Getting Started with Swift
Session 404

Dave Addey Xcode Documentation Engineer
Brian Lanier Swift Documentation Team Manager
Alex Martini Swift Documentation Engineer
The Basics

Dave Addey Xcode Documentation Engineer
let language: String = "Swift"
let language: String = "Swift"
let language: String = "Swift"
let language: String = "Swift"
let language: String = "Swift"
let language: String = "Swift"
Constants

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

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

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

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

```swift
let language = "Swift"
let introduced = 2014
let isAwesome = true
```
Type Inference

let language = "Swift"  // Inferred as String
let introduced = 2014  // Inferred as Int
let isAwesome = true   // Inferred as Bool
let language = "Swift"
let introduced = 2014
let isAwesome = true
var version = 1
Variables

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

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

Error at isAwesome = false
let conference = "WWDC"

let message = "Hello, " + conference + "!
// "Hello, WWDC!"
String Interpolation

```swift
let conference = "WWDC"

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

```swift
let conference = "WWDC"
let year = 2016
let message = "Hello, \(conference) \(year + 1)!"
// "Hello, WWDC 2017!"
```
let instruction = "Beware of the 🐶🐮"
let instruction = "Beware of the 🐶🐮"
let internationalHarmony = "🇬🇧😍"
let instruction = "Beware of the 🐶🐮"
let internationalHarmony = "#$😍"

let π = 3.1415927
let 鼠标 = "⿷⿷"
let dogString = "Dog🐶"
let dogString = "Dog🐶"
let dogString = "Dog🐶"  
print("\(dogString) is \(dogString.characters.count) characters long")
let dogString = "Dog?!🐶"
print("\(dogString) is \(dogString.characters.count) characters long")

Dog?!🐶 is 5 characters long.
let dogString = "Dog?!🐶"

for character in dogString.characters {
    print(character)
}
let dogString = "Dog?!🐶"
for character in dogString.characters {
    print(character)
}
Array and Dictionary
let names = ["Lily", "Santiago", "Justyn", "Aadya"]
let names = ["Lily", "Santiago", "Justyn", "Aadya"]

let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
let names = [
    "Lily",
    "Santiago",
    "Justyn",
    "Aadya"
]

let ages = [
    "Mohsen": 17,
    "Amy": 40,
    "Graham": 5
]
let names = ["Lily", "Santiago", "Justyn", "Aadya", 42]

let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
Array and Dictionary

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

let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```
let names = ["Lily", "Santiago", "Justyn", "Aadya", Bicycle()]

let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
Array and Dictionary

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

let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
```
let names: [String] = ["Lily", "Santiago", "Justyn", "Aadya"]

let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
let names = ["Lily", "Santiago", "Justyn", "Aadya"]
// an array of String values

let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
let names = ["Lily", "Santiago", "Justyn", "Aadya"]
// an array of String values

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

While and Repeat-While

```plaintext
while !endOfFile {
    readLine()
}

repeat {
    performTask()
} while tasksRemaining > 0
```
let dogString = "Dog🐶"  
for character in dogString.characters {
    print(character)
}
For-In Loop

Characters

```swift
let dogString = "Dog?!🐶"
for character in dogString.characters {
    print(character)
}
```
for number in 1...5 {
    print("\(number) times 4 is \(number \times 4)")
}

For-In Loop
Closed Ranges
for number in 1...5 {
    print("\(number) times 4 is \(number * 4)")
}

1 times 4 is 4
2 times 4 is 8
3 times 4 is 12
4 times 4 is 16
5 times 4 is 20
for number in 1...5 {
    print("\(number) times 4 is \(number * 4)")
}

1 times 4 is 4
2 times 4 is 8
3 times 4 is 12
4 times 4 is 16
5 times 4 is 20
let results = [7, 52, 9, 33, 6, 12, 86, 4, 22, 18, 3]
let maxResultCount = 5
for index in 0..<maxResultCount {
    print("Result \(index) is \(results[index])")
}
let results = [7, 52, 9, 33, 6, 12, 86, 4, 22, 18, 3]
let maxResultCount = 5
for index in 0..<maxResultCount {
    print("Result \(index) is \(results[index])")
}

