Swift workshop

Introduction to Swift

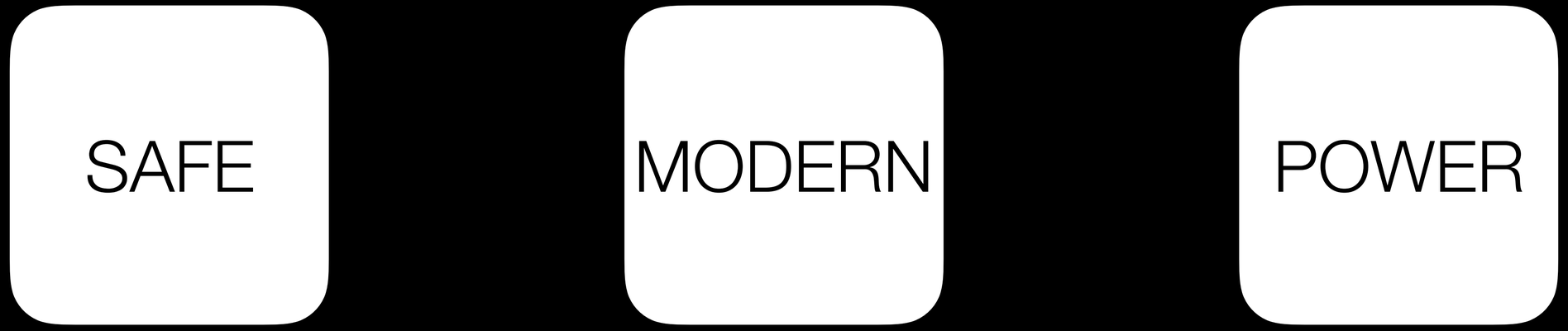
Andrej Šarić
Josip Ćavar


Swift

```c
#include <stdio.h>

int main() {
    printf("hello, world\n");
    return 0;
}
```

```
print("Hello, Futura")
```

What You Will Learn



Variables

```
var
```

Variables

```
var languageName
```

Variables

```
var languageName:
```

Variables

```
var languageName: String
```

Constants and variables

```
let languageName: String = “Swift"
```

Constants and variables

```
let languageName: String = “Swift”
var version: Double = 1.0
```

Constants and variables

```
let languageName: String = “Swift”
var version: Double = 1.0
var introduced: Int = 2014
```

Constants and variables

```
let languageName: String = “Swift”
var version: Double = 1.0
var introduced: Int = 2014
var isAwesome: Bool = true
```

Constants and variables

```
let languageName: String = “Swift”
var version: Double = 1.0
let introduced: Int = 2014
let isAwesome: Bool = true
```

Constants and variables

```
let languageName: String = “Swift”
var version: Double = 1.0
let introduced: Int = 2014
let isAwesome: Bool = true
```

Constants and variables

```
let languageName: String = “Swift”
var version: Double = 1.0
let introduced: Int = 2014
let isAwesome: Bool = true
```

Type inference

SAFE

```
let languageName = “Swift” // inferred as String
var version = 1.0          // inferred as Double
let introduced = 2014      // inferred as Int
let isAwesome = true       // inferred as Bool
```

Unicode names

```
let languageName = “Swift”
var version = 1.0
let introduced = 2014
let isAwesome = true
let 🐶🐱 = “dogcat”
```

String

String

```
let someString = “I appear to be a string”
```

String

```
let someString = “I appear to be a string”
// inferred to be of type String = "Swift"
```

String

```
let someString = “I appear to be a string”
// inferred to be of type String = “Swift

urlRequest.HTTPMethod = “POST”
```

String

```
let someString = “I appear to be a string”
// inferred to be of type String = “Swift

urlRequest.HTTPMethod = “POST”

let components = “~/Documents/Swift”
```

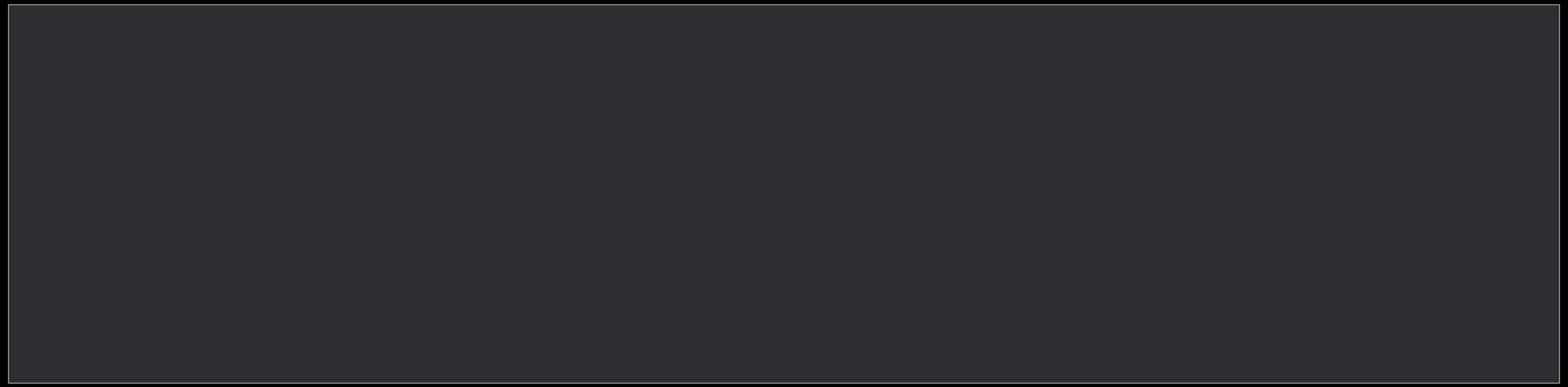
String

```
let someString = “I appear to be a string”
// inferred to be of type String = “Swift

urlRequest.HTTPMethod = “POST”

let components = “~/Documents/Swift”
// [“~”,”Documents”,”Swift”]
```

Character

Character

```
for character in “mouse”.characters {
    print(character)
}
```



Character

```
for character in “mouse”.characters {
    print(character)
}
```

```
m
o
u
s
e
```

Character

```
for character in “mús”.characters {
    print(character)
}
```

```
m
ú
s
```

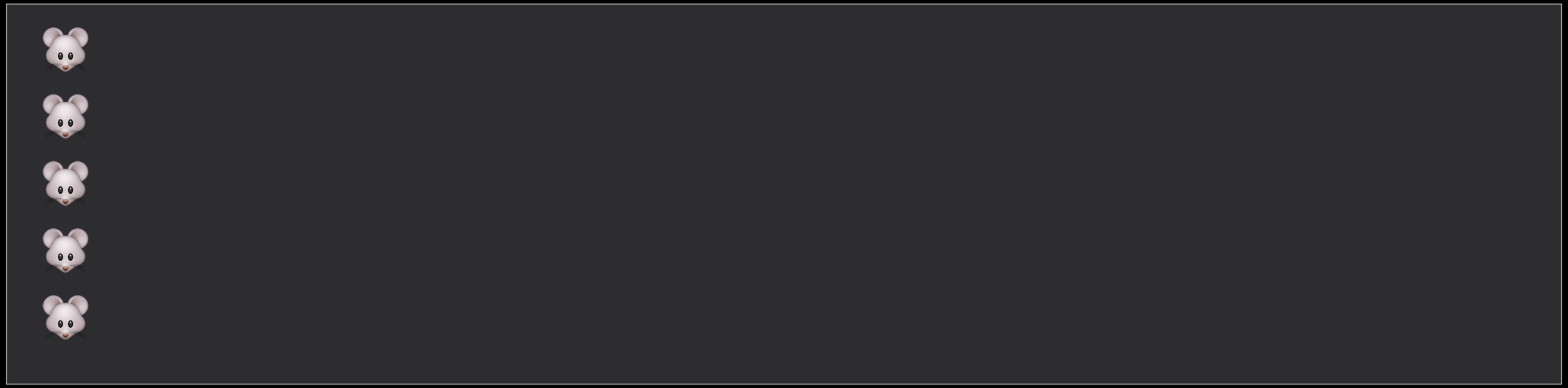
Character

```
for character in “мышь”.characters {
    print(character)
}
```

```
м
ы
ш
ь
```

Character

```
for character in “🐭🐭🐭🐭🐭”.characters {
    print(character)
}
```



Combining Strings and Characters

```
let dog: Character = “🐶”
```

Combining Strings and Characters

```
let cow: Character = “🐮”
```

Combining Strings and Characters

```
let dog: Character = “🐶”
let cow: Character = “🐮”
```

Combining Strings and Characters

```
let dog: Character = “🐶”
let cow: Character = “🐮”
let dogCow = “\(dog)\(cow)”
// dog cow is “🐶🐮”
```

Combining Strings and Characters

```
let dog: Character = “🐶”
let cow: Character = “🐮”
let dogCow = “\(dog)\(cow)”
// dog cow is “🐶🐮”

let instructions = “Beware of the ”
let fullInstructions = instructions.append(dog)
// instructions is “Beware of the 🐶”
```

