Introduction to Swift

Dan Goldsmith

Introduction

About Me

- Dr Daniel Goldsmith
- Lecturer in Ethical Hacking and Cyber Security.
- Linux User!

About Me 2

- Background is Pervasive Computing
  - Security of Wireless Sensor Networks
  - Reverse engineering
  - Radio's

Lectures:

- I Don't Like Lectures!
  - Standing up and talking for hours is boring :(
  - Also Programming is Practical.
- So we have a mix of practical and Talking

Swift Language

The Swift Language

- Developed in 2010 by Chris Lattner
- Improves Objective-C
- Swift 3.0 in 2016

More About Swift

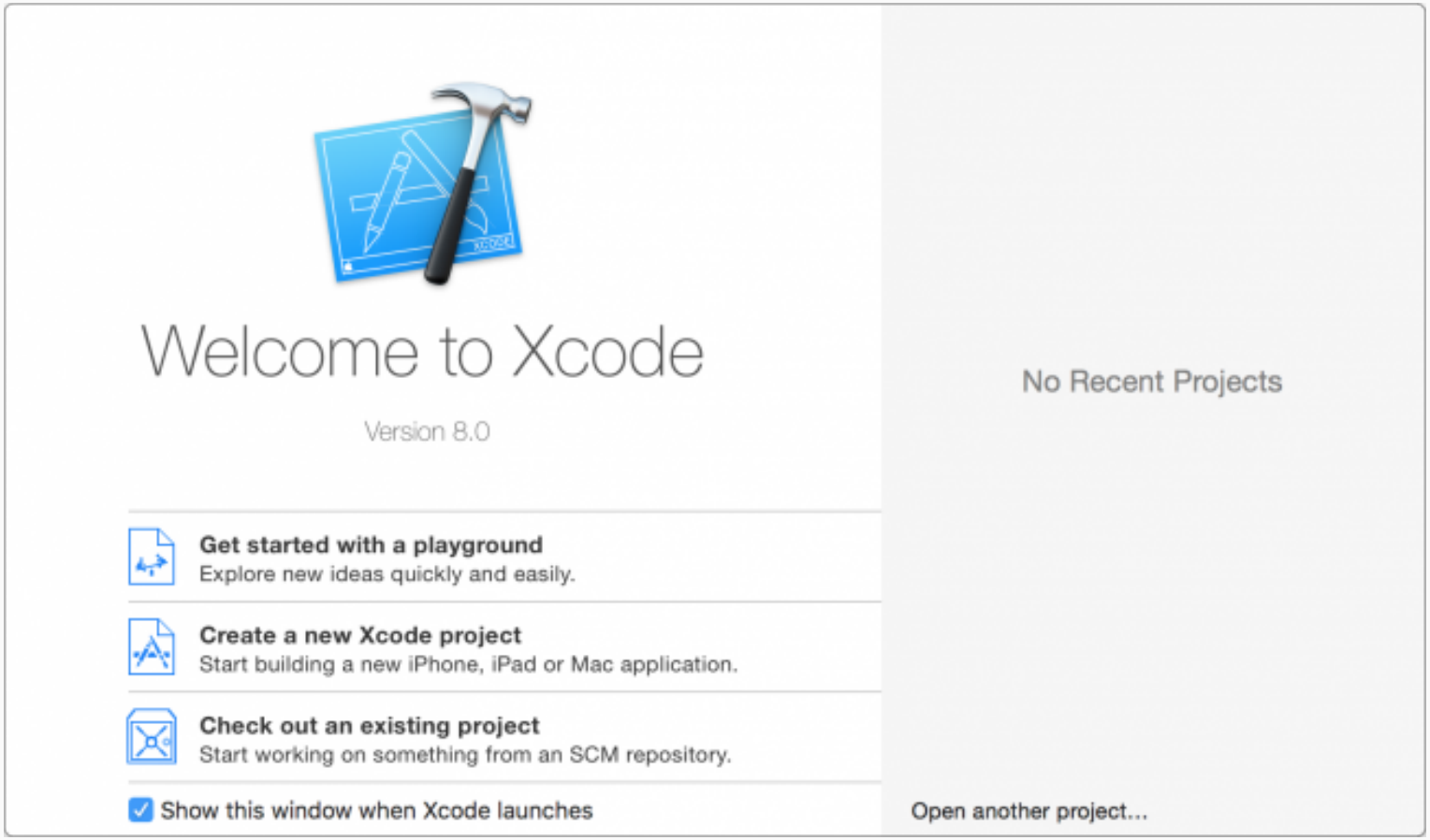
Swift is a new programming language for iOS, macOS, watchOS, and tvOS apps that builds on the best of C and Objective-C, without the constraints of C compatibility.

