GENERAL ASSEMBLY

MOBILE DEVELOPMENT

LESSON 04
OPERATORS, OPTIONALS, AND FUNCTIONS

Arthur Ariel Sabintsev
Lead Mobile Architect, ID.me

A BRIEF COMMENT ABOUT HOMEWORK

FIXING GITHUB ISSUES (FOR REAL THIS TIME)

LESSON 03 REVIEW

WHAT DID WE LEARN IN LESSON 03?

- Nomenclature
  - Defintion of Syntax and Source Code
- Swift and Playgrounds
  - Fundamental Data Types
  - Printing to the Console
  - Operators
  - Control Flow

QUESTIONS

- What does Syntax mean?
- What does Source Code mean?
- What is a keyword in Swift?
- What's the difference between a let and var?
- What's the difference between mutability and immutability?
- What does the modulo operator (%) do?
- What do the ++ and — operators do?
- When do you use an if-else statement?
- When do you use a while loop?
- When do you use a for-in loop?

LESSON 03
IN-CLASS ASSIGNMENT REVIEW

LESSON 04
LEARNING OBJECTIVES

LEARNING OBJECTIVES

- Operators Continued
  - Unary
  - Binary
  - Ternary
- Optionals
  - Optional Binding
  - Optional Unwrapping
- Functions
  - All different types!

OPERATORS (CONTINUED)

OPERATORS

- Operators perform an action on elements, like let or var.
  - Unary operators operate on one element
  - Binary operators operate on two elements
  - Ternary operators operate on three elements.

UNARY OPERATORS

- Unary Operators (That you already know)
  - You already know about ++ and —

```
1 var x = 5                    5
2 ++x                          6
3
4 var y = 5                    5
5 --y                          4
```

UNARY OPERATORS

- Negative Operator
  - Converts positive to negative and vice versa

```
1 let x = 1                    1
2 -x                           -1
```

UNARY OPERATORS

- Logical Negation or Logical NOT Operator
  - Converts true to false and vice versa

```
1 let x = true
2 !x
```

```
true
false
```

BINARY OPERATORS

- Binary Operators (that you already know)
  - The arithmetic operators (+, -, *, /)
  - The comparison operators (>, >=, <, <=)

BINARY OPERATORS

- Logical AND Operator
  - &&
  - Chains two conditions together. Both must be true for if statement to be true.

```
let x = 5
let y = 10

if (x >= 5) && (y >= 10) {
    println("Both conditions are true")
} else {
    println("At least one condition is
        false")
}
```

```
5
10


"Both conditions are true"
```

BINARY OPERATORS

- Logical OR Operator
  - ||
    - | is called the pipe. To create it, click **Shift** and \ button at the same time.
  - Chains two conditions together. Only one must be true for if statement to be true.

```
let x = 5
let y = 10

if (x >= 5) || (y <= 10) {
    println("At least one condition is
        true")
} else {
    println("Both conditions are false")
}
```

```
5
10


"At least one condition is true"
```

TERNARY OPERATOR

- Ternary Conditional Operator (By Example)

```
1  let x = 5
2  let stringTrue = "Condition is true."
3  let stringFalse = "Condition is false."
4
5  if (x > 0) {
6      stringTrue
7  } else {
8      stringFalse
9  }
10
11 // Same thing as the if-else conditional
12 (x > 0) ? stringTrue : stringFalse
13
14 let z = (x > 0) ? stringTrue : stringFalse
15
16 z
17
```

```
5
"Condition is true."
"Condition is false."


"Condition is true."


"Condition is true."

"Condition is true."

"Condition is true."
```

TERNARY OPERATOR

```
if (condition) {
      condition is true
} else {
      condition is false
}


(condition) ? condition is true : condition is false
```

OPTIONALS (BY EXAMPLE)

OPTIONALS

- Typically, your constants (let) and variables (var) have values.
- There may be a situation where you might not yet know the value of your constants or variables.
- Swift has a feature that allows you to create a variable without setting it equal to a value.
- These constants and variables are called optionals.
- Optionals have two possible states:
  - Have a value and know what it is
  - They are nil, meaning, they have no value.

