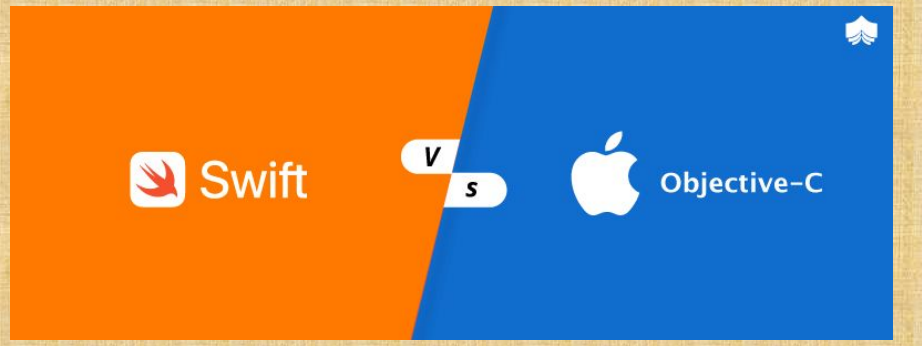
iOS Programming

Lecture 2

Recap

Swift vs Objective C



Language vs Framework

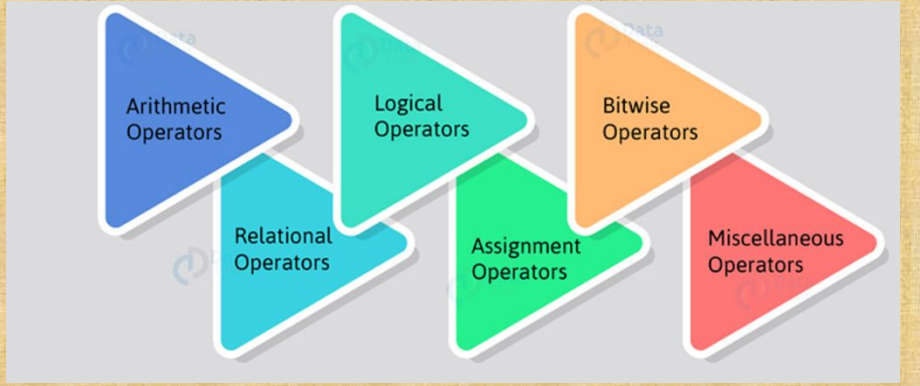
vs

XCode to create playground

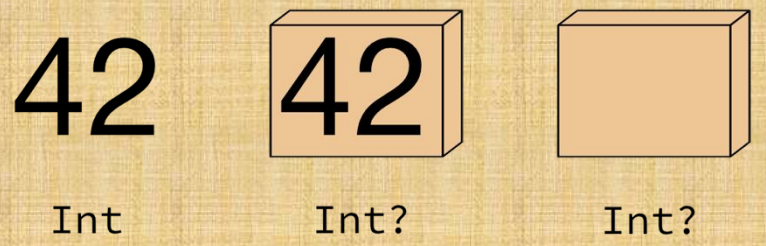
Today

Variables and Constants

Operators



Optionals



Swift - Variables

How to define

```
var questionNum = 1
var question = "Which language are we covering in this course?"
var isAnswered = false
var answerAttempted: String
```

Type Inference

```
var questionNum = 1
var question = "Which language are we covering in this course?"
var isAnswered = false
```

Type Annotation

```
var answerAttempted: String
```

Swift – Data Types

Data Type	Explanation	Variant
Int	This is used for whole numbers.	Int32, Int64, UInt32, UInt64
Float	This is used to represent a 32-bit floating-point number and numbers with smaller decimal points.	N/A
Double	This is used to represent a 64-bit floating-point number and used when floating-point values must be very large.	N/A
Bool	This represents a Boolean value which is either true or false.	N/A
Character	This is a single-character string literal. For example, "A"	N/A
String	This is an ordered collection of characters. For example, "Hello, World!"	N/A
Optional	This represents a variable that can hold either a value or no value.	N/A
Tuples	This is used to group multiple values in single Compound Value.	N/A

Swift – Variables – Swifty Code

What does it means for the code to be Swifty enough?

No semi colons

```
var questionNum = 1
```

Variables start with small case and Types with capital

```
var answerAttempted: String
```

Variable and colon has no space but we have a space between colon and type name.

