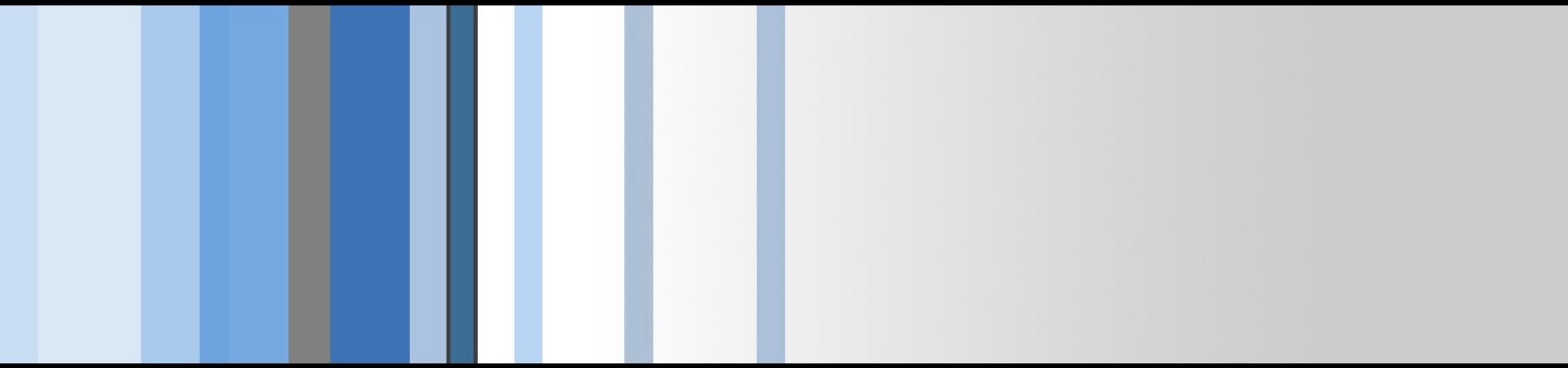
Class 13

CSC 496: iOS App Development
Swift Fundamentals: Fetching Data from APIs

Si Chen (schen@wcupa.edu)



```swift
let calculateDamage: (Int, Float) -> Int = { attackPower, defenseFactor in
    let damage = Float(attackPower) * (1 - defenseFactor)
    return Int(damage)
}

// Usage:
let damageDealt = calculateDamage(100, 0.25)
print(damageDealt)  // Output: 75
```

```swift
let checkLevelUp: (Int, Int) -> Bool = { currentLevel, currentXP in
    let xpNeeded = currentLevel * 100
    return currentXP >= xpNeeded
}

// Usage:
let didLevelUp = checkLevelUp(2, 250)
print(didLevelUp)  // Output: true
```

```
class ScoreTracker {
    var highestScore = 0
}

let scoreTracker = ScoreTracker()

let trackHighScore: (Int) -> Int = { newScore in
    if newScore > scoreTracker.highestScore {
        scoreTracker.highestScore = newScore
    }
    return scoreTracker.highestScore
}
```

Fetching data from APIs

- Most iOS app needs to interact with the internet, whether it's fetching images from a server, communicating with a database, or accessing various services. You do all of these via APIs (Application Programming Interfaces)

API Example: https://pokemon.wcpc.fun/id/1


```
{"base_experience":64,"base_happiness":70,"capture_rate":45,"color_id":5,"conquest_order":null,"evolution_chai
ender_differences":0,"hatch_counter":20,"height":7,"id":1,"identifier":"bulbasaur","is_baby":0,"is_default":1,
```


URL and URLComponents

- A URL (Uniform Resource Locator) is basically the address of a particular resource on the internet.
- In Swift, we have the **URL** and **URLComponents** classes that let's us work with URLs.
- **URL** is straightforward, and you typically use it to create a URL from a String like so:

```
let url = URL(string: "https://someapi.com/data")
```

- **URLComponents**, however, is more flexible. It represents the components of a URL and allows you to construct and manipulate URLs more granularly.

```
var urlComponents = URLComponents()
urlComponents.scheme = "https"
urlComponents.host = "api.swaggerhub.com"
urlComponents.path = "/apis/swagger-api/school/1.0.0"

let url = urlComponents.url
print(url!)
```

URL and URLComponents

```
var urlComponents = URLComponents()
urlComponents.scheme = "https"
urlComponents.host = "api.swaggerhub.com"
urlComponents.path = "/apis/swagger-api/school/1.0.0"

let url = urlComponents.url
print(url!)
```

Note: In the Swift programming language, the ! symbol is used for force-unwrapping an optional value.

