Functions and Closures

Allen Holub

http://holub.com | Allen Holub | @allenholub

```
func sayHello (first: String, var last: String) -> String {

    return “Hello \(first) \(last)"
}


func noReturn() {…}
```

```
func sayHello (first: String, var last: String) -> String {

    return “Hello \(first) \(last)"
}


func noReturn() ->() {…}
```

```
func takesOptionals( optionalArg:Int? ) -> Int? {
    return nil
}

if let returnValue = takesOptionals(nil) {
    …
}
```

```
func printName( first:String, last:String){
    print("\(first) \(last)")
}

printName("Wile E.", last: "Coyote")
```

```
func printName( given first:String, family last:String){
    print("\(first) \(last)")
}

printName( given: "Wile E.", family: "Coyote")
```

```
func printName( _     first:String, _      last:String){
    print("\(first) \(last)")
}

printName(        "Wile E.",         "Coyote")
```

```
func myOverload(x:Int          )                 {…}
func myOverload(x:Int, y:Int ) ->(Int,Int) {…}
func myOverload(x:Double       ) -> Double   {…}
func myOverload(x:(Int,Int)   ) -> Double   {…}


func myOverload(a a:Int          ){}
func myOverload(b b:Int          ){}
func myOverload(c c:Int          ){}

myOverload( a:10 )
myOverload( b:10 )
myOverload( c:10 )
```

```
func findCenter(start:(y:Int,x:Int), _ end:(Int,Int))
                                        -> (y:Int,x:Int) {
    let (xEnd, yEnd) = end;
    let xCenter = start.x + (xEnd - start.y)/2
    let yCenter = start.y + (yEnd - start.x)/2
    return (xCenter, yCenter)
}


let center = findCenter( (1,1), (5,5) )
doSomethingWith( center.x, center.y )
```

```
func findCenter(start:(y:Int,x:Int), _ end:(Int,Int))
                                        -> (y:Int,x:Int)?  {
    let (xEnd, yEnd) = end;
    let xCenter = start.x + (xEnd - start.y)/2
    let yCenter = start.y + (yEnd - start.x)/2
    return (xCenter, yCenter)
}


if let center = findCenter( (1,1), (5,5) ) {
        doSomethingWith( center.x, center.y )
}
```

```
func findCenter(start:(y:Int,x:Int), _ end:(Int,Int))
                                        -> (y:Int,x:Int)?  {
    let (xEnd, yEnd) = end;
    let xCenter = start.x + (xEnd - start.y)/2
    let yCenter = start.y + (yEnd - start.x)/2
    return (xCenter, yCenter)
}


if let (x,y)  = findCenter( (1,1), (5,5) ) {
        doSomethingWith( x, y )
}
```

```
func concatenate( strings:[String], delimitedBy:String = ", " )
                                                    -> String {
    var result = ""
    for word in strings {
        result += (word + withSep)
    }
    return result
}

concatenate(["a","b","c"], delimitedBy:”\n" )
```

```
func concatenate( strings:[String], delimitedBy:String = ", " )
                                                    -> String {
    var result = ""
    for word in strings {
        result += (word + withSep)
    }
    return result
}

concatenate(["a","b","c"])
```

```swift
func sub( a:Int = 0, b:Int = 0) -> Int { return b - a }

sub()
sub(a:10)
sub(b:10)
sub(  20, b:10)
```

```
func sub( a  a:Int = 0, b:Int = 0) -> Int { return b - a }

sub()
sub(a:10)
sub(b:10)
sub(a:20, b:10)
```

```swift
func sub( a  a:Int = 0, b:Int = 0) -> Int { return b - a }

sub()
sub(a:10)
sub(b:10)
sub(a:20, b:10)
sub(b:10, a:20)
```

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


sum( 1, 2, 3, 4 )
```

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

```swift
sum( numbers: 2, 3, 4 )
```

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

```swift
sum( 1, numbers: 2, 3, 4 )
```

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



sum( 1, 2, 3, 4 )
```

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

```
var left  = "L"
var right = "R"

swap(&left, &right)

left    // == R
right   // == L
```

```
enum Status { case success, failure }



func doSomething( inout result: String ) -> Status {
    result = "new value"
    return .success;
}
```

```
enum Status { case success, failure }
```

```
func doSomething() -> (Status:status, result:String) {
    return (.success, "newValue");
}
```

```
func sayHello(){ print("hello") }
var  talk = sayHello
```

```
func sayHello(){ print("hello") }
var  talk : ()->() = sayHello
```

```
func sayHello(){ print("hello") }
var  talk : ()->() = sayHello
talk()


func speak( talk:()->() ){
    talk()
}
speak( sayHello )
```

```swift
func moveTowardsZero( value: Int ) ->  (Int)->(Int) {
    func up   (input: Int ) -> Int { return input + 1 }
    func down (input: Int ) -> Int { return input - 1 }
    return (value < 0 ? up : down)
}


var i = 3
let mover = moveTowardsZero(i)

for _ in 1...3 {
    i = mover(i) // 3, 2, 1
}
```

