goto; copenhagen

GOTO Copenhagen 2019
Conference Nov. 18 - 20

What's New in Swift

Daniel H Steinberg

Follow us @gotocph

What's New in Swift

GoTo

Copenhagen, Denmark
November, 2019

Daniel H Steinberg

dimsumthinking.com

Big Things

Swift 5 ABI stability

Swift 5.1 Module stability

Little Things

```
func double(_ input: Int) -> Int {

}
```

```
func double(_ input: Int) -> Int {
    return  2 * input
}
```

SE-0255 Implicit returns from single-expression functions

```
func double(_ input: Int) -> Int {
    return  2 * input
}
```

SE-0255 Implicit returns from single-expression functions

```
func double(_ input: Int) -> Int {
    2 * input
}
```

SE-0255 Implicit returns from single-expression functions

```
func double(_ input: Int) -> Int {
    print("This is an error")
    2 * input
}
```

SE-0255 Implicit returns from single-expression functions

```
func double(_ input: Int) -> Int {
    if (condition) {
        2 * input
    } else {
        0
    }
}
```

SE-0255 Implicit returns from single-expression functions

```
func double(_ input: Int) -> Int {
    if (condition) {
        2 * input
    } else {
        0
    }
}
```

SE-0255 Implicit returns from single-expression functions

```
func double(_ input: Int) -> Int {
    if (condition) {
        return 2 * input
    } else {
        return 0
    }
}
```

SE-0255 Implicit returns from single-expression functions

```
var description: String {

}
```

SE-0255 Implicit returns from single-expression functions

```
var description: String {
    return "Double \(value) is \(double(value)"
}
```

SE-0255 Implicit returns from single-expression functions

```
var description: String {
    return "Double \(value) is \(double(value)"
}
```

SE-0255 Implicit returns from single-expression functions

```
var description: String {
    "Double \(value) is \(double(value)"
}
```

Raw Strings

SE-0200 Enhancing String Literals Delimiters to Support Raw Text

```
"I say Happy Birthday to \(name),
you're \(age * 7) years old."
```

SE-0200 Enhancing String Literals Delimiters to Support Raw Text

```
"I say Happy Birthday to \(name),
you're \(age * 7) years old."
```

I say Happy Birthday to Annabelle, you're 84 years old.

SE-0200 Enhancing String Literals Delimiters to Support Raw Text

```
"I say Happy Birthday to \(name),
you're \(age * 7) years old."
```

SE-0200 Enhancing String Literals Delimiters to Support Raw Text

```
"I say "Happy Birthday" to \(name),
you're \(age * 7) years old."
```

SE-0200 Enhancing String Literals Delimiters to Support Raw Text

```
"I say "Happy Birthday" to \(name),
you're \(age * 7) years old."
```

SE-0200 Enhancing String Literals Delimiters to Support Raw Text

```
"I say "Happy Birthday" to \(name),
you're \(age * 7) years old."
```

SE-0200 Enhancing String Literals Delimiters to Support Raw Text

```
#"I say "Happy Birthday" to \(name),
you're \(age * 7) years old."#
```

SE-0200 Enhancing String Literals Delimiters to Support Raw Text

```
#"I say "Happy Birthday" to \(name),
you're \(age * 7) years old."#
```

I say "Happy Birthday" to \Annabelle, you're \84 years old.

SE-0200 Enhancing String Literals Delimiters to Support Raw Text

```
#"I say "Happy Birthday" to \(name),
you're \(age * 7) years old."#
```

I say "Happy Birthday" to \Annabelle, you're \84 years old.

SE-0200 Enhancing String Literals Delimiters to Support Raw Text

```
#"I say "Happy Birthday" to \(name),
you're \(age * 7) years old."#
```

I say "Happy Birthday" to \Annabelle, you're \84 years old.

SE-0200 Enhancing String Literals Delimiters to Support Raw Text

```
#"I say "Happy Birthday" to \#(name),
you're \#(age * 7) years old."#
```

SE-0200 Enhancing String Literals Delimiters to Support Raw Text

```
#"I say "Happy Birthday" to \#(name),
you're \#(age * 7) years old."#
```

I say "Happy Birthday" to Annabelle, you're 84 years old.

SE-0200 Enhancing String Literals Delimiters to Support Raw Text

```
#"""
I say "Happy Birthday" to \#(name), \#
you're \#(age * 7) years old.
"""#
```

SE-0200 Enhancing String Literals Delimiters to Support Raw Text

```
#"""
I say "Happy Birthday" to \#(name), \#
you're \#(age * 7) years old.
"""#
```

I say "Happy Birthday" to Annabelle,
you're 84 years old.

String Interpolation

SE-0228 Fix ExpressibleByStringInterpolation

```
let dog = Dog(name: name, age: age)
```

SE-0228 Fix ExpressibleByStringInterpolation

```
extension String.StringInterpolation {



}
```

SE-0228 Fix ExpressibleByStringInterpolation

```
extension String.StringInterpolation {
    mutating func appendInterpolation(_ dog: Dog) {


    }
}
```

SE-0228 Fix ExpressibleByStringInterpolation

```
extension String.StringInterpolation {
    mutating func appendInterpolation(_ dog: Dog) {
        appendInterpolation("\(dog.name) is
                             \(dog.age * 7) years old")
    }
}
```

SE-0228 Fix ExpressibleByStringInterpolation

```
print("\(dog)")
```

"Annabelle is 84 years old"

KeyPath Expressions

SE-0249 KeyPath Expressions as Functions

```
let phrase1 = "Annabelle, my dog, is not the
                smartest animal in Ohio."

let phrase2 = "Madam, in Eden, I'm Adam."
```

SE-0249 KeyPath Expressions as Functions

```
func normalized(_ string: String) -> String {



}
```

SE-0249 KeyPath Expressions as Functions

```
func normalized(_ string: String) -> String {
    return string



}
```

SE-0249 KeyPath Expressions as Functions

```
func normalized(_ string: String) -> String {
    return string
        .filter{!$0.isWhitespace}


}
```

SE-0249 KeyPath Expressions as Functions

```
func normalized(_ string: String) -> String {
    return string
        .filter{!$0.isWhitespace}
        .filter{!$0.isPunctuation}

}
```

SE-0249 KeyPath Expressions as Functions

```
func normalized(_ string: String) -> String {
    return string
        .filter{!$0.isWhitespace}
        .filter{!$0.isPunctuation}
        .lowercased()
}
```

SE-0249 KeyPath Expressions as Functions

```
func normalized(_ string: String) -> String {
    return string
        .filter{!$0.isWhitespace}
        .filter{!$0.isPunctuation}
        .lowercased()
}
```

SE-0249 KeyPath Expressions as Functions

```
func normalized(_ string: String) -> String {
    return string
        .filter{!\isWhitespace}
        .filter{!\isPunctuation}
        .lowercased()
}
```

SE-0249 KeyPath Expressions as Functions

Didn't make it into 5.1, but soon

```
func normalized(_ string: String) -> String {
    return string
        .filter{!\isWhitespace}
        .filter{!\isPunctuation}
        .lowercased()
}
```

SE-0249 KeyPath Expressions as Functions

Example

SE-0249 KeyPath Expressions as Functions

```
extension Sequence {
    func sorted
                                                    {




    }
}
```

SE-0249 KeyPath Expressions as Functions

```
extension Sequence {
    func sorted<Attribute: Comparable>
                                                    {
    }
}
```

SE-0249 KeyPath Expressions as Functions

```
extension Sequence {
    func sorted<Attribute: Comparable>
                (by keyPath: KeyPath<Element, Attribute>,
                 isIncreasing: Bool = true) -> [Element] {









    }
}
```

SE-0249 KeyPath Expressions as Functions

```
extension Sequence {
    func sorted<Attribute: Comparable>
                (by keyPath: KeyPath<Element, Attribute>,
                 isIncreasing: Bool = true) -> [Element] {
        sorted{first, second in









        }
    }
}
```

SE-0249 KeyPath Expressions as Functions

```
extension Sequence {
    func sorted<Attribute: Comparable>
                (by keyPath: KeyPath<Element, Attribute>,
                 isIncreasing: Bool = true) -> [Element] {
        sorted{first, second in
            if isIncreasing {
               return first[keyPath: keyPath] < second[keyPath: keyPath]
            }


        }
    }
}
```

SE-0249 KeyPath Expressions as Functions

```
extension Sequence {
    func sorted<Attribute: Comparable>
                (by keyPath: KeyPath<Element, Attribute>,
                 isIncreasing: Bool = true) -> [Element] {
        sorted{first, second in
            if isIncreasing {
                return first[keyPath: keyPath] < second[keyPath: keyPath]
            } else {
                return first[keyPath: keyPath] > second[keyPath: keyPath]
            }
        }
    }
}
```

Result

SE-0235 Add Result to the Standard Library

```
public enum Result<Success, Failure: Error> {
    case success(Success)
    case failure(Failure)
}
```

SE-0235 Add Result to the Standard Library

Like Optional

```
public enum Result<Success, Failure: Error> {
    case success(Success)
    case failure(Failure)
}
```

SE-0235 Add Result to the Standard Library

Like Optional

```
public enum Result<Success, Failure: Error> {
    case success(Success)
    case failure(Failure)
}
```

SE-0235 Add Result to the Standard Library

```
public enum Result<Success, Failure: Error> {
    case success(Success)
    case failure(Failure)
}
```

SE-0235 Add Result to the Standard Library

```
public enum Result<Success, Failure: Error> {
    case success(Success)
    case failure(Failure)
}
```

SE-0235 Add Result to the Standard Library

Like Either

```
public enum Result<Success, Failure: Error> {
    case success(Success)
    case failure(Failure)
}
```

SE-0235 Add Result to the Standard Library

```
func doubled(_ input: Int) -> Result<Int, InputOutOfBoundsError> {




}
```

SE-0235 Add Result to the Standard Library

```
func doubled(_ input: Int) -> Result<Int, InputOutOfBoundsError> {




}
```

SE-0235 Add Result to the Standard Library

```
func doubled(_ input: Int) -> Result<Int, InputOutOfBoundsError> {




}
```

SE-0235 Add Result to the Standard Library

```
enum InputOutOfBoundsError          {

}
```

SE-0235 Add Result to the Standard Library

```
enum InputOutOfBoundsError: Error {

}
```

SE-0235 Add Result to the Standard Library

```
enum InputOutOfBoundsError: Error {
    case negativeNumber
    case numberIsTooLarge(amountOver: Int)
}
```

SE-0235 Add Result to the Standard Library

```
func doubled(_ input: Int) -> Result<Int, InputOutOfBoundsError> {




}
```

SE-0235 Add Result to the Standard Library

```
func doubled(_ input: Int) -> Result<Int, InputOutOfBoundsError> {
    guard input >= 0 else {
        return .failure(.negativeNumber)}



}
```

SE-0235 Add Result to the Standard Library

```
func doubled(_ input: Int) -> Result<Int, InputOutOfBoundsError> {
    guard input >= 0 else {
        return .failure(.negativeNumber)}
    guard input <= 10 else {
        return .failure( .numberIsTooLarge(
                    amountOver: input - 10))}
    return .success(2 * input)
}
```

SE-0235 Add Result to the Standard Library

```
func doubled(_ input: Int) -> Result<Int, InputOutOfBoundsError> {
    guard input >= 0 else {
        return .failure(.negativeNumber)}
    guard input <= 10 else {
        return .failure( .numberIsTooLarge(
                    amountOver: input - 10))}
    return .success(2 * input)
}
```

SE-0235 Add Result to the Standard Library

```
doubled(2)
```

success(4)

SE-0235 Add Result to the Standard Library

```
doubled(doubled(2))
```

?

