

# Swift

## Introduction to Swift

Swift is C-influenced but not a C-superset, like Objective-C.

Like in Python, the use of rounded parenthesis is not essential to swift. It is just for readability.

To learn Swift syntax, the Swift standard library and Foundation you need to know nothing about iOS or macOS programming

Chris Lattner is the first contributor to several Apple projects like Swift, Clang (the LLVM compiler) and Xcode

## Coding Style

CamelCase: programming style where strings obtained by chaining multiple words have all the words but the first one beginning with a capital letter.

Swift line codes doesn't require semicolons. They are allowed but not needed and the standard practice is not to use semicolons at all

## Where is the main entry-point ?

All swift files in an app are order-independent.

For example, you can define a enum type in file a.swift

```
enum Direction {  
    case north  
    case south  
    case west  
    case est  
}
```

and use it in another file b.swift

```
var dir = Direction.north
```

In swift a Top-level code is any swift executable statement not written within a function or class body.

Top-level code is not allowed in most of swift source files. The exception is a special file named "main.swift", which behaves much like a playground file, but is built with your app's source code.

Swift doesn't require a main method or function to say where the execution starts.

With "main.swift" you can take complete control over initialization. In effect, the first line of code to run in "main.swift" is implicitly defined as the main entry-point for the program.

If you add @UIApplicationMain to a regular Swift file, this causes the compiler to synthesize a main entry point for you, and eliminates the need for a "main.swift" file.

This is what happens when you start with a iOS template project.

## Exercise

1. Create a new project in Xcode
2. Comment out the line @UIApplicationMain in AppDelegate.swift
3. Add a new source file and call it main.swift, and copy the following code inside

```
9  import UIKit  
10  
11  print ("I'm using my own main")  
12  UIApplicationMain(0, nil, nil, NSStringFromClass(AppDelegate.self))|
```

## Playground

Playground is a valuable tool for quickly writing and testing swift code.

Playground files allow top-level code execution.  
Code within a playground file is order-dependent.  
Playground contain three areas

- code editor
- results
- timeline

## Useful buttons in playground



This is the quick Look and it is an API

All variables are declared with keyword `var`

Type inference means that swift infers the type of a variable from code we have written

Type of a variable can be defined by value assignment

```
var myVariable = "Hello"
```

or by type annotation

```
var myVariable : String
```

We cannot change the type of a variable

Types are `Int`, `Float`, `Double`, `Bool`, `String`, `Character`.

`Float` represents a 32-bit floating-point number

`Double` represents a 64-bit floating-point number.

Other valid types are `Int8`, `Int16`, `Int32`, `Int64`, `UInt`, `UInt8`, `UInt16`,...

### NOTE

Use `UInt` only when you specifically need an unsigned integer type with the same size as the platform's native word size. If this is not the case, `Int` is preferred, even when the values to be stored are known to be non-negative. A consistent use of `Int` for integer values aids code interoperability, avoids the need to convert between different number types, and matches integer type inference, as described in [Type Safety](#) and [Type Inference](#).

### NOTE

Type aliases are useful when you want to refer to an existing type by a name that is contextually more appropriate, such as when working with data of a specific size from an external source:

```
typealias AudioSample = UInt16
```

All constants are declared with keyword `let`

The value of a constant can be assigned only once. After that it cannot change anymore

In Objective-C a string variable had to be declare of type `NSString`, while a string constant had to be declared of type `NSString`. In Swift we use `var` and `let`, both for basic types and for more complex structures.

## The print function

The `print()` function is a global function that prints one or more values to an appropriate output. In Xcode, for example, the `print(_:separator:terminator:)` function prints its output in

Xcode's console. The separator and terminator parameter have default values, so you can omit them when you call this function (because it is a variadic function). By default, the function terminates the line it prints by adding a line break. To print a value without a line break after it, pass an empty string as the terminator—for example, `print(someValue, terminator: "")`.

```
print("cip", terminator: "")
print("ciop", terminator: "")
```

prints "cipciop"

The separator parameter works as follows

```
print("1", "2", "ciao", "a", separator: "-")
```

prints "1-2-ciao-a"

In many languages, in order to create strings with program dependent contents we use Concatenation

"Congratulation " + name + "." + CRLF + "Your score is " + score.toString()



or string Formatting

("Congratulation %@. Your score is %i", name, score)

