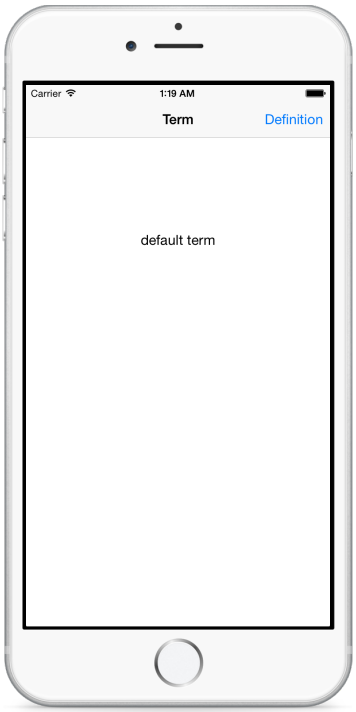
Flashcards
Lesson 3

Description

Add a `Deck` model to the project that encapsulates a collection of `Flashcard` objects.



Learning Outcomes

- Practice defining models as Swift classes with properties.
- Describe the purpose of initializers and initialization best practices.
- Describe the structure and behavior of dictionaries.
- Practice using a Swift `for-in` loop to iterate over a collection of objects.
- Practice expressing different forms of closure expressions with `map`.

Vocabulary

model	property	initialization
dictionary	key-value	dictionary literal
`for-in`	transformation	`map`
closure expression	type annotation	

Materials

- **Flashcards Lesson 3** Xcode project
- **Initialization** presentation

Opening

How might we model a collection of flashcards?

Agenda

- Discuss the need for a `Deck` model, representing a collection of `Flashcard` objects.
- Add a new (⌘N) `Deck` class to the project.

```
class Deck {

}
```

- Discuss how the `Deck` model will manage a collection of `Flashcard` objects, but the controller will use methods to "ask" a `Deck` for a card, rather than accessing the collection of `Flashcard` objects directly.
- Add a private `[Flashcard]` property to the `Deck` class.

```
private var cards = [Flashcard]()
```

- Discuss why the `cards` property is private, to hide how the `Deck` class manages the collection of `Flashcard` objects.
- Discuss how initializing a `Deck` should fill the `cards` array with a collection of `Flashcard` objects.
- Implement the `Deck` initializer, using a dictionary of term-definition pairs for `Flashcard` objects.

```
init() {
   let cardData = [
      "controller outlet" : "A controller view property, marked with
         IBOutlet.",
      "controller action": "A controller method, marked with IBAction,
         that is triggered by an interface event."
   ]
   for (term, definition) in cardData {
      cards.append(Flashcard(term: term, definition: definition))
   }
}
```

- Present the concept of initialization, if necessary.
- Explain how the initializer creates a dictionary of term-definition pairs using the Swift dictionary literal syntax.

- Explain the Swift `for-in` loop, and how each key-value pair in the `cardData` dictionary is assigned to the implicit `term` and `definition` constants for each iteration of the loop.
- Discuss how, for each iteration of the `for-in` loop, the `term` and `definition` values are used to instantiate a `Flashcard` object, which is then appended to the `cards` array.
- Discuss how the initializer is transforming an array of flashcard data into an array of `Flashcard` objects, which is an opportunity for using `map`.
- Replace the `for-in` loop with a verbose call of `map`.

```
cards = cardData.map(
   { (term: String, definition: String) -> Flashcard in
      return Flashcard(term: term, definition: definition)
   })
```

- Explain how the `map` function is passed a closure expression; and how `map` invokes the closure for each key-value pair in the dictionary, builds an array with each returned `Flashcard` object, and assigns the resulting array to the `cards` property.
- Explain how Swift can infer the type of the closure expression from the data type of the `cardData` dictionary and the `cards` array.
- Refactor the `map` call, removing the explicit type annotations.

```
cards = cardData.map( { term, definition in
   return Flashcard(term: term, definition: definition)
})
```

- Explain that, because the closure expression only contains one statement, Swift also infers an implicit `return`.
- Refactor the `map` call, removing the explicit `return`.

```
cards = cardData.map( { term, definition in
   Flashcard(term: term, definition: definition)
})
```

- Explain that, because the closure expression is the last argument to `map`, we can use the Swift trailing closure expression syntax; and explain how Swift provides shorthand argument names, removing the need for the explicit `term` and `definition` arguments.
- Refactor the `map` call, using a trailing closure expression and shorthand argument names.

```
cards = cardData.map { Flashcard(term: $0, definition: $1) }
```

- Discuss how the succinct `map` call compares to the `for-in` loop.

- Discuss how, because the initializer no longer appends `Flashcard` objects to the mutable `cards` array property, the property can now be constant.
- Modify the cards property declaration to a constant, without a default value.

```
private let cards: [Flashcard]
```

- Discuss how the `cards` property declaration no longer instantiates an empty `[Flashcard]` array, since the initializer uses `map` to assign the property its `[Flashcard]` value.

Closing

What are the differences between an array and a dictionary?

Modifications and Extensions

- Investigate how to store dictionaries of String values inside a property list (**.plist**) file. Store a collection of flashcard terms and definitions in a property list, so the values are not explicitly hard-coded in the `Deck` initializer. Think about how the app can load the data once, and pass the data as a dictionary to the `Deck` initializer.

Resources

Cocoa Core Competencies: Model Object https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/ModelObject.html

The Swift Programming Language: Classes and Structures https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ClassesAndStructures.html

The Swift Programming Language: Properties https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Properties.html

The Swift Programming Language: Initialization https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Initialization.html

The Swift Programming Language: Collection Types https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/CollectionTypes.html

The Swift Programming Language: Control Flow https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ControlFlow.html

The Swift Programming Language: Closures https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Closures.html