SE-0235 Add Result to the Standard Library

```
extension Result {
    func map<NewSuccess>(_ transform: (Success) -> NewSuccess)
                                    -> Result<NewSuccess, Failure> {









    }
}
```

SE-0235 Add Result to the Standard Library

```
extension Result {
    func map<NewSuccess>(_ transform: (Success) -> NewSuccess)
                                    -> Result<NewSuccess, Failure> {









    }
}
```

SE-0235 Add Result to the Standard Library

```
extension Result {
    func map<NewSuccess>(_ transform: (Success) -> NewSuccess)
                                    -> Result<NewSuccess, Failure> {









    }
}
```

SE-0235 Add Result to the Standard Library

```
extension Result {
    func map<NewSuccess>(_ transform: (Success) -> NewSuccess)
                                    -> Result<NewSuccess, Failure> {




    }
}
```

SE-0235 Add Result to the Standard Library

```
extension Result {
    func map<NewSuccess>(_ transform: (Success) -> NewSuccess)
                                    -> Result<NewSuccess, Failure> {
        switch self {




        }
    }
}
```

SE-0235 Add Result to the Standard Library

```
extension Result {
    func map<NewSuccess>(_ transform: (Success) -> NewSuccess)
                                    -> Result<NewSuccess, Failure> {
        switch self {
        case .failure(let error):
            return .failure(error)


        }
    }
}
```

SE-0235 Add Result to the Standard Library

```
extension Result {
    func map<NewSuccess>(_ transform: (Success) -> NewSuccess)
                                    -> Result<NewSuccess, Failure> {
        switch self {
        case .failure(let error):
            return .failure(error)
        case .success(let success):
            return .success(transform(success))
        }
    }
}
```

SE-0235 Add Result to the Standard Library

```
doubled(doubled(2))
```

?

SE-0235 Add Result to the Standard Library

```
doubled(2).map(doubled)
```

success(8)

Dynamic

```
struct Days {
    let values = ["mon", "tue", "wed",
                  "thu", "fri", "sat", "sun"]
}
```

```
struct Days {
    let values = ["mon", "tue", "wed",
                  "thu", "fri", "sat", "sun"]
}

let days = Days()
```

```
extension Days {
    subscript(index: Int) -> String {

    }
}
```

```
extension Days {
    subscript(index: Int) -> String {
        return values[index]
    }
}
```

days[1]

tue

```
extension Days {
    subscript(index: Int,
              reversed isReversed: Bool) -> String {
                                  return values[index]

    }
}
```

```
extension Days {
    subscript(index: Int,
              reversed isReversed: Bool) -> String {
        guard isReversed else {return values[index]}

    }
}
```

```
extension Days {
    subscript(index: Int,
                  reversed isReversed: Bool) -> String {
        guard isReversed else {return values[index]}
        return values[values.count - 1 - index]
    }
}
```

```
days[1, reversed: false]
```

tue

```
days[1, reversed: true]
```

sat

SE-0195 Dynamic Member Lookup

```
@dynamicMemberLookup

struct Dog {




}
```

SE-0195 Dynamic Member Lookup

```
@dynamicMemberLookup

struct Dog {
    let properties: [String: CustomStringConvertible]




}
```

SE-0195 Dynamic Member Lookup

```
@dynamicMemberLookup

struct Dog {
    let properties: [String: CustomStringConvertible]

    subscript(dynamicMember member: String) -> String? {

    }
}
```

SE-0195 Dynamic Member Lookup

```
@dynamicMemberLookup

struct Dog {
    let properties: [String: CustomStringConvertible]

    subscript(dynamicMember member: String) -> String? {
        properties[member]?.description
    }
}
```

SE-0195 Dynamic Member Lookup

```
let annabelle = Dog(properties: ["age": 12,
                                 "name": "Annabelle"])
```

SE-0195 Dynamic Member Lookup

```
let annabelle = Dog(properties: ["age": 12,
                                 "name": "Annabelle"])

annabelle.age
```

"12"

SE-0195 Dynamic Member Lookup

```
let annabelle = Dog(properties: ["age": 12,
                                 "name": "Annabelle"])

annabelle.name
```

"Annabelle"

SE-0195 Dynamic Member Lookup

```
let annabelle = Dog(properties: ["age": 12,
                                 "name": "Annabelle"])

annabelle.isVaccinated
```

nil

SE-0216 Dynamic Callable

```
@dynamicCallable
struct SillyExample {









}
```

SE-0216 Dynamic Callable

```
@dynamicCallable
struct SillyExample {
    func dynamicallyCall(withArguments args: [Any]) -> Int {
        args.count
    }

}
```

SE-0216 Dynamic Callable

```
@dynamicCallable
struct SillyExample {
    func dynamicallyCall(withArguments args: [Any]) -> Int {
        args.count
    }
}
```

```
let example = SillyExample()
example(1,2,3)
```

```
3
```

SE-0216 Dynamic Callable

```
@dynamicCallable
struct SillyExample {
    func dynamicallyCall(withArguments args: [Any]) -> Int {
        args.count
    }
}
```

```
let example = SillyExample()
example(1, "a")
```

2

SE-0216 Dynamic Callable

```
@dynamicCallable
struct SillyExample {
    func dynamicallyCall(withArguments args: [Any]) -> Int {
        args.count
    }

    func dynamicallyCall(withKeywordArguments args:
                         KeyValuePairs<String, CustomStringConvertible>)
                                                  -> [String: String] {




    }
}
```

SE-0216 Dynamic Callable

```
@dynamicCallable
struct SillyExample {
    func dynamicallyCall(withArguments args: [Any]) -> Int {
        args.count
    }

    func dynamicallyCall(withKeywordArguments args:
                         KeyValuePairs<String, CustomStringConvertible>)
                                                -> [String: String] {
        var result = [String: String]()



        return result
    }
}
```

SE-0216 Dynamic Callable

```
@dynamicCallable
struct SillyExample {
    func dynamicallyCall(withArguments args: [Any]) -> Int {
        args.count
    }

    func dynamicallyCall(withKeywordArguments args:
                         KeyValuePairs<String, CustomStringConvertible>)
                                                -> [String: String] {
        var result = [String: String]()
        for arg in args {
            result[arg.key] = arg.value.description
        }
        return result
    }
}
```

SE-0216 Dynamic Callable

```
@dynamicCallable
struct SillyExample { //…
    func dynamicallyCall(withKeywordArguments args:
                         KeyValuePairs<String, CustomStringConvertible>)
                                                  -> [String: String] {
        var result = [String: String]()
        for arg in args {
            result[arg.key] = arg.value.description
        }
        return result
    }
}
                         let example = SillyExample()
                         example(name: "Annabelle", age: 12)
```

["age": "12", "name": "Annabelle"]

Opaque Result Type

SE-0244 Opaque Result Types

```
var body: some View {
   Text("Hello World")
}
```

SE-0244 Opaque Result Types

```
var body: some View {
   Text("Hello World")
}
```

SE-0244 Opaque Result Types

```
var body: some View {
   Text("Hello World")
}
```

SE-0244 Opaque Result Types

Can't do this in Swift

```
var body: View {
   Text("Hello World")
}
```

SE-0244 Opaque Result Types

```
var body: some View {
   Text("Hello World")
}
```

SE-0244 Opaque Result Types

```
var body: some View {
   Text("Hello World")
}
```

SE-0244 Opaque Result Types

```
var body: some View {
    Image(uiImage: goToImage)
}
```

SE-0244 Opaque Result Types

```
let goToImage = UIImage(named: “GoTo")!


        var body: some View {
            Image(uiImage: goToImage)
        }
```

SE-0244 Opaque Result Types

```
let goToImage = UIImage(named: “GoTo")!
```

```
var body: some View {
   Image(uiImage: goToImage)
}
```

SE-0244 Opaque Result Types

```
var body: some View {
      if let image = goToImage {
          Image(uiImage: image)
      } else {
          Text("Hello, World!")
      }
}
```

SE-0244 Opaque Result Types

```
var body: some View {
      if let image = goToImage {
          Image(uiImage: image)
      } else {
          Text("Hello, World!")
      }
}
```

SE-0244 Opaque Result Types

```
var body: some View {
      if let image = goToImage {
          Image(uiImage: image)
      } else {
          Text("Hello, World!")
      }
}
```

SE-0244 Opaque Result Types

```
var body: some View {
      if let image = goToImage {
          Image(uiImage: image)
      } else {
          Text("Hello, World!")
      }
}
```

255 - Implicit returns from single-expression functions

```
var body: some View {
     if let image = goToImage {
         Image(uiImage: image)
     } else {
         Text("Hello, World!")
     }
}
```

255 - Implicit returns from single-expression functions

```
var body: some View {
    if let image = goToImage {
        Image(uiImage: image)
    } else {
        Text("Hello, World!")
    }
}
```

255 - Implicit returns from single-expression functions

```
var body: some View {
    if let image = goToImage {
        Image(uiImage: image)
    } else {
        Text("Hello, World!")
    }
}
```

255 - Implicit returns from single-expression functions

```
var body: some View {
    if let image = goToImage {
        return Image(uiImage: image)
    } else {
        return Text("Hello, World!")
    }
}
```

Still not fixed

```
var body: some View {
     if let image = goToImage {
         return Image(uiImage: image)
     } else {
         return Text("Hello, World!")
     }
}
```

Function declares an opaque return type,
but the return statements in its body
do not have matching underlying types

```
var body: some View {
    if let image = goToImage {
        return Image(uiImage: image)
    } else {
        return Text("Hello, World!")
    }
}
```

Function declares an opaque return type,
but the return statements in its body
do not have matching underlying types

```
var body: some View {
    if let image = goToImage {
        return Image(uiImage: image)
    } else {
        return Text("Hello, World!")
    }
}
```

Function declares an opaque return type,
but the return statements in its body
do not have matching underlying types

```
var body: some View {
    if let image = goToImage {
        return Image(uiImage: image)
    } else {
        return Text("Hello, World!")
    }
}
```

SE-0244 Opaque Result Types

```
var body: some View {
      if let image = hackingImage {
          return Image(uiImage: image)
      } else {
          return Text("Hello, World!")
      }
}
```

Function Builders


8:41
Start
goto;
conference
Hello, GoTo Copenhagen

```
var body: some View {

        Image(uiImage: goToImage)
        Text("Hello, GoTo Copenhagen")

}
```

```
var body: some View {
    VStack {
        Image(uiImage: goToImagge)
        Text("Hello, GoTo Copenhagen")
    }
}
```

```
var body: some View {
    VStack {
        Image(uiImage: goToImage)
        Text("Hello, GoTo Copenhagen")
    }
}
```


