Introduction to iOS development with Swift

Lesson 2

Adrien Humilière
Trainline

adhumi+dant@gmail.com

adhumi.fr/teaching

- Strings
- Functions
- Structures
- Classes and inheritance

Strings


Here's to the crazy ones.
The misfits. The rebels.
The troublemakers. The
round pegs in the
holes.

Basics

```
let greeting = "Hello"
var otherGreeting = "Salutations"
```

```
let joke = """
  Q: Why did the chicken cross the road?
  A: To get to the other side!
  """
print(joke)
// Q: Why did the chicken cross the road?
// A: To get to the other side!
```

Basics – escaping

```
let greeting = "It is traditional in programming to print
\"Hello, world!\""
```

`\"`	Double quote
`\\`	Backslash
`\t`	Tab
`\r`	Carriage return (return to beginning of the next line)

Basics – Empty

```
var myString = ""

if myString.isEmpty {
  print("The string is empty")
}
```

Basics – Characters

```
let a = "a" // 'a' is a string
let b: Character = "b" // 'b' is a Character
```

Concatenation

```
let string1 = "Hello"
let string2 = ", world!"
var myString = string1 + string2 // "Hello, world!"

myString += " Hello!" // "Hello, world! Hello!"
```

Interpolation

```
let name = "Rick"
let age = 30
print("\(name) is \(age) years old")

// Rick is 30 years old
```

```
let a = 4
let b = 5
print("If a is \(a) and b is \(b), then a + b equals \(a+b)")
```

String equality and comparison

```
let name = "Johnny Appleseed"
if name.lowercased() == "joHnnY aPPleseeD".lowercased() {
  print("The two names are equal.")
}
```

String equality and comparison

```
let greeting = "Hello, world!"

print(greeting.hasPrefix("Hello"))
print(greeting.hasSuffix("world!"))
print(greeting.hasSuffix("World!"))
```

String equality and comparison

```
let greeting = "Hi Rick, my name is Amy."
if greeting.contains("my name is") {
  print("Making an introduction")
}
```

String equality and comparison

```
let name = "Ryan Mears"
let count = name.count
let newPassword = "1234"

if newPassword.count < 8 {
    print("This password is too short. Passwords should have
at least 8 characters.")
}
```

String equality and comparison

```
let someCharacter: Character = "e"
switch someCharacter {
  case "a", "e", "i", "o", "u":
    print("\(someCharacter) is a vowel.")
  default:
    print("\(someCharacter) is not a vowel.")
}
```

String equality and comparison

```
let cow = "🐮"
let credentials = "résumé"
let myBook = "私の本"
print("∞".characters.count)
```

Functions

```
tieMyShoes()
```

```
tieMyShoes()
```

```
makeBreakfast(food: "scrambled eggs", drink: "orange juice")
```

Defining a function

```
func functionName (parameters) -> ReturnType {
    // Body of the function
}
```

Defining a function

```
func displayPi() {
  print("3.1415926535")
}

displayPi() // 3.1415926535
```

Parameters

```
func triple(value: Int) {
  let result = value * 3
  print("If you multiply \(value) by 3, you'll get \
(result).")
}

triple(value: 10) // If you multiply 10 by 3, you'll get 30.
```

Multiple parameters

```
func multiply(firstNumber: Int, secondNumber: Int) {
  let result = firstNumber * secondNumber
  print("The result is \(result).")
}

multiply(firstNumber: 10, secondNumber: 5)
// The result is 50.
```

Return values

```
func multiply(firstNumber: Int, secondNumber: Int) -> Int {
  let result = firstNumber * secondNumber
  return result
}
```

Return values

```
func multiply(firstNumber: Int, secondNumber: Int) -> Int {
 return firstNumber * secondNumber
}
```

```
let myResult = multiply(firstNumber: 10, secondNumber: 5)
print("10 * 5 is \(myResult)")
```

```
print("10 * 5 is \(multiply(firstNumber: 10, secondNumber:
5))")
```

Argument labels

```
func sayHello(firstName: String) {
  print("Hello, \(firstName)!")
}

sayHello(firstName: "Amy")
```

Argument labels

```
func sayHello(to: String, and: String) {
  print("Hello \(to) and \(and)")
}

sayHello(to: "Luke", and: "Dave")
```

Argument labels

```
func sayHello(to person: String, and anotherPerson: String) {
  print("Hello \(person) and \(anotherPerson)")
}

sayHello(to: "Luke", and: "Dave")
```

Argument labels

```
print("Hello, world!")
```

Argument labels

```
print("Hello, world!")
```

```
func add(_ firstNumber: Int, to secondNumber: Int) -> Int {
    return firstNumber + secondNumber
}

let total = add(14, to: 6)
```

