Universidad Autónoma De Nuevo León
Facultad de Ciencias Físico Matemáticas

Programación Básica
DOCUMENTACIÓN PIA

Docente. Osvaldo Habib González González

2069465 - Angel Adrian Cortes Zuñiga
2142757 - Rebeca Moreno González

Gpo. 076

MINUTAS DE TRABAJO

11/04/2025

GRUPO 2

Minuta 1

Miembros del equipo:

- Rebeca Moreno González
- Angel Adrian Cortes Zuñiga

El dia de hoy todo el equipo asistió

Roles asignados

- Código: Angel Adrian Cortes Zuñiga, Rebeca Moreno González

Objetivo del día:
El objetivo fue ver y analizar lo que íbamos a hacer en el proyecto, asi como cual API deberiamos de usar y dividirnos las tareas

Acuerdos tomados:
Se decidió que los dos trabajaremos tanto en el código y en la documentación juntos. Cada quien inicio la parte que le tocaba y para luego checar estas y unirlas

Dificultades encontradas:
Lo más difícil de este día fue decidir la API, por la gran variedad que hay

26/04/2025

GRUPO 2

Minuta 2

Miembros del equipo:

- Rebeca Moreno González
- Angel Adrian Cortes Zuñiga

El dia de hoy todo el equipo asistió

Objetivo del día:
El objetivo fue empezar el código, aunque sea la recaudación de información del API y realizar la documentación.

Tareas realizadas:
Se terminaron las primeras dos partes de la documentación que era acerca de la API y lo que íbamos a hacer con ella, y dividimos las partes para empezar a realizar el código.

Dificultades encontradas:
Lo más difícil de este día fue empezar al realizar el código

28/04/2025

GRUPO 2

Minuta 3

Miembros del equipo:

- Rebeca Moreno González
- Angel Adrian Cortes Zuñiga

El dia de hoy todo el equipo asistió

Objetivo del día:
El objetivo fue seguir con el código y terminar la documentación

Tareas realizadas:
Realizamos cada quien una parte del código y se verificó que funcionara.

Dificultades encontradas:
Lo más difícil de este día fue el pensar que estructuras utilizar y que instrucciones escribir en python

01/05/2025

GRUPO 2

Minuta 4

Miembros del equipo:

- Rebeca Moreno González
- Angel Adrian Cortes Zuñiga

El dia de hoy todo el equipo asistió

Objetivo del día:
El objetivo fue seguir con el código y terminar la documentación

Tareas realizadas:
Se terminó de realizar la documentación y seguimos con la elaboración del código.

Dificultades encontradas:
Lo más difícil de este día fue la realización del código.

08/05/2025

GRUPO 2

Minuta 5

Miembros del equipo:

- Rebeca Moreno González
- Angel Adrian Cortes Zuñiga

El dia de hoy todo el equipo asistió

Objetivo del día:
El objetivo fue seguir con el código, y subir el avance a github y a teams.

Tareas realizadas:
Seguimos realizando el código, ya en la parte de sacar estadísticas, y subimos la documentación, minutas de trabajo hasta ahora, y lo que llevamos del código, se empezó a pensar ideas para la realización del video.

Dificultades encontradas:
Lo más difícil de este día fue crear las estructuras para calcular las estadísticas, y lograr subir los archivos a github.

12/05/2025

GRUPO 2

Minuta 6

Miembros del equipo:

- Rebeca Moreno González
- Angel Adrian Cortes Zuñiga

El dia de hoy todo el equipo asistió

Objetivo del día:
El objetivo fue checar como iba el código

Tareas realizadas:
Seguimos realizando el código, checamos que hasta ahorita no hubiera ningún fallo y nos pusimos de acuerdo para realizar la documentación que faltaba.

Dificultades encontradas:
Lo más difícil de este día fue la realización del algoritmo, ya que eran muchos

14/05/2025

GRUPO 2

Minuta 7

Miembros del equipo:

- Rebeca Moreno González
- Angel Adrian Cortes Zuñiga

El dia de hoy todo el equipo asistió

Objetivo del día:
Terminar el código y documentación

Tareas realizadas:
Se terminó de realizar el código y la documentación y nos empezamos a poner de acuerdo en como íbamos a grabar el vídeo

Dificultades encontradas:
Lo más difícil de este día fue la realización del algoritmo

15/05/2025

GRUPO 2

Minuta 7

Miembros del equipo:

- Rebeca Moreno González
- Angel Adrian Cortes Zuñiga

El dia de hoy todo el equipo asistió

Objetivo del día:
Realizar video

Tareas realizadas:
Se realizó el guión y la grabación del video

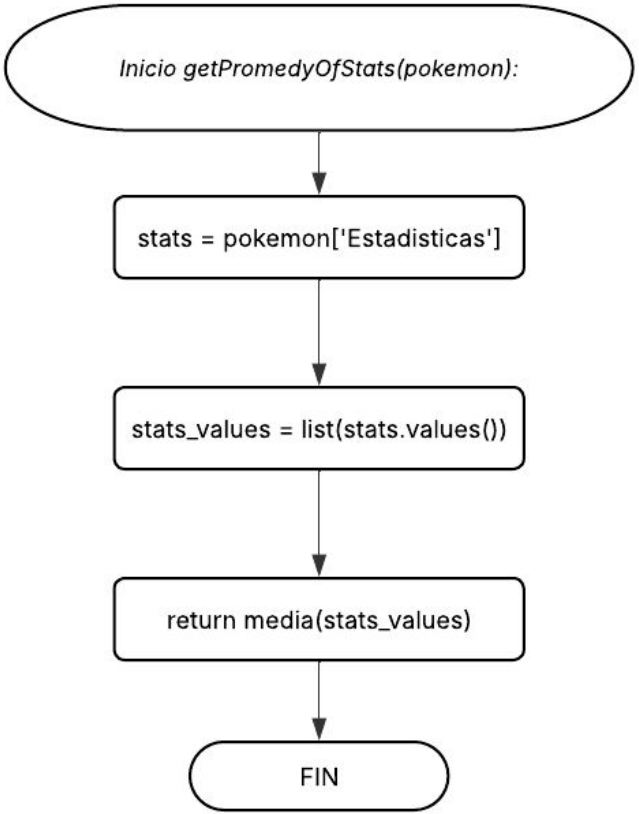
Dificultades encontradas:
Lo más difícil de este día fue ver como podíamos explicar nuestro proyecto de una manera sencilla