8:41
Start
goto;
conference
Hello, GoTo Copenhagen

xxx - Function builders

```
var body: some View {
    VStack {
        Image(uiImage: goToImage)
        Text("Hello, GoTo Copenhagen")
    }
}
```

xxx - Function builders

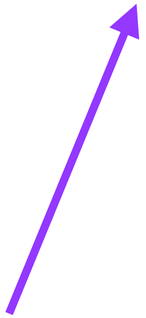
```
var body: some View {
    VStack {
        Image(uiImage: goToImage)
        Text("Hello, GoTo Copenhagen")
    }
}
```

xxx - Function builders

```
var body: some View {
    VStack.createFrom(Image(uiImage: goToImage),
                      and: Text("Hello, GoTo Copenhagen”))
}
```

xxx - Function builders

```
var body: some View {
    VStack.createFrom(Image(uiImage: goToImage),
                      and: Text("Hello, GoTo Copenhagen”))
}
```



Not really - just the idea

xxx - Function builders

```
var body: some View {
    let v = VStack()
    v.add(Image(uiImage: goToImage)
    v.add(Text("Hello, GoTo Copenhagen”))
}
```

xxx - Function builders

```
var body: some View {
    let v = VStack()
    v.add(Image(uiImage: goToImage)
    v.add(Text("Hello, GoTo Copenhagen”))
}
```

xxx - Function builders

```
var body: some View {
    let v = VStack()
    v.add(Image(uiImage: goToImage)
    v.add(Text("Hello, GoTo Copenhagen”))
}
```

xxx - Function builders

```
var body: some View {
    let v = VStack()
    v.add(Image(uiImage: goToImage)
    v.add(Text("Hello, GoTo Copenhagen”))
}
```

Again, not really - just the idea

```
var body: some View {
    VStack {
        Image(uiImage: goToImage)
        Text("Hello, GoTo Copenhagen")
    }
}
```

```
var body: some View {
    VStack {
        Image(uiImage: goToImage)
        Text("Hello, GoTo Copenhagen")
    }
}
```

@Viewbuilder: () -> Content

```
var body: some View {
    VStack {
        Image(uiImage: goToImage)
        Text("Hello, GoTo Copenhagen")
    }
}
```

```
var body: some View {
    VStack {
        Image(uiImage: goToImage)
            .resizable()
            .frame(width: 90, height: 90)
        Text("Hello, Go To Copenhagen”)
            .font(.headline)
            .padding()
        }
    }
}
```

```
var body: some View {
    VStack.createFrom(Image(uiImage: goToImage)
                            .resizable()
                            .frame(width: 90, height: 90),
                      and: Text("Hello, GoTo Copenhagen”)
                            .font(.headline)
                            .padding())
}
```

xxx - Function builders

```
var body: some View {
    VStack {
        Image(uiImage: goToImage)
            .resizable()
            .frame(width: 90, height: 90)
        Text("Hello, GoTo Copenhagen”)
            .font(.headline)
            .padding()
        }
    }
}
```