However, if urlComponents.url is nil, attempting to force-unwrap it will result in a runtime error. Typically, using optional binding or other safer methods of unwrapping is a better choice. For instance, you could do the following:

```
if let url = urlComponents.url {
    print(url)
} else {
    print("Invalid URL components")
}
```

URLSession

- **URLSession** is Swift's primary API for networking. With it, you can send and receive data, upload and download files, and do much more. Here's how you can fetch data from a URL:

```
let session = URLSession.shared
let task = session.dataTask(with: url!) { (data, response, error) in
    if let error = error {
        print("Error: \(error)")
    } else if let data = data {
        let str = String(data: data, encoding: .utf8)
        print("Received data:\n\(str!)")
    }
}
task.resume()
```

HTTP Methods (GET, POST, PUT, DELETE)

- HTTP methods define what action we want to perform to the resource. The most common methods you'll interact with are GET, POST, PUT, and DELETE.
  - **GET**: To fetch data.
  - **POST**: To send data.
  - **PUT**: To update existing data.
  - **DELETE**: To remove data.
- You can specify the HTTP method of your request like so:

```
var request = URLRequest(url: url!)
request.httpMethod = "POST" // or GET, PUT, DELETE
```

HTTP Status Codes

```
let session = URLSession.shared
let task = session.dataTask(with: url!) { (data, response, error) in
    if let error = error {
        print("Error: \(error)")
    } else if let httpResponse = response as? HTTPURLResponse {
        print("HTTP Status Code: \(httpResponse.statusCode)")
        if let data = data {
            let str = String(data: data, encoding: .utf8)
            print("Received data:\n\(str!)")
        }
    }
}
task.resume()
```

- HTTP status codes are three-digit numbers returned by servers to indicate the status of a web activity. These status codes are divided into five classes:
  - 2xx (Success): The action was received, understood and accepted.
  - 3xx (Redirection): Further action must be taken to complete the request.
  - 4xx (Client Error): The request contains bad syntax or cannot be fulfilled.
  - 5xx (Server Error): The server failed to fulfill a seemingly valid request.
- For example, a commonly seen status code is 200, which means the request has succeeded, or 404, which means the requested resource could not be found.

JSON

- **JSON** (JavaScript Object Notation) is a lightweight data-interchange format.
  - Other options: XML, YAML,…
- JSON is built on two structures:
  - A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
  - An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

```
{
    "title": "Design Patterns",
    "subtitle": "Elements of Reusable Object-Oriented Software",
    "author": [
        "Erich Gamma",
        "Richard Helm",
        "Ralph Johnson",
        "John Vlissides"
    ],
    "year": 2009,
    "weight": 1.8,
    "hardcover": true,
    "publisher": {
        "Company": "Pearson Education",
        "Country": "India"
    },
    "website": null
}
```

Parsing JSON with Codable

- To parse JSON in Swift, we'd use something called 'Codable'. It's a type alias for the Decodable & Encodable protocols.
- So when something is Codable, that means it can be encoded to or decoded from a JSON structure. Here's a simple example:

```
struct User: Codable {
    var name: String
    var email: String
}

let data = ...// some JSON data
let decoder = JSONDecoder()

do {
    let user = try decoder.decode(User.self, from: data)
    print(user.name)
} catch {
    print(error)
}
```

How to fetch and parse data from the API with Swift

- 1. Build a data model (based on the structure of the JSON)

```
import Foundation

struct PokemonData: Decodable{
  //
      {"base_experience":270,"base_happiness":100,"capture_rate":45,"color_id":6
      ,"conquest_order":null,"evolution_chain_id":78,"evolves_from_species_id":null
      ,"forms_switchable":0,"gender_rate":-1,"generation_id":1,"growth_rate_id":4
      ,"habitat_id":5,"has_gender_differences":0,"hatch_counter":120,"height":4
      ,"id":151,"id:1":151,"identifier":"mew","is_baby":0,"is_default":1,"name":"Mew"
      ,"order":182,"order:1":182,"shape_id":6,"species_id":151,"weight":40}

    var name: String

}
```

