STM JOURNALS

Recent Trends in
Programming Languages

ISSN: 2455-1821
Volume 13, Issue 1, 2026
January–April
DOI (Journal): 10.37591/RTPL

https://journals.stmjournals.com/rtpl

Review RTPL

Emerging Programming Languages Shaping Technology in 2026

V. Basil Hans*

Abstract

As technology keeps changing quickly, it's more crucial than ever for developers, students, and tech fans to choose the correct programming language. This article talks about the best programming languages to learn in 2026, focusing on their most important features, how they can be used in the real world, and how much demand there will be for them in the job market in the future. Each language is looked at in light of industry trends, community support, and job opportunities. Some languages are flexible and easy to learn, while others are high-performance and specialized. This guide gives you useful information that will help you make smart choices and stay competitive in an ever-changing digital world, whether you're just starting with coding or want to improve your skills. The IT world will be different in 2026, with new challenges and problems. New technologies like machine learning, blockchain, and automation will have an impact on which programming languages are most useful. Some languages are still the most popular because they are flexible and have a lot of community support. Others are becoming more popular because they are fast, secure, or good for specific tasks. This article will help you figure out which programming languages you should study in 2026 by looking at current trends, industry needs, and real-world uses. If you're a newbie just starting to learn how to code or an experienced developer trying to improve your skills, knowing which languages to focus on will help you remain ahead in an industry that is always changing and getting more competitive.

Keywords: Languages for programming, making software, skills in coding, trends in technology in 2026, jobs in IT, new technologies, tools for developers

INTRODUCTION

The goal of this analysis was to identify the programming languages that will be most useful for teaching, the job market, and the industry in 2026. This will help people who want to learn which languages to learn. The results are influenced by three essential and linked criteria: skills that provide a learner with more options, skills that make a student more likely to get a job, and skills that will prepare a learner for a field of development that is becoming more popular.

Data-driven assessments of job advertisements, programming language use, and skill demand help decide which languages to learn and which learning routes to focus on within a topic. The analysis depends on publicly available data sources that encompass the entire software development ecosystem, such as language features, supporting libraries and tooling ecosystems, runtime performance, and concurrency models. There is a set of languages that form the basis for future learning in a certain area of programming or domain, allowing these topics to be studied along pathways that lead to mastery [1].

***Author for Correspondence**
V. Basil Hans
E-mail: vhans2011@gmail.com

Research Professor, Department of Management and Commerce, Srinivas University, Mangaluru, Karnataka, India

Received Date: April 13, 2026
Accepted Date: April 16, 2026
Published Date: April 30, 2026

Citation: V. Basil Hans. Emerging Programming Languages Shaping Technology in 2026. Recent Trends in Programming Languages. 2026; 13(1): 24–34p.

HOW TO CHOOSE A LANGUAGE

The choice of programming languages should be based on clear criteria that show the importance of each language and how technology is changing. Languages differ significantly in terms of ease of

use, usefulness for employment, and applicability to different fields. Therefore, language choice has important practical effects for each person. The selection process is guided by widely shared goals: all candidates recognize the importance of an accessible introductory language, particularly those with educational aspirations; employability factors fluctuate based on economic conditions; and a small number of applicants engage in nonspecialized programming as a pastime. Ecosystem maturity remains the most useful measure. As people gain expertise, the practical usefulness of potential projects grows; however, the fields of application and project profiles can be very different. Based on these main needs, the study methodically looks at the first group of languages that together cover most learning paths.

Candidates have many job options in programming; therefore, it is still very important to choose the right one. Ecosystem maturity is the most important factor to consider, since domain applicability and personal desire have a significant impact on job prospects and financial growth. The survey structure is also influenced by factors such as accessibility, portability, interoperability, and modernity. These strategic criteria also have a significant impact on the importance of certain factors: accessibility is crucial for individuals seeking to enter the education market, influencing the choice of candidate languages; programming for business becomes more appealing during downturns, making job market signals more significant; cloud-native paradigms necessitate widespread portability; and pre-existing modernity influences long-term maintainability [2].

