# The Swift Programming Language

by Apple Inc.

## A Swift Tour

- Simple Values
- Control Flow
- Functions and Closures
- Objects and Classes
- Enumerations and Structures
- Protocols and Extensions
- Generics

## Language Guide

### The Basics

- Constants and Variables
    - Declaring Constants and Variables
    - Type Annotations
    - Naming Constants and Variables
    - Printing Constants and Variables
- Comments
- Semicolons
- Integers
    - Integer Bounds
    - Int
    - UInt
- Floating-Point Numbers
- Type Safety and Type Inference
- Numeric Literals
- Numeric Type Conversion
    - Integer Conversion
    - Integer and Floating-Point Conversion
- Type Aliases
- Booleans
- Tuples
- Optionals
    - nil
    - If Statements and Forced Unwrapping
    - Optional Binding
    - Implicitly Unwrapped Optionals
- Assertions
    - Debugging with Assertions
    - When to Use Assertions

### Basic Operators

- Terminology
- Assignment Operator
- Arithmetic Operators
    - Remainder Operator
    - Floating-Point Remainder Calculations
    - Increment and Decrement Operators
    - Unary Minus Operator
    - Unary Plus Operator

- Compound Assignment Operators
- Comparison Operators
- Ternary Conditional Operator
- Nil Coalescing Operator
- Range Operators
    - Closed Range Operator
    - Half-Open Range Operator
- Logical Operators
    - Logical NOT Operator
    - Logical AND Operator
    - Logical OR Operator
    - Combining Logical Operators
    - Explicit Parentheses

**Strings and Characters**

- String Literals
- Initializing an Empty String
- String Mutability
- String are Value Types
- Working with Characters
- Concatenating Strings and Characters
- String Interpolation
- Unicode
    - Unicode Scalars
    - Special Unicode Characters in Special Literals
    - Extended Grapheme Clusters
- Counting Characters
- Comparing Strings
    - String and Character Equality
    - Prefix and Suffix Equality
- Unicode Representations of Strings
    - UTF-8 Representation
    - UTF-16 Representation
    - Unicode Scalar Representation

**Collection Types**

- Mutability of Collections
- Arrays
    - Array Type Shorthand Syntax
    - Array Literals
    - Accessing and Modifying an Array
    - Iterating Over an Array
    - Creating and Initializing an Array
- Dictionaries
    - Dictionary Type Shorthand Syntax
    - Dictionary Literals
    - Accessing and Modifying a Dictionary
    - Iterating Over a Dictionary
    - Creating an Empty Dictionary
    - Hash Values for Dictionary Key Types

**Control Flow**

- For Loops
    - For-In
    - For-Condition-Increment
- While Loops
    - While

    - Do-While
- Conditional Statements
    - If
    - Switch
    - No Implicit Fallthrough
    - Range Matching
    - Tuples
    - Value Bindings
    - Where
- Control Transfer Statements
    - Continue
    - Break
    - Break in a Loop Statement
    - Break in a Switch Statement
    - Fallthrough
    - Labeled Statements

**Functions**

- Defining and Calling Functions
- Function Parameters and Return Values
    - Multiple Input Parameters
    - Functions Without Parameters
    - Functions Without Return Values
    - Functions with Multiple Return Values
    - Optional Tuple Return Values
- Function Parameter Names
    - External Parameter Names
    - Shorthand External Parameter Names
    - Default Parameter Values
    - External Names for Parameters with Default Values
    - Variadic Parameters
    - Constant and Variable Parameters
    - In-Out Parameters
- Function Types
    - Using Function Types
    - Function Types as Parameters Types
    - Function Types as Return Types
- Nested Functions

**Closures**

- Closure Expressions
    - The Sorted Function
    - Closure Expression Syntax
    - Inferring Type From Context
    - Implicit Returns from Single-Expression Closures
    - Shorthand Argument Names
    - Operator Functions
- Trailing Closures
- Capturing Values
- Closures Are Reference Types

**Enumerations**

- Enumeration Syntax
- Matching Enumeration Values with a Switch Statement
- Associated Values
- Raw Values

**Classes and Structures**

- Comparing Classes and Structures
    - Definition Syntax
    - Class and Structures Instances
    - Accessing Properties
    - Memberwise Initializers for Structure Types
- Structures and Enumerations Are Value Types
- Classes are Reference Types
    - Identity Operators
    - Pointers
- Choosing Between Classes and Structures
- Assignment and Copy Behavior for Strings, Arrays, and Dictionaries

