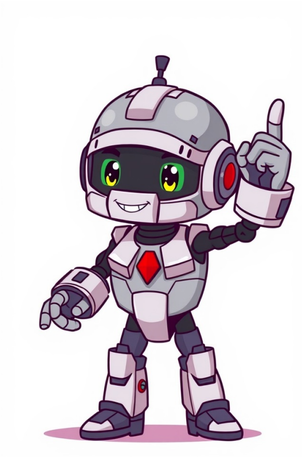
I'm not a robot



Tip: Sign in to track your progress - it's free. With our "Try it Yourself" editor, you can edit Python code and view the result. print("Hello, World!") Try it Yourself » Click on the "Try it Yourself" button to see how it works. Python File Handling In our File Handling section you will learn how to open, read, write, and delete files. Python Database Handling In our database section you will learn how to access and work with MySQL and MongoDB databases: Python MySQL Tutorial Python MongoDB Tutorial Python Exercises Many chapters in this tutorial end with an exercise where you can check your level of knowledge. See all Python Exercises Python Examples Learn by examples! This tutorial supplements all explanations with clarifying examples. See All Python Examples Python Quiz Test your Python skills with a quiz. Python Quiz View your completed tutorials, exercises, and quizzes Keep an eye on your progress and daily streaks Set goals and create learning paths Create your own personal website Sign Up for Free Note: This is an optional feature. You can study at W3Schools without creating an account. Python Reference You will also find complete function and method references: Download Python Download Python from the official Python web site: Get certified by completing the course Get certified w3schools CERTIFIED . 2025 In this section, you'll learn basic Python. If you're completely new to Python programming, this Python basics section is perfect for you.After completing the tutorials, you'll be confident in Python programming and be able to create simple programs in Python.Syntax – introduce you to the basic Python programming syntax.Variables – explain to you what variables are and how to create concise and meaningful variables.Strings – learn about string data and some basic string operations.Numbers – introduce to you the commonly-used number types including integers and floating-point numbers.Booleans – explain the Boolean data type, falsy and truthy values in Python.Constants – show you how to define constants in Python.Comments – learn how to make notes in your code.Type conversion – learn how to convert a value of one type to another e.g., converting a string to a number.Arithmetic operators – learn how to use arithmetic operators to perform mathematical operations.Assignment operators – guide on how to use assignment operators to assign values to variables.Comparison operators – introduce you to the comparison operators and how to use them to compare two values.Logical operators – show you how to use logical operators to combine multiple conditions.if...else statement – learn how to execute a code block based on a condition.Ternary operator – introduce you to the Python ternary operator that makes your code more concise.for loop with range() – show you how to execute a code block for a fixed number of times by using the for loop with range() function.while– show you how to execute a code block as long as a condition is True.break – learn how to exit a loop prematurely.continue – show you how to skip the current loop iteration and start the next one.pass – show you how to use the pass statement as a placeholder.Python functions – introduce you to functions in Python, and how to define functions, and reuse them in the program.Default parameters – show you how to specify the default values for function parameters.Keyword arguments – learn how to use the keyword arguments to make the function call more obvious.Recursive functions – learn how to define recursive functions in Python.Lambda Expressions – show you how to define anonymous functions in Python using lambda expressions.Docstrings – show you how to use docstrings to document a function.Dictionary – introduce you to the dictionary type.Dictionary comprehension – show you how to use dictionary comprehension to create a new dictionary from an existing one.Set – explain to you the Set type and show you how to manipulate set elements effectively.Set comprehension – explain to you the set comprehension so that you can create a new set based on an existing set with a more concise and elegant syntax.Union of Sets – show you how to union two or more sets using the union() method or set union operator (|).Intersection of Sets – show you how to intersect two or more sets using the intersection() method or set intersection operator (&).Difference of sets – learn how to find the difference between sets using the set difference() method or set difference operator (-).Symmetric Difference of sets – guide you on how to find the symmetric difference of sets using the symmetric_difference() method or the symmetric difference operator (^).Subset – check if a set is a subset of another set.Superset – check if a set is a superset of another set.Disjoint sets – check if two sets are disjoint.try...except – show you how to handle exceptions more gracefully using the try...except statement.try...except...finally – learn how to execute a code block whether an exception occurs or not.try...except...else – explain to you how to use the try...except...else statement to control the follow of the program in case of exceptions.for...else – explain to you the for else statement.while...else – discuss the while else statement.do...while loop emulation – show you how to emulate the do...while loop in Python by using the while loop statement.Unpacking tuples – show you how to unpack a tuple that assigns individual elements of a tuple to multiple variables.*args Parameters – learn how to pass a variable number of arguments to a function.**kwargs Parameters – show you how to pass a variable number of keyword arguments to a function.Partial functions – learn how to define partial functions.Type hints – show you how to add type hints to the parameters of a function and how to use the static type checker (mypy) to check the type statically.Modules – introduce you to the Python modules and show you how to write your own modules.Module search path – explain to you how the Python module search path works when you import a module.__name__ variable – show you how to use the __name__ variable to control the execution of a Python file as a script or as a module.Packages – learn how to use packages to organize modules in more structured ways.Private functions – define private functions in Python.F-strings – learn how to use the f-strings to format text strings in a clear syntax.Raw strings – use raw strings to handle strings that contain the backslashes.Backslash – explain how Python uses the backslashes (\) in string literals. Notice: While JavaScript is not essential for this website, your interaction with the content will be limited. Please turn JavaScript on for the full experience. ⚡Support Python for everyone by grabbing a 30% discount on PyCharm- ALL proceeds go to the Python Software Foundation. Offer ends soon, so grab it today! GET 30% OFF PYCHARM With the popularity of Python comes a multitude of resources available to learn to code using this language, including online courses, certification, boot camps, and more. Here are some important things to know:Python was one of the highest-demand programming languages in 2025 among recruiters [1].Additionally, Python's versatility means that adding this programming language to your repertoire can help you work in a wide range of roles, including data scientist, artificial intelligence engineer, and software developer.You build your Python skills by taking a quiz, course, or a more in-depth certification.Learn about essential Python programming skills to focus on developing, how to go about learning them, and various Python career possibilities. Then, consider enrolling in the University of Michigan's Python for Everybody Specialization, where you'll build foundational programming knowledge in as little as two months. Essential Python coding skillsTo work with the Python programming language, you need to have skills and knowledge within areas such as data structures, Python libraries, machine