In swift we use Interpolation, with the notation `\( .. )`

`print("Congratulation \(name). Your score is \(score + 12)")`

## UTF (Unicode Transformation Format)

<pre>var circledi: Character = "\u{24D8}" var circledà: Character = "\u{e0}\u{20dd}"</pre>	 
<pre>let caffè = "caff\u{E8}" let caffèCorretto = "caffe" + "\u{300}"</pre>	"caffè" "caffè"
<pre>print(caffè.count)</pre>	"5\n"
<pre>print(caffè.utf16.count)</pre>	"5\n"
<pre>print(caffèCorretto.count)</pre>	"5\n"
<pre>print(caffèCorretto.utf16.count)</pre>	"6\n"

Some fonts have more than one glyphs for some of their characters.

In Swift all strings can also be treated as an array of characters.

However they also have the property `.utf16`

This is because Swift's strings natively store international characters as individual characters, e.g. the letter "é" is stored as precisely that (like `caffè` constant above). However, UIKit was written in Objective-C before Swift's strings came along, and it uses a different character system called UTF-16 where the accent and the letter are stored separately (like `caffèCorretto`).

It's a subtle difference, and often it isn't a difference at all, but it's becoming increasingly problematic because of the rise of emoji – those little images that are frequently used in messages. Emoji are actually just special character combinations behind the scenes, and they are measured differently with Swift strings and UTF-16 strings: Swift strings count them as 1 letter strings, but UTF-16 considers them to be 2 letter strings. This means if you use `.count` with UIKit methods, you run the risk of miscounting the string length.

Let me try to give you a simple rule: when you're working with UIKit, SpriteKit, or any other Apple framework, use `.utf16.count` for the character count. If it's just your own code – i.e. looping over characters and processing each one individually – then use `.count` instead.

## Typecasting

For type conversion use builtin functions `Int()`, `Double()`, `String()`, ..

## Tuples

Tuples are ordered set of value of any type. They are very useful as return values of function. For example a HTTP status code is returned by a HTTP server whenever you request a web page. This could be given with the following tuple

```
let http404Error = (404, "Not Found")
```

```
..
```

```
let (statusCode, statusMessage) = http404Error
```

Then you can access the single parts of a tuple

```
print("\(statusCode)")
```

```
or
```

```
print(http404Error.0)
```

NOTE

Tuples are useful for temporary groups of related values. They are not suited to the creation of complex data structures. If your data structure is likely to persist beyond a temporary scope, model it as a class or structure, rather than as a tuple. For more information, see

[Classes and Structures](#).

## Optionals and Unwrapping

You use *optionals* in situations where a value may be absent.

Here's an example of how optionals can be used to cope with the absence of a value. Swift's `Int` type has an initializer which tries to convert a `String` value into an `Int` value. However, not every string can be converted into an integer.

The example below uses the initializer to try to convert a `String` into an `Int`:

```
let possibleNumber = "123"
```

```
let convertedNumber = Int(possibleNumber) // convertedNumber is inferred to be of type
"Int?", or "optional Int"
```

Once you're sure that the optional *does* contain a value, you can access its underlying value by adding an exclamation mark (!) to the end of the optional's name. The exclamation mark effectively says, "I know that this optional definitely has a value; please use it." This is known as *forced unwrapping* of the optional's value.

You use *optional binding* to find out whether an optional contains a value, and if so, to make that value available as a temporary constant or variable. Optional binding can be used with `if` and `while` statements to check for a value inside an optional, and to extract that value into a constant or variable, as part of a single action.

Example:

```
if let newvar = optionalvar {
    //use newvar with the unwrapped value of optionalvar
}
```

When it is clear from a program's structure that an optional will *always* have a value, after that value is first set, it is useful to remove the need to check and unwrap the optional's value every time it is accessed. In this cases use an *implicitly unwrapped optional* by placing an exclamation mark after the optional type.

Example:

```
@IBOutlet var mmap: MKMapView!
```

## Error Handling

Swift has powerful means to detect, propagate and evaluate error conditions.

