

Getting Started with Swift

Session 404

Dave Addey Xcode Documentation Engineer

Brian Lanier Swift Documentation Team Manager

Alex Martini Swift Documentation Engineer

The Basics

Dave Addey Xcode Documentation Engineer

Constants

```
let language: String = "Swift"
```

Constants

```
let language: String = "Swift"
```

Constants

```
let language: String = "Swift"
```

Constants

```
let language: String = "Swift"
```

Constants

```
let language: String = "Swift"
```

Constants

```
let language: String = "Swift"
```


Constants

```
let language: String = "Swift"  
let introduced: Int = 2014
```

Constants

```
let language: String = "Swift"  
let introduced: Int = 2014  
let isAwesome: Bool = true
```

Constants

```
let language: String = "Swift"  
let introduced: Int = 2014  
let isAwesome: Bool = true
```

Naming

```
let language: String = "Swift"  
let introduced: Int = 2014  
let isAwesome: Bool = true
```

Naming

```
let language: String = "Swift"  
let introduced: Int = 2014  
let isAwesome: Bool = true
```

Type Inference

```
let language: String = "Swift"  
let introduced: Int = 2014  
let isAwesome: Bool = true
```

Type Inference

```
let language = "Swift"  
let introduced = 2014  
let isAwesome = true
```

Type Inference

```
let language = "Swift"           // Inferred as String
let introduced = 2014             // Inferred as Int
let isAwesome = true             // Inferred as Bool
```


Variables

```
let language = "Swift"  
let introduced = 2014  
let isAwesome = true  
var version = 1
```

Variables

```
let language = "Swift"  
let introduced = 2014  
let isAwesome = true  
var version = 1
```

Variables

```
let language = "Swift"  
let introduced = 2014  
let isAwesome = true  
var version = 1  
version = 3
```

Variables

```
let language = "Swift"  
let introduced = 2014  
let isAwesome = true  
var version = 1  
version = 3  
isAwesome = false
```

Variables

```
let language = "Swift"
```

```
let introduced = 2014
```

```
let isAwesome = true
```

```
var version = 1
```

```
version = 3
```

```
isAwesome = false
```

 Error

Building Strings

```
let conference = "WWDC"  
  
let message = "Hello, " + conference + "!"  
// "Hello, WWDC!"
```

String Interpolation

```
let conference = "WWDC"
```

```
let message = "Hello, \(conference)!"
```

```
// "Hello, WWDC!"
```

String Interpolation

```
let conference = "WWDC"  
let year = 2016  
let message = "Hello, \(conference) \(year)!"  
// "Hello, WWDC 2016!"
```


String Interpolation

```
let conference = "WWDC"  
let year = 2016  
let message = "Hello, \(conference) \(year + 1)!"  
// "Hello, WWDC 2017!"
```

Unicode

```
let instruction = "Beware of the 🐶🐮"
```

Unicode

```
let instruction = "Beware of the 🐶🐮"  
let internationalHarmony = "🇬🇧🇺🇸💕"
```

Unicode

```
let instruction = "Beware of the 🐶🐮"
```

```
let internationalHarmony = "🇬🇧🇺🇸💕"
```

```
let  $\pi$  = 3.1415927
```

```
let 鼠标 = "🐭"
```

Characters

```
let dogString = "Dog?!🐶"
```

Characters

```
let dogString = "Dog?!"🐶"
```

Characters

```
let dogString = "Dog?!🐶"  
print("\(dogString) is \(dogString.characters.count) characters long")
```

Characters

```
let dogString = "Dog?!🐶"  
print("\(dogString) is \(dogString.characters.count) characters long")
```

Dog?!🐶 is 5 characters long.

Characters

```
let dogString = "Dog?!🐶"  
for character in dogString.characters {  
    print(character)  
}
```

Characters

```
let dogString = "Dog?!🐶"  
for character in dogString.characters {  
    print(character)  
}
```

D

O

g

?!



Array and Dictionary

Array and Dictionary

```
let names = ["Lily", "Santiago", "Justyn", "Aadya"]
```

Array and Dictionary

```
let names = ["Lily", "Santiago", "Justyn", "Aadya"]
```

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

Array and Dictionary

```
let names = ["Lily", "Santiago", "Justyn", "Aadya"]
```

```
let ages = {"Mohsen": 17, "Amy": 40, "Graham": 5}
```

Array and Dictionary

```
let names = ["Lily", "Santiago", "Justyn", "Aadya", 42]
```

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

Array and Dictionary

```
let names = ["Lily", "Santiago", "Justyn", "Aadya", true]
```

```
let ages = {"Mohsen": 17, "Amy": 40, "Graham": 5}
```


Array and Dictionary

```
let names = ["Lily", "Santiago", "Justyn", "Aadya", Bicycle()]
```

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

Array and Dictionary

```
let names = ["Lily", "Santiago", "Justyn", "Aadya"]
```

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

Array and Dictionary

```
let names: [String] = ["Lily", "Santiago", "Justyn", "Aadya"]
```

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

Type Inference

```
let names = ["Lily", "Santiago", "Justyn", "Aadya"]  
// an array of String values
```

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

Type Inference

```
let names = ["Lily", "Santiago", "Justyn", "Aadya"]  
// an array of String values
```

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]  
// a dictionary with String keys and Int values
```

Loops

While and Repeat-While

```
while !eofFile {  
    readLine()  
}
```

```
repeat {  
    performTask()  
} while tasksRemaining > 0
```

For-In Loop

Characters

```
let dogString = "Dog?!🐶"  
for character in dogString.characters {  
    print(character)  
}
```

For-In Loop

Characters

```
let dogString = "Dog?!🐶"  
for character in dogString.characters {  
    print(character)  
}
```

D

o

g

?!



For-In Loop

Closed Ranges

```
for number in 1...5 {  
    print("\(number) times 4 is \(number * 4)")  
}
```

For-In Loop

Closed Ranges

```
for number in 1...5 {  
    print("\(number) times 4 is \(number * 4)")  
}
```

```
1 times 4 is 4  
2 times 4 is 8  
3 times 4 is 12  
4 times 4 is 16  
5 times 4 is 20
```

For-In Loop

Closed Ranges

```
for number in 1...5 {  
    print("\(number) times 4 is \(number * 4)")  
}
```

```
1 times 4 is 4  
2 times 4 is 8  
3 times 4 is 12  
4 times 4 is 16  
5 times 4 is 20
```

For-In Loop

Half-Closed Ranges

```
let results = [7, 52, 9, 33, 6, 12, 86, 4, 22, 18, 3]
let maxResultCount = 5
for index in 0..
```

For-In Loop

Half-Closed Ranges

```
let results = [7, 52, 9, 33, 6, 12, 86, 4, 22, 18, 3]
let maxResultCount = 5
for index in 0..
```

Result 0 is 7

Result 1 is 52

Result 2 is 9

Result 3 is 33

Result 4 is 6

For-In Loop

Half-Closed Ranges

```
let results = [7, 52, 9, 33, 6, 12, 86, 4, 22, 18, 3]
let maxResultCount = 5
for index in 0..
```

Result 0 is 7

Result 1 is 52

Result 2 is 9

Result 3 is 33

Result 4 is 6

For-In Loop

Arrays

```
for name in ["Lily", "Santiago", "Justyn", "Aadya"] {  
    print("Hello, \(name)!")  
}
```

For-In Loop

Arrays

```
for name in ["Lily", "Santiago", "Justyn", "Aadya"] {  
    print("Hello, \(name)!")  
}
```

Hello, Lily!

Hello, Santiago!

Hello, Justyn!

Hello, Aadya!

For-In Loop

Dictionaries

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
for (name, age) in ages {
    print("\(name) is \(age) years old")
}
```

For-In Loop

Dictionaries

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
for (name, age) in ages {
    print("\(name) is \(age) years old")
}
```

Mohsen is 17 years old

Amy is 40 years old

Graham is 5 years old

For-In Loop

Dictionaries

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
for (name, age) in ages {
    print("\(name) is \(age) years old")
}
```

Mohsen is 17 years old

Amy is 40 years old

Graham is 5 years old

Modifying an Array

Packing for WWDC

```
var packingList = ["Socks", "Shoes"]
```

```
["Socks", "Shoes"]
```

Modifying an Array

Packing for WWDC

```
var packingList = ["Socks", "Shoes"]  
print(packingList[0])
```

Modifying an Array

Packing for WWDC

```
var packingList = ["Socks", "Shoes"]  
print(packingList[0])
```

Modifying an Array

Packing for WWDC

```
var packingList = ["Socks", "Shoes"]  
print(packingList[0])
```

"Socks"

Modifying an Array

Packing for WWDC

```
var packingList = ["Socks", "Shoes"]  
print(packingList[0])  
packingList.append("Trousers")
```

```
["Socks", "Shoes", "Trousers"]
```


Modifying an Array

Packing for WWDC

```
var packingList = ["Socks", "Shoes"]  
print(packingList[0])  
packingList.append("Trousers")
```

```
["Socks", "Shoes", "Trousers"]
```

Modifying an Array

Packing for WWDC

```
var packingList = ["Socks", "Shoes"]  
print(packingList[0])  
packingList.append("Trousers")  
packingList[2] = "Jeans"
```

```
["Socks", "Shoes", "Jeans"]
```

Modifying an Array

Packing for WWDC

```
var packingList = ["Socks", "Shoes"]  
print(packingList[0])  
packingList.append("Trousers")  
packingList[2] = "Jeans"
```

```
["Socks", "Shoes", "Jeans"]
```

Modifying an Array

Packing for WWDC

```
var packingList = ["Socks", "Shoes"]  
print(packingList[0])  
packingList.append("Trousers")  
packingList[2] = "Jeans"  
packingList.append(contentsOf: ["Shorts", "Sandals", "Sunblock"])
```

```
["Socks", "Shoes", "Jeans", "Shorts", "Sandals", "Sunblock"]
```

Modifying an Array

Packing for WWDC

```
var packingList = ["Socks", "Shoes"]  
print(packingList[0])  
packingList.append("Trousers")  
packingList[2] = "Jeans"  
packingList.append(contentsOf: ["Shorts", "Sandals", "Sunblock"])
```

```
["Socks", "Shoes", "Jeans", "Shorts", "Sandals", "Sunblock"]
```

Modifying an Array

Packing for WWDC

```
var packingList = ["Socks", "Shoes"]  
print(packingList[0])  
packingList.append("Trousers")  
packingList[2] = "Jeans"  
packingList.append(contentsOf: ["Shorts", "Sandals", "Sunblock"])  
packingList[3...5] = ["Hoodie", "Scarf"]
```

```
["Socks", "Shoes", "Jeans", "Hoodie", "Scarf"]
```

Modifying an Array

Packing for WWDC

```
var packingList = ["Socks", "Shoes"]  
print(packingList[0])  
packingList.append("Trousers")  
packingList[2] = "Jeans"  
packingList.append(contentsOf: ["Shorts", "Sandals", "Sunblock"])  
packingList[3...5] = ["Hoodie", "Scarf"]
```

```
["Socks", "Shoes", "Jeans", "Hoodie", "Scarf"]
```

Modifying a Dictionary

```
var ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

```
["Mohsen": 17, "Amy": 40, "Graham": 5]
```


Modifying a Dictionary

```
var ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]  
ages["Justyn"] = 67      // Adds a new value for "Justyn"
```

```
["Mohsen": 17, "Amy": 40, "Graham": 5, "Justyn": 67]
```

Modifying a Dictionary

```
var ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]  
ages["Justyn"] = 67      // Adds a new value for "Justyn"  
ages["Justyn"] = 68      // Changes the value for "Justyn"
```

```
["Mohsen": 17, "Amy": 40, "Graham": 5, "Justyn": 68]
```

Retrieving a Value from a Dictionary

```
let ages = {"Mohsen": 17, "Amy": 40, "Graham": 5}
```

Retrieving a Value from a Dictionary

```
let ages = {"Mohsen": 17, "Amy": 40, "Graham": 5}  
// Devon?
```

Retrieving a Value from a Dictionary

```
let ages = {"Mohsen": 17, "Amy": 40, "Graham": 5}  
// Devon?  
// Daryl?
```

Retrieving a Value from a Dictionary

```
let ages = {"Mohsen": 17, "Amy": 40, "Graham": 5}  
// Devon?  
// Daryl?  
// Daniel?
```

Optionals

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

```
let possibleAge = ages["Amy"]
```

Optionals

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

```
let possibleAge = ages["Amy"]
```

A value of 40, perhaps?

Optionals

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

```
let possibleAge = ages["Daryl"]
```

Optionals

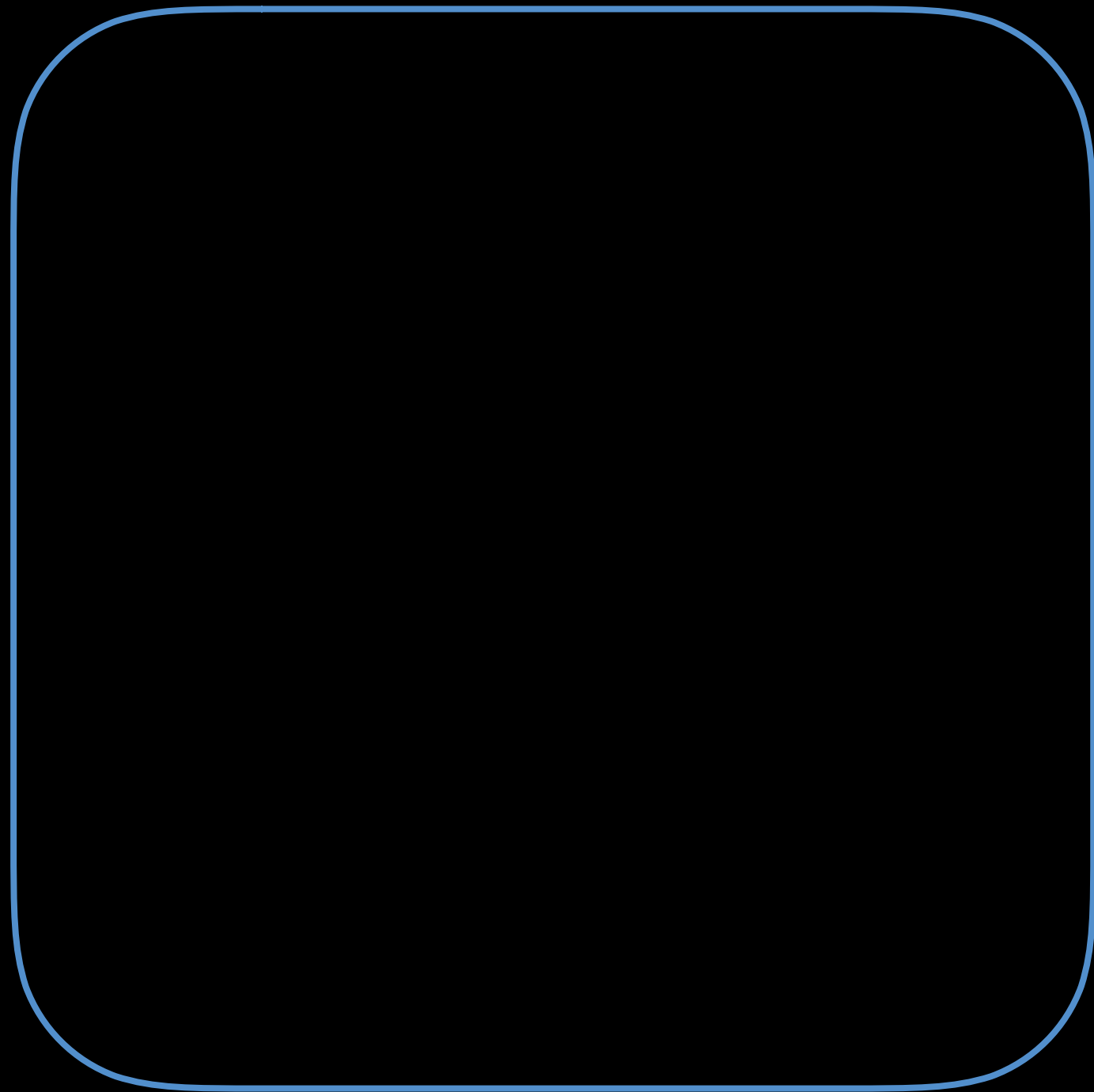
```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

```
let possibleAge = ages["Daryl"]
```

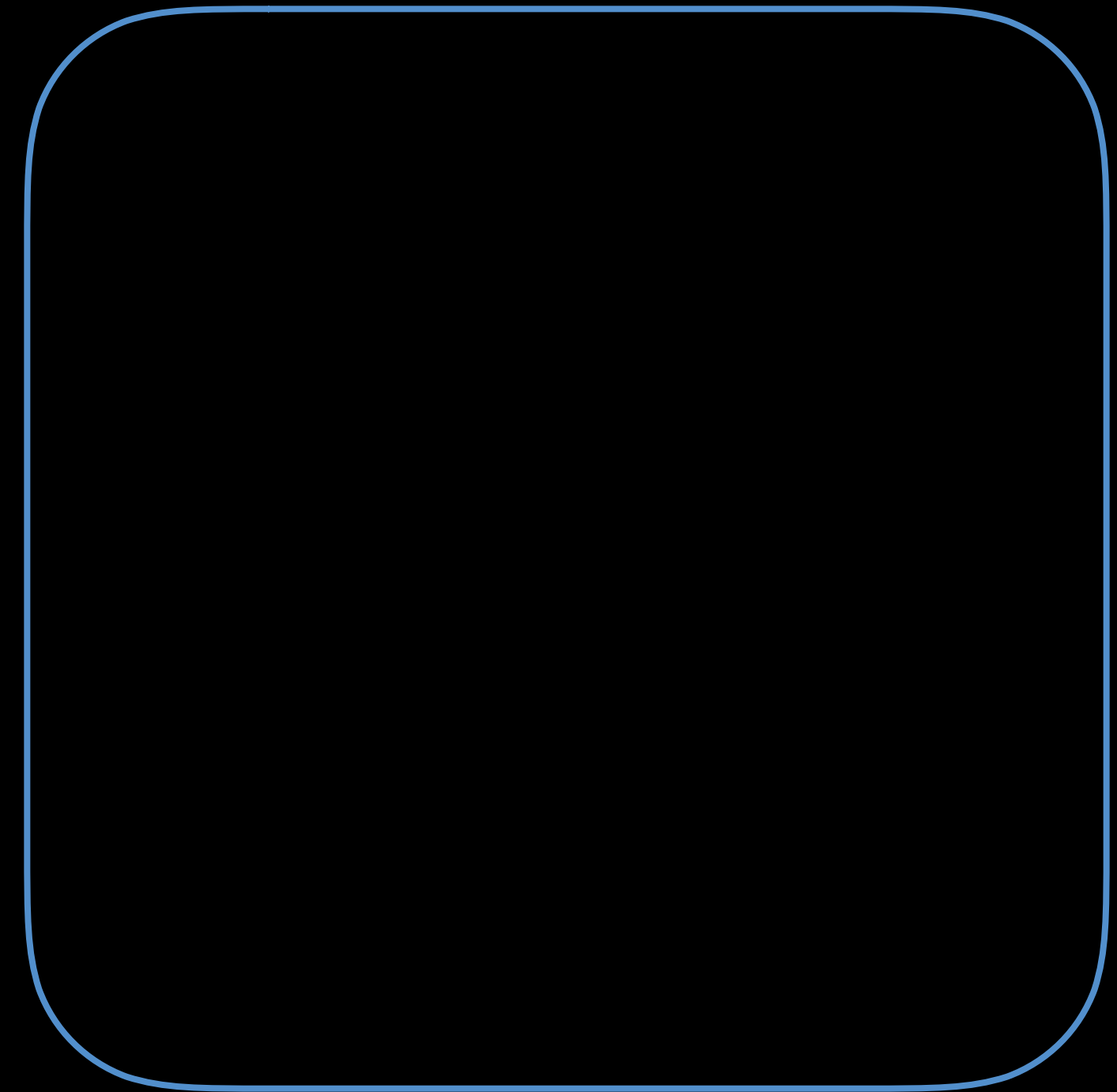
?????