Swift and X-Code

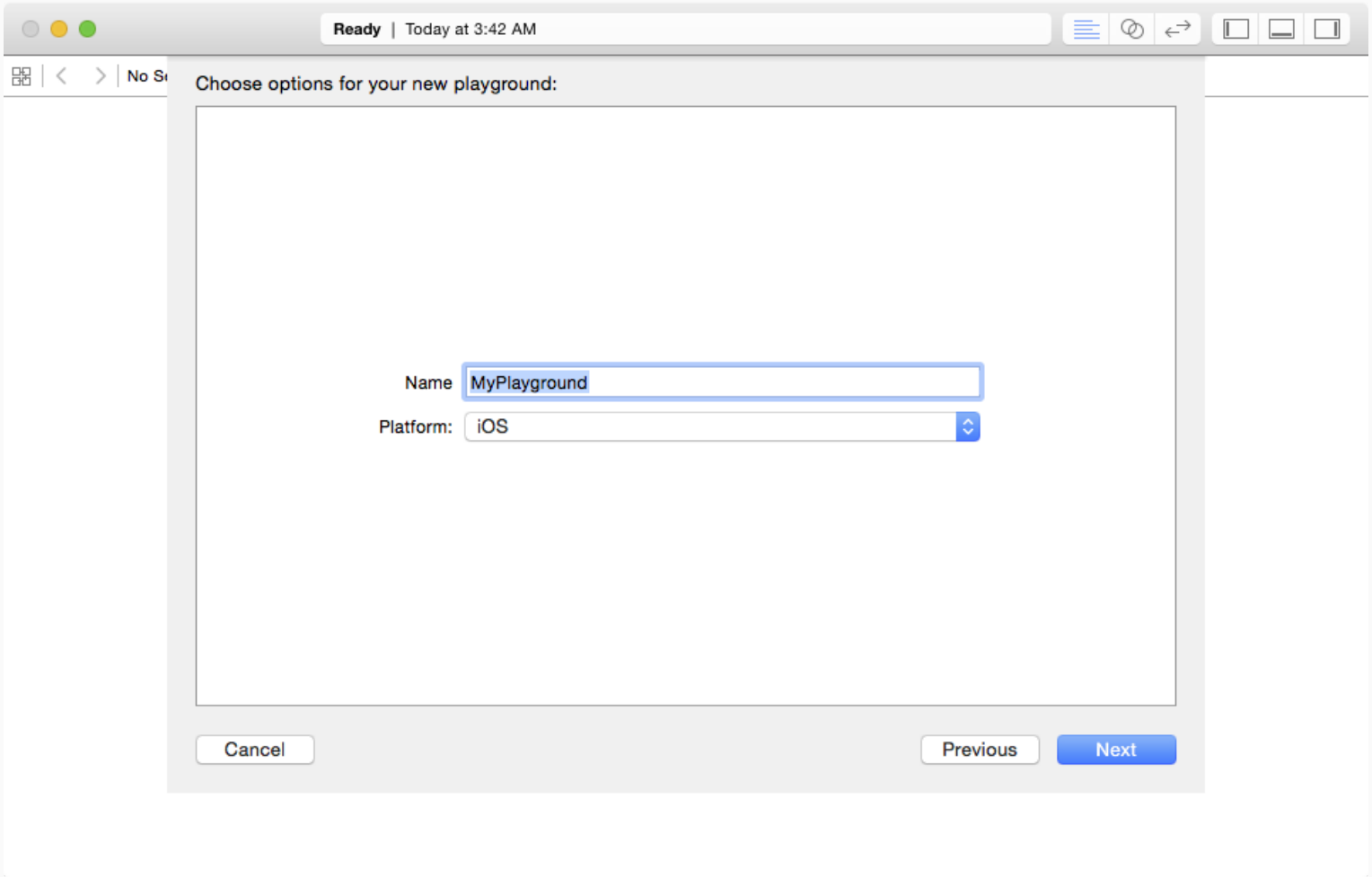
- Two Independent things:
  - X-Code is the IDE
  - Swift is the Language.

