

Give me please a Popularity list (or table) of programming languages between 1960 and 2020.

ChatGPT

Creating a popularity list of programming languages from 1960 to 2020 involves understanding the historical context and evolution of programming languages over six decades. The list below summarizes some of the most popular programming languages for each decade, highlighting their rise in popularity and significance during those periods.

1960s:

1. **FORTRAN** (1957) - Dominant in scientific and engineering applications.
2. **COBOL** (1959) - Widely used in business, finance, and administrative systems.
3. **Lisp** (1958) - Important in academic research, particularly in artificial intelligence.

1970s:

1. **C** (1972) - Became extremely popular due to its efficiency and control, leading to its use in system/software development.
2. **Pascal** (1970) - Used extensively in teaching and academia.
3. **BASIC** (1964) - Popular among beginners and in educational contexts.

1980s:

1. **C** - Continued dominance, particularly in systems programming and application development.
2. **C++** (1985) - Gained popularity for its object-oriented features, extending C.
3. **Ada** (1980) - Used by the Department of Defense and in safety-critical applications.
4. **Smalltalk** (1980) - Pioneered object-oriented programming.
5. **Perl** (1987) - Gained traction for scripting and text manipulation.

1990s:

1. **Java** (1995) - Became widely popular for its portability across platforms (Write Once, Run Anywhere).
2. **Python** (1991) - Started to gain popularity for its simplicity and readability.
3. **JavaScript** (1995) - Essential for web development.
4. **PHP** (1995) - Popular for server-side web development.
5. **Visual Basic** (1991) - Used for rapid application development in Windows environments.

2000s:

1. **Java** - Continued strong presence in enterprise environments.
2. **C#** (2000) - Emerged as a powerful language for Windows development with the .NET framework.
3. **Python** - Gained significant popularity in web development, data science, and scripting.
4. **JavaScript** - Became crucial with the rise of dynamic web applications.
5. **Ruby** (1995) - Gained attention with the Ruby on Rails framework for web development.

2010s:

1. **JavaScript** - Dominated due to the rise of frameworks and libraries like React, Angular, and Node.js.
2. **Python** - Exploded in popularity for data science, machine learning, and general scripting.
3. **Java** - Maintained strong presence, particularly in Android development.
4. **C#** - Continued to be significant in enterprise environments and game development (with Unity).
5. **Swift** (2014) - Became popular for iOS and macOS app development.
6. **Go** (2009) - Gained traction for its efficiency and simplicity in backend development.
7. **Rust** (2010) - Recognized for its safety and performance, particularly in systems programming.

2020:

1. **Python** - Recognized as the most popular language due to its versatility in various domains.
2. **JavaScript** - Continued to be indispensable for web development.
3. **Java** - Remained crucial for enterprise applications.
4. **C#** - Continued relevance in enterprise and game development.
5. **Go** - Gaining further adoption for cloud and backend services.
6. **Rust** - Increasingly recognized for systems programming and safety features.

Popularity Trends

- **1960s-1970s:** Dominance of languages like FORTRAN, COBOL, and C.
- **1980s:** Rise of C, C++, and specialized languages like Ada and Perl.
- **1990s:** Shift towards web development with Java, JavaScript, PHP, and Visual Basic.
- **2000s:** Consolidation of Java, C#, and the emergence of Python and JavaScript as key players.
- **2010s:** Explosion of JavaScript and Python due to their roles in web development and data science, with newer languages like Swift, Go, and Rust making significant impacts.

These trends reflect the evolving needs and advancements in technology, from early scientific computing to modern web development, data science, and systems programming.

Give me please the development time duration list (or table) of individual programming languages between 1960 and 2020.

ChatGPT

Here is a list of some prominent programming languages developed between 1960 and 2020, along with their initial development and release years. This table includes the approximate time duration for their development based on available historical records.

Programming Language	Initial Development Year	Release Year	Duration
FORTRAN	1954	1957	3 years
Lisp	1956	1958	2 years
COBOL	1959	1960	1 year
ALGOL 60	1958	1960	2 years
BASIC	1963	1964	1 year

Programming Language	Initial Development Year	Release Year	Duration
PL/I	1964	1966	2 years
Pascal	1968	1970	2 years
C	1969	1972	3 years
Prolog	1972	1972	~0 years
ML	1973	1973	~0 years
Ada	1977	1980	3 years
Smalltalk	1972	1980	8 years
C++	1979	1985	6 years
Objective-C	Early 1980s	1984	~4 years
Perl	1987	1987	~0 years
Python	1989	1991	2 years
Haskell	1987	1990	3 years
Visual Basic	1990	1991	1 year
Ruby	1993	1995	2 years
Java	1991	1995	4 years
PHP	1994	1995	1 year
JavaScript	1995	1995	~0 years
Lua	1993	1994	1 year
Delphi	1993	1995	2 years
R	1992	1995	3 years
C#	1999	2000	1 year
Scala	2001	2003	2 years
F#	2002	2005	3 years
Go	2007	2009	2 years
Rust	2006	2010	4 years
Kotlin	2010	2011	1 year
Swift	2010	2014	4 years
Julia	2009	2012	3 years

Notes:

- The **initial development year** marks when the language was first conceptualized or when serious development began.
- The **release year** indicates when the language was publicly released or reached a stable version.
- **Duration** is the time between the initial development and the public release.

