Lab 02 – Xcode & Swift intro
(updated for Xcode 8 and Swift 3)

Instructor:
Mitja Hmeljak,
http://pages.iu.edu/~mitja
mitja@indiana.edu
Programming Tools for B581/Spring 2017

**Android** Programming tools for B581/Spring 2017:

- **Tools:** Android Studio
  
  *(should be installed soon! in STC Mac labs and STC Windows labs)*
  
- **Languages:** Java (CPU side) and GLSL (GPU side)

**iOS** Programming tools for B581/Spring 2017:

- **Tools:** Xcode 8.x
  
  *(in STC Mac labs, upgraded for Spring 2017 semester from previous Xcode 7.x)*
  
- **Languages:** Swift 3 (CPU side) and GLSL (GPU side)
  
- **Frameworks:** OpenGL ES 2.0, but also a bit of UIKit & Motion
  
- **Design Strategies:** MVC

**WebGL** Programming tools for B581/Spring 2017:

- **Tools:** a recent release of Firefox, Chrome, Safari, etc.
  
- **Languages:** Javascript (CPU side) and GLSL (GPU side)
Programming Tools for B581/Spring 2017

Android Programming tools:
• we're planning for everyone trying them out once in Lab 03

iOS Programming tools:
• everyone is trying these out once in Lab 02 today

WebGL Programming tools:
• we're planning for everyone trying them out once in Lab 03

what happens next in B581 about programming tools?
after Lab 03,
every B581 student will be asked to
choose one of those programming platforms
for all subsequent B581 / Spring 2017 assignments
A first OpenGL ES Swift app: "Hello OpenGL ES"

To develop OpenGL ES graphics iOS applications – usually referred to as apps – one needs:

• the iOS SDK, including:
  • the Xcode IDE
    (integrated development environment)
  • the iOS simulator
  • Xcode's Interface Builder
    to design GUI elements

• an x86-based host running OS X
Now start Xcode 8.x, and select the menu:

File → New → Project...

this brings up a choice of possible template projects.

Select the project template:

iOS → Application → Game

then click Next...
A first OpenGL ES Swift app: "Hello OpenGL ES"

as **Product Name** choose "Hello OpenGL ES", (no need to add an account for "Team"),

set **Language** to **Swift**,
set **Game technology** to **OpenGL ES**,
set **Devices** to **Universal**,
do **not** select
"**include Unit Tests**"
or "**include UI Tests**",
then click **Next**, then... (next slide)
A first Swift OpenGL ES Swift App

select "Desktop" either from the sidebar, or from the filesystem navigation pop-up menu,

**do not select**
"Source Control",

don't add to any project or workspace

and click **Create**.
Workaround to OpenGL ES template bug in Xcode 8

The template code provided in Xcode 8 for OpenGL ES-based apps written in Swift has a bug that causes the sample code to crash at first run. This bug is present in template code in all Xcode 8 versions from beta to 8.2.1.

As a workaround, please **either**:

1. download the provided please download the Xcode project example for a "Hello OpenGL ES 2.0" app, prepared for B581/Spring 2017 [https://www.cs.indiana.edu/classes/b581/2017/labs/lab04-code/HelloOpenGLES20iOS.zip](https://www.cs.indiana.edu/classes/b581/2017/labs/lab04-code/HelloOpenGLES20iOS.zip)

   or:

2. fix the template code bug (in the new project that you set up as from instructions) by changing the function in the source code file thus:

```swift
//uncio BUFFER_OFFSET(_ i: Int) -> UnsafeRawPointer {
//    return UnsafeRawPointer(bitPattern: i)!
//} Thread 1: EXC_BAD_INSTRUCTION

func BUFFER_OFFSET(_ i: Int) -> UnsafeRawPointer? {
    return UnsafeRawPointer(bitPattern: i)
}
```
To test your first OpenGL ES Swift App, you can run it in the iOS Simulator.

- In Xcode's toolbar (top-left of the main Xcode window) select "iPhone 6" note: do not select any phone model simulator other than "iPhone 6", since some other (simulated) models are buggy when running OpenGL ES
- then select the menu Product → Run in Xcode, to start the iOS simulator.
Getting around Xcode

The Xcode workspace window has several parts:

- **Toolbar**
- **Utility area**
- **Navigator area**
- **Editor area**
Getting around Xcode: source code files

To look at the source code, select the Project Navigator in the navigator area of the Xcode workspace window:

Look at the GameViewController.swift file, which contains most of the app's Swift (CPU) code and the Shader.fsh and Shader.vsh files, which contain the OpenGL Shader Language (GPU) code.
starting an OpenGL ES Swift app

In general, the main entry point for an application at launch time is usually the `main()` function.

