# Swift (programming language)

**Swift** is a high-level general-purpose, multi-paradigm, compiled programming language created by Chris Lattner in 2010 for Apple Inc. and maintained by the open-source community. Swift compiles to machine code, as it is an LLVM-based compiler. Swift was first released in June 2014,[11] and the Swift toolchain has shipped in Xcode since version 6, released in 2014.

Apple intended Swift to support many core concepts associated with Objective-C, notably dynamic dispatch, widespread late binding, extensible programming, and similar features, but in a "safer" way, making it easier to catch software bugs; Swift has features addressing some common programming errors like null pointer dereferencing and provides syntactic sugar to help avoid the pyramid of doom. Swift supports the concept of protocol extensibility, an extensibility system that can be applied to types, structs and classes, which Apple promotes as a real change in programming paradigms they term "protocol-oriented programming"[12] (similar to traits and type classes).[13]

Swift was introduced at Apple's 2014 Worldwide Developers Conference (WWDC).[14] It underwent an upgrade to version 1.2 during 2014 and a major upgrade to Swift 2 at WWDC 2015. Initially a proprietary language, version 2.2 was made open-source software under the Apache License 2.0 on December 3, 2015, for Apple's platforms and Linux.[15][16]

Through version 3.0 the syntax of Swift went through significant evolution, with the core team making source stability a focus in later versions.[17][18] In the first quarter of 2018 Swift surpassed Objective-C in measured popularity.[19]

Swift 4.0, released in 2017, introduced several changes to some built-in classes and structures. Code written with previous versions of Swift can be updated using the migration functionality built into Xcode. Swift 5, released in March 2019, introduced a stable binary interface on Apple platforms, allowing the Swift runtime to be incorporated into Apple operating systems. It is source compatible with Swift 4.[20]

Swift 5.1 was officially released in September 2019. Swift 5.1 builds on the previous version of Swift 5 by extending the stable features of the language to compile-time with the introduction of module stability. The introduction of module stability makes it possible to create and share binary frameworks that will work with future releases of Swift.[21]

Swift 5.5, officially announced by Apple at the 2021 WWDC, significantly expands language support for concurrency and asynchronous code, notably introducing a unique version of the actor model.[22]

**Swift**

Logo

| | |
|---|---|
| **Paradigm** | Multi-paradigm: protocol-oriented, object-oriented, functional, imperative, block structured, declarative, concurrent |
| **Designed by** | Chris Lattner, Doug Gregor, John McCall, Ted Kremenek, Joe Groff, and Apple Inc.[1] |
| **Developer** | Apple Inc. and open-source contributors |
| **First appeared** | June 2, 2014[2] |
| **Stable release** | 5.10[3] / 5 March 2024 |
| **Preview release** | 5.9 |
| **Typing discipline** | Static, strong, inferred |
| **Memory management** | Automatic Reference Counting |
| **OS** | Apple's operating systems (Darwin, iOS, iPadOS, macOS, tvOS, watchOS), Linux, Windows, Android, z/OS |
| **License** | Apache License 2.0 (Swift 2.2 and |

Swift 5.9, was released in September 2023 and includes a macro system, generic parameter packs, and ownership features like the new `consume` operator.[23]

The current version, Swift 5.10, was released in March 2024. This version improves the language's concurrency model, allowing for full data isolation to prevent data races. It is also the last release before Swift 6.[24] Version 5.10 is currently available for macOS, Windows and, in experimental use, for Linux.[25]

| | |
|---|---|
| | later)<br>Proprietary (up to Swift 2.2)[4][5] |
| **Filename extensions** | .swift, .SWIFT |
| **Website** | www.swift.org (https://www.swift.org/)<br>developer.apple.com/swift/ (https://developer.apple.com/swift/) |
| **Influenced by** | |
| Objective-C,[6] Rust, Haskell, Ruby, Python, C#, CLU,[7] D[8] | |
| **Influenced** | |
| Rust,[9] V (Vlang)[10] | |

# History

Development of Swift started in July 2010 by Chris Lattner, with the eventual collaboration of many other programmers at Apple. Swift was motivated by the need for a replacement for Apple's earlier programming language Objective-C, which had been largely unchanged since the early 1980s and lacked modern language features. Swift took language ideas "from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list".[7] On June 2, 2014, the Apple Worldwide Developers Conference (WWDC) application became the first publicly released app written with Swift.[26] A beta version of the programming language was released to registered Apple developers at the conference, but the company did not promise that the final version of Swift would be source code compatible with the test version. Apple planned to make source code converters available if needed for the full release.[26]

*The Swift Programming Language*, a free 500-page manual, was also released at WWDC, and is available on the Apple Books Store and the official website.[27]

Swift reached the 1.0 milestone on September 9, 2014, with the *Gold Master* of Xcode 6.0 for iOS.[28] Swift 1.1 was released on October 22, 2014, alongside the launch of Xcode 6.1.[29] Swift 1.2 was released on April 8, 2015, along with Xcode 6.3.[30] Swift 2.0 was announced at WWDC 2015, and was made available for publishing apps in the App Store on September 21, 2015.[31] Swift 3.0 was released on September 13, 2016.[32] Swift 4.0 was released on September 19, 2017.[33] Swift 4.1 was released on March 29, 2018.[34]

Swift won first place for *Most Loved Programming Language* in the Stack Overflow Developer Survey 2015[35] and second place in 2016.[36]

On December 3, 2015, the Swift language, supporting libraries, debugger, and package manager were open-sourced under the Apache 2.0 license with a Runtime Library Exception,[37] and Swift.org (https://swift.org) was created to host the project. The source code is hosted on GitHub (https://github.com/apple/swift), where it is easy for anyone to get the code, build it themselves, and even create pull requests to contribute code back to the project.

In December 2015, IBM announced its Swift Sandbox website, which allows developers to write Swift code in one pane and display output in another.[38][39][40] The Swift Sandbox was deprecated in January 2018.[41]

During the WWDC 2016, Apple announced an iPad exclusive app, named Swift Playgrounds, intended to teach people how to code in Swift. The app is presented in a 3D video game-like interface which provides feedback when lines of code are placed in a certain order and executed.[42][43][44]

In January 2017, Chris Lattner announced his departure from Apple for a new position with Tesla Motors, with the Swift project lead role going to team veteran Ted Kremenek.[45][46]

During WWDC 2019, Apple announced SwiftUI with Xcode 11, which provides a framework for declarative UI structure design across all Apple platforms.[47]

Official downloads for the Ubuntu distribution of Linux have been available since Swift 2.2, with more distros added since Swift 5.2.4, CentOS and Amazon Linux.[48] There is an unofficial SDK and native toolchain package for Android too.[49][50]

## Platforms

The platforms Swift supports are Apple's operating systems (Darwin, iOS, iPadOS, macOS, tvOS, watchOS), Linux, Windows, and Android.[51][52]

A key aspect of Swift's design is its ability to interoperate with the huge body of existing Objective-C code developed for Apple products over the previous decades, such as Cocoa and the Cocoa Touch frameworks. On Apple platforms,[53] it links with the Objective-C runtime library, which allows C, Objective-C, C++ and Swift code to run within one program.[54]