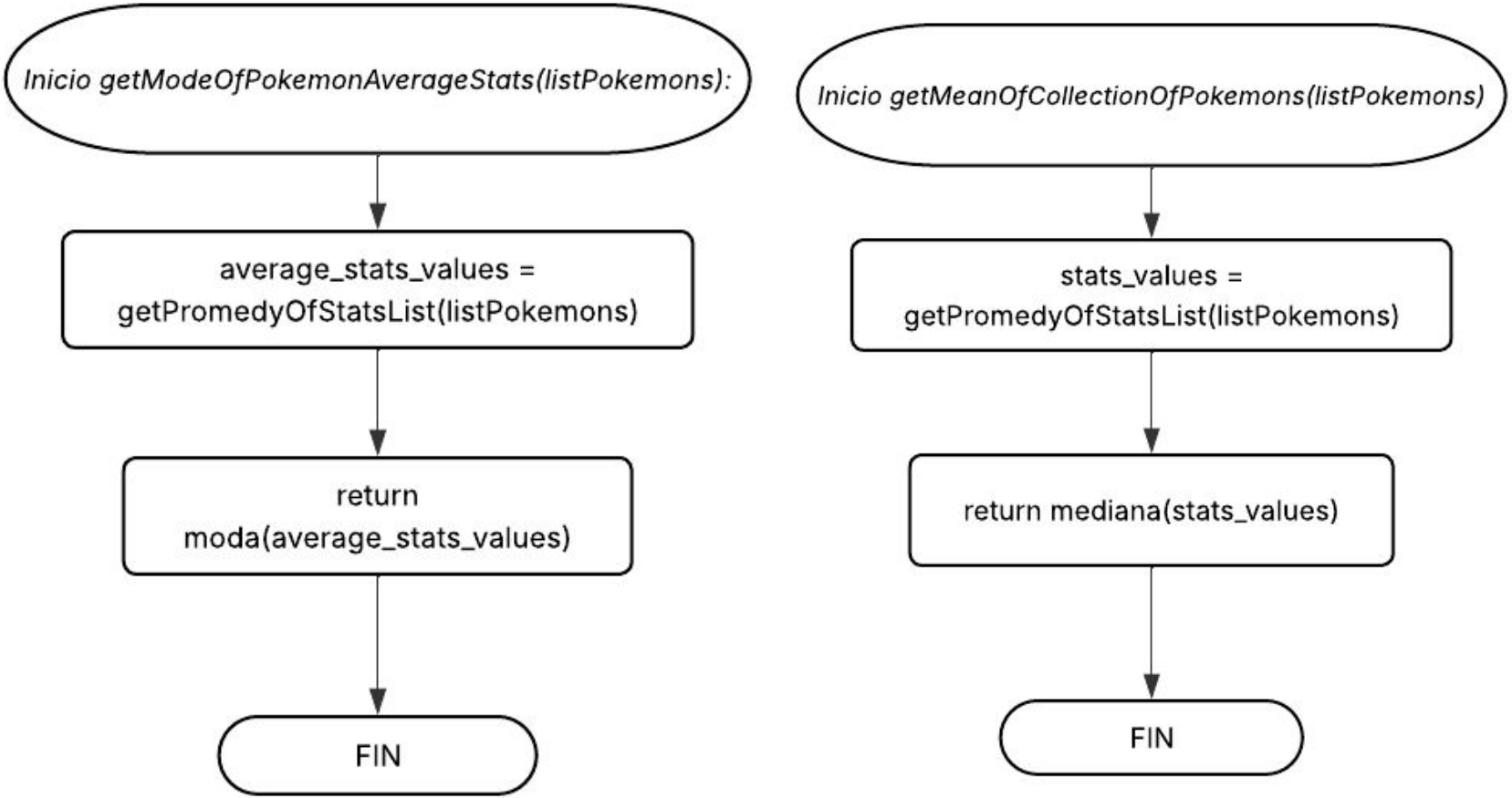
CUADRO COMPARATIVO

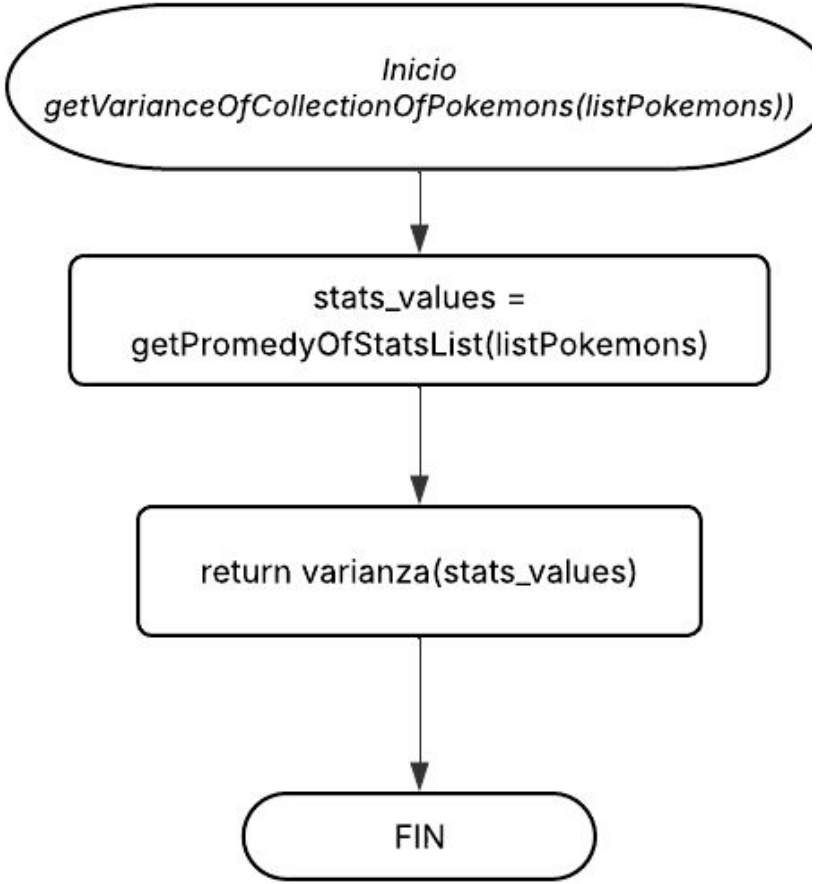
API	PROPÓSITO	DATOS QUE CONTIENE	AUTENTICACIÓN	LÍMITE DE USO	LINK
PokéAPI	Información detallada del universo Pokémon.	Pokémon, habilidades, movimientos, especies, etc.	No requiere	100 llamadas por hora por IP	https://pokeapi.co
OpenWeatherMap	Datos meteorológicos actuales, históricos y de pronóstico.	Clima actual, pronóstico, históricos	Requiere API Key (gratis)	La suscripción gratuita incluye 1000 llamadas a la API diarias	https://openweathermap.org/api
NASA APIs Exoplanet	Datos sobre exoplanetas descubiertos por misiones espaciales	Nombre, masa, tamaño, distancia, métodos de descubrimiento	Requiere API Key (gratis)	1000 solicitudes por hora	https://api.nasa.gov
FakerAPI.dev	Generación de datos falsos para pruebas.	Usuarios, textos, productos, direcciones, etc.	No requiere	Ilimitado	https://fakerapi.it/en

