David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan) (https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/) (https://www.quora.com/profile/David-J-Malan) (https://twitter.com/davidjmalan)

Pokédex

Distribution Code

Download this project's distribution code (https://cdn.cs50.net/2019/fall/tracks/ios/pokedex/pokedex.zip).

What To Do

- Searching
- Catching
- Saving State
- Sprites
- Description

Searching

Let's add some new functionality to our Pokédex app! First, let's give users the ability to search the Pokédex for their favorite Pokémon.

First, add a search bar to the top of our `UITableView`. Open up your Storyboard, and then add a new UI element. Search for `search bar`, and then drag the search bar onto your `UITableView`, just above where it says `Prototype Cells`. You should now see a search bar above the `UITableView` —nice!

Next, create an `@IBOutlet`, and wire it up, as you did with other UI elements.

Now, let's implement the search functionality. We want to search over Pokémon names as the user types, and only display the ones that match. Specifically, your search should match any substring in a Pokémon's name. For instance, the search "saur" should match Bulbasaur, Ivysaur, and Venusaur; the search "v" should only match Ivysaur and Venusaur; an empty search should match everything.

To get started, you'll want to make your `PokemonListViewController` inherit from `UISearchBarDelegate`. Recall that *delegation* is one way for objects to pass information between each other. In this case, whenever the text changes in the search bar, we'd like to delegate that to our `PokemonListViewController`.

So, your class declaration should look something like this:

```
class PokemonListViewController: UITableViewController, UISearchBarDelegate {
```

Next, we want to set the `delegate` property of our search bar to be the instance of our `PokemonListViewController`. You can use the `viewDidLoad` method to do so. Assuming you called your search bar variable `searchBar`, you can do something like this:

```
override func viewDidLoad() {
  super.viewDidLoad()
  searchBar.delegate = self
}
```

Now, whenever the user changes the text in the search bar, the below method will be called:

```
func searchBar(_ searchBar: UISearchBar, textDidChange searchText: String) {
```

```
func searchBar(_ searchBar: UISearchBar, textDidChange searchText: String) {
  // implement your search here!
}
```

We can implement our search inside of this method. (Don't forget to call `tableView.reloadData` !) You probably want to store the results of the search in another class variable (whose type is a list of `PokemonListResult` s), so you don't lose your copy of the list containing all Pokémon (i.e., the `pokemon` variable). Then, rather than using `pokemon` inside of methods like `tableView( **_** tableView: UITableView, numberOfRowsInSection section: Int)` and `tableView( **_** tableView: UITableView, cellForRowAt indexPath: IndexPath)` , use your newly created variable, which should only contain the matching Pokémon.

Catching

Any good Pokédex keeps track of which Pokémon have been caught and which haven't. Let's add that functionality to our Pokédex as well.

First, add a button (a `UIButton` to be specific) to `PokemonViewController` storyboard. You can set the text of this button to whatever you'd like, but we'll go with `Catch` for simplicity. You'll also want to create a corresponding `@IBOutlet` and wire it up.

Next, create a method that will be called when the button is tapped. We'll need to mark this method as an `@IBAction` so that we can use it in our storyboard. Let's call it `toggleCatch` :

```
@IBAction func toggleCatch() {
  // gotta catch 'em all!
}
```

To wire up this `@IBAction` , head back to your storyboard, click on your button, hold Control, and click and drag from the button to `Pokemon View Controller` on the left. When you unclick, you should see a popover with `toggleCatch` under `Sent Events` . Click on that, and this method will be called each time the button is pressed.

Now, we can implement catching. To start, add a new boolean class variable that keeps track of whether or not the Pokémon is caught. If a Pokémon is caught, change the text of the button to something like `Release` , and vice-versa when it's released. The `UIButton` method `setTitle(_ title: String?, for: UIControl.State)` method will come in handy.

Saving State

You'll notice that if you stop running your app and then run it again, your Pokédex will forget which Pokémon are caught and which aren't! Let's fix that by saving that state to disk.

As your last task, use the `UserDefaults` class to save which Pokémon are caught. With this class, you can store state that will be remembered each time your app launches, which is just what you need. How you store this state is up to you—you might consider storing a list of all Pokémon that are caught, or you might consider using a map from Pokémon to boolean values.

Here's an example:

```
UserDefaults.standard.set("cs50", forKey: "course")
let course = UserDefaults.standard.string(forKey: "course")
print(course) // will print "cs50"
```

To test saving state, you should be able to catch a Pokémon, stop the simulator, start the simulator again, and still see that Pokémon as caught.

Sprites

Every Pokémon aficionado has noticed by now that our Pokédex doesn't yet have arguably its most important feature: the ability to display what each Pokémon looks like! Luckily for us, the API we chose contains links to images for each Pokémon.

Let's add that functionality to our app. First, add a new `UIImageView` to your storyboard, and wire it up to an `@IBOutlet` .

Next, when parsing the response from the API call, take a look at the key called `sprites` . You'll notice that it's a dictionary, and the key `front_default` contains a URL pointing to an image of a Pokémon. Use the value of that key to load in an image to your `UIImageView` . You'll want to follow a similar pattern—create structs with types that match the keys and types returned by the API.

Here are a few methods you might find useful:

- `UIImage(data: Data)`
- `Data(contentsOf: URL)`
- `URL(string: String)`

- `URL(string: String)`

Description

Let’s add one last feature to our Pokédex: a description of each Pokémon. From the API documentation, we can see that we can use /api/v2/pokemon-species/{id} to retrieve a description for a given Pokémon: https://pokeapi.co/docs/v2.html#pokemon-species (https://pokeapi.co/docs/v2.html#pokemon-species). For instance, the URL `https://pokeapi.co/api/v2/pokemon-species/133/` will give you the description text for everyone’s favorite Pokémon.

Specifically, what we’re looking for can be found in the key called `flavor_text_entries`. This key happens to contain entries for several different languages, but we’re just concerned with English for now. You might need a few additional structs to model the data for these new keys.

After a user selects a Pokémon from the list, make a separate API call to this second endpoint to retrieve the description of the selected Pokémon. Filter for just the first English description, and then display it somewhere on the screen. (Some Pokémon have more than one English description, and it suffices to just display the first one.) You’ll probably want to wire up a new `UILabel` or `UITextView` to display this final piece of data.

How to Submit

To submit your code with `submit50`, you may either: (1) upload your code to CS50 IDE and run `submit50` from inside of your IDE, or (2) install `submit50` on your own computer by running `pip3 install submit50` (assuming you have Python 3 (https://www.python.org/downloads/) installed).

Execute the below, logging in with your GitHub username and password when prompted. For security, you’ll see asterisks (`*`) instead of the actual characters in your password.

```
submit50 cs50/problems/2020/x/tracks/ios/pokedex
```