Example:

```
func makeASandwich() throws {
```

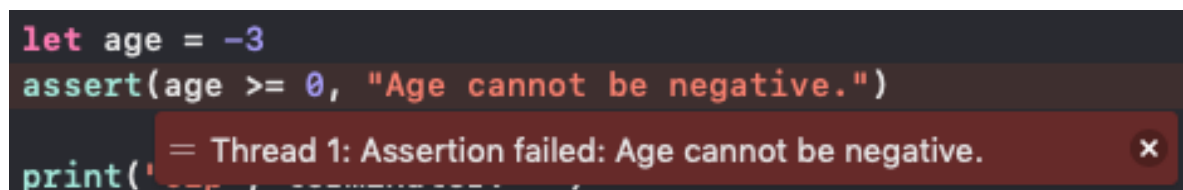
```
// ...
}

do {
    try makeASandwich()
    eatASandwich()
} catch Error.OutOfCleanDishes {
    washDishes()
} catch Error.MissingIngredients(let ingredients) {
    buyGroceries(ingredients)
}
```

## Using assertions

An assertion is a way to test a variable or expression is true and terminate the app with a message when it is found false.

Example:



Use an assertion whenever a condition has the potential to be false, but must *definitely* be true in order for your code to continue execution. Suitable scenarios for an assertion check include:

- An integer subscript index is passed to a custom subscript implementation, but the subscript index value could be too low or too high.
- A value is passed to a function, but an invalid value means that the function cannot fulfil its task.
- An optional value is currently nil, but a non-nil value is essential for subsequent code to execute successfully.

### NOTE

Assertions cause your app to terminate and are not a substitute for designing your code in such a way that invalid conditions are unlikely to arise. Nonetheless, in situations where invalid conditions are possible, an assertion is an effective way to ensure that such conditions are highlighted and noticed during development, before your app is published.

### NOTE

The advantage to assertions versus `print()` is that their check code is never executed in a live app. When you build a release version of your app – i.e., when you ship your app to the App Store – Xcode automatically disables your assertions so they won't reach your users.

## Switch

In Swift the round braces around the condition statement is not necessary, instead the curly braces around the action statements are mandatory, even you have a single line instruction

The statement

```
if x { .. }
```

works in C even though `x` is an integer value. In Swift it is not valid because the conditional expression must be a Bool value

In Swift there is not the automatic fall through behaviour in switch statement.

For example, in C you can write

```
switch myVar {
    case 1:
    case 2:
    case 3:
```

```

        x = 1
    break;
    case 4:
        x = 21
    break;
    default:
    break;
}

```

You can do this because the break keyword holds.

In swift you must write explicitly a statement for each condition

```

switch myVar {
    case 1:
        x = 1
    case 2:
        x = 1
    case 3:
        x = 1
    case 4:
        x = 21
    default:
        break
}

```

You can also use the range operator

```

switch myVar {
    case 1...3:
        x = 1
    case 4:
        x = 21
    default:
        break
}

```

The swift for loop uses the syntax

for each-item in some-collection

```

    // use each-item
{

```

Example

```

for k in 1..>100 {
    print("current k is \(k)")
}

```

Example

```

let myName = "Gabriele"
for eachChar in myName {
    print("next letter in name is current \(eachChar)")
}

```

## Functions

Function with no arguments and no return declaration is

```

func myFunction() {
    print("Hello")
}

```

//Function call

```
myFunction()
```

A function has a type. The function above has type `() -> ()`

Function with one argument and no return declaration is

```
func myFunction( name : String ) {  
    println("Hello \(name)")  
}
```

The function type is `(String) -> ()`

Function call

```
myFunction("Jane")
```

In Swift, function parameters are considered constants, so you cannot change them, unless you add keyword `var` in front of the parameter inside the function parameters list

```
func myFunction( name : String, var age : Int ) {  
    print("Hello \(name). Next year you will be \(age + 1)" years old)  
}
```

//Function call

```
myFunction("Jane", 21)
```

Add a return to the function:

```
func myFunction( name : String, var age : Int ) -> String {  
    return "Hello \(name). Next year you will be \(age + 1)" years old)  
}
```

Function type is `(String, Int) -> String`

Example:

```
func COSTestExec( name : String, var datarate : Int ) -> Bool {  
    switch name {  
        case "ergo":  
            ...  
        case "dmc":  
            ...  
        default:  
            break  
    }  
    return False  
}
```

`(Bool)COSTestExec:(String *)name`

You can assign a function to a variable of the same type.

```

5 func myFunc (employee emp: String, age: Int?) -> Bool {
6     if let a = age {
7         print("the age of " + emp + " is \(a)")
8         return true
9     }
10    return false
11 }
12
13 var myVar : (String, Int?) -> Bool
14 myVar = myFunc
15 myVar("Pippo", nil)
16 myVar("Gabriele", 48)

```

- Variadic parameters: You can have function with an arbitrary number of parameters

```

5 func myFunc (myParam par : Int, words : String...) {
6     print (words.count)
7 }
8
9 myFunc(myParam: 22, words: "cippa", "ciuppa", "ciappa")

```

A *variadic function* is a function that accepts any number of parameters.  
The system `print()` is a variadic function.

