SWIFT – THE ESSENTIALS

Nick Chen
Fall 2015
talentspark.io

SYLLABUS

Week 1	Swift 2.0, Practical Xcode
Week 2	First iOS app and TableViews
Week 3	TableViews and Auto Layout
Week 4	More TableViews with Parse SDK
Week 5	Animation
Week 6	Core Location & Mapkit
Week 7	*Project*
Week 8	*Project*

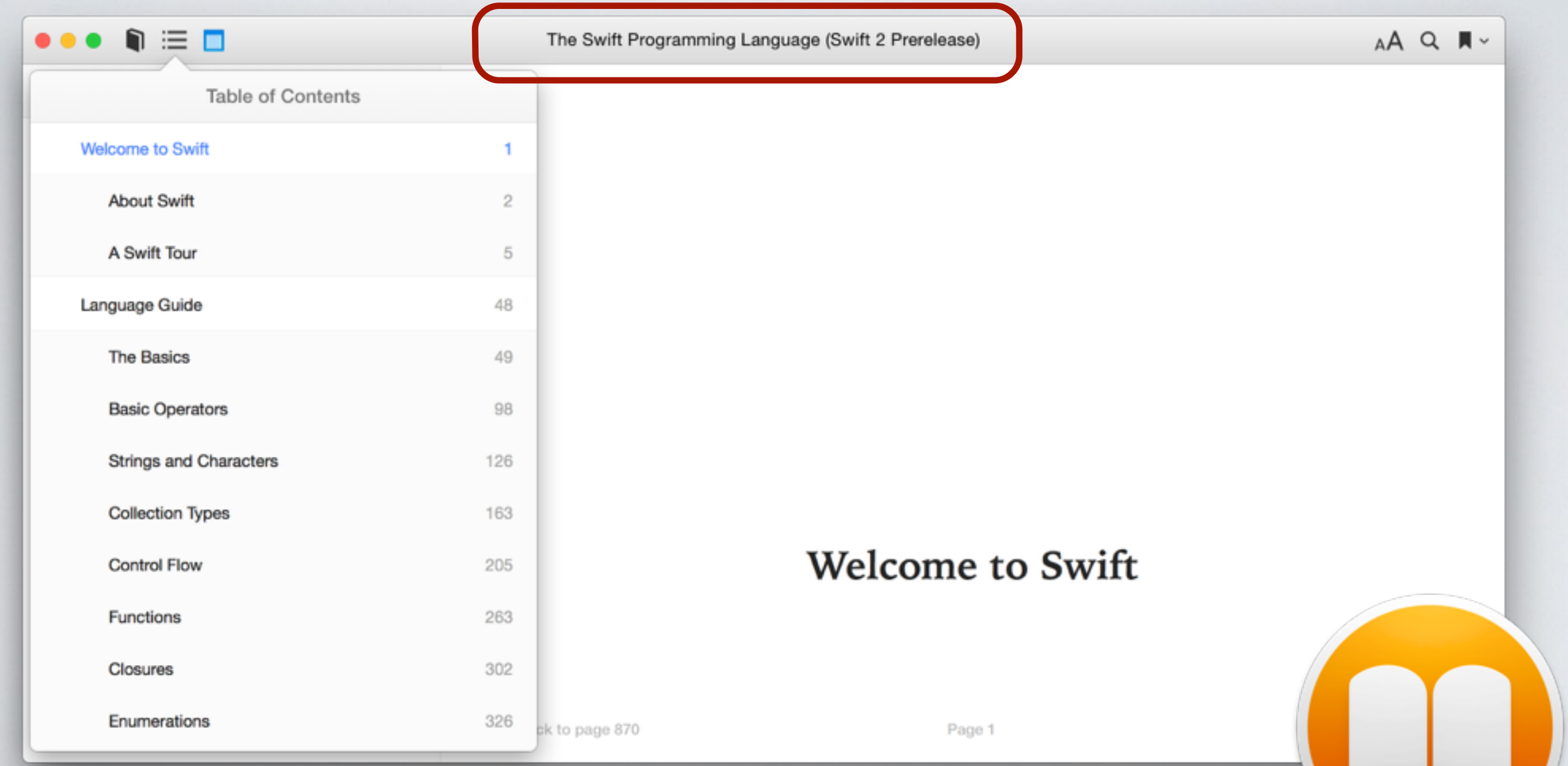
INTRO

- *New* programming language for iOS, OS X and watch OS.

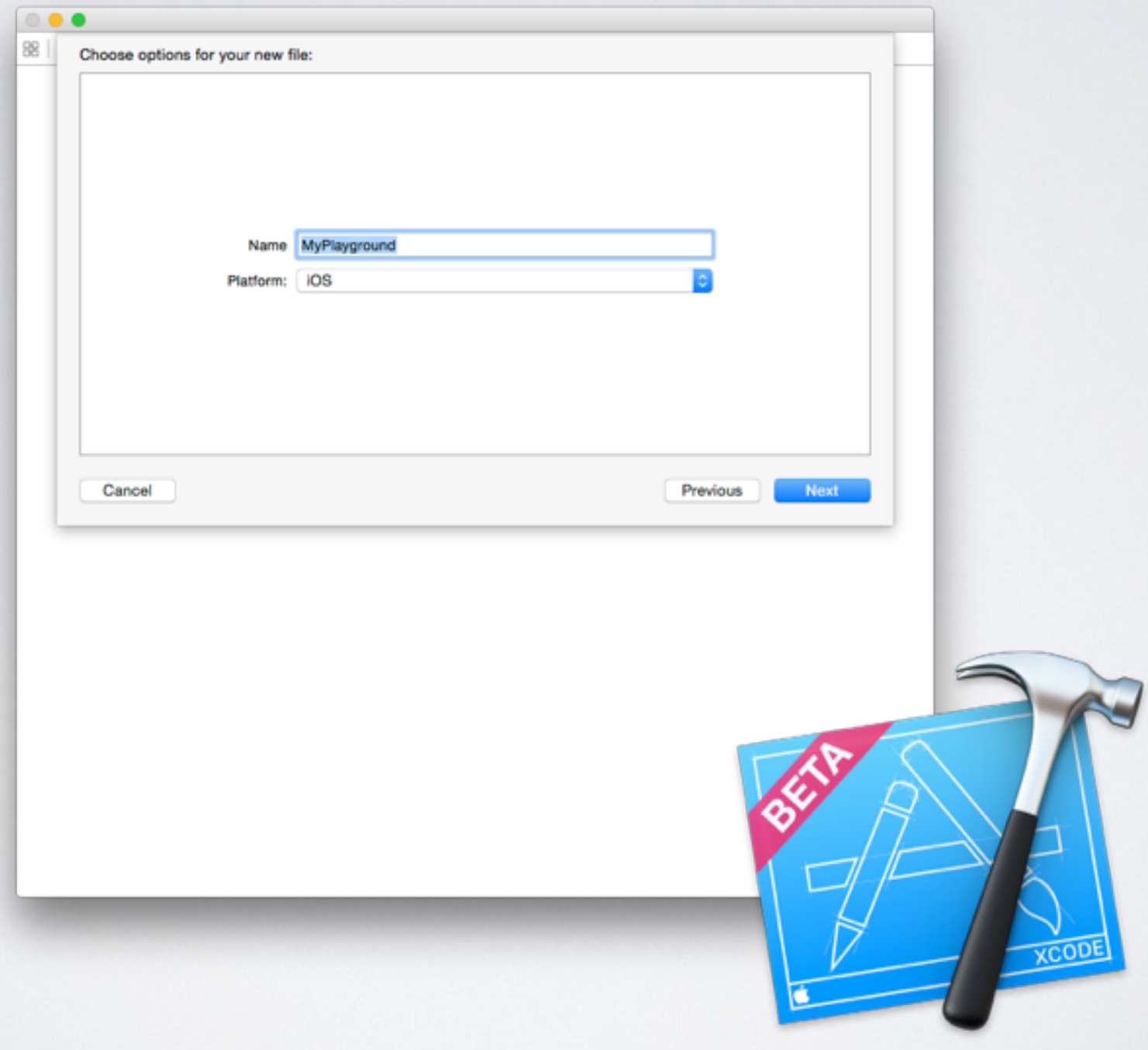
- Swift 1 introduced in WWDC 2014.
  Swift 2 introduced in WWDC 2015.

- "…drawing ideas from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list."

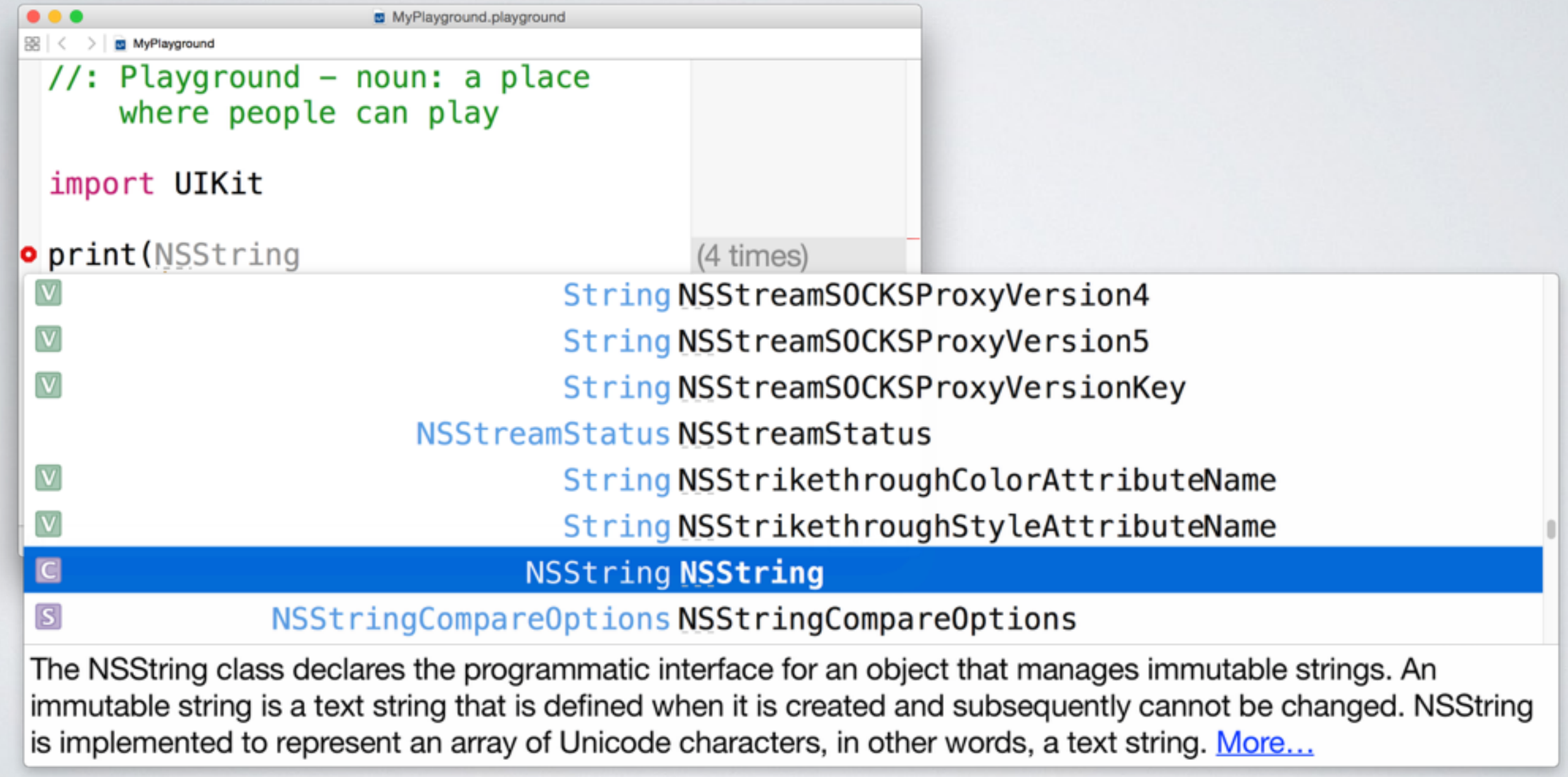
REFERENCE


The Swift Programming Language (Swift 2 Prerelease)
Table of Contents
Welcome to Swift 1
About Swift 2
A Swift Tour 5
Language Guide 48
The Basics 49
Basic Operators 98
Strings and Characters 126
Collection Types 163
Control Flow 205
Functions 263
Closures 302
Enumerations 326
ck to page 870
Page 1
Welcome to Swift

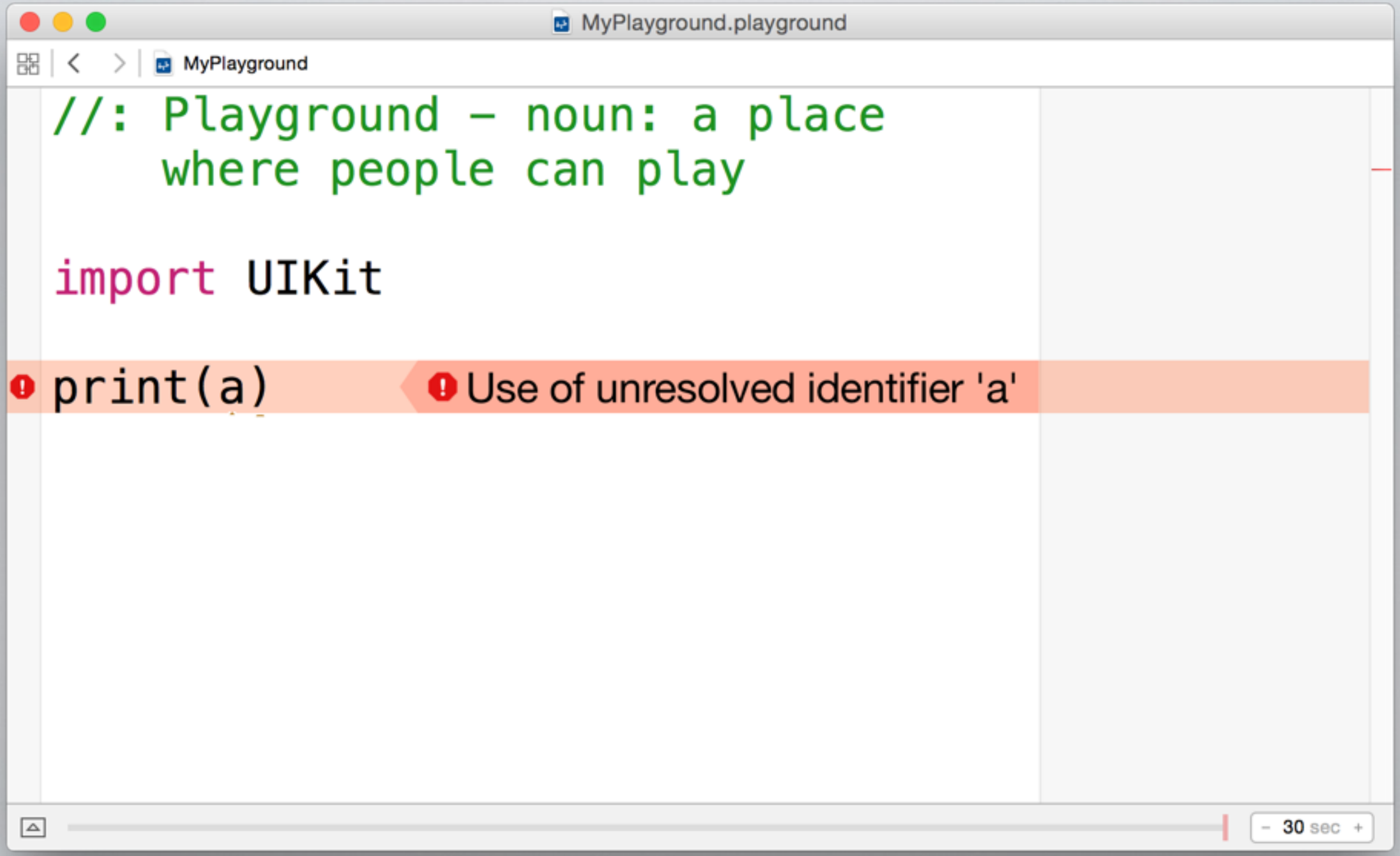
PLAYGROUNDS



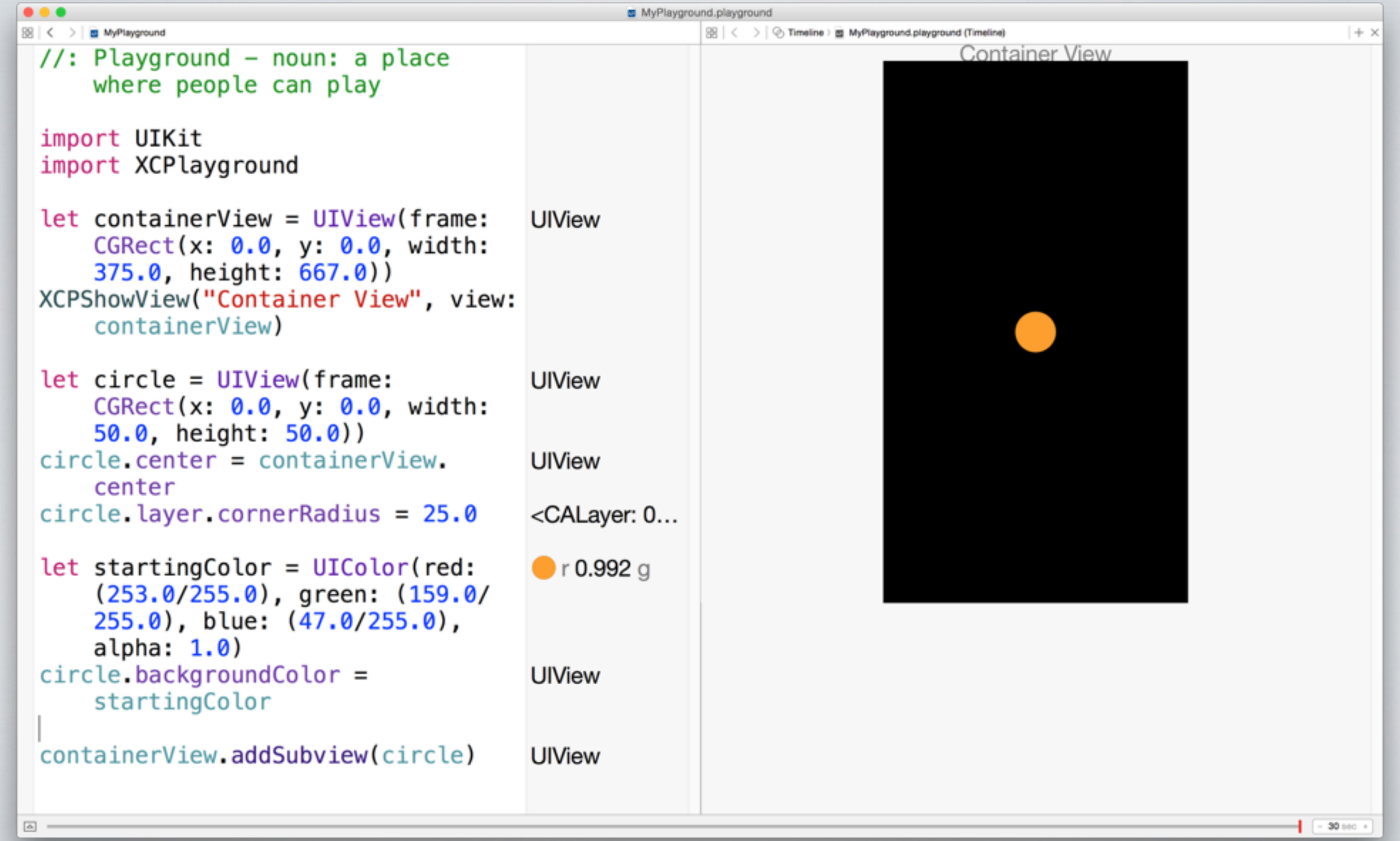
PLAYGROUNDS



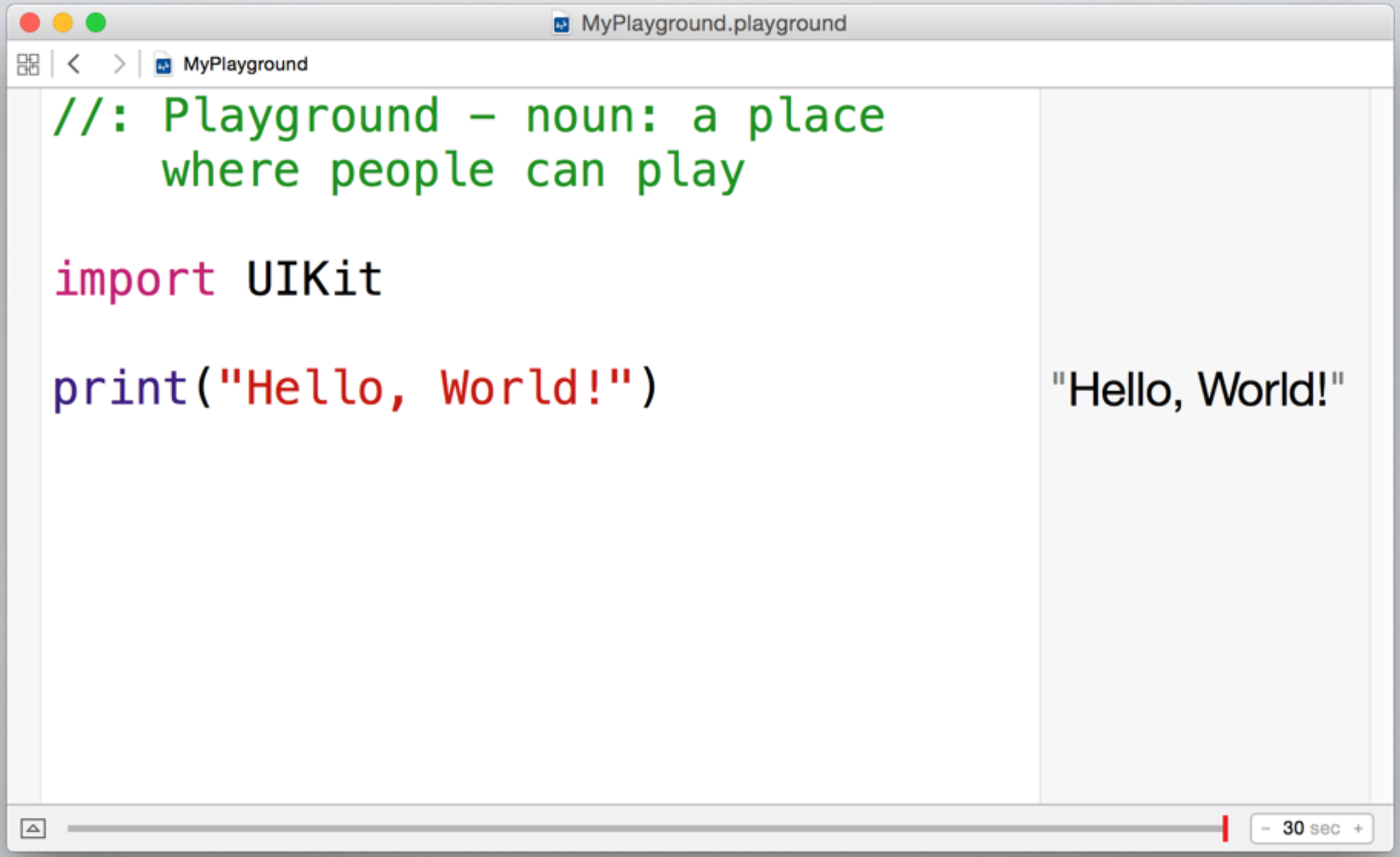
PLAYGROUNDS



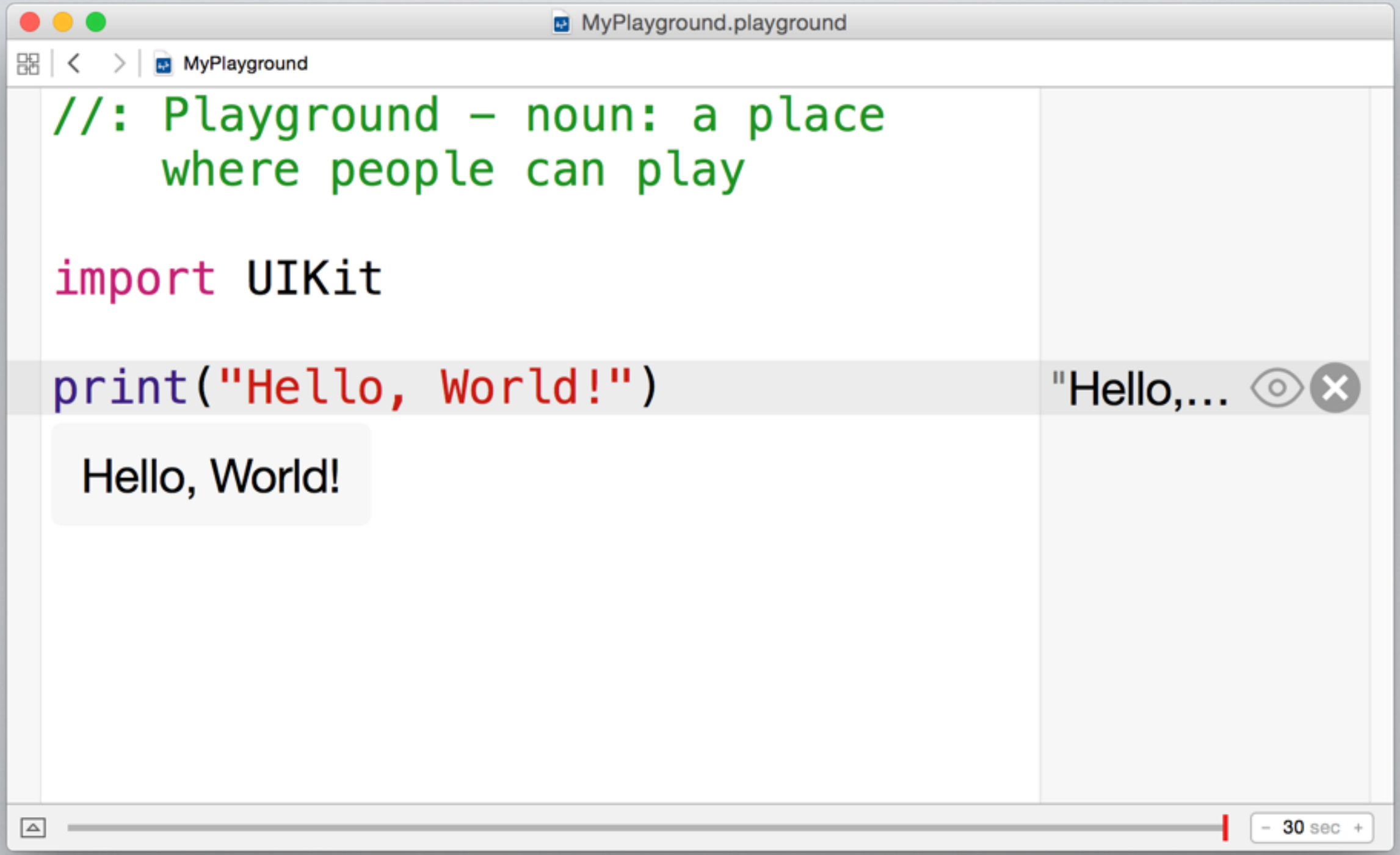
PLAYGROUNDS



HELLO, WORLD!


