Swift Meetup Welcome!

Oct 7, 2014
Organizer John Cao @jobacao

Agenda

- John - OO to Functional Swift
- Darren - Optionals 101
- Darren - Error Handling with Enums
- Dinner - Albert Centre Market - 270 Queen St

Functional Programming

- Mathematical function that **takes in arguments and outputs a value**, e.g. F(A) = X

- **Guarantees** a function call with the same input will produce the same output every time… 1,10,100,1000 times

5 Benefits

- **Safer** Code
- **Parallel** Programming
- More **Concise** Code
- Easily Model **Math Problems**
- Faster **Team** Development

3 Issues

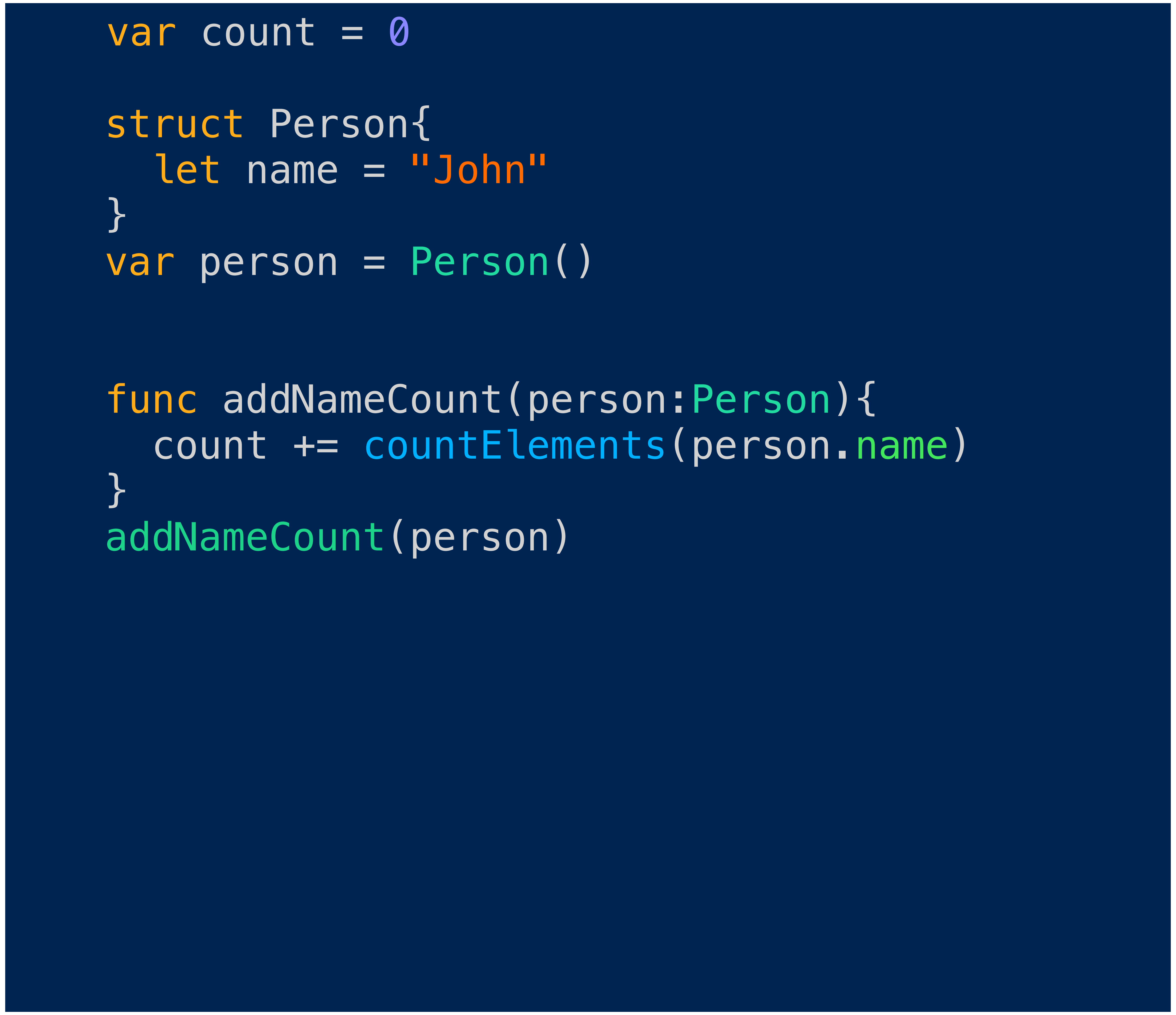
- Unknown Input - **User Input** (Web Apps)
- Performance - **New Objects** vs Update
- State Programming - **Devices** (Mouse,Cameras)

