WIKIPEDIA

Swift (programming language)

Swift is a general-purpose, multi-paradigm, compiled programming language developed by Apple Inc. and the open-source community. First released in 2014, Swift was developed as a replacement for Apple's earlier programming language Objective-C, as Objective-C had been largely unchanged since the early 1980s and lacked modern language features. Swift works with Apple's Cocoa and Cocoa Touch frameworks, and a key aspect of Swift's design was the ability to interoperate with the huge body of existing Objective-C code developed for Apple products over the previous decades. It is built with the open source LLVM compiler framework and has been included in Xcode since version 6, released in 2014. On Apple platforms,[11] it uses the Objective-C runtime library, which allows C, Objective-C, C++ and Swift code to run within one program.[12]

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"[13] (similar to traits).[14]

Swift was introduced at Apple's 2014 Worldwide Developers Conference (WWDC).[15] 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.[16][17]

Through version 3.0 the syntax of Swift went through significant evolution, with the core team making source stability a focus in later versions.[18][19] In the first quarter of 2018 Swift surpassed Objective-C in measured popularity.[20]

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.[21]

Swift

Swift

Logo

Paradigm	Multi-paradigm: protocol-oriented, object-oriented, functional, imperative, block structured, declarative
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	5.5.1[3] / 26 October 2021
Preview release	5.5 branch[4]
Typing discipline	Static, strong, inferred
OS	Apple's operating systems (Darwin, iOS, iPadOS, macOS, tvOS, watchOS), Linux, Windows 10, Android
License	Apache License 2.0 (Swift 2.2 and later) Proprietary (up to Swift 2.2)[5][6]
Filename	.swift, .SWIFT

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.[22]

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.[23]

extensions	
Website	swift.org (https://swift.org)
Influenced by	
Objective-C,[7] Rust, Haskell, Ruby, Python, C#, CLU,[8] D[9]	
Influenced	
Rust[10]	

Contents

History

Development of Swift started in July 2010 by Chris Lattner, with the eventual collaboration of many other programmers at Apple. Swift took language ideas "from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list".[8] On June 2, 2014, the Apple Worldwide Developers Conference (WWDC) application became the first publicly released app written with Swift.[24] 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.[24]

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.[25]

Swift reached the 1.0 milestone on September 9, 2014, with the *Gold Master* of Xcode 6.0 for iOS.[26] Swift 1.1 was released on October 22, 2014, alongside the launch of Xcode 6.1.[27] Swift 1.2 was released on April 8, 2015, along with Xcode 6.3.[28] Swift 2.0 was announced at WWDC 2015, and was made available for publishing apps in the App Store in September 21, 2015.[29] Swift 3.0 was released on September 13, 2016.[30] Swift 4.0 was released on September 19, 2017.[31] Swift 4.1 was released on March 29, 2018.[32]

Swift won first place for *Most Loved Programming Language* in the Stack Overflow Developer Survey 2015[33] and second place in 2016.[34]

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,[35] 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.[36][37][38] The Swift Sandbox was deprecated in January 2018.[39]

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.[40][41][42]

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.[43][44]

During WWDC 2019, Apple announced SwiftUI with Xcode 11, which provides a framework for declarative UI structure design across all Apple platforms.[45]

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.[46] There is an unofficial SDK and native toolchain package for Android too.[47][48]

Platforms

The platforms Swift supports are Apple's operating systems (Darwin, iOS, iPadOS, macOS, tvOS, watchOS), Linux, Windows, and Android.[49][50]