Optionals

ages["Amy"]



ages["Daryl"]



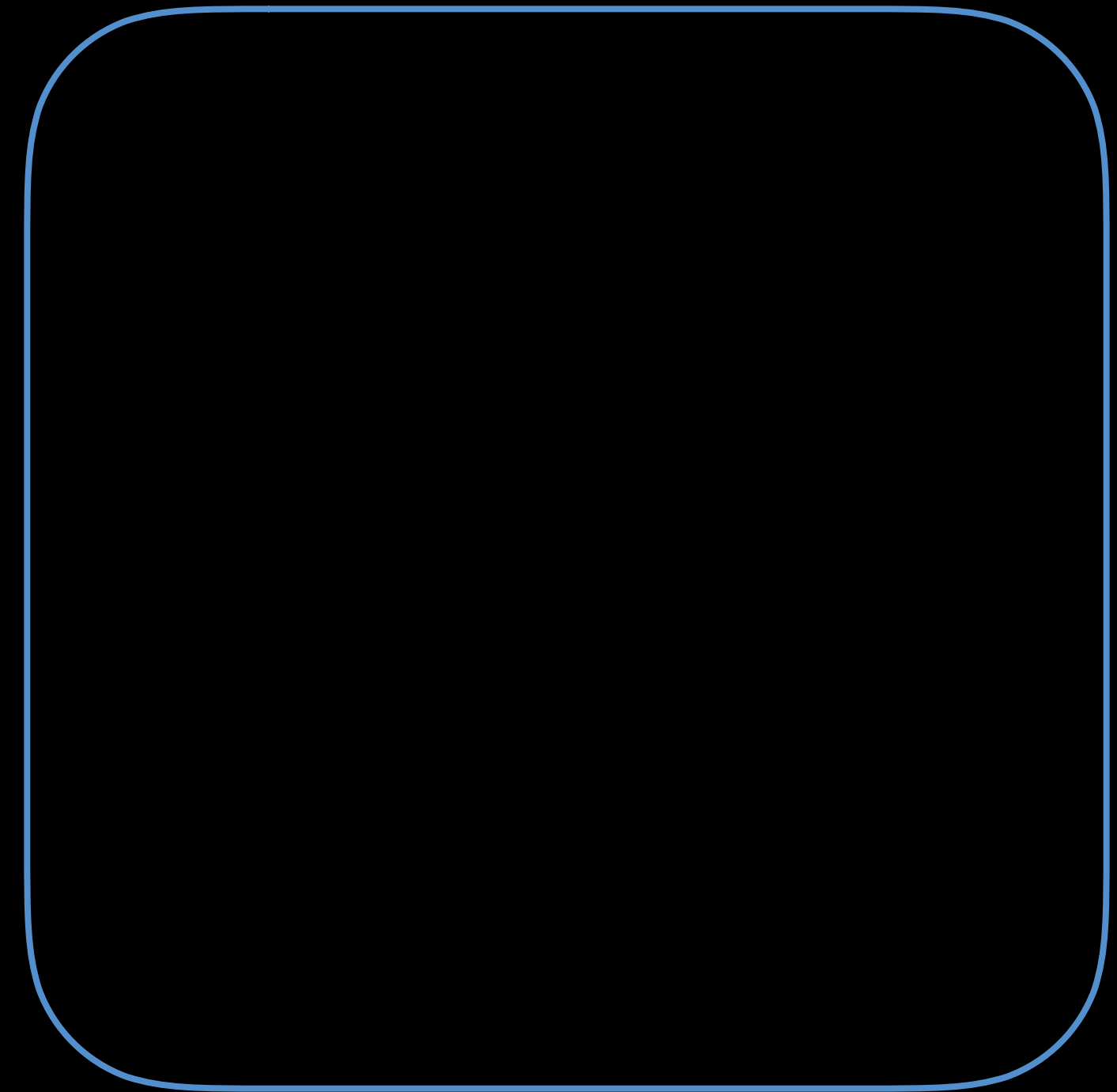
Optionals

ages["Amy"]

40

Int

ages["Daryl"]



Optionals

ages["Amy"]

40

Int

ages["Daryl"]

No
Value

No Int

Optionals

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

```
let possibleAge: Int? = ages["Daryl"]
```

Optionals

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

```
let possibleAge: Int? = ages["Daryl"]
```

Checking for an Optional Value

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

```
let possibleAge: Int? = ages["Daryl"]
```

```
if possibleAge == nil {  
    print("Age not found.")  
}
```


Checking for an Optional Value

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

```
let possibleAge: Int? = ages["Daryl"]
```

```
if possibleAge == nil {  
    print("Age not found.")  
}
```

Checking for an Optional Value

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

```
let possibleAge: Int? = ages["Daryl"]
```

```
if possibleAge == nil {  
    print("Age not found.")  
}
```

Age not found.

Checking for an Optional Value

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

```
let possibleAge: Int? = ages["Amy"]
```

```
if possibleAge == nil {  
    print("Age not found.")  
}
```

If-Let Statement

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]

if let age = ages["Amy"] {
    print("An age of \(age) was found.")
}
```

If-Let Statement

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```

```
if let age = ages["Amy"] {  
    print("An age of \(age) was found.")  
}
```

If-Let Statement

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]

if let age = ages["Amy"] {
    print("An age of \(age) was found.")
}
```

If-Let Statement

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]

if let age = ages["Amy"] {
    print("An age of \(age) was found.")
}
```

If-Let Statement

```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]

if let age = ages["Amy"] {
    print("An age of \(age) was found.")
}
```

An age of 40 was found.

If Statement

```
let age = 32

if age == 1 {
    print("Happy first birthday!")
} else if age == 40 {
    print("Happy 40th birthday!")
} else {
    print("Happy plain old boring birthday.")
}
```

If Statement

```
let age = 32

if age == 1 {
    print("Happy first birthday!")
} else if age == 40 {
    print("Happy 40th birthday!")
} else {
    print("Happy plain old boring birthday.")
}
```

If Statement

```
let age = 32

if age == 1 {
    print("Happy first birthday!")
} else if age == 40 {
    print("Happy 40th birthday!")
} else {
    print("Happy plain old boring birthday.")
}
```

Switch Statement

```
let age = 1  
switch age {
```

```
}
```

Switch Statement

```
let age = 1  
switch age {
```

```
}
```

Switch Statement

```
let age = 1
switch age {
case 1:
    print("Happy first birthday!")

}
```

Switch Statement

```
let age = 15
switch age {
case 1:
    print("Happy first birthday!")
case 13...19:
    print("Happy birthday, teenager!")

}
```

Switch Statement

```
let age = 40
switch age {
case 1:
    print("Happy first birthday!")
case 13...19:
    print("Happy birthday, teenager!")
case let decade where decade % 10 == 0:
    print("Happy significant \("\(decade)th birthday!")")

}
```


Switch Statement

```
let age = 40
switch age {
case 1:
    print("Happy first birthday!")
case 13...19:
    print("Happy birthday, teenager!")
case let decade where decade % 10 == 0:
    print("Happy significant \ (decade)th birthday!")

}
```

Switch Statement

```
let age = 40
switch age {
case 1:
    print("Happy first birthday!")
case 13...19:
    print("Happy birthday, teenager!")
case let decade where decade % 10 == 0:
    print("Happy significant \ (decade)th birthday!")

}
```

Switch Statement

```
let age = 40
switch age {
case 1:
    print("Happy first birthday!")
case 13...19:
    print("Happy birthday, teenager!")
case let decade where decade % 10 == 0:
    print("Happy significant \(decade)th birthday!")

}
```

Switch Statement

```
let age = 40
switch age {
case 1:
    print("Happy first birthday!")
case 13...19:
    print("Happy birthday, teenager!")
case let decade where decade % 10 == 0:
    print("Happy significant \(decade)th birthday!")

}
```

Switch Statement

```
let age = 41
switch age {
case 1:
    print("Happy first birthday!")
case 13...19:
    print("Happy birthday, teenager!")
case let decade where decade % 10 == 0:
    print("Happy significant \(decade)th birthday!")

}
```

Switch Statement

```
let age = 97
switch age {
case 1:
    print("Happy first birthday!")
case 13...19:
    print("Happy birthday, teenager!")
case let decade where decade % 10 == 0:
    print("Happy significant \(decade)th birthday!")

}
```

Switch Statement

```
let age = 56
switch age {
case 1:
    print("Happy first birthday!")
case 13...19:
    print("Happy birthday, teenager!")
case let decade where decade % 10 == 0:
    print("Happy significant \((decade)th birthday!")

}
```

Switch Statement

```
let age = 32
switch age {
case 1:
    print("Happy first birthday!")
case 13...19:
    print("Happy birthday, teenager!")
case let decade where decade % 10 == 0:
    print("Happy significant \ (decade)th birthday!")
default:
    print("Happy plain old boring birthday.")
}
```


Switch Statement

```
let userName = "admin"  
let passwordIsValid = true
```

Switch Statement

```
let userName = "admin"  
let passwordIsValid = true
```

Switch Statement

```
let userName = "admin"
```

```
let passwordIsValid = true
```

Switch Statement

[illegible]

Switch Statement

```
let userName = "admin"  
let passwordIsValid = true  
switch (userName, passwordIsValid) {  
case ("admin", true):  
    print("Welcome back, administrator!")  
  
}
```

Switch Statement

```
let userName = "admin"
let passwordIsValid = true
switch (userName, passwordIsValid) {
case ("admin", true):
    print("Welcome back, administrator!")
case ("guest", _):
    print("Guests are not allowed in this restricted area.")

}
```

Switch Statement

```
let userName = "admin"
let passwordIsValid = true
switch (userName, passwordIsValid) {
  case ("admin", true):
    print("Welcome back, administrator!")
  case ("guest", _):
    print("Guests are not allowed in this restricted area.")
}
```

Switch Statement

```
let userName = "admin"
let passwordIsValid = true
switch (userName, passwordIsValid) {
case ("admin", true):
    print("Welcome back, administrator!")
case ("guest", _):
    print("Guests are not allowed in this restricted area.")
case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```


Switch Statement

```
let userName = "admin"
let passwordIsValid = true
switch (userName, passwordIsValid) {
case ("admin", true):
    print("Welcome back, administrator!")
case ("guest", _):
    print("Guests are not allowed in this restricted area.")
case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```

Switch Statement

```
let userName = "admin"
let passwordIsValid = true
switch (userName, passwordIsValid) {
  case ("admin", true):
    print("Welcome back, administrator!")
  case ("guest", _):
    print("Guests are not allowed in this restricted area.")
  case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```

Switch Statement

```
let userName = "admin"
let passwordIsValid = true
switch (userName, passwordIsValid) {
case ("admin", true):
    print("Welcome back, administrator!")
case ("guest", _):
    print("Guests are not allowed in this restricted area.")
case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```

Switch Statement

```
let userName = "admin"
let passwordIsValid = true
switch (userName, passwordIsValid) {
case ("admin", true):
    print("Welcome back, administrator!")
case ("guest", _):
    print("Guests are not allowed in this restricted area.")
case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```

Switch Statement

```
let userName = "admin"
let passwordIsValid = true
switch (userName, passwordIsValid) {
  case ("admin", true):
    print("Welcome back, administrator!")
  case ("guest", _):
    print("Guests are not allowed in this restricted area.")
  case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```

Switch Statement

```
let userName = "admin"
let passwordIsValid = true
switch (userName, passwordIsValid) {
  case ("admin", true):
    print("Welcome back, administrator!")
  case ("guest", _):
    print("Guests are not allowed in this restricted area.")
  case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```

Switch Statement

```
let userName = "admin"
let passwordIsValid = true
switch (userName, passwordIsValid) {
case ("admin", true):
    print("Welcome back, administrator!")
case ("guest", _):
    print("Guests are not allowed in this restricted area.")
case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```

Welcome back, administrator!

Switch Statement

```
let userName = "guest"
let passwordIsValid = true
switch (userName, passwordIsValid) {
case ("admin", true):
    print("Welcome back, administrator!")
case ("guest", _):
    print("Guests are not allowed in this restricted area.")
case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```


Switch Statement

```
let userName = "guest"
let passwordIsValid = true
switch (userName, passwordIsValid) {
case ("admin", true):
    print("Welcome back, administrator!")
case ("guest", _):
    print("Guests are not allowed in this restricted area.")
case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```

Guests are not allowed in this restricted area.

Switch Statement

```
let userName = "bob"
let passwordIsValid = true
switch (userName, passwordIsValid) {
  case ("admin", true):
    print("Welcome back, administrator!")
  case ("guest", _):
    print("Guests are not allowed in this restricted area.")
  case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```

Switch Statement

```
let userName = "bob"
let passwordIsValid = true
switch (userName, passwordIsValid) {
  case ("admin", true):
    print("Welcome back, administrator!")
  case ("guest", _):
    print("Guests are not allowed in this restricted area.")
  case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```

Welcome to the restricted area!

Switch Statement

```
let userName = "bob"
let passwordIsValid = false
switch (userName, passwordIsValid) {
  case ("admin", true):
    print("Welcome back, administrator!")
  case ("guest", _):
    print("Guests are not allowed in this restricted area.")
  case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```

Switch Statement

```
let userName = "bob"
let passwordIsValid = false
switch (userName, passwordIsValid) {
  case ("admin", true):
    print("Welcome back, administrator!")
  case ("guest", _):
    print("Guests are not allowed in this restricted area.")
  case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```

ACCESS DENIED.

Switch Statement

```
let userName = "bob"
let passwordIsValid = false
switch (userName, passwordIsValid) {
case ("admin", true):
    print("Welcome back, administrator!")
case ("guest", _):
    print("Guests are not allowed in this restricted area.")
case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```

Switch Statement

```
let userName = "bob"
let passwordIsValid = false
switch (userName, passwordIsValid) {
  case ("admin", true):
    print("Welcome back, administrator!")
  case ("guest", _):
    print("Guests are not allowed in this restricted area.")
  case (_, let isValid):
    print(isValid ? "Welcome to the restricted area!" : "ACCESS DENIED.")
}
```

Functions and Closures

Brian Lanier Swift Documentation Team Manager

Functions

```
func sendMessage() {  
    let message = "Hey there!"  
    print(message)  
}
```

Functions

```
func sendMessage() {  
    let message = "Hey there!"  
    print(message)  
}  
sendMessage()
```

Functions

```
func sendMessage() {  
    let message = "Hey there!"  
    print(message)  
}  
sendMessage()
```

Hey there!

Parameters

```
func sendMessage(shouting: Bool) {  
    var message = "Hey there!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}
```

Parameters

```
func sendMessage(shouting: Bool) {  
    var message = "Hey there!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}
```

Argument Labels

```
func sendMessage(shouting: Bool) {  
    var message = "Hey there!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
sendMessage(shouting: true)
```

Argument Labels

```
func sendMessage(shouting: Bool) {  
    var message = "Hey there!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
sendMessage(shouting: true)
```

HEY THERE!

Multiple Parameters

```
func sendMessage(recipient: String, shouting: Bool) {  
    var message = "Hey there, \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}
```


Multiple Parameters

```
func sendMessage(recipient: String, shouting: Bool) {  
    var message = "Hey there, \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
sendMessage(recipient: "Morgan", shouting: false)
```

Multiple Parameters

```
func sendMessage(recipient: String, shouting: Bool) {  
    var message = "Hey there, \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
sendMessage(recipient: "Morgan", shouting: false)
```

Hey there, Morgan!

Multiple Parameters

```
func sendMessage(recipient: String, shouting: Bool) {  
    var message = "Hey there, \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
sendMessage(recipient: "Morgan", shouting: false)
```

Multiple Parameters

```
func sendMessage(to: String, shouting: Bool) {  
    var message = "Hey there, \$(to)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
sendMessage(to: "Morgan", shouting: false)
```

Multiple Parameters

```
func sendMessage(to: String, shouting: Bool) {  
    var message = "Hey there, \!(to)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
sendMessage(to: "Morgan", shouting: false)
```

Multiple Parameters

```
func sendMessage(to: String, shouting: Bool) {  
    var message = "Hey there, \!(to)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
sendMessage(to: "Morgan", shouting: false)
```

Multiple Parameters

```
func sendMessage(to: String, shouting: Bool) {  
    var message = "Hey there, \(to)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
sendMessage(to: "Morgan", shouting: false)
```

Explicit Argument Labels



```
func sendMessage(to recipient: String, shouting: Bool) {  
    var message = "Hey there, \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
  
sendMessage(to: "Morgan", shouting: false)
```


Explicit Argument Labels



```
func sendMessage(to recipient: String, shouting: Bool) {  
    var message = "Hey there, \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
sendMessage(to: "Morgan", shouting: false)
```

Explicit Argument Labels



```
func sendMessage(to recipient: String, shouting: Bool) {  
    var message = "Hey there, \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
sendMessage(to: "Morgan", shouting: false)
```

Argument Labels

```
func sendMessage(message: String, to recipient: String, shouting: Bool) {  
    var message = "\(message), \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}
```

Argument Labels

```
func sendMessage(message: String, to recipient: String, shouting: Bool) {  
    var message = "\(message), \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
  
sendMessage(message: "See you at the Bash", to: "Morgan", shouting: false)
```

Argument Labels

```
func sendMessage(message: String, to recipient: String, shouting: Bool) {  
    var message = "\(message), \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
sendMessage(message: "See you at the Bash", to: "Morgan", shouting: false)
```

See you at the Bash, Morgan!

Argument Labels

```
func sendMessage(message: String, to recipient: String, shouting: Bool) {  
    var message = "\(message), \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
sendMessage(message: "See you at the Bash", to: "Morgan", shouting: false)
```

Omitting Argument Labels



```
func sendMessage(_ message: String, to recipient: String, shouting: Bool) {  
    var message = "\(message), \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}
```

Omitting Argument Labels



```
func sendMessage(_ message: String, to recipient: String, shouting: Bool) {  
    var message = "\(message), \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
  
sendMessage("See you at the Bash", to: "Morgan", shouting: false)
```


Parameters with Default Values

```
func sendMessage(_ message: String, to recipient: String, shouting: Bool = false) {  
    var message = "\(message), \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}
```

Parameters with Default Values

```
func sendMessage(_ message: String, to recipient: String, shouting: Bool = false) {  
    var message = "\(message), \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
  
sendMessage("See you at the Bash", to: "Morgan")
```

Parameters with Default Values

```
func sendMessage(_ message: String, to recipient: String, shouting: Bool = false) {  
    var message = "\(message), \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
sendMessage("See you at the Bash", to: "Morgan")
```

See you at the Bash, Morgan!

Parameters with Default Values

```
func sendMessage(_ message: String, to recipient: String, shouting: Bool = false) {  
    var message = "\(message), \(recipient)!"  
    if shouting {  
        message = message.uppercased()  
    }  
    print(message)  
}  
  
sendMessage("See you at the Bash", to: "Morgan")
```

Function Return Values

Function Return Values

```
func firstString(havingPrefix prefix: String, in strings: [String]) -> String {...}
```

Function Return Values

```
func firstString(havingPrefix prefix: String, in strings: [String]) -> String {...}
```

Function Return Values

```
func firstString(havingPrefix prefix: String, in strings: [String]) -> String {...}
```


Function Return Values

```
func firstString(havingPrefix prefix: String, in strings: [String]) -> String {...}
```

Function Return Values

```
func firstString(havingPrefix prefix: String, in strings: [String]) -> String {  
    for string in strings {  
  
    }  
  
}
```

Function Return Values

```
func firstString(havingPrefix prefix: String, in strings: [String]) -> String {  
    for string in strings {  
        if string.hasPrefix(prefix) {  
            return string  
        }  
    }  
}
```

Function Return Values

```
func firstString(havingPrefix prefix: String, in strings: [String]) -> String {  
    for string in strings {  
        if string.hasPrefix(prefix) {  
            return string  
        }  
    }  
}
```

Function Return Values

```
func firstString(havingPrefix prefix: String, in strings: [String]) -> String {  
    for string in strings {  
        if string.hasPrefix(prefix) {  
            return string  
        }  
    }  
    return  
}
```

Function Return Values

```
func firstString(havingPrefix prefix: String, in strings: [String]) -> String {  
    for string in strings {  
        if string.hasPrefix(prefix) {  
            return string  
        }  
    }  
    return ""  
}
```

Returning Optional Values

```
func firstString(havingPrefix prefix: String, in strings: [String]) -> String? {  
    for string in strings {  
        if string.hasPrefix(prefix) {  
            return string  
        }  
    }  
    return nil  
}
```

Returning Optional Values

```
var guests = ["Jack", "Kumar", "Anita", "Anna"]  
  
if let guest = firstString(havingPrefix: "A", in: guests) {  
    print("See you at the party, \(guest)!")  
} else {  
    print("Invite must be in the mail.")  
}
```


Returning Optional Values

```
var guests = ["Jack", "Kumar", "Anita", "Anna"]  
  
if let guest = firstString(havingPrefix: "A", in: guests) {  
    print("See you at the party, \(guest)!")  
} else {  
    print("Invite must be in the mail.")  
}
```

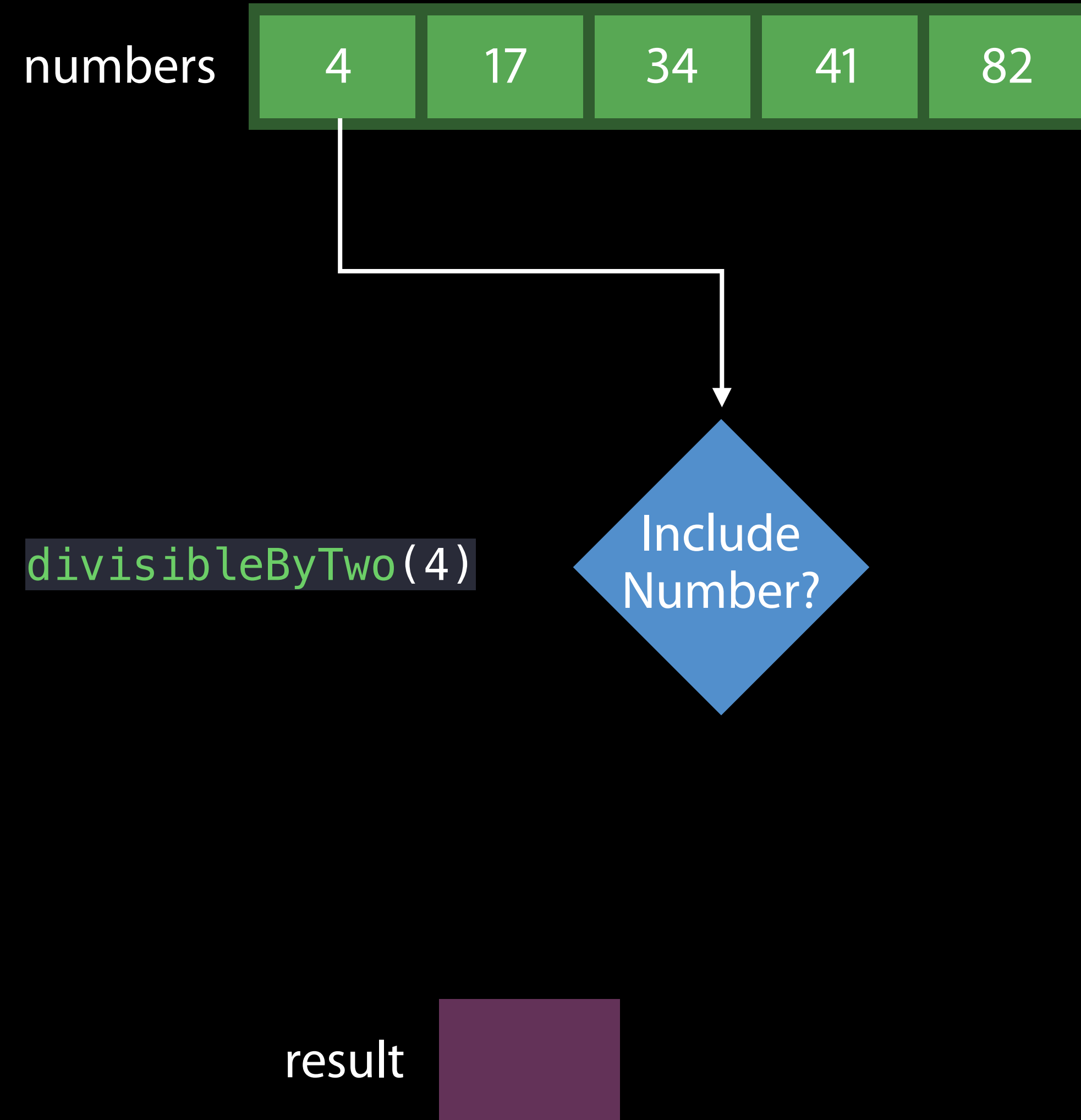
See you at the party, Anita!