- Existentials: you can combine class type and protocols to create new types

```

class Appliance { .. }
class Hairdryer: Appliance { .. }
protocol CanCook { .. }

```

```

class Oven: Appliance, CanCook { .. }

```

and define function argument types

```

func makeDinner (using: Appliance & CanCook) { .. }

```

The synthetic way to write a method is to list the external parameter names followed by :  
For example:

```

func image(_ image: UIImage, didFinishSavingWithError error: Error?, contextInfo:
UnsafeRawPointer) { ... }

```

is written

```

image(_:didFinishSavingWithError:contextInfo:)

```

## Type cast

A as B, returns A with type B

A as! B, returns A with type B, forced unwrapped

as?

is

## Any and AnyObject

Any is an alias for any type.

```

Var cassetto = [Any]()

```

è un array di qualsiasi tipo



```

7   var arr = [Any]()
8   arr.append("Hello")
9   arr.append(12)
10  arr.append(0.22)

```

AnyObject is an alias for any object

## For loop

```

for var k in 0...10 {
    print("\(k)")
}

```

## Timer

```

import UIKit

class ViewController: UIViewController {

    var time = 0

    func result () {
        time = time + 1
        print("\(time)")
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        var timer = NSTimer()
        timer = NSTimer.scheduledTimerWithTimeInterval(1, target: self, selector: #selector(ViewController.result),
            userInfo: nil, repeats: true)
    }
}

```

The stopwatch app

```

class ViewController: UIViewController {

    var time = 0
    var timer = NSTimer()

    @IBOutlet var timerLabel: UILabel!

    func increaseTimer (sender: AnyObject) {
        time += 1
        self.timerLabel.text = String(time)
    }

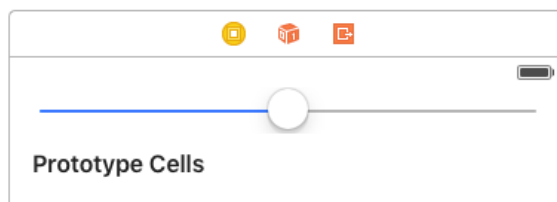
    @IBAction func play(sender: AnyObject) {
        if timer.valid {
            return
        }
        timer = NSTimer.scheduledTimerWithTimeInterval(1, target: self,
            selector: #selector(ViewController.increaseTimer), userInfo: nil,
            repeats: true)
    }

    @IBAction func stop(sender: AnyObject) {
        timer.invalidate()
        time = 0
        timerLabel.text = "0"
    }

    @IBAction func pause(sender: AnyObject) {
        timer.invalidate()
    }
}

```

## Slider



```

8
9 import UIKit
10
11 class ViewController: UIViewController,
12     UITableViewDelegate {
13     @IBOutlet var sliderValue: UISlider!
14     @IBAction func sliderMoved(sender: AnyObject) {
15         print(self.sliderValue)
16     }
17     override func viewDidLoad() {

```

## Tip for quick search of class or protocol methods

to find the method prototype of a protocol, just Cmd+click on protocol name,

```

class ViewController: UIViewController,
    UITableViewDelegate {
    @IBOutlet var sliderValue: UISlider!

```

find and copy the desired methods

```

@available(iOS 2.0, *)
public func tableView(tableView: UITableView,
    numberOfRowsInSection section: Int) -> Int

```

then click the left arrow on top bar to go back, and paste



## How to automatically update cells of a TableView

Simply create an outlet "myTable" by Ctrl+click and drag the TableView from storyboard to ViewController.swift, then call  
myTable.reloadData()

Note: sometimes you fill an empty cello of a TV and then reload the TV. A better approach is using myTable.insertRows() lets us tell the table view that a new row has been placed at a specific place in the array so that it can animate the new cell appearing. Adding one cell is also significantly easier than having to reload everything! We do that in Project5.

