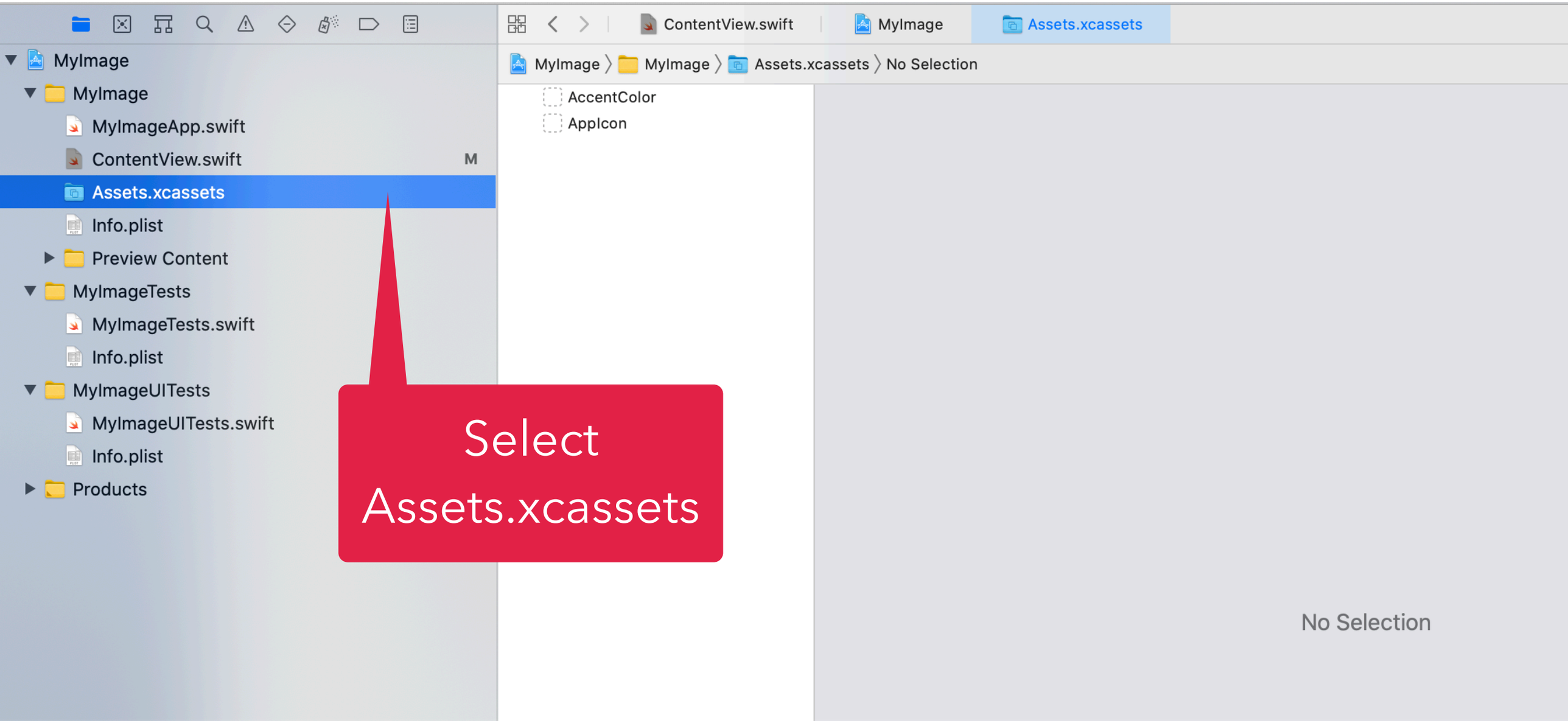
