Recap

Session 101

Alex Telek
Clarence Ji
KCL Tech iOS Engineers

Variables

Variables

```
var languageName: String = “Swift”
```

Array and Dictionary

```
var names = [“Anna”, “Brian”, “Jack”]
```

```
var numberOfLegs = [“ant”: 6, “snake”: 0]
```

Loops

Loops

```
while hungry {
  eatCake()
}
```

Loops

```
while hungry {
  eatCake()
}
```

```
for var i = 0; i < 10; i++ {
  eat(i)
}
```

Loops

```
while hungry {
  eatCake()
}
```

```
var index = 0
repeat {
  index++
} while index < 5
```

```
for var i = 0; i < 10; i++ {
  eat(i)
}
```

If Statements

If Statements

```
if legCount == 0 {

  print(“It slides”)

} else {

  print(“It walks”)
}
```

Switch

Switch

```
switch legCount {
  case 0:
    print(“It slides”)

  case 1,3,5,7,9:
    print(“It hops”)

  case 2,4,6,8,10:
    print(“It walks”)

  default:
    print(“No idea”)
}
```

Functions

Functions

```
func sayHello(name: String) -> String {
  return “Hello ” + name
}
```

Functions

```
func sayHello(name: String) -> String {
  return “Hello ” + name
}

let greeting = sayHello(“WWDC”)
```

Functions

```
func sayHello(name: String) -> String {
  return “Hello ” + name
}

let greeting = sayHello(“WWDC”)

println(greeting)
```

Functions

```
func sayHello(name: String) -> String {
  return “Hello ” + name
}

let greeting = sayHello(“WWDC”)

println(greeting)
```

```
Hello, WWDC
```

Challenge 1: Prime Number

itunesu.kcl.tech

Challenge 1: Prime Number

Solve the coding assignment (More Swift and Xcode - Challenge 1) on iTunes U

itunesu.kcl.tech

More Swift & Xcode Auto Layout

Session 102

Alex Telek
Clarence Ji
KCL Tech iOS Engineers

Optionals

Optionals

You use optionals in situations where a value may be absent.

Optionals

You use optionals in situations where a value may be absent.

- There is a value, and it is equal to x.

Optionals

You use optionals in situations where a value may be absent.

- There is a value, and it is equal to x.
- There is no value at all.

Optionals

Optionals

```
let numberOfLegs = [“ant” : 6 , “snake” : 0]
```

Optionals

```
let numberOfLegs = [“ant” : 6 , “snake” : 0]

let possibleLeg: Int? = numberOfLegs[“frog”]
```

Optionals

```
let numberOfLegs = [“ant” : 6 , “snake” : 0]

let possibleLeg: Int? = numberOfLegs[“frog”]


if possibleLeg == nil {

  print(“Frog wasn’t found”)

} else {

  let legCount = possibleLeg!
  print(“A frog has \(possibleLeg) legs”)

}
```

Unwrapping optionals

SAFE

Unwrapping optionals

SAFE

```
if let legCount = possibleLeg {

   print(“A frog has \(legCount) legs”)

}
```

Unwrapping optionals


```
if let legCount = possibleLeg {
  print(“A frog has \(legCount) legs”)
}

if let legCount = possibleLeg, value = anotherOptional {
  print(“What happens now?”)
}
```

Tuples

MODERN

Tuples

MODERN

```
func refreshWebPage() -> (Int, String) {

  //…try to refresh…

  return (200, “Success”)

}
```

Tuples

Tuples

```
(3.79, 2.23, 5.65)      //(Double, Double, Double)
```

Tuples

```
(3.79, 2.23, 5.65)        //(Double, Double, Double)

(404, “Not found”)        //(Int, String)
```

Tuples

```
(3.79, 2.23, 5.65)        //(Double, Double, Double)

(404, “Not found”)        //(Int, String)

(2, “banana”, 3.21)       //(Int, String, Double)
```

Tuples

Tuples

```
func refreshWebPage() -> (Int, String) {

  return (200, “Success”)

}
```

Tuples

```
func refreshWebPage() -> (Int, String) {

  return (200, “Success”)

}

let (statusCode, message) = refreshWebPage()
```

Tuples

```
func refreshWebPage() -> (Int, String) {

  return (200, “Success”)

}

let (statusCode, message) = refreshWebPage()

print(“Received \(statusCode): \(message)”)
```

Tuples

```
func refreshWebPage() -> (Int, String) {
  return (200, “Success”)
}

let (statusCode, message) = refreshWebPage()
print(“Received \(statusCode): \(message)”)
```

```
Received 200: Success
```