MyPlayground.playground
MyPlayground
//: Playground - noun: a place
where people can play
import UIKit
print("Hello, World!")
"Hello, World!"
30 sec

HELLO, WORLD!


MyPlayground.playground
MyPlayground
//: Playground - noun: a place
where people can play
import UIKit
print("Hello, World!")
"Hello,...
Hello, World!
30 sec

VARIABLES VS CONSTANTS

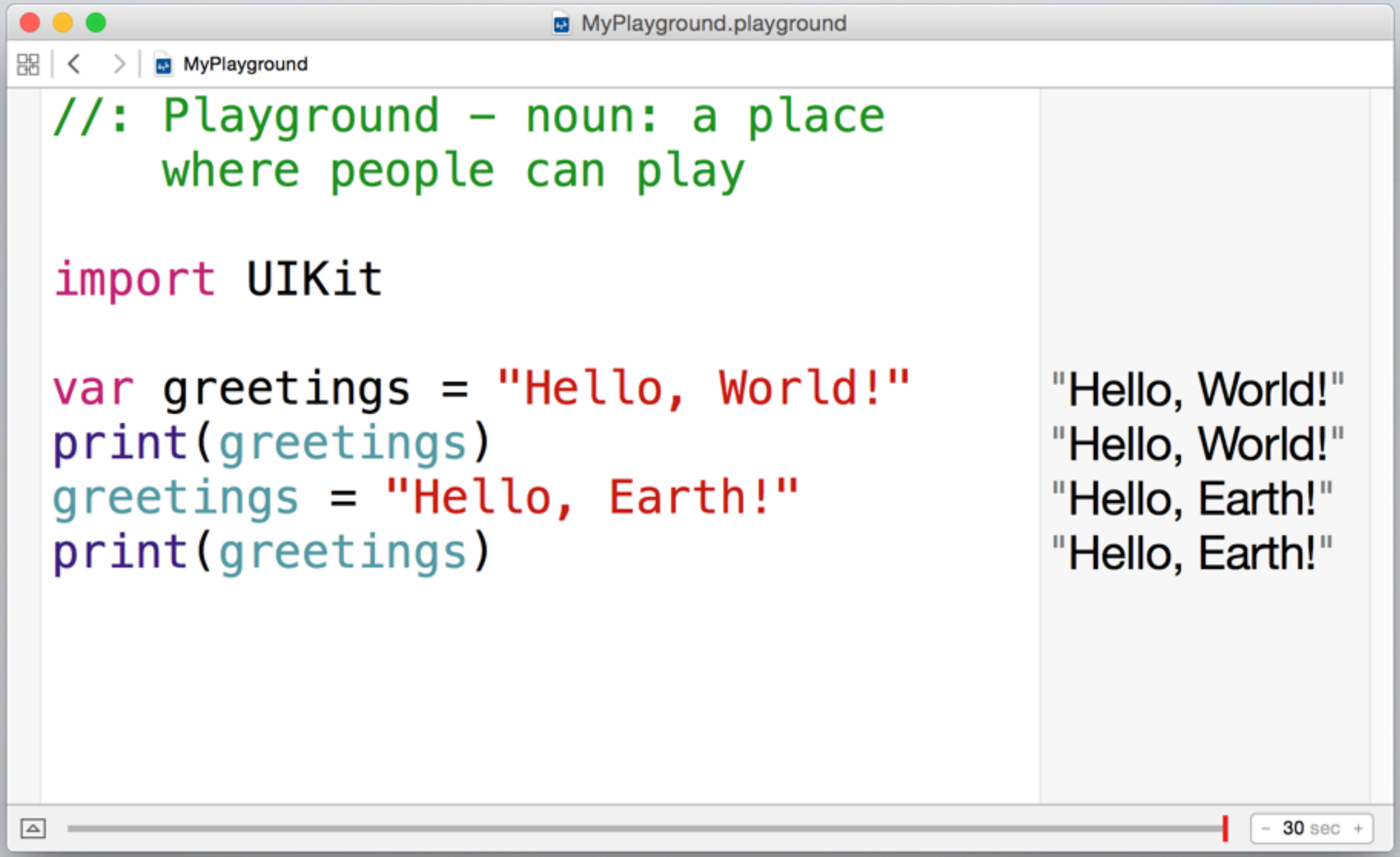
MyPlayground.playground

MyPlayground

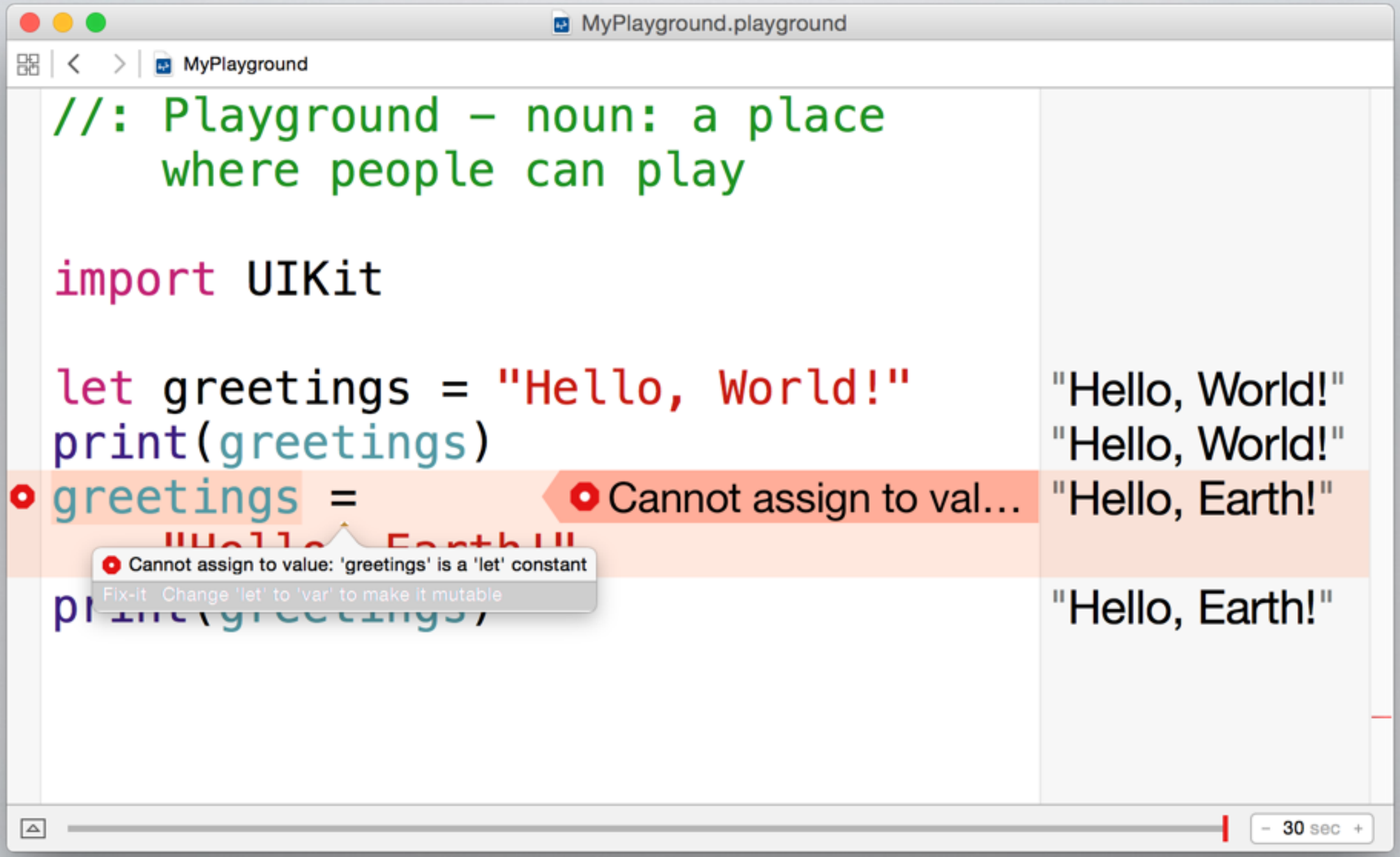
```
//: Playground - noun: a place
    where people can play

import UIKit

var greetings = "Hello, World!"          "Hello, World!"
print(greetings)                         "Hello, World!"
```

30 sec

VARIABLES VS CONSTANTS



VARIABLES VS CONSTANTS



PRIMITIVE TYPES

MyPlayground.playground

MyPlayground

```
//: Playground - noun: a place
    where people can play

import UIKit

let greetings = "Hello, World!"                "Hello, World!"
greetings is String                            true
greetings.dynamicType                          String.Type
```

30 sec

PRIMITIVE TYPES

MyPlayground.playground

MyPlayground

```
//: Playground - noun: a place
    where people can play

import UIKit

let greetings: String = "Hello,
    World!"                          "Hello, World!"
greetings is String                  true
greetings.dynamicType                String.Type
```

30 sec

PRIMITIVE TYPES

MyPlayground.playground

MyPlayground

```
//: Playground - noun: a place
    where people can play

import UIKit

let myNumber = 1234                          1234
myNumber.dynamicType                         Int.Type

// You probably won't need to use
// Int16, Int32, Int64
// UInt16, UInt32, UInt64
let myUInt16: UInt16 = 1234                  1234
myNumber.dynamicType                         Int.Type
```

30 sec

PRIMITIVE TYPES

MyPlayground.playground

MyPlayground

```
//: Playground - noun: a place
    where people can play

import UIKit

let myFloatingPoint = 1.234                1.234
myFloatingPoint.dynamicType                Double.Type

let anotherFloatingPoint: Float =          1.234
    1.234
anotherFloatingPoint.dynamicType           Float.Type
```

30 sec

PRIMITIVE TYPES

MyPlayground.playground

MyPlayground

```
//: Playground - noun: a place
    where people can play

import UIKit

let myBoolean = true                    true
myBoolean.dynamicType                   Bool.Type

// Does not let you do this
// unlike other languages
let i = 1
if i {
    Type 'Int' does not conform to protocol 'B…
```

30 sec

STRING INTERPOLATION

MyPlayground.playground

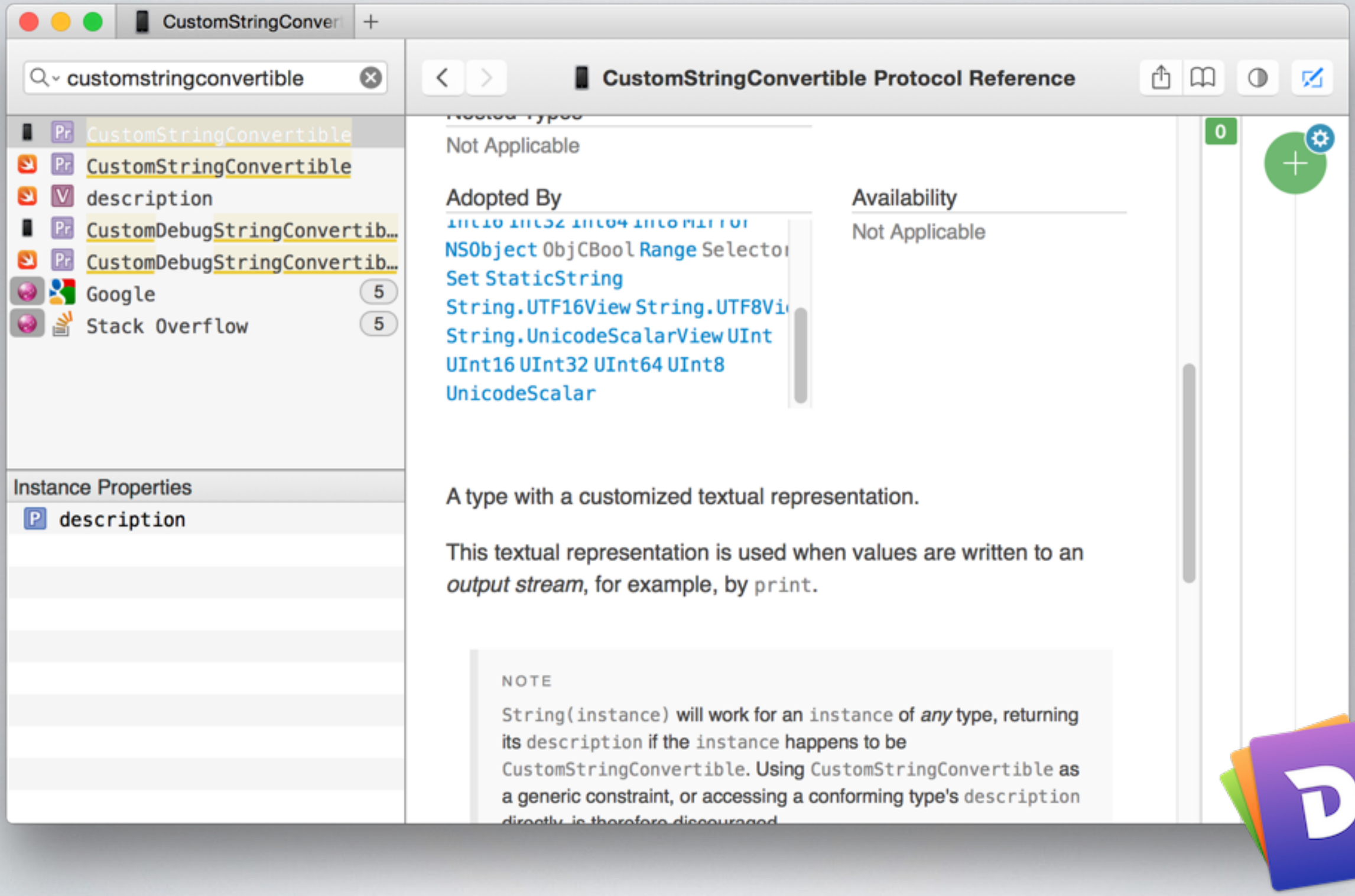
MyPlayground

```
//: Playground - noun: a place
    where people can play

import UIKit

let myInt = 2                                   2
let myBool = false                              false
print("The value of myInt is \                 "The valu...
    (myInt) and the value of myBool
    is \(myBool)")
```

The value of myInt is 2 and the value of myBool is false

30 sec

STRING INTERPOLATION



COLLECTIONS

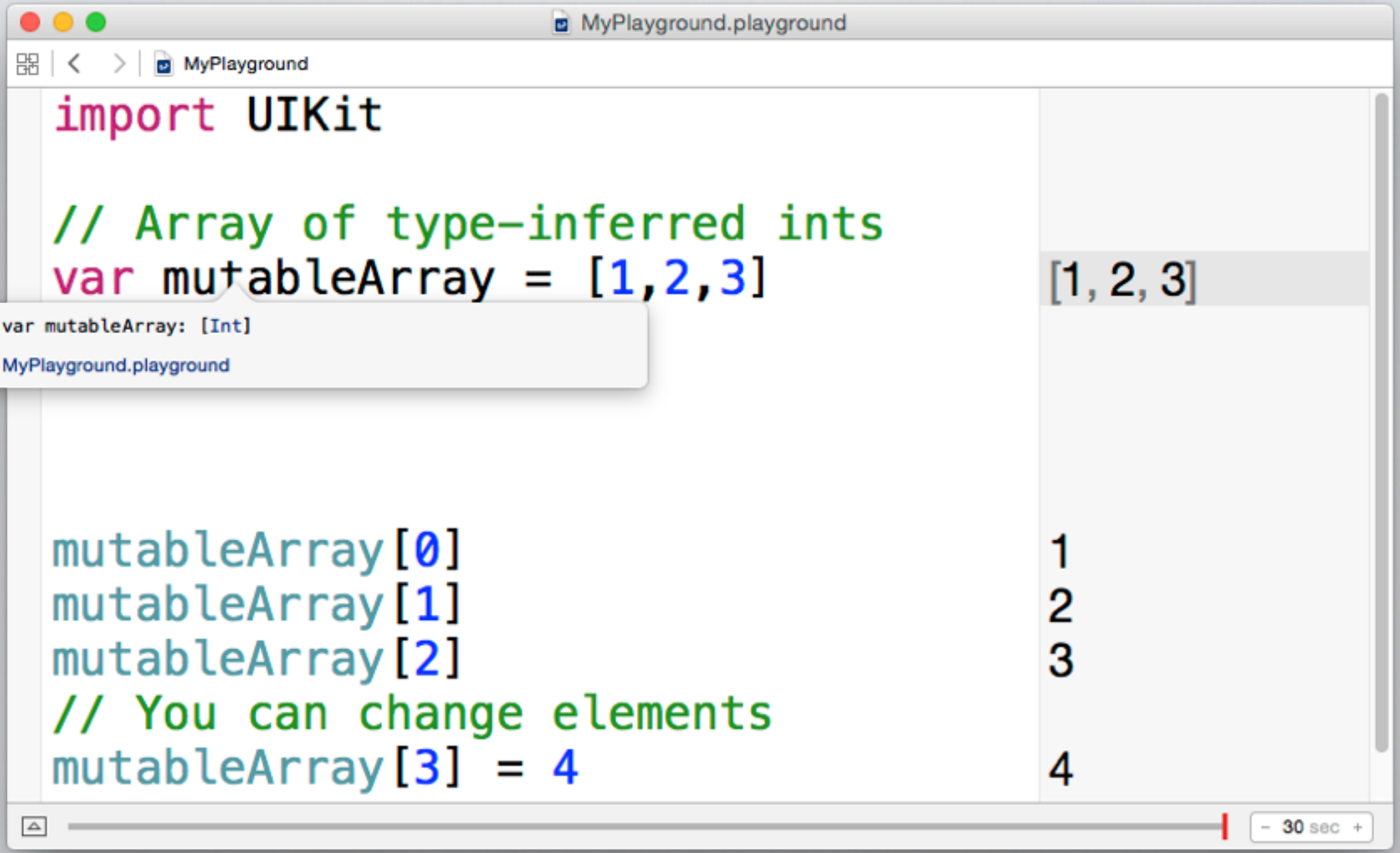
MyPlayground.playground

MyPlayground

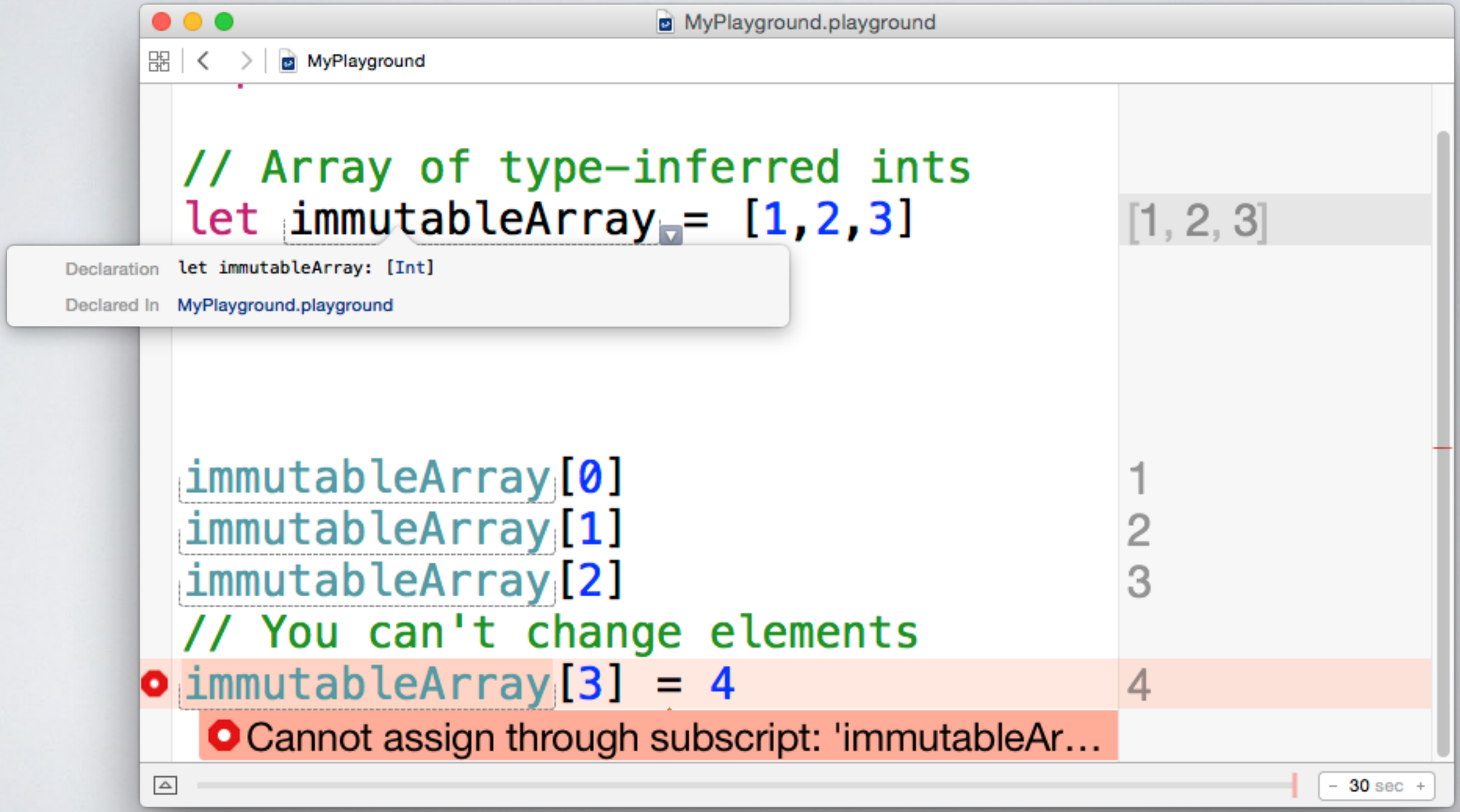
```
//: Playground - noun: a place
    where people can play

import UIKit

let myTuple = (firstVal: "a",          (.0 "a", .1 1, ....
    secondVa1: 1,
    thirdVal: true)
myTuple.firstVal                       "a"
myTuple.0                              "a"
myTuple.secondVa1                      1
myTuple.1                              1
```