Lets Get Started:

- Start X-Code
- Create a new Playground



Playground Options



Initial Code

Writing Code Documentation

Comments

- Comments are super important
  - Let others understand your code
  - Let YOU understand your code
- Follows the "Principle of Minimal Surprise"

Comments in Swift

- Single Line Comment

```
1  // This is a comment
```

- They can also stack

```
1  // First Line of Comment
2  // Second Line of comment
```

Multiline Comments

- But a better way is:

```
1  /* This is also a comment.
2     Over many..
3     many...
4     many lines. */
```

Documenting Code

- You do not need to comment every line
  - Some stuff should be self-explanatory
- Try to capture the Logic
  - WHY did you do something
  - WHAT does a particular class / function do.

Documenting Code: Example

```
1  // we create a new Todo object
2  var newList = Todo()
3  // we now call the addItem method to add two strings to the list
4  newList.addItem("Cheese")
5  newList.addItem("Milk")
```

Documenting Code: Functions

```
1       /// adds a new item to the list
2       /// - parameters:
3       ///   - Int: The index of the list item to be returned.
4       /// - throws: A `TodoError.indexOutOfRange` error, if the index is invalid.
5       /// - returns: A string containing the list item.
6       func getItem(atIndex index:Int) throws -> String {
7           ...
8       }
```

Getting Started

Hello World!

- The Traditional first program
- Type in the following, and click the run button.

```
1  //print Hello to the screen
2  print("Hello, world!")
```

Some things to note:

- Depending on languages you are familiar with (C,Java,C++):
  - No need to import libraries
  - No Semicolons at the end of each line

Core Variables

Constants and Variables:

- Constants: Defined Once, cannot change
- Variables: Can have different values

Constants and Variables:

```
1  //A Constant
2  let pi=3.14
3
4  //A Variable
5  var radius=5.0
6
7  //And another
8  var circumference = 2 * pi * radius
```

Constants and Variables: 2

- Lets Change some values

```
1  //Constant
2  let pi=3.14
3
4  //Variable
5  var radius=5.0
6
7  //Change the Variable
8  radius = 2.0
```

Constants and Variables: 3

- Lets break something

```
1  //Constant
2  let pi=3.14
3
4  //Variable
5  var radius=5.0
6
7  //Try to change the constant
8  pi = 3.14
```

What about Types?

- If we define an initial value swift is clever enough to work out the type.
- However, some times we need to define it ourselves

```
1  // Round numbers
2  var number: Int = 10
3  // Decimal Numbers
4  var decimal: Double = 3.14
5
6  // Text
7  var text: String = "Hello World"
8
9  //Boolean
10 var status = true
```

More Types

```
1  //Multiple definions
2  var decimalOne, decimalTwo: Double
3
4  decimalOne = 5.0
5  decimalTwo = 22.5
```

Gotchas: Integers

- Integer Only Arithmetic. Drops decimal numbers:
  - What Happens, How do we fix it

```
1  var number = 22
2  var value = 7
3  //This is 3
4  var output = number / value
```

Gotchas: Type Safety

- Values types are Locked once defined
  - Swift will not allow you to pass a float to a string etc.
- What happens here, How do we fix it?

```
1  //Define a string
2  var text: String
3  //Try to set a value
4  text = 3.14
```

Gotchas: Conversion

- By default no conversion is performed
  - Means we have difficulties combining items and need to cast it to the correct type
  - Try the following with and without the conversion.

```
1  //Define our number
2  var number = 42
3  //And build a string from it
4  var text = "The meaning of Life is " + String(number)
```

