```swift
var people = ["Dave", "Brian", "Alex", "Peter"]
let name = "Alex"
if let index = find(people, name) {
    println("\(name) is person \(index + 1)"
} else {
    println("Unable to find \(name) in list")
```
370,000

iBooks Store downloads
#include <stdio.h>

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

println("hello, WWDC")
What You Will Learn
What You Will Learn
What You Will Learn

SAFE

MODERN
What You Will Learn

SAFE

MODERN

POWER
The Basics

Dave Addey
Developer Publications Engineer
Variables
Variables

var
Variables

```javascript
var languageName
```
Variables

var languageName:
Variables

var languageName: String
Variables

var languageName: String = "Swift"
Variables

let languageName: String = "Swift"
Constants and Variables

let languageName: String = "Swift"
let languageName: String = "Swift"
var version: Double = 1.0
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

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

var version: Double = 1.0

let introduced: Int = 2014

let isAwesome: Bool = true
let languageName: String = "Swift"
var version: Double = 1.0
let introduced: Int = 2014
let isAwesome: Bool = true
Type Inference

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
let languageName = "Swift"
var version = 1.0
let introduced = 2014
let isAwesome = true
let \( \pi \) = 3.1415927
let languageName = "Swift"
var version = 1.0
let introduced = 2014
let isAwesome = true
let π = 3.1415927
let 🐶🐮 = "dogcow"
String
let someString = "I appear to be a string"
let someString = "I appear to be a string"
// inferred to be of type String
let someString = "I appear to be a string"
// inferred to be of type String

urlRequest.HTTPMethod = "POST"
let someString = "I appear to be a string"
// inferred to be of type String

urlRequest.HTTPMethod = "POST"

let components = "~/Documents/Swift".pathComponents
let someString = "I appear to be a string"
// inferred to be of type String

urlRequest.HTTPMethod = "POST"

let components = "~/Documents/Swift".pathComponents
// ["~", "Documents", "Swift"]
Character
for character in "mouse" {
    println(character)
}

Character
for character in "mouse" { 
    println(character)
}
Character

for character in "mús" {
    println(character)
}

m
ú
ús
Character

for character in "мышь" {
    println(character)
}

М
ышь
Character

for character in "鼠标" {
    println(character)
}

鼠标
for character in "🐱🐱🐱🐱🐱" {
    println(character)
}
Combining Strings and Characters

let dog: 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
// dogCow is "🐶🐮"
Combining Strings and Characters

let dog: Character = "🐶"
let cow: Character = "🐮"
let dogCow = dog + cow
// dogCow is "🐶🐮"

let instruction = "Beware of the " + dog
// instruction 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"
let a = 3, b = 5

// "3 times 5 is 15"

let mathResult = "\(a) times \(b) is \(a \times b)"
String Interpolation

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

// "3 times 5 is 15"

let mathResult = "\(a) times \(b) is \(a * b)"

// "3 times 5 is 15"
```
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

```javascript
var variableString = "Horse"
variableString += " and carriage"
```
String Mutability

```javascript
var variableString = "Horse"
variableString += " and carriage"
// variableString is now "Horse and carriage"
```
String Mutability

```swift
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 += " and another Highlander"
String Mutability

```javascript
var variableString = "Horse"
variableString += " and carriage"
// variableString is now "Horse and carriage"

let constantString = "Highlander"
constantString += " and another Highlander"
// error - constantString cannot be changed
```
Array and Dictionary
let components = "~/Documents/Swift".pathComponents
let components = "~/Documents/Swift".pathComponents
// ["~", "Documents", "Swift"]
// returns an Array, not an NSArray
Array and Dictionary Literals
Array and Dictionary Literals

```javascript
var names = ["Anna", "Alex", "Brian", "Jack"]
```
Array and Dictionary Literals

var names = ["Anna", "Alex", "Brian", "Jack"]
Array and Dictionary Literals

```javascript
var names = ["Anna", "Alex", "Brian", "Jack"]

var numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]
```
Array and Dictionary Literals

```javascript
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]
Arrays and Dictionaries

var names = ["Anna", "Alex", "Brian", "Jack"]

var numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]
Arrays and Dictionaries

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"]
var names = ["Anna", "Alex", "Brian", "Jack", 42]
var names = ["Anna", "Alex", "Brian", "Jack", true]
Typed Collections