## Version history

| Swift version | Release date | macOS | Linux | Windows |
|---|---|---|---|---|
| 1.0 | September 9, 2014 | Yes | No | No |
| 1.1 | October 22, 2014 | Yes | No | No |
| 1.2 | April 8, 2015 | Yes | No | No |
| 2.0 | September 21, 2015 | Yes | No | No |
| 2.1 | October 20, 2015 | Yes | No | No |
| 2.2 | March 21, 2016 | Yes | Yes | No |
| 2.2.1 | May 3, 2016 | Yes | Yes | No |
| 3.0 | September 13, 2016 | Yes | Yes | No |
| 3.0.1 | October 28, 2016 | Yes | Yes | No |
| 3.0.2 | December 13, 2016 | Yes | Yes | No |
| 3.1 | March 27, 2017 | Yes | Yes | No |
| 3.1.1 | April 21, 2017 | Yes | Yes | No |
| 4.0 | September 19, 2017 | Yes | Yes | No |
| 4.0.2 | November 1, 2017 | Yes | Yes | No |
| 4.0.3 | December 5, 2017 | Yes | Yes | No |
| 4.1 | March 29, 2018 | Yes | Yes | No |
| 4.1.1 | May 4, 2018 | No | Yes | No |
| 4.1.2 | May 31, 2018 | Yes | Yes | No |
| 4.1.3 | July 27, 2018 | No | Yes | No |
| 4.2 | September 17, 2018 | Yes | Yes | No |
| 4.2.1 | October 30, 2018 | Yes | Yes | No |
| 4.2.2 | February 4, 2019 | No | Yes | No |
| 4.2.3 | February 28, 2019 | No | Yes | No |
| 4.2.4 | March 29, 2019 | No | Yes | No |
| 5.0[55] | March 25, 2019 | Yes | Yes | No |
| 5.0.1 | April 18, 2019 | Yes | Yes | No |
| 5.0.2 | July 15, 2019 | No | Yes | No |
| 5.0.3 | August 30, 2019 | No | Yes | No |
| 5.1 | September 10, 2019 | Yes | Yes | No |
| 5.1.1 | October 11, 2019 | No | Yes | No |
| 5.1.2 | November 7, 2019 | Yes | Yes | No |
| 5.1.3 | December 13, 2019 | Yes | Yes | No |
| 5.1.4 | January 31, 2020 | No | Yes | No |
| 5.1.5 | March 9, 2020 | No | Yes | No |
| 5.2 | March 24, 2020 | Yes | Yes | No |
| 5.2.1 | March 30, 2020 | No | Yes | No |
| 5.2.2 | April 15, 2020 | Yes | Yes | No |
| 5.2.3 | April 29, 2020 | No | Yes | No |

| Swift version | Release date | macOS | Linux | Windows |
|---|---|---|---|---|
| 5.2.4 | May 20, 2020 | Yes | Yes | No |
| 5.2.5 | August 5, 2020 | No | Yes | No |
| 5.3 | September 16, 2020 | Yes | Yes | Yes[56] |
| 5.3.1 | November 13, 2020 | Yes | Yes | Yes |
| 5.3.2 | December 15, 2020 | Yes | Yes | Yes |
| 5.3.3 | January 25, 2021 | No | Yes | Yes |
| 5.4[57] | April 26, 2021 | Yes | Yes | Yes |
| 5.4.1 | May 25, 2021 | No | Yes | Yes |
| 5.4.2 | June 28, 2021 | Yes | Yes | Yes |
| 5.4.3 | September 9, 2021 | No | Yes | Yes |
| 5.5 | September 20, 2021 | Yes | Yes | Yes |
| 5.5.1 | October 27, 2021 | Yes | Yes | Yes |
| 5.5.2 | December 14, 2021 | Yes | Yes | Yes |
| 5.5.3 | February 9, 2022 | No | Yes | Yes |
| 5.6[58] | March 14, 2022 | Yes | Yes | Yes |
| 5.6.1[59] | April 9, 2022 | No | Yes | Yes |
| 5.6.2[60] | June 15, 2022 | No | Yes | Yes |
| 5.6.3[61] | September 2, 2022 | No | Yes | Yes |
| 5.7[62] | September 12, 2022 | Yes | Yes | Yes |
| 5.7.1[63] | November 1, 2022 | Yes | Yes | Yes |
| 5.8[64] | March 30, 2023 | Yes | Yes | Yes |
| 5.8.1[65] | June 1, 2023 | Yes | Yes | Yes |
| 5.9[66] | September 18, 2023 | Yes | Yes | Yes |
| 5.9.1[67] | October 19, 2023 | Yes | Yes | Yes |
| 5.9.2[68] | December 11, 2023 | Yes | Yes | Yes |
| 5.10[24] | March 5, 2024 | Yes | Yes | Yes |

# Features

Swift is a general purpose programming language that employs modern programming-language theory concepts and strives to present a simple, yet powerful syntax. Swift incorporates innovations and conventions from various programming languages, with notable inspiration from Objective-C, which it replaced as the primary development language on Apple Platforms.

Swift was designed to be safe and friendly to new programmers while not sacrificing speed. By default Swift manages all memory automatically and ensures variables are always initialized before use. Array accesses are checked for out-of-bounds errors and integer operations are checked for overflow. Parameter names allow creating clear APIs. Protocols define interfaces that types may adopt, while extensions allow developers to add more function to existing types. Swift enables object-oriented programming with the support for classes, subtyping, and method overriding. Optionals allow nil values to be handled explicitly and safely. Concurrent

programs can be written using async/await syntax and actors isolate shared mutable state in order to eliminate data races.[69][70]

## Basic syntax

Swift's syntax is similar to C-style languages. Code begins executing in the global scope by default.[71] Alternatively, the `@main` attribute can be applied a structure, class, or enumeration declaration to indicate that it contains the program's entry point.[72]

Swift's "Hello, World!" program is:

```
print("Hello, world!")
```

The `print(_:separator:terminator:)` function used here is included in Swift's standard library, which is available to all programs without the need to import external modules. Statements in Swift don't have to end with a semicolon, however semicolons are required to separate multiple statements written on the same line. Single-line comments begin with `//` and continue until the end of the current line. Multiline comments are contained by `/*` and `*/` characters. Constants are declared with the **`let`** keyword and variables with the **`var`** keyword. Values must be initialized before they are read. Values may infer their type based on the type of the provided initial value. If the initial value is set after the value's declaration, a type must be declared explicitly.[71]

```
let highScoreThreshold = 1000 // A constant with type Int. The type was inferred based on the provided value.

var currentScore = 980 // A variable with type Int.
currentScore = 1200 // The value of variables can change over time.

let playerMessage: String // A constant with explicit type String.
if currentScore > highScoreThreshold {
    playerMessage = "You are a top player!"
} else {
    playerMessage = "Better luck next time."
}

print(playerMessage) // Prints "You are a top player!"
```

Control flow in Swift is managed with if-else, guard, and switch statements, along with while and for-in loops. If statements take a boolean parameter and execute the body of the if statement if the condition is true, otherwise it executes the optional **`else`** body. **`if-let`** syntax provides syntactic sugar for checking for the existence of an optional value and unwrapping it at the same time.

```
let someNumber = 42
if someNumber % 2 == 0 { // Use the remainder operator to find the remainder of someNumber divided by 2.
    print("\(someNumber) is even.")
} else{
   print("\(someNumber) is odd.")
}

// Prints "42 is even."
```

Functions are defined with the **`func`** keyword. Function parameters may have names which allow function calls to read like phrases. An underscore before the parameter name allows the argument label to be omitted from the call site. Tuples can be used by functions to return multiple pieces of data at once.

```
func constructGreeting(for name: String) -> String {
    return "Hello \(name)!"
}

let greeting = constructGreeting(for: "Craig")
```

```
print(greeting) // Prints "Hello Craig!"
```

Functions, and anonymous functions known as closures, can be assigned to properties and passed around the program like any other value.

```
func divideByTwo(_ aNum: Int) -> Int {
    return aNum / 2
}

func multiplyByTwo(_ aNum: Int) -> Int {
    return aNum * 2
}

let mathOperation = multiplyByTwo

print(mathOperation(21)) // Prints "42"
```