Function Builders

Example

Peter Henderson's
Picture Language 1980s

F

F

Rotate F

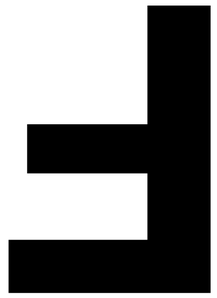
F

UnRotate F

Rotate^{-1} F

Rotate F

Rotate Rotate F



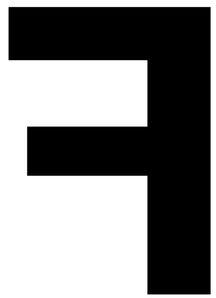
Rotate Rotate Rotate F

Rotate^{-1} F = Rotate Rotate Rotate F

F

F

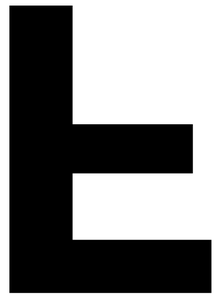
Flip F



F

F

Vertical F



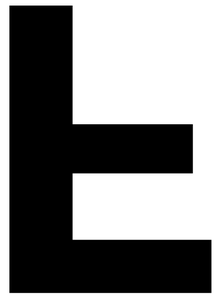
F

F

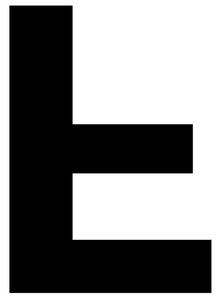
Rotate F

Flip Rotate F

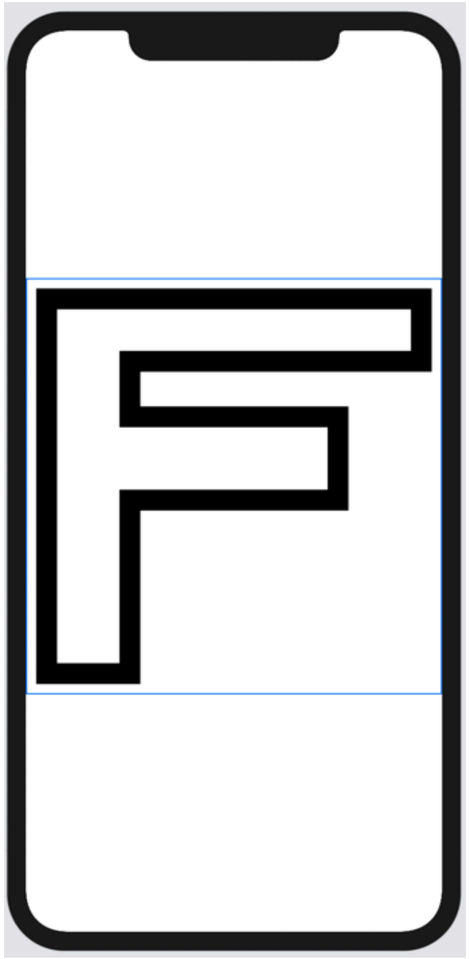
Rotate^{-1} Flip Rotate F



VerticalFlip F = Rotate^{-1} Flip Rotate F

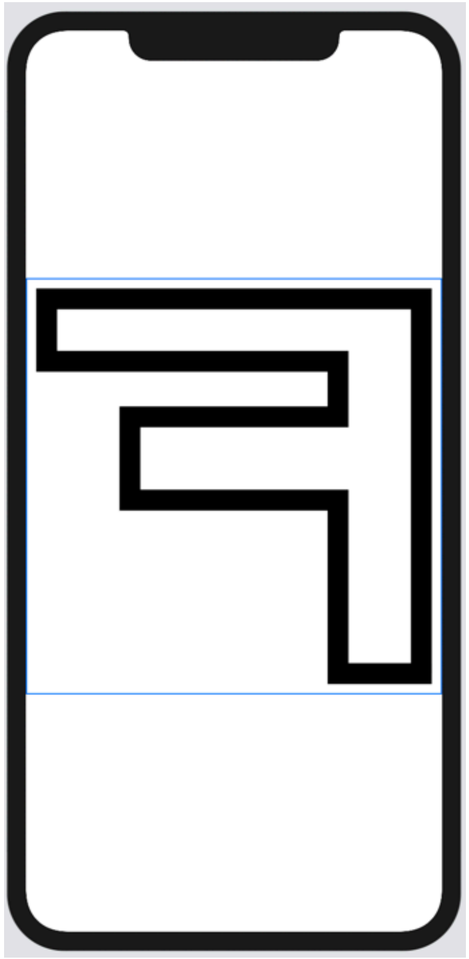


Start with a shape

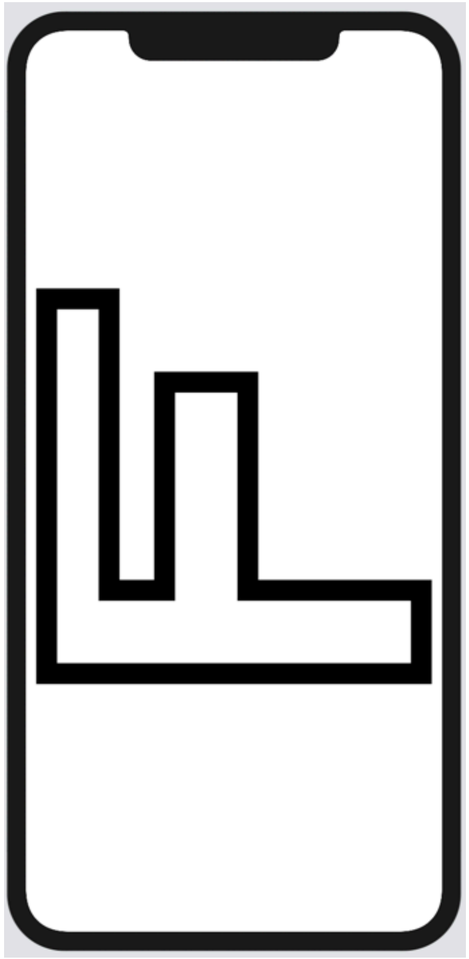


Define fundamental transformations

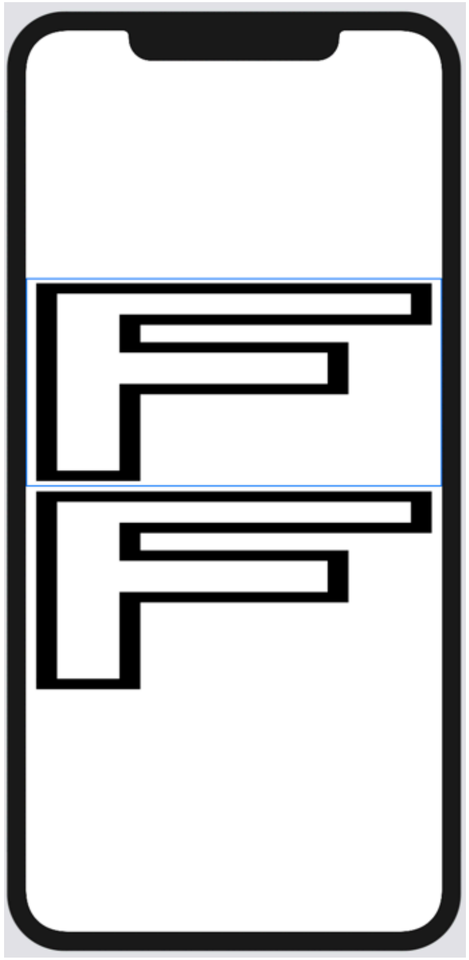
Flip



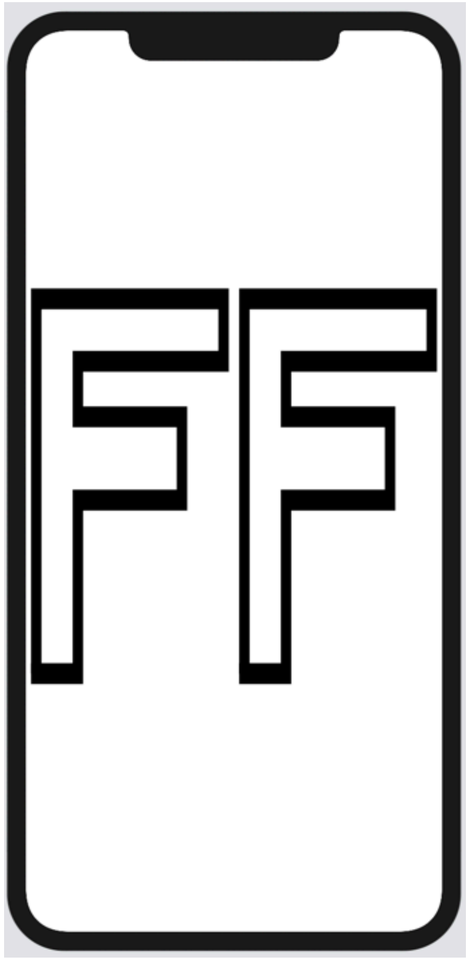
Rotate



Above

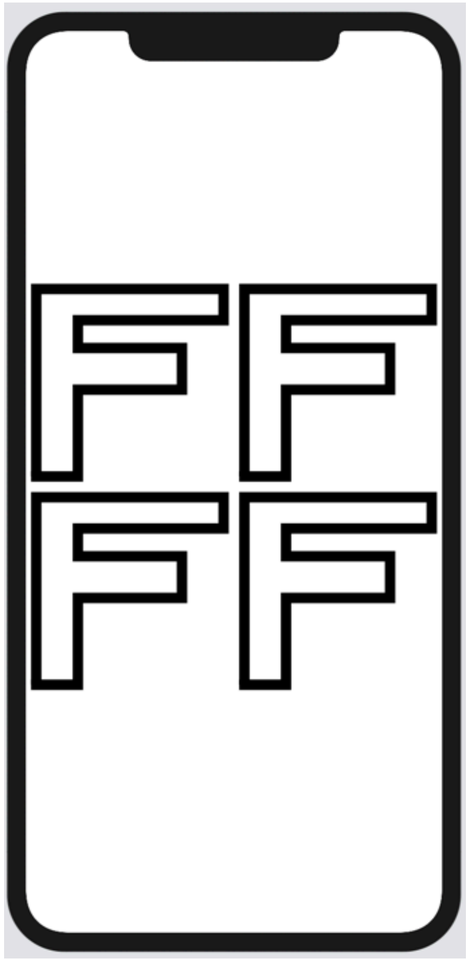


Beside



Higher Order

Quartet



Quartet

```
Above {
    Beside {
        topLeadingView
        topTrailingView
    }
    Beside {
        bottomLeadingView
        bottomTrailingView
    }
}
```

Quartet

```
Above {
    Beside {
        topLeadingView
        topTrailingView
    }
    Beside {
        bottomLeadingView
        bottomTrailingView
    }
}
```

Quartet

```
Above {
    Beside {
        topLeadingView
        topTrailingView
    }
    Beside {
        bottomLeadingView
        bottomTrailingView
    }
}
```

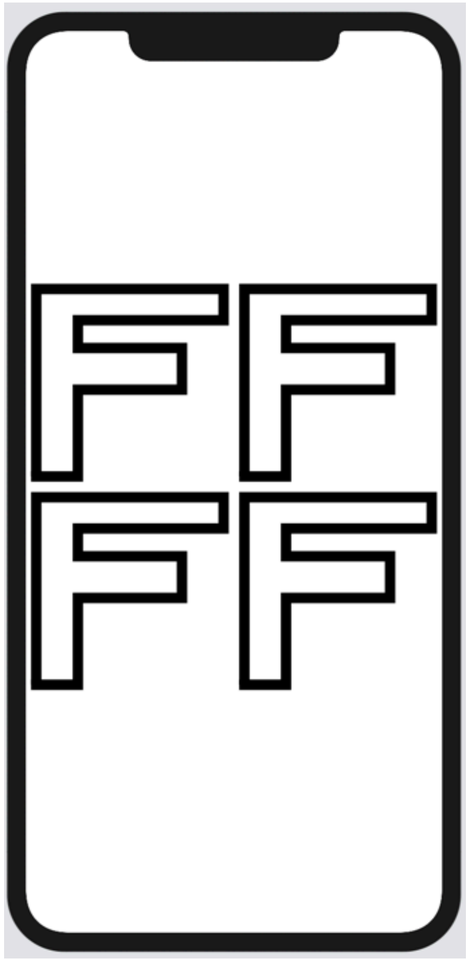
Quartet

```
Above {
    Beside {
        topLeadingView
        topTrailingView
    }
    Beside {
        bottomLeadingView
        bottomTrailingView
    }
}
```

Quartet

```
Above {
    Beside {
        topLeadingView
        topTrailingView
    }
    Beside {
        bottomLeadingView
        bottomTrailingView
    }
}
```

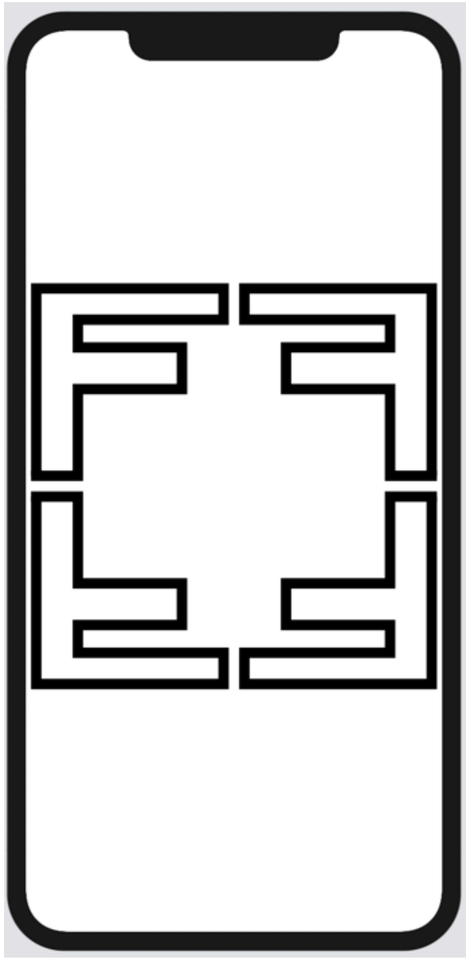
Quartet



Quartet

```
Quartet {
    F()
    Flip { F() }
    VerticalFlip { F()}
    Flip{VerticalFlip { F()}}
}
```

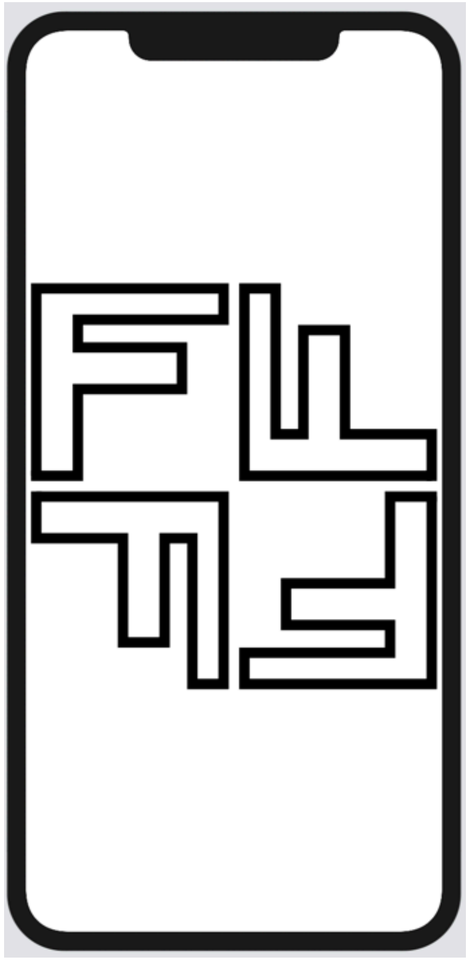
Quartet



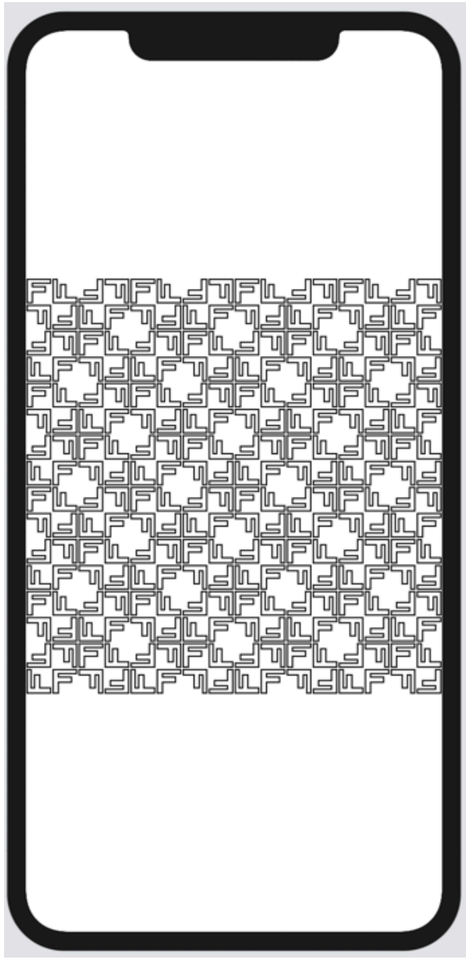
Cycle

```
Quartet {
    topLeadingView
    Rotate{topTrailingView}
    Rotate{Rotate{ Rotate{bottomLeadingView}}}
    Rotate{ Rotate{ bottomTrailingView}}
}
```

Cycle



Quartet Cycle Cycle Cycle F



Just iteration

Just iteration
Nothing fishy going on

Builder

```
struct RotateBuilder {
    static func buildBlock<Content: View>(degrees: Double,
                                          view: Content) -> Content {
        // details
    }
}
```

Builder

```
struct RotateBuilder {
    static func buildBlock<Content: View>(degrees: Double,
                                          view: Content) -> Content {
        // details
    }
}
```

Builder

```
struct RotateBuilder {
    static func buildBlock<Content: View>(degrees: Double,
                                          view: Content) -> Content {
        // details
    }
}
```

init

```
struct Rotate<Content>: View where Content: View {
    let view: Content
    let angle: Double

    init(angle: Double = 90,
         @RotateBuilder builder: () -> Content) {
        self.angle = angle
        self.view = builder()
    }
}
```

init

```
struct Rotate<Content>: View where Content: View {
    let view: Content
    let angle: Double

    init(angle: Double = 90,
         @RotateBuilder builder: () -> Content) {
        self.angle = angle
        self.view = builder()
    }
}
```

init

```
struct Rotate<Content>: View where Content: View {
    let view: Content
    let angle: Double

    init(angle: Double = 90,
         @RotateBuilder builder: () -> Content) {
        self.angle = angle
        self.view = builder()
    }
}
```

init

```
struct Rotate<Content>: View where Content: View {
    let view: Content
    let angle: Double

    init(angle: Double = 90,
         @RotateBuilder builder: () -> Content) {
        self.angle = angle
        self.view = builder()
    }
}
```

init

```
struct Rotate<Content>: View where Content: View {
    let view: Content
    let angle: Double

    init(angle: Double = 90,
         @RotateBuilder builder: () -> Content) {
        self.angle = angle
        self.view = builder()
    }
}
```

init

```
struct Rotate<Content>: View where Content: View {
    let view: Content
    let angle: Double

    init(angle: Double = 90,
         @RotateBuilder builder: () -> Content) {
        self.angle = angle
        self.view = builder()
    }
}
```