```javascript
var names = ["Anna", "Alex", "Brian", "Jack", Bicycle()]
```
Typed Collections

var names = ["Anna", "Alex", "Brian", "Jack"]
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

```swift
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

```swift
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

```swift
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 var doctor = 1; doctor <= 13; ++doctor {
    exterminate(doctor)
}
For-In: Strings and Characters

```swift
for character in "🐭🐭🐭🐭🐭" {
    println(character)
}
```
for number in 1...5 {
    println("\(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: Ranges

```swift
for number in 0..5 {
    println("\(number) times 4 is \(number * 4)")
}
```

0 times 4 is 0
1 times 4 is 4
2 times 4 is 8
3 times 4 is 12
4 times 4 is 16
for name in \["Anna", "Alex", "Brian", "Jack"\] { 
    println("Hello, \(name)!")
}
let numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]

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

ants have 6 legs
snakes have 0 legs
cheetahs have 4 legs
let numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]

for (animalName, legCount) in numberOfLegs {
    println("\(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

```javascript
var shoppingList = ["Eggs", "Milk"]
```
Modifying an Array

```javascript
var shoppingList = ["Eggs", "Milk"]
println(shoppingList[0])
```
Modifying an Array

```javascript
var shoppingList = ["Eggs", "Milk"]
println(shoppingList[0])
```

"Eggs"
Modifying an Array

```javascript
var shoppingList = ["Eggs", "Milk"]
println(shoppingList[0])
shoppingList += "Flour"
```
Modifying an Array

```javascript
var shoppingList = ["Eggs", "Milk"]
println(shoppingList[0])
shoppingList += "Flour"
```
Modifying an Array

```javascript
var shoppingList = ["Eggs", "Milk"]
println(shoppingList[0])
shoppingList += "Flour"
shoppingList += ["Cheese", "Butter", "Chocolate Spread"]
```
Modifying an Array

```javascript
var shoppingList = ["Eggs", "Milk"]
println(shoppingList[0])
shoppingList += "Flour"
shoppingList += ["Cheese", "Butter", "Chocolate Spread"]

["Eggs", "Milk", "Flour", "Cheese", "Butter", "Chocolate Spread"]
```
Modifying an Array

```javascript
var shoppingList = ["Eggs", "Milk"]
println(shoppingList[0])
shoppingList += "Flour"
shoppingList += ["Cheese", "Butter", "Chocolate Spread"]
shoppingList[0] = "Six eggs"

["Eggs", "Milk", "Flour", "Cheese", "Butter", "Chocolate Spread"]
```
Modifying an Array

```javascript
var shoppingList = ["Eggs", "Milk"]
println(shoppingList[0])
shoppingList += "Flour"
shoppingList += ["Cheese", "Butter", "Chocolate Spread"]
shoppingList[0] = "Six eggs"

["Six eggs", "Milk", "Flour", "Cheese", "Butter", "Chocolate Spread"]
```
Modifying an Array

```javascript
var shoppingList = ["Eggs", "Milk"]
println(shoppingList[0])
shoppingList += "Flour"
shoppingList += ["Cheese", "Butter", "Chocolate Spread"]
shoppingList[0] = "Six eggs"
shoppingList[3...5] = ["Bananas", "Apples"]
```

```
["Six eggs", "Milk", "Flour", "Cheese", "Butter", "Chocolate Spread"]
```
Modifying an Array

```javascript
var shoppingList = ["Eggs", "Milk"]
println(shoppingList[0])
shoppingList += "Flour"
shoppingList += ["Cheese", "Butter", "Chocolate Spread"]
shoppingList[0] = "Six eggs"
shoppingList[3...5] = ["Bananas", "Apples"]

["Six eggs", "Milk", "Flour", "Bananas", "Apples"]
```
Modifying a Dictionary
Modifying a Dictionary

```javascript
var numberOfLegs = {"ant": 6, "snake": 0, "cheetah": 4}
```
Modifying a Dictionary

```
var numberOfLegs = {"ant": 6, "snake": 0, "cheetah": 4}
```