Filtering Numbers

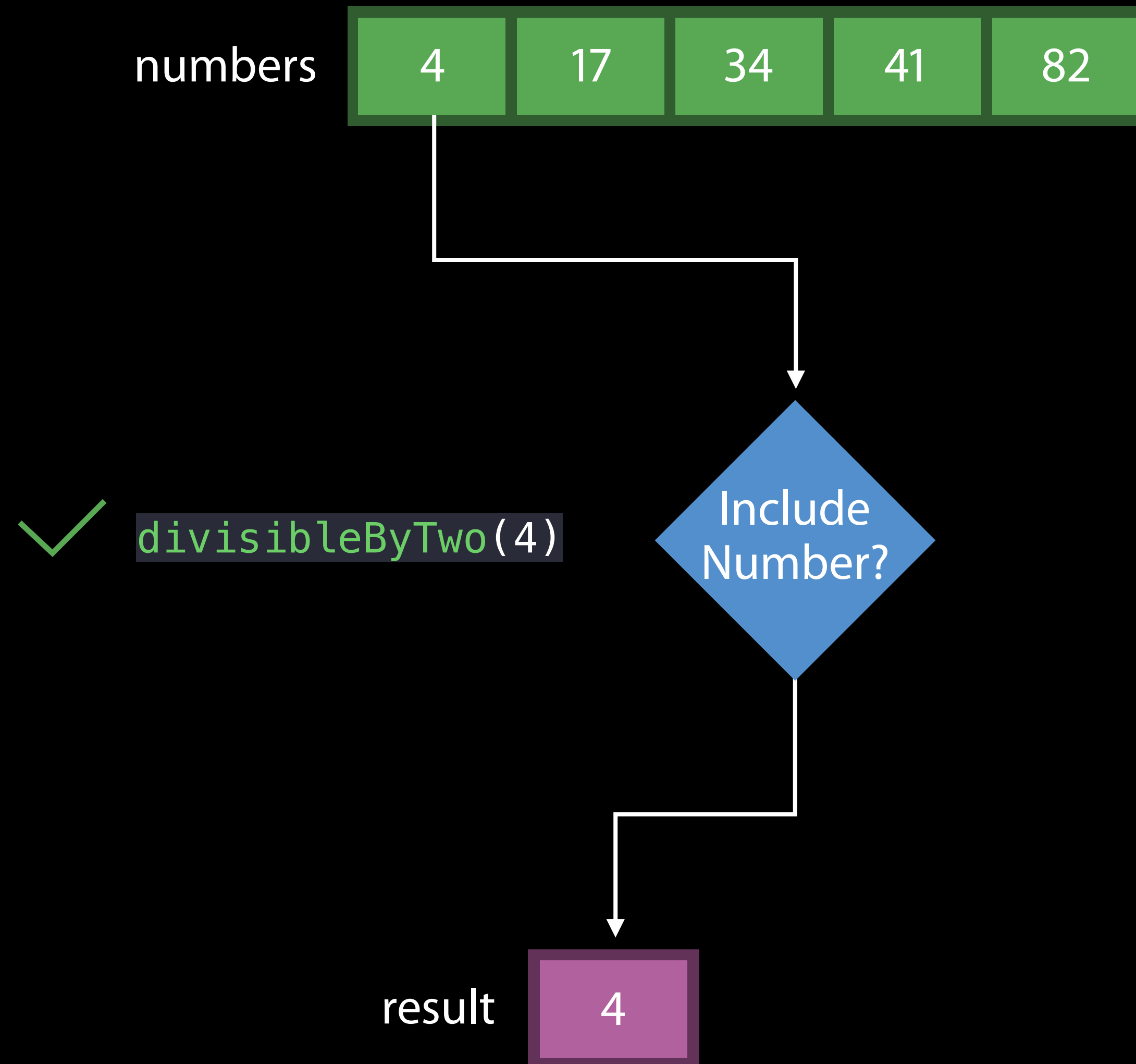
Filtering Numbers



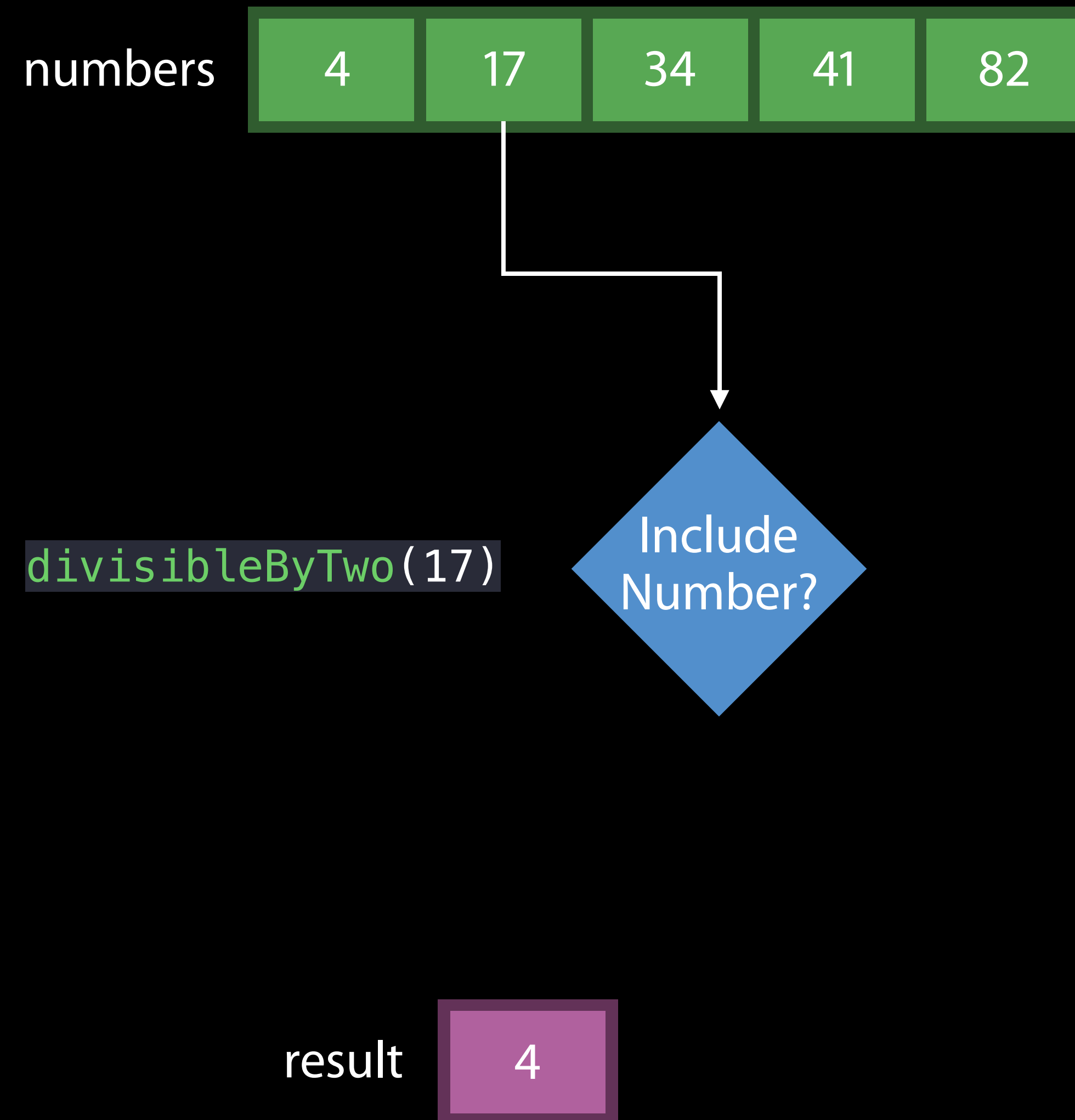
Filtering Numbers



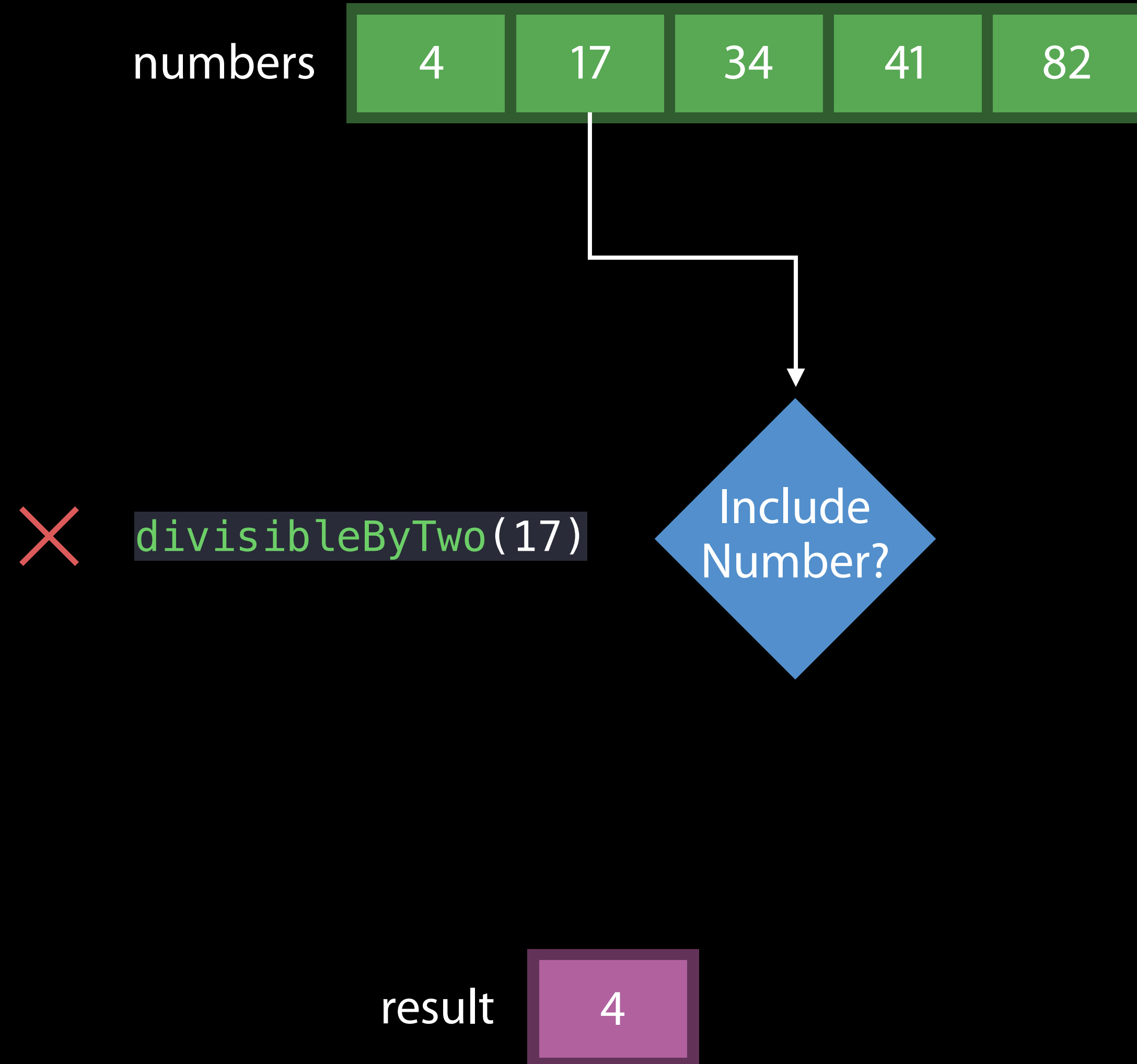
Filtering Numbers



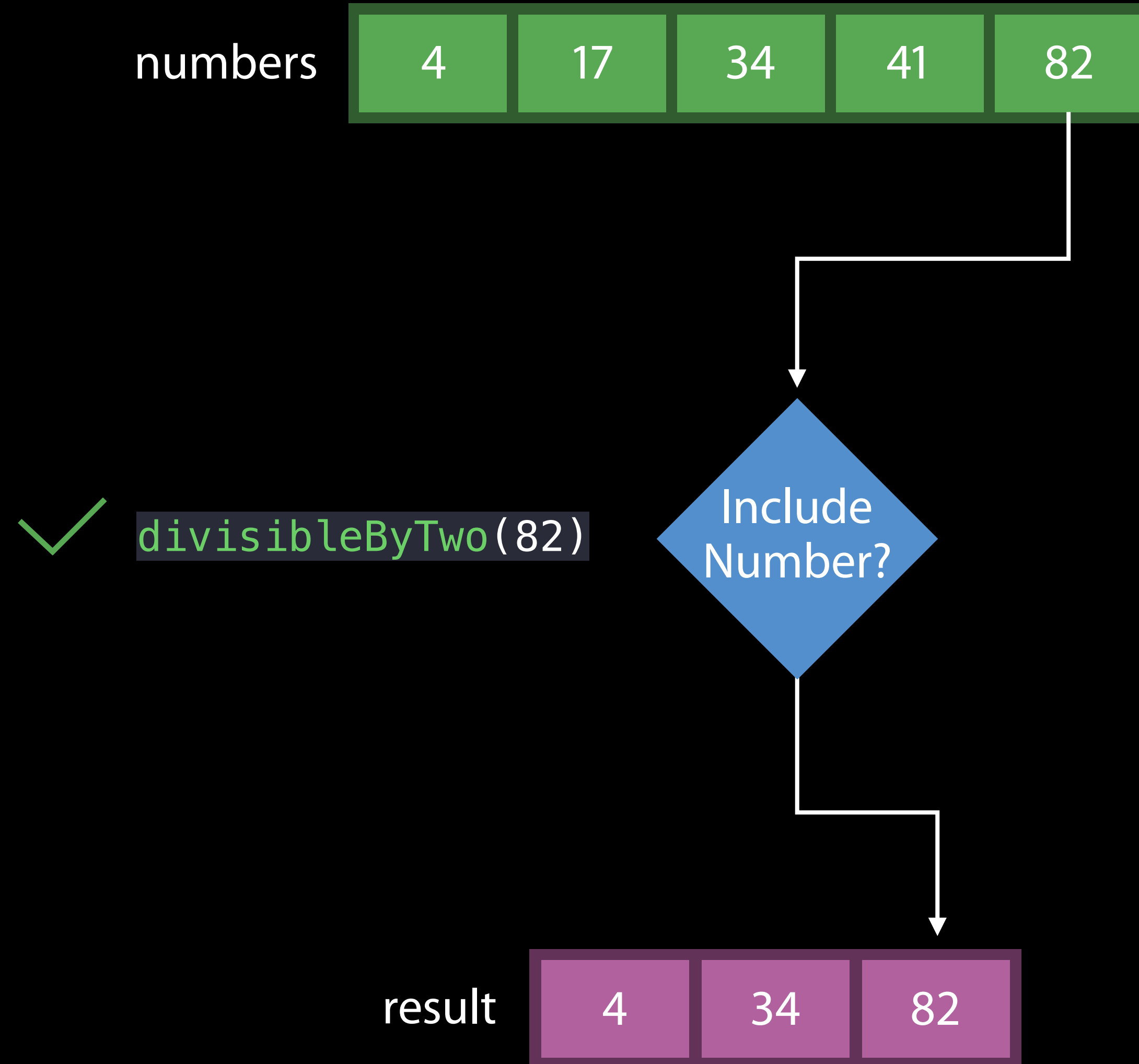
Filtering Numbers



Filtering Numbers



Filtering Numbers



Filtering Numbers

```
func filterInts(_ numbers: [Int], _ includeNumber: type) -> [Int] {...}
```

Function Types

(parameter types) -> return type

Function Types

(parameter types) -> return type

```
func sendMessage() {...}
```

Function Types

(parameter types) -> return type

```
func sendMessage() {...}
```

() -> Void

Function Types

(parameter types) -> return type

```
func firstString(havingPrefix prefix: String, in strings: [String]) -> String? {...}
```

Function Types

(parameter types) -> return type

```
func firstString(havingPrefix prefix: String, in strings: [String]) -> String? {...}
```

(String, [String]) -> String?

Functions as Parameters

```
func filterInts(_ numbers: [Int], _ includeNumber: type) -> [Int] {...}
```

Functions as Parameters

Functions as Parameters

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {  
    var result: [Int] = []  
    for number in numbers {  
  
    }  
    return result  
}
```

Functions as Parameters

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {  
    var result: [Int] = []  
    for number in numbers {  
        if includeNumber(number) {  
            result.append(number)  
        }  
    }  
    return result  
}
```

Functions as Parameters

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {  
    var result: [Int] = []  
    for number in numbers {  
        if includeNumber(number) {  
            result.append(number)  
        }  
    }  
    return result  
}
```

Functions as Arguments

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}  
  
let numbers = [4, 17, 34, 41, 82]  
func divisibleByTwo(_ number: Int) -> Bool {  
    return n % 2 == 0  
}
```

Functions as Arguments

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}
```

```
let numbers = [4, 17, 34, 41, 82]
```

```
func divisibleByTwo(_ number: Int) -> Bool {
```

```
(Int) -> Bool
```

```
}
```

Functions as Arguments

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}
```

```
let numbers = [4, 17, 34, 41, 82]
```

```
func divisibleByTwo(_ number: Int) -> Bool {  
    return number % 2 == 0  
}
```

```
let evenNumbers = filterInts(numbers, divisibleByTwo)
```

Functions as Arguments

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}
```

```
let numbers = [4, 17, 34, 41, 82]
```

```
func divisibleByTwo(_ number: Int) -> Bool {  
    return number % 2 == 0  
}
```

```
let evenNumbers = filterInts(numbers, divisibleByTwo)  
print(evenNumbers)
```

```
[4, 34, 82]
```

Functions as Arguments

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}
```

```
let numbers = [4, 17, 34, 41, 82]
```

```
func divisibleByTwo(_ number: Int) -> Bool {  
    return number % 2 == 0  
}
```

```
let evenNumbers = filterInts(numbers, divisibleByTwo)  
print(evenNumbers)
```

```
[4, 34, 82]
```


Functions as Arguments

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}
```

```
let numbers = [4, 17, 34, 41, 82]
```

```
func divisibleByTwo(_ number: Int) -> Bool {  
    return number % 2 == 0  
}
```

```
let evenNumbers = filterInts([4, 17, 34, 41, 82], divisibleByTwo)  
print(evenNumbers)
```

```
[4, 34, 82]
```

Functions as Arguments

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}
```

```
let numbers = [4, 17, 34, 41, 82]
```

```
func divisibleByTwo(_ number: Int) -> Bool {  
    return number % 2 == 0  
}
```

```
let evenNumbers = filterInts(numbers, divisibleByTwo)  
print(evenNumbers)
```

```
[4, 34, 82]
```

Functions as Arguments

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}

let numbers = [4, 17, 34, 41, 82]
func divisibleByTwo(_ number: Int) -> Bool {
    return number % 2 == 0
}

let evenNumbers = filterInts(numbers, divisibleByTwo)
print(evenNumbers)
```

```
[4, 34, 82]
```

Closure Expressions

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}

let numbers = [4, 17, 34, 41, 82]
func divisibleByTwo(_ number: Int) -> Bool {
    return number % 2 == 0
}

let evenNumbers = filterInts(numbers, divisibleByTwo)
print(evenNumbers)
```

```
[4, 34, 82]
```

Closure Expressions

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}

let numbers = [4, 17, 34, 41, 82]
func divisibleByTwo(_ number: Int) -> Bool {
    return number % 2 == 0
}

let evenNumbers = filterInts(numbers, { (number: Int) -> Bool    return number % 2 == 0 })
print(evenNumbers)
```

```
[4, 34, 82]
```

Closure Expressions

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}

let numbers = [4, 17, 34, 41, 82]
func divisibleByTwo(_ number: Int) -> Bool {
    return number % 2 == 0
}

let evenNumbers = filterInts(numbers, { (number: Int) -> Bool in return number % 2 == 0 })
print(evenNumbers)
```

```
[4, 34, 82]
```