**`guard`** statements require that the given condition is true before continuing on past the **`guard`** statement, otherwise the body of the provided **`else`** clause is run. The **`else`** clause must exit control of the code block in which the **`guard`** statement appears. **`guard`** statements are useful for ensuring that certain requirements are met before continuing on with program execution. In particular they can be used to create an unwrapped version of an optional value that is guaranteed to be non-nil for the remainder of the enclosing scope.

```
func divide(numerator: Int?, byDenominator denominator: Int) -> Int? {
    guard denominator != 0 else {
        print("Can't divide by 0.")
        return nil
    }

    guard let numerator else {
        print("The provided numerator is nil.")
        return nil
    }

    return numerator / denominator
}

let result = divide(numerator: 3, byDenominator: 0)
print("Division result is: \(result)")

// Prints:
// "Can't divide by 0."
// "Division result is: nil."
```

`switch` statements compare a value with multiple potential values and then executes an associated code block. `switch` statements must be made exhaustive, either by including cases for all possible values or by including a `default` case which is run when the provided value doesn't match any of the other cases. `switch` cases do not implicitly fall through, although they may explicitly do so with the `fallthrough` keyword. Pattern matching can be used in various ways inside `switch` statements. Here is an example of an integer being matched against a number of potential ranges:

```swift
let someNumber = 42

switch someNumber {
case ..<0:
    print("\(someNumber) negative.")
case 0:
    print("\(someNumber) is 0.")
case 1...9:
    print("\(someNumber) greater than 0, but less than 10.")
default:
    print("\(someNumber) is greater than 9.")
}

// Prints "42 is greater than 9."
```

`for-in` loops iterate over a sequence of values:

```swift
let names = ["Will", "Anna", "Bart"]
for name in names {
    print(name)
}
// Prints:
// Will
// Anna
// Bart
```

`while` loops iterate as long as the given boolean condition evaluates to `true`:

```swift
// Add together all the numbers from 1 to 5.
var i = 1
var result = 0

while i <= 5 { // The loop performs its body as long as i is less than or equal to 5.
    result += i // Add i to the current result.
    i += 1 // Increment i by 1.
}

print(result) // Prints "15"
```

## Closure support

Swift supports closures, which are self-contained blocks of functionality that can be passed around and used in code,[73] and can also be used as anonymous functions. Here are some examples:

```swift
// Closure type, defined by its input and output values, can be specified outside the closure:
let closure1: (Int, Int) -> Int = { arg1, arg2 in
    return arg1 + arg2
}

// …or inside it:
let closure2 = { (arg1: Int, arg2: Int) -> Int in
    return arg1 + arg2
}

// In most cases, closure's return type can be inferred automatically by the compiler.
let closure3 = { arg1: Int, arg2: Int in
    return arg1 + arg2
}
```

Closures can be assigned to variables and constants, and can be passed into other functions or closures as parameters. Single-expression closures may drop the **`return`** keyword.

Swift also has a trailing closure syntax, which allows the closure to be written after the end of the function call instead of within the function's parameter list. Parentheses can be omitted altogether if the closure is the

function's only parameter:

```
// This function takes a closure which receives no input parameters and returns an integer,
// evaluates it, and uses the closure's return value (an Int) as the function's return value.
func foo(closure bar: () -> Int) -> Int {
    return bar()
}

// Without trailing closure syntax:
foo(closure: { return 1 })

// With trailing closure syntax, and implicit return:
foo { 1 }
```

Starting from version 5.3, Swift supports multiple trailing closures:[74]

```
// This function passes the return of the first closure as the parameter of the second,
// and returns the second closure's result:
func foo(bar: () -> Int, baz: (Int) -> Int) -> Int {
    return baz(bar())
}

// With no trailing closures:
foo(bar: { return 1 }, baz: { x in return x + 1 })

// With 1 trailing closure:
foo(bar: { return 1 }) { x in return x + 1 }

// With 2 trailing closures (only the first closure's argument name is omitted):
foo { return 1 } baz: { x in return x + 1 }
```

Swift will provide shorthand argument names for inline closures, removing the need to explicitly name all of the closures parameters.[75] Arguments can be referred to with the names $0, $1, $2, and so on:

```
let names = ["Josephine", "Steve", "Chris", "Barbara"]

// filter calls the given closure for each value in names.
// Values with a character count less than 6 are kept, the others are dropped.
let shortNames = names.filter { $0.count < 6 }

print(shortNames) // Prints "["Steve", "Chris"]"
```

Closures may capture values from their surrounding scope. The closure will refer to this captured value for as long as the closure exists:

```
func makeMultiplier(withMultiple multiple: Int) -> (Int) -> (Int) {
    // Create and return a closure that takes in an Int and returns the input multiplied by the value of multiple.
    return {
        $0 * multiple
    }
}

let multiplier = makeMultiplier(withMultiple: 3)
print(multiplier(3)) // Prints "9"
print(multiplier(10)) // Prints "30"
```

## String support

The Swift standard library includes unicode-compliant `String` and `Character` types. String values can be initialized with a String literal, a sequence of characters surrounded by double quotation marks. Strings can be concatenated with the + operator:

```
var someString = "Hello,"
```

```
someString += " world!"
```

String interpolation allows for the creation of a new string from other values and expressions. Values written between parentheses preceded by a \ will be inserted into the enclosing string literal:[76]

```
var currentScore = 980
print("Your score is \(currentScore).")

// Prints "Your score is 980."
```

A for-in loop can be used to iterate over the characters contained in a string:

```
for character in "Swift" {
    print(character)
}
// S
// w
// i
// f
// t
```

When the Foundation framework is imported Swift invisibly bridges the String type to NSString, the String class commonly used in Objective-C.

## Callable objects

In Swift, callable objects are defined using `callAsFunction`.[77]

```
struct CallableStruct {
    var value: Int
    func callAsFunction(_ number: Int, scale: Int) {
        print(scale * (number + value))
    }
}
let callable = CallableStruct(value: 100)
callable(4, scale: 2)
callable.callAsFunction(4, scale: 2)
// Both function calls print 208.
```

## Access control

Swift supports five access control levels for symbols: `open`, **`public`**, **`internal`**, `fileprivate`, and **`private`**. Unlike many object-oriented languages, these access controls ignore inheritance hierarchies: **`private`** indicates that a symbol is accessible only in the immediate scope, `fileprivate` indicates it is accessible only from within the file, **`internal`** indicates it is accessible within the containing module, **`public`** indicates it is accessible from any module, and `open` (only for classes and their methods) indicates that the class may be subclassed outside of the module.[78]

## Optionals and chaining

An important feature in Swift is option types, which allow references or values to operate in a manner similar to the common pattern in C, where a pointer may either refer to a specific value or no value at all. This implies that non-optional types cannot result in a null-pointer error; the compiler can ensure this is not possible.

Optional types are created with the `Optional` enum. To make an Integer that is nullable, one would use a declaration similar to `var optionalInteger: Optional<Int>`. As in C#,[79] Swift also includes syntactic sugar for this, allowing one to indicate a variable is optional by placing a question mark after the type name, `var`

`optionalInteger: Int?`.[80] Variables or constants that are marked optional either have a value of the underlying type or are `nil`. Optional types *wrap* the base type, resulting in a different instance. `String` and `String?` are fundamentally different types, the former is of type `String` while the latter is an `Optional` that may be holding some `String` value.

To access the value inside, assuming it is not nil, it must be *unwrapped* to expose the instance inside. This is performed with the `!` operator:

```
let myValue = anOptionalInstance!.someMethod()
```

In this case, the `!` operator unwraps `anOptionalInstance` to expose the instance inside, allowing the method call to be made on it. If `anOptionalInstance` is nil, a null-pointer error occurs, terminating the program. This is known as force unwrapping. Optionals may be safely unwrapped using optional chaining which first tests whether the instance is nil, and then unwrap it if it is non-null:

```
let myValue = anOptionalInstance?.someMethod()
```

In this case the runtime calls `someMethod` only if `anOptionalInstance` is not nil, suppressing the error. A `?` must be placed after every optional property. If any of these properties are nil the entire expression evaluates as nil. The origin of the term *chaining* comes from the more common case where several method calls/getters are chained together. For instance:

```
let aTenant = aBuilding.tenantList[5]
let theirLease = aTenant.leaseDetails
let leaseStart = theirLease?.startDate
```

can be reduced to:

```
let leaseStart = aBuilding.tenantList[5].leaseDetails?.startDate
```

Swift's use of optionals allows the compiler to use static dispatch because the unwrapping action is called on a defined instance (the wrapper), versus occurring in a runtime dispatch system.

