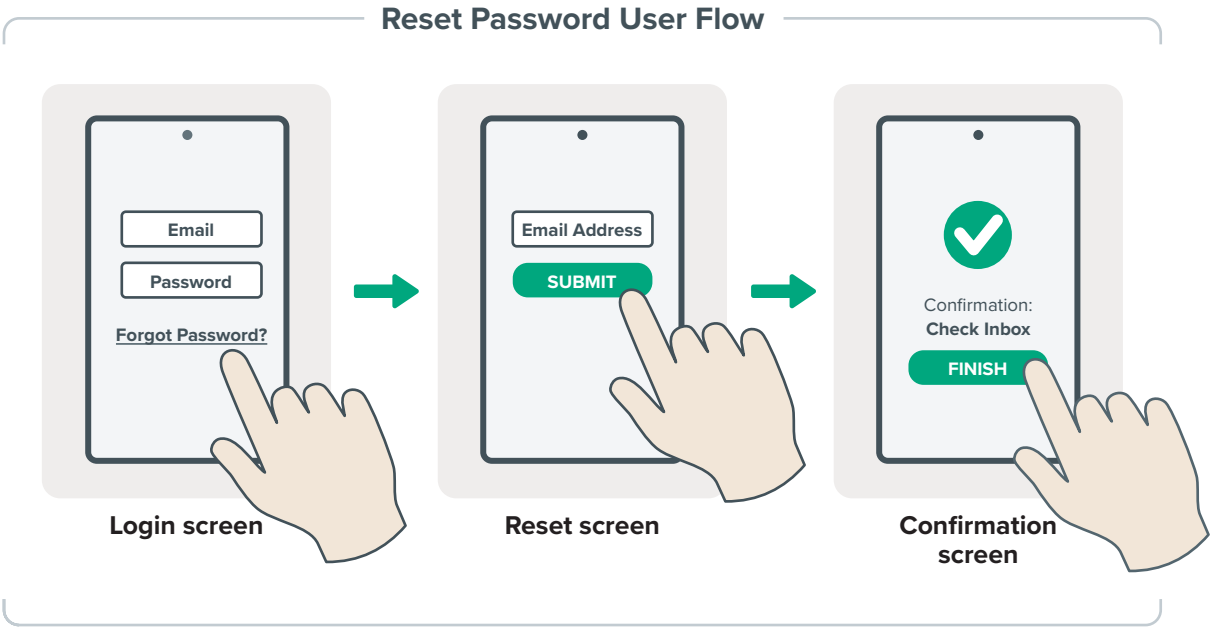
There is no single correct level of detail, but as a rule, someone who is not familiar with your project should be able to look at the sketch, say what the screen is for, and identify the main action without asking questions. If they cannot, the wireframe needs to be simplified or reorganized.

User Flows

A **user flow** is a simple diagram that shows the steps a person takes to achieve one goal in your app. Instead of focusing on a single screen, it tells the story of what happens from start to finish. User flows also help developers plan what the program should do at each step, including what happens when the user makes a mistake or enters invalid data.

You draw boxes for screens or important states and connect them with arrows. Each arrow represents an action or a result: clicking a button, entering valid data, getting an error, or reaching a confirmation.

For example, think of a basic "reset password" task. Someone might start on a login page, click a "Forgot password?" link, enter their email on a reset page, submit the form, and then receive a confirmation message telling them to check their inbox. If they enter an email that is not valid, the reset page might stay visible and show an error message instead of moving on.



As a user flow, that story becomes a short chain of boxes, with a side loop on the reset screen when the email is invalid. You should be able to trace that chain with your finger and explain it in one or two sentences.

User flows help you check whether a task is straightforward or confusing, whether there are missing steps, and whether there are dead ends where the user does not know what to do next.

Discussion Question

Where should an error message appear?

It is common to refine user flows in stages. First you sketch a very simple version that covers what happens when everything works correctly (this is often called "the happy path"). Then you add error cases and alternative routes. You might also mark which steps belong to the same screen but represent different states, such as "initial form," "form with errors shown," and "form after success." The goal is not to model every possible detail, but to make sure there is a clear entry, a clear exit, and understandable steps in between.

Career connections

Wireframes and user flows are used by UX designers, product designers, frontend developers, and software teams to plan how a user will move through a digital product.

Design a Simple App on Paper

Choose one simple task for an app. For example:

- Signing in to an account
- Adding a new item to a personal list
- Booking a seat at a school event
- Submitting a short feedback form

You will work through this task in two stages: first, create the wireframes, then the user flow.

To create your wireframes, decide which screens or states are needed. Often three is enough: a starting screen, a possible error state, and a success or results screen.

On a sheet of paper, draw a separate wireframe for each of those screens. For each one:

1. Add a short title so the purpose is clear.
2. Sketch the main input areas with simple labeled boxes.
3. Show the main button or action distinctly.
4. Decide where any important message will appear, such as an error or confirmation.

Remember that this process should not be a perfect drawing. Before you begin coding a new feature, pause long enough to sketch the main screens as wireframes and the main task as a user flow.

You only need to create rough sketches. What matters is whether someone who has never seen your code can understand what the screen is about and where their attention should go.

Write the user flow

After sketching the screens, describe the same task as a short story in plain language. Begin with what the person sees first, then describe what they do and what the app does in response, until the task is complete. Include what happens when something goes wrong, if that is realistic for your example.

Once you have this paragraph, turn it into a user flow on paper. Draw a box for each screen or state from your wireframes. Connect them with arrows that follow the story you wrote, and add short labels along the arrows to describe the action or condition that moves you from one box to another. When you are finished, check two things. There should be a clear starting box for the task. There should also be at least one clear ending box where the person has achieved what they wanted and there are no more arrows to follow.

Repeat the process with a second small task. You will start to see patterns: certain screens and flows feel natural, while others feel awkward. Those impressions are useful feedback about your design before a piece of code is even written.

Connecting design and code

Wireframes and user flows are not separate from software development. They help guide the interface, navigation, and feedback that will later be built in code.

If the sketches lack clarity, the code developed from them will lack clarity too. If the sketches are simple and clear, you have a strong foundation for the software you will build later.

After planning screens with wireframes and mapping tasks with user flows, teams often build prototypes to test and improve their ideas.

Prototyping

Because many design decisions involve screens, layouts, and user interactions, teams often represent their ideas visually before building the system. This is done using low-fidelity prototypes, such as wireframes, simple diagrams, and basic screen sketches. Prototyping allows teams to explore design ideas, identify issues early, and make improvements before moving on to implementation.

Advantages of Prototyping

Key benefits	Explanation
Better understanding of design content	Prototyping provides a clear visualization of the design, helping users understand the structure and usability of the final product. It also helps the team better understand what they are designing and who it is for.
Improved feedback process	Prototypes can be used to gather feedback from stakeholders at every stage of product development. This includes adding new features or redesigning parts of the product, and testing what works and what does not based on the application's objectives.
Validation of changes before development	Prototyping allows for multiple discussions regarding changes before entering the final development phase. This process facilitates the adoption of appropriate changes and ensures that realistic requirements are built that meet the application objective.
Time and cost savings from early changes	Early changes help you achieve your goals faster. Making adjustments in the final stages is expensive and may require radical restructuring and more thought and reformulation. Having a ready-made prototype enables you to make needed changes early before investing a lot of time and effort into creating the final product.

2.3

Prototyping categories

There are different approaches to modeling, and you should always select the one that works with your product and the resources available for the work. Prototyping methods are generally classified based on their fidelity into low-, medium-, and high-fidelity categories. Fidelity refers to how closely a prototype looks and works like the final product.

The low-fidelity prototype

- This model is usually created using paper in the initial stages of design and is constantly refined throughout the process.
- It helps to make changes easily and quickly, as it focuses more on how the system is used rather than its structure.
- As the product becomes more complex, it is difficult to keep the low-fidelity model in the development cycle, making paper prototypes ineffective in keeping up with the required depth of design.

The medium-fidelity prototype

- A model that is created to simulate and represent system functions, even limited ones, based on specific usage scenarios.
- This model is best for the intermediate stages of product development when moving from a low-fidelity prototype to a more detailed version.

The high-fidelity prototype

- This model is often confused with the final product because of the similarity in appearance and the effectiveness of some system functions.
  High-fidelity models provide a realistic experience similar to the final product, with actual functions.
- It is characterized by accuracy in estimating the required cost and time.
- It supports the analysis of more complex parts of the product in advanced stages, as showing this model in the initial stages may confuse stakeholders and not provide the necessary preliminary understanding.

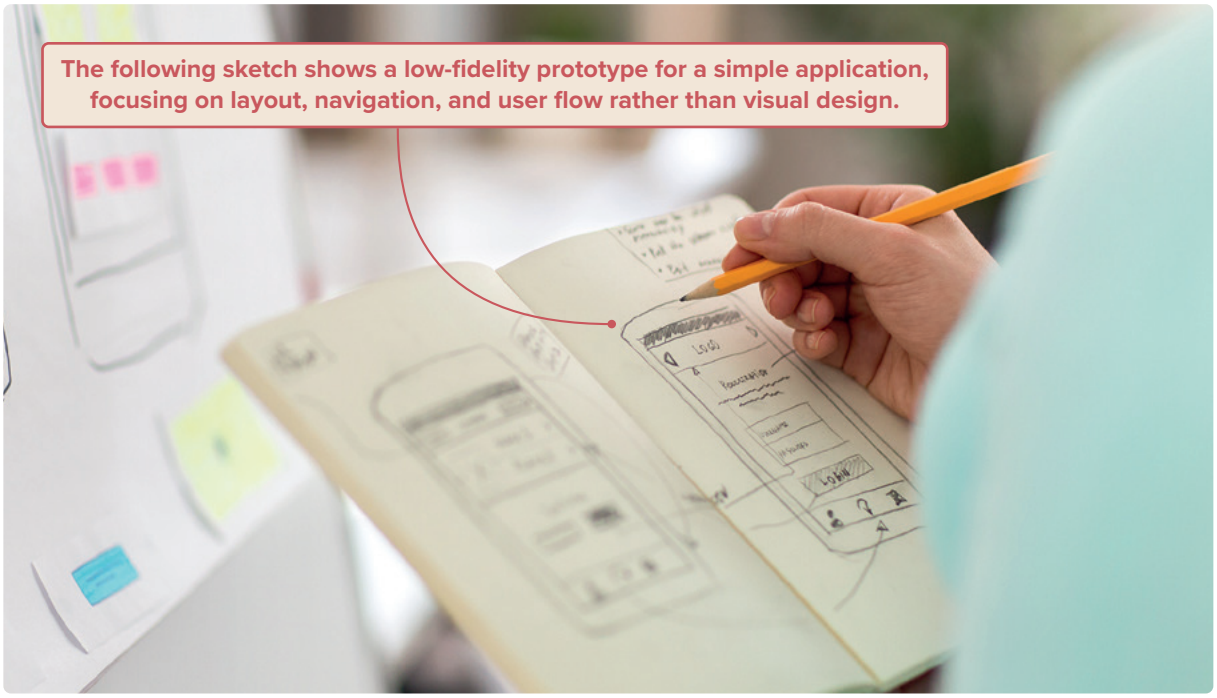
Example: Creating a low-fidelity prototype for a simple application

A low-fidelity prototype uses simple sketches to show how a user might move through an application. These sketches focus on layout, navigation, and basic functionality rather than visual design or technical details.

A typical low-fidelity prototype may include the following elements:

- **Home screen:** Shows the first screen a user sees when opening the application. It includes basic elements such as a title, buttons, or navigation options.
- **Navigation between screens:** Arrows or lines show how a user moves from one screen to another. Each connection represents a user action, such as clicking a button or selecting an option.
- **Key feature screens:** Screens are sketched to represent the main features of the system. Each screen focuses on one primary task the user can perform.
- **Content placeholders:** Boxes, lines, or simple symbols represent text, images, or lists. Exact wording or visual details are not required at this stage.
- **User flow completion:** The final screen shows the outcome of the user's actions, such as viewing results, receiving confirmation, or completing a task.

These sketches help teams quickly test ideas, identify missing steps, and discuss improvements before moving to detailed design or development.



Discussion Questions

- Why are paper sketches useful during early design?
- Why is it better to change designs early rather than later?

AI-Assisted Prototyping

AI tools can support designers during the early stages of prototyping by helping them explore ideas, reflect on user flows, and improve clarity. AI does not replace sketching, design decisions, or user understanding. Instead, it acts as a thinking aid that helps teams work more efficiently and thoughtfully. During low-fidelity prototyping, AI is most useful for idea generation, feedback, and refinement, not for creating final designs.

How AI can support early prototyping

AI tools can assist the prototyping process in several ways:

- **Suggesting interface ideas:** Based on a short project description, AI can suggest possible screens, layout ideas, or navigation structures.
- **Reviewing user flow logic:** AI can help check whether a user flow makes sense, identify missing steps, or point out unclear transitions between screens.
- **Generating alternative approaches**: Designers can ask AI to propose different ways a user might complete the same task, encouraging comparison and discussion.
- **Clarifying screen purpose**: AI can help rephrase or clarify what each screen is meant to achieve, ensuring that every screen has a clear role.

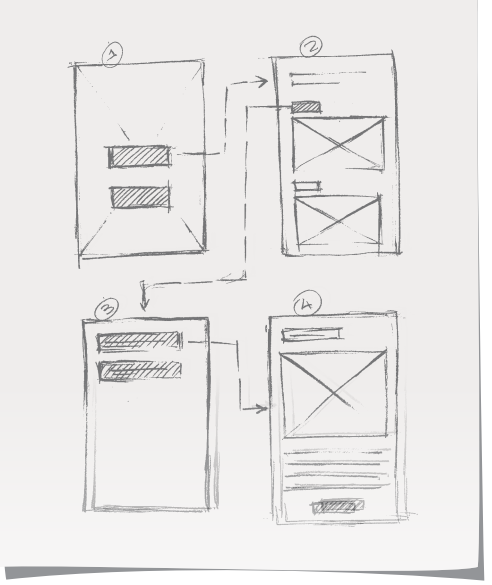
Using AI with a paper prototype

Suppose students have sketched a low-fidelity prototype for a homework reminder app. They might open an AI chatbot and ask questions such as:

- "Does this user flow make sense for a first-time user?"
- "What steps might be missing between the home screen and the reminder notification?"
- "Are there any unnecessary screens in this flow?"
- "What alternative navigation could simplify this process?"

The AI may suggest adding a confirmation screen, simplifying navigation, or merging two screens. Students then decide which suggestions align with their project goals and user needs.

Let's look at an example of how an AI assistant can be used to create a medium-fidelity prototype of an application. First, you start with a low-fidelity prototype, such as a hand-drawn sketch that shows the basic layout, screens, and user flow of the app. Then, you give the AI a clear and detailed prompt describing exactly what you want the prototype to become.



The AI uses your instructions to transform the sketch into a clean, medium-fidelity user interface, keeping the same screen structure and navigation while improving the visual design. This includes replacing rough shapes with proper UI elements (such as buttons, input fields, and navigation bars), applying a consistent color style, and organizing content so each screen is easy to understand.

This example shows how clear prompts lead to better AI results and how AI can support the design process by quickly turning ideas into more polished prototypes.

AI Tool

The prompt:

I am providing a low-fidelity prototype (hand-drawn sketch) of an application.

Convert it into a medium-fidelity UI prototype.

Requirements:

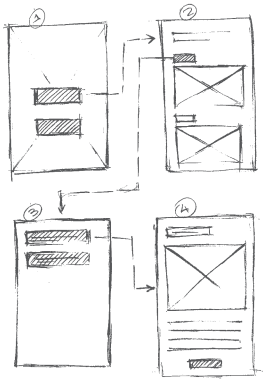
Preserve the original layout, screen structure, and user flow exactly as shown.

Replace rough shapes with clean UI components (buttons, input fields, cards, navigation bars).

Apply a cohesive color palette and modern visual style.

Use placeholder images and text where appropriate.

Present each screen clearly and consistently as a mobile app interface.



The Output:

This example shows a prompt you could enter into any AI chatbot.

The prompt structure (clear instructions, specific requirements) works the same way regardless of which tool you choose.

See the AI References and Resources section for details on this AI generated content.

2.3 **AI can suggest ideas, but it can also:**

- Make incorrect assumptions about users.
- Invent features that were not requested.
- Overcomplicate simple designs.

Human judgment is always required throughout the design and prototyping process. Designers must carefully review any suggestions generated by AI to ensure they align with the project requirements and truly support real user needs. AI-generated ideas should also be evaluated to confirm that they do not introduce unnecessary features or complexity that could make the system harder to use. In this way, AI acts as a tool to support and enhance human thinking, not as a replacement for thoughtful decision-making.

Using AI responsibly during prototyping

When using AI during low-fidelity prototyping, you should:

- Start with your own sketches and ideas.
- Use AI to ask questions, not to make final decisions.
- Treat AI feedback as suggestions, not instructions.
- Revise prototypes based on human discussion and reasoning.

Exercises

Answer each set of questions in your notebook.

1 Explain what a diagram is and why diagrams are useful in the analysis phase of the Software Development Life Cycle (SDLC).

2 Describe two reasons why software teams use diagrams before writing code and explain how diagrams help developers and stakeholders understand a system.

3 A student wants to borrow a book from the school library. Create a workflow diagram showing every step from start to finish.

Include:

- A start point
- At least one decision point (e.g., "Is the book available?" with Yes/No paths)
- Actions for both the student and the library system
- An end point

Exchange diagrams with a partner. Can they follow the flow without asking questions? If not, ask them to suggest one improvement.

4 Choose one of the following systems and create an input/output diagram showing what data enters the system and what results it produces.

A. A food delivery mobile app

B. A school exam grading system

C. A weather forecasting app

D. An ATM

Your diagram must include:

- At least three inputs (on the left side)
- The system name (in the center)
- At least three outputs (on the right side)
- Arrows showing the direction of data flow

5 Describe the differences between low-, medium-, and high-fidelity prototypes and explain when each type might be used during development.

6 Create a wireframe for a homework planner screen.

Exercises

7 Draw a user flow for submitting a school event registration form.

8 Which software development role is mainly responsible for creating a prototype of a product: Product Manager, UX Designer, Developer, Tester, or System Analyst? Explain why the person in this role is most likely to be responsible for building prototypes.

9 Design a low-fidelity paper prototype for a mobile app that helps students find and join school clubs, and use an AI chatbot to get feedback on your design.

A. Create your prototype

Begin by sketching the screens for your app.

Requirements for your prototype:

- Sketch at least four screens (for example: home screen, club list, club details, sign-up confirmation).
- Show navigation between screens using arrows.
- Include content placeholders (boxes for images, lines for text).
- Label each screen with its purpose.
- Do not use colors, detailed graphics, or final text.

B. Describe your design

Write 5-8 sentences describing your prototype:

- What does the user see first?
- How does the user find a club?
- How does the user sign up?

This description will help the AI understand your design.

C. Ask AI for feedback

Enter a prompt such as the following into an AI chatbot: "I am designing a mobile app that helps students find and join school clubs. My app has these screens: [describe your screens and what each one does]. Does this user flow make sense? What steps might be missing? Are there any unnecessary screens?"

D. Evaluate the AI suggestions and update your prototype based only on the suggestions you accepted. Sketch the improved version of your screens.

E. Give your prototype to a classmate. Ask them to "use" the app by pointing at the screens and describing what they would tap.

LESSON 4

User Experience Design and Accessibility

Previously, you planned what your system should do and how tasks are organized. This lesson focuses on how your screens feel to users. Two apps can serve the same purpose, but one may feel simple and easy to use, while the other seems cluttered and frustrating. The difference usually comes from a few basic UX principles, not from elaborate visual design.

User experience (UX) focuses on what it is like for a person to use and navigate a digital product. When you think about UX, you stop asking only "Does this work?" and start asking "Is this easy to understand? Does it behave the way people expect?"

What Good UX Means in Practice

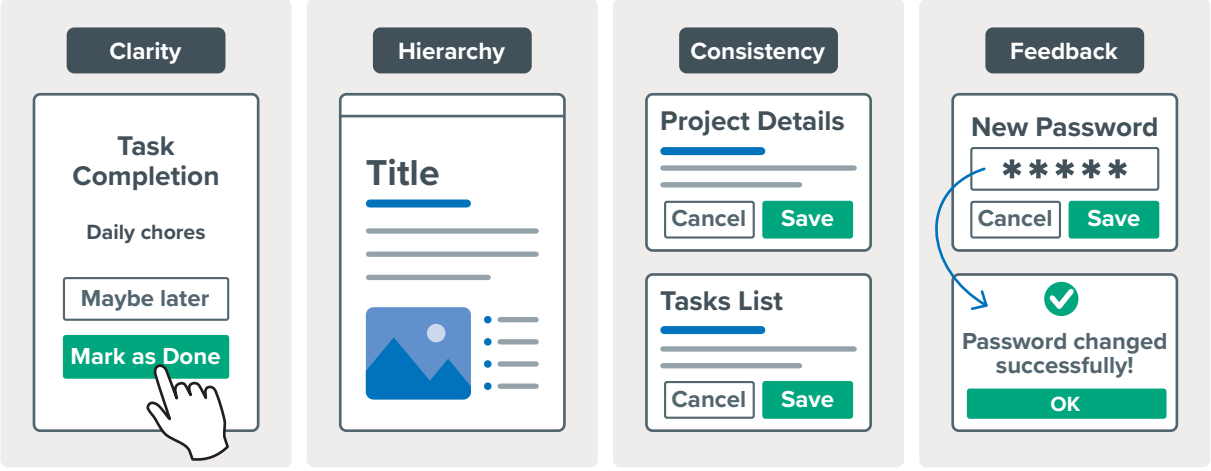
Good UX does not mean the interface looks fancy, it means an average user can understand the screen's purpose right away and complete their task easily.

If a screen needs a detailed explanation, if users keep clicking the wrong thing, or if they are not sure what happened after they pressed a button, the UX is poor, even if the code is perfect. You cannot control every reaction, but you can control how clear, consistent, and responsive the interface is.

You can think of good UX as reducing unnecessary effort. Users should focus on their actual goal, not on figuring out the layout or terminology.

Four simple principles

There are many UX rules, but four are especially useful: **clarity**, **hierarchy**, **consistency**, and **feedback**. Keep these rules in mind when you review a wireframe or a screen.



Clarity

Clarity means that the screen is easy to read, and its purpose is obvious. The main task should be visible in the title, the labels, and the main button. Text should be plain and direct. For example, a button that says "Send message" is clearer than one that says "OK." A heading that says "Payment methods" is clearer than "Options." If someone has to stop and ask "Wait, what does this button actually do?", clarity is missing.

> **Example: Self-checkout options**
>
> You are at a self-checkout machine that shows two buttons: "OK" and "Next." You are not sure what either one does. A clearer screen would say "Pay now" or "Scan next item." When words match the action, the user no longer has to guess.

Hierarchy

Hierarchy is about what the user notices first, second, and third. On a well-designed screen, the most important elements stand out more than the rest. Titles are larger or bolder than regular text, and the main action button is easier to see than secondary links. Related elements are placed close to each other.

If everything looks the same size and weight, the eye has no guidance, and users are not sure where to look first. A clear hierarchy gently guides their attention without them having to think about it.

Consistency

Consistency means that the interface follows its own rules. When similar elements look and behave in similar ways, the app feels predictable. For instance, a primary button should look the same on different screens. Error messages should appear in the same place relative to inputs. If you call a feature "Bookings" on one screen, you should not call it "Reservations" somewhere else.

When an app is consistent, people can rely on what they've already learned and navigate with less effort. When it isn't, they're forced to figure things out again each time, which makes the experience feel more difficult and tiring.

Feedback

Feedback is the system's response to what the user does. It answers the question "Did that work?" Feedback can be a button that briefly changes state when you click it, a loading indicator when something takes time, a check mark after saving, or an error message when input is wrong. Good feedback is specific. For example, "Password changed" is better than "Success." "Incorrect format for email address" is better than "Invalid input."

Without feedback, users press buttons and wait, unsure whether anything happened. With feedback, they can tell that the system has responded, creating a smoother experience.

> **Example: Homework planner app**
>
> A student opens a homework planner app to add a new assignment. The screen shows a title "Add Assignment" at the top, three input fields (subject, description, due date), and a clear "Save Assignment" button at the bottom. Below the button, there is a small "Cancel" link.

A screen as the one described above uses all four UX principles.

- Clarity: the title and button text tell the user exactly what the screen is for and what will happen.

- Hierarchy: the "Save Assignment" button is larger and more prominent than "Cancel," guiding the user toward the main action.
- Consistency: the layout follows the same pattern as other screens in the app, with the title at the top, inputs in the middle, and the main action at the bottom.
- Feedback: after the user presses "Save Assignment," a message appears saying "Assignment saved successfully," confirming that the action worked.

If any of these principles were missing, the screen would be harder to use. For example, if both buttons looked the same, the user might accidentally press "Cancel" instead of "Save." If there were no confirmation message, the user would not know whether the assignment was actually saved.

UX is not something you add at the end of a project; it is part of every decision you make about layout, text, and behavior. Clarity, hierarchy, consistency, and feedback may sound simple, but together they provide a strong framework for evaluating and improving design.

Discussion Question

- What happens when a screen gives no clear visual priority?

Career connections

UX principles are used by designers, software developers, and product teams to make apps easier to understand and use.

A Quick UX Check for Any Screen

When you finish a wireframe or a mockup, use a short checklist to review the screen as if you were a new user, and ask yourself:

- "Can I tell what this screen is for within five seconds?"
- "Do my eyes naturally find the main action, without searching around?"
- "Do similar actions and messages match patterns I have used on other screens?"
- "Is it obvious what happens after I press the main button, both when things go right and when something is wrong?"

If you hesitate on any of these questions, that is a signal to adjust the design before you turn it into code.

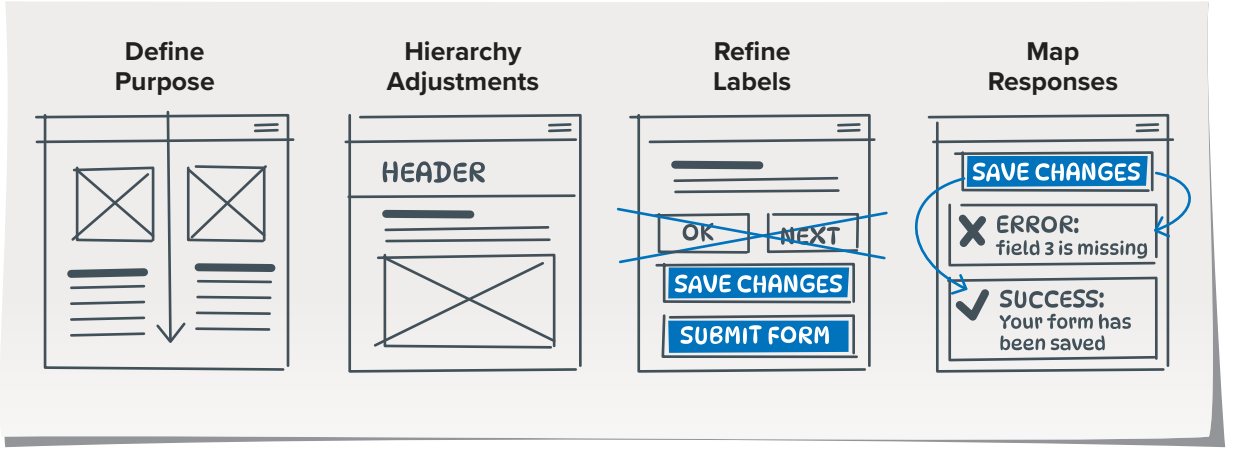
Improving your wireframes

Take the wireframes you drew in the previous lesson and lay them out in front of you. At the top of each page, write one sentence that starts with "The main purpose of this screen is to..." If you struggle to finish that sentence, the screen is probably trying to do too many things. Consider splitting it into two simpler screens, each with a single purpose.

Next, look at each wireframe and decide where you want the user's eyes to go first. That might be the main heading or the main input. Adjust your sketch so that this element is clearly more prominent. You can make it larger, add more space around it, or move it to a more central position. Then check that the main button is easy to find compared with less important controls.

Read every label and button text out loud. Replace vague words with specific ones. If you see "OK," "Next," or "Submit" without context, ask yourself whether you can write a more meaningful verb, such as "Save changes" or "Continue your review." Small wording changes can make a big difference in clarity.

Finally, indicate on your sketch how the interface will respond. Draw where error messages will appear. Write a short note that reminds you what the success state looks like. Think about what the user will see and feel immediately after pressing the main button. If that next step is not clear in your drawing, it will not be clear in your app. Repeat this process for each screen. You are training yourself to identify UX issues early, when changes are easy to make.



Before you move on, make sure you can look at your screens and easily understand what each screen is for, that the main actions are obvious, similar elements behave consistently, and the app gives clear responses to what the user does.

So far, you have focused on making screens clear and easy to use. The next step is making sure everyone can use them, including people with different abilities and needs. Not everyone using your app will see, move, read, or focus in the same way. Some people zoom their screen to 200 percent and still struggle to read small, low-contrast text; some navigate using only a keyboard because they cannot use a mouse; some use a screen reader that reads the page aloud and lets them jump between headings and controls; some find long, cluttered screens exhausting to process.

Accessibility means designing so more people can complete their tasks, regardless of ability. It isn't something extra or reserved for a certain group. In practice, everyone benefits; clear layouts, strong contrast, and informative error messages help all users, not just those with specific accessibility needs. You don't need to memorize the entire Web Content Accessibility Guidelines (WCAG). You only need a set of questions and habits that guide you in the right direction.

Discussion Question

What is one accessibility problem you have encountered?

Different People, Different Ways of Using Your App

When you hear "accessible design," it helps to think about real situations instead of abstract rules.

For instance, someone with low vision might zoom their browser. If your design uses tiny text on a light background, they may have to strain to read it, or they may just give up. A person who navigates using only the keyboard may find parts of your app difficult to operate if it requires using a mouse. A classmate with color blindness might not see the difference between “pass” in green and “fail” in red if there is no accompanying text. And a student who gets overwhelmed by long, dense blocks of text could engage better when you use clear headings and concise sentences.