Version history

Version	Release Date	macOS	Linux	Windows
Swift 1.0	September 9, 2014	Yes	No	No
Swift 1.1	October 22, 2014	Yes	No	No
Swift 1.2	April 8, 2015	Yes	No	No
Swift 2.0	September 21, 2015	Yes	No	No
Swift 2.1	October 20, 2015	Yes	No	No
Swift 2.2	March 21, 2016	Yes	Yes	No
Swift 2.2.1	May 3, 2016	Yes	Yes	No
Swift 3.0	September 13, 2016	Yes	Yes	No
Swift 3.0.1	October 28, 2016	Yes	Yes	No
Swift 3.0.2	December 13, 2016	Yes	Yes	No
Swift 3.1	March 27, 2017	Yes	Yes	No
Swift 3.1.1	April 21, 2017	Yes	Yes	No
Swift 4.0	September 19, 2017	Yes	Yes	No
Swift 4.0.2	November 1, 2017	Yes	Yes	No
Swift 4.0.3	December 5, 2017	Yes	Yes	No
Swift 4.1	March 29, 2018	Yes	Yes	No
Swift 4.1.1	May 4, 2018	No	Yes	No
Swift 4.1.2	May 31, 2018	Yes	Yes	No
Swift 4.1.3	July 27, 2018	No	Yes	No
Swift 4.2	September 17, 2018	Yes	Yes	No
Swift 4.2.1	October 30, 2018	Yes	Yes	No
Swift 4.2.2	February 4, 2019	No	Yes	No
Swift 4.2.3	February 28, 2019	No	Yes	No
Swift 4.2.4	March 29, 2019	No	Yes	No
Swift 5.0[51]	March 25, 2019	Yes	Yes	No
Swift 5.0.1	April 18, 2019	Yes	Yes	No
Swift 5.0.2	July 15, 2019	No	Yes	No
Swift 5.0.3	August 30, 2019	No	Yes	No
Swift 5.1	September 10, 2019	Yes	Yes	No
Swift 5.1.1	October 11, 2019	No	Yes	No
Swift 5.1.2	November 7, 2019	Yes	Yes	No
Swift 5.1.3	December 13, 2019	Yes	Yes	No
Swift 5.1.4	January 31, 2020	No	Yes	No
Swift 5.1.5	March 9, 2020	No	Yes	No
Swift 5.2	March 24, 2020	Yes	Yes	No
Swift 5.2.1	March 30, 2020	No	Yes	No
Swift 5.2.2	April 15, 2020	Yes	Yes	No
Swift 5.2.3	April 29, 2020	No	Yes	No
Swift 5.2.4	May 20, 2020	Yes	Yes	No

Version	Release Date	macOS	Linux	Windows
Swift 5.2.5	August 5, 2020	No	Yes	No
Swift 5.3	September 16, 2020	Yes	Yes	Yes[52]
Swift 5.3.1	November 13, 2020	Yes	Yes	Yes
Swift 5.3.2	December 15, 2020	Yes	Yes	Yes
Swift 5.3.3	January 25, 2021	No	Yes	Yes
Swift 5.4[53]	April 26, 2021	Yes	Yes	Yes
Swift 5.4.1	May 25, 2021	No	Yes	Yes
Swift 5.4.2	June 28, 2021	Yes	Yes	Yes
Swift 5.4.3	September 9, 2021	No	Yes	Yes
Swift 5.5	September 20, 2021	Yes	Yes	Yes

Features

Swift is an alternative to the Objective-C language that employs modern programming-language theory concepts and strives to present a simpler syntax. During its introduction, it was described simply as "Objective-C without the baggage of C".[54][55]

By default, Swift does not expose pointers and other unsafe accessors, in contrast to Objective-C, which uses pointers pervasively to refer to object instances. Also, Objective-C's use of a Smalltalk-like syntax for making method calls has been replaced with a dot-notation style and namespace system more familiar to programmers from other common object-oriented (OO) languages like Java or C#. Swift introduces true named parameters and retains key Objective-C concepts, including protocols, closures and categories, often replacing former syntax with cleaner versions and allowing these concepts to be applied to other language structures, like enumerated types (enums).[56]

Closure support

Swift supports closures (known as lambdas in other languages). Closures are self-contained blocks of functionality that can be passed around and used in your code.[57] Closures can be thought of as an unnamed function. Here is an example:

```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.
// However, this functionality may not work for too complex expressions.
let closure3 = { arg1: Int, arg2: Int in
    return arg1 + arg2
}
```