Modifying a Dictionary

```javascript
var numberOfLegs = [{'ant': 6, 'snake': 0, 'cheetah': 4}]
numberOfLegs['spider'] = 273
```
Modifying a Dictionary

```javascript
var numberOfLegs = {"ant": 6, "snake": 0, "cheetah": 4}
numberOfLegs["spider"] = 273

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

```javascript
var numberOfLegs = {"ant": 6, "snake": 0, "cheetah": 4}
numberOfLegs["spider"] = 273
numberOfLegs["spider"] = 8
```

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

```javascript
var numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]
// aardvark?
```
Retrieving a Value from a Dictionary

```javascript
var numberOfLegs = {"ant": 6, "snake": 0, "cheetah": 4}
// aardvark?
// dugong?
```
Retrieving a Value from a Dictionary

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

????????
Beyond the Basics

Tim Isted
Developer Publications Engineer
let numberOfLegs = "ant": 6, "snake": 0, "cheetah": 4

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

let possibleLegCount: Int? = numberOfLegs["aardvark"]
let numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]

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

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

Querying an Optional
let numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]

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

if possibleLegCount == nil {
    println("Aardvark wasn't found")
} else {
    let legCount = possibleLegCount!
    println("An aardvark has \(legCount) legs")
}
let numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]

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

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

    println("An aardvark has \((legCount) legs")
}
if let legCount = possibleLegCount {
    println("An aardvark has \(legCount) legs")
}
Unwrapping an Optional

```swift
if let legCount = possibleLegCount {
    println("An aardvark has \(legCount) legs")
}
```
If Statements

```java
if legCount == 0 {
    println("It slithers and slides around")
} else {
    println("It walks")
}
```
If Statements

```java
if (legCount == 0) {
    println("It slithers and slides around")
} else {
    println("It walks")
}
```
if legCount == 0 {
    println("It slithers and slides around")
} else {
    println("It walks")
}
If Statements

```java
if legCount == 0 {
    println("It slithers and slides around")
} else {
    println("It walks")
}
```
More Complex If Statements

```java
if legCount == 0 {
    println("It slithers and slides around")
} else if legCount == 1 {
    println("It hops")
} else {
    println("It walks")
}
```
switch legCount {
    case 0:
        println("It slithers and slides around")
    case 1:
        println("It hops")
    default:
        println("It walks")
}
Switch

switch sender {
    case executeButton:
        println("You tapped the Execute button")
    case firstNameTextField:
        println("You tapped the First Name text field")
    default:
        println("You tapped some other object")
}
Switch

```java
switch legCount {
    case 0:
        println("It slithers and slides around")
    case 1, 3, 5, 7, 9, 11, 13:
        println("It limps")
    case 2, 4, 6, 8, 10, 12, 14:
        println("It walks")
}
```
Switch

switch legCount {
    case 0:
        println("It slithers and slides around")
    
    case 1, 3, 5, 7, 9, 11, 13:
        println("It limps")
    
    case 2, 4, 6, 8, 10, 12, 14:
        println("It walks")
}
// error: switch must be exhaustive
Switch

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

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

  default:
    println("It walks")
}
```
switch legCount {
    case 0:
        println("It slithers and slides around")
    case 1, 3, 5, 7, 9, 11, 13:
        println("It limps")
    default:
        println("It walks")
}
Matching Value Ranges

```java
switch legCount {
    case 0:
        println("It has no legs")
    case 1...8:
        println("It has a few legs")
    default:
        println("It has lots of legs")
}
```
Matching Value Ranges

```java
switch legCount {
    case 0:
        println("It has no legs")
    case 1...8:
        println("It has a few legs")
    default:
        println("It has lots of legs")
}
```
Matching Value Ranges

```swift
switch legCount {
    case 0:
        println("It has no legs")
    case 1...8:
        println("It has a few legs")
    default:
        println("It has lots of legs")
}
```
switch legCount {
    case 0:
        println("It has no legs")
    case 1...8:
        println("It has a few legs")
    default:
        println("It has lots of legs")
}
Functions

```go
func sayHello() {
    println("Hello!")
}
```
func sayHello() {
    println("Hello!")
}

sayHello()
func sayHello() {
    println("Hello!")
}

sayHello()