learning, and many more. Fortunately, Python is a beginner-friendly programming language thanks to its efficient syntax, making it less intimidating so you can feel confident taking the first step toward your goals. However, Python isn't only for beginners as it offers an extensive list of tools, libraries, and frameworks to help you increase your productivity and complete a broad range of tasks.Basic Python coding skillsIf you're just getting started in Python, you can focus on getting comfortable with several core concepts. These include functions, loops, variables, and data structures. Different data types to know are strings, lists, numbers, sets, tuples, and dictionaries.Intermediate Python coding skillsAfter gaining proficiency in the fundamentals of Python, you can venture out to more intermediate concepts such as learning to use popular frameworks and libraries. Frameworks and libraries are especially valuable because they allow you to reuse code and perform tasks in specific areas such as machine learning and data science. Common Python frameworks and libraries to know are Django, Flask, CherryPy, Matplotlib, NumPy, and pandas.Advanced Python coding skillsArtificial intelligence skills are in high demand, and Python is the most popular programming language for developing these technologies. Python has several machine learning frameworks and libraries, giving you the ability to process massive amounts of data. Machine learning, deep learning, and natural language processing are possible through libraries such as PyTorch, TensorFlow, NLTK, and scikit-learn.How to improve Python coding skillsTo improve your Python coding skills, you can choose from a variety of resources—online courses, boot camps, and certifications—along your learning journey. While some people get their start in coding by earning a degree such as a bachelor's in computer science, you have plenty of other options, many of which you can access online. Learn how you can develop a robust Python skill set.Master the core concepts.Step one of improving your Python coding skills is to get comfortable with the basics. This will lay a strong foundation on which you can continue to build. It's key to have proficiency in fundamental components such as data types, variable types, object-oriented programming, generators, and iterators.Take an online course.Online courses are a valuable and often free resource for you to grow your Python skills. Whether you're a beginner, an advanced programmer, or anywhere in between, you can improve your Python coding skills on Coursera with credentials such as the Python for Everybody Specialization from the University of Michigan. With this Specialization, you can learn about Python programming, data structures, algorithms, and more. Enroll in a boot camp.Coding boot camps are a great way for you to learn how to code quickly while also ensuring you build the necessary skills for success in real-world environments. Generally, boot camps last around four months but can sometimes extend beyond six months. During this time period you will have a chance to get practical experience working on Python projects. By learning new skills through a coding boot camp, you can also potentially increase your salary.Get certified.Earning a Python certification will help you not only to validate your abilities to employers but also to stand out among other candidates. The Python Institute offers a range of certifications, from entry-level options to more advanced levels. For example, you can earn the PCEP – Certified Entry-Level Python Programmer credential and follow with the PCAP – Certified Associate Python Programmer, PCPP1 – Certified Professional Python Programmer Level 1, and PCPP2 – Certified Professional Python Programmer Level 2. Another certification option for you to consider is the Certified Python Developer credential, provided by the Global Tech Council.Use GitHub.GitHub is a place you can go to view code, have others review your code, collaborate, and share your work. This cloud-based platform also offers a student developer pack, where you can gain access to resources to learn programming skills. Take part in a hackathon.Hackathons are competitive events that typically take place over a multiple-day span where programmers get together to solve a problem. Here you will often work as part of a team, working toward a solution and challenging your coding skills along the way. Not only can you learn to use new tools and techniques at a hackathon, but it also serves as a great networking opportunity.Improve your Python coding skills on CourseraYou can continuously improve your Python coding skills by taking advantage of resources such as boot camps, certifications, and online courses. Get started today with our Python Skill Assessment, or explore these other free resources: Watch on YouTube: How Long Does It Really Take to Learn Python? Hear from an expert: 6 Questions with a Microsoft Data Analytics Leader Get our insights: Subscribe to our Career Chat newsletterWhether you want to develop a new skill, get comfortable with an in-demand technology, or advance your abilities, keep growing with a Coursera Plus subscription. You'll get access to over 10,000 flexible courses. Python is one of the most popular programming languages. It's simple to use, packed with features and supported by a wide range of libraries and frameworks. Its clean syntax makes it beginner-friendly. A high-level language, used in data science, automation, AI, web development and more.Known for its readability, which means code is easier to write, understand and maintain.Backed by strong library support, we don't have to build everything from scratch.Basic Code ExampleThe following is a simple program that displays the message "Hello, World!" on the screen. Python # Python Program to Print Hello World! print("Hello World!") To understand working of this code, refer to our article Python Introduction.Why Learn Python?Requires fewer lines of code compared to other programming languages like Java.Provides Libraries / Frameworks like Django, Flask and many more for Web Development, and Pandas, Tensorflow, Scikit-learn and many more for, AI/ML, Data Science and Data Analysis Cross-platform, works on Windows, Mac and Linux without major changes.Used by top tech companies like Google, Netflix and NASA. Many Python coding job opportunities in Software Development, Data Science and AI/ML.Try our ongoing free course Python Skillup with weekly topic coverage, notes, daily quizzes and coding problems.BasicsIn this section, we'll cover the basics of Python programming, including installing Python, writing first program, understanding comments and working with variables, keywords and operators. Before starting to learn python we need to install python on our system.FunctionsIn this section of Python 3 tutorial we'll explore Python function syntax, parameter handling, return values and variable scope. Along the way, we'll also introduce versatile functions like range(), map, filter and lambda functions. Data StructuresPython offers versatile collections of data types, including lists, string, tuples, sets, dictionaries and arrays. In this section, we will learn about each data types in detail.Python's collections module offers essential data structures, including the following:To learn data structure and algorithm with python in detail, you can refer to our DSA with Python Tutorial.OOP ConceptsIn this section, we'll explore the core principles of object-oriented programming (OOP) in Python. From encapsulation to inheritance, polymorphism, abstract classes and iterators, we'll cover the essential concepts that helps you to build modular, reusable and scalable code. Exception HandlingIn this section, we'll explore Python Exception Handling that how Python deals with unexpected errors, enabling us to write fault-tolerant code. We'll cover file handling, including reading from and writing to files.File HandlingIn this section, we will cover file handling, including reading from and