```swift
import UIKit
//: # Control Flow
/*:
 ### control flow statements
 * for-in loop
 * while loops
 * if, guard, and switch statements
 * break and continue statements
*/
//: ## For-In Loops
//: ### Example 1 : iterate over the items in an array:
let names = ["Anna", "Alex", "Brian", "Jack"]
for name in names {
    print("Hello, \(name)!")
}
// Hello, Anna!
// Hello, Alex!
// Hello, Brian!
// Hello, Jack!
//: ### Example 2 : iterate over a dictionary to access its key-value pairs:
let numberOfLegs = ["spider": 8, "ant": 6, "cat": 4]
for (animalName, legCount) in numberOfLegs {
    print("\(animalName)s have \(legCount) legs")
}
// ants have 6 legs
// cats have 4 legs
// spiders have 8 legs
//: ### Example 3 : using for-in loops with numeric ranges:
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
//: ### Note: index is a constant whose value is automatically set at the start of each iteration of the loop
//: * No need for a let declaration keyword
//: ### If you don’t need each value from a sequence --> use (_)
let base = 3
let power = 10
var answer = 1
for _ in 1...power {
    answer *= base
}
print("\(base) to the power of \(power) is \(answer)")
// Prints "3 to the power of 10 is 59049"
//: ### the half-open range operator (..<)
let minutes = 60
for _ in 0..<minutes {
    // render the tick mark each minute (60 times)
}
//: ### Use the stride(from:to:by:) function to skip the unwanted marks
let minuteInterval = 5
for tickMark in stride(from: 0, to: minutes, by: minuteInterval) {
    // render the tick mark every 5 minutes (0, 5, 10, 15 ... 45, 50, 55)
    print(tickMark, terminator : " ")
}
//: ### Closed ranges are also available, by using stride(from:through:by:) instead:
let hours = 12
let hourInterval = 3
for _ in stride(from: 3, through: hours, by: hourInterval) {
    // render the tick mark every 3 hours (3, 6, 9, 12)
}
//: ## While Loops
//: ## While
//: ### A while loop performs a set of statements until a condition becomes false
//: * These kinds of loops are best used when the number of iterations is not known
//: * while loops --> evaluates its condition at the start of each pass through the loop
//: * repeat-while --> evaluates its condition at the end of each pass through the loop
var index = 10

while index < 20 {
    print( "Value of index is \(index)")
    index = index + 1
}
//: ## Repeat-While
//: ### performs a single pass through the loop block first, before considering the loop’s condition
//: * It then continues to repeat the loop until the condition is false

repeat {
    print( "Value of index is \(index)")
    index = index + 1
} while index < 20
//: ### Note: The repeat-while loop in Swift is analogous to a do-while loop in other languages.
//: ## Conditional Statements
//: * if statement
//: * switch statement
//: ## if statement
var temperatureInFahrenheit = 30
if temperatureInFahrenheit <= 32 {
    print("It's very cold. Consider wearing a scarf.")
}
// Prints "It's very cold. Consider wearing a scarf."
//: ## if-else statement
temperatureInFahrenheit = 40
if temperatureInFahrenheit <= 32 {
    print("It's very cold. Consider wearing a scarf.")
} else {
    print("It's not that cold. Wear a t-shirt.")
}
// Prints "It's not that cold. Wear a t-shirt."
```

```swift
//: ### chaining multiple if statements
temperatureInFahrenheit = 90
if temperatureInFahrenheit <= 32 {
    print("It's very cold. Consider wearing a scarf.")
} else if temperatureInFahrenheit >= 86 {
    print("It's really warm. Don't forget to wear sunscreen.")
} else {
    print("It's not that cold. Wear a t-shirt.")
}
// Prints "It's really warm. Don't forget to wear sunscreen."
//: ### The final else clause is optional
temperatureInFahrenheit = 72
if temperatureInFahrenheit <= 32 {
    print("It's very cold. Consider wearing a scarf.")
} else if temperatureInFahrenheit >= 86 {
    print("It's really warm. Don't forget to wear sunscreen.")
}
// No message is printed
//: ## Switch
//: ### A switch statement considers a value and compares it against several possible matching patterns
//: ### Every switch statement must be exhaustive
//: ### default keyword is to cover any values that are not addressed explicitly and must always appear last
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"
//: ### The body of each case must contain at least one executable statement
//: ### This example code is not valid because the first case is empty:

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.
//: ### compound case
//: * each of the patterns is separated with a comma
//let anotherCharacter: Character = "a"
switch anotherCharacter {
case "a", "A":
    print("The letter A")
default:
    print("Not the letter A")
}
// Prints "The letter A"
//: ### Interval Matching
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).")
// Prints "There are dozens of moons orbiting Saturn."