30 sec

COLLECTIONS

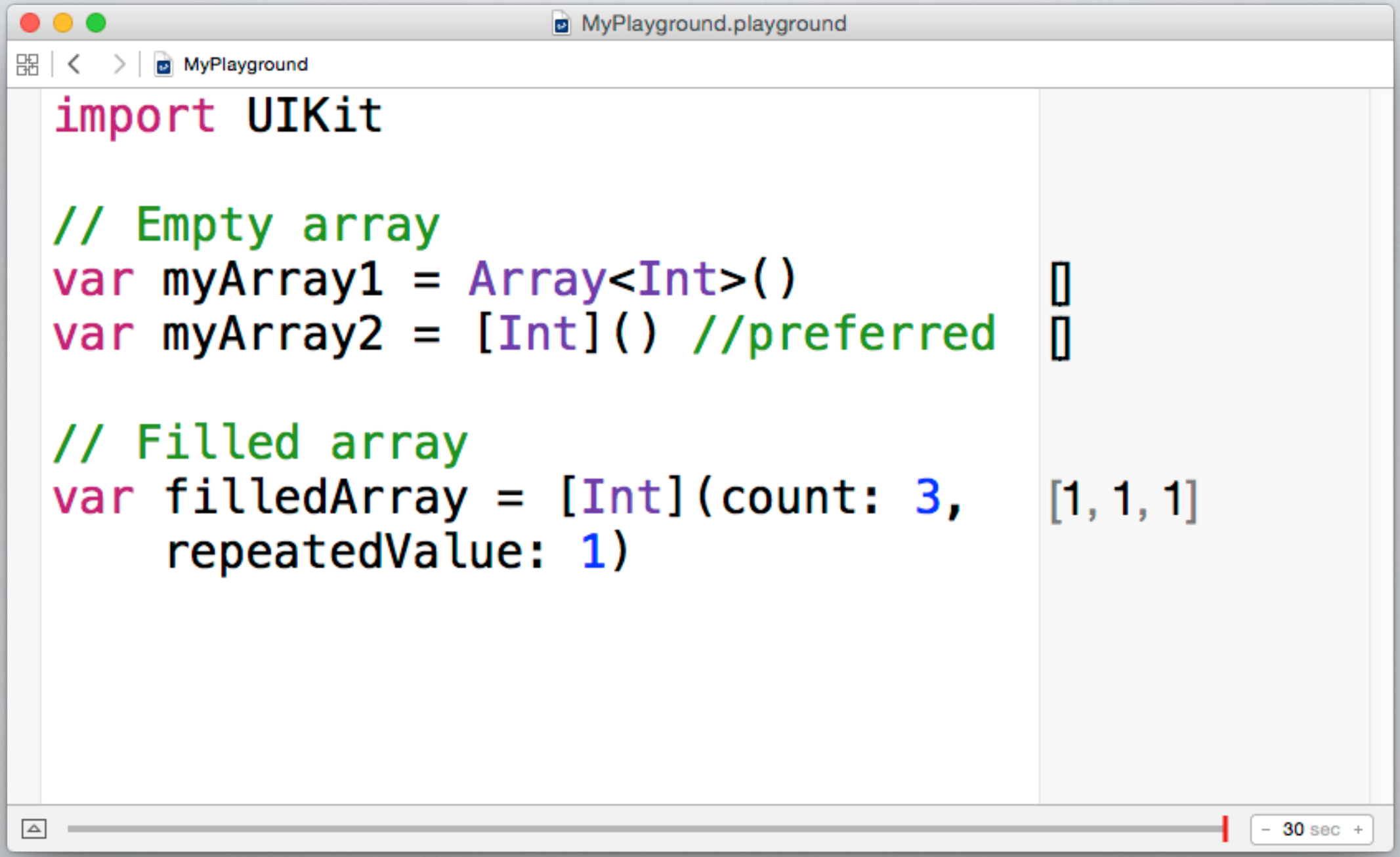


COLLECTIONS

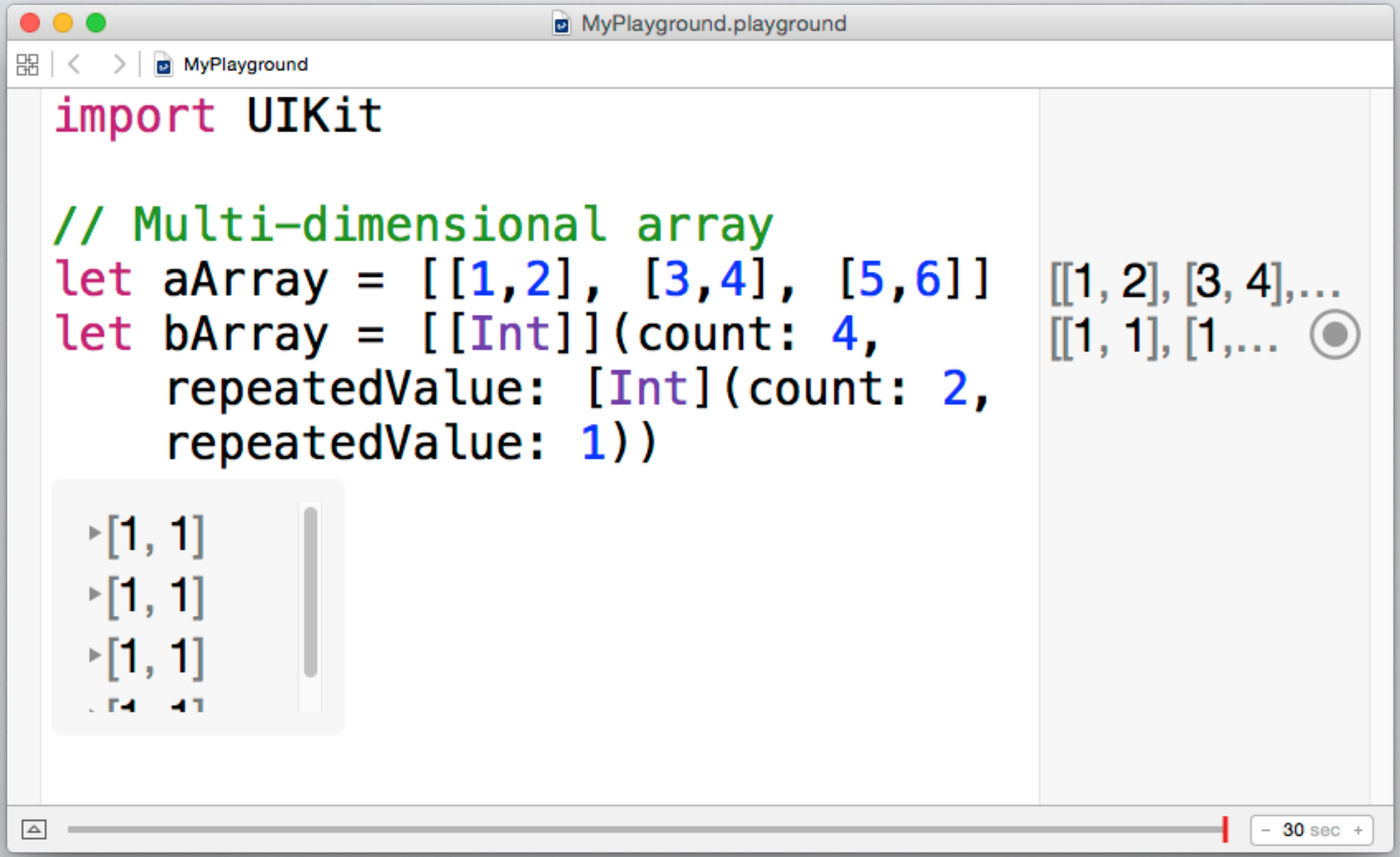


COLLECTIONS

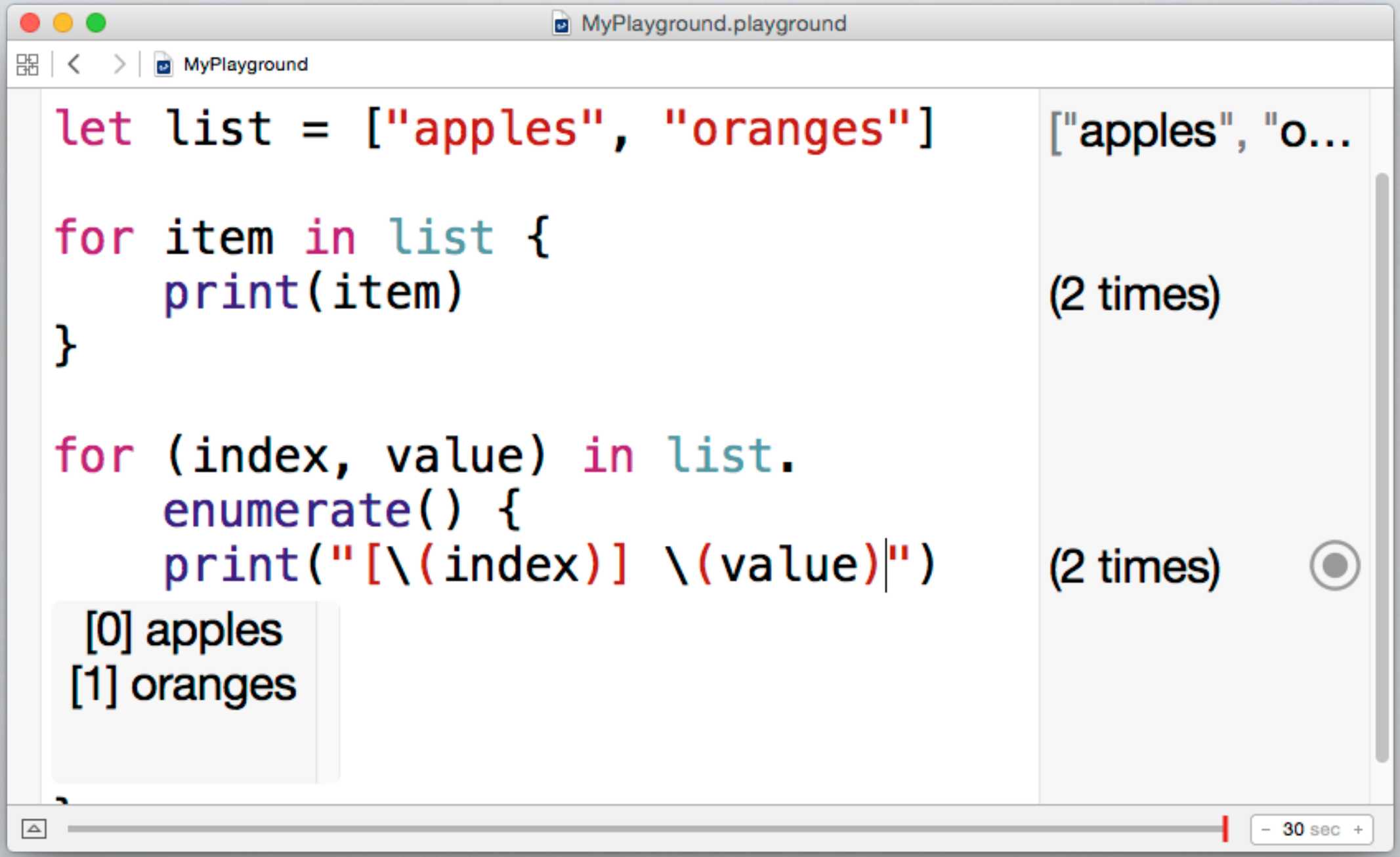
COLLECTIONS



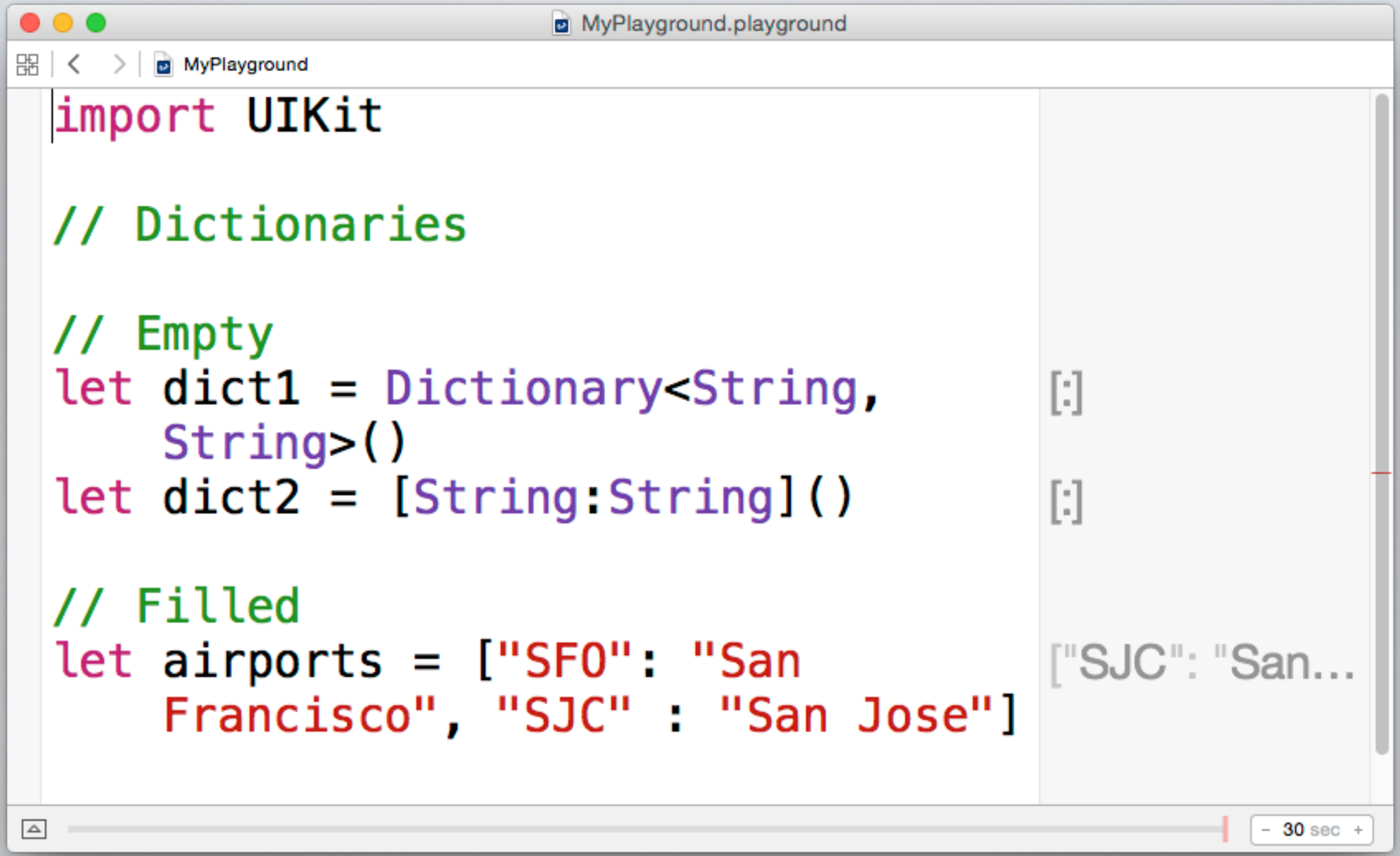
COLLECTIONS



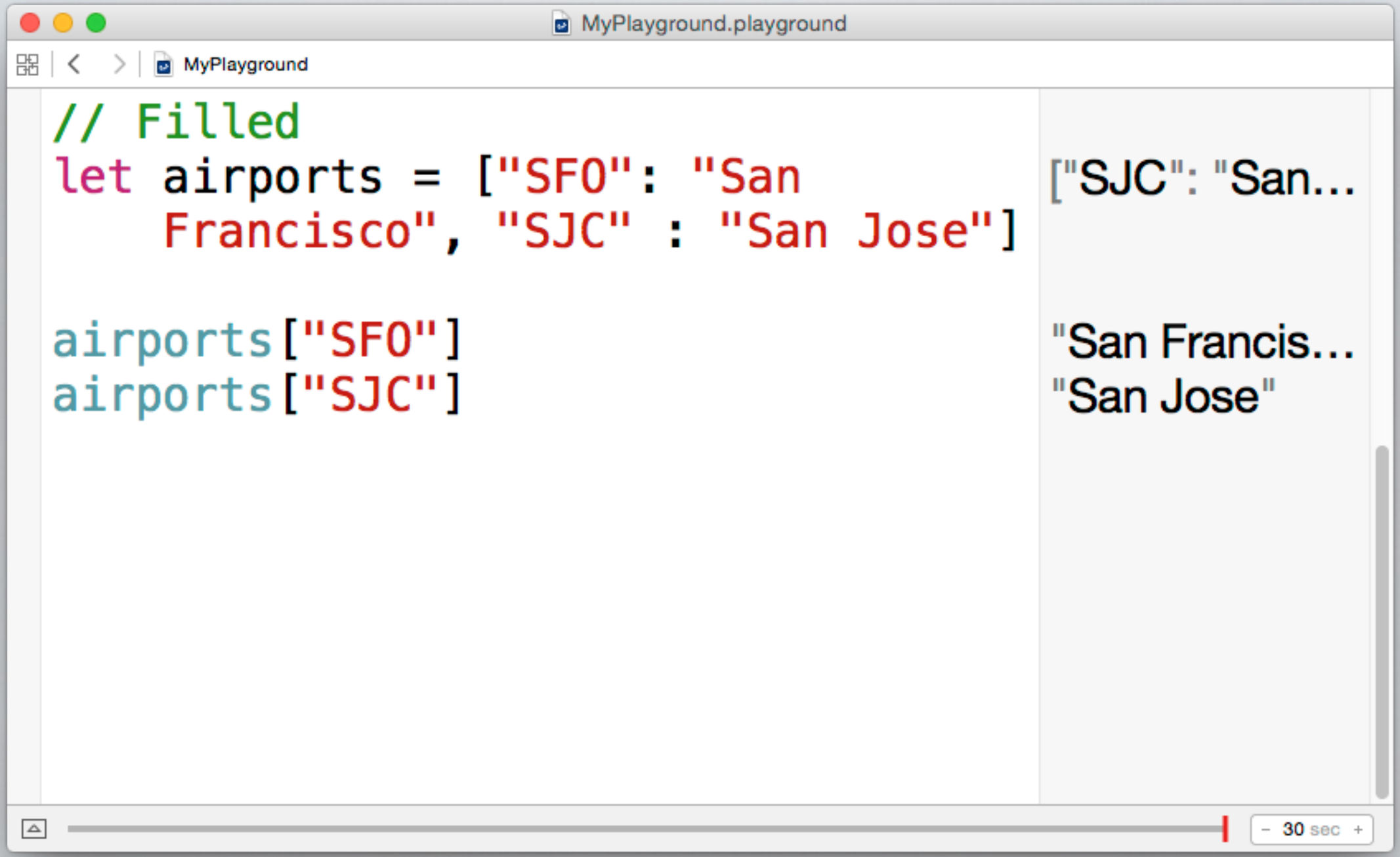
COLLECTIONS



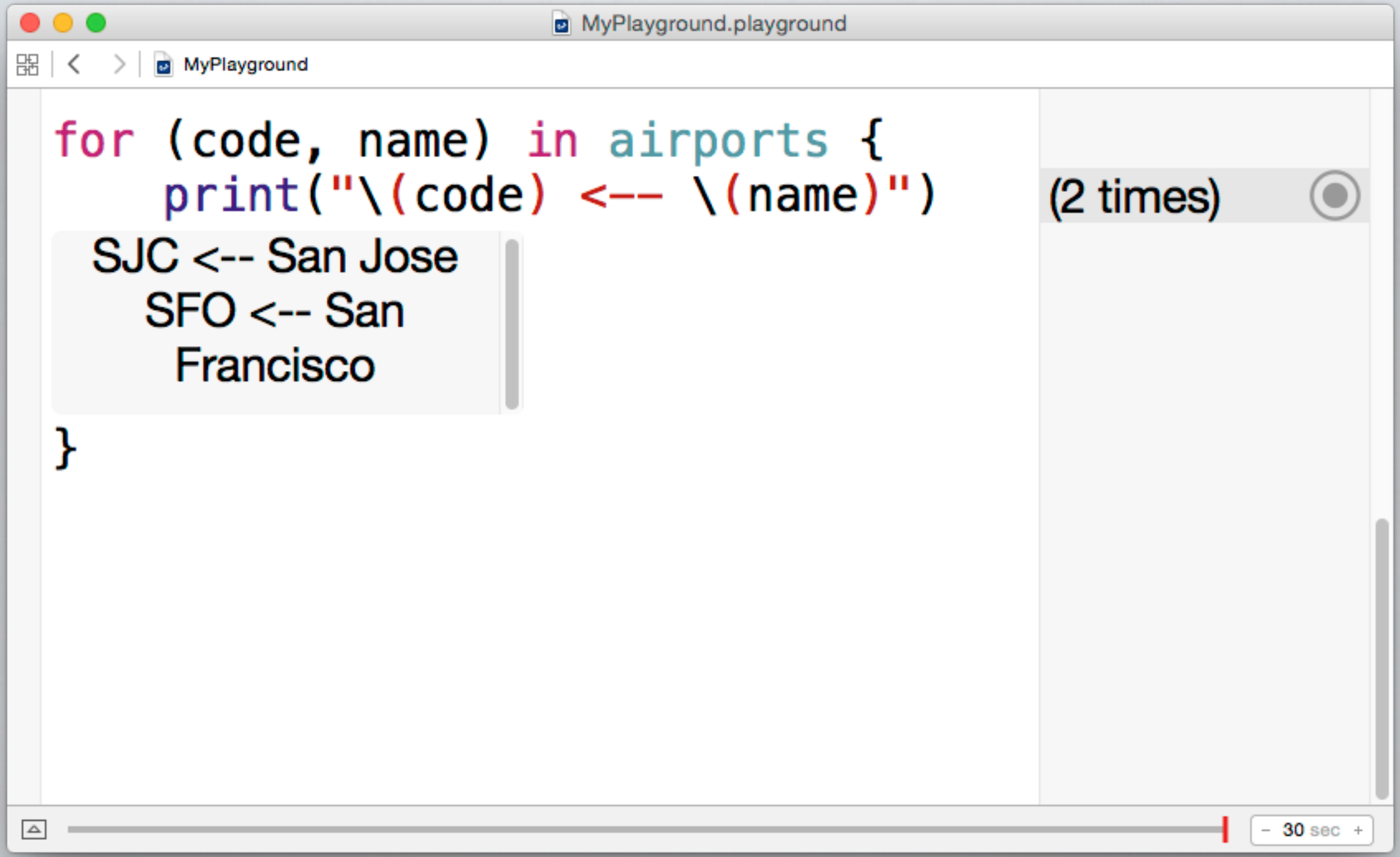
COLLECTIONS



COLLECTIONS

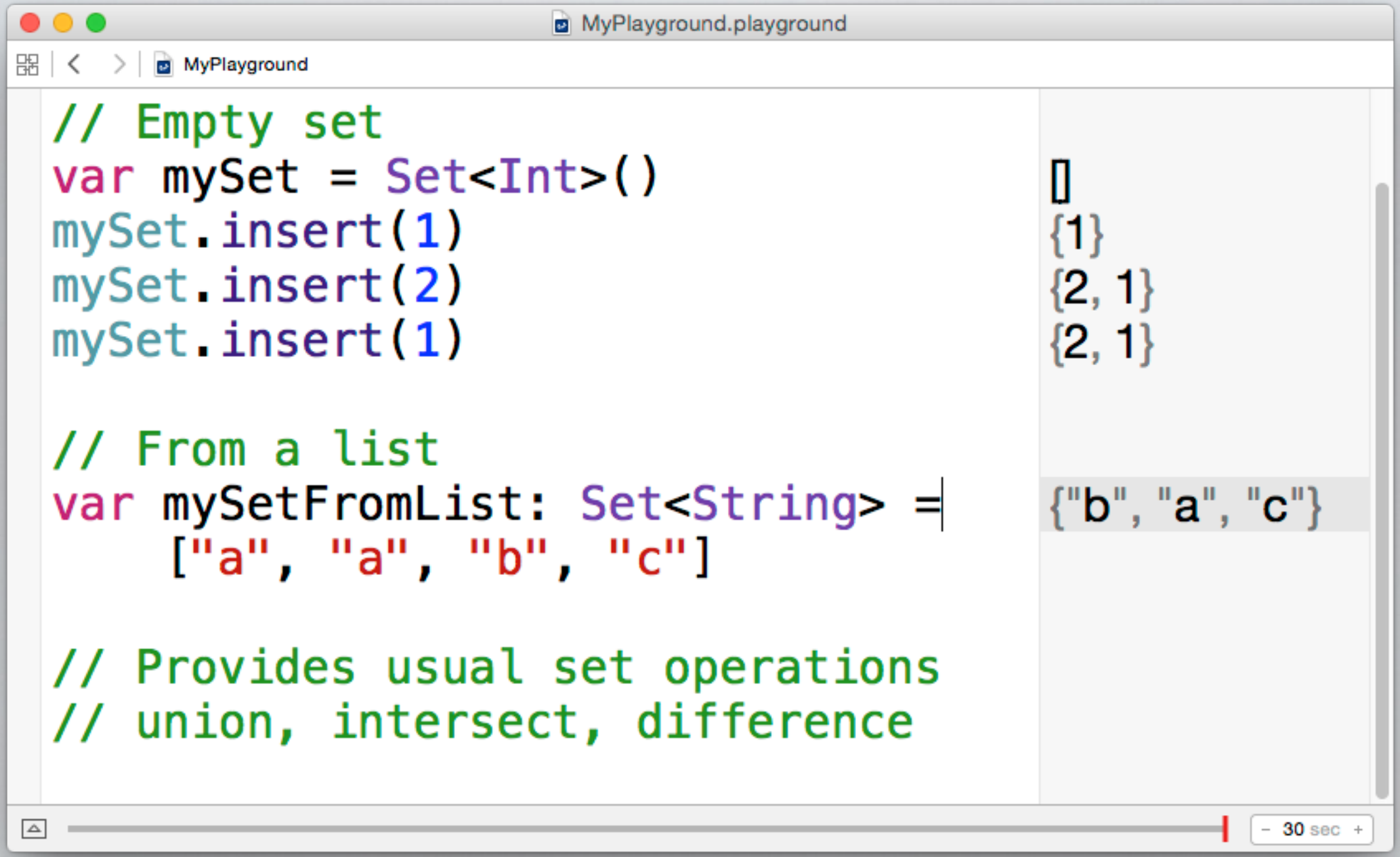


COLLECTIONS

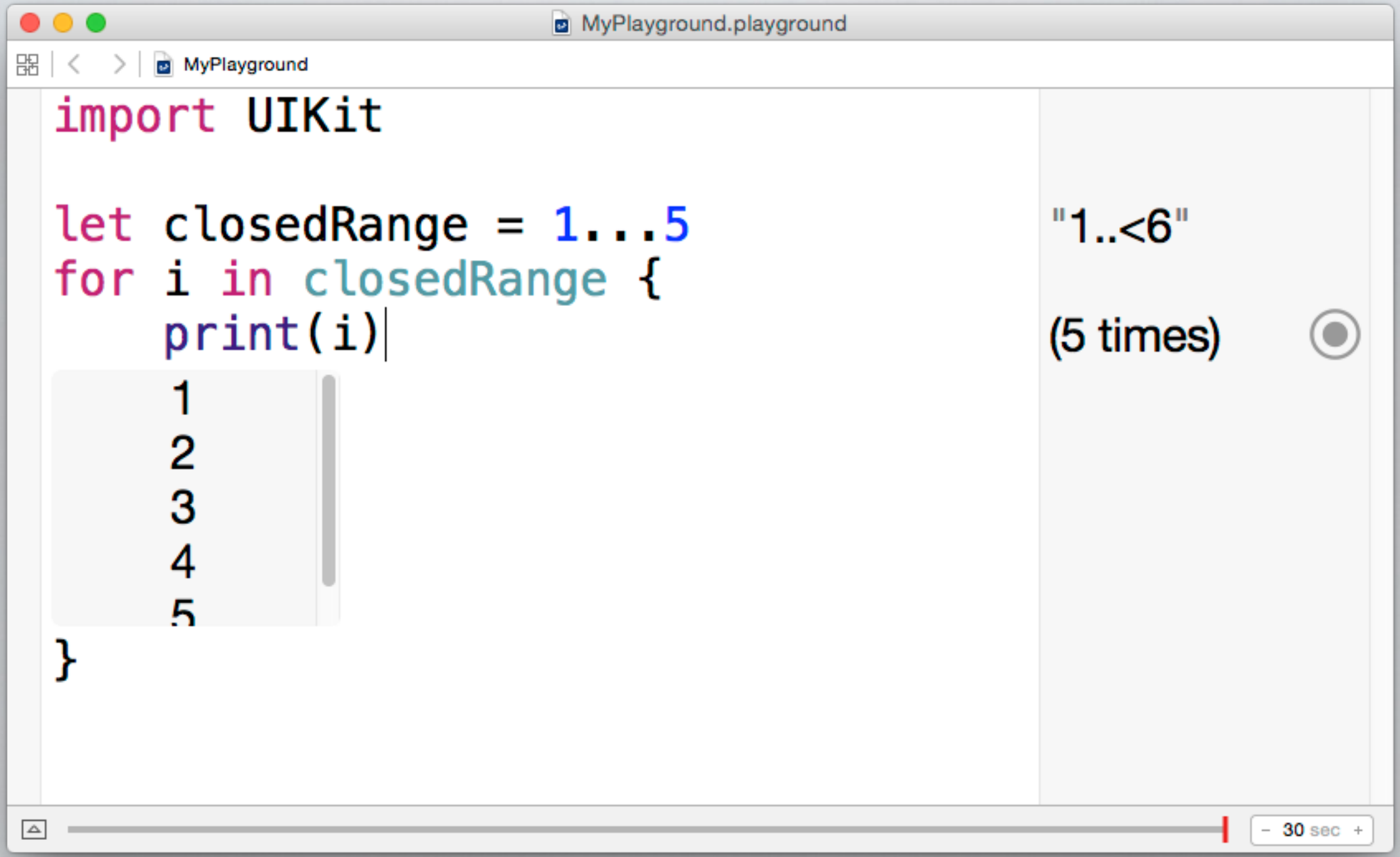


COLLECTIONS

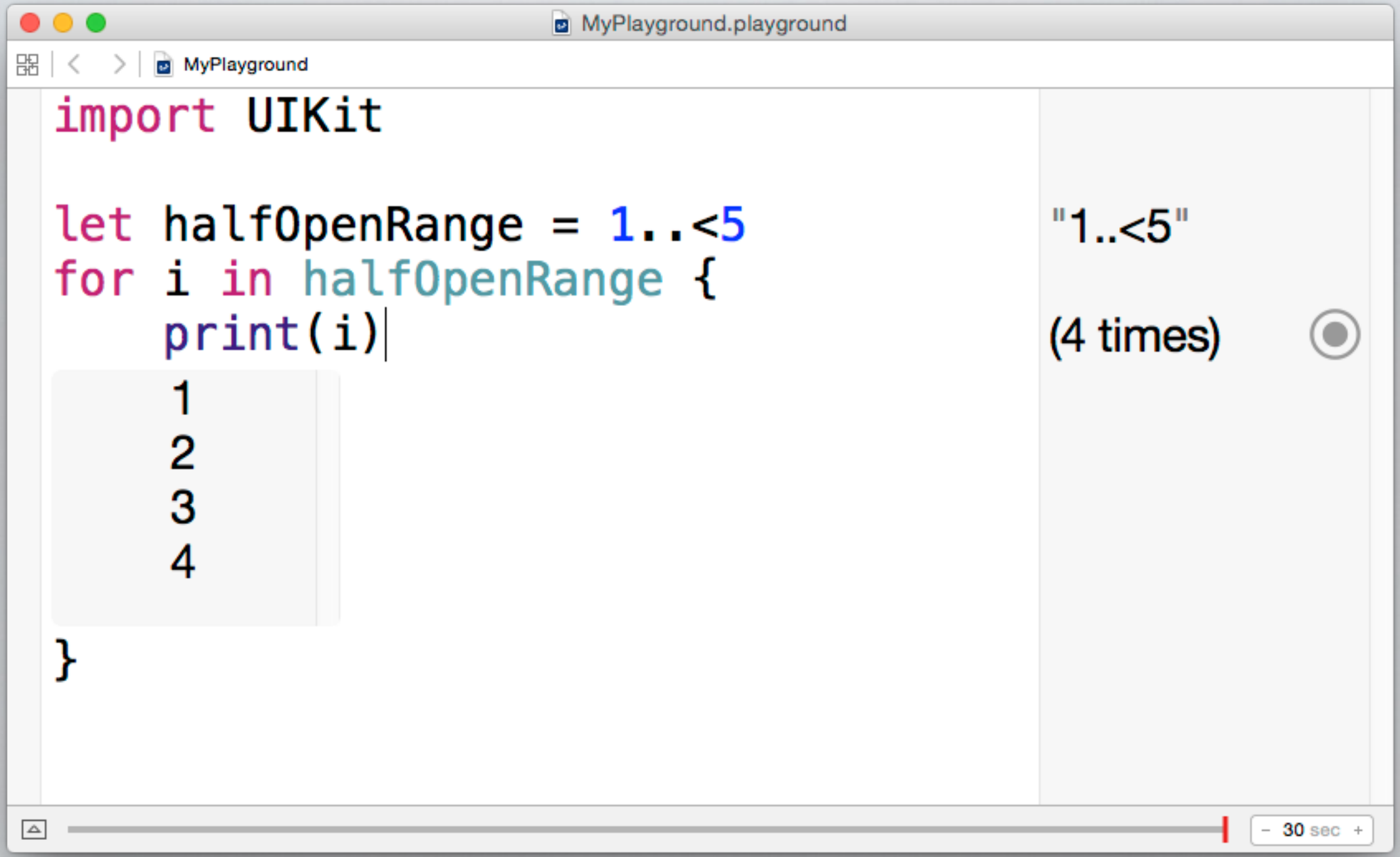
COLLECTIONS



RANGES


MyPlayground.playground
MyPlayground
import UIKit
let closedRange = 1...5
for i in closedRange {
print(i)
1
2
3
4
5
}
"1..<6"
(5 times)
30 sec