Closure Expressions

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}

let numbers = [4, 17, 34, 41, 82]
func divisibleByTwo(_ number: Int) -> Bool {
    return number % 2 == 0
}

let evenNumbers = filterInts(numbers, { (number: Int) -> Bool in return number % 2 == 0 })
print(evenNumbers)
```

```
[4, 34, 82]
```

Type Inference in Closures

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}

let evenNumbers = filterInts(numbers, { (number: Int) -> Bool in return number % 2 == 0 })
print(evenNumbers)
```

```
[4, 34, 82]
```


Type Inference in Closures

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}  
  
let evenNumbers = filterInts(numbers, { (number: Int) -> Bool in return number % 2 == 0 })  
print(evenNumbers)
```

```
[4, 34, 82]
```

Type Inference in Closures

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}  
  
let evenNumbers = filterInts(numbers, { number in return number % 2 == 0 })  
print(evenNumbers)
```

```
[4, 34, 82]
```

Type Inference in Closures

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}  
  
let evenNumbers = filterInts(numbers, { number in return number % 2 == 0 })  
print(evenNumbers)
```

```
[4, 34, 82]
```

Type Inference in Closures

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}  
  
let evenNumbers = filterInts(numbers, { number in number % 2 == 0 })  
print(evenNumbers)
```

```
[4, 34, 82]
```

Implicit Arguments in Closures

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}  
  
let evenNumbers = filterInts(numbers, { number in number % 2 == 0 })  
print(evenNumbers)
```

```
[4, 34, 82]
```

Implicit Arguments in Closures

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}

let evenNumbers = filterInts(numbers, { $0 % 2 == 0 })
print(evenNumbers)
```

```
[4, 34, 82]
```

Implicit Arguments in Closures

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}  
  
let evenNumbers = filterInts(numbers, { $0 % 2 == 0 })  
print(evenNumbers)
```

[4, 34, 82]

Implicit Arguments in Closures

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}  
  
let evenNumbers = filterInts(numbers, { $0 % 2 == 0 })  
print(evenNumbers)
```

[4, 34, 82]

Trailing Closures

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}  
  
let evenNumbers = filterInts(numbers, { $0 % 2 == 0 })  
print(evenNumbers)
```

[4, 34, 82]

Trailing Closures

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}  
  
let evenNumbers = filterInts(numbers) { $0 % 2 == 0 }  
print(evenNumbers)
```

[4, 34, 82]

Trailing Closures

```
let evenDigitSums = filterInts(numbers) { number in
    var sum = 0, number = number
    while number > 0 {
        //... calculate sum of digits
        //... 82 is 8 + 2 = 10, which is even
    }
    return sum % 2 == 0
}
```

Trailing Closures

```
let evenDigitSums = filterInts(numbers) { number in
    var sum = 0, number = number
    while number > 0 {
        //... calculate sum of digits
        //... 82 is 8 + 2 = 10, which is even
    }
    return sum % 2 == 0
}
print(evenDigitSums)
```

[4, 82]

Filtering Strings?

```
let names = ["Lily", "Santiago", "Aadya", "Jack", "Anna"]  
  
let shortNames = filterStrings(names) { name in  
    name.characters.count < 5  
}
```

Filtering Strings?

```
let names = ["Lily", "Santiago", "Aadya", "Jack", "Anna"]

let shortNames = filterStrings(names) { name in
    name.characters.count < 5
}

print(shortNames)
```

[Lily, Jack, Anna]

Filtering Numbers

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {  
    var result: [Int] = []  
    for number in numbers {  
        if includeNumber(number) {  
            result.append(number)  
        }  
    }  
    return result  
}
```

Filtering Numbers

```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {  
    var result: [Int] = []  
    for number in numbers {  
        if includeNumber(number) {  
            result.append(number)  
        }  
    }  
    return result  
}
```


Filtering Numbers

```
func filterStrings(_ strings: [Int], _ includeString: (Int) -> Bool) -> [Int] {  
    var result: [Int] = []  
    for string in strings {  
        if includeString(string) {  
            result.append(string)  
        }  
    }  
    return result  
}
```

Filtering Numbers

```
func filterStrings(_ strings: [Int], _ includeString: (Int) -> Bool) -> [Int] {  
    var result: [Int] = []  
    for string in strings {  
        if includeString(string) {  
            result.append(string)  
        }  
    }  
    return result  
}
```

Filtering Strings

```
func filterStrings(_ strings: [String], _ includeString: (String) -> Bool) -> [String] {  
    var result: [String] = []  
    for string in strings {  
        if includeString(string) {  
            result.append(string)  
        }  
    }  
    return result  
}
```

Filtering Strings

```
func filterStrings(_ strings: [String], _ includeString: (String) -> Bool) -> [String] {  
    var result: [String] = []  
    for string in strings {  
        if includeString(string) {  
            result.append(string)  
        }  
    }  
    return result  
}
```

Generic Filtering

```
func filter(_ source: [Element], _ includeElement: (Element) -> Bool) -> [Element] {  
    var result: [Element] = []  
    for element in source {  
        if includeElement(element) {  
            result.append(element)  
        }  
    }  
    return result  
}
```

Generic Type Parameters

```
func filter<Element>(_ source: [Element], _ includeElement: (Element) -> Bool) -> [Element] {  
    var result: [Element] = []  
    for element in source {  
        if includeElement(element) {  
            result.append(element)  
        }  
    }  
    return result  
}
```

Generic Type Parameters

```
func filter<Element>(_ source: [Element], _ includeElement: (Element) -> Bool) -> [Element] {  
    var result: [Element] = []  
    for element in source {  
        if includeElement(element) {  
            result.append(element)  
        }  
    }  
    return result  
}
```

Calling Generic Functions

```
func filter<Element>(_ source: [Element], _ includeElement: (Element) -> Bool) -> [Element] {  
    var result: [Element] = []  
    for element in source {  
        if includeElement(element) {  
            result.append(element)  
        }  
    }  
    return result  
}
```


Calling Generic Functions

```
func filter<Element>(_ source: [Element], _ includeElement: (Element) -> Bool) -> [Element] {  
    var result: [Element] = []  
    for element in source {  
        if includeElement(element) {  
            result.append(element)  
        }  
    }  
    return result  
}
```

```
let evenNumbers = filter(numbers) { $0 % 2 == 0 }
```

```
let shortNames = filter(names) { name in name.characters.count < 5 }
```

Filter and Map Methods

```
let names = ["Lily", "Santiago", "Aadya", "Jack", "Anna", "Andrés"]
```

```
let shortNames = names.filter { name in name.characters.count < 5 }
```

Filter and Map Methods

```
let names = ["Lily", "Santiago", "Aadya", "Jack", "Anna", "Andrés"]  
  
let shortNames = names.filter { name in name.characters.count < 5 }  
print(shortNames)
```

```
[Lily, Jack, Anna]
```

Filter and Map Methods

```
let names = ["Lily", "Santiago", "Aadya", "Jack", "Anna", "Andrés"]  
  
let shortNames = names.filter { name in name.characters.count < 5 }  
print(shortNames)  
  
let capitalizedShortNames = shortNames.map { name in name.uppercased() }
```

```
[Lily, Jack, Anna]
```

Filter and Map Methods

```
let names = ["Lily", "Santiago", "Aadya", "Jack", "Anna", "Andrés"]

let shortNames = names.filter { name in name.characters.count < 5 }
print(shortNames)

let capitalizedShortNames = shortNames.map { name in name.uppercased() }
print(capitalizedShortNames)
```

[Lily, Jack, Anna]

[LILY, JACK, ANNA]

Filter and Map Methods

```
let names = ["Lily", "Santiago", "Aadya", "Jack", "Anna", "Andrés"]
```

[illegible]

Filter and Map Methods

```
let names = ["Lily", "Santiago", "Aadya", "Jack", "Anna", "Andrés"]
```

[illegible]

Filter and Map Methods

```
let names = ["Lily", "Santiago", "Aadya", "Jack", "Anna", "Andrés"]

let capitalizedShortNames = names.filter { name in name.characters.count < 5 }
                                   .map { name in name.uppercased() }

print(capitalizedShortNames)
```

```
[LILY, JACK, ANNA]
```


Custom Types

Alex Martini Swift Documentation Engineer

Structures

```
struct Rectangle {  
    var width = 12  
    var height = 10  
}
```

```
var rectangle = Rectangle()  
rectangle.height = 4
```

Structures

```
struct Rectangle {  
    var width = 12  
    var height = 10  
}
```

```
var rectangle = Rectangle()  
rectangle.height = 4
```

Structures

```
struct Rectangle {  
    var width = 12  
    var height = 10  
}
```

```
var rectangle = Rectangle()  
rectangle.height = 4
```

Structures

```
struct Rectangle {  
    var width = 12  
    var height = 10  
}
```

```
var rectangle = Rectangle()  
rectangle.height = 4
```

Structures

```
struct Rectangle {  
    var width: Int  
    var height: Int  
}
```

```
var rectangle = Rectangle(width: 4, height: 5)
```

Structures

```
struct Rectangle {  
    var width: Int  
    var height: Int  
}
```

```
var rectangle = Rectangle(width: 4, height: 5)
```

Properties

```
struct Rectangle {  
    var width: Int  
    var height: Int  
    var area: Int  
}
```


Computed Properties

```
struct Rectangle {  
    var width: Int  
    var height: Int  
    var area: Int {  
        return width * height  
    }  
}
```

Computed Properties

```
struct Rectangle {  
    var width: Int  
    var height: Int  
    var area: Int {  
        return width * height  
    }  
}  
  
let rectangle = Rectangle(width: 4, height: 5)  
print("Width is \(rectangle.width) and area is \(rectangle.area).")
```

Computed Properties

```
struct Rectangle {  
    var width: Int  
    var height: Int  
    var area: Int {  
        return width * height  
    }  
}  
  
let rectangle = Rectangle(width: 4, height: 5)  
print("Width is \(rectangle.width) and area is \(rectangle.area).")
```

Width is 4 and area is 20.

Computed Properties

```
struct Rectangle {  
    var width: Int  
    var height: Int  
    var area: Int {  
        return width * height  
    }  
}  
  
let rectangle = Rectangle(width: 4, height: 5)  
print("Width is \(rectangle.width) and area is \(rectangle.area).")
```

Width is 4 and area is 20.

Computed Properties

```
struct Rectangle {  
    var width: Int  
    var height: Int  
}
```

Methods

```
struct Rectangle {  
    var width: Int  
    var height: Int  
  
    func fitsInside(_ other: Rectangle) -> Bool {  
        return (width < other.width) && (height < other.height)  
    }  
}
```

Methods

```
struct Rectangle {  
    var width: Int  
    var height: Int  
  
    func fitsInside(_ other: Rectangle) -> Bool {  
        return (width < other.width) && (height < other.height)  
    }  
}
```

```
let small = Rectangle(width: 1, height: 2)  
let large = Rectangle(width: 5, height: 5)  
small.fitsInside(large) // Returns true
```

Methods

```
struct Rectangle {  
    var width: Int  
    var height: Int  
  
    func fitsInside(_ other: Rectangle) -> Bool {  
        return (width < other.width) && (height < other.height)  
    }  
}
```

```
let small = Rectangle(width: 1, height: 2)  
let large = Rectangle(width: 5, height: 5)  
small.fitsInside(large) // Returns true
```


Creating a Rectangle

```
struct Rectangle {  
    var width: Int  
    var height: Int  
}
```

```
var rectangle = Rectangle(width: 4, height: 5)
```

Initializers

```
struct Rectangle {  
    var width: Int  
    var height: Int
```

```
    init(width: Int, height: Int) {  
        self.width = width  
        self.height = height  
    }  
}
```

```
var rectangle = Rectangle(width: 4, height: 5)
```

Initializers

```
struct Rectangle {  
    var width: Int  
    var height: Int  
  
    init(width: Int, height: Int) {  
        self.width = width  
        self.height = height  
    }  
}  
  
var rectangle = Rectangle(width: 4, height: 5)
```

Organizing Your Code

```
struct Rectangle {  
    var width: Int  
    var height: Int  
  
    func fitsInside(_ other: Rectangle) -> Bool {  
        return (width < other.width) && (height < other.height)  
    }  
  
    var area: Int {  
        return width * height  
    }  
}
```

Extensions

```
struct Rectangle {  
    var width: Int  
    var height: Int  
}  
  
extension Rectangle {  
    func fitsInside(_ other: Rectangle) -> Bool {...}  
    var area {...}  
}
```

Extensions

```
struct Rectangle {  
    var width: Int  
    var height: Int  
}
```

```
extension Rectangle {  
    func fitsInside(_ other: Rectangle) -> Bool {...}  
    var area {...}  
}
```

Extensions

```
struct Rectangle {  
    var width: Int  
    var height: Int  
}
```

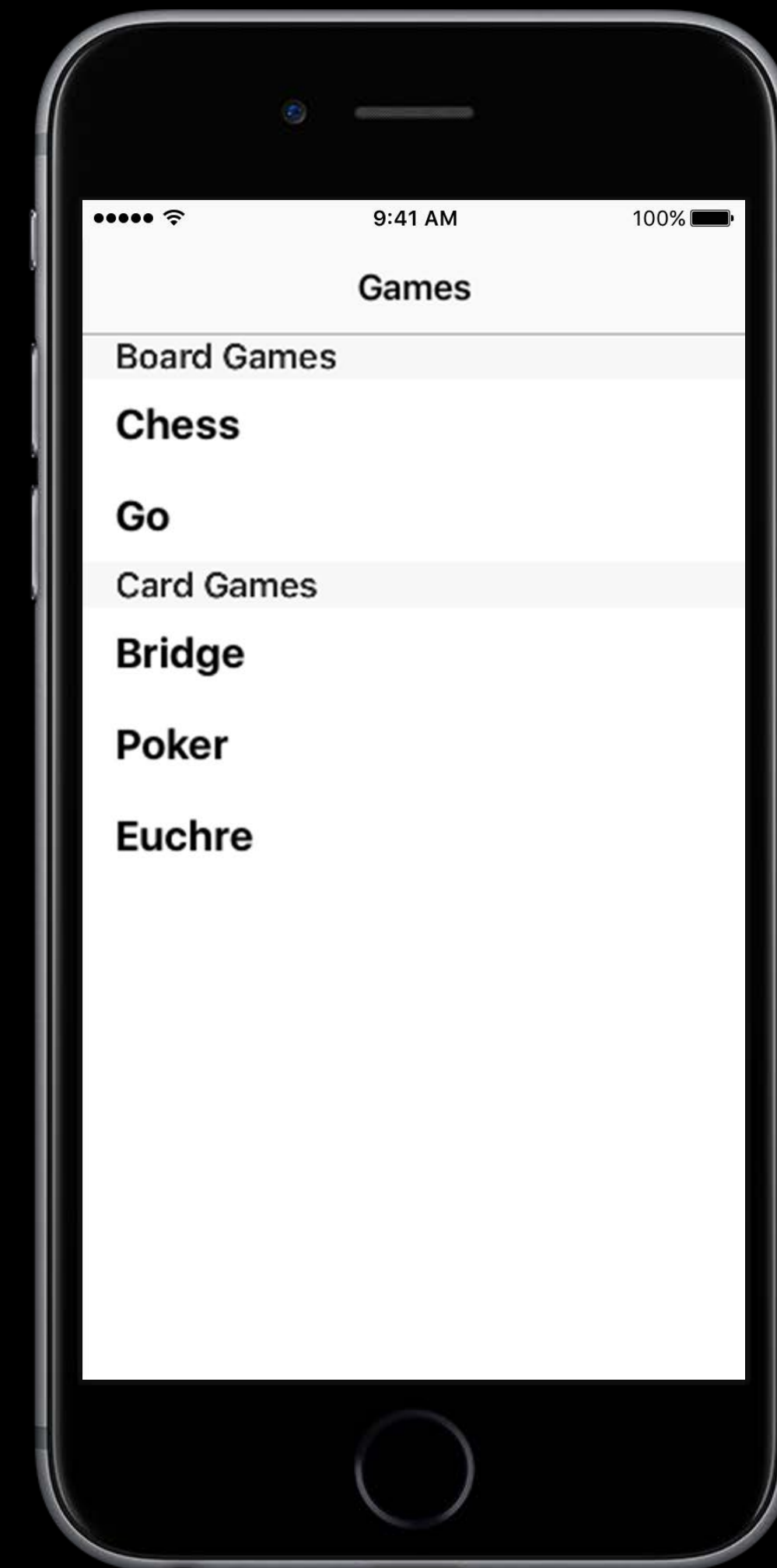
```
extension Rectangle {  
    func fitsInside(_ other: Rectangle) -> Bool {...}  
    var area {...}  
}
```

Generic Types

```
struct NamedArray<Element> {  
    var name: String  
    var items: [Element]  
}
```


Generic Types

```
struct NamedArray<Element> {  
    var name: String  
    var items: [Element]  
}
```



Generic Types

```
struct NamedArray<Element> {  
    var name: String  
    var items: [Element]  
}
```

```
let boardGames: NamedArray<String> = NamedArray(name: "Board Games", items: ["Chess", "Go"])  
let primes: NamedArray<Int> = NamedArray(name: "Primes", items: [1, 3, 5, 7, 13])
```

Generic Types

```
struct NamedArray<Element> {  
    var name: String  
    var items: [Element]  
}
```

```
let boardGames: NamedArray<String> = NamedArray(name: "Board Games", items: ["Chess", "Go"])  
let primes: NamedArray<Int> = NamedArray(name: "Primes", items: [1, 3, 5, 7, 13])
```

Generic Types

```
struct NamedArray<Element> {  
    var name: String  
    var items: [Element]  
}
```

```
let boardGames = NamedArray(name: "Board Games", items: ["Chess", "Go"])  
let primes = NamedArray(name: "Primes", items: [1, 3, 5, 7, 13])
```

Generic Types

```
struct NamedArray<Element> {  
    var name: String  
    var items: [Element]  
}
```

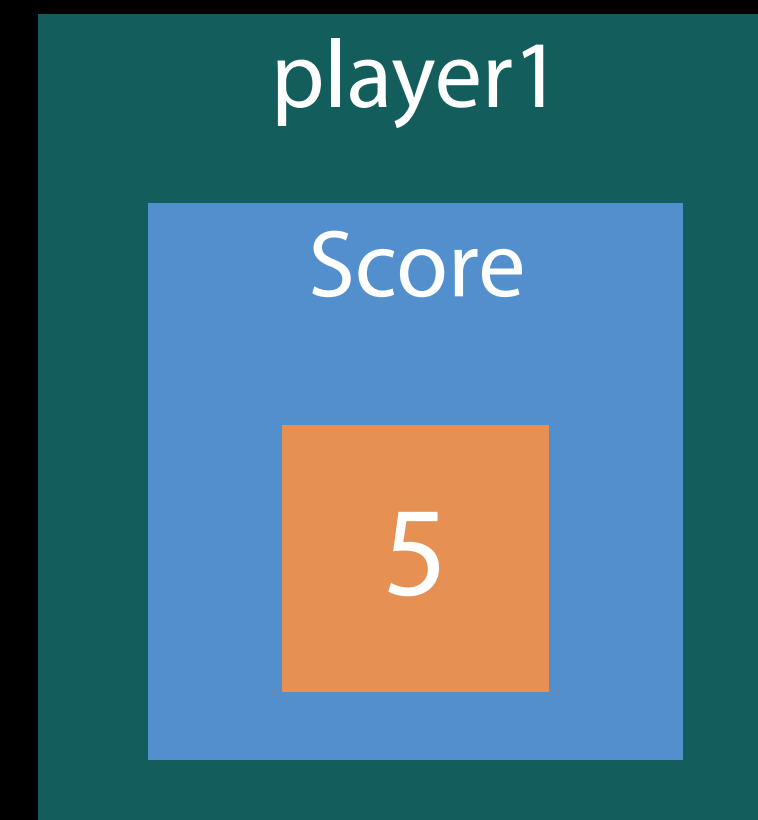
```
let boardGames = NamedArray(name: "Board Games", items: ["Chess", "Go"])  
let primes = NamedArray(name: "Primes", items: [1, 3, 5, 7, 13])
```

Classes

