iOS 项目实践

1 - iOS 开发 & Swift 基础

Adamas

(Adam)

Adamas

课程目标

考试?

作业?

编程语言?

iOS?

0.2 * 编程语言 + 0.5 * iOS + 0.3 * 真实项目流程

课程结构

总共 7 节课

1. iOS 开发 & Swift 基础
2. Swift 基础, 项目介绍, XCode & 项目结构
3. UI 组件
4. Debug, Design, Agile & Github
5. 网络访问 & 数据存储
6. 动画效果, 手势 & Demo
7. 作业问题解答, 内存管理, 扩展知识 & 职业展望

课程组成

课程内容

(随时在聊天窗提问)

+

5-15 分钟提问时间

+

课后作业

(**最终作业**)

iOS 开发

游戏:

3D: Unity (C#)

2D: Cocos 2D (C++ / JS)

跨平台:

Microsoft Xamarin: (C#)

facebook React Native: (JS)

More

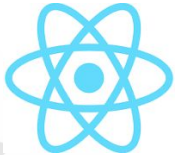
iOS 开发

原生

UIKit (Objective-C)

UIKit (Swift)

SwiftUI (Swift)



面向对象的编程语言

1 汇编语言

2 面向过程的语言: C 语言

3 面向对象的语言: C++

4 第二代面向对象的语言: Java / Objective C / C#

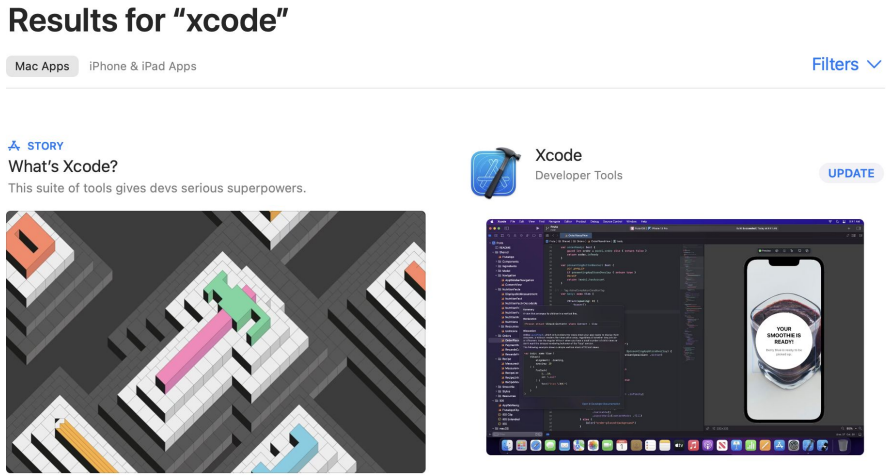
5 第三代面向对象的语言: Swift / Kotlin

Hello World


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

搭建 XCode 环境

1. App Store 搜索 “XCode”
2. 下载并安装
3. 点击运行
4. 点击 “Install” 安装命令行



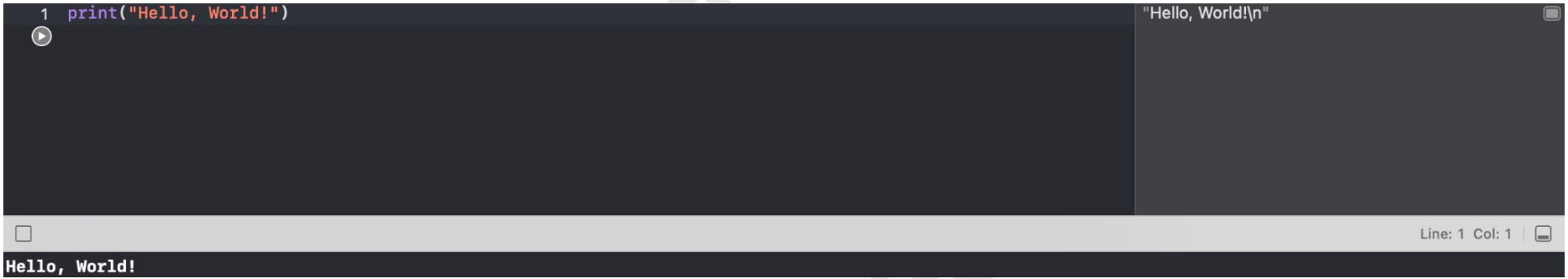
搭建 XCode 环境 - Troubleshooting

OS版本和 XCode 版本不兼容时

下载历史 XCode 版本

https://developer.apple.com/download/all/

Apple Playground



https://swiftfiddle.com/

行结束符

;

