TP 1 - Swift

Adrien Humilière 15/02/2019

Download and unarchive `SpaceAdventure.zip`, base project for this lab.

Part 1

- Using the Project Navigator (⌘+1), open main.swift.
- Swift programs generally contain a main.swift file, which contains code for the starting point, or "main entry point," of a Swift program.
- Run the program (⌘+R), and observe the console (⇧+⌘+C) to see the program's output.
- Declare two variables.

```
var numberOfPlanets: Int = 8
var diameterOfEarth: Float = 24859.82 // In miles, from pole to pole
```

- Swift single-line comments begin with `//`.
- Remove the printing of `Hello World!` and add some of your own `print` calls below the variable declarations.

```
print("Welcome to our solar system!")
print("There are \(numberOfPlanets) planets to explore.")
print("You are currently on Earth, which has a circumference of \(diameterOfEarth) miles.")
```

- Run the program, and observe the console output.
- Remove the type annotations from the two variable declarations.

```
var numberOfPlanets = 8
var diameterOfEarth = 24859.82 // In miles, from pole to pole
```

- Run the program, and observe how the program works the same.
- The values of `numberOfPlanets` and `diameterOfEarth` do not change while the program is running.

- Change the variable declarations to constant declarations.
- Run the program, and observe how the program works the same.
- You should always start with `let`, and fallback to `var` if needed.

Part 2

- We need to ask the user their name, to capture what they type, and to print it back on the console.
- Implement an idiomatic approach to capturing console input from the user with a provided utility function, `getln`.

```
1 print("What is your name?")
2 let name = getln()
3 print("Nice to meet you, \(name). My name is Eliza, I'm an ↩
    old friend of Siri.")
```

- Unlike `print`, which is part of the Swift Standard Library, the `getln` function is a "helper" function provided as a convenience with this particular Xcode project.
- Using the Project Navigator, locate and select the HelperFunctions.swift file.
- Xcode will compile all of the Swift source files within the Xcode project before running the application.
- `getln` function retrieves keyboard input from the console, and returns what the user has typed as a `String` value.
- Run the program, interact with the console, and observe the output.
- We could suggest an adventure, and ask the traveler if he or she would like the program to choose a random planet to visit.

```
1 print("Let's go on an adventure!")
2 print("Shall I randomly choose a planet for you to visit? (Y ↩
    or N)")
3 let decision = getln()
```

- We need the program to make a decision on what to do, based on what the traveler types, stored in the constant `decision`.
- Implement a decision using an `if` statement and an `else` clause.

```
if decision == "Y" {
    print("Ok! Traveling to...")
    // TODO: travel to random planet
} else {
    print("Ok, name the planet you would like to visit...")
    // TODO: let the user select a planet to visit
}
```

- Run the program, interact with the console, and enter `Y` or `N` to observe the respective output.

Part 3

- We need to ask the traveler if he wants to visit a random planet, and to prompt for another answer "as long as the traveler does not answer `Y` or `N`."
- Modify the existing user input capturing and decision making to leverage a while loop to carry out the repetitive task of waiting for the user to type `Y` or `N`.
- Run the program, interact with the console, try some arbitrary input, and observe that the program continues to prompt until `Y` or `N` is entered.

Part 4

- We need to model what happens during the space adventure, such as greeting the travelers, asking them what planets they want to travel to, and then traveling to the planets.
- Add a new Swift file (⌘+N) called SpaceAdventure.swift. Be sure that the SpaceAdventure group is selected, and that the SpaceAdventure target is checked.
- Explain the convention of using an individual file to contain a single class definition, and how the file name (SpaceAdventure.swift) alludes to the name of the class it contains.
- At a high level of thinking, the code in main.swift should has just two jobs: to create a `SpaceAdventure` object, and to start the adventure.
- Above the existing code within main.swift, instantiate a `SpaceAdventure` object.

```
import Foundation

let adventure = SpaceAdventure()

let numberOfPlanets = 8
...
```

- Observe the errors in the Xcode editor.
- We now have to write the `SpaceAdventure` class definition. Using the Project Navigator, select SpaceAdventure.swift and implement a basic class definition.
- Return to main.swift, and observe that the error notice disappears.
- We might call a method upon a `SpaceAdventure` object, telling it to `start`. Add a method call using the `SpaceAdventure` object.

```
1 adventure.start()
```

- Observe the error notice in the Xcode editor.

Part 5

- `SpaceAdventure` object does not know how to handle the `start` method call.
- Add an empty implementation of the `start` method to the `SpaceAdventure` class.
- Return to main.swift, and observe how the Xcode error notices disappear.
- Cut and paste the existing code from main.swift into the body of the `SpaceAdventure` `start` method implementation.
- Run the program, and interact with the console to demonstrate that the existing functionality remains intact.
- main.swift now only creates a `SpaceAdventure` object, and tells the `SpaceAdventure` object to `start`.

Part 6

- The `start` method seems to do three things: print an introduction, greet the user, and determine which planet to travel to.
- Extract the first few lines of `start` into a new private method called `displayIntroduction`. Replace the extracted code with a method call at the beginning of `start`.
  The `displayIntroduction` will only be called by the `start` method, and is marked `private` to indicate that only code within the same file will be able to call `displayIntroduction`.
- The `start` method uses a pair of print and `getln` methods twice, to prompt for and capture user input. Encapsulate the work of prompting for and capturing user input into a private method called `responseToPrompt`.
- Replace the relevant lines of code in `start` to use the new `responseToPrompt` method.
- Extract the greeting-related code in `start` into a new method called `greetAdventurer`.

- Extract the remaining code in `start` into a new method called `determineDestination`. Update the `start` method to call the new `greetAdventurer` and `determineDestination` methods.
- Run the program and confirm that the functionality remains unchanged.

Part 7

- We need to model a collection of planets, using a `PlanetarySystem` class.
- Add a new Swift file called PlanetarySystem.swift to the project.
- Using the Project Navigator, select PlanetarySystem.swift and implement a basic `PlanetarySystem` class definition.
- Add a property declaration to the `PlanetarySystem` class to represent the name of the planetary system. Swift requires that all constant properties be assigned values during instantiation, within the implementation of an initializer.
- Add a parameterized initializer to the `PlanetarySystem` class.
- SpaceAdventure should consist of a `PlanetarySystem` to travel within, we need to add a `PlanetarySystem` property to the `SpaceAdventure` class.
- Add the new `PlanetarySystem` property to the `SpaceAdventure` class.

```
1 class SpaceAdventure {
2     let planetarySystem = PlanetarySystem(name: "Solar System")
3 ...
```

- Update the implementation of displayIntroduction, removing some previous demonstration code, and using the `PlanetarySystem` name to display the introductory message.
- Discuss how name is a property of a `PlanetarySystem` object, and how `planetarySystem` is a property of a `SpaceAdventure` object.
- Run the program, and observe how the console output reflects the name of the planetary system.

Part 8

- Configure the Planetary System with a list of planets (in the initializer). Use this list of planets for the random destination.
- To go further, make it possible for the user to choose his Planetary System before choosing his destination planet.