5 Properties of Functional Swift

#1 Read Inputs, Write Outputs

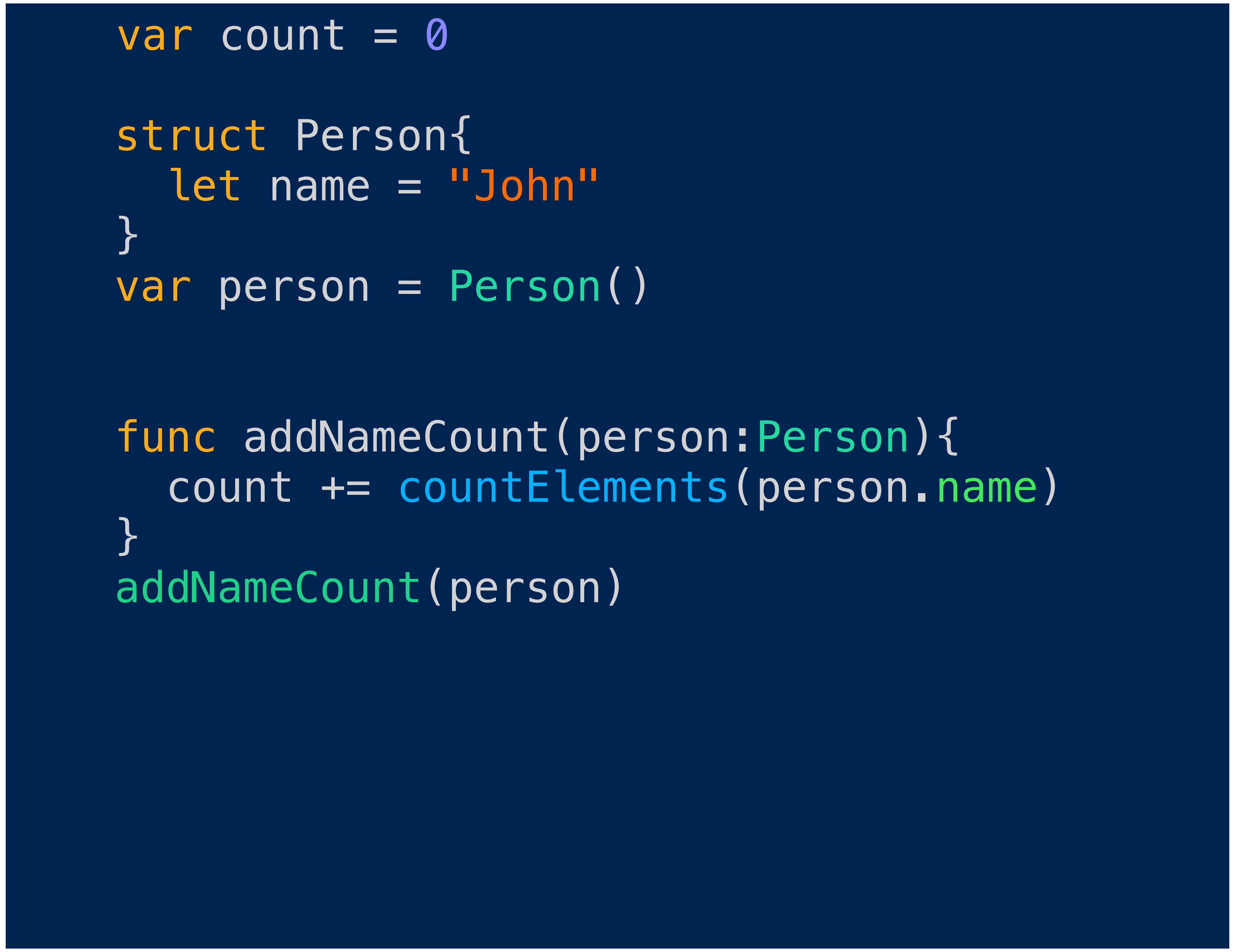
- Don’t access state
- Don’t modify inputs (arguments)
- Guarantee is maintained