RANGES


MyPlayground.playground
MyPlayground
import UIKit
let halfOpenRange = 1..<5
for i in halfOpenRange {
print(i)
1
2
3
4
}
"1..<5"
(4 times)
30 sec

FUNCTIONS

MyPlayground.playground

MyPlayground

```
import UIKit

// Function
// with no argument
// returns void
func myFunction() {
    print("myFunction")
}

// Call the function
myFunction()
```

"myFunction"

30 sec

FUNCTIONS

MyPlayground.playground

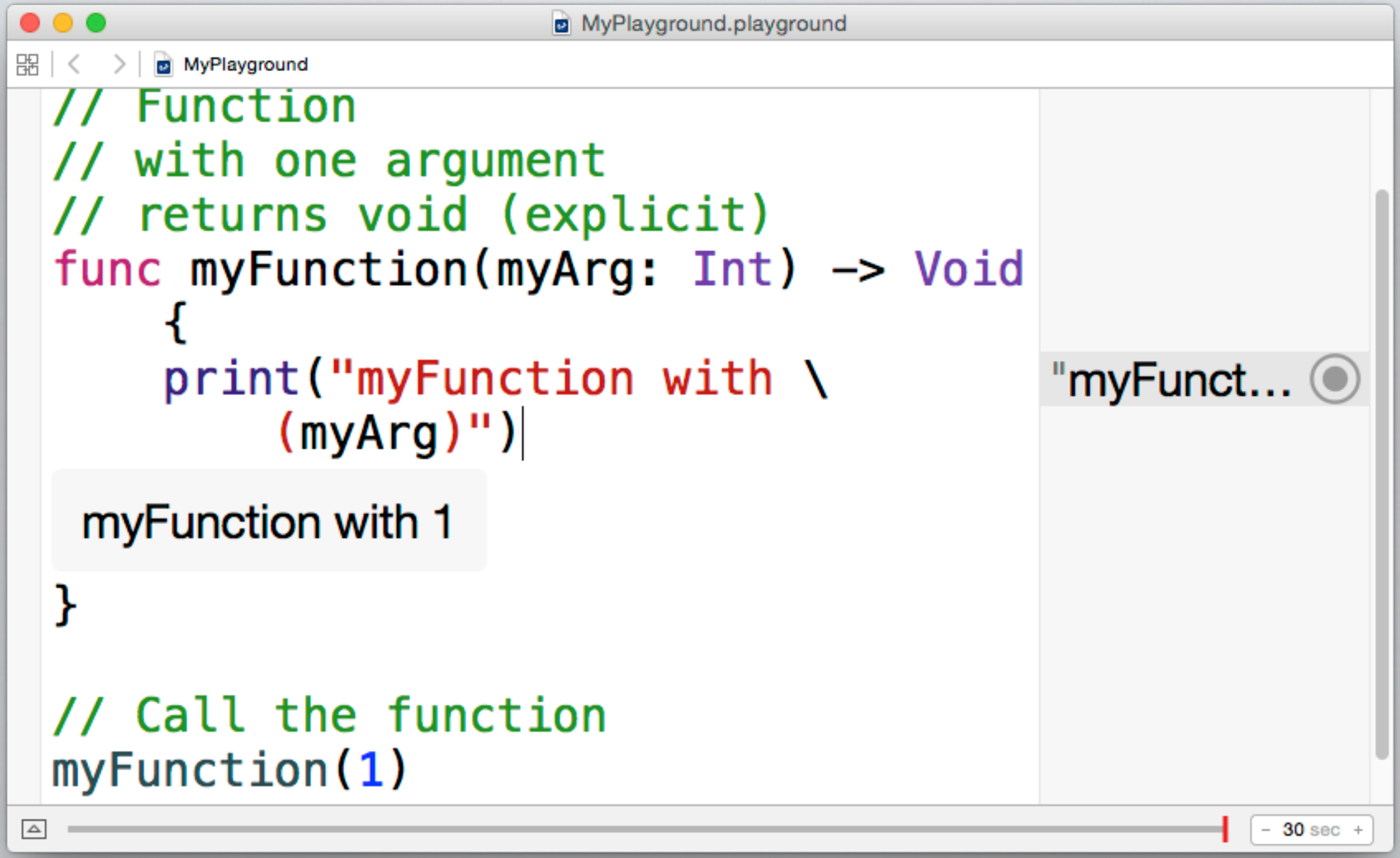
MyPlayground

```
import UIKit

// Function
// with no argument
// returns void (explicit)
func myFunction() -> Void {
    print("myFunction")
}

// Call the function
myFunction()
```

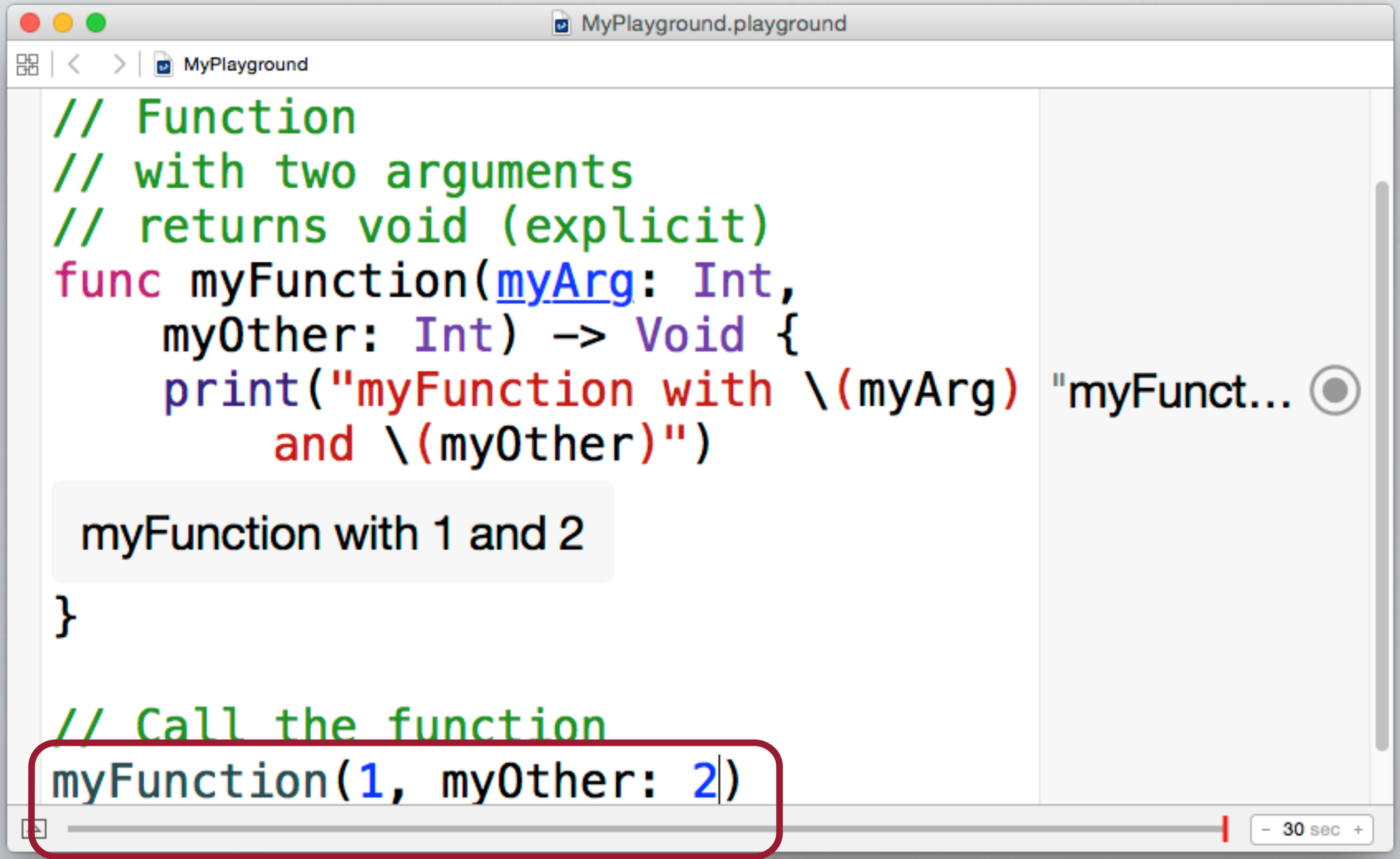
"myFunction"

30 sec

FUNCTIONS


MyPlayground.playground
MyPlayground
// Function
// with one argument
// returns void (explicit)
func myFunction(myArg: Int) -> Void
{
print("myFunction with \
(myArg)")
myFunction with 1
}
// Call the function
myFunction(1)
"myFunct...
30 sec

FUNCTIONS


MyPlayground.playground
MyPlayground
// Function
// with two arguments
// returns void (explicit)
func myFunction(myArg: Int,
myOther: Int) -> Void {
print("myFunction with \(myArg)
and \(myOther)")
myFunction with 1 and 2
}
// Call the function
myFunction(1, myOther: 2)
"myFunct...
30 sec

FUNCTIONS

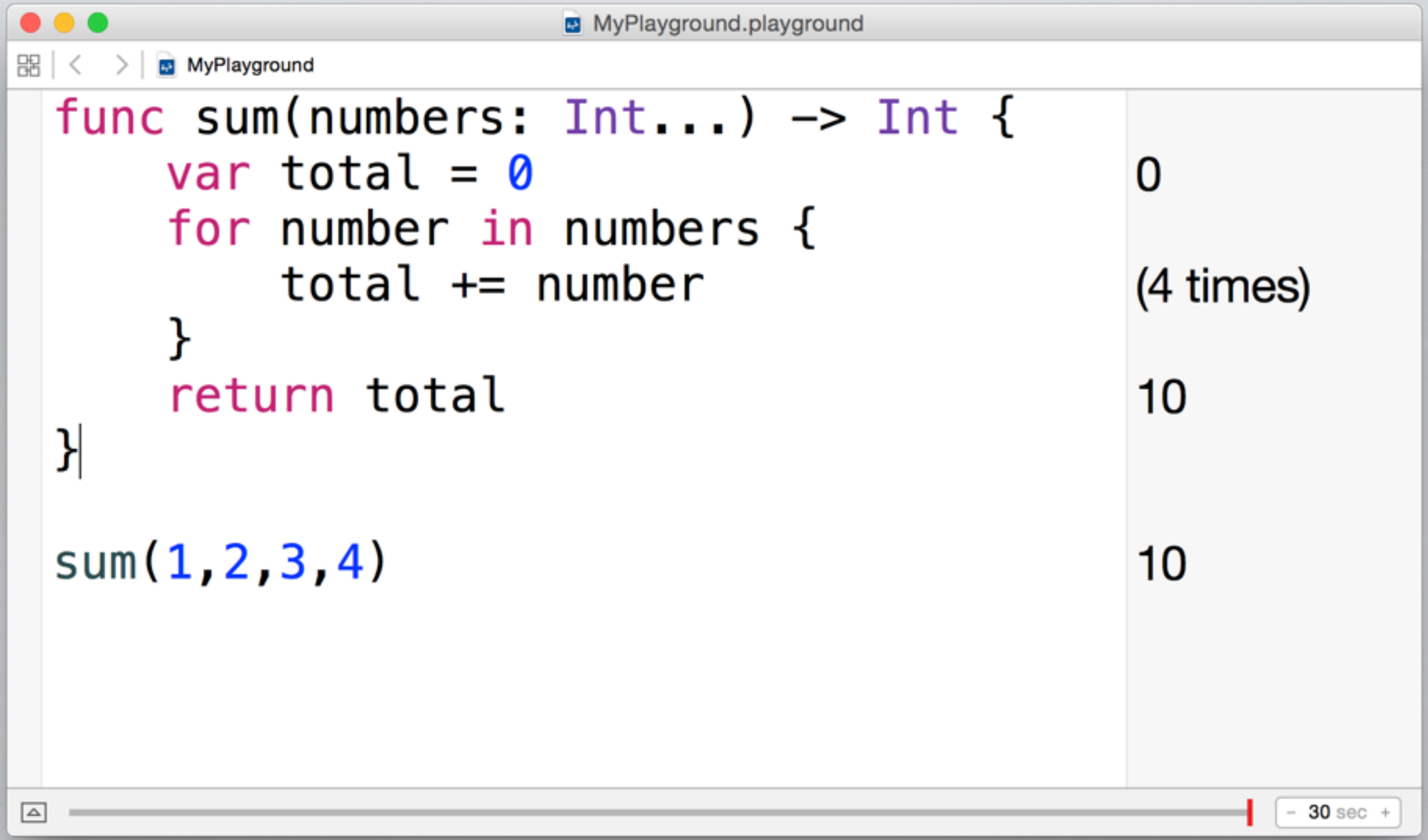
MyPlayground.playground

MyPlayground

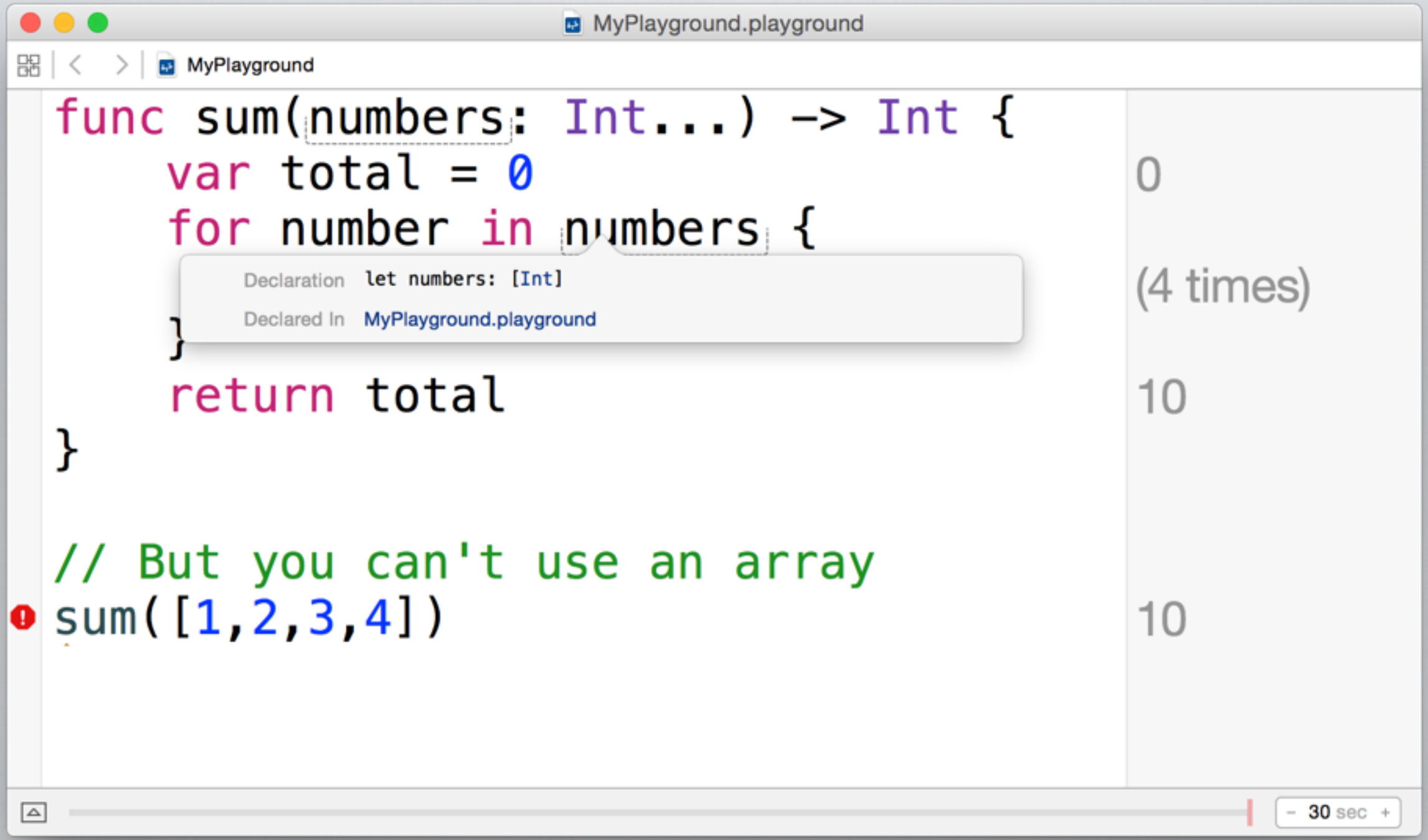
```
import UIKit

// Function
// with two arguments
// returns Int
func myFunction(myArg: Int,
    myOther: Int) -> Int {
    return myArg + myOther                3
}

// Call the function
myFunction(1, myOther: 2)                 3
```

30 sec

FUNCTIONS


MyPlayground.playground
MyPlayground
func sum(numbers: Int...) -> Int {
var total = 0
for number in numbers {
total += number
}
return total
}
sum(1,2,3,4)
0
(4 times)
10
10
30 sec

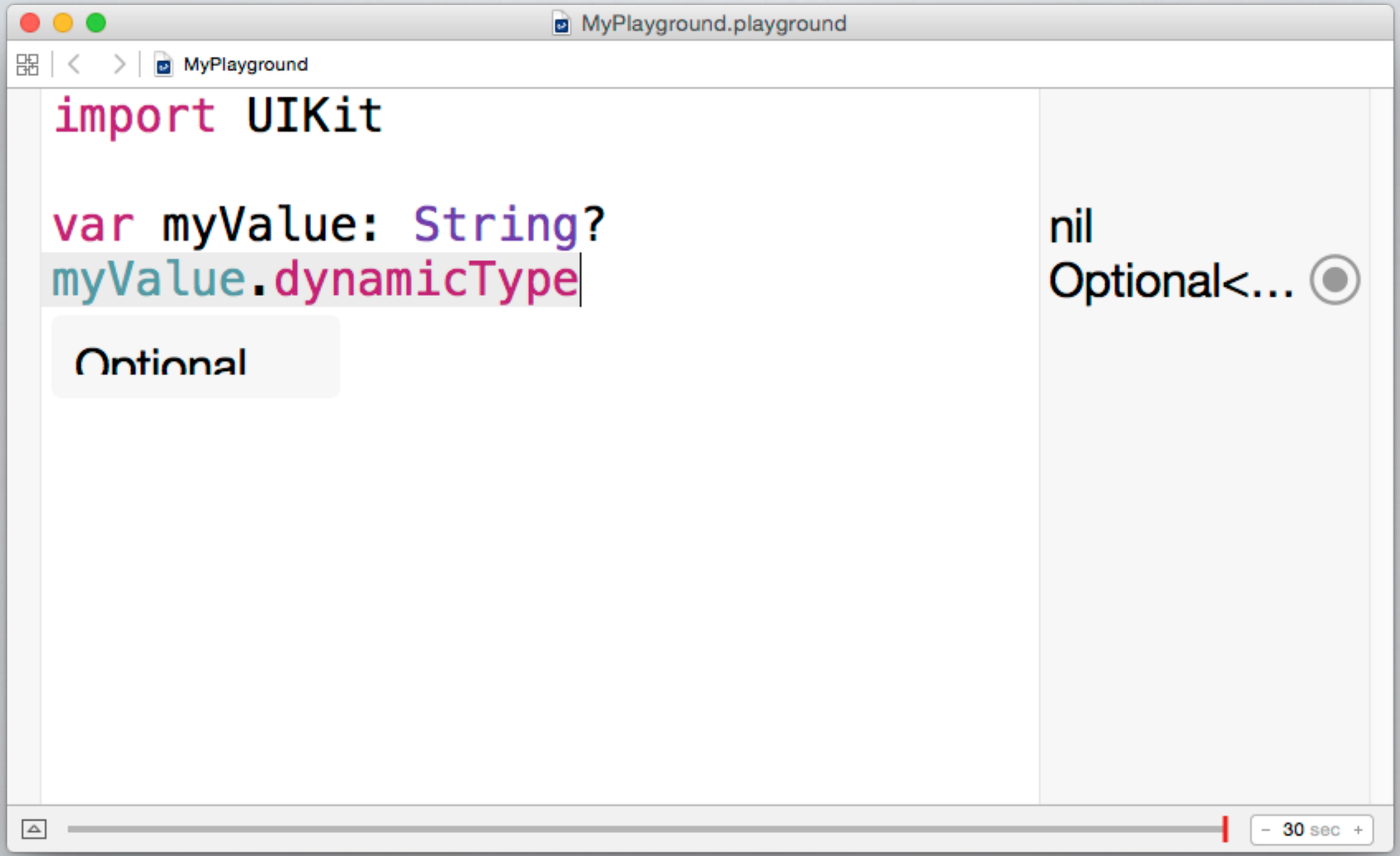
FUNCTIONS


MyPlayground.playground
MyPlayground
func sum(numbers: Int...) -> Int {
var total = 0
for number in numbers {
Declaration let numbers: [Int]
Declared In MyPlayground.playground
}
return total
}
// But you can't use an array
sum([1,2,3,4])
0
(4 times)
10
10
30 sec

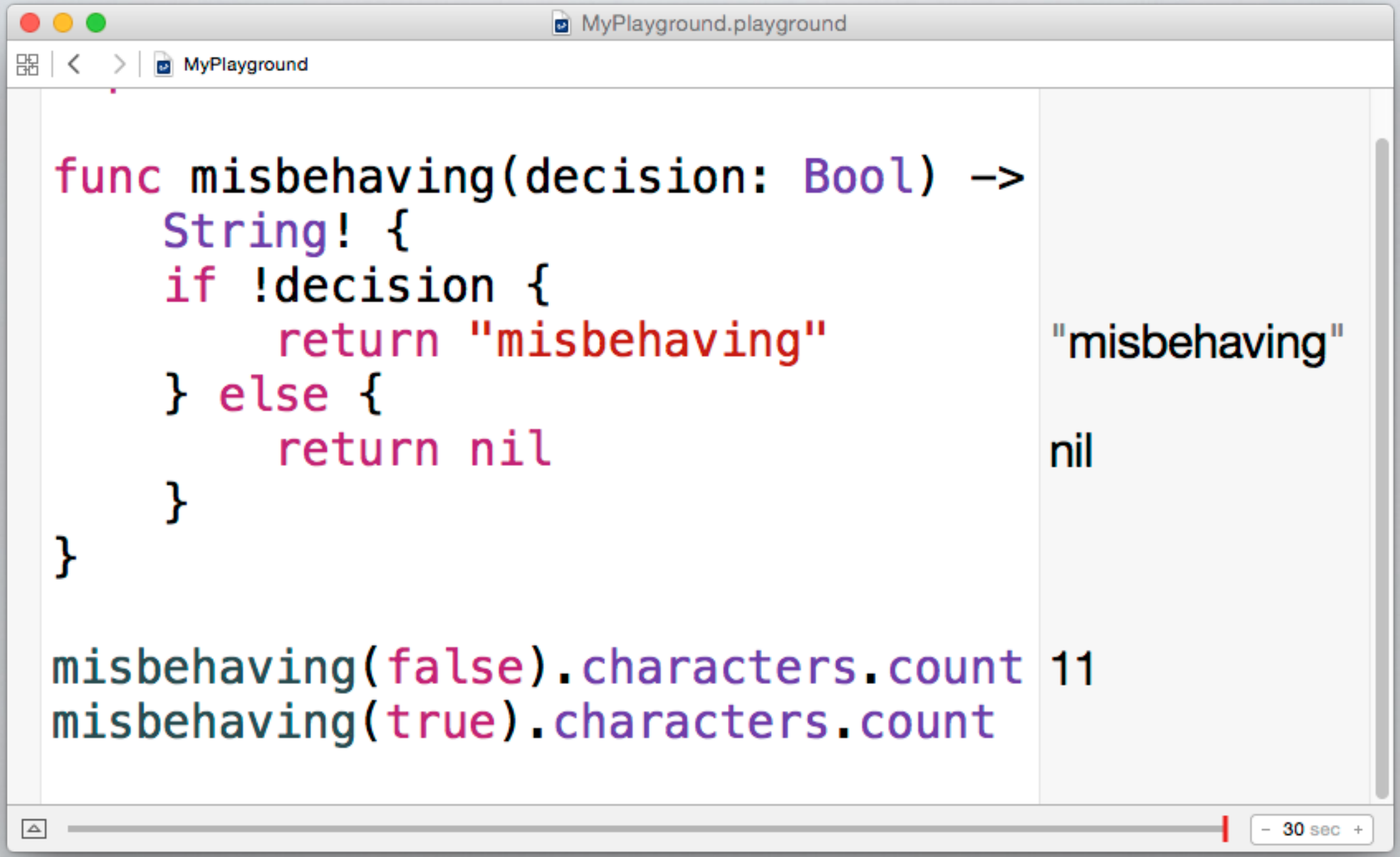
FUNCTIONS

- Other features not covered
  - default parameters
  - constant and variable parameters
  - inout parameters
  - throwing exceptions