Building Complex Strings

Building Complex Strings

```
let a = 3, b = 5
```

Building Complex Strings

```
let a = 3, b = 5

// “3 times 5  is 15”
```

String interpolation


```
let a = 3, b = 5

// “3 times 5  is 15”

let mathResult = “\(a) times \(b) is \(a * b)”
```

String interpolation


```
let a = 3, b = 5

// “3 times 5  is 15”

let mathResult = “\(a) times \(b) is \(a * b)”

// “3 times 5  is 15”
```

String interpolation


```
let a = 7, b = 4

// “7 times 4  is 28”

let mathResult = “\(a) times \(b) is \(a * b)”

// “7 times 4  is 28”
```

String mutability

String mutability

```
var variableString = “Horse”
```

String mutability

```
var variableString = “Horse”
variableString += “ and carriage”
```

String mutability

```
var variableString = “Horse”
variableString += “ and carriage”
// variableString is now “Horse and carriage”
```

String mutability

```
var variableString = “Horse”
variableString += “ and carriage”
// variableString is now “Horse and carriage”

let constantString = “Highlander”
```

String mutability

```
var variableString = “Horse”
variableString += “ and carriage”
// variableString is now “Horse and carriage”

let constantString = “Highlander”
constantString += “add another Highlander”
```

String mutability

```
var variableString = “Horse”
variableString += “ and carriage”
// variableString is now “Horse and carriage”

let constantString = “Highlander”
constantString += “add another Highlander”
// error – constantString cannot be changed
```

Array and Dictionary Literals

Array and Dictionary Literals

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”]
```

Array and Dictionary Literals

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”]
```

Array and Dictionary Literals

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”]

var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
```

Array and Dictionary Literals

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”]

var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
```

Array and Dictionary Literals

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”]

var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
```

Array and Dictionary Literals

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”]

var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
```

Typed Collections

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”]
```

Typed Collections

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”, 42]
```

Typed Collections

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”, true]
```

Typed Collections

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”, Bicycle()]
```

Typed Collections

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”]
```

Typed Collections

```
var names: [String] = [“Anna”,”Alex”,”Brian”,”Jack”]
```

Typed Collections

```
var names: [String] = [“Anna”,”Alex”,”Brian”,”Jack”]
```

Typed Collections

```
var names: [String] = [“Anna”,”Alex”,”Brian”,”Jack”]
```

Typed Collections

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”]
// an array of String values
```

Typed Collections

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”]
// an array of String values

var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
```

Typed Collections

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”]
// an array of String values

var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
```

Typed Collections

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”]
// an array of String values

var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
// a Dictionary with String keys and Int values
```

Typed Collections

SAFE

```
var names = [“Anna”,”Alex”,”Brian”,”Jack”]
// an array of String values

var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
// a Dictionary with String keys and Int values
```

Loops

Loops

```
while !sated {
    eatCake()
}

for doctor in 1...13 {
  exterminate(doctor)
}
```

For-In: Strings and Characters

POWER

```
for character in “🐭🐭🐭🐭🐭”.characters {
    print(character)
}
```

```
🐭
🐭
🐭
🐭
🐭
```

For-In: Ranges

POWER

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

```
1 times 5 is 5
2 times 5 is 10
3 times 5 is 15
4 times 5 is 20
5 times 5 is 25
```

For-In: Ranges

POWER

```
for index in 0..<5 {
     print("\(index) times 5 is \(index * 5)")
 }
```

```
0 times 5 is 0
1 times 5 is 5
2 times 5 is 10
3 times 5 is 15
4 times 5 is 20
```

For-In: Arrays

POWER

```
for name in [“Anna”,”Alex”,”Brian”,”Jack”] {
    print(“Hello, \(name)!”)
}
```

```
Hello, Anna!
Hello, Alex!
Hello, Brian!
Hello, Jack!
```

For-In: Dictionaries

POWER

```
var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]

for (animalName, legCount) in numberOfLegs {
    print(“\(animalName)s have \(legCount) legs”)
}
```

```
ants have 6 legs
snakes have 0 legs
cheetahs have 4 legs
```

For-In: Dictionaries

POWER

```
var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]

for (animalName, legCount) in numberOfLegs {
    print(“\(animalName)s have \(legCount) legs”)
}
```

```
ants have 6 legs
snakes have 0 legs
cheetahs have 4 legs
```

Modifying an Array

Modifying an Array

```
var shoppingList = [“Eggs”, ”Milk”]
```

Modifying an Array

```
var shoppingList = [“Eggs”, ”Milk”]
```

```
[“Eggs”, “Milk”]
```

Modifying an Array

```
var shoppingList = [“Eggs”, ”Milk”]
print(shoppingList[0])
```

Modifying an Array

```
var shoppingList = [“Eggs”, ”Milk”]
print(shoppingList[0])
```

```
“Eggs”
```

Modifying an Array

```
var shoppingList = [“Eggs”, ”Milk”]
print(shoppingList[0])
shoppingList += “Flour”
```

Modifying an Array

```
var shoppingList = [“Eggs”, ”Milk”]
print(shoppingList[0])
shoppingList += “Flour”
```

```
[“Eggs”, “Milk”, “Flour”]
```

Modifying an Array

```
var shoppingList = [“Eggs”, ”Milk”]
print(shoppingList[0])
shoppingList += “Flour”
shoppingList += [“Cheese”,”Butter”, “Chocolate Spread”]
```

Modifying an Array

```
var shoppingList = [“Eggs”, ”Milk”]
print(shoppingList[0])
shoppingList += “Flour”
shoppingList += [“Cheese”,”Butter”, “Chocolate Spread”]
```

```
[“Eggs”,“Milk”,“Flour”,”Cheese”,”Butter”,
”Chocolate Spread”]
```

Modifying an Array

```
var shoppingList = [“Eggs”, ”Milk”]
print(shoppingList[0])
shoppingList += “Flour”
shoppingList += [“Cheese”,”Butter”, “Chocolate Spread”]
shoppingList[0] = “Sugar”
```

```
[“Eggs”,“Milk”,“Flour”,”Cheese”,”Butter”,
”Chocolate Spread”]
```

Modifying an Array

```
var shoppingList = [“Eggs”, ”Milk”]
print(shoppingList[0])
shoppingList += “Flour”
shoppingList += [“Cheese”,”Butter”, “Chocolate Spread”]
shoppingList[0] = “Sugar”
```

```
[“Sugar”,“Milk”,“Flour”,”Cheese”,”Butter”,
”Chocolate Spread”]
```

Modifying an Array

```
var shoppingList = [“Eggs”, ”Milk”]
print(shoppingList[0])
shoppingList += “Flour”
shoppingList += [“Cheese”,”Butter”, “Chocolate Spread”]
shoppingList[0] = “Sugar”
shoppingList[3…5] = [“Bananas”, “Apples”]
```

```
[“Sugar”,“Milk”,“Flour”,”Cheese”,”Butter”,
”Chocolate Spread”]
```

Modifying an Array

```
var shoppingList = [“Eggs”, ”Milk”]
print(shoppingList[0])
shoppingList += “Flour”
shoppingList += [“Cheese”,”Butter”, “Chocolate Spread”]
shoppingList[0] = “Sugar”
shoppingList[3…5] = [“Bananas”, “Apples”]
```

```
[“Sugar”,“Milk”,“Flour”,”Bananas”,”Apples”]
```

Modifying a Dictionary

Modifying a Dictionary

```
var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
```

Modifying a Dictionary

```
var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
```

```
["ant": 6, "snake": 0, "cheetah":4]
```

Modifying a Dictionary

```
var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
numberOfLegs["spider"] = 273
```

Modifying a Dictionary

```
var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
numberOfLegs["spider"] = 273
```

```
["ant": 6, "snake": 0, "cheetah":4, "spider": 273]
```

Modifying a Dictionary

```
var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
numberOfLegs["spider"] = 273
numberOfLegs["spider"] = 8
```

```
["ant": 6, "snake": 0, "cheetah":4, "spider": 273]
```

Modifying a Dictionary

```
var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
numberOfLegs["spider"] = 273
numberOfLegs["spider"] = 8
```

```
["ant": 6, "snake": 0, "cheetah":4, "spider": 8]
```

Retrieving a Value from a Dictionary

```
var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
```

Retrieving a Value from a Dictionary

```
var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
// aardvark?
```

Retrieving a Value from a Dictionary

```
var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
// aardvark?
// dugong?
```

Retrieving a Value from a Dictionary

```
var numberOfLegs = [“ant”: 6,”snake”: 0,”cheetah”: 4]
// aardvark?
// dugong?
// venezuelan poodle moth?
```

```
???????
```

Beyond the Basics

Retrieving a Value from a Dictionary

```
let numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]

let possibleLegCount = numberOfLegs["aardvark"]
```

Optionals

```
let numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]

let possibleLegCount: Int? = numberOfLegs["aardvark"]
```

Querying an Optional

```
let numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]

let possibleLegCount: Int? = numberOfLegs["aardvark"]

if possibleLegCount == nil {
    print("Aardvark wasn't found")
}
```

Querying an Optional