Hello
func sayHello(name: String) {
    println("Hello \(name)!")
}
Functions with Parameters

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

sayHello("WWDC")
```
func sayHello(name: String) {
    println("Hello \$(name)!")
}

sayHello("WWDC")

Hello WWDC!
Default Parameter Values

```swift
func sayHello(name: String = "World") {
    println("Hello \(name)!")
}
```
Default Parameter Values

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

sayHello()
```
Default Parameter Values

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

sayHello()
```

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

sayHello()
sayHello(name: "WWDC")

Hello World!
Default Parameter Values

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

sayHello()
sayHello(name: "WWDC")
```

Hello World!
Hello WWDC!
Returning Values

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

let greeting = buildGreeting()
Returning Values

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

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

let greeting = buildGreeting()

println(greeting)
Returning Values

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

let greeting = buildGreeting()

println(greeting)
```

Hello World
func refreshWebPage() -> (Int, String) {
    // ...try to refresh...
    return (200, "Success")
}
Returning Multiple Values

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

(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

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

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

let (statusCode, message) = refreshWebPage()
```
Decomposing a Tuple

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

let (statusCode, message) = refreshWebPage()

println("Received \(statusCode): \(message)")
```
Decomposing a Tuple

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

let (statusCode, message) = refreshWebPage()

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

Received 200: Success
Decomposing a Tuple

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

let (statusCode: Int, message: String) = refreshWebPage()

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

```
Received 200: Success
```
let numberOfLegs = ["ant": 6, "snake": 0, "cheetah": 4]

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

ants have 6 legs
snakes have 0 legs
cheetahs have 4 legs
func refreshWebPage() -> (Int, String) {
    // ...try to refresh...
    return (200, "Success")
}
Named Values in a Tuple

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

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

let status = refreshWebPage()

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

let status = refreshWebPage()

println("Received \(status.code): \(status.message)")

Received 200: Success
Closures

let greetingPrinter = {
    println("Hello World!")
}
Closures

let greetingPrinter: () -> () = {
    println("Hello World!")
}
Closures

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

func greetingPrinter() -> () {
    println("Hello World!")
}
Closures

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

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

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

greetingPrinter()
Closures

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

greetingPrinter()
Closures as Parameters

```swift
func repeat(count: Int, task: () -> ()) {
    for i in 0..<count {
        task()
    }
}
```
Closures as Parameters

```swift
func repeat(count: Int, task: () -> ()) {
    for i in 0..<count {
        task()
    }
}
repeat(2, {
    println("Hello!")
})
```
func repeat(count: Int, task: () -> ()) {
    for i in 0..<count {
        task()
    }
}
repeat(2, {
    println("Hello!")
})
Hello!
Hello!
Trailing Closures

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

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

Dave Addey
Developer Publications Engineer
Classes
class Vehicle {

}
Classes

class Vehicle {
}

class Vehicle {
}

}
class Vehicle {
    // properties
}

class Vehicle {
    // properties
    // methods
}
class Vehicle {
    // properties
    // methods
    // initializers
}
import "Vehicle.h"

class Vehicle {

}
import "Vehicle.h"

class Vehicle {
}

