

# **The Swift Language Handbook**

An Introduction  
and Guidebook for  
Pascal Programmers



**Henry Lowe**

The Swift Language Handbook  
An Introduction and Guidebook for  
Pascal Programmers

Copyright © 2023 by Henry Lowe

All Rights Reserved. This book was published by the author Henry Lowe under Ascriva Press. No part of this book may be reproduced in any form by any means without the express written permission of the author. This includes reprints, excerpts, photocopying, recording, or any future means of reproducing text.

If you would like to do any of the above, please seek permission first by contacting the author at [hlowe@ascriva.com](mailto:hlowe@ascriva.com)

Published in the United States by  
Ascriva Health Informatics LLC, Stanford, California.

[www.ascriva.com](http://www.ascriva.com)



- Preface ..... 1
  - Origins of This Book ..... 1
  - The Pascal Programming Language ..... 2
  - The Swift Programming Language ..... 3
- Tools and References ..... 3
  - The Swift Playgrounds App ..... 3
  - Code Examples ..... 4
  - Swift’s Print() Statement ..... 5
  - References ..... 6
- 1 Introduction ..... 7
  - 1.1 The Form of a Program ..... 7
  - 1.2 Declaration ..... 9
    - 1.2.1 Case Sensitivity ..... 9
    - 1.2.2 Constants ..... 10
    - 1.2.3 Variables ..... 12
    - 1.2.4 Constant and Variable Names ..... 13
- 2 Operators ..... 15
  - 2.1 Assignment ..... 15
  - 2.2 Equality ..... 15
  - 2.3 Non-Equality ..... 16
  - 2.4 Arithmetic Operators ..... 17
  - 2.4 Logical Operators ..... 19
  - 2.5 Range Operators ..... 21
- 3 Data Types ..... 24
  - 3.1 Numbers ..... 24
    - 3.1.1 Integer ..... 24

3.1.2 Floating Point Number .....	25
3.1.3 Numeric Type Conversion .....	26
3.1.4 Rounding .....	26
3.1.5 Truncation .....	28
3.1.6 Generating Random Numbers .....	28
3.1.7 Numeric Functions .....	29
3.1.8 Converting Numbers to Strings .....	30
3.2 Booleans .....	31
3.3 Strings and Characters .....	32
3.3.1 Character Declaration .....	32
3.3.2 String Declaration .....	33
3.3.3 String Literals .....	34
3.3.4 String Length .....	37
3.3.5 String Character Indices .....	38
3.3.6 String Concatenation .....	43
3.3.7 String Comparison .....	45
3.3.8 String Interpolation .....	46
3.3.9 String Insertion and Removal .....	47
3.3.10 String Search .....	50
3.3.11 String Replace .....	51
3.3.12 String Split .....	52
3.3.13 Substrings .....	53
3.4 Arrays .....	54
3.4.1 Declaration .....	55
3.4.2 Initialization .....	55
3.4.3 Element Operations .....	56
3.4.4 Searching an Array .....	59

3.4.5	Sorting an Array.....	65
3.5	Sets .....	66
3.5.1	Declaration.....	67
3.5.3	Set Operations.....	68
3.5.4	Sorting a Set.....	70
3.6	Dictionaries.....	70
3.6.1	Declaration.....	71
3.6.2	Adding and Deleting Entries.....	72
3.6.3	Optionals.....	73
3.6.4	Iteration.....	76
3.6.5	Sorting Entries .....	77
3.6.6	Searching Dictionaries.....	77
3.7	Records and Structures .....	80
3.7.1	Declaration.....	81
3.7.2	Property Values.....	83
3.7.3	Instance Methods.....	85
3.8	Enumeration.....	86
3.8.1	Declaration.....	87
3.9	Type Alias .....	93
4	Control Flow .....	94
4.1	If Statement.....	94
4.2	If Then Else Statement .....	95
4.3	Guard Statement .....	97
4.4	Switch Statement .....	99
4.5	Repeat Loops .....	101
4.6	Repeat While Loops .....	105
4.7	Control Transfer Statements .....	107

