Swift程序设计实践

Swift Programming Experiments

张 亮 | 计算机学院 | 北京航空航天大学

函数

定义和调用

函数分类

无形参函数、无返回值函数、多个返回值函数、
可变形参函数、inout类型参数

函数类型

函数类型的变量、参数、返回值

嵌套函数

定义和调用

函数名

用来标识函数的，可以理解为一个函数的代号。

函数名要能够比较清楚而简略的描述该函数所完成的任务，从而大大提高函数的可读性。

参数表

要定义函数的传入值的类型，即：形参的类型。

返回值

要定义函数执行完毕后返回值的类型。

函数体

要定义函数体，即：一系列执行语句，用来完成特定的任务。

函数调用

通过函数名来调用函数，要根据形参类型来传入值。

注意:函数有返回值时，接收函数返回值的变量类型要和返回值类型一致。

定义和调用

函数名

用来标识函数的，可以理解为一个函数的代号。

函数名要能够比较清楚而简略的描述该函数所完成的任务，从而大大提高函数的可读性。

参数表

要定义函数的传入值的类型，即：形参的类型。

返回值

要定义函数执行完毕后返回值的类型。

函数体

要定义函数体，即：一系列执行语句，用来完成特定的任务。

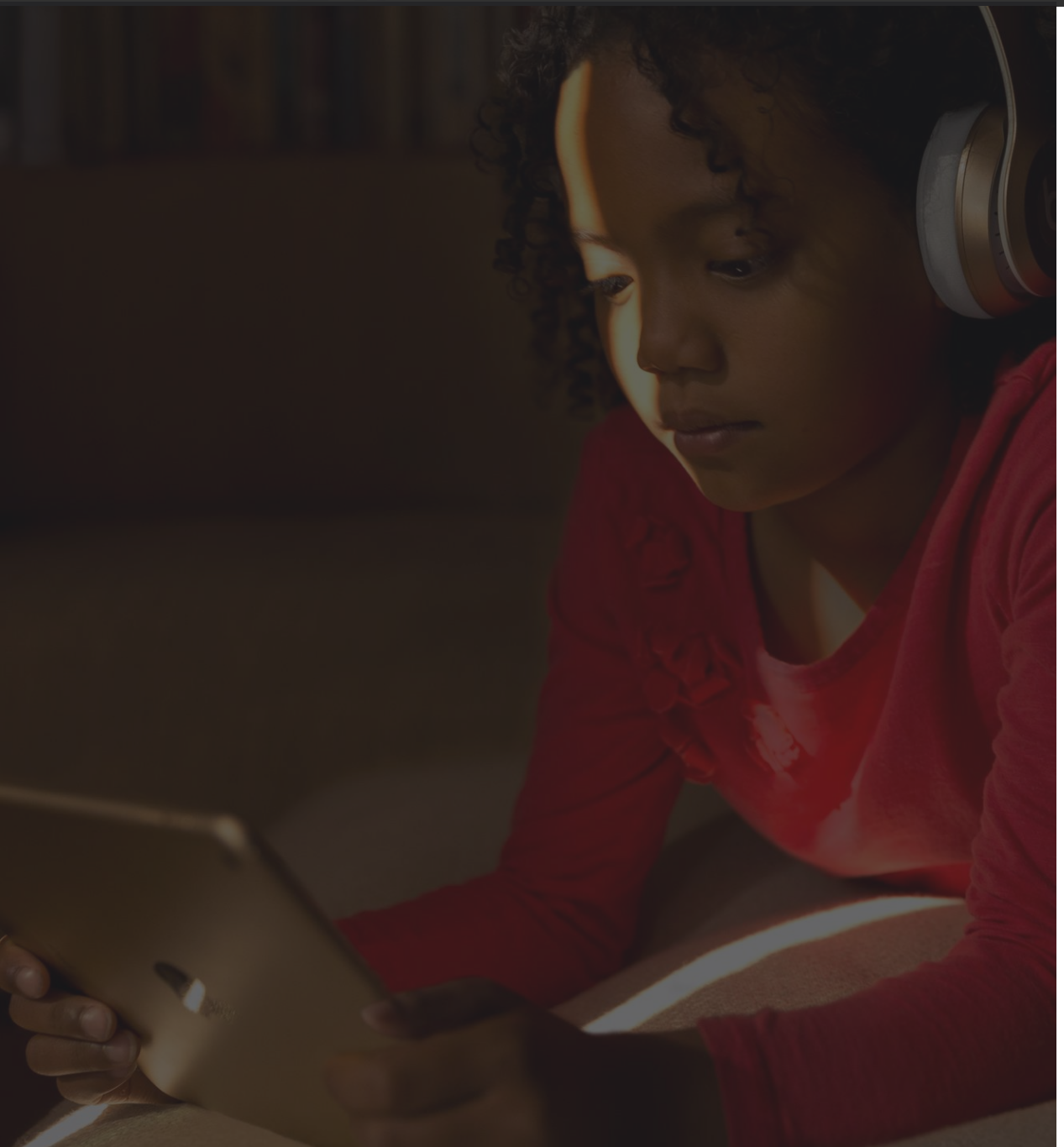
函数调用

通过函数名，并根据形参的类型传入值。当函数有返回值的时候，还要注意接收函数返回值的变量的类型要和返回值类型一致。

例子

```
func mulAdd(mul1:Int,mul2:Int,add:Int) -> Int {
    let result = mul1*mul2 + add
    return result
}

var result : Int

result = mulAdd(3, mul2: 5, add: 3)

print("The result of mulAdd is \(result)")
```

```
18
18


18

"The result of mulAdd is 18\n"
```



函数分类

无形参函数

无返回值函数

多个返回值函数

可变形参函数

inout类型参数

无形参函数

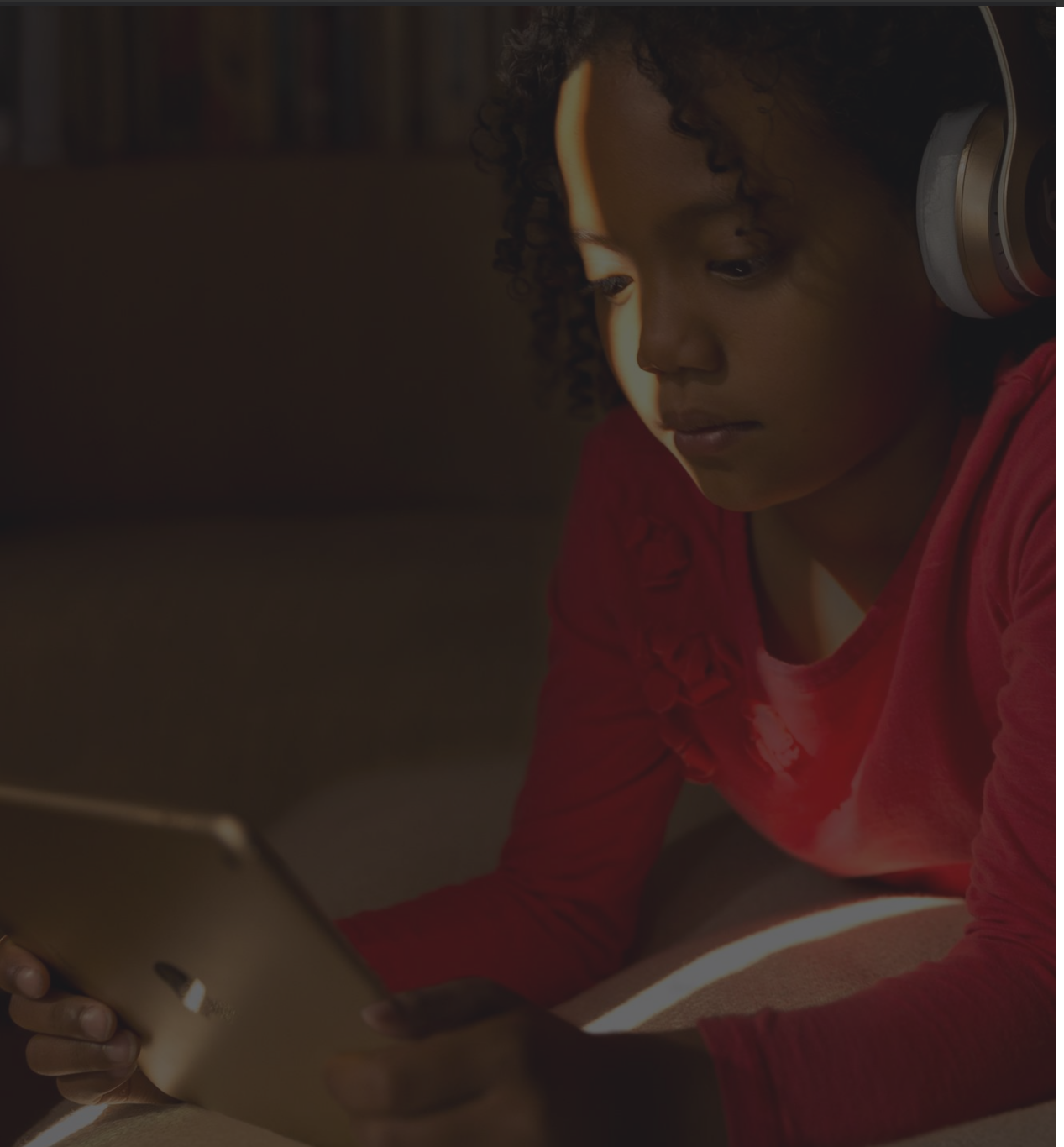
函数可以有一个形参或多个形参，也可以没有形参

不带形参的函数在调用的时候，仍然需要在函数名后面跟着一对空的括号

例子

```
func mulAdd() -> Int {
    let mul1,mul2,add : Int
    mul1 = 3
    mul2 = 5
    add = 3
    return mul1*mul2 + add
}

var result = mulAdd()

print("The result of mulAdd is \(result)")
```

```
3
5
3
18

18

"The result of mulAdd is 18\n"
```