Even beginner languages must be able to handle difficult technological problems. Most of the time, fixing system problems requires either low-level design-based languages that provide a lot of control or high-level systems languages that do not include design at all. Serious programming always requires coordination across different languages, which means that you need to know how to work with other languages. Equipment resources severely limit the possible scope, thus emphasizing the necessity for expandable candidates to surpass their respective levels.

BASIC LANGUAGES FOR FOUNDATIONS

The four main languages mentioned above that meet the selection criteria in Section 2 are a good starting point for learning how to program. They all have simple syntax, many users, many libraries, many support resources, and the ability to work with other languages. Before moving on to other languages that fulfil the original criteria in more specific areas, it is important to explain why these four are part of the foundational block and how they help with learning the basics.

Python is usually the language of choice for teaching basic programming. Its simple and clear syntax makes it easy to focus on the basics and start coding immediately. Because it has many libraries, it is a logical choice for processing and analyzing data in real-world situations. In schools, it is the sole language taught at all levels, from elementary school to college. This means that it is one of the few languages that students learn in both high school and college.

This is particularly important for businesses, where the demand for skilled Python programmers is still growing. However, the language has some performance limitations. When performing comput-intensive operations with large amounts of data, speed can be quite important. In these cases, it is commonly used as a “glue” language to connect faster parts written in C, C++, Java, or Rust [3].

Python

The main ideas behind this essay’s recommendations come from the needs of most development projects, which include systems, data, and AI, and online and mobile, as well as the general guidelines for choosing programming languages. These pillars include many different options; however, they are mostly focused on certain application areas. Additionally, a limited selection of recommendations guides all learners, irrespective of their prior experience, towards the build, learn, and grow stages of their programming careers.

Python is the best programming language for teaching. Guido van Rossum, the creator of Python, chose Unicode support and a vast standard library over simplicity and clarity, but he kept the trade-off small. This focus on readability and simplicity goes beyond lexical issues to include semantic ones, such as the use of indentation to create blocks and the lack of explicit variable declarations. Because of these reasons, new developers may focus on writing algorithms and logic instead of getting stuck on the details of syntax. Readability also makes it easier to explore Python's huge library ecosystem, which has many libraries that work for both data and AI tasks. Another major benefit of Python in teaching is that it makes it easy to quickly develop and test new ideas. Speculation costs a lot of money, and releasing non-trivial software for general usage is much more than just developing the implementation itself. Therefore, it is important to have the luxury of a prototyping approach in schools to prepare students for employment in the real world. However, beginners who want low-level system access, excellent performance, or web clients that are not specialized should be careful. The primary implementation of Python gives up speed and control over memory management in exchange for ease of use. It uses specially optimized libraries when performance is important [4].

Java

Java is suitable for a wide range of applications because it works well on many platforms, has many standard libraries, and has a huge ecosystem. It is especially useful for enterprise backends, Android development, and scientific computing. Java's libraries make it possible to build complicated user interfaces, perform predictive modelling, mine data, and analyze massive amounts of data. The strong Java Virtual Machine (JVM) environment helps with challenges that arise because there is no compiler-supported pipe-and-filter architecture; however, the language itself can make strictly functional programming less natural. Groovy and Clojure are dynamic languages that allow the use of domain-specific languages that are easier for developers to use, even though they are not as easy to understand as templates made by hand.

Though people often complain that Java is too verbose and has old ideas, it is still the most popular language for enterprise backends and cloud-native microservices. In addition, the need for Kotlin expertise has grown significantly since Kotlin became the official language for Android development and Gradle became more popular as a build system. The JVM ecosystem makes it easier for multiple languages to work together and use libraries from these languages. Java is a good choice for safely delivering large-scale system-level services with mission-critical reliability because it is predictable, type-safe, and has many tools for development, testing, profiling, and deployment. This is especially true when interoperability with existing codes and libraries is important [5].

C

C is still a popular choice for systems programming because it provides low-level access to memory, allows hardware interaction, and is fast. As a compiled language, it can be made more efficient by manually allocating and freeing memory; however, these capabilities also make memory leaks more likely. Occasionally, programming language systems are intentionally rendered useless. For instance, C does not stop pointer arithmetic; thus, the code must be carefully checked to ensure that arrays are not accessed outside their allotted memory bounds. In terms of prototyping, C work is not as fast because modifications usually involve many long compilation stages and do not receive any help from safety checks or garbage collection at run time.

