CSE 3218 - Lab1
Introduction to Swift

Kaniz Fatema Isha
Lecturer,
CSE, KUET

Introduction

- Swift is a general-purpose programming language built using a modern approach to safety, performance, and software design patterns.
- Apple created Swift, an open-source programming language, as a replacement for all languages based on C, including Objective C, C++, and C.
- Swift was designed from the outset to be safer than C-based languages, and eliminates entire classes of unsafe code.

Introduction

Swift defines away large classes of common programming errors by adopting modern programming patterns:

- Variables are always initialized before use.
- Array indices are checked for out-of-bounds errors.
- Integers are checked for overflow.
- Optionals ensure that nil values are handled explicitly.
- Memory is managed automatically.
- Type safe language

Declaring Constants and Variables

- Constants and variables must be declared before they’re used.
- The syntax to declare a variable is : var variable_name: Datatype

```
var siteName:String
var id: Int
```

Declaring Constants and Variables

- There are 2 ways to assign values into variables:

```
var siteName: String
siteName = "programiz.com"

print(siteName)
```

```
var siteName = "programiz.com"
print(siteName) // programiz.com
```

Declaring Constants and Variables

- The value of a constant cannot be changed. For example

```
let const:String ="Hello World !"
const="Hello KUET!!"
print(const)
```

ERROR!!

- If we are sure that the value of a variable won't change throughout the program, it's recommended to use let.

Swift Data Types

- There are six basic types of data types in Swift programming.

Data Types	Example	Description
`Character`	`"s"`, `"a"`	a 16-bit Unicode character
`String`	`"hello world!"`	represents textual data
`Int`	`3`, `-23`	an integer number
`Float`	`2.4`, `3.14`, `-23.21`	represents 32-bit floating-point number
`Double`	`2.422342412414`	represents 64-bit floating-point number
`Bool`	`true` and `false`	Any of two values: true or false

Swift Input Output

- In Swift, we can simply use the print() function to print output. For example,

```
print("Hello World !")
```

```
Hello World !
```

- We can print a string and variable together by using string interpolation

Swift Input Output

- We can print a string and variable together by using string interpolation.

```
var year = 2014
print("Swift was introduced in \(year)")
```

```
Swift was introduced in 2014
```

Swift Input Output

- We can use the readLine() function to take input from users.
- Swift always assumes that the newline is not a part of the input.
- Swift takes the input as String. To take input any other Datatype, we must do type conversion:

```
var k:Int
k=Int(readLine()!)!
print(k)
```

Swift Comments

- There are two ways to add comments in Swift:

```
//This is a single line comment
print("Hello World !")

/* This is a muli-line
comment */
print("Hello Kuet")
```

Swift Operators

There are different types of Swift operators :

- Arithmetic operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- Special Operators (Ternary,Nil-Coalescing Operator,Range Operator)

Nil-Coalescing Operator

- The nil-coalescing operator (a ?? b) unwraps an optional a if it contains a value, or returns a default value b if a is nil. The expression a is always of an optional type.
- The expression b must match the type that's stored inside a.

```
let name: String? = nil
let unwrappedName = name ?? "Anonymous"
```

Range Operator

- The closed range operator (a...b) defines a range that runs from a to b, and includes the values a and b. The value of a must not be greater than b.
- The half-open range operator (a..<b) defines a range that runs from a to b, but doesn't include b.
- Both these operators can also be expressed in One-Sided Ranges.

```
// Closed Range Operator
0...2 // 0, 1, 2

// Half Open Range Operator
0..<2 // 0, 1
```

Tuples in Swift

- A tuple is a group of different values. Each value inside a tuple can be of different data types.

```
let complex = (1.0, -2.0)  // Compound Type: (Double, Double)
let (real, imag) = complex // Decompose
let (real, _) = complex    // Underscores ignore value

// Access by index
let real = complex.0
let imag = complex.1

// Name elements
let complex = (real: 1.0, imag: -2.0)
let real = complex.real
```

Optional in Swift

- An optional in Swift is basically a constant or variable that can hold a value OR no value. The value can or cannot be nil. It is denoted by appending a “?” after the type declaration

```
var tweet: String?
```

- We must “unwrap” an optional to use it. There are 3 ways to unwrap an optional

Optional in Swift

1. **Forced Unwrapping:**
    - Forced Unwrapping is denoted by “!” to the optional’s name.
    - The exclamation indicates the guarantee of optional having a value.

```
var tweet: String?
tweet = “Now assigning a string to tweet, thus giving tweet a value”
print(tweet!)
```

What happens when force unwrapping is applied but there was no value in optional?

