Fachhochschule
Nordwestschweiz

# iOS Modul
## ViewBuilder & Shape & Access Modifier

## Access Control

- Zugriff kann auf Typen gesteuert werden: calls, struct, enum.
- Ebenso Properties, Methoden, Initialisierer.
- Swift hat einen Default-Zugriffsschutz (Internal), der für die meisten Apps ausreicht.
- Wichtig wird das Thema vor allem bei Modulen. Z.B. bei der Entwicklung eines Frameworks.

| Bezeichner | Bedeutung |
|---|---|
| Public | Jeder kann lesen und schreiben |
| Internal | Code im gleichen Modul kann lesen und schreiben, andere Module nicht |
| File Private | Code in der gleichen Datei kann lesen und schreiben |
| Private | Zugriff nur durch Code im gleichen Typ oder Extension |

## Access Control

Beispiel für einen private Setter

```
struct TrackedString {
       private(set) var numberOfEdits = 0

       var value: String = "" {
          didSet {
            numberOfEdits += 1
          }
       }
}
```

## ViewBuilder

- Annotation, mit der Listen von Views erstellt werden können. Analog wie es z.B. im HStack möglich ist
- Kann auf Funktionen und computed properties angewendet werden, sofern diese eine View zurück geben
- Ergebnis ist dann eine TupleView mit 2 bis 10 Elementen
- Beispiel: TupleView<RoundedRectangle, RoundedRectangle, Text>

```
@ViewBuilder
func front(of card: Card) -> some View {
      RoundedRectangle(cornerRadius: 10)
      RoundedRectangle(cornerRadius: 10).stroke()
      Text(card.content)
}
```

## ViewBuilder

- ViewBuilder kann auch als Parameter verwendet werden, der eine View zurück gibt.
- In einer ViewBuilder-Funktion dürfen zur Steuerung nur if-else-Anweisungen verwendet werden.
  - Es können keine Variablen deklariert werden.
  - Durch Bedingungen entstehen Conditional-Views, d.h. es können auch EmptyViews entstehen.
- ViewBuilder werden vor allem dazu genutzt den Code zu strukturieren.

## Shape

- Protokoll, welches von View erbt.
- Beispiele: RoundedRect, Circle, Capsule, …
- Standardmässig werden Shapes mit der aktuellen Foreground Color gezeichnet
  - Dies kann mit .stroke() und .fill() angepasst werden

```
@func fill(_ whatToFillWith: S) -> View where S: ShapeStyle
```

  - S ist ein Generic, welches das ShapeStyle-Protokoll erfüllen muss:
    - Color, ImagePaint, AngularGradient, LinearGradient

## Shape

- Auch eigene Shapes sind möglich, indem das Shape-Protokoll implementiert wird
- Dabei muss die path-Methode implementiert werden

```
func path(in rect: CGRect) -> Path
```

- Mit Path ist das Zeichnen von beliebigen Grafiken möglich. (Siehe Apple Doku)
  - Es können Primitive wie Linien, BezierKurven, etc. verwendet werden um eine Shape zu erzeugen
- Mehr zur Shape in der Demo

## Animation

- Animationen in Apps sind relativ wichtig, damit sich die App natürlicher anfüllt und dem Nutzer visuelles Feedback gibt.
- SwiftUI lässt Animationen mit wenig Aufwand umsetzen.
- Animationen können durch Anpassung von Shapes und durch ViewModifiers umgesetzt werden.

## ViewModifier

- Funktionen, die Views anpassen. Z.B. aspectRatio() oder padding()
  - Rufen wiederum die Funktion modifier auf:

```
.modifier(AspectModifier(2/3))
```

  - AspectModifier folgt dem ViewModifier-Protokoll. Dieses hat eine Funktion.

```
protocol ViewModifier {
   associatedtype Content //Generic in protocol
   func body(content: Content) -> some View
}
```

  - Content ist die View, die modifiziert werden soll. Die body-Funktion passt die View entsprechend an.

## ViewModifier

Beispiel: Gesucht ist ein ViewModifier, der aus einer View eine Spielkarte macht, d.h. eine View mit Vorder und Rückseite.

```
Text("😛").modifier(Cardify(isFaceUp: true))
struct Cardify: ViewModifier {
       var isFaceUp: Bool
       func body(content: Content) -> some View {
       ZStack {
         if isFaceUp {
           RoundedRectangle(cornerRadius: 10).fill(Color.white)
           RoundedRectangle(cornerRadius: 10).stroke()
           content
}
else { RoundedRectangle(cornerRadius: 10) } } } }
```

## ViewModifier

Anstelle von `Text("😛").modifier(Cardify(isFaceUp: true))`

Soll jedoch diese Syntax genutzt werden können: `Text("😛").cardify(isFaceUp: true)`

Dies lässt sich sehr einfach durch eine Extension umsetzen:

```
extension View {
        func cardify(isFaceUp: Bool) -> some View {
          return self.modifier(Cardify(isFaceUp: isFaceUp))
        }
}
```