Default parameter values

```
func display(teamName: String, score: Int = 0) {
  print("\(teamName): \(score)")
}

display(teamName: "Wombats", score: 100)
display(teamName: "Wombats")
```

Structures

```
struct Person {
  var name: String
}
```

- Capitalize type names
- Use lowercase for property names

Accessing property values

```
struct Person {
  var name: String
}

let person = Person(name: "Jasmine")
print(person.name) // Jasmine
```

Adding functionality

```
struct Person {
  var name: String

 func sayHello() {
    print("Hello there! My name is \(name)!")
  }

}

let person = Person(name: "Jasmine")
person.sayHello() // Hello there! My name is Jasmine!
```

Instances

```
struct Shirt {
  var size: String
  var color: String
}

let myShirt = Shirt(size: "XL", color: "blue")

let yourShirt = Shirt(size: "M", color: "red")
```

```
struct Car {
  var make: String
  var year: Int
  var color: String

  func startEngine() {...}

  func drive() {...}

  func park() {...}

  func steer(direction: Direction) {...}
}

let firstCar = Car(make: "Peugeot", year: 2010, color: "blue")
let secondCar = Car(make: "Ford", year: 2013, color: "black")

firstCar.startEngine()
firstCar.drive()
```

Initializers

```
let string = String.init() // ""
let integer = Int.init() // 0
let bool = Bool.init() // false
```

Initializers

```
let string = String() // ""
let integer = Int() // 0
let bool = Bool() // false
```

Default values

```
struct Odometer {
  var count: Int = 0
}

let odometer = Odometer()
print(odometer.count) // 0
```

```
let odometer = Odometer(count: 27000)
print(odometer.count) // 27000
```

```
struct Shirt {
  let size: String
  let color: String
}

let myShirt = Shirt(size: "XL", color: "blue")


struct Car {
  let make: String
  let year: Int
  let color: String
}

let firstCar = Car(make: "Honda", year: 2010, color: "blue")
```

Custom initializers

```
struct Temperature {
  var celsius: Double
}

let temperature = Temperature(celsius: 30.0)
```

Custom initializers

```
struct Temperature {
  var celsius: Double
}

let temperature = Temperature(celsius: 30.0)
```

```
let fahrenheitValue = 98.6
let celsiusValue = (fahrenheitValue - 32) / 1.8

let newTemperature = Temperature(celsius: celsiusValue)
```

```
struct Temperature {
  var celsius: Double

  init(celsius: Double) {
    self.celsius = celsius
  }

  init(fahrenheit: Double) {
    celsius = (fahrenheit - 32) / 1.8
  }
}

let tempFromCelsius = Temperature(celsius: 18.5)
let tempFromFahrenheit = Temperature(fahrenheit: 212.0)
```

Instance methods

```
struct Size {
  var width: Double
  var height: Double

  func area() -> Double {
    return width * height
  }
}

var someSize = Size(width: 10.0, height: 5.5)

let area = someSize.area() // Area is assigned a value of 55.0
```

Mutating methods

```
struct Odometer {
  var count: Int = 0 // Assigns a default value to the 'count'
}
```

Need to:

- → Increment the mileage
- → Reset the mileage

```
struct Odometer {
  var count: Int = 0

  mutating func increment() {
    count += 1
  }

  mutating func increment(by amount: Int) {
    count += amount
  }

  mutating func reset() {
    count = 0
  }
}
```

Computed properties

Computed properties

```
struct Temperature {
  let celsius: Double
  let fahrenheit: Double
  let kelvin: Double
}

let temperature = Temperature(celsius: 0, fahrenheit: 32, kelvin: 273.15)
```

```
struct Temperature {
  var celsius: Double
  var fahrenheit: Double
  var kelvin: Double

  init(celsius: Double) {
    self.celsius = celsius
    fahrenheit = celsius * 1.8 + 32
    kelvin = celsius + 273.15
  }

  init(fahrenheit: Double) {
    self.fahrenheit = fahrenheit
    celsius = (fahrenheit - 32) / 1.8
    kelvin = celsius + 273.15
  }

  init(kelvin: Double) {
    self.kelvin = kelvin
    celsius = kelvin - 273.15
    fahrenheit = celsius * 1.8 + 32
  }
}
```

Computed properties

```
struct Temperature {
  let celsius: Double

  var fahrenheit: Double {
    return celsius * 1.8 + 32
  }

  var kelvin: Double {
    return celsius + 273.15
  }
}
```

Property observers

```
struct StepCounter {
    var totalSteps: Int = 0 {
        willSet {
            print("About to set totalSteps to \(newValue)")
        }
        didSet {
            if totalSteps > oldValue  {
                print("Added \(totalSteps - oldValue) steps")
            }
        }
    }
}
```