These examples are not unusual. In fact, many of your future users will experience similar situations at some point in their lives. Accessibility begins with the belief that everyone should be able to use your app without unnecessary frustration.

WCAG describes four broad goals: content should be perceivable, operable, understandable, and robust. In this course, you can begin by thinking about three practical questions: Can I see it? Can I use it? Do I understand it?

Presentation of information (Can I see it?)

Text needs to be readable. That means a reasonable size and enough contrast with its background. Light gray text on white might look modern, but for many people it is almost invisible. In your wireframes, thin, faint text and overcrowded areas suggest the design could be hard to read. There are IT tools for assessing if the contrast of your design is accessible, but there is also a simple test you can use: if you imagine your screen printed in black and white and squint a little, do important headings and labels still stand out?

A page is not accessible if colors alone are used to show important information. If "error" is red and "OK" is green, without any further distinction, a user with color blindness may not be able to tell them apart. Add a word like "Error" or an icon so the information is still clear even without color. Think about charts too. Two lines that differ only in color are hard to tell apart; a different line style or clear labels can help.

Images and icons also need attention. If an icon is the only way to understand something, ask yourself whether a short label would help the user. When you start to build real interfaces, you will learn about "alt text" for images so that screen readers can describe them. For now, in your sketches, you can simply note what an image is supposed to communicate: "product photo," "map of route," "illustration, decorative only."

Low-contrast design

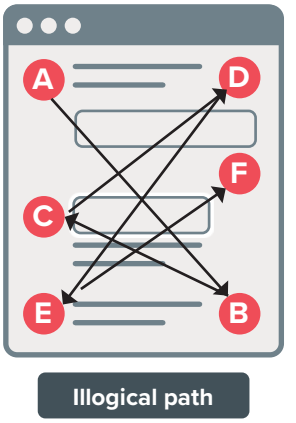
High-contrast design

Discussion Questions

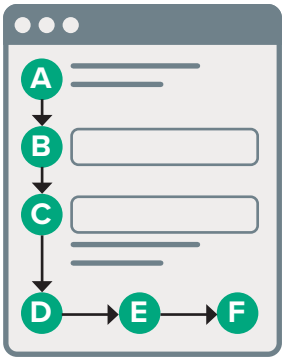
- Why shouldn't color be the only way to show important information?
- What is one way to improve visual contrast?

Interaction (Can I use it?)

Keyboard access is one of the most important aspects of accessibility. Many people use the Tab key to move between controls. If the order of elements jumps around in a strange way, the interface feels confusing. On your wireframes, imagine the user's focus moving from the top left of the screen through all inputs and buttons. Does your design follow that path, or does it require the user to jump around the page? If so, the layout should be adjusted.



Illogical path



Logical path

Hit areas matter too. These are the clickable or tappable areas of an interface that respond to user input. If a button or link is very small, it is hard to click accurately on a laptop trackpad and almost impossible for someone with fine-motor challenges. When reviewing your sketches, note when clickable targets appear too small or are placed too close to each other.

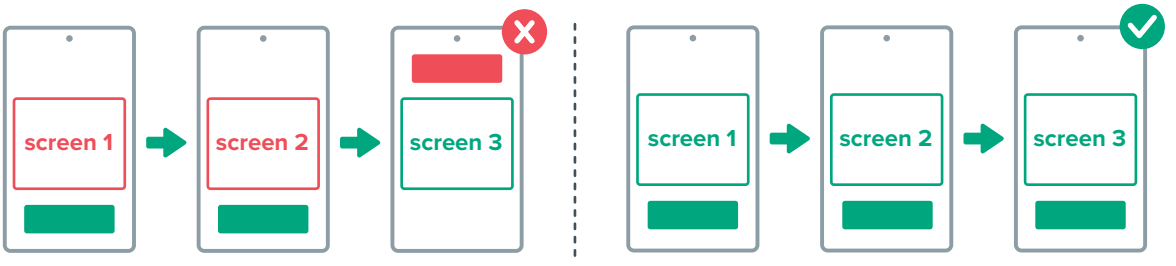
Think about time as well. Some designs assume very quick reactions, like a message that disappears after two seconds or a quiz that auto-submits. People who read more slowly, translate as they read, or get distracted may struggle. A more accessible choice is to keep important messages visible until the user dismisses them, or at least ensure they have enough time to read.

Cognitive load (Do I understand it?)

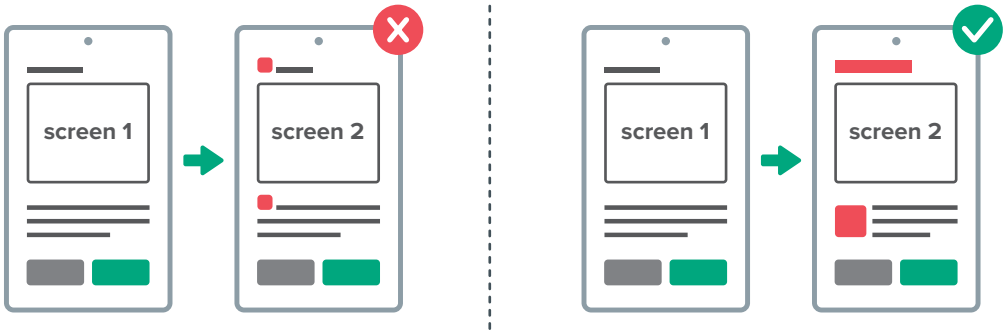
Accessible design should feel easy to follow, even for someone who is tired or distracted.

Language plays a big role. Labels and messages should be clear, direct, and specific. For example, "Name is required" is more helpful than "Invalid input." "Password must be at least 8 characters" tells you exactly what to change. Long blocks of text with no structure are hard to scan. Breaking content into short paragraphs with meaningful headings helps everyone, including those who find reading more difficult.

Layout and behavior should be predictable. If every screen in your app places the main button in the bottom right, do not move it to the top left on one screen for no reason. If error messages normally appear next to the field that has the problem, do not occasionally show them in a separate panel. A user who has learned your pattern once should be able to rely on it.



Also think about changes that are not obvious. If the screen updates in a way that is easy to miss, such as a small number changing in a corner, some users will not notice and will feel lost. Indicate important state changes clearly, close to where the user is looking.



Discussion Questions

- What makes a screen feel overwhelming?
- What is one way to reduce cognitive load?

Career connections

Accessibility is important in the work of UX designers, frontend developers, QA testers, and accessibility specialists who ensure digital products are usable for everyone.

A Closer Look at Forms and Messages

Accessibility issues show up quickly in forms, and they can be very frustrating for users.

An accessible form tells the user what each field is for, what is optional, and what format is needed. Labels stay visible even when you are typing, instead of disappearing inside the field. Required fields are marked clearly. When something goes wrong, the error message appears near the field with the problem and uses specific, helpful language.

It helps to imagine a user who can only see one small part of the screen clearly at a time. When they press Tab to enter a field, can they tell what information belongs there? When they press Enter to submit, if there is an error, can they see exactly what needs fixing?

Similar considerations apply to status messages. When the user triggers something that takes time, such as "upload file" or "generate report," it is important to show that progress is happening. When the process finishes, make that success visible. For accessibility, consider how the screen works for someone using a screen reader: is there a message, heading, or text that announces “Upload complete” or “No results found,” or does the app change silently?

An accessibility review for your sketches

Take one of your wireframes from the previous lesson and evaluate its accessibility.

First, imagine that all color is removed. Will the user still know what is important, where errors appear, and what the main action is? If the answer is no, adjust your sketch: add labels, icons, or headings that convey meaning.

Next, trace the order in which focus would move through the controls if you used only the keyboard. You can even number them lightly in pencil. If the sequence jumps unpredictably, ask yourself how you could rearrange the layout so that the path flows more naturally from top to bottom or from left to right.

Then, read each label and message aloud. If you encounter a vague label like "Submit" with no context, or "OK" as the only button text, choose a clearer verb. If an error message says only "Invalid data," rewrite it on your sketch to explain more clearly what is wrong.

Finally, ask yourself how a screen reader user would experience this page. Screen readers usually move between headings, links, and form fields. Does your wireframe include a clear main heading? Do interactive elements have clear text that explains what they do? Or are you relying on visual placement that users who rely on screen readers cannot see?

You are still working on paper, but you are training yourself to spot accessibility problems before you build them into code.

Redesigning an inaccessible screen

Here is a simple exercise you can do on your own or in class.

Imagine a sign-up page that has several problems: the text is very small and light gray, the instructions appear as one long block of text, there are two empty text fields with no labels, a small red message reads “Invalid” when something goes wrong, the only button is a tiny "O" in the corner, and there is no clear feedback indicating you have successfully signed up.

On a fresh sheet of paper, redraw this screen to make it usable. Make the text larger and use a darker color for better contrast. Break the long paragraph into shorter, easier-to-read instructions. Add clear labels next to or above each text field. Replace “Invalid” with a message that explains the problem. Create a larger button with a clear label. Add a visible success message so users know they have signed up.

When you are done, compare the two designs. The second one is not just "nicer"; it is more accessible because it makes seeing, using, and understanding the interface easier for more people.

Accessibility is less about following rules and more about thoughtful design. Keep in mind how people with varying abilities might see, navigate, and interpret what’s on the screen.

Exercises

Answer each set of questions in your notebook.

1. Explain what is meant by user experience (UX).
2. Briefly explain the terms clarity, hierarchy, consistency, and feedback.
3. Explain why “OK” is often a weak button label.
4. Explain what accessibility means in app design.
5. Explain why color should not be the only way to communicate important information.
6. Describe how keyboard navigation affects accessibility.
7. Describe two ways to reduce cognitive load on a screen.
8. Mark the keyboard focus order on one wireframe.
9. Redesign a weak sign-up screen to improve readability and clarity.
10. Redesign one of your earlier wireframes so that it is more accessible, and explain the changes you made.

LESSON 5

Planning for Testing and Managing Risks

Once a software system has been analyzed, designed, and represented through diagrams and prototypes, the next step is to think carefully about how the system will be tested and what could go wrong. Testing and risk planning are not activities that happen only at the end of a project. Instead, they should begin before any code is written.

Early testing and risk planning help teams build software that is reliable, safe, and aligned with user needs. Testing is the process of checking whether a system behaves as expected. If planning how to test our software is left until the end of development, problems can be harder and more expensive to fix.

Planning testing early helps teams:

- Clarify requirements and expected behavior.
- Detect misunderstandings before development begins.
- Reduce errors and unexpected system behavior.
- Build confidence that the system will work in real situations.

Testing is closely connected to requirements: if a requirement cannot be tested, it may not be clear enough.

Creating test cases before coding

A **test case** is a clear description of how the system should respond to a specific input or user action. User stories and requirements are used when defining test cases.

A simple test case usually includes:

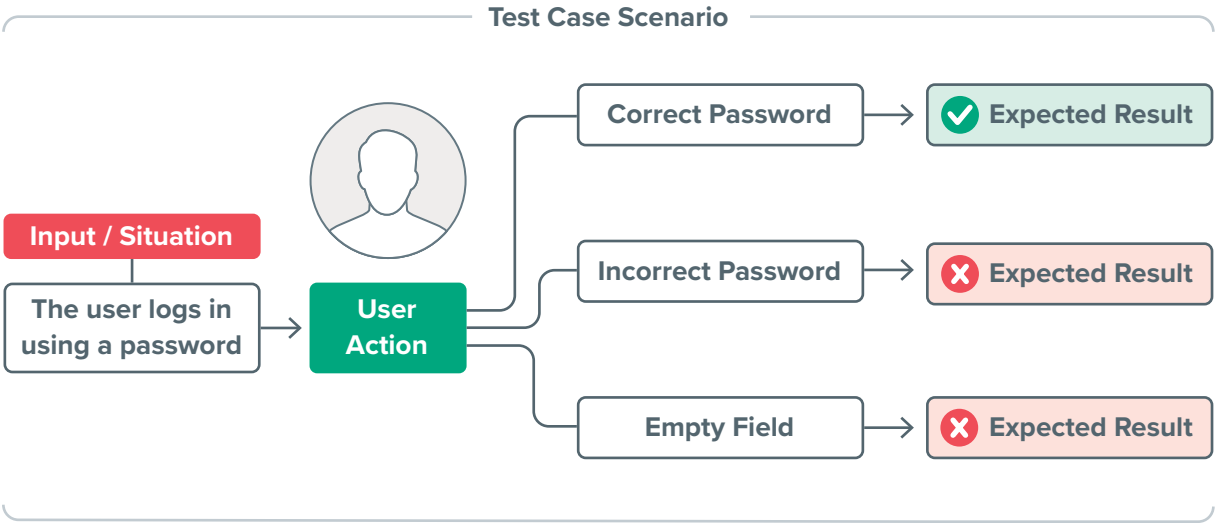
- The situation or input
- The action taken by the user
- The expected result

Writing test cases before coding helps teams:

- Confirm that requirements are complete and specific.
- Agree on what "correct behavior" means.
- Guide future development and debugging.
- Prepare for systematic testing later.

At this stage, testing is about planning and thinking ahead, not running tests on finished software. Teams define what should be tested and what problems to watch for, while the actual execution of tests happens later, after development.

For example, if a system requirement states that a user must log in using a password, the team can plan several test cases in advance. They might check how the system responds when the correct password is entered, when an incorrect password is used, or when the password field is left empty.



Even before any code is written, thinking through these scenarios helps teams understand expected system behavior and identify potential problems early.

Discussion Questions

- Why is testing easier when it is planned early in the project?
- What problems might occur if testing is not planned until the software has been written?

Example: Login system test case

Test case:

Input: username = student1
password = correct password

User action: click Login

Expected result:
The system logs the user in and displays the dashboard.

Understanding risks in software projects

A risk is a potential problem that could negatively affect the software, the users, or the project timeline. Identifying risks early allows teams to reduce their impact or prevent them altogether.

2.5

From Risks to Edge Cases

Once potential risks have been identified, the next step is to think about how those risks might appear when the system is actually used. Many problems do not happen during normal use, but instead occur in unusual or unexpected situations. These situations are called edge cases, and they help teams understand the limits of a system and how it behaves when things do not go as planned.

Types of risks

Risk category	What it Involves	Examples
Technical risks	Problems related to how the system is built and how well it functions	• Features that are difficult to implement • Performance problems (slow response times) • Compatibility issues with devices or platforms
Ethical risks	Concerns about how the system affects users and society	• Collecting more user data than necessary • Lack of transparency about how data is used • Unfair or biased system behavior
AI-related risks	Risks introduced when AI tools are used in the development process	• AI hallucinations, where AI generates incorrect or misleading information • Overreliance on AI-generated suggestions • AI outputs that do not match real user needs

Risk planning is a proactive approach to managing potential problems.

Considering edge cases

An **edge case** is an unusual situation that may not happen often, but can still cause serious problems if it is ignored. Edge cases help teams test the boundaries of a system rather than only its typical behavior.

Examples of edge cases include:

- Empty input fields
- Extremely large or very small values
- Unexpected user actions
- Missing or incomplete data

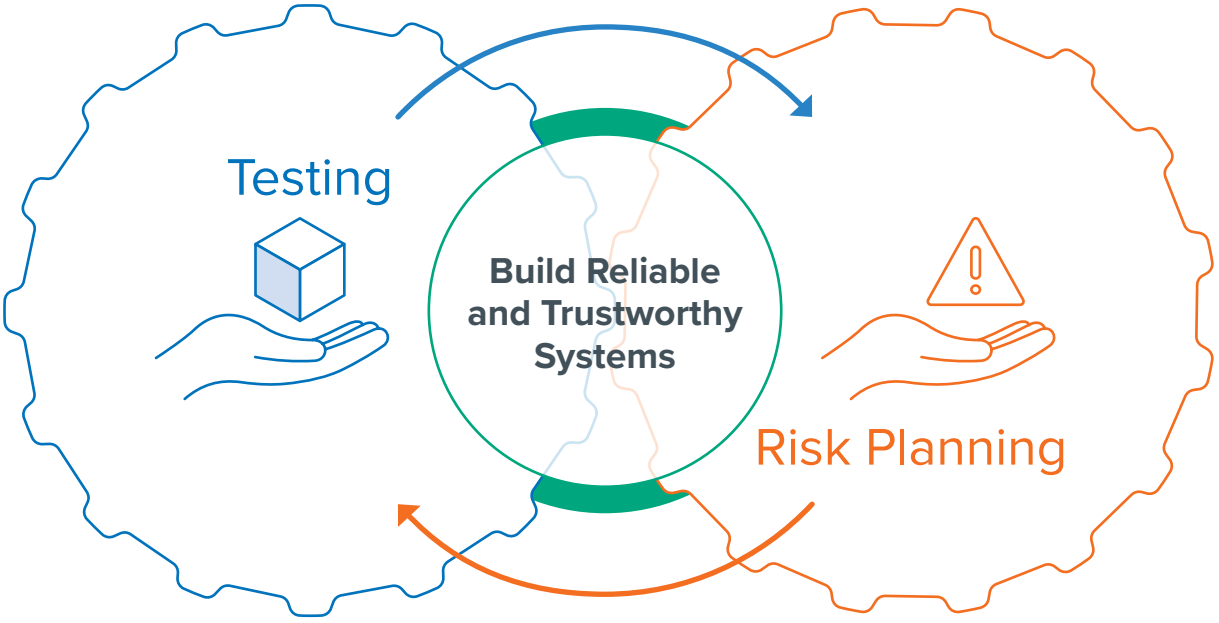
By identifying edge cases early, teams can:

- Prevent system crashes or errors
- Improve system reliability
- Create a smoother and more predictable user experience

Test planning and risk analysis

Test planning involves identifying which features are most important to test and determining which requirements need corresponding test cases. It also includes planning when testing will take place—ideally early, frequently, and repeatedly—and deciding who will be responsible for testing the system, whether developers, other colleagues, or users.

Test planning works closely together with **risk analysis**. Test cases help uncover technical problems, while risk analysis highlights areas that require extra attention during testing. In addition, edge cases often relate directly to identified risks. By planning testing and risk management together, teams can build systems that are more robust and trustworthy.



Test planning helps ensure testing is organized and intentional rather than rushed at the end.

Example: Edge case in a login form

Normal case: User enters username and password.

Edge cases:

- User leaves fields empty
- User enters extremely long text
- User enters special characters

Testing these cases helps prevent crashes and errors.

Later in the course, you will turn these planned test cases into actual checks while building Python programs.

Exercises

Answer each set of questions in your notebook.

1

A user wants to access their account through a login form that requires a username and password. Describe one test case that could be used to verify whether the login feature works correctly. Include the input, the user action, and the expected result.

2

A system requirement states that students must receive a reminder notification one day before a homework deadline. Create one test case that could be used to check whether this feature works correctly.

3

Explain how test cases, risk analysis, and edge cases work together during software development. Why is it important to consider all three when planning a system?

4

A website allows users to search for books in an online library. Create one test case that checks how the system behaves when the search query returns no results. Explain what the system should display to the user.

5

Edge cases are unusual situations that can cause unexpected behavior. For each system below, brainstorm at least three edge cases the developers should test.

System A:

Online bookstore search: A user types a search query to find books.

System B:

Student registration form: A new student fills out a form with their name, email address, and phone number.

System C:

Homework reminder app: Students input due dates for assignments and receive notification reminders.

Why is it important to think about edge cases before writing code, not after?

Project

Designing a software solution

In this project, you will design a software solution by applying what you have learned about requirements, system diagrams, prototyping, and early testing considerations. The focus is on planning and thinking, not building or coding the system.

1. Propose a software solution

- Clearly describe the purpose of the system.
- Identify the target users.
- Explain the problem the system is meant to solve.

2. Define system requirements

Using what you have learned about the analysis phase of the SDLC, decide what the system must do.

- Write functional requirements (what the system should do).
- Write non-functional requirements (usability, performance, security, reliability).
- Organize requirements clearly using lists or tables.
- Optionally, use AI to suggest requirements, then review and refine them.

3. Create system diagrams

- Visually represent how the system works.
- Create a workflow diagram showing the main steps or processes in the system.
- Create a use case diagram to identify users and their interactions with the system.
- Optionally, create an input/output diagram to show what data enters the system and what outputs are produced.
- Create a wireframe to sketch the main interface screens.

4. Design a low-fidelity prototype

Create a paper-based or simple digital prototype.

- Sketch key screens of the system.
- Show navigation between screens using arrows.
- Focus on layout, structure, and user flow.
- Do not include colors, detailed graphics, or final text.

5. Plan for testing and risks

- Create basic test cases based on the system requirements.
- Use clear labels and readable text.
- Use large buttons and simple navigation.
- Maintain high contrast between text and background.
- Provide clear error and success messages.
- Avoid using color alone to show important information.
- Identify possible risks (technical, ethical, AI-related).
- Consider edge cases that could cause problems.

6. Peer review and AI-supported critique

- Share your project with classmates.
- Review your classmate's plan, focusing on the requirements, the diagrams, and the prototype.
- Provide constructive feedback.
- Optionally, use AI tools to assess clarity or completeness.
- Make final improvements to your plan based on the feedback you have received.

2 Wrap-Up

This unit covered how to:

> Analyze a problem and define the scope of a simple software project.
> Identify the main tasks required to complete a software project.
> Use a timeline to organize tasks in a logical order based on the project scope.
> Translate user needs into clear requirements and user stories.
> Use diagrams and prototypes to communicate system ideas.
> Plan tests and identify risks during system development.
> Create a simple UX prototype using wireframes, user flows, and design notes.
> Describe how AI tools can support UI mockups.
> Use AI tools to support planning tasks while critically evaluating AI output.

Key Terms

- acceptance criteria
- accessibility
- AI-assisted prototyping
- edge case
- clarity
- consistency
- feedback
- flow diagram
- functional requirements
- hierarchy
- high-fidelity prototype
- hit area
- input/output diagram
- interview
- low-fidelity prototype
- medium-fidelity prototype
- mockup
- non-functional requirements
- observation
- project
- project planning
- project scope
- prototype
- questionnaire
- requirements
- risk analysis
- storyboard
- test case
- test planning
- tree diagram
- use case diagram
- user experience (UX)
- user flow
- user story
- wireframe diagram
- workflow diagram

Python Foundations 3

3

Introduction

This unit introduces the implementation phase of software development using Python programming. The focus is on translating algorithms into code by working with variables, data types, input and output, operators, and conditional statements.

AI is used as a support tool to generate ideas, debug code, and improve solutions.

Learning Objectives

In this unit, you will:

- Explain how programming connects to the SDLC implementation phase.
- Define what a program and algorithm are.
- Represent solutions using flowcharts.
- Create and use variables in Python.
- Differentiate between numeric and string variables.
- Perform arithmetic calculations based on user input.
- Use conditional statements to make decisions in a program.
- Apply multiple and nested conditions to solve more complex problems.
- Use an AI assistant to help generate and improve code.

LESSON 1

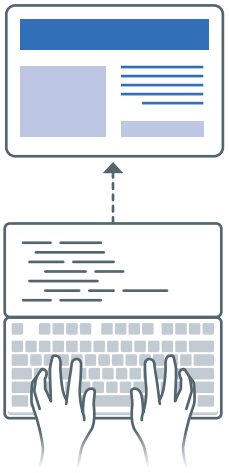
How Programs Think: An Introduction

Earlier units introduced the Software Development Life Cycle (SDLC) and how software systems are planned. In this unit, you will move into the implementation phase, where those plans are turned into working Python programs.

Good programming does not begin by immediately writing code. Instead, it starts with clear thinking and careful planning. Tools such as algorithms and flowcharts help programmers organize the steps, decisions, and actions needed to solve a problem. These tools act as a bridge between the design stage and the actual programming, making it easier to translate ideas into code.

How Do I Write a Program?

A **program** is created by a programmer using a programming language. Computers can only understand instructions written in a series of 1s and 0s, called machine code, which is the lowest level of software that a computer's processor can execute. However, writing programs directly in machine code is extremely challenging and impractical. To solve this, programmers use higher-level programming languages like Python.



These languages allow programmers to create instructions in a more human-friendly way. Special tools, such as compilers or interpreters, then convert the program into machine code that the computer can understand and execute. Before we can start programming in Python, we must first decide the exact steps we want the computer to follow in our program.

Discussion Question

- What problems might happen if a programmer starts coding without planning?

Following the rules

We start learning rules from the day we are born. We have to follow rules as we grow up, and we live and work with rules all our lives. For example, when we get up every morning, we perform a set of actions in order. Rules are not always clear in life and sometimes people need to make new rules according to specific situations.

However, computers cannot make decisions by themselves. They have to follow very specific instructions. Computers only do what people tell them to do. If you give them the wrong instructions, the result will also be wrong or the work will not be completed. In programming, these precise rules are written as step-by-step instructions called algorithms.

3.1

Algorithms: the bridge from design to code

After understanding the problem, gathering requirements, and designing how a system should work, the next step is to decide exactly what actions the system must take. Computers cannot think or make assumptions on their own, so every action must be clearly planned in advance. To do this, we describe the solution as a series of precise steps that the computer can follow. These step-by-step instructions are called algorithms.

An **algorithm** is a set of clear, step-by-step instructions used to solve a problem or complete a task. The steps must be easy to understand, must be followed in the correct order, and the process must stop after it gives a result (output).

> **Example: Following a recipe to bake a cake**
>
> When baking a cake, you need to follow a specific set of steps in the correct order. This includes measuring ingredients, mixing them, preheating the oven, and baking for a certain amount of time. To ensure the cake turns out correctly, the recipe acts as an algorithm, a clear sequence of instructions that must be followed step by step. Each step depends on the previous one, and skipping or changing the order can affect the final result.

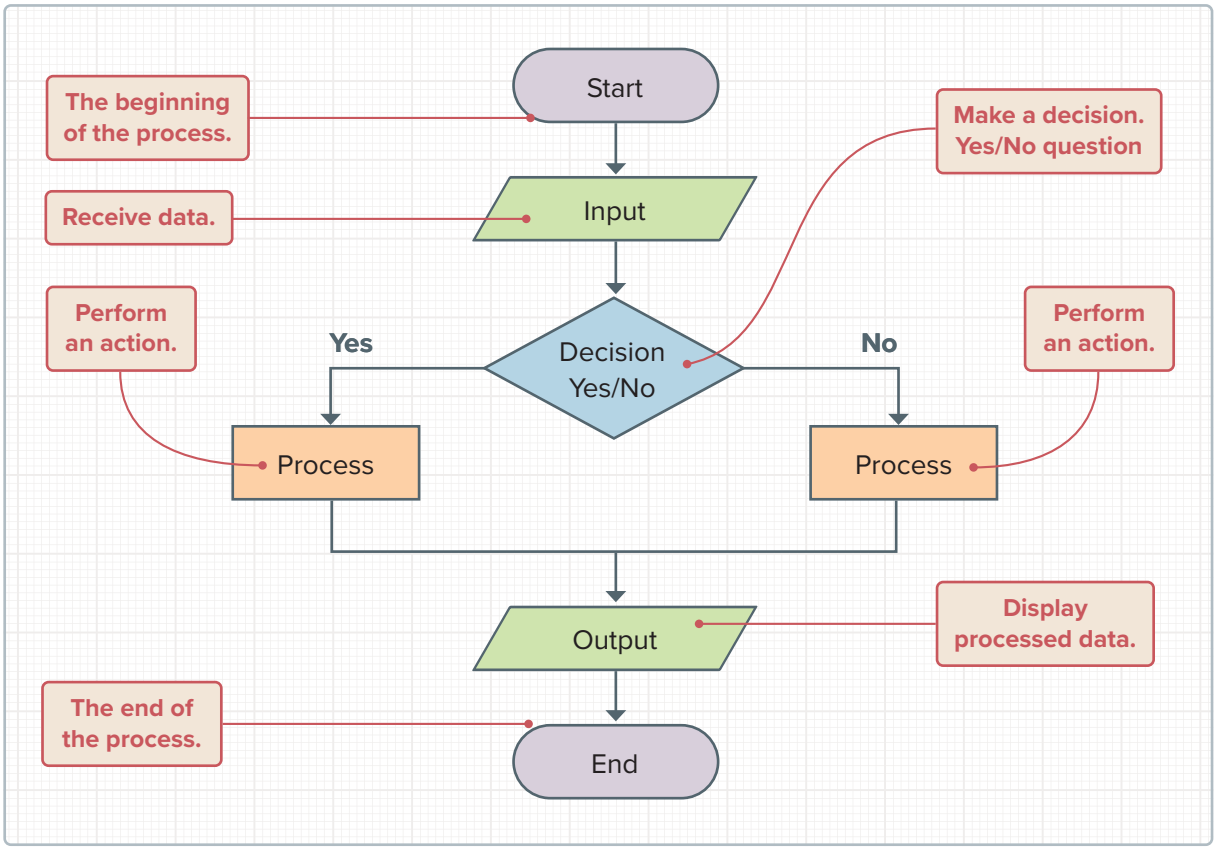
Flowcharts

A **flowchart** is a type of diagram that represents an algorithm. It outlines the steps you need to follow and their correct order. This diagram gives a clear step-by-step solution to a problem broken into smaller tasks or specific instructions. You can make flowcharts to describe your thoughts about how to solve a problem using a computer before you actually start writing the program.

You can represent the steps of an algorithm by drawing four different types of shapes to reflect different actions and connect the shapes with arrows to indicate their order.

Flowchart elements

Type of shape	Description
Start/End	Used to mark the beginning and end of the process.
Input/Output	Used to receive data and display processed data (input and output).
Process	Used to perform an action (do calculations or give commands).
Decision	Used to make a decision, usually a Yes/No or a True/False question.
	Used to indicate the order in which the steps flow.



Flowchart guidelines

A flowchart must have one Start symbol and one End symbol.

- The flow should move from top to bottom or left to right.
- All the steps of the algorithm must appear in the flowchart.
- The arrows show the order of the steps.
- Decision symbols must have two paths (for example, Yes/No).

Career connections

Software developers, system analysts, and testers use algorithms and flowcharts to plan how a system should behave before writing code. These tools help professionals break complex problems into clear steps, making it easier to design reliable apps, websites, and software that people use every day.

Discussion Questions

- How do flowcharts help programmers understand the solution?
- What might happen if a flowchart is missing a step or has steps in the wrong order?