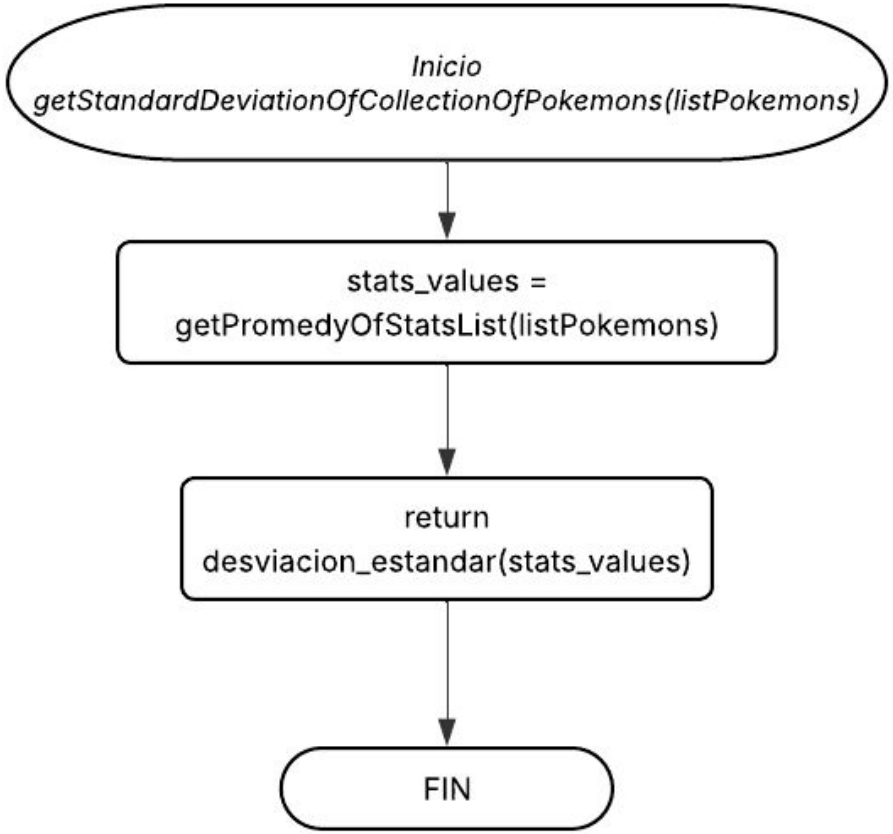
DIAGRAMAS DE FLUJO

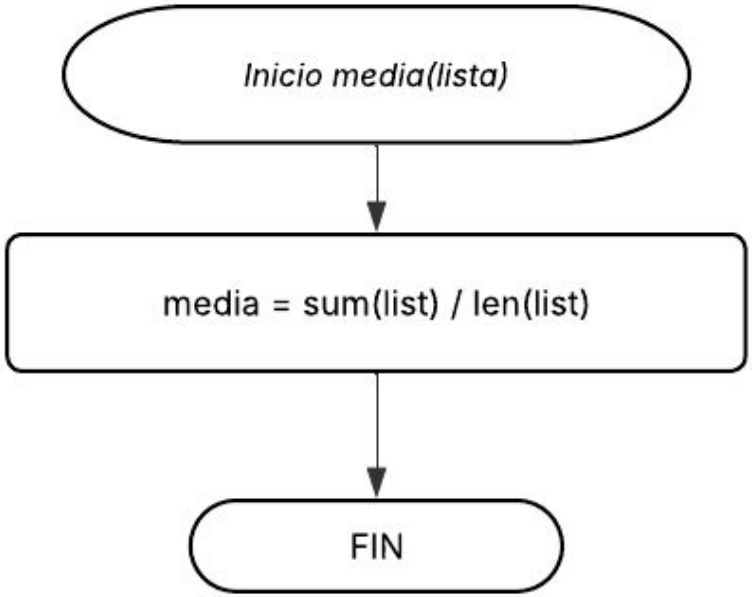

Inicio getPromedyOfStatsList(listPokemons)
stats_values = []
for pokemon in listPokemons
No
True
stats_values.append(getPromedyOfStats(pokemon))
return stats_values
FIN

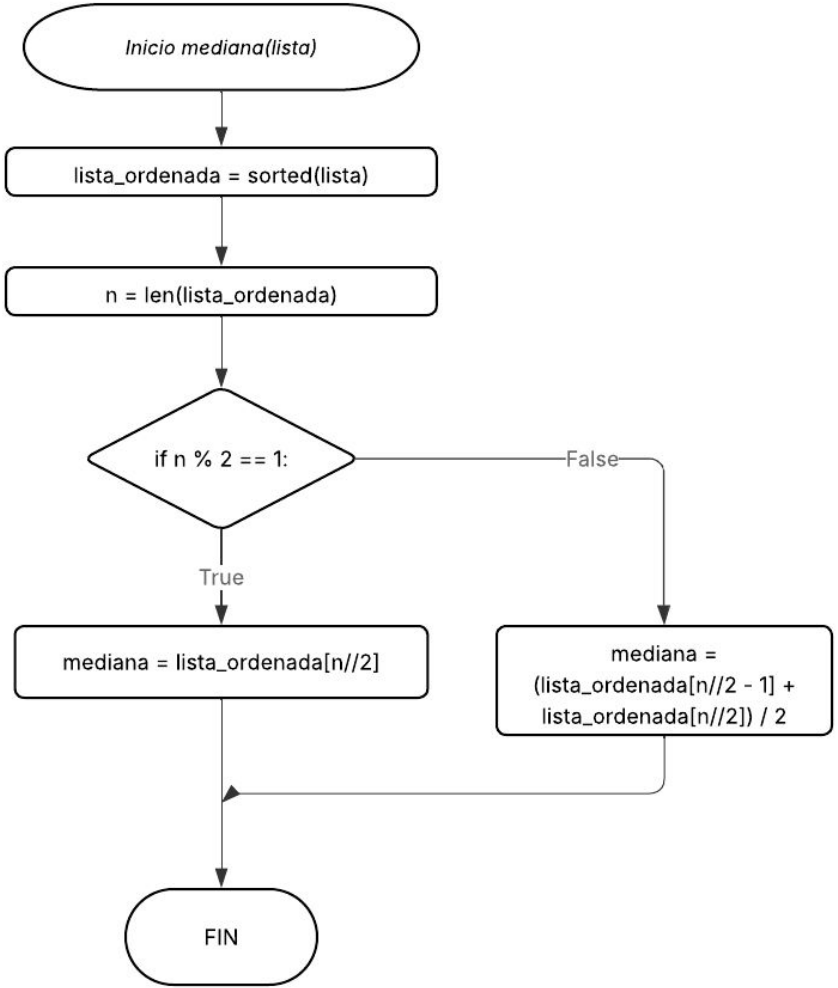

Inicio getPromedyOfStats(pokemon):
stats = pokemon['Estadisticas']
stats_values = list(stats.values())
return media(stats_values)
FIN