However, in the Swift programming language there is no need for a `main()` function (although it's possible to have one) since the Swift “code written at global scope is used as the entry point for the program, so you don’t need a `main()` function.”
running an OpenGL ES Swift app

to test the Hello OpenGL ES app, you can run it in the iOS simulator. Pick a simulated device of your choice, then select the menu Product → Run.
testing an OpenGL ES Swift app in the iOS Simulator
1. When you're done experimenting with your first OpenGL Swift app project:
   1. select the Xcode menu *Product ➔ Clean*
   2. quit Xcode, then rename your "Hello OpenGL ES" folder to "HelloOpenGLES\textit{yourusername}"
   3. place your "HelloOpenGLES\textit{yourusername}" folder in your lab02 local github repository copy, then \textit{add, commit & push}
   4. (see instructions for IU github at: \url{https://www.cs.indiana.edu/classes/b581/2017/resources/github.html})

(\textit{where yourusername is your IU username, not the word "yourusername"})
Lab 02 deliverables

if you're done with all material the previous pages, you may consider your Lab 02 task completed: turn it in to IU Github as instructed.

...you may look at the subsequent slides if you are interested in more details about how iOS apps are written in Swift and run on the iOS Simulator...
the Swift programming language was the WWDC 2014 surprise

A summary of Swift features from Fall 2014: the language was still in beta at the time, and quite a few things have changed since then...

...for an up-to-date Swift reference, check:
Swift: Objective-C without the "C"?

- Swift language documentation:

iOS Application Development: Swift and Xcode


let's try a Swift Playground

In Xcode 8.x, select the menu
File → New → File:

and select
iOS → Source → Playground
then click Next...
a Swift Playground

name it "SwiftPlay.playground", (or something similar) and save to Desktop:

(this alert may pop up: in which case you'll need to authenticate with your username/password)
a Playground in Xcode 8.x

what you type is what you get →
on the right side of the window

```swift
// Playground - noun: a place where people can play

import UIKit

var str = "Hello, playground"

"Hello, playground"
```
Swift – the language
compared to Python and
compared to C-derived languages (Java, C++, etc.)

in Swift – just like in Python, and unlike C-derived languages:
• there is no `main()` function
• there is no need for `;` (semicolons) to terminate a line

in Swift – just like in C-derived languages – and unlike Python:
• there is no compulsory indentation (but it helps visual clarity)
• blocks don't depend on indentation level (safer!)

in Swift – just like in C-derived languages – and unlike Python:
• `{curly brackets}` are required to delimit code blocks.
  For example in `if` statements.
Swift constants and variables declarations using **let** (constants) and **var** (variables) keywords:

Use **let** every time a value will never change: use immutable constants instead of variables, whenever possible. Makes code **safer** (e.g. in a multithreaded environment) and **cleaner** (i.e. more readable, knowing a value won't change).
the type of a Swift variable (or constant) can be either *explicit* as in C, or *inferred* as in Python (as seen in the previous slide). Floating-point numbers can be converted to integers, and `Int()` always truncates the numbers: it does not round.

```swift
import UIKit

var str = "Hello, playground"

let studentsInI399: Int = 25
let studentsInWaitlist: Float = 2
// the above are explicitly declared types -- different ones!
// --> so that this would cause an error:
// let totalStudents = studentsInI399 + studentsInWaitlist

// but we can convert a Float to an Int, thus making it work:
var totalStudentsInI399 = studentsInI399 + Int(studentsInWaitlist)
```

"Hello, playground"

25

2.0

27
basic Swift types

floats, doubles, bools, strings:

```swift
// floating point numbers: Float and Double types
let floatingExplicit: Double = 78.65
let floatingInferred = 78.65
// the above are both Double, because Double is the default
let floatingLessPrecise: Float = 78.65

// boolean logic values: Bool types
let aBooleanValueInferred = true
let aBooleanValueExplicit: Bool = false
// using actual "true"/"false" values, unlike in C

// text and strings: the String type
let protagonistInferred = "Jack Spratt"
let protagonistExplicit: String = "Landen Parke-Laine"

// string "interpolation" is not the same as "point interpolation"...
println("\(protagonistExplicit) weights \(floatingInferred) kg")
// it's actually a handy way to substitute something in a string
```
the *type* of a Swift variable or constant can be either: explicit (as in C), or inferred as in Python

```swift
let myFloat = 1.2       // it's a Double, not a Float
let π = 3.15           // some Unicode is fine...
//let 🌟 = 3.14         // ...but not too much?

for i in 1...10 {
    println("myFloat is \(myFloat) at count \(i)\")
    var myInt = Int(myFloat)
    // var myInt2 = myFloat as Int //-- won't work
    println("myFloat is \(myInt) at count \(i)\")
}
```

casting vs. conversion:
floating-point numbers can be converted to integers using the type name to initialize another variable e.g. `Int(33.22)`