Stages of program creation

1. Define the problem
The first thing you have to do is to identify the problem and write the steps needed to solve it.

2. Create the algorithm
Then put the steps in a logical and sequential order to form the algorithm.

3. Design the flowchart
The next step is to draw a flowchart, which represents the logical sequence of the algorithm.

4. Write the code
The last step is to write the program in Python.

Define the problem

Before you start designing a program, you must first define and understand the problem that needs to be solved and determine the necessary steps to reach a solution.

For example, suppose you want to calculate the area of a rectangle. You begin by identifying the steps required to find the answer. In this case, you need to know two measurements: the length and the width of the rectangle. Then, you can apply the formula:

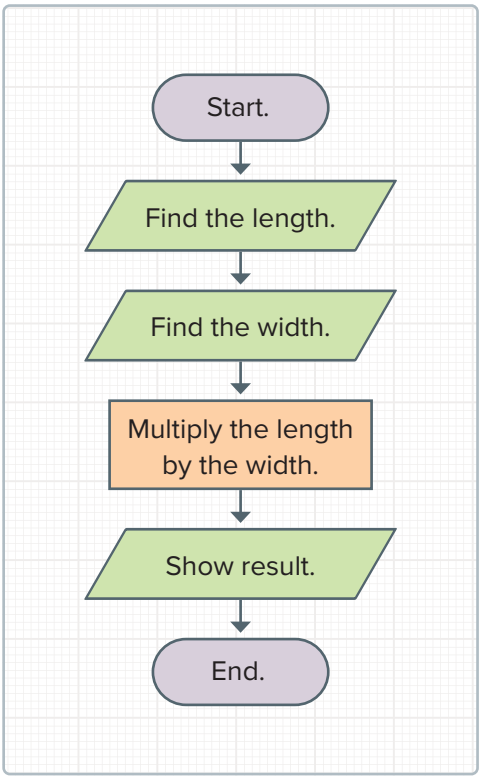
Area = Length x Width

The algorithm

Steps of the algorithm

1. Find the length.
2. Find the width.
3. Multiply the length by the width.
4. Show the result.

The flowchart



Algorithm: Step-by-step plan

Program: Algorithm written in a programming language.

Using AI to plan algorithms and create flowcharts

AI tools can help you describe a problem as a clear sequence of steps. For example, you can ask an AI assistant to help you outline the steps needed to complete a task or to represent those steps in a flowchart. This can make it easier to organize your ideas and check whether your solution makes sense.

AI Tool

The prompt:

Create a step-by-step algorithm to calculate the area of a rectangle.

The output:

1. Step-by-Step Algorithm: Calculate the Area of a Rectangle
2. Start.
3. Enter the length of the rectangle.
4. Enter the width of the rectangle.
5. Multiply the length by the width.
6. Store the result as the area.
7. Display the area.
8. End.

Check whether the algorithm has clear steps, follows the correct order, and includes defined inputs and outputs.

See the AI References and Resources section for details on this AI generated content.

Exercises

Answer each set of questions in your notebook.

1 Why do programmers use algorithms before writing code?

2 Below are the steps for making a cup of tea, but they are not in the correct order. Rearrange them to create a correct algorithm.

1. Pour the hot water into the cup.
2. Put a tea bag in a cup.
3. Boil water.
4. Drink the tea.
5. Wait for the tea to brew.

3 Create a flowchart for the following problem. A program asks the user to enter a number and then displays the square of the number. Your flowchart should include:

1. Start.
2. Input number.
3. Calculate the square of the number.
4. Display the result.
5. End.

4 Put the steps below in the correct order to create an algorithm that asks the user to enter the prices of two products and then calculates the total cost.

- Display the total price on the screen.
- Ask the user to enter the prices of the two products.
- Calculate the total price by adding the two values.
- Read the two prices.

5 Create a simple flowchart to show the steps you would take to organize a trip. Use these three main questions to guide your flowchart:

- Where will you go?
- How will you get there?
- Will you travel by car, train, or plane?

6

Write an algorithm to calculate a student's final grade. The final grade is calculated as the average of three grades.

Below are the steps to create the algorithm, in random order. Arrange the steps correctly, and then create the flowchart of the algorithm.

- Calculate student's final grade by adding the grades and dividing by 3.
- Ask the user to enter the three grades.
- Display the result on the screen.
- Read the three grades.

7

Below are the instructions for a program that calculates a library late fee:

- Ask the user to enter the number of days the book is late.
- If the book is 0 days late, display "No fee".
- If the book is 1 to 5 days late, charge $1 per day.
- If the book is more than 5 days late, charge $2 per day for each late day.
- Display the total fee.

1. Define the problem.
2. Write the algorithm in clear step-by-step form.
3. Draw a flowchart using the correct symbols.
4. Explain what the input, process, output, and decision are.

LESSON 2

Variables, Data, and Output

In the previous lesson, you learned how programmers plan solutions using algorithms and flowcharts. In this lesson, you move into the implementation phase of the Software Development Life Cycle, where those planned steps are translated into real Python code. You will begin by learning how to write simple programs, display information on the screen, and store values using variables. The environment you will use for writing and running your Python code is Microsoft Visual Studio Code.

Microsoft Visual Studio Code

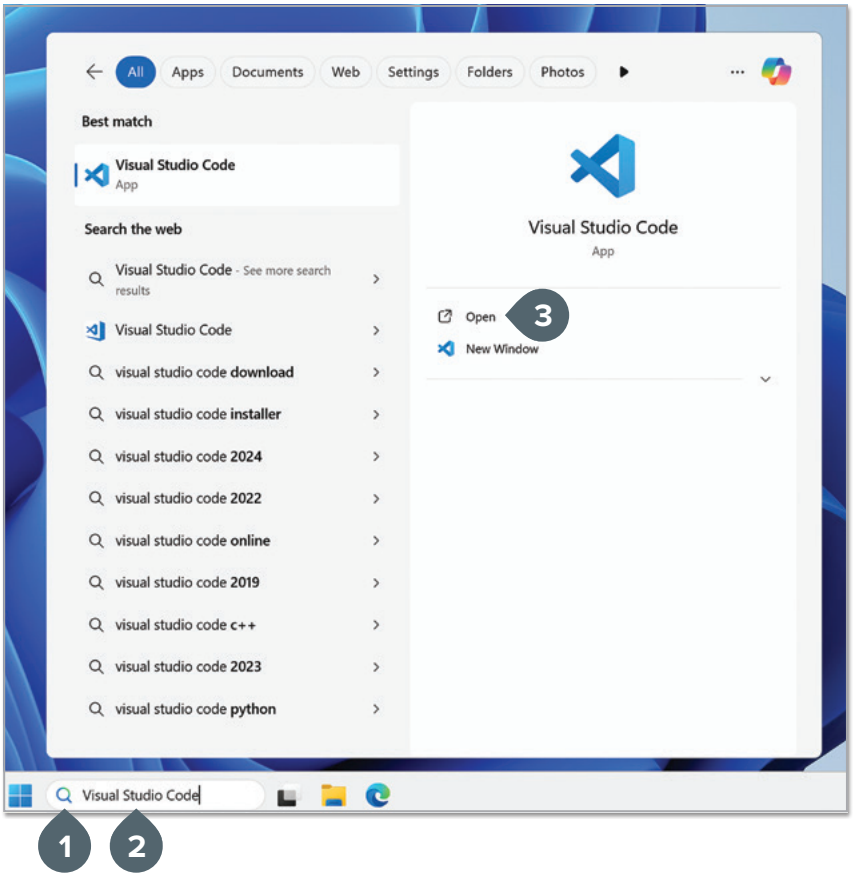
Visual Studio Code (VSC) is a code editor developed by Microsoft. You can use VS Code to view, edit, run, and debug your code. It supports multiple programming languages (Python, Java, HTML, etc.) and also offers many extensions, so if the user wants to use a programming language not already included, they can download the appropriate extension and use it.

You can download Visual Studio Code for free from the Microsoft Store.

To open Visual Studio Code:

> Click the **Search** button on the **Taskbar**. 1

> Type "**Visual Studio Code**" in the search bar 2 and then click **Open**. 3

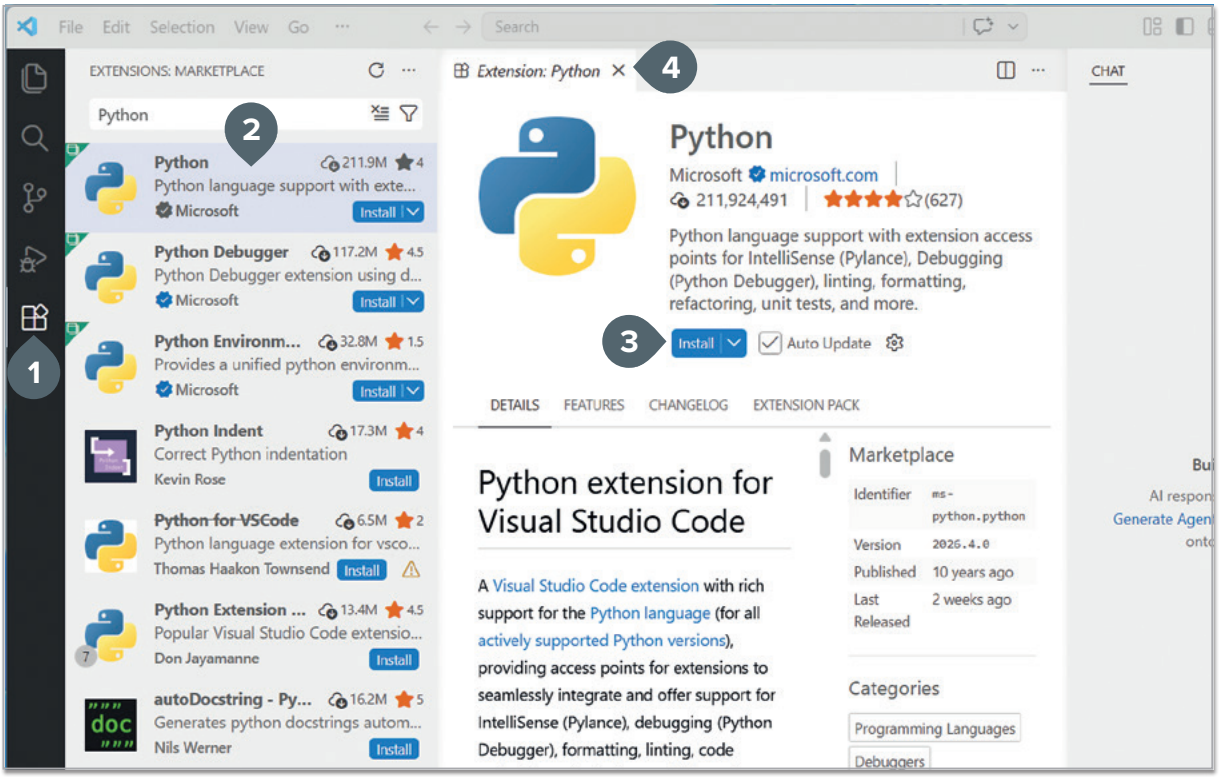


A high-level **programming language** uses words, symbols, and structures that are easier for humans to read and write than machine code. It has words that you need to learn and syntax rules that you have to follow just like with a human language. Python, Visual Basic, and JavaScript are all high-level programming languages.

VS Code supports many different programming languages. To run Python programs, you need to install the Python extension.

To install the Python extension:

> Click the **Extensions** icon. 1
> Select **Python**. 2
> Click **Install**. 3
> Close the tab to start typing your code. 4



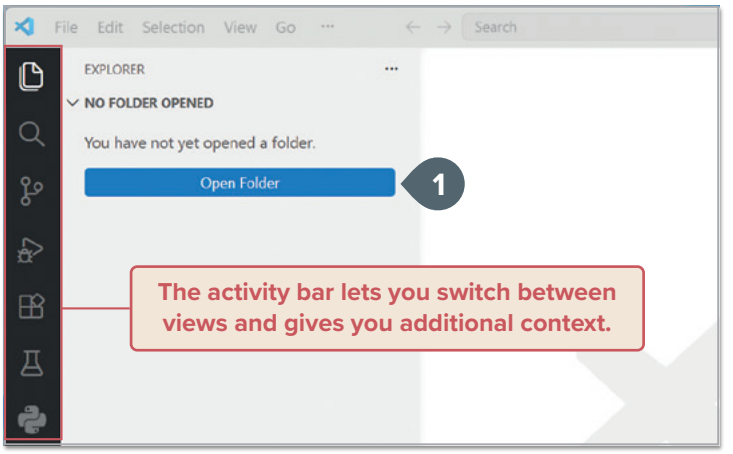
3.2

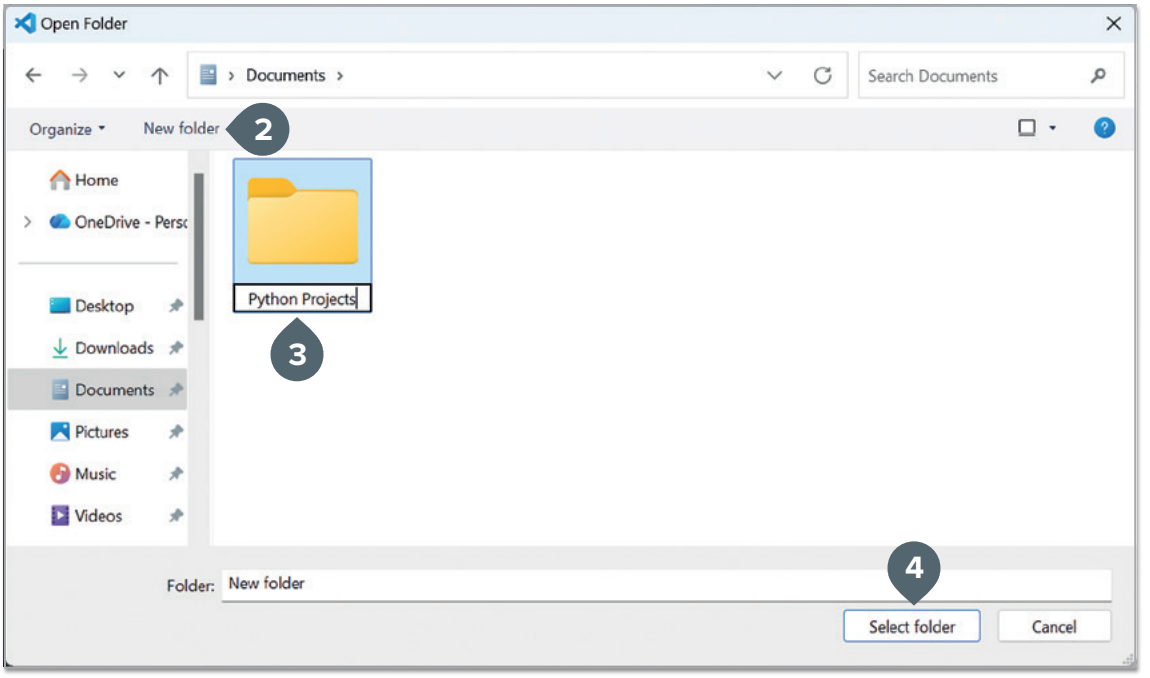
To keep your program files organized, you need a folder to save them in.

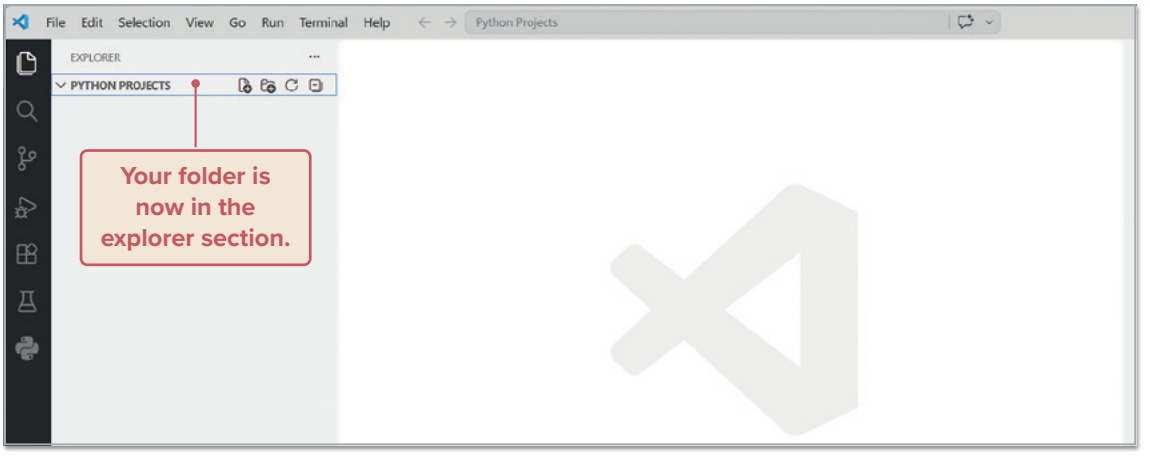
VS Code gives you the option to open a folder to provide more convenient access to your files.

To create a folder for your files:

> Click **Open folder.** 1
> Click **New folder** 2 and type a name. 3
> Click **Select Folder.** 4



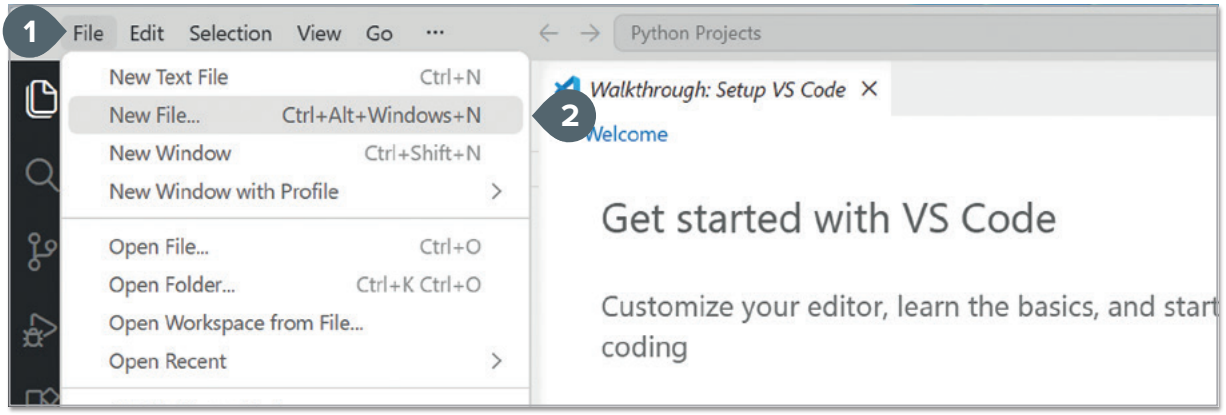


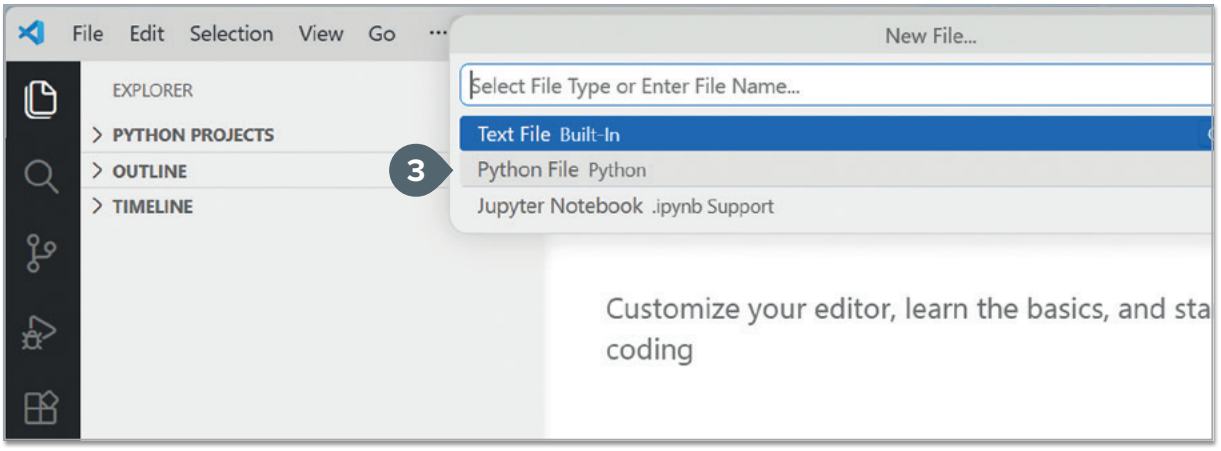


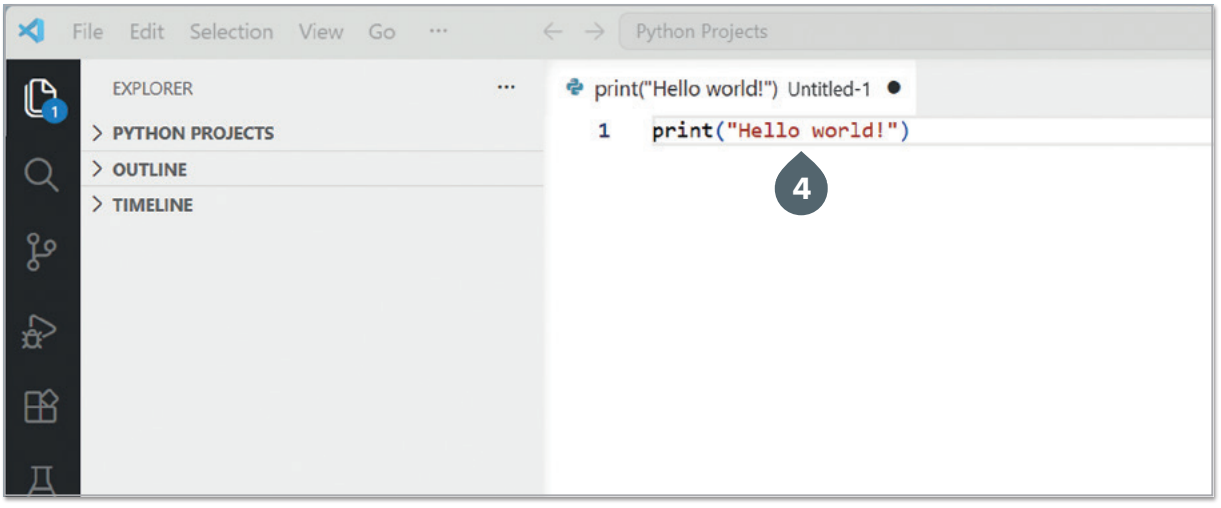
Now, it's time to create your first Python file in VS Code.

To create a new file:

> Click **File** 1 and select **New File**. 2
> Click **Python File**. 3
> Type the simple Python statement **print("Hello world!")**. 4



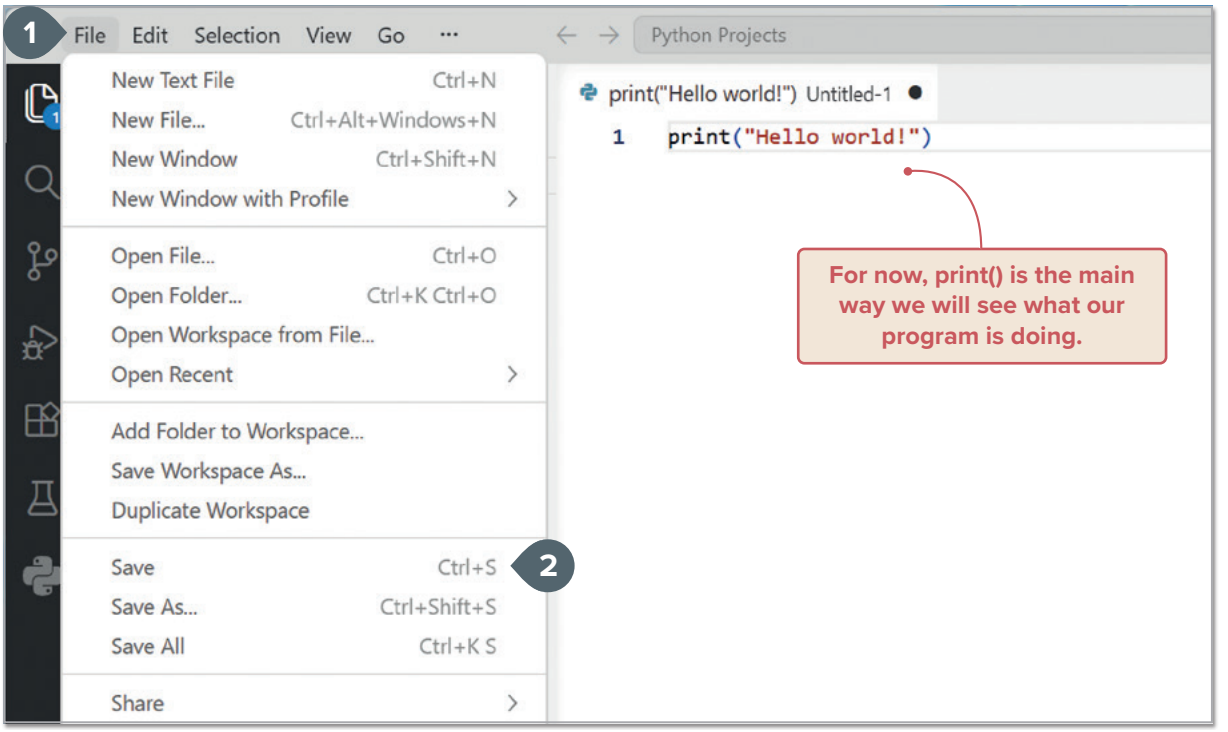


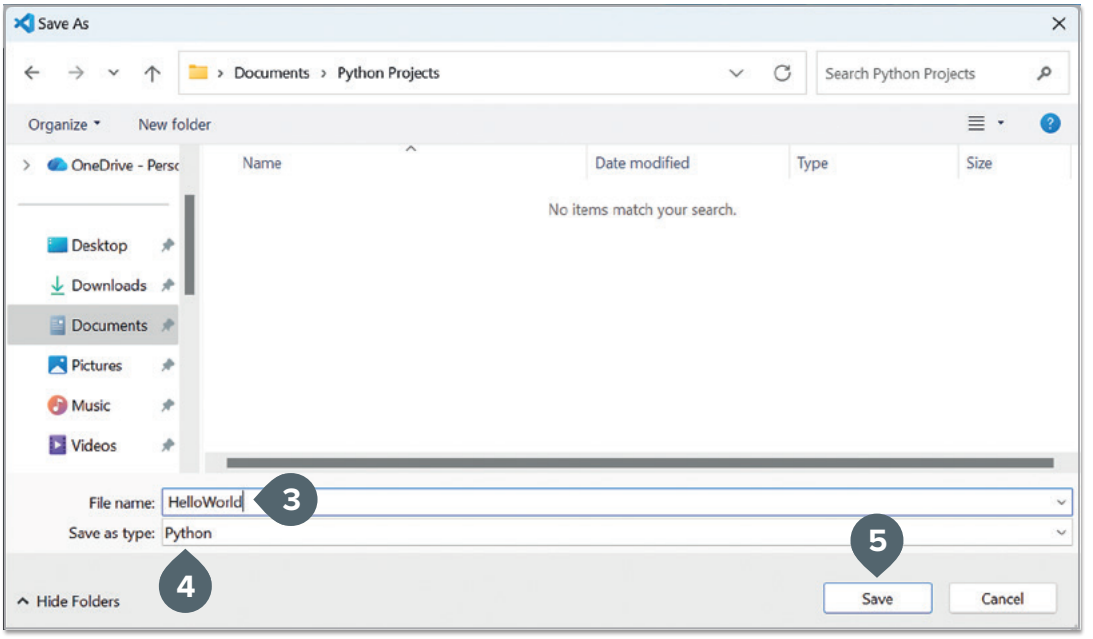


3.2 The next step is to save your file. Normally, the program will select the Python file type automatically. If not, you need to manually set the **".py"** extension.