4.7.1 Break Statement.....	107
4.7.2 Continue Statement.....	108
4.7.3 Fallthrough Statement.....	109
5 Functions and Procedures .....	110
5.1 Function Definition.....	112
5.2 Calling Swift Functions .....	113
5.3 Argument Labels.....	113
5.4 Return Values.....	115
5.5 Function Parameters .....	115
5.5.1 Variadic Parameters .....	117
5.6 Emulating Pascal Procedures.....	118
5.7 Tuples .....	121
5.8 Mutating Functions.....	124
5.9 Generic Functions.....	127
5.10 Defer .....	128
6 File Management.....	129
6.1 URL .....	130
6.6.1 Declaration.....	131
6.6.2 Methods .....	133
6.3 FileManager.....	135
6.3.1 Declaration.....	135
6.3.2 Methods .....	135
6.3.3 Functions & Methods .....	137
6.4 Data.....	140
6.4.1 Declaration.....	140
6.4.2 Iteration.....	141

- 6.4.3 Reading and Writing Data Files.....141
- 6.4.4 Splitting .....142
- 7 Error Handling.....143
  - 7.1 Throw Statement.....143
  - 7.2 Do-Catch Statement.....145
  - 7.3 Using Optionals .....146
- 8 Extensions.....149
- 9 Modules .....153
  - 9.1 Access Control.....153
- 10 Protocols .....156
- Index.....160
- About the Author .....177

# Preface

## Origins of This Book

I first programmed in Pascal in 1982, using Apple UCSD Pascal for the Apple II family of computers. I was fascinated by Pascal's support for structured programming, significantly enhancing the design, implementation, and maintenance of code. The ability to create rich user-defined data types in Pascal was also a powerful feature of the language. At the time, Pascal impressed me as an elegant, compact, intuitive, and extensible language. It was also a "safe" language because of strong type checking and sparse use of pointers.

In 1982 I discovered Randy Clarke and Stephen Koehler's wonderful book: The UCSD Pascal Handbook - A Reference and Guidebook for Programmers. Using this well-designed book, one could quickly review UCSD Pascal's features, language and syntax - plus code examples. As I began using the Swift programming language 40 years later, I yearned for a similar short reference guide.

Not finding one that met my needs, I decided to create this book, which is designed to help those familiar with Pascal begin exploring Swift. Where possible, I have tried to



demonstrate how these two elegant languages intersect, hopefully creating a conceptual bridge between them.

This book is not intended to be a comprehensive reference guide to Swift. Nor does it cover all aspects of the Swift language. Instead, it attempts to map the core concepts of Standard Pascal to the equivalent language concepts in Swift to hopefully facilitate that “Aha!” moment when one gets a firm conceptual foothold on a new programming paradigm.

## **The Pascal Programming Language**

The Pascal programming language was designed by Nicklaus Wirth at the Federal Institute of Technology in Zürich, Switzerland, in 1970. Pascal is descended from the Algol programming language, which was first described in 1958. By the 1980’s Pascal had become the standard programming language used to teach college-level computer science. It evolved to become an object-oriented language (Object Pascal) and was used in the development of the original Apple Macintosh. In time, Pascal was eclipsed by other languages, particularly C, C++, Objective-C and Java. Pascal lives on today as Delphi Pascal and the Free Pascal Compiler (FPC).

## **The Swift Programming Language**

Swift is a compiled general-purpose object-oriented programming language developed by Apple Inc. and the open-source community. Development of Swift began at Apple in 2010 by [Chris Lattner](#), with the eventual collaboration of many other programmers. Swift was designed to replace Apple's earlier programming language Objective-C, which lacked many modern language features. Swift was first released in 2014 and, at the time of writing (April 2023), is at version 5.8, supporting all Apple platforms and Linux.

Henry Lowe  
Stanford 2023

## **Tools and References**

### **The Swift Playgrounds App**

To experiment with the Swift code examples in this book I recommend installing Apple's [Playgrounds app](#) for Mac and iPad. Playgrounds provides an interactive Read Eval Print Loop (REPL) environment into which you type Swift code and get instant feedback.

Launch the Playgrounds app and select ‘New Blank Playground’ from the Mac File Menu or tap ‘+ Playground’ in the iPad version of the app. This gives you a new empty playground into which you can enter Swift code. Entering ‘import Foundation’ as the first line of the playground ensures that Swift’s Foundation framework is available when you use the examples in this book.