writing to files.Database HandlingIn this section we will learn how to access and work with MySQL and MongoDB databasesPython MongoDB TutorialPython MySQL TutorialPackages or LibrariesPython is a huge collection of Python Packages standard libraries that make development easier. These libraries help with a wide range of tasks and can save you a lot of time by providing ready-to-use tools.Some commonly used types of libraries in Python include:PackagesBuilt-in Modules DSA LibrariesGUI LibrariesData Science 1. Foundational Libraries: These are the libraries that form the base for all data science work. Start here to build a strong foundation.2. Advanced Visualization and Statistical Tools: Once you're comfortable with basic data handling and visualization, move to creating cleaner visuals and performing statistical analysis.3. Machine Learning Libraries: After data manipulation and visualization, learn machine learning, starting with simpler models and moving to advanced ones.Scikit-learnXGBoost /LightGBM4. Deep Learning Frameworks: If you're interested in AI and deep learning, these libraries will allow you to build and train neural networks.TensorFlow and KerasPyTorchTo learn more, you can refer to Python for Data Science.Web Development 1. Core Web Frameworks (Backend Development with Python): These are the tools for building Python-based web applications.2. Database Integration: Learn how to connect Python web frameworks to databases for storing and retrieving data.SQLiteSQLAlchemyDjango ORM3. Front-End and Backend Integration: Learn how to connect Python backends with front-end technologies to create dynamic, full-stack web applications.Jinja2 (Flask)Django Templates4. API Development: Learn to build APIs (Application Programming Interfaces) for connecting your backend with front-end apps or other services.Flask-RESTfulDjango REST Framework (DRF)To learn more, you can refer to Python for Web Development.PracticePython quiz page covers topics including variables, data types, input, output, lists, tuples, dictionaries and sets. The Python Coding Practice Problems page offers exercises on loops, functions, lists, strings, dictionaries, sets and advanced structures like heaps and deques. QuizzesPython Coding Problems This Python tutorial is updated based on latest Python 3.13.1 version. Python Introduction How Python Programs are Executed Comments in Python Variables in Python List Introduction Set in Python Starting with the Python 3.11.0, Python 3.10.7, and Python 3.9.14 releases, CPython release artifacts are signed with Sigstore. See our dedicated Sigstore Information page for how it works. OpenPGP verification Python versions before 3.14 are also signed using OpenPGP private keys of the respective release manager. In this case, verification through the release manager's public key is also possible. See our dedicated OpenPGP Verification page for how it works. See PEP 761 for why OpenPGP key verification was dropped in Python 3.14. Windows (Updated for Azure Trusted Signing, which applies for all releases chronologically from 3.14.0a1) The Windows installers and all binaries produced as part of each Python release are signed using an Authenticode signing certificate issued to the Python Software Foundation. This can be verified by viewing the properties of any executable file, looking at the Digital Signatures tab, and confirming the name of the signer. Our full certificate subject is CN = Python Software Foundation, O = Python Software Foundation, L = Beaverton, S = Oregon, C = US and as of 14th October 2024 the certificate authority is Microsoft Identity Verification Root Certificate Authority. Our previous certificates were issued by DigiCert. Note that some executables may not be signed, notably, the default pip command. These are not built as part of Python, but are included from third-party libraries. Files that are intended to be modified before use cannot be signed and so will not have a signature. macOS installer packages Installer packages for Python on macOS downloadable from python.org are signed with with an Apple Developer ID Installer certificate. As of Python 3.11.4 and 3.12.0b1 (2023-05-23), release installer packages are signed with certificates issued to the Python Software Foundation (Apple Developer ID BMM5U3QVKW). Installer packages for previous releases were signed with certificates issued to Ned Deily (DJ3H93M7VJ). Other useful items Looking for third-party Python modules? The Python Package Index has many of them. You can view the standard documentation online, or you can download it in HTML, EPUB and other formats. See the main Documentation page. Tip: even if you download a ready-made binary for your platform, it makes sense to also download the source. This lets you browse the standard library (the subdirectory Lib) and the standard collections of tools (Tools) that come with it. There's a lot you can learn from the source! Want to contribute? Want to contribute? See the Python Developer's Guide to learn about how Python development is managed. Welcome to the LearnPython.org interactive Python tutorial. Whether you are an experienced programmer or not, this website is intended for everyone who wishes to learn the Python programming language. You are welcome to join our group on Facebook for questions, discussions and updates. After you complete the tutorials, you can get certified at LearnX and add your certification to your LinkedIn profile. Just click on the chapter you wish to begin from, and follow the instructions. Good luck! Learn the Basics Coding for Kids Advanced Tutorials Other Python Tutorials Contributing Tutorials Read more here: Contributing Tutorials This site is generously supported by DataCamp. DataCamp offers online interactive Python Tutorials for Data Science. Join over a million other learners and get started learning Python for data science today! NEW Understand quantum computing basics like qubits, superposition, and entanglement. Then use Python Qiskit to build your first quantum circuit. Dec 01, 2025 intermediate data-science If you're starting fresh with programming and wondering how to use Python, this tutorial will give you the foundation you need to get started with Python step by step. You'll learn how to install Python, run your first program, and understand the language's clear syntax. You'll also explore variables, loops, functions, classes, and error handling while discovering tools, coding style, and libraries that help you write effective Python code from day one. By the end of this tutorial, you'll understand that: You can install Python on Windows, macOS, and Linux using binaries, package managers, or distributions like Anaconda. You can use the Python REPL to experiment interactively before writing full scripts, modules, and projects. Built-in data types like strings, lists, dictionaries, and sets provide powerful ways to structure information. You can handle errors with syntax checks, exceptions, and debugging practices to keep your code running smoothly. Tools like editors, integrated development environments (IDEs), and AI assistants can boost your productivity when writing Python code. You'll go through your first steps with clear examples so that you can start coding in Python confidently and build on solid ground. Take the Quiz: Test your knowledge with our interactive "How to Use Python: Your First Steps" quiz. You'll receive a score upon completion to help you track your learning progress: Interactive Quiz How to Use Python: Your First Steps Review the basics of Python with this quiz. Practice syntax, keywords, variables, errors, and tools every beginner should know. Python is a high-level, interpreted, interactive, and object-oriented programming language that's a great choice as a first language because its code reads like English. It's flexible, powerful, and allows you to do many things, both big and small. With Python, you can write basic programs and scripts, as well as create complex and large-scale enterprise solutions. Here's a sampling