#1 Read Inputs, Write Outputs

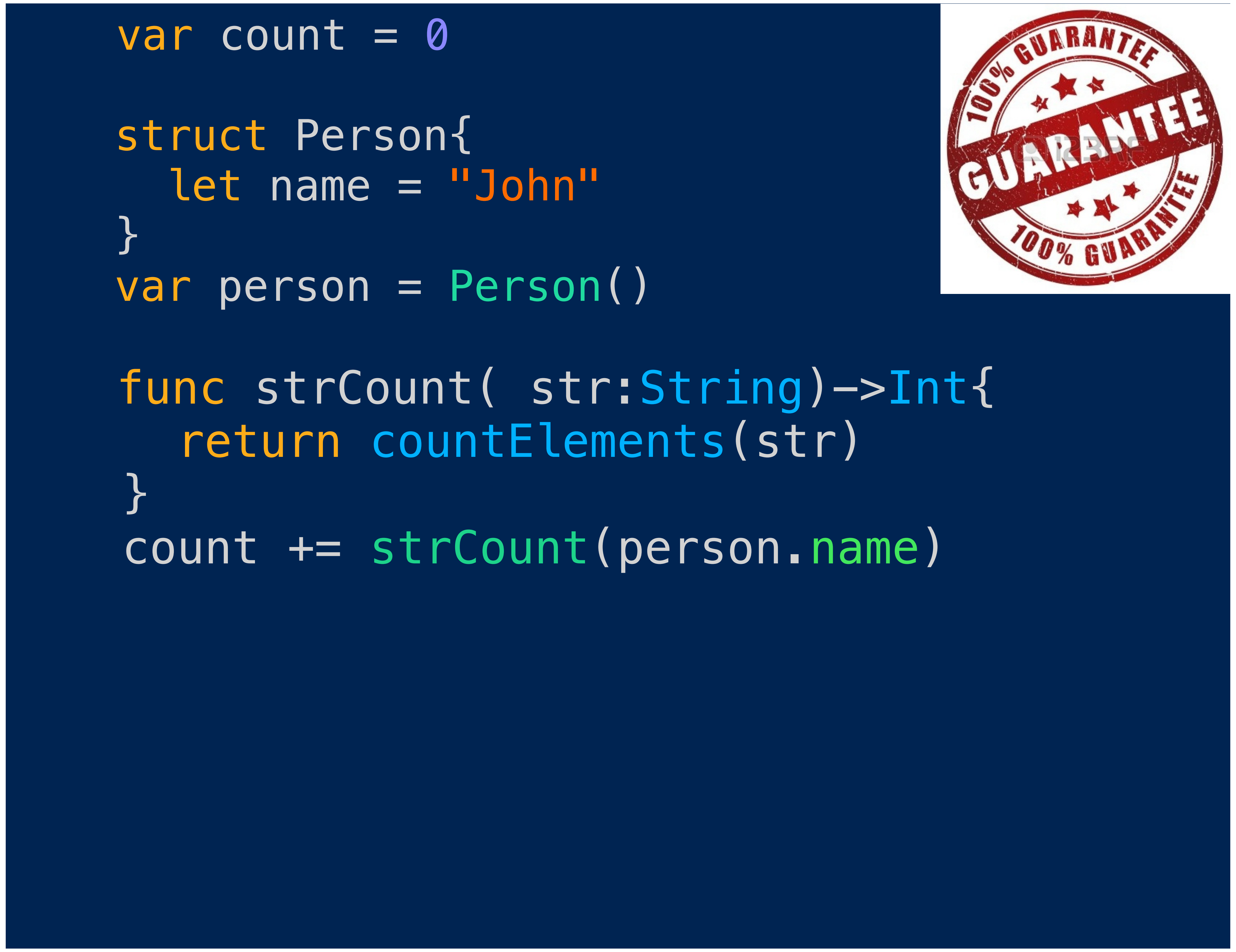


```
var count = 0

struct Person{
  let name = "John"
}
var person = Person()


func addNameCount(person:Person){
  count += countElements(person.name)
}
addNameCount(person)
```

#1 Read Inputs, Write Outputs



```
var count = 0

struct Person{
  let name = "John"
}
var person = Person()


func addNameCount(person:Person){
  count += countElements(person.name)
}
addNameCount(person)
```



```
var count = 0

struct Person{
  let name = "John"
}
var person = Person()

func strCount( str:String)->Int{
   return countElements(str)
 }
 count += strCount(person.name)
```