Use it

```
Rotate(angle: 60){ F() }
```

Use it

```
Rotate{ F() }
```

Just iteration
Nothing fishy going on

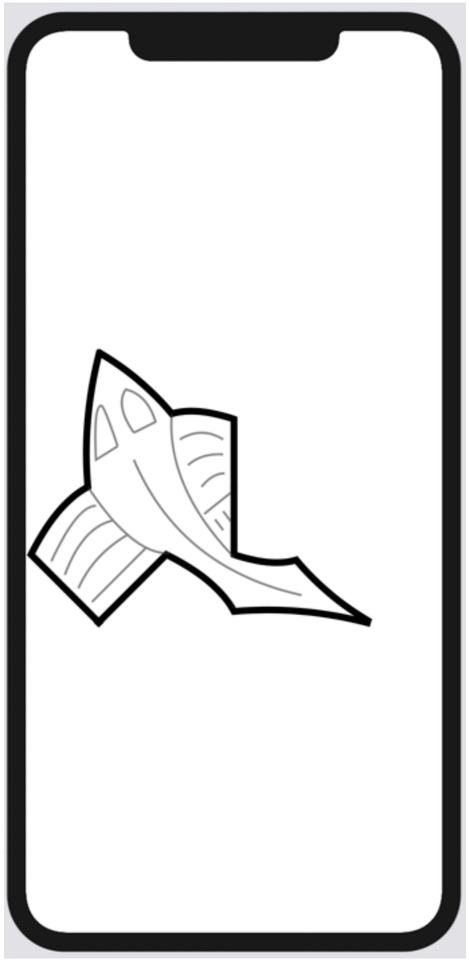
Just iteration

Nothing fishy going on

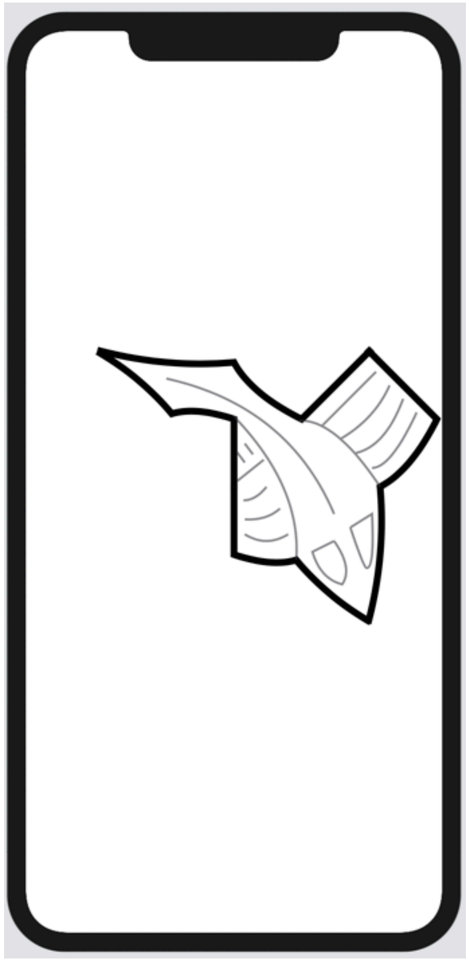
Well not so far

Fish

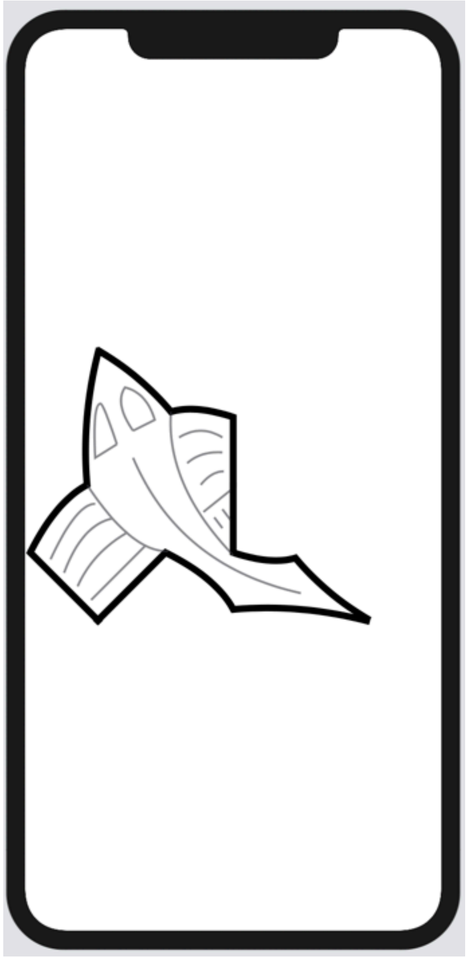
Fish

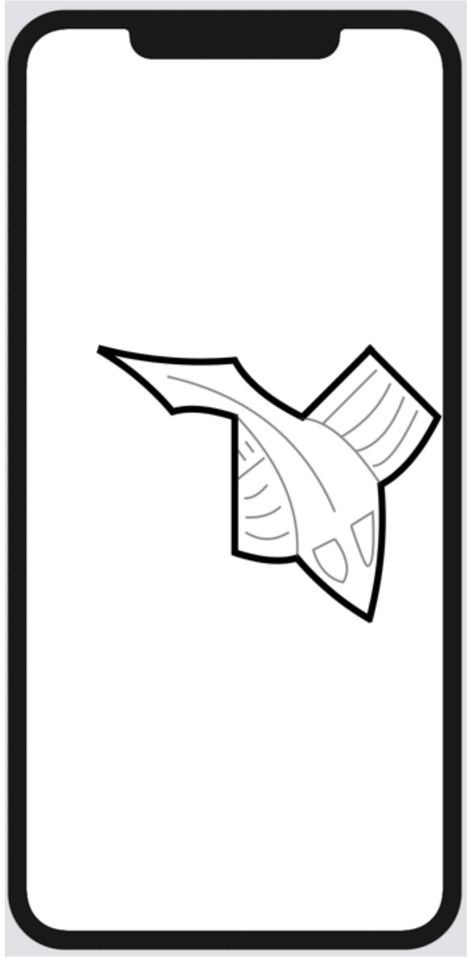


Rotate Rotate Fish



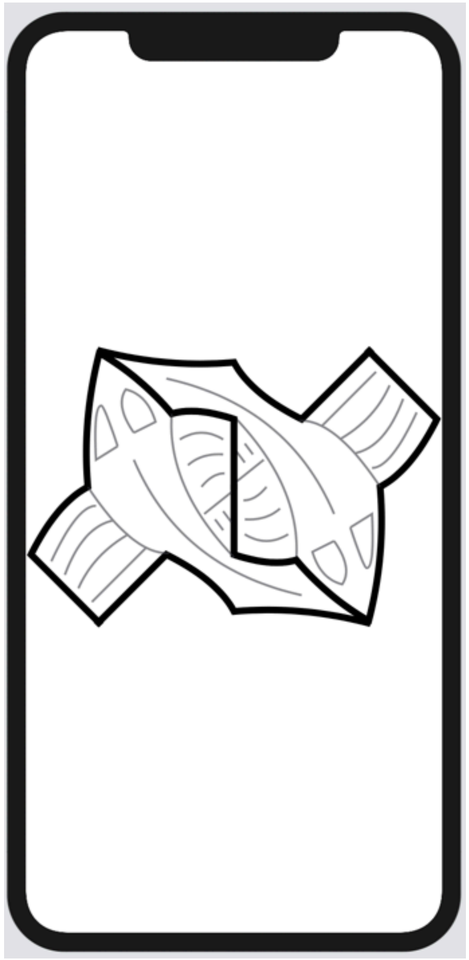
Rotate Rotate Fish



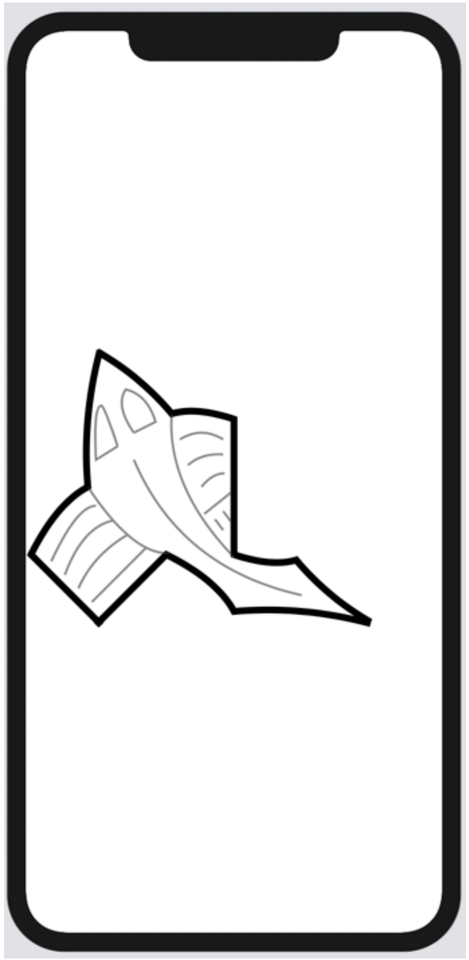


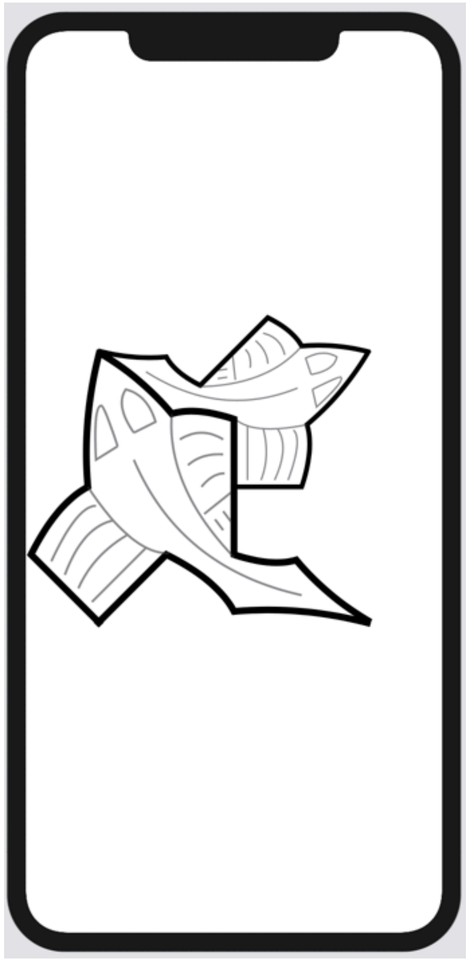
Add a New Primitive: Over

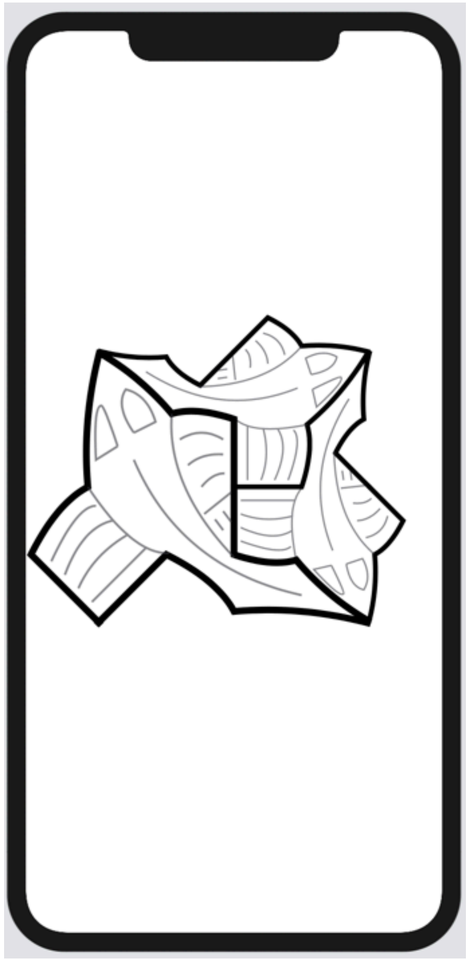
Over {Fish Rotate {Rotate Fish}}



Wait! There's more

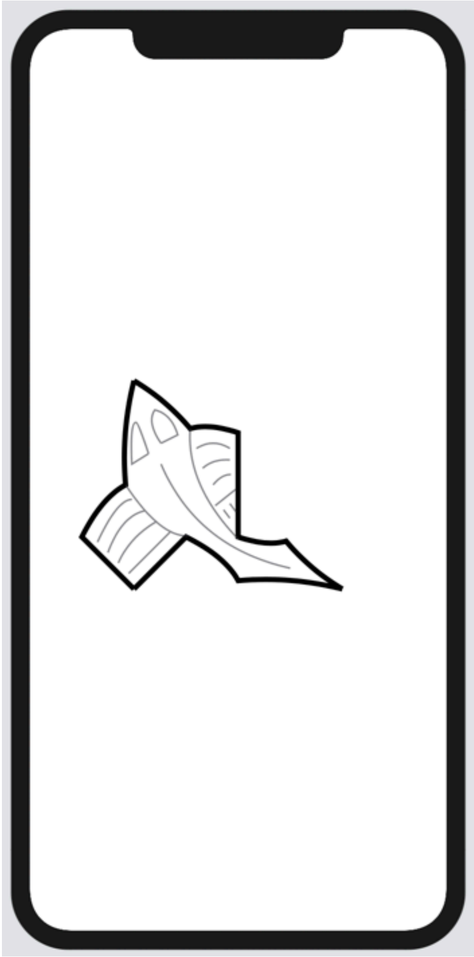


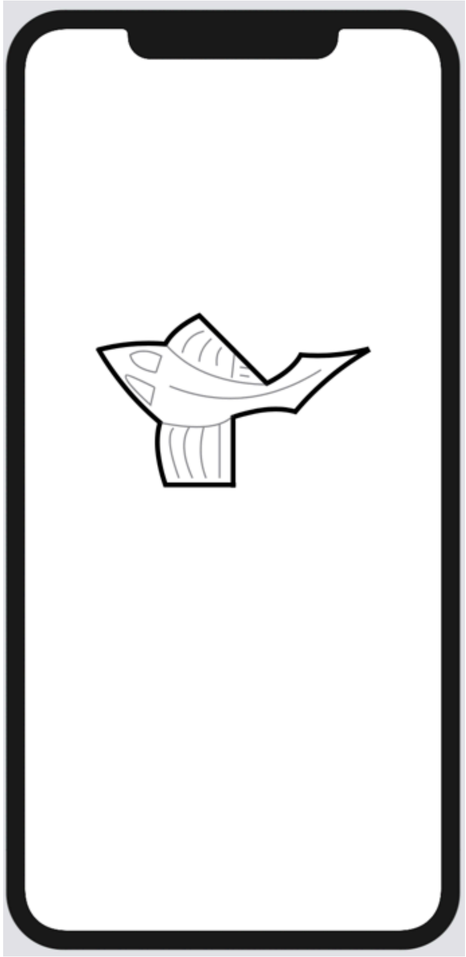




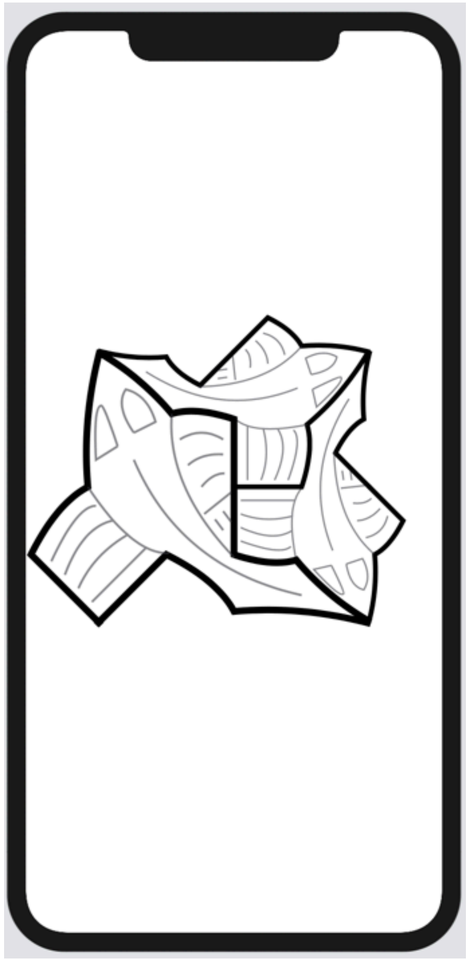
One more More Primitive: Rotate45

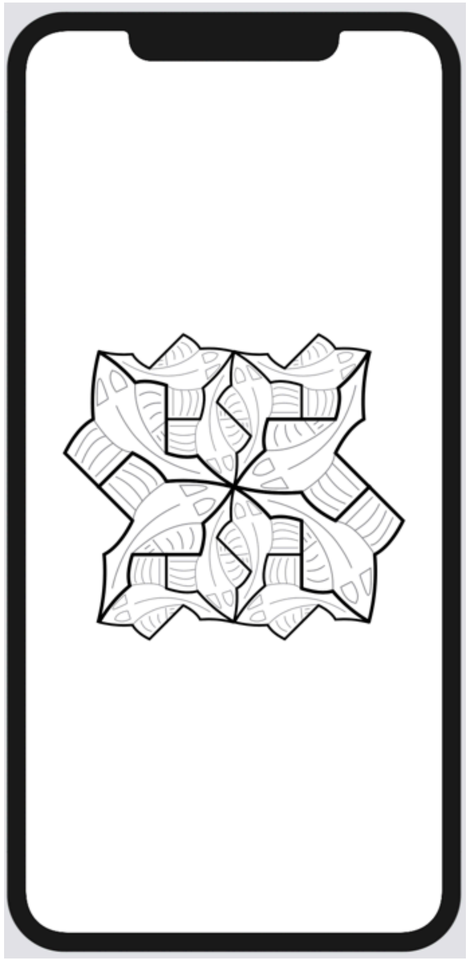
Rotate45





Iterate

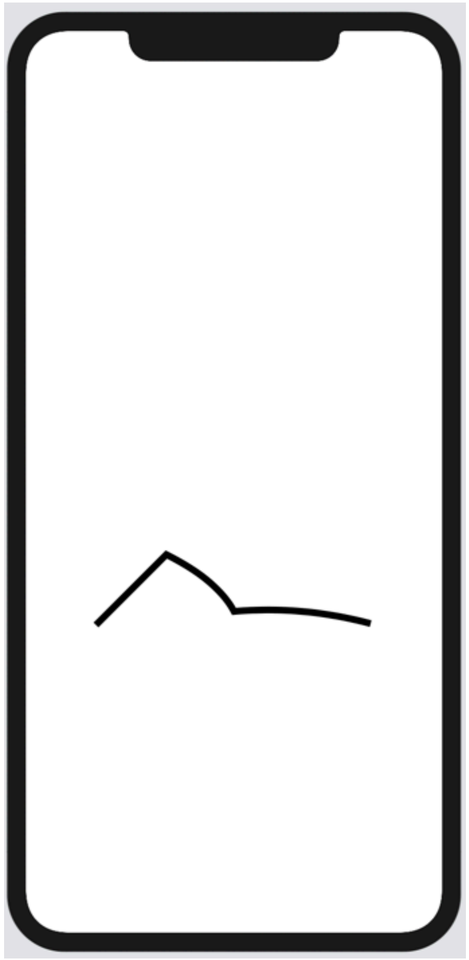




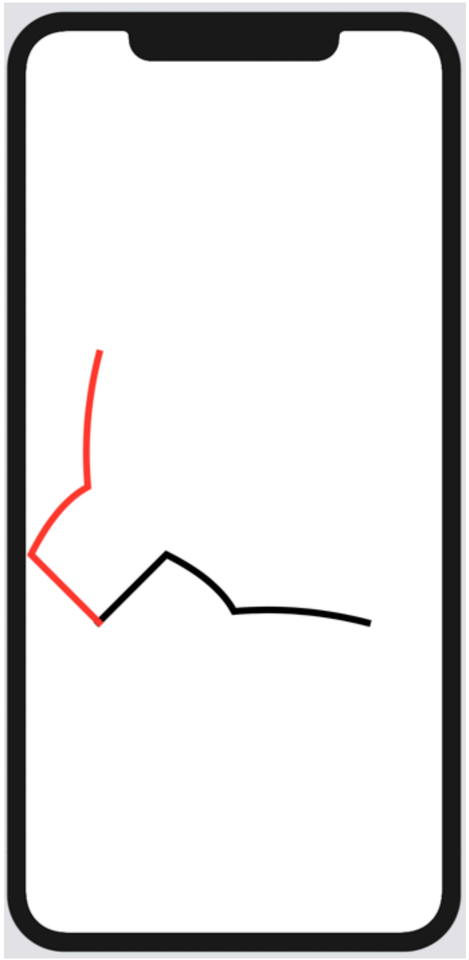
There is something fishy going on

Escher's Fish

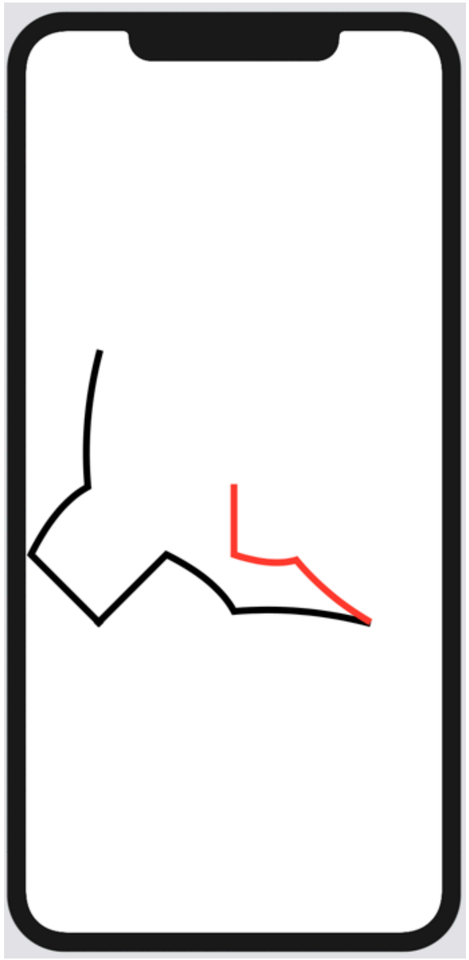
The base



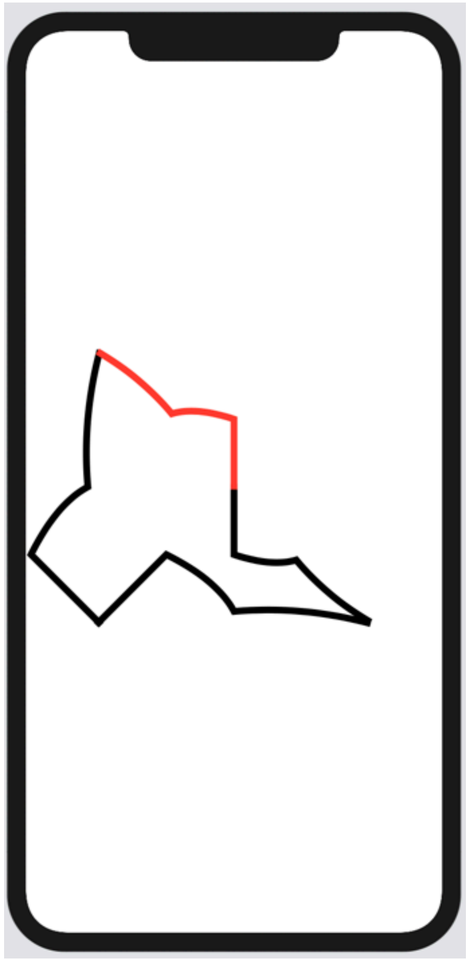
Rotate it

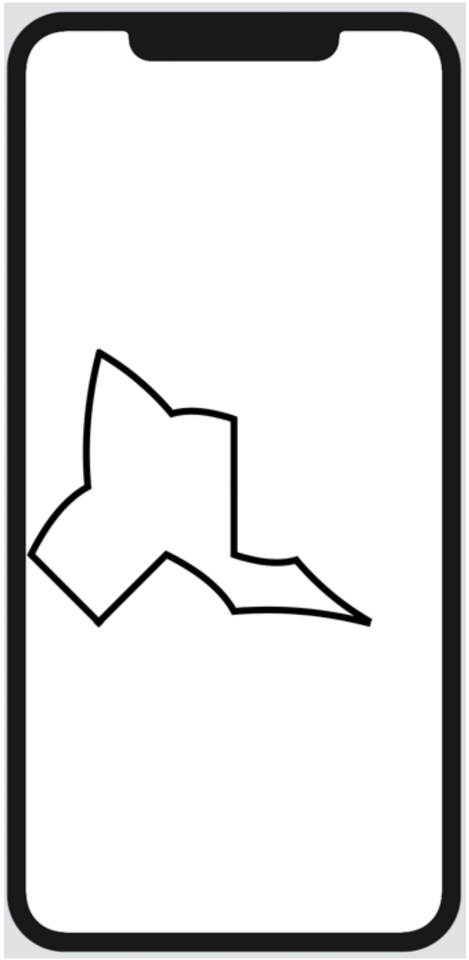


Rotate the other way and scale

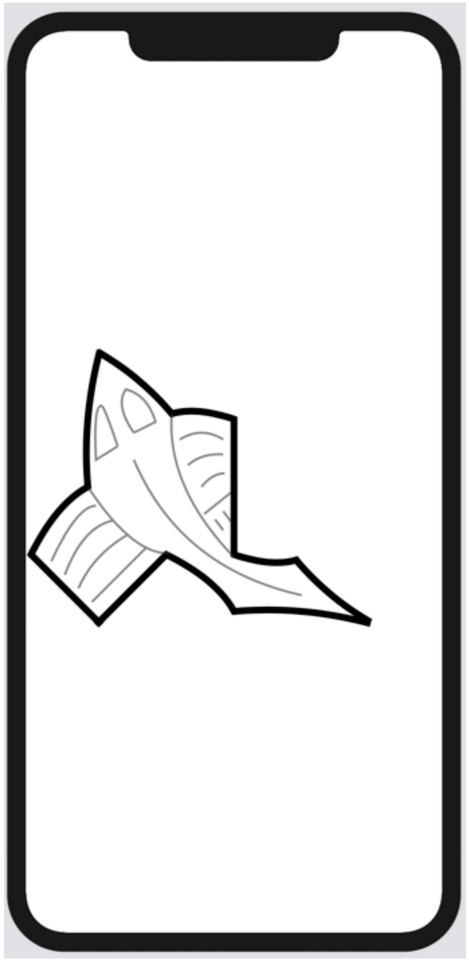


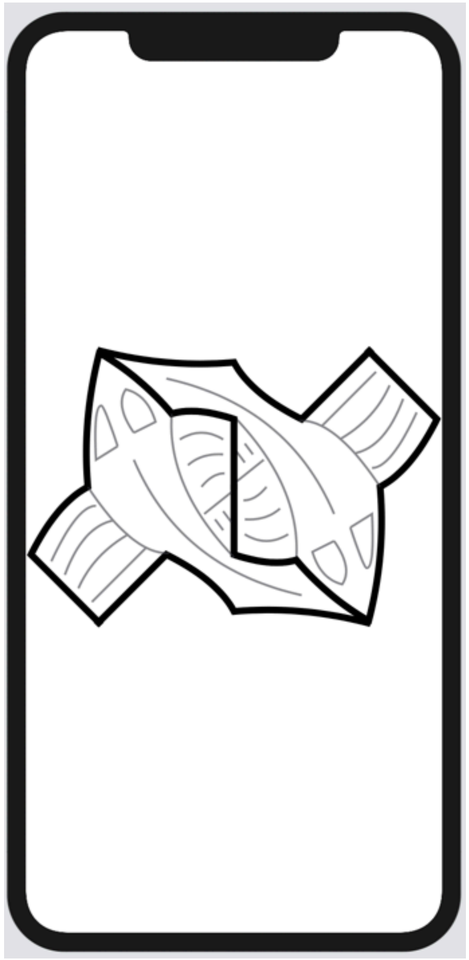
One more scaled copy

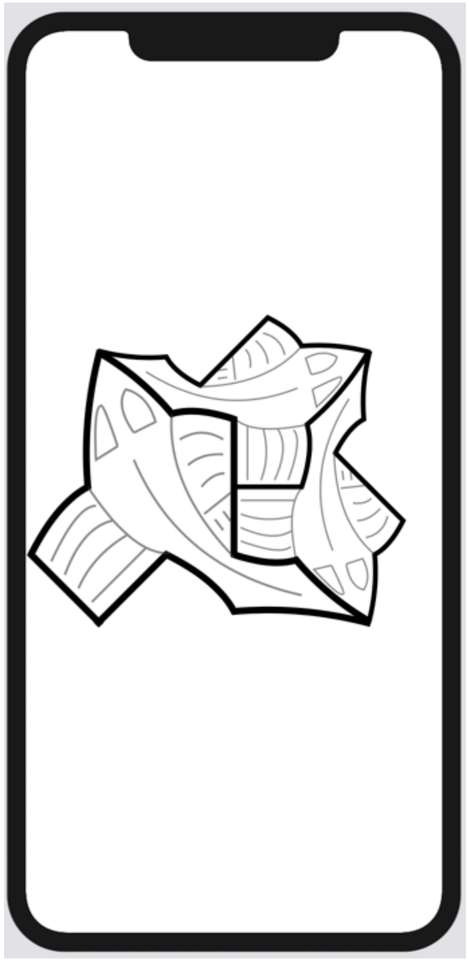


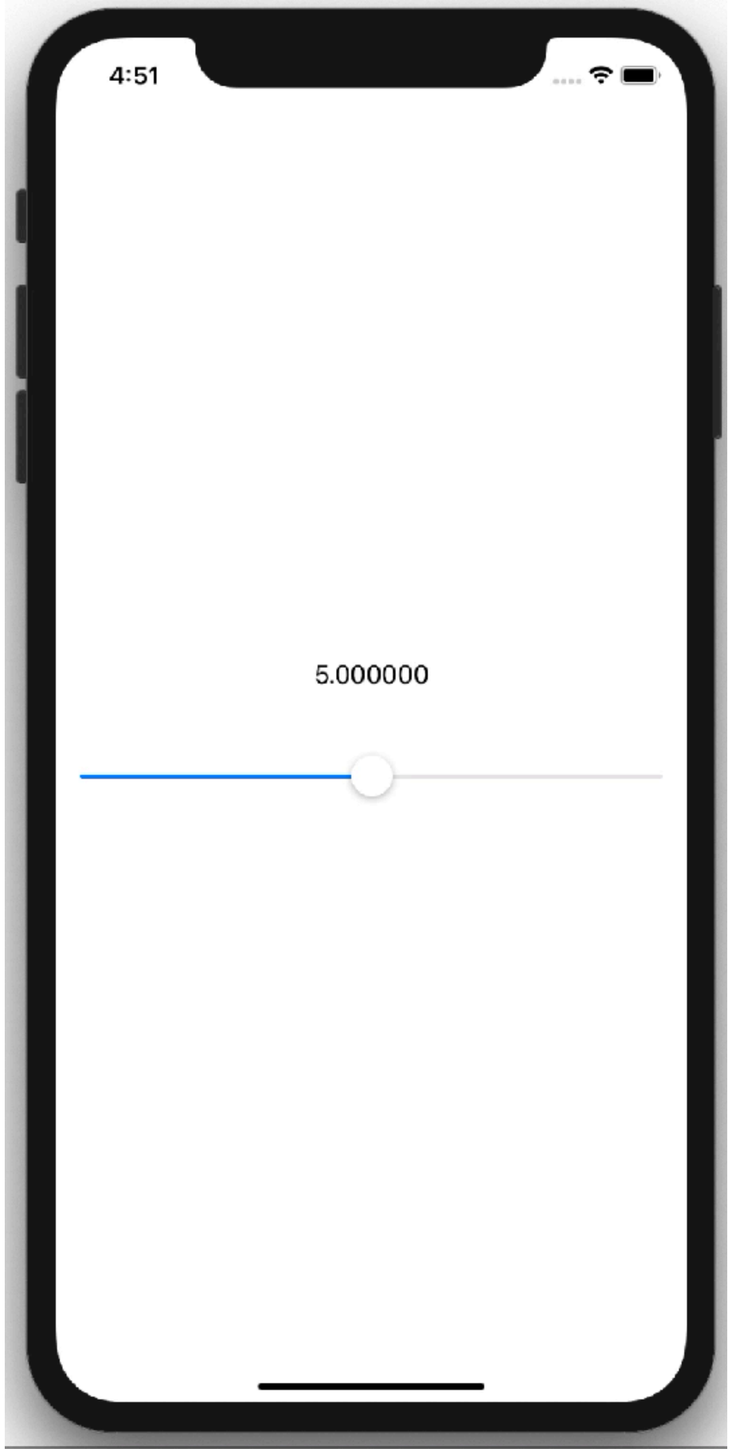


Now draw the rest of the %&$@ fish








4:51
5.000000

```
struct ContentView: View {


    var body: some View {
        VStack(alignment: .center) {
            Text("????")
                .padding()

            Slider(value: ????,
                   from: 0,
                   through: 10,
                   by: 0.00001)
                .padding()
        }
    }
}
```

```
struct ContentView: View {


    var body: some View {
        VStack(alignment: .center) {
            Text("????")
                .padding()

            Slider(value: ????,
                   from: 0,
                   through: 10,
                   by: 0.00001)
                .padding()
        }
    }
}
```

```
struct ContentView: View {


    var body: some View {
        VStack(alignment: .center) {
            Text("????")
                .padding()

            Slider(value: ????,
                   from: 0,
                   through: 10,
                   by: 0.00001)
                .padding()
        }
    }
}
```

```
struct ContentView: View {


    var body: some View {
        VStack(alignment: .center) {
            Text("????")
                .padding()

            Slider(value: ????,
                   from: 0,
                   through: 10,
                   by: 0.00001)
                .padding()
        }
    }
}
```

```
struct ContentView: View {


    var body: some View {
        VStack(alignment: .center) {
            Text("????")
                .padding()

            Slider(value: ????,
                   from: 0,
                   through: 10,
                   by: 0.00001)
                .padding()
        }
    }
}
```

```
struct ContentView: View {
    @State var value: Double = 5

    var body: some View {
        VStack(alignment: .center) {
            Text("????")
                .padding()

            Slider(value: ????,
                   from: 0,
                   through: 10,
                   by: 0.00001)
                .padding()
        }
    }
}
```

```
struct ContentView: View {
    @State var value: Double = 5

    var body: some View {
        VStack(alignment: .center) {
