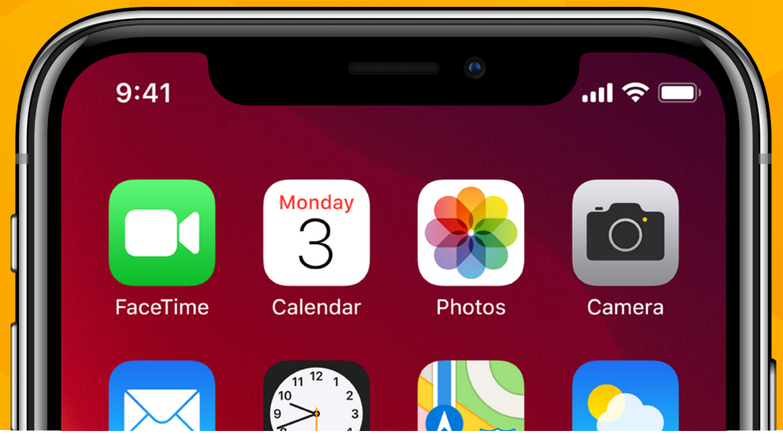


iOS Application Development

Lecture 2: Introduction to Swift & Seminar Topics

Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

Winter Semester ’22/’23

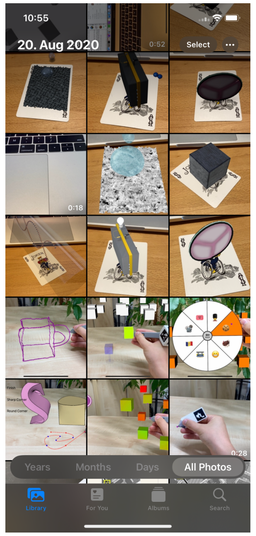
hci.rwth-aachen.de/ios

RWTH AACHEN UNIVERSITY

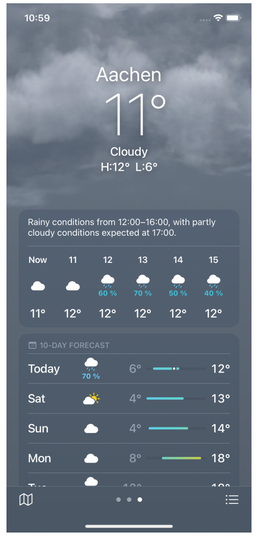
Recap

- Mobile device characteristics
  - Context
  - Screen size
  - One app at a time
- Application Styles
  - Productivity
  - Utility
  - Immersive

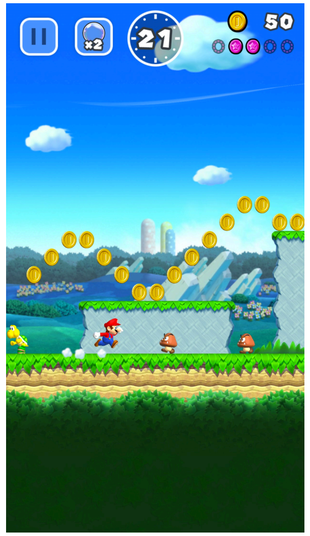
VS.



Photos



Weather



Super Mario Run

Swift

History

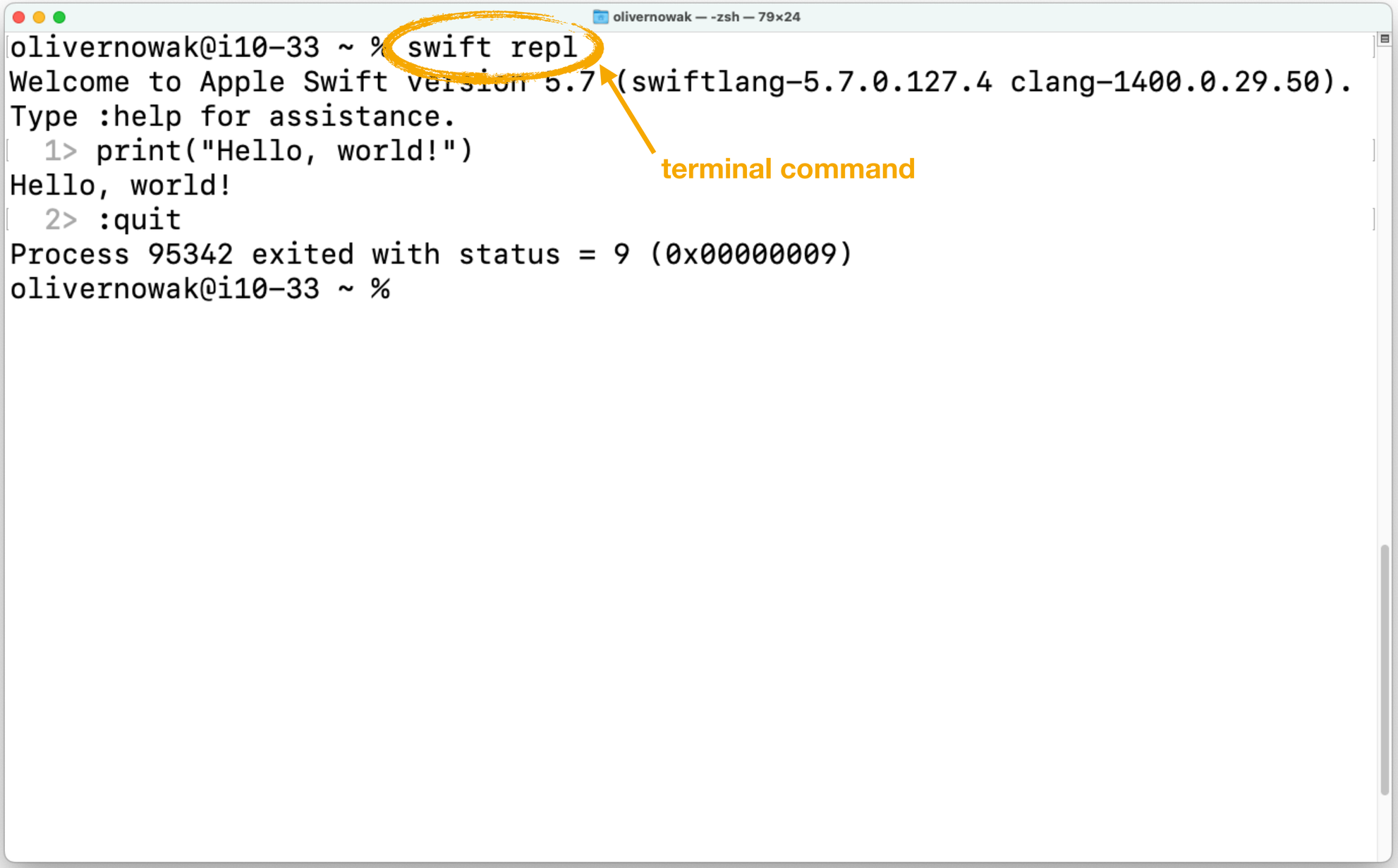
- Introduced at WWDC 2014
- Influenced by C and Objective-C
- But designed to be easier to learn and not depend on older languages
- “Safe, fast, and expressive”
- Open source since 2015
- Replacing Objective-C throughout iOS & macOS