```
Classes

class Vehicle {
}

class Vehicle: ?????????
{
}

Classes
class Vehicle: ????????
{
}

Classes
Classes

class Vehicle: NSObject {
}

class Vehicle {
}

Class Inheritance

class Vehicle {
}

class Bicycle: Vehicle {
}
Class Inheritance

class Vehicle {
}

class Bicycle: Vehicle {
}
Class Inheritance

class Vehicle {
}

class Bicycle: Vehicle {
}
class Vehicle {
}

Class Inheritance
Properties

class Vehicle {
    var numberOfWheels = 0
}

Properties

class Vehicle {
    var numberOfWheels = 0
}

Properties

class Vehicle {
    let numberOfWheels = 0
}

Properties

class Vehicle {
    var numberOfWheels = 0
}

class Vehicle {
    var numberOfWheels = 0
}

Properties
class Vehicle {
  var numberOfWheels = 0
}

Properties
Stored Properties

class Vehicle {
    var numberOfWheels = 0
}

Computed Properties
Computed Properties

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

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

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 {
        return "\(numberOfWheels) wheels"
    }
}
Computed Properties

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

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

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

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

let someVehicle = Vehicle()
Type Inference

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

let someVehicle: Vehicle = Vehicle()
Default Values

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

let someVehicle = Vehicle()
let someVehicle = Vehicle()
Dot Syntax

let someVehicle = Vehicle()

println(someVehicle.description)
let someVehicle = Vehicle()

println(someVehicle.description)
// 0 wheels
let someVehicle = Vehicle()
println(someVehicle.description)
// 0 wheels

someVehicle.numberOfWheels = 2
let someVehicle = Vehicle()

println(someVehicle.description)
// 0 wheels

someVehicle.numberOfWheels = 2

println(someVehicle.description)
let someVehicle = Vehicle()

println(someVehicle.description)
// 0 wheels

someVehicle.numberOfWheels = 2

println(someVehicle.description)
// 2 wheels
class Bicycle: Vehicle {

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

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

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

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

let myBicycle = Bicycle()
println(myBicycle.description)
class Bicycle: Vehicle {
    init() {
        super.init()
        numberOfWheels = 2
    }
}

let myBicycle = Bicycle()
println(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
}

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 {
    }
}
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

```swift
let myCar = Car()

println(myCar.description)
// 4 wheels, 0.0 mph
```
let myCar = Car()

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

myCar.speed = 35.0
let myCar = Car()

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

myCar.speed = 35.0

println(myCar.description)
// 4 wheels, 35.0 mph
Property Observers
Property Observers

class ParentsCar: Car {
}

class ParentsCar: Car {
    override var speed: Double {

    }
}

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

        }
        didSet {

        }
        didSet {

        }
    }
}
class ParentsCar: Car {
    override var speed: Double {
        didSet {
            // newValue is available here
        }
        didSet {
        }
        didSet {
        }
    }
}
class ParentsCar: Car {
    override var speed: Double {
        didSet {
            // oldValue is available here
        }
    }
}
Property Observers

class ParentsCar: Car {
    override var speed: Double {
        didSet {
            //update logic here
        }
    }
}

class ParentsCar: Car {
    override var speed: Double {
        didSet {
            if newValue > 65.0 {
                // Handle the change if the speed exceeds 65.0
            }
        }
    }
}
class ParentsCar: Car {
    override var speed: Double {
        didSet {
            if newValue > 65.0 {
                println("Careful now.")
            }
        }
    }
}
Methods
Methods

class Counter {
    var count = 0
}

}
class Counter {
    var count = 0
}

Methods

class Counter {
    var count = 0

    func increment() {
        count++
    }
}

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

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

Tim Isted
Developer Publications Engineer
Structures in Swift
Structures in Swift

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

struct Size {
    var width, height: Double
}

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

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

```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

```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
    }
}
```
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
    }
}
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 = Window(frame: frame)
Structure or Class?

```swift
var window
```

```swift
Window(frame: frame)
```
What if...

var window

setup(window)
What if...

```swift
var window

setup(window)
```
What if...