            Text("\(value)")
                .padding()

            Slider(value: $value,
                   from: 0,
                   through: 10,
                   by: 0.00001)
                .padding()
        }
    }
}
```

```
struct ContentView: View {
    @State var value: Double = 5

    var body: some View {
        VStack(alignment: .center) {
            Text("\(value)")
                .padding()

            Slider(value: $value,
                   from: 0,
                   through: 10,
                   by: 0.00001)
                .padding()
        }
    }
}
```

```
struct ContentView: View {
    @State var value: Double = 5

    var body: some View {
        VStack(alignment: .center) {
            Text("\(value)")
                .padding()

            Slider(value: $value,
                   from: 0,
                   through: 10,
                   by: 0.00001)
                .padding()
        }
    }
}
```

```
struct ContentView: View {
    @State var value: Double = 5

    var body: some View {
        VStack(alignment: .center) {
            Text("\(value)")
                .padding()

            Slider(value: $value,
                   from: 0,
                   through: 10,
                   by: 0.00001)
                .padding()
        }
    }
}
```

```
struct ContentView: View {
    @State var value: Double = 5

    var body: some View {
        VStack(alignment: .center) {
            Text("\(value)")
                .padding()

            Slider(value: $value,
                   from: 0,
                   through: 10,
                   by: 0.00001)
                .padding()
        }
    }
}
```

```
struct ContentView: View {
    @State var value: Double = 5

    var body: some View {
        VStack(alignment: .center) {
            Text("\(value)")
                .padding()

            Slider(value: $value,
                   from: 0,
                   through: 10,
                   by: 0.00001)
                .padding()
        }
    }
}
```

258 - Property Wrappers

```
@State
```

```
@propertyWrapper
struct ToTwoPlaces {
    private(set) var value: Double = 0

    private let multiplier = 100.0

    var wrappedValue: Double {
        get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double) {
         self.wrappedValue = initialValue
    }
}
```

```
@propertyWrapper
struct ToTwoPlaces {
    private(set) var value: Double = 0

    private let multiplier = 100.0

    var wrappedValue: Double {
        get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double) {
         self.wrappedValue = initialValue
    }
}
```

```
@propertyWrapper
struct ToTwoPlaces {
    private(set) var value: Double = 0

    private let multiplier = 100.0

    var wrappedValue: Double {
        get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double) {
         self.wrappedValue = initialValue
    }
}
```

```
@propertyWrapper
struct ToTwoPlaces {
    private(set) var value: Double = 0

    private let multiplier = 100.0

    var wrappedValue: Double {
        get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double) {
         self.wrappedValue = initialValue
    }
}
```

```
@propertyWrapper
struct ToTwoPlaces {
    private(set) var value: Double = 0

    private let multiplier = 100.0

    var wrappedValue: Double {
        get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double) {
         self.wrappedValue = initialValue
    }
}
```

```
@propertyWrapper
struct ToTwoPlaces {
    private(set) var value: Double = 0