无返回值函数

函数不仅可以没有参数，也可以没有返回类型。

Swift和C的差别

当函数没有返回类型的时候，在C语言中称为过程。在Swift中，并没有将无返回类型的函数作为一种特殊的情况来考虑。

例子

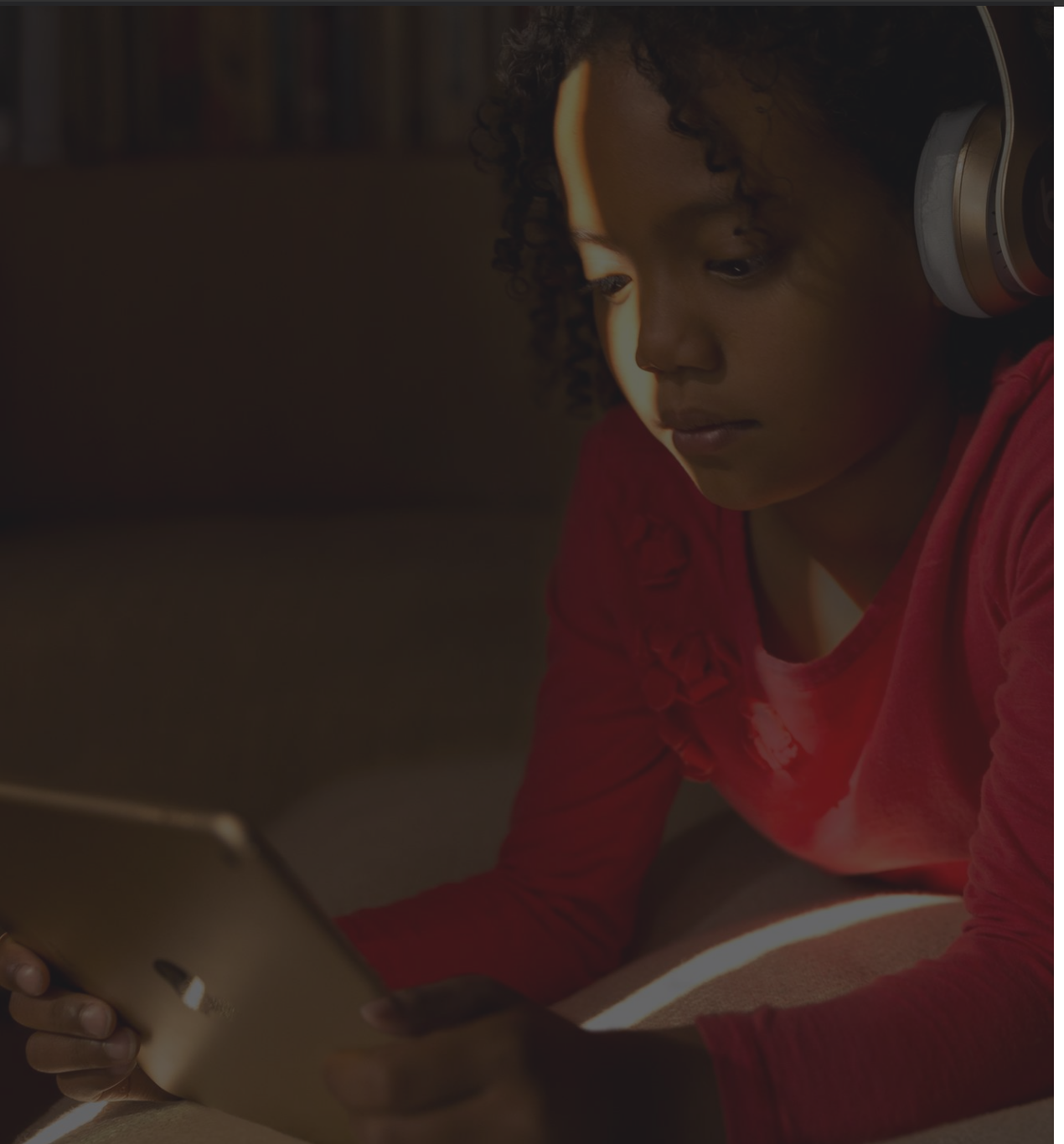
```
func mulAdd(mul1:Int,mul2:Int,add:Int){
    print("The result of mulAdd is \(mul1*mul2 + add)")
}

mulAdd(mul1:3, mul2:5, add:3)
```

多个返回值函数

如果函数的返回值为多个，就要用元组作为返回值类型。

例子

```
func climate(city:String)->(averageTemperature:Int,weather:String,wind:String) {
    var averageTemperature : Int
    var weather,wind : String
    switch city {
    case "beijing": averageTemperature = 25;weather = "dry";wind = "strong"
    case "shanghai": averageTemperature = 15;weather = "wet";wind = "weak"
    default : averageTemperature = 10; weather = "sunny"; wind = "normal"
    }

    return (averageTemperature,weather,wind)
}

var climateTemp = (0, "", "")
climateTemp = climate(city: "beijing")
```



可变形参函数

定义

除了确定个数的形参外，Swift还支持形参个数不确定的函数，即：在函数定义的时候只知道形参的类型，并不知道形参的个数。只有在调用函数的时候才能确定形参的个数。这种函数称为可变形参函数。

用法

可变形参在定义的时候，只需要定义参数名和参数类型，不同之处在于参数类型后面要加上“...”符号，表示该形参为可变形参。

可变形参在函数体内是以数组的形式存在的。

例子

```
func sum(numbers : Int...) -> Int{
    var result = 0
    for number in numbers {
        result = result + number
    }
    return result
}

sum(numbers: 1,2,3,4,5,6,7,8,9)
sum(numbers: 10,11,12)
```

inout类型参数

形参的局限性

形参只能在函数体内使用和改变，不影响函数体外传入的变量值。调用函数的时候，将函数外部变量作为参数传入函数体内的是变量值，而不是变量本身。当变量的值在函数体内发生变化时，函数体外的变量值并不会受到影响。

解决方案：inout

如果要通过函数对传入的变量产生影响时，就需要在函数定义时，将形参的类型中加上关键字“inout”，表明该参数值的变化会影响传值给它的外部变量。

inout实现机制

inout类型的参数是通过变量地址来实现的。

调用函数时，传递给函数中inout参数的不再是变量的值，而是变量的地址。在函数体中，当inout参数的值发生变化时，必然会导致指向同一地址的外部参数值发生改变。

注意

在调用函数时，不能将一个常量或者字面量传递给一个inout类型的参数。

inout类型参数

形参的局限性

形参只能在函数体内使用和改变，不影响函数体外传入的变量值。调用函数的时候，将函数外部变量作为参数传入函数体内的是变量值，而不是变量本身。当变量的值在函数体内发生变化时，函数体外的变量值并不会受到影响。

解决方案：inout

如果要通过函数对传入的变量产生影响时，就需要在函数定义时，将形参的类型中加上关键字“inout”，表明该参数值的变化会影响传值给它的外部变量。

inout实现机制

inout类型的参数是通过变量地址来实现的。

调用函数时，传递给函数中inout参数的不再是变量的值，而是变量的地址。在函数体中，当inout参数的值发生变化时，必然会导致指向同一地址的外部参数值发生改变。

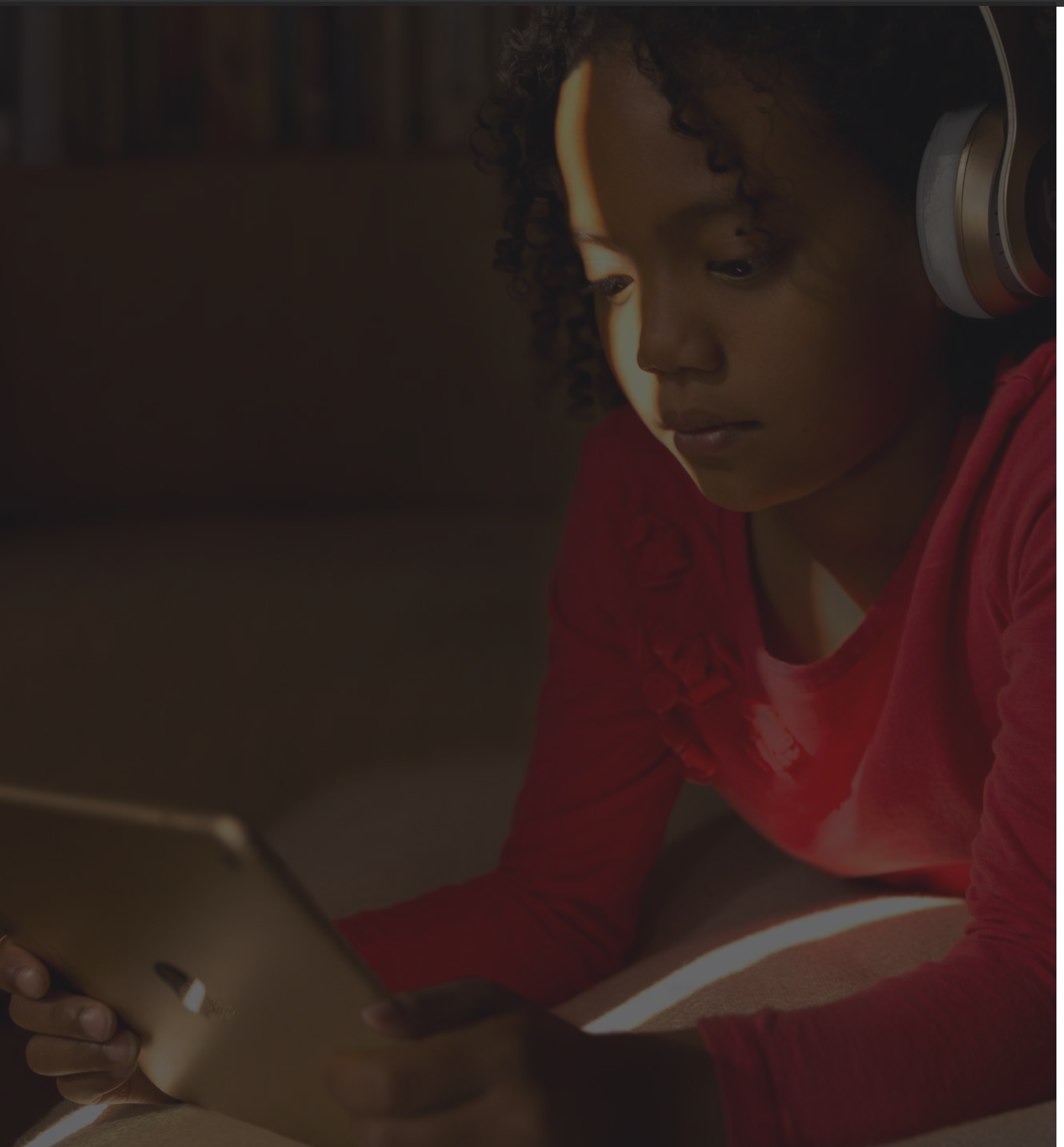
注意

在调用函数时，不能将一个常量或者字面量传递给一个inout类型的参数。

例子

```
func swap(a:inout Int, b:inout Int) {
    let temp = a
    a = b
    b = temp
}

var a = 5
var b = 6
swap(a: &a, b: &b)
print("a is \(a)")
print("b is \(b)")
```

取址运算符 &



函数类型

定义

每一个函数都有特定的函数类型。函数类型由形参类型和返回值类型共同组成。

例子

```
func add(a: Int,b: Int) ->Int{
    return a+b
}

func helloWorld() {
    print("Hello world!")
}
```