of its uses: You'll find Python across many high-traffic websites. For example, Reddit is written in Python. Dropbox's earliest prototypes were in Python, and it remains central there. YouTube uses Python among its back-end languages. Meanwhile, Instagram runs on Django, and Pinterest has historically used Python with a modified Django stack. Python offers many features that make it attractive as your first programming language: Readable, beginner-friendly syntax: Python's design favors code readability, so you spend more time learning programming ideas and less time fighting syntax. Accessible: People of all ages, from school children to retirees, have learned Python, and so can you. Batteries included: The standard library ships with modules for file processing, networking, mathematics, date and time processing, testing, and more. Large community and abundant resources: There's a vast ecosystem of tutorials, videos, forums, and local meetups for every topic and skill level. Proven in the real world: From startups to enterprises and research labs, Python powers production systems, data pipelines, and AI tooling across industries. Versatile and scalable: It can be used for quick scripts and automation, as well as web applications, data analysis, machine learning, and even game development. Free and cross-platform: Python runs on Windows, macOS, and Linux, and it's free for both personal and commercial use. Open source: Python source code is publicly available under the Python Software Foundation License Version 2, which grants broad rights to use, modify, and distribute, including in proprietary software. Additionally, anyone can contribute to its development. Compared to other programming languages, Python offers several key features: Interpreted: It's portable and quicker to experiment with than compiled languages. Multiparadigm: It lets you write code in different styles, including object-oriented, imperative, and functional. Dynamically typed: It checks variable types at runtime, so you don't need to declare them explicitly. Strongly typed: It won't let unsafe operations on incompatible types go unnoticed. There's a lot more to learn about Python. But by now, you should have a better idea of why Python is so popular and why you should consider learning to program with it. Before you can learn how to use Python, you need to install it. Python works on Linux, macOS, Windows, and several other platforms. You can download and install the latest version from the official download page. You also have the option to install and use different Python versions for different projects. To check what Python version has been installed globally on your operating system, open the terminal or command line and run the following: This command prints the version of your system's default Python installation. Note that on macOS and Linux, you'll typically run the interpreter with python3. On many systems, python now also points to Python 3, but this isn't always guaranteed, especially on Linux. If python isn't found or starts the wrong version, then use python3 instead. Regardless of your operating system, you can download an appropriate version of Python from the official site. Go there and select the appropriate 32-bit or 64-bit version for your operating system and processor. Selecting and downloading a Python binary from the language's official site is often a good choice. However, there are some OS-specific alternatives: macOS: You have the option of installing Python from Homebrew. Linux: You can install several Python versions using your distribution's package manager. Windows: You can install Python from the Microsoft Store on Windows 10 and later versions. Before doing so, make sure the publisher is the Python Software Foundation (PSF). After installation, Python, pip, and IDLE will be available. You can also use the Anaconda distribution to install Python with a rich set of packages and libraries, or you can use Miniconda to install only the packages you need. Note: There are several options for managing Python versions and environments. Choosing the right tools for this task can be overwhelming when you're starting with the language. For guidance on setting up your working environment, check out An Effective Python Environment: Making Yourself at Home. Once you've installed Python on your system using your preferred method, you can start using the Python interpreter in interactive mode. You can do a quick test to ensure Python is installed correctly. Open your terminal or command line and run the python3 command. This will start a Python interactive session, also known as the REPL. Your command prompt should look similar to this: While you're in this interactive session, you can run your first line of code: When you press Enter, Python prints Hello, World! to the screen. That's it! You've just written your first Python program in an interactive session! You can use exit() or quit() to leave a Python interactive session and return to the system shell. If you're using Python 3.13 or later, then you can also use the exit and quit commands (no parentheses) to leave the REPL. Alternatively, you can use the following key combinations to finish a session: Ctrl+D on macOS and Linux Ctrl+Z and Enter on Windows Keep your terminal or command line open. You still have more to do and learn! You'll start with the basics of Python syntax. The Python syntax is clear, concise, and focused on readability. Readability is arguably one of the most appealing features of the language itself. It makes Python ideal for those learning to program. In this section, you'll learn about several key components of Python syntax: In the following sections, you'll learn the essentials of Python's syntax. With this knowledge, you'll understand the basics of how to use Python to write real programs. Comments are pieces of text that live in your code but are ignored by the Python interpreter as it executes the code. You can use comments to quickly document certain parts of your code so that other developers can understand what the code does or why it's written a certain way. To write a comment in Python, just add a hash mark (#) before your comment text: The Python interpreter ignores the text after the hash mark up to the end of the line. You can also add inline comments to your code. In other words, you can combine a Python expression or statement with a comment in a single line, given that the comment is at the end of the line: You should use inline comments sparingly to clear up pieces of code that aren't obvious on their own. In practice, your comments should be short and to the point. PEP 8 advises keeping comments at 72 characters or less. If your comment is approaching or exceeding that length, then you might want to spread it out over multiple lines: If you need more room for a comment, then you can use multiple lines with a hash mark on each. This way, you can keep your comments under 72 characters. In Python, variables are names attached to a particular object. They hold a reference, or pointer, to the memory address at which an object is stored. Once you assign an object to a variable, you can access the object using that variable name. To use a Python variable in your code, you need to define it in advance. Here's the syntax: You should use a naming scheme that makes your variables intuitive and readable. The variable name should provide some indication as to what the values assigned to it are. Sometimes programmers use short variable names, such as x and y. These are perfectly suitable names in the context of math, algebra, and so on. In other contexts, you should avoid single-character names and use something more descriptive. That way, other developers can make an educated guess of what your variables hold. Think of others and your future self when writing your programs. Your future self will thank you. Here are some examples of valid and invalid variable names in Python: Your variable names can be any length and can consist of uppercase and lowercase