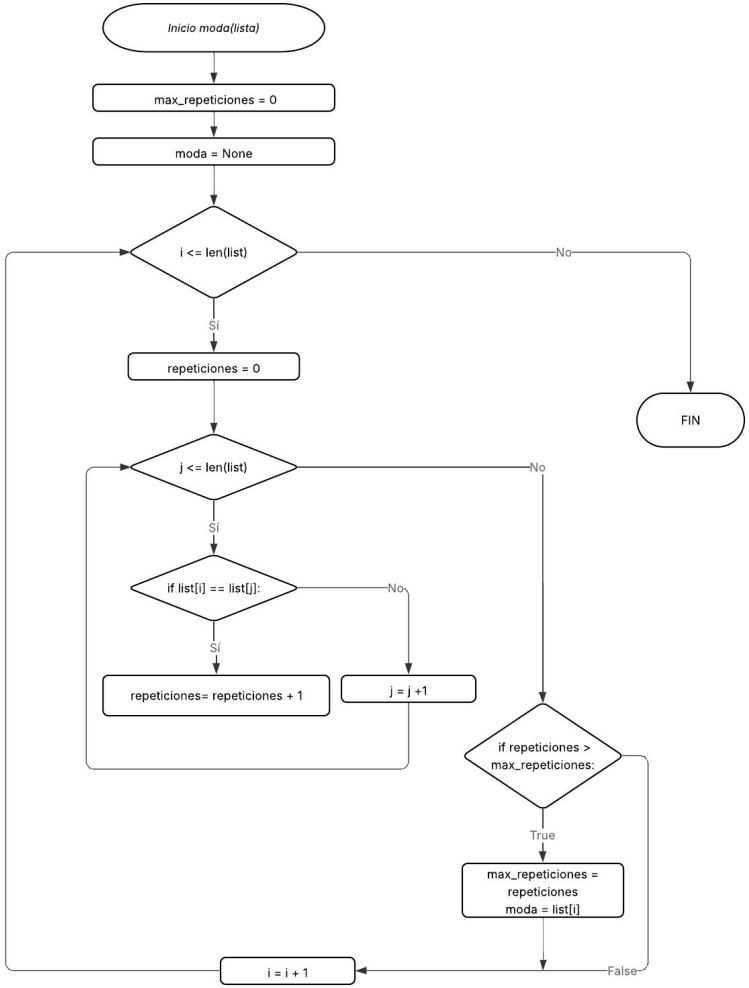

Inicio getModeOfPokemonAverageStats(listPokemons):
average_stats_values = getPromedyOfStatsList(listPokemons)
return moda(average_stats_values)
FIN
Inicio getMeanOfCollectionOfPokemons(listPokemons)
stats_values = getPromedyOfStatsList(listPokemons)
return mediana(stats_values)
FIN

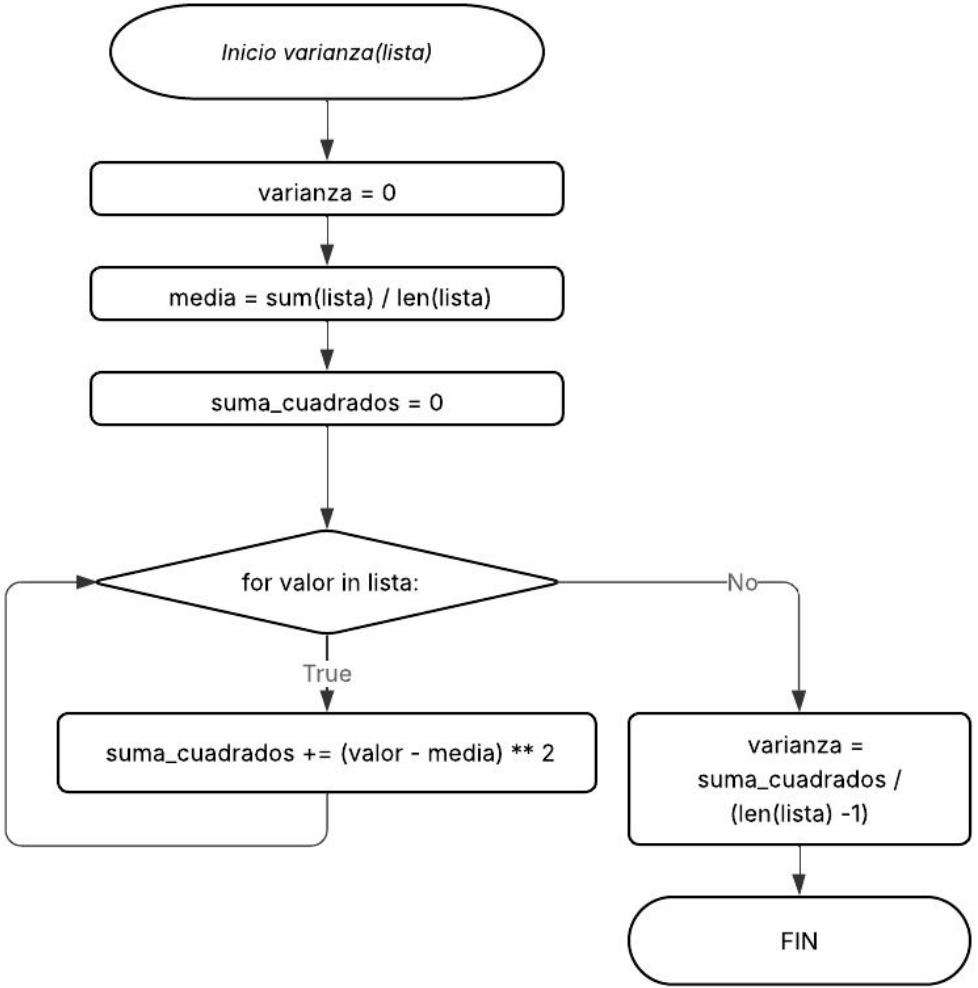

