SWIFT

INTRODUÇÃO A LINGUAGEM DA MAÇÃ

Hello!

I Am Vinicius

I am here because I love to technologies for iOS.

You can contact me at

@ViniciusDeep

Swift

iOS +

“

“Swift is like C”
“Swift is CLIKE”

THE BASICS

- ✓ Constants and Variables
- ✓ Types
- ✓ Casting

THE BASICS

✓ Constants and Variables

```
let maximumNumberOfLoginAttempts = 10
var currentLoginAttempt = 0
```

THE BASICS

- ✓ Constants and Variables
- ✓ Types

```
• let maximumNumberOfLoginAttempts : Int
• var currentLoginAttempt : Int
• let π  : Double = 3.1415
• let nameOfUser  : String = “Michael Douglas”
• let haveCount = true
```

THE BASICS

- ✓ Constants and Variables
- ✓ Types
- ✓ Casting

```
• let three :Int = 3
• let pointOneFourOneFiveNine = 0.14159
• let pi = Double(three) + pointOneFourOneFiveNine
```

THE BASICS

✓ Tuples

```
let http404Error = (404, "Not Found")

let http200Status = (statusCode: 200, description: "OK")
```

THE BASICS

✓ Strings are Value Types

```
let string1 = "hello"
let string2 = " there"
var welcome = string1 + string2
```

```
let multiplier = 3
let message = "\(multiplier) times 2.5 is \(Double(multiplier) * 2.5)"
```

OPTIONALS?

- let possibleNumber = "123"
- let convertedNumber = Int(possibleNumber)

nil?

- var serverResponseCode: Int? = 404
- // serverResponseCode contains an actual Int value of 404
- serverResponseCode = nil
- // serverResponseCode now contains no value

Hands On!

IF STATEMENTS

```swift
let name = "world"
if name == "world" {
    print("hello, world")
} else {
    print("I'm sorry \(name), but I don't recognize you")
}
// Prints "hello, world", because name is indeed equal to "world".
```

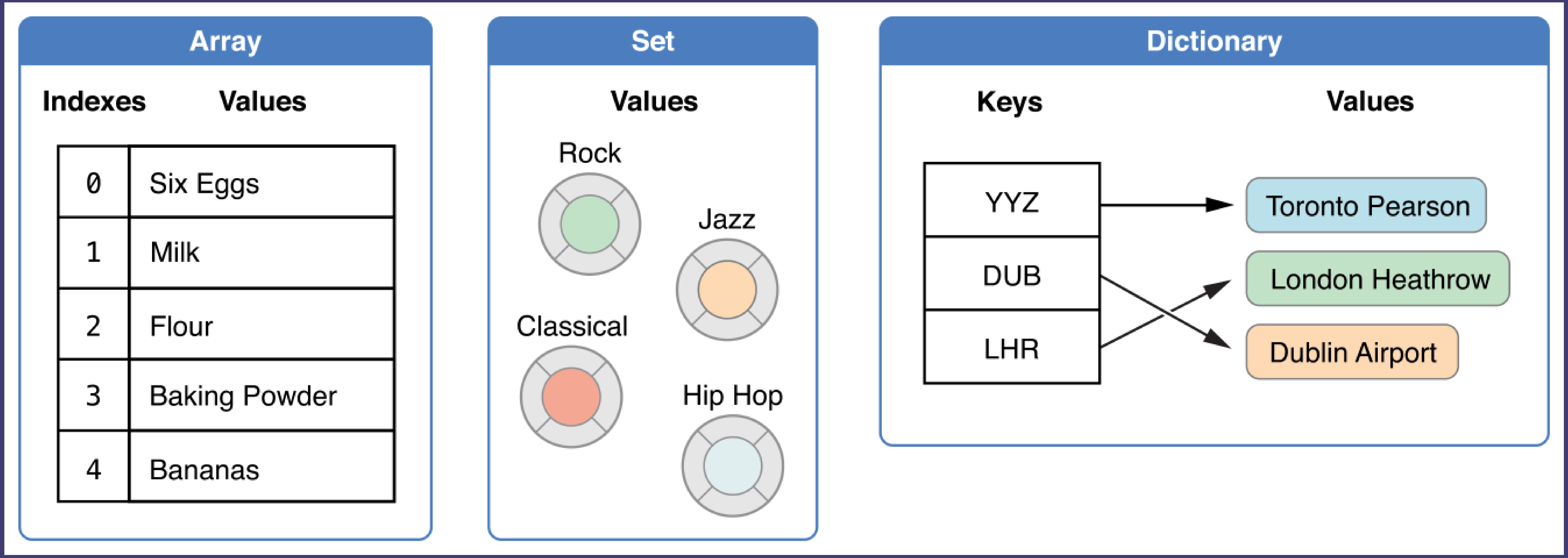
FOR RANGE

```swift
let names = ["Anna", "Alex", "Brian", "Jack"]
let count = names.count
for i in 0..<count {
    print("Person \(i + 1) is called \(names[i])")
}
// Person 1 is called Anna
// Person 2 is called Alex
// Person 3 is called Brian
// Person 4 is called Jack
```