Characteristics

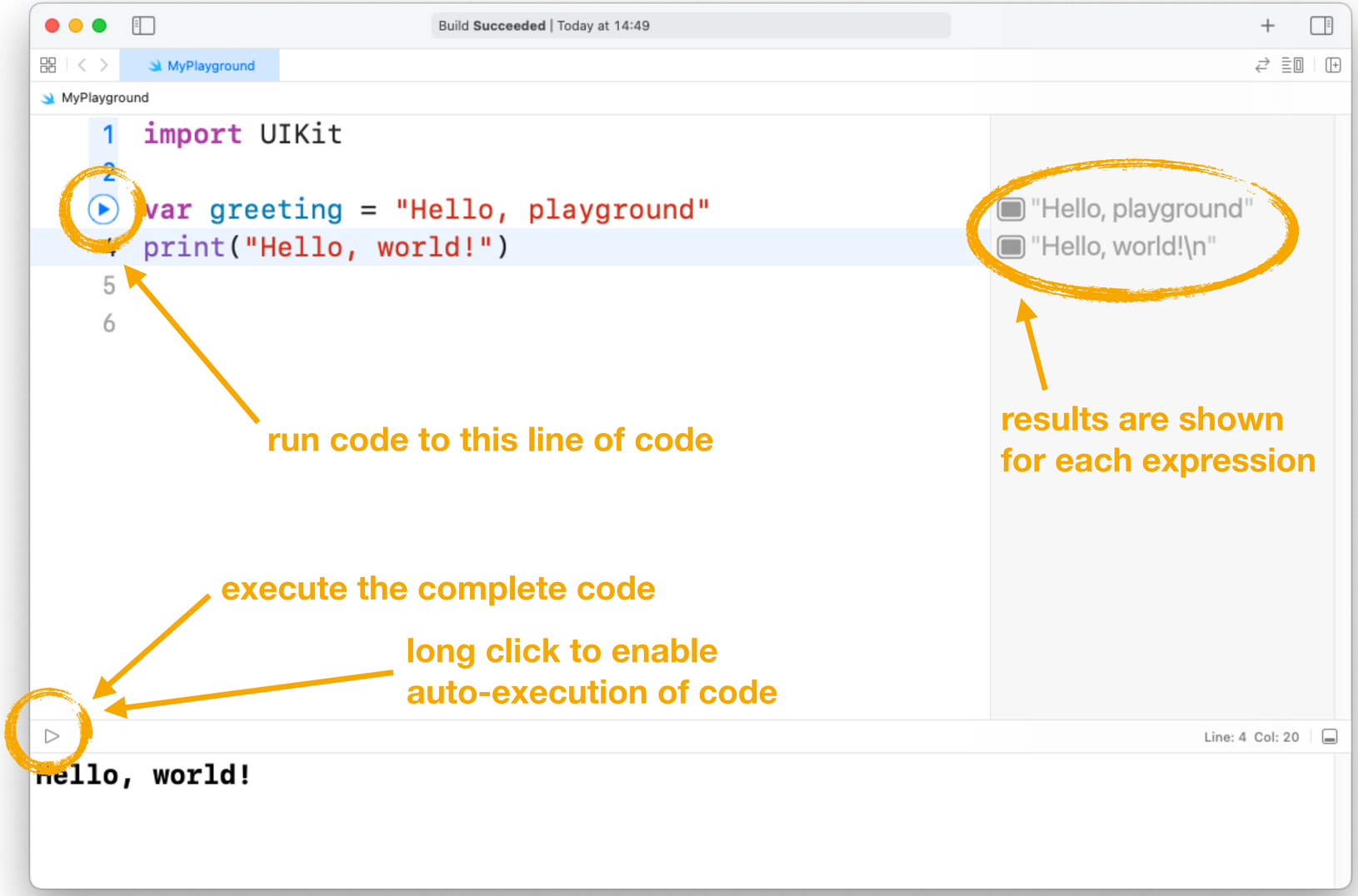
- Clean syntax
- Type safety
- Type inference
- Automatic Reference Counting (ARC)
- Optionals

Characteristics

- Tuples and multiple return values
- Generics
- Fast and concise iteration over collections
- Structs that support methods, extensions, and protocols
- Map, filter, reduce, and other functional programming patterns
- Powerful error handling



Playground



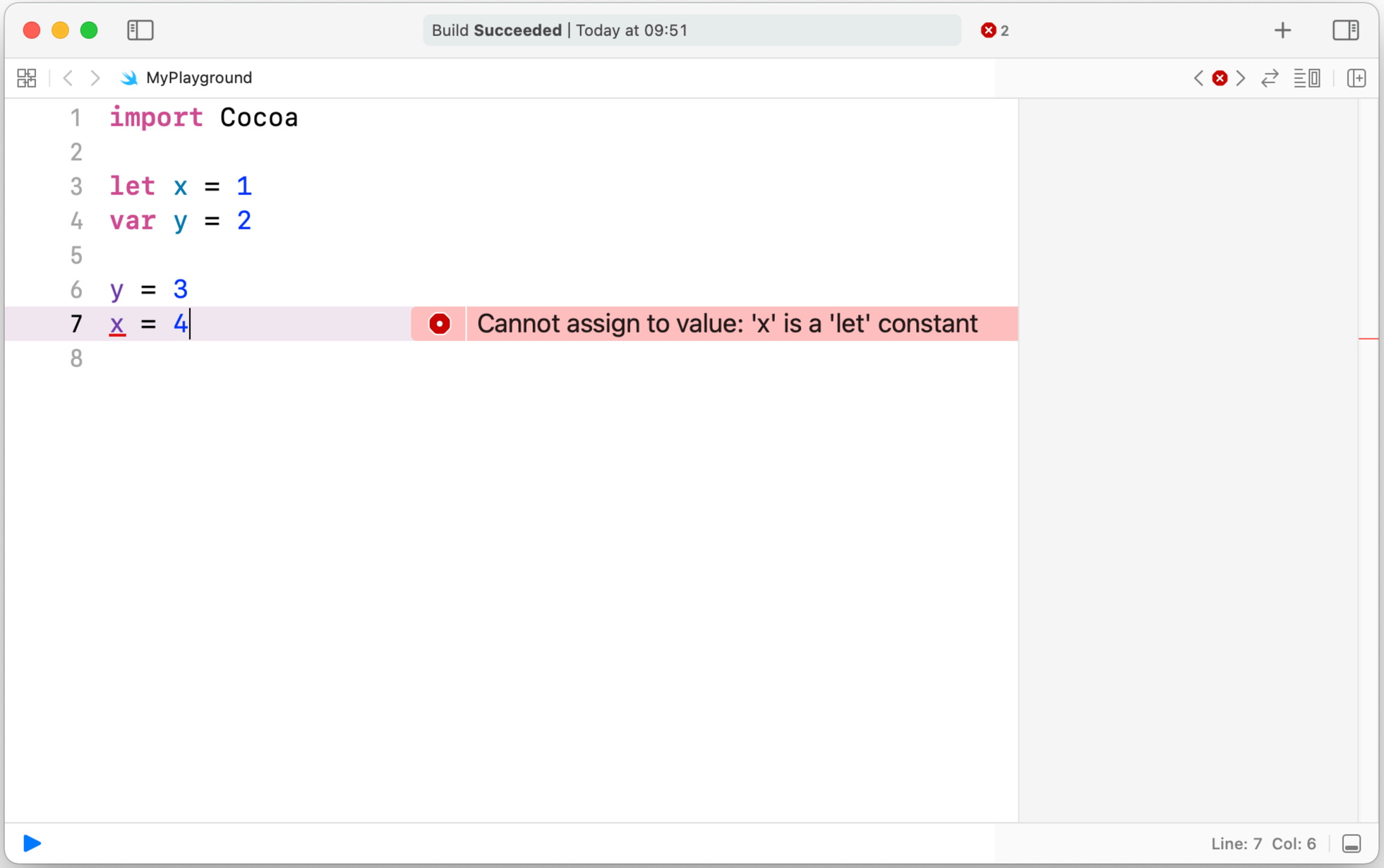
Variables and Constants

- Variables are declared with `var`

```
var x = 100
```

- Constants are declared with `let`

```
let pi = 3.14
```



Type Inference

- Swift automatically chooses the adequate data type for a variable/constant

```
var x = 100
    x = 99.5  // Error! x is of type Integer

var x = 99.5
    x = 100   // Correct (x is 100.0, type: Double)
```

- You can also explicitly specify the type

```
var aString : String
```

Data Types & Type Inference

```
var x1 = 100                // x1 is Int

var x2 = 0.5                // x2 is Double

var x3 = x1 + x2            // Error! Can’t add Int and Double

var x3 = Double(x1) + x2    // Works! explicit type casting to Double

var x4 = 0.5 + 100          // Works! compiler adds before setting the data type

print("x4 = \(x4)")         // Output: x4 = 100.5

var 😂 = "LOL"              // 😂 is String
```

Optionals

- By default, variables and constants cannot be nil
- Optionals are variables that can also be nil

```
var i:Int? = 3

i = nil
```

- Normal variables and Optionals are **incompatible** to each other

```
var number = Int("42")      // Type of number is Int? because Int() returns Int?

print(number + 3)           // Error! Int? != Int

var i:Int = number          // Error! Int? != Int
```

