Control Flow

Outline

1. Loops
2. Conditional Statements
3. Control Transfer Statements
4. Early Exit
5. Checking API Availability

1. Loops

1. *For-in* loops
2. *While* loops
3. *Repeat-while* loops
4. Range Operators

1.1 For-in Loops

- You can use the *for-in* loop to iterate over a sequence (e.g: array, ranges of numbers, characters in a string,...)

```
let str = "Hello world"
var charArray = [Character]()
for character in str {
    charArray.append(character)
}

print(charArray)
// ["H", "e", "l", "l", "o", " ", "w", "o", "r", "l", "d"]
```

1.1 For-in Loops (cont)

- Iterate over a dictionary

```
let numberOfLegs = ["spider": 8, "ant": 6, "cat": 4,
"human": 2]
for (animalName, legCount) in numberOfLegs {
    print("\(animalName)s have \(legCount) legs")
}
// cats have 4 legs
// ants have 6 legs
// humans have 2 legs
// spiders have 8 legs
```

1.1 For-in Loops (cont)

- Iterate over a range of numbers

```
let step = 2
for number in stride(from: 0, through: 10, by: step) {
    print(number)
}
// 0, 2, 4, 6, 8, 10
```

1.2 While Loops

- A *while* loop performs a set of statements until a condition becomes *false*

```
var number = 45856
var sumOfNumbers = 0
while number > 0 {
    sumOfNumbers += number % 10
    number /= 10
}
print(sumOfNumbers) // 28
```

1.3 Repeat - While

- Swift replaces *do-while* syntax in C with *repeat-while*

```
var sum = 0
repeat {
    sum += 2
} while sum < 10
print(sum) // 10
```

1.4 Range Operators

- The *closed range operator* **`(a...b)`** defines a range that run from **`a`** to **`b`**, and includes the values **`a`** and **`b`**.

```
for index in 1 ... 5 {
    print("\(index) times 5 is \(index * 5)")
}
// 1 times 5 is 5
// 2 times 5 is 10
// 3 times 5 is 15
// 4 times 5 is 20
// 5 times 5 is 25
```

1.4 Range Operators

- The *half-open range operator* `(a..<b)` defines a range that runs from `a` to `b`, but doesn't include `b`.

```
let fruits = ["Apple", "Orange", "Pine Apple", "Coconut"]
let count = fruits.count
for index in 0 ..< count {
    print("Fruit number \(index + 1) is \(fruits[index])")
}
```

1.4 Range Operators

- The *one-sided range* is an alternative form for ranges that continue as far as possible in one direction.

```
let fruits = ["Apple", "Orange", "Pine Apple", "Coconut"]
for fruit in fruits[2...] {
    print(fruit)
}
// Pine Apple
// Coconut

for fruit in fruits[..<2] {
    print(fruit)
}
// Apple
// Orange
```

2. Conditional Statements

1. *If* Expression
2. *Switch* Expression

2.1 *If* Expression

- As other programming languages, Swift has *if* expression. Traditional

```
var max = a
if a < b {
    max = b
}

// With else
if a > b {
    max = a
} else {
    max = b
}
```

2.2 *Switch* Expression

- A *Switch* statement provides an alternative to the *if* statement for responding to multiple potential states

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

2.2 *Switch* Expression (cont)

- In contrast with *switch* statements in C and Objective-C, *switch* statements in Swift do not fall through the bottom of each case and into the next one by default

```
let animalName = "chicken"
switch animalName {
case "cow":
    print("\(animalName)s have 4 legs")
case "chicken":
    print("\(animalName)s have 2 legs")
case "snake":
    print("\(animalName)s have no legs")
default:
    print("Other animal")
}
```

2.2 *Switch* Expression (cont)