Swift – Variable Scope

What do we mean by scope of variables?

```
var questionNum = 1
var question = "Which language are we covering in this course?"
var isAnswered = false
var answerAttempted: String

func demonstrateLocal(){
   var innerVar = "I am local"
   print(innerVar)
}

print(question)
print(innerVar)
```

Use of unresolved identifier 'innerVar'

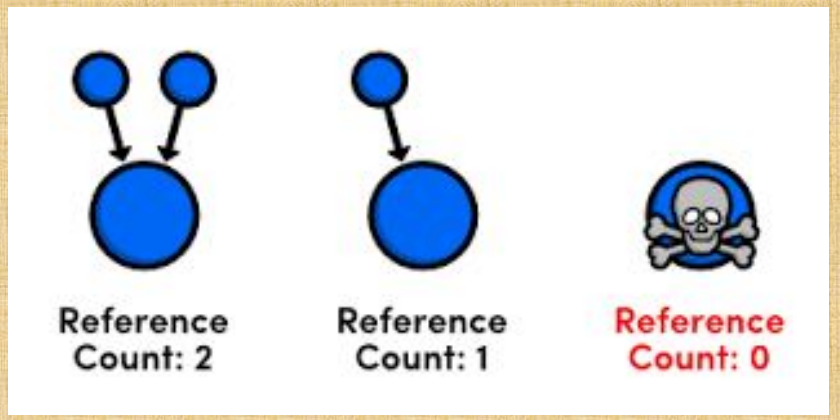
Why the error?

Swift – Variable Scope

Why is scope important?

Automatic Reference Counting

Swift



Keep track of when the memory can be reclaimed.

Swift – Constants

Variable - var

```
var count = 0
//Let's change count to 1
count = 1
```

Constant - let

```
let minCount = 1
//Let's change min count to 0
minCount = 0
```

Cannot assign to value: 'minCount' is a 'let' constant

Swift – Constants – Swifty Code

Clear

Explicit

In Swift world, you will see lots of constants being used and that is intentional.

Compile Time Errors

Swift – Constants

- Swift encourages to the extent of issuing compile time warnings if a variable makes sense to be constant.
- Why?? – Swift compiler does memory optimizations for let vs var so there is objective improvement.

```
func demonstrateVarVsLet(){
    var sayHello = "Say Hello"
                   ⚠ Variable 'sayHello' was never mutated; consider changing to 'let'...
    print(sayHello)
}
```

Swift – Constants – One more thing

Constants and Variables must be initialized before use. But, it doesn't means that constants need to be initialized at definition. As long as we have type annotation performed for a constant we are good. We can later read the constant value from a network response or a local file.

```
let maxCounters: Int
//Let's do some network calls and get the value of maxCounters
maxCounters = 99
```

Swift – Constants & Variables

Must be initialized before first use. Otherwise, the swift compiler will catch it for us.

```
let maxCounters: Int
print(maxCounters)
```

Constant 'maxCounters' used before being initialized

```
//Let's do some network calls and get the value of maxCounters
maxCounters = 99
```

Swift – Operators – Basic Arithmetic

Let's start with basic arithmetic operators. For examples below, let's say X=10 and Y=2

Operator	Description	Example
+	Adds two operands	X + Y = 12
–	Subtracts second operand from the first	X – Y = 8
*	Multiplies both operands	X * Y = 20
/	Divides numerator by denominator	X / Y = 5
%	Modulus Operator and remainder of after an integer/float division	X % Y = 0

Swift – Operators – Type Inference

When we do these arithmetic operations, what's the expected type?

Can you answer this?

```
let x = 10
let y = 3
let z = x / y
```

How can we be sure?

```
type(of: z)
```

In Swift there is no implicit conversion between data types.

Swift – Operators – Type Conversion

Now, if I want to convert types – what should I do?

```
let x = 10
let y = 3

//It's as easy as telling swift what type you want
let z = Double(x) / Double(y)
```

Swift – Comparison Operators

In the example below, let's say X=10 and Y=2

Operator	Description	Example
==	Checks if the values of two operands are equal or not; if yes, then the condition becomes true.	(X == Y) is not true.
!=	Checks if the values of two operands are equal or not; if values are not equal, then the condition becomes true.	(X != Y) is true.
>	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	(X > Y) is true.
<	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	(X < Y) is not true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.	(X >= Y) is true.
<=	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	(X <= Y) is not true.

Swift – Comparison Operators

An example where it is handy

```
var scoredPercentage = 70.3
var passingPercentage = 70.0

if (scoredPercentage >= passingPercentage){
    print("Congratulations!!!! You have passed")
}
```

Swift – Logical Operators

In the example below, let's say X=1 and Y=0

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(X && Y) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(X || Y) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.	!(X && Y) is true.

