ATLS 4120/5120: Mobile Application Development
Week 3: Swift Intro

Swift
Swift is an object-oriented programming language developed by Apple for developing iOS and OS X apps
Announced in June 2014 at the WorldWide Developer Conference with Xcode 6
Works side-by-side with Objective-C

Playground
Xcode
File | New | Playground
A playground is a type of file that allows you to test out Swift code, and see the results of each line in the sidebar.
Name: swift intro
Platform: iOS
Save
Delete what's there so you start with an empty file
(swift1 playground)

Variables and constants

- Variables and constants associate a name with a value of a specific type.
  - The value of a variable can change
  - The value of a constant cannot change
- Only use variables when the value will change, otherwise use constants.
- Variables use the keyword `var`, constants use `let`
- Swift is a strongly typed language.
- Can use almost any character in a variable or constant name
- Once you've declared a constant or variable of a certain type, you can't change its type.
- You can't change a constant into a variable or a variable into a constant.

```
var message : String = "Hello class"
let classMax : Int = 20
var age : Int
age = 20
age="old" (gives error, wrong type)
```

Data Types
Swift has all the standard data types(all listed in book and docs)

- **Int** represents integers
- **Double** represents 64-bit floating points
  - has a precision of at least 15 decimal digits
- **Float** represents 32-bit floating points
  - can have the precision be as little as 6 decimal digits
- **String** represents characters
  - Values must be in quotes
  - String is a value type
  - Use \ to escape a quote in a String
  - String is a value type so it's value is copied if it's assigned or passed

- **Bool** represents Boolean
  - Values true or false

Type Inference

- If you provide an initial value for a variable or constant Swift can infer the data type
- Because Swift is type safe, it performs type checks when compiling your code and flags any mismatched types as errors.
- Without an initial value for a variable or constant you must provide the type

```
var name = "Aileen"
firstName="Aileen" (error, needs var or let)
var firstName="Aileen"
name="Pierce"
name=20 (gives error, wrong type)
```

- print is a global function for output
- To print out the value of a variable or constant use a backslash and parentheses.
- var name : String = "Aileen"
- print("My name is \(name)")

```
print("Who am I?")
print("\(firstName)")
print("\(firstName)"+"\(age)")
print("\(firstName)" + " \(name)")
```

Swift does not provide implicit type conversion when you assign a variable of one numeric type to a variable of another numeric type, you must do the conversion explicitly.

```
let a = 42
let b = 0.123
let c = a + b (error)
```
You must convert the Int to a Double
```
let c = Double(a) + b
```

Tuples

- A tuple is a way to group values
  - Any type
  - As many values as you want
- Useful way to pass multiple values around easily

```
let violet = ("#EE82EE", 238, 130, 238)
print("Violet is \(violet.0)")
let (hex, red, green, blue) = violet
print("Violet is \(hex)")
```

Operators

- Swift supports the standard arithmetic and comparison operators(in books and docs)
- Equality is ==
- String concatenation: +
- Don't use the ++ or -- shorthand, it's going away in Swift 3

        - Use += or -= instead
- Logical operators
    - Logical NOT: !
    - Logical AND: &&
    - Logical OR: ||
- Range operators are shortcuts for expressing a range of values
    - The closed range operator (a...b) defines a range that includes a and b.
    - The half-open range operator (a..<b) defines a range that includes a but NOT b

Conditionals

If/else

- if and if/else statements are the same as in other languages.
- The test condition does NOT need to be in parentheses
- if and else bodies MUST be in curly braces, even if it's only 1 line

```
if age<21 {
    print("you're young")
}
else {
    print("you're not so young")
}
```

(change the value you're testing age against and see how the playground changes)

switch

- A switch statement compares a value against possible matching cases
- Case statements can have multiple values separated by commas, or can be a range
- Cases do not automatically fall through so you don't need a break in each case.
- Switch statements must be exhaustive
    - A case for each possible value OR
    - Default case for any values that don't match a case

```
switch age{
     case 0...5: print("You're a wee bitty one")
     case 6...21: print("Enjoy school")
     case 22...55: print("Welcome the real world")
default: print("I don't know what you're doing")
}
```

before the switch statement add

```
age=50
```

(watch the playground change)

Loops

- Swift supports the for and while loops
- Swift 3 removes support for the C-style for loop
- The **for-in** loop is used to iterate over collections of items
- Loops don't need parentheses around the conditional
- No need for 'var' for the counter in a for loop
- Swift has both a while loop and a do-while loop

- While loops evaluate its condition at the start of each pass through the loop
- Repeat-while loops evaluate its condition at the end of each pass through the loop

```
for count in 0...age
{
    print("\(count)")
}
```

Optionals

One of the unique aspects of Swift is the concept of optionals(book page 800)

- Defining a variable as an optional says it might have a value or it might not
- If it does not have a value it has the value nil
  - Nil is the absence of a value
- Optionals of any type can have the value nil
- A '?' after the type indicates it's an optional

```
var score : Int?
print("Score is \(score)")
score=80
print(score)
print("score is \(score!)")
```

To access the value of an optional you must add an '!'. This is called forced unwrapping.
If you force unwrap an optional that is nil your program will crash so you should use an if statement to find out if an optional has a value before unwrapping it.

```
if score != nil {
    print("The score is \(score!)")
}
```

This is so common in Swift that there's a shorthand for it called optional binding. You can conditionally unwrap an optional and if it contains a value, assigns it to a temporary variable or constant.

```
if let currentScore = score {
    print("My current score is \(currentScore)")
}
```

Sometimes it's clear that an optional will always have a value, after the first value is set
We can unwrap these optionals without the need to check it each time
These are called implicitly unwrapped optionals
'!' after the type indicates it's an implicitly unwrapped optional
Implicitly unwrapped optionals should not be used when there is a possibility of a variable becoming nil at a later point.

```
let newScore : Int! = 95
print("My new score is \(newScore)")
```

- No "!" is needed to access the optional because it's an implicitly unwrapped optional

Functions

Functions provide a way to group a set of instructions that perform a specific task.

- Keyword func

- Function name

```
func sayHi () {
    print("Hello class")
}
```

Call the function
```
sayHi()
```

Parameters are named and typed(can't be inferred like variables)
```
func sayHello (first: String, last: String){
    print("Hi \(first) \(last)")
}
```

When calling a function with parameters you don't have to name the first parameter but you must name all subsequent parameters.
```
sayHello("Bill", last: "Adams")
```

Functions can also have external parameter names to be used when the function is called

- Place the external parameter name before the local one
- If no external parameter is supplied it's taken to be the same as the internal parameter name
- An underscore in the parameter list means there's no external parameter name so you don't need to supply one when calling the function
    - You will see this in the SDK methods a lot

```
func sayWhat (firstName first: String, lastName last: String){
    print("What \(first) \(last)?")
}
```

Call a function using the external parameter name.
```
sayWhat(firstName: "Bill", lastName: "Adams")
```

Functions can also return a value

- List the return type with an arrow after the parameter list
- You can use a tuple to return multiple values

```
func sayWho(#firstName : String, #lastName : String) -> String {
                return "Who" + firstName + " " + lastName + "?"
}

let msg2=sayWho(firstName: "Jim", lastName: "Adams")
print(msg2)
```

Comments

- Comments are used to include notes, citations, or explanations in your code.
- // is used for single line comments
- /* comment */ is used for multi-line comments

```
// that's all folks!
```