OPTIONALS


MyPlayground.playground
MyPlayground
import UIKit
var myValue: String?
myValue.dynamicType
nil
Optional<...
Optional
30 sec

OPTIONALS


MyPlayground.playground
MyPlayground
func misbehaving(decision: Bool) ->
String! {
if !decision {
return "misbehaving"
} else {
return nil
}
}
misbehaving(false).characters.count
misbehaving(true).characters.count
"misbehaving"
nil
11
30 sec

OPTIONALS

MyPlayground.playground

MyPlayground

```
func misbehaving(decision: Bool) ->
    String! {
    if !decision {
        return "misbehaving"
    } else {
        return nil
    }
}

misbehaving(false).characters.count
misbehaving(true).characters.count
```

"misbehaving"

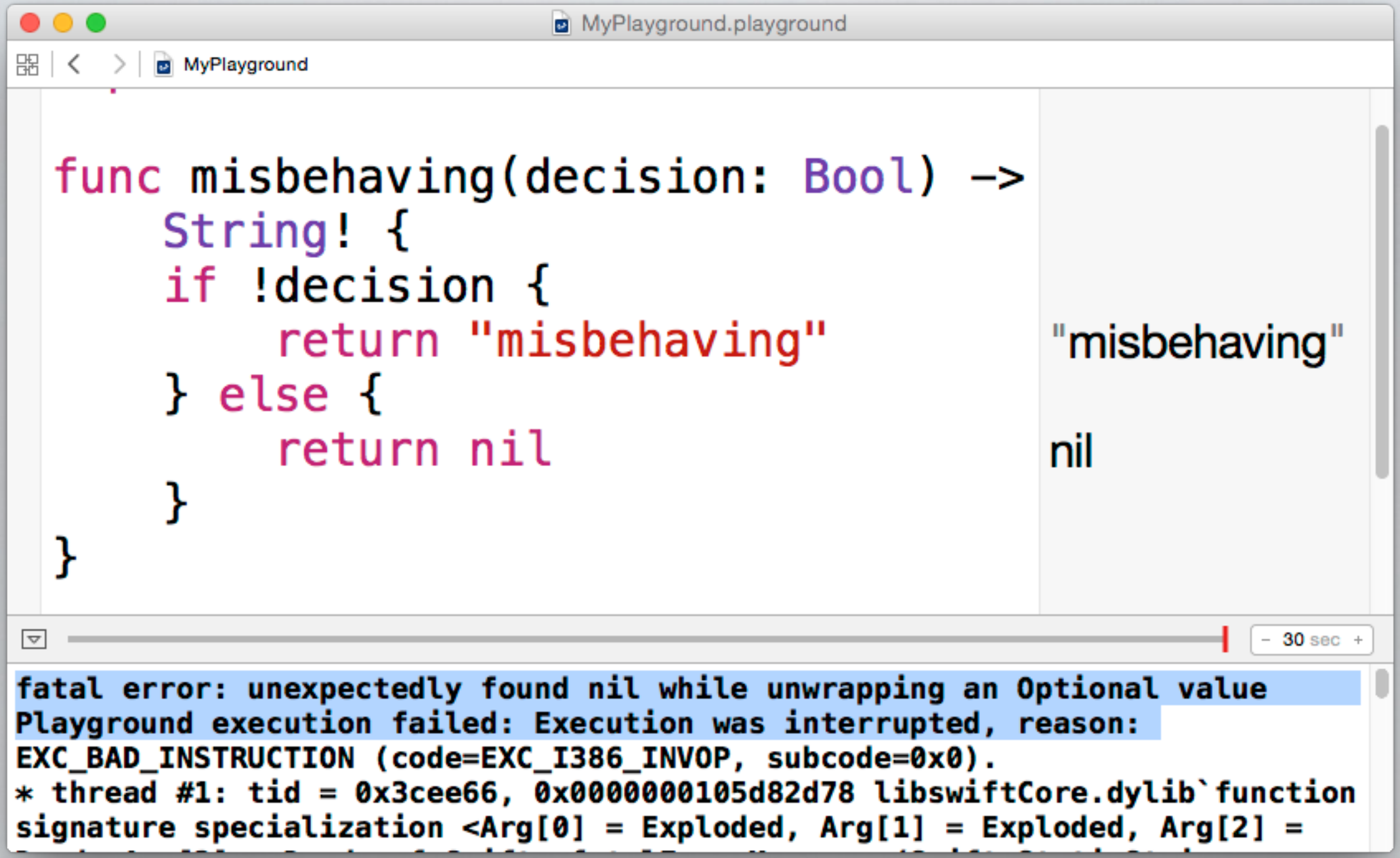
nil

11

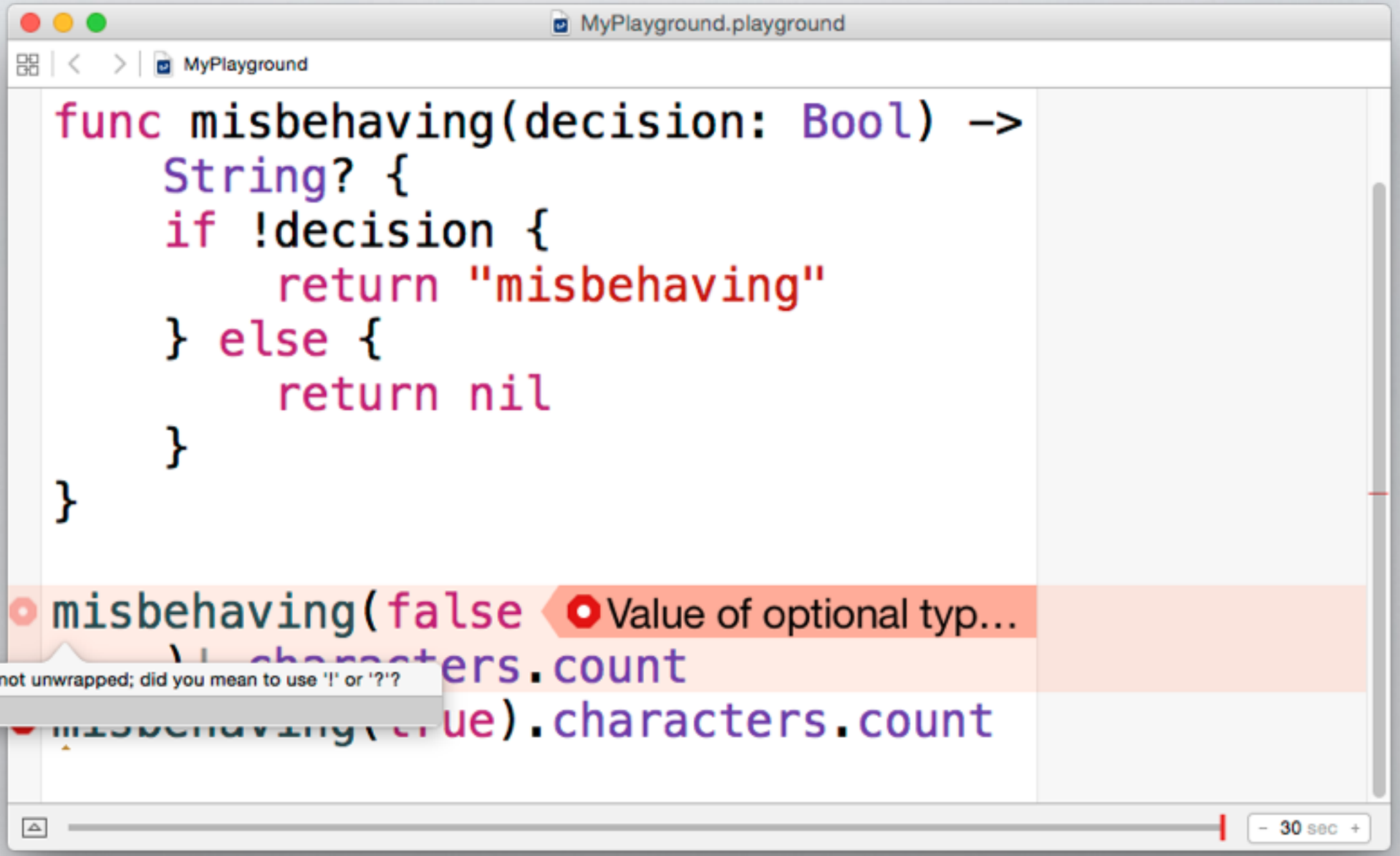
30 sec

What happens here?

OPTIONALS


MyPlayground.playground
MyPlayground
func misbehaving(decision: Bool) ->
String! {
if !decision {
return "misbehaving"
} else {
return nil
}
}
"misbehaving"
nil
30 sec
fatal error: unexpectedly found nil while unwrapping an Optional value
Playground execution failed: Execution was interrupted, reason:
EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0).
* thread #1: tid = 0x3cee66, 0x0000000105d82d78 libswiftCore.dylib`function
signature specialization <Arg[0] = Exploded, Arg[1] = Exploded, Arg[2] =

OPTIONALS


MyPlayground.playground
MyPlayground
func misbehaving(decision: Bool) ->
String? {
if !decision {
return "misbehaving"
} else {
return nil
}
}
misbehaving(false
Value of optional typ…
ers.count
ue).characters.count
Value of optional type 'String?' not unwrapped; did you mean to use '!' or '?'?
Fix-it Insert "!"
30 sec

OPTIONALS

MyPlayground.playground

MyPlayground

```
func misbehaving(decision: Bool) ->
    String? {                                    
    if !decision {                               
        return "misbehaving"                     "misbehaving"
    } else {                                     
        return nil                               nil
    }                                            
}                                                
                                                 
misbehaving(false)?.characters.                  11
    count                                        
misbehaving(true)?.characters.count              nil
```

30 sec

OPTIONALS

MyPlayground.playground

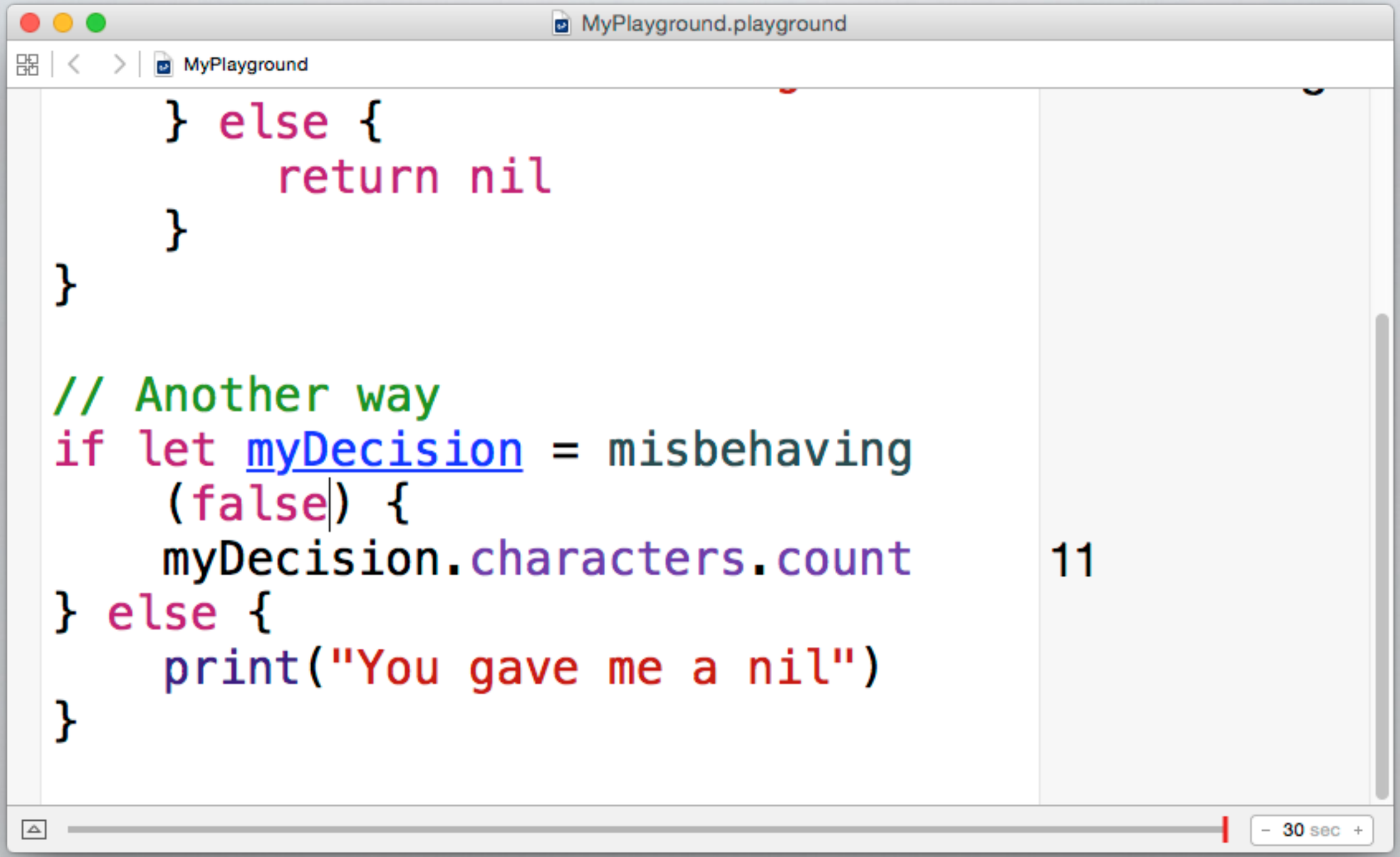
MyPlayground

```
func misbehaving(decision: Bool) ->
    String? {                                    
    if !decision {                               
        return "misbehaving"                     "misbehaving"
    } else {                                     
        return nil                               nil
    }                                            
}                                                
                                                 
misbehaving(false)?.characters.                  11
    count                                        
misbehaving(true)?.characters.count              nil
```

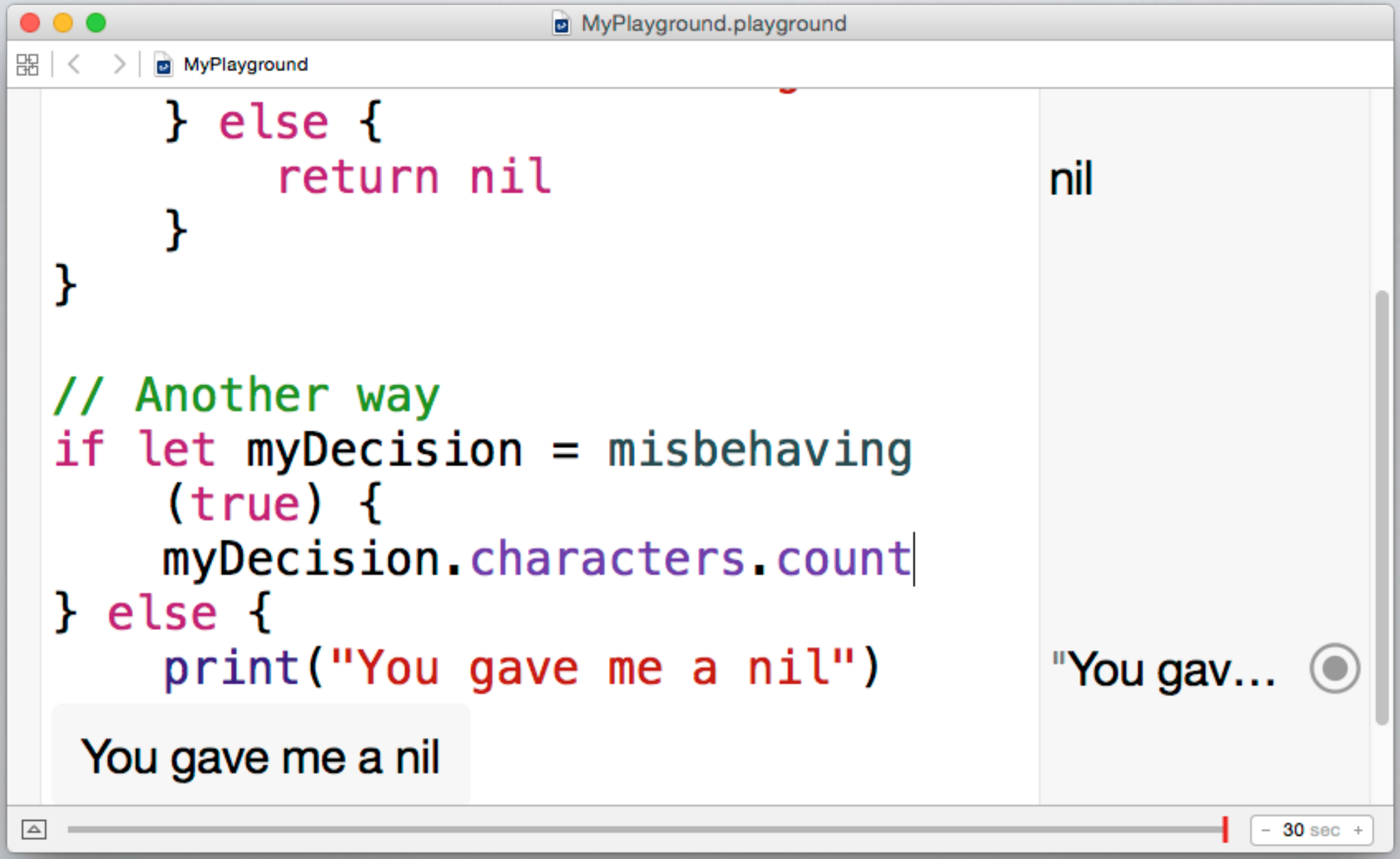
30 sec

If nil, don't call the rest of the methods after the '.'

OPTIONALS


MyPlayground.playground
MyPlayground
} else {
return nil
}
}
// Another way
if let myDecision = misbehaving
(false) {
myDecision.characters.count
11
} else {
print("You gave me a nil")
}
30 sec

OPTIONALS

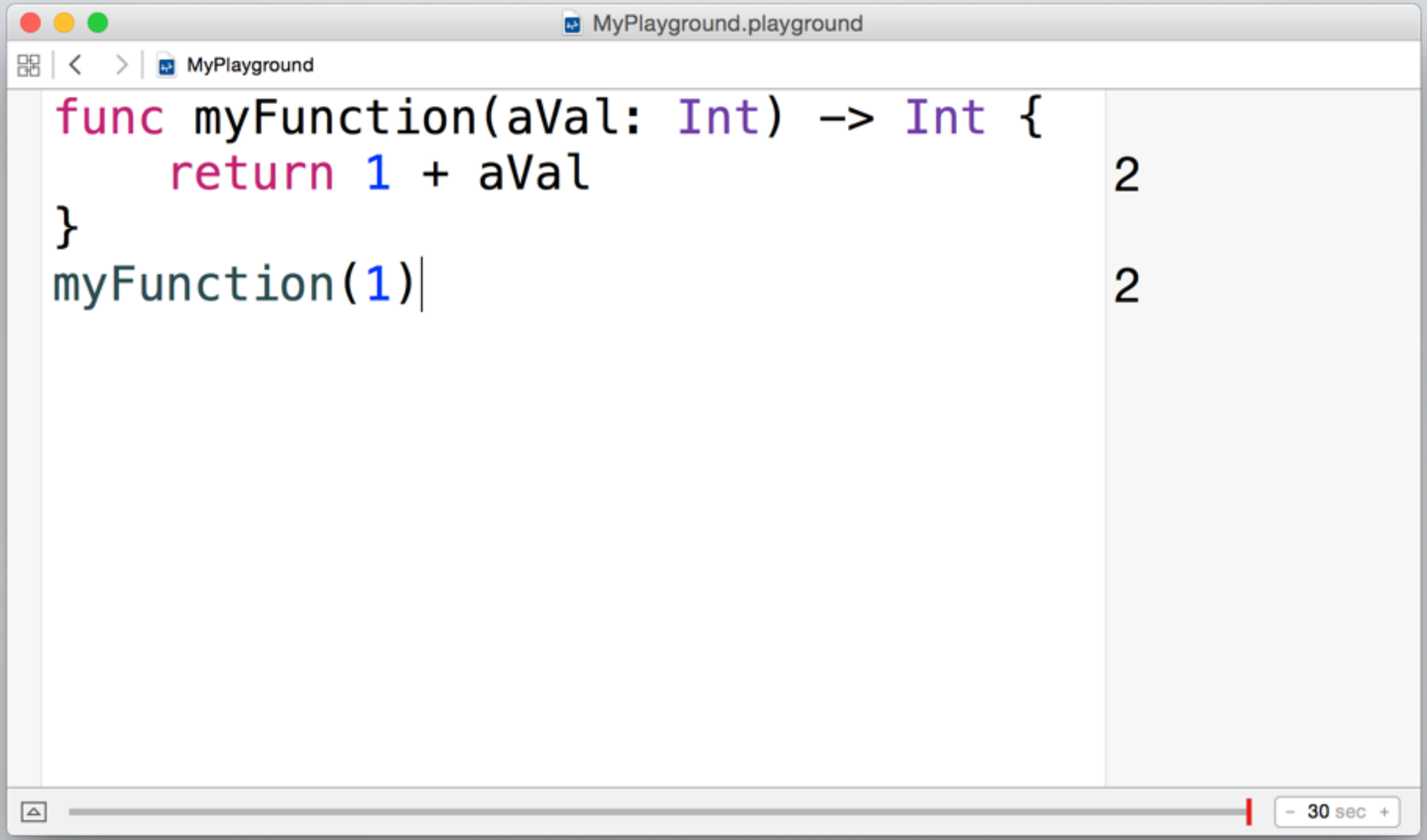

MyPlayground.playground
MyPlayground
} else {
return nil
}
}
// Another way
if let myDecision = misbehaving
(true) {
myDecision.characters.count
} else {
print("You gave me a nil")
nil
"You gav...
You gave me a nil
30 sec

CLOSURES

Closures are self-contained blocks of functionality that can be passed around and used in your code.

Closures can capture and store references to any constants and variables from the context in which they are defined.