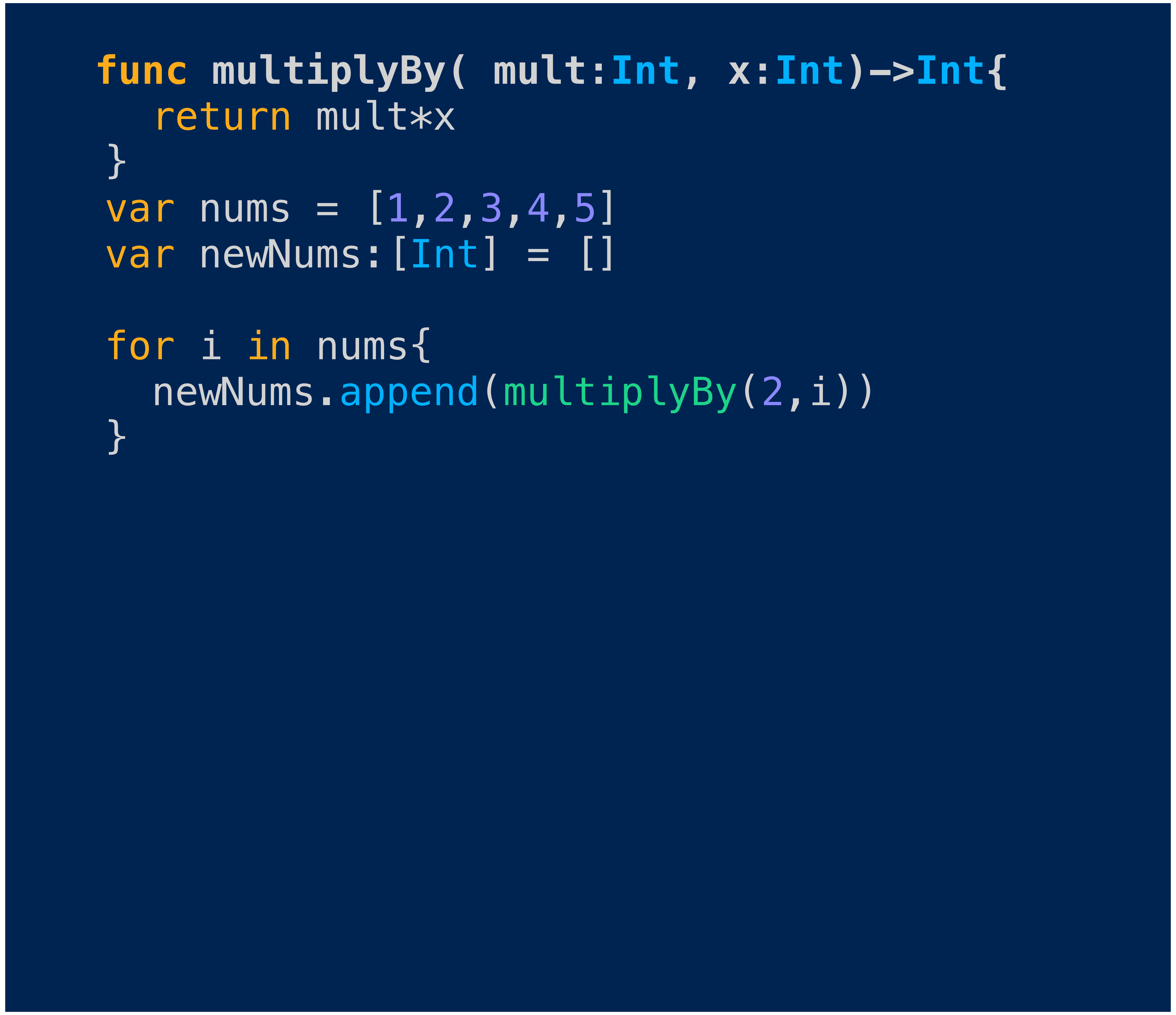
#2 No Control Loops

- More recursion, No For, While Loops
- More concise and succinct
- Less documentation