Swift has a trailing closure syntax like this:

```
// 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:
foo { return 1 }
```

Starting from version 5.3, Swift supports multiple trailing closures:[58]

```
// 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:
a(bar: { return 1 }) { x in return x + 1 })

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

Here are the criteria for the trailing closure syntax:

- If the last arguments of a function are closures you can use the trailing closure syntax.
- The parameter name of the first trailing closure must be omitted.
- The parameter names of the remaining trailing closures must not be omitted.
- If all the arguments given to a function are trailing closures, you may omit the parentheses after the function's name.
- Calls to a function with trailing closures must be parenthesized if used in a `guard` statement.[59]

String support

Under the Cocoa and Cocoa Touch environments, many common classes were part of the Foundation Kit library. This included the NSString string library (using Unicode, UTF-8 in Swift 5, changed from UTF-16), the NSArray and NSDictionary collection classes, and others. Objective-C provided various bits of syntactic sugar to allow some of these objects to be created on-the-fly within the language, but once created, the objects were manipulated with object calls. For instance, in Objective-C concatenating two NSStrings required method calls similar to this:

```
NSString *str = @"hello,";
str = [str stringByAppendingString:@" world"];
```

In Swift, many of these basic types have been promoted to the language's core, and can be manipulated directly. For instance, strings are invisibly bridged to NSString (when Foundation is imported) and can now be concatenated with the + operator, allowing greatly simplified syntax; the prior example becoming:[60]

```
var str = "hello,"
str += " world"
```

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.[61]

Optionals and chaining

An important new 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 refer to a value or may be null. 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` mechanism—to make an Integer that is nullable, one would use a declaration similar to `var optionalInteger: Optional<Int>`. As in C#,[62] 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?`.[63] 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 latter has more in common with `Int?` than `String`.

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. This can be annoying in practice, so Swift also includes the concept of optional chaining to test 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. Normally this requires the programmer to test whether `myValue` is nil before proceeding. 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
```

The ? syntax circumvents the pyramid of doom.

Swift 2 introduced the new keyword `guard` for cases in which code should stop executing if some condition is unmet:

```
guard let leaseStart = aBuilding.TenantList[5]?.leaseDetails?.startDate else
{
    //handle the error case where anything in the chain is nil
    //else scope must exit the current method or loop
}
//continue, knowing that leaseStart is not nil
```

Using `guard` has three benefits. While the syntax can act as an `if` statement, its primary benefit is inferring non-nullability. Where an `if` statement requires a case, `guard` assumes the case based on the condition provided. Also, since `guard` contains no scope, with exception of the `else` closure, `leaseStart` is presented as an unwrapped optional to the guard's super-scope. Lastly, if the `guard` statement's test fails, Swift requires the `else` to exit the current method or loop, ensuring `leaseStart` never is accessed when `nil`. This is performed with the keywords `return`, `continue`, `break`, or `throw`, or by calling a function returning a `Never` (e.g. `fatalError()`).

Objective-C was weakly typed and allowed any method to be called on any object at any time. If the method call failed, there was a default handler in the runtime that returned nil. That meant that no unwrapping or testing was needed, the equivalent statement in Objective-C:

```
leaseStart = [[[aBuilding tenantList:5] leaseDetails] startDate]
```

Would return nil, and this could be tested. However, this also demanded that all method calls be dynamic, which introduces significant overhead. Swift's use of optionals provides a similar mechanism for testing and dealing with nils, but does so in a way that allows the compiler to use static dispatch because the unwrapping action is called on a defined instance (the wrapper), versus occurring in the 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. Also, pass-by-reference inherently requires a dereferencing operation, which can produce noticeable overhead in some operations, typically those used with these basic value types, like mathematics.