Result 0 is 7
Result 1 is 52
Result 2 is 9
Result 3 is 33
Result 4 is 6
let results = [7, 52, 9, 33, 6, 12, 86, 4, 22, 18, 3]
let maxResultCount = 5
for index in 0..<maxResultCount {
    print("Result \(index) is \(results[index])")
}
for name in \["Lily", "Santiago", "Justyn", "Aadya"\] {
    print("Hello, \(name)!")
}

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

Hello, Lily!
Hello, Santiago!
Hello, Justyn!
Hello, Aadya!
let ages = \["Mohsen": 17, "Amy": 40, "Graham": 5\]

for (name, age) in ages {
    print("\(name) is \(age) years old")
}

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

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

Mohsen is 17 years old
Amy is 40 years old
Graham is 5 years old
Modifying an Array

Packing for WWDC

```javascript
var packingList = ["Socks", "Shoes"]
```
Modifying an Array

Packing for WWDC

```javascript
var packingList = ["Socks", "Shoes"]
print(packingList[0])
```
Modifying an Array

Packing for WWDC

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

print(packingList[0])
```
Modifying an Array

Packing for WWDC

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

"Socks"
Modifying an Array

Packing for WWDC

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

["Socks", "Shoes", "Trousers"]
```
Modifying an Array
Packing for WWDC

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

["Socks", "Shoes", "Trousers"]
Modifying an Array

Packing for WWDC

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

["Socks", "Shoes", "Jeans"]
```
Modifying an Array

Packing for WWDC

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

["Socks", "Shoes", "Jeans"]
```
Modifying an Array

Packing for WWDC

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

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

Modifying an Array

Packing for WWDC

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

["Socks", "Shoes", "Jeans", "Hoodie", "Scarf"]
```
Modifying an Array

Packing for WWDC

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

["Socks", "Shoes", "Jeans", "Hoodie", "Scarf"]
Modifying a Dictionary

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

```javascript
["Mohsen": 17, "Amy": 40, "Graham": 5]
```
Modifying a Dictionary

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

```javascript
["Mohsen": 17, "Amy": 40, "Graham": 5, "Justyn": 67]
```
Modifying a Dictionary

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

```javascript
["Mohsen": 17, "Amy": 40, "Graham": 5, "Justyn": 68]
```
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
Retrieving a Value from a Dictionary

let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
// Devon?
Retrieving a Value from a Dictionary

```swift
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
// Devon?
// Daryl?
```
Retrieving a Value from a Dictionary

```swift
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]
// Devon?
// Daryl?
// Daniel?
```
Optionals

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

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

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

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

A value of 40, perhaps?
let ages = ["Mohsen": 17, "Amy": 40, "Graham": 5]

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

let possibleAge = ages["Daryl"]
Optionals

ages["Amy"]

ages["Daryl"]
Optionals

ages["Amy"]

40

Int

ages["Daryl"]
Optionals

ages["Amy"]

40

Int

ages["Daryl"]

No Value

No Int
Optionals

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

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

```swift
let ages = \"Mohsen\": 17, \"Amy\": 40, \"Graham\": 5

let possibleAge: Int? = ages[\"Daryl\"]
```
Checking for an Optional Value

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

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

if possibleAge == nil {
    print("Age not found.")
}
```
let ages = 
{"Mohsen": 17, "Amy": 40, "Graham": 5} 

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

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


Checking for an Optional Value

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

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

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

Age not found.
Checking for an Optional Value

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

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

if possibleAge == nil {
    print("Age not found.")
}
```
If-Let Statement

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

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

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

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

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

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

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

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

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

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

An age of 40 was found.
If Statement