## Code Examples

Throughout this book you will find illustrative code examples.

Pascal code examples are printed as plain text, e.g:

```
Program Hello;  
    Var Greeting:String;  
    Begin  
        Greeting:='Hello world';  
        Writeln(Greeting);  
    End.
```

Swift code examples are displayed within a colored text box:

```
let greeting = "Hello world"  
print(greeting)
```

## Swift's Print() Statement

Swift's `print()` statement is used in the code examples throughout this book. The `print()` statement sends output to the console. If you are using the Playgrounds app to experiment with Swift, the `print()` statement is a useful way to display constants, variables and data in the console pane of a playground window. For example, enter `'print("Hello world!")'` And the text 'Hello world' is displayed in the playground's console pane.

When using Swift's `print()` statement all non-string variables are converted to strings automatically.

The full definition of `print()` is:

```
print(items, separator, terminator)
```

The 'items' parameter contains what we wish to display either as a literal, constant, variable or expression.

The optional 'separator' parameter defines the character used to separate multiple items within `print()`. The default separator is a single space character.

The optional ‘terminator’ parameter defines the character used to terminate the display output. The default terminator is new line character “\n”. Alternatives include: the tab character “\t” and the space character.

We can use these optional print() parameters as:

```
let word1 = "hello"  
let word2 = "world"  
print(word1,word2,separator:"-",terminator:"\t")  
// displays hello-world
```

## References

The definitive Swift reference is Apple’s ‘The Swift Programming Language’, which is available for free from the Apple Bookstore using Apple’s Books app. I highly recommend using this book as your standard Swift reference.

Online Swift documentation is available on the Apple Developer Site (subscription required) - <https://developer.apple.com/documentation/swift/> and on the (free) Swift Open Source site at <https://www.swift.org>.

# 1 Introduction

**P**ascal and Swift share many core conceptual features.

## 1.1 The Form of a Program

A Pascal program has the general structure shown below:

```
Program HelloWorld;  
    Var (* Declare a variable *)  
        Greeting: String;  
Begin  
    Greeting:='Hello World';  
    Writeln(Greeting);  
End.
```

A Swift program may be written as:

```
var greeting: String //Declare a variable  
greeting = "Hello World"  
print(greeting)
```

Let's start with some obvious differences in the example: Unlike Pascal, Swift has no construct to indicate the start of a program. A Swift program begins with the first line of code, in this case: `var greetings: String`.

Pascal requires that statements end with a semicolon. This is not a requirement in Swift. However, a semicolon can be used to separate multiple statements written in the same line of Swift source code. While Swift does not require a semicolon after each statement, you can use one if you wish.

A major difference between Pascal and Swift is Swift's (and many other modern languages) use of 'curly braces' i.e. { } to define blocks of code. A code block is a scope (i.e., the set of all variable-name bindings visible to the compiler within a part of a program) that determines the lifetime of variables declared within the block. Variables declared within a block are not valid when control exits the block. Curly braces are also used to define the beginning and end of logical programming structures such as loops. Pascal uses Begin and End statements to define blocks of code. More on the use of curly braces later.

A single line comment in Swift begins with // and multi-line comments begin with /\* and end with \*/. In Pascal source code comments can be enclosed in 'curly brackets' or parentheses e.g.: {Comment} (\* Comment \*).

## 1.2 Declaration

All variables in a Pascal program have a specific type. Some types are built into the language, while others are user-defined. Swift follows a similar model.

### 1.2.1 Case Sensitivity

Pascal is a case-insensitive language. You can use all upper case, all lower case or a mix of cases when writing Pascal code. Identifiers, constants, and variables are also case-insensitive. One consequence of this is that in Pascal the variable 'MyVariable' and 'myVariable' reference the same data.

Swift is a case-sensitive language: 'MyVariable' and 'myVariable' are two completely different variables.

One common problem that Pascal programmers encounter when starting to use Swift is forgetting to use the correct case when entering identifiers. Most Swift commands are all lower case. For example, use 'print' not 'Print' or 'PRINT':

```
print("Hello World") // is fine
Print("Hello World") // fails - uppercase 'P'
```



Swift's built-in data types are written with the initial character in uppercase e.g.: String, Int, Double, Bool, Array etc.

