We already have seen a piece of Swift program while setting up environment. Let’s start once again with the following **Hello, World!** program created for OS X playground, which includes `import Cocoa` as shown below:

```swift
import Cocoa

/* My first program in Swift */
var myString = "Hello, World!"
println(myString)
```

If you create same program for iOS playground, then it will include `import UIKit` and same program will look like as follows:

```swift
import UIKit

var myString = "Hello, World!"
println(myString)
```

When we run above program using appropriate playground, we will get the following result.

```
Hello, World!
```

Let us now see the basic structure of Swift program, so that it will be easy for you to understand basic building blocks of the Swift programming language.

**Import in Swift**

You can use `import` statement to import any Objective-C framework or library directly into Swift program. For example above `import cocoa` statement makes all Cocoa libraries, APIs, and runtimes that form the development layer for all of OS X, available in Swift.

Cocoa is implemented in Objective-C, which is a superset of C, so it is easy to mix C and even C++ into your Swift applications.

**Tokens in Swift**

A Swift program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol. For example, the following Swift statement consists of three tokens:

```swift
println("test!")
The individual tokens are:
println (    "test!"
)
```

**Comments**

Comments are like helping text in your Swift program and they are ignored by the compiler. Multi-line comment starts with `/*` and terminates with the characters `*/` as shown below:

```swift
/* My first program in Swift */
```

Multi-line comments can be nested in Swift. Following is a valid comment in Swift:

```swift
/* My first program in Swift is Hello, World! */
/* Where as second program is Hello, Swift! */
```
Single comment is written using // in the beginning of the comment.

// My first program in Swift

**Semicolons**

Swift does not require you to write a semicolon ; after each statement in your code, though its optional and if you use a semicolon then compiler does not complain about it.

But, if you are using multiple statements in the same line then its required to use a semicolon as a delimiter, otherwise compiler will raise syntax error. You can write above Hello, World! program as follows:

```swift
import Cocoa
/* My first program in Swift */
var myString = "Hello, World!"; println(myString)
```

**Identifiers**

A Swift identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore _ followed by zero or more letters, underscores, and digits 0 to 9.

Swift does not allow punctuation characters such as @, $, and % within identifiers. Swift is a case sensitive programming language. Thus Manpower and manpower are two different identifiers in Swift. Here are some examples of acceptable identifiers:

- Azad       zara    abc   move_name  a_123
- myname50   _temp   j    a23b9      retVal

To use a reserved word as an identifier, you will need to put a backtick ` before and after it. For example, `class` is not a valid identifier, but `class` is valid.

**Keywords**

The following keywords are reserved in Swift. These reserved words may not be used as constant or variable or any other identifier names, unless they're escaped with backticks:

**Keywords used in declarations**

- class       deinit     enum      extension
- func        import     init      internal
- let         operator   private   protocol
- public      static     struct    subscript
- typealias   var

**Keywords used in statements**

- break       case       continue   default
- do           else       fallthrough  for
- if           in         return     switch
- where        while
Keywords used in expressions and types

as   dynamicType   false   is
nil   self   Self   super
true   _COLUMN_   _FILE_   _FUNCTION_
_LINE_

Keywords used in particular contexts

associativity   convenience   dynamic   didSet
final   get   infix   inout
lazy   left   mutating   none
nonmutating   optional   override   postfix
precedence   prefix   Protocol   required
right   set   Type   unowned
weak   willSet

Whitespaces

A line containing only whitespace, possibly with a comment, is known as a blank line, and a Swift compiler totally ignores it.

Whitespace is the term used in Swift to describe blanks, tabs, newline characters and comments. Whitespace separates one part of a statement from another and enables the compiler to identify where one element in a statement, such as int, ends and the next element begins. Therefore, in the following statement:

```swift
var age
```

There must be at least one whitespace character usually as space between `var` and `age` for the compiler to be able to distinguish them. On the other hand, in the following statement:

```swift
int fruit = apples + oranges //get the total fruits
```

No whitespace characters are necessary between `fruit` and `=`, or between `=` and `apples`, although you are free to include some if you wish to use for readability purpose.

Literals

A literal is the source code representation of a value of an integer, floating-point number, or string type. The following are examples of literals:

```swift
92       // Integer literal
4.24159  // Floating-point literal
"Hello, World!"   // String literal
```