To save your file:

> Click **File** 1 and select **Save**. 2
> Type a name for your file. 3
> From the drop-down list of file types, select **Python**. 4
> Click **Save**. 5

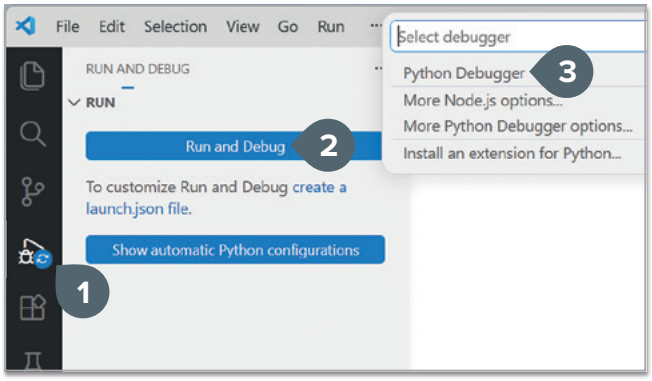


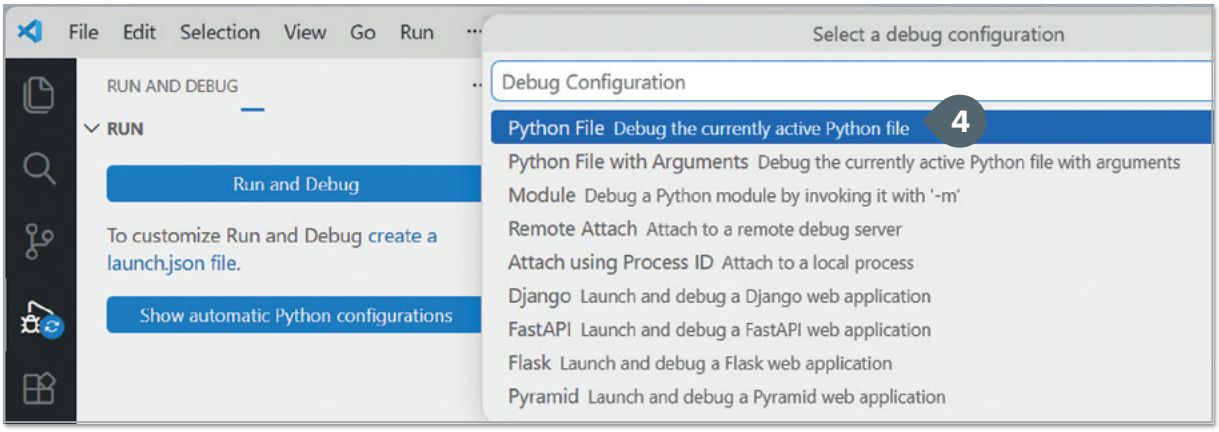


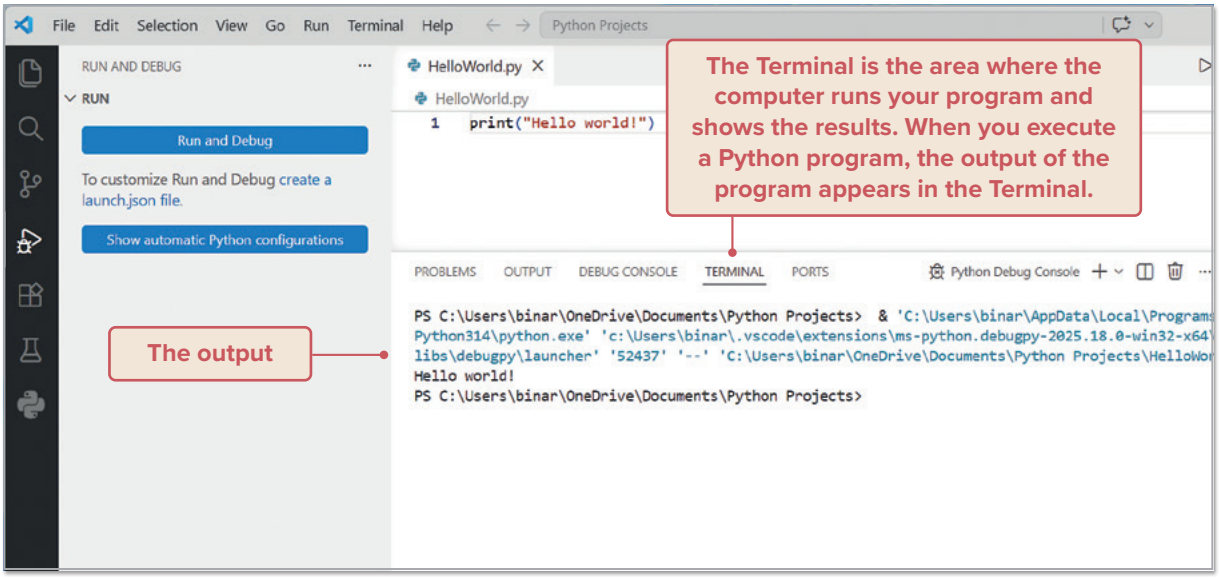
Now you are ready to run your first program in VS Code.

To run the program:

> Click the **Run and Debug** icon. 1
> Click the **Run and Debug** button. 2
> The first time you run a Python program, you need to install the **Python Debugger** extension. 3
> Select **Python File** from the drop-down list. 4







If your Python program does not run in Visual Studio Code, you can ask an AI assistant for help suggesting possible solutions. You can use a prompt like:
"My Python program won't run in VS Code. How do I fix it?"

Print

The simplest statement you can write in Python displays text on the screen. In Python, the function you use to display something on the screen is the **print()** function. When you want to display text, you have to put it inside quotes.

```
print("I use the print function!")
```

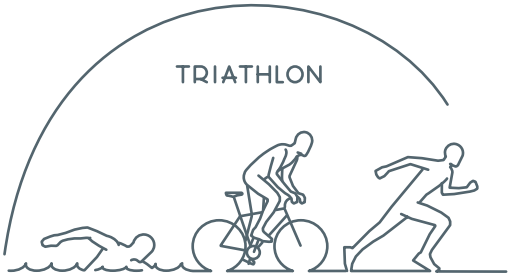
```
I use the print function!
```

The result is displayed on the screen.

Anything inside quotation marks is treated as text and printed exactly as written.

Triathlon game

In this unit, you are going to program a triathlon game. The program asks for the name of the athlete and the athlete's score in the three triathlon sports. It calculates the final score of the athlete and the athlete's best performance. To write our program, we will need to learn about variables and comments, but we can already write the first line of our program, which displays the name of the game. So let's start!




```
print("Triathlon Game")
```

Variables

A **variable** is a name used to store information in a program. You can think of a variable as a labeled container where a program keeps data such as numbers or text. The value stored in a variable can change while the program is running. In Python, you assign a value to a variable using the assignment operator (=).

General form of a variable

variable name = content

Name of the variable

Content of the variable

Clear variable names make programs easier to test and maintain.

Let's create a variable called "age" that stores the value 15. Later in the program, the value of the variable can change.

```
age = 15
```

The two main categories of variables are numbers and text. Python supports two types of numbers, integers and floating point (decimal) numbers.

Numbers (numeric variables)

```
level=3
score=1200
TotalAmount=120.50
```

Text (string variables)

```
Message="Do you want to play again Y/N?"
MyName="John"
EmailAddress="john@binary-academy.com"
```

Discussion Questions

- Why is a variable useful in a program?
- What could happen if a programmer uses unclear variable names?

Variable names

A variable can have a short name (like x or y) or a more descriptive name (like age, carname, total_volume, etc.). When naming a variable in Python, there are certain rules that must be followed.

A variable's name:

- Must start with a letter or the underscore character (_)
- Cannot start with a number
- Can only contain alpha-numeric characters and underscores (A-Z, a-z, 0-9, and _)
- Is case-sensitive (age, Age, and AGE are three different variables)
- Should be descriptive. It is more effective to give variables names that represent their content and purpose, so you can easily understand what each variable represents when reading your code.

Some names cannot be used because they are special Python keywords, such as if, return, while, else, True, False, None, and import.

3.2 Assigning a value to a variable

The equals sign (=) is used to assign a value to a variable. The number, text or variable to the right of the equals sign is assigned to the variable on the left. You can also put a calculation to the right of the equals sign, and the result is assigned to the variable. For example:

```
x=15
y=20
Total=x+y
print(Total)
```

To print the value of a variable, you use the function **print()** and the name of the variable without quotes.

```
35
```

In coding, the equals sign (=) is not used like in mathematics. For example, x=15 means that you take the value 15 as a number and assign it to the variable named x.

Triathlon game

Let's continue the triathlon game.

```
print("Triathlon Game")
SwimmingScore=70
CyclingScore=40
RunningScore=60
```

The program follows instructions in order, from top to bottom.

String variables

Variables that store text are called string variables. To assign text to a variable, place the text inside quotation marks.

```
name="John"
print(name)
```

```
John
```

To make the output clearer, we can change our code to:

```
name="John"
print("My name is:",name)
```

```
My name is: John
```

Can you see the difference? Now it's clearer.

Comments

Comments are notes in your code that explain its functionality and help you or others understand it later. They are especially useful when you need to change your program in the future. In Python, you create a comment by placing a hash symbol (#) at the beginning of a line. Comments are ignored by the computer but are very important for human readers.

```
#assigns a value to the variable name
name="John"
#prints the value of the variable
print("My name is: ", name)
```

This is a comment.

```
My name is: John
```

The computer ignores comments because they are intended for human readers to understand and work with the code more effectively.

Now, let's add some comments to our triathlon game so that we can understand the code more easily if we want to change it later.

Triathlon game

```
print("Triathlon Game")
SwimmingScore=70
CyclingScore=40
RunningScore=60
#Calculate athlete's score in the triathlon
TriathlonScore=SwimmingScore+CyclingScore+RunningScore
print("The total triathlon score is:",TriathlonScore)
```

```
Triathlon Game
The total triathlon score is: 170
```

Career connections

Developers, data analysts, and automation specialists use variables and input data to build systems that respond to user actions and process information.

Discussion Questions

- When printing text and variables together (for example, "My name is:" , name), why is the text in quotation marks but not the variable name?
- How can clear variable names and comments help someone else understand your code?

Exercises

Answer each set of questions in your notebook.

1

Run the following programs. What happens in each case? Why do you think the second program behaves differently?

```
a=5
a=a+2
print(a)
```

```
a="5"
a=a+2
print(a)
```

2

The following code has some errors. Find and correct them.

```
*This program prints a person's name and age
Name="John"
print(Name, "is", age, years old)
age=15
```

3

Read the following program. Before running the program, predict the output.

```
x=10
y=3
z=x+y
print(z)
```

- Then run the code and check your answer.
- What value is stored in the variable z?
- How does Python calculate the result?

4

Create a Python program that calculates the total grade for three subjects: Mathematics, Science, and Literature. Use appropriate variable names for each subject's grade, perform the calculation, and display the total grade in a clear and descriptive message.

5 Run the following program:

```
name = "Alex"
city = "Rome"
print(name)
print(city)
```

Modify the program so that it prints the following message: "Alex lives in Rome."

6 Write a Python program that stores the following information in variables:

- A person's name
- Their favorite sport

Then display a message that combines the two pieces of information.

For example: "Alex likes basketball."

LESSON 3

Input, Data Types, and Operators

In the previous lesson, you learned how to store values in variables and display results using statements. Until now, the values in our programs were written directly in the code. However, programs often need information from the user. In this lesson, you will learn how to write a program that receives input from the user and uses it in calculations.

Python uses the **input()** function to receive input from the user. When the **input()** function is used, the program stops and waits for the user to enter the data.

For example, in the following program, the user is asked to enter a value for the variable x. When the user enters the value 10 and presses Enter ↵, the value 10 is assigned to the variable x.

```
print("Please enter a value for x: ")
x=input()
print("The value of x is: ",x)

Please enter a value for x:
10
The value of x is: 10
```

The program asks the user for a value for the variable x.

The user enters the value 10 and presses Enter ↵.

The value 10 is assigned to the variable x.

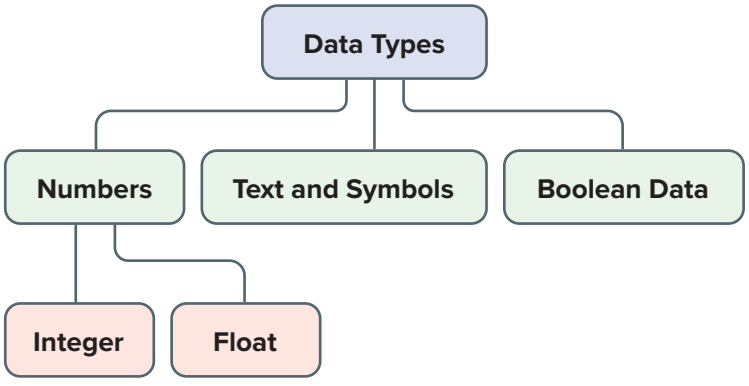
Data Types

A **data type** describes what kind of value a piece of data holds. It defines the kinds of values the data can have and the operations that can be performed on it. Python provides some built-in data types. The basic data types are:

- Numbers
- Text and symbols
- Boolean data

Numbers can be:

- Integer numbers
- Floating-point numbers



Every value stored in a variable has a data type. The data type determines what operations can be performed on that value.

Examples of data types

Data type	Python name	Example
Integers	int	12, -999, 0, 90000
Floating-point numbers	float	4.5, 0.003, -90.5, 3.0
Text	str	"a", "John", "$$$"
Boolean	bool	True, False

Boolean variables can take only two values, True or False.

To use numbers input by the user, you need to use the functions:

> `int(input())` for integers
> `float(input())` for floating-point numbers

Let's look at an example.

```
print("Please enter a value for x: ")
x=int(input())
print("Please enter a value for y: ")
y=int(input())
Total=x+y
print("The sum of x and y is: ", Total)
```

```
Please enter a value for x:
10
Please enter a value for y:
5
The sum of x and y is: 15
```

By default, the input() function stores the value as text (a string). If you want to perform calculations, you must convert the input to a number using:

- **int() for integers**
- **float() for decimal numbers**

The **input()** function can also use a string as a message that will help the user understand what kind of data has to be entered. The string "Please enter a value for a:" is an optional argument of the input() function.

```
a=int(input("Please enter a value for a:"))
print(a)
```

Triathlon game

Now you know how to ask the player of the game for the athlete's name. You can either use:

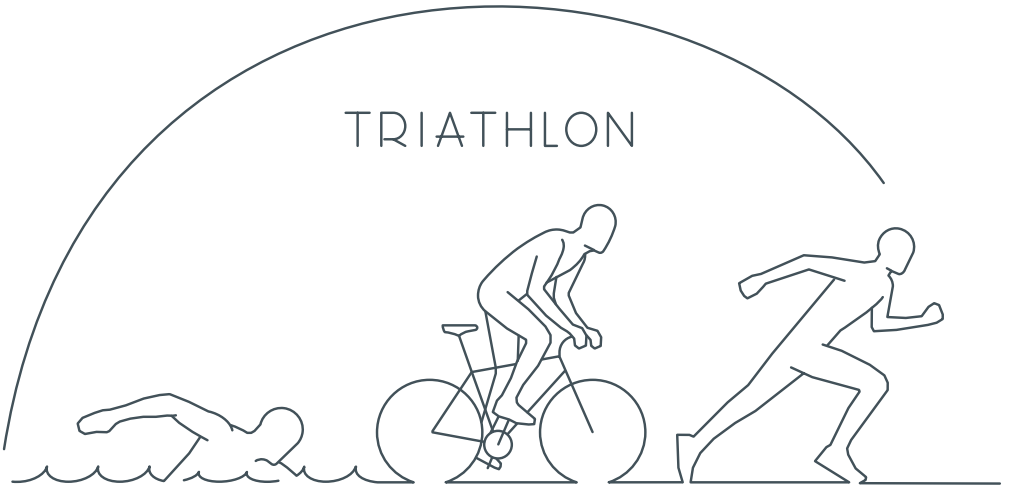
```
print("Type the name of the athlete: ")
AthleteName=input()
```

or:

```
AthleteName=input("Type the name of the athlete: ")
```

Let's see how the program is developed. The program should ask for the name of the athlete and the athlete's score in the three sports.

```
print("Triathlon Game")
# We read athlete's name
print("Type the name of the athlete: ")
AthleteName=input()
# We read athlete's number
AthleteNumber=int(input("Type the number of the athlete: "))
# We read athlete's performance in the 3 sports
SwimmingScore=float(input("Type Swimming Score: "))
CyclingScore=float(input("Type Cycling Score: "))
RunningScore=float(input("Type Running Score: "))
```



Score: Score: Score:

Operators in Python

You now have all the input data needed for our triathlon game, but to calculate the total score, you need to learn how to use operators. An **operator** in a programming language is a symbol used to perform a specific operation on variables and values.

Python offers different classes of operators. The four most frequently used classes of operators in Python are:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Boolean operators

Operators

Operator class	Operators	Description

Arithmetic operators	`+`
	`-`
	`*`
	`/`
	`**`
		Used to perform arithmetic operations: addition, subtraction, multiplication, division, etc.

Assignment operators	`=`
	`+=`
	`-=`
	`*=`
	`/=`
		Used to assign values to variables.

Comparison operators	`>`
	`<`
	`==`
	`<=`
	`>=`
	`!=`
		Used to compare values while writing conditional statements.

Boolean operators	`and`
	`or`
	`not`
		Used to check more than one condition in a single conditional clause, and also used to check the opposite of a condition. These operators allow programs to make decisions based on complex logical expressions.

In this lesson, we will explore the arithmetic and assignment operators, as these are the operator classes we need for our program.

Arithmetic Operators

You can use Python to perform all kinds of calculations, such as addition, subtraction, multiplication, and division. In coding, you write these calculations a little differently than you do in mathematics. You use the following symbols to represent the basic operations:

Arithmetic operators	
+	addition
-	subtraction
*	multiplication
/	division
**	exponent
%	modulo
//	floor division

Math	Python coding
4 + 2	4 + 2
4 - 2	4 - 2
4 x 2	4 * 2
4 ÷ 2	4 / 2
x^2	x ** 2
10 | 4 2 | 2	10 % 4
	10//4

The following examples demonstrate the use of arithmetic operators:

```
x=5+2
print(x)
```

```
7
```

```
x=6*3
print(x)
```

```
18
```

```
x=5**2
print(x)
```

```
25
```

```
x=6/3
print(x)
```

```
2.0
```

```
x=10%6
print(x)
```

```
4
```

```
x=10//6
print(x)
```

```
1
```

Just like in math, in Python the order of the operations is important. The rules you encountered in Microsoft Excel for the use of parentheses apply here, too.

Multiplication and division are calculated before addition and subtraction. This means that the output of 4 + 2 * 5 is 14, not 30. You can use parentheses to define a different sequence of calculations.

```
print(4 + 2 * 5)
print((4 + 2) * 5)
```

```
14
30
```

The order of precedence of the arithmetic operators is as follows:

1. ( ) parentheses
2. ** exponentiation
3. * / multiplication and division
4. + - addition and subtraction

Operators that are at the same level of precedence are performed in order from left to right.

Discussion Questions

- Imagine a program asks the user to enter their age. Why might it be important to check whether the input is actually a number? What problems could happen if the user types something unexpected?
- Sometimes two programs produce different answers for the same calculation because of the use of parentheses or operator order. Why do you think programming languages have strict rules about the order of operations?

Triathlon game

Let's use what you've learned to make the calculations in the program. You want to display the points an athlete has earned on the screen. In the triathlon, the average score is multiplied by the event's coefficient, which is 0.6.

```
print("Triathlon Game")
# We read athlete's name
print("Type the name of the athlete: ")
AthleteName=input()
# We read athlete's number
AthleteNumber=int(input("Type the number of the athlete: "))
# We read athlete's performance in the 3 sports
SwimmingScore=float(input("Type Swimming Score: "))
CyclingScore=float(input("Type Cycling Score: "))
RunningScore=float(input("Type Running Score: "))
# The coefficient in triathlon is 0.6
TriathlonPoints=0.6*(SwimmingScore+CyclingScore+RunningScore)/3
print("Your points in triathlon: ",TriathlonPoints)
```

Assignment Operators

You have learned how to use arithmetic operators, such as + and -, but your program also uses =, which is an **assignment operator**. Assignment operators in Python are used to assign values to variables.

Assignment operators	
=	Assignment operator
+=	Addition assignment
-=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment

Math	Python coding
s =x + y	
x=x+y	x += y
x=x-y	x -= y
x=x*y	x *= y
x=x/y	x /= y

The following examples demonstrate the use of assignment operators:

```
x=6
x+=3
print(x)
```

```
9
```

```
x=6
x-=3
print(x)
```

```
3
```

```
x=6
x*=3
print(x)
```

```
18
```

```
x=6
x/=3
print(x)
```

```
2.0
```

Using AI to Generate Code

Developers often use AI tools to explain errors, suggest starter code, generate test ideas, and improve documentation. However, programmers must still check that the code is correct, readable, and secure.

Before using AI, always think about:

- What inputs does my program need?
- What calculations should it perform?
- What should it display?

In the following example, you ask the AI assistant to generate code for a Python program that asks the user to enter the number of hours they worked and their hourly pay rate. The program then calculates the total salary. If the user worked more than 40 hours, the program adds a bonus of $100. Finally, it displays the final salary.

Before using this prompt: write your own version of this program first. After AI generates its response, compare the two approaches. Test both on at least two inputs, including one edge case.

AI Tool

The prompt:

Create a Python program that asks the user to enter the number of hours worked and the hourly pay rate.

Calculate the total salary by multiplying hours by the rate.

If the hours worked are greater than 40, add a bonus of 100 dollars to the salary.

Print the final salary with a clear message.

The output:

```
hours = float(input("Enter the number of hours worked: "))
rate = float(input("Enter the hourly pay rate: "))

salary = hours * rate

if hours > 40:
    salary += 100

print("The final salary is:", salary, "dollars.")
```

See the AI References and Resources section for details on this AI generated content.

You can now save the program and run it to see how it works.

After running the code, always double-check the results and review the logic carefully. AI assistants can help you write programs faster, but they may also make mistakes or misunderstand the requirements. It is your responsibility to test the program, verify the calculations, and make sure the code behaves as expected.

Discussion Questions

- If a program asks users to enter numbers for a calculation, what strategies could a programmer use to reduce errors caused by incorrect input?
- What do you think is the best way to use AI as a learning tool without depending on it too much?

Exercises

Answer each set of questions in your notebook.

1

Convert these mathematical expressions into Python code.

1. ax^2+bx+c

2. $2x-3(x-ac3/5ac5)$

3. $x=a^{3x+2}+\frac{x+1}{x^3-2}$

4. $x=-b/a$

5. $x=a^2b^n+4k$

2

Explain why input() returns text by default and why int() or float() may be needed.

3

What will be the output of the following program if the input value of a is 8 and b is 4?

```
a=int(input("Type a value for a: "))
b=int(input("Type a value for b: "))
z=(a+b)/2
print("z=  ",z)
a=(a-z)**2
b=(b+z)**2
z=a+b/2
print("a= ",a,"b= ",b,"z= ",z)
```

4

Examine the following expressions:

```
print(10 + 5 * 2)
print((10 + 5) * 2)
```

- Why are the results different?
- How do parentheses change the calculation?

5

You want to create a program that calculates the average score for four exams.

Before writing any code, consider:

- What inputs will the program need?
- What calculations must the program perform?
- What output should the program display?

Write a short algorithm describing the steps.

After completing your program, use an AI tool to improve or test your solution.

6

Consider the following program:

```
age = int(input("Enter your age: "))
print("Next year you will be", age + 1)
```

- What might happen if the user enters text instead of a number?
- How could a programmer improve the program to handle this situation?

LESSON 4

Conditional Statements

In the previous lesson, you learned how to store data and receive input from the user. In this lesson, you will learn how to make decisions using that data. Programs often need to make choices based on different situations. To do this, programmers use conditional statements, which allow a program to choose different actions depending on a condition. This is an important skill, and it allows programs to respond in different ways depending on the user's input.

For example, a program can give different answers based on what the user types. This makes programs more interactive and useful for the user.

Comparison Operators in Python

In programming, we use **comparison operators** to make decisions. Comparison operators compare values and return a result of True or False. The following table provides a list of the comparison operators available in Python.

Two important types of operators in Python are comparison operators and Boolean (logical) operators. These operators are like building blocks that help us compare values and check conditions in a program.

Let's explore how they work before using them in conditional statements.

"=" means assign a value.

"==" means compare two values.

Operator	Meaning
>	Greater than
<	Less than
==	Equal to
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal to

Here are some examples of the use of comparison operators in which two values are compared, and the Python program returns the Boolean result: True or False.

```
x=5
y=6
k=x<y
print(k)
```

```
True
```

```
x=5
y=6
z=x==y
print(z)
```

```
False
```

```
x=5
y=5
m=x-y<=0
print(m)
```

```
True
```

```
x=5
y=6
n=x+y!=15
print(n)
```

```
True
```

If you are not sure whether a comparison condition is correct, test it with simple values and then ask AI for help if needed.

Boolean Operators in Python

Boolean operators are used in conditions and conditional statements. The main operators are **and**, **or**, and **not**, and they are used to allow programs to make decisions based on True or False values. The meanings of these operators are shown in the table below, and the truth table further illustrates all possible input combinations for a Boolean operation and the resulting output.

Operator	Meaning
and	Returns True if both statements are True.
or	Returns True if at least one of the statements is True.
not	Returns the opposite Boolean value: False if the result is True, and True if the result is False.

Truth table

AND			OR			NOT	
A	B	A **and** B	A	B	A **or** B	A	**not** A
True	True	True	True	True	True	True	False
True	False	False	True	False	True	False	True
False	True	False	False	True	True		
False	False	False	False	False	False		

Operator precedence in programming			
1	()	5	== > < <= >= !=
2	**	6	not
3	* /	7	and
4	+ -	8	or

The order of precedence of operators in programming starts with the arithmetic operators, then comparison operators, and finally the Boolean operators.

3.4

Before learning how Boolean operators work in code, let's look at a real-life situation.

Imagine a school club has rules for joining. A student can join the club only if:

- their school attendance is above 90%, and
- their grade is above 80.

This means both conditions must be true for the student to join. For example:

- If a student has 95% attendance and a grade of 85: they can join.
- If a student has 92% attendance and a grade of 70: they cannot join.
- If a student has 85% attendance and a grade of 90: they cannot join.

This situation uses the AND condition, where both requirements must be satisfied. In Python, we use the "and" operator to represent this type of decision.

The following examples show how Boolean operators are used in Python.

```
x=5
y=6
k=x<10 and y<8
print(k)
```

```
True
```

```
x=True
y=False
z=x==y
print(z)
```

```
False
```

```
x=True
y=False
m=(x or y) and (not False)
print(m)
```

```
True
```

```
x=5
y=6
n=x>y and (not y==6)
print(n)
```

```
False
```

Discussion Questions

- What might happen if a programmer uses "=" instead of "==" in a condition?
- Can you think of a situation where using "or" would be better than "and"?

Types of Conditional Statements

To make a decision in Python, you use the **if statement**. There are three ways to express the **if** statement.

if
statement

```
if condition:
    statement
```

if... else
statement

```
if condition:
    statement
else:
    statement
```

if... elif... else
statement

```
if condition:
    statement
elif:
    statement
else:
    statement
```

Note that the colon (:) following the expression is required.

Use **if** when you only need to do something if a single condition is true. Use **if...else** when there are two possible outcomes (one if true, another if false). Use **if...elif...else** when you have three or more possible outcomes to handle.

The Simple If Statement

In this lesson, you will use the simple **if** statement.

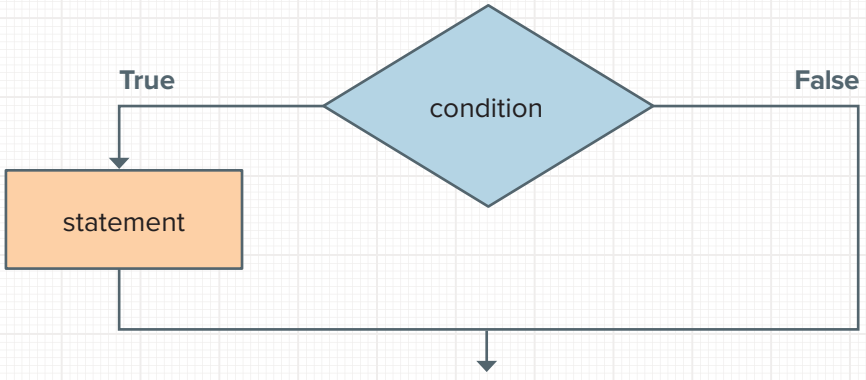
- If the condition is True, the statement(s) that follow the **if** statement will be executed.
- If the condition is False, the statement(s) will not be executed.

Python uses indentation to indicate the statements that depend on the condition.

if
statement

```
if condition:
    statement
```

The flowchart of the **if** statement



3.4 Indentation

Indentation is important in Python because it tells the program which statements depend on a condition.

- Everything under **if**, **elif**, or **else** must be indented, the same amount.
- The colon (:) at the end of a conditional line shows which indented lines belong to that condition.
- Lines that are not indented, and whose execution doesn't depend on the condition, will run even if the condition isn't met.

If you don't indent correctly after a conditional statement, Python will give an error.

```
grade=int(input("Type the score"))
if grade>=18:
print("Excellent")
```

Error message

```
IndentationError: expected an indented block after 'if' statement
```

If you get an IndentationError message, you can ask an AI assistant to help you fix it. The AI can point out where indentation is missing and show you the correct structure.

```
grade=19
if grade>=18:
   print("Excellent")
print("Continue to develop your skills")
```

The second **print** statement is not part of the **if** statement, so it is executed regardless of the result of the **if** condition.

```
Excellent
Continue to develop your skills
```

Let's try the following example:

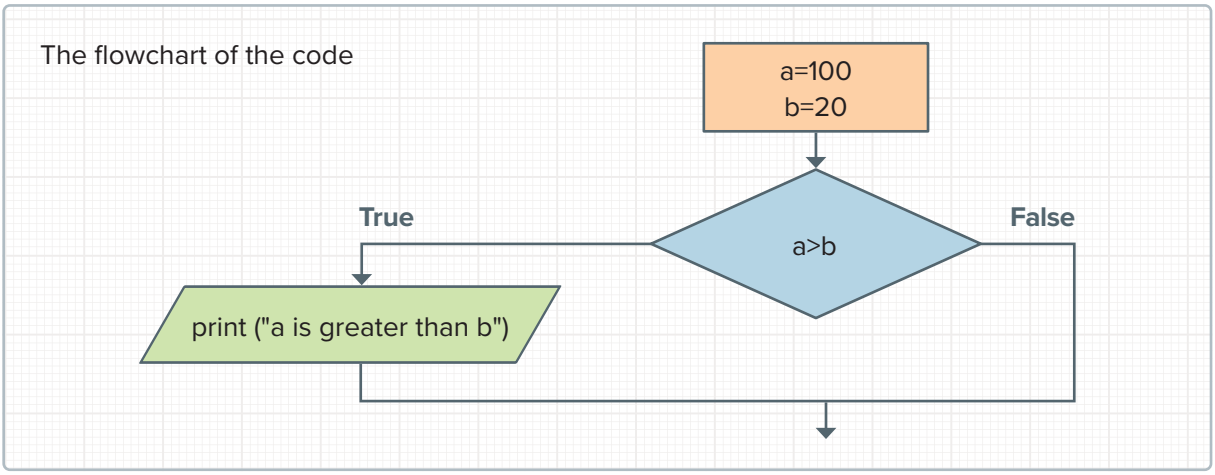
```
a = 100
b = 20
if a > b:
   print("a is greater than b")
```

The condition

The statement

```
a is greater than b
```

Code that uses clever shortcuts, very short variable names, or compressed logic that is hard to understand at first glance is sometimes called hard-to-read code. AI tools often produce this type of code. They make code appear correct and concise. However, the code can be difficult for another person (or even for you later) to understand or fix.



Here's another example of a simple **if** statement.