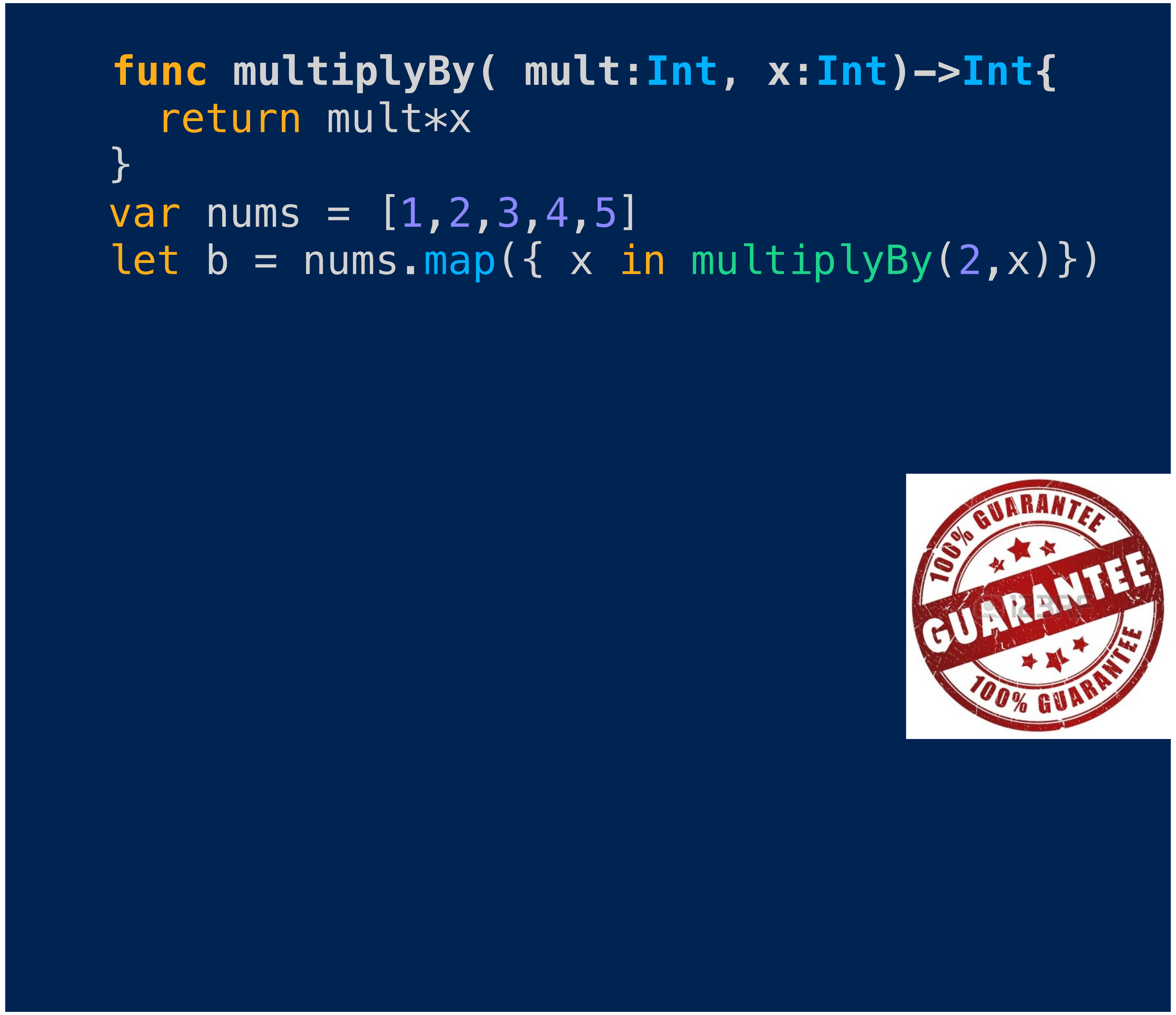
#2 No Control Loops

```swift
func multiplyBy( mult:Int, x:Int)->Int{
  return mult*x
}
var nums = [1,2,3,4,5]
var newNums:[Int] = []

for i in nums{
  newNums.append(multiplyBy(2,i))
}
```

#2 No Control Loops



```
func multiplyBy( mult:Int, x:Int)->Int{
  return mult*x
}
var nums = [1,2,3,4,5]
var newNums:[Int] = []

for i in nums{
  newNums.append(multiplyBy(2,i))
}
```



```
func multiplyBy( mult:Int, x:Int)->Int{
  return mult*x
}
var nums = [1,2,3,4,5]
let b = nums.map({ x in multiplyBy(2,x)})
```