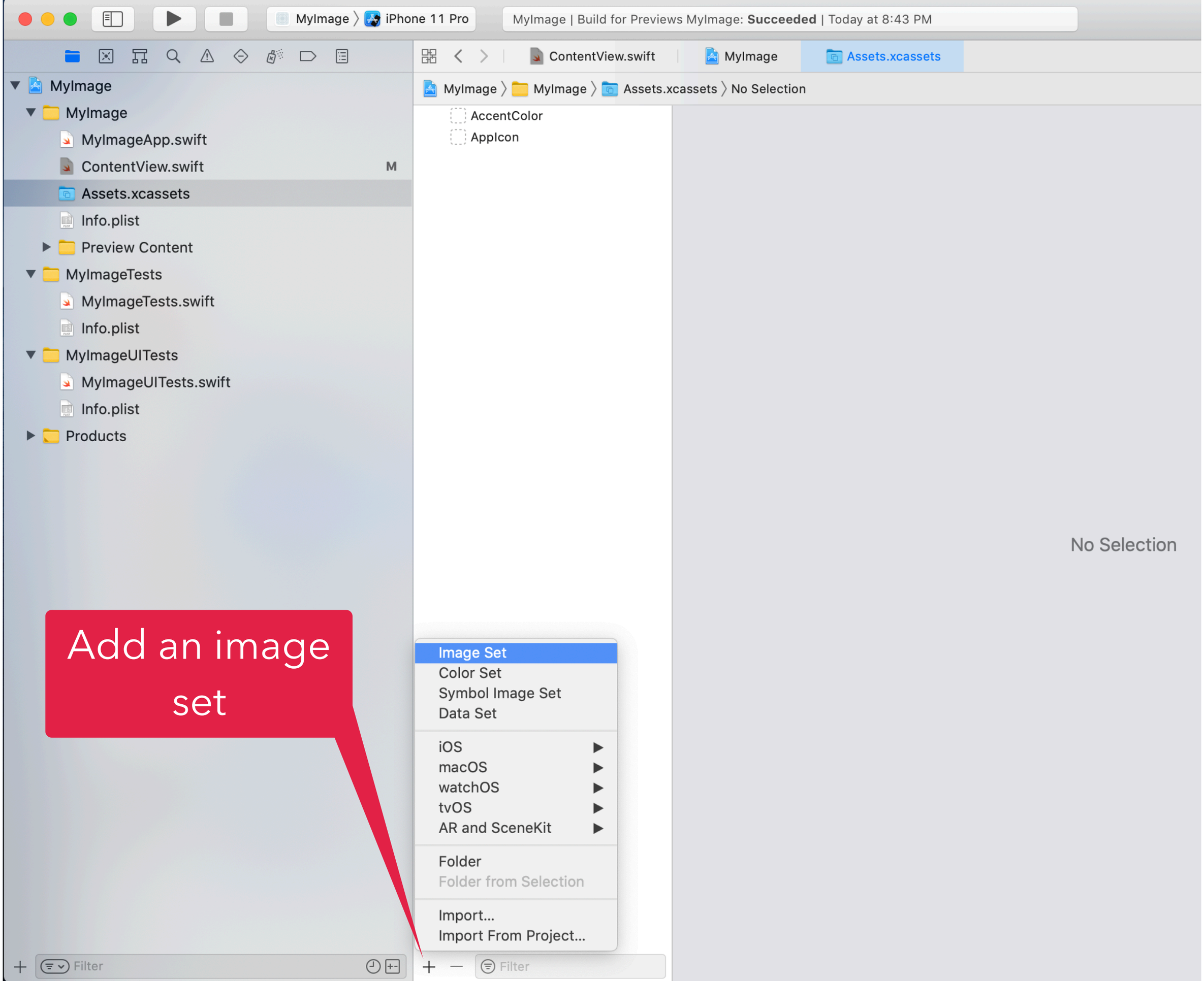
Guided Tour Playground SwiftUI Image