//: ### Tuples
//: ### test multiple values in the same switch statement
//: ### use (_) to match any possible value
//: * (x, y) point as (Int, Int) tuple on a coordinate system
/*:
 ![Coordinate System](xy1.png)
*/
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"
//: ### Swift allows multiple switch cases to consider the same value or values
//: * the point (0, 0) could match all four of the cases in this example
//: * if multiple matches are possible, the first matching case is always used
//: * The point (0, 0) would match case (0, 0) first, and so all other matching cases would be ignored
//: ### Value Bindings
let anotherPoint = (2, 0)
switch anotherPoint {
```

```
case (let x, 0):
    print("on the x-axis with an x value of \(x)")
case (0, let 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"
// default case is not needed because this case matches all possible remaining
 values
//: ### Where
//:* check for additional conditions
/*:
 ![Coordinate System](xy2.png)
*/
//: ### in the code below the switch case matches the current value of point only if the where clause's condition evaluates to true for that value
//: ### the final case matches all possible remaining values, and so a default case is not needed to make the switch statement exhaustive
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"
//: ### Compound Cases
//: * Compound cases can also include value bindings
let stillAnotherPoint = (9, 0)
switch stillAnotherPoint {
case (let distance, 0), (0, let distance):
    print("On an axis, \(distance) from the origin")
default:
    print("Not on an axis")
}
// Prints "On an axis, 9 from the origin"
//: ## Control Transfer Statements
/*:
 ### Swift has five control transfer statements
 * continue
 * break
 * fallthrough
 * return
 * throw
*/
//: ## Continue
let puzzleInput = "great minds think alike"
var puzzleOutput = ""
```

```swift
let charactersToRemove: [Character] = ["a", "e", "i", "o", "u", " "]
for character in puzzleInput {
    if charactersToRemove.contains(character) {
        continue
    } else {
        puzzleOutput.append(character)
    }
}
print(puzzleOutput)
// Prints "grtmndsthnklk"
//: ## Break
//: ### Break in a Loop Statement
//: ### break statement in for loop
for i in 1...5 {
    if i == 4 {
        break
    }
    print("i = \(i)")
}
print("The end")
//: ### break statement in while loop
var currentLevel:Int = 1, finalLevel:Int = 2
var isLifeAvailable = true
while (isLifeAvailable) {

    if currentLevel > finalLevel {
        print("Game Completed. No level remaining")
        break
    }
    //play game and go to next level
    currentLevel += 1
    print("next level")
}
print("outside of while loop")
//: ### break statement with nested loops
for j in 1...2 {
    for i in 1...5 {
        if i == 4 {
            break
        }
        print("i = \(i)")
    }
    print("j = \(j)")
}
//: ### break statement with switch
//: * Swift's switch statement is exhaustive and does not allow empty cases
//: * Always use a break statement to ignore a switch case
let numberSymbol: Character = "三"  // Chinese symbol for the number 3
var possibleIntegerValue: Int?
switch numberSymbol {
case "1", "١", "一", "๑":
    possibleIntegerValue = 1
case "2", "٢", "二", "๒":
    possibleIntegerValue = 2
case "3", "٣", "三", "๓":
    possibleIntegerValue = 3
case "4", "٤", "四", "๔":
    possibleIntegerValue = 4
default:
    break
}
if let integerValue = possibleIntegerValue {
    print("The integer value of \(numberSymbol) is \(integerValue).")
} else {
    print("An integer value could not be found for \(numberSymbol).")
}
// Prints "The integer value of 三 is 3."
//: ## Fallthrough
//: ### Example 1:
let integerToDescribe = 5
var description = "The number \(integerToDescribe) is"
switch integerToDescribe {
case 2, 3, 5, 7, 11, 13, 17, 19:
    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."
//: ### Example 2:
let dayOfWeek = 4
switch dayOfWeek {
case 1 :
    print("Sunday")
case 2:
    print("Monday")
case 3:
    print("Tuesday")
case 4:
    print("Wednesday")
    fallthrough
case 5:
    print("Thursday")
case 6:
    print("Friday")
case 7:
    print("Saturday")
default:
    print("Invalid day")
}
```

```swift
//: ## Labeled Statements
/*:
 ### Statements that have prefixes in the form (label : Statement) are called
 as labeled statement
 * With a conditional statement, you can use a statement label with the break statement to end the execution
   of the labeled statement
 * With a loop statement, you can use a statement label with the break or continue statement to end or
   continue the execution of the labeled statement
*/
//: ### Example without labeled statement
for x in 1...3 {
    for y in 1...3 {
        if y == 2{
            continue
        }
        print("x = \(x), y = \(y)")
    }
}
//x = 1, y = 1
//x = 1, y = 3
//x = 2, y = 1
//x = 2, y = 3
//x = 3, y = 1
//x = 3, y = 3
//: ### Now usig the above code with labeled statements
outerloop: for x in 1...3 {
    innerloop: for y in 1...3 {
        if y == 2{
            continue outerloop
        }
        print("x = \(x), y = \(y)")
    }
}
//x = 1, y = 1
//x = 2, y = 1
//x = 3, y = 1
//: ### Labeled Statement with break
outerloop: for j in 1...2{
    innerloop: for i in 1...5 {
        if i == 4 {
            break outerloop
        }
        print("i = \(i)")
    }
    print("j = \(j)")
}
//: ## Early Exit
//: ### guard statement
//: ### unlike if, the guard statements only run when certain conditions are not met
//: ### Example: labeled statement with a conditional statement
var num = 55
```

```
OuterIf : if num is Int {

    guard num > 0 else {
        print("\(num) is not > 0")
        break OuterIf
    }
    guard Int(num) % 2 == 0 else {
        print("\(num) is not even")
        break OuterIf
    }
    print("\(num) is positive and even")
}
```