Tuples

- Tuples can contain multiple elements of different types

```
var tuple = (42, 23.0, “hello”, true)

var (a,b,c,d) = tuple  // a = 42, b = 23.0, etc.

print(tuple.2)         // Prints “hello” (index starts at 0)
```

Control Flow

- if/else

```
var x=3

if x<0 {
    print("x is negative")
} else if x==0 {
    print ("x is zero”)
} else {
    print("x is positive")
}
```

- Ternary Operator

```
var largest: Int
let a = 15
let b = 4

if a > b {
     largest = a
}
else {
     largest = b
}

// Can be written as
largest = a > b ? a : b
```

Control Flow

- switch

```
let pt = (0.0, 0.0)

switch pt {
case (0,0):
     print("Origin.")
case (_,0):
     print ("On x-axis.")
case (0,_):
     print ("On y-axis.")
default:
     print ("Elsewhere.")
}
```

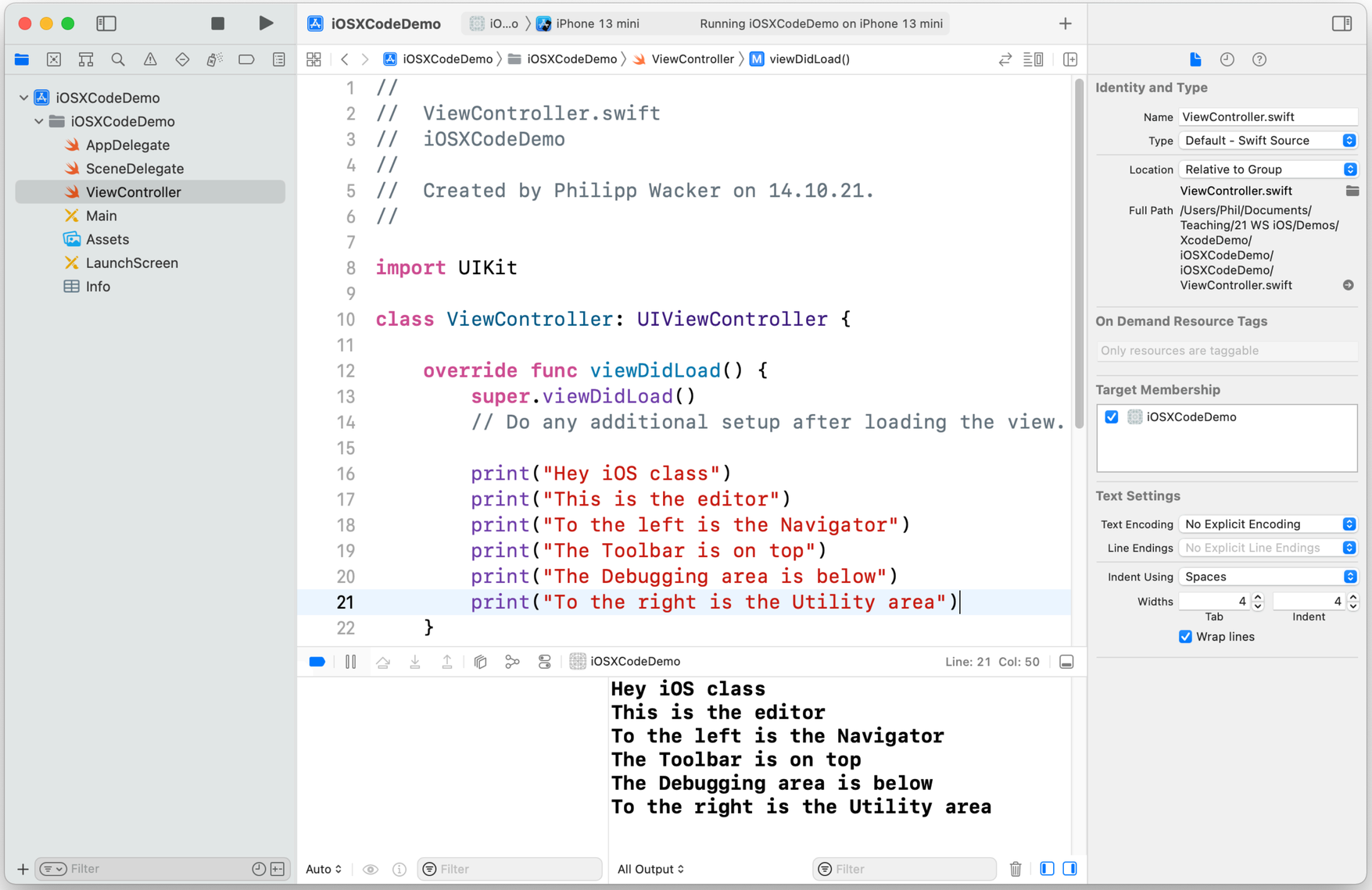
```
let distance = 5

switch distance {
case 0...9:
     print("You are close.”)
case 10...500:
     print("Take the train.”)
default:
     print("Too far away.")
}
```

