# Swift Basics

# Number Guessing Game



## 8.1 Lecture - Introduction to the Number Guessing Game

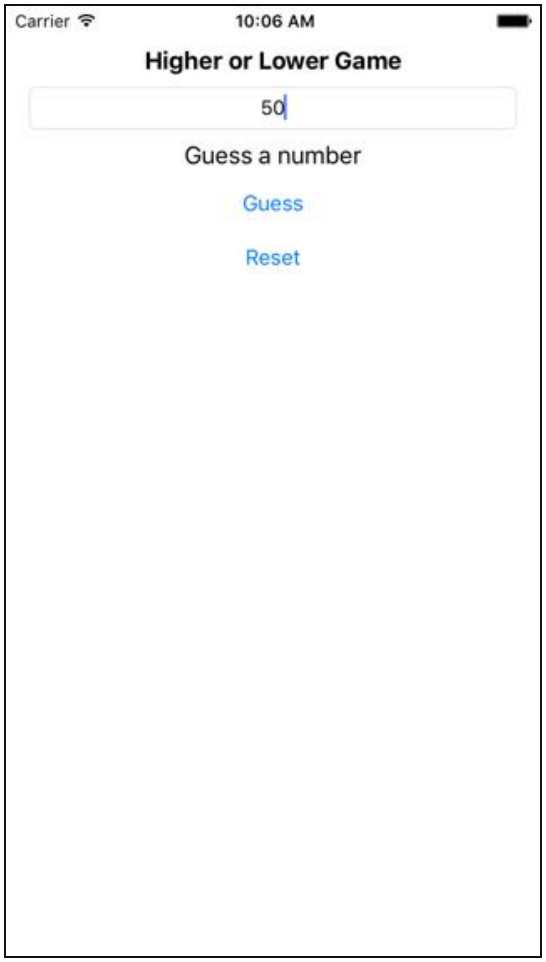
In this lesson you will build a new iPhone app from start to finish. The app will teach you how to design an apps UI (user interface), get user input, and write code to power your app. This lesson will reinforce the Swift Basics concepts with a practical application.

## 8.2 Tutorial - Design the User Interface in Xcode

Your first goal is to get more comfortable with Xcode and the `Storyboard` user interface files. Drag labels, text fields, and buttons to create the prototype app UI. Right now it will be non-functional, but you'll fix that in an upcoming step.

### Design Your User Interface Top to Bottom

Open your `Main.Storyboard` file and from the Object Library on the bottom of the right panel in Xcode drag UI elements onto the `ViewController` canvas.




1. Create a title `UILabel`: `Higher or Lower Game`
    1. Adjust the font to bold and point size
    2. Resize the label using `Xcode > Editor > Size to Fit Content`
        1. Keyboard shortcut: `Command + =`
2. Add an input `UITextField` for the user to type numbers
    1. Adjust the font centering
3. Add a message `UILabel` with a starting message: `Guess a number`
4. Add a `UIButton` named `Guess`
5. Add a `UIButton` named `Reset`

Run your app by pressing the `Play` button in the top left and you should see the start of your iPhone app.

## UITextField Keyboard Types

When you first tap on the `textField` to enter a value, it shows the alphabet keyboard instead of numbers. Change this using the Attributes Inspector on the right panel in Xcode.

1. In Storyboard click on the `UITextField`
2. Change Keyboard Type from `Default` to `Numbers and Punctuation` or `Number Pad`

# 8.3 Tutorial - Connect the UI to Code With Outlets and Actions

To make your apps responsive and interactive to user input, you need to create a connection between the UI and the code.

## How to Make Outlet and Action Connections

1. Open the Assistant Editor using the Venn Diagram button in the top right corner (`Command` + `Alt` + `Enter`)
2. Right-click and drag from a UI element (i.e.: UIButton) to the code file (two-finger click and drag on a trackpad)
    1. Sometimes Xcode displays Manual or nothing (a bug!) in the top jump bar above the code on the right side.
    2. If that happens, click on where it says `Manual` and change it to `Automatic`
3. Type a name for the variable and choose Outlet or Action for the type of connection (see below)

## Outlet and Action Connections

1. Make `Outlet` connections for any UI element you want to change or access in code (i.e.: message label and the input text field)
2. Make `Action` connections for any UI element you want to make something happen (i.e.: Guess and Reset buttons)

## Create Outlets and Actions Using the Following Names

1. Outlet for the UITextField named: textField
2. Outlet for the message UILabel named: messageLabel
3. Action for the Guess UIButton named: guessButtonPressed
4. Action for the Reset UIButton named: resetButtonPressed

### Spelling and Capitalization

Spelling and capitalization are important. You will use camel case for the name, which is common in programming languages for variable and method names. The first letter is lower-case, and every word after is capitalized to make it easy to read.