```
let numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]

let possibleLegCount: Int? = numberOfLegs["aardvark"]

if possibleLegCount == nil {
     print("Aardvark wasn't found")
} else {
     let legCount = possibleLegCount!
     print("An aardvark has \(legCount) legs")
}
```

Querying an Optional

```
let numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]

let possibleLegCount: Int? = numberOfLegs["aardvark"]

if possibleLegCount == nil {
    print("Aardvark wasn't found")
} else {
    let legCount = possibleLegCount!
    print("An aardvark has \(legCount) legs")
}
```

Querying an Optional

```
let numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]

let possibleLegCount: Int? = numberOfLegs["aardvark"]

if possibleLegCount == nil {
    print("Aardvark wasn't found")
} else {
    let legCount: Int = possibleLegCount!
    print("An aardvark has \(legCount) legs")
}
```

Querying an Optional

```
if possibleLegCount {
    let legCount = possibleLegCount!

    print("An aardvark has \(legCount) legs")

}

//ERROR
```

Unwrapping an Optional

```
if let legCount = possibleLegCount  {
    print("An aardvark has \(legCount) legs")
}
```

Unwrapping an Optional

SAFE

```
if let legCount = possibleLegCount  {
    print("An aardvark has \(legCount) legs")
}
```

If Statements

```
if legCount == 0 {
    print("It slithers and slides around")
} else {
    print("It walks")
}
```

If Statements

```
if legCount == 0 {
   print("It slithers and slides around")
} else {
   print("It walks")
}
```

If Statements

```
if legCount == 0 {
   print("It slithers and slides around")
} else {
   print("It walks")
}
```

More Complex If Statements

```
if legCount == 0 {
    print("It slithers and slides around")
} else if legCount == 1 {
    print("It hops")
} else {
    print("It walks")
}
```

Switch

```
switch legCount {
 case 0:
  print("It slithers and slides around")

 case 1:
  print("It hops")

 default:
  print("It walks")
}
```

Switch

```
switch sender {
 case executeButton:
   print("You tapped the Execute button")

 case firstNameTextField:
   print("You tapped the First Name text field")

 default:
   print("You tapped some other object")

}
```

Switch

```
switch legCount {
 case 0:
   print("It slithers and slides around")

 case 1, 3, 5, 7, 9, 11, 13:
   print("It limps")

 case 2, 4, 6, 8, 10, 12, 14:
   print("It walks")
}
```

Switch

```
switch legCount {
 case 0:
  print("It slithers and slides around")

 case 1, 3, 5, 7, 9, 11, 13:
  print("It limps")

 case 2, 4, 6, 8, 10, 12, 14:
  print("It walks")
}
// error: switch must be exhaustive
```

Switch

```
switch legCount {
 case 0:
   print("It slithers and slides around")

 case 1, 3, 5, 7, 9, 11, 13:
   print("It limps")

 default:
   print("It walks")
}
```

Switch

SAFE

```
switch legCount {
 case 0:
   print("It slithers and slides around")

 case 1, 3, 5, 7, 9, 11, 13:
   print("It limps")

 default:
   print("It walks")
}
```

Matching Value Ranges

```
switch legCount {
 case 0:
   print("It has no legs")

 case 1...8:
   print("It has a few legs")

 default:
   print("It has lots of legs")
}
```

Matching Value Ranges

SAFE

```
switch legCount {
 case 0:
   print("It has no legs")

 case 1...8:
   print("It has a few legs")

 default:
   print("It has lots of legs")
}
```

Matching Value Ranges

```
switch legCount {
 case 0:
   print("It has no legs")

 case 1...8:
   print("It has a few legs")

 default:
   print("It has lots of legs")
}
```

Functions

```
func sayHello() {
 print("Hello!")
}

sayHello()
```

Functions

```
func sayHello() {
 print("Hello!")
}

sayHello()
```

```
Hello
```

Functions with parameters

```
func sayHello(name:String) {
    print("Hello\(name)!")
}
```

Functions with parameters

```
func sayHello(name:String) {
    print("Hello\(name)!")
}

sayHello("Futura")
```

Functions with parameters

```
func sayHello(name:String) {
     print("Hello\(name)!")
}

sayHello("Futura")
```

```
Hello Futura!
```

Default Parameter Values

```
func sayHello(name: String = "World") {
    print("Hello\(name)!")
}

sayHello()
```

Default Parameter Values

```
func sayHello(name: String = "World") {
    print("Hello\(name)!")
}

sayHello()
```

```
Hello World!
```

Default Parameter Values

```
func sayHello(name: String = "World") {
    print("Hello\(name)!")
}

sayHello()
sayHello(name: "Futura")
```

```
Hello World!
```

Default Parameter Values

```
func sayHello(name: String = "World") {
    print("Hello\(name)!")
}

sayHello()
sayHello(name: "Futura")
```

```
Hello World!
Hello Futura!
```

Returning Values

```
func buildGreeting(name: String = "World") -> String {
 return "Hello " + name
}

let greeting = buildGreeting()
```

Returning Values

```
func buildGreeting(name: String = "World") -> String {
 return "Hello " + name
}

let greeting: String = buildGreeting()
```

Returning Values

```
func buildGreeting(name: String = "World") -> String {
 return "Hello " + name
}

let greeting = buildGreeting()

print(greeting)
```

Returning Values

```
func buildGreeting(name: String = "World") -> String {
 return "Hello " + name
}

let greeting = buildGreeting()

print(greeting)
```

```
Hello World!
```

Returning Multiple Values

```
func refreshWebPage() -> (Int, String) {
      // ...try to refresh...
      return (200, "Success")
}
```

Returning Multiple Values

MODERN

```
func refreshWebPage() -> (Int, String) {
      // ...try to refresh...
      return (200, "Success")
}
```

Tuples

MODERN

```
(3.79, 3.99, 4.19)

(404, "Not found")

(2, "banana", 0.72)
```

Tuples

```
(3.79, 3.99, 4.19)    // (Double, Double, Double)

(404, "Not found")    // (Int, String)

(2, "banana", 0.72)   // (Int, String, Double)
```

Returning Multiple Values

```
func refreshWebPage() -> (Int, String) {
    // ... try to refresh...
    return (200, "Success")
}
```

Decomposing a Tuple

```
func refreshWebPage() -> (Int, String) {
    // ... try to refresh...
    return (200, "Success")
}

let (statusCode, message) = refreshWebPage()
```

Decomposing a Tuple

```
func refreshWebPage() -> (Int, String) {
    // ... try to refresh...
    return (200, "Success")
}

let (statusCode, message) = refreshWebPage()

print("Received \(statusCode): \(message)")
```

Decomposing a Tuple

```
func refreshWebPage() -> (Int, String) {
    // ... try to refresh...
    return (200, "Success")
}

let (statusCode, message) = refreshWebPage()

print("Received \(statusCode): \(message)")
```

```
Received 200: Success
```

Tuple Decomposition for Enumeration

```
let numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]

for (animalName, legCount) in numberOfLegs {
    print("\(animalName)s have \(legCount) legs")
}
```

```
ants have 6 legs
snakes have 0 legs
cheetahs have 4 legs
```

Named Values in a Tuple

```
func refreshWebPage() -> (Int, String) {
    // ... try to refresh...
    return (200, "Success")
}
```

Named Values in a Tuple

```
func refreshWebPage() -> (code: Int, message: String) {
    // ... try to refresh...
    return (200, "Success")
}
```

Named Values in a Tuple

```
func refreshWebPage() -> (code: Int, message: String) {
    // ... try to refresh...
    return (200, "Success")
}

let status = refreshWebPage()

print("Received \(status.code): \(status.message)")
```

Named Values in a Tuple

```
func refreshWebPage() -> (code: Int, message: String) {
    // ... try to refresh...
    return (200, "Success")
}

let status = refreshWebPage()

print("Received \(status.code): \(status.message)")
```

```
Received 200: Success
```

Closures

```
let greetingPrinter = {
     print("Hello World!")
}
```

Closures

```
let greetingPrinter: () -> () = {
     print("Hello World!")
}
```

Closures

```
let greetingPrinter: () -> () = {
     print("Hello World!")
}

func greetingPrinter() -> () {
     print("Hello World!")
}
```

Closures

```
let greetingPrinter: () -> () = {
     print("Hello World!")
}

func greetingPrinter() -> () {
      print("Hello World!")
}
```

Closures

```
let greetingPrinter: () -> () = {
     print("Hello World!")
}

greetingPrinter()
```

Closures

```
let greetingPrinter: () -> () = {
     print("Hello World!")
}

greetingPrinter()
```

```
Hello World!
```

Closures as Parameters

```
func repeat(count: Int, task:() -> ()) {
    for i in 0..count {
        task()
    }
}
```

Closures as Parameters

```
func repeat(count: Int, task:() -> ()) {
    for i in 0..count {
        task()
    }
}
repeat(2, {
    print("Hello!")
})
```

Closures as Parameters

```
func repeat(count: Int, task:() -> ()) {
     for i in 0..count {
          task()
     }
}
repeat(2, {
    print("Hello!")
})
```