Inicio
getVarianceOfCollectionOfPokemons(listPokemons))
stats_values =
getPromedyOfStatsList(listPokemons)
return varianza(stats_values)
FIN

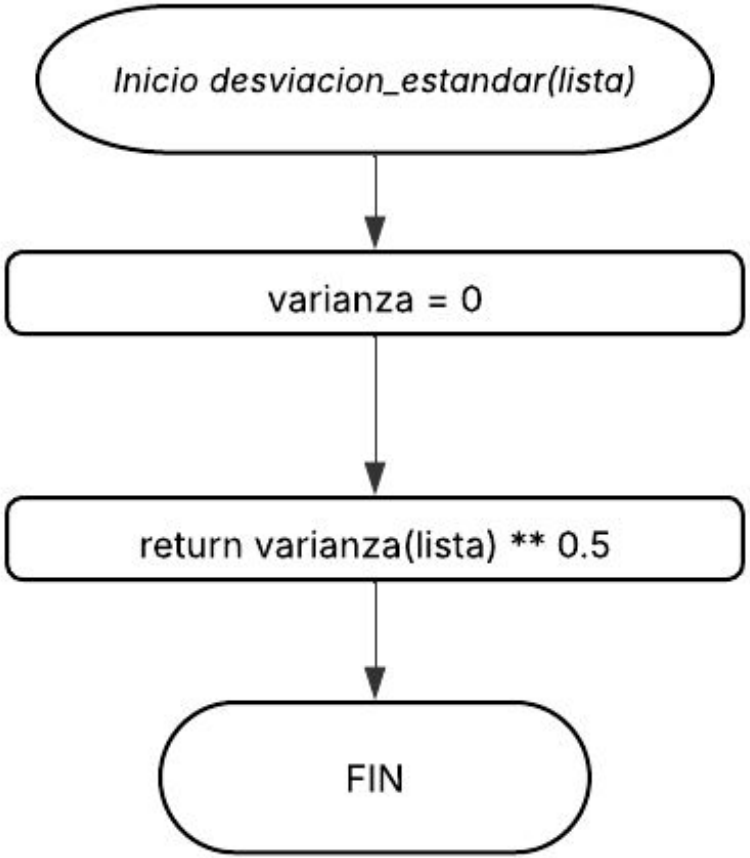

Inicio
getStandardDeviationOfCollectionOfPokemons(listPokemons)
stats_values =
getPromedyOfStatsList(listPokemons)
return
desviacion_estandar(stats_values)
FIN