## Controlling Keyboard

It is possible to get control of the sw keyboard. For example suppose we want the keyboard disappears when tapping the active view and/or the return button

```
class ViewController: UIViewController, UITextFieldDelegate {

    @IBOutlet var myTextField: UITextField!
    @IBOutlet weak var myLabel: UILabel!
    @IBAction func myHereButton (sender: AnyObject) {

        self.myLabel.text = myTextField.text
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        self.myTextField.delegate = self
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    override func touchesBegan(touches: Set<UITouch>, withEvent event: UIEvent?) {
        self.view.endEditing(true)
    }

    func textFieldShouldReturn(textField: UITextField) -> Bool {
        myTextField.resignFirstResponder()
        return true
    }
}
```

## Scope of variables

If you want a variable be accessible to all methods of a class, declare it within the class body but outside any method

```
import UIKit

class FirstViewController: UIViewController {

    var classScopeVar = 0
}
```

If the variable has to be accessed to all project's classes, then declare it outside any class body

```
import UIKit

var projectScopeVar = 0

class FirstViewController: UIViewController {
```

## Update the view content

Use the method `viewDidAppear` to update the content of the view every time the view appears, not just when it is created

```
class FirstViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {

    @IBOutlet var toDoListTable: UITableView!

    override func viewDidAppear(animated: Bool) {
        toDoListTable.reloadData()
    }
}
```

## Add the delete function by swipe-left to the cell editing

To add the delete function by swipe left to the table view cell editing use the following method of the `UITableViewDelegate` protocol

```
//to add the remove function by left swipe
func tableView(tableView: UITableView, commitEditingStyle editingStyle: UITableViewCellEditingStyle, forRowAtIndexPath: NSIndexPath) {
    if editingStyle == UITableViewCellEditingStyle.Delete {
        //add here what to do when deleted
    }
}
```

## String manipulation

To cycle over all string characters

```
var str = "This is a String"
for c in str.characters {
    print(c)
}
```

get substrings

<pre>var stringa: String = "vaffanculo" var s = NSString(string: stringa) s.substringFromIndex(3) s.substringToIndex(5) s.substringWithRange(NSRange(location: 6, length: 4))</pre>	<pre>"vaffanculo" "vaffanculo" "fanculo" "vaffa" "culo"</pre>
---	---

others

<pre>aString.componentsSeparatedByString(" ") aString.uppercaseString aString.lowercaseString</pre>	<pre>["Ciao", "il", "mio", "nome", "è", "Gabriele"] "CAIO IL MIO NOME È GABRIELE" "ciao il mio nome è gabriele"</pre>
---	---

## Download web contents

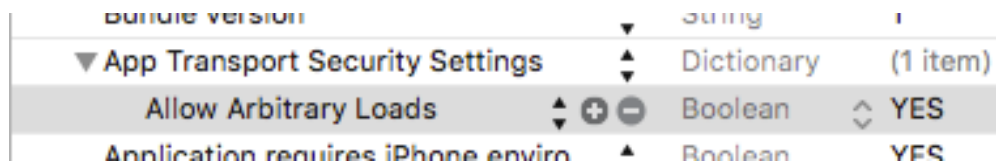
The simplest way to display content from a web site

```
13 @IBOutlet var webView: UIWebView!
14 override func viewDidLoad() {
15     super.viewDidLoad()
16
17     //define a url string. Because NSURL converter returns an optional we decide to unwrap it right now
18     let url = NSURL(string: "https://www.stackoverflow.com")!
19
20     //the simplest way to display the web content in the web view
21     webView.loadRequest(NSURLRequest(URL: url))
```

A more sophisticated way consists in downloading the binary data from the web site, convert it in UTF8 html and then display in in the web view, all that executed within a task

```
13 @IBOutlet var webView: UIWebView!
14 override func viewDidLoad() {
15     super.viewDidLoad()
16
17     //define a url string. Because NSURL converter returns an optional we decide to unwrap it right now
18     let url = NSURL(string: "https://www.stackoverflow.com")!
19
20     //the simplest way to display the web content in the web view
21     //webView.loadRequest(NSURLRequest(URL: url))
22
23     //fetch the url data content by creating a virtual browser task
24     let task = NSURLSession.sharedSession().dataTaskWithURL(url) { (data, response, error) in
25         //this code (closure) will happen when task completes
26         if let urlContent = data {
27             //convert the binary url content in a UTF readable format (html)
28             let webContent = NSString(data: urlContent, encoding: NSUTF8StringEncoding)
29             //print(webContent)
30
31             //display the web content in a web page
32             dispatch_async(dispatch_get_main_queue(), {
33                 self.webView.loadHTMLString(String(webContent), baseURL: nil)
34             })
35         }
36         else if let urlError = error {
37             print(urlError)
38         }
39     }
40
41     //run the task
42     task.resume()
43 }
```