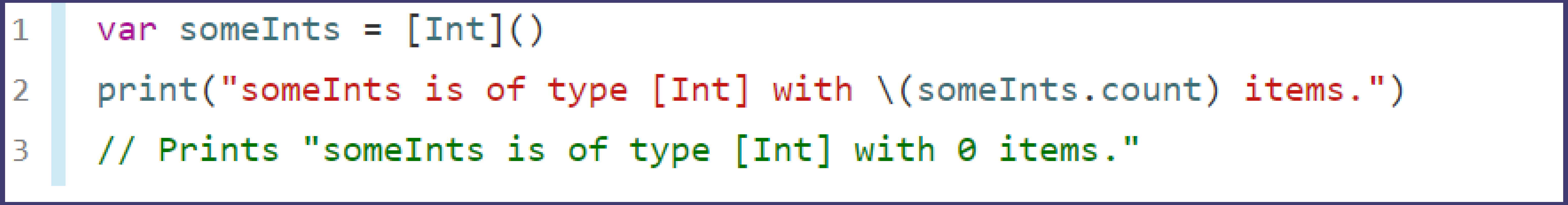
OS X Mavericks
GAMES
Final Cut Pro X
I migliori nuovi giochi
Mac App Store?

Hands On!

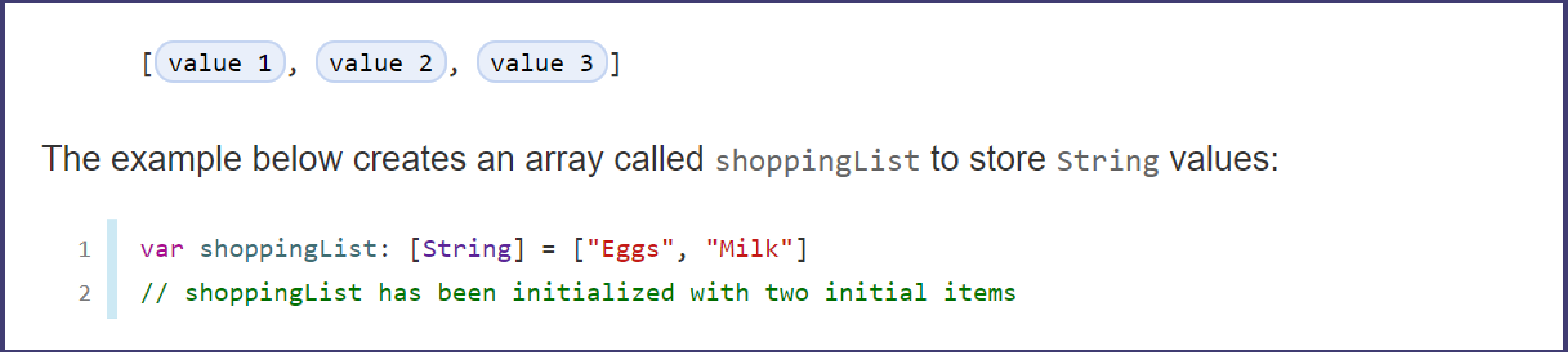
COLLECTION TYPES



ARRAYS



```swift
var someInts = [Int]()
print("someInts is of type [Int] with \(someInts.count) items.")
// Prints "someInts is of type [Int] with 0 items."
```