**Properties**

- Stored Properties
    - Stored Properties of Constant Structure Instances
    - Lazy Stored Properties
    - Stored Properties and Instance Variables
- Computed Properties
    - Shorthand Setter Declaration
    - Read-Only Computed Properties
- Property Observers
- Global and Local Variables
- Type Properties
    - Type Property Syntax
    - Querying and Setting Type Properties

**Methods**

- Instance Methods
    - Local and External Parameter Names for Methods
    - Modifying External Parameter Name Behavior for Modifying
    - The `self` Property
    - Modifying Value Types from Within Instance Methods
    - Assigning to `self` Within a Mutating Method
- Type Methods

**Subscripts**

- Subscript Syntax
- Subscript Usage
- Subscript Options

**Inheritance**

- Defining a Base Class
- Subclassing
- Overriding
    - Accessing Superclass Methods, Properties, and Subscripts
    - Overriding Methods
    - Overriding Properties
    - Overriding Property Getters and Setters
    - Overriding Property Observers
- Preventing Overrides

**Initialization**

- Setting Initial Values for Stored Properties
    - Initializers
    - Default Property Values
- Customizing Initialization
    - Initialization Parameters
    - Local and External Parameter Names
    - Initializer Parameter Without External Names
    - Optional Property Names
    - Modifying Constant Properties During Initialization
- Default Initializers
    - Memberwise Initializers for Structure Types
- Initializer Delegation for Value Types
- Class Inheritance and Initialization
    - Designated Initializers and Convenience Initializers
    - Syntax for Designated and Convenience Initializers
    - Initializer Delegation for Class Types
    - Two-Phase Initialization
    - Initializer Inheritance and Overriding
    - Automatic Initializer Inheritance
    - Designated and Convenience Initializers in Action
- Failable Initializers
    - Failable Initializers for Enumerations
    - Failable Initializers for Enumerations with Raw Values
    - Failable Initializers for Classes
    - Propagation of Initialization Failure
    - Overriding a Failable Initializer
    - The init! Failable Initializer
- Required Initializers
- Setting a Default Property Value with a Closure or Function

**Deinitialization**

- How Deinitialization Works
- Deinitializers in Action

## Automatic Reference Counter

- How ARC Works
- ARC in Action
- Strong Reference Cycles Between Class Instances
- Resolving Strong Reference Cycles Between Class Instances
    - Weak References
    - Unowned References
    - Unowned References and Implicitly Unwrapped Optional Properties
- Strong Reference Cycles for Closures
- Resolving Strong Reference Cycles for Closures
    - Defining a Capture List
    - Weak and Unowned References

**Optional Chaining**

- Optional Chaining as an Alternative to Forced Unwrapping
- Defining Model Classes for Optional Chaining
- Access Properties Through Optional Chaining
- Calling Methods Through Optional Chaining
- Accessing Subscripts Through Optional Chaining
    - Accessing Subscripts of Optional Type
- Linking Multiple Levels of Chaining
- Chaining on Methods With Optional Return Values

**Type Casting**

- Defining a Class Hierarchy for Type Casting
- Checking Type
- Downcasting
- Type Casting for Any and AnyObject
    - AnyObject
    - Any

**Nested Types**

- Nested Types in Action
- Referring to Nested Types

**Extensions**

- Extension Syntax
- Computed Properties
- Initializers
- Methods
    - Mutating Instance Methods
- Subscripts
- Nested Types

**Protocols**

- Protocol Syntax
- Property Requirements
- Method Requirements
- Mutating Method Requirements
- Initializer Requirements
- Protocols as Types
- Delegation
- Adding Protocol Conformance with an Extension
    - Declaring Protocol Adoption with an Extension
- Collections of Protocol Types
- Protocol Inheritance
- Protocol Composition
- Checking for Protocol Conformance
- Optional Protocol Requirements

**Generics**

- The Problem That Generics Solve
- Generic Functions
- Type Parameters
- Naming Type Parameters
- Generic Types
- Extending a Generic Type
- Type Constraints
    - Type Constraint Syntax
    - Type Constraints in Action
- Associated Types
    - Associated Types in Action
    - Extending an Existing Type to Specify an Associated Type
- Where Clauses