OPTIONAL BINDING AND FORCED UNWRAPPING

- Optional Binding lets you check to see your optional to see if it
  - has a value
  - is nil

- The concept of optionals can only be learned by example, so let's go to Xcode!

- An article I wrote about Optionals:
  - https://medium.com/arthurs-coding-tips/optionals-in-swift-c94fd231e7a4

IN-CLASS ASSIGNMENT #1

GETTING STARTED

EXERCISE

KEY OBJECTIVE(S)

Create and use a ternary operator. Also, create and use optionals.

TIMING

20 min 1. Code with partner

5 min 2. Debrief

DELIVERABLE

No deliverable. Practice and ask questions.

FUNCTIONS (BY EXAMPLE)

WHAT IS A FUNCTION? (PT. 1)

- A function is a series of repeatable steps
  - Contains a Beginning, Middle, End
  - May contain input (e.g., initial conditions)
    - May contain multiple inputs
  - May contain output (e.g., return value)
    - May contain multiple outputs (e.g., tuple)
  - May contain constants and variables that are visible only inside the function

WHAT IS A FUNCTION? (PT. 2)

- Functions are blocks of code that are runnable from anywhere
- Functions can take parameters and return values
- When a function is called from within our code, code execution steps into the function until it returns
- When defining a function, return stops all execution of the function and kicks you out of the function

DEFINING FUNCTIONS (WITHOUT PARAMETERS)

```
func testFunction() {
    println("Inside a function!")
}
```

```
// Call testFunction() by simply writing testFunction()
testFunction()
```

DEFINING FUNCTIONS (WITH 1 PARAMETER)

```
func aSecondTestFunction(name: String) {
    println(name)
}
```

```
// Call aSecondTestFunction() by:
aSecondTestFunction("Arthur")
```

DEFINING FUNCTIONS (WITH MULTIPLE PARAMETERS)

```
func aThirdTestFunction(name: String, age: Int) {
    println(name)
    println(age)
}
```

```
// Call aThirdTestFunction() by:
aThirdTestFunction("Arthur", 29)
```

DEFINING FUNCTIONS (WITH A RETURN TYPE)

```
func aFourthTestFunction(name: String, age: Int) -> String {
    let statement = "My name is \(name) and I am \(age) years old."
    return statement
}
```

```
// Call aFourthTestFunction() by:
let sentence = aFourthTestFunction("Arthur", 29)
```

FUNCTIONS (WITH OPTIONALS)

```
func aFifthTestFunction(name: String, age: Int?) -> String? {

    var statement: String?
    if let myAge = age {
        statement = "My name is \(name) and I am \(myAge) years old."
    } else {
        statement = "My name is \(name)."
    }

    return statement
}


// Call aFifthTestFunction() by:
let sentenceWithAge = aFifthTestFunction("Arthur", 29)

let sentenceWithoutAge = aFifthTestFunction("Arthur", nil)
```

WHY USE FUNCTIONS?

```
/*
    Area of a Triangle
    Takes two parameters; base and height

    Return the area of a Triangle
*/

func areaOfTriangle(base: Int, height: Int) -> Int {
    let area = (1/2)*base*height
    return area
}
```

COMMON CONVENTIONS

- Descriptive function names
- Keep the contents of your functions under 50 lines (if possible)
- Make your functions abstract

- Two principles to keep in mind:
  - KISS: Keep It Simple, Stupid
  - DRY: Don’t Repeat Yourself

IN-CLASS ASSIGNMENT #2

GETTING STARTED

EXERCISE

KEY OBJECTIVE(S)

Create and use functions with parameters and return values.

TIMING

30 min 1. Code with partner

5 min 2. Debrief

DELIVERABLE

To the best of your ability, complete the provided playground file. If you hit a question you don't feel comfortable with, ask an instructor.

HOMEWORK

HOMEWORK

- You should be close to finishing these chapters:
  - The Basics Chapter
  - Basic Operators Chapter

- At your own pace, read the following:
  - Control Flow chapter in Apple's Swift book
    - Link: Control Flow in the Official Swift Book
  - Functions chapter in Apple's Swift book
    - Link: Functions in the Official Swift Book