## Value types

In many object-oriented languages, objects are represented internally in two parts. The object is stored as a block of data placed on the heap, while the name (or "handle") to that object is represented by a pointer. Objects are passed between methods by copying the value of the pointer, allowing the same underlying data on the heap to be accessed by anyone with a copy. In contrast, basic types like integers and floating-point values are represented directly; the handle contains the data, not a pointer to it, and that data is passed directly to methods by copying. These styles of access are termed *pass-by-reference* in the case of objects, and *pass-by-value* for basic types.

Both concepts have their advantages and disadvantages. Objects are useful when the data is large, like the description of a window or the contents of a document. In these cases, access to that data is provided by copying a 32- or 64-bit value, versus copying an entire data structure. However, smaller values like integers are the same size as pointers (typically both are one word), so there is no advantage to passing a pointer, versus passing the value.

Swift offers built-in support for objects using either pass-by-reference or pass-by-value semantics, the former using the `class` declaration and the latter using `struct`. Structs in Swift have almost all the same features as

classes: methods, implementing protocols and using the extension mechanisms. For this reason, Apple terms all data generically as *instances*, versus objects or values. Structs do not support inheritance, however.[81]

The programmer is free to choose which semantics are more appropriate for each data structure in the application. Larger structures like windows would be defined as classes, allowing them to be passed around as pointers. Smaller structures, like a 2D point, can be defined as structs, which will be pass-by-value and allow direct access to their internal data with no indirection or reference counting. The performance improvement inherent to the pass-by-value concept is such that Swift uses these types for almost all common data types, including `Int` and `Double`, and types normally represented by objects, like `String` and `Array`.[81] Using value types can result in significant performance improvements in user applications as well.[82]

`Array`, `Dictionary`, and `Set` all utilize copy on write so that their data are copied only if and when the program attempts to change a value in them. This means that the various accessors have what is in effect a pointer to the same data storage. So while the data is physically stored as one instance in memory, at the level of the application, these values are separate and physical separation is enforced by copy on write only if needed.[83]

## Extensions

Extensions add new functionality to an existing type, without the need to subclass or even have access to the original source code. Extensions can add new methods, initializers, computed properties, subscripts, and protocol conformances.[84] An example might be to add a spell checker to the base `String` type, which means all instances of `String` in the program gain the ability to spell-check. The system is also widely used as an organizational technique, allowing related code to be gathered into library-like extensions.

Extensions are declared with the `extension` keyword.

```
struct Rectangle {
    let width: Double
    let height: Double
}

extension Rectangle {
    var area: Double {
        return height * width
    }
}
```

## Protocol-oriented programming

Protocols promise that a particular type implements a set of methods or properties, meaning that other instances in the system can call those methods on any instance implementing that protocol. This is often used in modern object-oriented languages as a substitute for multiple inheritance, although the feature sets are not entirely similar.

In Objective-C, and most other languages implementing the protocol concept, it is up to the programmer to ensure that the required methods are implemented in each class.[85] Swift adds the ability to add these methods using extensions, and to use generic programming (generics) to implement them. Combined, these allow protocols to be written once and support a wide variety of instances. Also, the extension mechanism can be used to add protocol conformance to an object that does not list that protocol in its definition.[86]

For example, a protocol might be declared called `Printable`, which ensures that instances that conform to the protocol implement a `description` property and a `printDetails()` method requirement:

```swift
// Define a protocol named Printable
protocol Printable {
    var description: String { get } // A read-only property requirement
    func printDetails() // A method requirement
}
```

This protocol can now be adopted by other types:

```swift
// Adopt the Printable protocol in a class
class MyClass: Printable {
    var description: String {
        return "An instance of MyClass"
    }

    func printDetails() {
        print(description)
    }
}
```

Extensions can be used to add protocol conformance to types. Protocols themselves can also be extended to provide default implementations of their requirements. Adopters may define their own implementations, or they may use the default implementation:

```swift
extension Printable { // All Printable instances will receive this implementation, or they may define their own.
    func printDetails() {
        print(description)
    }
}

// Bool now conforms to Printable, and inherits the printDetails() implementation above.
extension Bool: Printable {
    var description: String {
        return "An instance of Bool with value: \(self)"
    }

}
```

In Swift, like many modern languages supporting interfaces, protocols can be used as types, which means variables and methods can be defined by protocol instead of their specific type:

```swift
func getSomethingPrintable() -> any Printable {
    return true
}

var someSortOfPrintableInstance = getSomethingPrintable()
print(someSortOfPrintableInstance.description)

// Prints "An instance of Bool with value: true"
```

It does not matter what concrete type of `someSortOfPrintableInstance` is, the compiler will ensure that it conforms to the protocol and thus this code is safe. This syntax also means that collections can be based on protocols also, like `let printableArray = [any Printable]`.

Both extensions and protocols are used extensively in Swift's standard library; in Swift 5.9, approximately 1.2 percent of all symbols within the standard library were protocols, and another 12.3 percent were protocol requirements or default implementations.[87] For instance, Swift uses extensions to add the `Equatable` protocol to many of their basic types, like Strings and Arrays, allowing them to be compared with the `==` operator. The `Equatable` protocol also defines this default implementation:

```swift
func !=<T : Equatable>(lhs: T, rhs: T) -> Bool
```

This function defines a method that works on any instance conforming to `Equatable`, providing a *not equals* operator. Any instance, class or struct, automatically gains this implementation simply by conforming to `Equatable`.[88]

Protocols, extensions, and generics can be combined to create sophisticated APIs. For example, constraints allow types to conditionally adopt protocols or methods based on the characteristics of the adopting type. A common use case may be adding a method on collection types only when the elements contained within the collection are `Equatable`:

```
extension Array where Element: Equatable {

    // allEqual will be available only on instances of Array that contain Equatable elements.
    func allEqual() -> Bool {
        for element in self {
            if element != self.first {
                return false
            }
        }
        return true
    }
}
```

## Concurrency

Swift 5.5 introduced structured concurrency into the language.[89] Structured concurrency uses Async/await syntax similar to Kotlin, JavaScript, and Rust. An async function is defined with the `async` keyword after the parameter list. When calling an async function the `await` keyword must be written before the function to indicate that execution will potentially suspend while calling function. While a function is suspended the program may run some other concurrent function in the same program. This syntax allows programs to clearly call out potential suspension points and avoid a version of the Pyramid of doom (programming) caused by the previously widespread use of closure callbacks.[90]

```
func downloadText(name: String) async -> String {
    let result = // ... some asynchronous downloading code ...
    return result
}

let text = await downloadText("text1")
```

The `async let` syntax allows multiple functions to run in parallel. `await` is again used to mark the point at which the program will suspend to wait for the completion of the `async` functions called earlier.

```
// Each of these calls to downloadText will run in parallel.
async let text1 = downloadText(name: "text1")
async let text2 = downloadText(name: "text2")
async let text3 = downloadText(name: "text3")

let textToPrint = await [text1, text2, text3] // Suspends until all three downloadText calls have returned.
print(textToPrint)
```

Tasks and TaskGroups can be created explicitly to create a dynamic number of child tasks during runtime:

```
let taskHandle = Task {
    await downloadText(name: "someText")
}

let result = await taskHandle.value
```