letters (A-Z, a-z), digits (0-9), and the underscore character (_). In summary, variable names should be alphanumeric, but note that even though variable names can contain digits, their first character can't be a digit. Note: The lower_case_with_underscores naming convention, also known as snake case, is the recommended convention in Python. It isn't enforced by the language, but it's a widely adopted standard. Python offers full Unicode support, so you can also use Unicode characters in your variable names, such as the π variable shown above. However, while Python permits Unicode characters in identifiers, most style guides (including PEP 8) recommend sticking to ASCII letters, digits, and underscores for variable names. This helps avoid confusion, tooling issues, and potential security pitfalls. Like any other programming language, Python has a set of special words that are part of its syntax. These words are known as keywords. To get the complete list of keywords available in your current Python installation, you can run the following code in an interactive session: Each of these keywords plays a role in Python syntax. They have specific meanings and purposes in the language, so you shouldn't use them for anything but those specific purposes. For example, you shouldn't use them as variable names in your code. In fact, Python will prevent this by raising a syntax error if you try. Python also has what's known as soft keywords. These are keywords that retain their meaning only in specific contexts. Unlike normal keywords, you can use soft keywords as identifiers if needed. Here's how you can access the whole list of Python keywords and soft keywords: The keyword module provides objects and functions that allow you to work with Python keywords. For example, keyword.kwlist and keyword.softkwlist hold lists of all the current keywords and soft keywords in Python. Python has a handful of built-in data types, such as numbers (integers, floats, and complex numbers), Booleans, strings, bytes, lists, tuples, dictionaries, and sets. You can manipulate the built-in data types using different tools. Here are some of them: Operators Built-in functions Methods In the following sections, you'll learn how to use Python's built-in data types, including numbers, Booleans, strings, bytes, lists, tuples, dictionaries, and sets, with quick practical examples. Python provides integers, floating-point numbers, and complex numbers. Integers and floating-point numbers are the most commonly used numeric types in day-to-day programming, while complex numbers have specific use cases in math and science. Here's a summary of their more relevant features: Number Description Examples Data Type Integer Whole numbers 1, 2, 42, 476, -99999 int Floating-point Numbers with decimal points 1.0, 2.2, 42.09, 476.1, -99999.9 float Complex Numbers with a real part and an imaginary part complex(1, 2), complex(-1, 7), complex("1+2j") complex Integer numbers have unlimited precision. The precision information for floating-point numbers is available in sys.float_info. Complex numbers have a real and an imaginary part, which are both floating-point numbers. When you combine math operators with numbers, you form expressions that Python can evaluate. Arithmetic operators represent common operations such as addition, subtraction, multiplication, division, and so on: These operators work with two operands. The operands can be numbers or variables that point to numbers. Besides operators, Python provides built-in functions that allow you to manipulate numbers. These functions are always available to you. In other words, you don't have to do anything to use them in your programs. Consider the float() function. Given an integer number or a string representing a number, float() returns a floating-point number: With float(), you can convert integer numbers and strings representing numbers into floating-point numbers, but you can't convert a complex number into a floating-point number. Similarly, int() returns an integer when you call it with a floating-point number or a string as an argument. This function doesn't round a float input up to the nearest integer. Instead, it truncates the input, throwing out anything after the decimal point, and returns the resulting integer. For example, an input of 10.6 returns 10 instead of 11. Likewise, 3.25 returns 3: Note that you can pass a string representing an integer to int(), but you can't pass a string representing a floating-point number. Complex numbers don't work either. Besides these built-in functions, there are a few methods associated with each numeric type. You can access them through attribute references using dot notation: In the case of integer numbers, to access their methods through a literal, you need to use a pair of parentheses. Otherwise, you get a SyntaxError. In Python, Booleans are implemented as a subclass of integers with only two possible values: True or False. Note that these values must start with a capital letter. You use Boolean values to express the truth value of an expression or object. Booleans are handy when you're writing predicate functions or using comparison operators, such as greater than (>), less than (18 is false, so execution jumps to the elif clause. The condition in this clause is true, so Python runs the associated code block and prints You're exactly 18 years old. Sometimes, you need to traverse an iterable of data or repeat a piece of code several times. In this scenario, you can use a loop. Python provides two types of loops: Python's for loops are designed to iterate over the items in a collection, such as lists, tuples, strings, and dictionaries. In contrast, while loops are useful when you need to execute a block of code repeatedly as long as a given condition remains true. Here's the general syntax for creating a for loop: This type of loop normally performs as many iterations as items in the target iterable. You commonly use each iteration to perform a given operation on or with the value of loop_var. The else clause is optional and runs when the loop finishes. The break and continue statements are also optional. Here's a quick example of a for loop that allows you to iterate over a tuple of numbers: When the loop processes the last number in the tuple, the flow of execution jumps to the else clause and prints The loop wasn't interrupted on your screen. That's because your loop wasn't interrupted by a break statement. You commonly use an else clause in loops that include at least one break statement in their code block. Otherwise, there's no need for it. If the loop finds a break_condition, then the break statement interrupts the loop's execution and jumps to the next statement below the loop, without consuming the rest of the items in iterable: When i == 3 is true, the loop prints Number found: 3 on your screen and then hits the break statement. This interrupts the loop, and the execution jumps to the line below the loop without running the else clause. If you change the condition to i == 6 or any other number that's not in the tuple, then the loop doesn't hit the break statement and prints Number not found. If the continue_condition is true, then the continue statement resumes the loop without running the remaining statements in the loop's code block: This time, the continue statement restarts the loop when i == 3. That's why you don't see the number 3 in the output. Both break and continue should be wrapped in a conditional. Otherwise, the loop will always break when it hits break, and continue when it hits continue. You typically use a while loop when you don't know beforehand how many iterations you need to complete a given operation. Here's the general syntax for a while loop in Python: This loop works similarly to a for loop, but it'll keep iterating until condition becomes false. A common problem with this type of loop comes when you provide a condition that never evaluates to False. In such cases, you'll have a potentially infinite loop. Here's an example of how the while