CLOSURES


MyPlayground.playground
MyPlayground
func myFunction(aVal: Int) -> Int {
return 1 + aVal
}
myFunction(1)
2
2
30 sec

CLOSURES

MyPlayground.playground

MyPlayground

```
var myFunction = { (aVal: Int) ->          Int -> Int
    Int in
    return 1 + aVal                        2
}

myFunction(1)                              2
myFunction.dynamicType                     (Int -> Int).Type
```

30 sec

CLOSURES

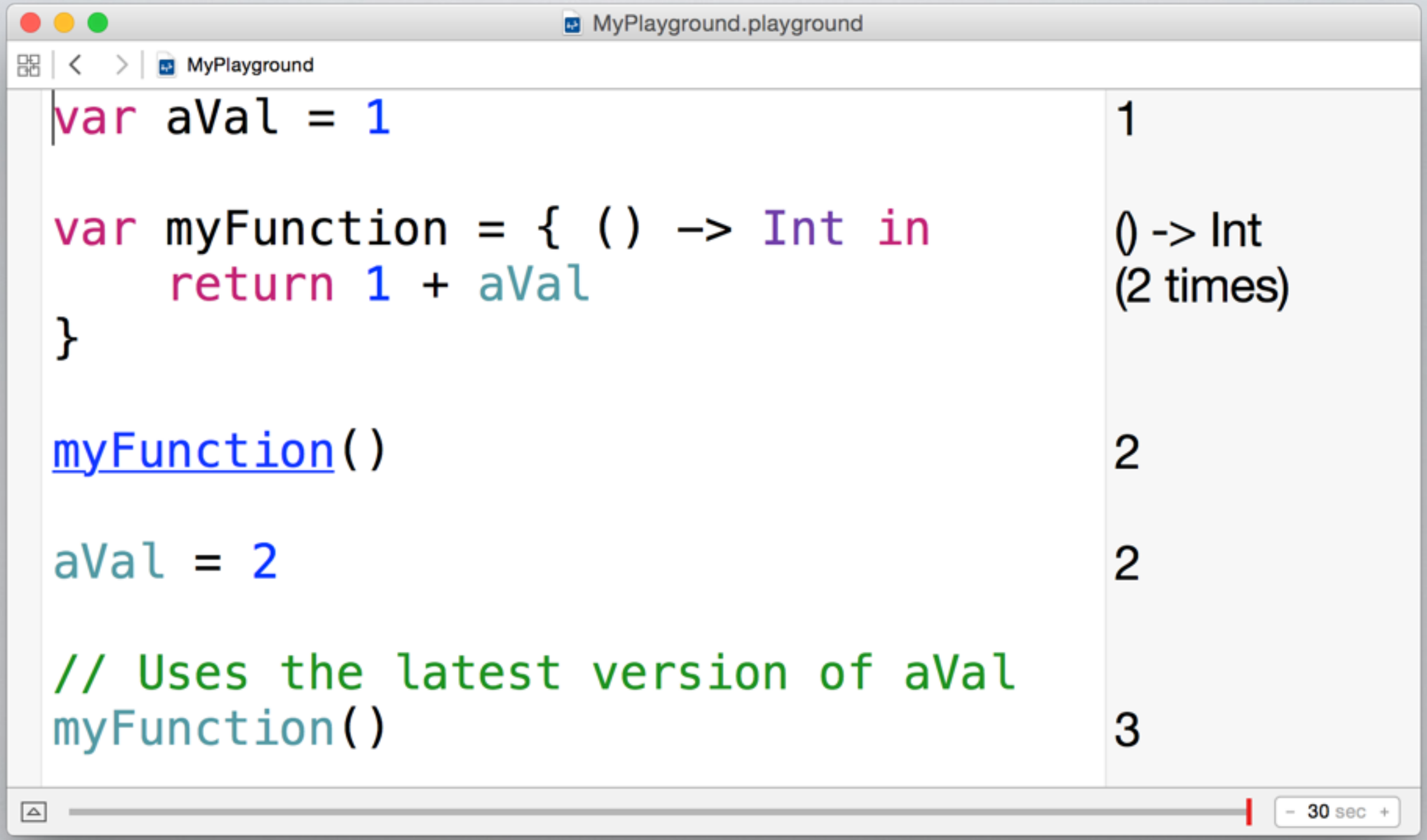
MyPlayground.playground

MyPlayground

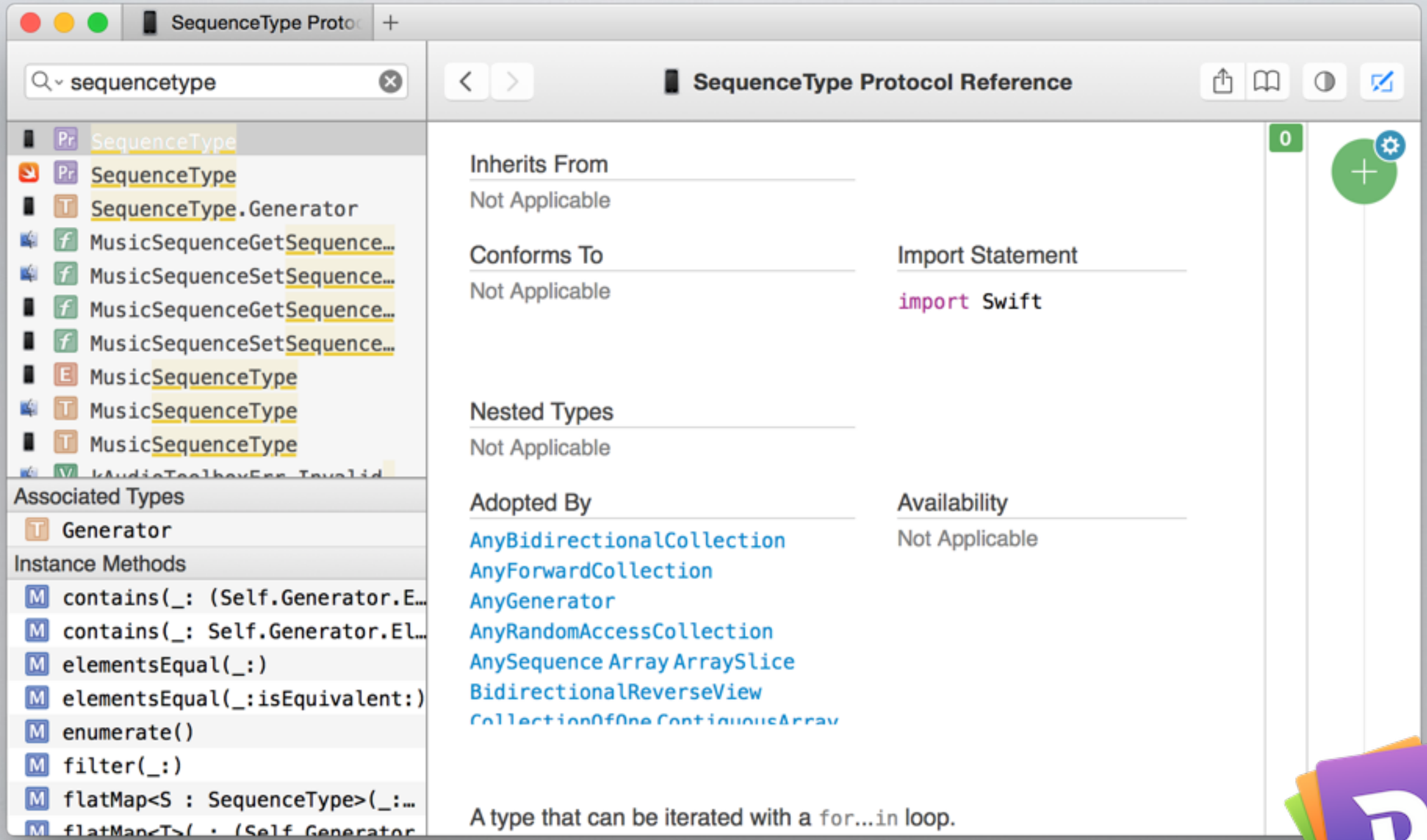
```
// This is outside the function
// Not passed as an argument
var aVal = 1                                    1

var myFunction = { () -> Int in                 () -> Int
    return 1 + aVal                             2
}

myFunction()                                    2
```

30 sec

CLOSURES


MyPlayground.playground
MyPlayground
var aVal = 1
1
var myFunction = { () -> Int in
return 1 + aVal
() -> Int
(2 times)
}
myFunction()
2
aVal = 2
2
// Uses the latest version of aVal
myFunction()
3
30 sec

CLOSURES


SequenceType Proto
sequencetype
SequenceType Protocol Reference
SequenceType
SequenceType
SequenceType.Generator
MusicSequenceGetSequence…
MusicSequenceSetSequence…
MusicSequenceGetSequence…
MusicSequenceSetSequence…
MusicSequenceType
MusicSequenceType
MusicSequenceType
Associated Types
Generator
Instance Methods
contains(_: (Self.Generator.E…
contains(_: Self.Generator.El…
elementsEqual(_:)
elementsEqual(_:isEquivalent:)
enumerate()
filter(_:)
flatMap<S : SequenceType>(_:…
Inherits From
Not Applicable
Conforms To
Not Applicable
Import Statement
import Swift
Nested Types
Not Applicable
Adopted By
AnyBidirectionalCollection
AnyForwardCollection
AnyGenerator
AnyRandomAccessCollection
AnySequence Array ArraySlice
BidirectionalReverseView
Availability
Not Applicable
A type that can be iterated with a for...in loop.
0

CLOSURES

MyPlayground.playground

MyPlayground

```
let myFruits = ["apples", "pears",
    "oranges"]
myFruits.map { (fruit: String) ->
    String in
    fruit.uppercaseString
}
```

["apples", "...

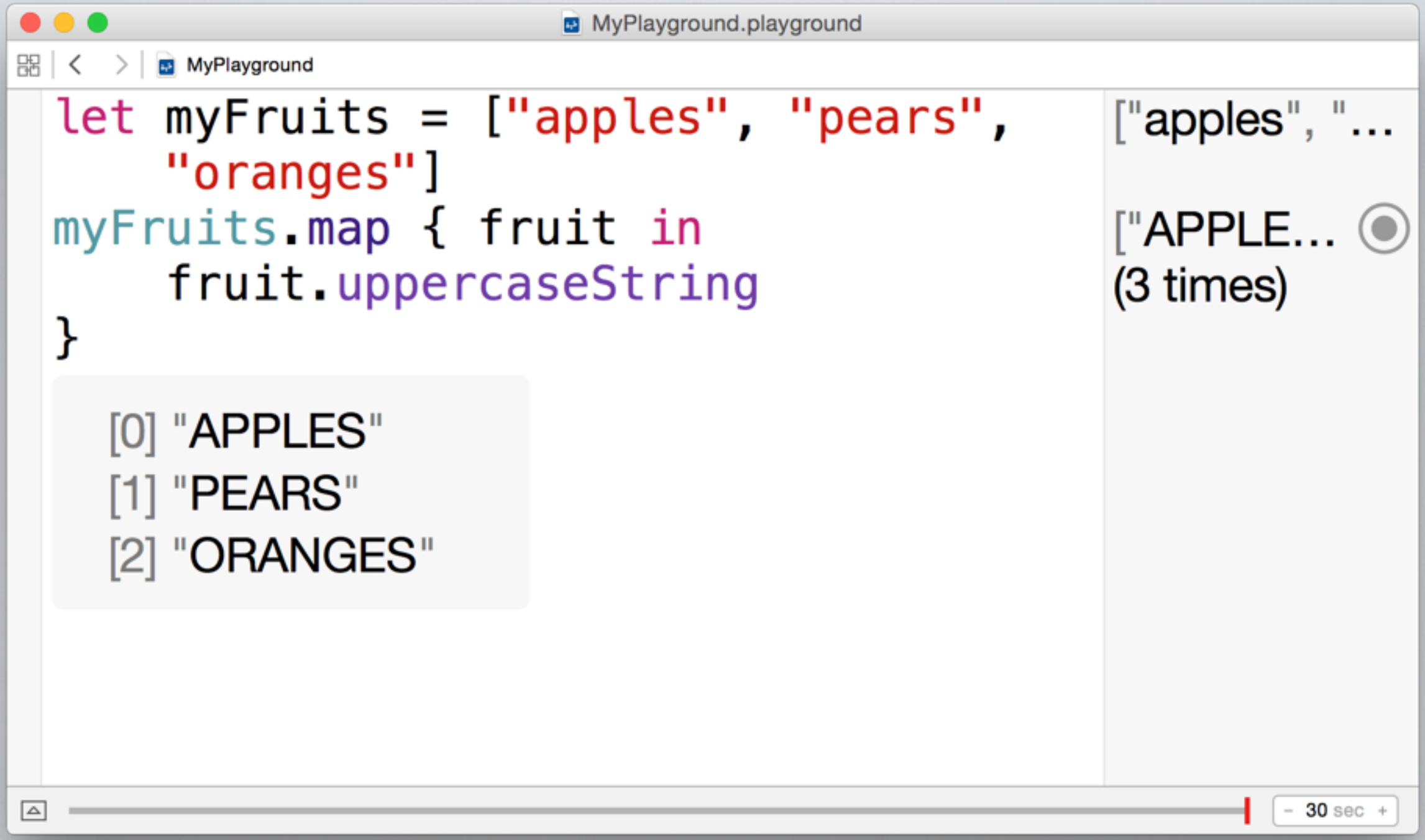
["APPLE...

(3 times)

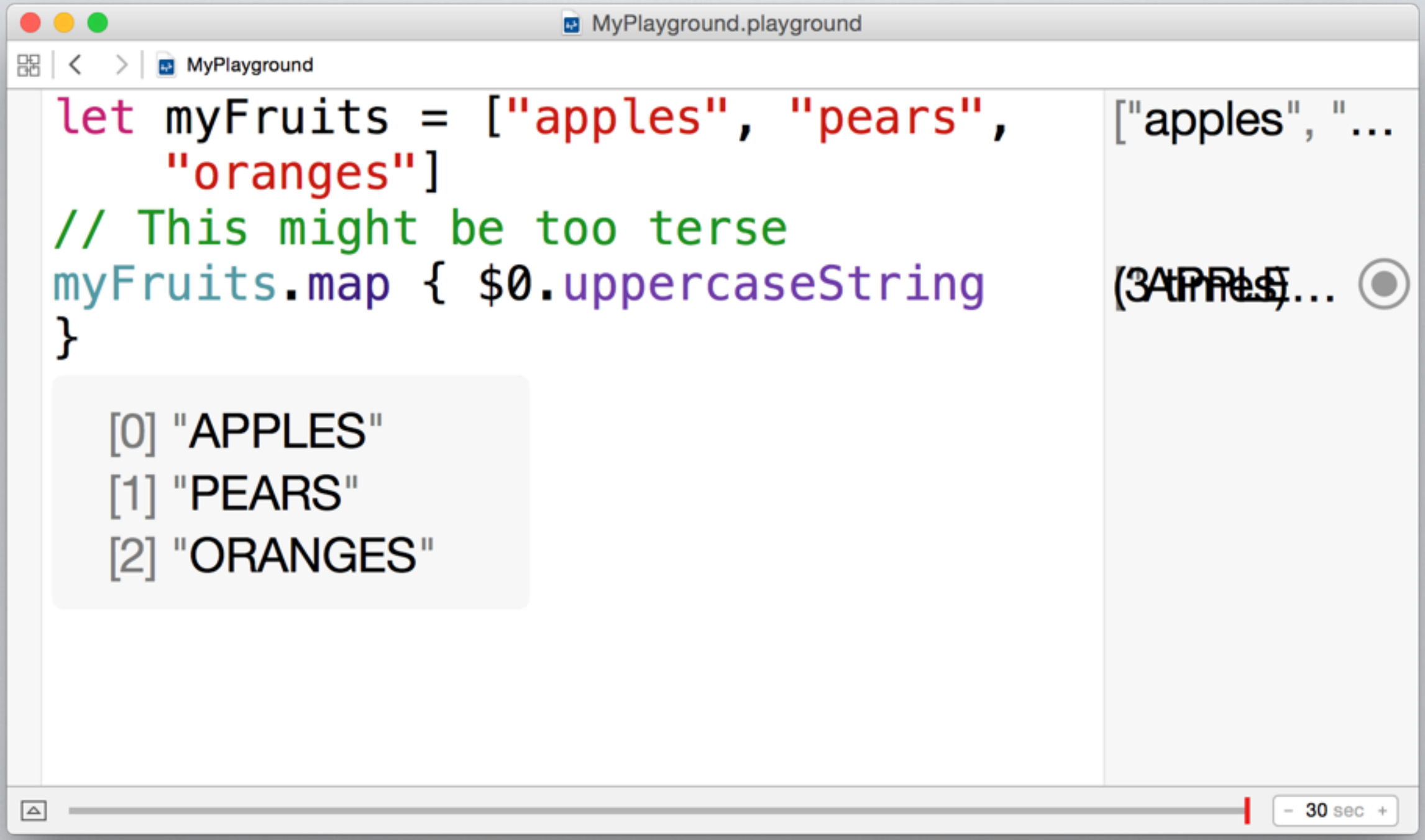
```
[0] "APPLES"
[1] "PEARS"
[2] "ORANGES"
```

30 sec

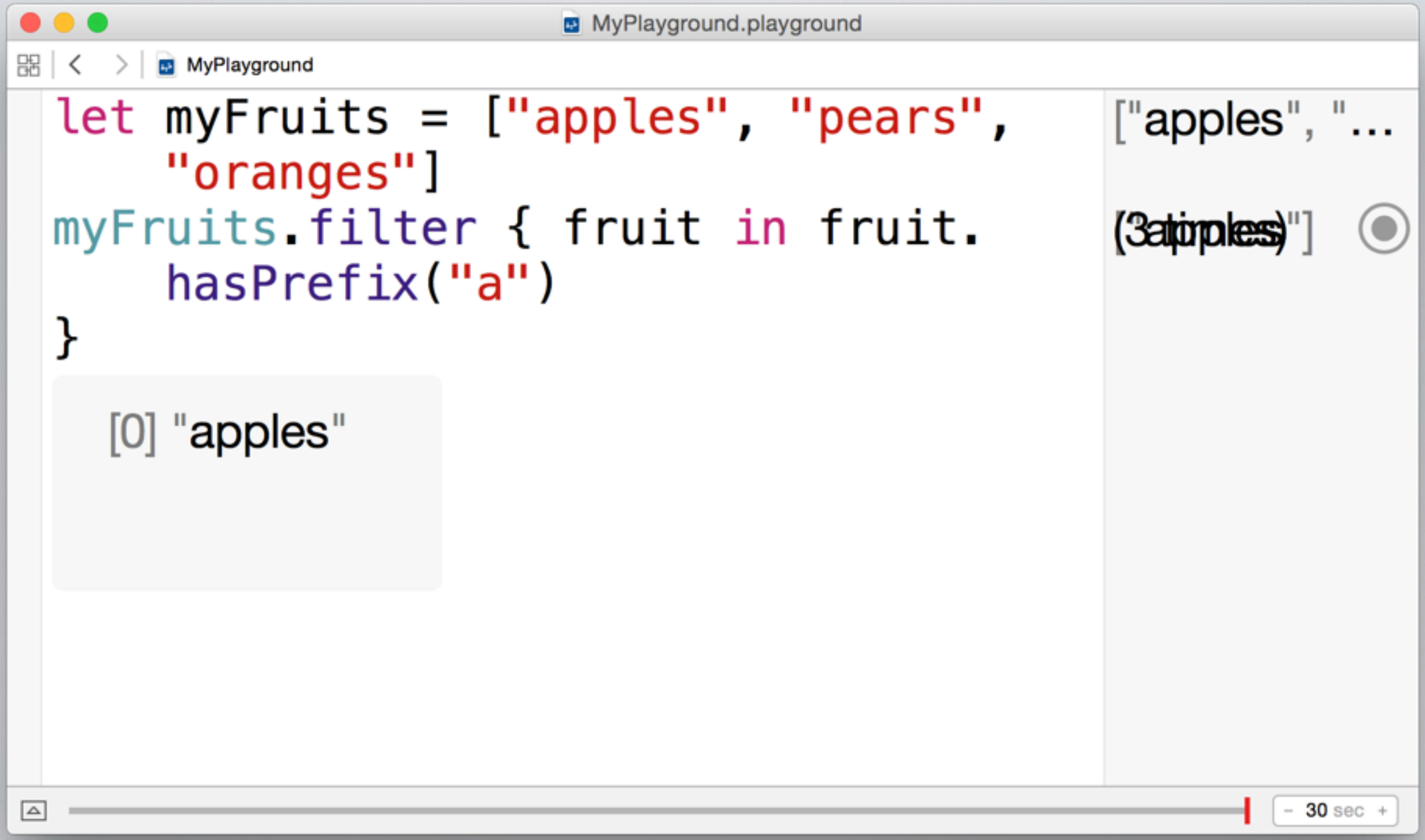
CLOSURES


MyPlayground.playground
MyPlayground
let myFruits = ["apples", "pears",
"oranges"]
myFruits.map { fruit in
fruit.uppercaseString
}
["apples", "...
["APPLE...
(3 times)
[0] "APPLES"
[1] "PEARS"
[2] "ORANGES"
30 sec

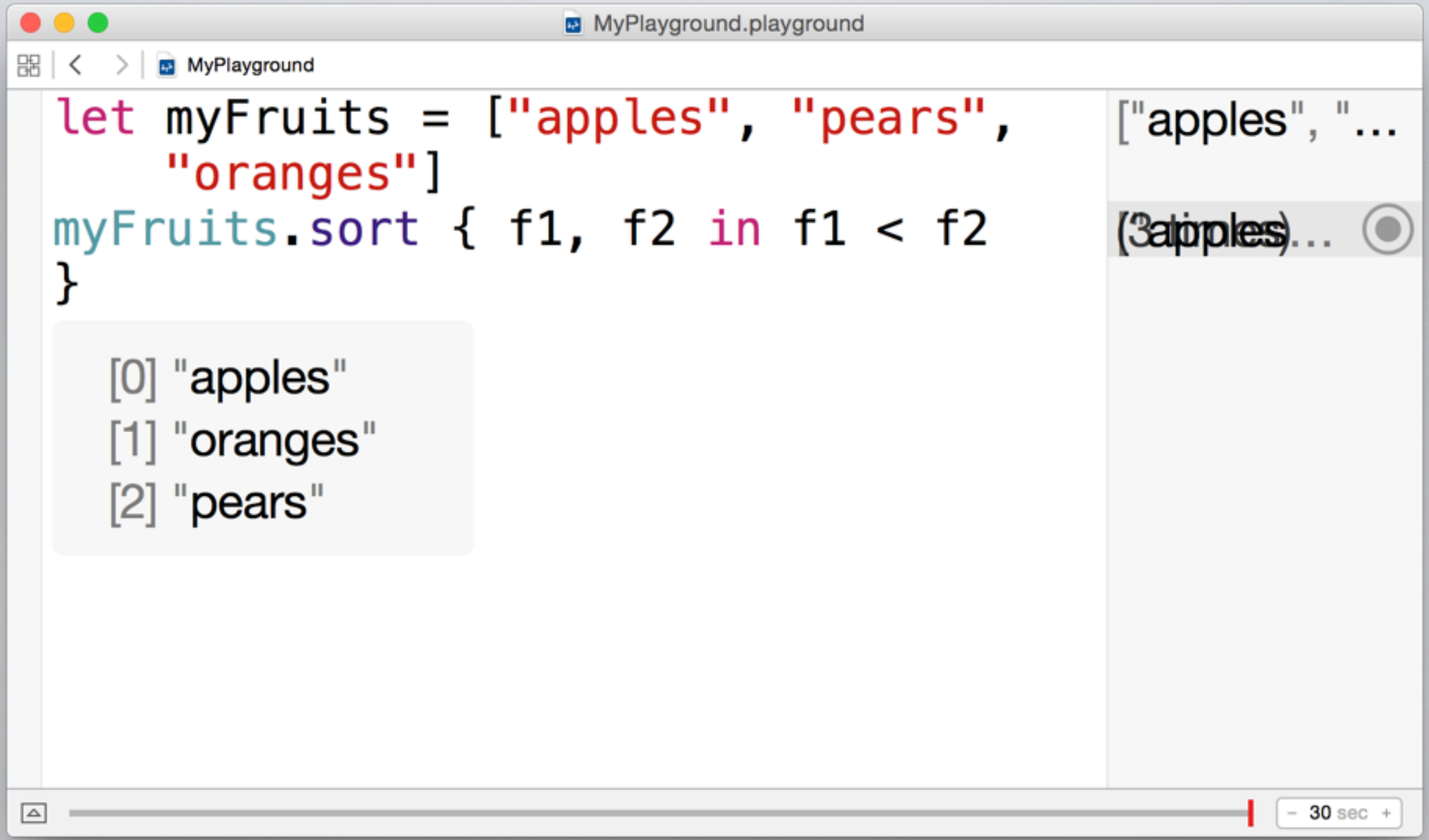
CLOSURES


MyPlayground.playground
MyPlayground
let myFruits = ["apples", "pears",
"oranges"]
// This might be too terse
myFruits.map { $0.uppercaseString
}
["apples", "...
[0] "APPLES"
[1] "PEARS"
[2] "ORANGES"
30 sec

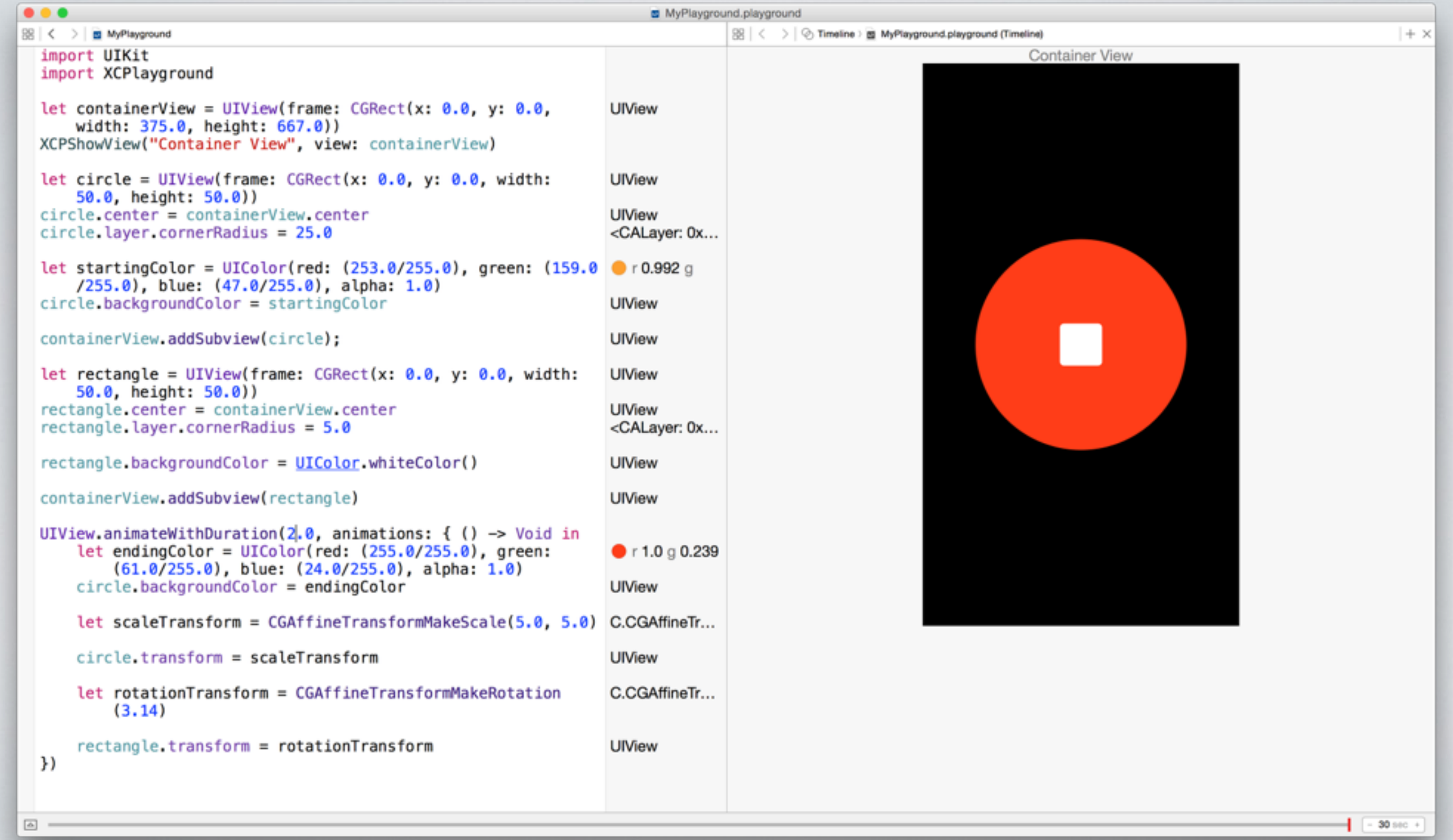
CLOSURES


MyPlayground.playground
MyPlayground
let myFruits = ["apples", "pears",
"oranges"]
myFruits.filter { fruit in fruit.
hasPrefix("a")
}
["apples", "...
[0] "apples"
30 sec

CLOSURES


MyPlayground.playground
MyPlayground
let myFruits = ["apples", "pears",
"oranges"]
myFruits.sort { f1, f2 in f1 < f2
}
["apples", "...
[0] "apples"
[1] "oranges"
[2] "pears"
30 sec