**Access Control**

- Modules and Source Files
- Access Levels
  - Guiding Principle of Access Levels
  - Default Access Levels
  - Access Levels for Single-Target Apps
  - Access Levels for Frameworks
- Access Control Syntax
- Custom Types
  - Tuple Types
  - Function Types
  - Enumeration Types
  - Raw Values and Associated Values
  - Nested Types
- Subclassing
- Constants, Variables, Properties, and Subscripts
  - Getters and Setters
- Initializers
  - Default Initializers
  - Default Memberwise Initializers for Structure Types
- Protocols
  - Protocol Inheritance
  - Protocol Conformance
- Extensions
  - Adding Protocol Conformance with an Extension
- Generics
- Type Aliases

### Advanced Operators

- Bitwise Operators
  - Bitwise NOT Operator
  - Bitwise AND Operator
  - Bitwise OR Operator
  - Bitwise XOR Operator
  - Bitwise Left and Right Shift Operators
  - Shifting Behavior for Unsigned Integers
  - Shifting Behavior for Signed Integers
- Overflow Operators
  - Value Overflow
  - Value Underflow
  - Division by Zero
- Precedence and Associativity
- Operator Functions
  - Prefix and Postfix Operators
  - Compound Assignment Operators
  - Equivalence Operators
- Custom Operators
  - Precedence and Associativity for Custom Infix Operators

## Language Reference

### About the Language Reference

- How to Read the Grammar

### Lexical Structure

- Whitespace and Comments
- Identifiers

- Keywords and Punctuation
- Literals
    - Integer Literals
    - Floating-Point Literals
    - String Literals
- Operators

**Types**

- Type Annotation
- Type Identifier
- Tuple Type
- Function Type
- Array Type
- Dictionary Type
- Optional Type
- Implicitly Unwrapped Optional Type
- Protocol Composition Type
- Metatype Type
- Type Inheritance Clause
- Type Inference

**Expressions**

- Prefix Expressions
- Binary Expressions
    - Assignment Operator
    - Ternary Conditional Operator
    - Type-Casting Operators
- Primary Expressions
    - Literal Expressions
    - Self Expression
    - Superclass Expression
    - Closure Expression
    - Implicit Member Expression
    - Parenthesized Expression
    - Wildcard Expression
- Postfix Expressions
    - Function Call Expression
    - Initializer Expression
    - Explicit Member Expression
    - Postfix Self Expression
    - Dynamic Type Expression
    - Subscript Expression
    - Forced-Value Expression
    - Optional-Chaining Expression

**Statements**

- Loop Statements
    - For Statement
    - For-In Statement
    - While Statement
    - Do-While Statement
- Branch Statements
    - If Statement
    - Switch Statement
    - Switch Statements Must be Exhaustive
    - Execution Does Not Fall Through Cases Implicitly
- Labeled Statement

- Control Transfer Statements
    - Break Statement
    - Continue
    - Fallthrough Statement
    - Return Statement

**Declarations**

- Top-Level Code
- Code Blocks
- Import Declaration
- Constant Declaration
- Variable Declaration
    - Stored Variables and Stored Variable Properties
    - Computed Variables and Computed Properties
    - Stored Variable Observers and Property Observers
    - Class and Static Variable Properties
- Type Alias Declaration
- Function Declaration
    - Parameter Names
    - Special Kinds of Parameters
    - Special Kinds of Methods
    - Curried Functions
- Enumeration Declaration
    - Enumeration with Cases of Any Type
    - Enumeration with Cases of Raw-Value Type
    - Accessing Enumeration Cases
- Structure Declaration
- Class Declaration
- Protocol Declaration
    - Protocol Property Declaration
    - Protocol Method Declaration
    - Protocol Initializer Declaration
    - Protocol Subscript Declaration
    - Protocol Associated Type Declaration
- Initializer Declaration
- Deinitializer Declaration
- Extension Declaration
- Subscript Declaration
- Operator Declaration
- Declaration Modifiers
    - Access Control Levels

**Attributes**

- Declaration Attributes
    - Declaration Attributes Used by Interface Builder
- Type Attributes

**Patterns**

- Wildcard Pattern
- Identifier Pattern
- Value-Binding Pattern
- Tuple Pattern
- Enumeration Case Pattern
- Type-Casting Pattern
- Expression Pattern

**Generic Parameters and Arguments**

- Generic Parameters Clause
  - Where Clauses
- Generic Argument Clause

**Summary of the Grammar**

- Statements
- Generic Parameters and Arguments
- Declarations
- Patterns
- Attributes
- Expressions
- Lexical Structure
- Types

**Revision History**