```swift
let age = 32

if age == 1 {
    print("Happy first birthday!")
} else if age == 40 {
    print("Happy 40th birthday!")
} else {
    print("Happy plain old boring birthday.")
}
```
let age = 32

if age == 1 {
    print("Happy first birthday!")
} else if age == 40 {
    print("Happy 40th birthday!")
} else {
    print("Happy plain old boring birthday.")
}
let age = 32

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

}
Switch Statement

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

Switch Statement

```swift
let age = 15
switch age {
    case 1:
        print("Happy first birthday!")
    case 13...19:
        print("Happy birthday, teenager!")
}
```
let age = 40
switch age {
    case 1:
        print("Happy first birthday!")
    case 13...19:
        print("Happy birthday, teenager!")
        print("Happy birthday, teenager!")
    case let decade where decade % 10 == 0:
        print("Happy significant \(decade)th birthday!")

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

Switch Statement

```swift
let age = 40
switch age {
    case 1:
        print("Happy first birthday!")
    case 13...19:
        print("Happy birthday, teenager!")
        print("Happy birthday, teenager!")
    case let decade where decade % 10 == 0:
        print("Happy significant \(decade)th birthday!")
}
```
let age = 41
switch age {
    case 1:
        print("Happy first birthday!")
    case 13...19:
        print("Happy birthday, teenager!")
        print("Happy birthday, teenager!")
    case let decade where decade % 10 == 0:
        print("Happy significant \(\text{(decade)}\)th birthday!")
        print("Happy significant \(\text{(decade)}\)th birthday!")
}


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

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

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

let userName = "admin"
let passwordIsValid = true
Switch Statement

let userName = "admin"
let passwordIsValid = true
Switch Statement

let userName = "admin"
let passwordIsValid = true
Switch Statement

let userName = "admin"
let passwordIsValid = true

switch (userName, passwordIsValid) {
}
Switch Statement

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

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


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

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

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

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

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

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

Guests are not allowed in this restricted area.
let userName = "bob"
let passwordIsValid = true

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

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

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

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

Brian Lanier Swift Documentation Team Manager
Functions

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

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

sendMessage()
```

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

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

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

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

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

sendMessage(recipient: "Morgan", shouting: false)
Multiple Parameters

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

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

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

sendMessage(message: "See you at the Bash", to: "Morgan", shouting: false)
Omitting Argument Labels

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

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

```swift
func firstString(havingPrefix prefix: String, in strings: [String]) -> String {
    // Implementation goes here
}
```
func firstString(havingPrefix prefix: String, in strings: [String]) -> String {…}
Function Return Values

```swift
func firstString(havingPrefix prefix: String, in strings: [String]) -> String {
    // Implementation...
}
```
Function Return Values