These durations reflect the complexity and scope of the languages, as well as the resources available to their developers. Some languages were developed rapidly to meet specific needs, while others underwent extensive design and iteration before release.

Thanks. Now I would like to have the definition of "Duration" the lifetime of language between first and last public release.

ChatGPT

To define the "duration" of a programming language as the lifetime between its first and last public release, we need to consider the span of time from the initial release of the language to its latest stable or significant release version. Here's a list of some prominent programming languages developed between 1960 and 2020, along with their first release year, last significant release year (up to 2020), and their duration.

Duration of Programming Languages (First to Last Public Release):

Programming Language	First Release Year	Last Significant Release Year	Duration (Years)
FORTRAN	1957	2020	63
Lisp	1958	2020	62
COBOL	1960	2020	60
ALGOL 60	1960	1973	13
BASIC	1964	2020 (various dialects)	56
PL/I	1966	2020 (IBM Enterprise PL/I)	54
Pascal	1970	2020 (Free Pascal)	50
C	1972	2018 (C18 standard)	46
Prolog	1972	2020	48
ML	1973	2005 (Standard ML 1997 revised)	32
Ada	1980	2012 (Ada 2012 standard)	32
Smalltalk	1980	2020	40
C++	1985	2020 (C++20 standard)	35
Objective-C	1984	2016 (Last major updates)	32
Perl	1987	2020 (Perl 5/6)	33
Python	1991	2020 (Python 3.9)	29
Haskell	1990	2020	30
Visual Basic	1991	2008 (VB 6) / 2020 (VB .NET)	29 (VB 6) / 29 (VB .NET)
Ruby	1995	2020 (Ruby 2.7)	25
Java	1995	2020 (Java 15)	25
PHP	1995	2020 (PHP 7.4)	25
JavaScript	1995	2020 (ECMAScript 2020)	25
Lua	1994	2020 (Lua 5.4)	26
Delphi	1995	2020 (Delphi 10.4 Sydney)	25
R	1995	2020 (R 4.0.3)	25
C#	2000	2020 (C# 9.0)	20
Scala	2003	2020 (Scala 2.13)	17
F#	2005	2020 (F# 4.7)	15
Go	2009	2020 (Go 1.15)	11
Rust	2010	2020 (Rust 1.47)	10
Kotlin	2011	2020 (Kotlin 1.4)	9
Swift	2014	2020 (Swift 5.3)	6
Julia	2012	2020 (Julia 1.5)	8

Notes:

- The **first release year** marks when the language was first publicly available.
- The **last significant release year** indicates the most recent stable release or significant update up to 2020.
- **Duration** is the number of years between the first and last significant release.

This table provides a snapshot of the longevity and sustained development of various programming languages over time.

Can you extend the same table with a column that marks by whom the language was standardized, like: "None" / "de facto standard" / "formal standard" (by whom)

ChatGPT

Certainly! Below is the extended table including a column indicating whether the language was standardized, and if so, by whom.

Duration and Standardization of Programming Languages

Programming Language	First Release Year	Last Significant Release Year	Duration (Years)	Standardization
FORTRAN	1957	2020	63	Formal standard (ANSI, ISO)
Lisp	1958	2020	62	None
COBOL	1960	2020	60	Formal standard (ANSI, ISO)
ALGOL 60	1960	1973	13	Formal standard (ISO)
BASIC	1964	2020 (various dialects)	56	None / De facto standard
PL/I	1966	2020 (IBM Enterprise PL/I)	54	Formal standard (ANSI)
Pascal	1970	2020 (Free Pascal)	50	Formal standard (ISO)
C	1972	2018 (C18 standard)	46	Formal standard (ANSI, ISO)
Prolog	1972	2020	48	Formal standard (ISO)
ML	1973	2005 (Standard ML 1997 revised)	32	De facto standard
Ada	1980	2012 (Ada 2012 standard)	32	Formal standard (ISO)
Smalltalk	1980	2020	40	None
C++	1985	2020 (C++20 standard)	35	Formal standard (ISO)
Objective-C	1984	2016 (Last major updates)	32	None
Perl	1987	2020 (Perl 5/6)	33	None
Python	1991	2020 (Python 3.9)	29	De facto standard
Haskell	1990	2020	30	De facto standard
Visual Basic	1991	2008 (VB 6) / 2020	29 (VB 6) / 29	None / De facto