- No fallthrough

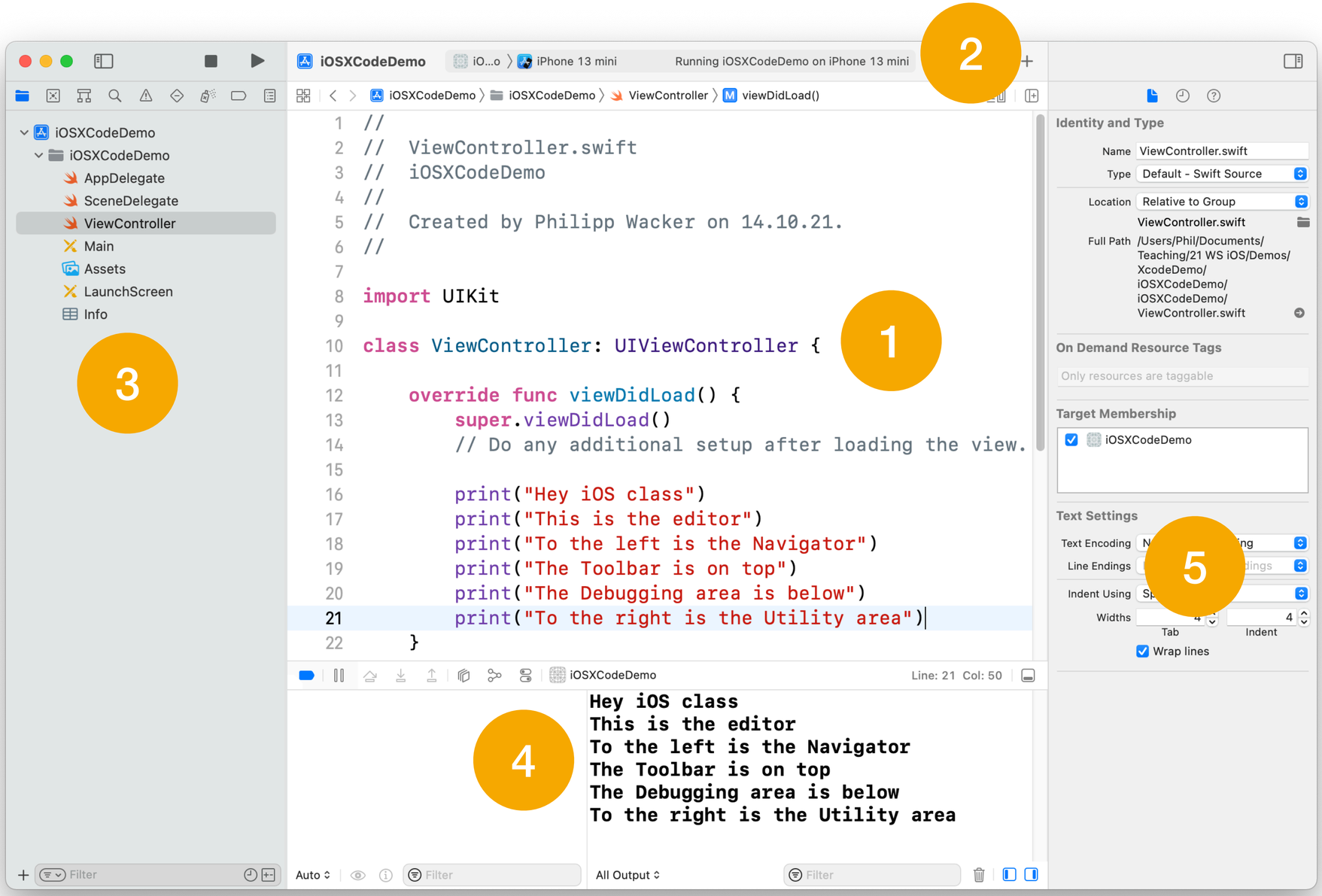
Demo: Xcode Development Environment

Xcode



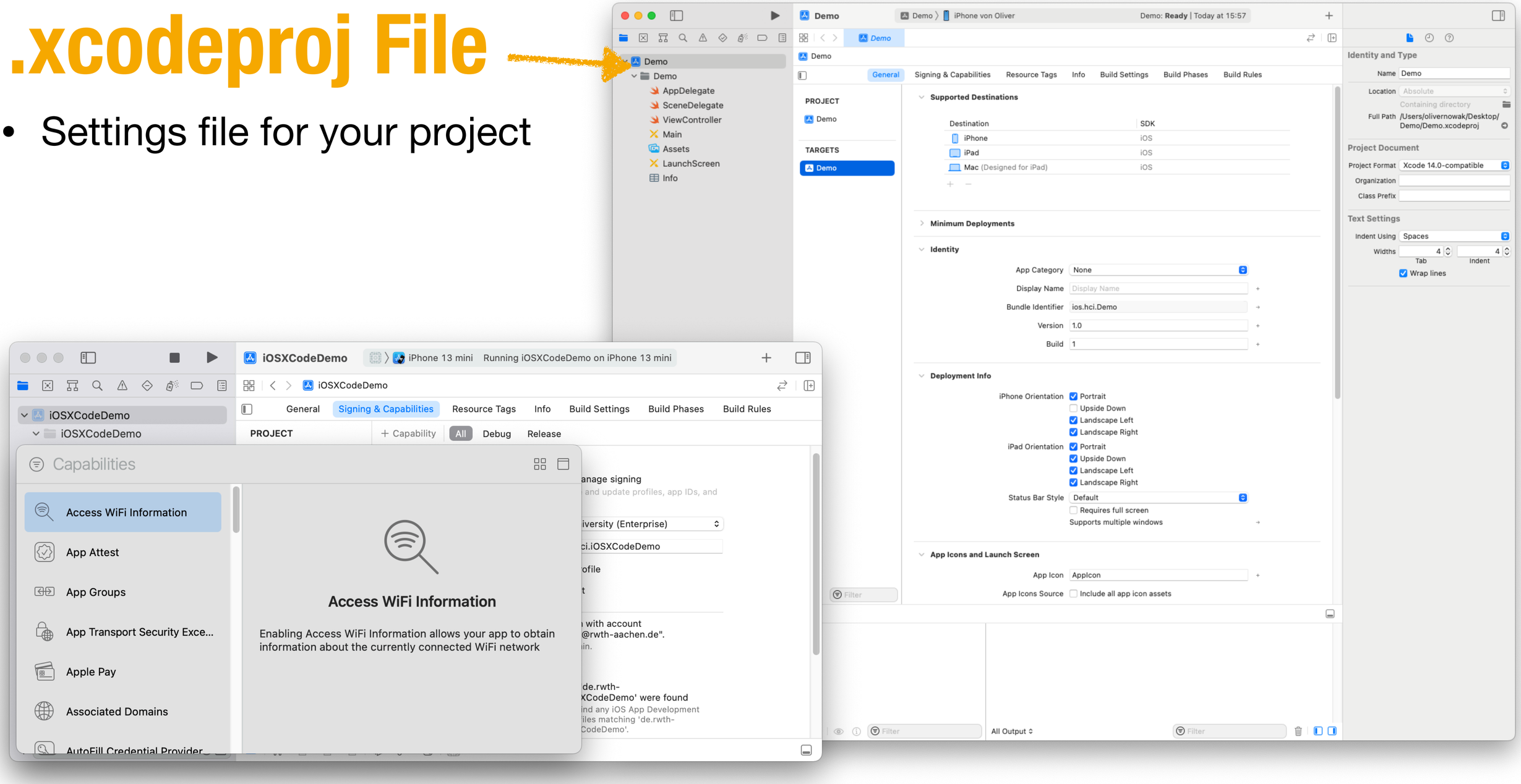
Xcode

- 5 areas
  1. Editor
  2. Toolbar
  3. Navigator
  4. Debugging
  5. Utility



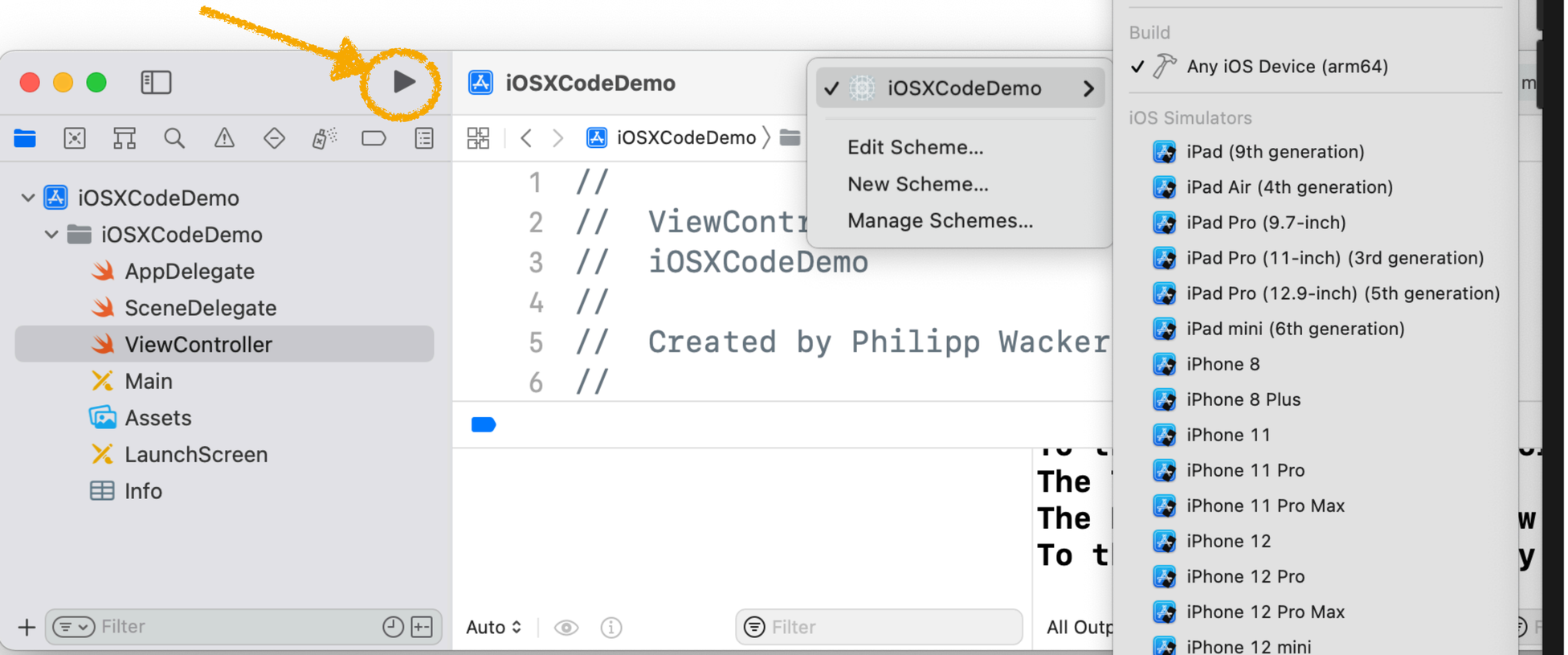
.xcodeproj File

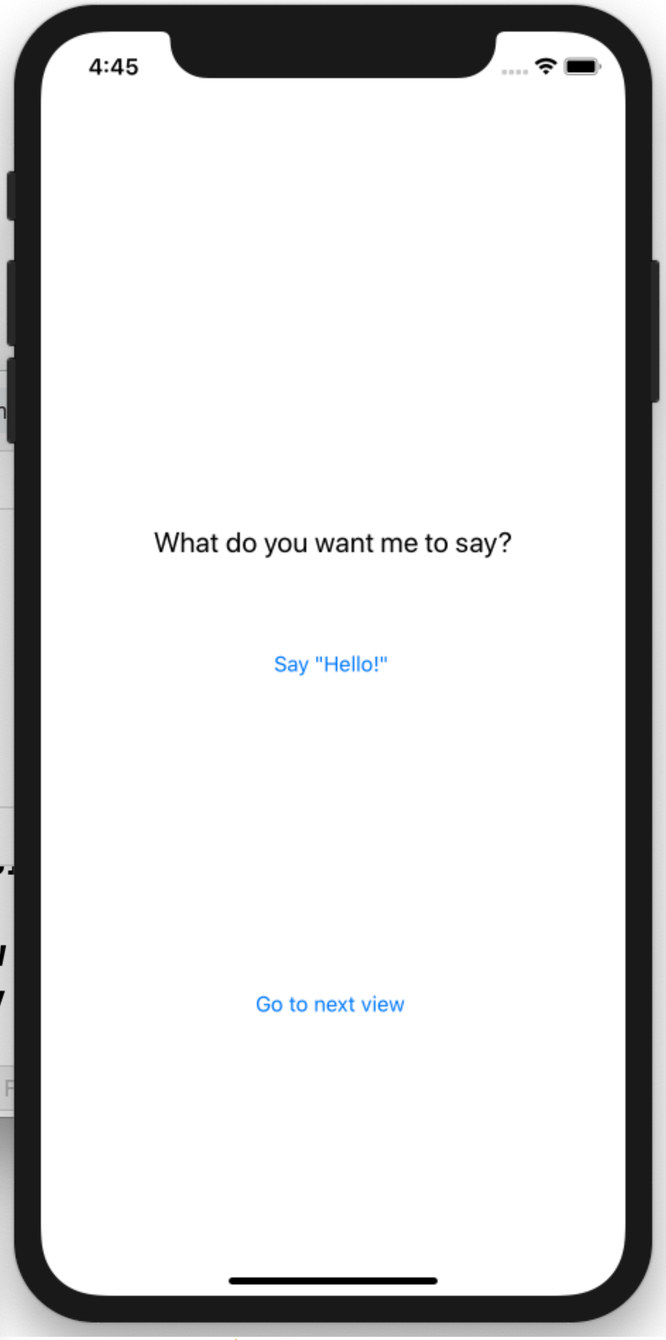
- Settings file for your project



Building/Running

- Run on selected device