C is still the language of choice for developers working on huge codebases. Though other languages, such as Rust, now enable native code, standard library wrapper code still works best with C APIs. Because there are many different types of computers, C code that is meant to work on a certain type of computer may require a lot of work to make it work on other types of computers. C code is still called from many different languages, even though it is a long-lived language with many connections to other programming languages. This is why mastering C is still an important part of follow-up courses in low-level systems programming [6].

JavaScript

Almost all online applications use JavaScript on both the client and server sides. It can also be the main language for server-based software, which means that all the parts of a full-stack application can be written in one language. This makes development much easier because the same code, tools, and libraries can be used. JavaScript, along with HTML and CSS, is a must-have skill for almost all online development. It is also one of the most popular languages in all fields of study. It features a huge library ecosystem and many high-quality, highly specialized third-party tools. Most large cloud businesses have created huge libraries for their services that can be used for both backend and frontend development in JavaScript. Many high-quality online courses are available because they are so popular, which is why it is often the first programming language taught. However, JavaScript has some problems and strange syntax, and the ECMAScript specification and library ecosystem have long, inconsistent, and erratic change histories. Technical architecture often results in unsafe programming methods that facilitate hacking, partly owing to the inherent characteristics of web applications. The peculiar ways in which the language seems to employ objects can be confusing for people who are just starting.

JavaScript is still an excellent language to learn first, despite its problems. However, without type safety and breathing-type checking, it can be difficult to deal with huge enterprise scales. Because JavaScript is mostly a dynamically typed language, it is not possible to perform a full code analysis without executing the entire application. However, Artistic Visual Studio cannot fully understand all dependencies. TypeScript can help with type checking in large-scale business applications [7].

LANGUAGES FOR SYSTEMS AND PERFORMANCE

The languages for systems and performance are widely used and in high demand in industry. They are similar to low-level operations. Rust is the best language for systems programming, Go is the best for cloud-native development, and C++ is the best language for all industries.

Rust has memory safety built-in, so it does not require garbage collection. This makes resource management easier. Its guarantee of being free of data races makes concurrent programming easier. When combined, these capabilities make it possible to quickly and reliably install system software, such as operating systems, compilers, and embedded programs.

Go puts a lot of emphasis on simplicity and fast compilation, which makes it easier to get started and keep things running. Improved concurrency support is still very important because an increasing number of software stacks are using parallel processing. Goroutines and channels help Go overcome these issues. They are important for cloud-native approaches and microservices design [8].

C++ strikes a good balance between speed and flexibility with its template system and multiparadigm approach. There are many standard libraries and frameworks that cover a wide range of fields, such as games, finance, media, Internet of Things (IoT), and web development. C++ is one of the few languages that is widely used in business and will be useful in the future.

Rust

Rust is a set of clear rules that enables the safe and simultaneous construction of low-level applications. Using ideas such as ownership, the system makes things safer by checking things at compile-time instead of throwing runtime errors. Within a certain scope, you can only access a single value in one way: either immutable or changeable. Rust achieves this without a garbage collector 3, which is impressive. This means that there are no pauses in the runtime, which makes the performance more consistent, as seen in applications such as ripgrep and tokio. The language is not very popular in areas where it is useful, such as system initialization and debugging, bootloaders, device drivers, operating systems, file systems, and embedded software. The Open Systems Accounting Protocol (OSAP) is an excellent Rust application for managing the energy use of real-time systems with high accuracy and minimal latency. It can also be used as a reference for people who have trouble in similar situations.

Go

The language is easy to understand, which is great for novice programmers. Go gets rid of inheritance, operator overloading, and other hard-to-use characteristics that are common in C++ and Java. Quick compilation makes it easier to build prototypes and applications quickly. Using goroutines and channels, the language has built-in support for concurrency, making it an excellent choice for cloud-native development. Many people also use protocol buffers. Cloud service companies, in particular, tend to use Go extensively, with platforms and tools built-in the language [9].