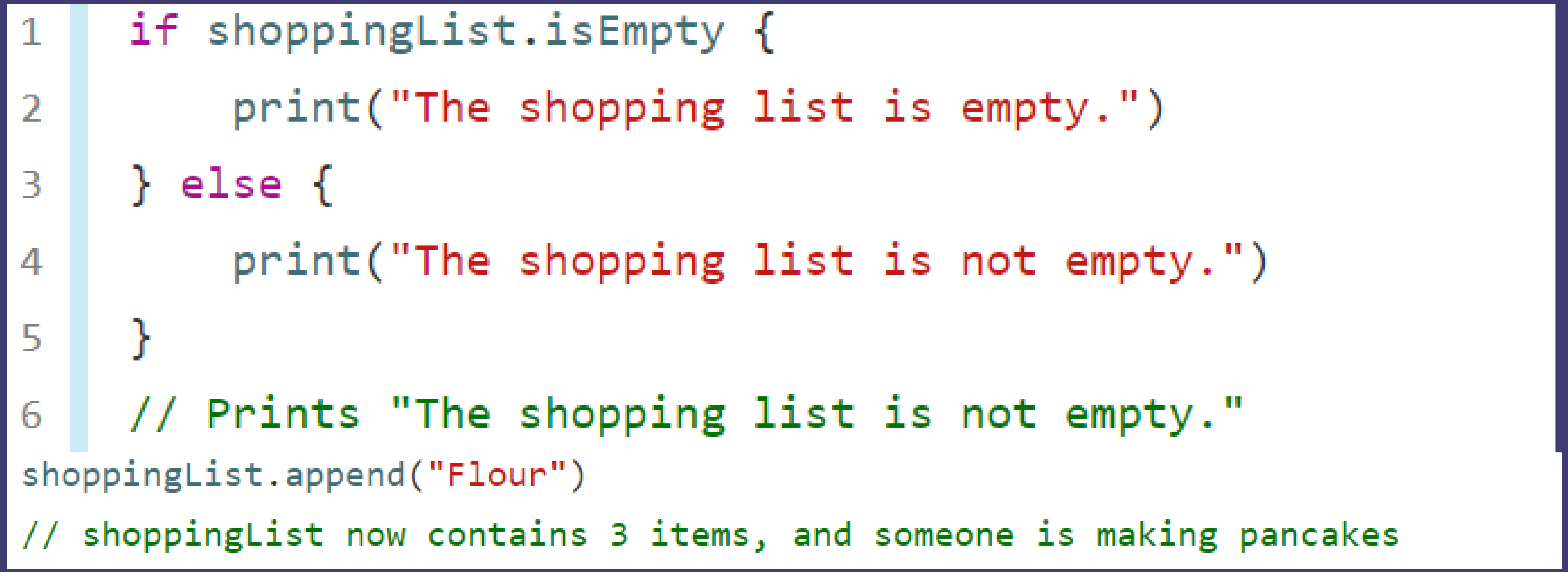


```
[value 1, value 2, value 3]
```

The example below creates an array called `shoppingList` to store `String` values:

```swift
var shoppingList: [String] = ["Eggs", "Milk"]
// shoppingList has been initialized with two initial items
```

ARRAYS



```swift
if shoppingList.isEmpty {
    print("The shopping list is empty.")
} else {
    print("The shopping list is not empty.")
}
// Prints "The shopping list is not empty."
shoppingList.append("Flour")
// shoppingList now contains 3 items, and someone is making pancakes
```

```swift
for item in shoppingList {
    print(item)
}
```

```swift
let mapleSyrup = shoppingList.remove(at: 0)
```