Optional in Swift

2. Implicitly Unwrapped Optionals:

- Implicitly unwrapped optionals are similar to optionals since they're allowed to have nil value but they do not need to be checked before accessing.
- An implicitly unwrapped optional is declared by placing an exclamation point (String!) rather than a question mark (String?) after the type of the optional.

Optional in Swift

2. Implicitly Unwrapped Optionals:

```
// Used mainly for class initialization
var assumedInt: Int! // set to nil
assumedInt = 42

var implicitInt: Int = assumedInt // do not need an exclamation mark
assumedInt = nil
implicitInt = assumedInt // ❌ RUNTIME ERROR!!!
```

Optional in Swift

3. Optional Binding:

- In this case, we check if a variable has a value or not by writing a codeblock:

```
var tweet: String?
tweet = "Now assigning a string to tweet, thus giving tweet a value"

if let actualTweet = tweet {
 print("The value is: \(actualTweet)")
} else {
 print("Gracefully go to this line when tweet is nil")
}
```

if-else in Swift

In Swift, there are three forms of the if...else statement.

- if statement
- if...else statement
- if...else if...else statement

if-else in Swift

```
let temperature = 40

var feverish: Bool
if temperature > 37 {
    feverish = true
} else {
    feverish = false
}
```

if-else in Swift

- Parenthesis() are optional and by convention are often omitted
- But Braces{} are always required even if the body contains only one statement.

```
if  temperature > 37  { feverish = true } // OK
if (temperature > 37) { feverish = true } // OK
if (temperature > 37)   feverish = true   // ❌ ERROR!!!
```

Switch in Swift

- The syntax of the switch statement in Swift is:

```
switch (expression)  {
  case value1:
    // statements

  case value2:
    // statements

  ...
  ...

  default:
    // statements
}
```

Switch in Swift

- Example:

```
let someCharacter: Character = "z"
switch someCharacter {
case "a":
    print("The first letter of the Latin alphabet")
case "z":
    print("The last letter of the Latin alphabet")
default:
    print("Some other character")
}
// Prints "The last letter of the Latin alphabet"
```

Switch in Swift

- Example:

```
let anotherCharacter: Character = "a"
let message = switch anotherCharacter {
case "a":
    "The first letter of the Latin alphabet"
case "z":
    "The last letter of the Latin alphabet"
default:
    "Some other character"
}

print(message)
// Prints "The first letter of the Latin alphabet"
```

Switch in Swift

- Switch is always exhaustive, there is always a value to assign.

```
let anotherCharacter: Character = "a"
switch anotherCharacter {
case "a": // Invalid, the case has an empty body
case "A":
    print("The letter A")
default:
    print("Not the letter A")
}
// This will report a compile-time error.
```

Switch in Swift

- Generally, switch cases in Swift doesnt require break to choose the statements of the chosen case.
- However, if we use the **fallthrough** keyword inside the case statement, the control proceeds to the next case even if the case value does not match with the switch expression.

Switch in Swift

```swift
let dayOfWeek = 2
switch dayOfWeek {
  case 1:
    print("Sunday")

  case 2:
    print("Monday")

  case 7:
    print("Saturday")
  default:
    print("Invalid day")
}
```

```
Monday
```

Switch in Swift

```
let dayOfWeek = 2
switch dayOfWeek {
  case 1:
    print("Sunday")

  case 2:
    print("Monday")
    fallthrough
  case 7:
    print("Saturday")
  default:
    print("Invalid day")
}
```

```
Monday
Saturday
```

Switch in Swift

- We can generate compound cases.

```
let anotherCharacter: Character = "a"
switch anotherCharacter {
case "a", "A":
    print("The letter A")
default:
    print("Not the letter A")
}
// Prints "The letter A"
```

Switch in Swift

- Values in switch cases can be checked for their inclusion in an interval.

```
let approximateCount = 62
let countedThings = "moons orbiting Saturn"
let naturalCount: String
switch approximateCount {
case 0:
    naturalCount = "no"
case 1..<5:
    naturalCount = "a few"
case 5..<12:
    naturalCount = "several"
case 12..<100:
    naturalCount = "dozens of"
case 100..<1000:
    naturalCount = "hundreds of"
default:
    naturalCount = "many"
}
print("There are \(naturalCount) \(countedThings).")
```

Switch in Swift

- In Swift, we can also use tuples in switch statements.

```
let somePoint = (1, 1)
switch somePoint {
case (0, 0):
    print("\(somePoint) is at the origin")
case (_, 0):
    print("\(somePoint) is on the x-axis")
case (0, _):
    print("\(somePoint) is on the y-axis")
case (-2...2, -2...2):
    print("\(somePoint) is inside the box")
default:
    print("\(somePoint) is outside of the box")
}
// Prints "(1, 1) is inside the box"
```