### ***1.2.2 Constants***

Constants are declared once in a program and their value cannot be changed during program execution. Constants can be declared anywhere in a program, but they must be declared before they are referenced. In Pascal constants are declared using the Const keyword:

#### **Const**

```
Dozen = 12;  
Pi = 3.14159;  
ErrorMessage = 'An error occurred';
```

Swift does not use the Const keyword but instead declares constants using the 'let' keyword e.g

```
let Dozen = 12  
let Pi = 3.14159  
let errorMessage = "An error occurred"
```

As in Pascal, Swift does not require that you define a constant's type, but you can if you wish. This is referred to as 'Type Annotation':

```
let Dozen: Int = 12
let Pi: Double = 3.14159
let errorMessage: String = "An error occured"
```

In Swift multiple constants can be declared on a single line separated by commas e.g:

```
let minValue = 1, maxValue = 31
```

Or

```
let minValue: Int = 1, maxValue: Int = 99
```

As in Pascal, Swift also allows the use of expressions in constants:

```
let square2 = 2 * 2
```

In Swift, for performance reasons, it is recommended that you use constants to hold data that will not change during program execution.

### 1.2.3 Variables

In Swift (as in Pascal) a variable's value can be changed during program execution. Variables must be declared before they are referenced. Swift variables are declared as:

```
var userName: String
var startValue: Int = 100
var mean, median, mode: Int
```

Note that Swift variable declarations do not require ending the declaration with a semicolon, as is the case in Pascal, though you can add a semicolon if you so desire.

In Swift the following variable declarations are legal:

```
var userName: String
var userName: String;
```

In Swift it is usually not necessary (or required) to declare the variable type (Type Annotation), as the Swift compiler can usually infer the type from the declaration (Type Inference). The following variable declarations are legal:

```
var greetingMessage = "Hello World"
var startValue = 10
```

Once a variable or constant has been declared to hold values of a specific type, you cannot redeclare that constant or variable later in the program to hold values of a different type.

### ***1.2.4 Constant and Variable Names***

Swift constant and variable names can contain almost any character, including Unicode characters e.g.:

```
let 🐸 = "Frog"
```

Variable names cannot start with a number. Names cannot contain whitespace characters, math symbols, arrows, private-use Unicode scalar characters or line and box-drawing characters.

It is recommended (for readability purposes) to use a ‘Camel case’ naming convention when naming constants and variables in Swift. Camel case uses a lowercase letter for the first character of the first ‘word’ in a variable name followed by a capital letter for each subsequent ‘word’ e.g:

```
var myVariableName: String
```

Pascal programmers often use a similar convention, called 'Pascal Case' - in which the first character of a constant or variable name made of compound words is upper case - as opposed to lower case in Swift.

Var: MyFileName: String; - Pascal Case

var myFileName: String - Camel Case in Swift

Constant and variable names in Swift can be of any length and all characters are significant. As Swift is a case-sensitive language the following are considered separate variables:

```
var MYVARIABLENAME: String
var myvariablename: String
var myVariableName: String
```

## 2 Operators

Pascal and Swift support a similar set of operators, though with different syntax.

### 2.1 Assignment

Pascal uses the ‘:=’ operator for assignment. For example:

```
Var Int1: Integer;  
Int1:=42;
```

Swift uses the ‘=’ operator for assignment. For example:

```
var int1,int2,int3:Int  
int1 = 42
```

### 2.2 Equality

The equality operator in Pascal is ‘=’. For example:

```
If int1 = int2 then ...
```

In Swift the equality operator is '==', e.g.:

```
var int1,int2:Int
int1 = 1;int2 = 1
if int1 == int2 {
    let equal = true }
```

## 2.3 Non-Equality

In Pascal we test for non-equality using the '<>' operator:

```
If int1 <> int2 then ...
```

In Swift we test for non-equality using the '!=' operator:

```
var int1,int2:Int
int1 = 1
int2 = 1
if int1 != int2 {
    let equal = false
}
```

In both Pascal and Swift to test if one variable is less than another we use the '<' operator, and use the '>' operator to test if one variable is greater than another:

```
var int1,int2:Int
int1 = 1
int2 = 1
if int1 > int2 {
    let greaterThan = false
}
```

The Greater Than Or Equals To operator '>=' and the Less Than Or Equals To operator '<=' are also the same in both Pascal and Swift.