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

Function Return Values

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

Function Return Values
func firstString(havingPrefix prefix: String, in strings: [String]) -> String {
    for string in strings {
        if string.hasPrefix(prefix) {
            return string
        }
    }
    return
}
func firstString(havingPrefix prefix: String, in strings: [String]) -> String {
  for string in strings {
    if string.hasPrefix(prefix) {
      return string
    }
  }
  return ""
}
func firstString(havingPrefix prefix: String, in strings: [String]) -> String? {
    for string in strings {
        if string.hasPrefix(prefix) {
            return string
        }
    }
    return nil
}
Returning Optional Values

```swift
var guests = ["Jack", "Kumar", "Anita", "Anna"]

if let guest = firstString(havingPrefix: "A", in: guests) {
    print("See you at the party, \(guest)!"")
} else {
    print("Invite must be in the mail.")
}
```
Returning Optional Values

```
var guests = ["Jack", "Kumar", "Anita", "Anna"]

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

See you at the party, Anita!
Filtering Numbers
Filtering Numbers

numbers
4  17  34  41  82

Include Number?

result
Filtering Numbers

numbers: 4 17 34 41 82

divisibleByTwo(4)

Include Number?

result
Filtering Numbers

numbers: 4 17 34 41 82

includeNumber: divisibleByTwo(4)

result: 4
Filtering Numbers

divisibleByTwo(17)

Include Number?

4 17 34 41 82

result 4
Filtering Numbers

numbers: 4, 17, 34, 41, 82

divisibleByTwo(17) = False

Include Number?: False

result: 4
Filtering Numbers

numbers:
- 4
- 17
- 34
- 41
- 82

Include Number?

result:
- 4
- 34
- 82

divisibleByTwo(82)
func filterInts(_ numbers: [Int], _ includeNumber: type) -> [Int] {...}
Function Types

( parameter types ) -> return type
Function Types

(parameter types) -> return type

def sendMessage() {...}
Function Types

( parameter types ) -> return type

func sendMessage() {...}

() -> Void
Function Types

(\textbf{parameter types}) \rightarrow \textbf{return type}

\textbf{func} firstString(havingPrefix prefix: \texttt{String}, in strings: [\texttt{String}]) \rightarrow \texttt{String}? \{\ldots\}
Function Types

(\text{parameter types}) \rightarrow \text{return type}

\textbf{func} \ firstString(havingPrefix \ prefix: \text{String}, \text{in} \ strings: \text{[String]}) \rightarrow \text{String}\? \{\ldots\}

(\text{String, [String]}) \rightarrow \text{String}\?
Functions as Parameters

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

}
Functions as Parameters

```swift
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {
    var result: [Int] = []
    for number in numbers {
        // Your condition here
    }
    return result
}
```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {
    var result: [Int] = []
    for number in numbers {
        if includeNumber(number) {
            result.append(number)
        }
    }
    return result
}
Functions as Parameters

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

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

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

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

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

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

let evenNumbers = filterInts(numbers, divisibleByTwo)
```
Functions as Arguments

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

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

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

[4, 34, 82]
```swift
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {
    let evenNumbers = filterInts(numbers, divisibleByTwo)
    print(evenNumbers)
}

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

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

[4, 34, 82]
Functions as Arguments

```swift
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {
    // Function implementation
}

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

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

```
[4, 34, 82]
```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {...}

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

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

let evenNumbers = filterInts(numbers, divisibleByTwo)

print(evenNumbers)

[4, 34, 82]
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {
    let evenNumbers = numbers.filter { includeNumber($0) }  
    return evenNumbers
}

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

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

[4, 34, 82]
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {…}

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

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

[4, 34, 82]
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {
    ...}

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

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

[4, 34, 82]
Closure Expressions

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

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

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

[4, 34, 82]
```
Closure Expressions

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

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

// Define a closure that checks if a number is divisible by 2
func divisibleByTwo(_ number: Int) -> Bool {
    return number % 2 == 0
}

// Filter the numbers based on the closure
let evenNumbers = filterInts(numbers, { (number: Int) -> Bool in return number % 2 == 0 })

print(evenNumbers)
```

[4, 34, 82]
Type Inference in Closures

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

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

[4, 34, 82]
```
Type Inference in Closures

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

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

[4, 34, 82]
```
Type Inference in Closures

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

let evenNumbers = filterInts(numbers, {
    number in return number % 2 == 0
})
print(evenNumbers)

// Output: [4, 34, 82]
```
Type Inference in Closures

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

let evenNumbers = filterInts(numbers, {
    number in return number % 2 == 0
})

print(evenNumbers)
```

[4, 34, 82]
Type Inference in Closures

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

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

[4, 34, 82]
Implicit Arguments in Closures

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

let evenNumbers = filterInts(numbers, { number in
    return number % 2 == 0
})

print(evenNumbers)

// Output: [4, 34, 82]
```
Implicit Arguments in Closures

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

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

print(evenNumbers)
```

[4, 34, 82]
Implicit Arguments in Closures

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

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

[4, 34, 82]
Implicit Arguments in Closures

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

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

[4, 34, 82]
Trailing Closures

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

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

[4, 34, 82]
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {
    // Implementation...
}