Switch in Swift

- A switch case can also use a where clause to check for additional conditions.

```
let yetAnotherPoint = (1, -1)
switch yetAnotherPoint {
case let (x, y) where x == y:
    print("(\(x), \(y)) is on the line x == y")
case let (x, y) where x == -y:
    print("(\(x), \(y)) is on the line x == -y")
case let (x, y):
    print("(\(x), \(y)) is just some arbitrary point")
}
// Prints "(1, -1) is on the line x == -y"
```

Loop in Swift

- **For-In Loops** : It is used to iterate over a sequence, such as items in an array, ranges of numbers, characters in a string, numeric ranges, dictionaries.

```
let names = ["Anna", "Alex", "Brian", "Jack"]
for name in names {
    print("Hello, \(name)!")
}
// Hello, Anna!
// Hello, Alex!
// Hello, Brian!
// Hello, Jack!
```

Loop in Swift

- **While and repeat while** : These are used to run a specific code until a certain condition is met.

```
var i = 1, n = 5

// while loop from i = 1 to 5
while (i <= n) {
  print(i)
   i = i + 1
}
```

Loop in Swift

- **While and repeat while** : These are used to run a specific code until a certain condition is met.

```
var i = 1, n = 5

// repeat...while loop from 1 to 5
repeat {

  print(i)

  i = i + 1

} while (i <= n)
```

Function in Swift

- Syntax of function defining in Swift:

```
func functionName(parametername1: datatype,parametername2:
datatype,.....)-> Return Datatype {

// function body statements….

}
```

- Syntax of function calling in Swift:

```
functionName(argumentlabel1: actual_value1,...)
```

Function in Swift

```
func greet(person: String, alreadyGreeted: Bool) -> String {
    if alreadyGreeted {
        return greetAgain(person: person)
    } else {
        return greet(person: person)
    }
}
print(greet(person: "Tim", alreadyGreeted: true))
// Prints "Hello again, Tim!"
```

Function in Swift

- We can define a default value for any parameter in a function by assigning a value to the parameter after that parameter's type. If a default value is defined, we can parameter when calling the function.

```
func someFunction(parameterWithoutDefault: Int, parameterWithDefault: Int = 12) {
    // If you omit the second argument when calling this function, then
    // the value of parameterWithDefault is 12 inside the function body.
}
someFunction(parameterWithoutDefault: 3, parameterWithDefault: 6) // parameterWithDefault
someFunction(parameterWithoutDefault: 4) // parameterWithDefault is 12
```

Function in Swift

- We can use a tuple type as the return type for a function to return multiple values as part of one compound return value.

```
func minMax(array: [Int]) -> (min: Int, max: Int) {
    var currentMin = array[0]
    var currentMax = array[0]
    for value in array[1..<array.count] {
        if value < currentMin {
            currentMin = value
        } else if value > currentMax {
            currentMax = value
        }
    }
    return (currentMin, currentMax)
}
```

Function in Swift

- A variadic parameter accepts zero or more values of a specified type.
- We use a variadic parameter to specify that the parameter can be passed a varying number of input values when the function is called.
- A variadic parameters by is defined by inserting three period characters (...) after the parameter’s type name.
- A function can have multiple variadic parameters

Function in Swift

```swift
func arithmeticMean(_ numbers: Double...) -> Double {
    var total: Double = 0
    for number in numbers {
        total += number
    }
    return total / Double(numbers.count)
}
arithmeticMean(1, 2, 3, 4, 5)
// returns 3.0, which is the arithmetic mean of these five numbers
arithmeticMean(3, 8.25, 18.75)
// returns 10.0, which is the arithmetic mean of these three numbers
```

Closure in Swift

- A closure is a special type of function without the function name
- Syntax:

```
{ (parameters) -> returnType in

   // statements

}
```

- We don't use the func keyword to create closure

Closure in Swift

```
// closure definition
var findSquare = { (num: Int) -> (Int) in
  var square = num * num
  return square
}

// closure call
var result = findSquare(3)

print("Square:",result)
```

Closure in Swift

- A closure can capture constants and variables from the surrounding context in which it's defined. The closure can then refer to and modify the values of those constants and variables from within its body, even if the original scope that defined the constants and variables no longer exists.
- In Swift, the simplest form of a closure that can capture values is a nested function, written within the body of another function.
- For a more clear understanding, see the documentation .

Closure in Swift

```
func makeIncrementer(forIncrement amount: Int) -> () -> Int {
    var runningTotal = 0
    func incrementer() -> Int {
        runningTotal += amount
        return runningTotal
    }
    return incrementer
}
```

Reference

1. Web
2. Documentation
3. Programiz