```
class ScoreLogFile {  
    var highScores: [Score]  
    func record(score: Score, for player: Player) -> Void {...}  
}
```

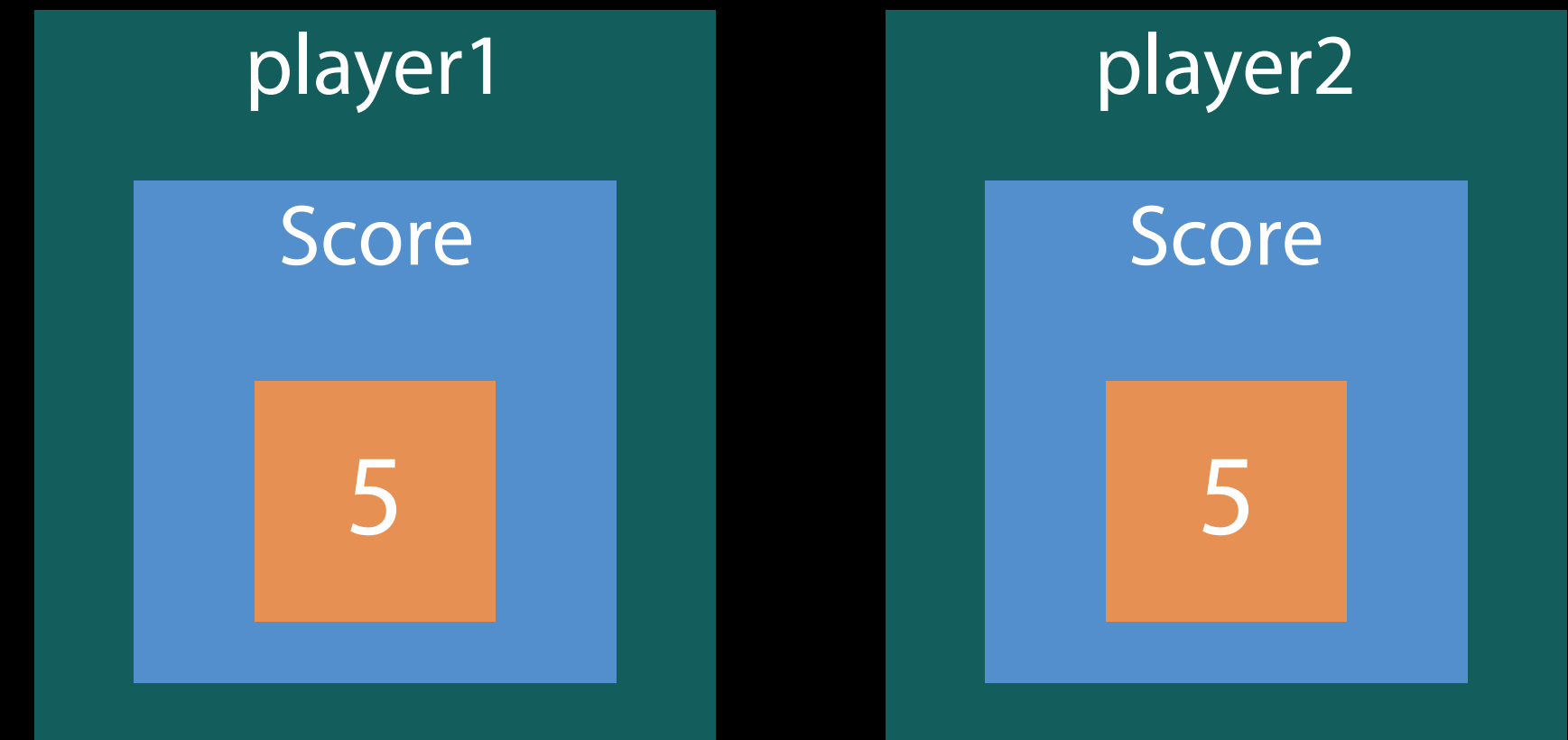
Values and References

```
struct Score { var value: Int }  
player1.score = Score(value: 5)
```



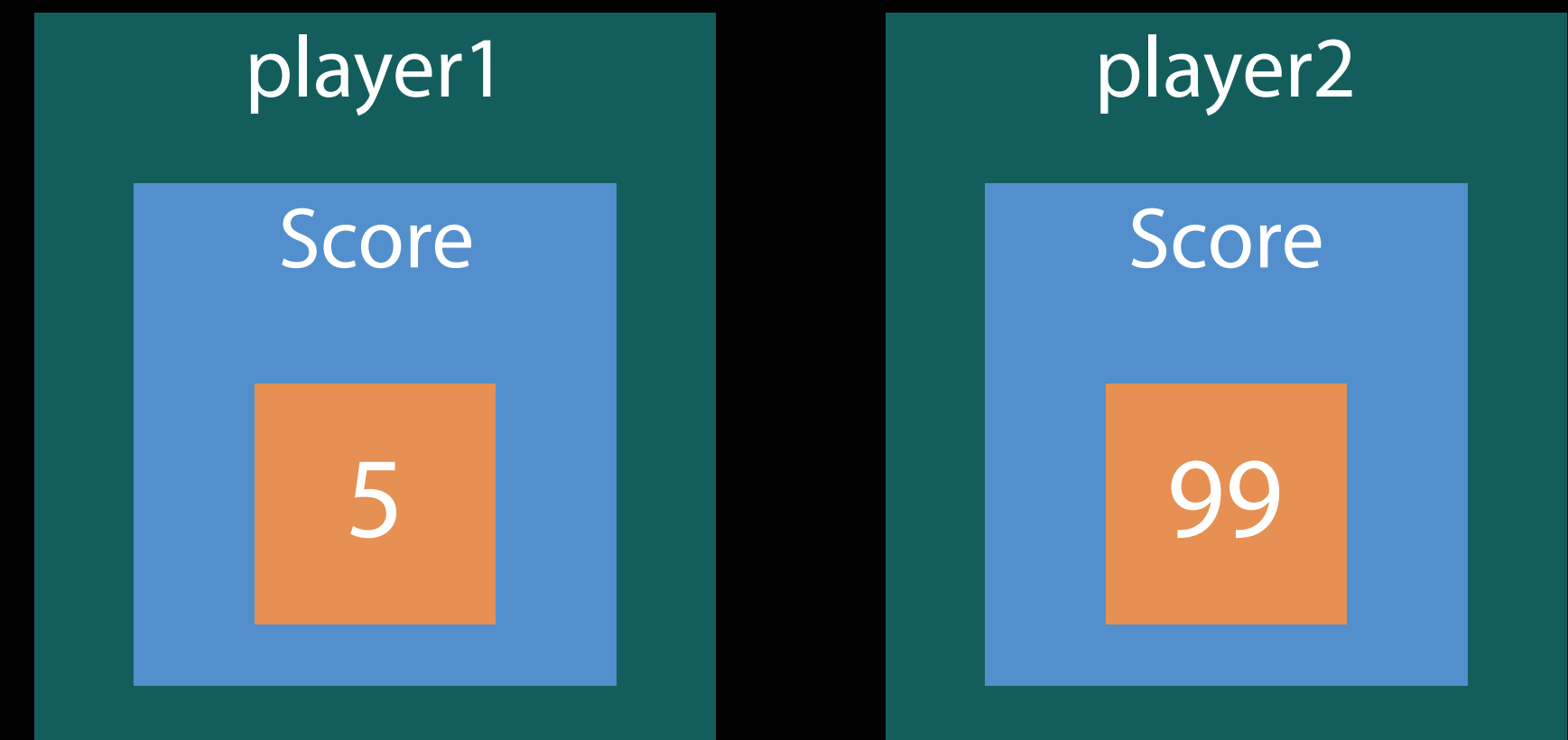
Values and References

```
struct Score { var value: Int }  
player1.score = Score(value: 5)  
player2.score = player1.score
```



Values and References

```
struct Score { var value: Int }  
player1.score = Score(value: 5)  
player2.score = player1.score  
player2.score.value = 99
```



Values and References

```
struct Score { var value: Int }  
player1.score = Score(value: 5)  
player2.score = player1.score  
player2.score.value = 99
```

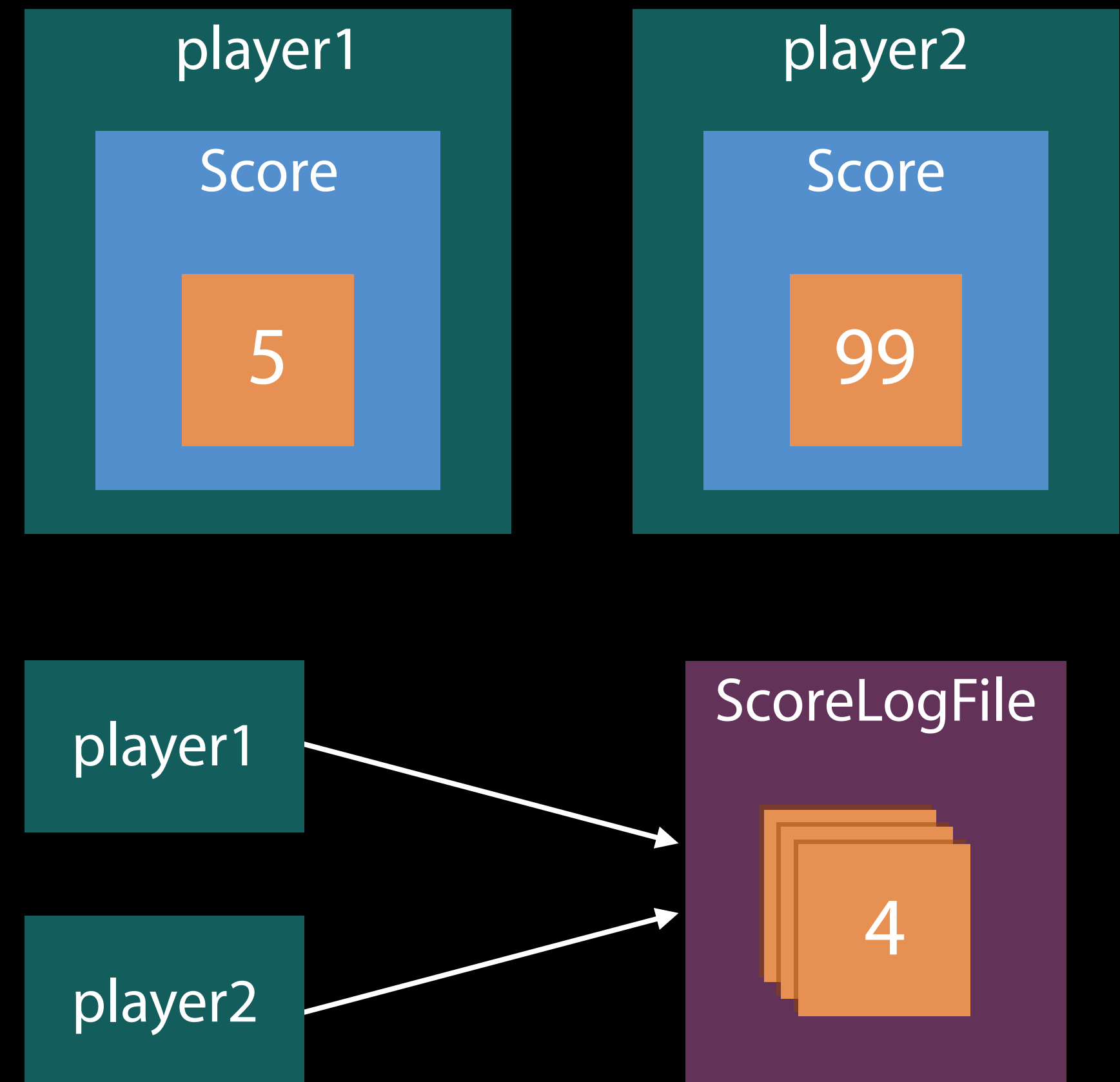
```
class ScoreLogFile {...}  
let scoreLog = ScoreLogFile()
```



Values and References

```
struct Score { var value: Int }  
player1.score = Score(value: 5)  
player2.score = player1.score  
player2.score.value = 99
```

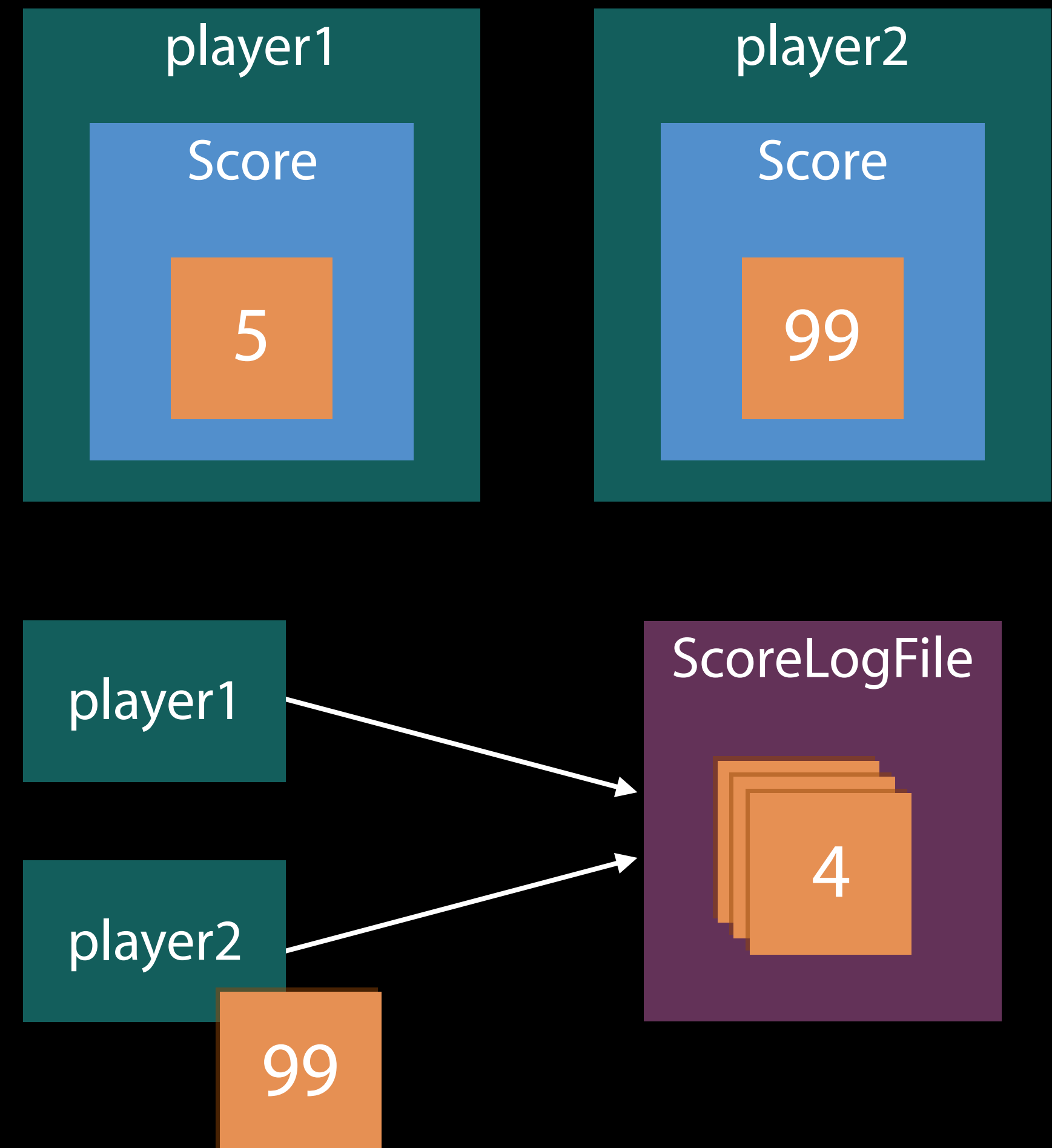
```
class ScoreLogFile {...}  
let scoreLog = ScoreLogFile()  
player1.scoreLog = scoreLog  
player2.scoreLog = scoreLog  
player2.logCurrentScore()
```



Values and References

```
struct Score { var value: Int }  
player1.score = Score(value: 5)  
player2.score = player1.score  
player2.score.value = 99
```

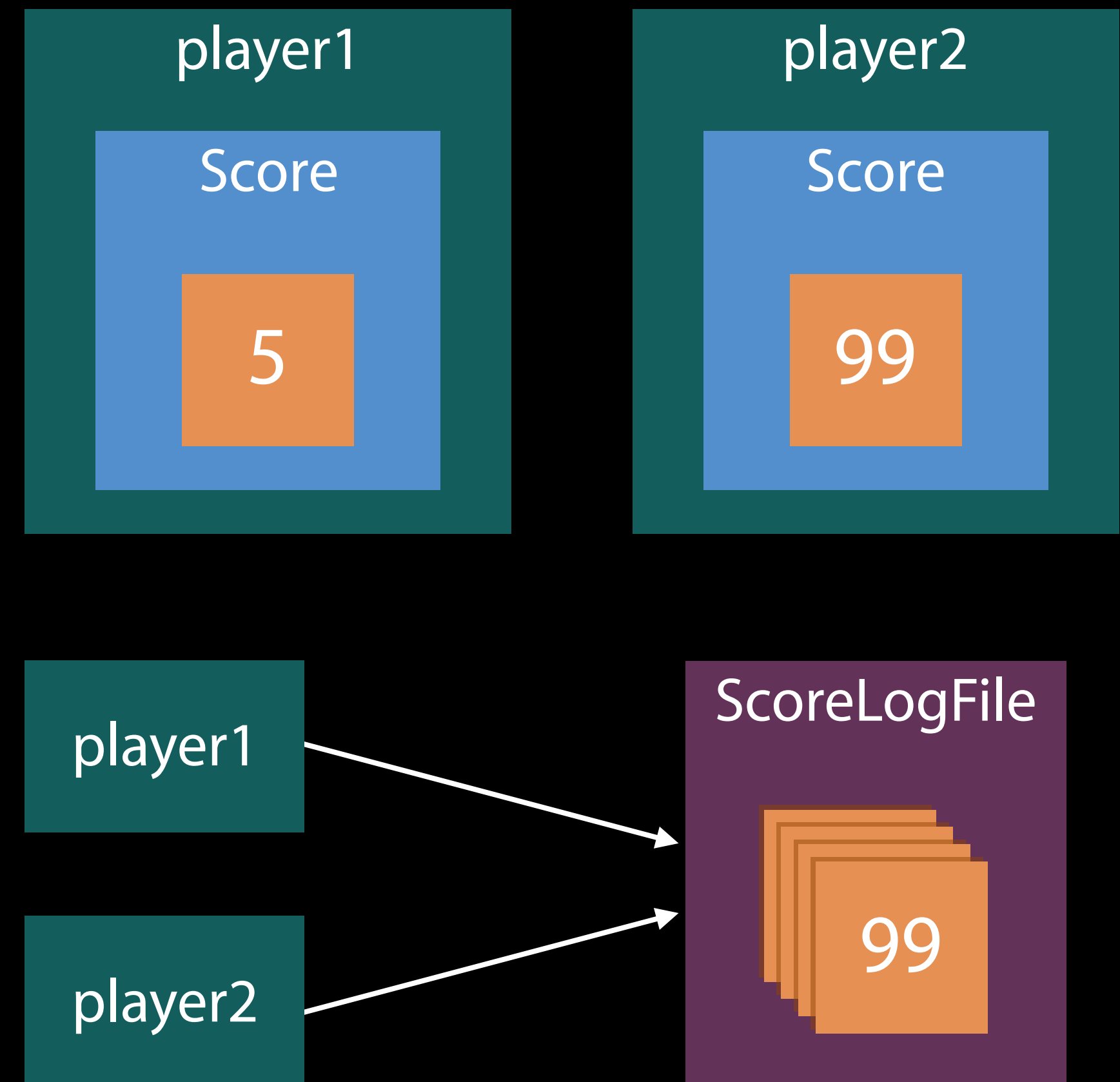
```
class ScoreLogFile {...}  
let scoreLog = ScoreLogFile()  
player1.scoreLog = scoreLog  
player2.scoreLog = scoreLog  
player2.logCurrentScore()
```



Values and References