函数类型的变量

函数类型可以像其他数据类型一样使用。

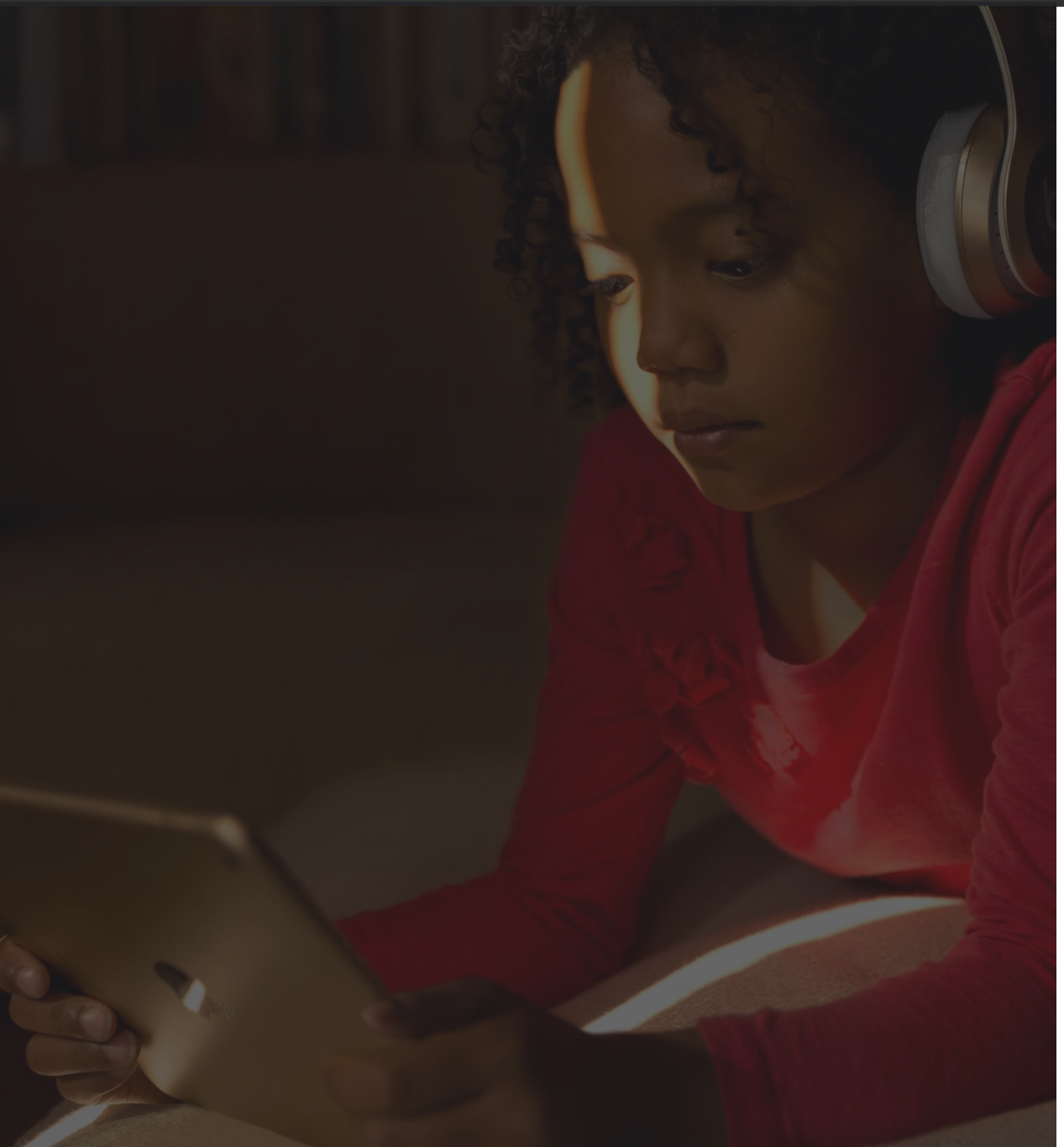
将函数类型赋值给一个变量。

变量的类型不需要显式的指出，编译器会通过类型推断来得到变量类型。

例子

```
var mathOperation : (Int,Int)->Int = add
var sayOperation : ()->() = helloWorld

mathOperation(5,6)
sayOperation()

var operation = add
operation(6, 7)
```

```
(Int, Int) -> Int
() -> ()

11


(Int, Int) -> Int
13
```



函数类型的参数

作为函数的参数

函数类型还可以作为另一个函数的参数类型来使用。

运行期确定函数类型

增加了函数定义的灵活性，可以在调用函数的时候，再根据需要来决定哪个函数被调用，而不需要在编写代码时就确定。

例子

```
func add(a: Int,b: Int) ->Int{
    return a + b
}

func sub(a: Int,b: Int) ->Int{
    return a - b
}

func printResult(operation: (Int,Int)->Int, a:Int, b:Int) {
    let result : Int
    if a>b {
        result = operation(a,b)
    } else {
        result = operation(b,a)
    }
    print("the result is \(result)")
}

printResult(operation: sub, a: 3, b: 9)
```

函数类型的参数

作为函数的参数

函数类型还可以作为另一个函数的参数类型来使用。

运行期确定具体函数

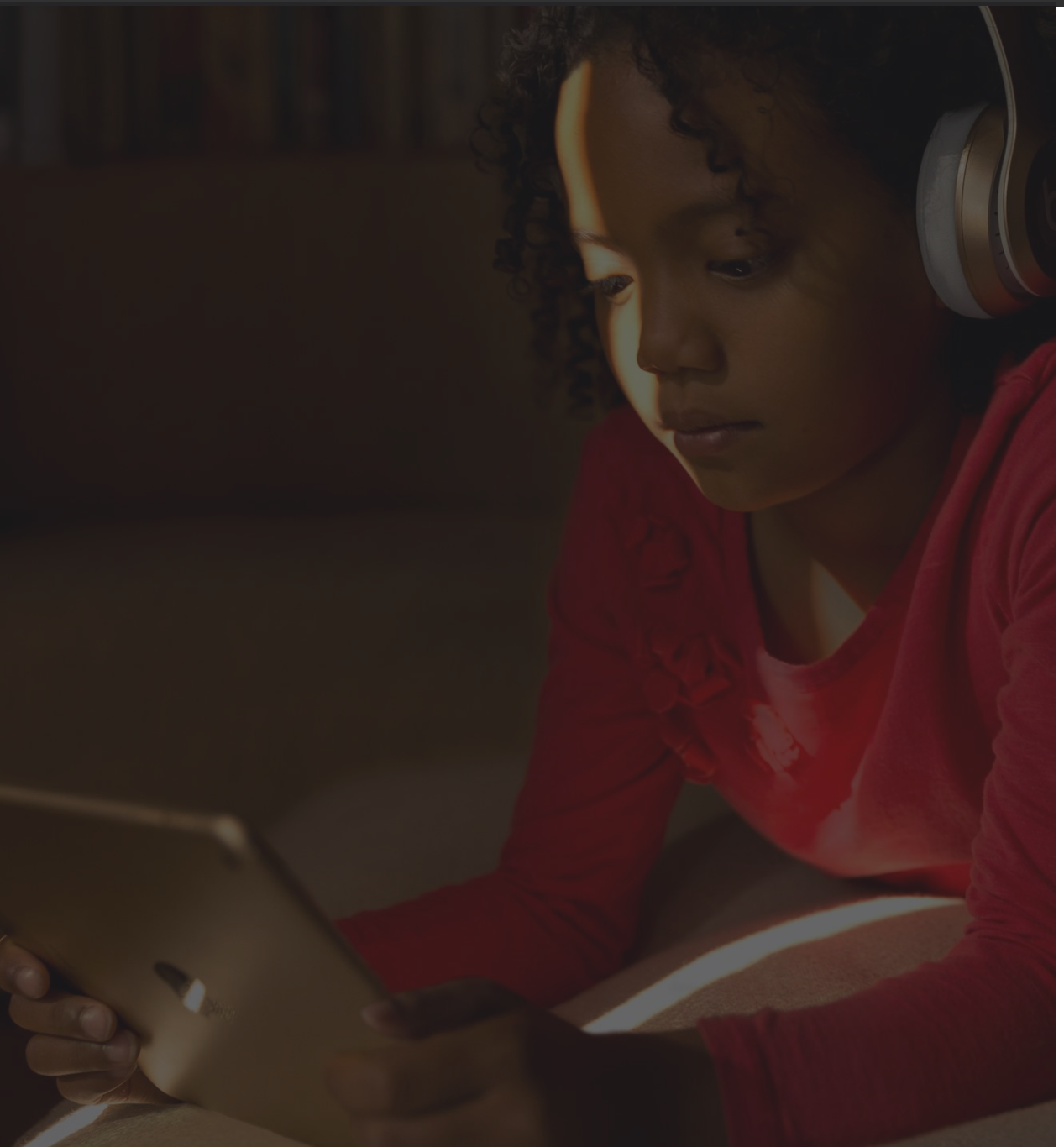
增加了函数定义的灵活性。在调用函数的时候，再来确定具体函数，而不需要在编写代码时候就写死。

函数类型的返回值

函数类型也可以作为另一个函数的返回值类型来使用。

例子

```
func mathOperation(op: String) -> (Int,Int)->Int {
    if op == "sub" {
        return sub
    }else {
        return add
    }
}

let result = mathOperation(op: "sub")
result(6,3)
```



嵌套函数

定义

在函数体内定义新的函数，称为嵌套函数。

使用范围

嵌套函数只能在函数体内使用，一般来说对于函数体外是不可见的。

例子

```
func printResult(a:Int, b:Int) {
    func add(a: Int,b: Int) ->Int{
        return a + b
    }

    func sub(a: Int,b: Int) ->Int{
        return a - b
    }

    let result : Int
    if a>b {
        result = sub(a,b: b)
    } else {
        result = add(b,b: a)
    }
    print("the result is \(result)")
}

printResult(6,b: 3)
```

```
3

3

"the result is 3\n"
```