Similarly to C# and in contrast to most other OO languages, 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.[64]

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 dereference. 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`.[64] Using value types can result in significant performance improvements in user applications as well.[65]

To ensure that even the largest structs do not cause a performance penalty when they are handed off, Swift uses copy on write so that the objects 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.[66]

Protocol-oriented programming

A key feature of Objective-C is its support for *categories*, methods that can be added to extend classes at runtime. Categories allow extending classes in-place to add new functions with no need to subclass or even have access to the original source code. An example might be to add spell checker support to the base `NSString` class, which means all instances of NSString in the application gain spell checking. The system is also widely used as an organizational technique, allowing related code to be gathered into library-like extensions. Swift continues to support this concept, although they are now termed *extensions*, and declared with the keyword `extension`. Unlike Objective-C, Swift can also add new properties accessors, types, and enums to extant instances.

Another key feature of Objective-C is its use of *protocols*, known in most modern languages as *interfaces*. Protocols promise that a particular class implements a set of methods, meaning that other objects in the system can call those methods on any object supporting that protocol. This is often used in modern OO languages as a substitute for multiple inheritance, although the feature sets are not entirely similar. A common example of a protocol in Cocoa is the `NSCopying` protocol, which defines one method, `copyWithZone`, that implements deep copying on objects.[67]

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.[68] 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.[67]

For example, a protocol might be declared called `StringConvertible`, which ensures that instances that conform to the protocol implement a `toString` method that returns a `String`. In Swift, this can be declared with code like this:

```
protocol StringConvertible
{
    func toString() -> String
}
```

This protocol can now be added to String, with no access to the base class's source:

```
extension String: StringConvertible
{
    func toString() -> String
    {
        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:

```
var someSortOfPrintableObject: StringConvertible
...
print(someSortOfPrintableObject.toString())
```

It does not matter what sort of instance `someSortOfPrintableObject` 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 = [StringConvertible]`.

As Swift treats structs and classes as similar concepts, both extensions and protocols are extensively used in Swift's runtime to provide a rich API based on structs. For instance, Swift uses an extension to add the `Equatable` protocol to many of their basic types, like Strings and Arrays, allowing them to be compared with the == operator. A concrete example of how all of these features interact can be seen in the concept of *default protocol implementations*:

```
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* function. Any instance, class or struct, automatically gains this implementation simply by conforming to `Equatable`. As many instances gain `Equatable` through their base implementations or other generic extensions, most basic objects in the runtime gain equals and not equals with no code.[69]

This combination of protocols, defaults, protocol inheritance, and extensions allows many of the functions normally associated with classes and inheritance to be implemented on value types.[67] Properly used, this can lead to dramatic performance improvements with no significant limits in API. This concept is so widely used within Swift that Apple has begun calling it a *protocol-oriented programming language*. They suggest addressing many of the problem domains normally solved through classes and inheritance using protocols and structs instead.

Libraries, runtime and 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.[70] Swift and Objective-C code can be used in one program, and by extension, C and C++ also. In contrast to C, C++ code cannot be used directly from Swift. An Objective-C or C wrapper must be created between Swift and C++.[71] 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.[72] The converse is not true: a Swift class cannot be subclassed in Objective-C.[73]

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.[74]

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.[75]

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.[76]

Memory management

Swift uses Automatic Reference Counting (ARC) to manage memory. Apple used to require manual memory management in Objective-C, but introduced ARC in 2011 to allow for easier memory allocation and deallocation.[77] One problem with ARC is the possibility of creating a *strong reference cycle*, where objects reference each other in a way that you can reach the object you started from by following references (e.g. A references B, B references A). This causes them to become leaked into memory as they are never released. Swift provides the keywords `weak` and `unowned` to prevent strong reference cycles. Typically a parent-child relationship would use a strong reference while a child-parent would use either `weak` reference, where parents and children can be unrelated, or `unowned` where a child always has a parent, but parent may not have a child. Weak references must be optional variables, since they can change and become `nil`.[78]

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*.