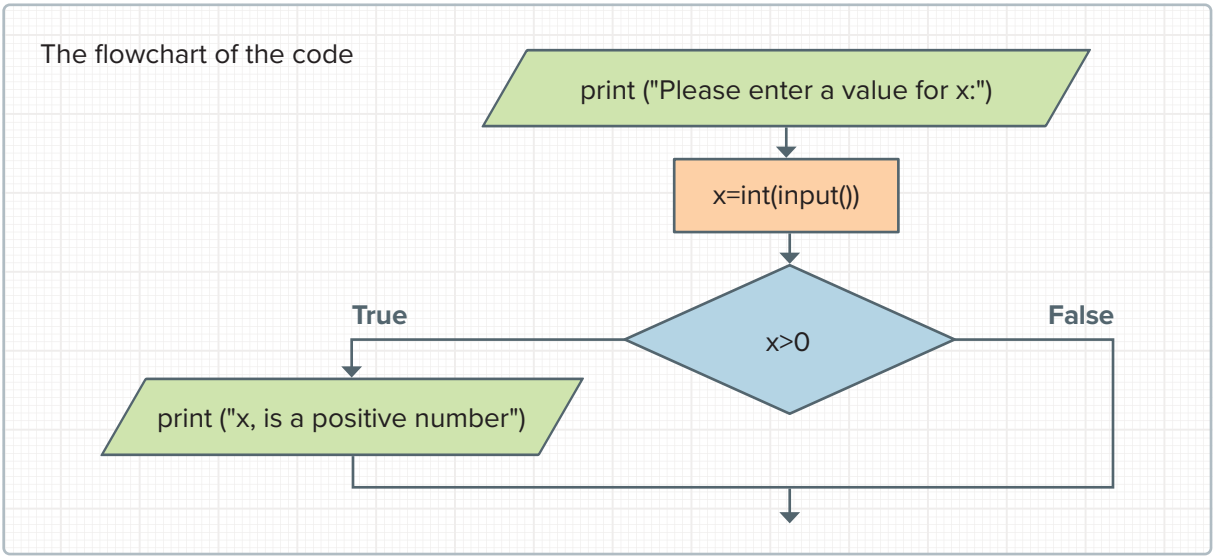
```
print("Please enter a value for x: ")
x=int(input())
if x>=0:
   print(x,"is a positive number")
```

If the condition is False, Python skips the indented line and continues with the rest of the program.

```
Please enter a value for x:
5
5 is a positive number
```

If your program does not print what you expected and you cannot find the problem by inspecting your code carefully and testing different possibilities, you can copy your code and ask an AI assistant "Why is this if statement not working as I expected?"

AI can help you check your condition, your indentation, and your use of comparison operators. When you review AI-generated code, readability is part of the quality check, not something to add later.



Discussion Questions

- What problems might happen if the condition is written incorrectly?
- Why is consistency important in indentation?

Career connections

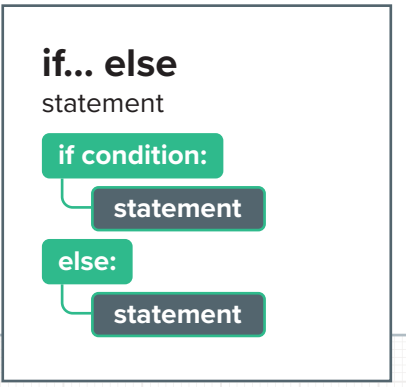
Conditional logic for decision-making is used in many jobs, including software development, data science, cybersecurity, and business systems design.

Understanding how to apply logic in code prepares you for careers that rely on data-driven decision-making.

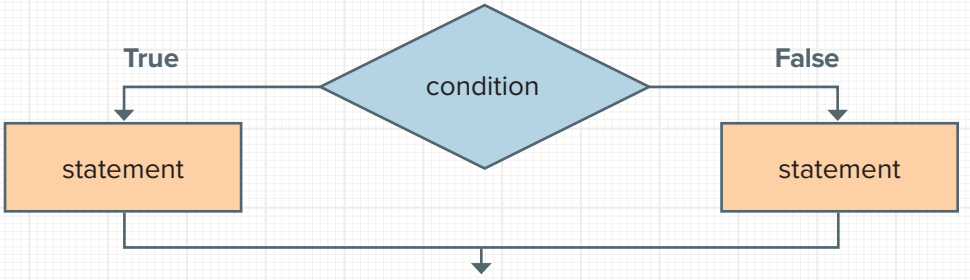
The If...Else Statement

In an **if...else statement**, if the condition is True, the statement(s) that follow the **if** statement will be executed. If the condition is False, the statement(s) that are under the **else** will be executed.

As with simple **if** statements, indentation is used to indicate the statements that will be executed in each case.



The flowchart of the **if...else** statement



It's time to explore a couple of examples.

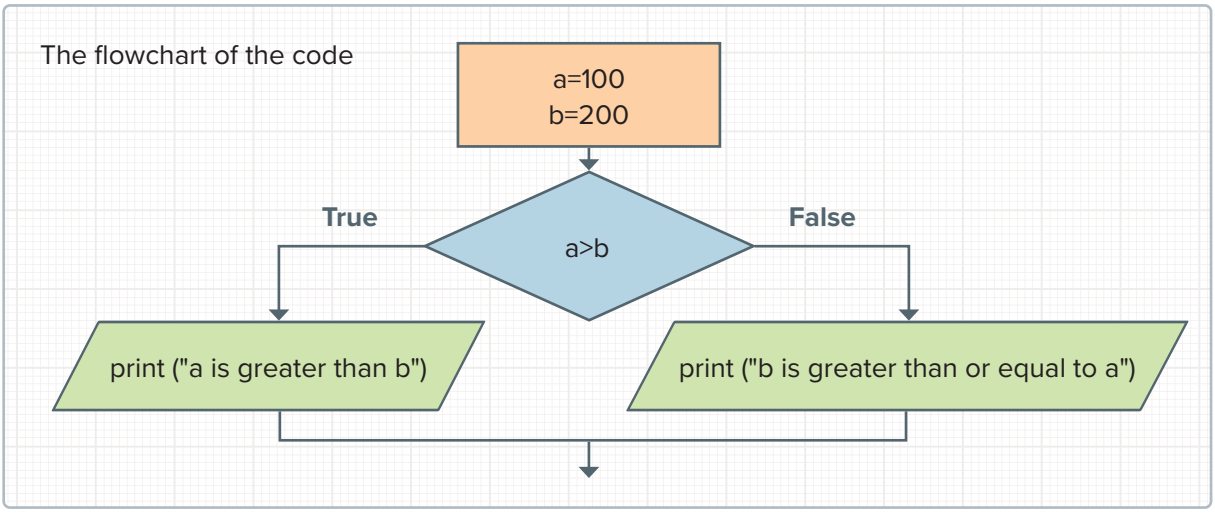
```
a = 100
b = 200
if a > b:
   print("a is greater than b")
else:
   print("b is greater than or equal to a")
```

The condition

if statement

else statement

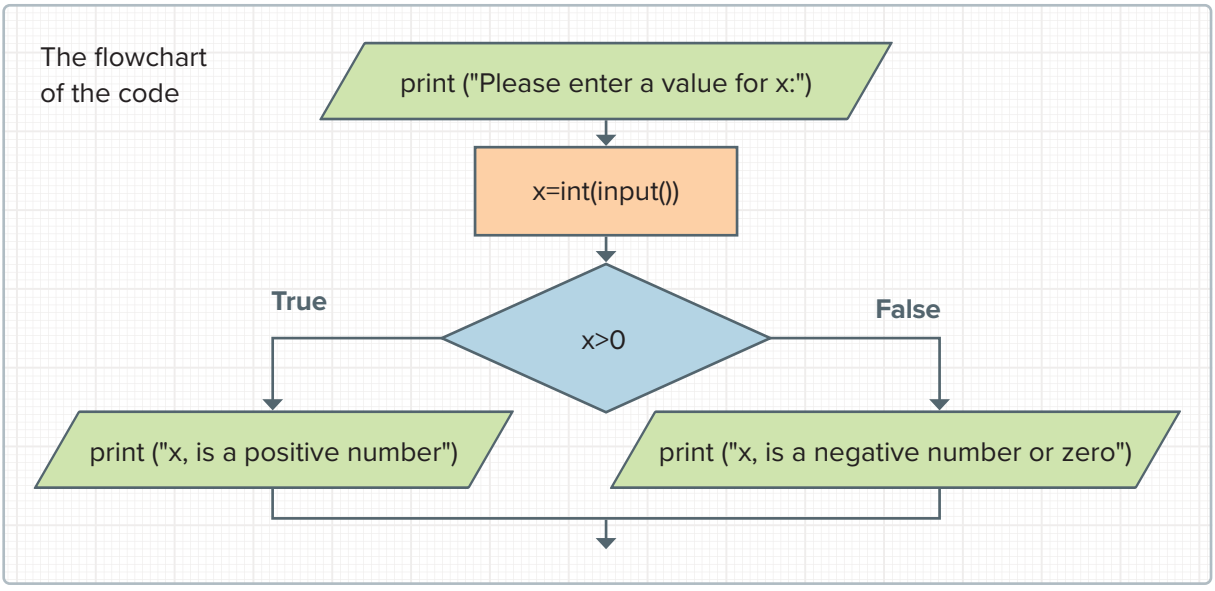
```
b is greater than or equal to a
```




```
print("Please enter a value for x: ")
x=int(input())
if x>=0:
   print(x,"is a positive number")
else:
   print(x,"is a negative number or zero")
```

In an if...else statement, only one part of the code will execute. If the condition is True, the if block runs. Otherwise, the else block runs.

```
Please enter a value for x:
-2
-2 is a negative number or zero
```



Discussion Questions

- Can you think of a real-life situation with only two possible outcomes?
- What happens if a condition is always True?

3.4

The If...Elif...Else Statement

In the previous examples of conditional statements, the program had to choose between two options.

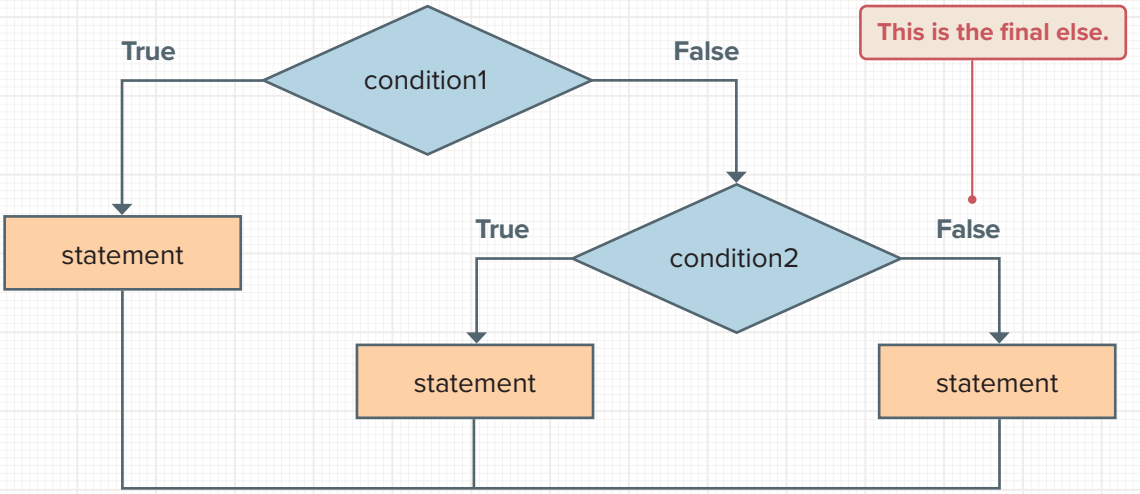
An **if...elif...else** structure is used when a program must choose among multiple possible outcomes. The **if** statements are executed from the top down.

The program checks the conditions one by one, and if one condition is True, the statement under that condition will be executed. The rest of the statements will be bypassed. If neither of the conditions is True, then the final **else** statement will be executed.

if... elif... else
statement

```
if condition:
    statement
elif:
    statement
else:
    statement
```

The flowchart of the **if...elif...else** statement

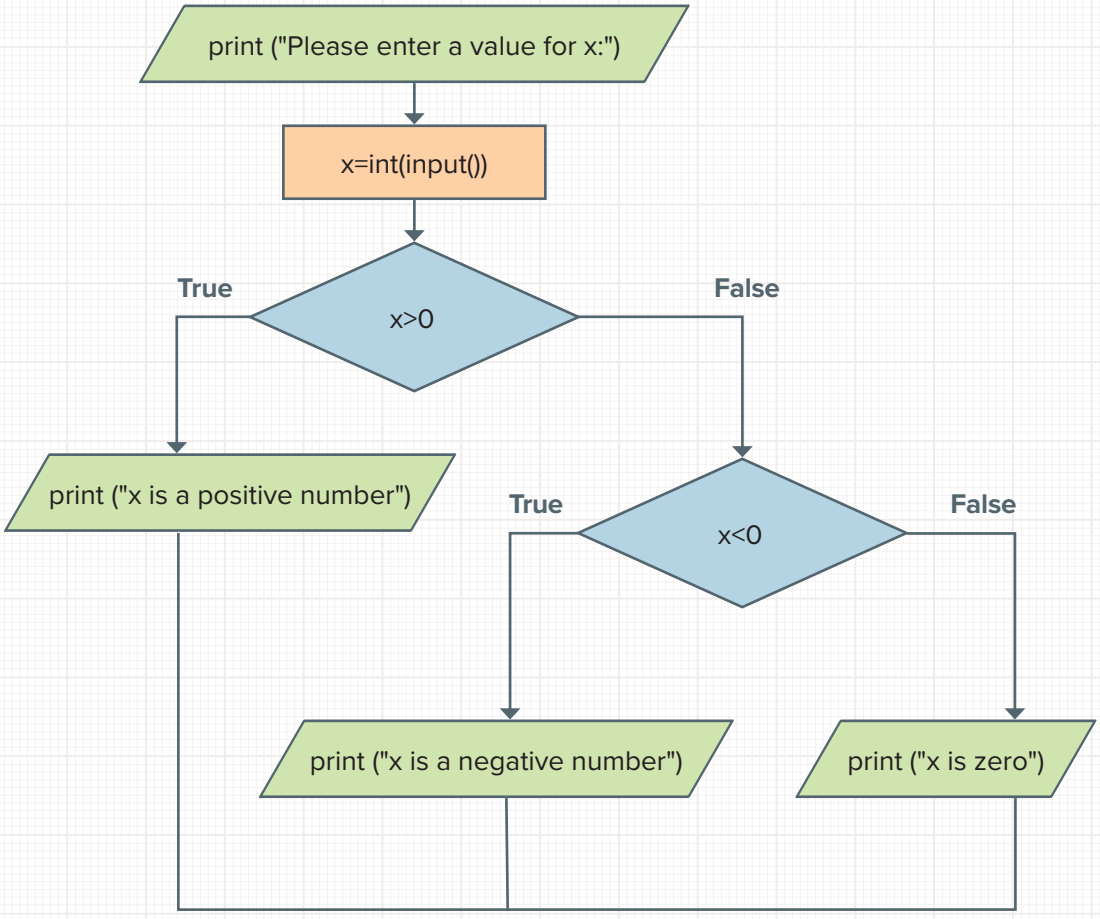


Here are some examples of **if...elif...else** statements.

```
print("Please enter a value for x: ")
x=int(input())
if x>0:
   print("x is a positive number")
elif x<0:
   print("x is a negative number")
else:
   print("x is zero")
```

```
Please enter a value for x:
-45
x is a negative number
```

The flowchart of the code



The following program asks the user to enter a grade and checks if it is valid (between 0 and 10). If the grade is valid, it uses conditional statements to display a message based on the score.

Student grades

```
print("Please enter a grade: ")
g=int(input())
if g<0 or g>10:
   print("Invalid grade")
elif g>=8:
   print("Excellent!")
elif g>=5:
   print("Very good!")
else:
   print("Try harder!")
```

```
Please enter a grade:
12
Invalid grade
```

Nested If Statements

Sometimes, a decision depends on multiple conditions that need to be checked in sequence. In such cases, a nested **if** statement is used. A **nested if statement** is an **if** statement inside another **if** statement. It lets you check one condition first, and if that condition is true, you can check another condition inside it.

This is useful when you need to make a decision based on multiple levels of conditions. Any number of statements in any combination can be nested inside one another.

Nested if...
statement

```
if condition:
    statement
    if condition 2:
        statement
    else:
        statement
elif condition 3:
    statement
    if condition 4:
        statement
else:
    statement
```

The only way to make sense of the nesting is the indentation. Each level of indentation represents a new condition inside another condition.

- Nested if = an if inside another if (runs only when the outer if is True).
- elif = "else if" (a new condition checked when the previous if condition was False).
- else = runs when all conditions above were False.

Student grades

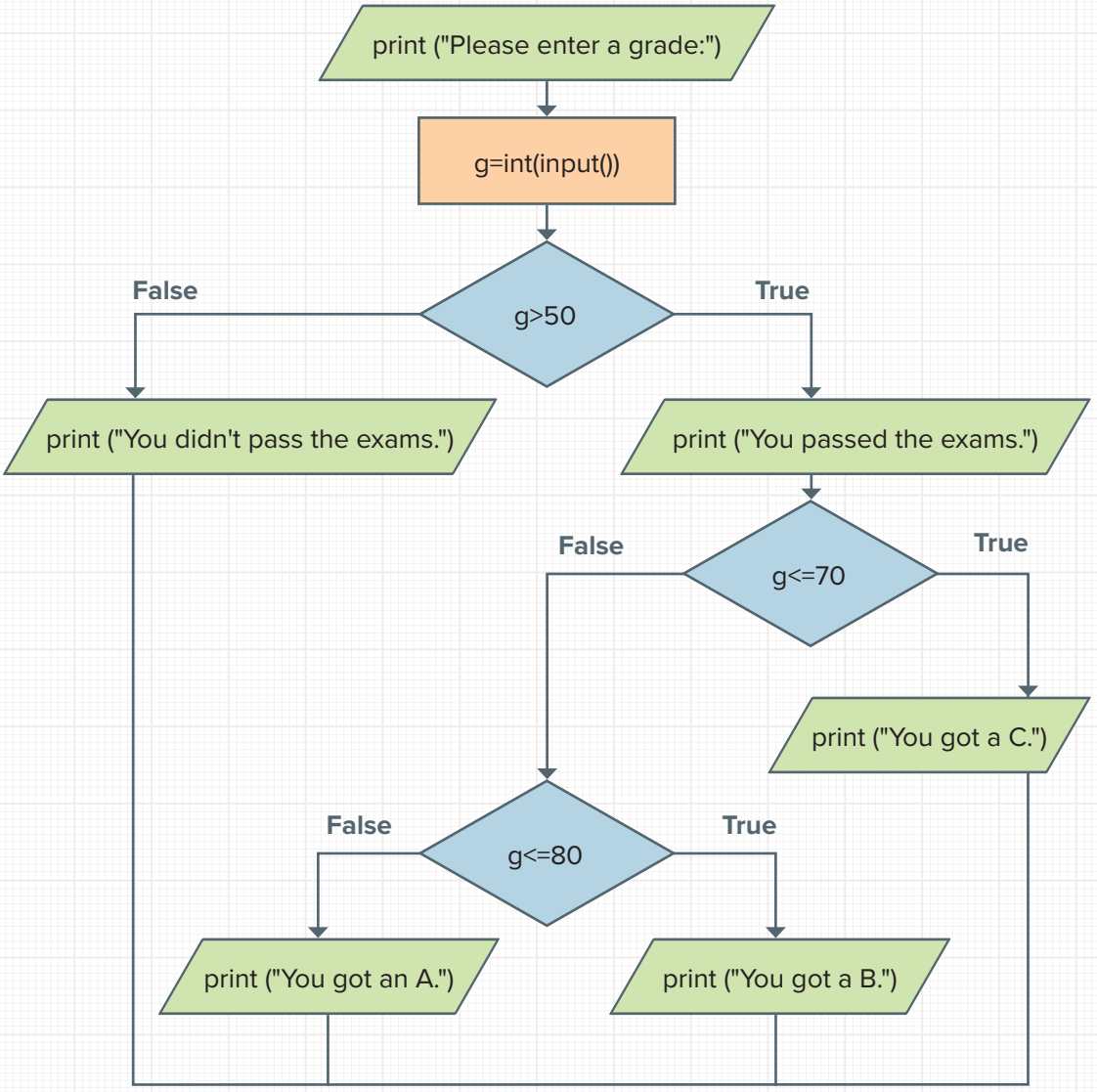
The following program calculates a student's letter grade. It uses **nested if** statements to tell a student if they passed their exams and what grade they got.

```
print("Please enter a grade: ")
g=int(input())
if g>50:
   print("You passed the exams.")
   if g<=70:
      print("You got a C.")
   elif g<=80:
      print("You got a B.")
   else:
      print("You got an A.")
else:
   print("You didn't pass the exams.")
```

We could achieve the same result using an **if...elif...else** statement.

Nested conditions can be difficult to understand. If you think your code is too complex, you can ask an AI assistant to rewrite it more simply. It may be able to suggest a simpler structure to help improve the readability of your code.

A flowchart of the **nested if** statements



Discussion Questions

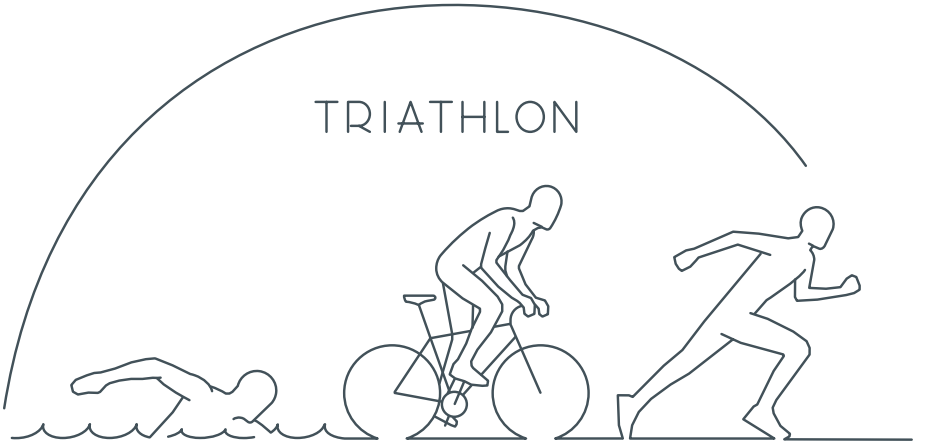
- Why must conditions be checked in order?
- What could happen if the order of conditions is wrong?
- How can you design conditions so that no values are missed?
- Why can nested conditions become difficult to understand?
- How does indentation help you understand nested logic?
- Should programmers rely on AI for decision logic? Why or why not?

Triathlon game

To improve the Triathlon game program, you will extend the simple **if** statement by using an **if...elif...else** structure. These new lines allow the program to evaluate multiple score ranges instead of checking only one condition. Now, the program can classify the athlete's performance into four different levels.

We could use **nested if** statements to write this code, but because the athlete can only have one performance level (outstanding, excellent, or good) this is not necessary. In this case, **nested if** statements would make the code harder to read and add unnecessary indentation.

```
print("Triathlon Game")
# We read athlete's name
print("Type the name of the athlete: ")
AthleteName=input()
# We read athlete's number
AthleteNumber=int(input("Type the number of the athlete: "))
# We read athlete's performance in the 3 sports
SwimmingScore=float(input("Type Swimming Score: "))
CyclingScore=float(input("Type Cycling Score: "))
RunningScore=float(input("Type Running Score: "))
# The coefficient in triathlon is 0.6
TriathlonPoints=0.6*(SwimmingScore+CyclingScore+RunningScore)/3
 print("Your points in triathlon: ",TriathlonPoints)
if TriathlonPoints >= 60:
    print("Outstanding athlete")
elif TriathlonPoints >= 50:
    print("Excellent performance")
elif TriathlonPoints >= 40:
    print("Good effort")
else:
    print("Needs improvement. Keep training!")
```



Score: Score: Score:

Exercises

Answer each set of questions in your notebook.

1 What will the following program display for each input value?

A. 25

B. 16

C. 7

```
age=int(input("Enter your age: "))
if age>=18:
    print("adult")
else:
    if age>=13:
        print("teenager")
    else:
        print("child")
```

2 Why is the order of conditions important in an if...elif...else structure? Give an example.

3 Why can a badly ordered if...elif...else structure produce incorrect results?

4 Rewrite the following program using an if...elif...else structure.

```
grade = int(input())

if grade >= 50:
    if grade >= 80:
        print("A")
    else:
        print("B")
else:
    print("Fail")
```

5 A student can join the robotics club if their math grade is greater than 80 and their science grade is between 70 and 90. The maximum grade is 100.

1. Write a program that reads the math and science grades of a student and displays whether they can join the robotics club.
2. Create the flowchart of the program.

Exercises

6 Write an example of a nested if statement. Then rewrite your statement using only if... elif...else. Compare the readability of your two programs.

7 Create a program that asks the user to enter two numbers and one of the arithmetic operators (+, -, *, /).

The program must make the necessary calculations based on the operator the user enters. Pay special attention when using the division operator.

8 A shop offers a discount based on the total price. If the total is $100 or more, the customer receives a 10% discount. Otherwise, no discount is given.

Write a Python program that:

- Asks the user for the total price.
- Uses an if...else statement.
- Prints whether the discount is applied.

If the program does not behave as expected, review your condition and test again.

Project

Payment planner

You will create a program to calculate the payments for buying a laptop and its accessories (a bag and a keyboard). The program will take the prices of the items and calculate the total cost, how much you need to pay upfront (30% of the total), and the monthly payments for the remaining amount. It also checks if the total cost is over $1,000. If it is, the program will give a 10% discount before doing the calculations.

1. **Design the algorithm and create the flowchart**

   Before writing the Python program, create the algorithm and the flowchart of the program.

2. **AI support (planning)**

   Write the Python program. You may use AI to help you plan your solution. If you do, carefully review the response and make sure that it is clear and logical.

3. **Enter the prices of the laptop and the accessories from the user**
   - Prompt the user to input the price of the laptop.
   - Prompt the user to input the price of the bag.
   - Prompt the user to input the price of the keyboard.

4. **Calculate the total cost**

   Add the prices of the laptop, bag, and keyboard to determine the total cost.

5. **Determine if the total cost exceeds $1,000**
   - If yes, apply a 10% discount to the total cost.
   - If no, proceed without applying a discount.

6. **Calculate payment details**
   - Calculate the upfront payment (30% of the total cost).
   - Calculate the remaining balance after the upfront payment.
   - Divide the remaining balance into six equal monthly installments.

7. **Display the results**
   - Display the total cost (after applying any discount).
   - Display the upfront payment.
   - Display the amount of each monthly installment.

8. **Test the program**
   - Write down example prices and calculate the results manually (e.g., use a calculator or pen and paper).
   - Input the same values into your program and compare the results.
   - Ensure the program is accurate and follows the logic correctly.

Wrap-Up

This unit covered how to:

> Explain what a program is and how it interacts with a computer.
> Identify the purpose of programming languages.
> Define and create an algorithm.
> Create a flowchart to represent a problem.
> Write simple Python programs.
> Use variables.
> Differentiate between different types of variables.
> Use conditional statements to implement decision-making in code.
> Apply nested and multiple decision structures to handle complex conditions.
> Use an AI assistant to generate code.

Key Terms

- algorithm
- arithmetic operator
- assignment operator
- Boolean operator
- comment
- comparison operator
- conditional statements
- data type
- flowchart
- if statement
- if...else statement
- if...elif...else statement
- nested if statements
- program
- programming language
- Python
- variable

Interactive Data Applications 7

7

Introduction

This unit introduces the ideas behind building interactive data apps in Python using Streamlit. You will learn how to collect user input with widgets, display results in a browser, organize apps using forms and layouts, and connect your programs to online data sources through web APIs. You will also explore how to turn API data into tables and charts, and how AI tools can help improve the clarity and usefulness of an app without replacing your own decisions.

Learning Objectives

In this unit, you will:

- explain how Streamlit reruns scripts when widget values change.
- use widgets such as text inputs, sliders, selectboxes, and buttons to collect user input.
- display results using Streamlit UI functions such as **st.write**, **st.dataframe**, and charts.
- use forms to group related inputs and submit them together.
- organize app layouts using columns and the sidebar.
- fetch data from public web APIs.
- convert JSON API responses into Python data structures and DataFrames.
- visualize data using tables, bar charts, and line charts.
- explain how AI can help improve the text and features of a Streamlit app.

LESSON 1

Introduction to Data App Development

Up to now, you have been writing Python programs that run in an editor or a terminal. They read input from the keyboard, print output as text, and then stop. That is useful, but it is not how most people expect to interact with software.

Streamlit gives you a different way to use your Python code. The same language and logic you already know can power web apps that run in the browser. This has the following advantages:

- Users type into text boxes instead of the terminal.
- They move sliders and click buttons instead of answering `input()` prompts.
- Program output appears as formatted text, tables, and charts instead of lines of plain text in the console.

At this stage, you do not need HTML, CSS, or JavaScript to build these applications. You continue using Python and ask Streamlit to handle input and output. In this unit, you will treat Streamlit as a thin layer around the programs you already know how to write. The focus is on three ideas:

1. **Connecting widgets to variables:** Every input widget, for example a text box or slider, returns a value. You store that value in a Python variable and use it in your code.
2. **Displaying results as UI elements**: Instead of `print`, you call functions like `st.write`, `st.dataframe`, or `st.line_chart`. These functions display output on a web page.
3. **Thinking in terms of live apps**: Whenever a user changes a widget, Streamlit reruns your script from top to bottom. The app always reflects the current state of the inputs.

In this lesson, the goal is not only to learn a tool, but to understand how interactive software responds to user input and updates results dynamically.

What Streamlit Does for You

Streamlit sits between your Python code and the browser.

From your point of view:

- You write a normal Python file.
- You import `streamlit`.
- You call `st.*` functions to create inputs and outputs.

From the user's point of view:

- They open a web page.
- They type in boxes and click buttons.
- Results update as they interact.

You do not manage routes, templates, or HTTP requests. Streamlit takes your script and turns it into an interactive web page.