Programming Language	First Release Year	Last Significant Release Year	Duration (Years)	Standardization
		(VB .NET)	(VB .NET)	standard
Ruby	1995	2020 (Ruby 2.7)	25	De facto standard
Java	1995	2020 (Java 15)	25	Formal standard (ISO, ECMA)
PHP	1995	2020 (PHP 7.4)	25	De facto standard
JavaScript	1995	2020 (ECMAScript 2020)	25	Formal standard (ECMA)
Lua	1994	2020 (Lua 5.4)	26	De facto standard
Delphi	1995	2020 (Delphi 10.4 Sydney)	25	None
R	1995	2020 (R 4.0.3)	25	De facto standard
C#	2000	2020 (C# 9.0)	20	Formal standard (ECMA, ISO)
Scala	2003	2020 (Scala 2.13)	17	De facto standard
F#	2005	2020 (F# 4.7)	15	De facto standard
Go	2009	2020 (Go 1.15)	11	De facto standard
Rust	2010	2020 (Rust 1.47)	10	De facto standard
Kotlin	2011	2020 (Kotlin 1.4)	9	De facto standard
Swift	2014	2020 (Swift 5.3)	6	None
Julia	2012	2020 (Julia 1.5)	8	De facto standard

Notes:

- **Formal standard** indicates the language has an official standard maintained by recognized bodies such as ANSI, ISO, ECMA, etc.
- **De facto standard** indicates that the language has widely accepted conventions and standards, but no official standardization body.
- **None** indicates that the language does not have an official or widely recognized standard.

40

What are the reasons if a programming language is getting a standardization status "none", "Formal standard", "de facto Standard"

ChatGPT

The status of a programming language's standardization—whether "none," "formal standard," or "de facto standard"—depends on various factors, including the language's history, development process, community adoption, and use cases. Here are some reasons for each standardization status:

None

1. Niche or Specialized Use:

- Some languages are designed for very specific tasks or niches and do not gain wide enough adoption to warrant formal standardization.

2. Proprietary Development:

- Languages developed and controlled by a single company or organization (e.g., Objective-C by Apple) may not seek or achieve formal standardization.

3. Rapid Evolution:

- Languages that evolve rapidly or have frequent changes may not settle into a standardized form (e.g., Swift).

4. Experimental Nature:

- Some languages are experimental or research-oriented, focusing more on exploring new concepts rather than widespread industry adoption.

Formal Standard

1. Wide Adoption and Industry Demand:

- Languages that become widely adopted across different industries often seek formal standardization to ensure consistency and interoperability (e.g., C, C++).

2. Longevity and Stability:

- Mature languages that have been in use for a long time are more likely to be standardized formally (e.g., COBOL, FORTRAN).

3. Diverse Implementations:

- Languages that have multiple implementations benefit from a formal standard to ensure compatibility and avoid fragmentation (e.g., SQL, Ada).

4. Regulatory or Government Use:

- Languages used in regulated environments, such as government or safety-critical systems, often require formal standardization (e.g., Ada).

5. Organizational Backing:

- Standardization bodies (ANSI, ISO, ECMA) and industry consortiums may back the standardization process, providing resources and formal procedures (e.g., Java by ISO and ECMA).

De Facto Standard

1. Widespread Community Adoption:

- Some languages become de facto standards due to their widespread use and community-driven development, even without formal standardization (e.g., Python, JavaScript).

2. Dominant Implementation:

- When a single implementation of a language is so dominant that it sets the standard for others, it can become a de facto standard (e.g., Perl, Ruby).

3. Open Source Ecosystem:

- Open source languages often rely on community consensus rather than formal standardization, leading to de facto standards (e.g., Go, Rust).

4. Flexibility and Evolution:

- De facto standards can evolve more flexibly and quickly than formally standardized languages, adapting to new needs and technologies.

5. **Vendor Neutrality:**

- Languages that are developed and maintained by open communities rather than single vendors often become de facto standards due to their neutral, inclusive development process.

Summary

- **None:** Often niche, proprietary, rapidly evolving, or experimental languages without broad adoption or multiple implementations.
- **Formal Standard:** Widely adopted, stable, mature languages often used in regulated or critical systems, backed by formal standardization bodies.
- **De Facto Standard:** Languages widely adopted by communities and industry, usually open source or driven by a dominant implementation, with flexible and rapid evolution.

Each standardization status reflects the language's development history, community, and usage context, balancing the need for consistency with the benefits of flexibility.