```
Hello!
Hello!
```

Trailing Closures

```
func repeat(count: Int, task:() -> ()) {
     for i in 0..count {
          task()
     }
}
repeat(2) {
    print("Hello!")
}
```

```
Hello!
Hello!
```

Trailing Closures

MODERN

```
func repeat(count: Int, task:() -> ()) {
     for i in 0..count {
          task()
     }
}
repeat(2) {
    print("Hello!")
}
```

```
Hello!
Hello!
```

Classes

Classes

```
class Vehicle {



}
```

Classes

```
class Vehicle {



}
```

Classes

```
class Vehicle {



}
```

Classes

```
class Vehicle {
    // properties


}
```

Classes

```
class Vehicle {
    // properties
    // methods


}
```

Classes

```
class Vehicle {
    // properties
    // methods
    // initializers

}
```

Classes

```
import "Vehicle.h"

class Vehicle {

}
```

Classes

```
import "Vehicle.h"

class Vehicle {

}
```

Classes

```
class Vehicle {

}
```

Classes

```
class Vehicle: ??????? {

}
```

Classes

```
class Vehicle: ???????? {

}
```

Classes

```
class Vehicle: NSObject {

}
```

Classes

```
class Vehicle {

}
```

Class Inheritance

```
class Vehicle {

}

class Bicycle: Vehicle {

}
```

Class Inheritance

```
class Vehicle {

}

class Bicycle: Vehicle {

}
```

Class Inheritance

```
class Vehicle {

}

class Bicycle: Vehicle {

}
```

Class Inheritance

```
class Vehicle {

}
```

Properties

```
class Vehicle {
    var numberOfWheels = 0
}
```

Properties

```
class Vehicle {
    var numberOfWheels = 0
}
```

Properties

```
class Vehicle {
    let numberOfWheels = 0
}
```

Properties

```
class Vehicle {
    var numberOfWheels = 0
}
```

Properties

```
class Vehicle {
    var numberOfWheels = 0
}
```

Properties

```
class Vehicle {
    var numberOfWheels = 0
}
```

Stored Properties

```
class Vehicle {
    var numberOfWheels = 0
}
```

Computed Properties

Computed Properties

```
class Vehicle {
    var numberOfWheels = 0
    var description: String {
        get {
            return "\(numberOfWheels) wheels"
        }
    }
}
```

Computed Properties

```
class Vehicle {
    var numberOfWheels = 0
    var description: String {
        get {
            return "\(numberOfWheels) wheels"
        }
    }
}
```

Computed Properties

```
class Vehicle {
    var numberOfWheels = 0
    var description: String {
        get {
            return "\(numberOfWheels) wheels"
        }
    }
}
```

Computed Properties

```
class Vehicle {
    var numberOfWheels = 0
    var description: String {
        get {
            return "\(numberOfWheels) wheels"
        }
    }
}
```

Computed Properties

```
class Vehicle {
    var numberOfWheels = 0
    var description: String {
        get {
            return "\(numberOfWheels) wheels"
        }
        set {
        }
    }
}
```

Computed Properties

```
class Vehicle {
    var numberOfWheels = 0
    var description: String {
        get {
            return "\(numberOfWheels) wheels"
        }
    }
}
```

Computed Properties

```
class Vehicle {
    var numberOfWheels = 0
    var description: String {
        var description: String {
            return "\(numberOfWheels) wheels"
        }
    }
}
```

Computed Properties

```
class Vehicle {
    var numberOfWheels = 0
    var description: String {
        var description: String {
            return "\(numberOfWheels) wheels"
        }
    }
}
```

Initializer Syntax

```
class Vehicle {
    var numberOfWheels = 0
    var description: String {
        var description: String {
            return "\(numberOfWheels) wheels"
        }
    }
}

let someVehicle = Vehicle()
```

Initializer Syntax

```
class Vehicle {
    var numberOfWheels = 0
    var description: String {
        var description: String {
            return "\(numberOfWheels) wheels"
        }
    }
}

let someVehicle = Vehicle()
```

Automatic Memory Allocation

```
class Vehicle {
    var numberOfWheels = 0
    var description: String {
        var description: String {
            return "\(numberOfWheels) wheels"
        }
    }
}

let someVehicle = Vehicle()
```

Type Inference

```
class Vehicle {
    var numberOfWheels = 0
    var description: String {
        var description: String {
            return "\(numberOfWheels) wheels"
        }
    }
}

let someVehicle: Vehicle = Vehicle()
```

Default Values

```
class Vehicle {
    var numberOfWheels = 0
    var description: String {
        var description: String {
            return "\(numberOfWheels) wheels"
        }
    }
}

let someVehicle = Vehicle()
```

Dot Syntax

```
let someVehicle = Vehicle()
```

Dot Syntax

```
let someVehicle = Vehicle()

print(someVehicle.description)
```

Dot Syntax

```
let someVehicle = Vehicle()

print(someVehicle.description)
// 0 wheels
```

Dot Syntax

```
let someVehicle = Vehicle()

print(someVehicle.description)
// 0 wheels

someVehicle.numberOfWheels = 2
```

Dot Syntax

```
let someVehicle = Vehicle()

print(someVehicle.description)
// 0 wheels

someVehicle.numberOfWheels = 2

print(someVehicle.description)
```

Dot Syntax

```
let someVehicle = Vehicle()

print(someVehicle.description)
// 0 wheels

someVehicle.numberOfWheels = 2

print(someVehicle.description)
// 2 wheels
```

Class Initialization

```
class Bicycle: Vehicle {



}
```

Class Initialization

```
class Bicycle: Vehicle {
    init() {

    }
}
```

Class Initialization

```
class Bicycle: Vehicle {
    init() {

    }
}
```

Class Initialization

```
class Bicycle: Vehicle {
    init() {
        super.init()
    }
}
```

Class Initialization

```
class Bicycle: Vehicle {
    init() {
        super.init()
        numberOfWheels = 2
    }
}
```

Class Initialization

```
class Bicycle: Vehicle {
    init() {
        super.init()
        numberOfWheels = 2
    }
}
```

Class Initialization

```
class Bicycle: Vehicle {
    init() {
        super.init()
        numberOfWheels = 2
    }
}

let myBicycle = Bicycle()
```

Class Initialization

```
class Bicycle: Vehicle {
    init() {
        super.init()
        numberOfWheels = 2
    }
}

let myBicycle = Bicycle()
print(myBicycle.description)
// 2 wheels
```

Overriding a Property

Overriding a Property

```
class Car: Vehicle {



}
```

Overriding a Property

```
class Car: Vehicle {
    var speed = 0.0     // inferred as Double

}
```

Overriding a Property

```
class Car: Vehicle {
    var speed = 0.0
    init() {
        super.init()
        numberOfWheels = 4
    }
}
```

Overriding a Property

```
class Car: Vehicle {
	var speed = 0.0
	init() {
		super.init()
		numberOfWheels = 4
	}
	var description: String{

	}
}
```

Overriding a Property

```
class Car: Vehicle {
	var speed = 0.0
	init() {
		super.init()
		numberOfWheels = 4
	}
	override var description: String{

	}
}
```

Overriding a Property

SAFE

```
class Car: Vehicle {
      var speed = 0.0
      init() {
            super.init()
            numberOfWheels = 4
      }
      override var description: String{

      }
}
```

Overriding a Property

```
class Car: Vehicle {
      var speed = 0.0
      init() {
            super.init()
            numberOfWheels = 4
      }
      override var description: String{
          return super.description + ", \(speed) mph"
      }
}
```

Overriding a Property

```
class Car: Vehicle {
      var speed = 0.0
      init() {
            super.init()
            numberOfWheels = 4
      }
      override var description: String{
          return super.description + ", \(speed) mph"
      }
}
```

Overriding a Property

```
let myCar = Car()
```

Overriding a Property

```
let myCar = Car()

print(myCar.description)
// 4 wheels, 0.0 mph
```

Overriding a Property

```
let myCar = Car()

print(myCar.description)
// 4 wheels, 0.0 mph

myCar.speed = 35.0
```

Overriding a Property

```
let myCar = Car()

print(myCar.description)
// 4 wheels, 0.0 mph

myCar.speed = 35.0

print(myCar.description)
// 4 wheels, 35.0 mph
```

Property Observers

Property Observers

```
class ParentsCar: Car {




}
```

Property Observers

```
class ParentsCar: Car {
    override var speed: Double {




    }
}
```

Property Observers

```
class ParentsCar: Car {
    override var speed: Double {
        willSet {

        }
        didSet {

        }
    }
}
```

Property Observers

```
class ParentsCar: Car {
    override var speed: Double {
        willSet {
            // newValue is available here
        }
        didSet {

        }
    }
}
```

Property Observers

```
class ParentsCar: Car {
    override var speed: Double {
        willSet {

        }
        didSet {
            // oldValue is available here
        }
    }
}
```

Property Observers

```
class ParentsCar: Car {
    override var speed: Double {
        willSet {



        }
    }
}
```

Property Observers

```
class ParentsCar: Car {
    override var speed: Double {
        willSet {
            if newValue > 65.0 {