基础数据类型

	Java	Swift
整数	int / Integer	Int
小数	float / Float / double / Double	Double
布尔	boolean / Boolean	Bool
字符	char / Character	Character
字符串	String	String
枚举类型	enum	enum
结构体	/ (Class)	struct

基础数据类型 - Int

Java

```
Integer i = 1;
```

Swift

```
let i: Int = 1
```

基础数据类型 - Double

Java

```
Double d = 1.0;

Float f = 1f;
```

Swift

```
let d: Double = 1.0
```

基础数据类型 - Bool

Java

```
Boolean b = true;
```

Swift

```
let b: Bool = true
```

基础数据类型 - Character

Java

```
Character c = 'a';
```

Swift

```
let c: Character = "a"
```

基础数据类型 - String

Java

```
String str = "string";
```

Swift

```
let str: String = "string"
```

基础数据类型 - String 占位符

	Java	Swift
整数	%d	%d
小数	%f	%f
字符串	%s	%@

基础数据类型 - String 占位符

Java

```
String name = "Adamas";

Integer age = 100;


String.format("I am %s who is %d years old.", name, age);
```

Swift

```
let name = "Adamas"
let age = 100


String(format: "I am %@ who is %d years old.", name, age)


"I am \(name) who is \(age) years old."
```

基础数据类型 - enum

Java

```
enum E {
    CASE1
}

E e = E.CASE1;
```

Swift

```
enum E {
    case case1
}

let e = E.case1
```

Enum 能有自己的
运算变量以及函数

基础数据类型 - struct

Java

```
class S {
    private int i;
        S(int i) {
            this.i = i;
        }
        int getI() {
            return i;
        }
}
S s = new S(1);
```

Swift

```
struct S {
    let i: Int
}

let s: S = S(i: 1)
```

数组

Java

```
String[] strs1 = {"String1",
"String2"};

String[] strs2 = {"String3",
"String4"};

// Merge using some functions

for (String str : strs1) {

    System.out.println(str);

}
```

Swift

```
let strs1: [String] = ["String1",
"String2"]
let strs2: Array<String> =
["String3", "String4"]
let strs = strs1 + strs2
print(strs)
```

词典 Dictionary

Java

```
HashMap<String, String>
personalInfo = new
HashMap<String, String>();

personalInfo.put("Firstname",
"Adamas");

personalInfo.put("Lastname",
"Zhu");

personalInfo.put("Title", "Mr");
```

Swift

```
var personalInfo =
["Firstname": "Adamas",
"Lastname": "Zhu",
"Title": "Mr"]
personalInfo["Gender"] = "M"
```

运算符

Java

+ - * /

+= -= == != !

> < >= <=

?:

&& ||

++ --

Swift

+ - * /

+= -= == != !

> < >= <=

?:

&& ||

~~++ --~~ **DEPRECATED**

Condition1 && Condition2
Condition1, Condition2

数据类型转换

```
let i: Int = Int(3.5)
```

“\(i)”

自定义转换函数: toString()

```
let i: Int = 3.5
```

???

变量

Java

```
Integer i = 100;
```

Swift

```
var i: Int = 100
var i = 100

i += 1

let c: Character = "a"    ???
```

常量

Java

```
final Integer i = 100;
```

Swift

```
let i: Int = 100
let i = 100

i += 1
```

XXX

函数

Java

```
public void test(int i) {
    // Do something
}

test(1);
```

Swift

```
public func test(_ i: Int) -> Void {
    // Do something
}
public func test(_ i: Int) {
    // Do something
}

test(1)
```

函数重载 Overload

```
public func test(name: String) -> Void {
    print(name)
}

public func test(address: String) -> Void {
    print(address)
}

test(name: "Adamas")
test(address: "1 Spring St")
```

Override?