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

[4, 34, 82]
Trailing Closures

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

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

[4, 82]
let names = ["Lily", "Santiago", "Aadya", "Jack", "Anna"]

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

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

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

print(shortNames)
```

```
[Lily, Jack, Anna]
```
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {
    var result: [Int] = []
    for number in numbers {
        if includeNumber(number) {
            result.append(number)
        }
    }
    return result
}
func filterInts(_ numbers: [Int], _ includeNumber: (Int) -> Bool) -> [Int] {
    var result: [Int] = []
    for number in numbers {
        if includeNumber(number) {
            result.append(number)
        }
    }
    return result
}
func filterStrings(_ strings: [Int], _ includeString: (Int) -> Bool) -> [Int] {
    var result: [Int] = []
    for string in strings {
        if includeString(string) {
            result.append(string)
        }
    }
    return result
}
func filterStrings(_ strings: [Int], _ includeString: (Int) -> Bool) -> [Int] {
    var result: [Int] = []
    for string in strings {
        if includeString(string) {
            result.append(string)
        }
    }
    return result
}
func filterStrings(_ strings: [String], _ includeString: (String) -> Bool) -> [String] {
    var result: [String] = []
    for string in strings {
        if includeString(string) {
            result.append(string)
        }
    }
    return result
}
func filterStrings(_ strings: [String], _ includeString: (String) -> Bool) -> [String] {
    var result: [String] = []
    for string in strings {
        if includeString(string) {
            result.append(string)
        }
    }
    return result
}
func filter(_ source: [Element], _ includeElement: (Element) -> Bool) -> [Element] {
    var result: [Element] = []
    for element in source {
        if includeElement(element) {
            result.append(element)
        }
    }
    return result
}
Generic Type Parameters

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

Calling Generic Functions

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

let evenNumbers = filter(numbers) { $0 % 2 == 0 }
let shortNames = filter(names) { name in name.characters.count < 5 }
```
Filter and Map Methods

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

let shortNames = names.filter { name in name.characters.count < 5 }
```
Filter and Map Methods

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

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

["Lily", "Jack", "Anna"]
```
Filter and Map Methods

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

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

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

[Lily, Jack, Anna]
Filter and Map Methods

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

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

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

[Lily, Jack, Anna]
[LILY, JACK, ANNA]
```
let names = ['Lily', 'Santiago', 'Aadya', 'Jack', 'Anna', 'Andrés']

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

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

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

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

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

print(capitalizedShortNames)

[LILY, JACK, ANNA]
```
Custom Types

Alex Martini Swift Documentation Engineer
Structures

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

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

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

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

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

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

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

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

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

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

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

```swift
struct Rectangle {
    var width: Int
    var height: Int
    var area: Int
}
```
Computed Properties

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

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

let rectangle = Rectangle(width: 4, height: 5)
print("Width is \(rectangle.width) and area is \(rectangle.area).")
```
Computed Properties

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

let rectangle = Rectangle(width: 4, height: 5)
print("Width is \(rectangle.width) and area is \(rectangle.area).")
```

Width is 4 and area is 20.
Computed Properties

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

let rectangle = Rectangle(width: 4, height: 5)
print("Width is \(rectangle.width) and area is \(rectangle.area).")
```

Width is 4 and area is 20.
Computed Properties

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

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

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

let small = Rectangle(width: 1, height: 2)
let large = Rectangle(width: 5, height: 5)
small.fitsInside(large)  // Returns true
struct Rectangle {
    var width: Int
    var height: Int

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

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

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

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

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

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

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

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

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

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

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

    var area: Int {
        return width * height
    }
}
Extensions

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

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

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

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

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

extension Rectangle {
    func fitsInside(_ other: Rectangle) -> Bool {...}
    var area {...}
}
```
struct NamedArray<Element> {
    var name: String
    var items: [Element]
}
struct NamedArray<Element> {
    var name: String
    var items: [Element]
}
Generic Types