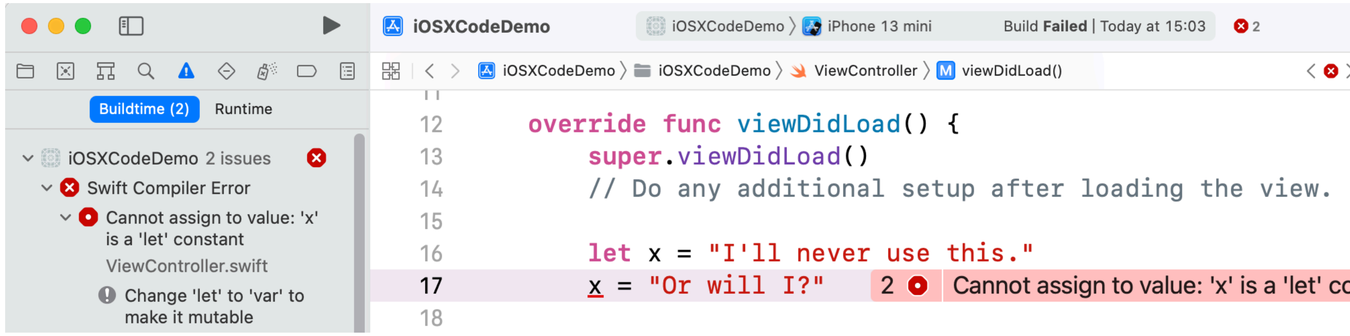




Warnings & Errors

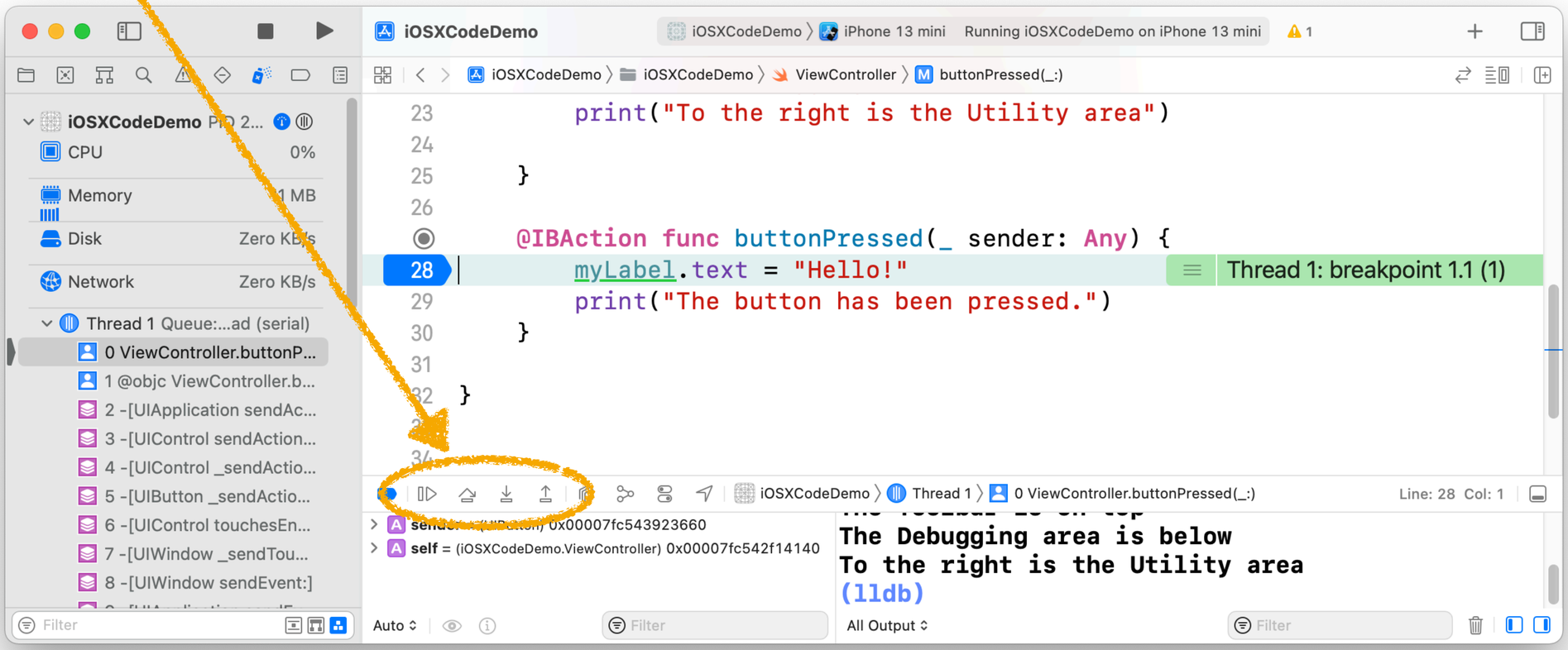
- **Warnings** don’t prevent your app from compiling & running
  - Code that never gets executed
  - Variable that does not change
  - Deprecated code

- **Errors** prevent your app from building
  - Invalid code (typo, variable declaration, function calling)
  - Xcode often provides suggestions & fixes