## 2.4 Arithmetic Operators

Pascal and Swift use the same operators for addition '+', subtraction '-', division '/' and multiplication '\*':

```
var int1,int2,int3:Int
int1 = 1
int2 = 1
int3 = (int1 + int2) * (int2 / int1) - int1
```



The Remainder operator returns the remainder when one

```
Int3:=int2 Mod Int1;
```

number is divided by another. In Pascal this is called ‘Mod’, which is short for Modulo:

Swift uses the Remainder operator ‘%’ in a similar fashion:

```
var int1,int2,int3:Int  
int3 = int2 % int1
```

In both Pascal and Swift one can change the sign of a number by prefixing a numeric value with the ‘-’ operator (the Unary Minus Operator). This operator toggles the sign of a number:

```
var int1:Int  
int1 = 1  
int1 = -int1 // int1 = -1  
int1 = -int1 // int1 = 1
```

The Unary Plus Operator ‘+’ does not change the sign of the number and its inclusion in both Pascal and Swift appears to be for symmetry.

## 2.4 Logical Operators

Both Pascal and Swift support the Logical Operators AND, OR and NOT:

The AND operator is 'AND' in Pascal and '&&' in Swift.

In Pascal:

```
Var Bool1,Bool2:Boolean;  
Bool1:=True;  
Bool2:=False;  
If Bool1 AND Bool2 then  
    Writeln('True')  
    Else  
    Writeln('False');
```

In Swift:

```
var bool1:Bool = true  
var bool2:Bool = false  
  
if bool1 && bool2 {  
    print ("True")  
} else {  
    print ("False")  
}
```

The OR operator is 'OR' in Pascal and '||' in Swift.

In Pascal:

```
Var Bool1,Bool2:Boolean;  
Bool1:=True;  
Bool2:=False;  
If Bool1 OR Bool2 then  
    Writeln('True')  
Else  
    Writeln('False');
```

In Swift:

```
if bool1 || bool2 {  
    print ("True")  
} else {  
    print ("False")  
}
```

The NOT operator is 'NOT' in Pascal and '!' in Swift.

In Swift Logical NOT is a prefix operator (i.e., appears just before the value it operates on, without whitespace) that toggles a Boolean value.

```
let bool1 = false  
let bool2 = !bool1 // bool2 = true
```

In Pascal **NOT** is used in a similar fashion:

```
Bool1:=False;  
Bool2:=Not Bool1; {Bool2 is True}
```

## 2.5 Range Operators

Swift includes several range operators that define a range of values. The closest equivalent construct in Pascal are subranges. In Pascal we can define a type, constant or variable as a subrange of any scalar type. For example:

```
digits = 0..9; (* Subrange of Integer *)
Grade = "A,B,C,D,E,F" (* User-defined type *)
pass = 'A'..'C'; (* Subrange of Grade *)
```

Pascal may also use subranges when defining arrays:

```
myArray[1..99] of String;
```

Swift's Closed Range Operator ('x...y') defines a range extending from the value of x to the value of y. For example, 2...7 is a range of integers from 2 to 7. The value of x must not be greater than the value of y. Note the three periods used by Swift as opposed to the two periods used in Pascal.

Swift's Half-Open Range Operator (x.. $<$ y) defines a range starting with the value of x but not including the value of y. For example, 2.. $<$ 7 is a range of integers starting with 2 and ending with 6. This type of range operator is useful when working with zero-based lists, such as Swift arrays. The

value of x must not be greater than the value of y. Note the use of two periods in this type of range.

Swift's One-Sided Range operator defines a range that starts with a value and extends as far as possible beyond that start value. For example, if an array contains 100 members, e.g. `Array[0..99]` then the One-Sided Range `[9...]` will include the array members `Array[9]` to `Array[99]`. We can also use a form of the One-Sided Range in which the left side of the range is undefined, e.g. `[...9]` would reference array members 0 to 9.