Google's Go programming language is cloud-native and simple but works well for developing scalable web services. Go is easy to learn because its syntax is straightforward and basic. Because it compiles quickly, it is a fantastic choice for quickly creating prototypes and apps. In addition, built-in support for concurrent programming is a significant improvement over the "manual" Pthreads methodology. Go is also a good choice for cloud-native applications, where the ability to scale both the software and the infrastructure on which it runs is very important. It has quickly become a common language for backend services, and most major cloud service providers offer Go SDKs for their platforms.

C++

C++ is known for its performance capabilities and widespread industrial use. C++ features a strong template system that allows you to write generic code, similar to C. C++ also supports many different programming styles, such as imperative, object-oriented, and functional programming. Therefore, C++ can be helpful in creating many different types of software, such as operating systems, web services, and video games.

However, C++ is still a difficult language to learn. It has good automatic memory management features, but it is not a real garbage-collected language, and it still allows you to control system resources at a low level. In addition, the standard library is not as large as those for the other languages discussed. There are many useful external libraries, but they may not always work in the same way or be of the same quality. Because of these problems, several companies have decided to use C++ less in their codebases and are looking for simpler options. However, many businesses in many different fields still hire hundreds or thousands of C++ developers to write software [10].

In summary, C++ is a fantastic choice for programmers who want to create high-performance applications or work at a large corporation creating systems or game-level programming. In addition, the fact that C++ applications can use memory more efficiently could be advantageous for workloads with limited resources, especially on lower-end hardware. However, it is important to note that using a language with good memory management features, such as Rust or Go, might make development easier and faster than coding in C or C++.

LANGUAGES FOR DATA AND AI

Trends show that people are moving away from Big Data frameworks and toward deeper statistical analyses with smaller datasets. Therefore, the following languages are not as important for general-purpose learning. However, they are very important for experts, especially in AI and data science.

Python. The use of Python in data research has been previously argued. Its large library ecosystem, especially NumPy and Pandas, makes it easy to work with data quickly and powerfully. It also works well with well-known data storage, such as thousands of relational SQL databases and the newest distributed NoSQL systems, to speed up the development of useful insights. The ongoing rapid increase in data science jobs shows that there is a need for people who know how to use Python. Though there are still issues regarding performance, they can be fixed by leveraging performance-critical code from other places and making Python's core function wrappers around optimized native code implementations.

R is still the best choice for statistical computations. It provides the most complete set of libraries for statistical inference, graphical statistical display, and data mining. It also has more advanced statistical plotting capabilities than Python's Matplotlib. Big firms such as Microsoft and Oracle, as well as dozens of smaller ones, make enterprise-class R products. Its prominence in academic research has made it a landmark study area that relies on complex statistics. R's data mining libraries and exploratory graphing tools are among the best in business when it comes to Educational Data Mining. It has become the standard in many fields, including genetics, biostatistics, machine learning, language analysis, pattern recognition, and spatial statistics. It is also considered the closest thing to Python for data science. However, the fact that there are no libraries for Big Data that are ready for production is still a problem, and these problems will probably keep it from getting a larger market share for a while [11].

Julia. Julia is a fast numerical computing language that is easy-to-use for people who are already familiar with Matlab or Python. Its syntax is clearly comparable to both, thus making it a good choice for people who are switching from those languages. It also attempts to fix their meta-programming problems while being quicker and able to call C libraries directly. A growing community of users is another sign of progress, but the environment is still limited, which makes it difficult to use the language for basic digital business objectives. However, it is becoming increasingly popular, and its use in artificial intelligence is still a goal.

Python (Again, for Data Science)

Python is the most popular language for data science and AI because it has a well-developed ecosystem with many libraries, frameworks, and tools. Most activities can be performed with just one line of code, which makes it easier for beginners and speeds up prototyping. However, because it has many abstractions, dynamic typing, and no compile time guarantees, it is not a good language for large systems. So, a lot of the most popular libraries are built-in C, C++, or Rust to avoid performance problems. There has been a lot of work lately to add just-in-time (JIT) compilation for speedup and type hints for compile time inspection.

R was designed specifically for statistical computation and is now the dominant language used by statisticians and data analysts in academia. R is also the best language for exploratory data analysis because it has powerful tools for creating custom data visualizations. The biggest problem with R is that it is usually difficult to bundle and deliver applications.