A Swift Tour

Tradition suggests that the first program in a new language should print the words “Hello, world!” on the screen. In Swift, this can be done in a single line:

Download Image


ContentView.swift
MyImage
Assets.xcassets
MyImage 〉 MyImage 〉 Assets.xcassets 〉 No Selection
MyImage
MyImage
MyImageApp.swift
ContentView.swift
M
Assets.xcassets
Info.plist
Preview Content
MyImageTests
MyImageTests.swift
Info.plist
MyImageUITests
MyImageUITests.swift
Info.plist
Products
AccentColor
AppIcon
Select Assets.xcassets
No Selection


MyImage 〉 iPhone 11 Pro
MyImage | Build for Previews MyImage: Succeeded | Today at 8:43 PM
ContentView.swift
MyImage
Assets.xcassets
MyImage 〉 MyImage 〉 Assets.xcassets 〉 No Selection
MyImage
MyImage
MyImageApp.swift
ContentView.swift M
Assets.xcassets
Info.plist
Preview Content
MyImageTests
MyImageTests.swift
Info.plist
MyImageUITests
MyImageUITests.swift
Info.plist
Products
AccentColor
AppIcon
No Selection
Add an image set
Image Set
Color Set
Symbol Image Set
Data Set
iOS
macOS
watchOS
tvOS
AR and SceneKit
Folder
Folder from Selection
Import...
Import From Project...
Filter
Filter

hone 11 Pro | MyImage | Build for Previews MyImage: **Succeeded** | Today at 8:43 PM