Debugging and other elements

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 that respond to code or debugger changes on-the-fly.[79] Playgrounds allow programmers to add in Swift code along with markdown documentation. If some code changes over time or with regard to some other ranged input value, the view can be used with the Timeline Assistant to demonstrate the output in an animated way. In addition, Xcode has debugging features for Swift development including breakpoints, step through and step over statements, as well as UI element placement breakdowns for app developers.

Apple says that Swift is "an industrial-quality programming language that's as expressive and enjoyable as a scripting language".[80]

Performance

Many of the features introduced with Swift have well-known performance and safety trade-offs. Apple has implemented optimizations that reduce this overhead.[81]

Comparisons to other languages

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

- Most C operators are used in Swift, but there are some new operators, for example to support integer operations with overflow (see under differences).
- 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 do not need to end with semicolons (`;`), though these must be used to allow more than one statement on a 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.[82]
- 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).[83]
  - 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.[84]

Development and other implementations

Since the language is open-source, there are prospects of it being ported to the web.[85] Some web frameworks have already been developed, such as IBM's Kitura, Perfect and Vapor.

An official "Server APIs" work group has also been started by Apple,[86] with members of the Swift developer community playing a central role.[87]

A second free implementation of Swift that targets Cocoa, Microsoft's Common Language Infrastructure (.NET), and the Java and Android platform exists as part of the *Elements Compiler* from RemObjects Software.[88]

By combining toolchains from LLVM and Macintosh Programmer's Workshop, it is possible to run a very small subset of the language on Mac OS 9.[89]

See also

- Comparison of programming languages
- Objective-C
- 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. https://github.com/apple/swift/releases/tag/swift-5.5.1-RELEASE; retrieved: 28 October 2021.
4. "Swift 5.5 Release Process" (https://forums.swift.org/t/swift-5-5-release-process/45644). March 12, 2021.
5. "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."
6. 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."
7. Lattner, Chris (June 3, 2014). "Chris Lattner's Homepage" (http://nondot.org/sabre/). Chris Lattner. 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."
8. 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."
9. "Building assert() in Swift, Part 2: __FILE__ and __LINE__" (https://developer.apple.com/swift/blog/?id=15). Retrieved September 25, 2014.
10. "Influences - The Rust Reference" (https://doc.rust-lang.org/reference/influences.html). *doc.rust-lang.org*. Retrieved May 2, 2020.
11. "The Swift Linux Port" (https://swift.org/blog/swift-linux-port/). *Swift.org*. Apple Inc. December 3, 2015. Retrieved August 3, 2016.
12. 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.
13. *Protocol-oriented Programming in Swift* (https://www.youtube.com/watch?v=g2LwFZatfTI). *Apple Inc*. YouTube.
14. "Concepts are similar to Rust Traits" (https://news.ycombinator.com/item?id=13225740).
15. 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.
16. "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.
17. "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.
18. "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.
19. "Swift-Evolution" (https://github.com/apple/swift-evolution#source-stability). *Swift Evolution*. Retrieved November 19, 2018.

