Swift - 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 −

42 // Integer literal

3.14159 // Floating-point literal

"Hello, world!" // String literal

## **Integer Literals**

An integer literal can be a decimal, binary, octal, or hexadecimal constant. Binary literals begin with 0b, octal literals begin with 0o, and hexadecimal literals begin with 0x and nothing for decimal.

Here are some examples of integer literals −

let decimalInteger = 17 // 17 in decimal notation

let binaryInteger = 0b10001 // 17 in binary notation

let octalInteger = 0o21 // 17 in octal notation

let hexadecimalInteger = 0x11 // 17 in hexadecimal notation

## **Floating-point Literals**

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or hexadecimal form.

Decimal floating-point literals consist of a sequence of decimal digits followed by either a decimal fraction, a decimal exponent, or both.

Hexadecimal floating-point literals consist of a 0x prefix, followed by an optional hexadecimal fraction, followed by a hexadecimal exponent.

Here are some examples of floating-point literals −

let decimalDouble = 12.1875

let exponentDouble = 1.21875e1

let hexadecimalDouble = 0xC.3p0

## **String Literals**

A string literal is a sequence of characters surrounded by double quotes, with the following form −

"characters"

String literals cannot contain an unescaped double quote ("), an unescaped backslash (\), a carriage return, or a line feed. Special characters can be included in string literals using the following escape sequences −

|  |  |
| --- | --- |
| **Escape sequence** | **Meaning** |
| \0 | Null Character |
| \\ | \character |
| \b | Backspace |
| \f | Form feed |
| \n | Newline |
| \r | Carriage return |
| \t | Horizontal tab |
| \v | Vertical tab |
| \' | Single Quote |
| \" | Double Quote |
| \000 | Octal number of one to three digits |
| \xhh... | Hexadecimal number of one or more digits |

The following example shows how to use a few string literals −

let stringL = "Hello\tWorld\n\nHello\'Swift 4\'"

print(stringL)

When we run the above program using playground, we get the following result −

Hello World

Hello'Swift 4'

## **Boolean Literals**

There are three Boolean literals and they are part of standard Swift 4 keywords −

* A value of **true** representing true.
* A value of **false** representing false.
* A value of **nil** representing no value.

## **Types of Swift Literals**

### **Integer Literals**

* **Binary Literals**
  + It is used to represent binary values.
  + It begins with 0b.
* **Octal Literals**
  + It is used to represent octal values.
  + It begins with 0o.
* **Hexadecimal Literals**
  + It is used to represent hexadecimal value.
  + It begins with 0x.
* **Decimal Literals**
  + It is used represent decimal value.
  + It begins with nothing. Everything you declare in integer literal is of type decimal.

### **Example of Integer Literal:**

1. **let** binaryNumber = 0b11111111
2. print(binaryNumber)
3. print(1231)

**Output:**

255

1231

The above example contains two integer literals 0b11111111 (binary literal) and 1231 (decimal literal). 255 is the decimal value of 11111111 that's why the print(binaryNumber) statement outputs 255 in the screen.

## **String & Character Swift literals**

A sequence of characters covered by double quotes is called **string literal** and a single character covered by double quotes is called **character literal**.

### **Example:**

1. **let** randomCharacter:Character = "C"
2. **let** randomString:String = "C is an awesome programming language"
3. print(randomCharacter)
4. print(randomString)

**Output:**

C

C is an awesome programming language

## **Floating Point Literals**

Floating point literals are used for **float** and **double** values. There are two types of floating point literals:

### **Decimal:**

It can store an optional exponent, indicated by an uppercase or lowercase **e**. For decimal numbers with an exponent of exp, the base number is multiplied by 10exp.

### **Example:**

1. **let** someFloat = 3.1416
2. **let** someAnotherFloat = 3.14e2
3. print(someFloat)
4. print(someAnotherFloat)

**Output:**

3.1416

314.0

### **Hexadecimal:**

Hexadecimal floats must contain an exponent, indicated by an uppercase or lowercase **p**. For hexadecimal numbers with an exponent of exp, the base number is multiplied by 2exp.

### **Example:**

1. **let** someFloat = 0xFp10
2. **let** someAnotherFloat = 0xFp-12
3. print(someFloat)
4. print(someAnotherFloat)

**Output:**

15360.0

0.003662109375

## **Boolean Literals**

There are two Boolean literals in Swift: **true** and **false**.

### **Example:**

1. **let** result1:Bool = **false**
2. **let** result2:Bool = **true**
3. print(result1)
4. print(result2)

**Output:**

false

true

## **Type Alias**

The **typealias** is used to create a new name for an existing type.

### **Syntax:**

1. **typealias** newname = type

### **Example:**

Let's take an example where we put "Raj" as another name for type Int.

1. **typealias** Raj = Int
2. **var** distance: Raj = 100
3. print(distance)

**Output:**

100

## **Type Safety**

Swift 4 is a type-safe language. If your code requires Int, then you can't use String. It performs type-checks when compiling your code and flags any mismatched types as errors.

### **Example:**

1. **var** varA = 12
2. varA = "Hello World!"
3. print(varA)

**Output:**

main.swift:2:8: error: cannot assign value of type 'String' to type 'Int'

varA = "Hello World!"

^~~~~~~~~~~~~~

You can see that the above program get a compile time error just because of type safety.

## **Type Inference**

Swift is a type inference language means when you compile the Swift code, it automatically check the type of value you provide. It automatically choose the appropriate data type for successful execution.