Introduction to iOS Development

Session 1 - Getting Started

Jay Lees

Swift

UI Components

Xcode

Swift

UI Components

Xcode

Variables, Constants and Data Types

Variables, Constants and Data Types

```
let name = "Jay"
```

Variables, Constants and Data Types

```
let name = "Jay"
var age = 19
```

Variables, Constants and Data Types

```
let name = "Jay"
var age = 19

age = age + 1
```

Variables, Constants and Data Types

```
let name = "Jay"
var age = 19

age += 1
```

Variables, Constants and Data Types

```
let name = "Jay"
var age = 19

age += 1


let surname: String = "Lees"
```

Variables, Constants and Data Types

```
let name = "Jay"
var age = 19

age += 1

let surname: String = "Lees"
let value: Double = 6.7
```

Variables, Constants and Data Types

```
let name = "Jay"
var age = 19

age += 1

let surname: String = "Lees"
let value: Double = 6.7
let anotherValue: Float = 8.9
```

Variables, Constants and Data Types

```
let name = "Jay"
var age = 19

age += 1

let surname = "Lees"
let value = 6.7
let anotherValue: Float = 8.9
```

Variables, Constants and Data Types

```
let name = "Jay"
var age = 19

age += 1

let surname = "Lees"
let value = 6.7
let anotherValue: Float = 8.9

print("Hello, \(name)")
```

Arrays

Arrays

```
var myArray = [1,2,3,4,5]
```

Arrays

```
var myArray = [1,2,3,4,5]
let myOtherArray = [1, 2, true, false]
```

Arrays

```
var myArray = [1,2,3,4,5]
//let myOtherArray = [1, 2, true, false]
```

Arrays

```
var myArray = [1,2,3,4,5]
//let myOtherArray = [1, 2, true, false]

let anotherArray: [(Int, Double)] = [(1, 1.1), (2, 2.2)]
```

Arrays

```swift
var myArray = [1,2,3,4,5]
//let myOtherArray = [1, 2, true, false]

let anotherArray: [(Int, Double)] = [(1, 1.1), (2, 2.2)]
let initArray = [Int]()
```

Arrays

```
var myArray = [1,2,3,4,5]
//let myOtherArray = [1, 2, true, false]

let anotherArray: [(Int, Double)] = [(1, 1.1), (2, 2.2)]
let initArray = [Int]()
myArray.count //5
```

Arrays

```swift
var myArray = [1,2,3,4,5]
//let myOtherArray = [1, 2, true, false]

let anotherArray: [(Int, Double)] = [(1, 1.1), (2, 2.2)]
let initArray = [Int]()
myArray.count
myArray.append(4) //[1,2,3,4,5,4]
```

Higher Order Functions

Higher Order Functions

```
myArray.map { (x) -> Int in
    return x * 2
}
//[1,2,3,4,5] => [2,4,6,8,10]
```

Higher Order Functions

```
myArray.map { (x) -> Int in
    return x * 2
}
myArray.filter { (x) -> Bool in
    return x % 2 == 0
}
//[1,2,3,4,5] => [2,4]
```

Higher Order Functions

```
myArray.map { (x) -> Int in
    return x * 2
}
myArray.filter { (x) -> Bool in
    return x % 2 == 0
}
myArray.reduce(0) { (result, x) -> Int in
    return result + x
}
//[1,2,3,4,5] => 15
```

Higher Order Functions

```
myArray.map({$0*2})
myArray.filter({$0 % 2 == 0})
myArray.reduce(0, +)
```

Loops

Loops

```
let subjects = ["Biology", "Chemistry", "Maths", "English", "CompSci"]
```

Loops

```
let subjects = ["Biology", "Chemistry", "Maths", "English", "CompSci"]
for subject in subjects {

}
```

Loops

```
var i = 10
while i > 0 {

}
```

Optionals

Optionals

```
var maybeInt: Int?
```

Optionals

```
var maybeInt: Int?

if let val = maybeInt {
    print("It has a value of \(val)")
} else {
    print("It does not have a value")
}
```

Optionals

```
var maybeInt: Int?

if let val = maybeInt {
    print("It has a value of \(val)")
} else {
    print("It does not have a value")
}

var maybeFloat: Float?
```

Optionals

```
var maybeInt: Int?

if let val = maybeInt {
    print("It has a value of \(val)")
} else {
    print("It does not have a value")
}

var maybeFloat: Float?
maybeFloat = 10.1
```

Optionals

```
var maybeInt: Int?

if let val = maybeInt {
    print("It has a value of \(val)")
} else {
    print("It does not have a value")
}

var maybeFloat: Float?
maybeFloat = 10.1

let solution = maybeFloat + 10.0
```

Optionals

```
var maybeInt: Int?

if let val = maybeInt {
    print("It has a value of \(val)")
} else {
    print("It does not have a value")
}

var maybeFloat: Float?
maybeFloat = 10.1

let solution = maybeFloat + 10.0
```

Error!

Optionals

```
var maybeInt: Int?

if let val = maybeInt {
    print("It has a value of \(val)")
} else {
    print("It does not have a value")
}

var maybeFloat: Float?
maybeFloat = 10.1

let solution = maybeFloat! + 10.0
```

Optionals

```
var maybeInt: Int?

if let val = maybeInt {
    print("It has a value of \(val)")
} else {
    print("It does not have a value")
}

var maybeFloat: Float?
maybeFloat = 10.1

let solution = maybeFloat? + 10.0
```

Optionals

```
var maybeInt: Int?

if let val = maybeInt {
    print("It has a value of \(val)")
} else {
    print("It does not have a value")
}

var maybeFloat: Float?
maybeFloat = 10.1

let solution = maybeFloat? + 10.0
```

Error!