Swift uses the Actor model to isolate mutable state, allowing different tasks to mutate shared state in a safe

manner. Actors are declared with the `actor` keyword and are reference types, like classes. Only one task may access the mutable state of an actor at the same time. Actors may access and mutate their own internal state freely, but code running in separate tasks must mark each access with the `await` keyword to indicate that the code may suspend until other tasks finish accessing the actor's state.

```
actor Directory {
    var names: [String] = []

    func add(name: String) {
        names.append(name)
    }
}

let directory = Directory()

// Code suspends until other tasks finish accessing the actor.
await directory.add(name: "Tucker")
print(await directory.names)
```

## Libraries, runtime, development

On Apple systems, Swift uses the same runtime as the extant Objective-C system, but requires iOS 7 or macOS 10.9 or higher. It also depends on Grand Central Dispatch.[91] Swift and Objective-C code can be used in one program, and by extension, C and C++ also. Beginning in Swift 5.9, C++ code can be used directly from Swift code.[92] In the case of Objective-C, Swift has considerable access to the object model, and can be used to subclass, extend and use Objective-C code to provide protocol support.[93] The converse is not true: a Swift class cannot be subclassed in Objective-C.[94]

To aid development of such programs, and the re-use of extant code, Xcode 6 and higher offers a semi-automated system that builds and maintains a *bridging header* to expose Objective-C code to Swift. This takes the form of an additional header file that simply defines or imports all of the Objective-C symbols that are needed by the project's Swift code. At that point, Swift can refer to the types, functions, and variables declared in those imports as though they were written in Swift. Objective-C code can also use Swift code directly, by importing an automatically maintained header file with Objective-C declarations of the project's Swift symbols. For instance, an Objective-C file in a mixed project called "MyApp" could access Swift classes or functions with the code `#import "MyApp-Swift.h"`. Not all symbols are available through this mechanism, however—use of Swift-specific features like generic types, non-object optional types, sophisticated enums, or even Unicode identifiers may render a symbol inaccessible from Objective-C.[95]

Swift also has limited support for *attributes*, metadata that is read by the development environment, and is not necessarily part of the compiled code. Like Objective-C, attributes use the `@` syntax, but the currently available set is small. One example is the `@IBOutlet` attribute, which marks a given value in the code as an *outlet*, available for use within Interface Builder (IB). An *outlet* is a device that binds the value of the on-screen display to an object in code.

On non-Apple systems, Swift does not depend on an Objective-C runtime or other Apple system libraries; a set of Swift "Corelib" implementations replace them. These include a "swift-corelibs-foundation" to stand in for the Foundation Kit, a "swift-corelibs-libdispatch" to stand in for the Grand Central Dispatch, and an "swift-corelibs-xctest" to stand in for the XCTest APIs from Xcode.[96]

As of 2019, with Xcode 11, Apple has also added a major new UI paradigm called SwiftUI. SwiftUI replaces the older Interface Builder paradigm with a new declarative development paradigm.[97]

## Memory management

Swift uses Automatic Reference Counting (ARC) to manage memory. Every instance of a class or closure maintains a reference count which keeps a running tally of the number of references the program is holding for on to. When this count reaches 0 the instance is deallocated. This automatic deallocation removes the need for a garbage collector as instances are deallocated as soon as they are no longer needed.

A *strong reference cycle* can occur if two instances each strongly reference each other (e.g. A references B, B references A). Since neither instances reference count can ever reach zero neither is ever deallocated, resulting in a memory leak. Swift provides the keywords `weak` and `unowned` to prevent strong reference cycles. These keywords allow an instance to be referenced without incrementing its reference count. `weak` references must be optional variables, since they can change and become `nil`.[98] Attempting to access an `unowned` value that has already been deallocated results in a runtime error.

A closure within a class can also create a strong reference cycle by capturing self references. Self references to be treated as weak or unowned can be indicated using a *capture list*.

```swift
class Person {
    let name: String
    weak var home: Home? // Defined as a weak reference in order to break the reference cycle. weak references do not increment the reference count of the instance that they refer to.

    init(name: String) {
        self.name = name
    }

    deinit { print("De-initialized \(name)") }
}

class Home {
    let address: String
    var owner: Person?

    init(address: String, owner: Person?) {
        self.address = address
        self.owner = owner
    }

    deinit { print("De-initialized \(address)") }
}

var stacy: Person? = Person(name: "Stacy")
var house21b: Home? = Home(address: "21b Baker Street", owner: stacy)

stacy?.home = house21b // stacy and house42b now refer to each other.

stacy = nil // The reference count for stacy is now 1, because house21b is still holding a reference to it.
house21b = nil // house21b's reference count drops to 0, which in turn drops stacy's count to 0 because house21b was the last instance holding a strong reference to stacy.

// Prints:
// De-initialized 21b Baker Street
// De-initialized Stacy
```

## Debugging

A key element of the Swift system is its ability to be cleanly debugged and run within the development environment, using a read–eval–print loop (REPL), giving it interactive properties more in common with the scripting abilities of Python than traditional system programming languages. The REPL is further enhanced with playgrounds, interactive views running within the Xcode environment or Playgrounds app that respond to code or debugger changes on-the-fly.[99] Playgrounds allow programmers to add in Swift code along with markdown documentation. Programmers can step through code and add breakpoints using LLDB either in a console or an IDE like Xcode.

# Comparisons to other languages

Swift is considered a C family programming language and is similar to C in various ways:

- Most operators in C also appear in Swift, although some operators such as + have slightly different behavior. For example, in Swift, + traps on overflow, whereas &+ is used to denote the C-like behavior of wrapping on overflow.
- Curly braces are used to group statements.
- Variables are assigned using an equals sign, but compared using two consecutive equals signs. A new identity operator, ===, is provided to check if two data elements refer to the same object.
- Control statements `while`, `if`, and `switch` are similar, but have extended functions, e.g., a `switch` that takes non-integer cases, `while` and `if` supporting pattern matching and conditionally unwrapping optionals, `for` uses the `for i in 1...10` syntax.
- Square brackets are used with arrays, both to declare them and to get a value at a given index in one of them.

It also has similarities to Objective-C:

- Basic numeric types: `Int, UInt, Float, Double`
- Class methods are inherited, like instance methods; `self` in class methods is the class the method was called on.
- Similar `for...in` enumeration syntax.

Differences from Objective-C include:

- Statements need not end with semicolons (`;`), though these must be used to allow more than one statement on one line.
- No header files.
- Uses type inference.
- Generic programming.
- Functions are first-class objects.
- Enumeration cases can have associated data (algebraic data types).
- Operators can be redefined for classes (operator overloading), and new operators can be defined.
- Strings fully support Unicode. Most Unicode characters can be used in either identifiers or operators.
- No exception handling. Swift 2 introduces a different and incompatible error-handling model.[100]
- Several features of earlier C-family languages that are easy to misuse have been removed:
  - Pointers are not exposed by default. There is no need for the programmer to keep track of and mark names for referencing or dereferencing.
  - Assignments return no value. This prevents the common error of writing `i = 0` instead of `i == 0` by throwing a compile-time error.
  - No need to use `break` statements in `switch` blocks. Individual cases do not fall through to the next case unless the `fallthrough` statement is used.
  - Variables and constants are always initialized and array bounds are always checked.
  - Integer overflows, which result in undefined behavior for signed integers in C, are trapped as a run-time error in Swift. Programmers can choose to allow overflows by using the special arithmetical operators &+, &−, &∗, &/ and &%. The properties `min` and `max` are defined in Swift for all integer types and can be used to safely check for potential overflows, versus relying on constants defined for each type in external libraries.
  - The one-statement form of `if` and `while`, which allows for the omission of braces around the statement, is unsupported.
  - C-style enumeration `for (int i = 0; i < c; i++)`, which is prone to off-by-one errors, is

unsupported (from Swift 3 onward).[101]

- The pre- and post- increment and decrement operators (`i++`, `--i` ...) are unsupported (from Swift 3 onward), more so since C-style `for` statements are also unsupported from Swift 3 onward.[102]

# Development and other implementations

Because Swift can run on Linux, it is sometimes also used as a server-side language.[103] Some web frameworks have already been developed, such as IBM's Kitura (now discontinued), Perfect and Vapor.