loop works: Again, the else clause is optional, and you'll commonly use it with a break statement in the loop's code block. The break and continue statements work the same way in a for loop. There are situations where you need an infinite loop. For example, GUI applications run in an infinite loop that manages the user's events. This loop needs a break statement to terminate the loop when, for example, the user exits the application. Otherwise, the application would continue running independently. You often create this type of intentionally infinite loop using the while True: pattern. In Python, a function is a named code block that performs actions and optionally computes the result, which can be returned to the calling code. You can use the following syntax to define a function: The def keyword starts the function header. Then you need the function's name and a list of arguments in parentheses. Note that the list of arguments is optional, but the parentheses are syntactically required. Next, you can define the function's code block, which will begin one level of indentation to the right. The return statement is also optional and is the statement you use if you need to send a return_value back to the caller code. Note: The full syntax to define functions and their arguments is beyond the scope of this tutorial. For an in-depth resource on this topic, check out Defining Your Own Python Function. To use a function, you need to call it with the appropriate arguments if needed. A function call consists of the function's name, followed by the function's arguments in parentheses: You can have functions that don't require arguments when called, but parentheses are always needed. If you forget them, then you won't be calling the function but referencing it as a function object. Classes let you bundle data (attributes) and behavior (methods) into reusable blueprints for objects. They're a core part of object-oriented programming in Python and help you model concepts from your problem domain. Here's a quick example of how to create a class and instantiate it: The class keyword allows you to define the class. Then, you have the .__init__(), which runs when you create a new object and initializes its attributes. Once you've defined a class, you can create instances using the class constructor with appropriate arguments. Finally, you can access the attributes and methods on the instance. Imports allow you to reuse code by bringing modules and packages into your main program. They're a critical tool for programs where you split the code into multiple .py files. Here are some quick examples of common syntax constructs that you'll use to import modules and objects in your Python code: In these examples, you import math and use dot notation to call sqrt(). Then, you import the sqrt() function directly into your current session and call it again. Finally, you import pi as PI, which is an alias. Errors can frustrate programmers at every level of experience, and identifying and handling them is a core skill. In Python, there are two types of code-based errors—syntax errors and exceptions—and you may also face semantic errors. Syntax errors occur when the syntax of your code isn't valid in Python. They automatically stop the execution of your programs. For example, the if statement below is missing a colon at the end of its header, and Python quickly points out the error: The missing colon at the end of the if statement is invalid Python syntax. Python's parser catches the problem and immediately raises a SyntaxError exception. The ^ character indicates where the parser found the problem. Exceptions are raised by syntactically correct code at runtime to signal a problem during program execution. For example, consider the following math expression: The expression 12 / 0 is syntactically correct from the Python parser's perspective. However, it raises a ZeroDivisionError exception when the interpreter tries to evaluate the expression. Note: In Python, you'll commonly rely on exceptions to handle errors. Python developers favor this coding style, known as EAFP (Easier to Ask for Forgiveness than Permission), over the coding style known as LBYL (Look Before You Leap), which is based on using if statements. For more information on these two coding styles, check out LBYL vs EAFP: Preventing or Handling Errors in Python. Python provides several convenient built-in exceptions that allow you to catch and handle errors in your code. Semantic errors happen as a result of one or more problems in the logic of a program. These errors can be difficult to find, debug, and fix because no error message is generated. The code runs but generates unexpected, incorrect output or no output at all. A classic example of a semantic error is an unintentional infinite loop, which most programmers experience at least once in their coding lifetime. If you're stuck, start by asking the interpreter for help. It's built in and always available. Perhaps you need to know how a specific function, method, class, or object works. In this situation, you can open an interactive session and call the built-in help() function. That'll take you directly to Python's interactive help system: Once there, you can enter the name of a Python object to get helpful information about it: When you type the name len at the help> prompt and press Enter, you get help content related to that built-in function. To leave the content and get back to the help> prompt, you can press Q. To leave the help utility, type quit and press Enter. You can also use help() with the name of an object as an argument to get information about that object: The built-in dir() function allows you to inspect the methods and attributes that are available in a particular object. There are several tools you can use to write and run Python code. You'll often combine them as your projects grow. The Python interactive interpreter—also known as the REPL (Read-Eval-Print Loop)—is perfect for experimenting and learning how to use Python. To write code that you can reuse and share, you typically put it into Python scripts and modules, which you usually edit in a code editor or an integrated development environment (IDE). In addition to these traditional tools, modern AI coding assistants can help you scaffold Python projects, write code and tests, debug, suggest improvements, and more. Think of them as a productivity booster that pairs nicely with the terminal, editors, and IDEs. The sections below introduce each option and how they fit together. Although you can write complex code in an interactive session, you typically use the REPL for one-line expressions and statements or for short compound statements to get quick feedback from your code. Open your Python interpreter and type the following: The interpreter evaluates 24 + 10 and outputs the sum 34. Now, try one more: Take a moment to read the output of this import. It states some important principles in Python, which will help you write better and more Pythonic code. So far, you've used the standard Python REPL, which ships with the official CPython distribution. However, this isn't the only REPL out there. Third-party REPLs provide many useful features, such as syntax highlighting, code completion, and history. Here are some popular options: IPython provides a rich toolkit to help you code in Python interactively. bpython is an interface to the Python interpreter for Linux, BSD, and macOS. ptpython is a Python REPL that also works on Linux, BSD, macOS, and Windows. Keep in mind that once you close the REPL session, your code is gone. In other words, the code typed into a REPL isn't saved, so you can't reuse it. As a developer, you want code that you can reuse to save precious keystrokes. In this situation, code editors and IDEs come in handy. Code editors are often good for learning purposes. Why? Because when you're learning something new, you want to peel off as many layers of complexity as possible. Adding a complex IDE into the mix can make the task of learning Python more