Functions

Functions

```
func
```

Functions

```
func double(              )
```

Functions

```
func double(input: Int)
```

Functions

```
func double(input: Int) -> Int
```

Functions

```
func double(input: Int) -> Int{
    return input * 2
}
```

Functions

```
func double(input: Int) -> Int{
    return input * 2
}

func add(a: Int, b: Int) -> Int {
    return a + b
}
```

Functions

```
func double(input: Int) -> Int{
    return input * 2
}

func add(a: Int, b: Int) -> Int {
    return a + b
}

func doNothing(){

}
```

Functions

Functions

```
func square(input: Int) -> Int {
    return input * input
}
```

Functions

```
func square(input: Int) -> Int {
    return input * input
}

square(input: 10)
```

Functions

```
func square(input: Int) -> Int {
    return input * input
}

square(input: 10)

func square(_ input: Int) -> Int {
    return input * input
}
```

Functions

```
func square(input: Int) -> Int {
    return input * input
}

square(input: 10)

func square(_ input: Int) -> Int {
    return input * input
}

square(10)
```

Functions

```
func square(input: Int) -> Int {
    return input * input
}

square(input: 10)

func square(_ input: Int) -> Int {
    return input * input
}
square(10)

func square(thisValue input: Int) -> Int {
    return input * input
}
```

Functions

```
func square(input: Int) -> Int {
    return input * input
}

square(input: 10)

func square(_ input: Int) -> Int {
    return input * input
}

square(10)

func square(thisValue input: Int) -> Int {
    return input * input
}

square(thisValue: 10)
```

Classes

Classes

```
public class Animal {

}
```

Classes

```
public class Animal {
    var name: String




}
```

Classes

```
public class Animal {
    var name: String
    private var age: Int




}
```

Classes

```
public class Animal {
    var name: String
    private var age: Int

    init(                          ){



    }
}
```

Classes

```
public class Animal {
    var name: String
    private var age: Int

    init(name: String, age: Int){


    }
}
```

Classes

```
public class Animal {
    var name: String
    private var age: Int

    init(name: String, age: Int){
        self.name = name
        self.age = age
    }

}
```

Classes

```
public class Animal {
    var name: String
    private var age: Int

    init(name: String, age: Int){
        self.name = name
        self.age = age
    }

    func hadBirthday(){
        age += 1
    }



}
```

Classes

```
public class Animal {
    var name: String
    private var age: Int

    init(name: String, age: Int){
        self.name = name
        self.age = age
    }

    func hadBirthday(){
        age += 1
    }

    func getAge() -> Int{
        return age
    }
}
```

Extending Classes

Extending Classes

```
public class Dog:          {



}
```

Extending Classes

```
public class Dog: Animal {




}
```

Extending Classes

```
public class Dog: Animal {

    var color: String



}
```

Extending Classes

```
public class Dog: Animal {

    var color: String
    init(name: String, age: Int, color: String) {



    }

}
```

Extending Classes

```
public class Dog: Animal {

    var color: String
    init(name: String, age: Int, color: String) {
        self.color = color
        super.init(name: name, age: age)
    }

}
```

Extending Classes

```
public class Dog: Animal {

    var color: String
    init(name: String, age: Int, color: String) {
        self.color = color
        super.init(name: name, age: age)
    }

    func woof() -> String {
        return "WOOF!"
    }



}
```

Extending Classes

```
public class Dog: Animal {

    var color: String
    init(name: String, age: Int, color: String) {
        self.color = color
        super.init(name: name, age: age)
    }

    func woof() -> String {
        return "WOOF!"
    }

    override func getAge() -> Int {
        return age * 7
    }
}
```

Enums

Enums

```
enum FurColor {



}
```

Enums

```
enum FurColor {
    case

}
```

Enums

```
enum FurColor {
    case Black


}
```

Enums

```
enum FurColor {
    case Black
    case Brown
    case White
    case Golden
    case Other
}
```

Implementing Enums

```
public class Dog: Animal {

    var color: String
    init(name: String, age: Int, color: String) {
        self.color = color
        super.init(name: name, age: age)
    }

    func woof() -> String {
        return "WOOF!"
    }

    override func getAge() -> Int {
        return age * 7
    }
}
```

Implementing Enums

```
public class Dog: Animal {

    var color: FurColor
    init(name: String, age: Int, color: FurColor) {
        self.color = color
        super.init(name: name, age: age)
    }

    func woof() -> String {
        return "WOOF!"
    }

    override func getAge() -> Int {
        return age * 7
    }
}
```

Implementing Enums

```
public class Dog: Animal {

    var color: FurColor
    init(name: String, age: Int, color: FurColor) {
        self.color = color
        super.init(name: name, age: age)
    }

    func woof() -> String {
        return "WOOF!"
    }

    override func getAge() -> Int {
        return age * 7
    }
}

let dog = Dog(name: "Scooby", age: 12, color: “Black”)
```

Implementing Enums

```
public class Dog: Animal {

    var color: FurColor
    init(name: String, age: Int, color: FurColor) {
        self.color = color
        super.init(name: name, age: age)
    }

    func woof() -> String {
        return "WOOF!"
    }

    override func getAge() -> Int {
        return age * 7
    }
}

let dog = Dog(name: "Scooby", age: 12, color: .Black)
```

Swift

UI Components

Xcode

UIKit