An official "Server APIs" work group has also been started by Apple,[104] with members of the Swift developer community playing a central role.[105]

A second free implementation of Swift that targets Cocoa, Microsoft's Common Language Infrastructure (.NET Framework, now .NET), and the Java and Android platform exists as part of the *Elements Compiler* from RemObjects Software.[106]

Subsets of Swift have been ported to additional platforms, such as Arduino[107] and Mac OS 9.[108]

# See also

- Comparison of programming languages
- Objective-C
- D (programming language)
- Kotlin (programming language)
- Python (programming language)
- Nim (programming language)


# References

1. U.S. patent no. 9329844
2. "Swift Has Reached 1.0" (https://developer.apple.com/swift/blog/?id=14). Apple. September 9, 2014. Retrieved March 8, 2015.
3. Error: Unable to display the reference properly. See the documentation for details.
4. "Swift, Objectively" (https://www.drdobbs.com/architecture-and-design/swift-objectively/240168424). "Swift is proprietary and closed: It is entirely controlled by Apple and there is no open source implementation."
5. Lattner, Chris (June 11, 2014). "Re: [LLVMdev] [cfe-dev] [Advertisement] open positions in Apple's Swift compiler team" (https://web.archive.org/web/20140714201921/http://lists.cs.uiuc.edu/pipermail/llvmdev/2014-June/073698.html). Archived from the original (http://lists.cs.uiuc.edu/pipermail/llvmdev/2014-June/073698.html) on July 14, 2014. Retrieved June 12, 2014. "You can imagine that many of us want it to be open source and part of LLVM, but the discussion hasn't happened yet, and won't for some time."
6. "Chris Lattner's Homepage" (http://nondot.org/sabre/). Chris Lattner. June 3, 2014. Retrieved June 3, 2014. "The Swift language is the product of tireless effort from a team of language experts, documentation gurus, compiler optimization ninjas, and an incredibly important internal dogfooding group who provided feedback to help refine and battle-test ideas. Of course, it also greatly benefited from the experiences hard-won by many other languages in the field, drawing ideas from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list."

7. Lattner, Chris (June 3, 2014). "Chris Lattner's Homepage" (http://nondot.org/sabre). Chris Lattner. Retrieved June 3, 2014. "I started work on the Swift Programming Language in July of 2010. I implemented much of the basic language structure, with only a few people knowing of its existence. A few other (amazing) people started contributing in earnest late in 2011, and it became a major focus for the Apple Developer Tools group in July 2013 [...] drawing ideas from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list."
8. "Building assert() in Swift, Part 2: __FILE__ and __LINE__" (https://developer.apple.com/swift/blog/?id=15). Retrieved September 25, 2014.
9. "Influences - The Rust Reference" (https://doc.rust-lang.org/reference/influences.html). *doc.rust-lang.org*. Retrieved May 2, 2020.
10. "influenced by - V documentation" (https://github.com/vlang/v/blob/master/doc/docs.md#introduction). *github.com*. Retrieved November 3, 2023.
11. Lardinois, Frederic (June 2, 2014). "Apple Launches Swift, A New Programming Language For Writing iOS And OS X Apps" (https://techcrunch.com/2014/06/02/apple-launches-swift-a-new-programming-language-for-writing-ios-and-os-x-apps/). *TechCrunch*. Retrieved September 7, 2022.
12. *Protocol-oriented Programming in Swift* (https://www.youtube.com/watch?v=g2LwFZatfTI). *Apple Inc*. YouTube.
13. "Concepts are similar to Rust Traits" (https://news.ycombinator.com/item?id=13225740).
14. Williams, Owen (June 2, 2014). "Tim Berners-Lee's sixtieth birthday Apple announces Swift, a new programming language for iOS" (https://thenextweb.com/apple/2014/06/02/apple-announces-swift-new-programming-language-ios). *The Next Web*. Retrieved June 2, 2014.
15. "Apple's new programming language Swift is now open source" (https://www.theverge.com/2015/12/3/9842854/apple-swift-open-source-released). *The Verge*. December 3, 2015. Retrieved December 5, 2015.
16. "Apple Open Sources Swift in Latest Pitch to the Enterprise" (https://blogs.wsj.com/cio/2015/12/03/apple-open-sources-swift-in-latest-pitch-to-the-enterprise/). CIO Journal. *The Wall Street Journal Blogs*. December 3, 2015. Retrieved December 5, 2015.
17. "Looking back on Swift 3 and ahead to Swift 4" (https://forums.swift.org/t/looking-back-on-swift-3-and-ahead-to-swift-4/3610). *Swift Forums*. July 29, 2016. Retrieved November 19, 2018.
18. "Swift-Evolution" (https://github.com/apple/swift-evolution#source-stability). *Swift Evolution*. Retrieved November 19, 2018.
19. "The RedMonk Programming Language Rankings: January 2018 – tecosystems" (https://redmonk.com/sogrady/2018/03/07/language-rankings-1-18/). *redmonk.com*. March 7, 2018. Retrieved November 20, 2018.
20. Kremenek, Ted (March 25, 2019). "Swift 5 Released!" (https://swift.org/blog/swift-5-released/).
21. Kremenek, Ted (September 20, 2019). "Swift 5.1 Released!" (https://web.archive.org/web/20220226203243/https://www.swift.org/blog/swift-5-1-released/). Archived from the original (https://swift.org/blog/swift-5-1-released/) on February 26, 2022. Retrieved October 28, 2019.
22. Hudson, Paul (June 6, 2021). "What's new in Swift 5.5?" (https://www.hackingwithswift.com/articles/233/whats-new-in-swift-5-5). *HackingWithSwift.com*. Hacking with Swift. Retrieved June 8, 2021.
23. "Swift 5.9 Released" (https://swift.org/blog/swift-5.9-released/). *Swift.org*. September 18, 2023. Retrieved October 9, 2023.
24. Borla, Holly (March 5, 2024). "Swift 5.10 Released" (https://swift.org/blog/swift-5.10-released/). *Swift.org*. Retrieved March 13, 2024.
25. "Swift.org" (https://swift.org/). *Swift.org*. March 2014. Retrieved April 28, 2024.
26. Platforms State of the Union, Session 102, Apple Worldwide Developers Conference, June 2, 2014
27. *The Swift Programming Language* (https://itunes.apple.com/book/swift-programming-language/id881256329?mt=11). Apple. June 2, 2014. Retrieved June 2, 2014.
    - "Documentation" (https://swift.org/documentation/). *Swift*.
28. "Swift Has Reached 1.0" (https://developer.apple.com/swift/blog/?id=14). September 9, 2014. Retrieved September 10, 2014.