    private let multiplier = 100.0

    var wrappedValue: Double {
        get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double) {
         self.wrappedValue = initialValue
    }
}
```

```
@propertyWrapper
struct ToTwoPlaces {
    private(set) var value: Double = 0

    private let multiplier = 100.0

    var wrappedValue: Double {
        get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double) {
          self.wrappedValue = initialValue
    }
}
```

```
@propertyWrapper
struct ToTwoPlaces {
    private(set) var value: Double = 0

    private let multiplier = 100.0

    var wrappedValue: Double {
        get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double) {
          self.wrappedValue = initialValue
    }
}
```

```
struct ContentView: View {
    @ToTwoPlaces var amount = 1.23456

    var body: some View {
        print(amount)
        return Text("\(amount)")
    }
}
```

```
struct ContentView: View {
    @ToTwoPlaces var amount = 1.23456

    var body: some View {
        print(amount)
        return Text("\(amount)")
    }
}
```

```
struct ContentView: View {
    @ToTwoPlaces var amount = 1.23456

    var body: some View {
        print(amount)
        return Text("\(amount)")
    }
}
```

1.23

```
struct ContentView: View {
    @ToTwoPlaces var amount = 1.23456

    var body: some View {
        print(amount)
        return Text("\(amount)")
    }
}
```

```
struct ContentView: View {
    @ToTwoPlaces var amount = 1.23456

    var body: some View {
        print(amount)
        return Text("\(amount)")
    }
}
```

Wait. Check this out.

```
struct ContentView: View {
    @ToTwoPlaces var amount = 1.23456

    var body: some View {
        print(amount)
        return Text("\(amount)")
    }
}
```

NSHipster

```
@propertyWrapper
struct ToTwoPlaces {
    private(set) var value: Double = 0

    private let multiplier = 100.0

    var wrappedValue: Double {
        get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double) {
         self.wrappedValue = initialValue
    }
}
```

```
@propertyWrapper
struct RoundedTo {
    private(set) var value: Double = 0
    let precision: Int

    var multiplier: Double {…}

    var wrappedValue: Double {
          get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double, _ precision: Int) {
        precondition(precision > 0)
        self.precision = precision
        self.wrappedValue = initialValue
    }
}
```

```
@propertyWrapper
struct RoundedTo {
    private(set) var value: Double = 0
    let precision: Int

    var multiplier: Double {…}

    var wrappedValue: Double {
          get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double, _ precision: Int) {
        precondition(precision > 0)
        self.precision = precision
        self.wrappedValue = initialValue
    }
}
```

```
@propertyWrapper
struct RoundedTo {
    private(set) var value: Double = 0
    let precision: Int

    var multiplier: Double {…}

    var wrappedValue: Double {
          get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double, _ precision: Int) {
        precondition(precision > 0)
        self.precision = precision
        self.wrappedValue = initialValue
    }
}
```

```
@propertyWrapper
struct RoundedTo {
    private(set) var value: Double = 0
    let precision: Int

    var multiplier: Double {…}

    var wrappedValue: Double {
          get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double, _ precision: Int) {
        precondition(precision > 0)
        self.precision = precision