Closures

Functional

- Lightweight
- Powerful
- Easy to read
- Easy to maintain
- Less duplication

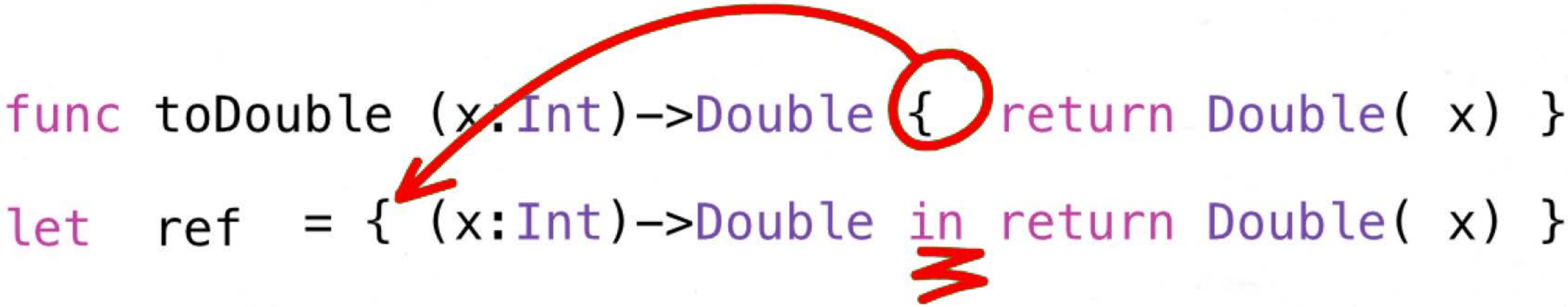
Object Oriented

- Heavyweight
- Wordy
- Domain modeling
- Message passing


1

```
func toDouble (x:Int)->Double {  return Double( x) }

let   ref  = toDouble
```

```
func toDouble (x:Int)->Double {  return Double( x) }

let  ref  = { (x:Int)->Double in return Double( x) }
```



Closure

Lambda Function

```
func toDouble (x:Int)->Double {  return Double( x) }

let   ref  :    (  Int)->Double
           = { (x:Int)->Double in return Double( x) }
```

```
func toDouble (x:Int)->Double {  return Double( x) }

let   ref  :    (  Int)->Double
           = {  x                in return Double( x) }
```

```
func toDouble (x:Int)->Double {  return Double( x) }

let   ref  :    (  Int)->Double
           = {                  return Double($0) }
```

```
func toDouble (x:Int)->Double {  return Double( x) }

let   ref  :    (  Int)->Double
           = {                                  Double($0) }
```

```
func sort( (String,String)->Bool )
```

```
func sort( (String,String)->Bool )

var names = [ "Fred", "Wilma", "Barney", "Betty" ]
names.sort(
    { (s1:String,s2:String)->Bool in return  s1 < s2 }
)
```

```
func sort( (String,String)->Bool )

var names = [ "Fred", "Wilma", "Barney", "Betty" ]
names.sort(
    {                                          $0 < $1 }
)
```

```
func sort( (String,String)->Bool )

var names = [ "Fred", "Wilma", "Barney", "Betty" ]
names.sort( ){ $0 < $1 }
```

trailing closure

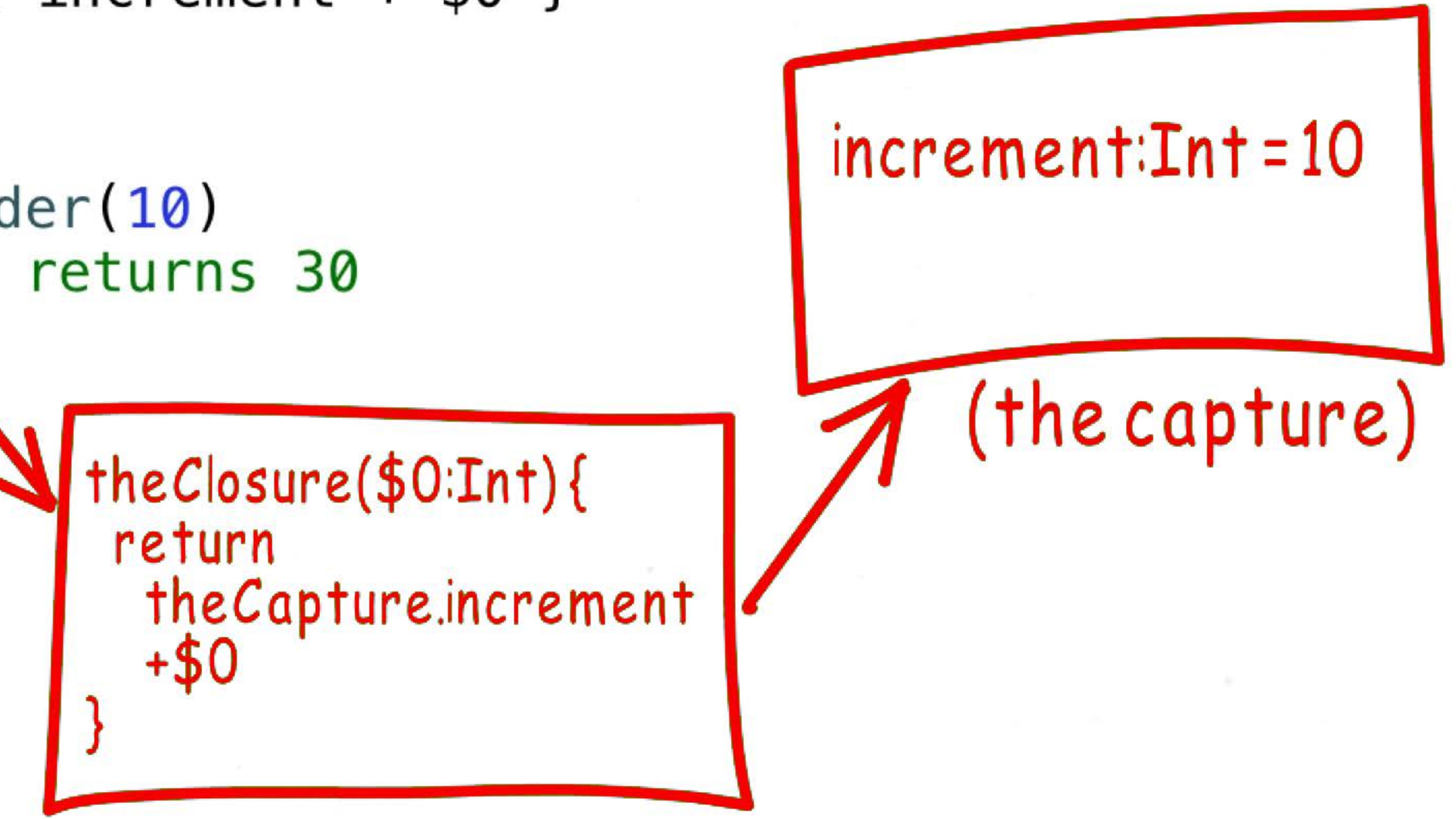
```
func sort( (String,String)->Bool )

var names = [ "Fred", "Wilma", "Barney", "Betty" ]
names.sort{ $0 < $1 }
```

```
func adder(increment: Int) -> (Int)->Int {
    return { increment + $0 }
}

var add = adder(10)
add(20)    // returns 30
```

same as:

```
func sum($0:Int)->Int{
  return increment+$0
}
return sum
```

```
func adder(increment: Int) -> (Int)->Int {
    return { increment + $0 }
}

var add = adder(10)
add(20)    // returns 30
```

