Instructions, Documentation, and Participation

Github Link: https://github.com/avery-feldman/206project

Instructions

1. Open the project in VSCode or another IDE
    a. Go to github repository and copy link
    b. Git clone to folder of preference
    c. Open project in IDE of choice.
2. Open the functions.py file
3. Scroll down to bottom of file, to def main()
4. Go to the line that says:
    a. pokemonDiction= server.getPokemonNameTypes(cur,conn,25)
    b. Changing the value of the number adjusts the limit of how much data goes into all of the tables (max amount). Feel free to change if needed.
5. Run functions.py. The code will begin running, with each function showing when it is completed from print statements located in def main()
6. Open SQLite and open database created by the program (PokeDatabase.db)
7. Take a look at the tables to see information stored about the Pokemon
8. Visualizations will also appear as a pop up tab at the end when functions.py is ran
    a. Close the current visual to get to the next

Documentation

206 Project Folder File Paths

1. functions.py: Location of the runnable code
2. Report folder: Holds information regarding the Report of the project- this includes:
    a. Visualizations of the data
    b. Data Diagram
    c. Goals and Problems
    d. Calculation results made using data
    e. Instructions and Documentations (this folder)
3. ReadMe.md: No information pertinent to the project

Functions.py Methods

All implementations apart from the main function are located in the Pokemon Class.

- **def main():**
    - Located outside of the Pokemon class, runs the code for the project.

- **def createStructure(self,cur,conn):**
    - Initializes database and tables
    - Requires connection to database (cur,conn)
- **def getPokemonNameTypes(self,cur,conn,limit):**
    - Requires limit of number of pokemon (in main function, it is declared as 25) and connection to database (cor,conn)
    - Chooses the pokemon being focused on, returns a dictionary where the key is pokemon and value are their types
    - {pokemon1: type, pokemon2: type, pokemon3: type}
    - There are only about 18 types of pokemon, so we will never go over 25 entries for this table.
- **def getPokemonMoves(self,cur,conn,pokemonDiction):**
    - Requires pokemonDiction returned from getPokemonNameTypes and connection to database (cur, conn)
    - Returns a dictionary where the key is pokemon and value are their list of moves
    - {pokemon1: [move1,move2,move3], pokemon2: [move1,move2,move3,move4], pokemon3:[move1,move2,move3,move4]}
- **def getPokemonAbilities(self,cur,conn,pokemonDiction):**
    - Requires pokemonDiction returned from getPokemonNameTypes and connection to database (cur,conn)
    - Returns a dictionary where the key is pokemon and value are their list of moves
    - {pokemon1: [ability1,ability2,ability3], pokemon2: [ability1,ability2,ability3,ability4], pokemon3:[ability1,ability2,ability3,ability4]}
- **def getAbilityCount(self,returnedPokemonAbilityDiction):**
    - Requires returnedPokemonAbilityDiction from getPokemonAbilities
    - Returns a dictionary where the key is an ability name and the value is how many pokemon have said ability in the pokeapi
    - {ability1: count, ability2: count, ability3: count...}
    - Keeps track of previous abilities that have been counted to ensure each ability is counted only once
- **def getMoveInfo(self,pokemonMoveDiction):**
    - Requires pokemonMoveDiction returned from getPokemonMoves
    - Returns a dictionary where the key is a pokemon move and value is a dictionary where type,power, accuracy are keys, and values are their actual values

- {move1:
  {
  power: power1
  type: type1
  value: value1
  },
  move2:
  {
  power: power2
  type: type2
  value: value2
  },
  move3:
  {
  power: power3
  type: type3
  value: value3
  }
  }

- **def insertTypeData(self,cur,conn, pokemonDiction, pokemonMNAPDiction):**
  - Requires pokemonDiction returned from getPokemonNameTypes (pogoAPI), pokemonMNAPDiction returned from getMoveInfo (bulbapedia), and connection to database (cur, conn)
  - Inserts data into type table
- **def insertMoveData(self,cur,conn,pokemonMNAPDiction):**
  - Requires pokemonMNAPDiction returned from getMoveInfo (bulbapedia requirement), connection to database (cur, conn)
  - Inserts data into moves table using Types Table for foreign key
- **def insertAbilityData(self,cur,conn,pokemonAbilityDiction,abilityCountDiction):**
  - Requires dictionary returned from getPokemonAbilities (pokeAPI requirement), dictionary returned from getAbilityCount (pokeAPI Requirement), and connection to database (cur,conn)
  - Inserts data into ability table