Swift – Logical Operators

An example where it is handy

```
var scoredPercentage = 60.3
var passingPercentageCoreExam = 70.0
var passingPercentageSecondaryExam = 60.0
var coreExam = false


if ((coreExam && scoredPercentage >= passingPercentageCoreExam) ||
(!coreExam && scoredPercentage >= passingPercentageSecondaryExam)){
    print("Congratulations!!!! You have passed.")
}else{
    print("Sorry!!!! Better luck next time.")
}
```

Swift – Bitwise Operators

Bitwise operators work on bits and perform bit by bit operation. The truth tables for &, |, and ^ are as follows –

p	q	p&q	p|q	p^q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Swift – Bitwise Operators

Bitwise operators supported by Swift 4 language are listed in the following table. Assume variable A holds 60 and variable B holds 13, then 7−

Operator	Description	Example
&	Binary AND Operator copies a bit to the result, if it exists in both operands.	(A & B) will give 12, which is 0000 1100
|	Binary OR Operator copies a bit, if it exists in either operand.	(A | B) will give 61, which is 0011 1101
^	Binary XOR Operator copies the bit, if it is set in one operand but not both.	(A ^ B) will give 49, which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61, which is 1100 0011 in 2's complement form.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	(A << 2 will give 240, which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15, which is 0000 1111

Swift – Bitwise Operators

& - Both values have to be 1

| - One of the values need to be 1 at least

^ - Exclusive OR – One and only one of the values must be 1

```
let x = 2     //00000010
let y = 21    //00010101
var z  = x&y  //00000000
print(z)      //Answer is 0

z = x|y       //00010111
print(z)      //Answer is 23

z = x^y       //00010111
print(z)      //Answer is 23
```

Swift – Assignment Operators

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assigns the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assigns the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assigns the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assigns the result to left operand	C %= A is equivalent to C = C % A

Swift – Assignment Operators

Operator	Description	Example
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Swift – Assignment Operators

Assignment Operator

```
var stepCount = 2
stepCount += stepCount
print(stepCount)
```

Another example

```
var stepCount = 2
stepCount *= stepCount
print(stepCount)
```

Swift – Range Operators

Range operators are shortcuts for expressing a range of values

Operator	Description	Example
Closed Range	(a...b) defines a range that runs from a to b, and includes the values a and b.	1...5 gives 1, 2, 3, 4 and 5
Half-Open Range	(a..< b) defines a range that runs from a to b, but does not include b.	1..< 5 gives 1, 2, 3, and 4
One-sided Range	a… , defines a range that runs from a to end of elements …a , defines a range starting from start to a	1… gives 1 , 2,3… end of elements …2 gives beginning… to 1,2

Swift – Range Operators

Closed Range

```
for counter in 1...5 {
    print("Counter loop index is \(counter)")
}
```

Output

```
Counter loop index is 1
Counter loop index is 2
Counter loop index is 3
Counter loop index is 4
Counter loop index is 5
```

Swift – Range Operators

Half-Open Range

```
for counter in 1..<5 {
    print("Counter loop index is \(counter)")
}
```

Output

```
Counter loop index is 1
Counter loop index is 2
Counter loop index is 3
Counter loop index is 4
```

Swift – One Sided Range

One sided Range

```
let numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for counter in numbers[8...] {
    print("Counter loop index is \(counter)")
}
print("---------------------------")
for counter in numbers[...2] {
    print("Counter loop index is \(counter)")
}
```

Output

```
Counter loop index is 8
Counter loop index is 9
---------------------------
Counter loop index is 0
Counter loop index is 1
Counter loop index is 2
```

Swift – Misc Operators

Misc operators are operators including numeric toggle and ? : that are explained in the following table.

Operator	Description	Example
Unary Minus	The sign of a numeric value can be toggled using a prefixed -	-3 or -4
Unary Plus	Returns the value it operates on, without any change.	+6 gives 6
Ternary Conditional	Condition ? X : Y	If Condition is true ? Then value X : Otherwise value Y

Swift – Ternary Operator

Ternary operator usage is shown below

```
var isCool = true
print( isCool == true ? "Swift Development is cool!!!" :  "Swift Development is not what i thought?")
```

Ternary operator is a shorthand. Eg: Above represents below code

```
var isCool = true
if isCool == true {
    print ("Swift Development is cool!!!")
}else{
    print ("Swift Development is not what i thought?")
}
```

Swift – Operator Precedence

Operator	Description	Example
Primary Expression Operators	() [] . expr++ expr--	left-to-right
Unary Operators	* & + - ! ~ ++expr --expr * / % + - >> << < > <= >= == !=	right-to-left
Binary Operators	& ^ | && ||	left-to-right
Ternary Operator	?:	right-to-left
Assignment Operators	= += -= *= /= %= >>= <<= &=^= |=	right-to-left
Comma	,	left-to-right

Swift – Optionals

In Swift variables and constants are not initialized to default values

If you have used some other languages like Java then you are used to seeing global members being initialized to defaults like Booleans to false, Strings to null.