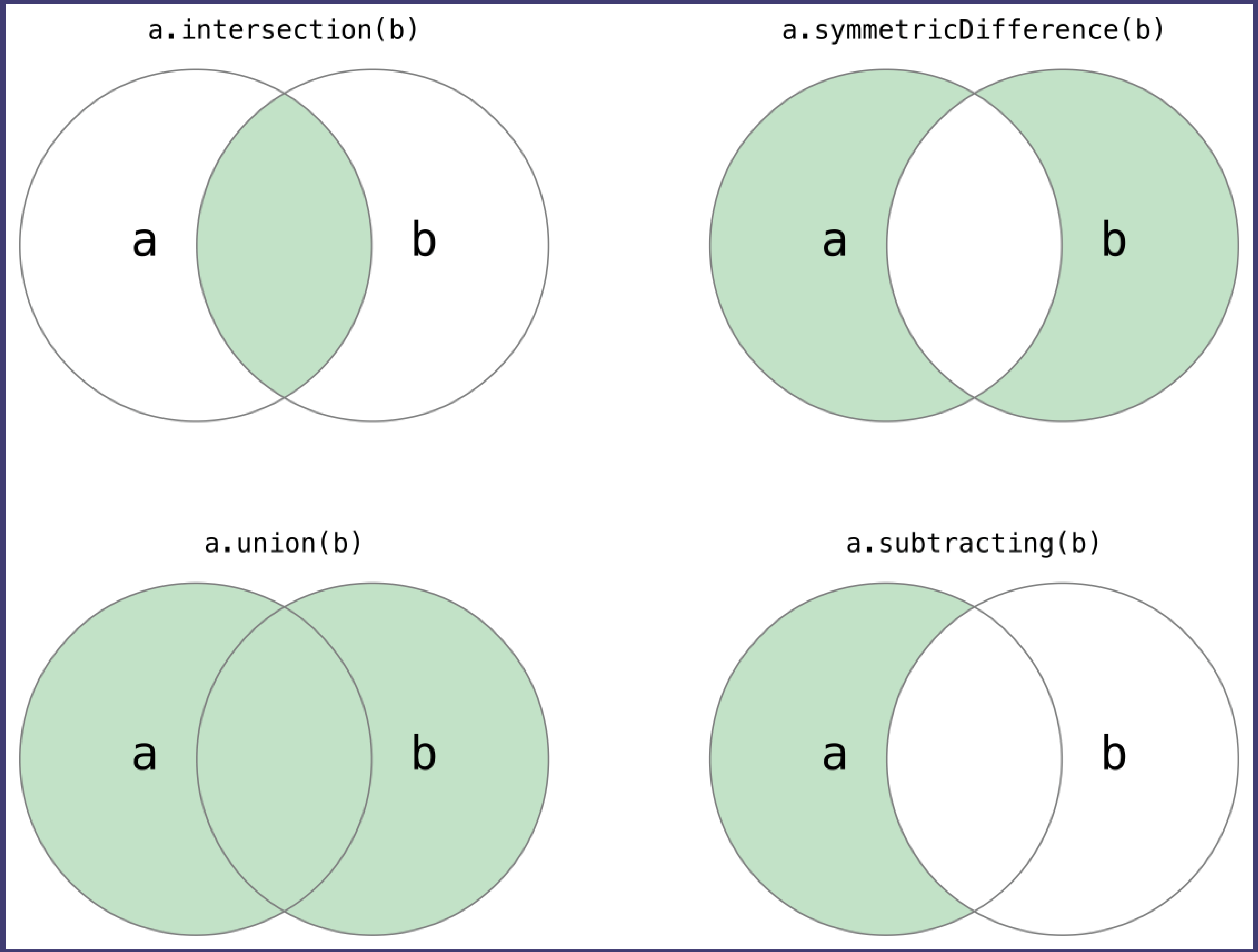
SET

- var letters = Set<Character>()
- print("letters is of type Set<Character> with \(letters.count) items.")
- // Prints "letters is of type Set<Character> with 0 items."

- var favoriteGenres: Set<String> = ["Rock", "Classical", "Hip hop"]
- // favoriteGenres has been initialized with three initial items

favoriteGenres.insert("Jazz")

SET


a.intersection(b)
a
b
a.symmetricDifference(b)
a
b
a.union(b)
a
b
a.subtracting(b)
a
b

DICITIONARY

- var namesOfIntegers = [Int: String]()
- // namesOfIntegers is an empty [Int: String] dictionary
  - namesOfIntegers[16] = "sixteen"
  - // namesOfIntegers now contains 1 key-value pair
  - namesOfIntegers = [:]
  - // namesOfIntegers is once again an empty dictionary of type [Int: String]

DICITIONARY

```
var airports = ["YYZ": "Toronto Pearson", "DUB": "Dublin"]
```

```
var airports: [String: String] = ["YYZ": "Toronto Pearson", "DUB": "Dublin"]
```

```
if airports.isEmpty {
    print("The airports dictionary is empty.")
} else {
    print("The airports dictionary is not empty.")
}
// Prints "The airports dictionary is not empty."
```

```
if let oldValue = airports.updateValue("Dublin Airport", forKey: "DUB") {
    print("The old value for DUB was \(oldValue).")
}
```

For in Loops

```
let names = ["Anna", "Alex", "Brian", "Jack"]
for name in names {
    print("Hello, \(name)!")
}
// Hello, Anna!
// Hello, Alex!
// Hello, Brian!
// Hello, Jack!
```

While

```
while condition {
    statements
}
```

While

```
var square = 0
var diceRoll = 0
while square < finalSquare {
    // roll the dice
    diceRoll += 1
    if diceRoll == 7 { diceRoll = 1 }
    // move by the rolled amount
    square += diceRoll
    if square < board.count {
        // if we're still on the board, move up or down for a snake or a ladder
        square += board[square]
    }
}
print("Game over!")
```

Switch

```
let someCharacter: Character = "z"
switch someCharacter {
case "a":
    print("The first letter of the alphabet")
case "z":
    print("The last letter of the alphabet")
default:
    print("Some other character")
}
// Prints "The last letter of the alphabet"
```

Hands On!