- **def insertPokemonData(self,cur,conn,pokemonDiction, pokemonMoveDiction, pokemonAbilityDiction):**
    - Requires dictionaries returned from getPokemonNameTypes (pogoAPI requirement), getPokemonMoves (pokeAPI), getPokemonAbilities (pokeAPI), as well as connection to database (cur,conn)
    - Inserts data into pokemon table using Type, Moves, and Ability table for foreign keys
- **def calculationsFile(self, cur, con):**
    - Requires connection to database (cur, con)
    - Takes data from all tables that originate from all apis/website
    - Uses data to output a "poke_calculations.txt" file that contains the calculations of
        - The strongest pokemon type, the weakest pokemon type, the most common pokemon type, and the least common pokemon type
        - The strongest move type, the weakest move type, the most common move type, and the least common move type
        - The strongest ability, the weakest ability, the most common ability, and the least common ability
    - Does so by averaging the calculated strengths by how length, and by counting type and ability occurrences
- **def powerAccuracyVisualization(self, cur, conn):**
    - Requires connection to database (cur,con)
    - Returns a scatterplot graph of each move's accuracy (x) to its power (y)
        - Beautifies the graph
    - Draws regression line and writes correlation coefficient of the data points on the upper left portion of the graph
- **def moveTypeStrVisualization1(self, cur, conn):**
    - Requires connection to database (cur,con)
    - Graphs each move's overall strength (power * accuracy) relative to its type
        - Keeps track of the type of each move to color code it properly
        - Beautifies the graph
    - Returns graph of type (x) to move strength (y)
- **def moveTypeStrVisualization2(self, cur, conn):**
    - Requires connection to database (cur, conn)
    - Graphs the average of each move's overall strength relative to its type

        - Keeps track of the type of each move to color code it properly
        - Beautifies the graph
    - Does so by dividing the total overall strength of all the moves of said type by the amount of moves of said type
    - Returns bar graph of averages
- **def pokemonTypeStrVisualization1(self, cur, conn):**
    - Requires connection to database (cur,con)
    - Graphs each pokemon's overall strength (by averaging the overall strength of its moves) relative to its type
        - Keeps track of the type of each move to color code it properly
        - Beautifies the graph
    - Returns graph of type (x) to pokemon strength (y)
- **def pokemonTypeStrVisualization2(self, cur, conn):**
    - Requires connection to database (cur, conn)
    - Graphs the average of each pokemon's overall strength relative to its type
        - Keeps track of the type of each move to color code it properly
        - Beautifies the graph
    - Does so by dividing the total overall strength of all the moves of said type by the amount of moves of said type
    - Returns bar graph of averages
- **def AbilityCommonalityVisualization(self, cur, conn):**
    - Requires connection to database (cur, con)
    - Returns a scatterplot of each ability's occurrences/commonality (x) to its overall strength (y)
        - Beautifies the graph
    - Does so by averaging its overall strength based on the number of pokemon that have said ability and the overall strength of said pokemon
    - Draws regression line and writes correlation coefficient of the data points on the upper left portion of the graph

***there also obtains additional documentation for each respective function within the functions.py file itself in the form of comments**

Participation

- Rishabh Verma
    - Created database diagram
    - Created structure of database (createStructure function)
    - getPokemonMoves function
    - getPokemonAbilities function
    - getMoveInfo Count
    - insertTypeData function
    - insertMoveData function
    - insertAbilityData function
- Avery Feldman
    - calculationFiles function
    - getAbilityCount function
    - moveTypeStrVisualization1 function
    - moveTypeStrVisualization2 function
    - pokemonTypeStrVisualization1 function
    - pokemonTypeStrVisualization2 function
    - Created visualizations
    - Created calculation file
- Jonah Feldman
    - Created basic structure for functions.py
    - Created Repository for Project
    - getPokemonNameTypes function
    - insertPokemonData function
    - Main function
    - powerAccuracyVisualization function
    - abilityCommonalityVisualization function
- Group
    - Final Project Plan
    - Goals and Problems
    - Resources
    - Instructions, Documentation, and Problems