ContentView.swift | MyImage | Assets.xcassets

MyImage 〉 MyImage 〉 Assets.xcassets 〉 fall-leaves

- AccentColor
- AppIcon
- fall-leaves

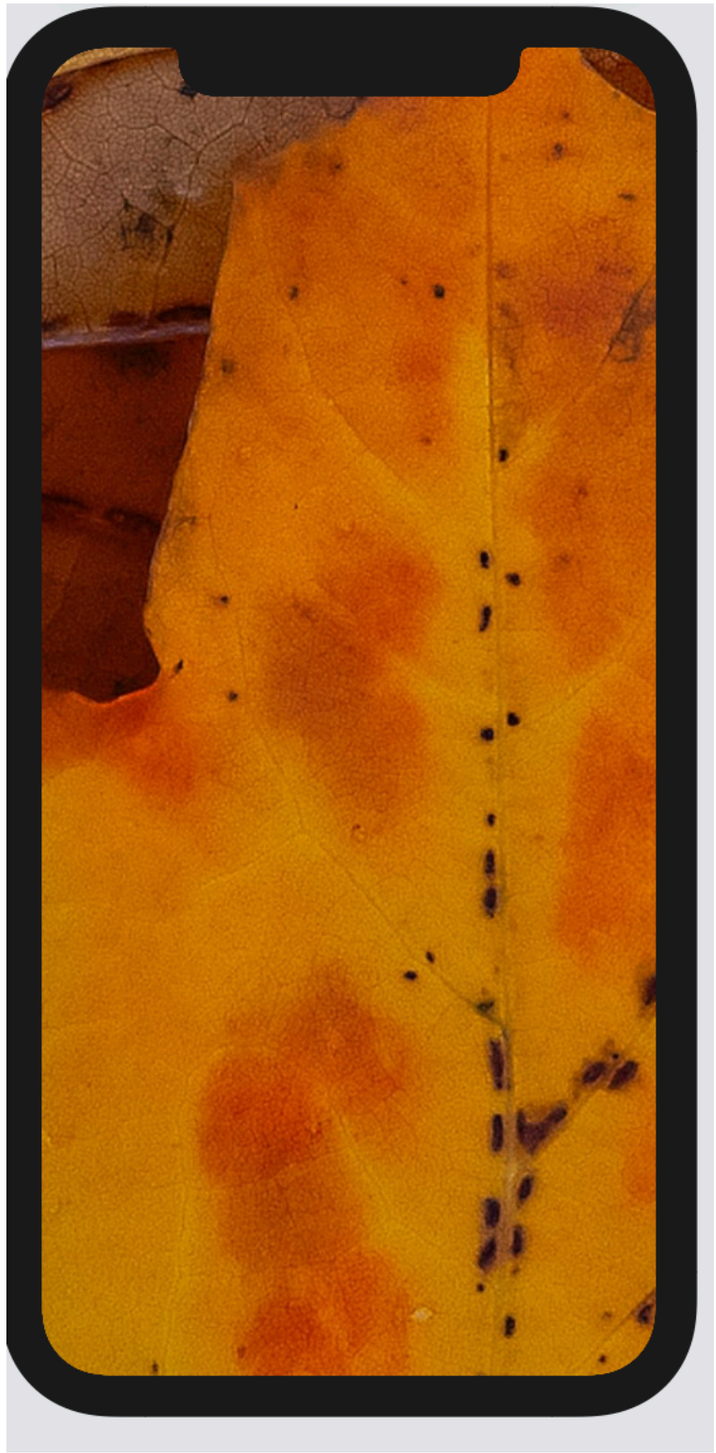
Key in Image name

fall-leaves

1x	2x	3x
	Universal	

Drag image to here

```
struct ContentView: View {
    var body: some View {
        Image("fall-leaves")
    }
}
```



```swift
struct ContentView: View {
    var body: some View {
        Image("fall-leaves")
            .resizable()
    }
}
```

```swift
struct ContentView: View {
    var body: some View {
        Image("fall-leaves")
            .resizable()
            .scaledToFit()
    }
}
```

1

```
struct ContentView: View {
    var body: some View {
        guard let img = UIImage(named: "fall-
leaves") else{
            fatalError("Unable to load!")
        }
        return Image(uiImage: img)
            .resizable()
            .scaledToFit()
    }
}
```

使用guard let錯誤處理

Assignment

```swift
var myVariable = 42      // variable (can't be nil)
let pi = 3.1415926        // constant
let (x, y) = (10, 20)    // x = 10, y = 20
let explicitDouble: Double = 1_000.000_1    // 1,000.0001
let piText = "Pi = \(pi)"     // String interpolation
var optionalString: String? = "optional"    // Can be nil
optionalString = nil
```

Assign nil to an optional string
允許未定義

Type of optional string
String?

```swift
import UIKit

var str = "Hello, playground"
var myVariable = 42      // variable (can't be nil)
let pi = 3.1415926        // constant
let (x, y) = (10, 20)    // x = 10, y = 20
let explicitDouble: Double = 1_000.000_1     // 1,000.0001
let piText = "Pi = \(pi)"     // String interpolation
var optionalString: String? = "optional"     // Can be nil
optionalString = nil
```

```
"Hello, playground"
42
3.1415926
1000.0001
"Pi = 3.1415926"
"optional"
nil
```

An array of String

```
//Array
var shoppingList = ["catfish", "water", "lemons"]
shoppingList[1] = "bottle of water"      // update
print(shoppingList.count)   // size of array (3)
shoppingList.append("eggs")
shoppingList += ["Milk"]
```

Append an item to it

shopList is an object
Its size is shoppingList.count

```
12 var shoppingList = ["catfish", "water", "lemons"]
13 shoppingList[1] = "bottle of water"      // update
14 print(shoppingList.count)    // size of array (3)
15 shoppingList.append("eggs")
16 shoppingList += ["Milk"]
17
```

```
["catfish", "wat...
"bottle of water"
"3\n"
["catfish", "bott...
["catfish", "bott...
```

An array of integers

```
// Array slicing
var fibList = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 5]
print(fibList[4..<6])
print(fibList[0..<(fibList.endIndex-1)]) // all except last
item
print(Array(fibList[0..<4]))
```