100% GUARANTEE

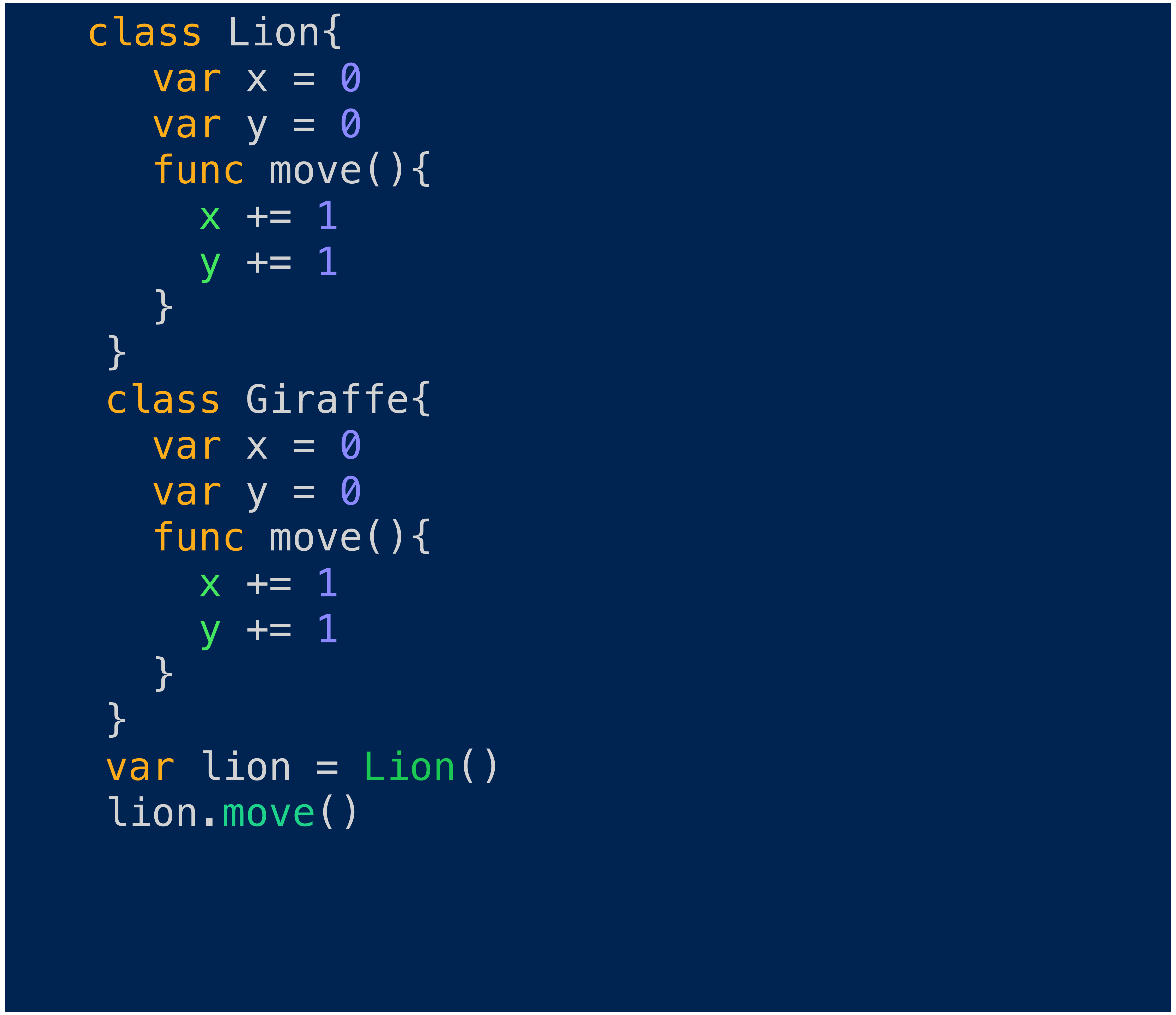
GUARANTEE

100% GUARANTEE

#3 More Functions,Less Objects