29. "Xcode 6.1 Release Notes" (https://developer.apple.com/library/ios/releasenotes/DeveloperTools/RN-Xcode/Chapters/xc6_release_notes.html). October 22, 2014. Retrieved January 23, 2015.
30. "Xcode 6.3 Release Notes" (https://developer.apple.com/library/ios/releasenotes/DeveloperTools/RN-Xcode/Chapters/xc6_release_notes.html). April 8, 2015. Retrieved April 8, 2015.
31. "Swift 2 Apps in the App Store" (https://developer.apple.com/swift/blog/?id=32). *Swift Blog*. Retrieved March 13, 2016.
32. "Swift 3.0 Released!" (https://web.archive.org/web/20161014162805/https://swift.org/blog/swift-3-0-released/). *Swift.org*. September 13, 2016. Archived from the original (https://swift.org/blog/swift-3-0-released/) on October 14, 2016. Retrieved October 26, 2016.
33. "Swift 4.0 Released!" (https://web.archive.org/web/20190328104443/https://swift.org/blog/swift-4-0-released/). *Swift.org*. September 17, 2017. Archived from the original (https://swift.org/blog/swift-4-0-released/) on March 28, 2019. Retrieved March 1, 2018.
34. "Swift 4.1 Released!" (https://web.archive.org/web/20190425153237/https://swift.org/blog/swift-4-1-released/). *Swift.org*. March 29, 2018. Archived from the original (https://swift.org/blog/swift-4-1-released/) on April 25, 2019. Retrieved March 30, 2018.
35. "Stack Overflow Developer Survey Results 2015" (https://stackoverflow.com/research/developer-survey-2015#tech-super).
36. "Stack Overflow Developer Survey Results 2016" (https://stackoverflow.com/research/developer-survey-2016#technology-most-loved-dreaded-and-wanted).
37. "Swift.org and Open Source" (https://swift.org/about/#swiftorg-and-open-source). *Swift.org*. Apple Inc. Retrieved February 25, 2019.
38. "Introducing the IBM Swift Sandbox — Swift" (https://developer.ibm.com/swift/2015/12/03/introducing-the-ibm-swift-sandbox/). *Swift*. Retrieved December 5, 2015.
39. Mayo, Benjamin (December 4, 2015). "Write Swift code in a web browser with the IBM Swift Sandbox" (https://9to5mac.com/2015/12/04/swift-web-browser-code-ibm-sandbox/). *9to5Mac*. Retrieved December 5, 2015.
40. "After Apple open sources it, IBM puts Swift programming in the cloud | ZDNet" (https://www.zdnet.com/article/after-apple-open-sources-it-ibm-puts-swift-in-the-cloud/). *ZDNet*. Retrieved December 5, 2015.
41. "Swift Package Catalog and Swift Sandbox Deprecation" (https://developer.ibm.com/swift/2017/12/07/package-catalog-sandbox-deprecation/). Retrieved November 9, 2018.
42. "Swift Playgrounds" (https://developer.apple.com/swift/playgrounds/). *Apple Developer*. Retrieved June 19, 2016.
43. "Swift Playgrounds — Preview" (https://www.apple.com/swift/playgrounds/). *Apple*. Retrieved June 19, 2016.
44. Mayo, Benjamin (June 13, 2016). "Apple announces Swift Playgrounds for iPad at WWDC, public release in fall" (https://9to5mac.com/2016/06/13/apple-announces-swift-playgrounds-for-ipad/). *9to5Mac*. Retrieved June 19, 2016.
45. Cunningham, Andrew (January 10, 2017). "Longtime Apple programmer and Swift creator leaves Apple for Tesla" (https://arstechnica.com/apple/2017/01/longtime-apple-programmer-and-swift-creator-leaves-apple-for-tesla). Ars Technica.
46. Wuerthele, Mike (January 13, 2017). "New Swift project head Ted Kremenek said to be running the show behind the scenes for some time" (https://appleinsider.com/articles/17/01/13/new-swift-project-head-ted-kremenek-said-to-be-running-the-show-behind-the-scenes-for-some-time). AppleInsider.
47. Daniel Eran Dilger. "WWDC19: SwiftUI was the brightest star in a galaxy of new ideas" (https://appleinsider.com/articles/19/06/19/wwdc19-swiftui-was-the-brightest-star-in-a-galaxy-of-new-ideas). *AppleInsider*. Retrieved July 19, 2019.
48. "Swift.org - Download Swift" (https://swift.org/download/#releases). Retrieved June 21, 2020.
49. "Android SDKs for Swift" (https://github.com/buttaface/swift-android-sdk). *GitHub*. Retrieved September 10, 2021.
50. "swift-lang package versions" (https://repology.org/project/swift-lang/versions). Retrieved September 10, 2021.
51. Readdle (January 15, 2020). "Swift for Android: Our Experience and Tools" (https://blog.readdle.com/why-we-use-swift-for-android-db449feeacaf). *Medium*. Retrieved August 20, 2020.

52. Anderson, Tim (March 30, 2020). "Official tailored Swift for Windows support promised in 5.3: Swift on more platforms – provided you do not need a GUI" (https://www.theregister.com/2020/03/30/official_swift_programming_for_windows/). *The Register*. Retrieved September 18, 2020.
53. "The Swift Linux Port" (https://swift.org/blog/swift-linux-port/). *Swift.org*. Apple Inc. December 3, 2015. Retrieved August 3, 2016.
54. Timmer, John (June 5, 2014). "A fast look at Swift, Apple's new programming language" (https://arstechnica.com/apple/2014/06/a-fast-look-at-swift-apples-new-programming-language/). *Ars Technica*. Condé Nast. Retrieved June 6, 2014.
55. Kremenek, Ted (March 25, 2019). "Swift 5 Released!" (https://swift.org/blog/swift-5-released/). *Swift.org*. Retrieved March 28, 2019.
56. "Download Swift" (https://swift.org/download/#releases). *Swift.org*. Apple. Retrieved December 15, 2020.
57. Kremenek, Ted (April 26, 2021). "Swift 5.4 Released!" (https://web.archive.org/web/20210426183534/https://swift.org/blog/swift-5-4-released/). *Swift.org*. Apple. Archived from the original (https://swift.org/blog/swift-5-4-released/) on April 26, 2021. Retrieved April 26, 2021.
58. Kremenek, Ted (March 14, 2022). "Swift 5.6 Released!" (https://www.swift.org/blog/swift-5.6-released/). *Swift.org*. Apple. Retrieved March 14, 2022.
59. "Release Swift 5.6.1 Release · apple/Swift" (https://github.com/apple/swift/releases/tag/swift-5.6.1-RELEASE). *GitHub*.
60. "Release Swift 5.6.2 Release · apple/Swift" (https://github.com/apple/swift/releases/tag/swift-5.6.2-RELEASE). *GitHub*.
61. "Release Swift 5.6.3 Release · apple/Swift" (https://github.com/apple/swift/releases/tag/swift-5.6.3-RELEASE). *GitHub*.
62. Borla, Holly (September 12, 2022). "Swift 5.7 Released!" (https://www.swift.org/blog/swift-5.7-released/). *Swift.org*. Apple. Retrieved September 23, 2022.
63. "Release Swift 5.7.1 Release · apple/Swift" (https://github.com/apple/swift/releases/tag/swift-5.7.1-RELEASE). *GitHub*.
64. "Release Swift 5.8 Release · apple/Swift" (https://github.com/apple/swift/releases/tag/swift-5.8-RELEASE). *GitHub*.
65. "Release Swift 5.8.1 Release · apple/swift" (https://github.com/apple/swift/releases/tag/swift-5.8.1-RELEASE). *GitHub*. Retrieved June 14, 2023.
66. "Release Swift 5.9 Release · apple/swift" (https://github.com/apple/swift/releases/tag/swift-5.9-RELEASE). *GitHub*. Retrieved September 18, 2023.
67. "Release Swift 5.9.1 Release · apple/swift" (https://github.com/apple/swift/releases/tag/swift-5.9.1-RELEASE). *GitHub*. Retrieved October 19, 2023.
68. "Release Swift 5.9.2 Release · apple/swift" (https://github.com/apple/swift/releases/tag/swift-5.9.2-RELEASE). *GitHub*. Retrieved December 11, 2023.
69. "Documentation" (https://docs.swift.org/swift-book/documentation/the-swift-programming-language/aboutswift). *docs.swift.org*. Retrieved November 17, 2023.
70. "Eliminate data races using Swift Concurrency - WWDC22 - Videos" (https://developer.apple.com/wwdc22/110351). Apple Inc. Retrieved November 17, 2023.
71. "Documentation" (https://docs.swift.org/swift-book/documentation/the-swift-programming-language/guidedtour/). *docs.swift.org*. Retrieved October 15, 2023.
72. "Documentation" (https://docs.swift.org/swift-book/documentation/the-swift-programming-language/attributes/#main). *docs.swift.org*. Retrieved October 15, 2023.
73. "Closures — The Swift Programming Language (Swift 5.5)" (https://docs.swift.org/swift-book/LanguageGuide/Closures.html). *docs.swift.org*. Retrieved August 31, 2021.
74. Macomber, Kyle; Yaskevich, Yavel; Gregor, Doug; McCall, John. "Multiple Trailing Closures" (https://github.com/apple/swift-evolution/blob/b394ae8fff585c8fdc27a50422ea8a90f13138d2/proposals/0279-multiple-trailing-closures.md). *GitHub*. Retrieved October 19, 2020.
75. "Documentation" (https://docs.swift.org/swift-book/documentation/the-swift-programming-language/closures/). *docs.swift.org*. Retrieved October 16, 2023.