Ranges are often used in Swift for-in loops. In Pascal we might define a for-loop as:

```
for index:= 1 to 9
    writeln(index);
```

In Swift we can iterate in a similar fashion using ranges:

```
var index1, index2, index3: Int
var myArray: [Int] = [1,2,3,4,5]
for index1 in 1...9 {
    print(index1) }
for index2 in 1..<10 {
    print(index2) }
for index3 in myArray[1...] {
    print(myArray[index3]) }
```

Because Swift ranges are a type, there are range properties and methods that we can use:

```
let myRange = 0...9
print(myRange.contains(5)) //true
print(myRange.lowerBound) // 0
print(myRange.upperBound) // 9
print(myRange.isEmpty) //false
```

Swift also supports comparing ranges:

```
let myRange = 0...9
let anotherRange = 1...8
print(myRange == anotherRange) //false
print(myRange != anotherRange) //true
```

We can also check if ranges overlap:

```
print(myRange.overlaps(anotherRange)) //true
```

# Index

## A

- Absolute value of a number, 29
- Access Control, 153
  - Fileprivate, 155
  - Internal, 154
  - Open, 154
  - Private, 155
  - Public, 154
- Algol programming language, 1
- And, 19
- Array, 21, 54, 63, 70, 71, 72
  - Accessing all elements, 59
  - Appending new elements, 57
  - Changing elements, 58
  - Counting elements, 57
  - Declaration, 55
  - Determining if empty, 57
  - Element operations, 56
  - Filtering, 64
  - Index, 56
  - Initialization, 55
  - Inserting elements, 57
  - Obtaining element index and value, 59
  - Removing elements, 58
  - Retrieving elements, 58
  - Searching, 59
  - Shuffle, 66
  - Zero-indexed, 55

Assignment, 15  
Associative Array, 70

## **B**

Bool type in Swift, 31  
Booleans, 31  
Break statement, 107  
Break to label, 108

## **C**

Case sensitivity, 9  
Ceil(), 26  
Chaining of methods, 133  
Change the sign of a number. *See* Operators, Unary Minus  
Character, useful methods, 33  
Characters, 32  
Closures, 78  
Code block, definition, 8  
Comments, in source code, 8  
ComparisonResult, 46, 61  
Constant and variables, naming, 13  
Constants, 10  
Continue statement, 108  
Control Flow, 94  
Control transfer statements, 107  
Converting Numbers to Strings, 30  
Cubed root of a number, 29  
Curly braces { }, 8



## **D**

### Data

- Declaration, 140
- Iteration, 141
- Reading and writing data files, 141
- Splitting, 142

### Data type, 140

### Data types, 9

### Data Types

- Double, 25
- Float, 25
- Floating point numbers, 25
- Uint, 25

### Defer statement, 128

### Delphi Pascal, 1

### Dictionary

- Adding entries, 72
- Case sensitivity, 77
- Changing entries, 72
- Converting to array, 76
- Counting key-value pairs, 77
- Creating from arrays, 72
- Declaration, 71
- Definition, 70
- Display values, 74
- Empty, 71
- Extensions, 78
- Iteration, 76
- Removing keys, 75
- Removing key-value pair, 75
- retrieving key list, 76

- Retrieving value list, 76
- Searching, 77
- Sorting keys and values, 77
- Subscript syntax, 72
- Subscripts, 72
- Updating entries, 73

Do-Catch Statement, 145

Double, 25

## **E**

Emulating Pascal procedures, 118

eNum, 46

Enum

- AllCases, 88
- Associated values, 89
- Case, 86
- CaseIterable, 88
- Count cases, 88
- Creating new constants or variables from, 88
- Declaration, 87
- Definition, 86
- Dot notation, 88
- List case labels, 88
- Raw values, 89
- Using collection properties with, 91