            }
        }
    }
}
```

Property Observers

```
class ParentsCar: Car {
    override var speed: Double {
        willSet {
            if newValue > 65.0 {
                print("Careful now.")
            }
        }
    }
}
```

Methods

Methods

```
class Counter {
    var count = 0



}
```

Methods

```
class Counter {
    var count = 0



}
```

Methods

```
class Counter {
    var count = 0
    func increment() {
        count += 1

    }
}
```

Methods

```
class Counter {
    var count = 0
    func incrementBy(amount: Int) {
        count += amount
    }
}
```

Methods

```
class Counter {
    var count = 0
    func incrementBy(amount: Int) {
        count += amount
    }
}
```

Methods

```
class Counter {
    var count = 0
    func incrementBy(amount: Int) {
        count += amount
    }
    func resetToCount(count: Int) {
        self.count = count
 }
}
```

Structures in Swift

Structures in Swift

```
struct Point {
    var x, y: Double
}

struct Size {
    var width, height: Double
}

struct Rect {
  var origin:Point
  var size: Size
}
```

Structures in Swift

```
var point = Point(x: 0.0, y: 0.0)

var size = Size(width: 640.0, height: 480.0)

var rect = Rect(origin: point, size: size)
```

Structures in Swift

```
struct Rect {
  var origin:Point
  var size: Size
}
```

Structures in Swift

```
struct Rect {
  var origin:Point
  var size: Size

  var area: Double {
      return size.width * size.height
  }
}
```

Structures in Swift

```
struct Rect {
  var origin:Point
  var size: Size

  var area: Double {
      return size.width * size.height
  }

  func isBiggerThanRect(other: Rect) -> Bool {
      return self.area > other.area
  }
}
```

Structures in Swift

POWER

```
struct Rect {
  var origin:Point
  var size: Size

  var area: Double {
      return size.width * size.height
  }

  func isBiggerThanRect(other: Rect) -> Bool {
      return self.area > other.area
  }
}
```

Structures in Swift

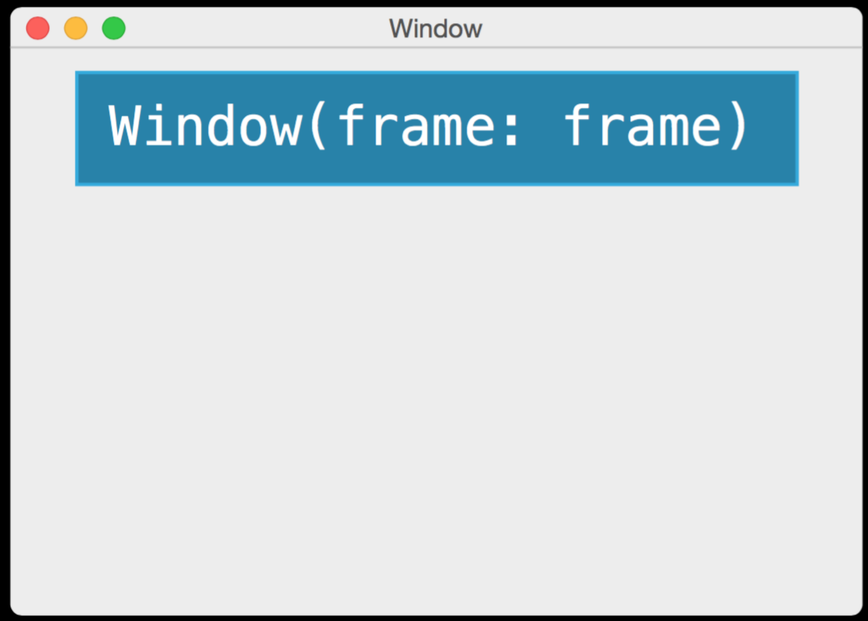
```
struct Rect {
  var origin:Point
  var size: Size

  var area: Double {
      return size.width * size.height
  }
}

class Window {
    var frame: Rect
    ...
}
```

Structure or Class?

```
var window = Window(frame: frame)
```

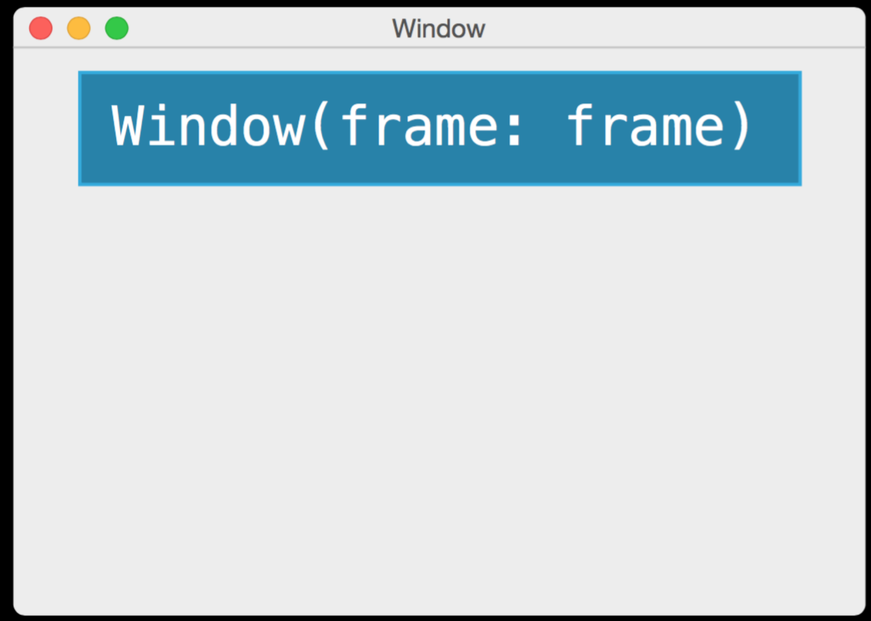
Structure or Class?

```
var window                  =
```



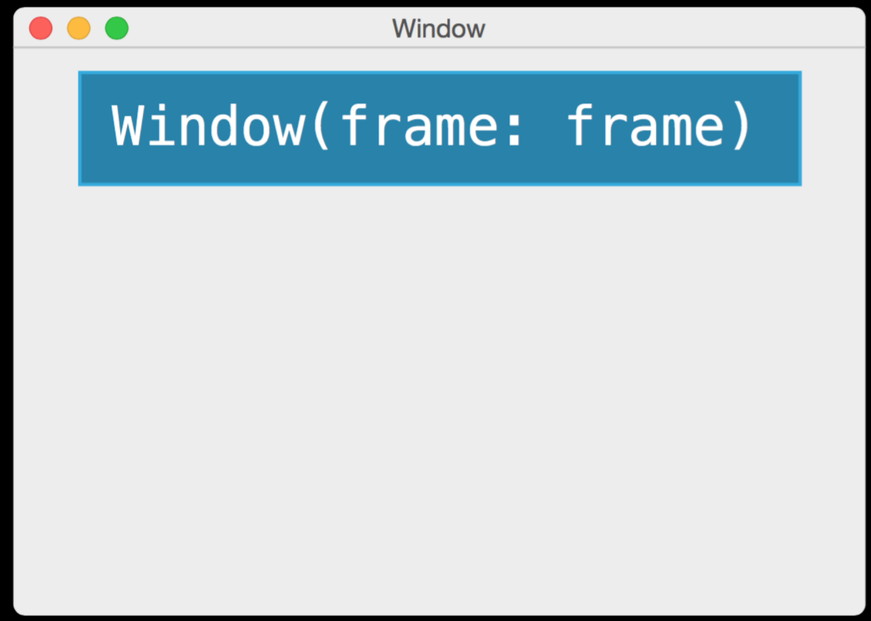
Structure or Class?

```
var window
```



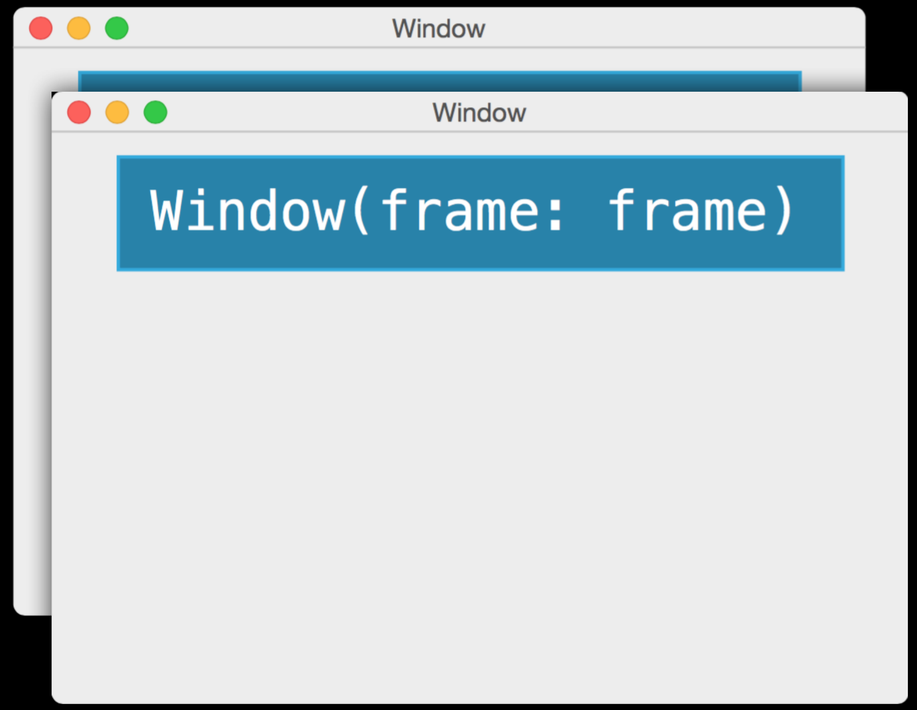
What if...