Converting Strings (2)

- These is an even easier way to convert strings
  - Use \(...)

```
1  //Define Number
2  var number = 42
3  //Build String
4  var text = "The Meaning of Life is \(number)"
```

Arithmatic

- Normal Maths applies

```
1  //Add Numbers
2  var number = 4+4
3  //Subtract
4  var number = 4-2
5  //Divide
6  var number = 4/2
7  //Multiply
8  var number = 4*2
```

Running Totals

- We can also add to existing numbers

```
1  //Define
2  var number = 4
3  //Add 5 to the number
4  number = number + 5
5  //Or Shorter version
6  number += 5
```

Printing Things

- Sometimes we want output

```
1  //Define Number
2  var number = 42
3  //Build String
4  var text = "The Meaning of Life is \(number)"
5  //Print to screen
6  print(text)
```

Printing Things (one liner)

- We can also do this without the intermediate Variable

```
1  //Define number
2  var number = 42
3
4  //Print
5  print ("The meaning of life is \(number)")
```

Your Turn:

- Type the basic program below

```
1  //Constant
2  let pi = 3.14
3  //Variable
4  var radius = 5
5
6  //Calculations
7  var circumference = 2 * pi * radius
8
9  //Output
10 print("Cirumference of Circle with Radius \(radius) is \(circumference)")
```

Your Turn:

- You have 10 Minutes to Modify the code to:
  - Store your name as a variable
  - Print the area of the Circle (Pi * R^2)
  - Print "Hello <your name>"

Lists and Dictionaries

Collections of Variables

- So far we have looked at primitive variables
- Lists and Dictionaries allow us to deal with groups of objects

Lists

- Allow us to store collections of items
- We use Square Brackets []

```
1  //Define a list
2  var shoppingList = ["Orange", "Water", "USB Drive"]
3
4  //Or an Empty List
5  var emptyList = [String]()
```

Getting data from lists

- We use the List Index
  - Starts at 0 (it does make sense in terms of Memory Management)

```
1  //Define a list
2  var shoppingList = ["Orange", "Water", "USB Drive"]
3
4  //Print the 1st (0th) Item
5  print(shoppingList[0])
6
7  //Change the 2nd value ("water")
8  shoppingList[1] = "Bottle of Water"
```

How many Items

```
1  //Define a list
2  var shoppingList = ["Orange", "Water", "USB Drive"]
3
4  //This should print 3
5  print ("The Size of the shopping list is \(shoppingList.count)")
```

Adding Items

```
1  //Define a list
2  var shoppingList = ["Orange", "Water", "USB Drive"]
3
4  //Add an Item
5  shoppingList.append("Book")
6
7  //What happens if we want to add it at a specific place
8  shoppingList.append("Beer", at: 0)
```

Removing Items

```
1  //Define a list
2  var shoppingList = ["Orange", "Water", "USB Drive"]
3
4  //Remove the First item
5  shoppingList.remove(at: 0)
```

Printing all items in an List

- We can also Iterate over the array

```
1  //Define a list
2  var shoppingList = ["Orange", "Water", "USB Drive"]
3
4  for item in shoppingList {
5      print(item)
6  }
```

Dictionaries

- Allow us to store items as "Key":"Value" pairs
  - We then access the item using the Key

```
1  //Define a dictonary
2  var occupations = ["Dan" : "Lecturer", "James": "Senior Lecturer"]
3
4  //And print some values (Will print "Lecturer")
5  print(occupation["Dan"])
6
7  //Give James a Promotion
8  occupation["James"] = "Professor"
```

Adding Items to dictionaries

- NOTE: The change to the layout

```
1  //Define a dictonary
2  var occupations = ["Dan" : "Lecturer",
3                     "James": "Senior Lecturer",
4                    ]
5  //Add a new person
6  occupations["Mark"] = "Teaching Assistant"
```