`Int()` always truncates the numbers: it does not round.
Swift type casting

```swift
let myFloat = 1.2 // it's a Double, not a Float
let π = 3.15 // some Unicode is fine...
// let 🇺🇸 = 3.14 // ...but not too much?

for i in 1...10 {
    println("myFloat is \(myFloat) at count \(i)")
    var myInt = Int(myFloat)
    // var myInt2 = myFloat as Int <-> won't work
    println("myFloat is \(myInt) at count \(i)")
}
```

casting == to treat an instance (e.g. myFloat above) as if it were a different superclass or subclass from somewhere else in its own class hierarchy.

casting wouldn't work in the above example: "myFloat as Int" gives an error.

An example of where casting does work:

```swift
for thing in things {
    switch thing {
    case 0 as Int:
        println("zero as an Int")
    case 0 as Double:
        println("zero as a Double")
    }
```
Swift collection types

**arrays** and **dictionaries** are *typed* collections

```swift
let vehicleTypes: [String] = ["bike", "unicycle", "boat"]
let numberOfWheels: [String: Int] = ["bike": 2, "unicycle": 1, "boat": 0]

for (vehicle, wheelCount) in numberOfWheels {
    println("\(vehicle)s have \(wheelCount) wheels")
}
```

**tuples** are groupings of values

... they are not a separate data type.

Tuples are useful to pass multiple values around.
**Swift tuples**

**named tuples** are groupings of values

... that can be used for example to return multiple values and refer to them separately:

```swift
let numberOfWheels: [String: Int] = ["bike": 2, "unicycle":1, "boat":0]

func returnWheelNo(someT: [String: Int], st:String) -> (name: String, wheels: Int) {
    return (st, someT[st as String]!)
}

let checkWheels = returnWheelNo(numberOfWheels, "bike")

println( "the vehicle type \(checkWheels.name) has \(checkWheels.wheels) wheels")
```
Swift control flow statements: *if*

if statements look like C, but **curly brackets** are compulsory:
Swift control flow statements: *for* loops

*for* loops in Swift can be written similarly to Python:

```swift
// this is a dictionary of arrays:
let interestingNumbers = [
    // these are Arrays of values
    // (they're defined as Swift standard library types)
    "Prime": [2, 3, 5, 7, 11, 13],
    "Fibonacci": [1, 1, 2, 3, 5, 8],
    "Square": [1, 4, 9, 16, 25],
]
var largest = 0

// this is a for loop to go through all kinds of numbers:
for (kind, numbers) in interestingNumbers {
    // this is a for loop to go through each number
    // (in the current kind of numbers):
    for number in numbers {
        if number > largest {
            largest = number
        }
    }
}
println("the largest number is \(largest)")
```

```
0
(5 times)
"the largest number is 25"
```
Swift  

for loops

for loops can be Python-like with ranges, or C-like with 3 declarations:

```swift
let a=0
let b=10

for i in a...b { // a closed range, Python-style
    print(i) // "i" hasn't been defined before!
}
println()

for i in © { // a closed range, Python-style
    print(i) // this prints out one char at a time
}
println()

for i in a..<b { // a half-closed range, Python-style
    print(i)
}
println()

for var i=0; i<10; i++ { // here i needs to be defined first!
    print(i)
}
println()
```
Swift control flow statements: `switch/case`

`switch/case` statements, C-like, but a bit different in Swift:

```swift
let glassesStatus = "new"

switch glassesStatus {
    case "scratched":
        let glassesOrder = "Add scratch resistant coating."
    case "can't read":
        let glassesOrder = "Take a vision test."
    default:
        let glassesOrder = "Enjoy your new glasses."
}
```

the switch statement *must* be exhaustive!

the switch statement does *not* "fall through"
Swift functions

*functions* in Swift have *input* arguments and output *return* values declared thus:

```swift
// functions in Swift are declared with the func keyword
// any arguments are listed in parentheses as "name: Type",
// and the return value type is declared after a "->"

func introduceYourself(name: String, age: Int, title: String) -> String {
    return "Hello, I'm \(name), I'm \(age) years old and I'm a \(title)."
}

print("introduceYourself("Thursday Next", 35, "literary detective")")
```

Console Output

```
Hello, I'm Thursday Next, I'm 35 years old and I'm a literary detective.
```
Swift functions

Functions in Swift can be nested! That means that you can define a function inside a function, and the inner function will be available only to code inside the outer function...

...maybe an example will make it clearer :-)
References:

http://web.stanford.edu/class/cs193p/cgi-bin/drupal/