闭包

闭包表达式

全局函数类型的闭包、闭包表达式、闭包的定义和调用、闭包表达式的简化、无参数无返回值闭包、尾随闭包的定义和写法。

闭包的应用

捕获闭包作用域的变量和常量、通过闭包来定制集合类型数据的迭代操作（包括：forEach、filter、map、reduce）。

闭包

定义

闭包是一种功能性自包含模块，可以捕获和存储上下文中任意常量和变量的引用。

闭包的三种形式

全局函数：有名字但不会捕获任何值的闭包。

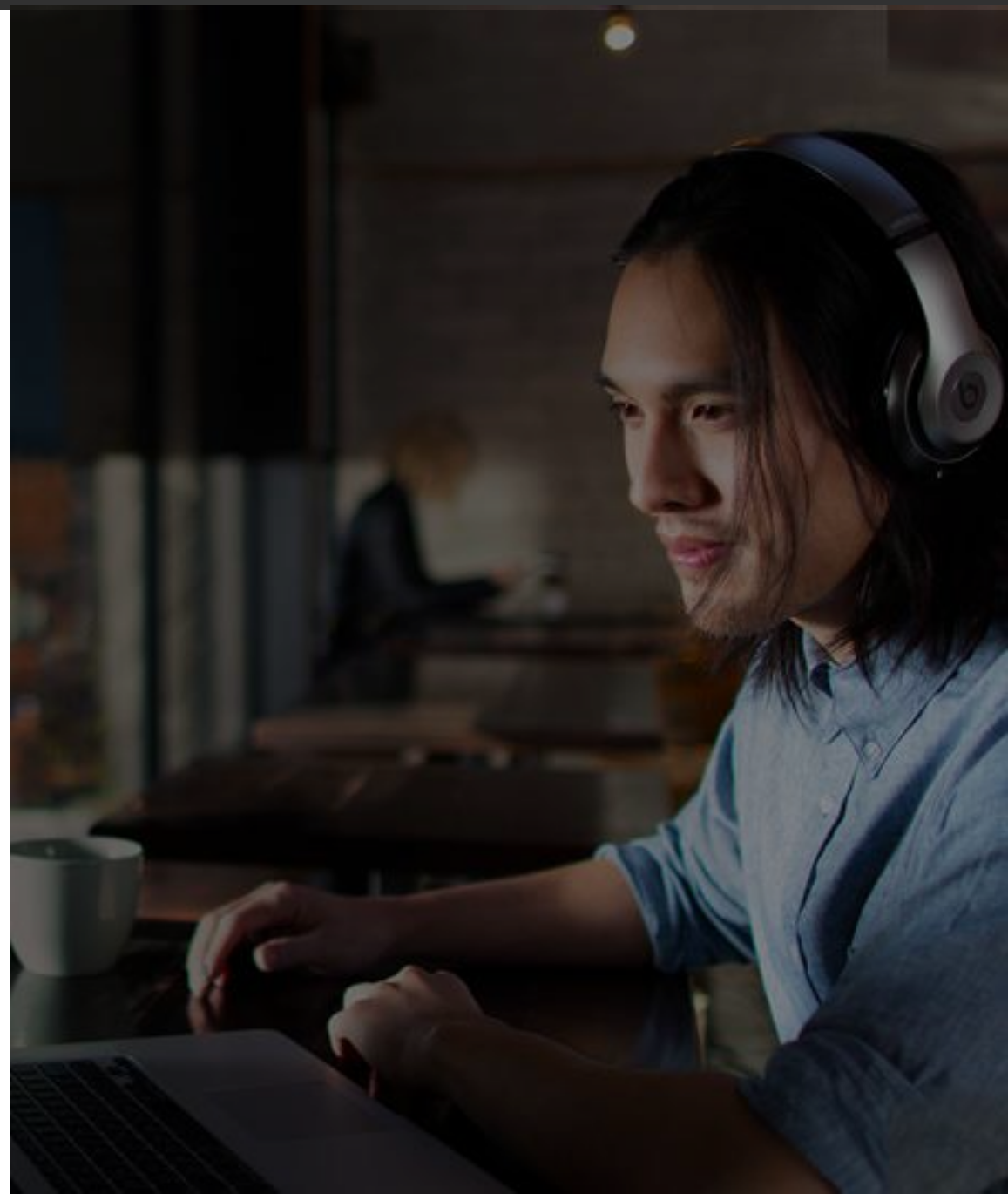
嵌套函数：有名字并可以捕获其封闭函数域内值的闭包。

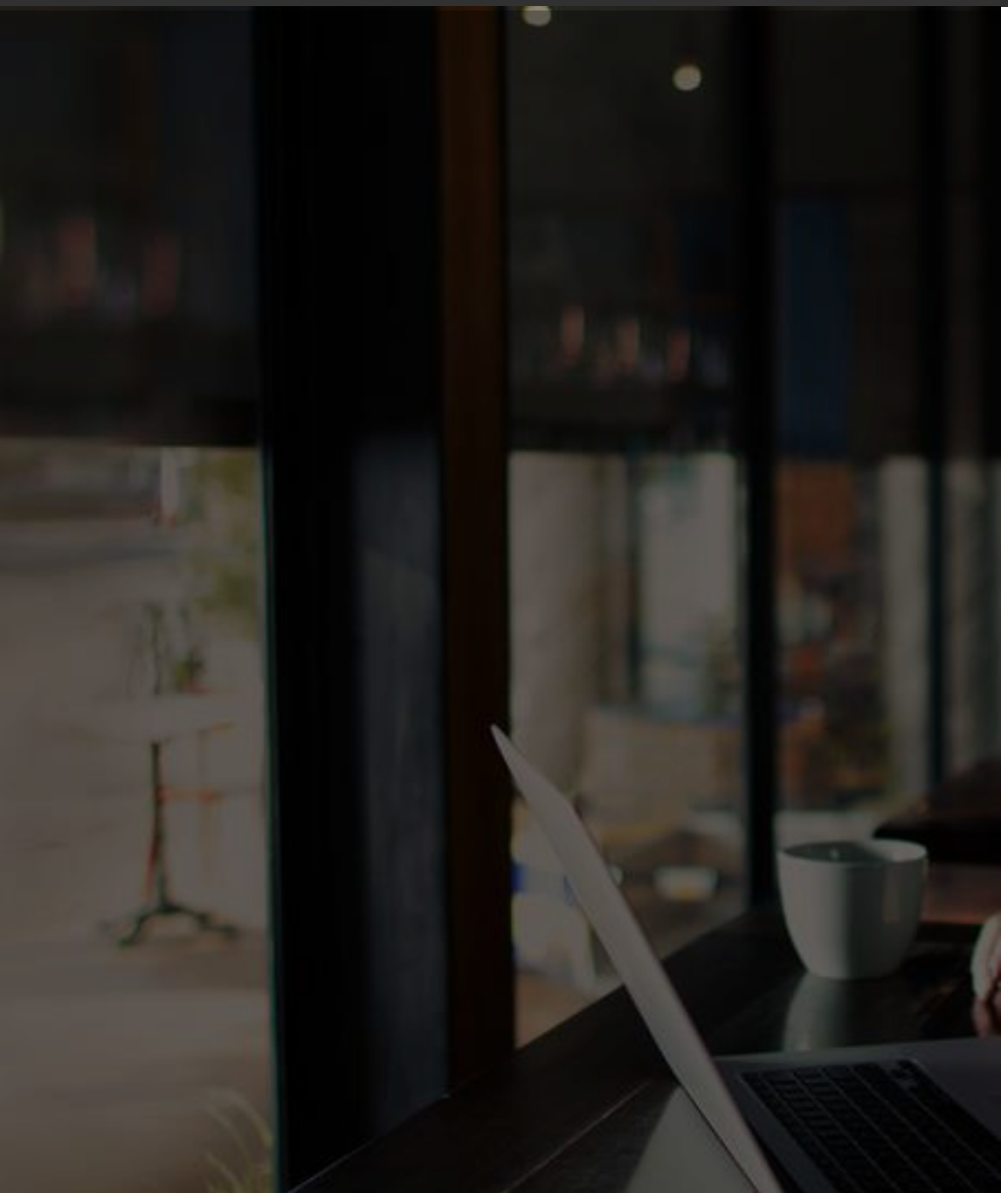
闭包表达式：没有名字但可以捕获上下文中变量和常量值的闭包。

嵌套函数 VS 闭包表达式

嵌套函数是一种在复杂函数中命名和定义自包含代码块的简洁方式。

闭包表达式则是一种利用内联闭包的方式实现的更为简洁的方式。





闭包表达式

格式

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

使用规则

闭包表达式的参数可以为常量和变量，也可以使用inout类型，但不能提供默认值。

在参数列表的最后可以使用可变参数。

可以使用元组作为参数和返回值。

闭包表达式

例子

```
func exchange(s1:String, s2:String)->Bool {
    return s1 > s2
}

let cityArray =
    ["Beijing","Shanghai","Guangzhou","Hangzhou","Suzhou"]

var descendingArray = cityArray.sorted(by: exchange)
```

```
(10 times)

["Beijing", "Shanghai", "Guangzhou", "Hangzhou", "Suzhou"]

["Suzhou", "Shanghai", "Hangzhou", "Guangzhou", "Beijing"]
```

用闭包来改写

```
var descengdingArrayByClosures = cityArray.sorted(by:
    {(s1:String, s2:String)->Bool in return s1 > s2})
print(descengdingArrayByClosures)
```

```
(11 times)

"["Suzhou", "Shanghai", "Hangzhou", "Guangzhou", "Beijing"]\n"
```