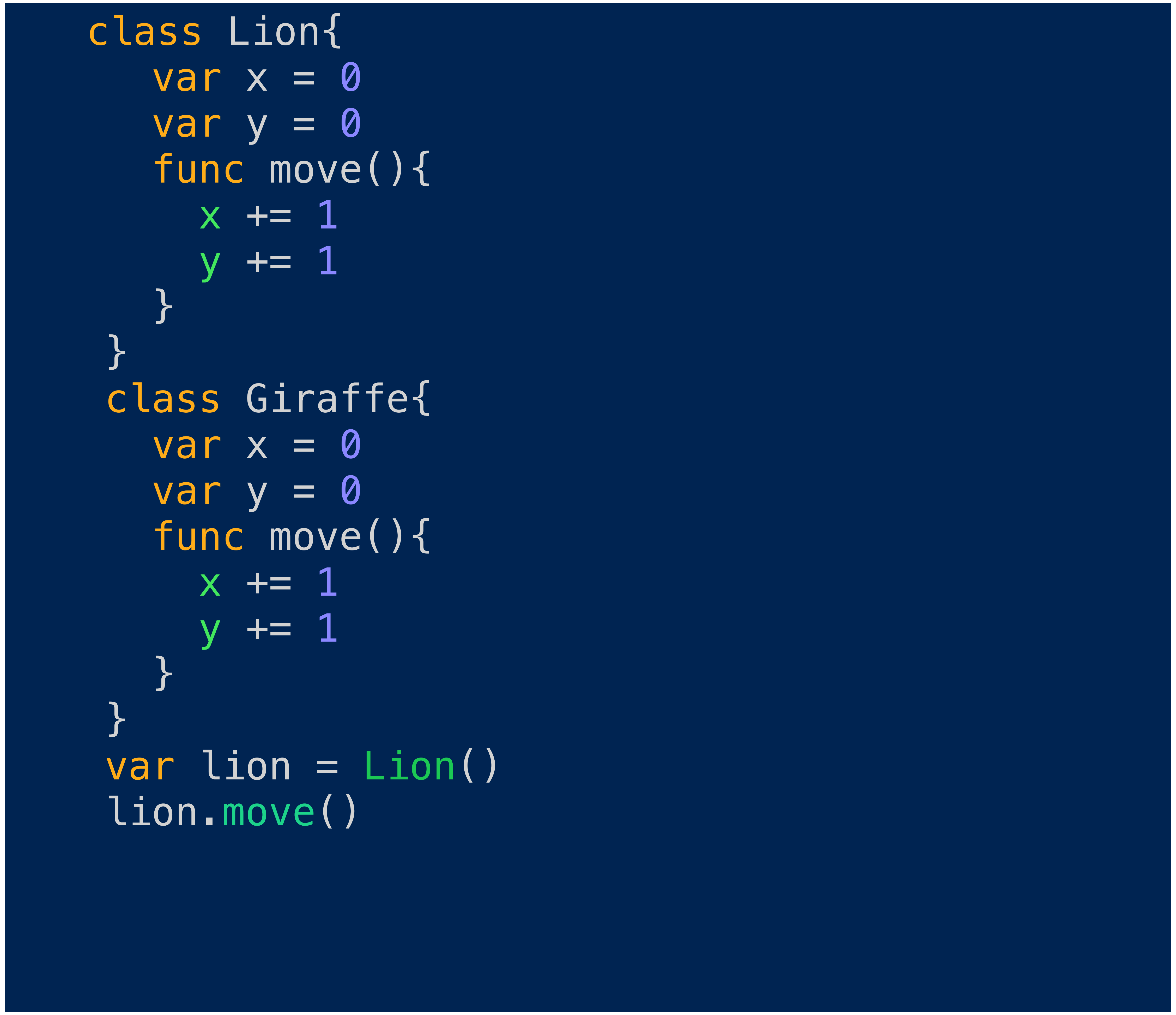
- Objects has state
- State breaks parallel programming
- State is harder to test and debug