This is not true in Swift. Swift is explicit, hence you must initialize a variable or constant before use otherwise you will get a compile time error. Not a run time but a compile time error itself.

Swift – Optionals - Type

```
var middleName: String?
```

What's the type of String? It is Optional String

```
var age: Int?
```

What's the type of Integer? It is Optional Int

Swift – Optionals – Example#1

Let’s look at an example. Some of our customers will have middle name and some won’t

```
var middleName: String
//But Middle Name may exist?
print(middleName)
```

Variable 'middleName' used before being initialized

How to address? Use Optional String instead of String, that means String is allowed to have nil value

```
var middleName: String?
//But Middle Name may exist?
print(middleName)
```

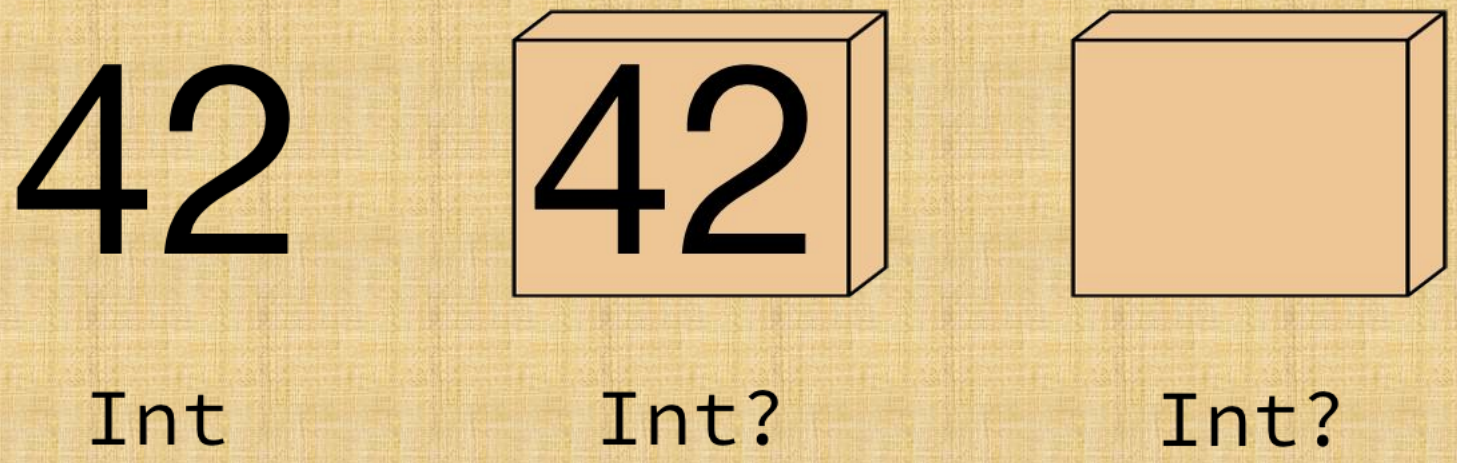
Swift – Optionals – Example#2

Look at the below code and tell me what's the expected output?

```
var name = "Disney"
var disneyName = name
print("Disney's name is: \(disneyName!)")
print("Default name is: \(name!)")
name = nil
name = "Marvel"
print("Disney's name is: \(disneyName!)")
print("Default name is: \(name!)")
```

Swift – Optionals – Unwrapping

Conceptually to create an optional you are talking a data type like Int or String and putting it in a container of Optional.



Now, if you want to use it then you must un-wrap the container first.

Swift – Optionals – Unwrapping

Let’s consider simple arithmetic operation on optional

```
var currentIndex: Int?
currentIndex = 3
var nextIndex = currentIndex + 1
```

Value of optional type 'Int?' must be unwrapped to a value of t...

This is because we know current Index is an optional Int so may be devoid of value and if so then the addition operation doesn’t makes sense.

Swift – Optionals – Unwrapping

Solution #1

This is most verbose and least used.

```
var currentIndex: Int?
currentIndex = 3
if currentIndex != nil {
    //this is forced un-wrapping
    let nextIndex = currentIndex! + 1
    print(nextIndex)
}
```

Swift – Optionals – Unwrapping

Solution #2 – Optional Binding

Swifty Way

This is the swift way of doing it

```
var currentIndex: Int?
currentIndex = 3
if var nextIndex = currentIndex {
    nextIndex += 1
    print(nextIndex)
}
```

Parting Notes

In world of software use the below loop:

1. Understand Theory
2. Practice – Write Code
3. Identify Gaps
4. Repeat Step 1 through 3 – Iterate

- Do create your variables and constants
- Do practice with ALL operators
- Do practice and understand Optionals – they are everywhere.