格式

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

使用规则

闭包表达式的参数可以为常量和变量，也可以使用inout类型，但不能提供默认值。

在参数列表的最后可以使用可变参数。

可以使用元组作为参数和返回值。

闭包类型参数

sorted方法

数组类型有一个sorted方法，功能是对数组中的值进行排序。

排序规则

由一个已知的闭包来提供。该方法的返回值为一个排过序的新数组。

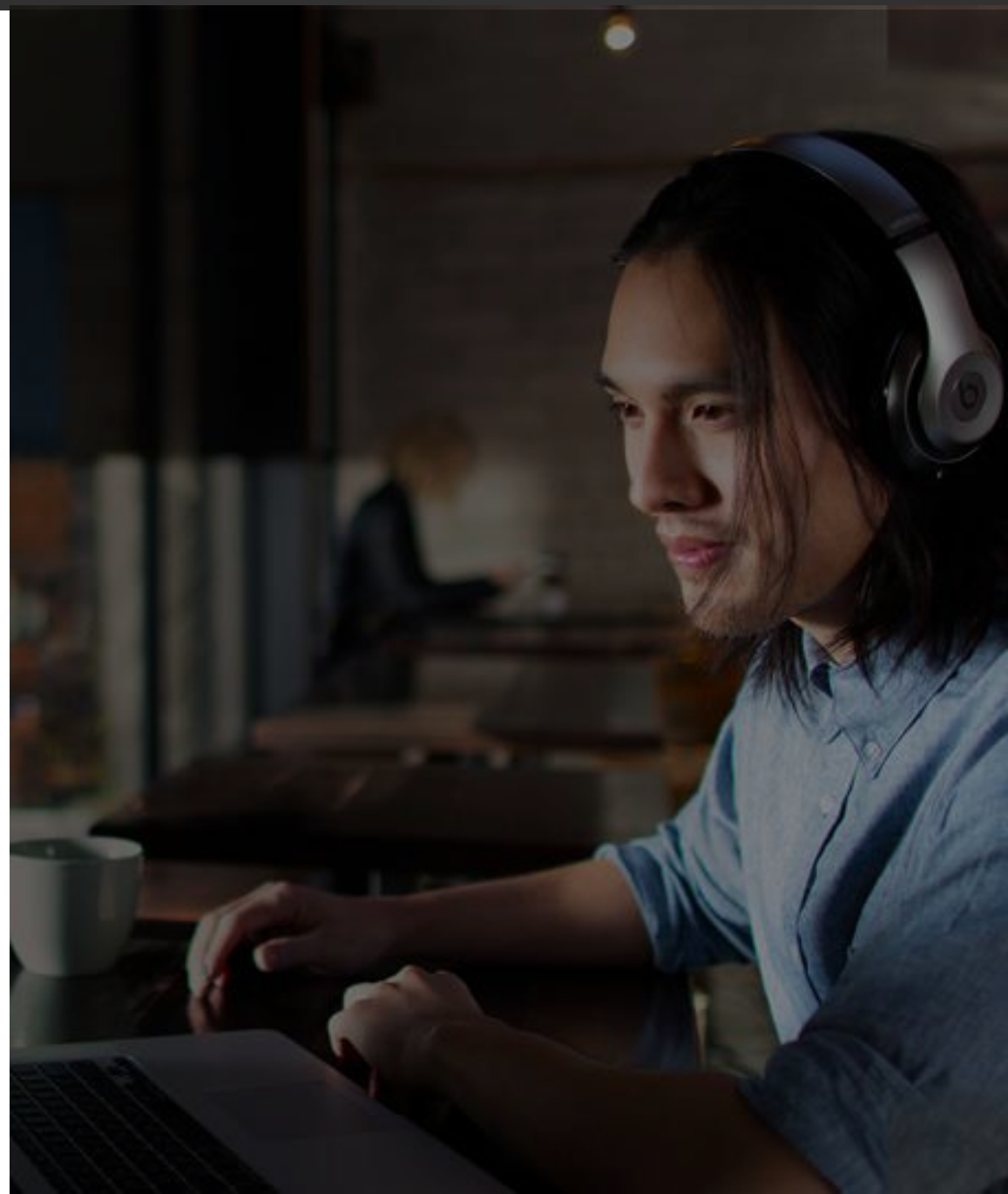
sorted方法的参数

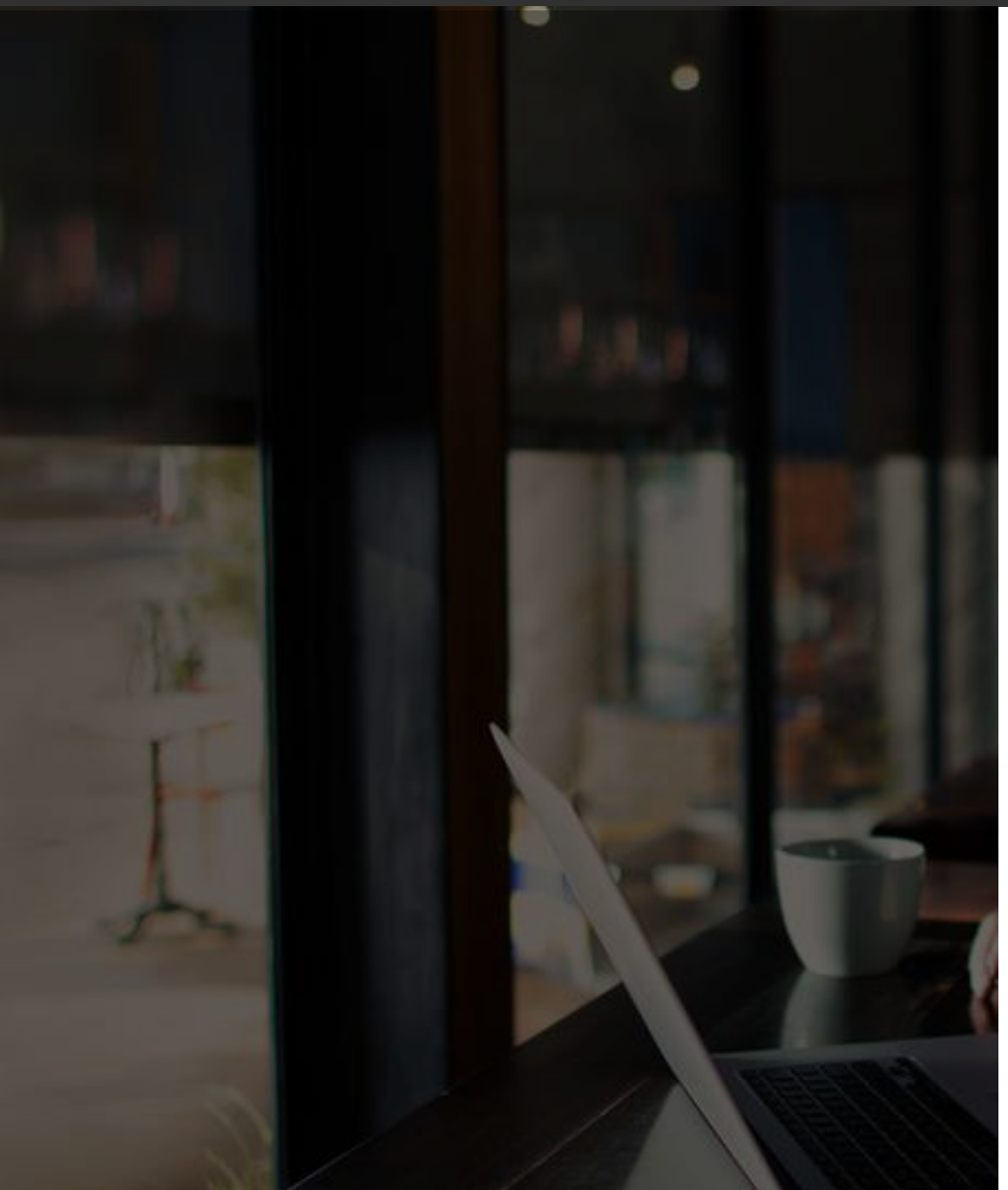
sorted方法有一个闭包类型参数。

该闭包有两个相同类型的参数，参数类型与数组元素的类型一致。

闭包的返回值

闭包的返回值要求为布尔型。当返回值为true时，表示第一个元素和第二个元素位置保持不变。当返回值为false时，表示第一个元素应该和第二个元素交换位置。





数组排序

例子

用方法sorted对一个String类型的数组按照字母降序排列

数组的初始值为

["Beijing","Shanghai","Guangzhou","Hangzhou","Suzhou"]。

那么，sorted方法的参数类型就为(String,String)->Bool的闭包。

```
func exchange(s1:String, s2:String)->Bool {
    return s1 > s2
}

let cityArray = ["Beijing","Shanghai","Guangzhou","Hangzhou","Suzhou"]

var descendingArray = cityArray.sorted(by: exchange)
print(descendingArray)
```

闭包的应用

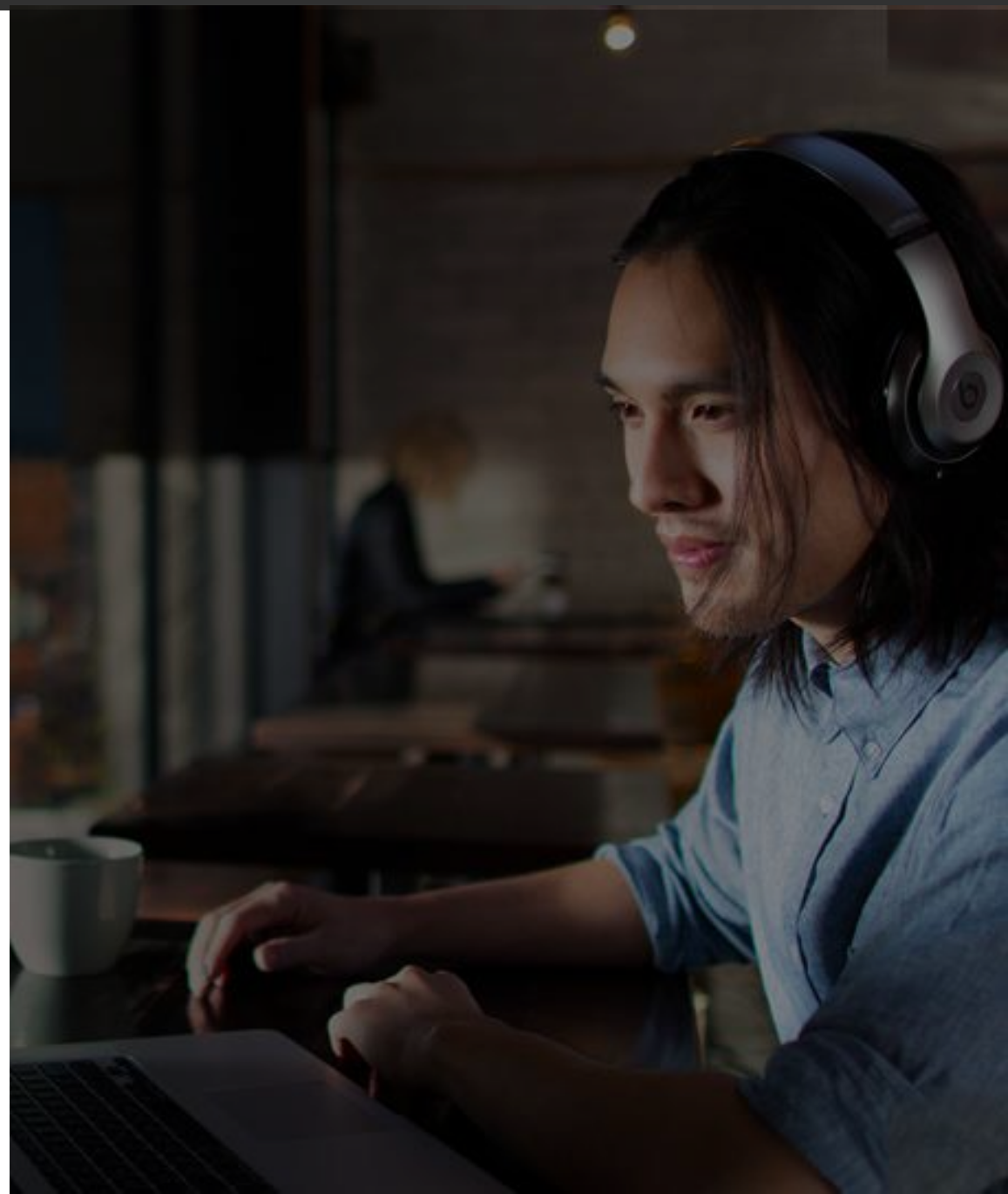
参数

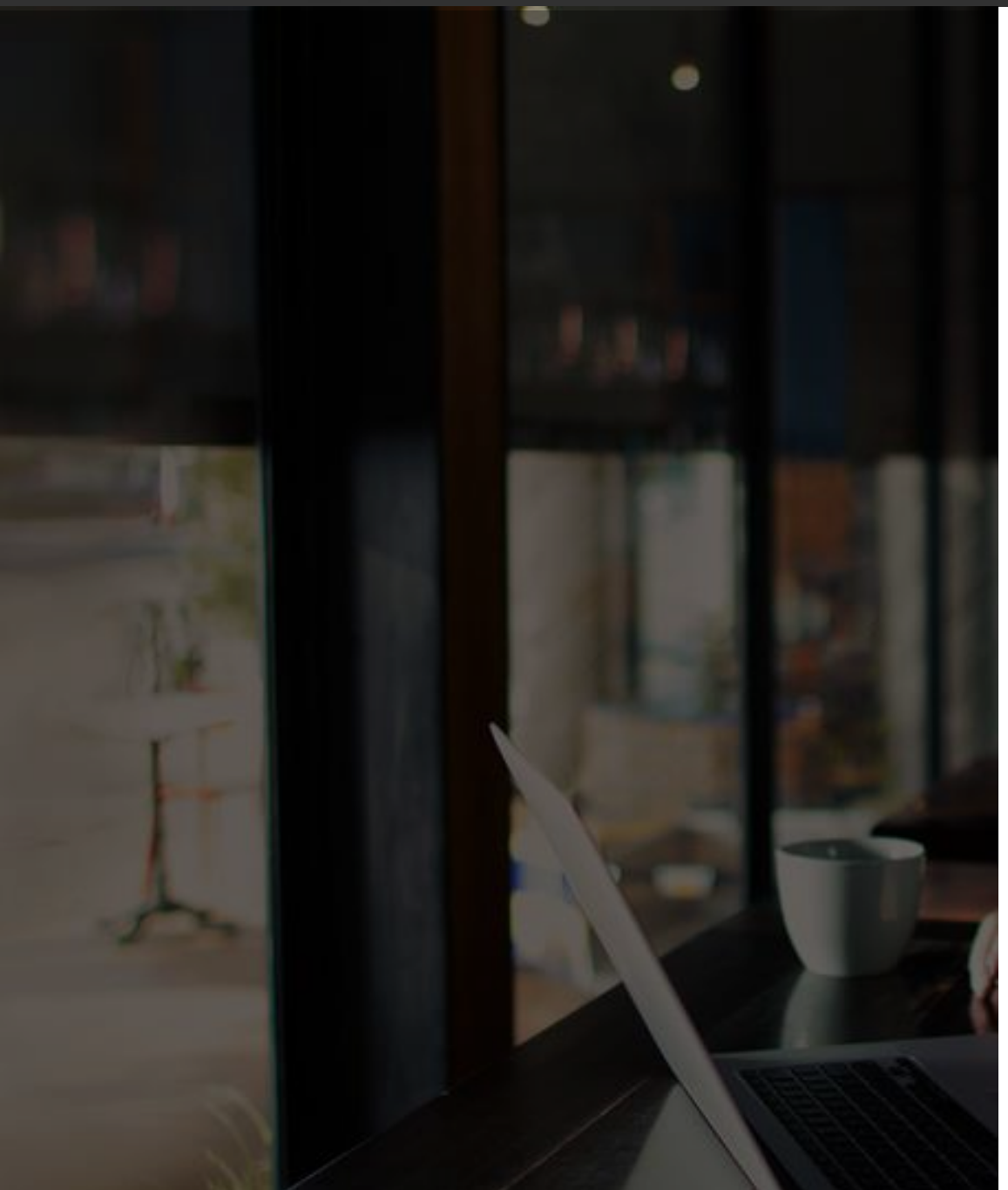
作为sorted方法的闭包类型的参数

```
var descengdingArrayByClosures = cityArray.sorted(by: {(s1:String, s2:String)->Bool
    in return s1 > s2})
print(descengdingArrayByClosures)
```

闭包表达式的简化

```
var descendingArray2 = cityArray.sorted(by: {s1,s2 in return s1>s2})
print(descendingArray2)

var descendingArray3 = cityArray.sorted(by: {s1,s2 in s1>s2})
print(descendingArray3)

var descendingArray4 = cityArray.sorted(by: {$0 > $1})
print(descendingArray4)

var descendingArray5 = cityArray.sorted(by: >)
print(descendingArray5)
```





无参数无返回值的闭包

定义

当闭包的类型为()->Void时，称之为无参数无返回值闭包。

说明：

一对空括号表示没有参数，但不能省略。

Void表示无返回值，也不能省略，否则编译器无法断定其为闭包。

声明

在定义一个变量为闭包类型时，闭包的具体类型可以显式声明，也可以通过赋初始值来隐式声明，两者的效果是一样的。

无参数无返回值的闭包

例子

```
var voidClosureWithTypeDeclaration : () -> Void =
    {
    print("A closure without parameters and
        return value")
}

var voidClosure = {
    print("Another clousre without parameters and
        return value")
}
voidClosureWithTypeDeclaration()
voidClosure()
```

```
() -> ()

()


() -> ()
()
```