20. "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.
21. Kremenek, Ted (March 25, 2019). "Swift 5 Released!" (https://swift.org/blog/swift-5-released/).
22. Kremenek, Ted (September 20, 2019). "Swift 5.1 Released!" (https://swift.org/blog/swift-5-1-released/).
23. 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.
24. Platforms State of the Union, Session 102, Apple Worldwide Developers Conference, June 2, 2014
25. *The Swift Programming Language* (https://itunes.apple.com/book/swift-programming-language/id881256329?mt=11). Apple. June 2, 2014. Retrieved June 2, 2014. Lay summary (https://swift.org/documentation/).
26. "Swift Has Reached 1.0" (https://developer.apple.com/swift/blog/?id=14). September 9, 2014. Retrieved September 10, 2014.
27. "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.
28. "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.
29. "Swift 2 Apps in the App Store" (https://developer.apple.com/swift/blog/?id=32). *Swift Blog*. Retrieved March 13, 2016.
30. Inc., Apple (September 13, 2016). "Swift 3.0 Released!" (https://swift.org/blog/swift-3-0-released/). *Swift.org*. Retrieved October 26, 2016.
31. Inc., Apple (September 17, 2017). "Swift 4.0 Released!" (https://swift.org/blog/swift-4-0-released/). *Swift.org*. Retrieved March 1, 2018.
32. Inc., Apple (March 29, 2018). "Swift 4.1 Released!" (https://swift.org/blog/swift-4-1-released/). *Swift.org*. Retrieved March 30, 2018.
33. "Stack Overflow Developer Survey Results 2015" (https://stackoverflow.com/research/developer-survey-2015#tech-super).
34. "Stack Overflow Developer Survey Results 2016" (https://stackoverflow.com/research/developer-survey-2016#technology-most-loved-dreaded-and-wanted).
35. "Swift.org and Open Source" (https://swift.org/about/#swiftorg-and-open-source). *Swift.org*. Apple Inc. Retrieved February 25, 2019.
36. "Introducing the IBM Swift Sandbox — Swift" (https://developer.ibm.com/swift/2015/12/03/introducing-the-ibm-swift-sandbox/). *Swift*. Retrieved December 5, 2015.
37. 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.
38. "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.
39. "Swift Package Catalog and Swift Sandbox Deprecation" (https://developer.ibm.com/swift/2017/12/07/package-catalog-sandbox-deprecation/). Retrieved November 9, 2018.
40. "Swift Playgrounds" (https://developer.apple.com/swift/playgrounds/). *Apple Developer*. Retrieved June 19, 2016.
41. "Swift Playgrounds — Preview" (https://www.apple.com/swift/playgrounds/). *Apple*. Retrieved June 19, 2016.

42. 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.
43. 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.
44. 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.
45. Wednesday, Daniel Eran Dilger; June 19; 2019; PT, 11:26 am. "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.
46. "Swift.org - Download Swift" (https://swift.org/download/#releases). Retrieved June 21, 2020.
47. "Android SDKs for Swift" (https://github.com/buttaface/swift-android-sdk). *GitHub*. Retrieved September 10, 2021.
48. "swift-lang package versions" (https://repology.org/project/swift-lang/versions). Retrieved September 10, 2021.
49. 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.
50. 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.
51. Kremenek, Ted (March 25, 2019). "Swift 5 Released!" (https://swift.org/blog/swift-5-released/). *Swift.org*. Retrieved March 28, 2019.
52. "Download Swift" (https://swift.org/download/#releases). *Swift.org*. Apple. Retrieved December 15, 2020.
53. Kremenek, Ted (April 26, 2021). "Swift 5.4 Released!" (https://swift.org/blog/swift-5-4-released/). *Swift.org*. Apple. Retrieved April 26, 2021.
54. Metz, Rachel (June 3, 2014). "Apple Seeks a Swift Way to Lure More Developers" (https://www.technologyreview.com/news/527821/apple-seeks-a-swift-way-to-lure-more-developers). Technology Review.
55. Weber, Harrison (June 2, 2014). "Apple announces 'Swift,' a new programming language for macOS & iOS" (https://venturebeat.com/2014/06/02/apple-introduces-a-new-programming-language-swift-objective-c-without-the-c/). VentureBeat.
56. "Advantages Of Using Swift" (https://themindstudios.com/blog/advantages-of-using-swift-over-objective-c/). *themindstudios.com*. February 24, 2017. Retrieved February 24, 2017.
57. "Closures — The Swift Programming Language (Swift 5.5)" (https://docs.swift.org/swift-book/LanguageGuide/Closures.html). *docs.swift.org*. Retrieved August 31, 2021.
58. 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.
59. Lattner, Chris. "Allow trailing closures in `guard` conditions" (https://github.com/apple/swift-evolution/blob/main/proposals/0056-trailing-closures-in-guard.md). *GitHub*. Retrieved October 19, 2020.
60. "Strings and Characters" (https://developer.apple.com/library/prerelease/ios/documentation/swift/conceptual/swift_programming_language/StringsAndCharacters.html). *developer.apple.com*. Apple Inc. Retrieved July 16, 2014.