R

R was created expressly for statisticians and quickly became the most popular choice among data analysts and statisticians. R is nevertheless very useful for data-related activities such as statistical analysis, visualization, and reporting, even though it is not as flexible as general-purpose programming languages. R is also a strong player in the fields of statistical computing and data research. It encourages contributions from the community by being open source. There are many places where you can learn R programming, including many free online classes. The Importance of being earnest (TIOBE) index currently ranks R as the ninth most in-demand programming language, and the number of people who want to learn it has steadily increased since 2021. This is a major reason why new programmers learn R [12].

Julia

Julia is a programming language that is highly productive and fast. It was developed for technical and scientific computing. It allows people to write complicated calculations in a very simple way and still obtain speeds similar to C. Julia can be used interactively as a scripting language or to create entire programs. You can use Julia for numerical computing jobs just as well as you can with any other language because it has many of the most popular numerical libraries. The development of community packages and libraries is progressing rapidly, and a dedicated package registry makes it easier for users to find and install new packages. Although Julia is still a young language, it has already made a name for itself in the engineering and scientific communities as a high-performance language for technical, scientific, and data computation.

Trends in Web and Mobile Frontend/Backend

JavaScript and WebAssembly are used on both the client and server; however, there are some trends to consider. TypeScript is becoming increasingly popular on the client side, and Kotlin is becoming increasingly popular on the backend. Swift is the best choice for iOS/macOS development. It may be tempting to include all the UI/web development landscape for all clients in a single language like Jack, the polymorphic scripting language developed at Bell Labs for UNIX. However, these are multi-language developments that support established patterns of component development and code reuse. TypeScript should be seen as an improvement to JavaScript for big business client development that uses the tools that are most popular right now, not as a replacement. Many large companies, including Google, use Kotlin for server-side programming. It also provides a more up-to-date method for scripting the Java environment.

TypeScript is an object-oriented language built on JavaScript. It supports type declarations, type checking, and class-based programming. This is important when large teams are building systems that are too large and complicated to use simple tools. The resulting toolset for larger JavaScript/TypeScript client applications is the best way to take advantage of JavaScript's benefits for building component-based UIs while still following the normal web browser security architecture. This UI development methodology also makes it possible to reuse dynamic but safe code across many clients, such as browsers and mobile screens. The existence of similar functionalities in other enterprise tool frameworks, such as Flutter, diminishes the benefits of consolidating the entire UI into a single language. The primary advantage of the shift towards scripting languages and interpreter support lies in the facilitation of rapid development and incremental-dynamic updates.

TypeScript

An increasing number of businesses are switching from JavaScript to TypeScript, which is a superset of JavaScript that allows you to check types and perform other tasks. This makes programmers feel more confident when writing code. This makes it easier to adapt to the start-up period of new platforms, where new languages and paradigms are used to help generate new ideas. Institutions, mostly big businesses, are investing more money in education and training programs that teach people how to use TypeScript and its related frameworks, such as Angular and Ionic. HTML, JavaScript, and other languages, along with CSS, are always among the most popular languages on the market. Java is a general-purpose programming language created in the mid-1990s. It is used to create systems that can run on any platform and have sophisticated application programming interfaces and routines. Java has weathered the test of time and remains one of the most popular programming languages in the global job market around the world.

Many companies are coming together to use the new TypeScript programming language for JavaScript-based development. This language adds optional static type checking to JavaScript syntax, yet it still works with all the existing JavaScript code. The statically typed syntax helps find typos, circumstances that could cause runtime exceptions, and violations of stated interfaces both during development and at runtime. Many programming environments support the language and provide syntax checking and tool support on the fly. It is also the target of specialized industry education and certification [13].

Kotlin

Kotlin is important because it has a large ecosystem of established libraries and frameworks that make it easy to build various types of apps. Kotlin's current design also includes many features that make it more productive than Java. These include short syntax, static type inference, smart Integrated Development Environment (IDE) support, built-in null safety, support for delegation, destructuring declarations, and data classes. The language also supports both functional and object-oriented programming styles, which allows developers to choose the best method for their individual requirements. In Kotlin, functions are first-class entities. This means that they can be assigned to variables, provided as parameters, or returned from other functions.