CLOSURES


MyPlayground.playground
MyPlayground
Timeline
MyPlayground.playground (Timeline)
Container View
import UIKit
import XCPlayground
let containerView = UIView(frame: CGRect(x: 0.0, y: 0.0, width: 375.0, height: 667.0))
XCPShowView("Container View", view: containerView)
let circle = UIView(frame: CGRect(x: 0.0, y: 0.0, width: 50.0, height: 50.0))
circle.center = containerView.center
circle.layer.cornerRadius = 25.0
let startingColor = UIColor(red: (253.0/255.0), green: (159.0/255.0), blue: (47.0/255.0), alpha: 1.0)
circle.backgroundColor = startingColor
containerView.addSubview(circle);
let rectangle = UIView(frame: CGRect(x: 0.0, y: 0.0, width: 50.0, height: 50.0))
rectangle.center = containerView.center
rectangle.layer.cornerRadius = 5.0
rectangle.backgroundColor = UIColor.whiteColor()
containerView.addSubview(rectangle)
UIView.animateWithDuration(2.0, animations: { () -> Void in
let endingColor = UIColor(red: (255.0/255.0), green: (61.0/255.0), blue: (24.0/255.0), alpha: 1.0)
circle.backgroundColor = endingColor
let scaleTransform = CGAffineTransformMakeScale(5.0, 5.0)
circle.transform = scaleTransform
let rotationTransform = CGAffineTransformMakeRotation(3.14)
rectangle.transform = rotationTransform
})
UIView
UIView
UIView <CALayer: 0x...
r 0.992 g
UIView
UIView
UIView
UIView <CALayer: 0x...
UIView
r 1.0 g 0.239
UIView
C.CGAffineTr...
UIView
C.CGAffineTr...
UIView

CLASSES

MyPlayground.playground

MyPlayground

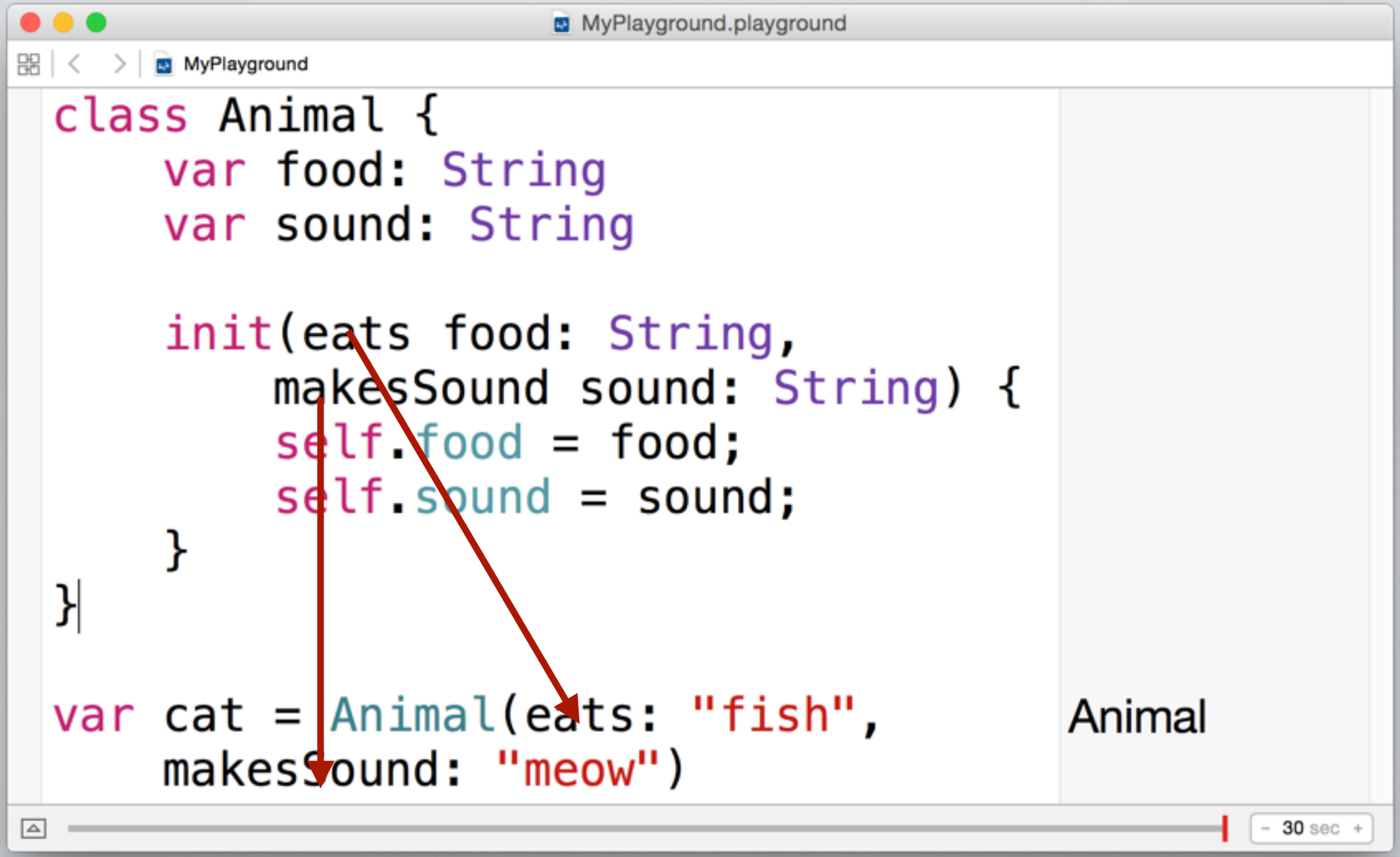
```
// Why does this give a compile
    error?
class Animal {
    var food: String
    var sound: String
}
```

30 sec

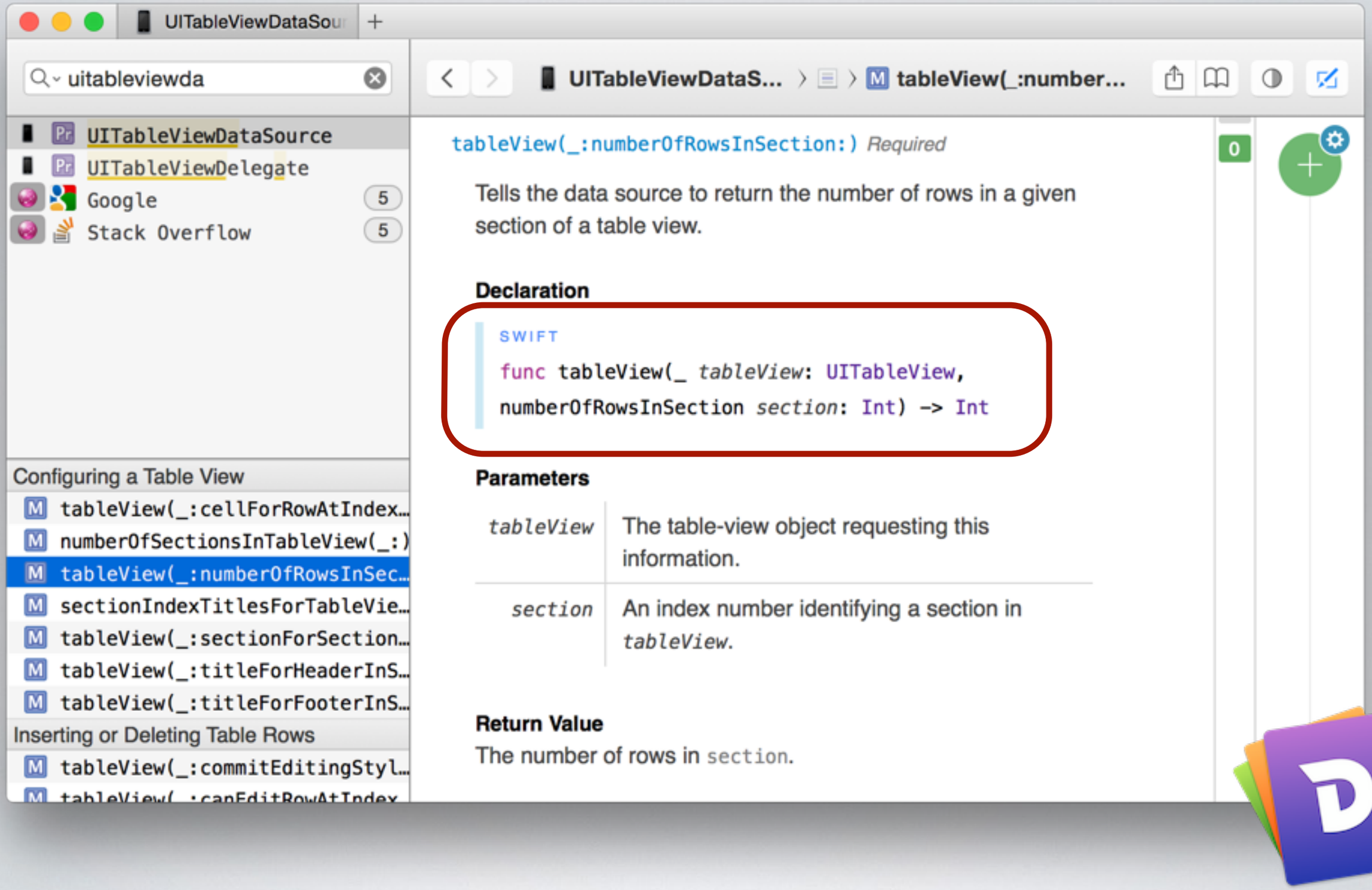
CLASSES


MyPlayground.playground
MyPlayground
class Animal {
var food: String
var sound: String
init(food: String, sound:
String) {
self.food = food;
self.sound = sound;
}
}
var cat = Animal(food: "fish",
sound: "meow")
Animal
30 sec

CLASSES


MyPlayground.playground
MyPlayground
class Animal {
var food: String
var sound: String
init(eats food: String,
makesSound sound: String) {
self.food = food;
self.sound = sound;
}
}
var cat = Animal(eats: "fish",
makesSound: "meow")
Animal
30 sec

METHODS


UITableViewDataSou
uitableviewda
UITableViewDataSource
UITableViewDelegate
Google 5
Stack Overflow 5
UITableViewDataS...
tableView(_:number...
tableView(_:numberOfRowsInSection:) Required
Tells the data source to return the number of rows in a given section of a table view.
Declaration
SWIFT
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
Parameters
tableView
The table-view object requesting this information.
section
An index number identifying a section in tableView.
Return Value
The number of rows in section.
0
Configuring a Table View
tableView(_:cellForRowAtIndex...
numberOfSectionsInTableView(_:)
tableView(_:numberOfRowsInSec...
sectionIndexTitlesForTableVie...
tableView(_:sectionForSection...
tableView(_:titleForHeaderInS...
tableView(_:titleForFooterInS...
Inserting or Deleting Table Rows
tableView(_:commitEditingStyl...

CLASSES

MyPlayground.playground

MyPlayground

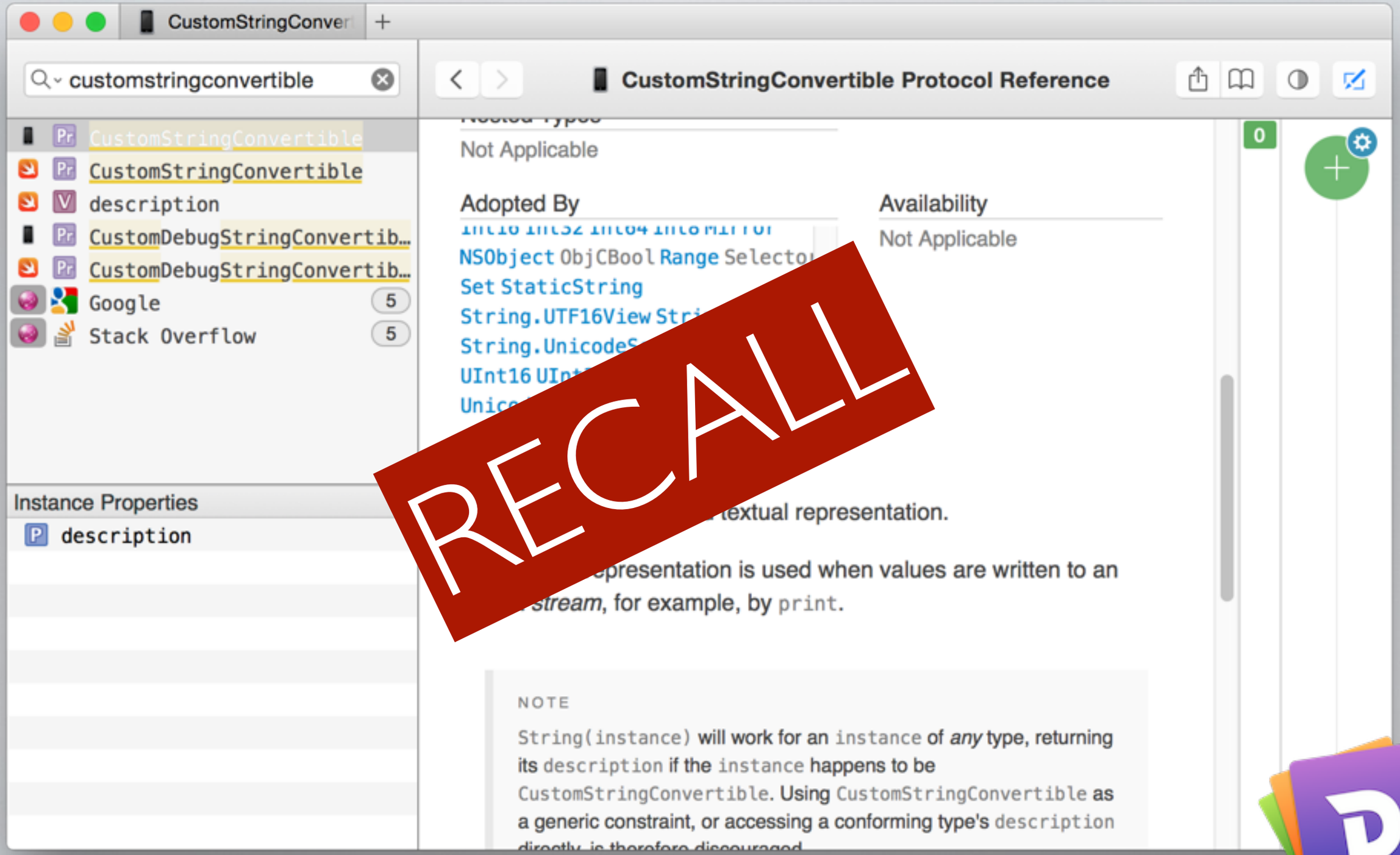
```
class Animal {
    var food: String
    var sound: String

    init(eats food: String,
         makesSound sound: String) {
        self.food = food;
        self.sound = sound;
    }
}

var cat = Animal(eats: "fish",
    makesSound: "meow")
```

Animal

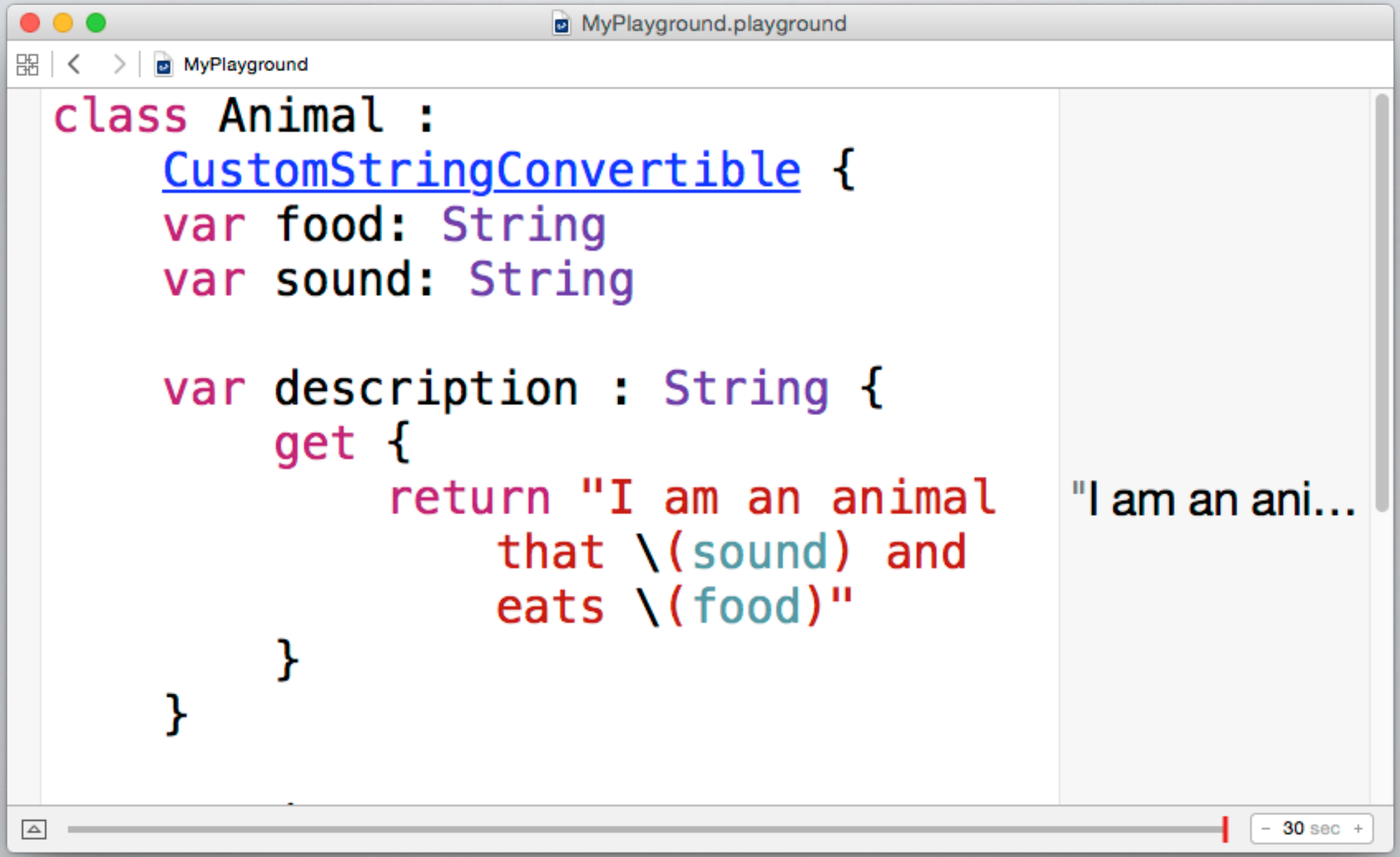
Why doesn't this print the variables?

30 sec

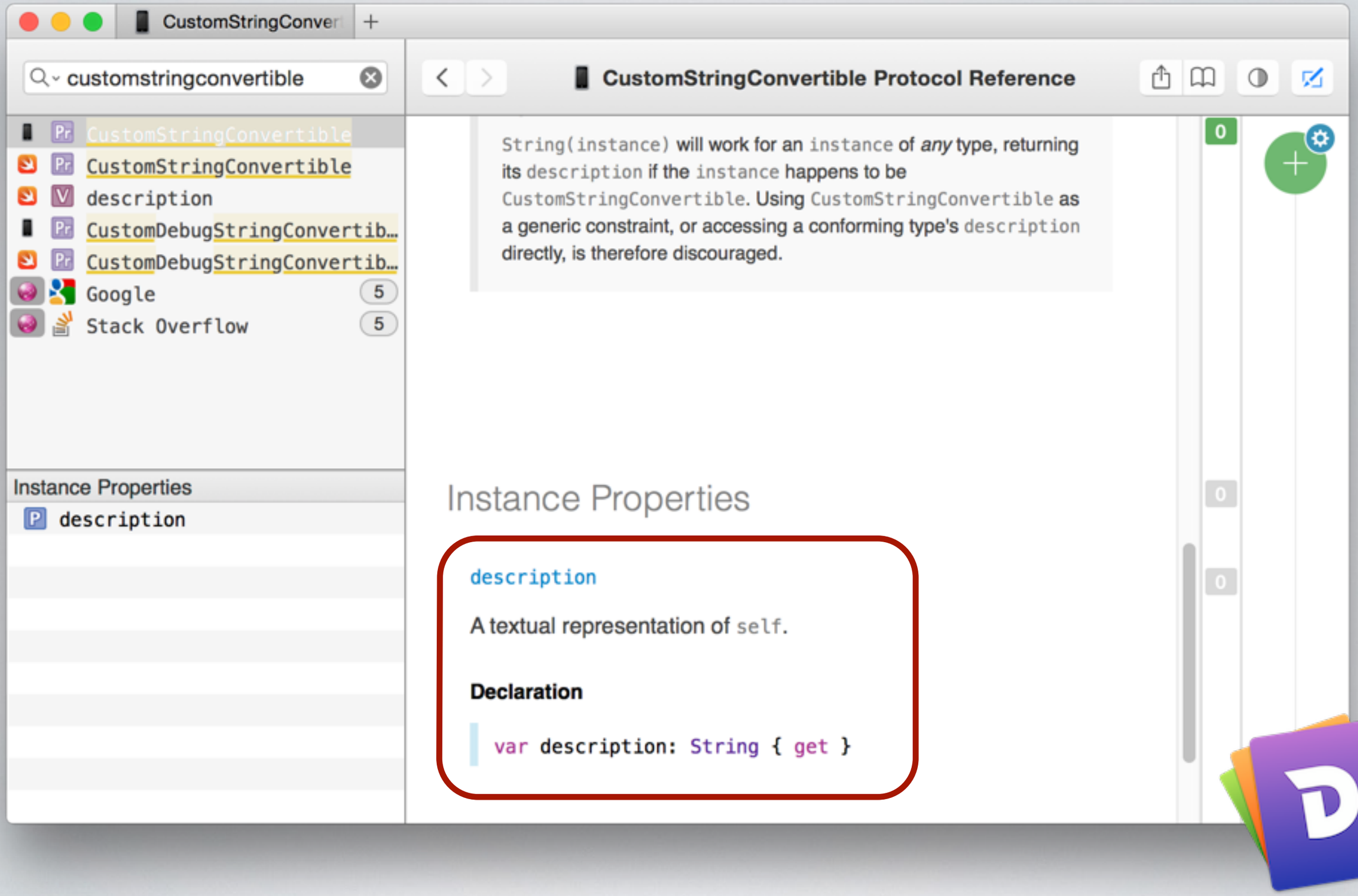
STRING INTERPOLATION



PROTOCOLS


MyPlayground.playground
MyPlayground
class Animal :
CustomStringConvertible {
var food: String
var sound: String
var description : String {
get {
return "I am an animal
that \(sound) and
eats \(food)"
}
}
"I am an ani...
30 sec

PROTOCOLS


CustomStringConver
customstringconvertible
CustomStringConvertible
CustomStringConvertible
description
CustomDebugStringConvertib…
CustomDebugStringConvertib…
Google
5
Stack Overflow
5
Instance Properties
description
CustomStringConvertible Protocol Reference
String(instance) will work for an instance of any type, returning its description if the instance happens to be CustomStringConvertible. Using CustomStringConvertible as a generic constraint, or accessing a conforming type's description directly, is therefore discouraged.
Instance Properties
description
A textual representation of self.
Declaration
var description: String { get }
0
0
0

PROTOCOLS

MyPlayground.playground

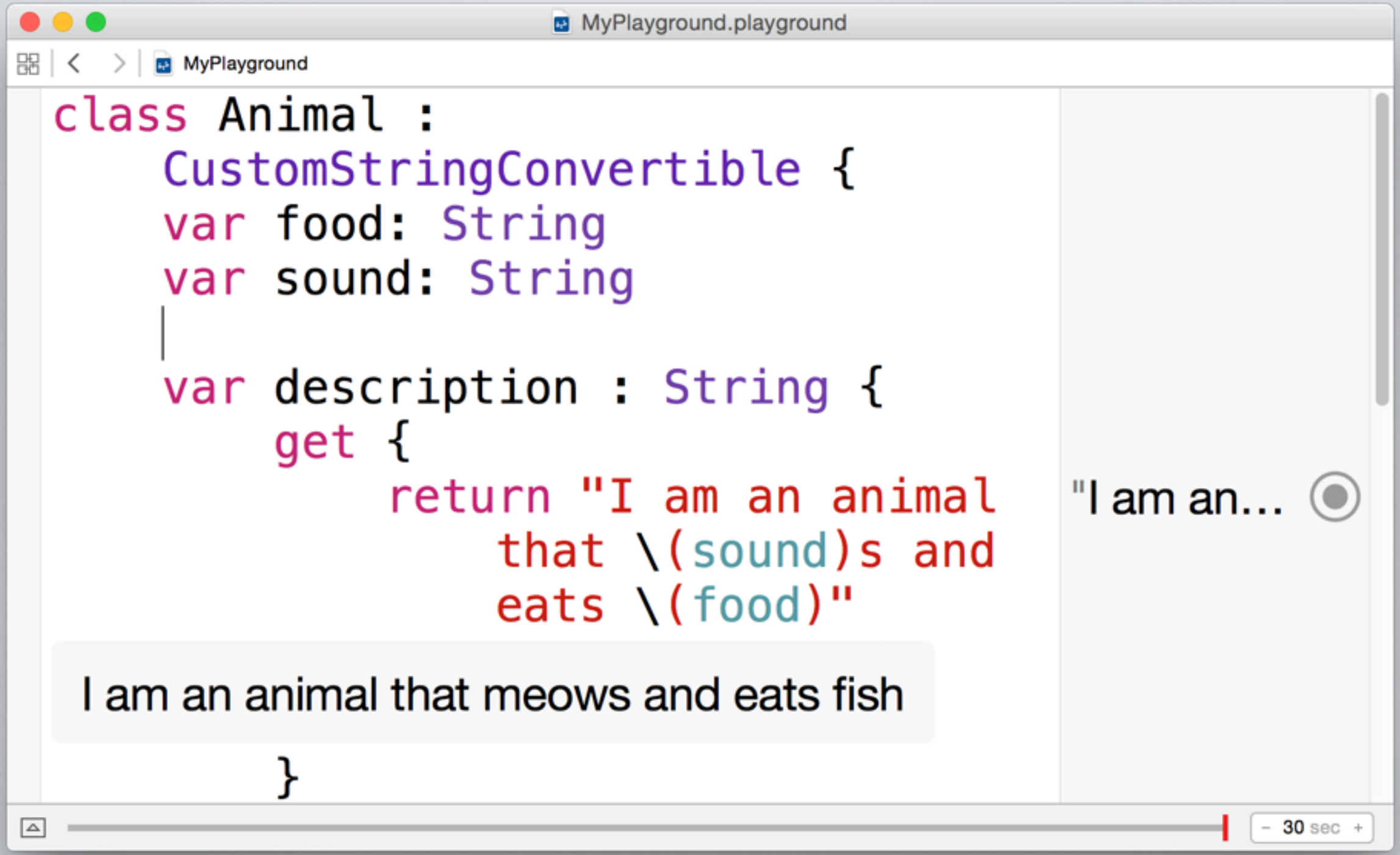
MyPlayground

```
class Animal :
    CustomStringConvertible {
    var food: String
    var sound: String

    var description : String {
        get {
            return "I am an animal
                that \(sound) and
                eats \(food)"
        }
    }
```

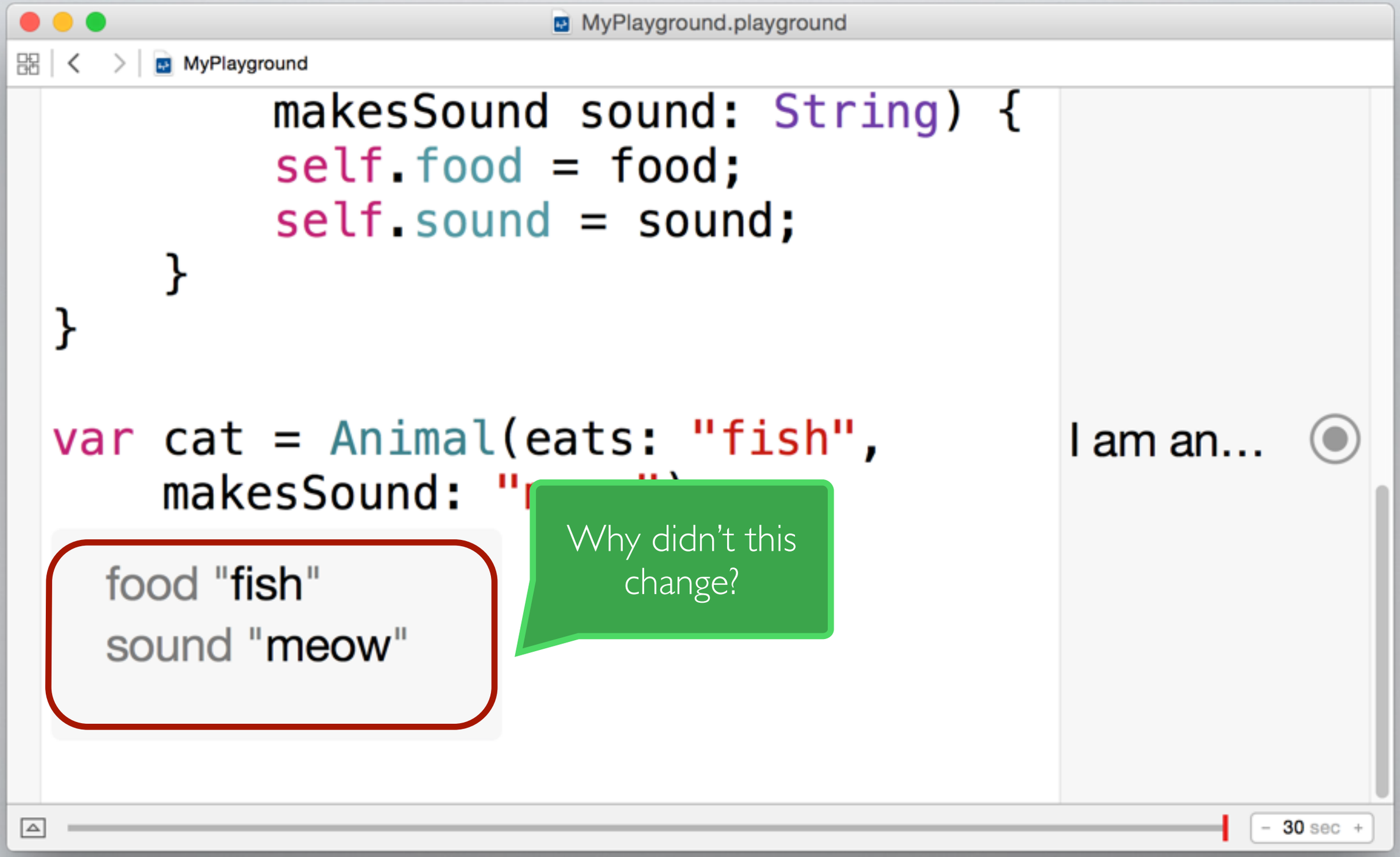
"I am an ani...