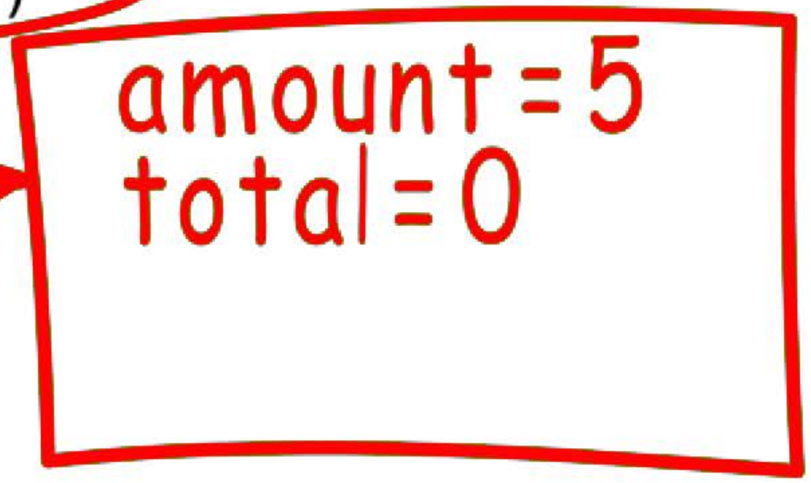


```
class Adder {
    var increment: Int;

    init(_ increment: Int ){
        self.increment = increment
    }

    func add(value: Int) -> Int {
        return increment + value
    }
}

var theAdder = Adder(10)
theAdder.add(20)  // returns 30
```

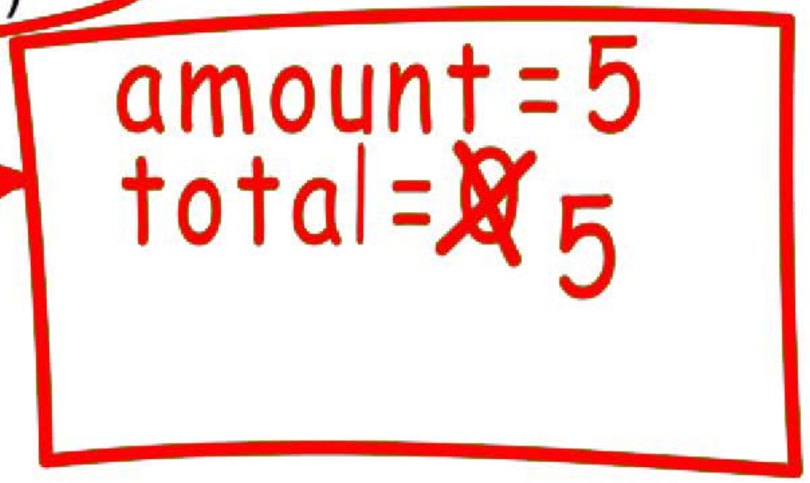
```
func getIncrementers(amount:Int) -> (increment:()->Int,
                                     decrement:()->Int) {
  var total = 0
  return ( {total += amount; return total},
           {total -= amount; return total} )
}

let (up5,down5) = getIncrementers(5)
up5()     // == 5
up5()     // == 10
down5()   // == 5
down5()   // == 0

let (up1,down1) = getIncrementers(1)
up5()    // 5
up1()    // 1
up5()    // 10
up1()    // 2
```

```
func getIncrementers(amount:Int) -> (increment:()->Int,
                                     decrement:()->Int) {
  var total = 0
  return ( {total += amount; return total},
           {total -= amount; return total} )
}

let (up5,down5) = getIncrementers(5)
up5()     // == 5
up5()     // == 10
down5()   // == 5
down5()   // == 0

let (up1,down1) = getIncrementers(1)
up5()    // 5
up1()    // 1
up5()    // 10
up1()    // 2
```

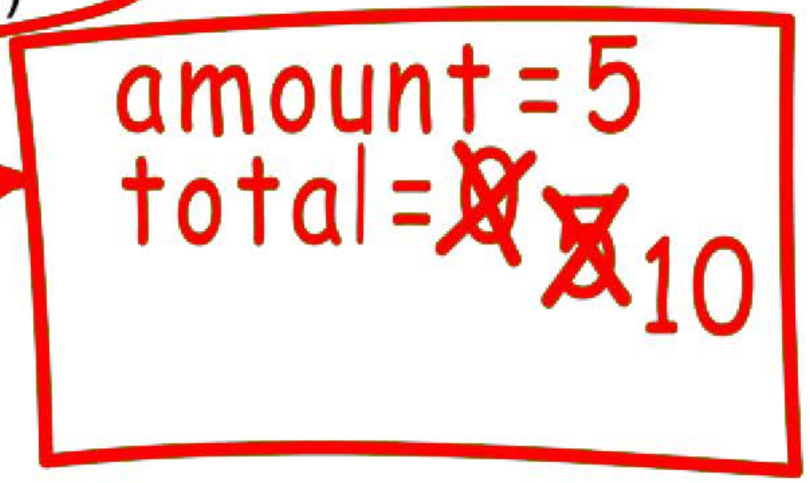
```
func getIncrementers(amount:Int) -> (increment:()->Int,
                                     decrement:()->Int) {
  var total = 0
  return ( {total += amount; return total},
           {total -= amount; return total} )
}

let (up5,down5) = getIncrementers(5)
up5()     // == 5
up5()     // == 10
down5()   // == 5
down5()   // == 0

let (up1,down1) = getIncrementers(1)
up5()    // 5
up1()    // 1
up5()    // 10
up1()    // 2
```