JetBrains made Kotlin, a general-purpose programming language, public in 2011 and published its first stable version in 2016. Its main goal is to provide Java developers with a modern language that can be easily added to projects that are already in use. Kotlin adopts ideas from modern languages, such as C# and Swift, and simplifies typical tasks by reducing the amount of code required. The language runs on a JVM, which makes it portable and compatible with Java. This means that the code developed in both languages can be used in the same project. In 2017, Google officially backed Kotlin as one of the main languages for developing Android apps, which made it even more popular. Because of this support, a large Kotlin community has been formed, and an increasing number of tutorials and courses on the language have been created.

Swift

Swift was first introduced at the 2014 Worldwide Developers Conference. It changes the way applications are developed for Apple's iOS and macOS platforms. Swift might be good for programming in general, but it seems to work well only in the Apple ecosystem, as seen by its availability on Windows 11. Nevertheless, Swift is worth considering because the iOS and macOS industries are so large and important. In 2022, Apple's App Store generated $85 billion, with games accounting for 39% of that. The macOS share was $10 billion, or 12% of overall sales. The significant expansion of iOS and macOS software is aided by job openings that go beyond programming to include analysis, architecture, and development. Students are learning Swift in iOS development classes at colleges such as Harvard and Stanford (for example, CS50 and CS193p). There are also separate courses on platforms like Udemy that add to these classes.

Swift is a programming language that combines speed, safety, and design. It is said to be faster than Objective-C and close to C++, and its safety features lower the number of errors and eliminate whole classes of issues. Model–View–Controller (MVC) is a common software design pattern that is encouraged by the use of familiar syntax, numerous built-in methods, and a large standard library. Swift also allows the use of object-oriented, procedural, and functional programming styles. Students can now create and visually check the accuracy of programs without having to compile them first. This is because programs that can run on the "electronic paper" workspaces of the Swift Playgrounds app no longer need to be compiled. Pocket Code, a popular iOS app, uses the Scratch model, which is easy-to-use for people who have never programmed before.

In terms of speed of production, it is better than Objective-C and C++. Code-writing companies claim that developing Swift code takes 30% less time than writing Objective-C code and that it has the same capacity to compile code multiple times as scripting languages. Swift's modern syntax has much in common with newer programming languages, which makes it even better for teaching.

The following are some of these features:

- Explicit optional (to protect against the "nothing" argument),
- Closures as first-class objects, with shorthand parameter names,
- Inference of contiguity between arrays, dictionaries, and sets,
- Unified and concise syntax for generics related to types or functions,
- Clear designations of global and local functions.

NEW AND SPECIALIZED LANGUAGES

Dart was once a failing project, but it became popular after Flutter, a cross-platform UI toolkit that makes it easier to develop native-like apps for web, Android, iOS, Desktop, and embedded systems, became popular. Dart features a rather advanced compiler that works well when you compile to JavaScript. It also focuses on developer experience and productivity. Even though UI frameworks generally go away, big firms started using Flutter because it focuses on building cross-platform apps that can grow. Dart works nicely with machine code, which makes for great experiences on all devices

and in embedded systems. Code that compiles to JavaScript runs at about the same speed, the architecture supports hot reload, and Flutter comes with a full set of UI tools.

Built on the Erlang platform, Elixir inherits the strong concurrency and distributed capabilities of Erlang. It also has a flexible and expressive vocabulary that makes it easy to build a web backend that can handle errors. The expanding number of libraries, the ability to work with Phoenix for real-time enterprise web systems, and the general feature set of Elixir have attracted large enterprises to the platform.

WebAssembly and related languages: WebAssembly is a portable binary instruction format that allows high-performance applications to run on any platform or device in language-agnostic sandbox settings. It supports several languages, such as Rust, C, C++, Assembly Script, Go, and bocraft-simple. As more languages are compiled to WebAssembly, the idea of a real polyglot future gets closer. In the future, programmers will be able to choose the optimal tool for each task without having to worry about the language of all the components that work together. In addition, straight cross-language calls without any inter-process communication add another level of possible integration that makes the programmer's job easier and improves performance.

Dart

Though they are not as well-known, new languages such as Dart and Go may be interesting to some people. The Flutter framework, which is created in Dart, is becoming a popular alternative for cross-platform UI development that works on the web, desktop, and mobile from a single codebase. It usually works well, but its built binaries are larger than those of many native programs.

