

Pokedex

You've always wanted one, and now you have the skills to make one. It is time to make a Pokedex!

The best part is the data is all freely available from the API at <https://pokeapi.co/>.

Note: the API is free, but they ask that you don't ruin it for everyone by making too many requests per day. Try to limit yourselves to 1000 or less a day to be safe.

Structure

As we discussed in class, JavaScript is asynchronous - which means we can't structure our program the same way we might for a synchronous language. This means some processes take an indeterminant amount of time to complete. We don't want to wait for those to finish, so we use callback functions instead. For instance, consider reading a line of input from the user. We have no idea how long it is going to take the user to get around to typing a response in, so why waste time waiting? Simply ask the user to enter some information and let JavaScript continue doing some other work while it is waiting! Here's an example:

```
// Load the readline library
const readline = require("readline");
// Setup readline to listen on the stdin stream
const rl = readline.createInterface(process.stdin,
process.stdout);

... Somehwere else in our code where we wish to get input ...

rl.question("What is your name? ", (response) => {
    console.log("Hello, " + response + "!");
});
```

If we had another statement immediately after the prompt, it would run **before our user could enter anything**. This is because JS doesn't know how long it will take for the user to input their name. Therefore, it simply goes on working, and the function

```
(response) => {
    console.log("Hello, " + response + "!");
}
```

gets called when they are finished.

If we had wanted the program to continue and do something else **only after the hello message was printed**, we would have had to put it inside this function body, after the `console.log` line.

Our program will have the following functions:

- **showMenu()** - will display all the menu options.
- **prompt(cb)** - will use `readline` to ask the user for a search term. It will then call the function passed into it (which is what `cb` is - a callback function), and pass the data the user entered as a parameter.
- **searchPoke(term)** - will query the API for a particular Pokemon (passed in as `term`). If it receives a valid response, it will call `printPoke(json)` with the json to print out the name, weight, height, base experience, and all the moves for that Pokemon. It will then call `run()` again to reprompt.
- **printPoke(json)** - print the data for the Pokemon in a neat and clean way.
- **searchItem(term)** - works exactly like the `searchPoke()` function, except searches the Item endpoint for an item. Calls the corresponding `printItem(json)` method. Calls `run()` to reprompt.
- **printItem(json)** - prints item data neatly. Pick at least four fields to display from the endpoint's data.
- **searchMove(term)** - works exactly like the `searchPoke()` function, except searches the Move endpoint for a move. Calls the corresponding `printMove(json)` method.
- **printMove(json)** - prints the move data in a neatly formatted way. Calls `run()` to reprompt.
- **run()** - will call `showMenu()`, then use `readline` to ask the user to enter their choice. We will call the `prompt` function and pass to it the name of the function we wish to use for searching.

An example flow of data through the program:

1. `run()` is called. It calls `showMenu()`. It prompts the user for a choice from the menu.
2. User selects 1. The `prompt()` function is called and `searchPoke` is passed as the parameter.
3. `prompt()` asks for a search term, then calls `searchPoke` with that search term.
4. `searchPoke()` calls upon the API with the provided search term. On receipt of data, it takes the json from the response and calls `printPoke()`, passing in the JSON data extracted from the response.

5. `printPoke(json)` extracts and neatly prints the data from the JSON. It then calls `run()` to start the process over again.

If you don't understand this flow of data, or why we must structure our code this way, don't start on the project. You will only code yourself into a corner. Instead, come to office hours, talk to other students, and try to understand why it works this way.

Grading

This is a partner project, but **NOT** an AI-assistant project.

Category	Points	Description	Evaluation Focus
1. Code Functionality & Flow	20	The Pokédex runs without syntax or logic errors and follows the required flow (<code>run() → showMenu() → prompt() → search → print → run()</code>).	Program runs correctly and re-prompts user. All menu options (Pokemon, Item, Move) work. Handles invalid entries gracefully.
2. API Interaction (Asynchronous Code)	20	Demonstrates correct use of asynchronous code (<code>fetch</code> , Promises, or <code>async/await</code>).	Uses <code>fetch()</code> correctly with proper endpoints. Handles both success and error responses. Demonstrates understanding of async logic flow.
3. Callback & Program Flow Understanding	15	Correct use of callbacks, asynchronous flow, and non-blocking design.	Correctly uses callbacks (e.g. <code>prompt(cb)</code>). Code structure reflects non-blocking JS execution. No nesting confusion or misplaced async calls.
4. Output Formatting & Data Presentation	15	Neat, user-friendly output for Pokémon, Item, and Move data.	Uses template literals or formatted strings. Displays key data fields clearly (name, height, weight, abilities, etc.). Output is visually readable and structured.

5. Code Quality & Documentation	15	Code is clean, well-commented, and uses meaningful identifiers.	Comments explain why not just what. Indentation and spacing are consistent. Variables and functions have descriptive names. Uses let/const correctly.
6. Error Handling & Edge Cases	15	Handles bad input or missing API data without crashing.	Invalid Pokémon name handled gracefully. Displays friendly message instead of errors. Uses try...catch or .catch() for promise rejection.