APPLE DEVELOPERS GROUP
ADG MANIPAL

iOS Dev 101 : Swift Basics

Swift is a programming language for iOS, macOS, watchOS and tvOS application development. Let's jump right into the basics:

Variables & Constants

When building an application, you will often be dealing with some sort of data, or information, which needs to be passed from one part of the application to another. This is done through the use of variables and constants.
Constants and variables associate a name with a value of a particular type (such as the number 10 or the string "Hello"). The value of a *constant* can't be changed once it is set, whereas a *variable* can be set to a different value in the future.

In Swift, you can simply define a variable by providing its initial value:

```
var myDecimal = 3.14 // This is a float(decimal) typed variable
var myVariable = "This is a string typed variable"
var myInteger = 10 // This is an integer-typed variable
```

In Swift, constants are defined using the keyword 'let':

```
let maxCount = 10
```

You can declare multiple constants or multiple variables on a single line, separated by commas:

```
var x = 0.0, y = 0.0, z = 0.0
```

Type Annotations

You can provide a *type annotation* when you declare a constant or variable, to be clear about the kind of values the constant or variable can store. Write a type annotation by placing a colon after the constant or variable name, followed by a space, followed by the name of the type to use.

The following example shows the declaration of a variable with it data type:

```
var welcomeMessage: String
```

A value can be assigned to this variable later at any stage during the program.

Data types

In Swift, there are a number of different data types. They are:

Numeric data types

- Integer. These store whole numbers, such as 5, -42, or 32000.
- Float. These store decimal numbers, such as 3.1415 or -2.76.

Alphanumeric data types

- Character. A single alphanumeric character, for example ‘a’ or ‘6' (note the quotation marks).

- String. A string is a collection of characters, for example ‘Hello world.’ String variables can have a number of operations done to them such as appending, pretending, and concatenation, but they cannot be used in arithmetic operations.

Other data types

- Boolean. This is a TRUE/FALSE data type, and can only contain one of those two values. (0 and 1 convention is not allowed)
- Array. An array is a collection of independent pieces of data. For example, you might have an array of strings that make up a collection of fruits:

  ```
  var fruits = [‘apple’, ‘banana’, ‘orange’]
  ```

Or, you might store the progression of the Fibonacci sequence within an array:

```
let fibbionaci = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

Arrays are really useful for moving collections of data around your application. For example, you might create an array that has a list of RSS feeds in your application.

- Dictionary. A dictionary is a special type of array in Swift that helps index the data.

  Syntax:

  ```
  var variableName = [ data : index, data : index , …]
  ```

  An example of a dictionary is given in The Swift Programming Language:

```
var occupations = [“Malcolm”:“Captain”, “Kaylee”:“Mechanic”,  “Jayne”:
“PublicRelations”]
```

Printing Constants and Variables

You can print the current value of a constant or variable with the print(_:separator:terminator:) function:

```
print(friendlyWelcome)
```

Swift uses *string interpolation* to include the name of a constant or variable as a placeholder in a longer string, and to prompt Swift to replace it with the current value of that constant or variable. Wrap the name in parentheses and escape it with a backslash before the opening parenthesis:

```
print("The current value of friendlyWelcome is \(friendlyWelcome)")
// Prints "The current value of friendlyWelcome is Bonjour!"
```

Type Safety and Type Inference

Swift is a *type-safe* language. A type safe language encourages you to be clear about the types of values your code can work with. If part of your code requires a String, you can’t pass it an Int by mistake.

Because Swift is type safe, it performs *type checks* when compiling your code and flags any mismatched types as errors. This enables you to catch and fix errors as early as possible in the development process.

Type-checking helps you avoid errors when you’re working with different types of values. However, this doesn’t mean that you

have to specify the type of every constant and variable that you declare. If you don't specify the type of value you need, Swift uses *type inference* to work out the appropriate type. Type inference enables a compiler to deduce the type of a particular expression automatically when it compiles your code, simply by examining the values you provide.

Because of type inference, Swift requires far fewer type declarations than languages such as C or Objective-C. Constants and variables are still explicitly typed, but much of the work of specifying their type is done for you.

Type inference is particularly useful when you declare a constant or variable with an initial value. This is often done by assigning a *literal value* (or *literal*) to the constant or variable at the point that you declare it. (A literal value is a value that appears directly in your source code, such as 42 and 3.14159 in the examples below.)

For example, if you assign a literal value of 42 to a new constant without saying what type it is, Swift infers that you want the constant to be an Int, because you have initialized it with a number that looks like an integer:

```
1.let meaningOfLife = 42
2.// meaningOfLife is inferred to be of type Int
```

Likewise, if you don't specify a type for a floating-point literal, Swift infers that you want to create a Double:

```
1.let pi = 3.14159
2.// pi is inferred to be of type Double
```

Swift always chooses Double (rather than Float) when inferring the type of floating-point numbers.

If you combine integer and floating-point literals in an expression, a type of Double will be inferred from the context:

```
1.let anotherPi = 3 + 0.14159
2.// anotherPi is also inferred to be of type Double
```

The literal value of 3 has no explicit type in and of itself, and so an appropriate output type of Double is inferred from the presence of a floating-point literal as part of the addition.

Conditionals and loops (control flow)

There are times when you are putting together an application when the logic you are building is dependent on either the state of a variable or constant, or needs to loop through a collection of data, performing the same task multiple times.

if/ else/ switch

'If,' 'else,' 'else if', and 'switch' are used for examining data and running a set of instructions based on the state of that data. The syntax in Swift is:

```
let variableName = value
if variableName < 1.0 {
        //perform operations and print statements
}
```

For longer or complicated conditionals, it's recommended to use switch instead of else, for reasons of both performance and code legibility. This looks something like the following:

```
switch let variableName {
case 0:
```

```
// perform required operations

case 1:
// do something else.

case 3:
// a minor change

default:
// some default condition
}
```

for/while

The conditionals ‘for’, ‘for-in’, and ‘while’, and ‘do-while’ are used for looping through sets of data. This saves you a lot of time coding when you know you will be running a repetitive task against a group of data.

The ‘for’ conditional is used when you know your range, or are working with a known set of data. For example, from the Swift language documentation:

```
let individualScores = [75, 43, 103, 87, 12]
var teamScore = 0
for score in individualScores {
        if score > 50 {
                teamScore += 3
        }else {
                teamScore += 1
        }
}
print(teamScore) // teamscore is 11.
```

The ‘while’ conditional is used for as long as a given condition is true. For example, from the Swift documentation:

```
var square = 0
var diceRoll = 0
while square < finalSquare {
        // roll the dice
        if ++diceRoll == 7 { diceRoll = 1 }
        // move by the rolled amount
        square += diceRoll
        if square < board.count {
                // if we are still on the board, move up or down for a snake or a
ladder
                square += board[square]
        }
}
println("Game over!")
```

Functions

An application is, generally, actually a collection of several smaller applications, called functions. Functions are mini-programs that live within your application to perform very specific tasks. For example, in your email client, a function exists that connects to your mail server and asks if there are any new messages. If there are, it will pass the work on to a different function that downloads the mail to your computer. Yet another function exists that displays the new messages in your mail client, so you can read them.

In Swift, it is mandatory that you specify the types of both the parameters and the return values. An example of a function:

```
func functionName(variable1: dataType, variable2: dataType, …) -> returnType {
        // Perform some operation
}
```