Lambda architecture is becoming less popular as microservices become more prevalent. There needs to be more emphasis on developing backend systems that can handle multiple users simultaneously, are fault-tolerant, and can be easily scaled. Elixir, which is based on Erlang, provides a useful way to design these types of systems. Elixir is becoming the top choice for modern online services because it features easy-to-use abstractions for creating systems that can handle a large amount of traffic and are fault-tolerant. It also has a thriving community of web backends. However, its complicated underlying model has made it much tougher to use in places other than web services.

It is worth noting WebAssembly and the languages that target its virtual machine. WebAssembly is a safe and efficient virtual machine with clear interfaces for networking and user interface. It is also a highly portable environment for high-performance applications. An increasing number of languages are starting to work with WebAssembly. The ability of WebAssembly for concurrent compilation, lack of a common runtime, and well-defined external interfaces could make it a useful addition to current cross-language methods in the long run.

Elixir

Elixir is excellent for designing distributed systems that can handle faults. This makes it an excellent choice for building web backends that can handle a large amount of traffic. Erlang, the programming language that powers it, was first created for a telephone switching system that could keep working even if some parts broke down. The syntax of Elixir is based on the Lisp programming language family and focuses on being clear and short.

Concurrent development is much easier with built-in tools for managing the processes and channels. For web development and deployment on embedded devices, libraries such as Phoenix and Nerves improve Elixir. Elixir is a niche language, but it has a good chance of lasting because both the language and its community are growing in size.

WebAssembly and other Languages that are Similar

WebAssembly (Wasm) is a binary instruction that can be used as a portable compilation target for many programming languages. With Wasm, web developers can use high-level languages such as C, C++, Rust, and Go to create applications for both the client and server. These programs run faster than JavaScript, follow modern software engineering best practices, and work on a wide range of platforms, including x86, ARM, and WebGPU. These benefits, along with the fact that many languages can be compiled into Wasm, make Wasm and its related languages great for IoT apps.

LEARNING PATHWAYS AND CURRICULUM

The curriculum is based on preferred languages and programming areas where those languages work effectively. Anyone can start with Python and then move on to data analysis, AI/ML, Java, or frontend and backend development. More advanced learners should start with Rust or with Go. If they wish, they can learn C++ or Python for data science. Once you know a lot about one language, it is enjoyable to try out others, such as Kotlin for mobile programming, TypeScript, Dart, Elixir, or even Julia. The suggested curriculum is divided into phases, with assessment milestones. It starts with fundamental languages and then moves on to other languages as students naturally improve.

Ultimately, the course you choose to learn should be based on your job and the industry in which you want to work. The average number of job ads for a certain language is a good place to start, but jobs themselves are too unpredictable to help people figure out what professional path to take. The saying "the path of least resistance" is probably true for those who are just starting out in their careers: choosing a language that is common in job advertising is more likely to lead to good results. As a career progresses, wage offers and the places where people want to go become the most important signals for job seekers. As always, professional analysis and projections should help you make decisions about your career, but they should never tell you what to do or how to do it.

EFFECTS ON INDUSTRY AND CAREER

In 2026, choosing a language to learn programming will go beyond personal taste and educational interest. It also includes professional opportunities, job market signals, and trends in skill adoption, which are all important reasons for learning programming. Three separate levels of information should be considered: wage rankings for key, in-demand workplace languages, trends showing whether demand for certain languages is increasing or decreasing, and messages from career pages and job advertisements asking about languages affected by those changes.

Overall, the rankings of salaries remain the same: C and C++ are at the top, followed by C# and Go, which are just ahead of Java, TypeScript, Python, SQL, Kotlin, and Rust. JavaScript, PHP, and Ruby were at the bottom. The key languages that jobs are saying are becoming harder to find do not seem to be heading in the same direction, or at least they do not seem to have a clear path through the center or along the main diagonal. For example, although Go is becoming more popular, it is not as popular as TypeScript, Groovy, Assembly language, or even Ruby. In contrast, Visual Basic and Rust are becoming less popular, so they could either go out of style or come back into style.

