Closures

closure 是自包含的代码 block，可以从定义其的上下文捕获常量/变量，嵌套的函数是 closure 的特殊情况，一般 closure 有三种形式

- 全局函数是有名且不捕获值的 closure
- 嵌套函数是有名且能从外部函数捕获值的 closure
- closure expression 是无名且能从上下文捕获值的 closure

Closure Expression

是相比嵌套函数更简洁的形式

考虑 swift 提供的 `sorted(by:)` 函数，根据提供的 closure 来排序，其接受的参数是一个函数，接受两个数组内的值的类型，并返回一个 bool 值，如果第一个参数要排在前面则返回 true，否则返回 false

closure expression 的语法形如

```
{ (parameters) -> return_type in
    statements
}
```

参数表中的参数不能有默认值，但是可以是 in-out parameter，也可以是可变长的参数

```
reversedNames = names.sorted(by: { (s1: String, s2: String) ->
Bool in
    return s1 > s2
})
```

closure expression 可以从上下文推导出参数类型和返回值类型，因此可以写得更简洁

```
reversedNames = names.sorted(by: { s1, s2 in return s1 > s2 } )
```

当将 closure expression 作为参数传递给函数时，总是能推导出类型

如果只有一条表达式，可以省去 return

```
reversedNames = names.sorted(by: { s1, s2 in s1 > s2 } )
```

甚至可以省去参数列表而使用参数的简写，在这种情况下参数列表和 in 都可以省去

```
reversedNames = names.sorted(by: { $0 > $1 } )
```

或是直接使用操作符

```
reversedNames = names.sorted(by: >)
```

Trailing Closure

对于接受函数作为参数的函数，如果 closure expression 太长，可以将其写在函数调用的括号之后而非之内

```
// Here's how you call this function without using a trailing closure:

someFunctionThatTakesAClosure(closure: {
    // closure's body goes here
})

// Here's how you call this function with a trailing closure instead:

someFunctionThatTakesAClosure() {
    // trailing closure's body goes here
}
```

这种称为 trailing closure，如果 closure 是唯一的参数，且写成了 trailing closure 的形式，则可以省去调用的括号

```
reversedNames = names.sorted { $0 > $1 }
```

在 closure 过长的情况下都可以采用这种方式

如果有多个参数，可以省去第一个参数的 argument label，然后在之后的参数使用 label

```
loadPicture(from: someServer) { picture in
    someView.currentPicture = picture
} onFailure: {
    print("Couldn't download the next picture.")
}
```

Capturing Values

closure 能从上下文捕获常量和变量，然后在之后访问修改被捕获的值，即使定义这些值的上下文已经消亡了

最简单的形式就是嵌套函数

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

内部函数对外部函数的变量的引用确保了在函数调用结束后这些变量不会消亡

虽然可以将 closure 赋给常量，但是仍能修改值，这是因为 closure/function 本质是 reference type，即如果将一个 closure 赋给两个变量，这两个变量指向同一个 closure

Escaping Closures

如果一个 closure 被作为参数传给一个函数，但却在函数返回后被调用，则称为这个 closure escape 了这个函数，可以在参数类型前加上 `@escaping` 来显式指明

```
var completionHandlers = [() -> Void]()
func someFunctionWithEscapingClosure(completionHandler:
@escaping () -> Void) {
    completionHandlers.append(completionHandler)
}
```

这种情况下不加上 `@escaping` 会触发一个编译时错误

Autoclosures

autoclosure 是一个自动创建的 closure，用于包裹并传递一个表达式给函数

```swift
var customersInLine = ["Chris", "Alex", "Ewa", "Barry",
"Daniella"]
print(customersInLine.count)
// Prints "5"

let customerProvider = { customersInLine.remove(at: 0) }
print(customersInLine.count)
// Prints "5"

print("Now serving \(customerProvider())!")
// Prints "Now serving Chris!"
print(customersInLine.count)
// Prints "4"
```

这种情况可以延后表达式的计算，只有在 closure 被调用时才计算

如果用 `@autoclosure` 标记，则在传参时可以直接传返回类型

```swift
// customersInLine is ["Alex", "Ewa", "Barry", "Daniella"]
func serve(customer customerProvider: () -> String) {
    print("Now serving \(customerProvider())!")
}
serve(customer: { customersInLine.remove(at: 0) } )
// Prints "Now serving Alex!"

// customersInLine is ["Ewa", "Barry", "Daniella"]
func serve(customer customerProvider: @autoclosure () ->
String) {
    print("Now serving \(customerProvider())!")
}
serve(customer: customersInLine.remove(at: 0))
// Prints "Now serving Ewa!"
```