30 sec

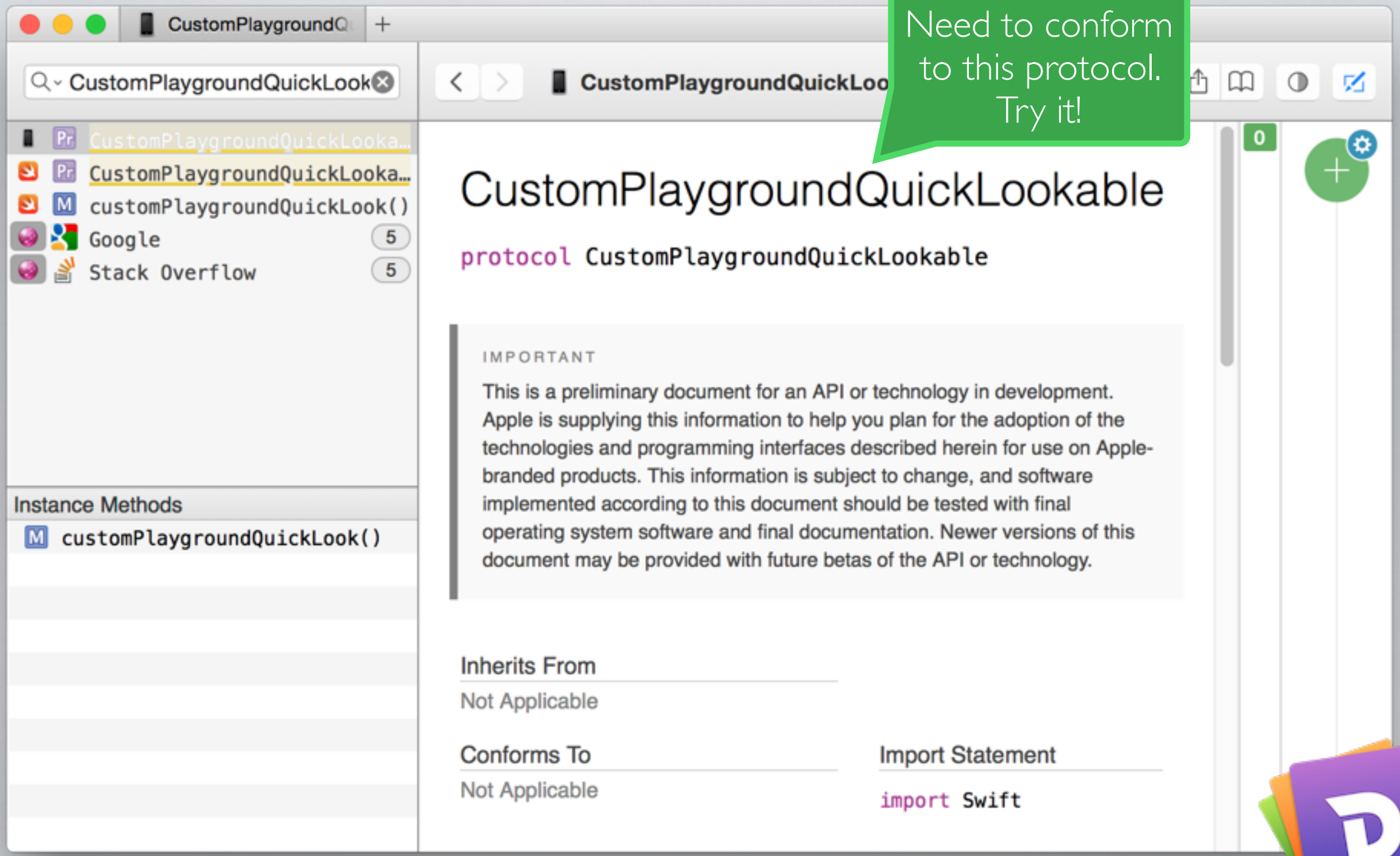
PROTOCOLS


MyPlayground.playground
MyPlayground
class Animal :
CustomStringConvertible {
var food: String
var sound: String
var description : String {
get {
return "I am an animal
that \(sound)s and
eats \(food)"
"I am an...
I am an animal that meows and eats fish
}
30 sec

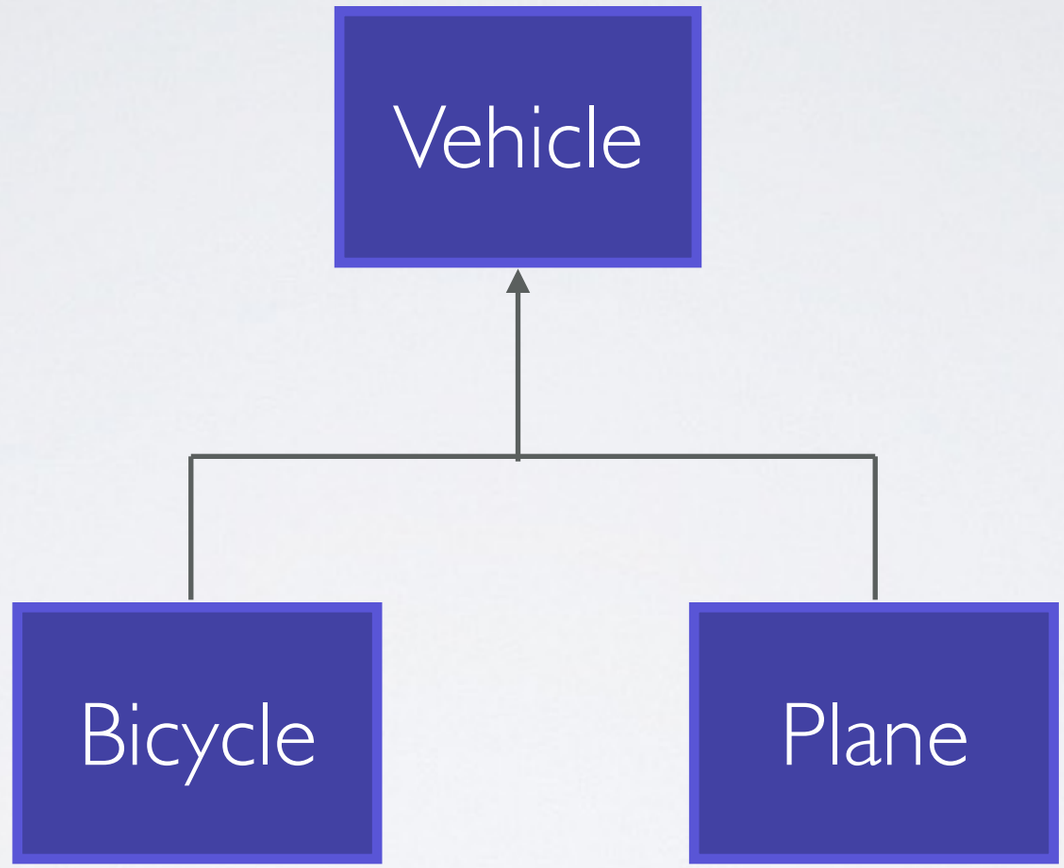
PROTOCOLS


MyPlayground.playground
MyPlayground
makesSound sound: String) {
self.food = food;
self.sound = sound;
}
}
var cat = Animal(eats: "fish",
makesSound: "
I am an...
food "fish"
sound "meow"
Why didn't this change?
30 sec

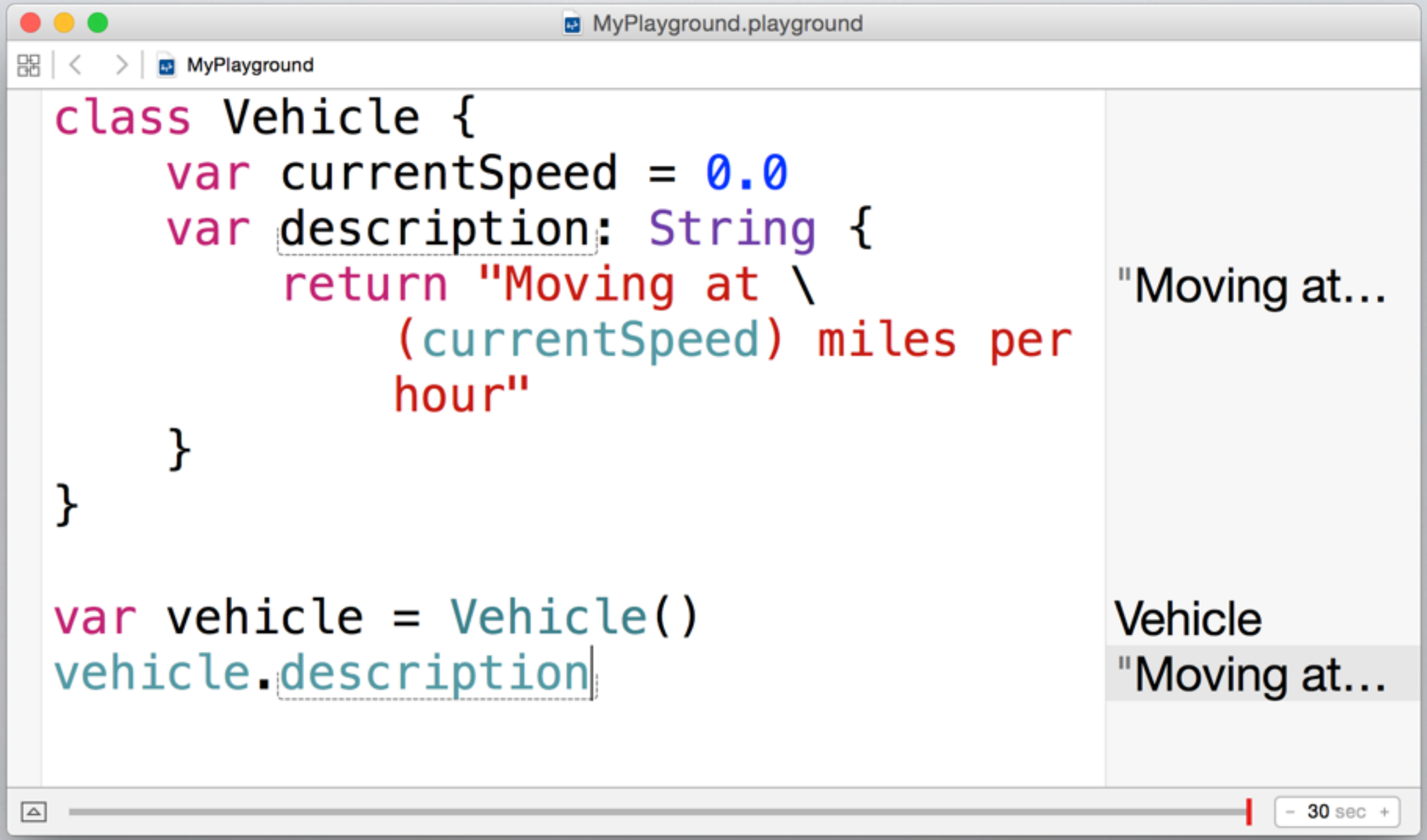
PROTOCOLS


CustomPlaygroundQ
CustomPlaygroundQuickLook
CustomPlaygroundQuickLooka…
CustomPlaygroundQuickLooka…
customPlaygroundQuickLook()
Google 5
Stack Overflow 5
CustomPlaygroundQuickLoo
Need to conform to this protocol. Try it!
CustomPlaygroundQuickLookable
protocol CustomPlaygroundQuickLookable
IMPORTANT
This is a preliminary document for an API or technology in development. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein for use on Apple-branded products. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future betas of the API or technology.
Instance Methods
customPlaygroundQuickLook()
Inherits From
Not Applicable
Conforms To
Not Applicable
Import Statement
import Swift
0

CLASSES



CLASSES


MyPlayground.playground
MyPlayground
class Vehicle {
var currentSpeed = 0.0
var description: String {
return "Moving at \
(currentSpeed) miles per
hour"
}
}
var vehicle = Vehicle()
vehicle.description
"Moving at...
Vehicle
"Moving at...
30 sec

CLASSES

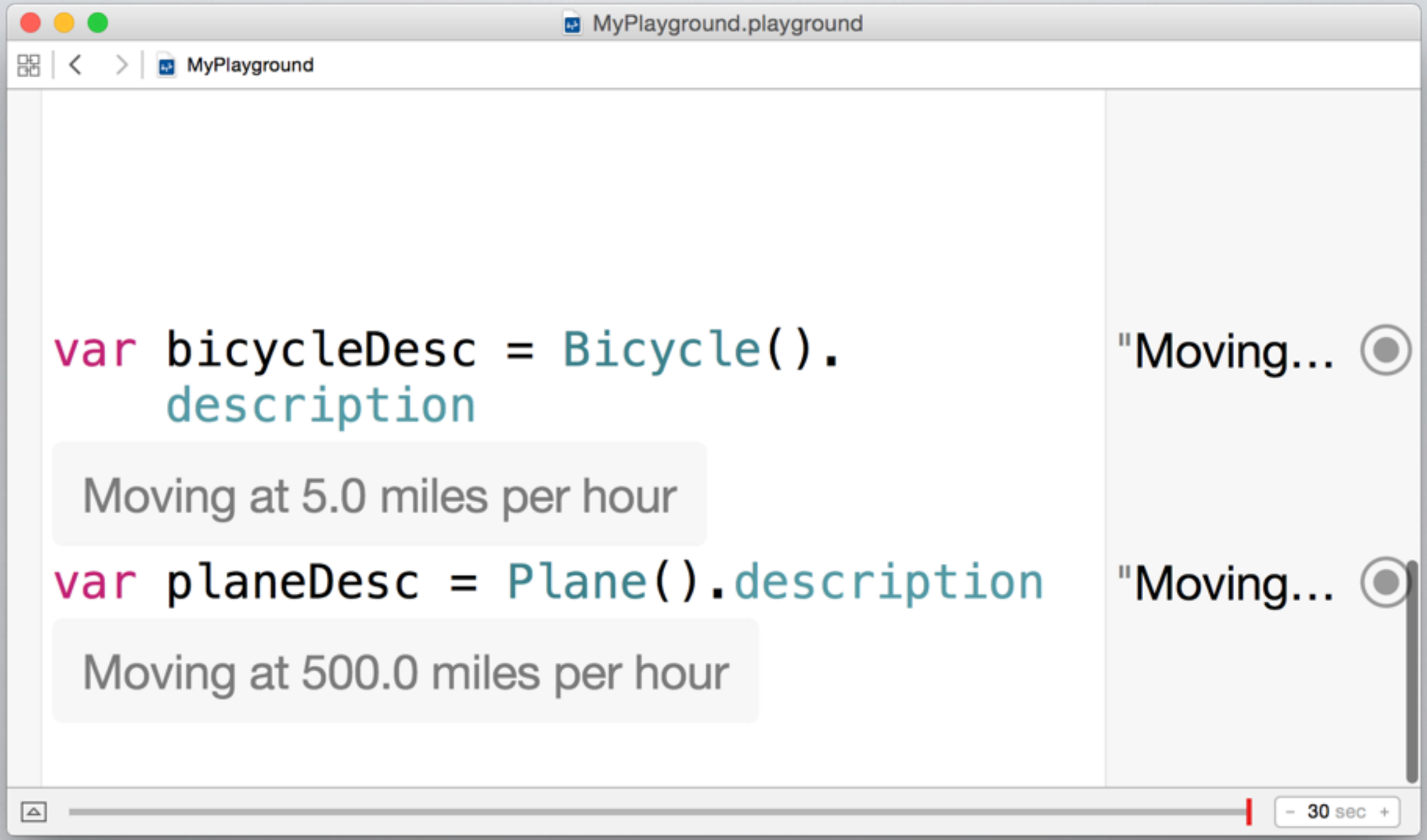
MyPlayground.playground

MyPlayground

```
class Bicycle: Vehicle {
    override init() {
        super.init()
        currentSpeed = 5.0
    }
}

class Plane: Vehicle {
    override init() {
        super.init()
        currentSpeed = 500.0
    }
}
```

30 sec

CLASSES


MyPlayground.playground
MyPlayground
var bicycleDesc = Bicycle().
description
"Moving...
Moving at 5.0 miles per hour
var planeDesc = Plane().description
"Moving...
Moving at 500.0 miles per hour
30 sec

STRUCTS

MyPlayground.playground

MyPlayground

```
struct Point {
    var x: Int = 0
    var y: Int = 0
}

var point0 = Point()                    Point
var point1 = Point(x:1, y:1)            Point
// You must follow the property
    declaration order
var point2 = Point(y:2, x:2)
```

Argument 'x' must precede argument 'y'

30 sec

STRUCTS

MyPlayground.playground

MyPlayground

```
var point1 = Point(x: 1, y:1)                    Point
var point2 = point1                              Point
point2.x = 5 // doesn't change the               Point
    original point1
print(point1)                                    "Point(x:...
```

Point(x: 1, y: 1)

```
print(point2)                                    "Point(x:...
```

Point(x: 5, y: 1)

30 sec

STRUCTS

MyPlayground.playground

MyPlayground

```
struct Point {
    var x: Int = 0
    var y: Int = 0

    // Attempting to change my value
    func changeMyValues() {
        self.x = 1   Cannot assign to prop...
        self.y = 2
    }
}
```

30 sec

STRUCTS

MyPlayground.playground

MyPlayground

```
struct Point {
    var x: Int = 0
    var y: Int = 0

    // Attempting to change my value
    mutating func changeMyValues() {
        self.x = 1
        self.y = 2
    }
}
```

30 sec

CLASSES VS STRUCTS

- Structs for small-ish types where values are more important than identity: Currency, Point, Dates, etc.

- Structs for cases where you don't want to share data (concurrency)

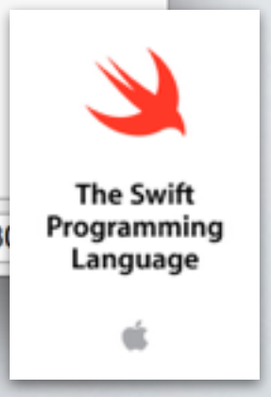
- Structs if you think in functional programming

ENUMS

MyPlayground.playground

MyPlayground

```
enum CompassPoint {
    case North
    case South
    case East
    case West
}


var directionToHead = CompassPoint.    .South
    South
```



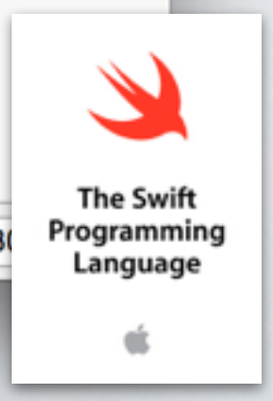
ENUMS

MyPlayground.playground

MyPlayground

```
switch directionToHead {
case .North:
    print("Lots of planets have a
        north")
case .South:
    print("Watch out for penguins")
case .East:
    print("Where the sun rises")
case .West:
    print("Where the skies are
        blue")
}
```

"Watch out...



WHEN TO USE ENUMS?

- When you have a *collection* of *data* that belong together

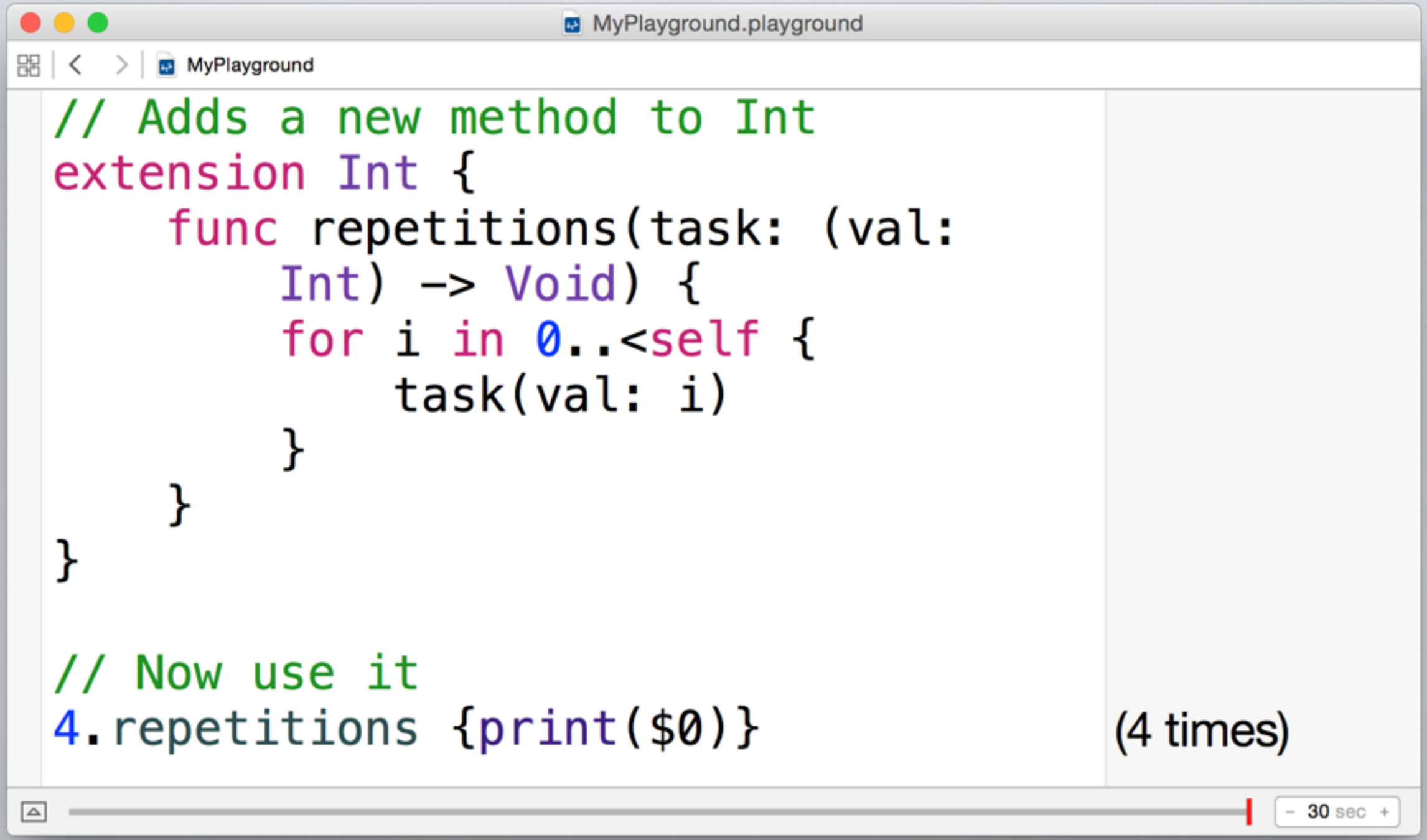
- When the collection of data is *bounded*, e.g., enum Directions { North, South, East, West}

- If you want to get creative, pattern matching and functional programming

EXTENSIONS

Extensions add new functionality to an existing class, structure, enumeration, or protocol type. This includes the ability to extend types for which you do not have access to the original source code

EXTENSIONS


MyPlayground.playground
MyPlayground
// Adds a new method to Int
extension Int {
func repetitions(task: (val:
Int) -> Void) {
for i in 0..<self {
task(val: i)
}
}
}
// Now use it
4.repetitions {print($0)}
(4 times)
30 sec

NEXT TIME

- Applying what we have learned to make HTTP requests
- Skim through AlamoFire https://github.com/Alamofire/Alamofire/tree/swift-2.0/Source code to test our knowledge of Swift