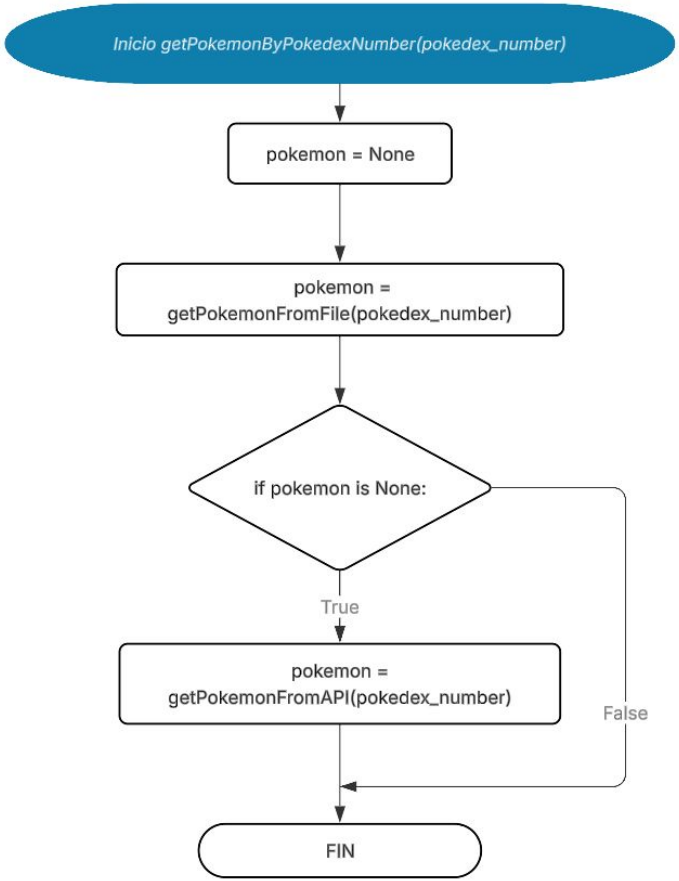

Inicio media(lista)
media = sum(list) / len(list)
FIN

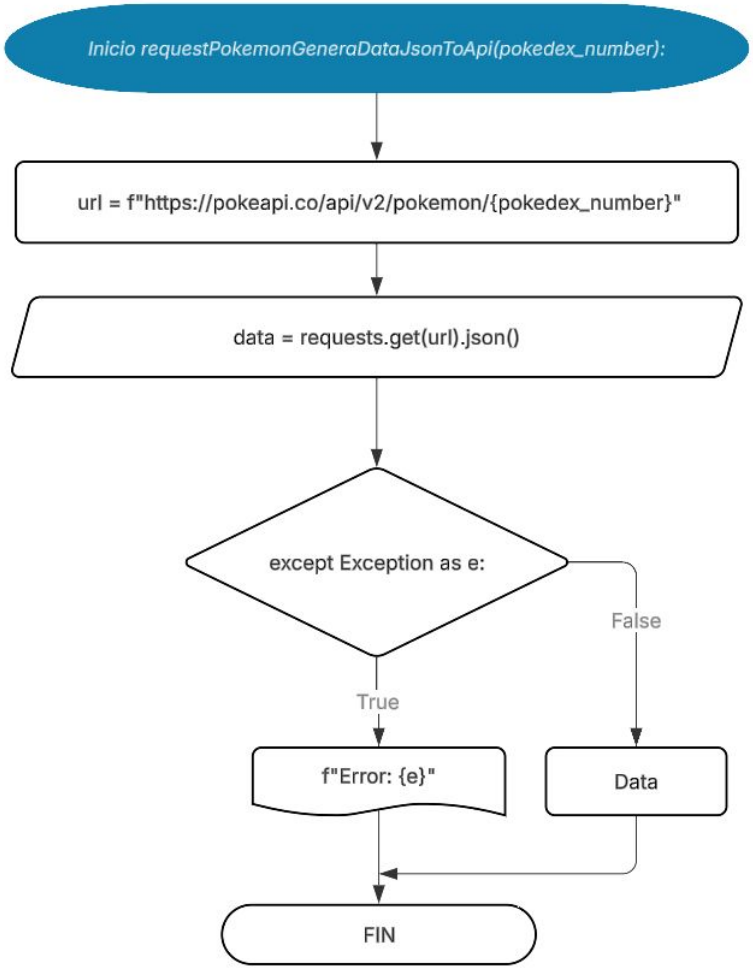

Inicio mediana(lista)
lista_ordenada = sorted(lista)
n = len(lista_ordenada)
if n % 2 == 1:
False
True
mediana = lista_ordenada[n//2]
mediana = (lista_ordenada[n//2 - 1] + lista_ordenada[n//2]) / 2
FIN