- Values in *switch* cases can be check for their inclusion in an interval

```
var httpCode = 404
switch httpCode {
case 300 ... 308:
    print("It was transferred to a different URL. I'm sorry for causing you trouble")
case 400 ... 451:
    print("An error occurred on the application side. Please try again later!")
case 500 ... 511:
    print("A server error occurred. Please try again later!")
default:
    print("An error occurred. Please try again later!")
}
// Prints "An error occurred on the application side. Please try again later!"
```

2.2 *Switch* Expression (cont)

- A *switch* case can name the value or values it matches to temporary constants or variables, for use in the body of the case.

```
let point = (2, 0)
switch point {
case let (x, 0):
    print("on the x-axis with an x value of \(x)")
case let (0, y):
    print("on the y-axis with a y value of \(y)")
case let (x, y):
    print("somewhere else at (\(x), \(y))")
}
// Prints "on the x-axis with an x value of 2"
```

2.2 *Switch* Expression (cont)

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

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

3. Control Transfer Statements

1. continue
2. break
3. fallthrough

3.1 Continue

- The *continue* statement tells a loop to stop what it is and start again at the beginning of the next iteration through the loop.

```
let names = ["Mike", "Matt", "Nancy", "Adam", "Jenny", "Nancy", "Carl"]
var uniqueNames = [String]()
for name in names {
    if uniqueNames.contains(name) { continue }
    uniqueNames.append(name)
}
print(uniqueNames) // ["Mike", "Matt", "Nancy", "Adam", "Jenny", "Carl"]
```

3.2 Break

- The *break* statement has the following two usages:
  - When a *break* statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
  - It can be used to terminate a case in *switch* statement

3.2 Break (cont)

- *Break* in a Loop Statement

```
var index = 10
repeat {
    index += 1
    if index == 15 {
        break
    }
    print("Value of index is \(index)")
} while index < 20

// Value of index is 11
// Value of index is 12
// Value of index is 13
// Value of index is 14
```

3.2 Break (cont)

- *Break* statement causes execution of a *switch* statement to immediately end its execution.

```
let x = 0
switch x {
case 0:
    break
case 1:
    print("One for the money ... ")
case 2:
    print("Two for the road ... ")
default:
    print("Any other values")
}
print("Finished")
```

3.3 Fallthrough

- if Swift, *switch* statement don't fall through the bottom of each case and into the next one. If you need C-style fallthrough behavior, you can use *fallthrough* keyword.

```
let integerToDescribe = 5
var description = "The number \(integerToDescribe) is"
switch integerToDescribe {
case 2, 3, 5, 7, 11, 13, 17:
    description += " a prime number, and also"
    fallthrough
default:
    description += " an integer."
}
print(description)
// Prints "The number 5 is a prime number, and also an integer."
```

4. Early Exit

- A *guard* statement executes statements depending on the Boolean value of an expression:
  - ➔ It requires that a condition must be true in order to execute the code after the *guard* statement. Any variables or constants assigned values using optional binding are available for the rest of the code block.

  - ➔ It always has an *else* clause. That branch must transfer control to exit the code block with control transfer statement (*return, break, continue, throw*) or call a function or method that doesn't return `fatalError(_:file:line:)`

4. Early Exit (cont)

```swift
func greet(person: [String: String]) {
    guard let name = person["name"] else {
        return
    }
    print("Hello \(name)!")

    guard let location = person["location"] else {
        print("I hope the weather is nice near you.")
        return
    }
    print("I hope the weather is nice in \(location).")
}

greet(person: ["name": "John"])
// Prints "Hello John!"
// Prints "I hope the weather is nice near you."
greet(person: ["name": "Jane", "location": "Cupertino"])
// Prints "Hello Jane!"
// Prints "I hope the weather is nice in Cupertino."
```

5. Checking API Availability

- Swift has built-in support for checking API availability to ensure that you don't accidentally use APIs that are unavailable on a given development target.

```
let refreshControl = UIRefreshControl()

if #available(iOS 10.0, *) {
    tableView.refreshControl = refreshControl
} else {
    tableView.addSubView(refreshControl)
}
```

Question & Answer?