BEGINNING iOS PROGRAMMING

GEORGIA TECH RESEARCH NETWORK OPERATIONS CENTER (RNOC)

*slides are based on the presentations by Professor Russell J Clark and Brian Davidson
OPTIONS

- Swift
- Objective-C
- RubyMotion
- Xamarin
- Hybrid
Swift

Safe, Modern and Powerful

PROGRAMMING 101

Programming

Data
- Mutable
- Immutable

Code
- Procedural
- Object-Oriented
In programming you use a convenience name to represent memory where the data is actually stored.

```javascript
let namedArea = DATAValue
```

```javascript
var namedArea = DATAValue
```
Immutable data is never modified

let greeting : String = "Hello World"

let greeting = "Hello World"
Mutable data is can be modified

```javascript
var greeting : String = "Hello World"
```

```javascript
var greeting = "Hello World"
```

```
var greeting = "Hello World"
```
Comments are a way for the programmer to add notes to himself or others who will read the code, but the compiler will ignore and they won’t exist in the final binary.

// this is a single line comment

/* this is a multi-line comment */
Besides strings you have...

**Fundamental**

- **Int**
  - Whole numbers
  - Eg. 42

- **Double & Float**
  - Floating-point numbers
  - Eg. 42.0, 42.034

- **Bool**
  - True or false

**Collection**

- **Array**
  - Items accessed via index
  - Index starts from 0

- **Dictionary**
  - Store key-value pairs
  - Access element via keys

- **Tuple**
An array is a collection of data items which can be accessed via an index beginning with 0.

```swift
var cardSuits: [String] = ["Spades", "Diamonds", "Hearts", "Clubs"]
var cardSuits = ["Spades", "Diamonds", "Hearts", "Clubs"]
println(cardSuits[0])
```
A dictionary is similar to a hash table and allows you to store key-value pairs and access the value by providing the key.

```javascript
var cards = ["Jack" : 11, "Queen" : 12, "King" : 13]
cards["Ace"] = 1
println(cards["Ace"])
cards["Ace"] = 14
println(cards["Ace"])```
There are instances where you sometimes need to denote the lack of a value which is different from an empty value or zero, for this swift has optionals which says either

– There is a value, and it equals x

or

– There isn’t a value at all
For instance, if I was writing code to convert a string to an integer I would like to note when it was unable to convert:

- \texttt{Int("42") \rightarrow Int(42)}
- \texttt{Int("0") \rightarrow Int(0)}
- \texttt{Int("foobar") \rightarrow ?}

As not all strings can be integers, I need to signify that the result of my code isn’t just an \texttt{Int} but an \texttt{Int} that may not exist:

- \texttt{Int?}
A special value used to represent the lack of a value

• `var someString : String?`
• `// someString is automatically set to nil`

Or you can specifically assign a value to nil

• `var someInt: Int? = 42`
• `// someInt contains an actual Int value of 42`
• `someInt = nil`
• `// someInt now contains no value`
You can check if an optional value has a value or not by comparing it to nil

- ( someString == nil )
  - // will succeed if someString doesn’t have a value
  - // will fail if someString is an empty string or any other string

- ( someInt != nil )
  - // will succeed if someInt has a value
  - // will succeed if someInt has the value zero
Once you know if a variable has a value you can access the inner value by force unwrapping it with (!)

- let unwrappedInt : Int = someInt!
- let unwrappedInt = someInt!
Loop over the collections

- while
- do–while
- for
- for–in
Conditional Statements

• if
• if–else
• if–else if–else
• switch
CONTROL FLOW – while

Execute a block of code when something is true

```java
while incomplete {
    println(“Downloading”)
}
```

```java
while !complete{
    println(“Downloading”)
}
```
Execute a block of code at least once

```javascript
var incomplete = false
do {
    println(“Downloading”)  
} while incomplete
```
Execute a block of code for every item or a subset of items in a collection

```swift
for var i = 0; i < cardSuits.count; i++ {
    println(cardSuits[i])
}
```

```swift
for cardSuit in cardSuits {
    println(cardSuit)
}
```
Execute a block of code that meets a condition

```java
if cardValue == 11 {
    println("Jack")
} else if cardValue == 12 {
    println("Queen")
} else {
    println("Not found")
}
```
Execute a block of code that matches a condition or run a default block

```java
switch cardValue {
    case 11:
        println("Jack")
    case 12:
        println("Queen")
    case 13:
        println("King")
    case 14:
        println("Ace")
    default:
        println("Not found")
}
```
We’ve been using println so how do we create our own?

```swift
func printCard()
{
    println("Ace")
}

func printCard(cardName: String)
{
    println(cardName)
}

func printCard(cardName: String, cardValue: Int)
{
    println("\(cardName) : \(cardValue)")
}
```
FUNCTIONS

No return value

```swift
func printCard(cardName : String, cardValue : Int){
    println("\(cardName) : \(cardValue)")
}
```

Return value

```swift
func buildCard(cardName : String, cardValue : Int) -> String {
    return "\(cardName) : \(cardValue)"
}
```
A class holds data as well as functions that act upon that data

class Animal{
    let species: String
    let weight: Double
    let age: Int
}

A class holds data as well as functions that act upon that data

class Animal{
    let species: String
    let weight: Double
    let age: Int

    init(species: String, weight: Double, age: Int){
        self.species = species
        self.weight = weight
        self.age = age
    }
}
A class holds data as well as functions that act upon that data

class Animal{
    let species: String
    let weight: Double
    let age: Int

    .
    .
    .
    
    func printDetails(){
        println(“Species- \
          (self.species): Weight- \
          (self.weight): Age- \
          (self.age)”)
    }
}
A series of objects can be modeled into a hierarchy. A Cat is an Animal therefore it is a subclass of an Animal

```swift
class Cat: Animal {
    let furColor: String

    init(species: String, weight: Double, age: Int){
        furColor = "Brown"
        super.init(species, weight: weight, age: age)
    }

    init(species: String, weight: Double, age: Int, furColor: String){
        self.furColor = furColor
        super.init(species, weight: weight, age: age)
    }
}
```
XCODE – WORKSPACE TOOLBAR

- Run button
- Scheme menu
- Editor configuration buttons
- Stop button
- Activity viewer
- Workspace configuration buttons
**XCODE – NAVIGATION AREA**

**Project navigator.** Add, delete, group, and otherwise manage files in your project, or choose a file to view or edit its contents in the editor area.

**Symbol navigator.** Browse the symbols in your project as a list or hierarchy. Buttons on the left of the filter bar let you limit the shown symbols to only classes and protocols, just those in your project, and containers.

**Find navigator.** Use search options and filters to quickly find any string within your project.

**Issue navigator.** View issues such as diagnostics, warnings, and errors found when opening, analyzing, and building your project.

**Test navigator.** Create, manage, run, and review unit tests.

**Debug navigator.** Examine the running threads and associated stack information at a specified point or time during program execution.

**Breakpoint navigator.** Fine-tune breakpoints by specifying characteristics such as triggering conditions.

**Report navigator.** View the history of your build, run, debug, continuous integration, and source control tasks.
• **Source editor.** Write and edit source code.

• **Interface Builder.** Graphically create and edit user interface files.

• **Project editor.** View and edit how your apps should be built, such as by specifying build options, target architectures, and app entitlements.
TODO LIST

WALKTHROUGH DEMO