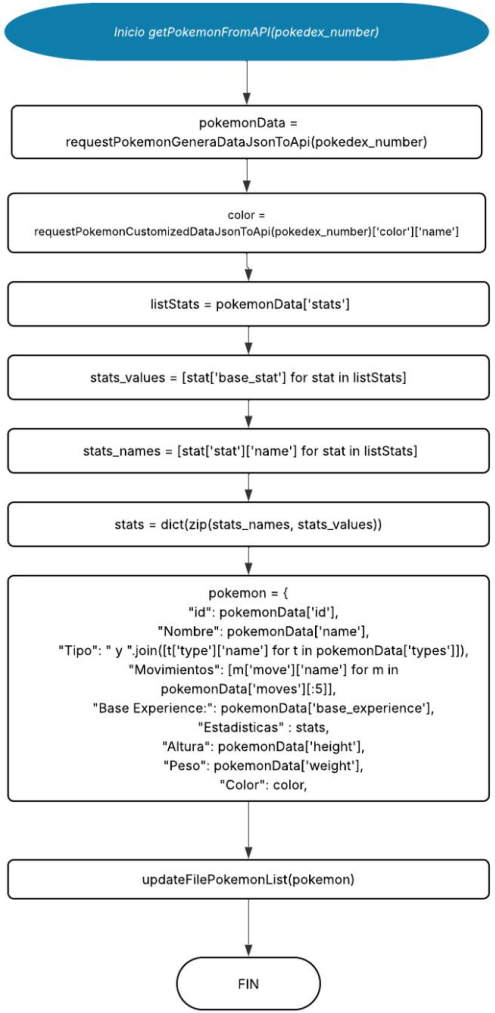

Inicio moda(lista)
max_repeticiones = 0
moda = None
i <= len(list)
No
Sí
repeticiones = 0
FIN
j <= len(list)
No
Sí
if list[i] == list[j]:
No
Sí
repeticiones= repeticiones + 1
j = j +1
if repeticiones > max_repeticiones:
True
max_repeticiones = repeticiones
moda = list[i]
i = i + 1
False

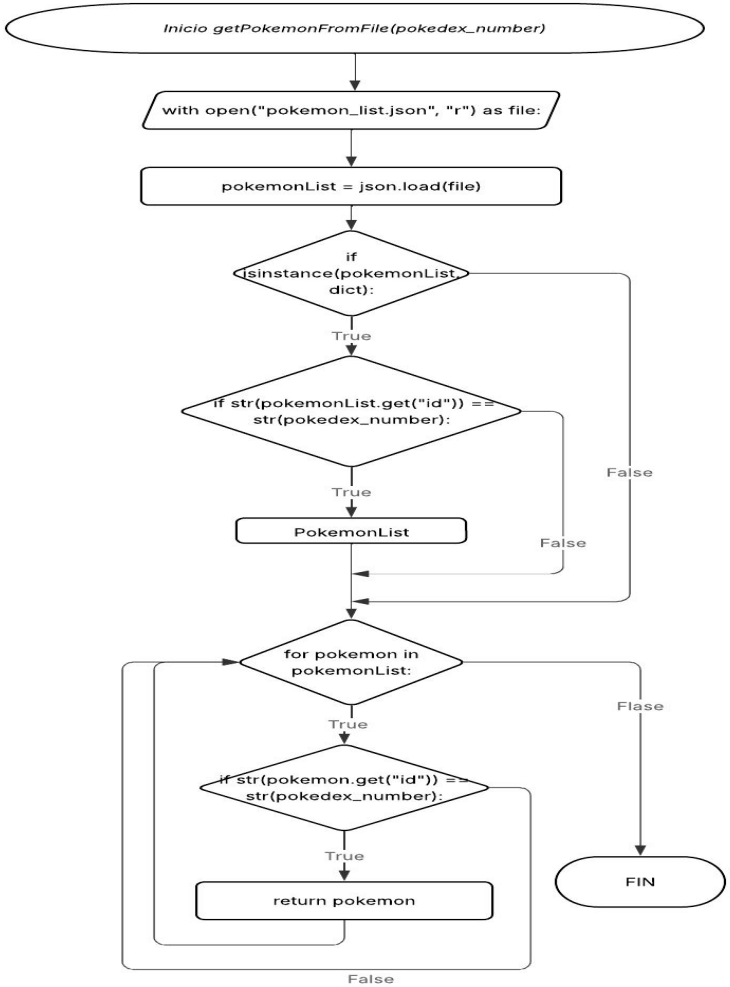

Inicio varianza(lista)
varianza = 0
media = sum(lista) / len(lista)
suma_cuadrados = 0
for valor in lista:
No
True
suma_cuadrados += (valor - media) ** 2
varianza = suma_cuadrados / (len(lista) -1)
FIN

