COMP 433 Exam 1

Basic View and Layout

Stacks

- **HStack** → Arranges elements from left to right in a horizontal way
- **VStack** → Arranges elements from top to bottom in a vertical way
- **ZStack** → Arranges elements along Z-Axis

We can nest these stacks together and they can also hold **TWO** parameters: *alignment:* & *spacing:*

View Modifiers

Stack Alignment

For HStack:

- **.leading** → Starting from the left most side
- **.center** → Starting from the center
- **.trailing** → Starting from the right most side

For VStack:

- **.top** → Starting from the top
- **.center** → Starting from the center
- **.bottom** → Starting from the bottom

Spacing

Changes the amount of space between the various number of elements

Spacer()

- Evenly spaces out elements to as tall or wide as they can, pushing other elements away from each other

Backgrounds, Borders, Padding, etc.

These are typically AFTER an element where order MATTERS

- **.background** → Gives a background of a color for a text or object
- **.border** → Gives a border based on a given color
- **.padding** → Gives padding around an object

```
Text("example")
    .background(Color.green)
    .padding(16)
// This is incorrect and will give the
    ↪ padding OUTSIDE the background
```

We can also wrap a modifier around an entire stack

Basics and General Syntax of Swift

Declaring Variables

- **Let** → A constant variable
- **Var** → A variable that will change

Syntax

Most things do not need parenthesis such as if statements

Switch cases example:

```
switch errorCodes {
    case 100 ..< 200: // from 100 to less
        ↪  than 200
        print("Informational")
    case 200 ..< 300:
        print("Success!")
    ...
}
```

Inputting a variable in a print:

```
Text("Hello my name is: \(name)")
```

For functions, we must specify the parameters:

```
func greet(name: String, hometown: String
    ↪ ) {...}
greet(name: "me", hometown: "Asheville")
```

Moreover, we can omit external names with _ and have different external and internal names for readability:

```
func divide(_ a: Double, by b: Double) ->
    ↪  Double {...}
let result = divide(4.0, by: 2.0)
```

Structs vs Classes

They both share many characteristics: properties, methods, initializers, etc.

However, they do have their differences :

- **Struct** → values types
- **Class** → reference types

This difference in typings mean that if you create a variable that references a class, then it will automatically change the reference values, like a pointer!

However, with structs, when you declare a variable that is set to a struct object, it will make a **COPY**

Interactive UIs

Buttons

Two parameters: action and label

```
Button { print("Pressed!") } label: { "
    ↪ Press Me!" } // 1 example that we
    ↪ did in class

Button("Press me", systemImage: "scope")
    ↪ { print("Pressed!")} // similar
    ↪ but icon and dif. syntax
```

@State and @Binding

- **@State** → Declares that this property is part of app's state, a dynamic information
- **@Binding** → A proxy to @State, writing to this variable will UPDATE @State variable

Computed Properties

Usually within struct, always up to date as it takes from fields directly

Lists

```
List(restaurants, id: \.self ) {
    ↪ restaurant in }
```

- **KeyPaths** → gives instructions on how to find a property
- **IDs** → we can also give whole structs ids, by giving it a UUID(), thus we can pass id to a list
- **Identifiable** → or we can give it the identifiable type for a struct and we can simply do List(courses)

Moreover, we can use the $ sign to bind it with a @State property!

Navigation

Conceptually, they are very similar to the data structure: Stacks, where we pop and push

```
NavigationStack {
    NavigationLink {
        Text("Destination view!")
    } label: { Text("Go to destination!")
        ↪  }
}
```

We can add levels but we only need ONE nav. stack