```swift
var window

setup(window)
```

```swift
Window(frame: frame)
```
Class Instances are Passed by Reference

```swift
var window

setup(window)

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

var newFrame = window.frame
What if...

```javascript
var newFrame = window.frame

newFrame.origin.x = 20.0
```
What if...

```javascript
let newFrame = window.frame!
newFrame.origin.x = 20.0
```
Structures are Passed by Value

var newFrame = window.frame

newFrame.origin.x = 20.0
let window = Window(frame: frame)
let window = Window(frame: frame)
let window → Window(frame: frame)
let window

window.title = "Hello!"
let window

window.title = "Hello!"
let window

window.title = "Hello!"

window = Window(frame: frame)
let window

window.title = "Hello!"

window = Window(frame: frame)

// error: Cannot mutate a constant!
var point1 = Point(x: 0.0, y: 0.0)
var point1 = Point(x: 0.0, y: 0.0)
Constants and Variables: Value Types

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

point1.x = 5
```
var point1 = Point(x: 0.0, y: 0.0)

point1.x = 5
Constants and Variables: Value Types

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

point1.x = 5

let point2 = Point(x: 0.0, y: 0.0)
```
Constants and Variables: Value Types

```swift
var point1 = Point(x: 0.0, y: 0.0)
point1.x = 5

let point2 = Point(x: 0.0, y: 0.0)
point2.x = 5
```
Constants and Variables: Value Types

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

`point1`.x = 5

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

`point2`.x = 5

// error: Cannot mutate a constant!
Mutating a Structure

```swift
struct Point {
    var x, y: Double
}
```
Mutating a Structure

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

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

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

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

```swift
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

```swift
enum Planet: Int {
    case Mercury = 1, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune
}
```
Enum Enumerations: Raw Values

class Enum

e = planets

e = earth

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

Enumerations: Raw Values

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

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

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

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

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

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

```swift
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

```swift
enum TrainStatus {
    case OnTime
    case Delayed(Int)
}
```
Enumerations: Associated Values

```swift
enum TrainStatus {
    case OnTime
    case Delayed(Int)
}
```
Enumerations: Associated Values

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

var status = TrainStatus.OnTime
// status is inferred to be a TrainStatus
```
Enumerations: Associated Values

eenum TrainStatus {
    case OnTime
    case Delayed(Int)
}

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

status = .Delayed(42)
Enumerations

```swift
enum TrainStatus {
    case OnTime, Delayed(Int)
}
```
Enumerations: Initializers

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

```swift
var status = TrainStatus()

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

```swift
var status = TrainStatus()

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

status = .Delayed(42)
```
var status = TrainStatus()

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

status = .Delayed(42)

println("The train is now \(status.description)")
// The train is now delayed by 42 minute(s)
class Train {
    enum Status {
        case OnTime, Delayed(Int)
        init() {
            self = OnTime
        }
        var description: String {
            ...
        }
    }
    var status = Status()
}
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({
    println("Hello!")
})
extension Int {
    func repetitions(task: () -> ()) {
        for i in 0..<self {
            task()
        }
    }
}

500.repetitions {
    println("Hello!")
}
A Non-Generic Stack Structure

```swift
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

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

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

    mutating func pop() -> Int {
        return elements.removeLast()
    }
}
```
A Generic Stack Structure

```swift
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

```swift
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

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

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

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

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

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

```swift
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

```
for person in people {
    let name = person
    if let index = find(person, name) {
        println("Found \(name) at index \(index) in \(people)"
            delegate?.didFindPersonWithName(name,""
    } else {
        println("Unable to find \(name) in \(people)"
    }
}
```

Using Swift with Cocoa & Objective-C
Resources

- **Intermediate Swift**
  - Presidio
  - Wednesday 2:00PM

- **Advanced Swift**
  - Presidio
  - Thursday 11:30AM
Resources

- Intermediate Swift
  - Presidio
  - Wednesday 2:00PM

- Advanced Swift
  - Presidio
  - Thursday 11:30AM

- Integrating Swift with Objective-C
  - Presidio
  - Wednesday 9:00AM

- Swift Interoperability in Depth
  - Presidio
  - Wednesday 3:15PM
More Information

Dave DeLong
Developer Tools Evangelist
delong@apple.com

Documentation
The Swift Programming Language
Using Swift with Cocoa and Objective-C
http://developer.apple.com

Apple Developer Forums
http://devforums.apple.com
Apple WWDC14