// [3, 5]. Note: the end range value is exclusive

// Subscripting returns the Slice type, instead of the Array type.
// You may need to cast it to Array in order to satisfy the type checker

```
19 var fibList = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 5]                [0, 1, 1, 2, 3, 5,...
20 print(fibList[4..<6])                                             "[3, 5]\n"
21 print(fibList[0..<(fibList.endIndex-1)]) // all except            "[0, 1, 1, 2, 3, 5...
       last item
   print(Array(fibList[0..<4]))                                      "[0, 1, 1, 2]\n"
23                                                                   [{lowerBound 1...
```

```
// Variants of creating an array. All three are equivalent.
var emptyArray1 = [String]()
var emptyArray2: [String] = []
var emptyArray3: [String] = [String]()
```

```
32  let anArray = Array(1...8)
33  print(anArray)
34  var ages = [ 21, 55, 19, 47, 22, 37,
        88, 71 ]
35  ages.count
36  ages.last
37  ages[3]
38  ages.append(99)
39  ages.insert(44, at:4)
40  ages.sort()
41  ages.reverse()
42  ages.shuffle()
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
"[1, 2, 3, 4, 5, 6, 7, 8]\n"
[21, 55, 19, 47, 22, 37, 88, 71]
8
71
47
[21, 55, 19, 47, 22, 37, 88, 71, 99]
[21, 55, 19, 47, 44, 22, 37, 88, 71, 99]
[19, 21, 22, 37, 44, 47, 55, 71, 88, 99]
[99, 88, 71, 55, 47, 44, 37, 22, 21, 19]
[47, 88, 71, 99, 44, 37, 19, 22, 55, 21]
```

Playground_SwiftUI

```
//: A UIKit based Playground for presenting

import UIKit
import PlaygroundSupport
import SwiftUI

struct ContentView: View {
    var body: some View {
        Text("SwiftUI in a Playground!")
    }
}

let viewController =
    UIHostingController(rootView: ContentView())
// Present the view controller in the Live View
window
PlaygroundPage.current.liveView = viewController
```

```swift
import UIKit
import PlaygroundSupport
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack{
            Text("SwiftUI in a Playground!")
            Spacer()
            Text("SwiftUI in a Playground!")
        }
    }
}

let viewController =
    UIHostingController(rootView: ContentView())
// Present the view controller in the Live View window
PlaygroundPage.current.liveView = viewController
```

SwiftUI in a Playground!

SwiftUI in a Playground!

MyPlayground_swiftUI

```swift
import UIKit
import PlaygroundSupport
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack{
            Text("SwiftUI in a Playground!")
            Spacer()
            Text("SwiftUI in a Playground!")
            Spacer()
            Button("Click Me"){
                print("Hello World")
            }

        }
    }
}

let viewController =
    UIHostingController(rootView: ContentView())
// Present the view controller in the Live View window
PlaygroundPage.current.liveView = viewController
```

SwiftUI.VSt...
SwiftUI.Tup...
SwiftUI.Text
SwiftUI.Sp...
SwiftUI.Text
SwiftUI.Sp...
SwiftUI.But...
()
SwiftUI.Tup...
SwiftUI.VSt...
<_TtGC7S...

SwiftUI in a Playground!
SwiftUI in a Playground!
Click Me

Hello World

A Swift Tour

Tradition suggests that the first program in a new language should print the words “Hello, world!” on the screen. In Swift, this can be done in a single line:

Guided Tour

Simple Values

Guided Tour
Simple Values

GuidedTour | Build GuidedTour (Playground): **Succeeded** | Today at 5:29 PM

Enumerations and Structures | Objects and Classes.xcplaygroundpage | *Simple Values.xcplaygroundpage* | Functions and Closures

GuidedTour 〉 Simple Values

A Swift Tour

Tradition suggests that the first program in a new language should print the words “Hello, world!” on the screen. In Swift, this can be done in a single line:

```
5 print("Hello, world!")
6 print("Hello, HHualien!")
```

```
5  print("Hello, world!")
6
```

```
15  var myVariable = 42
16  myVariable = 50
17  let myConstant = 42
18
```

變數可以改變內容

常數不可以改變內容

代表整數

```
23  let implicitInteger = 70
24  let implicitDouble = 70.0
25  let explicitDouble: Double = 70
26
```

明顯宣告為
double實數

```
32  let label = "The width is "
33  let width = 94
34  let widthLabel = label + String(width)
35
```

將整數轉換為字串

```swift
let apples = 3
let oranges = 5
let appleSummary = "I have \(apples) apples."
let fruitSummary = "I have \(apples + oranges) pieces of fruit."