```
struct Score { var value: Int }  
player1.score = Score(value: 5)  
player2.score = player1.score  
player2.score.value = 99
```

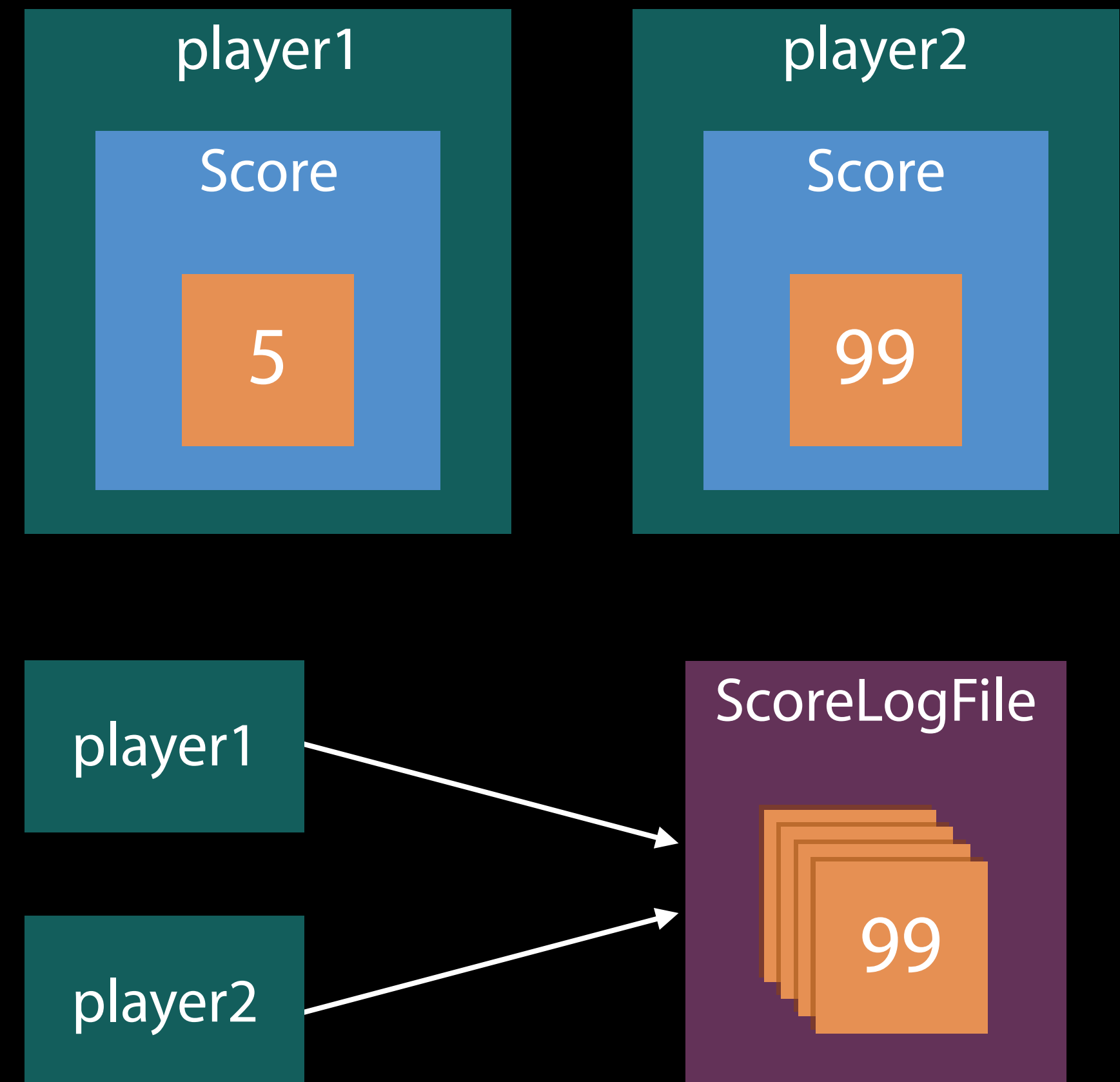
```
class ScoreLogFile {...}  
let scoreLog = ScoreLogFile()  
player1.scoreLog = scoreLog  
player2.scoreLog = scoreLog  
player2.logCurrentScore()
```



Values and References

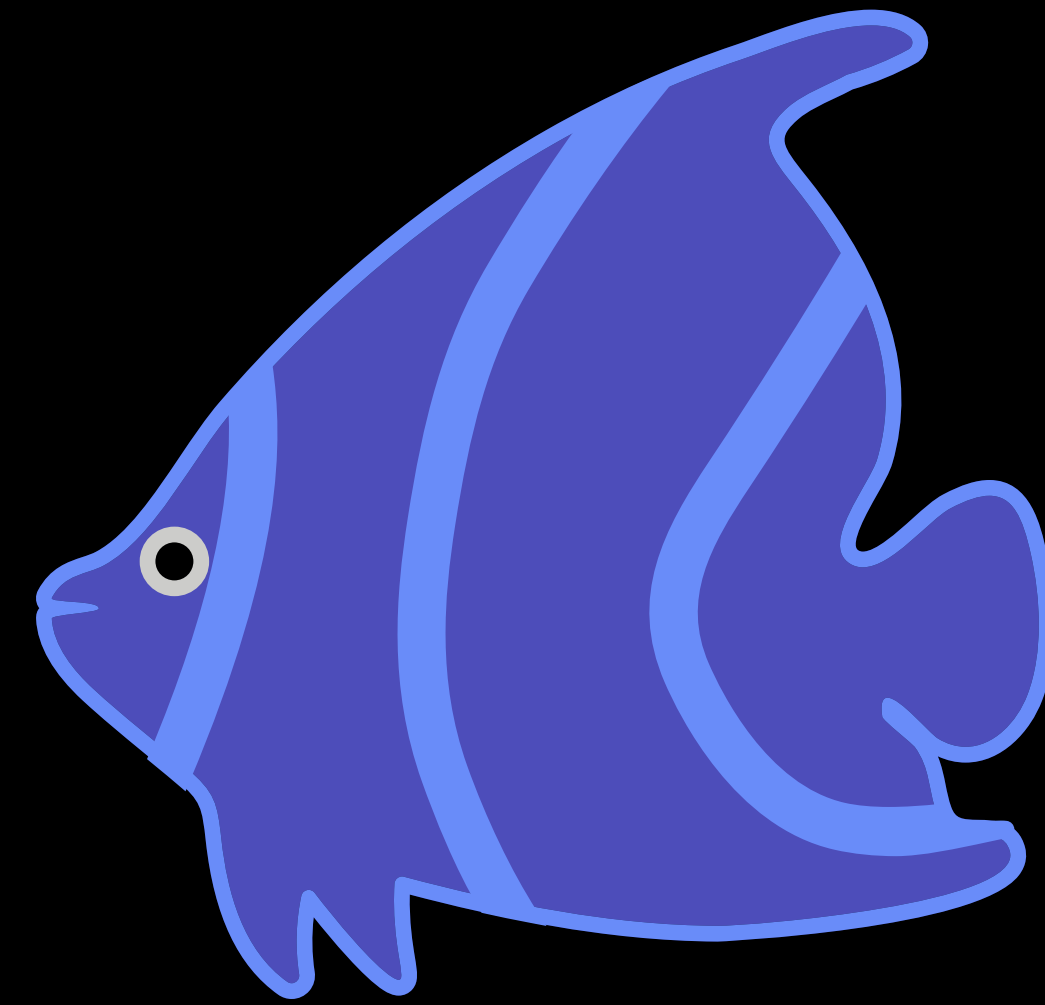
```
struct Score { var value: Int }  
player1.score = Score(value: 5)  
player2.score = player1.score  
player2.score.value = 99
```

```
class ScoreLogFile {...}  
let scoreLog = ScoreLogFile()  
player1.scoreLog = scoreLog  
player2.scoreLog = scoreLog  
player2.logCurrentScore()
```



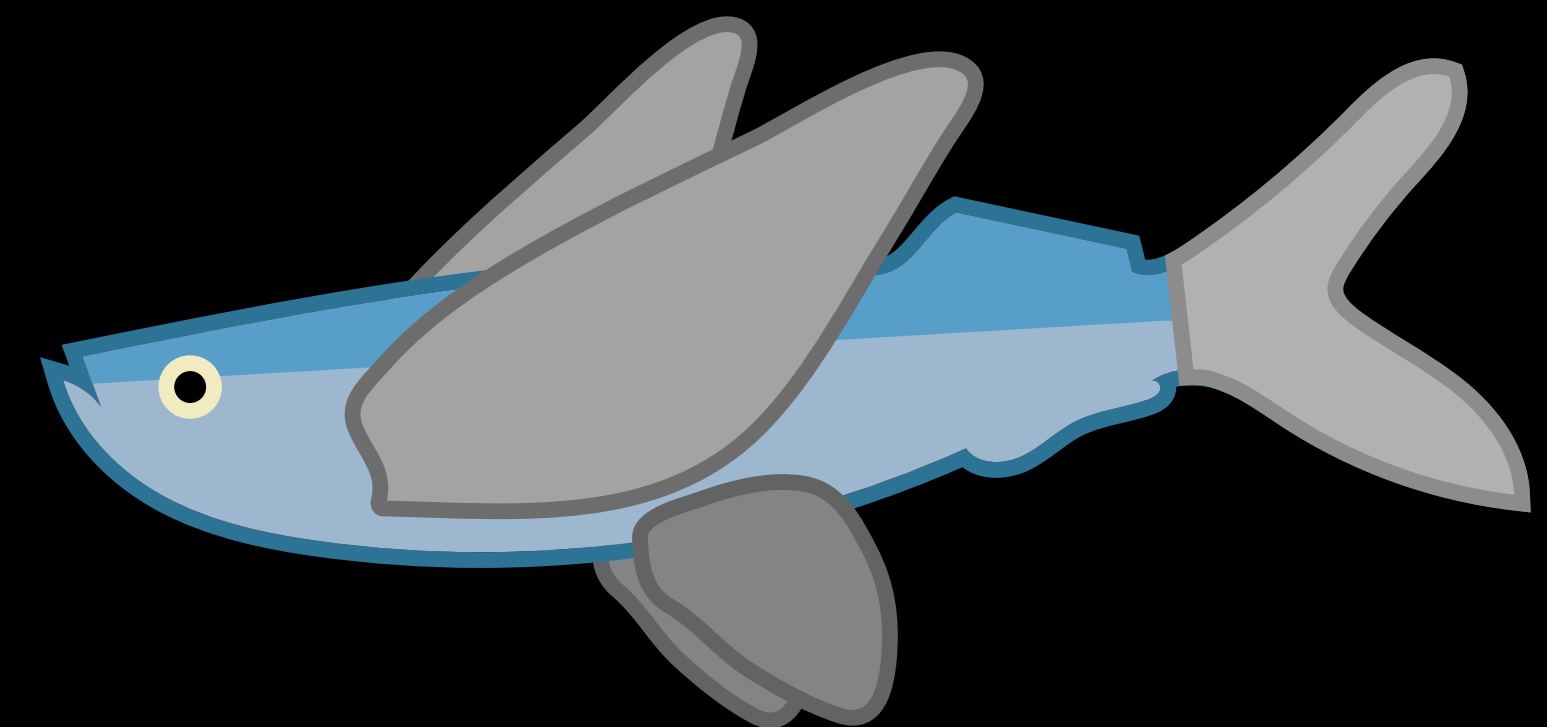
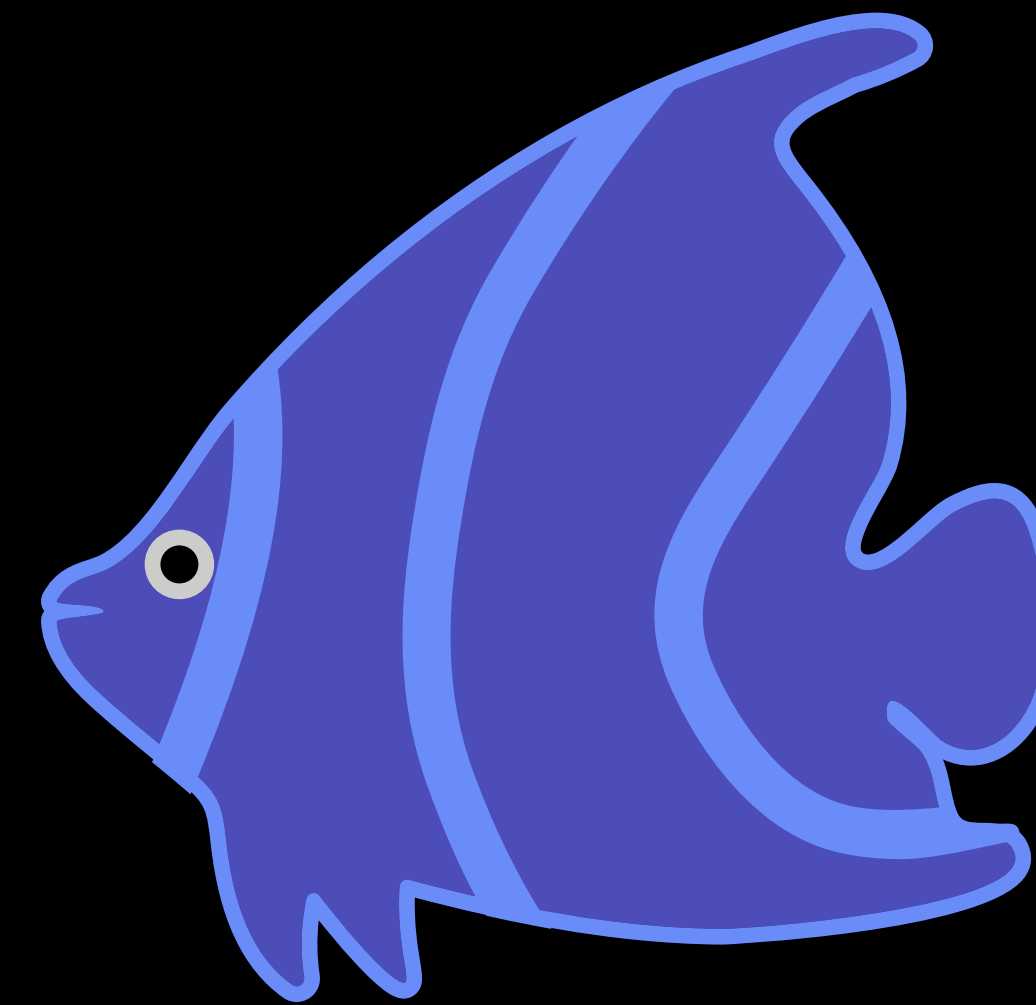
Subclasses

```
class Fish {  
    func swim() {  
        print("I'm swimming.")  
    }  
}
```



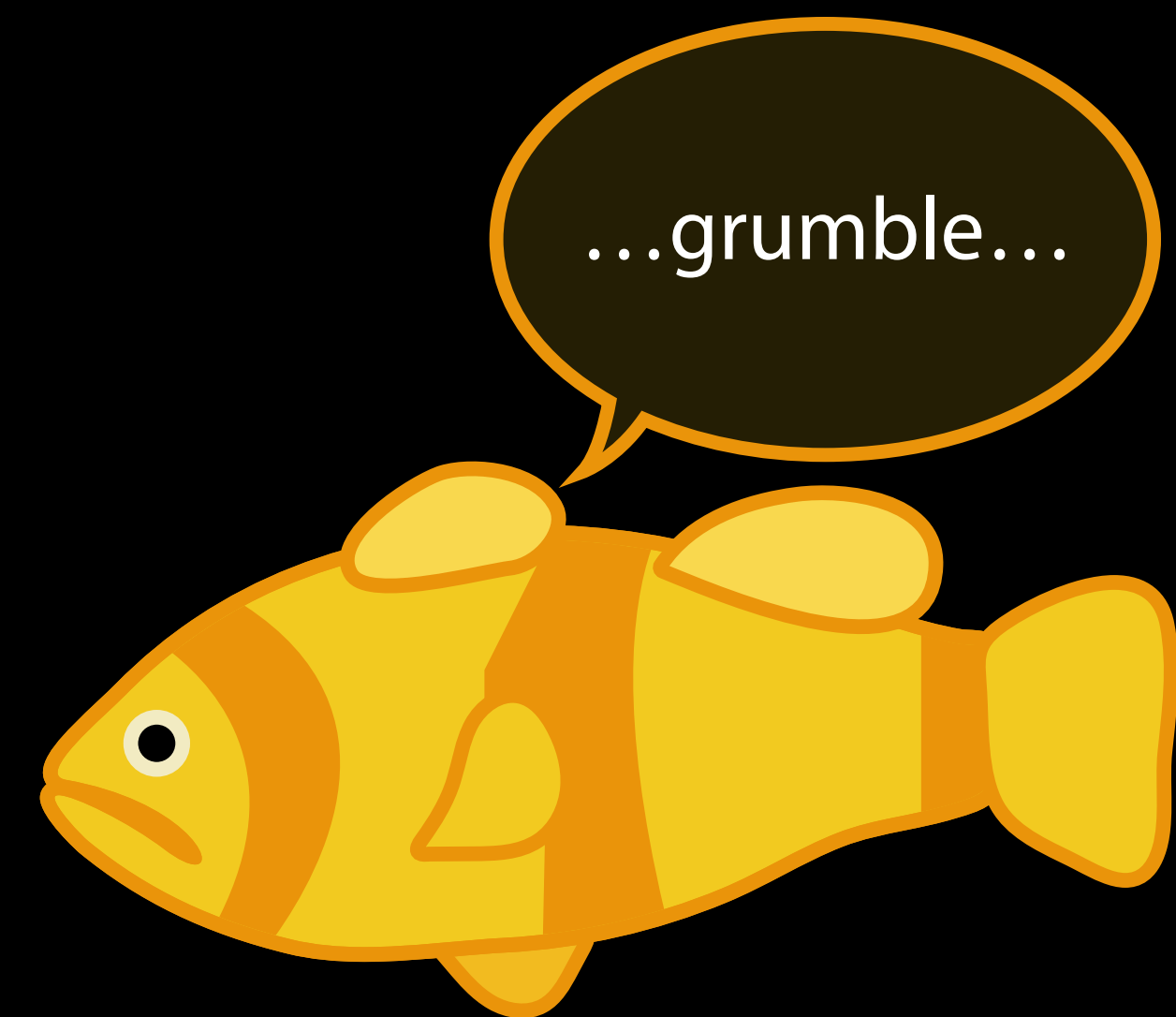
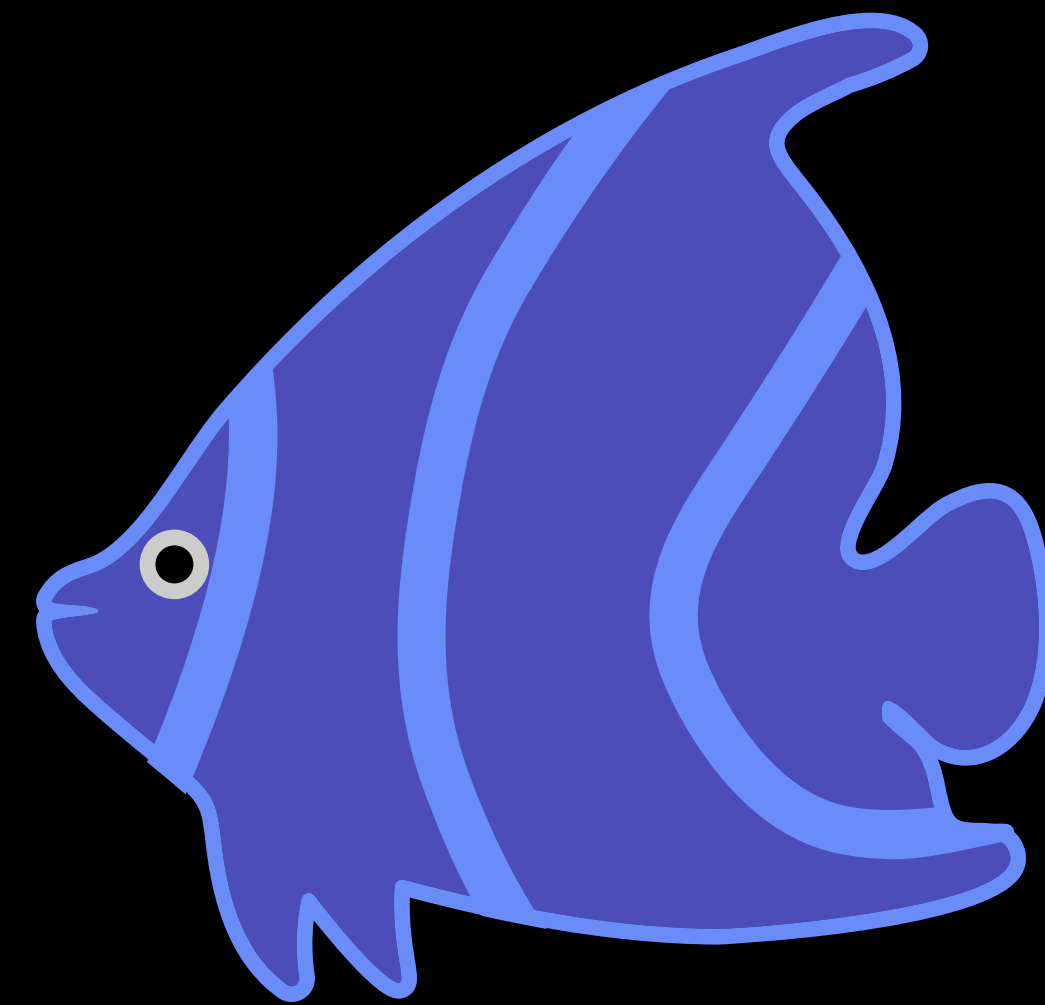
Subclasses Can Add Functionality