76. "Strings and Characters" (https://docs.swift.org/swift-book/documentation/the-swift-programming-language/stringsandcharacters/). *docs.swift.org*. Retrieved October 16, 2023.
77. "Declarations — The Swift Programming Language (Swift 5.6)" (https://docs.swift.org/swift-book/ReferenceManual/Declarations.html). *docs.swift.org*. Retrieved February 28, 2022.
78. "Access Control" (https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/AccessControl.html). *The Swift Programming Language*. Apple Developer. Retrieved October 25, 2016.
79. "Nullable Types" (https://msdn.microsoft.com/en-us/library/1t3y8s4s.aspx), C# Programming Guide, Microsoft. Archived (https://web.archive.org/web/20170221214236/https://msdn.microsoft.com/en-us/library/1t3y8s4s.aspx) February 21, 2017, at the Wayback Machine.
80. "Types" (https://developer.apple.com/library/prerelease/ios/documentation/swift/conceptual/swift_programming_language/Types.html). *The Swift Programming Language*. Apple Developer. Retrieved July 16, 2014.
81. "Classes and Structures" (https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ClassesAndStructures.html). *The Swift Programming Language*. Apple Developer. Archived (https://web.archive.org/web/20160325202335/https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ClassesAndStructures.html) from the original on March 25, 2016.
82. Guhit, Fiel (February 14, 2015). "Performance Case Study on Swift 1.1, Swift 1.2, and Objective-C" (https://medium.com/@fielgood/swift-1-1-swift-1-2-and-objective-c-a-performance-case-study-d86f7a333e2a). *Medium*. Archived (https://web.archive.org/web/20231212084350/https://medium.com/@fielgood/swift-1-1-swift-1-2-and-objective-c-a-performance-case-study-d86f7a333e2a) from the original on December 12, 2023.
83. *Building Better Apps with Value Types* (https://developer.apple.com/videos/wwdc/2015/?id=414). Apple.
84. "Extensions" (https://docs.swift.org/swift-book/documentation/the-swift-programming-language/extensions/). *docs.swift.org*. Retrieved November 28, 2023.
85. "Working with Protocols" (https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/WorkingwithProtocols/WorkingwithProtocols.html). *Programming with Objective-C*. Apple Developer Documentation Archive. September 17, 2014. Archived (https://web.archive.org/web/20160429195614/https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/WorkingwithProtocols/WorkingwithProtocols.html) from the original on April 29, 2016.
86. "NSCopying Protocol Reference" (https://developer.apple.com/library/prerelease/ios/documentation/Cocoa/Reference/Foundation/Protocols/NSCopying_Protocol/index.html). *Apple*.
87. "Swift standard library statistics" (https://swiftinit.org/stats/swift/swift#ss:interface-breakdown). *swiftinit.org*. Swiftinit. Retrieved October 2, 2023.
88. Thompson, Mattt (September 2, 2014). "Swift Default Protocol Implementations" (https://nshipster.com/swift-default-protocol-implementations/). *NSHipster*.
89. "swift-evolution/proposals/0304-structured-concurrency.md at main · apple/swift-evolution" (https://github.com/apple/swift-evolution/blob/main/proposals/0304-structured-concurrency.md). *GitHub*. Retrieved October 16, 2023.
90. "swift-evolution/proposals/0296-async-await.md at main · apple/swift-evolution" (https://github.com/apple/swift-evolution/blob/main/proposals/0296-async-await.md#motivation-completion-handlers-are-suboptimal). *GitHub*. Retrieved October 16, 2023.
91. "Do Swift-based apps work on macOS 10.9/iOS 7 and lower?" (https://stackoverflow.com/questions/24001778/do-swift-based-apps-work-on-os-x-10-9-ios-7-and-lower/24038997#24038997), StackOverflow
92. Inc, Apple (September 18, 2023). "Swift 5.9 Released" (https://swift.org/blog/swift-5.9-released/). *Swift.org*. Retrieved October 9, 2023. {{cite web}}: |last= has generic name (help)
93. "Writing Swift Classes with Objective-C Behavior" (https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/BuildingCocoaApps/WritingSwiftClassesWithObjective-CBehavior.html), Apple Inc.
94. "Migrating Your Objective-C Code to Swift" (https://developer.apple.com/library/ios/documentation/Swift/Conceptual/BuildingCocoaApps/Migration.html).
95. "Swift and Objective-C in the Same Project" (https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/BuildingCocoaApps/MixandMatch.html#//apple_ref/doc/uid/TP40014216-CH10-XID_77), Apple Inc.

96. "Apple: search "corelib" " (https://github.com/apple?utf8=%E2%9C%93&q=corelibs&type=&language=). *GitHub*.
97. "Xcode - SwiftUI- Apple Developer" (https://developer.apple.com/xcode/swiftui/). *developer.apple.com*. Retrieved February 1, 2021.
98. Lanier, Brian; Groff, Joe. "Intermediate Swift" (https://developer.apple.com/videos/wwdc/2014/?include=403#403). Apple. Retrieved July 3, 2014.
99. Metz, Cade. "Why Coders Are Going Nuts Over Apple's New Programming Language" (https://www.wired.com/2014/06/apple-swift-language/). *Wired*. Retrieved July 16, 2014.
100. "Error-Handling in Swift-Language" (https://stackoverflow.com/a/26749528). *stackoverflow.com*.
101. "apple/swift-evolution" (https://github.com/apple/swift-evolution/blob/master/proposals/0007-remove-c-style-for-loops.md). *GitHub*. Retrieved April 4, 2016.
102. "apple/swift-evolution" (https://github.com/apple/swift-evolution/blob/master/proposals/0004-remove-pre-post-inc-decrement.md). *GitHub*. Retrieved April 4, 2016.
103. Barbosa, Greg (February 22, 2016). "IBM brings Swift to the cloud, releases web framework Kitura written in Apple's programming language" (https://9to5mac.com/2016/02/22/ibm-swift-cloud-kitura/). *9to5Mac*. Retrieved May 16, 2016.
104. "Server APIs Work Group" (https://swift.org/blog/server-api-workgroup/). *Swift.org*. October 25, 2016. Retrieved October 28, 2016.
105. "Swift.org" (https://web.archive.org/web/20210510193318/https://swift.org/server-apis/). *Swift.org*. Archived from the original (https://swift.org/server-apis/) on May 10, 2021. Retrieved October 28, 2016.
106. "RemObjects Elements Compiler" (https://www.elementscompiler.com/silver). Retrieved January 17, 2016.
107. "Swift for Arduino" (https://www.swiftforarduino.com/).
108. Rose, Jordan (April 1, 2020). "Swift on Mac OS 9" (https://belkadan.com/blog/2020/04/Swift-on-Mac-OS-9/). *-dealloc*.

## External links

- Official website (https://swift.org/)
- Swift (https://developer.apple.com/swift/) at Apple Developer
- Swift source code (https://github.com/apple/swift) on GitHub
- Swift Example (https://iosexamples.com/)
- Server-side Swift: The Vapor Framework (https://vapor.codes/)