# Swift Standard Library

Solve complex problems and write high-performance, readable code.

## Overview

The Swift standard library defines a base layer of functionality for writing Swift programs, including:

- Fundamental data types such as `Int`, `Double`, and `String`
- Common data structures such as `Array`, `Dictionary`, and `Set`
- Global functions such as `print(_:separator:terminator:)` and `abs(_:)`
- Protocols, such as `Collection` and `Equatable`, that describe common abstractions.
- Protocols, such as `CustomDebugStringConvertible` and `CustomReflectable`, that you use to customize operations that are available to all types.
- Protocols, such as `OptionSet`, that you use to provide implementations that would otherwise require boilerplate code.

Note

Experiment with Swift standard library types and learn high-level concepts using visualizations and practical examples. Learn how the Swift standard library uses protocols and generics to express powerful constraints. Download the playground below to get started.

Swift Standard Library.playground

## Topics

## Values and Collections

Numbers and Basic Values

Model data with numbers, Boolean values, and other fundamental types.

Strings and Text

Work with text using Unicode-safe strings.

Collections

Store and organize data using arrays, dictionaries, sets, and other data structures.

Time

Measure how long an operation takes and determine schedules in the future.

## Tools for Your Types

Basic Behaviors

Use your custom types in operations that depend on testing for equality or order and as members of sets and dictionaries.

Encoding, Decoding, and Serialization

Serialize and deserialize instances of your types with implicit or customized encoding.

Initialization with Literals

Allow values of your type to be expressed using different kinds of literals.

## Programming Tasks

Input and Output

Print values to the console, read from and write to text streams, and use command line arguments.

Debugging and Reflection

Fortify your code with runtime checks, and examine your values’ runtime representation.

Macros

Generate boilerplate code and perform other compile-time operations.

Concurrency

Perform asynchronous and parallel operations.

Key-Path Expressions

Use key-path expressions to access properties dynamically.

Manual Memory Management

Allocate and manage memory manually.

Type Casting and Existential Types

Perform casts between types or represent values of any type.

C Interoperability

Use imported C types or call C variadic functions.

Operator Declarations

Work with prefix, postfix, and infix operators.

## Deprecated

Deprecated

---

## See Also

### Standard Library

`struct Int`

A signed integer value type.

`struct Double`

A double-precision, floating-point value type.

`struct String`

A Unicode string value that is a collection of characters.

`struct Array`

An ordered, random-access collection.

`struct Dictionary`

A collection whose elements are key-value pairs.