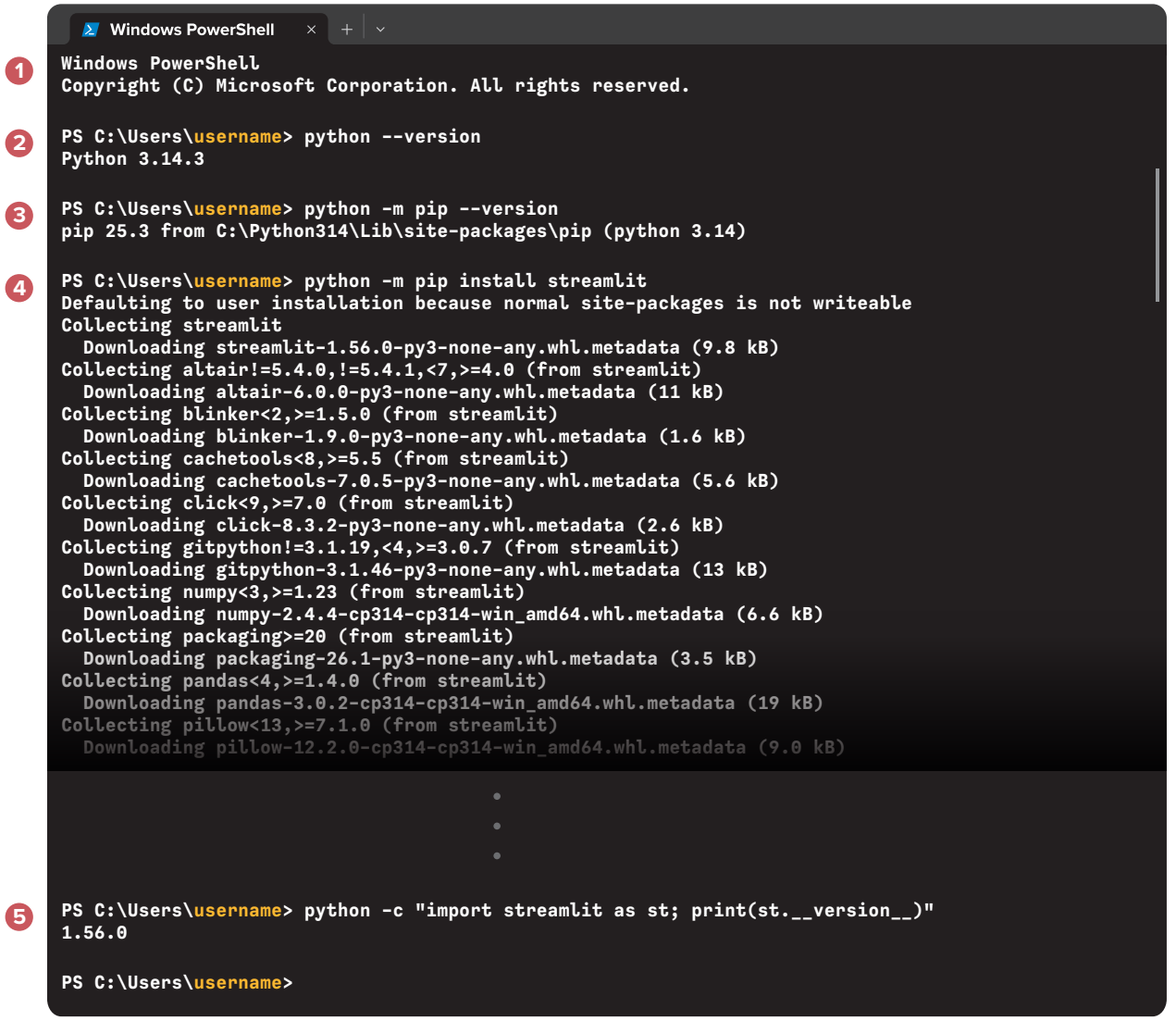
Installing Streamlit

You need Python and `pip` installed and working on your machine.

1. Open a terminal (PowerShell on Windows, Terminal on macOS or Linux).
2. Check that Python is installed: `python --version`
3. Check that pip is installed: `python -m pip --version`
4. Install Streamlit: `python -m pip install streamlit`
5. Quick check: `python -c "import streamlit as st; print(st.__version__)"`

If this prints a version number without errors, you are ready.

The programs in this unit use libraries such as Streamlit, pandas, and requests. AI-generated code assumes a certain version is installed, but library versions change over time, so some functions may work differently or no longer exist.



If AI-generated code gives an ImportError or AttributeError, the cause is often a version difference. Read the error message, check the documentation for your installed version, and see whether the function was renamed or deprecated.

Running a Streamlit app

To run any Streamlit app, you always use the same command.

1. Create a file, for example:

   ```
   import streamlit as st

   st.set_page_config(page_title="Hello Streamlit", layout="centered")

   st.title("Hello Streamlit")

   st.write("This is my first Streamlit app.")
   ```

   Save as `hello_streamlit.py`.

2. In a terminal, go to the folder with that file:

   ```
   cd path/to/your/folder
   ```

3. Run:

   ```
   python -m streamlit run hello_streamlit.py
   ```

Streamlit will start a local server and open the app in your browser at a URL such as `http://localhost:8501`

To stop the app, return to the terminal where it is running and press `Ctrl + C`. For the rest of this unit, whenever you need to run a `.py` file, you will run it with:

```
python -m streamlit run filename.py
```

Only the filename changes.

The basic Streamlit pattern

Most apps in this unit use the same basic program structure:

1. Import Streamlit.

   ```
   import streamlit as st
   ```

2. Configure the page.

   ```
   st.set_page_config(page_title="Some title", layout="wide")
   ```

3. Add a page title.

   ```
   st.title("Some title")
   ```

4. Create input widgets and read their values.

   ```
   name = st.text_input("Your name")
   ```

5. Perform some standard operations using Python.

   ```
   greeting = f"Hello {name}" if name else "Hello"
   ```

6. Display results.

   ```
   st.write(greeting)
   ```

Every time the user changes a widget, Streamlit reruns the script from the top and redraws the page. The current widget values are used to compute and display the new result.

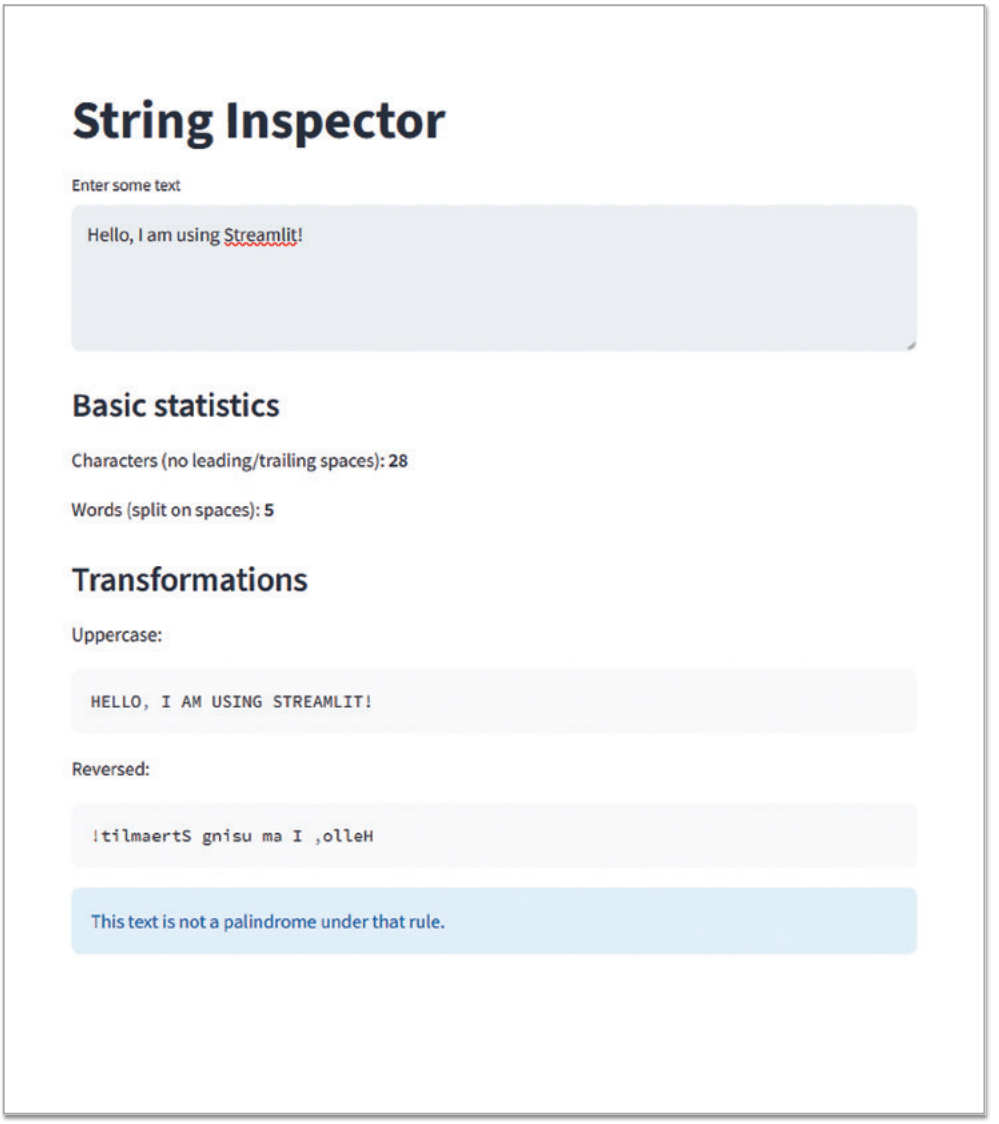
7.1

String Inspector

Now let's explore an app called String Inspector that uses these ideas. This program lets the user type text and then displays:

- How many characters it has.
- How many words it has.
- Some simple transformations.
- A short message about whether it reads the same forwards and backwards under a simple rule.

This program uses the same structure as any Streamlit script and introduces a few common functions.



Full program

```
import streamlit as st

st.set_page_config(page_title="String Inspector", layout="centered")

st.title("String Inspector")

text = st.text_area("Enter some text", height=150)

if text:
    stripped = text.strip()
    words = stripped.split()
    num_chars = len(stripped)
    num_words = len(words)

    st.subheader("Basic statistics")
    st.write(f"Characters (no leading/trailing spaces): **{num_chars}**")
    st.write(f"Words (split on spaces): **{num_words}**")

    st.subheader("Transformations")
    st.write("Uppercase:")
    st.code(stripped.upper())
    st.write("Reversed:")
    st.code(stripped[::-1])

    cleaned = "".join(ch.lower() for ch in stripped if ch.isalnum())
    if cleaned and cleaned == cleaned[::-1]:
        st.success("This text is a palindrome if you ignore spaces and
punctuation.")
    else:
        st.info("This text is not a palindrome under that rule.")
else:
    st.info("Type something in the box above.")
```

Save the code above in a file named `string_inspector.py` and run it in a terminal using: `python -m streamlit run string_inspector.py`

Let's take a look at the code in more detail:

`st.set_page_config`

Configures the page:

1. `st.set_page_config(page_title="String Inspector", layout="centered")`
   - `page_title` sets the browser tab title.
   - `layout="centered"` keeps content in a single centered column.

st.title

Adds the main heading:

```
st.title("String Inspector")
```

st.text_area

Adds multi-line text input:

```
text = st.text_area("Enter some text", height=150)
```

- The first argument is the label above the box.
- `height` controls the box height in pixels.
- The return value `text` is the string currently in the box, or `""` if it is empty.

st.subheader

Adds a section heading:

```
st.subheader("Basic statistics")
```

- Used to group related outputs.

st.write

Used for generic output:

```
st.write(f"Characters (no leading/trailing spaces): **{num_chars}**")
```

- `st.write` understands simple Markdown in strings such as `**bold**`.

st.code

Used for a fixed-width block:

```
st.code(stripped.upper())
```

- Useful for showing changes that have been made to text strings.

st.success and st.info

Prints status messages:

```
st.success("This text is a palindrome if you ignore spaces and punctuation.")
st.info("This text is not a palindrome under that rule.")
```

- `st.success` is used to show that something has worked correctly.
- `st.info` is used to give neutral information.
- `st.warning` and `st.error` are also available.

The script relies on the following standard string operations:

```
stripped = text.strip()      # remove leading and trailing spaces
words = stripped.split()     # use spaces to split text into words
num_chars = len(stripped)    # count characters
num_words = len(words)       # count words
```

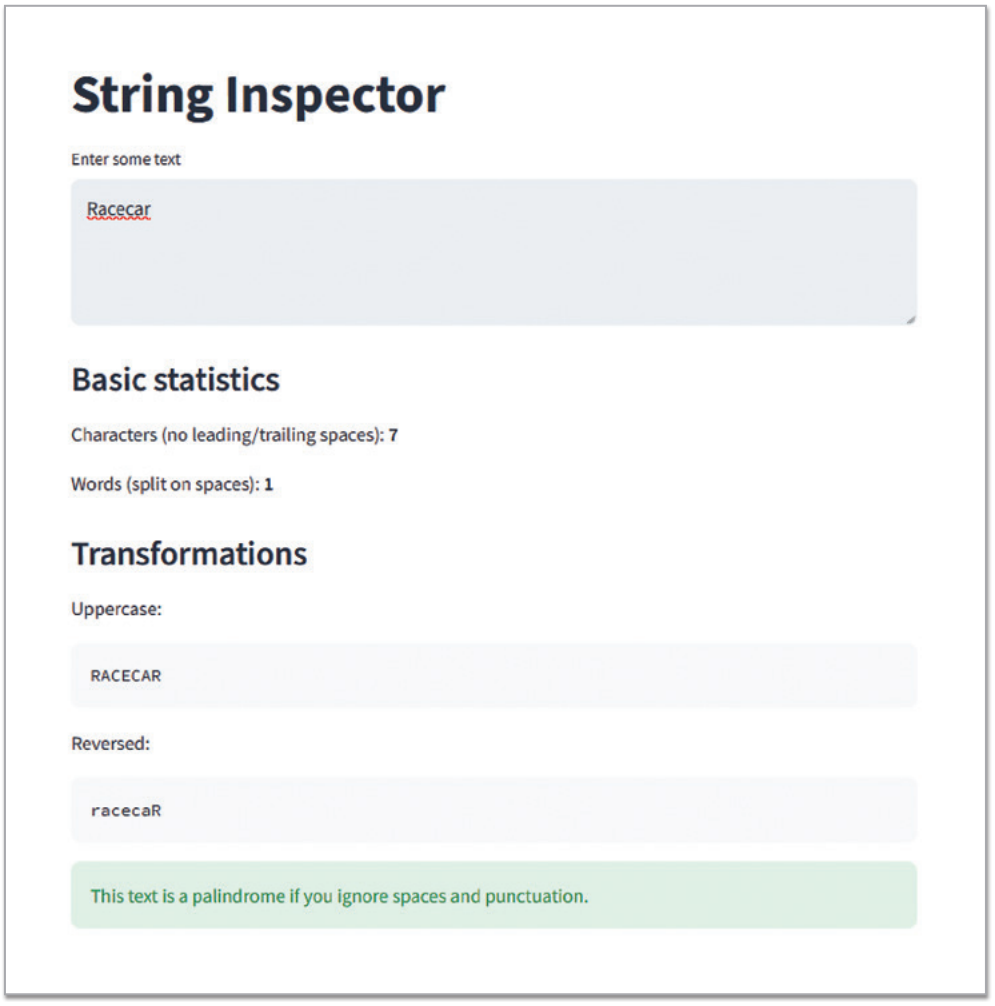
To reverse the string, we use:

```
reversed_text = stripped[::-1]
```

And to perform a simple check if the text is a palindrome, ignoring spaces and punctuation:

```
cleaned = "".join(ch.lower() for ch in stripped if ch.isalnum())
if cleaned and cleaned == cleaned[::-1]:
    # text matches its reverse under   this rule
else:
    # text does not match its reverse
```

`.isalnum()` keeps only letters and digits, and `.lower()` makes the comparison case insensitive.



Career connections

Interactive input and output design is used by software developers, frontend developers, data app developers, and product designers to create tools that respond to user actions. It helps make software more intuitive and effective for real users.

7.1

Unit Converter

Let's examine another program, called **Unit Converter**, that uses the same Streamlit structure but focuses on numbers.

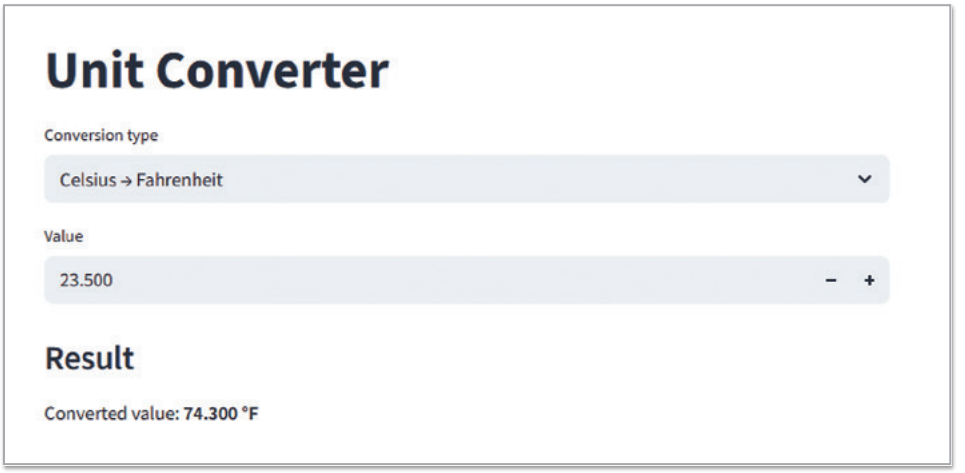
Unit Converter converts between different units for temperature and distance. The program:

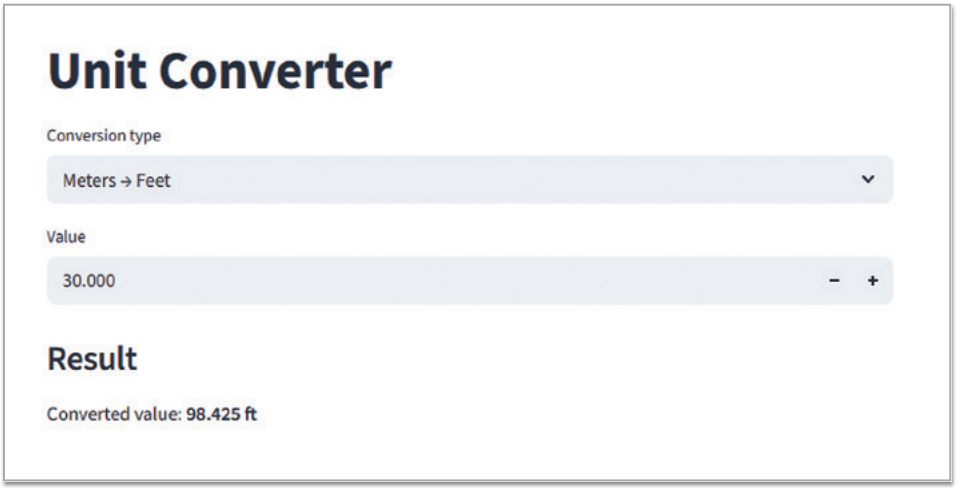
- Lets the user pick a conversion type
- Lets the user type a value
- Shows the converted result with units

This example introduces two new widgets:

- `st.selectbox` for a drop-down choice
- `st.number_input` for numeric input

The rest of the program (`set_page_config, title, subheader, write`) follows the pattern we have already seen in our String Inspector program.





Full program

```
import streamlit as st

st.set_page_config(page_title="Unit Converter", layout="centered")

st.title("Unit Converter")

mode = st.selectbox(
    "Conversion type",
    ["Celsius > Fahrenheit", "Fahrenheit > Celsius", "Meters > Feet", "Feet > Meters"],
)

value = st.number_input("Value", value=0.0, format="%.3f")

result = None
label = ""

if mode == "Celsius > Fahrenheit":
    result = value * 9 / 5 + 32
    label = "F"
elif mode == "Fahrenheit > Celsius":
     result = (value - 32) * 5 / 9
     label = "C"
elif mode == "Meters > Feet":
    result = value * 3.28084
    label = "ft"
elif mode == "Feet > Meters":
    result = value / 3.28084
    label = "m"

if result is not None:
    st.subheader("Result")
    st.write(f"Converted value: **{result:.3f} {label}**")
```

Save the program as `unit_converter.py` and run it in a terminal using:
`python -m streamlit run unit_converter.py`

`st.selectbox`

Adds a drop-down list for choosing one option from several:

```
mode = st.selectbox(
"Conversion type",
["Celsius > Fahrenheit", "Fahrenheit > Celsius", "Meters > Feet", "Feet > Meters"],
)
```

- The first argument is the label shown above the box.
- The second argument is the list of options.
- The return value mode is the selected string (for example "Meters > Feet").

You typically use the selected value in an `if` / `elif` chain.

`st.number_input`

Adds a numeric input box:

```
value = st.number_input("Value", value=0.0, format="%.3f")
```

- `"Value"` is the label that will be displayed on the input box.
- `value=0.0` sets the initial value.
- `format="%.3f"` controls how many decimal places are shown.
- The return value `value` is a `float`.

The program also uses components we have already seen:

```
st.set_page_config(page_title="Unit Converter", layout="centered")
st.title("Unit Converter")
st.subheader("Result")
st.write(f"Converted value: **{result:.3f} {label}**")
```

These are used the same way as in the String Inspector program.

The conversion logic uses arithmetic and a selection based on `mode`:

```
result = None
label = ""
if mode == "Celsius > Fahrenheit":
    result = value * 9 / 5 + 32
    label = "F"
elif mode == "Fahrenheit > Celsius":
    result = (value - 32) * 5 / 9
    label = "C"
elif mode == "Meters > Feet":
    result = value * 3.28084
    label = "ft"
elif mode == "Feet > Meters":
    result = value / 3.28084
    label = "m"
```

- `mode` selects the formula.
- `result` holds the numeric result.
- `label` holds the unit to show with the result.

Finally, the program checks whether `result` has been set before printing it:

```
if result is not None:
 st.subheader("Result")
 st.write(f"Converted value: **{result:.3f} {label}**")
```

If the result is `not None`, the program displays the formatted value with units.

Password Generator

We will now use Streamlit to build a Password Generator.

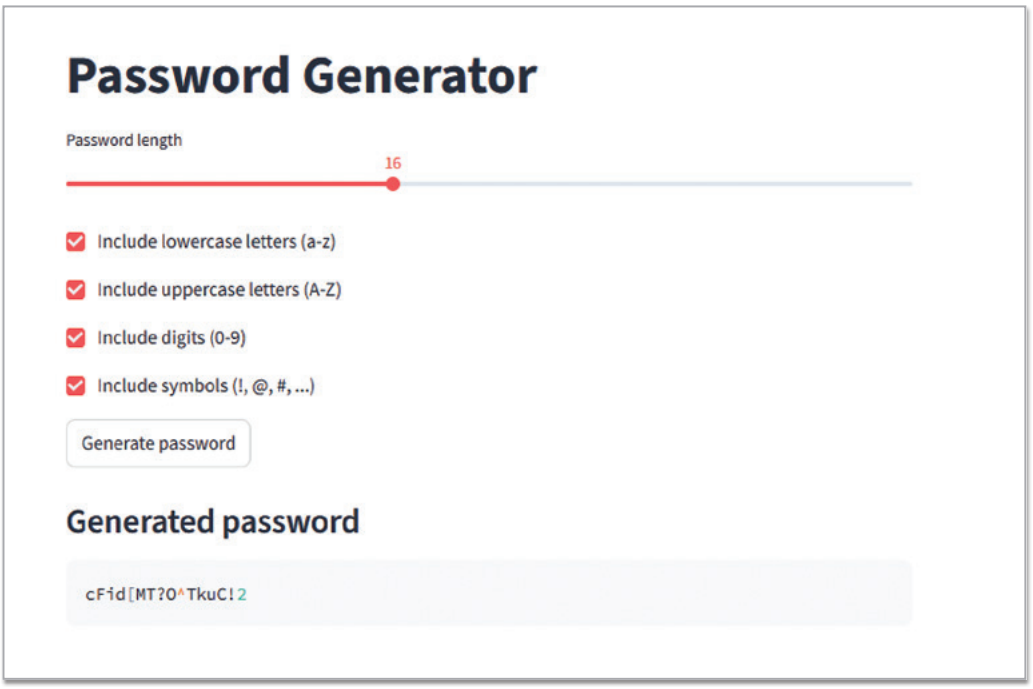
This program:

- Lets the user choose a password length
- Lets the user choose which character types to include
- Generates a random password when a button is pressed
- Shows an error if no character types are selected

It introduces three new widgets:

- `st.slider` for numeric input with a bar
- `st.checkbox` for on/off switches
- `st.button` for actions that run once when clicked

It also uses `st.error` to show errors.



Full program

```
import streamlit as st
import random
import string

st.set_page_config(page_title="Password Generator", layout="centered")

st.title("Password Generator")
```

```
length = st.slider("Password length", min_value=6, max_value=32, value=12)
use_lower = st.checkbox("Include lowercase letters (a-z)", value=True)
use_upper = st.checkbox("Include uppercase letters (A-Z)", value=True)
use_digits = st.checkbox("Include digits (0-9)", value=True)
use_symbols = st.checkbox("Include symbols (!, @, #, ...)", value=False)

alphabet = ""

if use_lower:
    alphabet += string.ascii_lowercase
if use_upper:
    alphabet += string.ascii_uppercase
if use_digits:
    alphabet += string.digits
if use_symbols:
    alphabet += "!@#$%^&*()-_=+[]{};:,.?/"

generate = st.button("Generate password")

if generate:
    if not alphabet:
        st.error("Select at least one character type.")
    else:
        password = "".join(random.choice(alphabet) for _ in range(length))
        st.subheader("Generated password")
        st.code(password)
```

`st.slider`

Adds a **slider** for numeric values:

1 `length = st.slider("Password length", min_value=6, max_value=32, value=12)`

- `"Password length"` is the slider label.
- `min_value` and `max_value` set the range.
- `value` is the initial position of the slider.
- The return value `length` is an `int`.

`st.checkbox`

Adds a **checkbox** for Boolean choices:

2
```
use_lower = st.checkbox("Include lowercase letters (a-z)", value=True)
use_upper = st.checkbox("Include uppercase letters (A-Z)", value=True)
use_digits = st.checkbox("Include digits (0-9)", value=True)
use_symbols = st.checkbox("Include symbols (!, @, #, ...)", value=False)
```

Each checkbox returns `True` or `False`.

st.button

Adds a **button** that is `True` only in the run where it was clicked:

3 `generate = st.button("Generate password")`

- When the user clicks it, `generate` becomes `True` for that run.
- On the next automatic rerun, it returns to `False` until clicked again.

st.error

Used to show an error box:

4 `st.error("Select at least one character type.")`

This message appears when the user has not made a valid selection.

The program also uses:

```
st.set_page_config(page_title="Password Generator", layout="centered")
st.title("Password Generator")
st.subheader("Generated password")
st.code(password)
```

These are used in the same way as in our earlier programs.

The program builds a character-set string based on the choices in the checkboxes, then uses `random.choice` to pick from this character set.

5
```
alphabet = ""
if use_lower:
    alphabet += string.ascii_lowercase
if use_upper:
    alphabet += string.ascii_uppercase
if use_digits:
    alphabet += string.digits
if use_symbols:
    alphabet += "!@#$%^&*()-_=+[]{};:,.?/"
```

If none of the checkboxes are selected, `alphabet` stays empty, and the program shows an error.

Password creation is achieved by the line:

6 `password = "".join(random.choice(alphabet) for _ in range(length))`

- This chooses `length` random characters from `alphabet`.
- These characters are joined into a single string.

The result is shown using `st.code` so that it stands out clearly on the page.

Discussion Question

- What makes a widget such as a text box or slider useful in an interactive app?

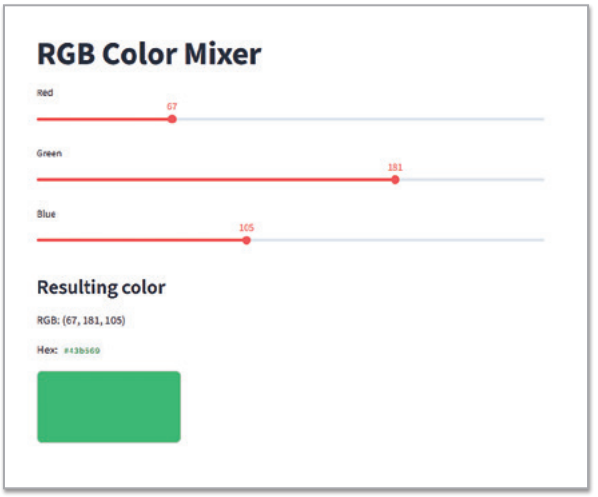
RGB Color Mixer

The final program in this lesson is **RGB Color Mixer**.

This program:

- Uses three sliders for red, green, and blue values
- Shows the numeric RGB and hex color codes
- Draws a colored rectangle that updates as the user moves the sliders

It reuses `st.slider` and introduces using `st.markdown` with HTML for a simple visual.



Full program

```
import streamlit as st

st.set_page_config(page_title="RGB Color Mixer", layout="centered")

st.title("RGB Color Mixer")

r = st.slider("Red", min_value=0, max_value=255, value=128)
g = st.slider("Green", min_value=0, max_value=255, value=128)
b = st.slider("Blue", min_value=0, max_value=255, value=128)

hex_code = f"#{r:02x}{g:02x}{b:02x}"

st.subheader("Resulting color")
st.write(f"RGB: ({r}, {g}, {b})")
st.write(f"Hex: `{hex_code}`")
```

1.
2.

```
st.markdown(
    f"""
    <div style="
        width: 200px;
        height: 100px;
        border-radius: 8px;
        border: 1px solid #ccc;
        background-color: {hex_code};
    "></div>
    """,
    unsafe_allow_html=True,
)
```

(3)

(1) This program creates three slider widgets:

```
r = st.slider("Red", min_value=0, max_value=255, value=128)
g = st.slider("Green", min_value=0, max_value=255, value=128)
b = st.slider("Blue", min_value=0, max_value=255, value=128)
```

All three widgets work like the password-length slider we saw before, but here they represent related values.

(2) The RGB hex code is derived from the values of the three sliders:

```
hex_code = f"#{r:02x}{g:02x}{b:02x}"
```

`:02x` formats an integer as two hexadecimal digits, adding a leading zero if needed.

The final string will look something like `#8040ff`.