#3 More Functions,Less Objects

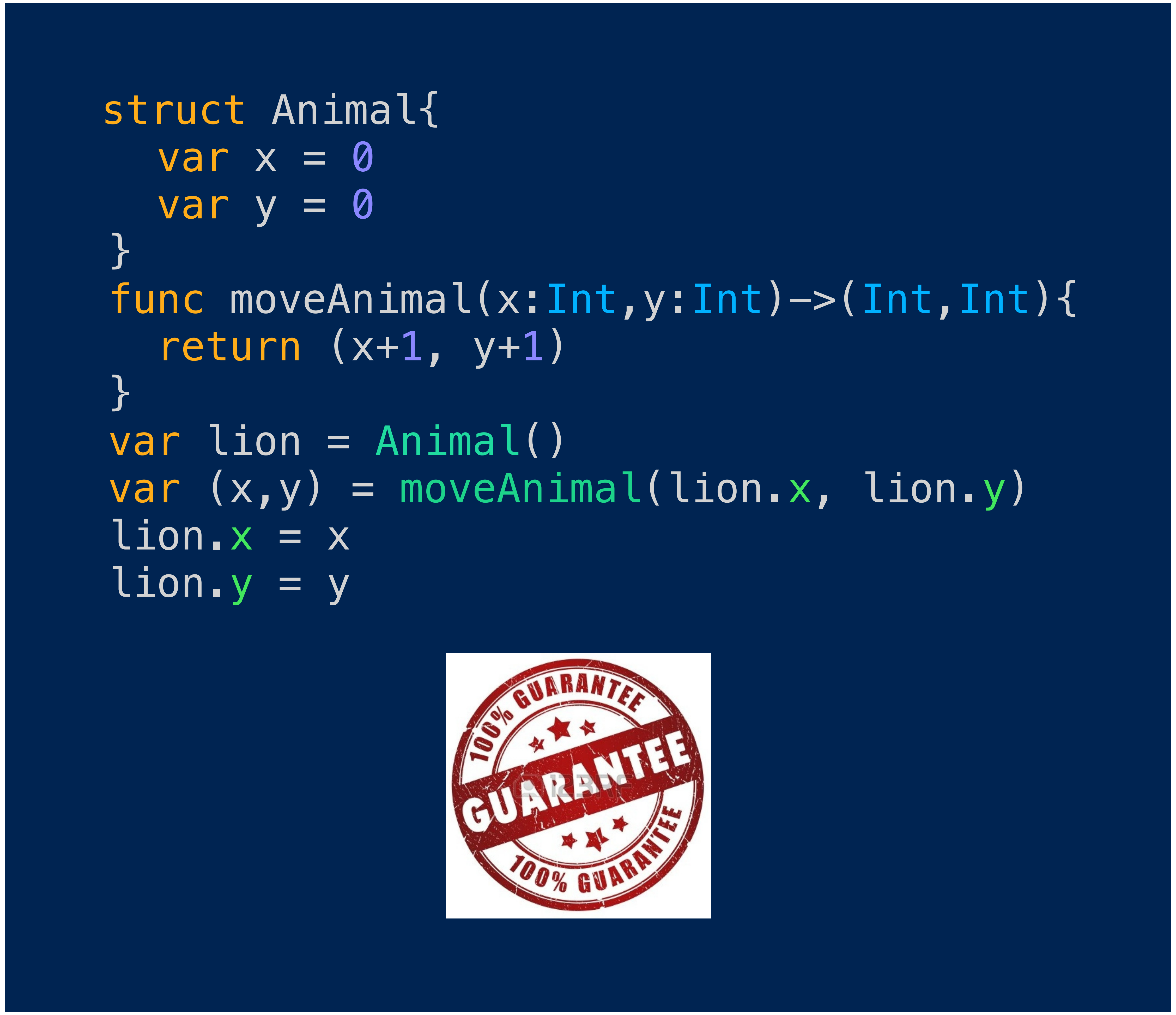


```
class Lion{
   var x = 0
   var y = 0
   func move(){
     x += 1
     y += 1
   }
 }
 class Giraffe{
   var x = 0
   var y = 0
   func move(){
     x += 1
     y += 1
   }
 }
 var lion = Lion()
 lion.move()
```

#3 More Functions,Less Objects



```
class Lion{
   var x = 0
   var y = 0
   func move(){
     x += 1
     y += 1
   }
 }
 class Giraffe{
   var x = 0
   var y = 0
   func move(){
     x += 1
     y += 1
   }
 }
 var lion = Lion()
 lion.move()
```



```
struct Animal{
  var x = 0
  var y = 0
}
func moveAnimal(x:Int,y:Int)->(Int,Int){
  return (x+1, y+1)
}
var lion = Animal()
var (x,y) = moveAnimal(lion.x, lion.y)
lion.x = x
lion.y = y
```

#4 Pretty Pipelining

- F1(N1) => F2(N2) => F3(N3) … FN(NN) => **Result**
- Easier to understand
- Maps - Returns transformed collections
- Filter - Find matching collections items
- Reduce -   Transform collection items into 1 value

#4 Pretty Pipelining

```
    var words = ["hi","hello","no",
"house","car","I","are"]
    // sum all words with length less than 3
    var count = 0
    for w in words{
      // map
      let size = countElements(w)
      // filter out
      if( size < 3 ){
        // reduce
        count += size
      }
    }
```

#4 Pretty Pipelining

```
    var words = ["hi","hello","no",
"house","car","I","are"]
    // sum all words with length less than 3
    var count = 0
    for w in words{
      // map
      let size = countElements(w)
      // filter out
      if( size < 3 ){
        // reduce
        count += size
      }
    }
```

```
var words =
[“hi","hello","no", "house","car","I","are"]

var count =
    words.map({ w in countElements(w) } )
    .filter({size in size < 3 })
    .reduce(0, { (sum,size) in sum+size})
```

#5 Leave OO ASAP

- Impossible to be 100% Functional
- User Input (Swipes, Types, Clicks)
- Device Programming (DB, Mouse, Screen)
- Reduce OO code to what’s necessary (Input Readers, Writers)

#5 Leave OO ASAP

```
var userLabel = UILabel()
var shortNameLabel = UILabel()

func getShortName(str:String){
  return ....
}
let shortName = getShortName(userLabel.text!)
```

5 Properties Recap

1. Read Inputs, Write Outputs
2. No Control Loops
3. More Functions, Less Objects
4. Pretty Pipeline
5. Leave OO ASAP

Questions?

- John Cao @jobacao
- Thanks!