Before running the app, we need to add the following item to the info.plist file



In order to isolate the substring "my substring" from the string "what precedes my substring and what follows it"

1. define an array of string myArray
2. call method myArray.componentsSeparatedByString("precedes ")
3. pick myArray[1]
4. define an array of string anotherArray = myArray[1]
5. call method anotherArray.componentsSeparatedByString(" and")
6. pick anotherArray[1]

Example:

```
var webSiteArray = [NSString]()
webSiteArray = webContent!.componentsSeparatedByString("3 Day Weather Forecast Summary:</b><span class=
\"read-more-small\"><span class=\"read-more-content\"> <span class=\"phrase\">")

if webSiteArray.count > 1 {
    let weatherArray = webSiteArray[1].componentsSeparatedByString("</span>")
    if weatherArray.count > 1 {
        let weatherSummary = weatherArray[0].stringByReplacingOccurrencesOfString("&deg;", withString: "°")
        dispatch_async(dispatch_get_main_queue(), { () -> Void in
            self.resultLabel.text = weatherSummary
            wasSuccessful = true
        })
    }
}
```

## GIF Animations

A lot of GIF animations can be downloaded for free from

<http://www.heathersanimations.com>

Save one of them on mac.

Install GIMP and the export plug-in.

Open the GIF file with GIMP and export the single frames of the animation as frameN.bmp, N=1,2,...

Import the frames in Assets.xcassets and then load the first one in a UIImageView

## Animate the UIImage with the frames of a animated GIF

To reproduce a GIF animation we can change the content of a UIImageView on time basis, making use of a timer.

```
11 class ViewController: UIViewController {
12     var i = 0
13     var timer = NSTimer()
14     @IBOutlet var alienImage: UIImageView!
15     @IBAction func animateAlien(sender: AnyObject) {
16         if timer.valid {
17             timer.invalidate()
18         } else {
19             timer = NSTimer.scheduledTimerWithTimeInterval(0.2, target: self, selector:
20                 #selector(ViewController.updateAnimation), userInfo: nil, repeats: true)
21         }
22     }
23
24     func updateAnimation () {
25         alienImage.image = UIImage(named: "frame\(i % 5 + 1).png")
26         i = i+1
27     }
28 }
```

## Adding animation at launch time

To create animation at launch time we can override two methods of the UIViewController class,

viewDidLoadLayoutSubviews and viewWillAppear

## Shifting a image in the screen

Suppose we want the initial scene appearing from one side of the screen (top, bottom, left, right).

```
// this method is called right before the subviews are displayed
override func viewDidLoadLayoutSubviews() {
    alienImage.center = CGPoint(x: alienImage.center.x - 400, y: alienImage.center.y)
}

// this method is called right after the subviews are displayed
override func viewWillAppear(animated: Bool) {
    UIView.animateWithDuration(1.0) {
        self.alienImage.center = CGPointMake(self.alienImage.center.x + 400, self.
        alienImage.center.y)
    }
}
```

In the example above the image centre is shifted to the left right before it is displayed, then it is shifted to the right of the same amount right after it is displayed, and the transition is made 1 second lasting. The effect is that the alien image move into the scene from the left when the app is launched.