定义

当闭包的类型为()->Void时，称之为无参数无返回值闭包。

说明：

一对空括号表示没有参数，但不能省略。

Void表示无返回值，也不能省略，否则编译器无法断定其为闭包。

声明

在定义一个变量为闭包类型时，闭包的具体类型可以显式声明，也可以通过赋初始值来隐式声明，两者的效果是一样的。

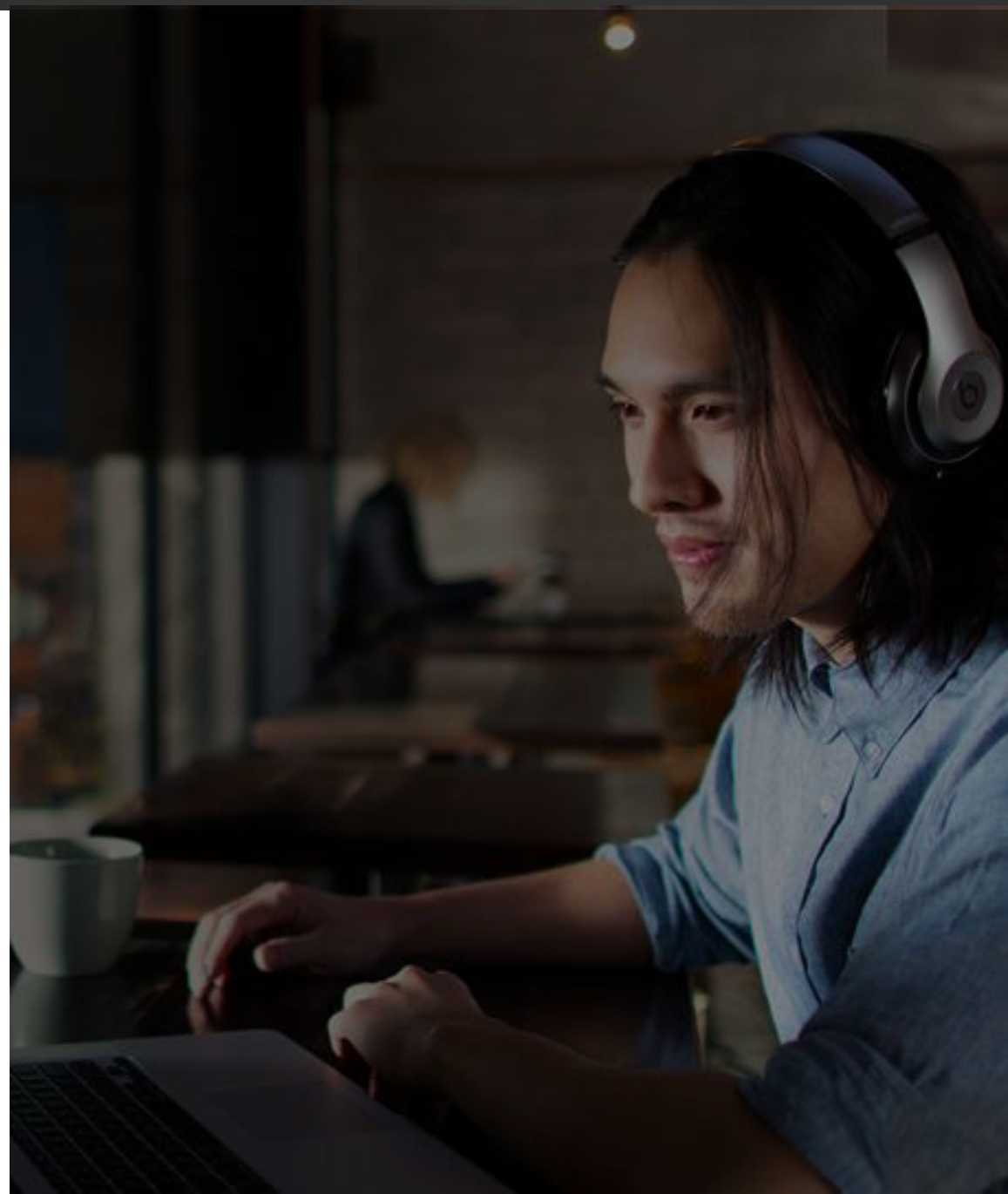
尾随闭包（一种写法）

使用时机

如果闭包表达式是函数的最后一个参数，那么可以使用尾随闭包的方式来增加代码的可读性。

简写方式

尾随闭包是将整个闭包表达式从函数的参数括号里移到括号外面的一种书写方式。



尾随闭包

使用时机

如果闭包表达式是函数的最后一个参数，那么可以使用尾随闭包的方式来增加代码的可读性。

简写方式

尾随闭包是将整个闭包表达式从函数的参数括号里移到括号外面的一种书写方式。

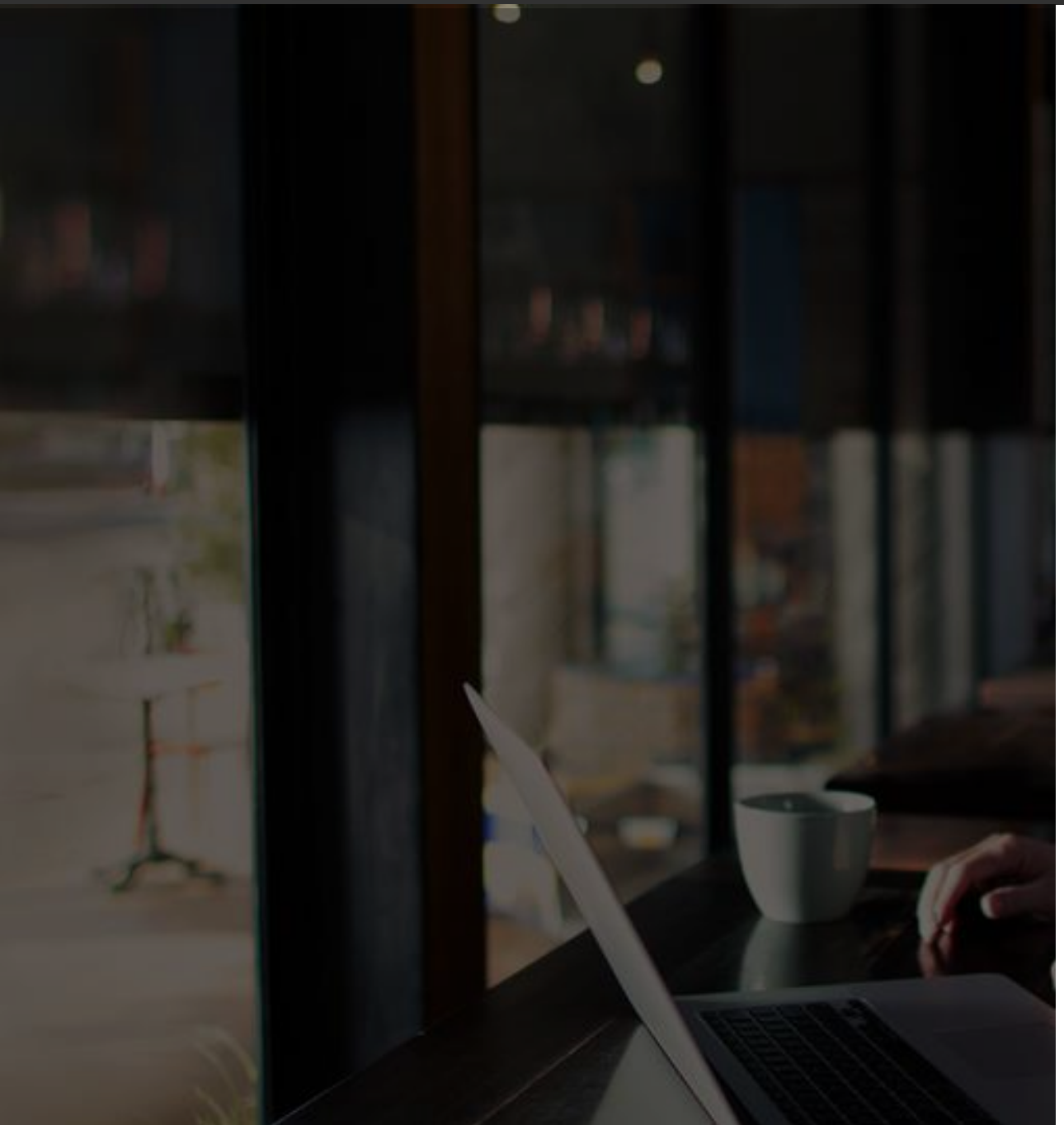
例子

```
func mathCompute(opr:String,n1:Int,n2:Int,
    compute:(Int,Int)->Int) {
    switch opr {
    case "+": print("\(n1)+\(n2) = \(compute
        (n1,n2))")
    case "-": print("\(n1)-\(n2) = \(compute
        (n1,n2))")
    case "*": print("\(n1)*\(n2) = \(compute
        (n1,n2))")
    case "/": print("\(n1)/\(n2) = \(compute
        (n1,n2))")
    default : print("not support this
        operator")
    }
}

mathCompute("+", n1: 9, n2: 3, compute: {(n1:
    Int,n2:Int)->Int in n1 + n2})
mathCompute("*", n1: 9, n2: 3)
    {(n1:Int,n2:Int)->Int in n1 * n2}
```

```
"9+3 = 12\n"
"9*3 = 27\n"
12
27
```