```
var window

setup(window)
```



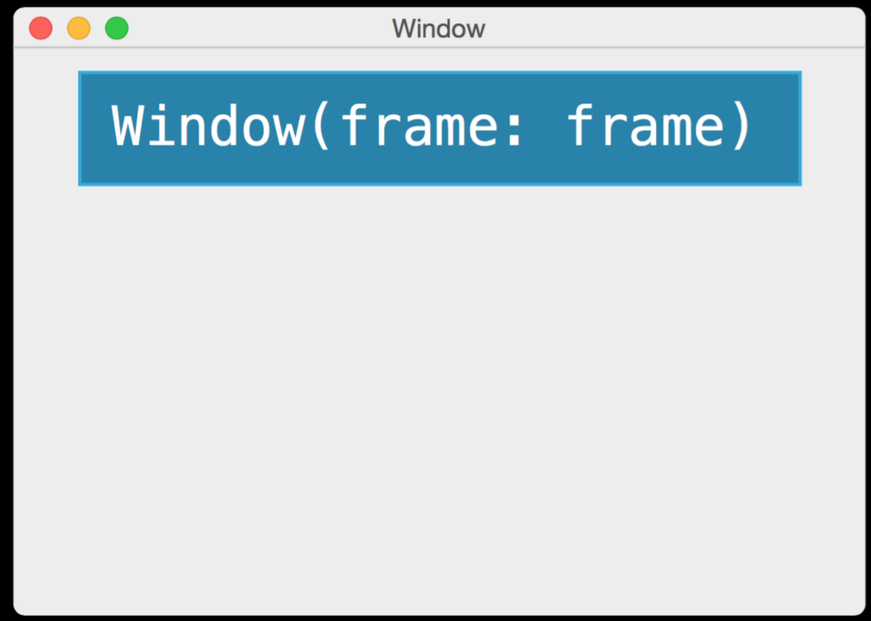
What if...

```
var window

setup(window)
```



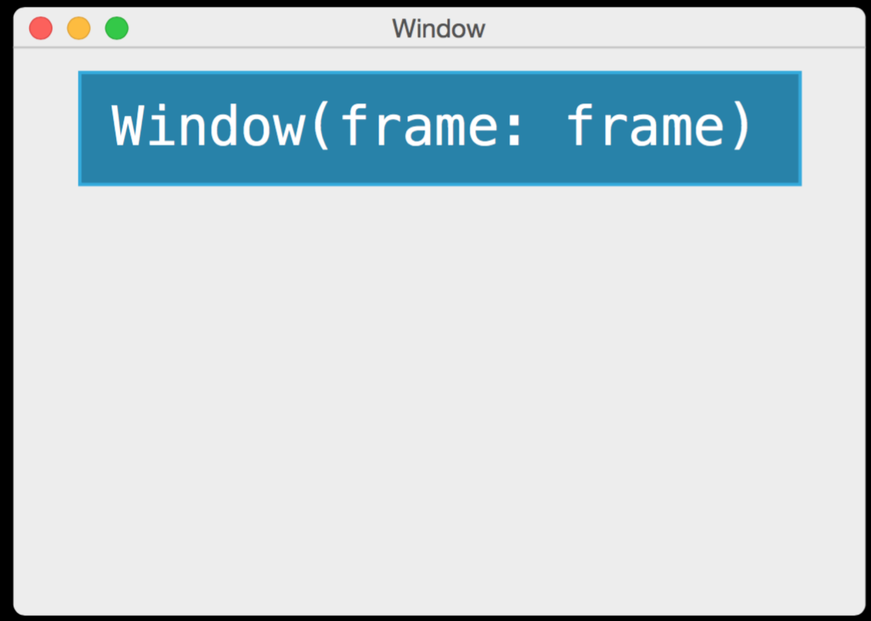
What if...

```
var window

setup(window)
```



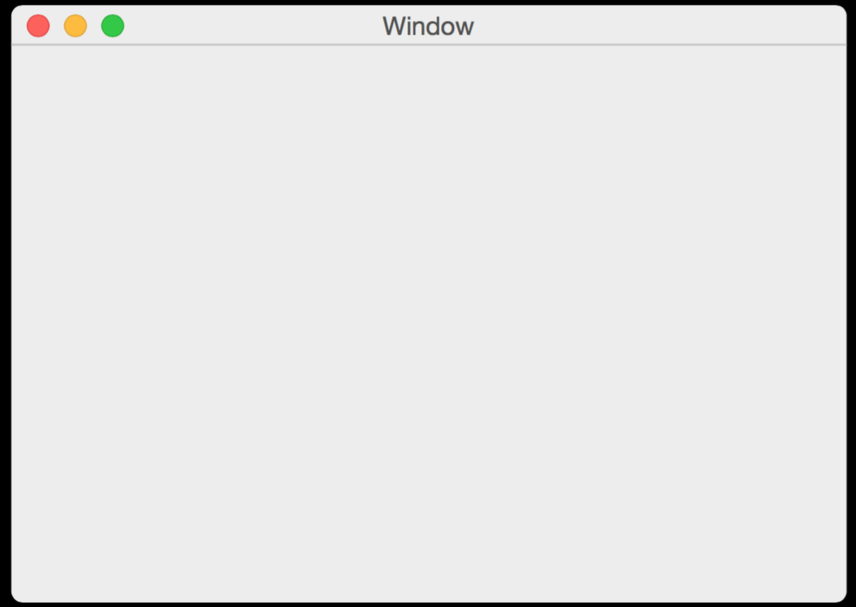
Class Instances are Passed by Reference

```
var window

setup(window)



func setup(window: Window) {
    // do some setup
}
```



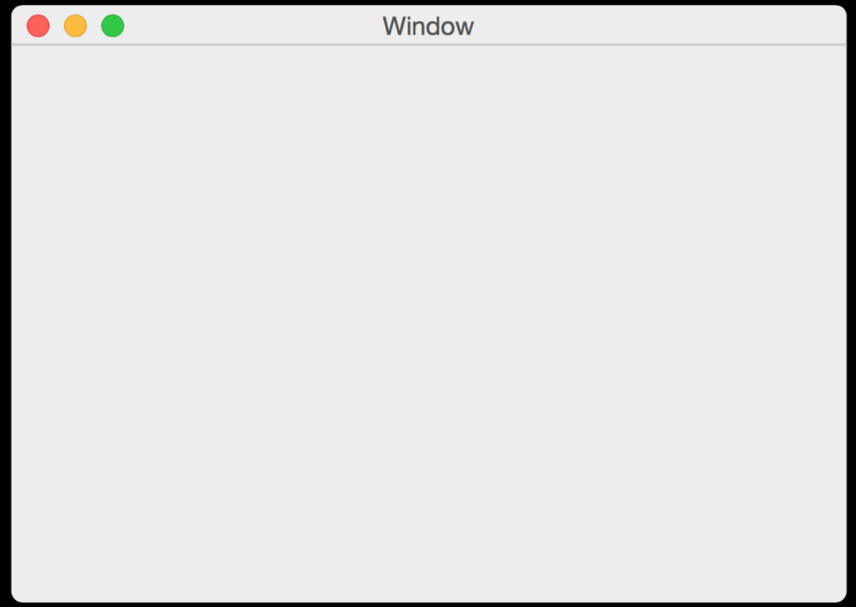
What if...

```
var newFrame = window.frame
```

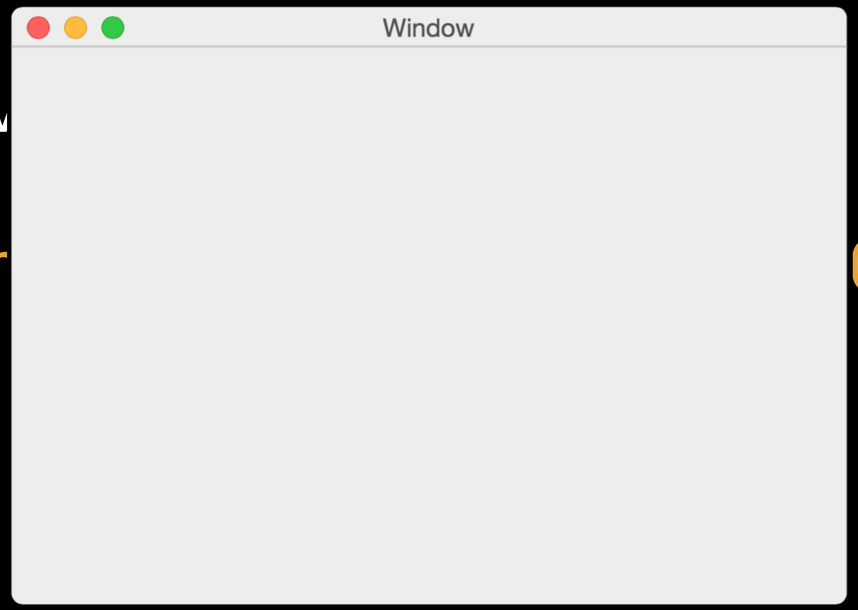


What if...