Functions

```
func greet(person: String) -> String {
    let greeting = "Hello, " + person + "!"
    return greeting
}

 print(greet(person: "Anna"))
 // Prints "Hello, Anna!"
 print(greet(person: "Brian"))
 // Prints "Hello, Brian!"
```

```
func greet(person: String) {
    print("Hello, \(person)!")
}
greet(person: "Dave")
// Prints "Hello, Dave!
```

Hands On!

Enumerations Syntax

```
enum CompassPoint {
    case north
    case south
    case east
    case west
}
```

Enumerations

```
directionToHead = .south
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")
}
```

Enumerations

8 85909 51226 3

Enumerations

```
enum Barcode {
    case upc(Int, Int, Int, Int)
    case qrCode(String)
}


var productBarcode = Barcode.upc(8, 85909, 51226, 3)

productBarcode = .qrCode("ABCDEFGHIJKLMNOP")



switch productBarcode {
case .upc(let numberSystem, let manufacturer, let product, let check):
    print("UPC: \(numberSystem), \(manufacturer), \(product), \(check).")
case .qrCode(let productCode):
    print("QR code: \(productCode).")
}
```

STRUCTS vs. CLASSES

Structs

```
struct SomeStructure {
    // structure definition goes here
}
```

Define properties to store values
Define methods to provide functionality
Define subscripts to provide access to their values using subscript syntax
Define initializers to set up their initial state
Be extended to expand their functionality beyond a default implementation
Conform to protocols to provide standard functionality of a certain kind
Value Type

Classes

```
class SomeStructure {
    // structure definition goes here
}
```

Define properties to store values
Define methods to provide functionality
Define subscripts to provide access to their values using subscript syntax
Define initializers to set up their initial state
Be extended to expand their functionality beyond a default implementation
Conform to protocols to provide standard functionality of a certain kind
Reference Type

Classes

Classes have additional capabilities that structures don’t have:

Inheritance enables one class to inherit the characteristics of another.
Type casting enables you to check and interpret the type of a class instance at runtime.
Deinitializers enable an instance of a class to free up any resources it has assigned.
Reference counting allows more than one reference to a class instance.

STRUCTS vs. CLASSES

Structs vs Classes

```
struct Resolution {
    var width = 0
    var height = 0
}
class VideoMode {
    var resolution = Resolution()
    var interlaced = false
    var frameRate = 0.0
    var name: String?
}
```

```
let someResolution = Resolution()
let someVideoMode = VideoMode()
```

Structs vs Classes

```
let someResolution = Resolution()
let someVideoMode = VideoMode()

someVideoMode.resolution.width = 1280
print("The width of someVideoMode is now \(someVideoMode.resolution.width)")

let vga = Resolution(width: 640, height: 480)
```

Structs vs Classes

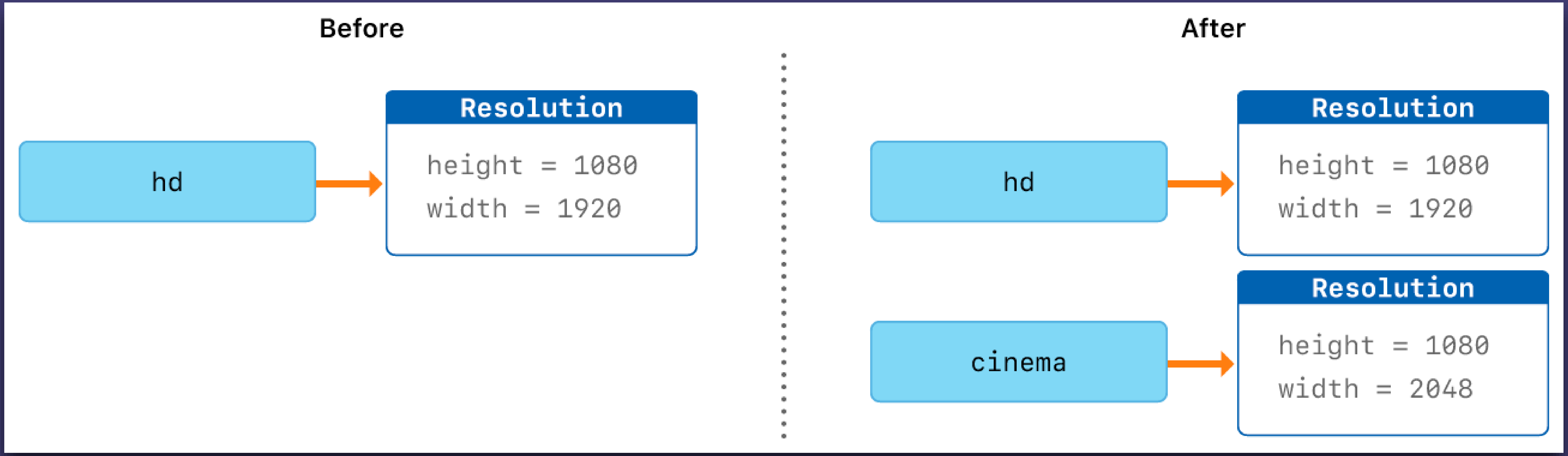
```
let hd = Resolution(width: 1920, height: 1080)
var cinema = hd

cinema.width = 2048


print("cinema is now \(cinema.width) pixels wide")
// Prints "cinema is now 2048 pixels wide"


print("hd is still \(hd.width) pixels wide")
// Prints "hd is still 1920 pixels wide"
```