```

將整數轉為字串，內插在列印字串中

運算後，在將整數轉為字串，內插在列印字串中

將常數quotation設定為
一段跨行的文字

```
51  let quotation = """
52  I said "I have \(apples) apples."
53  And then I said "I have \(apples +
        oranges) pieces of fruit."
54  """
55
```

將整數轉為字串，內
插在字串中

將常數quotation設定為
一段跨行的文字

```
58  var shoppingList = ["catfish", "water",
        "tulips"]
59  shoppingList[1] = "bottle of water"
60
61  var occupations = [
62      "Malcolm": "Captain",
63      "Kaylee": "Mechanic",
64    ]
65  occupations["Jayne"] = "Public Relations"
66
```

改變字串陣列中的第二個元素

字典陣列
Key:value

附加字串到陣列
shoppingList中

```
69  shoppingList.append("blue paint")
70  print(shoppingList)
71
```

```swift
var oc = [String: Float]()
oc["key"] = 1.0
let va = oc["key"]!
print(va)
```

空的字串陣列

```swift
74 let emptyArray = [String]()
75 let emptyDictionary = [String: Float]()
76
```

空的字典陣列

```
79  shoppingList = []
80  occupations = [:]
81
82
```

空的字典陣列

```
5  let individualScores = [75, 43, 103, 87, 12]
6  var teamScore = 0
7  for score in individualScores {
8      if score > 50 {
9          teamScore += 3
10     } else {
11         teamScore += 1
12     }
13 }
14 print(teamScore)
15
```

列舉陣列individualScores中的元素，依序代入score，執行迴圈

宣告為optional字串
可以被指定內容，但不是必須被指定

```
20 var optionalString: String? = "Hello"
21 print(optionalString == nil)
22
23 var optionalName: String? = "John Appleseed"
24 var greeting = "Hello!"
25 if let name = optionalName {
26     greeting = "Hello, \(name)"
27 }
28
```

是否無定義

當let指令成功設定name，if條件才會成立，代表optionalName有定義

```swift
let nickname: String? = nil
let fullName: String = "John Appleseed"
let informalGreeting = "Hi \(nickname ?? fullName)"

```

如果nickname沒有被指定，以fullname取代

2.1

根據變數vegetable內容不同的狀況，執行不同指令

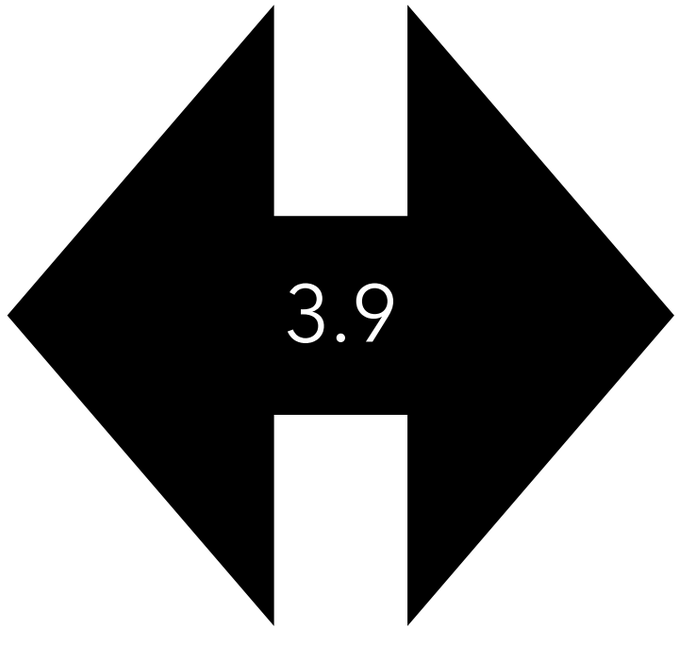
```
43  switch vegetable {
44      case "celery":
45          print("Add some raisins and make ants
                on a log.")
46      case "cucumber", "watercress":
47          print("That would make a good tea
                sandwich.")
48      case let x where x.hasSuffix("pepper"):
49          print("Is it a spicy \(x)?")
50      default:
51          print("Everything tastes good in
                soup.")
52  }
```