## Fading

If we want our image to fade in when the application launches, we proceed as described above, this time leveraging the alpha property of the UIImage

```

override func viewDidLoadSubviews() {
    alienImage.alpha = 0
}

override func viewWillAppear(animated: Bool) {
    UIView.animateWithDuration(1.0) {
        self.alienImage.alpha = 1.0
    }
}

```

## Moving and scaling

This is another effect we can use. The image appears in a moving rectangle frame

```

override func viewDidLoadSubviews() {
    alienImage.frame = CGRectMake(100, 20, 0, 0)
}

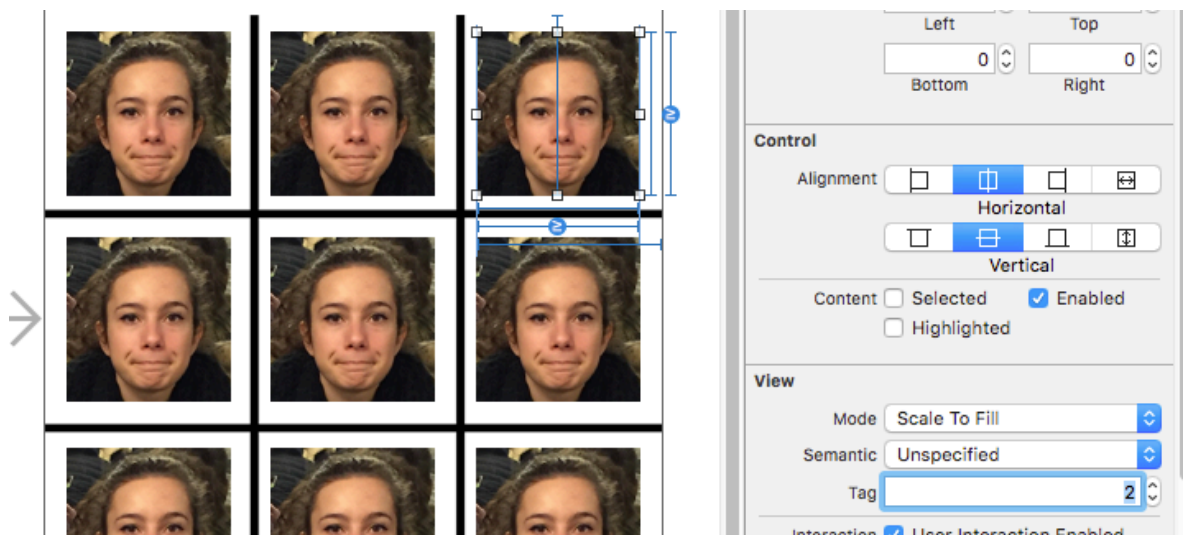
override func viewWillAppear(animated: Bool) {
    UIView.animateWithDuration(6.0) {
        self.alienImage.frame = CGRectMake(100, 20, 150, 300)
    }
}

```

## Using tags

Every view object contained in a screen has a integer that you can change in order to identify the object from code.

In this case we have 9 buttons within the main view and we have assigned them the tags 0 to 8. We also take care to assign a different tag (e.g. 10) to all other objects.



Another example is the xylophone application, in which we have 7 buttons all connected to the same IBAction. They have assigned a different tag, in such a way we'll be able to emit a different note when tapped





## Using maps

To add a map pointing in a specific location

1. add a Map Kit View object to the storyboard
2. import UIMapKit in ViewController.swift and add the MKMapViewDelegate protocol to the class definition
3. Create an outlet for the map view in the ViewController.swift
4. Point the map on a specific location. To do that declare the following constants and then call the setRegion method

```

17     override func viewDidLoad() {
18         super.viewDidLoad()
19         let latitude: CLLocationDegrees = 41.688959
20         let longitude: CLLocationDegrees = 12.642983
21         let location: CLLocationCoordinate2D = CLLocationCoordinate2DMake(latitude, longitude)
22
23         let latDelta: CLLocationDegrees = 0.001
24         let lonDelta: CLLocationDegrees = 0.001
25         let span: MKCoordinateSpan = MKCoordinateSpanMake(latDelta, lonDelta)
26
27         let region: MKCoordinateRegion = MKCoordinateRegionMake(location, span)
28         map.setRegion(region, animated: true)
29     }

```

## Using a dictionary

To declare a empty dictionary for storing String:String key-value pairs

```

11 var places = [Dictionary<String,String>()]

```

To append a new triple of pairs:

```

21 places.append(["Name":"Taj Mahal","Lat":"27.175277","Lon":"70.042128"])

```

or, in another example,

```

//add a new item to the list of memorable places
places.append(["Name":title,"Lat":"\(coordinateOnMap.latitude)","Lon":"\(coordinateOnMap.longitude)"])

```

To access the value string associated to the "Name" key:

```

43 cell.textLabel?.text = places[indexPath.row]["Name"]

```



## Making the code compatible with different iOS versions

Maybe that some versions of iOS support different versions of Swift.