`st.markdown` with HTML

Streamlit allows simple HTML if you pass `unsafe_allow_html=True`:

(3)

```
st.markdown(
    f"""
    <div style="
        width: 200px;
        height: 100px;
        border-radius: 8px;
        border: 1px solid #ccc;
        background-color: {hex_code};
    "></div>
    """,
    unsafe_allow_html=True,
)
```

This draws a colored box whose background color comes from the selected `hex_code`.

Exercises

Answer each set of questions in your notebook.

1. Explain how using Streamlit is different from running a Python program in the terminal.
2. Describe what we mean when we say Streamlit reruns a script from top to bottom.
3. Explain how widgets such as `text_input`, `slider`, and `selectbox` are connected to Python variables.
4. Explain why an interactive app reruns when widget values change.
5. Compare terminal input/output with browser-based interaction.
6. Create a simple idea for a Streamlit app similar to the examples in this lesson, and explain what inputs it would collect and what outputs it would display.
7. Modify the Unit Converter so it also supports kilograms and pounds.
8. Improve the Password Generator so it warns users when the password is short.

LESSON 2

Widgets, Forms, and Layout

In the previous lesson, you built simple tools that updated as soon as you changed any widget, so every slider move or keystroke caused an immediate rerun and a new result.

In this lesson, you will work with programs where you fill in several related fields, submit them all at once, and then a clear result or preview appears in a separate section of the page.

You will create three programs: **Poll Builder**, **Expense Splitter**, and **Issue Reporter**.

All three use:

- `st.form` and `st.form_submit_button` to group inputs
- `st.columns` to organize inputs on one side of the page and results on the other
- Status messages such as `st.info`, `st.error`, and `st.success`

Forms in Streamlit

With the widgets you have used so far (text inputs, sliders, checkboxes), the values were applied immediately. Sometimes, you may want different behavior. A **form** lets you change several fields and then press a button once to "submit" them all together. Inside a form, you use standard widgets. Then, at the end of the form, you add a submit button. The button returns `True` only when it is clicked.

Here is an example:

```
with st.form("example_form"):
    name = st.text_input("Name")
    age = st.number_input("Age", min_value=0, max_value=120, value=18)

    submitted = st.form_submit_button("Save")
if submitted:
    st.write(f"Saved: {name}, {age} years old")
```

Key points:

- `with st.form("example_form"):` starts the form block.
- `st.form_submit_button("Save")` returns True only when the submit button is clicked.
- Typically, you handle all input validation and processing of the results inside the `if submitted:` block, so that this logic only runs after the user submits the form.

Forms are useful in software development because they group related inputs, reduce errors, and make programs easier for users to understand.

The three programs in this lesson all follow this pattern.

7.2

Poll Builder

You will now explore a program that builds a simple poll. It lets you:

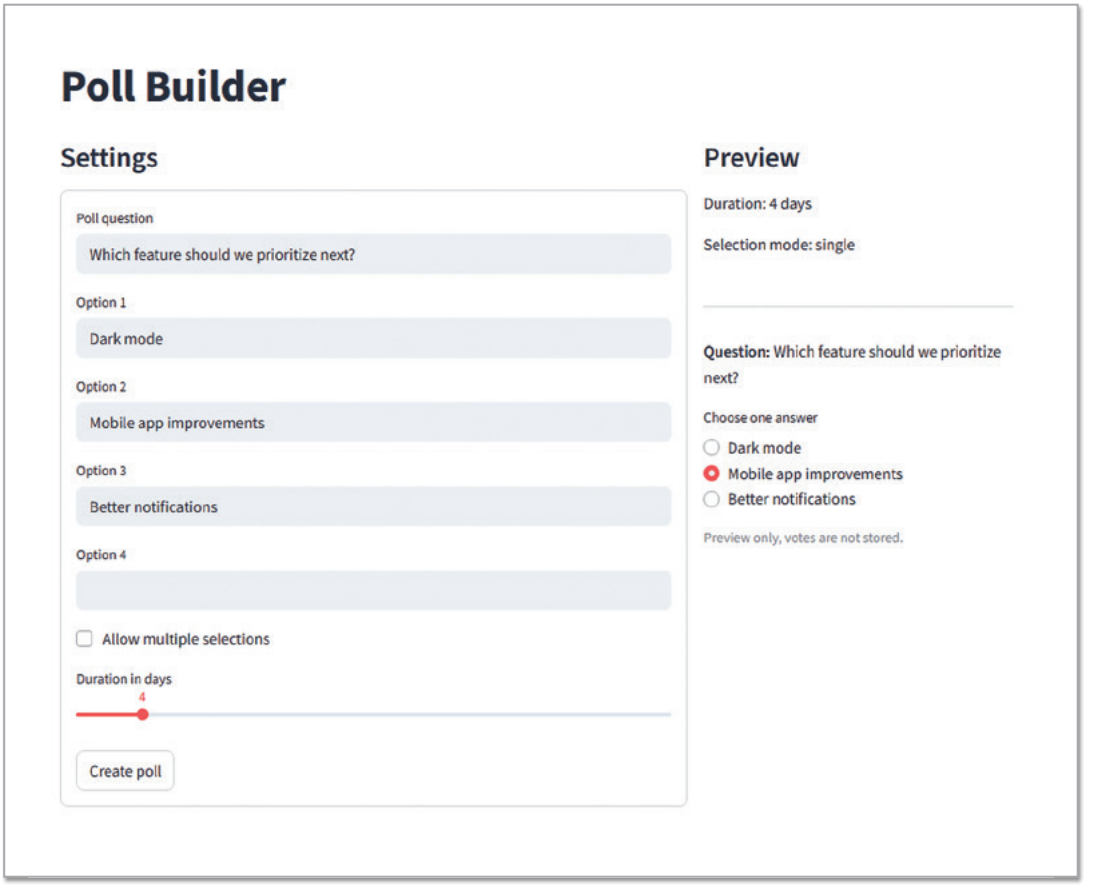
- Write a poll question
- Enter up to four options for responses
- Choose whether multiple answers are allowed
- Set a duration in days
- See how the poll would look

The layout has two columns:

- Left side: settings inside a form
- Right side: preview of the poll

The main components of this program are:

- `st.form` and `st.form_submit_button`
- `st.columns([2, 1])` to split the page
- Using `st.checkbox` or a `st.radio` to show a mock poll



Full program

```
import streamlit as st

st.set_page_config(page_title="Poll Builder", layout="wide")

st.title("Poll Builder")

builder_col, preview_col = st.columns([2, 1])

with builder_col:
    st.subheader("Settings")

    with st.form("poll_form"):
        question = st.text_input("Poll question")

        option1 = st.text_input("Option 1")
        option2 = st.text_input("Option 2")
        option3 = st.text_input("Option 3", value="")
        option4 = st.text_input("Option 4", value="")

        allow_multi = st.checkbox("Allow multiple selections")
        duration_days = st.slider("Duration in days", 1, 30, 7)

        submitted = st.form_submit_button("Create poll")

    if submitted:
        st.success("Poll created. Preview on the right.")

with preview_col:
    st.subheader("Preview")

    options = [o for o in [option1, option2, option3, option4] if o.strip()]

    if not question or len(options) < 2:
        st.info("Enter a question and at least two options to preview the
        poll.")
    else:
        st.write(f"Duration: {duration_days} days")
        st.write("Selection mode: " + ("multiple" if allow_multi else
        "single"))
        st.markdown("---")
        st.markdown(f"**Question:** {question}")
        if allow_multi:
            for opt in options:
                st.checkbox(opt)
        else:
            st.radio("Choose one answer", options)

        st.caption("Preview only, votes are not stored.")
```

7.2 **The key statements to understand in this program are:**

1. `st.set_page_config(page_title="Poll Builder", layout="wide")`
   This sets the title and uses a wide layout.

2. `builder_col, preview_col = st.columns([2, 1])`
   This creates two columns (left twice as wide as right).

3. `with st.form("poll_form"):`
   This groups all the poll settings inside a form.

4. `question = st.text_input("Poll question")`
   This creates a text input for the question, `option1` to `option4`. These provide four text boxes where the user can type options for the question. Options 3 and 4 can be left empty.

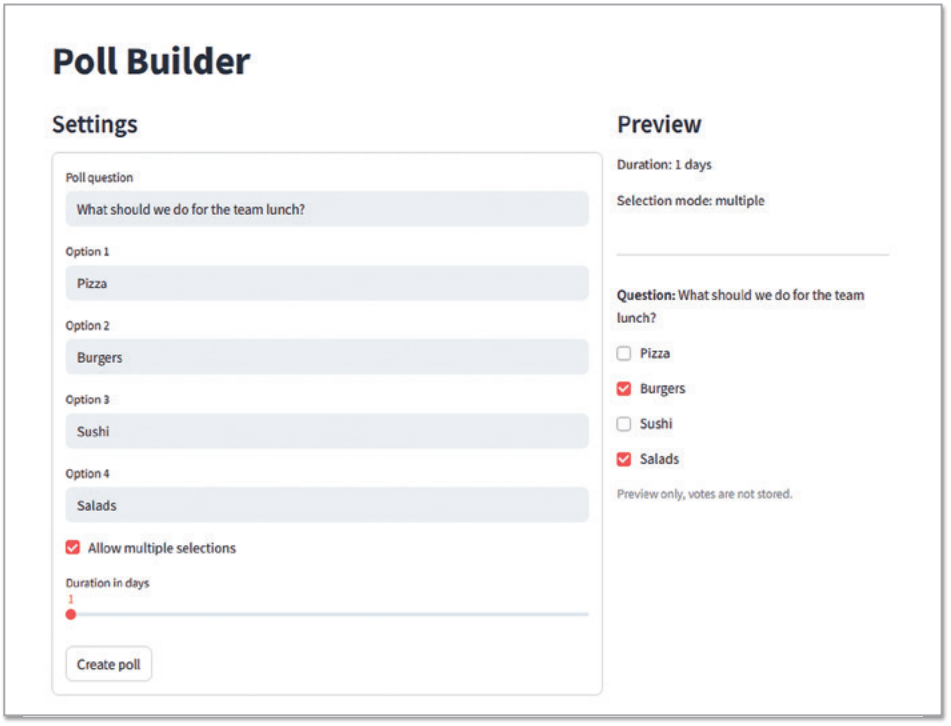
5. `allow_multi = st.checkbox("Allow multiple selections")`
   This is a Boolean that determines if the poll is single choice or multiple choice.

6. `duration_days = st.slider("Duration in days", 1, 30, 7)`
   This is a slider that sets how many days the poll stays open.

7. `submitted = st.form_submit_button("Create poll")`
   This is the submit button for the form.

In the "preview" column that is created on the screen, the code:

8. `options = [o for o in [option1, option2, option3, option4] if o.strip()]`
   Builds a list with only the non-empty options. If there is no question or fewer than two options, the program displays an info message. Otherwise, it shows the poll duration, selection mode, and a preview that uses `st.checkbox` for multiple-choice polls or `st.radio` for single-choice polls.



Expense Splitter

Now let's examine a program that helps three people split the bill at a restaurant.

This program lets the user:

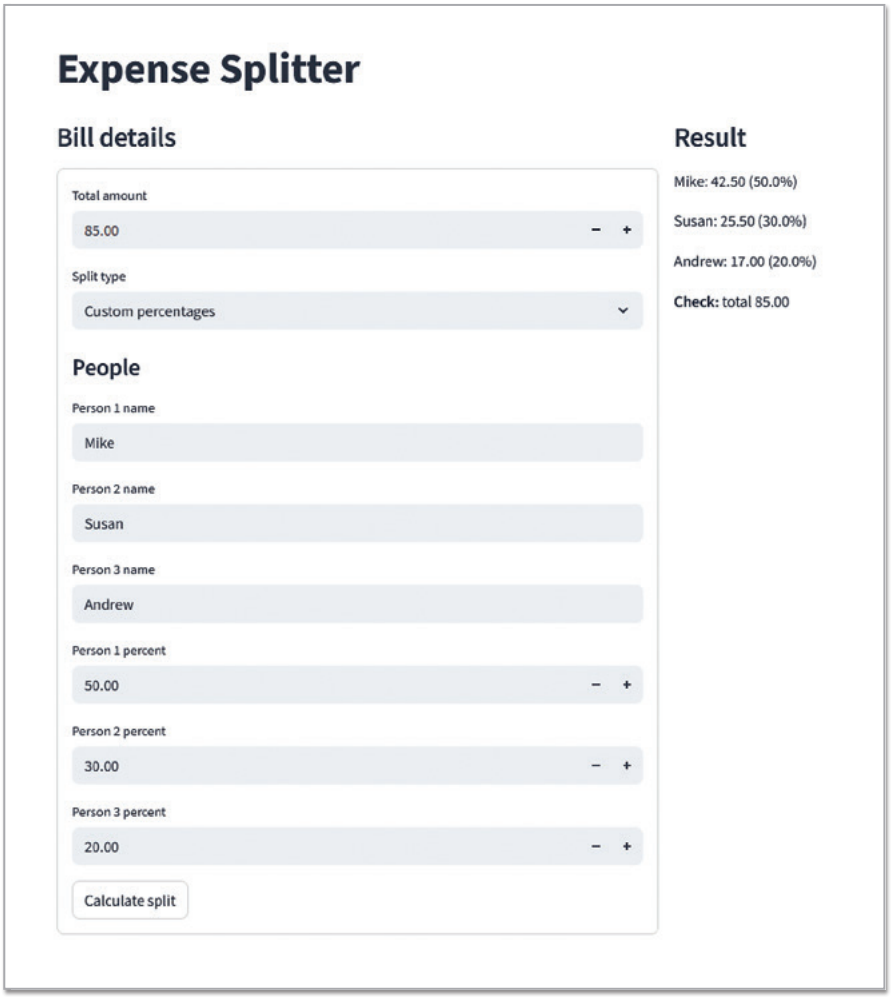
- Enter a total amount
- Choose between an equal split and custom percentages
- Enter three names
- If needed, enter three percentages that must add up to 100
- See how much each person should pay

Again, the layout has two columns:

- Left side: the form for entering the data
- Right side: result and validation messages

There are some new points to be noted here:

- The program performs validation (the total must be positive).
- It also checks that the three percentages add up to 100.
- It uses `st.error` to give clear feedback.



Full program

```
import streamlit as st

st.set_page_config(page_title="Expense Splitter", layout="wide")

st.title("Expense Splitter")

form_col, result_col = st.columns([2, 1])

with form_col:
    st.subheader("Bill details")

    with st.form("split_form"):
        total = st.number_input("Total amount", min_value=0.0, value=100.0,
step=1.0)

        split_mode = st.selectbox(
            "Split type",
            ["Equal", "Custom percentages"],
        )

        st.markdown("#### People")

        name1 = st.text_input("Person 1 name", value="Alice")
        name2 = st.text_input("Person 2 name", value="Bob")
        name3 = st.text_input("Person 3 name", value="Charlie")

        p1 = p2 = p3 = None
        if split_mode == "Custom percentages":
            p1 = st.number_input("Person 1 percent", min_value=0.0, max_
value=100.0, value=33.0)
            p2 = st.number_input("Person 2 percent", min_value=0.0, max_
value=100.0, value=33.0)
            p3 = st.number_input("Person 3 percent", min_value=0.0, max_
value=100.0, value=34.0)

        submitted = st.form_submit_button("Calculate split")

with result_col:
    st.subheader("Result")

    if not submitted:
        st.info("Fill the form and press Calculate split.")
    else:
        names = [name1 or "Person 1", name2 or "Person 2", name3 or "Person
3"]

        if total <= 0:
            st.error("Total amount must be greater than zero.")
        else:
            if split_mode == "Equal":
                share = total / 3
                for n in names:
                    st.write(f"{n}: {share:.2f}")
                st.markdown(f"**Check:** 3 x {share:.2f} = {total:.2f}")
            else:
                percents = [p1, p2, p3]
                if any(p is None for p in percents):
                    st.error("Enter all three percentages.")
```

```
                else:
                    percent_sum = sum(percents)
                    if abs(percent_sum - 100.0) > 0.01:
                        st.error(f"Percentages must add up to 100. Current
sum: {percent_sum:.2f}")
                    else:
                        for n, p in zip(names, percents):
                            amount = total * p / 100.0
                            st.write(f"{n}: {amount:.2f} ({p:.1f}%)")
                        st.markdown(f"**Check:** total {total:.2f}")
```

The most important statements to focus on in this code are:

1. `form_col, result_col = st.columns([2, 1])`

   This splits the page into inputs and result.

2. `total = st.number_input("Total amount", min_value=0.0, value=100.0, step=1.0)`

   This allows input for the total bill.

3. `split_mode = st.selectbox("Split type", ["Equal", "Custom percentages"])`

   This allows the user to choose between an equal split or custom percentages.

There are also three text inputs for names. If no names are given, generic labels are used.
If custom mode is selected, three `st.number_input` statements allow percentages to be entered.

4. `submitted = st.form_submit_button("Calculate split")`

   This is the submit button for the form.

In the right-hand column of the page:

- If the form has not yet been submitted, an info message is shown.
- If `total <= 0`, an error message is displayed.
- If an equal split has been chosen, the total is divided by three and displayed.
- If a custom split is selected, the program checks for missing percentages and ensures that they add up to 100.
- Each person's final bill and the percentage of the total are displayed.

Discussion Questions

- Why is a form useful when several inputs belong to one task?
- Why should some apps show validation messages before displaying a result?
- How can a two-column layout make an app easier to use?

Career connections

Forms, layouts, and validation are important in the work of UI designers, frontend developers, QA testers, and product teams who build software for real users.

7.2

Issue Reporter

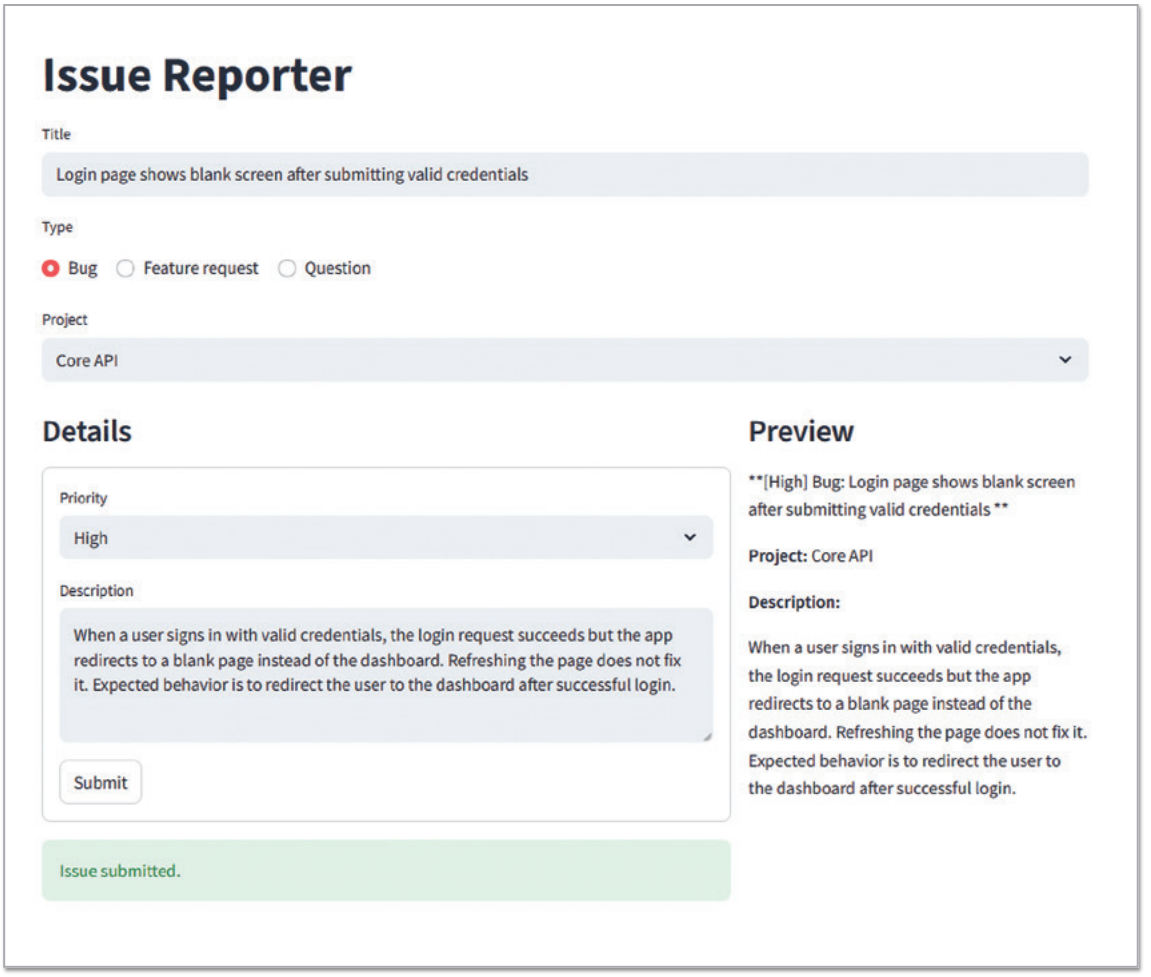
This Issue Reporter program is a simple version of the tools that software development teams use to track bugs and feature requests.

The program lets you:

- Enter a short title
- Choose what kind of issue you want to report (bug, feature request, question)
- Select which project it belongs to
- Set a priority
- Write a short description
- See the data you have entered formatted as a compact issue summary

The layout again uses two columns:

- left side: a form with the issue details
- right side: a preview after the form has been submitted



Full program

```
import streamlit as st

st.set_page_config(page_title="Issue Reporter", layout="wide")

st.title("Issue Reporter")

title = st.text_input("Title")
issue_type = st.radio(
    "Type",
    ["Bug", "Feature request", "Question"],
    horizontal=True,
)

project = st.selectbox(
    "Project",
    ["Core API", "Frontend UI", "Data pipeline", "Other"],
)

form_col, preview_col = st.columns([2, 1])

with form_col:
    st.subheader("Details")

    with st.form("issue_form"):
        priority = st.selectbox(
            "Priority",
            ["Low", "Medium", "High", "Critical"],
            index=0,
        )
        description = st.text_area(
            "Description",
            height=150,
        )

        submitted = st.form_submit_button("Submit")

    if submitted:
        st.success("Issue submitted.")

with preview_col:
    st.subheader("Preview")

    if not title and not description:
        st.info("Fill in the form to see a preview.")
    else:
        st.write(f"**[{priority}] {issue_type}: {title or '(no title)'}**")
        st.write(f"**Project:** {project}")
        st.markdown("**Description:**")
        st.write(description or "(no description)")
```

You have now used forms and two-column layouts to build multi-step tools with clear previews and validation. In the next lesson, you will learn how to connect Streamlit programs to external data and visualizations, so they react not only to user input but also to information from an API.

Exercises

Answer each set of questions in your notebook.

1. Explain the purpose of a form in Streamlit.
2. Describe the difference between widgets that update immediately and widgets grouped inside a form.
3. Explain how `st.columns` can improve the layout of a Streamlit app.
4. Describe how the Expense Splitter validates user input before showing results.
5. Explain why forms are useful when several inputs belong to one task.
6. Compare immediate widget updates with form-based submission.
7. Design a similar form-based Streamlit app that collects several related inputs and shows the result in a separate section of the page.
8. Redesign the Issue Reporter so the preview updates live as fields change.
9. Build a simple registration form with clear validation messages.

LESSON 3

Integrating APIs, Data, and Visualizations

In this lesson, you will connect your Streamlit programs to online data. You will:

- Send HTTP requests to public APIs
- Read JSON responses and turn them into Python data
- Build tables with `pandas` and show them with `st.dataframe`
- Draw charts with `st.line_chart` and `st.bar_chart`
- Handle errors so the page fails in a clear way instead of crashing

You will work with two programs: **Book Search** and **Major Cities Hourly Weather**. Both programs follow the same pattern:

1. Collect user choices with `Streamlit` widgets.
2. Call an online API with `requests`.
3. Read the JSON response.
4. Build a `pandas.DataFrame`.
5. Show the results as a table and as a chart.

Working with Web APIs in Streamlit

A web **API** is a URL that returns data instead of HTML. Many APIs provide data in JSON format. JSON maps directly to Python types:

- JSON object → Python dictionary
- JSON array → Python list
- JSON number → Python `int` or `float`
- JSON `null` → Python `None`
- JSON strings and booleans are unchanged

The basic steps for retrieving online data in JSON format and using it in Python are as follows:

1. Find the base URL for the data source, for example **https://example.com/api.**
2. Build a `params` dictionary for the query string.
3. Call `requests.get(url, params=params, timeout=10)` to connect to the API.
4. Call `resp.raise_for_status()` to detect HTTP errors.
5. Call `resp.json()` to convert the JSON body to Python data.

In Streamlit, you can enclose your API call in a `try/except` block and display `st.error` if something goes wrong. The following is an example of a Python web app making an API call with **error handling**.

```
import requests
import streamlit as st

url = "https://example.com/api"

params = {"q": "python"}

try:
    resp = requests.get(url, params=params, timeout=10)
    resp.raise_for_status()
    data = resp.json()
except Exception as e:
    st.error(f"Error while calling the API: {e}")
    st.stop()
```

The `try` block runs code that might fail, such as an API request. If an error occurs, the `except` block runs instead, so the app does not crash and can show a clear message. You can also use `raise` to trigger an error on purpose or pass it along, and `finally` to run code at the end no matter what happens, such as cleaning up resources.

Both programs in this lesson use this structure.

Some APIs require an API key, which is a private code for accessing an online service. It is like a password for your application. The APIs used in this unit, Open Library and Open-Meteo, do not require private keys, so this risk does not apply directly to these exercises. However, the following rule is important for future projects: never paste a real API key into an AI chat tool because anything you send in a prompt is shared with the AI service.

When asking AI for help with API code, write YOUR_API_KEY_HERE in the code you share. Replace it with your real key only in your own local file. Never include a real key in a prompt, message, or shared document.

Book Search

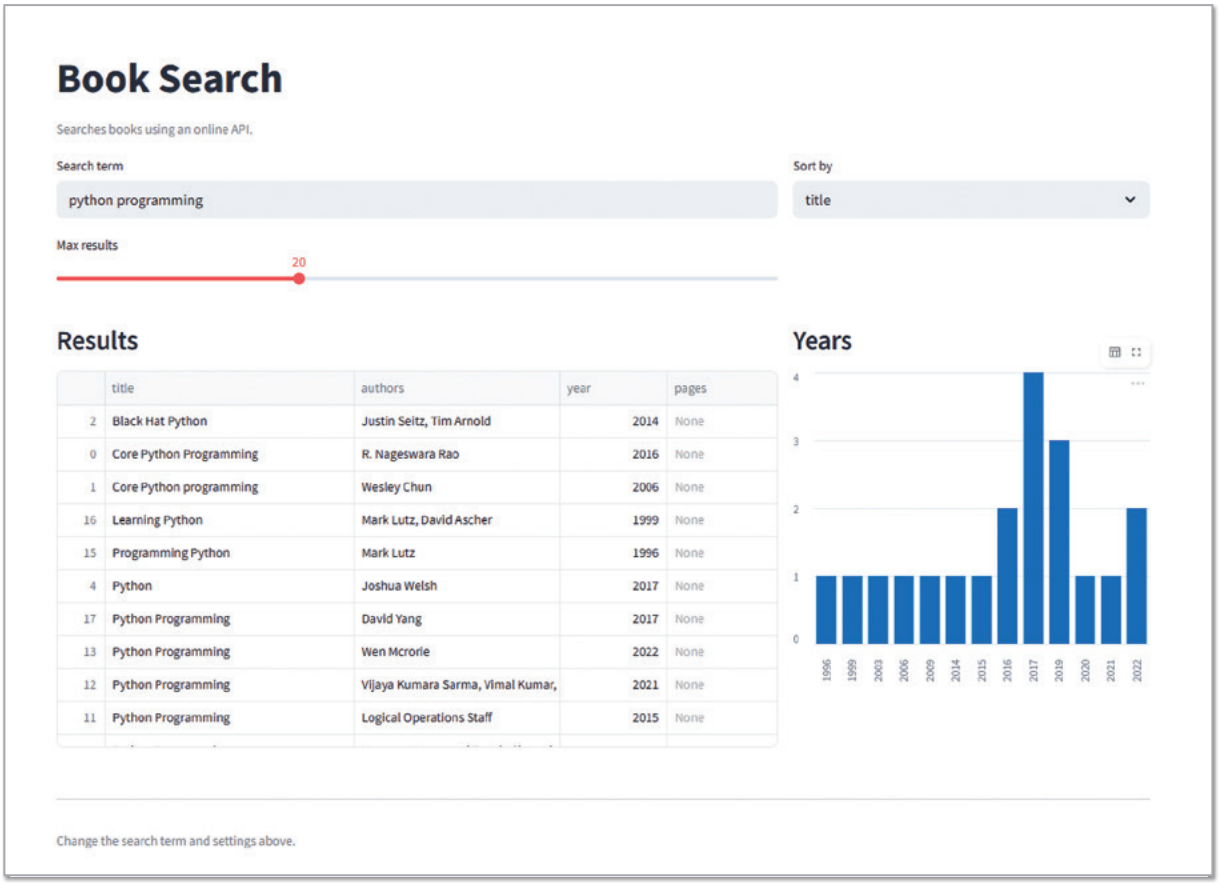
This Book Search program lets you explore books using the Open Library search API.

It lets you:

- Type a search term (for example `python programming`)
- Choose how many results to fetch
- Choose how to sort the results (by title, year, or number of pages)
- See the results in a table
- See a bar chart showing how many books appear per year

This program introduces the following new or important functions:

- It uses `requests.get` with query parameters to retrieve data from a URL.
- It reads a list of book records in JSON format.
- It builds a list of dictionaries and turns it into a `DataFrame`.
- It shows how to sort a `DataFrame` in different ways.
- It uses `st.dataframe` and `st.bar_chart` to display results.