內定的其他狀況

vegetable = "cucumber"

vegetable ="green pepper"

vegetable ="tomato"



```swift
let interestingNumbers = [
    "Prime": [2, 3, 5, 7, 11, 13],
    "Fibonacci": [1, 1, 2, 3, 5, 8],
    "Square": [1, 4, 9, 16, 25],
]
var largest = 0
for (_, numbers) in interestingNumbers {
    for number in numbers {
        if number > largest {
            largest = number
        }
    }
}
print(largest)
```

```
83  var n = 2
84  while n < 100 {
85        n *= 2
86  }
87  print(n)
88
89  var m = 2
90  repeat {
91        m *= 2
92  } while m < 100
93  print(m)
94
```

```
97  var total = 0
98  for i in 0..<4 {
99      total += i
100 }
101 print(total)
102
```

Functions and Closures

Use `func` to declare a function. Call a function by following its name with a list of arguments in parentheses. Use –> to separate the parameter names and types from the function's return type.

```
func greet(person: String, day: String) -> String {
    return "Hello \(person), today is \(day)."
}
greet(person: "Bob", day: "Tuesday")
```

By default, functions use their parameter names as labels for their arguments. Write a custom argument label before the parameter name, or write _ to use no argument label.

```
func greet(_ person: String, on day: String) -> String {
    return "Hello \(person), today is \(day)."
}
greet("John", on: "Wednesday")
```

Use a tuple to make a compound value — for example, to return multiple values from a function. The elements of a tuple can be referred to either by name or by number.

```
func calculateStatistics(scores: [Int]) -> (min: Int, max: Int, sum: Int) {
    var min = scores[0]
    var max = scores[0]
    var sum = 0

    for score in scores {
        if score > max {
            max = score
        } else if score < min {
            min = score
        }
        sum += score
    }

    return (min, max, sum)
}
let statistics = calculateStatistics(scores: [5, 3, 100, 3, 9])
print(statistics.sum)
// Prints "120".
print(statistics.2)
// Prints "120".
```

Functions can be nested. Nested functions have access to variables that were declared in the outer function. You can use nested functions to organize the code in a function that's long or complex.

```
func returnFifteen() -> Int {
    var y = 10
    func add() {
        y += 5
    }
    add()
    return y
}
returnFifteen()
```

Functions are a first-class type. This means that a function can return another function as its value.

```
func makeIncrementer() -> ((Int) -> Int) {
    func addOne(number: Int) -> Int {
        return 1 + number
    }
    return addOne
}
var increment = makeIncrementer()
increment(7)
```

A function can take another function as one of its arguments.

```
func hasAnyMatches(list: [Int], condition: (Int) -> Bool) -> Bool {
    for item in list {
        if condition(item) {
            return true
        }
    }
    return false
}
func lessThanTen(number: Int) -> Bool {
    return number < 10
}
var numbers = [20, 19, 7, 12]
hasAnyMatches(list: numbers, condition: lessThanTen)
```

Functions are actually a special case of closures: blocks of code that can be called later. The code in a closure has access to things like variables and functions that were available in the scope where the closure was created, even if the closure is in a different scope when it's executed — you saw an example of this already with nested functions. You can write a closure without a name by surrounding code with braces (`{}`). Use `in` to separate the arguments and return type from the body.

```
numbers.map({ (number: Int) -> Int in
    let result = 3 * number
    return result
})
```

You have several options for writing closures more concisely. When a closure's type is already known, such as the callback for a delegate, you can omit the type of its parameters, its return type, or both. Single statement closures implicitly return the value of their only statement.

```
let mappedNumbers = numbers.map({ number in 3 * number })
print(mappedNumbers)
// Prints "[60, 57, 21, 36]".
```

You can refer to parameters by number instead of by name — this approach is especially useful in very short closures. A closure passed as the last argument to a function can appear immediately after the parentheses. When a closure is the only argument to a function, you can omit the parentheses entirely.

```
let sortedNumbers = numbers.sorted { $0 > $1 }
print(sortedNumbers)
// Prints "[20, 19, 12, 7]".
```