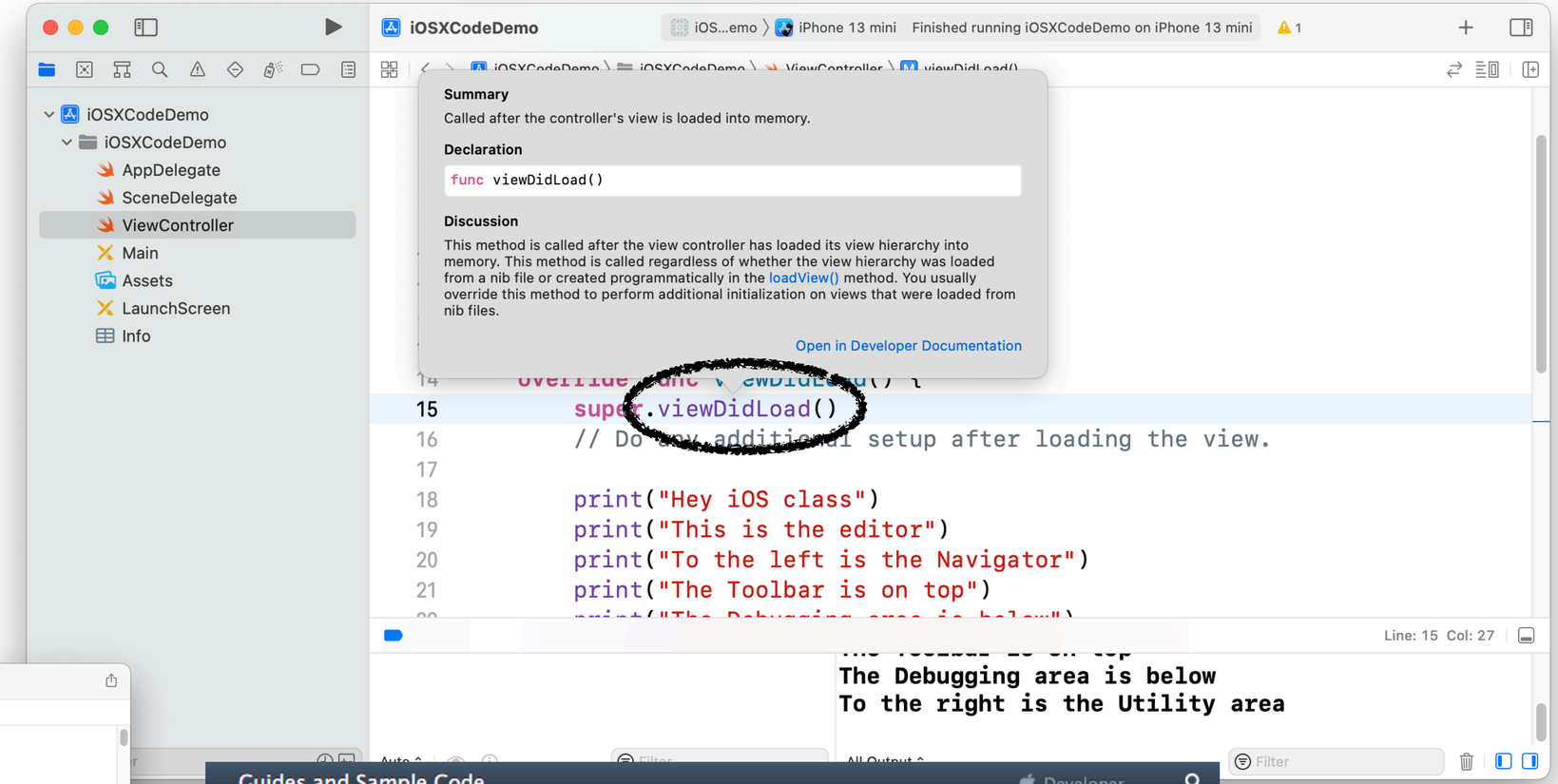
Debugging

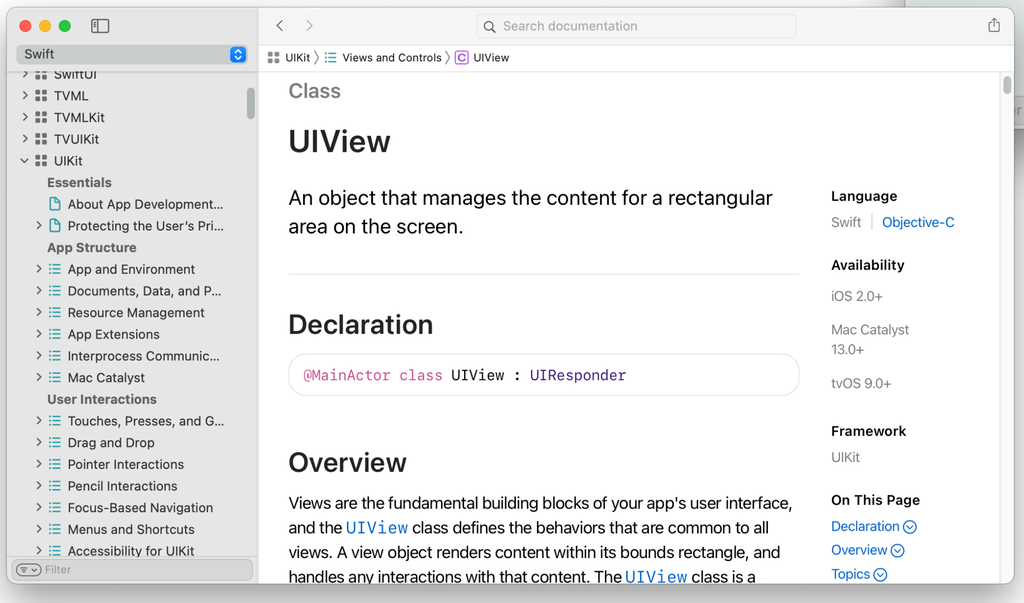
- Set breakpoints for execution on simulator and device
- Continue, Step over, Step into, Step out