```
var newFrame = window.frame

newFrame.origin.x = 20.0
```



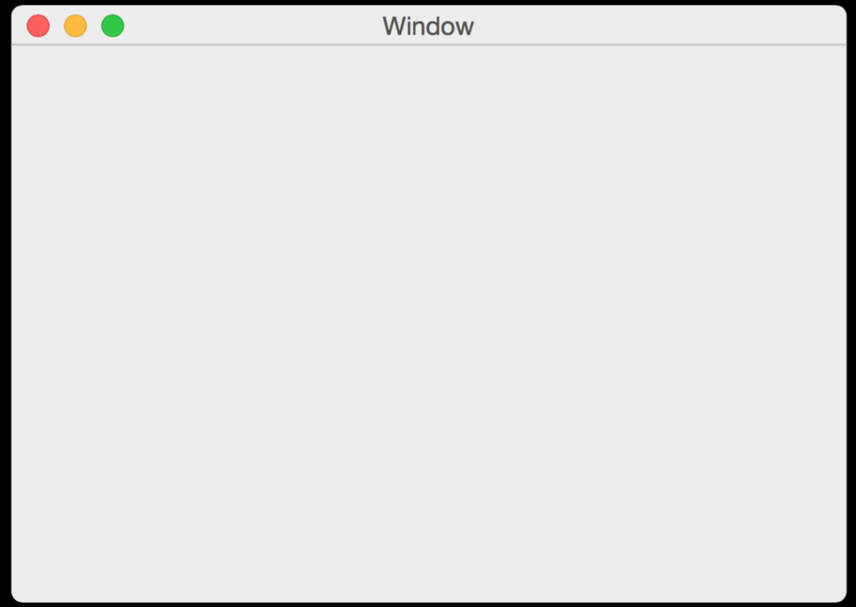
What if...




```
v                                    .frame

r                                    0.0
```

Structures are Passed by Value

```
var newFrame = window.frame

newFrame.origin.x = 20.0
```

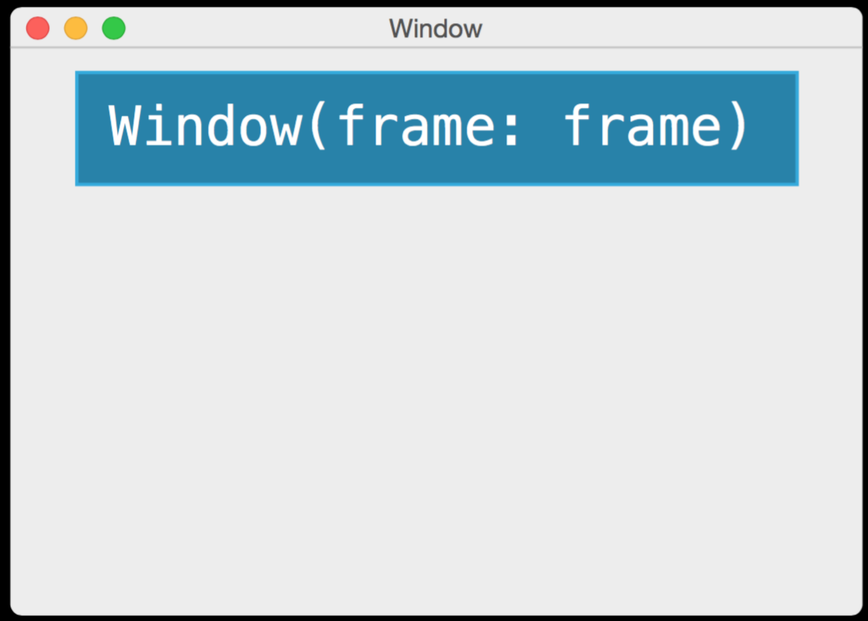


Constants and Variables:Reference Types

```
let window = Window(frame: frame)
```

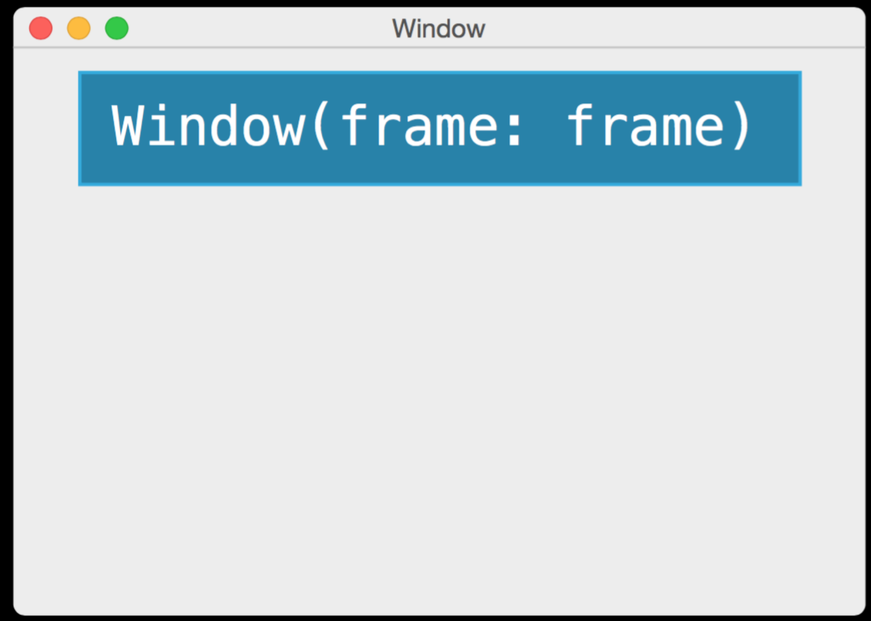
Constants and Variables:Reference Types

```
let window                  =
```



Constants and Variables:Reference Types

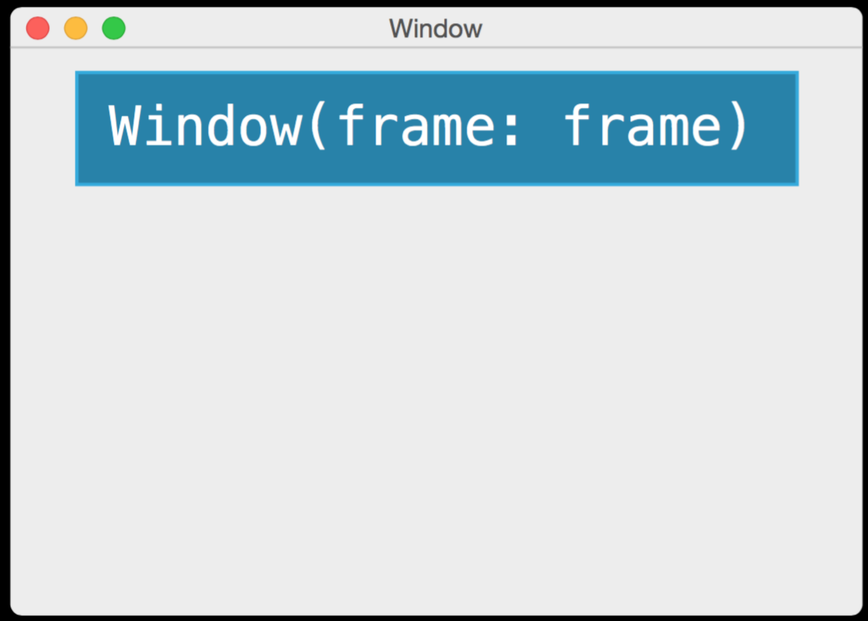
```
let window
```



Constants and Variables:Reference Types

```
let window
```

```
window.title = "Hello!"
```



Constants and Variables:Reference Types

```
let window
```

```
window.title = "Hello!"
```

Constants and Variables:Reference Types

```
let window
```


```
window.title = "Hello!"
```

```
window = Window(frame: frame)
```

Constants and Variables:Reference Types

```
let window
```


```
window.title = "Hello!"
```

```
window = Window(frame: frame)
// error: Cannot mutate a constant!
```

Constants and Variables: Value Types

```
var point1 = Point(x: 0.0, y: 0.0)
```

Constants and Variables: Value Types

```
var point1 = Point(x: 0.0, y: 0.0)
    point1
     x: 0.0
     y: 0.0
```

Constants and Variables: Value Types