Iterating over dictionaries

```
1  //Define a dictonary
2  var occupations = ["Dan" : "Lecturer",
3                     "James": "Senior Lecturer",
4                    ]
5
6  //Either have items returned as a tuple
7  for value in occupations {
8      print("Tuple is \(value)")
9  }
10
11 //Or decompose the tuple
12 for (name, job) in occupations {
13     print("\(name): works as a \(job)")
14 }
```

Your Turn: Reference

```
1  //Create an empty List
2  var shoppingList = ["Orange", "Water", "USB-Drive"]
3  //How many items in the list
4  print ("List Has \(shoppingList.count) items")
5  /Add an Item
6  shoppingList.append("Apple")
7  //And print it out
8  for item in shoppingList{
9      print ("Item in list \(item)")
10 }
```

Your turn

- Create a new list of numbers Grades and populate it with some scores
- Add a new grade to the list
- Iterate through the list and print all the grades
    - BONUS: Using another variable, try to calculate the average grade

Solution

Hopefully you have something like this

```
1  //Create a grades object
2  var grades = [70,65,72,50]
3
4  //Add a new grade
5  grades.append(70)
6
7  //Something to hold our total (Note its a floating point)
8  var total = 0.0
9
10 for item in grades{
11     total += item
12 }
13
14 print("Average grade is \(total / grades.count)")
```

Selection and Iteration

Selection and Iteration

- So far we have introduced variables, and some more complex data structures
- To write useful programs we need to do something with them
  - Selection: Choosing what to do based on an input
  - Iteration: Doing something many times

Conditions

- We have several conditions we can evaluate against
  - == Equal To
  - != Not Equal To
  - > Greater Than
  - < Less Than
  - >= Greater or Equal to
  - <= Less or Equal to

Conditions

```
1  5 == 5  //True
2  4 == 5  //False
3  10 > 5  //True
4  10 < 5  //False
5  5 >= 5  //True
```

Selection

- If condition is met, then do something

```
1  var value = 10
2
3  if value > 5 {
4      print("Value is Greater than 5")
5  }
```

Selection: Providing an alternative

- We can use Else

```
1  var value = 10
2
3  if value > 5 {
4       print("Value is Greater than 5")
5  }
6  else {  //Otherwise
7       print("Value is less than 5")
8  }
```

Selection, Multiple Choice

- Note Order is important here

```
1  value = 10
2  if value == 5 {
3      print("Value is equal to 5")
4  }
5  else if value > 5 {
6      print("Value is greater than 5")
7  }
8  else {
9      print("Value is less than 5")
10 }
```

Selection Task

- Lets write a (broken) grade calculator
  - Try running with different values for grade, what happens?
  - Can you fix the code to work correctly

```
1  var grade = 55
2
3  //Fail
4  if grade < 40 {
5       print ("Sorry, you failed")
6  } else if grade > 70 {
7       print ("Congratulations you got a 1st")
8  } else if grade >= 40{
9       print ("That sucks, a 3rd")
10 } else if grade >= 50 {
11      print ("OK, a 2:2")
12 } else if grade >= 60 {
13      print ("Not bad, a 2:1")
14 } else { //Catch things outside of expected range
15      print ("Grade outside of boundries")
16 }
```

More Selection

- We can also use Switch statements to achieve the same aim
  - Again, try the code. Does it need fixing.

```
1  var grade = 55
2
3  switch grade{
4      case 0..<40:
5          print ("Sorry, you failed")
6      case 70..<=100:
7          print ("Congratulations, a 1st")
8      case 60..<70:
9          print ("Not bad, a 2:1")
10     case 40..<50:
11         print ("That Sucks, a 3rd")
12     case 50..<60:
13         print ("OK, a 2:2")
14     default:  //Catch all
15         print ("Grade outside of boundries")
16 }
```

Iteration:

- Allows us to do things many times.
  - Go through the items in a list
  - repeat a task a given number of times
  - repeat a task until a condition is met

For and While Loops:

- FOR when we know how many items there are
  - Items in a list
  - Do things a set number of times
- WHILE stop when a condition is met
  - While we are still getting user input
  - To keep doing something until told to stop.