enumeration. *See* enum

Enumerations. *See* eNum

Equality, 15

Error Handling, 143

Error handling with optionals, 146

Error Protocol, 143

Exponential function, 30  
Extension Keyword, 150  
Extensions, 149

## **F**

Fallthrough statement, 109

File Management, 129

Filemanager

- Creating a new directory, 138

- Creating a new file, 139

- Declaration, 135

- Description, 135

- Methods and properties, 135

- Obtaining path to directory, 136

- Obtaining path to Documents directory, 136

- Reading a file, 140

- User domain mask, 137

- Writing to a file, 139

FileManager, 135

Fileprivate access level, 155

Float, 25

Floating point numbers, 25

Floor(), 27

Free Pascal Compiler, 1

Func. *See* Function

Function

- Ampersand '&', 120

- Argument labels, 113

- Calling, 113

- Calling without parameters, 115

- Defer, 128

- Definition, 112
- Generic, 127
- Inout keyword, 118
- Mutating, 124
- Parameter with multiple values, 117
- Parameters, 115
- Return keyword, 115
- Return values, 115
- Throwing, 143
- Type parameter, 127
- Using default parameter values, 116
- Variadic parameters, 117

Functions and procedures, 110

## **G**

- Generic functions, 127
- Generics, 54
- Goto. *See* Break
- Guard Statement, 97

## **H**

- Hash Table, 70
- Hash value, 71
- Hashable, 71

## **I**

- If Statement, 94
- If Then Else Statement, 95
- Import Foundation, 36
- Import Keyword, 153

Integer, 24  
Internal access level, 154

## **J**

JSON, 142

## **K**

Key-value pair, 75

## **L**

Lattner, Chris, 1

Let, 10

Logarithm, 30

Logical Operators, 19

LowerCamelCase, 82

## **M**

Mod. *See* Operators, Remainder

Modules, 153

Modulo. *See* Operators, Remainder

Mutating methods, 86

## **N**

Namespace, 153

Non-equality, 16

Not, 20

Numbers, 24

    Absolute value, 29

    Convert to String, 31

- Exponential function, 30
- Random, 28
- Rounding, 26
- Rounding down, 27
- Rounding rules, 27
- Rounding up. *See* Ceil()
- Rounding, decimal places, 27
- Square, 30
- Square root, 29
- Truncation, 28

## **O**

- Objective-C, 1
- Open access level, 154
- Operators
  - Addition, 17
  - And, 19
  - Arithmetic, 17
  - Assignment, 15
  - Division, 17
  - Equality, 15
  - Logical, 19
  - Multiplication, 17
  - Non-equality, 16
  - Not, 20
  - Or, 19
  - Range, 21
  - Remainder, 18
  - Subtraction, 17
  - Unary Minus, 18
  - Unary Plus, 18
- Optionals, 41, 73, 74, 75, 123

Or, 19

## **P**

parameters, 118

Pascal

Access control, 154

Apple UCSD Pascal, 2

Arrays, 54

Begin .. End, 95

Boolean, 31

Break, 107

Case, 99

Case sensitivity, 9

Char, 32

Code blocks, 95

Code examples, 4

Comments, source code, 8

Concat, 36

Constants, 10

Control Flow, 94

Copy, 36

Data types, 24

Declaration, 9

Delete, 47

Delphi, 1

Down to, 103

Emulating procedures in Swift, 118

File, 129

File management, 129

For, 102

Free Pascal Compiler, 1

Functions, 110

Goto, 107  
Halt, 107  
History, 1  
If, 94  
If then Else, 96  
Insert, 47  
Length, 36  
Numerics, 24  
Operators, 15  
Parameters, 111  
Pos, 36  
Procedures, 110  
Program structure, 7  
Random function, 28  
Real data type, 25  
Records, 80  
Repeat loops, 101  
Repeat Until, 106  
Repeat While, 105  
Return, 110  
Round function, 26  
Semicolon, ending statements with, 8  
Sets, 66  
Str function, 30  
Strings, 32  
Subranges, 21  
Trunc function, 28  
UCSD Pascal Handbook, 2  
Units, 153  
Value parameters, 111  
Variant Record, 91  
Pascal case, 14



Pascal's Copy(Source,Index,Size) function, 39  
Power function, numeric, 30  
Predicates, 61  
Print() statement, 5  
Private access level, 155  
Protocols, 156  
Public access level, 154