Property observers

```
var stepCounter = StepCounter()
stepCounter.totalSteps = 40
stepCounter.totalSteps = 100

// About to set totalSteps to 40
// Added 40 steps
// About to set totalSteps to 100
// Added 60 steps
```

Type properties and methods

```
struct Temperature {
  static var boilingPoint = 100.0

  static func convertedFromFahrenheit(_ temperatureInFahrenheit:
Double) -> Double {
    return(((temperatureInFahrenheit - 32) * 5) / 9)
  }

}

let boilingPoint = Temperature.boilingPoint
let currentTemperature = Temperature.convertedFromFahrenheit(99)
let positiveNumber = abs(-4.14)
```

Copying

```
var someSize = Size(width: 250, height: 1000)
var anotherSize = someSize

someSize.width = 500

print(someSize.width)
print(anotherSize.width)
```

self

```
struct Car {
  var color: Color

  var description: String {
    return "This is a \(self.color) car."
  }
}
```

self

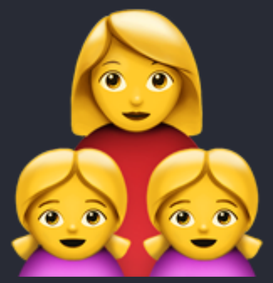
```
struct Car {
  var color: Color

  var description: String {
    return "This is a \(color) car."
  }
}
```

→ Not required when property or method names exist on the current object

self

```
struct Temperature {
  var celsius: Double

  init(celsius: Double) {
    self.celsius = celsius
  }
}
```

Classes and inheritance



```
class Person {
  let name: String

  init(name: String) {
    self.name = name
  }

  func sayHello() {
    print("Hello there!")
  }
}

let person = Person(name: "Jasmine")
print(person.name)
person.sayHello()
```

Inheritance

- Base class: `Vehicle`
- Subclass: `Tandem`
- Superclass: `Bicycle`

Inheritance

```
class Vehicle {
    var currentSpeed = 0.0

    var description: String {
        return "traveling at \(currentSpeed) km per hour"
    }

    func makeNoise() {
        // do nothing - a vehicle doesn't necessarily make noise
    }
}
```

Subclass

```
class SomeSubclass: SomeSuperclass {
    // subclass definition goes here
}
```

```
class Bicycle: Vehicle {
    var hasBasket = false
}
```

Subclass

```
class Tandem: Bicycle {
    var currentNumberOfPassengers = 0
}
```

Override methods

```
class Train: Vehicle {
    override func makeNoise() {
        print("Choo Choo!")
    }
}
```

Override methods

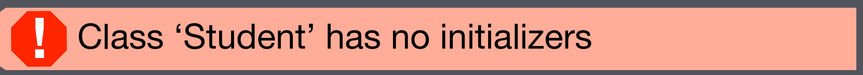
```
class Car: Vehicle {
    var gear = 1
    override var description: String {
        return super.description + " in gear \(gear)"
    }
}
```

Override init

```
class Person {
  let name: String

  init(name: String) {
    self.name = name
  }
}

class Student: Person {
  var favoriteSubject: String
}
```

Override init

```
class Person {
  let name: String

  init(name: String) {
    self.name = name
  }
}

class Student: Person {
  var favoriteSubject: String
}
```



```
class Person {
  let name: String

  init(name: String) {
    self.name = name
  }
}

class Student: Person {
  var favoriteSubject: String
  init(name: String, favoriteSubject: String) {
    self.favoriteSubject = favoriteSubject
    super.init(name: name)
  }
}
```

References

- When you create an instance of a class:
  - Swift returns the address of that instance
  - The returned address is assigned to the variable
- When you assign the address of an instance to multiple variables:
  - Each variable contains the same address
  - Update one instance, and all variables refer to the updated instance

```
class Person {
  let name: String
  var age: Int

  init(name: String, age: Int) {
    self.name = name
    self.age = age
  }
}

var jack = Person(name: "Jack", age: 24)
var myFriend = jack

jack.age += 1

print(jack.age) // 25
print(myFriend.age) // 25
```

```
struct Person {
  let name: String
  var age: Int
}

var jack = Person(name: "Jack", age: 24)
var myFriend = jack

jack.age += 1

print(jack.age) // 25
print(myFriend.age) // 24
```

Memberwise initializers

- Swift does not create memberwise initializers for classes
- Common practice is for developers to create their own for their defined classes

Class or structure?

- Start new types as structures
- Use a class:
  - When you’re working with a framework that uses classes
  - When you want to refer to the same instance of a type in multiple places
  - When you want to model inheritance

The End.