How to fetch and parse data from the API with Swift

- 2. Create a Class to fetch API Data and decode it based on the model

```
// Method to fetch Pokémon data from API.
// completionHandler is called when data is successfully fetched and decoded.
private func fetchAPIData(completionHandler: @escaping (PokemonData) -> Void, pokemonID: Int) {
    let url = URL(string: "https://pokemon.wcpc.fun/id/\(pokemonID)")!

    URLSession.shared.dataTask(with: url) { (data, response, error) in
        guard let data = data else { return }

        do {
            let pokemonData = try JSONDecoder().decode(PokemonData.self, from: data)

            // Move to the main thread
            DispatchQueue.main.async {
                completionHandler(pokemonData)
            }
        } catch {
            print(error.localizedDescription)
        }
    }.resume()
}
```

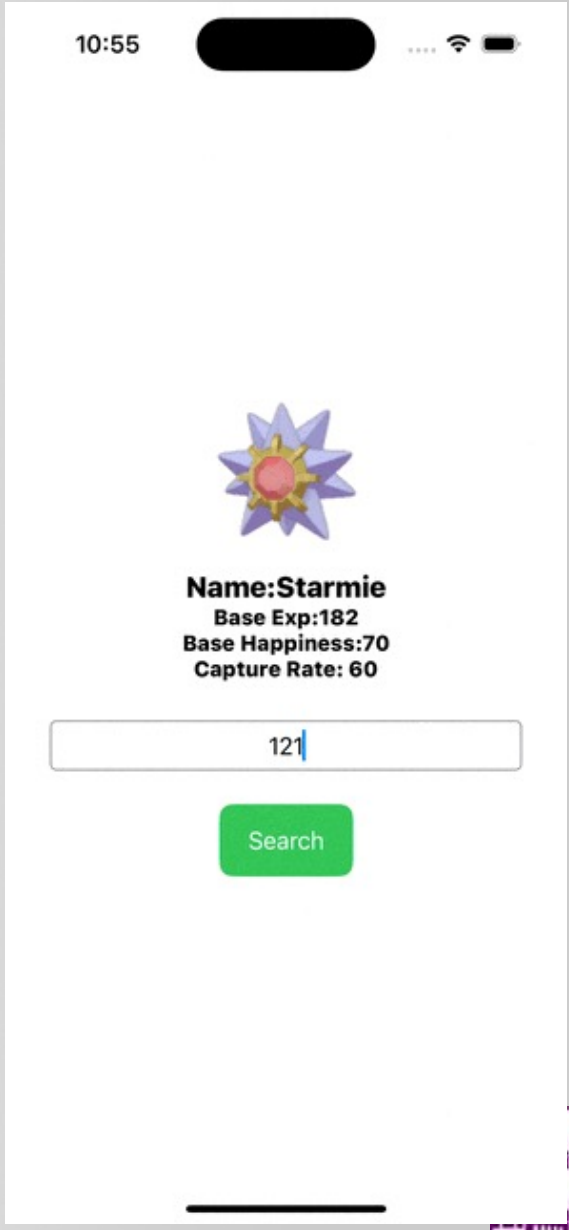
Pokédex Version 3

Task: Build the Pokédex Version 3

Lab2: Pokédex version 3

API Address: https://pokemon.wcpc.fun/id/1
https://pokemon.wcpc.fun/gpt/1

Task: Develop a Pokédex Application
Objective: Create a user-friendly mobile application to serve as a Pokédex (No.1 – 151). The app should display crucial attributes of each Pokémon, including a profile picture.
Requirements:
1.Core Attributes: Integrate at least Six attributes from the Pokémon API, with "name" being a mandatory field. Other attributes may include weight, height, base experience, etc.
2.User Interface: Develop an intuitive and visually appealing user interface that displays the Pokémon's profile picture alongside its attributes.
3.Search Functionality: Implement a search feature that allows users to input a Pokémon ID and retrieve corresponding information.
4.ChatGPT Descriptions: For every Pokémon, request data from https://pokemon.wcpc.fun/gpt/:pokemon_id to obtain a description provided by ChatGPT for the respective Pokémon ID. Display this description prominently within the app, enriching the information available to the user.
5.UI Design: Prioritize aesthetics and user experience. Aim to make the interface polished and visually engaging.



Q & A