CONCLUSION

A synthesized summary of the data offers direction for individuals selecting programming languages to study to optimize employment opportunities and maintain relevance in both the immediate and distant future. Expected programming language learning portfolios are determined by prospective job trajectories. Languages that are in high demand appear in many different job pathways. In contrast, learning new languages might help current learners get ahead of their future competition. However, these evaluations are not set in stone, and one cannot always trust the possible popularity contests. To find languages that will be important in the future, one must pay close attention to changing trends as they move from marginal and experimental use to widespread use. Emerging regional pockets are a second source that the future-proofing study from today did not consider. The response to the next up-and-coming specialty languages can change swiftly to take over a slot before they become popular.

Java, Python, TypeScript, and SQL are the main languages that should be studied to become a high-paying developer. Go, Kotlin, Swift, and Python Data Sci can assist in lowering the number of graduates who want these jobs, but Rust and Dart are less likely to get them. It appears clearer what you need to do to get a job in system-level software development. People in these jobs often know C, Rust, Go, and UNIX/Linux®. For entry-level jobs, they can study C++ and Python, and as schools and colleges focus more on data science, they are becoming more familiar with a wider range of programming languages. In computer science, learning programming languages is important for creating software systems that function well, are stable, and can grow. However, there is still a dispute about how well these languages follow scientific principles.

REFERENCES

1. Castro LM. It was never about the language: paradigm impact on software design decisions [preprint]. 2020. arXiv:2010.08292. doi:10.48550/arXiv.2010.08292.
2. Alomari Z, El Halimi O, Sivaprasad K, Pandit C. Comparative studies of six programming languages [preprint]. 2015. arXiv:1504.00693. doi:10.48550/arXiv.1504.00693.
3. Tripuramallu D, Singh S, Deshmukh S, Pinisetty S, Shivaji SA, Balusamy R, et al. Towards a transpiler for C/C++ to safer Rust [preprint]. 2024. arXiv:2401.08264. doi:10.48550/arXiv.2401.08264.
4. Costanzo M, Rucci E, Naiouf M, Giusti AD. Performance vs programming effort between Rust and C on multicore architectures: case study in N-body. 2021 XLVII Latin American Computing Conference (CLEI), Cartago, Costa Rica, 2021. p. 1–10. doi:10.1109/CLEI53233.2021.9640225.
5. Zeng A, Crichton W. Identifying barriers to adoption for Rust through online discourse [preprint]. 2019. arXiv:1901.01001. doi:10.48550/arXiv.1901.01001.
6. Giorgi FM, Ceraolo C, Mercatelli D. The R language: an engine for bioinformatics and data science. Life (Basel). 2022;12(5):648. doi:10.3390/life12050648. PMID: 35629316.
7. Churavy V, Godoy WF, Bauer C, Ranocha H, Schlottke-Lakemper M, Räss L, et al. Bridging HPC communities through the Julia programming language [preprint]. 2022. arXiv:2211.02740. doi:10.48550/arXiv.2211.02740.
8. Hansen MF, Oonyawongse R, Ackert JE. Rate of development, viability, vigor, and virulence of Ascaridia galli ova cultured respectively in air and in water. Am Midl Nat. 1953;49(3):743–750. doi:10.2307/2485206.
9. Reid IR. Relationships among body mass, its components, and bone. Bone. 2002;31:547–555. doi:10.1016/S8756-3282(02)00864-5. PMID: 12477567.
10. Putranto BPD, Saptoto R, Jakaria OC, Andriyani W. A comparative study of Java and Kotlin for Android mobile application development. 2020 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), Yogyakarta, Indonesia, 2020. p. 383–388. doi:10.1109/ISRITI51436.2020.9315483.
11. Goodwill J, Matlock W. The Swift programming language. In: Beginning Swift Games Development for iOS. Berkeley (CA): Apress; 2015. p. 219–244. doi:10.1007/978-1-4842-0400-9_17.
12. Zhang Y, Liu M, Wang H, Ma Y, Huang G, Liu X. Research on WebAssembly runtimes: a survey. ACM Trans Softw Eng Methodol. 2025;34(8):1–47. doi:10.1145/3714465.
13. Hans VB. The scientific foundations of programming languages: bridging theory and practical application. Int J Comput Sci Lang. 2025;3(1):32–41.