Tuples

```
func refreshWebPage() -> (Int, String) {

  return (200, “Success”)

}
```

```
Received 200: Success
```

Tuples

```
func refreshWebPage() -> (Int, String) {

  return (200, “Success”)

}

let status = refreshWebPage()

print(“Received \(status.code): \(status.message)”)
```

```
Received 200: Success
```

Closures

Closures

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

Closures

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

Similar to blocks in C and Objective-C

Closures

Closures

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

   statements
}
```

Closures

Closures

```
func methodWithClosure(closure: (String) -> ()) {
    closure("In a closure")
}
```

Closures

```
func methodWithClosure(closure: (String) -> ()) {
    closure("In a closure")
}

methodWithClosure { (returnString) -> () in
    print(returnString)
}
```

Demo

Sort closure

Enums

Enums

An enumerations 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

Enums

An enumerations 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

```
enum CompassPoint {
    case North
    case South
    case East
    case West
}
```

Enums

An enumerations 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

```
enum CompassPoint {
   case North
   case South
   case East
   case West
}

let direction = .North
```

Recursive Enums

Recursive Enums

Recursive enumeration is an enumeration that has another instance of the enumeration as the associated value for one or more of the enumeration members.

Recursive Enums

Recursive enumeration is an enumeration that has another instance of the enumeration as the associated value for one or more of the enumeration members.

```
enum ArithmeticExpression {
    case Number(Int)
    indirect case Addition(ArithmeticExpression, ArithmeticExpression)
    indirect case Multiplication(ArithmeticExpression,
    ArithmeticExpression)
    …
}
```

Recursive Enums

Recursive enumeration is an enumeration that has another instance of the enumeration as the associated value for one or more of the enumeration members.

```
indirect enum ArithmeticExpression {
   case Number(Int)
   case Addition(ArithmeticExpression, ArithmeticExpression)
   case Multiplication(ArithmeticExpression, ArithmeticExpression)
   …
}
```

Error Handling

Error Handling

Error handling is the process of responding to and recover from error conditions in your program

Error Handling

Error handling is the process of responding to and recover from error conditions in your program

```
enum MyError: ErrorType {
    case NotFound
    case Removed
    case Failed
    case NoInternetConnection
}
```

Error Handling

Error Handling

```
func loadData() throws {
    // Do something…

    throw MyError.Failed
}
```

Error Handling

```
func loadData() throws {
   // Do something…

   throw MyError.Failed
}

func refresh(){
   do {
      try loadData()
   } catch {
      print(error)
   }
}
```

Guard

MODERN

Guard

MODERN

```
func refresh() throws {

   guard isInternetAvailable() else {
      throw MyError.NoInternetConnection
   }

   do {
      try loadData()
   } catch {
      print(error)
   }
}
```

Defer

MODERN

Defer

MODERN

```
func refresh() throws {
    defer {
        print(“Clean up”)
    }

    guard isInternetAvailable() else {
        throw MyError. NoInternetConnection
    }

    do {
        try loadData()
    } catch {
        print(error)
    }
}
```

Extensions

MODERN

Extensions

MODERN

Extensions can add computed instance properties and computed type properties to existing types

Extensions

MODERN

Extensions can add computed instance properties and computed type properties to existing types

```
extension Double {
    var m: Double { return self }
    var km: Double { return self * 1000.0 }
}
```

Extensions

MODERN

Extensions can add computed instance properties and computed type properties to existing types

```
extension Double {
    var m: Double { return self }
    var km: Double { return self * 1000.0 }
}

3.0.km
```

Extensions

Extensions can add computed instance properties and computed type properties to existing types

```
extension Double {
    var m: Double { return self }
    var km: Double { return self * 1000.0 }
}

3.0.km
```

```
3000
```

Extensions

MODERN

Extensions

MODERN

```
extension String {
    func toArray() -> Array<Character> {
      return Array(self.characters)
    }
}
```

Extensions

MODERN

```
extension String {
    func toArray() -> Array<Character> {
      return Array(self.characters)
    }
}

“Hello there”.toArray
```

Extensions

MODERN

```
extension String {
    func toArray() -> Array<Character> {
      return Array(self.characters)
    }
}

“Hello there”.toArray
```

```
[“h”, “e”, “l”, “l”, “o”, “”, “t”, “h”, “e”, “r”, “e”]
```

Demo

Xcode Auto Layout

KCL TECH SOCIETY