For Loops (1)

- We have already met some for loops (called for-in loops):
  - Iterate through items in the list

```
1  //Define List
2  var thelist = ["foo","bar","baz"]
3  //For - In loop
4  for item in thelist {
5       print(item)
6  }
```

For Loops (2)

- We can also define a range of numbers to use

```
1  //A Range between 0 and 5
2  for index in 0..<5 {
3      print ("Index is \(index)")
4  }
```

For Loops (3)

- We can use the index to access items in a list
  - This is the longhand version of the for-in loop

```
1  //Define List
2  var thelist = ["foo","bar","baz"]
3
4  //Indexed For loop
5
6  for index in 0..<thelist.count {
7      print ("Item at Index \(index) is \(thelist[index])")
8  }
```

While Loops:

- Sometimes we dont know the number of items we need to deal with
- In this case we use a WHILE loop
  - WHIlE something is true, continue looping
- It is REALLY IMPORTANT to remember to change the condition otherwise you can get infinite loops.

While Loops (2):

- So Lets keep doubling a number

```
1  //Initialise Variable
2  var total=1
3  while total < 25 {
4     print("Total is \(total)")
5     //And add it to iteslf
6     total += total
7  }
```

While Loops (3):

- Using a While as a For
  - Question: Why not <= ?

```
1  //Define List
2  var thelist = ["foo","bar","baz"]
3
4  //and an index
5  var index = 0
6
7  while index < thelist.count {
8      print("Item at index \(index) is \(thelist[index])")
9      index += 1
10 }
```

Your Turn

- Remember the List of Grades?
- Remember the Classification Calculator
- Combine the two:
  - Print the score for each grade
  - Print the final grade classification

Functions / Methods

Functions and Methods:

- So far we have been writing all the code in the global namespace
  - This is a BadThing(TM) as it reduces modularity
  - We have to keep copying chunks of code
  - Leads to the potential for lots of mistakes.

Functions

- Allow us to break the code into "Logical" blocks
- We can then call the function from the code, to make use of it.
- For example, good candidates for functions are:
  - The calculate grade code we used before.
  - The Math we did to calculate parts of a circle.

Defining Functions

- We use the func syntax.

```
1  func <name>(<parameters>) -> <return> {
2     ...
3  }
```

Defining the grade function

- We know that the grade function:
  - Takes a value as input
  - Prints the grade message

Defining the grade function (1)

- The first cut of the grade function looks like this.

```
1  func grade(mark: Double) {
2      if mark < 40 {
3          print ("Sorry, you failed")
4      } else if mark > 70 {
5          print ("Congratulations you got a 1st")
6      } else if mark >= 60 {
7          print ("Not bad, a 2:1")
8      } else if mark >= 50 {
9          print ("OK, a 2:2")
10     }else if mark >= 40{
11         print ("That sucks, a 3rd")
12     } else { //Catch things outside of expected range
13         print ("Mark outside of boundries")
14 }
```

Calling the grade function

- We can then call the function

```
1  var score = 55
2
3  grade(score)
```

Improving the Grade function

- But there are some issues here:
  - Except for debugging Functions shouldnt really print things
  - It is more appropriate to have the function return a value (as it can be used anywhere)

Improving the Grade function

```
1  func grade(mark: Double) -> String {
2      if mark < 40 {
3          return "Sorry, you failed"
4      } else if mark > 70 {
5          return "Congratulations you got a 1st")
6      } else if mark >= 60 {
7         return "Not bad, a 2:1"
8      } else if mark >= 50 {
9          return "OK, a 2:2"
10     }else if mark >= 40{
11         return "That sucks, a 3rd"
12     } else { //Catch things outside of expected range
13         return "Mark outside of boundries"
14 }
```

Calling the improved grade function

```
1  mark = 55
2
3  var result = grade(mark)
4  print (result)
```

Documenting the Grade Function

- We should also document our grade function

```
1  func grade(mark:Double) -> String{
2      /* Convert a students grade into textual feedback
3         - parameters:
4             - mark:  Double representing the students numerical mark
5         - returns: A String representing text based feedback
6      */
7      if mark < 40
8          ...
9  }
```

