```
import UIKit

//: # Swift
//: ## A langauge overview
//: by Samuel Oechsler
//:  > "Announced in 2014, the Swift programming language has quickly become one of the fastest growing languages in history.
//: > Swift makes it easy to write software that is incredibly fast and safe by design. Our goals for Swift are ambitious:
//: > we want to make programming simple things easy, and difficult things possible." - [swift.org](https://swift.org)

//: ## History & information
//: - Designed by Chris Lattner (LLVM) & folks at Apple
//: - Swift was released to the public at WWDC 2014
//: - Replaces the old `Objective-C` language, Apple inherited from NeXT (back in 1997)
//: - Open sourced under the Apache License 2.0 since version 2.2
//: - Combines a lot of concepts from `Objective-C, Rust, Haskell, Ruby, Python, C# and D`
//: - Current stable version is Swift 5.1(.2)
let languageFather = UIImage(named: "chris.jpg")

//: ## Prerequisits
//: - Supported platforms: **macOS**, **Linux** and IBM z/OS
//: - No official support for **Windows**, only [Swift for Windows](https://swiftforwindows.github.io/)
//: - Available IDEs: **Xcode** and **AppCode** (from JetBrains)
//: - Swift Playgrounds on iPad (great place to start)

let appCode = UIImage(named: "appcode.png")
let xcodeMeme = UIImage(named: "xcode-meme.jpg")

//: ## Declaring variables
//: - Constants are declared using `let`
//: - Variables are declared using `var`
//: - Swift supports inferred types (can be left out)
let name = "Samuel Oechsler"
var age = 21
var height = 1.80
var lovesSwift = true
var languages = ["german", "english"]

//: **Fun with Swift:** Swift is UTF-8 based so we can do this:
let π = 3.14

let 💥 = "This will work!"
let 🐛 = "Gonna fix that later!"

//: ### Optionals
//: - Optionals are delcared by appending a `?` to the type
//: - The type can not be inferred on declaration, if the value is nil
var pohneNumber: Int?
var address: String?
```

```
//: ### Unsafely unwrapped optional
//: - Why does this exist in the first place?
var positiveNumber: Int! = 1
let isMultipleOfFiv = positiveNumber.isMultiple(of: 5)

//: ## Operators
//: - Swift supports all the expected operators
//: - Some may work a bit different ...
let theAnswer = 42

//: ## Conditions
//: - Swift supports the *"known"* types of conditions

//: ### If conditions
//: - Allways need curly braces *"improved readabilty"*
let myBirthDate = Birthdate(withDay: 21, month: 11, andYear: 1998)

if myBirthDate.hasBirthday {
    print("🎉 Congrats, Happy birthday!")
} else {
    print("🎈 I'm sorry, but today is not your birthday!")
}

//: ### Ternary operator
let present = myBirthDate.hasBirthday ? "🎁" : "💩"

//: ### Switch statements
//: - Switch sattements in Swift are really *"pattern matching"*
//: - They can match `Bool`, `Range` and `enum`
//: - Compared to other languages (Java, C) cases do not fall-trough by default
let cake: Cake = .apple

switch cake {
case .apple:
    print("🍏 Great choice, I also love Apple.")
case .brownie:
    print("☝ Don't do drugs kids!")
default:
    print("😭 No cake for you it seems")
}

//: ## Loops
//: - Swift supports all the *"known"* types of loops

//: ### While loop
//: - Allways need curly braces *"improved readabilty"*
//: - Most of the time you can just use a `for in` loop instead
var count = 1
while count <= 100 {
    print("I'm currently at \(count).")
    count += count + 1
}
```

```
//: ### Repeat while loop
//: - Runs the code in the `repeat` block at least once
//: - Allways need curly braces *"improved readabilty"*
//: - Most of the time you can just use a `for in` loop instead
repeat {
    count += 1000
} while(count <= 1000)

//: ### For in loop
//: - Works with all collection types (`Range` included)
//: - Prefered type of loop in Swift (just use it, you'll be fine 😅)
for number in 1...100 {
    if number % 2 == 0 {
        print("\(number) is an even number")
    }
}

//: **Fun with Swift:** Swift allows us to label loops
//: and lets us combine `for` with `if` in an SQL like syntax
totalWasteOfCompileTime: while count <= 0 {
    // do something ...
}

for number in 0...0 where number == 0 {
    // do something ...
}

//: ## Functions
//: - Functions look  and work simmilar to other C style languages
//: - Parameters can have a label for *improved readability* (or overloads)
func helloWorld() {
    print("Hello World")
}

func fibonacci(forNumber n: Int) -> Int {
    if case n = 0 {
        return 0
    } else if case n = 1 {
        return 1
    } else {
        return fibonacci(forNumber: n - 2) + fibonacci(forNumber: n - 1)
    }
}

helloWorld()
fibonacci(forNumber: 7)

//: ## Object orientation

//: ### Protocols
//: - In Swift interfaces are called `protocol`

protocol ReadyForDoomsday {
```

```
    var availableNukes: Int { get }
    func fireNuke(withLaunchCode code: String)
}

//: ### Classes
//: - In Swift `classes` are reference types
//: - Internaly a pointer is used to pass them around

class Nation {

}

class Germany: Nation {
    public let chancellor = "Angela Merkel"
}

class America: Nation, ReadyForDoomsday {
    public let president = "Donald Trump"

    private var _ownedNukes = 6_185
    private var _deployedNukes = 1_600

    public var availableNukes: Int {
        _deployedNukes
    }

    public func fireNuke(withLaunchCode code: String) {
        if code == "0000" {
            // fire a nuke here ...
            _deployedNukes -= 1
            _ownedNukes -= 1

            print("🚀 Launch in 3, 2, 1 ...")
        } else {
            print("☝ Ah-ah-a, Wrong code entered")
        }
    }
}

let germany = Germany()
let america = America()
america.fireNuke(withLaunchCode: "0000")

//: ### Structs
//: - In Swift `struct`'s are always value types
//: - The value gets copied on pass internaly

struct Vector {
    var x: Float
    var y: Float

    func dotProduct(_ vector: Vector) -> Float {
        x * vector.x + y * vector.y
    }
```

```
    mutating func scale(by scalar: Float) {
        x *= scalar
        y *= scalar
    }
}

var someVector = Vector(x: 3, y: 2)
someVector.scale(by: 2)
let dotProduct = someVector.dotProduct(Vector(x: 2, y: 4))

//: ### Enumerations
enum NeedForSleep {
    case low
    case high
}

let meOnMondays = NeedForSleep.high

//: ## Now its your turn
//:  Feel free to play around with the provided material or invent the **next big thing** down here!
//: - If there are any questions, I'm happy to answer them
print("Thank you for participating!")
```