闭包表达式

尾随闭包



闭包的应用

捕获闭包作用域的变量

方法forEach

方法filter

方法map

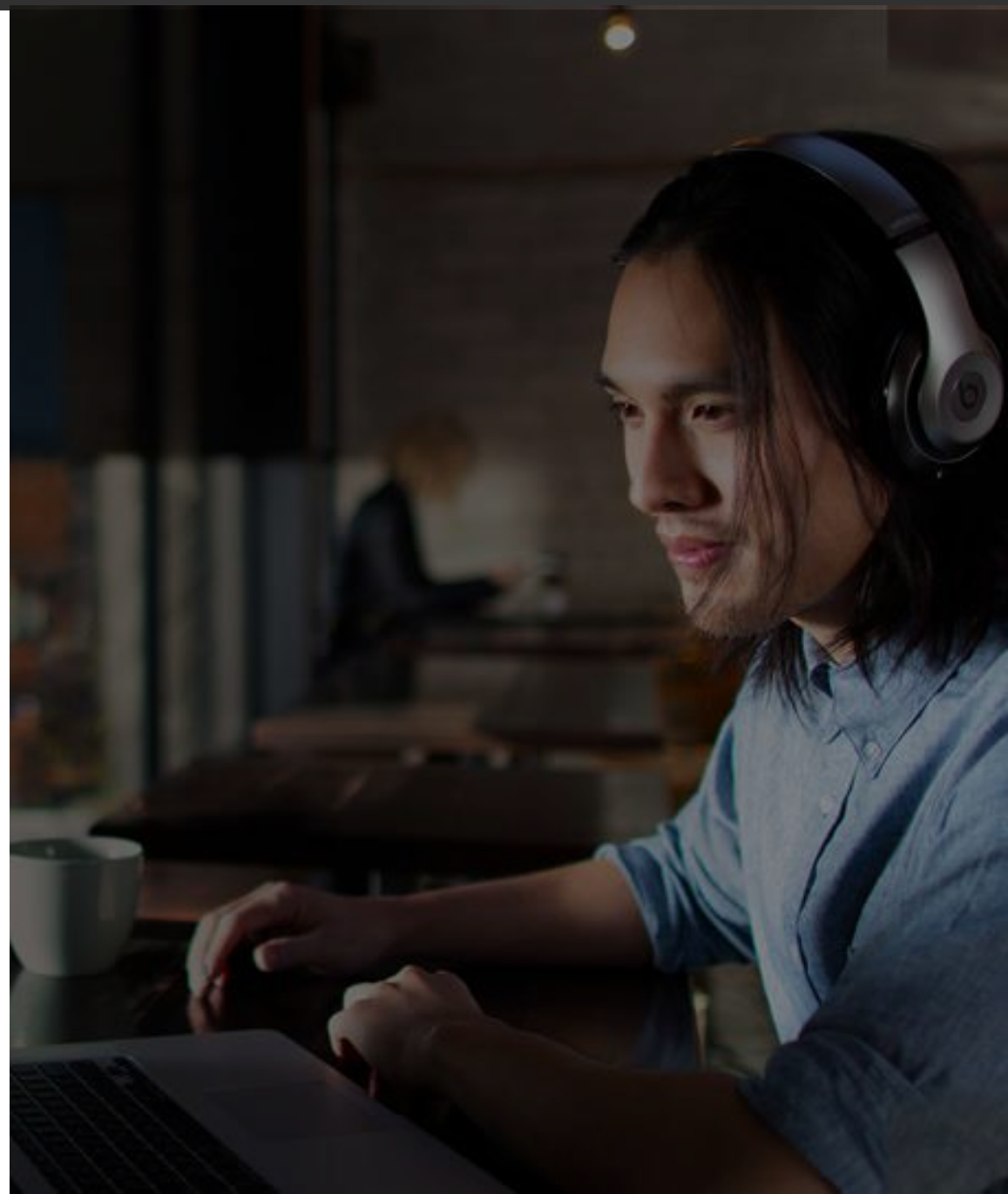
方法reduce

捕获闭包作用域的变量

利用闭包可以捕获其作用域中的任何变量。

不同闭包实例的作用域是相互独立的。

利用闭包还可以方便地对集合类型变量进行各种遍历操作，包括修改元素值、过滤出符合特定条件的元素等。（有兴趣自学）



捕获闭包作用域的变量

利用闭包可以捕获其作用域中的任何变量。

不同闭包实例的作用域是相互独立的。

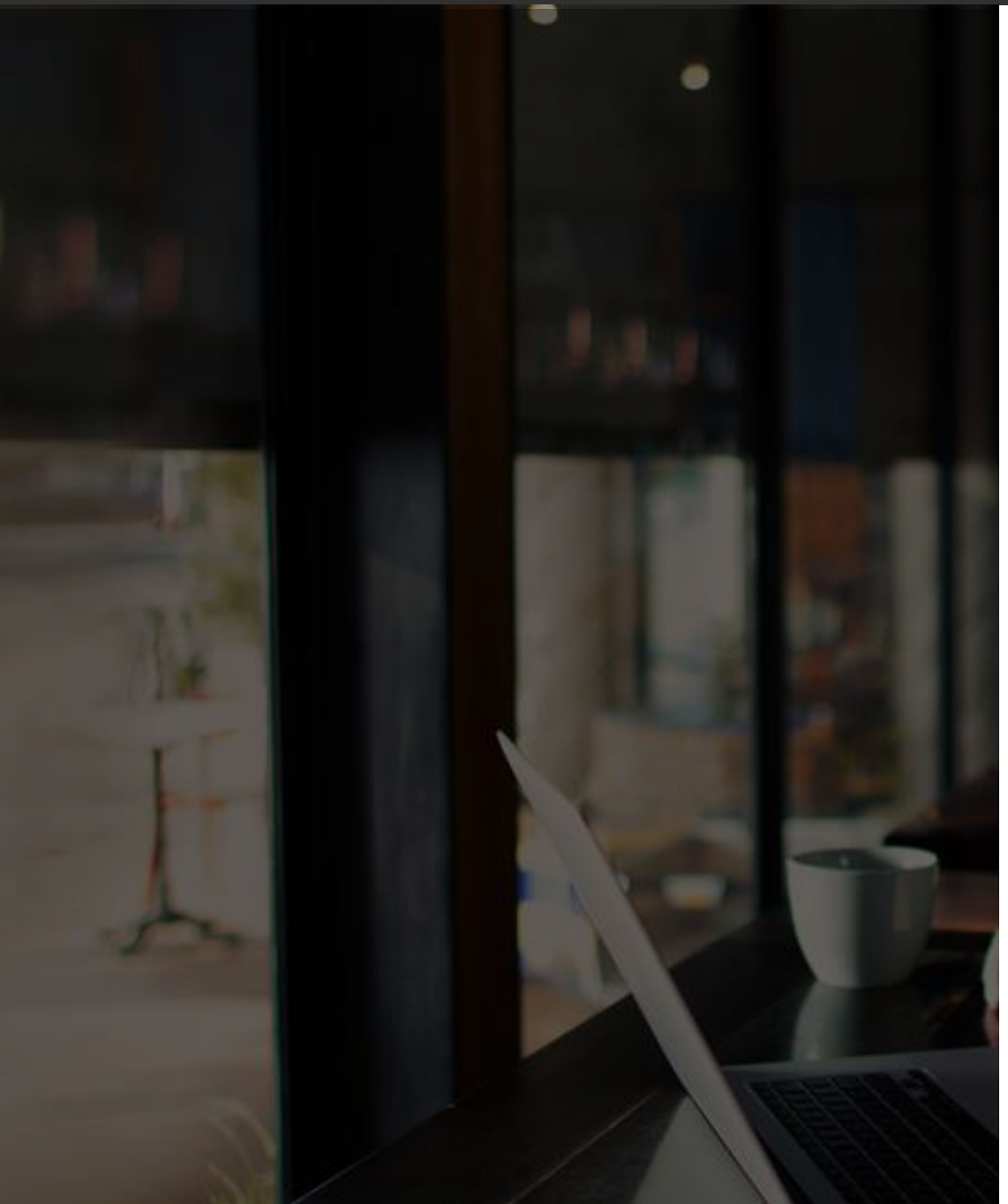
利用闭包还可以方便地对集合类型变量进行各种遍历操作，包括修改元素值、过滤出符合特定条件的元素等。（有兴趣自学）

```
var number = 100

let minusNumber = {
    number -= 1
}

minusNumber()
minusNumber()
minusNumber()
print(number)
```

```
100

() -> ()
(3 times)




"97\n"
```

```
func computeNumberClosure() -> ()->Int {
    var number = 100
    let minusNumber: ()->Int = {
        number -= 1
        return number
    }
    return minusNumber
}

let firstCountDown = computeNumberClosure()
let secondContDown = computeNumberClosure()

firstCountDown()
secondContDown()
secondContDown()
secondContDown()
firstCountDown()
secondContDown()
```

```
(2 times)
(2 times)
(6 times)
(6 times)

(2 times)



() -> Int
() -> Int

99
99
98
97
98
96
```



方法forEach

集合类型的forEach方法负责遍历集合中的每一个元素，而对元素的操作可以通过闭包来进行。

例子

```
22 var numbers = [1,2,3,4,5,6]                          [1, 2, 3, 4, 5, 6]
23 numbers.forEach{                                     [1, 2, 3, 4, 5, 6]
24     print("The Cube of \($0) is \($0*$0*$0)")         (6 times)
25 }
```

```
The Cube of 1 is 1
The Cube of 2 is 8
The Cube of 3 is 27
The Cube of 4 is 64
The Cube of 5 is 125
The Cube of 6 is 216
```