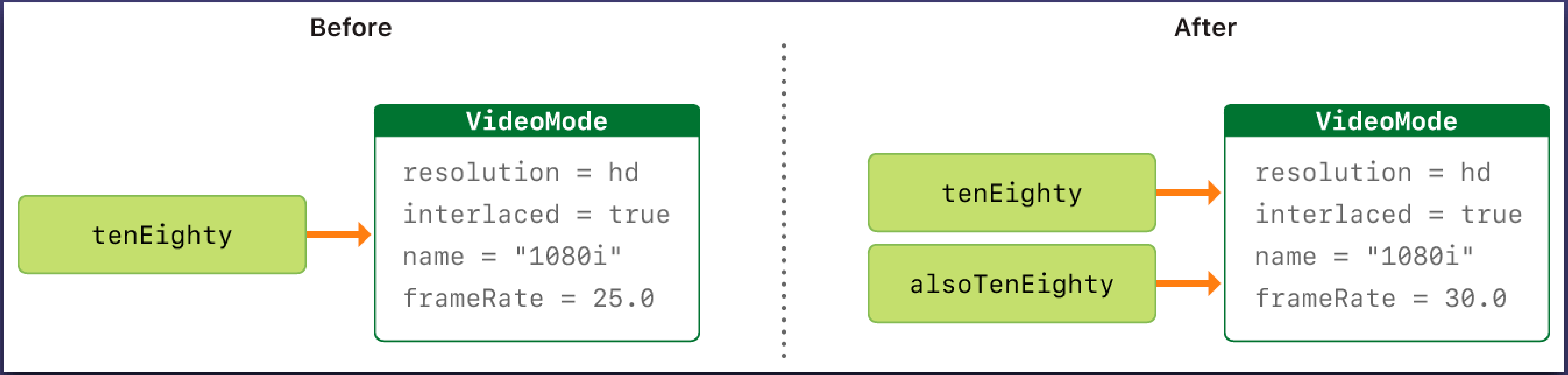
Structs vs Classes



Structs vs Classes

```
let tenEighty = VideoMode()
tenEighty.resolution = hd
tenEighty.interlaced = true
tenEighty.name = "1080i"
tenEighty.frameRate = 25.0

let alsoTenEighty = tenEighty
alsoTenEighty.frameRate = 30.0
```



09 - Em Swift, structs são Value Types (passadas por valor). Leia o código abaixo e assinale a resposta correta.

```
struct Resolution {
      var width: Float -
      var height: Float -
}

var hd = Resolution(width: 1920, height: 1080)
var cinema = hd
cinema.width = 2048
print("\(hd.width)")
```

CÓPIA DA REFERÊNCIA
HD != CINEMA

a) A instância referenciada pela variável hd receberá o valor 2048 na propriedade width.
b) As variáveis cinema e hd irão referenciar a mesma instância e essa instância receberá o valor 2048 na propriedade width.
c) A variável cinema irá referenciar uma instância diferente da referenciada pela variável hd. A instância referenciada por cinema receberá o valor 2048 na propriedade width. ✓
d) O valor 2048 será exibido no terminal. ×
e) O código não irá compilar.

1 2 A = 3 A = 1 A

Thanks!

Any questions?

You can find me at: @ViniciusDeep in gitHub