61. "Access Control" (https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/AccessControl.html). *developer.apple.com*. Apple Inc. Retrieved October 25, 2016.
62. "Nullable Types" (https://msdn.microsoft.com/en-us/library/1t3y8s4s.aspx), C# Programming Guide, Microsoft.
63. "Types" (https://developer.apple.com/library/prerelease/ios/documentation/swift/conceptual/swift_programming_language/Types.html). *developer.apple.com*. Apple Inc. Retrieved July 16, 2014.
64. "Classes and Structures" (https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ClassesAndStructures.html). *Apple.com*.
65. 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).
66. *Building Better Apps with Value Types* (https://developer.apple.com/videos/wwdc/2015/?id=414). Apple.
67. "NSCopying Protocol Reference" (https://developer.apple.com/library/prerelease/ios/documentation/Cocoa/Reference/Foundation/Protocols/NSCopying_Protocol/index.html). *Apple*.
68. "Working with Protocols" (https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/WorkingwithProtocols/WorkingwithProtocols.html). *Apple*.
69. Thompson, Mattt (September 2, 2014). "Swift Default Protocol Implementations" (https://nshipster.com/swift-default-protocol-implementations/). *NSHipster*.
70. "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
71. "Using Swift with Cocoa and Objective-C: Basic Setup" (https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/BuildingCocoaApps/index.html). *apple.com*. January 6, 2015.
72. "Writing Swift Classes with Objective-C Behavior" (https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/BuildingCocoaApps/WritingSwiftClassesWithObjective-CBehavior.html), Apple Inc.
73. "Migrating Your Objective-C Code to Swift" (https://developer.apple.com/library/ios/documentation/Swift/Conceptual/BuildingCocoaApps/Migration.html).
74. "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.
75. "Apple: search "corelib" " (https://github.com/apple?utf8=%E2%9C%93&q=corelibs&type=&language=). *GitHub*.
76. "Xcode - SwiftUI- Apple Developer" (https://developer.apple.com/xcode/swiftui/). *developer.apple.com*. Retrieved February 1, 2021.
77. "Automatic Reference Counting" (https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/Swift_Programming_Language/AutomaticReferenceCounting.html), Apple Inc.
78. Lanier, Brian; Groff, Joe. "Intermediate Swift" (https://developer.apple.com/videos/wwdc/2014/?include=403#403). Apple. Retrieved July 3, 2014.
79. 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.
80. About Swift (https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/Swift_Programming_Language/), Apple Inc.
81. "Optimizing Swift Performance" (https://developer.apple.com/videos/play/wwdc2015/409/). Apple, Inc. June 2015.

82. "Error-Handling in Swift-Language" (https://stackoverflow.com/a/26749528). *stackoverflow.com*.
83. "apple/swift-evolution" (https://github.com/apple/swift-evolution/blob/master/proposals/0007-remove-c-style-for-loops.md). *GitHub*. Retrieved April 4, 2016.
84. "apple/swift-evolution" (https://github.com/apple/swift-evolution/blob/master/proposals/0004-remove-pre-post-inc-decrement.md). *GitHub*. Retrieved April 4, 2016.
85. 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.
86. Inc., Apple (October 25, 2016). "Server APIs Work Group" (https://swift.org/blog/server-api-workgroup/). *Swift.org*. Retrieved October 28, 2016.
87. Inc., Apple. "Swift.org" (https://swift.org/server-apis/). *Swift.org*. Retrieved October 28, 2016.
88. "RemObjects Elements Compiler" (https://www.elementscompiler.com/silver). Retrieved January 17, 2016.
89. 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's source code (https://github.com/apple/swift) on GitHub
- Server-side Swift (https://developer.ibm.com/swift/)
- Swift Example (https://iosexample.com/)
- Server-side Swift - The Vapor Framework (https://vapor.codes/)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Swift_(programming_language)&oldid=1057974059"