Full program

```
import streamlit as st
import pandas as pd
import requests

st.set_page_config(page_title="Book Search", layout="wide")
st.title("Book Search")

st.caption("Searches books using an online API.")

left, right = st.columns([2, 1])

with left:
    query = st.text_input("Search term", "python programming")
    limit = st.slider("Max results", 5, 50, 20)

with right:
    sort_by = st.selectbox("Sort by", ["title", "year", "pages"], index=0)

if not query.strip():
    st.info("Enter a search term.")
    st.stop()
```

```
params = {
    "q": query,
    "limit": limit,
}

url = "https://openlibrary.org/search.json"

try:
    resp = requests.get(url, params=params, timeout=10)
    resp.raise_for_status()
    data = resp.json()
except Exception as e:
    st.error(f"Error: {e}")
    st.stop()

docs = data.get("docs", [])

rows = []
for d in docs:
    title = d.get("title", "Unknown title")
    authors = ", ".join(d.get("author_name", [])[:3])
    year = d.get("first_publish_year", None)
    pages = d.get("number_of_pages_median", None)
    rows.append(
        {
            "title": title,
            "authors": authors,
            "year": year,
            "pages": pages,
        }
    )

if not rows:
    st.warning("No results found.")
    st.stop()

df = pd.DataFrame(rows)

if sort_by == "year":
    df = df.sort_values(
        ["year", "title"],
        ascending=[True, True],
        na_position="last",
    )
elif sort_by == "pages":
    df = df.sort_values(
        ["pages", "title"],
        ascending=[True, True],
        na_position="last",
    )
else:
```

```
    df = df.sort_values("title")

table_col, chart_col = st.columns([2, 1])

with table_col:
    st.subheader("Results")
    st.dataframe(df)

with chart_col:
    st.subheader("Years")
    year_counts = (
        df["year"]
        .dropna()
        .astype(int)
        .value_counts()
        .sort_index()
    )
    if not year_counts.empty:
        st.bar_chart(year_counts)
    else:
        st.info("No year data.")

st.markdown("---")
st.caption("Change the search term and settings.")
```

The most important parts of this program are the following:

Page setup and title

```
st.set_page_config(page_title="Book Search", layout="wide")
st.title("Book Search")
st.caption("Searches books using an online API.")
```

Layout for controls

```
left, right = st.columns([2, 1])

with left:
    query = st.text_input("Search term", "python programming")
    limit = st.slider("Max results", 5, 50, 20)
with right:
    sort_by = st.selectbox("Sort by", ["title", "year", "pages"], index=0)
Guard for empty search term
if not query.strip():
    st.info("Enter a search term.")
    st.stop()
```

HTTP request and error handling

```
params = {
    "q": query,
    "limit": limit,
}
```

```
    url = "https://openlibrary.org/search.json"
4   try:
        resp = requests.get(url, params=params, timeout=10)
        resp.raise_for_status()
        data = resp.json()
    except Exception as e:
        st.error(f"Error: {e}")
        st.stop()
```

Extracting data for the table

```
5   docs = data.get("docs", [])

    rows = []
    for d in docs:
        title = d.get("title", "Unknown title")
        authors = ", ".join(d.get("author_name", [])[:3])
        year = d.get("first_publish_year", None)
        pages = d.get("number_of_pages_median", None)
        rows.append(
            {

               "title": title,
                "authors": authors,
                "year": year,
                "pages": pages,
            }
        )
```

Creating and sorting the DataFrame

```
7   if not rows:
        st.warning("No results found.")
        st.stop()

    df = pd.DataFrame(rows)

    if sort_by == "year":
        df = df.sort_values(
            ["year", "title"],
            ascending=[True, True],
            na_position="last",
        )
    elif sort_by == "pages":
        df = df.sort_values(
            ["pages", "title"],
            ascending=[True, True],
            na_position="last",
        )
```

```
else:
    df = df.sort_values("title")
```

Showing table and chart side by side

```
6 table_col, chart_col = st.columns([2, 1])

with table_col:

   st.subheader("Results")
    st.dataframe(df)

with chart_col:
    st.subheader("Years")
    year_counts = (
        df["year"]
        .dropna()
        .astype(int)
        .value_counts()
        .sort_index()
    )
    if not year_counts.empty:
        st.bar_chart(year_counts)
    else:
        st.info("No year data.")
```

Discussion Questions

- Why is error handling important when an app depends on online data?
- What is the advantage of turning API results into a DataFrame?

Major Cities Hourly Weather

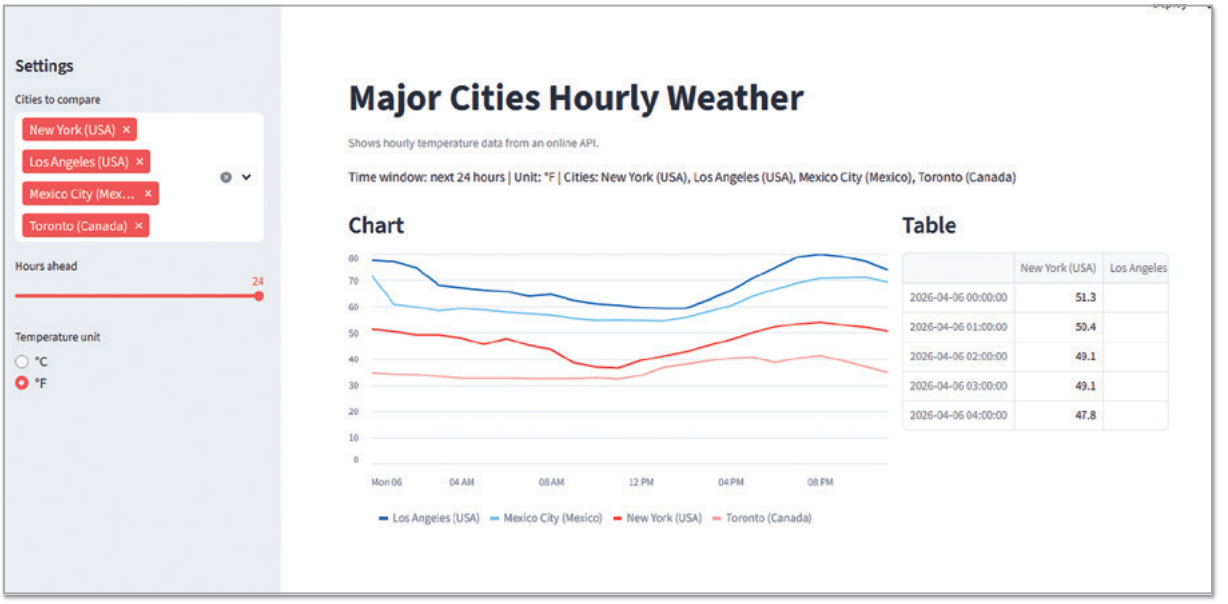
The next program, Major Cities Hourly Weather, compares temperature forecasts for different cities using the Open-Meteo API.

It lets you:

- Pick one or more cities from a list
- Choose how many hours ahead to show
- Switch between Celsius and Fahrenheit
- See a line chart that compares temperatures across cities
- See a table showing the first few hours of data

This program introduces the following new or important functions:

- It uses `st.sidebar` for configuration controls.
- It uses a **multiselect** widget to allow multiple cities to be selected.
- It keeps a shared time index across several data series.
- It combines multiple `pandas.Series` objects into one `DataFrame`.
- It shows warnings and errors from multiple API calls.



Full program

```
import streamlit as st
import pandas as pd
import requests

1 st.set_page_config(page_title="Major Cities Hourly Weather", layout="wide")
st.title("Major Cities Hourly Weather")

st.caption("Shows hourly temperature data from an online API.")

2 CITY_COORDS = {
    "New York (USA)": (40.7128, -74.0060),
    "Los Angeles (USA)": (34.0522, -118.2437),
    "London (UK)": (51.5074, -0.1278),
    "Paris (France)": (48.8566, 2.3522),
    "Berlin (Germany)": (52.5200, 13.4050),
    "Tokyo (Japan)": (35.6762, 139.6503),
    "Seoul (South Korea)": (37.5665, 126.9780),
    "Sydney (Australia)": (-33.8688, 151.2093),
    "Toronto (Canada)": (43.6510, -79.3470),
    "Mexico City (Mexico)": (19.4326, -99.1332),
}

3 st.sidebar.header("Settings")

selected_cities = st.sidebar.multiselect(
    "Cities to compare",
    list(CITY_COORDS.keys()),
    default=["New York (USA)", "London (UK)", "Tokyo (Japan)"],
)

hours_to_show = st.sidebar.slider(
    "Hours ahead",
    min_value=4,

    max_value=24,
    value=8,
    step=1,
)

unit_choice = st.sidebar.radio(
    "Temperature unit",
    ["°C", "°F"],
    index=0,
)

4 if not selected_cities:
    st.info("Select at least one city.")
    st.stop()
```

When programs are small, it is common to keep everything in one file and write code directly where you need it. That works at first, but as the app grows, organization starts to matter more than speed.

Storing city information in one dictionary keeps the code organized. To add a new city, you only update this one section.

```
st.markdown(
    f"Time window: next {hours_to_show} hours | "
    f"Unit: {unit_choice} | "
    f"Cities: {', '.join(selected_cities)}"
)

API_URL = "https://api.open-meteo.com/v1/forecast"

series_list = []
city_labels = []
time_index = None
error_messages = []

for city in selected_cities:
    lat, lon = CITY_COORDS[city]

    params = {
        "latitude": lat,
        "longitude": lon,
        "hourly": "temperature_2m",
        "timezone": "UTC",
        "forecast_days": 1,
    }

    try:
        resp = requests.get(API_URL, params=params, timeout=10)
        resp.raise_for_status()
        data = resp.json()
    except Exception as e:
        error_messages.append(f"{city}: {e}")
        continue

    hourly = data.get("hourly", {})
    times = hourly.get("time", [])
    temps_c = hourly.get("temperature_2m", [])

    if not times or not temps_c:
        error_messages.append(f"{city}: no data.")
        continue

    if time_index is None:
        times_slice = times[:hours_to_show]
        time_index = pd.to_datetime(times_slice)
    else:
        times_slice = times[: len(time_index)]
        if len(times_slice) < len(time_index):
            times_slice += [None] * (len(time_index) - len(times_slice))

    temps_c_slice = temps_c[: len(time_index)]
```

5

Keeping important constants together makes the app easier to maintain and update later.

This loop is built to work for any city in the dictionary. That is a scalable design because the app can grow without repeating code.

6

```
    if len(temps_c_slice) < len(time_index):
        temps_c_slice += [None] * (len(time_index) - len(temps_c_slice))

    if unit_choice == "°F":
        temps = [(t * 9 / 5 + 32) if t is not None else None for t in
temps_c_slice]
    else:
        temps = temps_c_slice

    s = pd.Series(temps, index=time_index)
    series_list.append(s)
    city_labels.append(city)

if not series_list:
    st.error("No data available.")
    if error_messages:
        with st.expander("Details"):
            for msg in error_messages:
                st.write(msg)
    st.stop()

df = pd.concat(series_list, axis=1)
df.columns = city_labels

left, right = st.columns([2, 1])

with left:
    st.subheader("Chart")
    st.line_chart(df)

with right:
    st.subheader("Table")
    st.dataframe(df.head().round(1))

if error_messages:
    with st.expander("Warnings"):
       for msg in error_messages:
            st.write(msg)
```

7.3 **The important points to note in the program are:**

Page setup and caption

```
1 st.set_page_config(page_title="Major Cities Hourly Weather", layout="wide")
  st.title("Major Cities Hourly Weather")

  st.caption("Shows hourly temperature data from an online API.")
```

Static dictionary of cities and coordinates

```
2 CITY_COORDS = {
      "New York (USA)": (40.7128, -74.0060),
      "Los Angeles (USA)": (34.0522, -118.2437),
      "London (UK)": (51.5074, -0.1278),
      "Paris (France)": (48.8566, 2.3522),
      "Berlin (Germany)": (52.5200, 13.4050),
      "Tokyo (Japan)": (35.6762, 139.6503),
      "Seoul (South Korea)": (37.5665, 126.9780),
      "Sydney (Australia)": (-33.8688, 151.2093),
      "Toronto (Canada)": (43.6510, -79.3470),
      "Mexico City (Mexico)": (19.4326, -99.1332),
  }
```

Controls in the sidebar

```
3 st.sidebar.header("Settings")
  selected_cities = st.sidebar.multiselect(
      "Cities to compare",
      list(CITY_COORDS.keys()),
      default=["New York (USA)", "London (UK)", "Tokyo (Japan)"],
  )

  hours_to_show = st.sidebar.slider(
      "Hours ahead",
      min_value=4,
      max_value=24,
      value=8,
      step=1,
  )

unit_choice = st.sidebar.radio(
    "Temperature unit",
    ["°C", "°F"],
    index=0,
)
```

Guard if no cities are selected

```
4 if not selected_cities:
      st.info("Select at least one city.")
      st.stop()

  Short summary line for current settings
  st.markdown(
      f"Time window: next {hours_to_show} hours | "
```

```
        f"Unit: {unit_choice} | "
        f"Cities: {', '.join(selected_cities)}"
    )
```

Loop over cities and call the API

```
5 API_URL = "https://api.open-meteo.com/v1/forecast"

    series_list = []
    city_labels = []

    time_index = None
    error_messages = []

    for city in selected_cities:
        lat, lon = CITY_COORDS[city]

        params = {
            "latitude": lat,
            "longitude": lon,
            "hourly": "temperature_2m",
            "timezone": "UTC",
            "forecast_days": 1,
        }

        try:
            resp = requests.get(API_URL, params=params, timeout=10)
            resp.raise_for_status()
            data = resp.json()
        except Exception as e:
            error_messages.append(f"{city}: {e}")
            continue
```

The example uses UTC (Coordinated Universal Time) so all cities can be compared on the same time scale.

Extracting times and temperatures, managing a shared time index

```
6       hourly = data.get("hourly", {})
        times = hourly.get("time", [])
        temps_c = hourly.get("temperature_2m", [])

        if not times or not temps_c:
            error_messages.append(f"{city}: no data.")
            continue

        if time_index is None:
            times_slice = times[:hours_to_show]

            time_index = pd.to_datetime(times_slice)
        else:
            times_slice = times[: len(time_index)]
            if len(times_slice) < len(time_index):
                times_slice += [None] * (len(time_index) - len(times_slice))
```

```
        temps_c_slice = temps_c[: len(time_index)]
        if len(temps_c_slice) < len(time_index):
            temps_c_slice += [None] * (len(time_index) - len(temps_c_slice))

    Unit conversion and building the series
        if unit_choice == "°F":
            temps = [(t * 9 / 5 + 32) if t is not None else None for t in temps_c_
    slice]
        else:
            temps = temps_c_slice

        s = pd.Series(temps, index=time_index)
        series_list.append(s)
        city_labels.append(city)
```

Handling the case where all API calls fail

```
7 if not series_list:
        st.error("No data available.")
        if error_messages:
            with st.expander("Details"):
                for msg in error_messages:
                    st.write(msg)
        st.stop()Combining series and showing chart plus table
8 df = pd.concat(series_list, axis=1)
    df.columns = city_labels
    left, right = st.columns([2, 1])

    with left:
        st.subheader("Chart")
        st.line_chart(df)

    with right:
        st.subheader("Table")
        st.dataframe(df.head().round(1))
```

Optional warnings

```
9 if error_messages:
        with st.expander("Warnings"):
            for msg in error_messages:
                st.write(msg)
```

Career connections

APIs, tables, and charts are used by software developers, data analysts, business intelligence teams, and AI developers who build tools that turn raw data into useful information.

You have now connected Streamlit programs to external APIs and turned the responses into tables and charts. Next, you will learn how AI can help refine these interfaces—improving labels, messages, and layouts without changing the underlying logic.

Exercises

Answer each set of questions in your notebook.

1. Explain what a web API is and how it is used in a Streamlit app.
2. Describe how JSON data can be turned into Python data structures.
3. Explain why `try` and `except` blocks are useful when working with online APIs.
4. Describe how a DataFrame can be used to organize and display API data.
5. Suggest one data app similar to the examples in this lesson that uses a public API, and explain what table or chart it could display.
6. Explain why API error handling is important even when the code is correct.
7. Describe the difference between JSON data and a DataFrame.
8. Add one more sort option to the Book Search app.
9. Extend the weather app to display the average temperature for each city.

LESSON 4

Designing with AI: Mockups and Prototypes

You now have several working Streamlit programs:

- Simple tools with text inputs, sliders, and checkboxes
- Larger pages with forms and previews
- Data-driven pages that call web APIs and draw charts

In this lesson, you will learn how to use an AI chatbot to improve your programs. You will discover how to:

- Ask the AI chatbot to rewrite labels and messages without breaking the code
- Ask for ideas that add useful features on top of your current logic
- Generate UI mockups from descriptions
- Evaluate and refine AI-generated designs

You stay in control the whole time:

- You decide what to share with the AI.
- You decide what to change.
- You can always keep the original version.

You will work with two programs: **Issue Reporter** (to refine labels and messages) and **Major Cities Hourly Weather** (to develop a new summary feature).

Use AI Safely

When using AI tools, do not upload or share personal data, passwords, API keys, private school information, or confidential project content. Provide only the code or text necessary for the task. Treat AI-generated output as a draft and review it carefully to ensure clarity, accessibility, and accuracy.

How to Ask the AI Chatbot to Improve a Program

When you ask the AI chatbot for help with code, how you ask makes a difference. The following tips can help you write effective AI chatbot prompts:

1. **Share only what is needed.**
   Copy a small section of the program. For example:
   A block that contains the labels and messages you want to change, or the part where you plan to add a new feature.

2. **Say what must not change.**
   Be very clear about what should stay the same. For example:
   - "Do not change any variable names."
   - "Do not change the logic; only rewrite text inside quotes."
   - "Keep the structure of the program; just suggest an extra section I can add at the end."

3. **Say what you want to improve.**
   For example:
   - "Make labels clearer for high school students."
   - "Make error messages shorter and more direct."
   - "Suggest an extra feature that helps users read this chart."

4. **Check the answer before using it.**
   Make sure the answer maintains the program's core functionality. If something is wrong or too complex, you can ask for a simpler version or ignore the parts you do not like.

You can think of the AI chatbot as a design partner for your program, but you are in charge.

Issue Reporter: Improving Labels and Messages

Let's start with the Issue Reporter from the earlier lesson. The program works well, but the labels and messages are quite generic. To improve them, you can copy the code block containing the labels and messages into the AI chatbot and ask it:

AI Tool

The prompt:

This is part of a Streamlit program for reporting issues.

Improve only the text inside quotes so it is clearer and more helpful for high school students.

See the AI References and Resources section for details on this AI generated content.

The AI chatbot will respond with a modified version of the code in which only the strings have changed. Here is one possible result you might accept, in which only the text has changed and the logic has stayed the same:

AI Tool

The output:

```
import streamlit as st

st.set_page_config(page_title="Issue Reporter", layout="wide")

st.title("Report an issue")

title = st.text_input("Short issue title")
issue_type = st.radio(
    "What kind of issue is this?",
    ["Bug", "Feature request", "Question"],
    horizontal=True,
)
project = st.selectbox(
    "Which project is affected?",
    ["Core API", "Frontend UI", "Data pipeline", "Other"],
)

form_col, preview_col = st.columns([2, 1])

with form_col:
    st.subheader("Issue details")

    with st.form("issue_form"):

        priority = st.selectbox(
            "Priority",
            ["Low", "Medium", "High", "Critical"],
            index=0,
        )
        description = st.text_area(
            "Describe what happens and what you expected",
            height=150,
        )

        submitted = st.form_submit_button("Submit issue")

    if submitted:
        st.success("Issue saved. Check the preview on the right.")

with preview_col:
    st.subheader("Issue preview")

    if not title and not description:
        st.info("Start filling in the fields on the left to see the
issue preview here.")
    else:
        st.write(f"**[{priority}] {issue_type}: {title or '(no
title)'}**")
        st.write(f"**Project:** {project}")
        st.markdown("**Details:**")
        st.write(description or "(no description yet)")
```

Points to notice:

- The variables and logic are unchanged.
- Only the text inside quotes has changed.
- The program is now more self-explanatory for someone who has never used an issue tracker.

This is the kind of change AI chatbots are very good at: rewriting text, not changing behavior.

Discussion Questions

- Why should you clearly state what must not change when asking the AI chatbot to help with code?
- Why is asking the AI chatbot to rewrite labels and messages usually safer than asking for a full code rewrite?
- What should you check before using AI-generated code in your program?

Career connections

Improving labels, messages, and useful features is part of the work of software developers, UX writers, product designers, and AI-enabled application teams. These professionals focus on making digital tools clearer, more accessible, and more responsive to user needs, ensuring technology communicates effectively and supports a positive user experience.

Major Cities Hourly Weather: Adding a Summary Feature

For the weather program, you will go further and add a feature. You start with a working program that already:

- Retrieves hourly temperature data from Open-Meteo
- Lets you choose cities, a time window, and a unit
- Displays a line chart and a data table

You will now add an automatic summary that provides the user with:

- A table showing the minimum, maximum, and average temperature for each city
- A short note identifying which city is currently the warmest and which is the coldest

The program you have calculates a `DataFrame` called `df` and then shows:

```
...

df = pd.concat(series_list, axis=1)
df.columns = city_labels

left, right = st.columns([2, 1])

with left:
    st.subheader("Hourly temperature comparison")
    st.line_chart(df)

with right:
    st.subheader("Table (first few rows)")
    st.dataframe(df.head().round(1))

...
```

The new feature will come after this code, using the same `df` and `city_labels`. You can describe the situation to the AI chatbot as follows:

AI Tool

The prompt:

I have a Streamlit program that compares hourly temperatures for several cities.

I already have a line chart and a small table.

Suggest a simple extra feature that helps users understand the data better.

The feature should use the existing df DataFrame and city_labels, and I should be able to add it at the end of the program in one block.

The AI chatbot will likely suggest min/max/average values or note the warmest and coldest cities. You can then refine the request:

AI Tool

The prompt:

Write Streamlit code that uses df and city_labels to show:

A table with min, max, and average temperature per city.

One line saying which city is currently warmest and which is currently coldest.

Assume df is not empty and its columns are the city names.

Put everything in a new section I can paste at the end of the program.

See the AI References and Resources section for details on this AI generated content.

You can then check the answer and adjust it if needed.

Here is a summary section you can add at the end of your existing program, right after the chart and table and the `if error_messages` block:

AI Tool

The output:

```
st.markdown("---")
st.subheader("Summary for this time window")

if df.empty:
    st.info("No data to summarize.")
else:
    col_min = df.min()
    col_max = df.max()
    col_mean = df.mean()

    summary_rows = []
    for city in city_labels:
        summary_rows.append(
            {
                "City": city,
                "Min": col_min[city],
                "Max": col_max[city],
                "Average": col_mean[city],
            }
        )

    summary_df = pd.DataFrame(summary_rows)
    summary_df[["Min", "Max", "Average"]] = summary_df[["Min", "Max",
"Average"]].round(1)
    st.dataframe(summary_df, hide_index=True)

    last_row = df.iloc[-1]
    warmest_city = last_row.idxmax()
    coldest_city = last_row.idxmin()
    warmest_value = last_row[warmest_city]
    coldest_value = last_row[coldest_city]

    st.markdown(
        f"Warmest now: **{warmest_city}** at **{warmest_value:.1f}{unit_
choice}**"
    )
    st.markdown(
        f"Coldest now: **{coldest_city}** at **{coldest_value:.1f}{unit_
choice}**"
    )
```

7.4 **What this does:**

- `col_min`, `col_max`, `col_mean` compute column-wise statistics per city.
- `summary_rows` and `summary_df` build a table with one row per city.
- `last_row` looks at the latest forecast time and finds the warmest and coldest city at that time.
- Two `st.markdown` calls display short, natural-language summaries.

This is a real functional change. The program now tells you something new about the data without you reading the chart in detail.

In this lesson, you used the AI chatbot in two different ways:

- To rewrite labels and messages in Issue Reporter while keeping the logic identical
- To design and add a summary section to the Major Cities Hourly Weather program that computes real statistics

The general pattern is:

1. Start from a working program.
2. Decide what you want to improve: wording, layout, or new features.
3. Give the AI chatbot a precise description of the existing code and clear rules about what must stay the same.
4. Take the useful parts of the answer and integrate them carefully into your program.

This is the kind of feature that the AI chatbot can help you design:

- You describe the program and the data you already have.
- You ask for a specific kind of summary.
- You check the code it proposes and adjust it so it fits your variable names and style.

AI as a Design Assistant

Creating detailed mockups by hand can be slow, especially if drawing is not an activity you enjoy. An AI tool that generates interface mockups from text can help you see what your wireframe might look like as a more polished screen, compare several versions of the same layout, and explore layout ideas such as spacing, grouping, and alignment that you might not have thought of.

At the same time, AI can also introduce problems. It might ignore your accessibility choices, hide important actions, or add distracting decoration. That is why you treat it as an assistant, not an automatic designer. You give clear instructions, you review the result, and you decide what to keep.

From wireframe to prompt

To use AI effectively, start with a solid wireframe and a clear description of the screen. Before you write any prompt, you should already know what the main purpose of the screen is, which elements must appear (for example the title, inputs, buttons, and messages), and how those elements are grouped on the page.

You then turn this into a short, precise description that an AI tool can follow. Think of it as "describing your wireframe in words."

For example, instead of writing "make a nice dashboard," you might say something like:

Design a simple dashboard screen for students. At the top, show the title "My Courses" and a short subtitle. Below that, show two main sections side by side: on the left, a list of current courses with course names and progress bars and on the right, a smaller panel titled "Upcoming deadlines" with a list of due dates. Use clear headings, high contrast text, and a calm, readable style. The layout should be easy to scan on a laptop screen.

This description tells the AI what the screen is for, what must be on it, how things are arranged, and what style you expect. If your wireframe is weak or vague, the prompt will be weak or vague as well, so the quality of your description depends on the quality of your earlier design work.

Evaluating an AI mockup

When an AI tool produces a mockup from your description, the first question is not "Do I like how this looks?" The first question is "Does this still respect my design decisions?"

Compare the mockup with your wireframe. Check whether the main purpose of the screen is still obvious and whether the main action is still easy to find. Look for any important elements that are missing or pushed out of sight. Notice whether the tool has added new items that do not belong to the task you described. Then review the mockup from your UX and accessibility perspective. Ask yourself if there is enough contrast between text and background, if the text appears large enough to read, and if any meaning depends only on color. Tiny labels, faint text, and decorative effects that overpower the main content are all warning signs.

If the mockup breaks your structure, hides important actions, or creates accessibility issues, you do not have to accept it. You can adjust your prompt by saying, for example, "Keep the two main panels the same size," or "Make the main button more visible," or "Use darker text for better contrast," and try again. You can also simply discard a version that goes in the wrong direction. Your wireframe is the reference point. The AI output is something you critique and edit.

Career connections

AI-assisted mockup generation can support the work of UX designers, product designers, and frontend teams, but human judgment is still needed to make design decisions.

Exercises

Answer each set of questions in your notebook.

1. Describe why it is important to state clearly what must not change when asking an AI chatbot to help with code.
2. Explain how an AI chatbot can improve labels and messages without changing program logic.
3. Describe one example of a useful feature an AI chatbot could help add to an existing data app.
4. Explain why it is important to avoid entering private or sensitive information into AI tools.
5. Describe two problems that may appear in an AI-generated mockup.
6. Describe two things you should check when evaluating an AI-generated mockup.
7. Choose one app from the earlier lessons and explain one UI improvement and one feature improvement you could ask an AI chatbot to help you make.
8. Ask an AI assistant to improve the wording of one earlier app, then compare the version before and after.
9. Add one small feature to an existing app and document what the AI suggested versus what you kept.

Project

An Interactive Data Application

In this project, you will build your own interactive data app in Streamlit. Your app should collect user input, process or retrieve data, and present the result in a clear and useful way. You may base your work on one of the ideas from this unit, such as a data viewer, comparison tool, or dashboard, or you may create a similar app of your own.

1. **Choose an app idea**

   Decide what your app will do and what problem it helps the user solve. Your app should have a clear purpose and should use at least two input controls, such as a text input, slider, selectbox, checkbox, radio button, or button.

2. **Build the interface**

   Create the app in Streamlit with a clear title and layout. Organize the inputs in a logical way using the main page, columns, forms, or the sidebar.

3. **Process or retrieve data**

   Your app should either:

   - Process user input using Python logic, or
   - Fetch data from a public web API and convert it into a usable form.

4. **Display the results**

   Show the results clearly using one or more of the following:

   - Formatted text
   - A table or DataFrame
   - A bar chart or line chart

5. **Improve the user experience**

   Add clear labels, messages, and validation. If helpful, you may use an AI chatbot to improve the wording of labels and messages or to suggest one extra feature, but you should review all suggestions and decide what to keep.

6. **Reflect on your design**

   Write a 200- to 350-word reflection on one responsible design choice in your app.

6. **(cont.) - Reflection Guide**

   Choose one:

   - Accessibility: Is your app easy to read and use for a wide range of people?
   - Data minimization: Does your app only collect the information it actually needs?
   - Transparency: Does your app clearly explain what it does, what the inputs mean, and what happens when something goes wrong?
   - Inclusive input design: Does your app work well for different kinds of names, dates, or user entries?
   - Efficiency: Does your app avoid unnecessary processing or repeated API requests?

   Explain:

   - which area you chose and what you tested, checked, or reviewed in your app
   - what you found and one improvement you would make with more time

   Be specific and honest. Use details from your actual app. Use the following expectations as a guide for the depth and quality of your reflection:

   - Strong: Clearly explains one responsible design choice, describes how it was checked, gives specific findings, and suggests a realistic improvement
   - Acceptable: Identifies one responsible design choice and gives a general reflection
   - Needs Revision: Reflection is too vague, incomplete, or unsupported

7

Wrap-Up

This unit covered how to:

> explain how Streamlit reruns scripts when widget values change.

> collect user input using widgets such as text inputs, sliders, selectboxes, and buttons.

> display results using Streamlit functions such as st.write, st.dataframe, and charts.

> group related inputs using forms.

> organize app layouts using columns and the sidebar.

> connect apps to public web APIs.

> convert JSON API responses into Python data structures and DataFrames.

> visualize data using tables, bar charts, and line charts.

> improve app text and features using AI-assisted suggestions.

Key Terms

- API
- bar chart
- button
- checkbox
- DataFrame
- error handling
- form
- JSON
- line chart
- multiselect
- radio button
- request
- selectbox
- sidebar
- slider
- submit button
- text input
- widget