Dealing with multiple parameters

- We can specify multiple parameters to a function

```
1  func area(pi: Double, radius: Double) -> Double {
2      /* Calculate the Area of a circle
3         - parameters:
4           - pi:  Value of Pi
5           - radius: Radius of cicle
6         - return:  The circles area
7         */
```

Functions: Your Turn

```
1  //Value for Pi
2  let pi = 3.14
3
4  //A List of Circles
5  var circles = [1.0, 2.0, 5.0, 10.0]
6
7  func area(pi: Double, radius: Double) -> Double {
8      /* Calculate the Area of a circle
9         - parameters:
10          - pi:  Value of Pi
11          - radius: Radius of cicle
12        - return:  The circles area
13        */
14     return pi * (radius * radius)
15 }
16
17 //Create a function to calculate and return the Circumference (2*pi*r)
18
19 //Get the program to calculate and print the Radius for each of the circles
```

Classes

Classes

- So Far our code has had no Class :)
- Classes are a way of abstracting behaviour and are core to OO programming.
- Classes represent a "thing" in our program
    - People
    - Shapes
    - Courses

Defining Classes

- We can use the class keyword

```
1  class Person {
2    ...
3  }
```

Creating Objects

- Instances of each class are known as Objects
- We can create them by putting parenthesis after the name

```
1  var Dan = Person()
```

Class Variables

- Class's also have attributes,
- These are the variables that make the class unique
- For example a person could have:
  - First (Given) Name
  - Last (Family) Name
  - Age

Adding Class Variables

```
1  class Person{
2      /* Defines a Person */
3      var givenName: String
4      var familyName: String
5      var age: Int
6  }
```

Accessing Class Variables

- Use Dotted Syntax

```
1  class Person{
2      /* Defines a Person */
3      var givenName: String
4      var familyName: String
5      var age: Int
6  }
7
8  //Create a person object
9  var Dan = Person()
10 //Set variables
11 Dan.givenName = "Daniel"
12 Dan.familyName = "Goldsmith"
13
14 //Print my Name
15 print("Full Name is \(Dan.givenName) \(Dan.familyName)")
```

Constructors

- Using Dotted syntax is clumsy when creating objects
- Instead we use Constructors
  - The special init method.
  - Takes parameters and is used to set variables

Constructors

```
1  class Person{
2      /* Defines a Person */
3      var givenName: String
4      var familyName: String
5      var age: Int
6
7      init(givenName: String, familyName: String){
8          //Create a new Person with provided names
9          //Note the use of Self to differentiate between class and parameters
10 self.givenName = givenName
11 self.familyName = familyName
12 }
13
14 var Dan = Person("Daniel","Goldsmith")
```

Constructors without the self

```
1  class Person{
2      /* Defines a Person */
3      var givenName: String
4      var familyName: String
5      var age: Int
6
7      init(given: String, family: String){
8          //Create a new Person with provided names
9          //Note the parameters are less readable
10 givenName = given
11 familyName = family
12 }
13
14 var Dan = Person("Daniel","Goldsmith")
```

Class Functions

- Each class will have a set of functions associate with it
- These can access the class variables to perform tasks
- Defined in a similar way to normal functions

Class Functions

```
1   class Person{
2       /* Defines a Person */
3       var givenName: String
4       var familyName: String
5       var age: Int
6
7       init(given: String, family: String){
8           //Create a new Person with provided names
9           //Note the parameters are less readable
10  givenName = given
11  familyName = family
12  }
13
14      func getName() -> String {
15          //No parameters,  Return full name as string
16          return "\(givenName) \(familyName)"
17      }
```

Calling Functions

- Call the function by using <object>.<function>

```
1  var Dan = Person("Daniel", "Goldsmith")
2
3  var theString = Dan.getName()
4  print(theString)
```

Your Turn

- Its time to make some Shapes
  - Create Classes for three different shapes (ie Square, Rectangle, Triangle)
  - Each Shape should have functions that return its Area, and Circumference.
  - Test the Shape functions out. Make sure they work.