

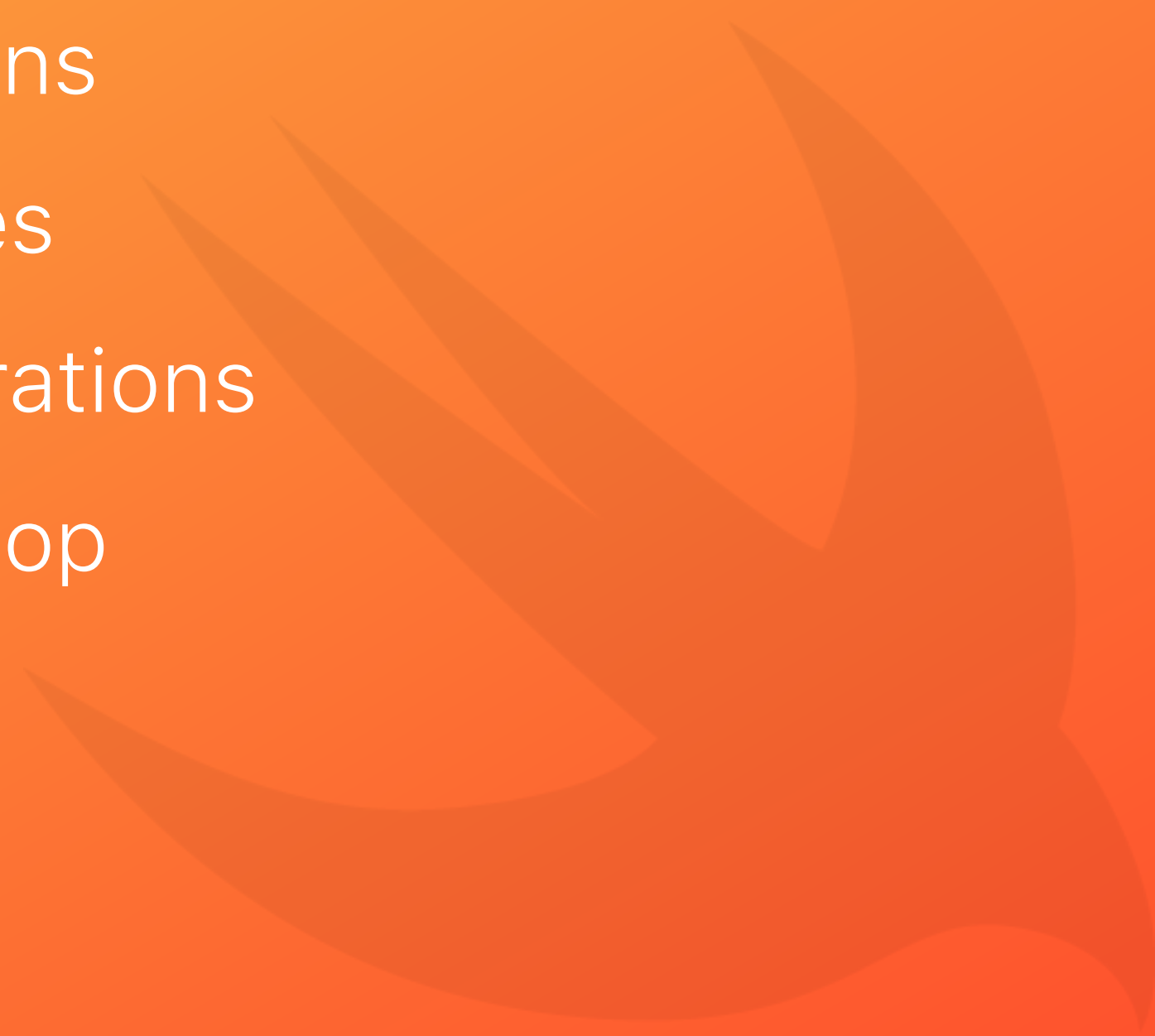
iOS Development with Swift

Functions & Closures & Enumerations

Week 3



Outline

- Functions
 - Closures
 - Enumerations
 - Workshop
- 

Functions

Functions are self-contained chunks of code that perform a specific task.

You give a function a name that identifies what it does, and this name is used to “call” the function to perform its task when needed.

```
func greet(person: String) -> String {  
    let greeting = "Hello, " + person + "!"  
    return greeting  
}
```

Defining Functions

You can define a function with ***name***, values that the function takes as input, known as ***parameters***. You can also optionally define a type of value that the function will pass back as output when it is done, known as its ***return*** type

```
func greet(person: String) -> String {  
    let greeting = "Hello, " + person + "!"  
    return greeting  
}
```

Function Parameters

You can either define a function with parameter(s) or without parameters.

Functions can have multiple input parameters, which are written within the function's parentheses, separated by commas.

```
func sayHelloWorld() -> String {  
    return "hello, world"  
}  
print(sayHelloWorld())  
// Prints "hello, world"
```

```
func greetAgain(person: String) -> String {  
    return "Hello again, " + person + "!"  
}  
print(greetAgain(person: "Anna"))  
// Prints "Hello again, Anna!"
```

Return Types

Functions can return values when unit of work is done.

Functions are not required to return any value. This is optional.

```
func greet(name:String) -> String{  
    return "Hello \ (name)"  
}  
  
print(greet(name: "Gokhan"))
```

Recursive Functions

A function that calls itself is called a recursive function.

Recursive functions are used in repetitive unit of works. e.g. fibonacci numbers, factorial calculation.

```
func fibonacci(val : Int) -> Int{  
    if val == 0{  
        return 0  
    }else if val == 1{  
        return 1  
    }else{  
        return fibonacci(val: val-1) + fibonacci(val: val-2)  
    }  
}
```


Closures

Closures are self-contained blocks of functionality that can be passed around and used in your code.

```
{ ( parameters ) -> return type in  
  statements  
}
```


Closures Cont'd

Following is a simple example of closure.

```
let divide = {(x: Int, y: Int) -> Int in  
    return x / y  
}  
  
let result = divide(200, 20)  
print(result)
```

In general, closures are block of code and they don't have a name and can capture values from their context.

Enumerations

An ***enumeration*** defines a common type for a group of related values and enables you to work with those values in a type-safe way within your code.

Alternatively, enumeration cases can specify associated values of any type to be stored along with each different case value, much as unions or variants do in other languages.

```
enum CompassPoint {  
    case north  
    case south  
    case east  
    case west  
}
```

Enumerations Cont'd

Enumeration value can be a string, a character, or a value of any integer or floating-point type.

```
enum Math : Double{  
    case pi = 3.14159  
    case e = 2.71828  
    case log10 = 1.0  
}  
  
print("π : \ (Math.pi.rawValue)")
```

Workshop - 1

Questions:

- Write a function named **greeting** takes one string parameter. Print that string with "**Hello**" prefix word.
- Write a function named **sum** takes two integer parameters and returns summation of two.
- Write a function named **multiply** that takes two double parameters and returns their multiplications.
- Write a recursive function named **factorial** that takes an integer **N** and returns it's factorial.

Workshop - 2

Question:

Write a recursive function **pow** that takes two numbers **x** and **y** as input and returns **x** to the power **y**.

Hint:

```
func pow(x: Int, y: Int) -> Int {  
  }  
  ▲
```

Workshop - 3

Question: A is an array containing integer values

$A = [3, 8, 1, 7, 3, 4, 6, 10, 12, 5]$

- Sort array by decreasing order.
- Sort array by increasing order
- Write a function named **average** that takes array as a parameter and returns average value of the array.