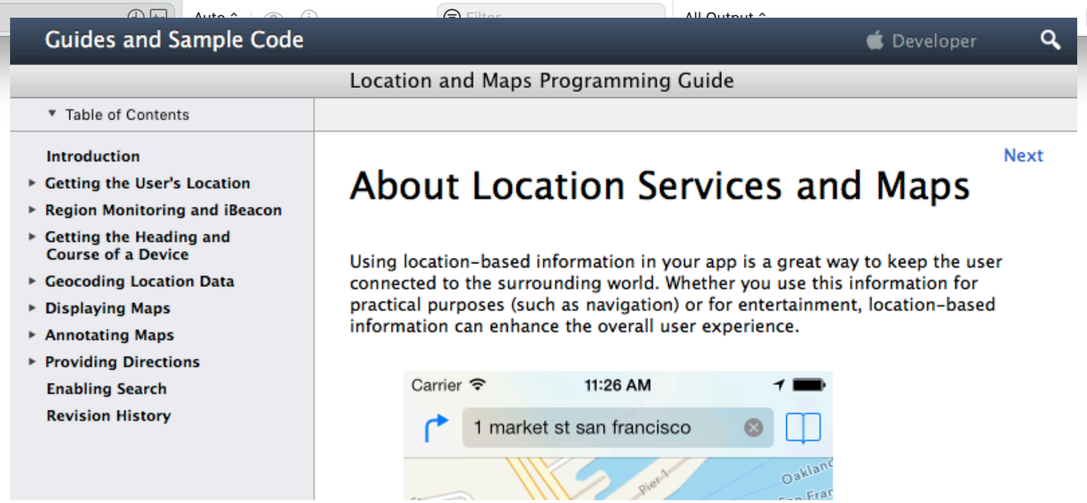


Documentation

- Quick Help (Option+Click)
- Documentation Browser
- Programming Guides

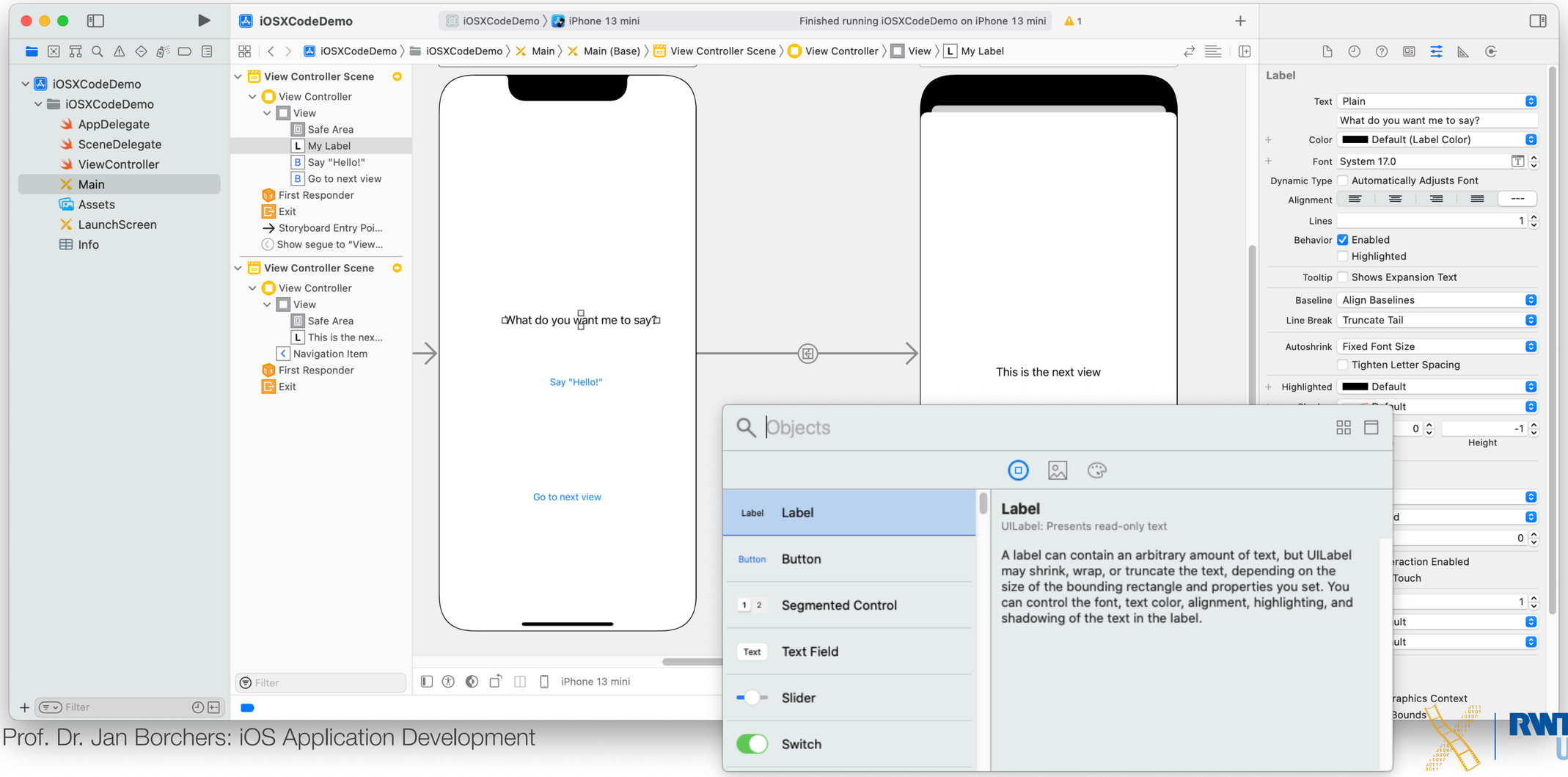






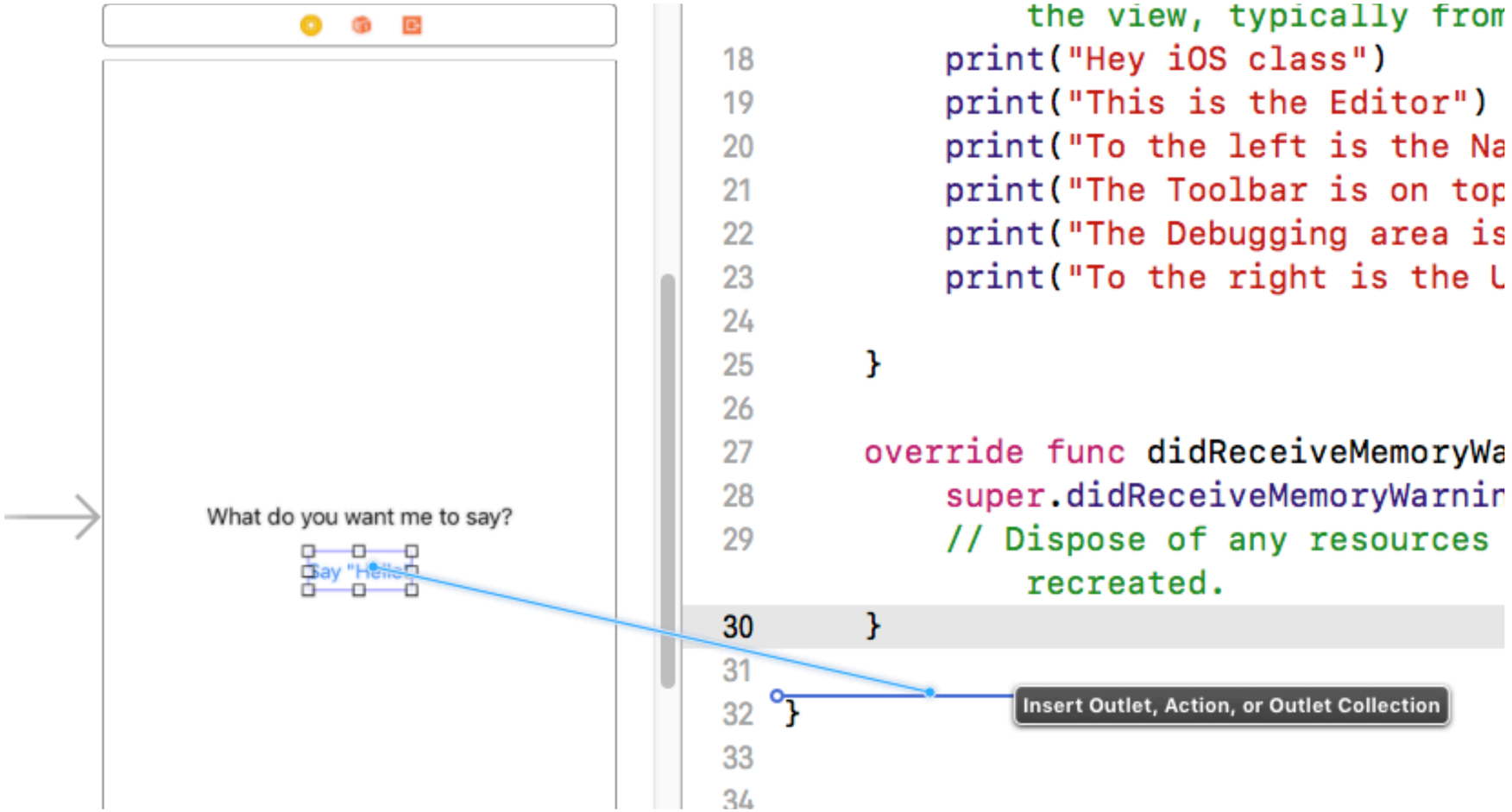
Interface Builder

- Visually define your UI



Outlets & Actions

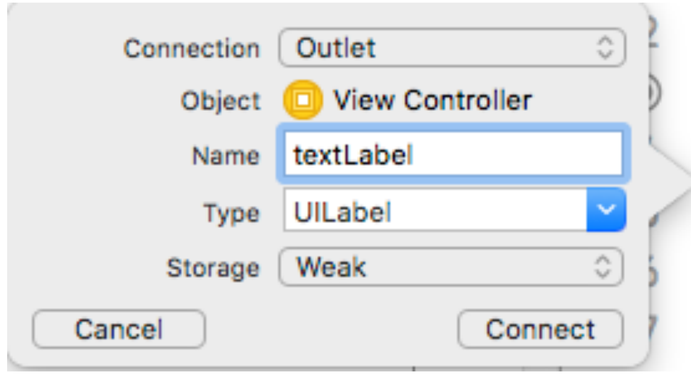
- Connect your UI elements with your code: Right-click + Drag