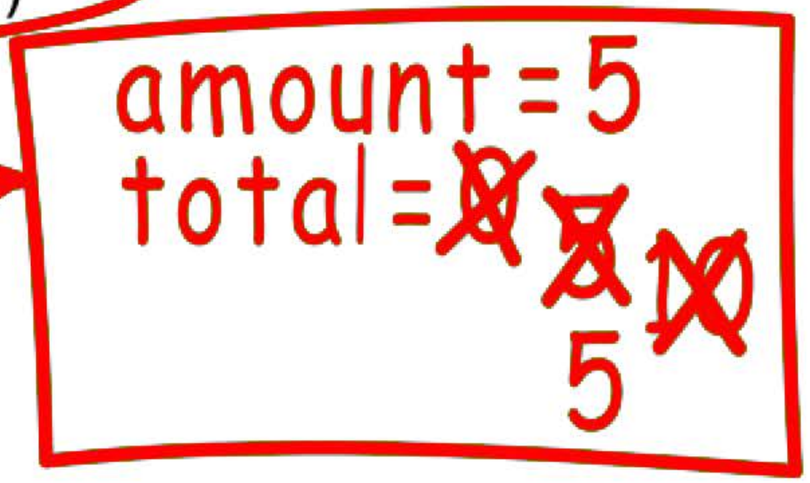
```
func getIncrementers(amount:Int) -> (increment:()->Int,
                                     decrement:()->Int) {
  var total = 0
  return ( {total += amount; return total},
           {total -= amount; return total} )
}

let (up5,down5) = getIncrementers(5)
up5()     // == 5
up5()     // == 10
down5()   // == 5
down5()   // == 0

let (up1,down1) = getIncrementers(1)
up5()    // 5
up1()    // 1
up5()    // 10
up1()    // 2
```



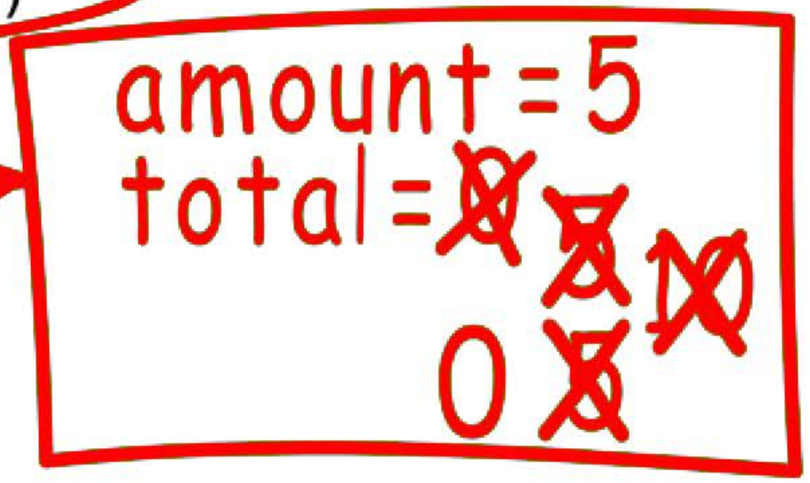
```
func getIncrementers(amount:Int) -> (increment:()->Int,
                                     decrement:()->Int) {
  var total = 0
  return ( {total += amount; return total},
           {total -= amount; return total} )
}

let (up5,down5) = getIncrementers(5)
up5()     // == 5
up5()     // == 10
down5()   // == 5
down5()   // == 0

let (up1,down1) = getIncrementers(1)
up5()    // 5
up1()    // 1
up5()    // 10
up1()    // 2
```



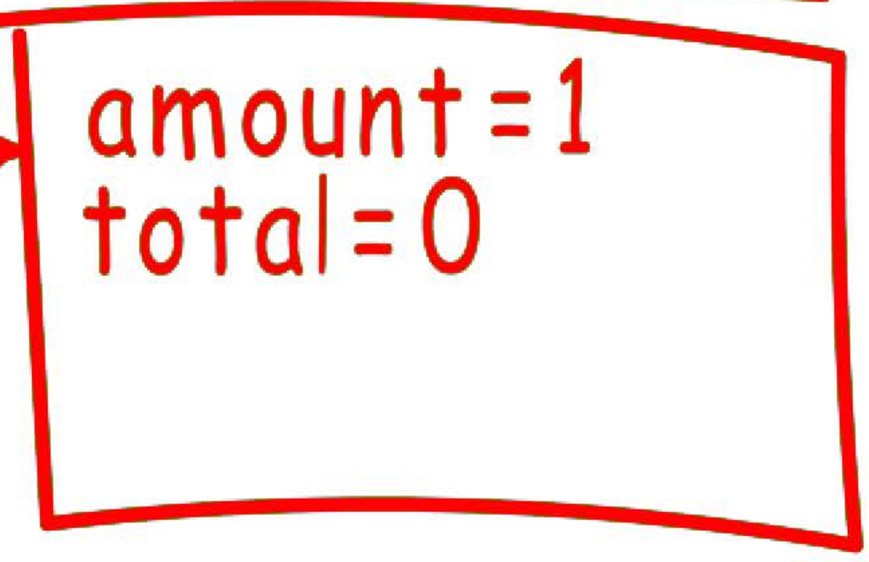
```
func getIncrementers(amount:Int) -> (increment:()->Int,
                                     decrement:()->Int) {
  var total = 0
  return ( {total += amount; return total},
           {total -= amount; return total} )
}

let (up5,down5) = getIncrementers(5)
up5()     // == 5
up5()     // == 10
down5()   // == 5
down5()   // == 0

let (up1,down1) = getIncrementers(1)
up5()    // 5
up1()    // 1
up5()    // 10
up1()    // 2
```