# 8.4 Bug Fix - iPhone App Crashes on Start

If your app crashes on start it is most likely because you broke one of your connections that you made between your Storyboard and code file.

This crash can happen if you rename an Outlet or Action connection in your code file, but don't change it in the Storyboard file.

1. In the bottom Console panel, scroll to the top of the error message looking for where it says `class is not key value coding-compliant for the key: YOUR_VARIABLE_NAME`
2. Track down the UI element that you gave the name (key) in your Storyboard file
3. Right-click on the UI element (i.e.: UILabel or UITextField)
4. Click the `x` button next to the old name
5. Reconnect the UI element to your code file
6. Run your app again
7. Repeat these steps if you have any other crashes

### Links

- 3 Common Beginner Crashes in Xcode - supereasyapps.com

# 8.5 Tutorial - Auto Layout for Beginners

When you run your iPhone app you will notice that the UI doesn't look the same as you might expect from the Storyboard file. Your center aligned UI elements are not centered in the iPhone app.

When you first design your user interface Xcode will pin your UI elements in place, they don't resize or move based on the size of the iPhone or iPad.

You need to explain to Xcode how to position and size your UI design using Auto Layout. Auto Layout is a rule based system where you specify positions and sizes of every element in your iPhone app's

user interface.

### Add Auto Layout Constraints in Storyboard

1. Open your `Main.storyboard` file
2. Right-click and drag from your UI elements to the canvas view or other UI elements to create new layout rules
3. You need to establish size and position for every UI element

### How to Think About UI Layout

When you think about app layouts, instead of thinking about exact pixels, you should think in terms of relative sizes and positions. On an iPhone 6S+ your UI will look slightly different than on an iPhone 6S because they have different screen sizes. Auto Layout helps you establish rules that the iPhone can follow so that your UI design will look good on any iPhone size.

Position your labels, buttons, images, and text fields relative to each other or anchored to the center and sides of your iPhone screen.

## 8.6 Tutorial - Random Numbers and User Input

Your first task is to get user input from the user. In this app you will work with Integer numbers (i.e.: whole numbers) and you will have the iPhone pick a number to guess.

### Pick a Random Number Between 1 and 100

1. Create a property to store a random `Int` called `numberToGuess`

```
var numberToGuess: Int = 50 // starting value
```

2. Create a method to pick a random number called `pickRandomNumber()`

```
func pickRandomNumber() {
}
```

3. Call the `pickRandomNumber()` at the bottom of your `viewDidLoad()` method

```
override func viewDidLoad() {
```

```
    super.viewDidLoad()

    // Pick a number to guess
    pickRandomNumber()
}
```

4. Use the `arc4random_uniform()` function to pick a number in the range [1, 100]

```
func pickRandomNumber() {
    numberToGuess = Int(arc4random_uniform(100) + 1)
    print("number: \(numberToGuess)") // Only for testing in Xcode, don't cheat!
}
```

## User Input From a UITextField

1. In the `guessButtonPressed(_:)` method you will need to get the text that the user typed

```
@IBAction func guessButtonPressed(sender: AnyObject) {
    if let guess = Int(numberTextField.text!) {
        messageLabel.text = "You guessed: \(guess)"
    }
}
```

2. Convert the text `String` into an `Int` number in order to compare it
3. Display the result using your `messageLabel`

## Links

- How to Create a Random Int Number in Swift 2 - supereasyapps.com
- arc4_uniform() manual page - apple.com

# 8.7 Tutorial - App Logic Using Conditional Statements

You now have the plumbing between the user interface and the code setup. You converted the input from the user from a text `String` to a number that allows you to do less than and greater than equality comparisons. Now you need to write the logic that will decide what to do based on the guessed number versus the current game number.

