Swift - Strings

Strings in Swift 4 are an ordered collection of characters, such as "Hello, World!" and they are represented by the Swift 4 data type **String**, which in turn represents a collection of values of **Character** type.

## **Create a String**

You can create a String either by using a string literal or creating an instance of a String class as follows −

// String creation using String literal

var stringA = "Hello, Swift 4!"

print( stringA )

// String creation using String instance

var stringB = String("Hello, Swift 4!")

print( stringB )

//Multiple line string

let stringC = """

Hey this is a

example of multiple Line

string by tutorialsPoint

"""

print(stringC)

When the above code is compiled and executed, it produces the following result

Hello, Swift 4!

Hello, Swift 4!

Hey this is a

example of multiple Line

string by tutorialsPoint

## **Empty String**

You can create an empty String either by using an empty string literal or creating an instance of String class as shown below. You can also check whether a string is empty or not using the Boolean property **isEmpty**.

// Empty string creation using String literal

var stringA = ""

if stringA.isEmpty {

print( "stringA is empty" )

} else {

print( "stringA is not empty" )

}

// Empty string creation using String instance

let stringB = String()

if stringB.isEmpty {

print( "stringB is empty" )

} else {

print( "stringB is not empty" )

}

When the above code is compiled and executed, it produces the following result −

stringA is empty

stringB is empty

## **String Constants**

You can specify whether your String can be modified (or mutated) by assigning it to a variable, or it will be constant by assigning it to a constant using **let** keyword as shown below −

// stringA can be modified

var stringA = "Hello, Swift 4!"

stringA + = "--Readers--"

print( stringA )

// stringB can not be modified

let stringB = String("Hello, Swift 4!")

stringB + = "--Readers--"

print( stringB )

When the above code is compiled and executed, it produces the following result −

Playground execution failed: error: <EXPR>:10:1: error: 'String' is not

convertible to '@lvalue UInt8'

stringB + = "--Readers--"

## **String Interpolation**

String interpolation is a way to construct a new String value from a mix of constants, variables, literals, and expressions by including their values inside a string literal.

Each item (variable or constant) that you insert into the string literal is wrapped in a pair of parentheses, prefixed by a backslash. Here is a simple example −

var varA = 20

let constA = 100

var varC:Float = 20.0

var stringA = "\(varA) times \(constA) is equal to \(varC \* 100)"

print( stringA )

When the above code is compiled and executed, it produces the following result −

20 times 100 is equal to 2000.0

## **String Concatenation**

You can use the + operator to concatenate two strings or a string and a character, or two characters. Here is a simple example −

let constA = "Hello,"

let constB = "World!"

var stringA = constA + constB

print( stringA )

When the above code is compiled and executed, it produces the following result −

Hello,World!

## **String Length**

Swift 4 strings do not have a **length** property, but you can use the global count() function to count the number of characters in a string. Here is a simple example −

var varA = "Hello, Swift 4!"

print( "\(varA), length is \((varA.count))" )

When the above code is compiled and executed, it produces the following result −

Hello, Swift 4!, length is 15

## **String Comparison**

You can use the == operator to compare two strings variables or constants. Here is a simple example −

var varA = "Hello, Swift 4!"

var varB = "Hello, World!"

if varA == varB {

print( "\(varA) and \(varB) are equal" )

} else {

print( "\(varA) and \(varB) are not equal" )

}

When the above code is compiled and executed, it produces the following result −

Hello, Swift 4! and Hello, World! are not equal

## **String Iterating**

Strings are again a collection of values in swift 4, so we can iterate over string using loops. −

for chars in "ThisString" {

print(chars, terminator: " ")

}

When the above code is compiled and executed, it produces the following result −

T h i s S t r i n g

## **Unicode Strings**

You can access a UTF-8 and UTF-16 representation of a String by iterating over its utf8 and utf16 properties as demonstrated in the following example −

var unicodeString = "Dog???"

print("UTF-8 Codes: ")

for code in unicodeString.utf8 {

print("\(code) ")

}

print("\n")

print("UTF-16 Codes: ")

for code in unicodeString.utf16 {

print("\(code) ")

}

When the above code is compiled and executed, it produces the following result −

UTF-8 Codes:

68

111

103

63

63

63

UTF-16 Codes:

68

111

103

63

63

63

## **String Functions & Operators**

Swift 4 supports a wide range of methods and operators related to Strings −

|  |  |
| --- | --- |
| **Sr.No** | **Functions/Operators & Purpose** |
| 1 | **isEmpty**  A Boolean value that determines whether a string is empty or not. |
| 2 | **hasPrefix(prefix: String)**  Function to check whether a given parameter string exists as a prefix of the string or not. |
| 3 | **hasSuffix(suffix: String)**  Function to check whether a given parameter string exists as a suffix of the string or not. |
| 4 | **toInt()**  Function to convert numeric String value into Integer. |
| 5 | **count()**  Global function to count the number of Characters in a string. |
| 6 | **utf8**  Property to return a UTF-8 representation of a string. |
| 7 | **utf16**  Property to return a UTF-16 representation of a string. |
| 8 | **unicodeScalars**  Property to return a Unicode Scalar representation of a string. |
| 9 | **+**  Operator to concatenate two strings, or a string and a character, or two characters. |
| 10 | **+=**  Operator to append a string or character to an existing string. |
| 11 | **==**  Operator to determine the equality of two strings. |
| 12 | **<**  Operator to perform a lexicographical comparison to determine whether one string evaluates as less than another. |
| 13 | **startIndex**  To get the value at starting index of string. |
| 14 | **endIndex**  To get the value at ending index of string. |
| 15 | **Indices**  To access the indeces one by one. i.e all the characters of string one by one. |
| 16 | **insert("Value", at: position)**  To insert a value at a position. |
| 17 | **remove(at: position)**  **removeSubrange(range)**  to remove a value at a position, or to remove a range of values from string. |
| 18 | **reversed()**  returns the reverse of a string |