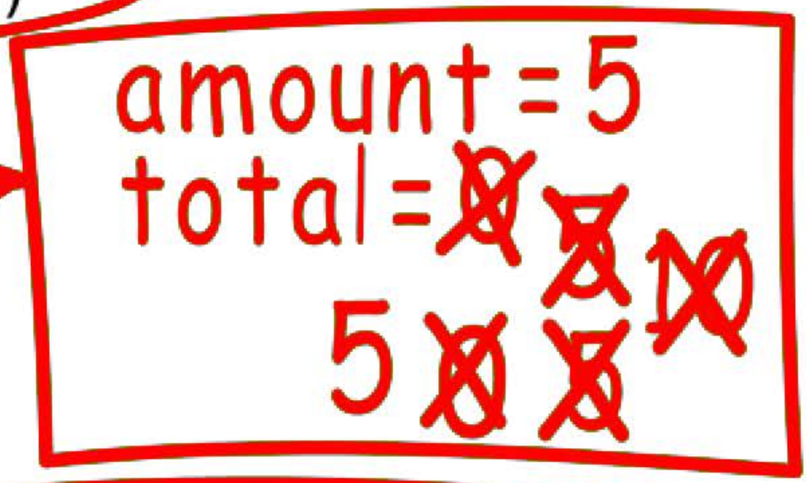
```
func getIncrementers(amount:Int) -> (increment:()->Int,
                                     decrement:()->Int) {
  var total = 0
  return ( {total += amount; return total},
           {total -= amount; return total} )
}

let (up5,down5) = getIncrementers(5)
up5()     // == 5
up5()     // == 10
down5()   // == 5
down5()   // == 0

let (up1,down1) = getIncrementers(1)
up5()    // 5
up1()    // 1
up5()    // 10
up1()    // 2
```

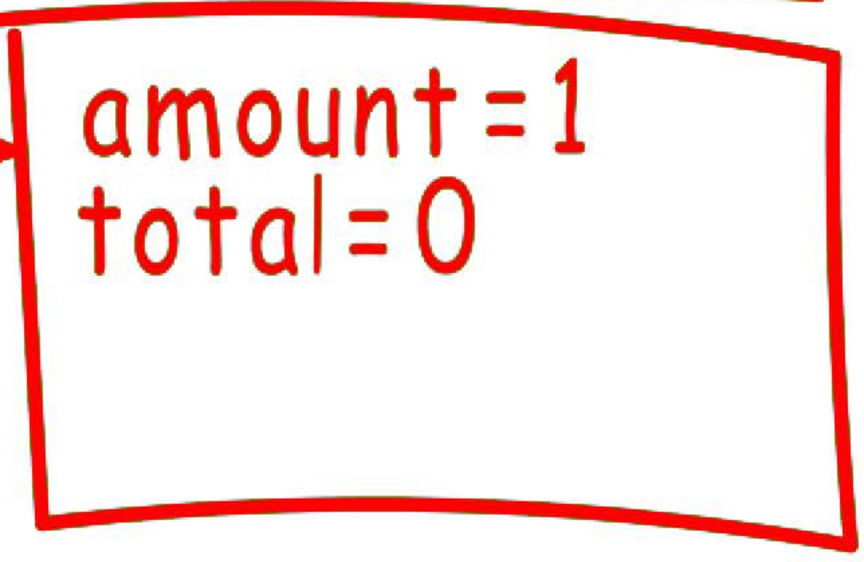


```
func getIncrementers(amount:Int) -> (increment:()->Int,
                                     decrement:()->Int) {
  var total = 0
  return ( {total += amount; return total},
           {total -= amount; return total} )
}

let (up5,down5) = getIncrementers(5)
up5()     // == 5
up5()     // == 10
down5()   // == 5
down5()   // == 0

let (up1,down1) = getIncrementers(1)
up5()    // 5
up1()    // 1
up5()    // 10
up1()    // 2
```

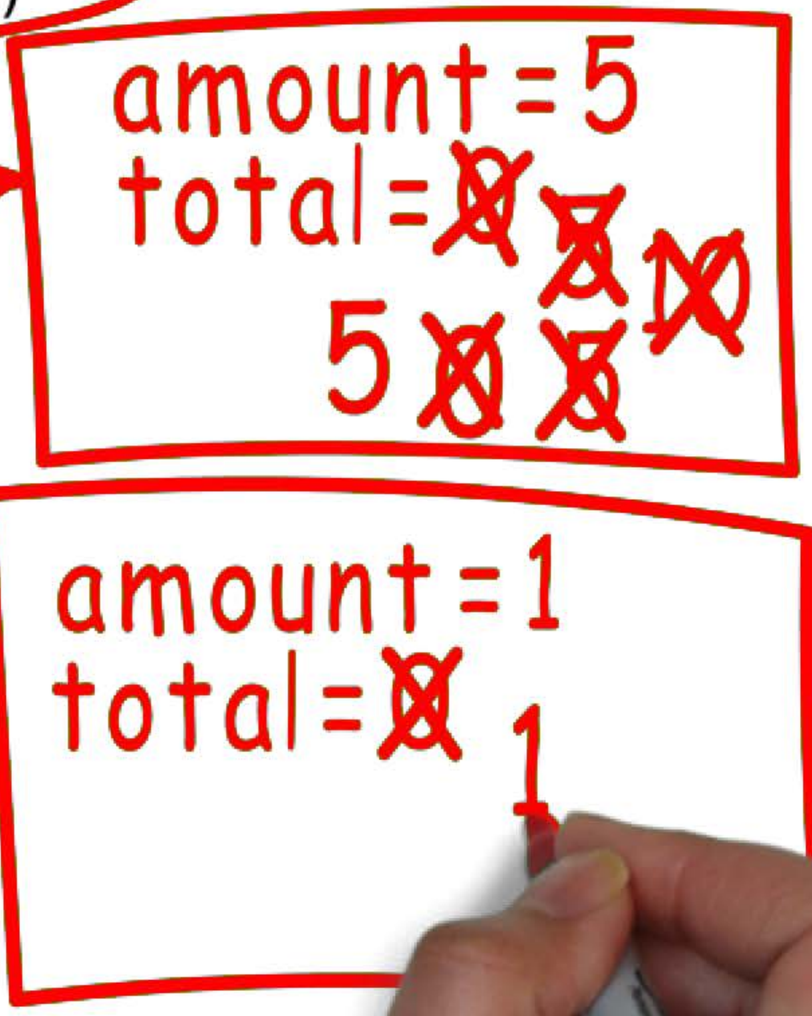


```
func getIncrementers(amount:Int) -> (increment:()->Int,
                                     decrement:()->Int) {
  var total = 0
  return ( {total += amount; return total},
           {total -= amount; return total} )
}

let (up5,down5) = getIncrementers(5)
up5()     // == 5
up5()     // == 10
down5()   // == 5
down5()   // == 0

let (up1,down1) = getIncrementers(1)
up5()    // 5
up1()    // 1
up5()    // 10
up1()    // 2
```