Swift 3.0 is a major release that is *not* source-compatible with Swift 2.2.

In these cases you could use the `#available` in an `if` or `guard` to protect code that should only be run on certain systems

The correct way to proceed is this

```
48         if #available(iOS 10.0, *) {
49             print("we are in iOS 10.0 or greater")
50         } else {
51             print("we are not yet in iOS 10.0")
52         }
```

Using guard statement

With the guard statement you can avoid nesting multiple if-else statements.

```
override func viewWillAppear(_ animated: Bool) {
    guard let hexColor = selectedCategory?.color else {return}
    guard let categoryName = selectedCategory?.name else {return}
    guard let navBar = navigationController?.navigationBar else {return}
    title = categoryName
    navBar.barTintColor = UIColor(hexString: hexColor)
}
```

## Bug reporting

If you see other errors, file a bug report at <https://bugreport.apple.com>, including details

## Swift Migration Assistant

To make a Swift 2.2 project a Swift 3 project: Edit>Convert>ToCurrentSwiftSyntax..

Eventually you will be presented with all the changes. The changes will be applied once you click on 'Save'.

Warning: some changes may not be suggested automatically, for example NSURL into URL

## Running Swift from terminal

One possibility is to use REPL (Read Evaluate Print Loop)

1. Open `.bash_profile` and add the line `swift="/Applications/<Xcode_version>.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/swift"`
2. Open a terminal and type `swift`. Then enter instructions

```
gfmacmini:~ gabrielefilosofi$ swift
Welcome to Apple Swift version 2.1.1 (swiftlang-700.1.101.15 clang-700.1.81). Type :help for assistance.
1> var myVar = 21
myVar: Int = 21
2> func increment(aVar: Int) -> Int {
3.     return aVar + 1
4. }
5>
6> increment(myVar)
$R0: Int = 22
```

To see LLDB debugging commands type `:help`

To escape from REPL mode type `Cmd+d`, or `:q`

Another possibility is

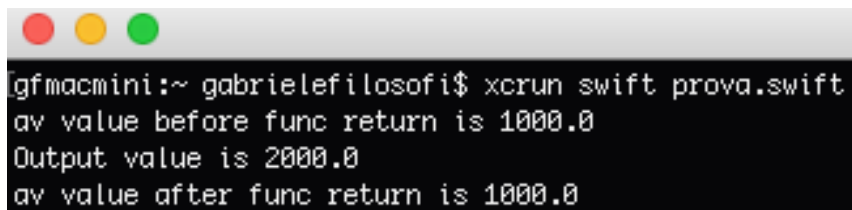
1. Edit the swift file, `<filename>.swift`
2. Open a terminal and type  
`$ xcrun swift <filename>.swift`  
or  
`$ swift <filename>.swift`

## Example

Edit prova.swift

```
var av: Double = 1200.0
var bv: Double = 2.0
func evaluate(a: inout Double, b: Double) -> Double {
    if a >= 1000.0 {
        a = 1000.0
    }
    print("av value before func return is \(av)")
    return a*b
}
var cv = evaluate(a: &av, b: bv)
print("Output value is \(cv)")
print("av value after func return is \(av)")
```

then

A terminal window with a title bar containing three colored circles (red, yellow, green). The terminal text shows the command to run the Swift script and its output.

```
[gfmacmini:~ gabrielefilosofi$ xcrun swift prova.swift
av value before func return is 1000.0
Output value is 2000.0
av value after func return is 1000.0
```

### Swift script

Another possibility is to use swift script files. Add the following *shebash* line on top of the script file

```
#!/usr/bin/xcrun swift
```

or

```
#!/usr/bin/env swift
```

### Example

```
firstPrimesSwiftScript x
1  #! /usr/bin/env swift
2  import Foundation
3
4  func isPrime(n: Int) -> Bool {
5      if n <= 2 {
6          return true
7      }
8      let d = Int(sqrt(Double(n)))
9      if d < 2 {
10         return true
11     }
12
13     for i in 2...d {
14         if n % i == 0 {
15             return false
16         }
17     }
18     return true
19 }
20
21 print("first primes: ")
22
23 for s in 2...10000000 {
24     if isPrime(n: s) == true {
25         print("\(s)", terminator: ", ")
26     }
27 }
```