```
var point1 = Point(x: 0.0, y: 0.0)
    point1
     x: 0.0
     y: 0.0

point1.x = 5
```

Constants and Variables: Value Types

```
var point1 = Point(x: 0.0, y: 0.0)
    point1
     x: 5.0
     y: 0.0

point1.x = 5
```

Constants and Variables: Value Types

```
var point1 = Point(x: 0.0, y: 0.0)
    x: 5.0
    y: 0.0

point1.x = 5

let point2 = Point(x: 0.0, y: 0.0)
    x: 0.0
    y: 0.0
```

Constants and Variables: Value Types

```
var point1 = Point(x: 0.0, y: 0.0)
     point1
      x: 5.0
      y: 0.0

point1.x = 5

let point2 = Point(x: 0.0, y: 0.0)
     point2
      x: 0.0
      y: 0.0

point2.x = 5
```

Constants and Variables: Value Types

```
var point1 = Point(x: 0.0, y: 0.0)
     x: 5.0
     y: 0.0

point1.x = 5

let point2 = Point(x: 0.0, y: 0.0)
     x: 0.0
     y: 0.0

point2.x = 5
// error: Cannot mutate a constant!
```

Mutating a Structure

```
struct Point {
 var x, y: Double
}
```

Mutating a Structure

```
struct Point {
    var x, y: Double

    func moveToTheRightBy(dx: Double) {
        x += dx
    }
}
```

Mutating a Structure

```
struct Point {
    var x, y: Double

    mutating func moveToTheRightBy(dx: Double) {
        x += dx
    }
}
```

Mutating a Structure

```
struct Point {
    var x, y: Double

    mutating func moveToTheRightBy(dx: Double) {
        x += dx
    }
}

let point = Point(x: 0.0, y: 0.0)
point.moveToTheRightBy(200.0)
// Error: Cannot mutate a constant!
```

Enumerations: Raw Values

```
enum Planet: Int {
    case Mercury = 1, Venus, Earth, Mars, Jupiter,
    Saturn, Uranus, Neptune
}
```

Enumerations: Raw Values

```
enum Planet: Int {
    case Mercury = 1, Venus, Earth, Mars, Jupiter,
    Saturn, Uranus, Neptune
}

let earthNumber = Planet.Earth.toRaw()
// earthNumber is 3
```

Enumerations: Raw Values

```
enum ControlCharacter: Character {
    case Tab = "\t"
    case Linefeed = "\n"
    case CarriageReturn = "\r"

}
```

Enumerations

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

Enumerations

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

var directionToHead = CompassPoint.West
// directionToHead is inferred to be a CompassPoint
```

Enumerations

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

var directionToHead = CompassPoint.West
// directionToHead is inferred to be a CompassPoint
directionToHead = .East
```

Enumerations

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

var directionToHead = CompassPoint.West
// directionToHead is inferred to be a CompassPoint
directionToHead = .East

let label = UILabel()
label.textAlignment = .Right
```

Enumerations: Associated Values

```
enum TrainStatus {
    case OnTime
    case Delayed(Int)
}
```

Enumerations: Associated Values


```
enum TrainStatus {
    case OnTime
    case Delayed(Int)
}
```

Enumerations: Associated Values


```
enum TrainStatus {
    case OnTime
    case Delayed(Int)
}

var status = TrainStatus.OnTime
// status is inferred to be a TrainStatus
```

Enumerations: Associated Values


```
enum TrainStatus {
    case OnTime
    case Delayed(Int)
}

var status = TrainStatus.OnTime
// status is inferred to be a TrainStatus

status = .Delayed(42)
```

Enumerations


```
enum TrainStatus {
    case OnTime, Delayed(Int)
}
```

Enumerations


```
enum TrainStatus {
    case OnTime, Delayed(Int)
    init() {
        self.OnTime
    }
}
```

Enumerations: Properties


```
enum TrainStatus {
     case OnTime, Delayed(Int)
     init() {
          self.OnTime
     }
     var description: String {
          switch self {
               case OnTime:
                  return "on time"
               case Delayed(let minutes):
                  return "delayed by \(minutes) minute(s)"
          }
     }
}
```

Enumerations


```
var status = TrainStatus()
```

Enumerations


```
var status = TrainStatus()

print("The train is \(status.description)")
// The train is on time
```

Enumerations


```
var status = TrainStatus()

print("The train is \(status.description)")
// The train is on time

status = .Delayed(42)
```

Enumerations


```
var status = TrainStatus()

print("The train is \(status.description)")
// The train is on time

status = .Delayed(42)

print("The train is now \(status.description)")
// The train is now delayed by 42 minute(s)
```

Nested Types

POWER

```
class Train {
    enum Status {
        case OnTime, Delayed(Int)
        init() {
            self = OnTime
        }
        var description: String { ... }
    }
    var status = Status()
}
```

Extensions

Extensions

```
extension Size {
   mutating func increaseByFactor(factor: Int) {
     width *= factor
     height *= factor
   }
}
```

Extensions

```
extension CGSize {
   mutating func increaseByFactor(factor: Int) {
     width *= factor
     height *= factor
   }
}
```

Extensions

```
extension Int {



}
```

Extensions

```
extension Int {
    func repetitions(task: () -> ()) {
        for i in 0..self {
            task()
        }
    }
}
```

Extensions

```
extension Int {
    func repetitions(task: () -> ()) {
        for i in 0..self {
            task()
        }
    }
}

500.repetitions({
    print("Hello!")
})
```

Extensions

```
extension Int {
    func repetitions(task: () -> ()) {
        for i in 0..self {
            task()
        }
    }
}

500.repetitions {
    print("Hello!")
}
```

A Non-Generic Stack Structure

```
struct IntStack {
    var elements = Int[]()

    mutating func push(element: Int) {
        elements.append(element)
    }

    mutating func pop() -> Int {
        return elements.removeLast()
    }
}
```

A Non-Generic Stack Structure

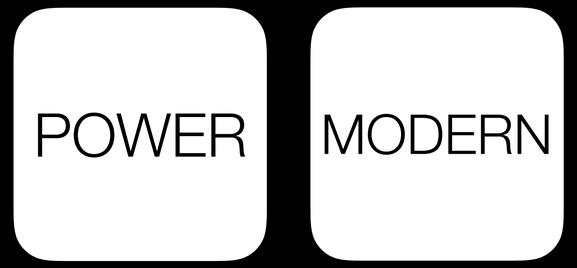
```
struct IntStack {
    var elements = Int[]()

    mutating func push(element: Int) {
        elements.append(element)
    }

    mutating func pop() -> Int {
        return elements.removeLast()
    }
}
```

A Generic Stack Structure

POWER MODERN

```
struct Stack<T> {
    var elements = T[]()

    mutating func push(element: T) {
        elements.append(element)
    }

    mutating func pop() -> T {
        return elements.removeLast()
    }
}
```

A Generic Stack Structure




```
struct Stack<T> {
    ...
}

var intStack = Stack<Int>()
intStack.push(50)
let lastIn = intStack.pop()
```

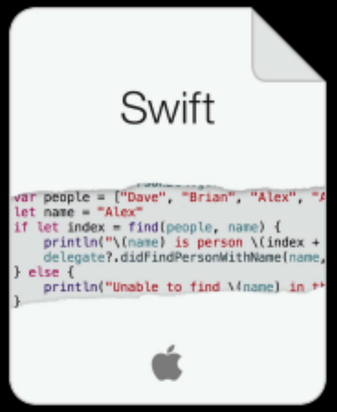
A Generic Stack Structure

POWER MODERN

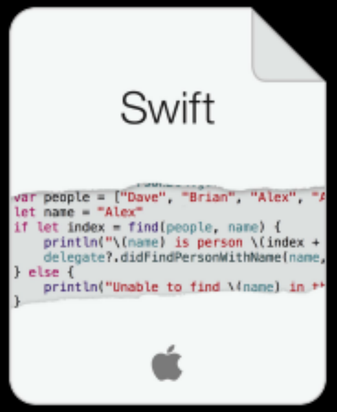
```
struct Stack<T> {
    ...
}

var intStack = Stack<Int>()
intStack.push(50)
let lastIn = intStack.pop()

var stringStack = Stack<String>()
stringStack.push("Hello")
println(stringStack.pop())
```

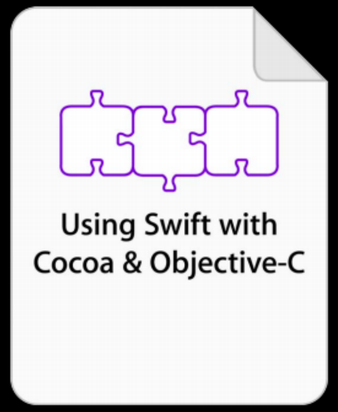
Resources

Resources


Swift
var people = ["Dave", "Brian", "Alex",
let name = "Alex"
if let index = find(people, name) {
println("\(name) is person \(index +
delegate?.didFindPersonWithName(name,
} else {
println("Unable to find \(name) in
}

Resources


Swift
var people = ["Dave", "Brian", "Alex",
let name = "Alex"
if let index = find(people, name) {
println("\(name) is person \(index +
delegate?.didFindPersonWithName(name,
} else {
println("Unable to find \(name) in
}


Using Swift with
Cocoa & Objective-C