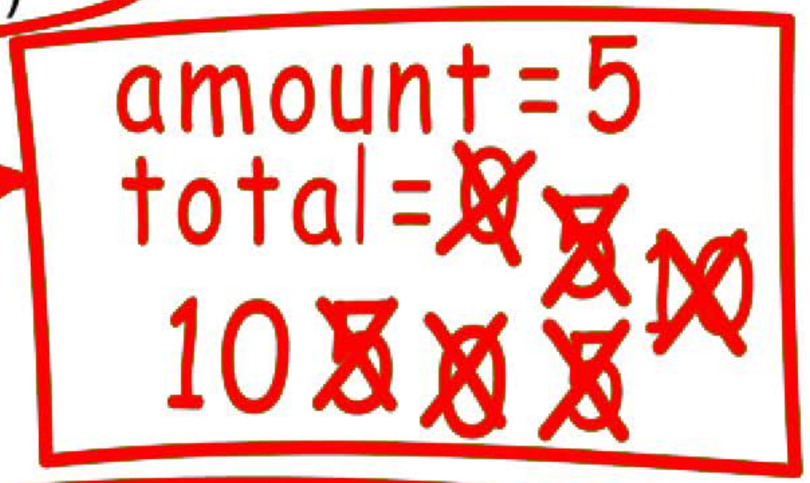


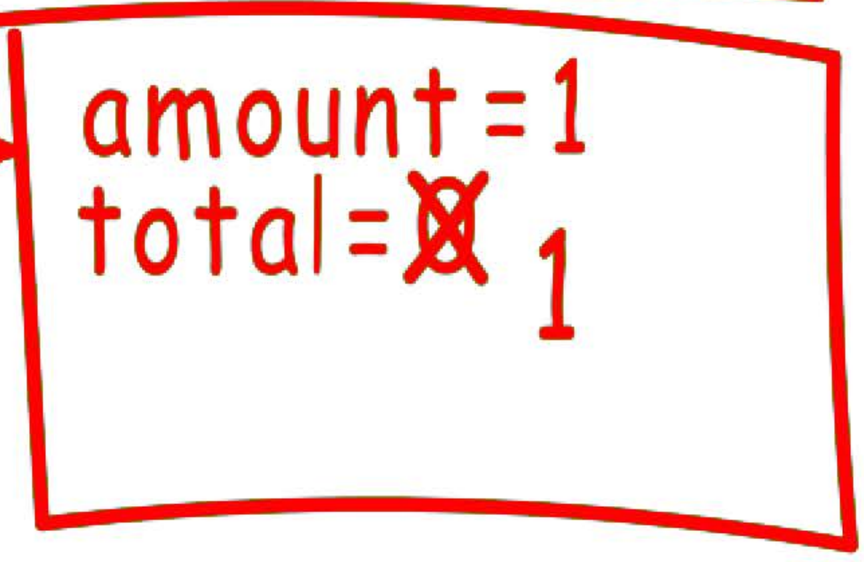


```
func getIncrementers(amount:Int) -> (increment:()->Int,
                                     decrement:()->Int) {
  var total = 0
  return ( {total += amount; return total},
           {total -= amount; return total} )
}

let (up5,down5) = getIncrementers(5)
up5()     // == 5
up5()     // == 10
down5()   // == 5
down5()   // == 0

let (up1,down1) = getIncrementers(1)
up5()    // 5
up1()    // 1
up5()    // 10
up1()    // 2
```