## App Logic - Pseudocode

There are three cases: when the numbers are equal, less than, or greater than each other. For each case you will need to inform the user if they were correct, higher, or lower.

1. If the `guess` is equal to the `numberToGuess` the user won!
2. If the guess is less than the numberToGuess tell the user their number is too low
3. If the guess is higher than the numberToGuess tell the user their number is too high

```
@IBAction func guessButtonPressed(sender: AnyObject) {
    if let guess = Int(numberTextField.text!) {
        if guess == numberToGuess {
            messageLabel.text = "You won by guessing the number, \(numberToGuess)!"
        } else if guess < numberToGuess{
            messageLabel.text = "\(guess) is lower than my number!"
        } else if guess > numberToGuess {
            messageLabel.text = "\(guess) is higher than my number!"
        }
    }
}
```

## String Interpolation

To provide feedback that makes the app feel responsive you should include the guess from the user in your messages. If you don't, it's unclear to a user if the app has done anything, which means your app could feel broken to a customer.

String interpolation is how you can insert the value of a number or variable inside a text `String`. Inside the `String` literal, surround a variable name with () to insert the current value.

```
var guess = 10
print("Your guess is \(guess)")
```

Will display:

```
"Your guess is 10"
```

## Custom Messages

Change your messages to the user to make them more playful. Be creative with how you communicate to the user.

# 8.8 Code Exercise - Reset the Game and Debugging App Logic

Awesome! Now you can play a full game and test out if your app works correctly. However, after you finish playing a game there is no way to restart. You need to write some code that will reset the game to the beginning state.

## Code Exercise

1. Create a new method called `resetNumber()`
2. Call the `resetNumber()` method in your `resetButtonPressed(_:)` method
3. Pick a new random `numberToGuess`
4. Reset the textField to “50” or some other value
5. Reset the message label to the starting message

## Debugging App Logic

Using Xcode you can insert breakpoints to pause the app and inspect what happens line by line. This is a great way to test new logic and understand what is happening when your app is running.

`if/else` statements only match one statement (conditional expression) and then they skip all the remaining statements. To accomplish the bonus you need to write your new `else if` statements in order or you need to nest them to create a nested `if/else` statement.

## Bonus

1. Display a custom message when the user guesses two numbers higher than the `numberToGuess`
2. Display a message when the user guesses two numbers lower than the `numberToGuess`
3. Hint 1: You will need to add new `else if` statements after you check if your guess is equal to (end game condition)
4. Hint 2: You can nest `if` statements inside of each other
5. Hint 3: You can use the absolute value function `abs()`

Try the Code Exercise and Bonus before you look at the solutions!

# 8.9 Solution - Reset the Game

If you got stuck, follow along and see what you missed.

## Code Exercise Solution

1. Create a new method called `resetNumber()`

```
func resetNumber() {
}
```

2. Call the `resetNumber()` method in your `resetButtonPressed(_:)` method

```
@IBAction func resetButtonPressed(sender: AnyObject) {
    resetNumber()
}
```

3. Pick a new random `numberToGuess`
4. Reset the `messageLabel.text` to the starting message
5. Reset the `textField.text` to `"50"` or some other value

```
func resetNumber() {
    pickRandomNumber()
    messageLabel.text = "Guess a number"
    numberTextField.text = "50"
}
```

## Bonus

1. Add an inner `if/else` statement when you check greater than or less than the `numberToGuess`.
2. Using the absolute value function `abs()` can help you write easier to write app logic

```
@IBAction func guessButtonPressed(sender: AnyObject) {
    if let guess = Int(numberTextField.text!) {
        if guess == numberToGuess {
            messageLabel.text = "You won by guessing the number, \(numberToGuess)!"
        } else if guess < numberToGuess{

            if abs(guess - numberToGuess) < 5 {
                messageLabel.text = "\(guess) is slightly lower than my number!"
            } else {
                messageLabel.text = "\(guess) is lower than my number!"
            }

        } else if guess > numberToGuess {
```

```
            if abs(guess - numberToGuess) < 5 {
                messageLabel.text = "\(guess) is slightly higher than my number!"
            } else {
                messageLabel.text = "\(guess) is higher than my number!"
            }
        }
    }
}
```

# 8.10 Quiz - Chapter 1

Take the Chapter 1 Quiz:

8.10 Quiz - Chapter 1 Topics