```
class Fish {  
    func swim() {  
        print("I'm swimming.")  
    }  
}  
  
class FlyingFish: Fish {  
    func fly() {  
        print("Flying throught the air!")  
    }  
    // Inherits swim() with no changes.  
}
```



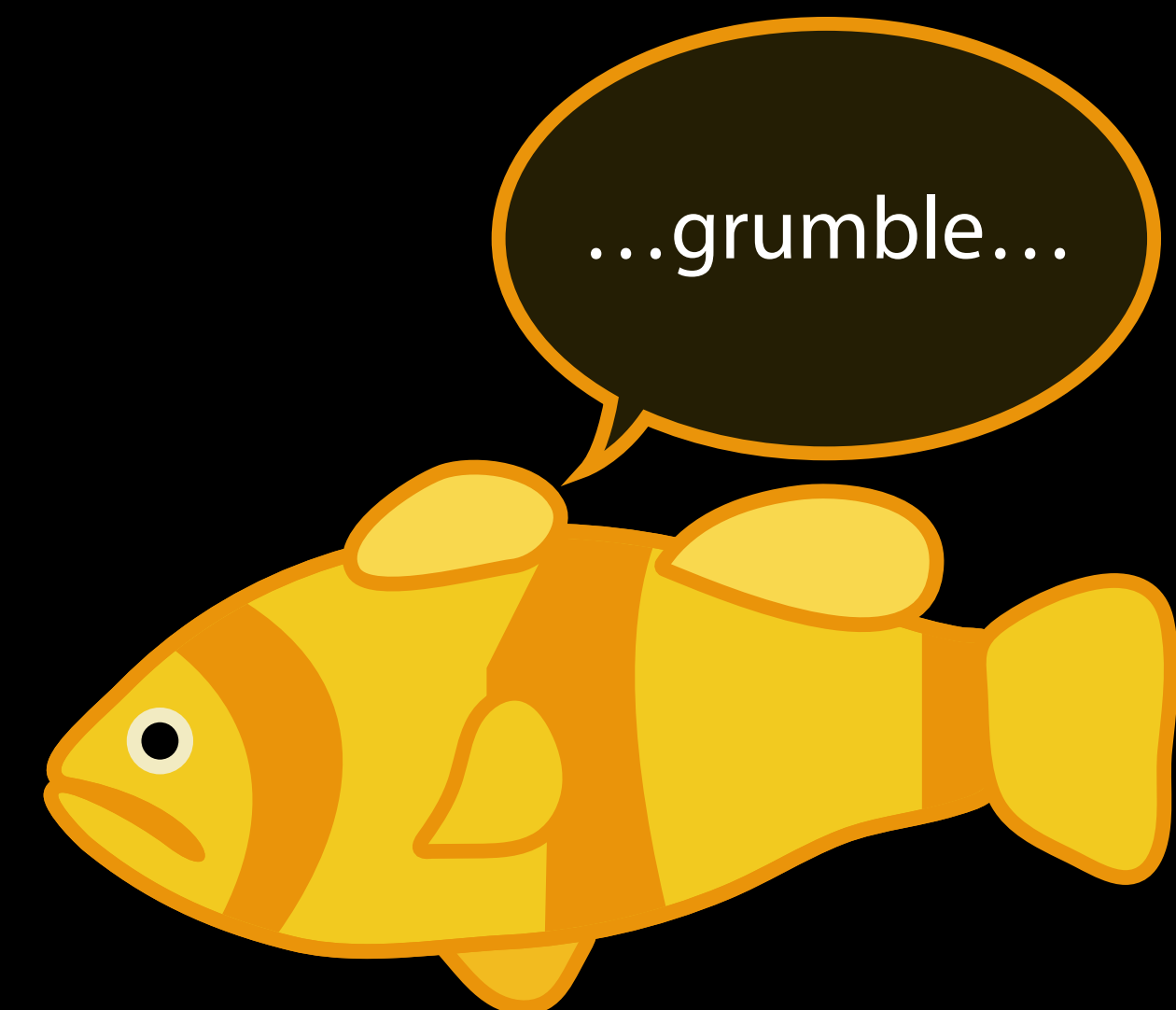
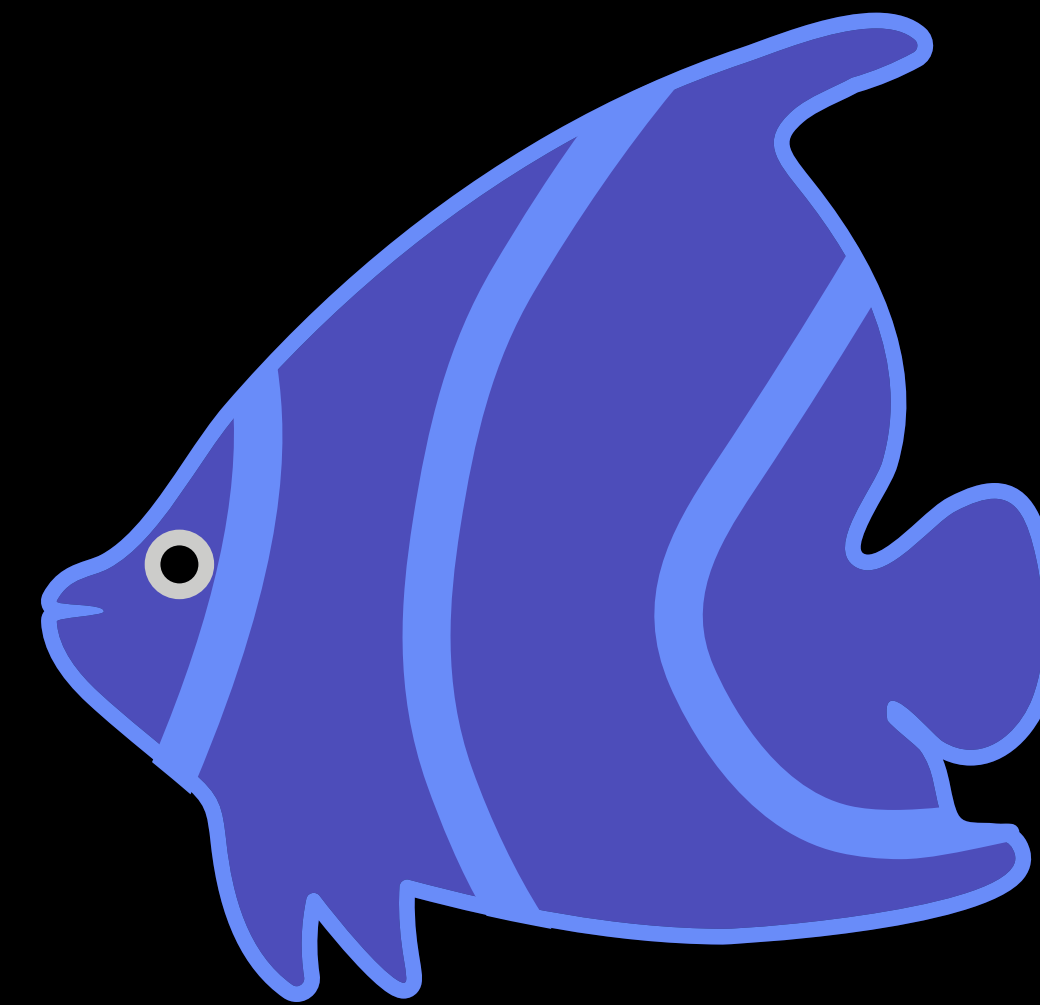
Subclasses Can Override Functionality

```
class Fish {  
    func swim() {  
        print("I'm swimming.")  
    }  
}  
  
class ComplainingFish: Fish {  
    func swim() {  
        print("Grumble grumble grumble...")  
        super.swim()  
    }  
}
```



Subclasses Can Override Functionality

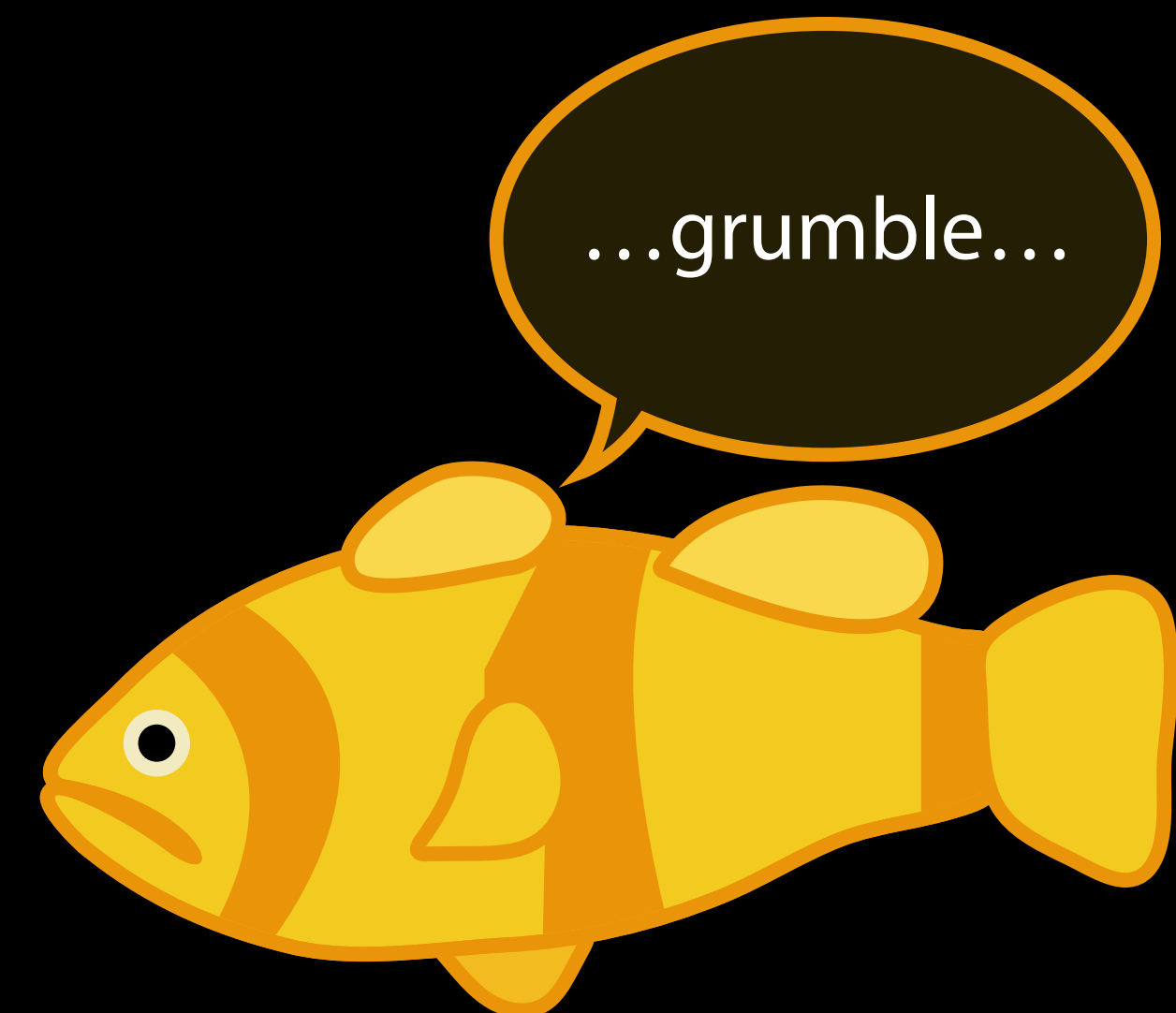
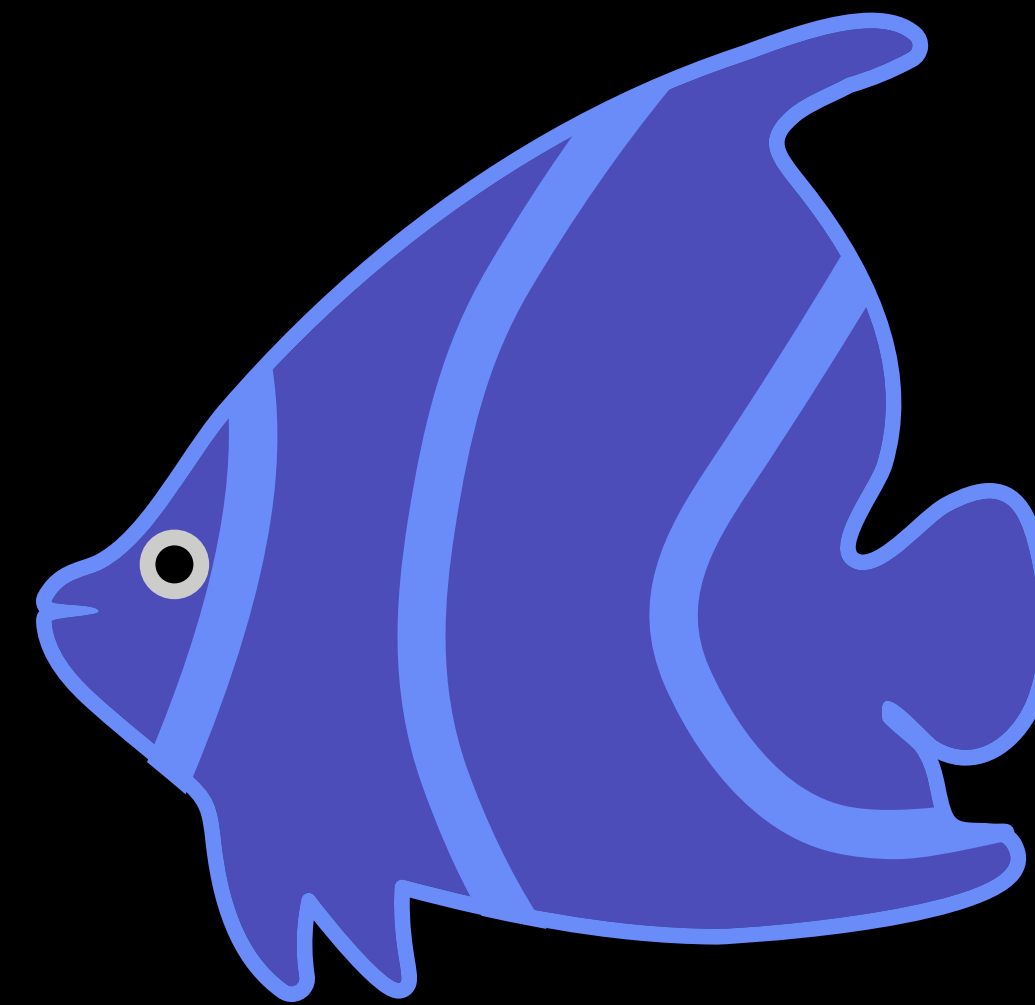
```
class Fish {  
    func swim() {  
        print("I'm swimming.")  
    }  
}  
  
class ComplainingFish: Fish {  
    func swim() {  
        print("Grumble grumble grumble...")  
        super.swim()  
    }  
}
```



Subclasses Can Override Functionality

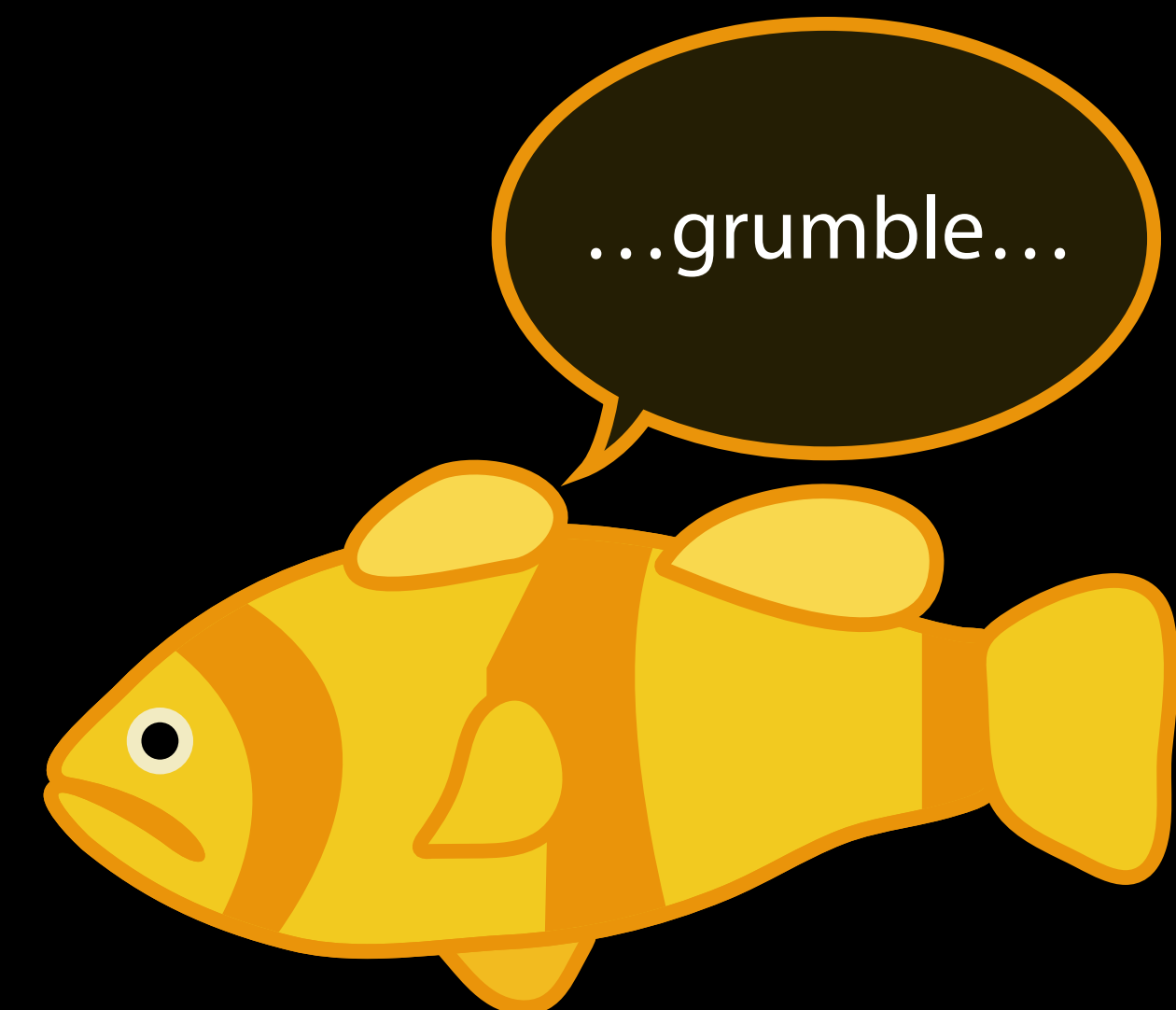
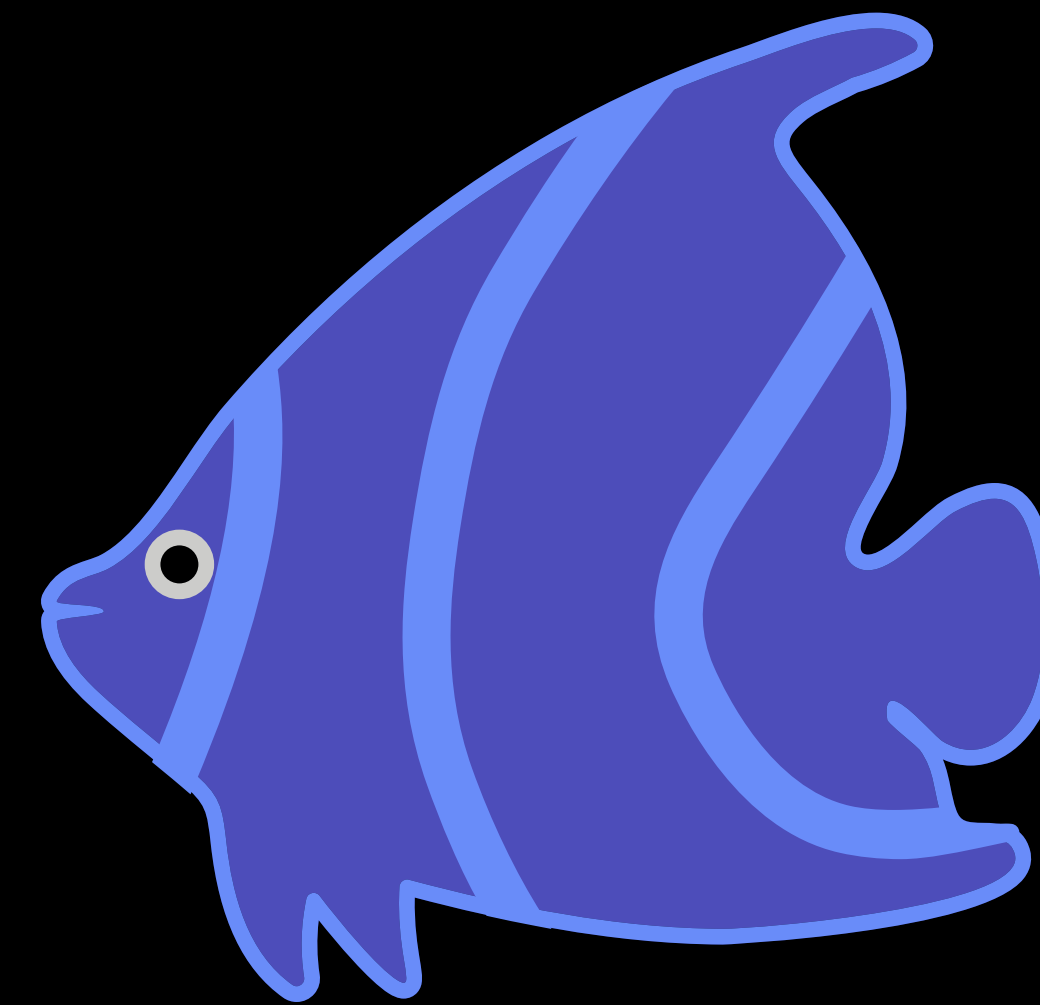
```
class Fish {  
    func swim() {  
        print("I'm swimming.")  
    }  
}  
  