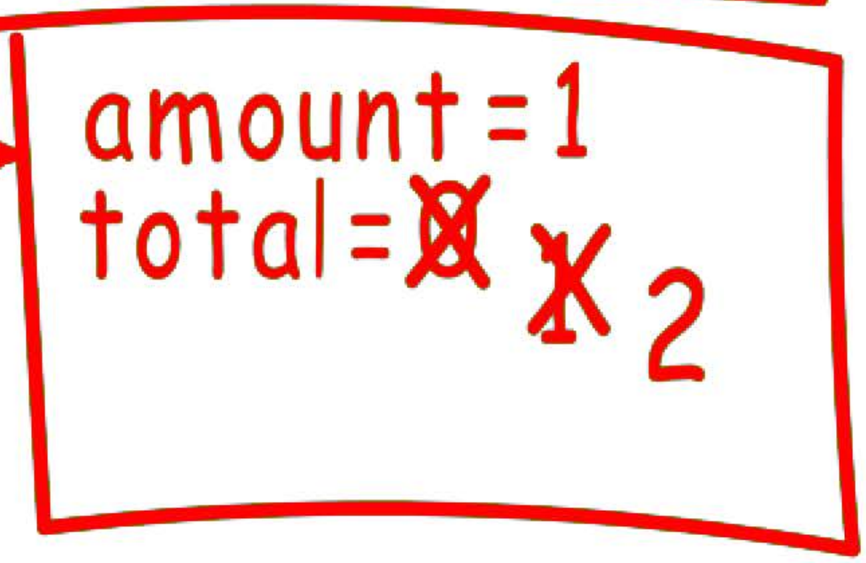


```
func getIncrementers(amount:Int) -> (increment:()->Int,
                                     decrement:()->Int) {
  var total = 0
  return ( {total += amount; return total},
           {total -= amount; return total} )
}

let (up5,down5) = getIncrementers(5)
up5()     // == 5
up5()     // == 10
down5()   // == 5
down5()   // == 0

let (up1,down1) = getIncrementers(1)
up5()    // 5
up1()    // 1
up5()    // 10
up1()    // 2
```



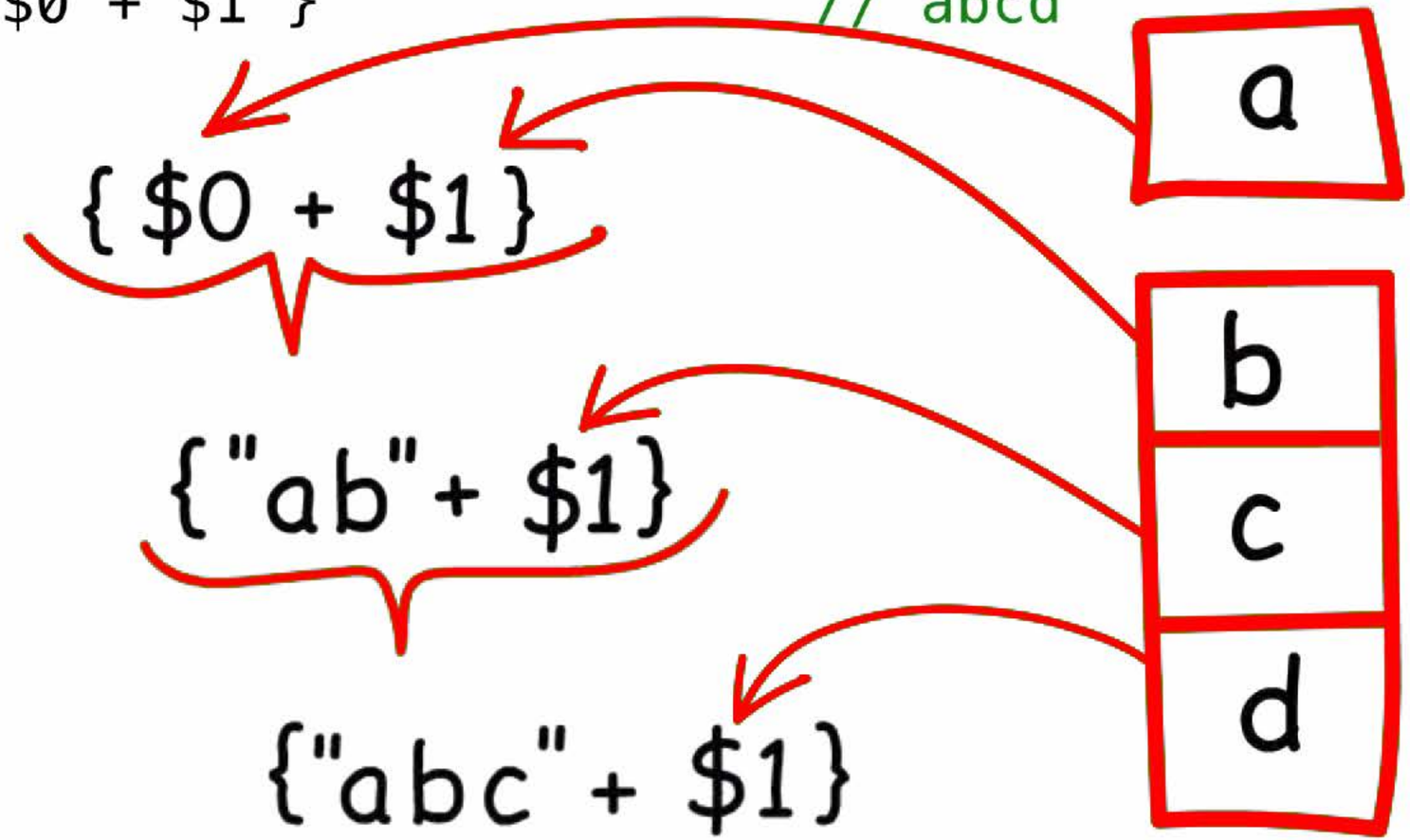


```
func getIncrementers(amount:Int) -> (increment:()->Int,
                                     decrement:()->Int) {
  var total = 0
  return ( {total += amount; return total},
           {total -= amount; return total} )
}

let (up5,down5) = getIncrementers(5)
up5()     // == 5
up5()     // == 10
down5()   // == 5
down5()   // == 0

let (up1,down1) = getIncrementers(1)
up5()    // 5
up1()    // 1
up5()    // 10
up1()    // 2
```



```
func getValueFromConfigFile
                    (key:String) -> String {…}


class SomeClass {

    let configurationItem: String = {
          getValueFromConfigFile("configKey")
     }()

}
```