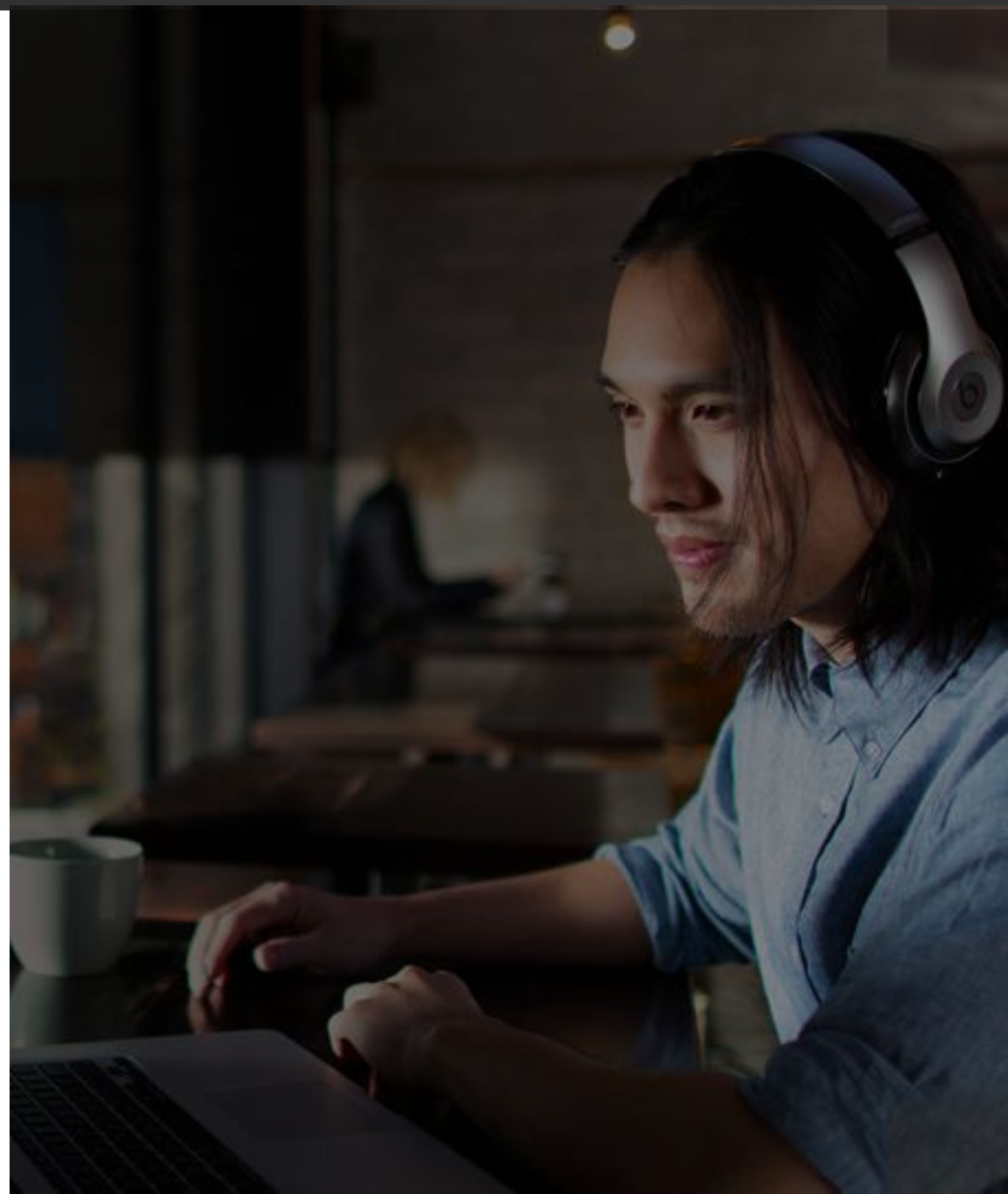
方法filter

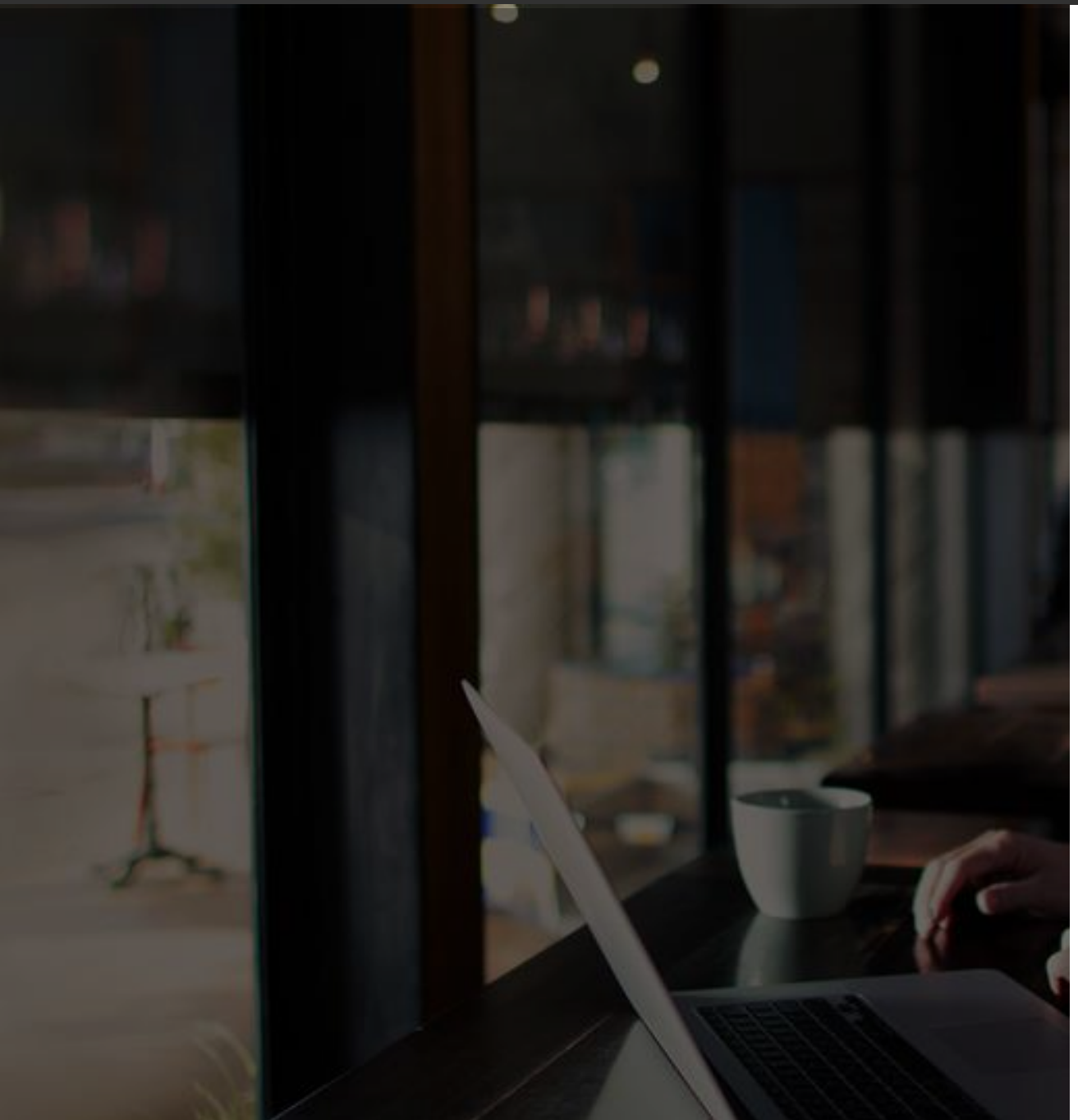
方法filter负责按照一定的条件将集合中的元素过滤出来，并形成一个新的集合，过滤的条件由闭包类型的参数提供。

例子

```
let filteredNumbers = numbers.filter{
    return $0 > 3
}
```

[4, 5, 6]
(6 times)





方法map

方法map负责遍历集合中的每一个元素，对其进行操作，并形成一个新的集合，对元素的操作由闭包类型的参数提供。

例子

```
let doubledNumbers = numbers.map{
    return $0 * 2
}
```

```
[2, 4, 6, 8, 10, 12]
(6 times)
```

方法reduce

两个参数

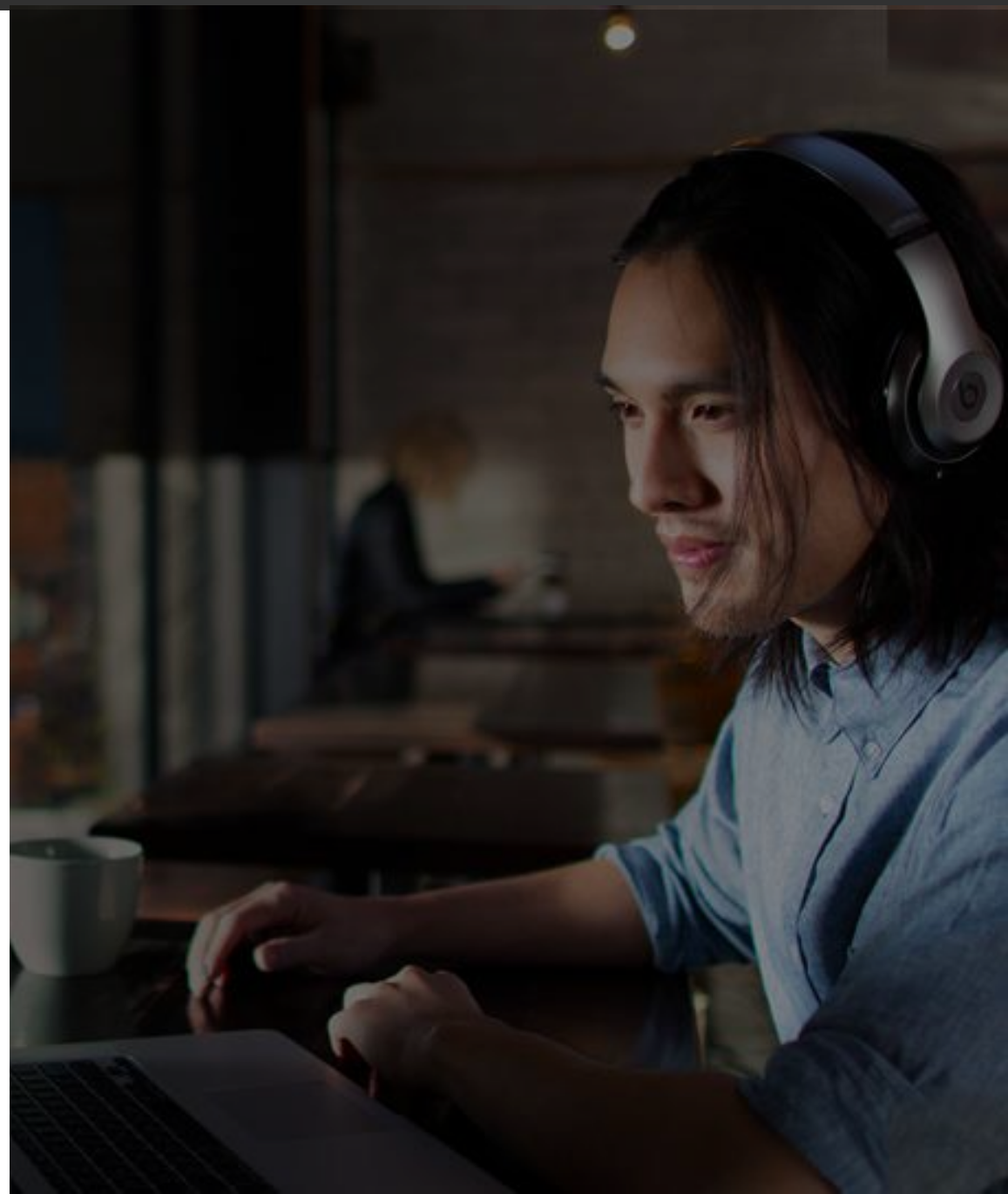
第一个参数为整型，作为当前值的起始值；

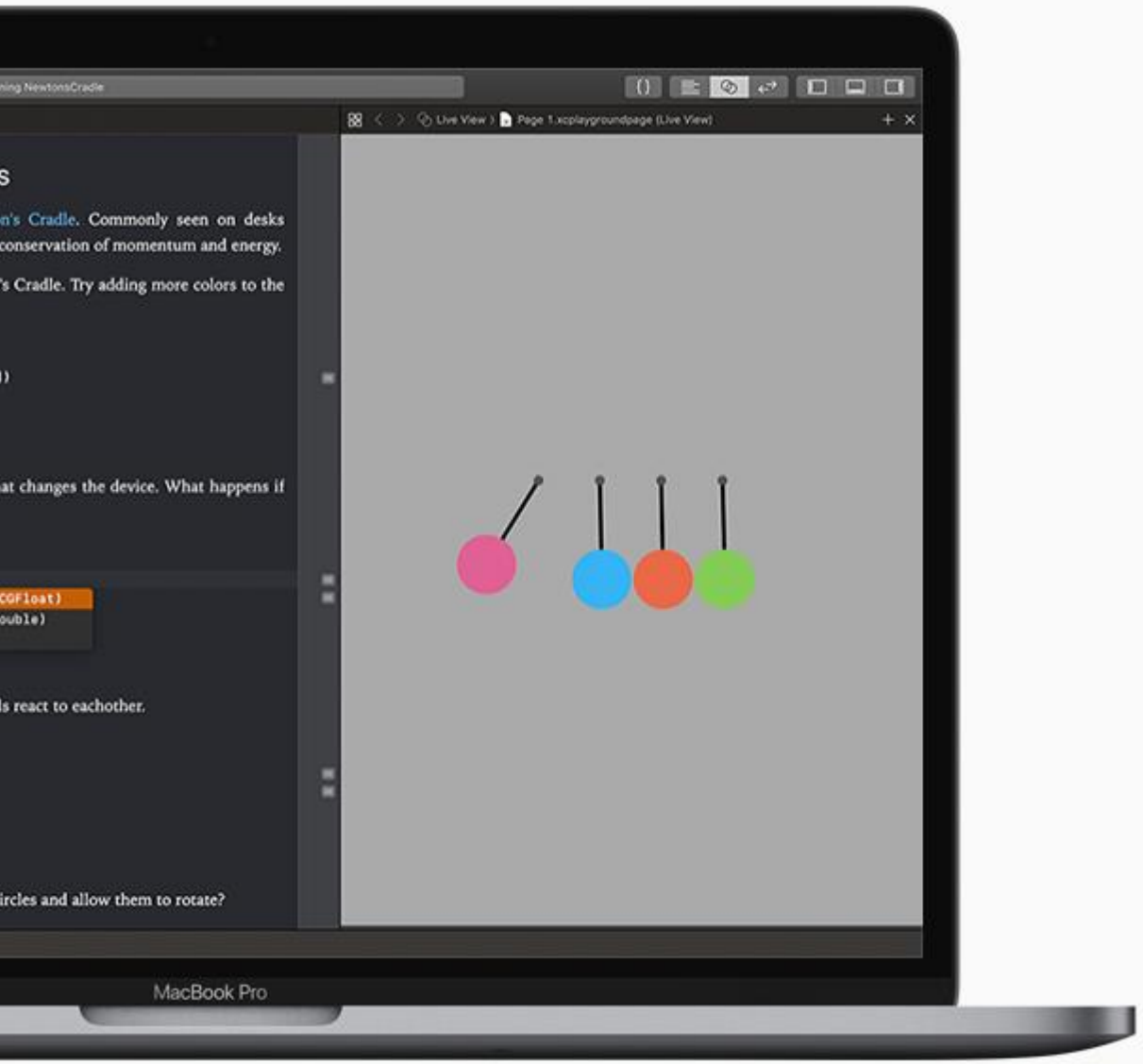
第二个参数为闭包类型。这个闭包有两个参数，一个为当前值，另一个为集合中的一个元素值。闭包的返回值作为传给自己的当前值参数。

例子

```
let sumOfNumbers = numbers.reduce(0){
    return $0 + $1
}
```

```
21
(6 times)
```





面向对象

枚举类型

定义、关联值及原始值的定义和用法

结构体和类

共同点和不同点，以及各自的应用场景

属性

存储属性、计算属性、属性观察器以及类型属性

方法

实例方法、类型方法、下标方法

继承性、基类、子类、重载

构造器和析构器

协议

泛型