Outlets & Actions

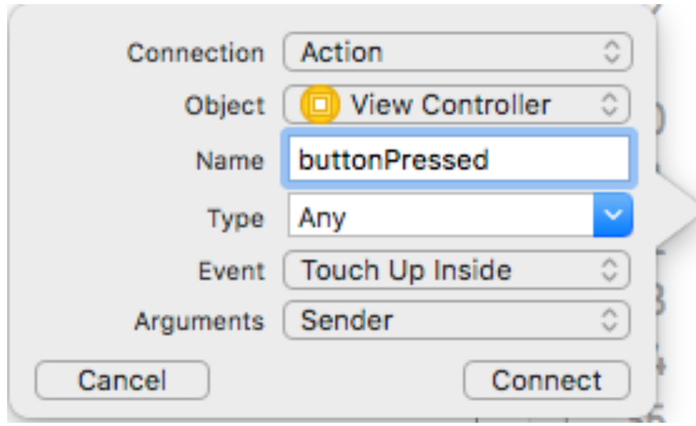
- IBOutlet
  - Access the UI element from code

```
@IBOutlet weak var textLabel: UILabel!
```



- IBAction
  - Receive UI events

```
@IBAction func buttonPressed(_ sender: Any) {}
```



Seminar

- 2 presentations per session
- Attendance is **mandatory**
  - Missing >1 time will lead to a 5.0 for the seminar
- 15 min presentation, ~10 min discussion
- 3 people per group
- Dates:
  - 22.11., 28.11., 29.11., 5.12., 6.12., 12.12., 13.12.
- Order is not fixed yet

- Finished version due one week before your presentation
- 15 min slide and content discussions one week before your presentation

Seminar

- Framework overview, conceptional structure
- Demo (small Playground app for Moodle)
  - Show how a problem can be solved elegantly using the framework
  - For most topics, your demo can be in either UIKit or SwiftUI
- Not a list of APIs;
  instead problem–solution oriented

- Structure:
  - Brief introduction & motivation
  - Basic steps to use the framework
  - Explain one or two advanced features, and show how to use them
  - Code demo
- Deliverables:
  slides and demo code

1. Core Animation

- **Drawing and animating what's on the screen**

- What to look at:
  Layers, paths, shapes, clipping, rasterization, keyframe animations, CADisplayLink

2. Haptics and Sound

- **Enriching interaction with sound and haptic feedback**

- What to look at:
  AVAudioPlayer, AVAudioSession, MPNowPlayingInfoCenter, UIFeedbackGenerator

3. Core Image + CI Filters

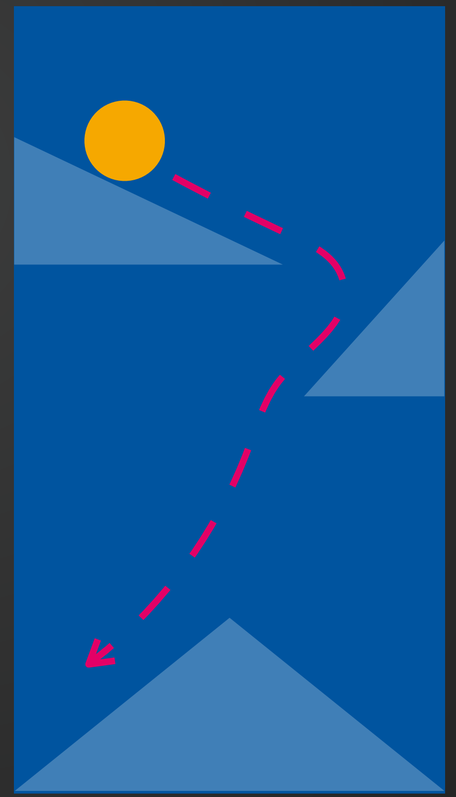
- **Fast image processing and analysis**

- What to look at:
  Automatic Enhancements, CIDetector

4. SpriteKit

- **2D games**

- What to look at:
  Nodes, scenes, actions, constraints, physics



5. Working with Files

- **How to save data to a file and find it in the Files app**

- What to look at:
  FileManager, FileHandle, DocumentBrowser,
  Files app integration

6. Combine

- **Declarative event processing**

- What to look at:
  Publishers & Subscribers, how can the Cancellables of Combine be used for declarative UIs with UIKit?

7. Debugging in Xcode

- **Using the debugger and Instruments**

- What to look at:
  print out, View Debugger, exception breakpoints, memory leaks, …

8. Displaying Rich Articles

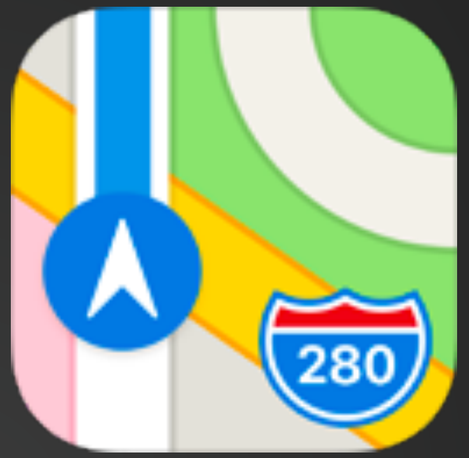
- **Displaying HTML contents in your app**

- What to look at:
  WKWebView, UITextView, NSAttributedString,
  NSParagraphStyle

9. MapKit

- **Interactive maps and directions**

- What to look at:
  MapKit, CLLocationManager, map styles, overlays, callouts, paths, …



10. UIPresentationController

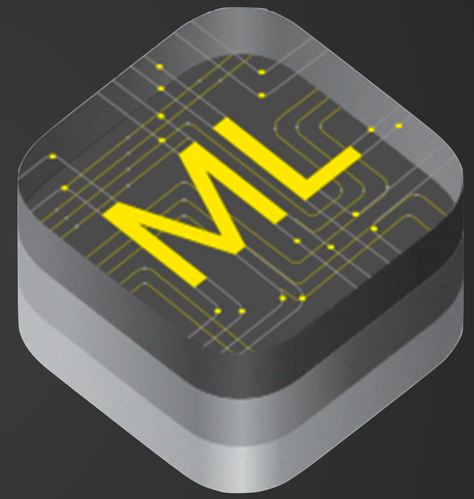
- **Create a custom view controller presentation style**

- What to look at:
  UIPresentationController,
  UIViewControllerAnimatedTransitioning,
  UIViewControllerTransitioningDelegate

11. Core ML + Create ML

- **Machine Learning in iOS**

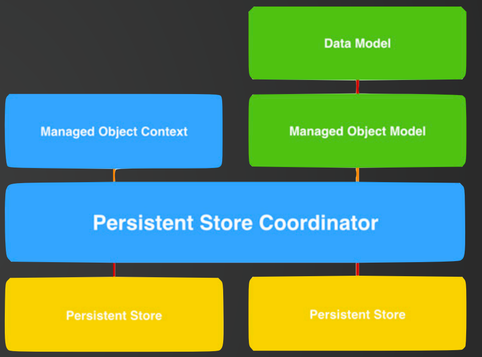
- What to look at:
  Framework overview in general, but focus on image classification



12. Core Data

- **Persistent database**

- What to look at:
  Managed objects, view context, fetch requests, predicates, entity relationship diagram in graphical model editor



13. watchOS

- **Designing native apps for the Watch**

- What to look at:
  Limitations of the UI toolkit?
  Communication between phone and watch?
  Layout in watch apps
  Special widgets

- Can use WatchKit or SwiftUI

14. ARKit

- **Showing AR content in a 3D graphics engine**

- What to look at:
  Session and Configuration, AR anchors, AR onboarding, plane detection, hit testing

15. RealityKit & Reality Composer

- **Simulate and render 3D content in AR**

- What do look at:
  Prototype AR scenes and apps, interaction with the environment

16. Advanced SwiftUI Layout

- **Using GeometryReader, priorities, fixed dimensions, alignment guides and more to create great UIs.**

- What to look at:
  How can you express relationships (e.g. resizing a view based on the contents of some other view) in SwiftUI?
  How can we achieve complex layering and scrolling (e.g. stretchy headers)?

Summary

- Swift: fast, safe, expressive
- Data types, control flow, tuples
- Development Environment
  - Xcode
- Next: Strings, classes, and structs

- Seminar Topics

What's Next?

Vote for your topic

The topics can be ranked in RWTHmoodle

Only one group member should do the ranking!

Deadline: Wednesday,19.10., at **13:00**

Results will be published on Thursday