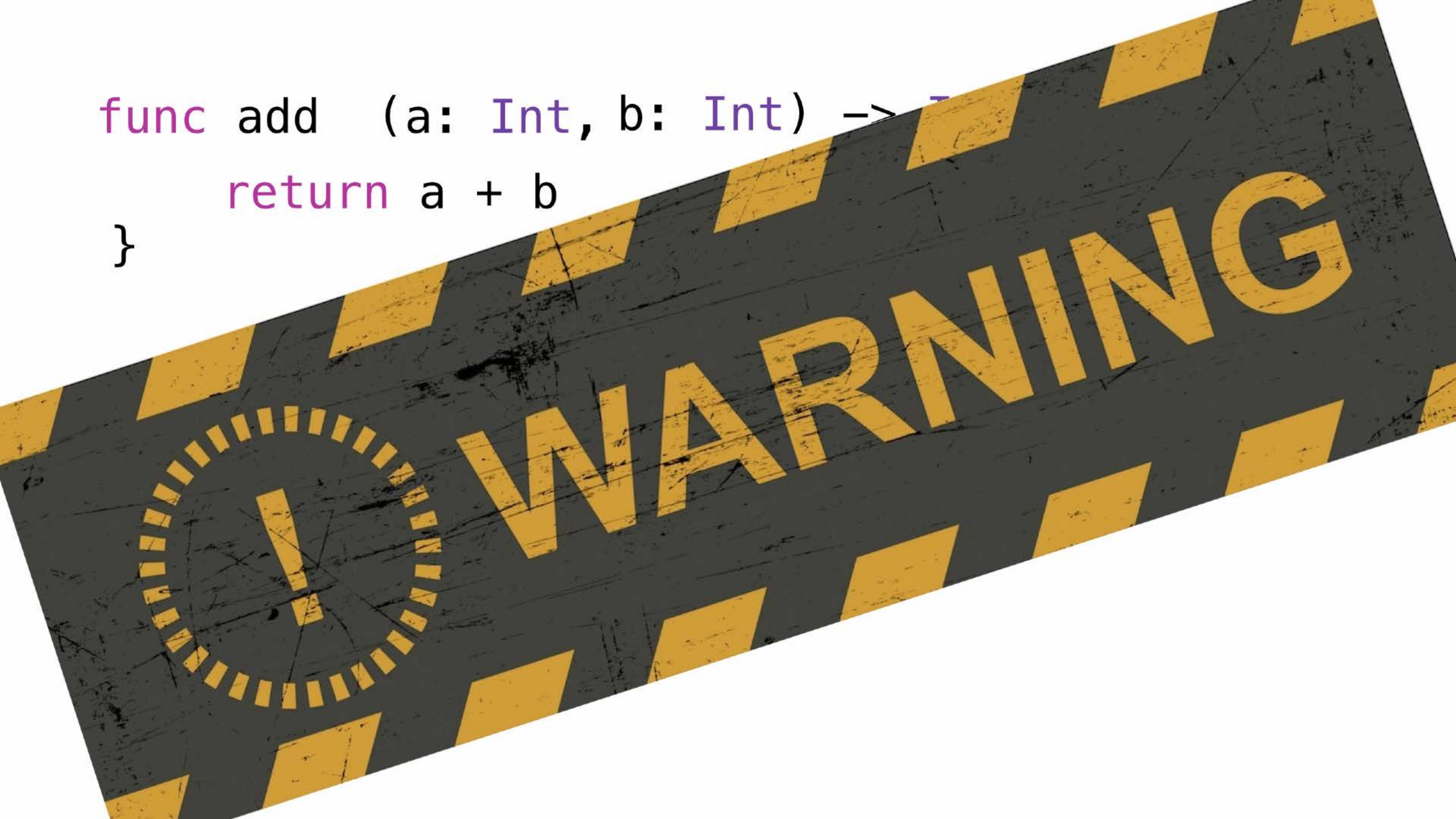
```
let letters = ["b", "c", "d"]

letters.map          { $0.uppercaseString }   // ["B", "C", "D"]

letters.filter       { $0 <= "c" }            // ["b", "c"]

letters.reduce("a"){ $0 + $1 }                // abcd
```



"Fred, Wilma"

```
var names = ["Fred Flintstone", "Wilma Flintstone",
             "Barney Rubble",   "Betty Rubble"]
let result =
names .filter    { $0.hasSuffix(" Flintstone") }
      .map       { $0.substringToIndex(advance($0.endIndex,-11))}
      .reduce(""){ let sep = $0.characters.count > 0 ? ", ":""
                   return "\($0)\(sep)\($1)"
                 }
```

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



```
func add  (a: I                    {
    return a +
}
```

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

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

```
add(1, b:2) == 3
```

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

func add2 (a: Int) -> ( b: Int) -> Int {
     return { ( b: Int)->Int in return a + b }
}
```

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

func add2 (a: Int) -> ( b: Int) -> Int {
     return { ( b: Int)->Int in return a + b }
}
```

add2(1)(b:2) == 3

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

func add2 (a: Int)    ( b: Int) -> Int {
     return {  b              in          a + b }
}
```

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

func add2 (a: Int)    ( b: Int) -> Int {
    return a + b
}
```

```
func appender( delimiter: String    ) ->
                          ( String ) -> String {

    var buffer = ""
    return {  buffer += $0 + delimiter
              return buffer
           }
}


 let append = appender(" ")
 append("Hello")                    // "Hello"
 append("world")                    // "Hello world"
 append("!")                        // "Hello world !"
 print ( append("") )
```

```
func appender( delimiter: String , var buffer: String ="")
                            ( String ) -> String {


                buffer += $0 + delimiter
                return buffer

 }


 let append = appender(" ")
 append("Hello")                    // "Hello"
 append("world")                    // "Hello world"
 append("!”)                        // "Hello world !"
 print ( append("") )
```

```
func appender( delimiter: String , var buffer: String ="")
               ( suffix:    String ) -> String {


    buffer += suffix  + delimiter
    return buffer

}


 let append = appender(" ")
 append("Hello")                    // "Hello"
 append("world")                    // "Hello world"
 append("!")                        // "Hello world !"
 print ( append("") )
```

functions == named closures

code

functions == named closures

encapsulate a small bit of code, so you can reuse it