difficult. A Python program, in its simplest form, consists of lines of text (code) saved in a file with a .py or .pyw extension. You can write Python code in something as basic as Notepad on Windows, but there are much better options available that make coding and learning easier. At its core, a code editor should provide several features to help programmers create programs. In most cases, you can customize the code editor to suit your needs and style. So, what should you look for in a code editor? The answer to this question might depend on your personal needs, but in general, you should look for at least the following features: Here's a non-exhaustive list of some modern code editors that you can use: Visual Studio Code is a full-featured code editor for Linux, macOS, and Windows. Sublime Text is a fast, cross-platform editor with a rich plugin ecosystem. Neovim is a modern Vim with an extensible Lua-based plugin system. Zed is a high-performance, collaborative editor available on macOS and Linux. Cursor is an AI-assisted editor built on the VS Code ecosystem, available on macOS, Windows, and Linux. Notepad++ is a popular, lightweight editor for Windows. GNU Emacs is an extensible, cross-platform editor. There are many different options, both free and commercial, when it comes to code editors. Do your research and don't be afraid to experiment! Keep in mind that your code editor should help you adhere to Python coding standards, best practices, and idioms. An integrated development environment (IDE) is a program dedicated to software development. IDEs commonly integrate features such as code editing, debugging, version control, test running, and environment management. Here are solid, up-to-date options that work great with Python: IDLE ships with Python and provides an interactive shell, a code editor, and a basic debugger. It's ideal for first steps. For a friendly walkthrough, see Getting Started With Python IDLE. Thonny is a beginner-friendly IDE with a clean UI and a step-through debugger. It's another great choice if you're just getting started. See Thonny: The Beginner-Friendly Python Editor to learn more. PyCharm is a full-featured IDE from JetBrains (Windows, macOS, Linux) with powerful refactoring, debugging, testing, and database and web tools. See PyCharm for Productive Python Development (Guide) for a comprehensive guide. Spyder is a scientific Python IDE with an IPython console, variable explorer, plots, and NumPy, pandas, and Matplotlib integrations. It's popular in data science workflows. If you're into data science, then you may use JupyterLab, which isn't a traditional IDE but is widely used for data analysis and teaching. This list isn't exhaustive. If you're new, start with Thonny. For larger projects and web work, PyCharm is a strong choice. Explore and experiment before you make your final choice. AI coding assistants can augment your workflow by generating code, explaining error messages, writing tests and documentation, and suggesting refactors. All of this comes from natural language prompts. Most popular and powerful AI coding assistants like GitHub Copilot Chat, Claude Code, Codex CLI, Gemini CLI, and JetBrains AI Assistant, are available as editor plugins or extensions, web apps, and command-line tools. When you're using these AI assistants, keep these best practices in mind: Be specific in your prompts: Include your Python version and library names and versions when they matter. Verify everything: Run the code, read the documentation, and add tests. Protect secrets: Don't share user credentials, private keys, or sensitive data in prompts. Treat it like a junior teammate: You're still the engineer responsible for code correctness and style. AI assistants are tools to speed up your development process. They're not replacements for understanding. Use them for learning, coding, and automating routine tasks, but always validate their output and keep building your understanding of Python and the libraries you use. PEP 8 is the official style guide for Python code. Although it's not required for writing working Python code, studying PEP 8 and applying it consistently in your Python code will make your programs more readable and maintainable. Luckily, you don't need to memorize PEP 8 to give your Python code a Pythonic style. Most code editors and IDEs that support Python internally implement automatic checks to flag PEP 8 violations. These checks help you consistently improve your code style and reinforce PEP 8's recommendations. You can also take advantage of code linters like Flake8, Pylint, and pycodestyle. You can even use code formatters, such as Black, isort, and Ruff, to consistently format your code. Some of these tools are conveniently integrated into most modern Python code editors and IDEs. To learn more about improving the quality of your code using PEP 8 and other style best practices, check out How to Write Beautiful Python Code With PEP 8 and Python Code Quality: Best Practices and Tools. So far, you've learned a few basic Python concepts and features. When you start to dive deeper into the language, you may find that you need a certain feature and decide to code it by yourself. If that's the case, then consider that you might be reinventing the wheel. Python's been in use for over three decades now. It has an incredibly large community of developers, and it's likely that someone else has run into the same problem as you. With a little research, you may be able to find a code snippet, library, framework, or other solution that can save you a lot of time and effort. The first place to look is the Python standard library. If you don't find anything there, then you can also look at the Python Package Index (PyPI). Finally, you can check out some other third-party libraries. One of the great things about Python is the plethora of available modules, packages, and libraries built into the Python core and made available by third-party developers. These modules, packages, and libraries can be very useful in your day-to-day work as a Python developer. Here are some of the most commonly used built-in modules: For example, here you import math to use pi, find the square root of a number with sqrt(), and raise a number to a power with the pow() function: Once you import math, you can use any function or object defined in that module. If you want a complete list of the functions and objects that live in math, then you can run something like dir(math) in an interactive session. You can also import specific functions directly from math or any other module: This kind of import statement brings the name sqrt() into your current namespace, so you can use it directly without needing to reference the containing module. If you're using modules such as math or random, then make sure not to use those same names for your custom modules, functions, or objects. Otherwise, you might run into name conflicts, which can cause unexpected behavior. The Python Package Index, also known as PyPI (pronounced "pie pea eye"), is a massive repository of Python packages that includes frameworks, tools, packages, and libraries. You can install any PyPI package using pip, uv, or another package installer. New coders frequently hit a wall when they're following an example and get a ModuleNotFoundError exception when they run the code. This error means that the code lives in a module, but that module isn't installed in the current Python environment, leading to a broken dependency. For example, say you're trying to run an application that uses pandas, but you don't have this library installed in your Python environment. In this situation, you can open your terminal and use pip as shown below: This command downloads pandas and its dependencies from PyPI and installs them in your current Python environment. Once the installation is finished, you can run your application again. If there are no other broken dependencies, the code will work. Here at Real Python, you'll find all kinds of resources to help you get started with the