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

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

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

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

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

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

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

let boardGames = NamedArray(name: "Board Games", items: ["Chess", "Go"])
let primes = NamedArray(name: "Primes", items: [1, 3, 5, 7, 13])
```
```swift
class ScoreLogFile {
    var highScores: [Score]
    func record(score: Score, for player: Player) -> Void {
    }
}
```
Values and References

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

player1.score = Score(value: 5)
player2.score = player1.score
Values and References

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

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

class ScoreLogFile {
}
let scoreLog = ScoreLogFile()
```
```swift
struct Score {
    var value: Int
}
player1.score = Score(value: 5)
player2.score = player1.score
player2.score.value = 99

class ScoreLogFile {
    // ...
}
let scoreLog = ScoreLogFile()
player1.scoreLog = scoreLog
player2.scoreLog = scoreLog
player2.logCurrentScore()
```
struct Score { var value: Int }
player1.score = Score(value: 5)
player2.score = player1.score
player2.score.value = 99

class ScoreLogFile {...}
let scoreLog = ScoreLogFile()
player1.scoreLog = scoreLog
player2.scoreLog = scoreLog
player2.logCurrentScore()
Values and References

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

class ScoreLogFile {...}
let scoreLog = ScoreLogFile()
player1.scoreLog = scoreLog
player2.scoreLog = scoreLog
player2.logCurrentScore()
Values and References

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

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

Protocol and Value Oriented Programming in UIKit Apps

Nob Hill  Friday 4:00PM
Subclasses

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

Subclasses Can Add Functionality

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

class FlyingFish: Fish {
    func fly() {
        print("Flying throught the air!")
    }
    // Inherits swim() with no changes.
}
Subclasses Can Override Functionality

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

class ComplainingFish: Fish {
    func swim() {
        print("Grumble grumble grumble...")
        super.swim()
    }
}

...grumble...
Subclasses Can Override Functionality

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

class ComplainingFish: Fish {
    func swim() {
        print("Grumble grumble grumble...")
        super.swim()
    }
}

...grumble...
Subclasses Can Override Functionality

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

class ComplainingFish: Fish {
    func swim() {
        print("Grumble grumble grumble...")
        super.swim()
    }
}

...grumble...
Subclasses Can Override Functionality

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

class ComplainingFish: Fish {
    override func swim() {
        print("Grumble grumble grumble...")
        super.swim()
    }
}
Subclasses Can Override Functionality

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

class ComplainingFish: Fish {
    override func swimmm() {
        print("Grumble grumble grumble...")
        super.swim()
    }
}

...grumble...
Subclass Initializers

class Fish {
    var name: String

    init(name: String) {
        self.name = name
    }
}

let fish = Fish(name: "Herring")
Subclass Initializers

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```swift
protocol Player {
    func takeTurn(on board: Board) { /* fatal error */ }
}

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

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

```swift
protocol Player {
    func takeTurn(on board: Board)
}

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

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

```swift
protocol Player {
    func takeTurn(on board: Board)
}

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

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

protocol Player {
    func takeTurn(on board: Board)
}

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

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

protocol Player {
    func takeTurn(on board: Board)
}

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

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

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

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

```swift
struct HumanPlayer: Player {
    var name: String
    var score: Int
    func takeTurn(on board: Board) {...}
}

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

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

let player = HumanPlayer(name: "Lynn", score: 0)
print(player)
protocol CustomStringConvertible {
    var description: String { get }
}

Protocols
Protocols and Extensions

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

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

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

struct HumanPlayer: Player {...}

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

let player = HumanPlayer(name: "Lynn", score: 0)
print(player)
Protocols and Extensions

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

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

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

Human player Lynn has a score of 0
struct HumanPlayer: Player {...}

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

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

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

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

let textAlignment = Alignment.left
```
Enumerations

code:

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

let textAlignment = Alignment.left
```
Enumerations

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

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

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

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

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

let textAlignment = Alignment.left

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

let textAlignment = Alignment.left

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

```swift
enum Alignment {
    case left, right, center
}

let textAlignment = Alignment.left

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

⚠️ switch must be exhaustive
Enumerations

```swift
enum Alignment {
    case left, right, center
}

let textAlignment = Alignment.left

switch textAlignment {
    case .left:
        print("Lean to the left")
    case .right:
        print("Lean to the right")
    case .center:
        print("Stand up straight")
}
```
Enumerations with Associated Values

```swift
enum Alignment {
    case left(padding: Double), right(padding: Double), center
}

let textAlignment = Alignment.left(padding: 42.7)
```
Enumerations with Associated Values

```swift
enum Alignment {
    case left(padding: Double), right(padding: Double), center
}

let textAlignment = Alignment.left(padding: 42.7)
```
Enumerations with Associated Values

```swift
enum Alignment {
    case left(padding: Double), right(padding: Double), center
}

let textAlignment = Alignment.left(padding: 42.7)

switch textAlignment {
    case .left(let padding):
        print("Left with \(padding) pixels of padding")
    ...
}
```
Enumerations with Associated Values

```swift
enum Alignment {
    case left(padding: Double), right(padding: Double), center
}

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

Left with 42.7 pixels of padding
Enumerations with Raw Values

```swift
enum ServerAddress: String {
    case staging = "https://staging.example.com"
    case production = "https://example.com"
}
```
Enumerations with Raw Values

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

func findPhotos(matchingQuery: String, from server: ServerAddress) {
    let serverAddress = server.rawValue
    ...
}

findPhotos(matchingQuery: "strawberry", from: .staging)
```
// Error Handling
enum SomeError: ErrorProtocol {
    case somethingWentWrong, somethingFailed
}

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

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

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

enum SomeError: ErrorProtocol {
    case somethingWentWrong, somethingFailed
}

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

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

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

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

eenum SomeError: ErrorProtocol {
    case somethingWentWrong, somethingFailed
}

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

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

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

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

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

enum SomeError: ErrorProtocol {
    case somethingWentWrong, somethingFailed
}

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

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

    guard let result = summarize(data) else { throw SomeError.somethingFailed }
    return result
}
enum SomeError: ErrorProtocol {
    case somethingWentWrong, somethingFailed
}

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

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

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

The Swift Programming Language > Error Handling
More Information

https://developer.apple.com/wwdc16/404
<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related Sessions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>What’s New in Swift</td>
<td>Presidio</td>
<td>Tuesday 9:00AM</td>
</tr>
<tr>
<td>Swift API Design Guidelines</td>
<td>Presidio</td>
<td>Tuesday 10:00AM</td>
</tr>
<tr>
<td>What’s New in Foundation for Swift</td>
<td>Mission</td>
<td>Tuesday 4:00PM</td>
</tr>
<tr>
<td>Introducing Swift Playgrounds</td>
<td>Mission</td>
<td>Wednesday 11:00AM</td>
</tr>
<tr>
<td>Going Server-Side with Swift Open Source</td>
<td>Mission</td>
<td>Friday 9:00AM</td>
</tr>
<tr>
<td>Protocol and Value Oriented Programming in UIKit Apps</td>
<td>Nob Hill</td>
<td>Friday 4:00PM</td>
</tr>
<tr>
<td>Event</td>
<td>Location</td>
<td>Time</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Swift Get-Together</td>
<td>Graphics, Games, and Media Lab A</td>
<td>Wednesday 6:15 PM</td>
</tr>
<tr>
<td>Swift Open Hours</td>
<td>Developer Tools Lab A</td>
<td>Tuesday 12:00PM</td>
</tr>
<tr>
<td>Swift Open Hours</td>
<td>Developer Tools Lab A</td>
<td>Wed–Fri 9:00AM</td>
</tr>
</tbody>
</table>