        self.wrappedValue = initialValue
    }
}
```

```
var i = 1
for _ in 1...precision {
    i *= 10
}
return Double(i)
```

```
@propertyWrapper
struct RoundedTo {
    private(set) var value: Double = 0
    let precision: Int

    var multiplier: Double {…}

    var wrappedValue: Double {
          get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double, _ precision: Int) {
        precondition(precision > 0)
        self.precision = precision
        self.wrappedValue = initialValue
    }
}
```

```
@propertyWrapper
struct RoundedTo {
    private(set) var value: Double = 0
    let precision: Int

    var multiplier: Double {…}

    var wrappedValue: Double {
          get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double, _ precision: Int) {
        precondition(precision > 0)
        self.precision = precision
        self.wrappedValue = initialValue
    }
}
```

```
@propertyWrapper
struct RoundedTo {
    private(set) var value: Double = 0
    let precision: Int

    var multiplier: Double {…}

    var wrappedValue: Double {
          get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double, _ precision: Int) {
        precondition(precision > 0)
        self.precision = precision
        self.wrappedValue = initialValue
    }
}
```

```
@propertyWrapper
struct RoundedTo {
    private(set) var value: Double = 0
    let precision: Int

    var multiplier: Double {…}

    var wrappedValue: Double {
          get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double, _ precision: Int) {
        precondition(precision > 0)
        self.precision = precision
        self.wrappedValue = initialValue
    }
}
```

```
@propertyWrapper
struct RoundedTo {
    private(set) var value: Double = 0
    let precision: Int

    var multiplier: Double {…}

    var wrappedValue: Double {
          get {value}
        set {
            value = ((newValue * multiplier).rounded()) / multiplier
        }
    }

    init(initialValue: Double, _ precision: Int) {
        precondition(precision > 0)
        self.precision = precision
        self.wrappedValue = initialValue
    }
}
```

```
struct ContentView: View {
    @ToTwoPlaces var amount = 1.23456

    var body: some View {
        print(amount)
        return Text("\(amount)")
    }
}
```

```
struct ContentView: View {
    @RoundedTo(3) var amount = 1.23456

    var body: some View {
        print(amount)
        return Text("\(amount)")
    }
}
```

```
struct ContentView: View {
    @RoundedTo(3) var amount = 1.23456

    var body: some View {
        print(amount)
        return Text("\(amount)")
    }
}
```

```
struct ContentView: View {
    @RoundedTo(3) var amount = 1.23456

    var body: some View {
        print(amount)
        return Text("\(amount)")
    }
}
```

1.235

```
struct ContentView: View {
    @RoundedTo(3) var amount = 1.23456

    var body: some View {
        print(amount)
        return Text("\(amount)")
    }
}
```

What's New in Swift

GoTo

Copenhagen, Denmark
November, 2019

Daniel H Steinberg

dimsumthinking.com

goto; copenhagen

GOTO Copenhagen 2019

Conference Nov. 18 - 20

Click ‘Rate Session’ to rate session and ask questions.

Follow us @gotocph

goto; copenhagen

GOTO Copenhagen 2019

Conference Nov. 18 - 20

Please

Remember to rate this session

Thank you!

Follow us @gotocph

goto; copenhagen

GOTO Copenhagen 2019

Conference Nov. 18 - 20

Did you **remember to rate** the previous session ?

Follow us @gotocph

goto;
copenhagen

Follow us @gotocph