函数参数

```
func append(_ string: String, to originalString: String) -> String {
    return originalString + string
}
var str = append(".png", to: "Test")

func printMessage(_ message: String, withAdditionalInfo additionalInfo:
String) {
    print(message + "\nNote: " + additionalInfo)
}
printMessage("I'm here", withAdditionalInfo: "Adamas")
```

函数参数 - 默认

```
func printCurrencyString(for double: Double,
                         usingCurrencySymbol currencySymbol: String = "$")
{
    print(currencySymbol + String(double))
}

printCurrencyString(for: 100) // $100.0
printCurrencyString(for: 100, usingCurrencySymbol: "¥") // ¥100.0
```

Block

```
func printMessage(_ message: String, withAdditionalInfo additionalInfo:
String) {
    print(message + "\nNote: " + additionalInfo)
}
let printMessageAction:(String, String) -> Void = {message,
additionalInfo in
    print(message + "\nNote: " + additionalInfo)
}
let printMessageAction = printMessage
printMessageAction("I'm here", "Adamas")
```

类 - Java

```
class S {
    private int i;
    S(int i) {
        setI(i);
    }
    int getI() {
        return i;
    }
    void setI(int newI) {
        i = newI > 0 ? newI : 0;
    }
}

S s = new S(-1);
```

类 - Swift

```
class S {
    var i: Int {
        didSet {
            i = i > 0 ? i : 0
        }
    }
    init(i: Int) {
        self.i = i
    }
    func setI(_ i: Int) {
        self.i = i > 0 ? i : 0
    }
}

let s = S(i: 0)
s.setI(-1)
s.i = -1
print(s.i)
```

类 - 继承

Java

```
class Car extends Vehicle {
    // Body
}
```

Swift

```
class Car: Vehicle {
    // Body
}
```

类 - Scope

	Java	Swift
公有	public	open
不可修改	final	final
公有 (仅本包可修改)	/	public
私有	private	private
包	default	internal (default)
包 / 子类	protected	/
只读	/	private (set)

类 - 当前对象

Java	Swift
this	self

类 - 父类

Java

super

Swift

super

Override

```
class A {
    func test() {
    }
}

class B: A {
    override func test() {
    }
}
```

运算变量

只读

```
class Person {
    private var yearOfBirth: Int
    init(yearOfBirth: Int) {
        self.yearOfBirth = yearOfBirth
    }
    var age: Int {
        return 2022 - yearOfBirth
    }
}
```

读写

```
var age: Int {
    get {
        return 2022 - yearOfBirth
    }
    set {
        2022 - newValue
    }
}
```

静态变量/函数

Java

```
static Integer i = 100;
static void test() {
    // Body
}
ClassName.i
ClassName.test()
```

Swift

```
static var i = 100
static func test() {
    // Body
}
Person.i
Person.test()
Self.i
Self.test()
```

常量

Java

```
final static Integer i = 100;
```

Swift

```
static let i = 100
```

Strings???

接口

Java

```
interface Testable {
    void test();
}
```

Swift

```
protocol Testable {
    var i: Int { get }
    func test()
}
```

实现接口

Java

```
class ComplexManager
implements Testable {

    // Body

}
```

Swift

```
class ComplexManager: Testable
{
    // Body
}
```

扩展

```
class S {
}

extension S {
    func test() {
        // Body
    }
}
```

```
protocol Testable {
}

extension Testable {
    func testNothing() {
        // Body
    }
}
```

不同文件中

```
class S1 {
}

protocol Testable {
    func test()
}

extension S1: S, Testable {
    func test() {
        // Body
    }
}
```

作业

在 Playground 中创建类ImageStorage, 包含如下内容:

- 常量imagesKey, 值为"images"
- 只读数组变量images, 类型为Data数组
- 无参数的构造函数
- 函数saveImage, 接受一个Data类型的参数, 内容为空
- 函数deleteImage, 接受一个Data类型的参数, 内容为空
- 私有函数hasImage, 接受一个Data类型的参数, 返回值为Bool类型, 内容为空

作业

在 Playground 中创建Enum类型NetworkError, 包含如下内容:

- String类型常量networkError, 值为"NetworkError"
- String类型常量networkError, 值为"OtherError"
- 两个case connection和other
- 只读String类型变量message
- 其返回值为上述定义的对应常量

作业

在 Playground 中创建类RandomImageGenerator, 包含如下内容:

- String类型常量endpoint, 值为"https://picsum.photos"
- 函数generateRandomImage, 接受如下参数, 内容为空
  - 一个Int类型的参数width
  - 一个Int类型的参数height
  - 一个仅接受Data输入类型的 Block
  - 一个仅接受NetworkError输入类型的 Block
- 接受一个Int类型参数的构造函数, 该参数的默认值是0

作业

创建前面三者的变量, 并尝试调用其中的函数和变量

END