language and learn Python: Tutorials help you learn Python through a detailed, in-depth approach. Video Courses provide in-depth content with a progressive learning approach. Learning Paths guide you through foundational and advanced Python topics from the ground up. Live Courses for Beginners connect you with expert instructors for hands-on Python learning in real time. References help you quickly look up Python and programming-related concepts. Quizzes let you test your understanding and measure your learning progress. The Real Python Podcast covers a wide range of topics, including Python best practices, career tips, and related software-development themes. Office Hours give you the opportunity to meet fellow Pythonistas, discuss your learning, ask questions, and explore Python concepts. Books help you take your Python skills to the next level and support our online resources. Community allows you to connect with the Real Python team and other Pythonistas who are actively looking to improve their skills. Most of these learning resources are free. Others cost a modest fee that supports the site and helps maintain high-quality content. If you're just beginning with Python, then check out the book Python Basics: A Practical Introduction to Python 3 and its companion Learning Path. They'll help you make the leap from beginner to intermediate Python developer. Of course, there are other Python courses, tutorials, and resources available online. The right choice is personal—compare a few and pick what fits your goals and learning style. The official Python documentation is a solid, free reference to keep handy. Just be aware that the material can be less beginner-friendly than what you'll find at Real Python. Above all, it's important that you don't fall into trying to find the best book or video ever and get lost in the process. Do some research. Ask around. But pick something and stick with it! Open your code editor and start coding a Python project! Make a commitment to yourself to find a way to bring your vision to life and complete your project. Coding is like riding a bike. You can watch people to learn how it's done, but in the end, you have to do it yourself. When you get stuck or need to learn a new concept, you can often work through the problem yourself by doing some research on the web or asking an AI chat. For example, if you get an error, then typing the exact error message into Google or an AI chat can help you solve the problem. Stack Overflow is another place you can visit when looking for answers. If you get stuck on a problem, then try these suggestions: Stop coding! Get a

piece of paper and map out how to solve the problem using plain words. Use a flowchart if necessary. Don’t use a try … except block until your code is working. The try can suppress valuable error messages that help identify problems in your code. Use print() to quickly inspect your variables and make sure they have the expected value. This is an effective, quick-and-dirty debugging technique. Use the rubber duck debugging technique. Explain your code, line by line, to the duck. You might find the solution to your problem in the process. If you’re still stumped, use the Python Visualizer. This tool allows you to step through your code as it executes. The Python Visualizer has examples to help you if needed. One final and important note: a frustrated brain won’t help. When you feel stuck because something isn’t working, take a break to clear your mind. Go for a quick walk or run, stretch, or do something unrelated. You’ll be amazed at just how effective this can be. More often than not, you’ll come back with fresh eyes and see a simple typo, a misspelled keyword, or another small oversight. You’ve taken your first steps with Python, learning how to install it, write and run your first programs, and understand its clear and readable syntax. You’ve explored the core building blocks of Python coding, including variables, data types, loops, functions, classes, and error handling, and you’ve discovered tools and resources that make programming in Python efficient and enjoyable. Along the way, you’ve learned about coding style, the Python standard library, and third-party packages. Understanding these foundational concepts is crucial for any Python developer. They’re the basis for writing clean, reliable, and maintainable code. In this tutorial, you’ve learned how to: Install and run Python on your operating system and use the REPL for interactive exploration Work with Python’s built-in data types, including strings, lists, dictionaries, and sets Use control flow constructs like conditionals and loops to manage program execution Define your own functions and classes to structure your code effectively and promote reusability Detect and handle errors and exceptions in your code Use Python’s built-in help system to get help interactively With these skills, you can confidently start writing and running Python programs, explore new features and libraries, and build on your foundation to tackle more complex projects. Now that you’ve taken your first steps with how to use Python, you can use the questions and answers below to check your understanding and recap what you’ve learned. These FAQs are related to the most important concepts you’ve covered in this tutorial. Click the Show/Hide toggle beside each question to reveal the answer. When you use Python, it checks types at runtime instead of requiring you to declare them. This makes it a dynamically typed language. It’s also strongly typed, meaning Python prevents unsafe operations on incompatible types and raises an error instead. To run the Python interpreter, open your terminal or command prompt and type python3 or python (on macOS/Linux), or py (on Windows). This will start an interactive session, known as the REPL, where you can type and execute Python code directly. You create a variable in Python by assigning a value to a name using the = operator. For example, x = 10 defines a variable named x that refers to the value 10. A variable name is the label you choose, while the variable value is the object stored in memory that the name refers to. You can use the name to access or modify the underlying value. Python provides built-in data types such as numbers, Booleans, strings, bytes, lists, tuples, dictionaries, and sets. Each of them comes with its own methods and operations for manipulation. An integer represents whole numbers without decimals, while a floating-point number represents numbers with decimal points. Floating-point values can approximate fractions but have limited precision. Booleans express truth values in Python. They only have two possible values, True or False, and are commonly used in conditions and comparisons. A list is mutable, so you can change its contents after creation. A tuple is immutable, which means once created, you can’t change its contents. A dictionary is a collection of key-value pairs where each key maps to a specific value. Keys must be unique and hashable, while values can be any object. You use comments to explain what your code does when it’s not clear or to clarify why you chose a specific approach or algorithm. This makes your code easier to read, understand, and maintain for you and others. The help() function opens Python’s interactive help system. You can use it to quickly access documentation about functions, methods, classes, or modules directly in your Python interactive session. A syntax error occurs when your code violates Python’s rules and prevents execution. An exception occurs at runtime in otherwise syntactically valid code when something unexpected happens, such as dividing by zero. Take the Quiz: Test your knowledge with our interactive “How to Use Python: Your First Steps” quiz. You’ll receive a score upon completion to help you track your learning progress: Interactive Quiz How to Use Python: Your First Steps Review the basics of Python with this quiz. Practice syntax, keywords, variables, errors, and tools every beginner should know.