## **R**

Random boolean, 29  
Random numbers, 28  
Range  
    Closed ('x..y'), 21  
    Comparing, 23  
    Contains, 23  
    Half-open ( $x..<y$ ), 21  
    IsEmpty, 23  
    Lowerbound, 23  
    Methods, 23  
    One-sided ('x...' or ('..y'), 22  
    Overlap, 23  
    Upperbound, 23  
    Use in iteration, 102  
    Used in loops, 22  
Read Eval Print Loop (REPL), 3  
Records and Structures, 80  
Reference types, 124  
Remainder, 18  
Repeat Loops, 101  
Repeat while loops, 105

Rounding numbers, 26

## **S**

Sandboxing, 132

Scope, definition, 8

Self, 124, 150

Semicolon, ending statements with, 8

Set

- Asymmetric difference, 69

- Declaration, 67

- Definition, 67

- Intersection, 69

- Iteration, 69

- Membership tests, 70

- Operations, 68

- Sorting, 70

- Subtract, 69

- Union, 69

Sets, 66

Square root of a number, 29

Static Keyword, 157

Stride, 104

string, 33

String Character Indices, 38

String Comparison, 45

String Concatenation, 43

String Insertion and Removal, 47

String Interpolation, 46

String Length, 37

String Literals, 34

String Replace, 51

String Search, 50

String Split, 52

## Strings

- Changing case, 50

- Comparison, 45

- Concatenation, 43

- Embedded special characters, 35

- Extended special characters, 177

- Extending Swift Strings, 37

- Indices, 38

- Initialization, 33

- Initialization from a file, 34

- Inserting characters, 47

- Length, 37

- Literals, 34

- Multiline literals, 34

- NSString.CompareOptions, 52

- Removing characters, 48

- Replacing characters, 51

- Searching, 50

- Splitting, 52

- Substrings, 53

- Testing if empty, 35

Strings and characters, 32

## StringStrings

- Interpolation, 46

## Struct

- Computed properties, 84

- Create new instance, 82

- Declaration, 81

- Default property values, 83

- Defining new, 82
- Embedded functions. *See* Instance methods
- Getter and setter methods, 84
- Getting and setting property values, 83
- Initializing properties, 83
- Instance methods, 85
- Properties, 83
- Self reference, 86
- Subscript syntax, 58, 72, 73, 74
- Substrings, 53
- Swift
  - Bridged methods, 36
  - Brief history, 1
  - Case sensitivity, 9
  - Code examples, 4
  - Let keyword, 10
  - Online documentation, 6
  - Playgrounds app, 3
  - Print() statement, 5
  - Program structure, 7
  - Swift.org, 6
  - The Swift Programming Language Reference, 6
- Swift Playgrounds App, 3
- Switch default case, 99
- Switch Statement, 99

## **T**

- Throw Statement, 143
- Throws keyword, 143
- Truncating Numbers, 28
- Try Statement, 146

- Try!, 145
- Try?, 146
- Tuple, 120, 121, 122, 123, 146, 147, 148
- Tuple Definition, 121
- Tuples, 121, 123
- Tuples in error handling, 146
- Type alias, 93
- Type annotation, 11
- Type Conversion
  - Numbers, 26
- Type inference, 12

## U

- UInt, 25
- Unicode, 13, 32, 33, 38
- Universal Remote Locator. *See* URL
- Unwrapping optionals, 74
- UpperCamelCase, 82
- URL
  - Accessing components, 134
  - As file reference, 130
  - Components, 130
  - Declaration, 131
  - Definition, 130
  - Determining if file is reachable, 134
  - Methods, 133
  - Obtaining file extension from, 134
  - Obtaining file name from, 134
  - Referencing relative to Home directory, 133
  - Standardized, 133

## **V**

Value types, 86, 124

Variables, 12

Variadic parameter, 45, 118

Void, 120

## **W**

Wirth, Nicklaus, 1



## About the Author



Henry Lowe MD, FACMI is Emeritus Professor at Stanford University and a Board-Certified Internal Medicine physician. From 2002 until 2013 Dr. Lowe was Chief Information Officer and Senior Associate Dean for Information Resources and Technology at Stanford University School of Medicine. He was also the founding Director of Stanford's Center for Clinical Informatics. He has extensive experience in clinical informatics research and development. Dr. Lowe is an elected Fellow of the American College of Medical Informatics (ACMI) and the Founder of Ascriva Health Informatics, a company based in Silicon Valley, California.