class ComplainingFish: Fish {  
    func swim() {  
        print("Grumble grumble grumble...")  
        super.swim()  
    }  
}
```

! Error




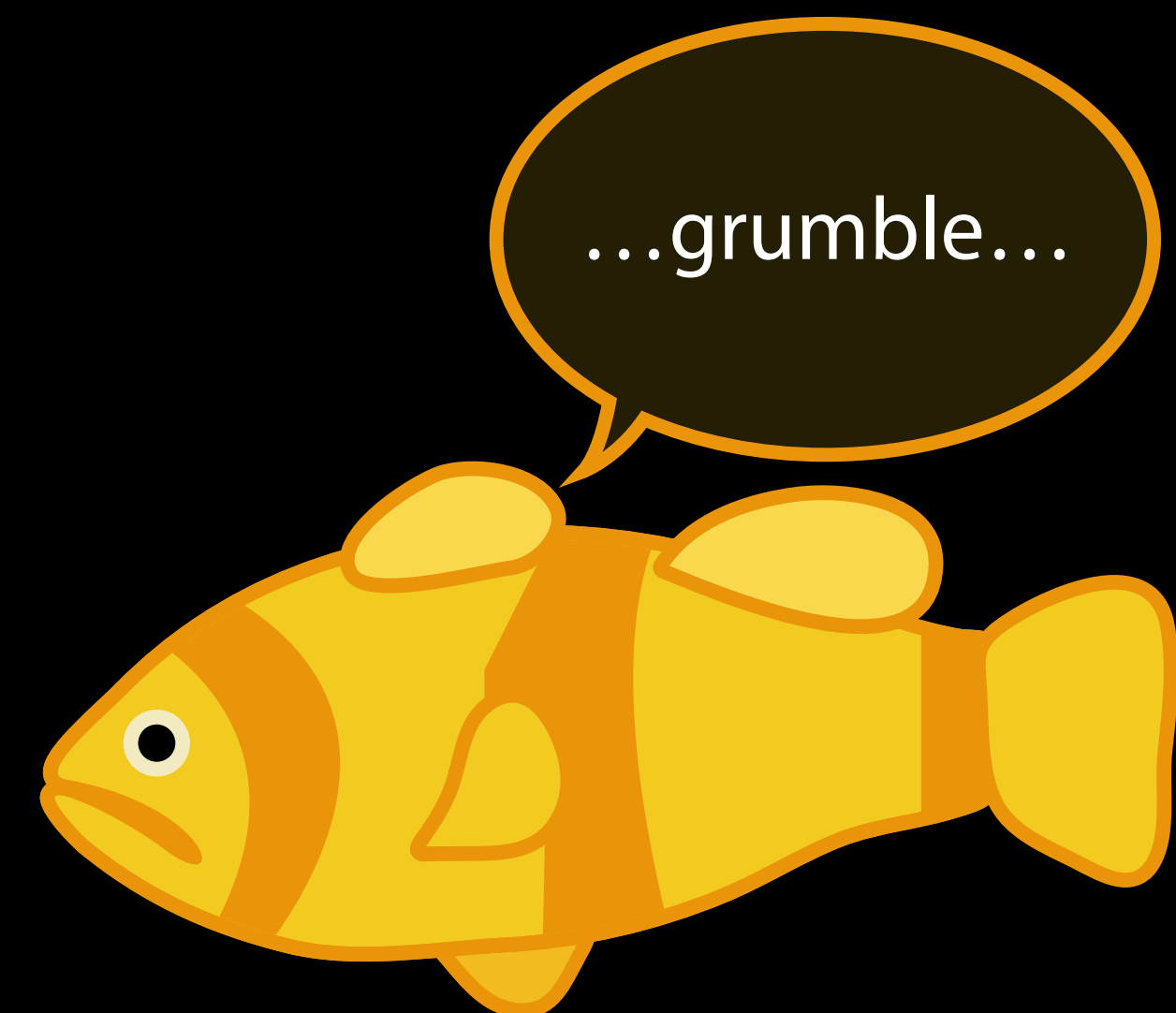
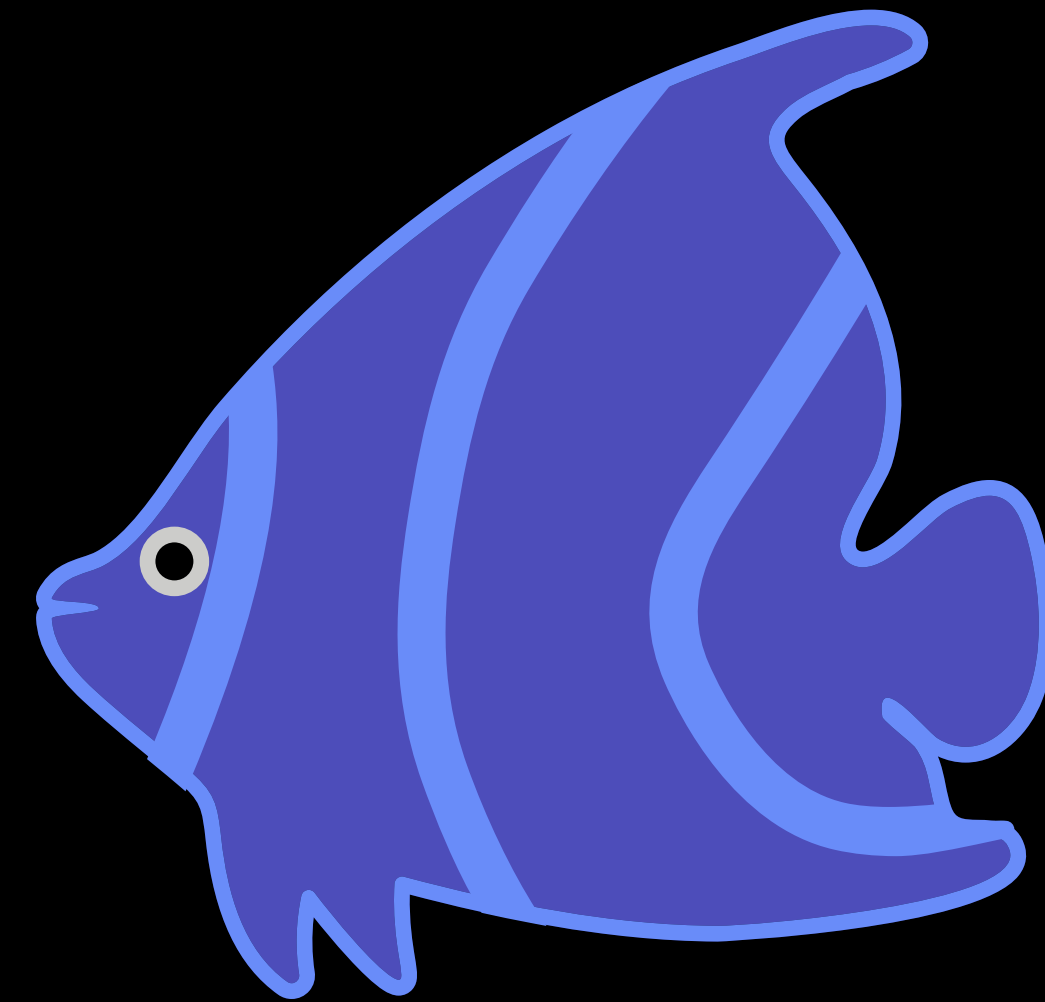
Subclasses Can Override Functionality

```
class Fish {  
    func swim() {  
        print("I'm swimming.")  
    }  
}  
  
class ComplainingFish: Fish {  
    override func swim() {  
        print("Grumble grumble grumble...")  
        super.swim()  
    }  
}
```



Subclasses Can Override Functionality

```
class Fish {  
    func swim() {  
        print("I'm swimming.")  
    }  
}  
  
class ComplainingFish: Fish {  
    override func swimmm() {  Error  
        print("Grumble grumble grumble...")  
        super.swim()  
    }  
}
```



Subclass Initializers

```
class Fish {  
  var name: String  
  init(name: String) {  
    self.name = name  
  }  
}
```

```
let fish = Fish(name: "Herring")
```

Subclass Initializers

```
class Fish {  
  var name: String  
  init(name: String) {  
    self.name = name  
  }  
}
```

```
class ComplainingFish: Fish {  
  var complaint: String  
  init(name: String, complaint: String) {  
  }  
}
```

```
let fish = ComplainingFish(name: "Salmon", complaint: "Grumble grumble grumble...")
```

Subclass Initializers

```
class Fish {  
    var name: String  
    init(name: String) {  
        self.name = name  
    }  
}
```

```
class ComplainingFish: Fish {  
    var complaint: String  
    init(name: String, complaint: String) {  
        self.complaint = complaint  
    }  
}
```

```
let fish = ComplainingFish(name: "Salmon", complaint: "Grumble grumble grumble...")
```


Subclass Initializers


```
class Fish {  
  var name: String  
  init(name: String) {  
    self.name = name  
  }  
}
```

```
class ComplainingFish: Fish {  
  var complaint: String  
  init(name: String, complaint: String) {  
    self.complaint = complaint  
    super.init(name: name)  
  }  
}
```

```
let fish = ComplainingFish(name: "Salmon", complaint: "Grumble grumble grumble...")
```

Subclass Initializers


```
class Fish {  
  var name: String  
  init(name: String) {  
    self.name = name  
  }  
}  
  
class ComplainingFish: Fish {  
  var complaint: String  
  init(name: String, complaint: String) {  
    self.complaint = complaint  
    super.init(name: name)  
  }  
}
```

A white line with an arrowhead points from the `super.init(name: name)` line in the `ComplainingFish` class to the `init(name: String) {` line in the `Fish` class, indicating that the superclass's `init` method is being called.

```
let fish = ComplainingFish(name: "Salmon", complaint: "Grumble grumble grumble...")
```

Subclass Initializers

```
class Fish {  
    var name: String  
    init(name: String) {  
        self.name = name  
    }  
}  
  
class ComplainingFish: Fish {  
    var complaint: String  
    init(name: String, complaint: String) {  
        self.complaint = complaint  
        super.init(name: name)  
    }  
}
```

A white line with an arrowhead points from the `init` method of the `ComplainingFish` class to the `init` method of the `Fish` class, indicating that `ComplainingFish` inherits the `init` method from `Fish`.

```
let fish = ComplainingFish(name: "Salmon", complaint: "Grumble grumble grumble...")
```

Subclass Problem

```
class Player {  
    func takeTurn(on board: Board) {...}  
}
```

Subclass Problem

```
class Player {  
    func takeTurn(on board: Board) {...}  
}  
  
class HumanPlayer: Player {  
    override func takeTurn(on board: Board) { /* Show move UI and wait */ }  
}
```

Subclass Problem

```
class Player {  
    func takeTurn(on board: Board) {...}  
}
```

```
class HumanPlayer: Player {  
    override func takeTurn(on board: Board) { /* Show move UI and wait */ }  
}
```

```
class ComputerPlayer: Player {  
    override func takeTurn(on board: Board) { /* Pick the best legal move using AI */ }  
}
```

Subclass Problem

```
class Player {  
    func takeTurn(on board: Board) {...}  
}  
  
class HumanPlayer: Player {  
    override func takeTurn(on board: Board) { /* Show move UI and wait */ }  
}  
  
class ComputerPlayer: Player {  
    override func takeTurn(on board: Board) { /* Pick the best legal move using AI */ }  
}
```

Subclass Problem

```
class Player {  
    func takeTurn(on board: Board) { /* fatal error */ }  
}  
  
class HumanPlayer: Player {  
    override func takeTurn(on board: Board) { /* Show move UI and wait */ }  
}  
  
class ComputerPlayer: Player {  
    override func takeTurn(on board: Board) { /* Pick the best legal move using AI */ }  
}
```


Protocols

```
protocol Player {  
    func takeTurn(on board: Board) { /* fatal error */ }  
}  
  
class HumanPlayer: Player {  
    override func takeTurn(on board: Board) { /* Show move UI and wait */ }  
}  
  
class ComputerPlayer: Player {  
    override func takeTurn(on board: Board) { /* Pick the best legal move using AI */ }  
}
```

Protocols

```
protocol Player {  
    func takeTurn(on board: Board)  
}  
  
class HumanPlayer: Player {  
    override func takeTurn(on board: Board) { /* Show move UI and wait */ }  
}  
  
class ComputerPlayer: Player {  
    override func takeTurn(on board: Board) { /* Pick the best legal move using AI */ }  
}
```

Protocols

```
protocol Player {  
    func takeTurn(on board: Board)  
}  
  
class HumanPlayer: Player {  
    override func takeTurn(on board: Board) { /* Show move UI and wait */ }  
}  
  
class ComputerPlayer: Player {  
    override func takeTurn(on board: Board) { /* Pick the best legal move using AI */ }  
}
```

Protocols

```
protocol Player {  
    func takeTurn(on board: Board)  
}  
  
class HumanPlayer: Player {  
    override func takeTurn(on board: Board) { /* Show move UI and wait */ }  
}  
  
class ComputerPlayer: Player {  
    override func takeTurn(on board: Board) { /* Pick the best legal move using AI */ }  
}
```

Protocols

```
protocol Player {  
    func takeTurn(on board: Board)  
}  
  
class HumanPlayer: Player {  
    func takeTurn(on board: Board) { /* Show move UI and wait */ }  
}  
  
class ComputerPlayer: Player {  
    func takeTurn(on board: Board) { /* Pick the best legal move using AI */ }  
}
```

Protocols

```
protocol Player {  
    func takeTurn(on board: Board)  
}  
  
struct HumanPlayer: Player {  
    func takeTurn(on board: Board) { /* Show move UI and wait */ }  
}  
  
struct ComputerPlayer: Player {  
    func takeTurn(on board: Board) { /* Pick the best legal move using AI */ }  
}
```

Protocols

```
struct HumanPlayer: Player {  
    var name: String  
    var score: Int  
    func takeTurn(on board: Board) {...}  
}  
  
let player = HumanPlayer(name: "Lynn", score: 0)
```

Protocols

```
struct HumanPlayer: Player {  
    var name: String  
    var score: Int  
    func takeTurn(on board: Board) {...}  
}  
  
let player = HumanPlayer(name: "Lynn", score: 0)  
print(player)
```


Protocols

```
struct HumanPlayer: Player {  
    var name: String  
    var score: Int  
    func takeTurn(on board: Board) {...}  
}  
  
let player = HumanPlayer(name: "Lynn", score: 0)  
print(player)
```

```
HumanPlayer(name: "Lynn", score: 0)
```

Protocols

```
protocol CustomStringConvertible {  
    var description: String { get }  
}
```

Protocols and Extensions

```
struct HumanPlayer: Player {...}
```

```
extension HumanPlayer: CustomStringConvertible {
```

```
}
```

Protocols and Extensions

```
struct HumanPlayer: Player {...}
```

```
extension HumanPlayer: CustomStringConvertible {
```

```
    var description: String {
```

```
        return "Human player \((name) has a score of \((score)"
```

```
    }
```

```
}
```

Protocols and Extensions

```
struct HumanPlayer: Player {...}

extension HumanPlayer: CustomStringConvertible {
    var description: String {
        return "Human player \((name) has a score of \((score)"
    }
}

let player = HumanPlayer(name: "Lynn", score: 0)
print(player)
```

Protocols and Extensions

```
struct HumanPlayer: Player {...}

extension HumanPlayer: CustomStringConvertible {
    var description: String {
        return "Human player \((name) has a score of \((score)"
    }
}

let player = HumanPlayer(name: "Lynn", score: 0)
print(player)
```

Human player Lynn has a score of 0

Protocols and Extensions

```
struct HumanPlayer: Player {...}

extension HumanPlayer: CustomStringConvertible {
    var description: String {
        return "Human player \(name) has a score of \(score)"
    }
}

let player = HumanPlayer(name: "Lynn", score: 0)
print(player)
```

Enumerations

```
enum Alignment {  
    case left  
    case right  
}
```


Enumerations

```
enum Alignment {  
    case left  
    case right  
}  
let textAlignment = Alignment.left
```

Enumerations

```
enum Alignment {  
    case left, right  
}  
let textAlignment = Alignment.left
```

Enumerations

```
enum Alignment {  
    case left, right  
}  
  
let textAlignment = Alignment.left  
  
switch textAlignment {  
case Alignment.left:  
    print("Lean to the left")  
case Alignment.right:  
    print("Lean to the right")  
}
```

Enumerations

```
enum Alignment {  
    case left, right  
}  
  
let textAlignment = Alignment.left  
  
switch textAlignment {  
case Alignment.left:  
    print("Lean to the left")  
case Alignment.right:  
    print("Lean to the right")  
}
```

Enumerations

```
enum Alignment {  
    case left, right  
}  
  
let textAlignment = Alignment.left  
  
switch textAlignment {  
case .left:  
    print("Lean to the left")  
case .right:  
    print("Lean to the right")  
}
```

Enumerations

```
enum Alignment {  
    case left, right  
}  
  
let textAlignment = Alignment.left  
  
switch textAlignment {  
case .left:  
    print("Lean to the left")  
case .right:  
    print("Lean to the right")  
}
```

Enumerations

```
enum Alignment {  
    case left, right, center  
}  
  
let textAlignment = Alignment.left
```

```
switch textAlignment {  
case .left:  
    print("Lean to the left")  
case .right:  
    print("Lean to the right")  
}
```

! switch must be exhaustive

Enumerations

```
enum Alignment {  
    case left, right, center  
}  
  
let textAlignment = Alignment.left  
switch textAlignment {  
case .left:  
    print("Lean to the left")  
case .right:  
    print("Lean to the right")  
case .center:  
    print("Stand up straight")  
}
```


Enumerations with Associated Values

```
enum Alignment {  
    case left(padding: Double), right(padding: Double), center  
}  
  
let textAlignment = Alignment.left(padding: 42.7)
```

Enumerations with Associated Values

```
enum Alignment {  
    case left(padding: Double), right(padding: Double), center  
}  
  
let textAlignment = Alignment.left(padding: 42.7)
```

Enumerations with Associated Values

```
enum Alignment {  
    case left(padding: Double), right(padding: Double), center  
}  
  
let textAlignment = Alignment.left(padding: 42.7)  
switch textAlignment {  
case .left(let padding):  
    print("Left with \(padding) pixels of padding")  
...  
}
```

Enumerations with Associated Values

```
enum Alignment {  
    case left(padding: Double), right(padding: Double), center  
}  
  
let textAlignment = Alignment.left(padding: 42.7)  
switch textAlignment {  
case .left(let padding):  
    print("Left with \(padding) pixels of padding")  
...  
}
```

Left with 42.7 pixels of padding

Enumerations with Raw Values

```
enum ServerAddress: String {  
    case staging = "https://staging.example.com"  
    case production = "https://example.com"  
}
```

Enumerations with Raw Values

```
enum ServerAddress: String {  
    case staging = "https://staging.example.com"  
    case production = "https://example.com"  
}  
  
func findPhotos(matchingQuery query: String, from server: ServerAddress) {  
    let serverAddress = server.rawValue  
    ...  
}  
  
findPhotos(matchingQuery: "strawberry", from: .staging)
```

```
// Error Handling
```

```
enum SomeError: ErrorProtocol {  
    case somethingWentWrong, somethingFailed  
}
```

```
func doSomething() throws -> Data {  
    progressBar.visible = true  
    defer { progressBar.visible = false }  
  
    let data: Data?  
    do {  
        data = try somethingThatMightFail()  
    } catch SomeError.somethingWentWrong {  
        data = nil  
    }  
  
    guard let result = summarize(data) else { throw SomeError.somethingFailed }  
    return result  
}
```

```
// Error Handling
```

```
enum SomeError: ErrorProtocol {  
    case somethingWentWrong, somethingFailed  
}
```

```
func doSomething() throws -> Data {  
    progressBar.visible = true  
    defer { progressBar.visible = false }  
  
    let data: Data?  
    do {  
        data = try somethingThatMightFail()  
    } catch SomeError.somethingWentWrong {  
        data = nil  
    }  
  
    guard let result = summarize(data) else { throw SomeError.somethingFailed }  
    return result  
}
```



```
// Error Handling
```

```
enum SomeError: ErrorProtocol {  
    case somethingWentWrong, somethingFailed  
}
```

```
func doSomething() throws -> Data {  
    progressBar.visible = true  
    defer { progressBar.visible = false }  
  
    let data: Data?  
    do {  
        data = try somethingThatMightFail()  
    } catch SomeError.somethingWentWrong {  
        data = nil  
    }  
  
    guard let result = summarize(data) else { throw SomeError.somethingFailed }  
    return result  
}
```

```
// Error Handling
```

```
enum SomeError: ErrorProtocol {  
    case somethingWentWrong, somethingFailed  
}
```

```
func doSomething() throws -> Data {  
    progressBar.visible = true  
    defer { progressBar.visible = false }  
}
```

```
let data: Data?  
do {  
    data = try somethingThatMightFail()  
} catch SomeError.somethingWentWrong {  
    data = nil  
}
```

```
guard let result = summarize(data) else { throw SomeError.somethingFailed }  
return result  
}
```

```
// Error Handling
```

```
enum SomeError: ErrorProtocol {  
    case somethingWentWrong, somethingFailed  
}
```

```
func doSomething() throws -> Data {  
    progressBar.visible = true  
    defer { progressBar.visible = false }  
}
```

```
let data: Data?  
do {  
    data = try somethingThatMightFail()  
} catch SomeError.somethingWentWrong {  
    data = nil  
}
```

```
guard let result = summarize(data) else { throw SomeError.somethingFailed }  
return result  
}
```

```
// Error Handling
```

```
enum SomeError: ErrorProtocol {  
    case somethingWentWrong, somethingFailed  
}
```

```
func doSomething() throws -> Data {  
    progressBar.visible = true  
    defer { progressBar.visible = false }  
}
```

```
let data: Data?  
do {  
    data = try somethingThatMightFail()  
} catch SomeError.somethingWentWrong {  
    data = nil  
}
```

```
guard let result = summarize(data) else { throw SomeError.somethingFailed }  
return result  
}
```

```
// Error Handling
```

```
enum SomeError: ErrorProtocol {  
    case somethingWentWrong, somethingFailed  
}
```

```
func doSomething() throws -> Data {  
    progressBar.visible = true  
    defer { progressBar.visible = false }  
}
```

```
let data: Data?  
do {  
    data = try somethingThatMightFail()  
} catch SomeError.somethingWentWrong {  
    data = nil  
}
```

```
guard let result = summarize(data) else { throw SomeError.somethingFailed }  
return result  
}
```

More Information

<https://developer.apple.com/wwdc16/404>

Related Sessions

What's New in Swift	Presidio	Tuesday 9:00AM
Swift API Design Guidelines	Presidio	Tuesday 10:00AM
What's New in Foundation for Swift	Mission	Tuesday 4:00PM
Introducing Swift Playgrounds	Mission	Wednesday 11:00AM
Going Server-Side with Swift Open Source	Mission	Friday 9:00AM
Protocol and Value Oriented Programming in UIKit Apps	Nob Hill	Friday 4:00PM

Labs

Swift Get-Together	Graphics, Games, and Media Lab A	Wednesday 6:15 PM
Swift Open Hours	Developer Tools Lab A	Tuesday 12:00PM
Swift Open Hours	Developer Tools Lab A	Wed–Fri 9:00AM