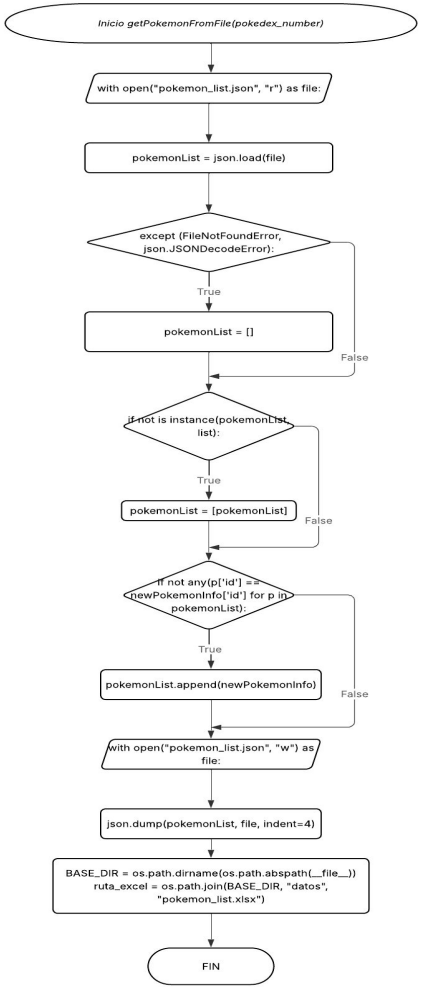

Inicio desviacion_estandar(lista)
varianza = 0
return varianza(lista) ** 0.5
FIN

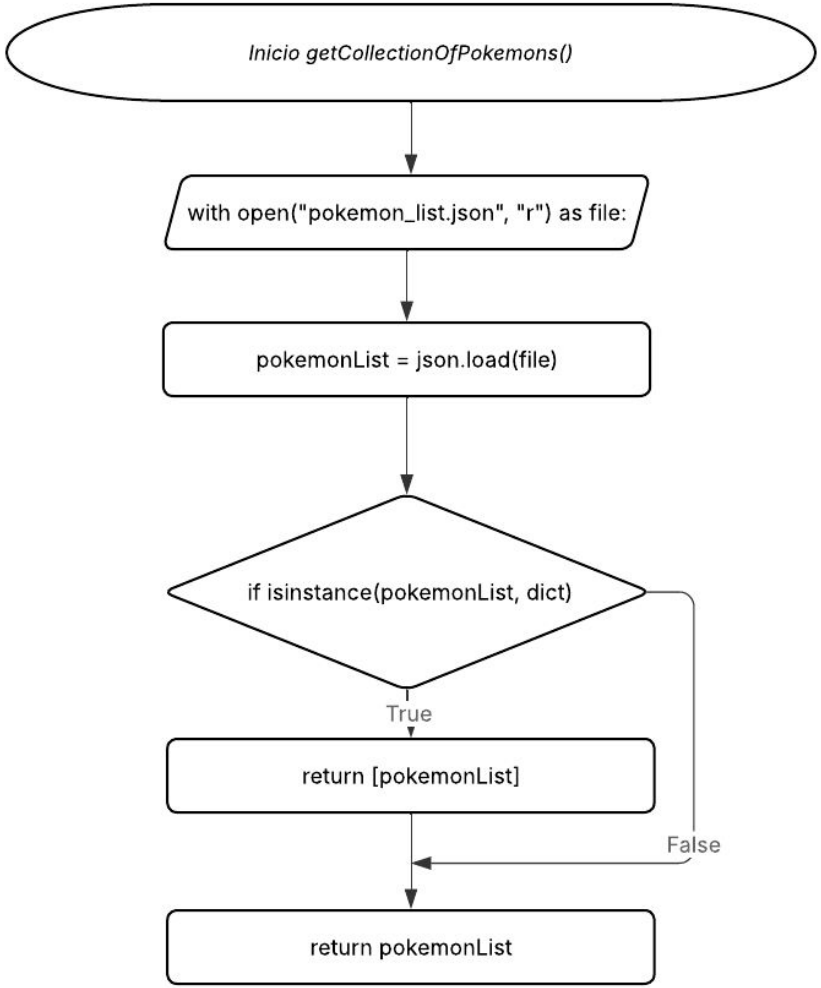

Inicio getPokemonByPokedexNumber(pokedex_number)
pokemon = None
pokemon = getPokemonFromFile(pokedex_number)
if pokemon is None:
True
pokemon = getPokemonFromAPI(pokedex_number)
False
FIN

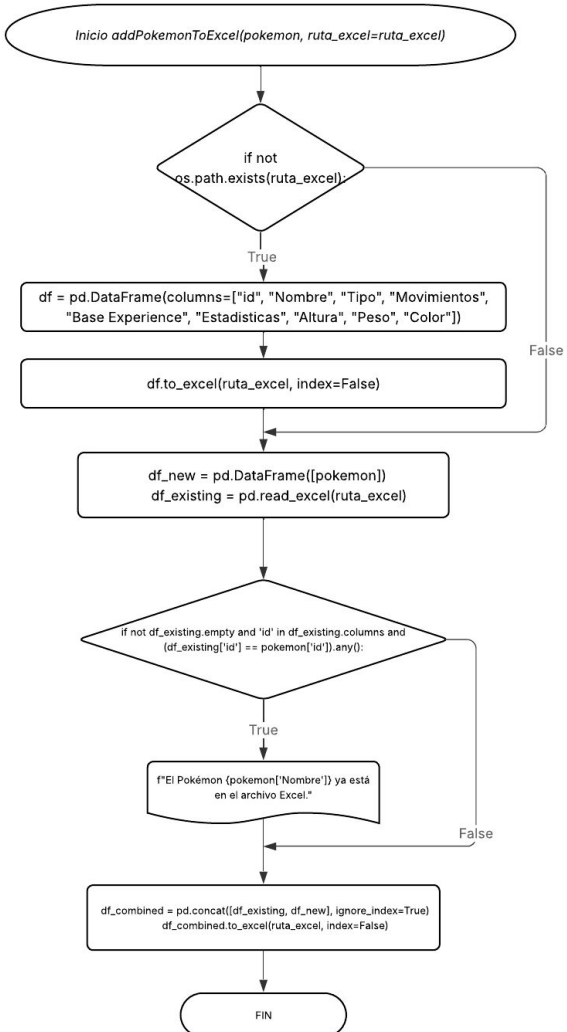

Inicio requestPokemonGeneraDataJsonToApi(pokedex_number):
url = f"https://pokeapi.co/api/v2/pokemon/{pokedex_number}"
data = requests.get(url).json()
except Exception as e:
True
False
f"Error: {e}"
Data
FIN

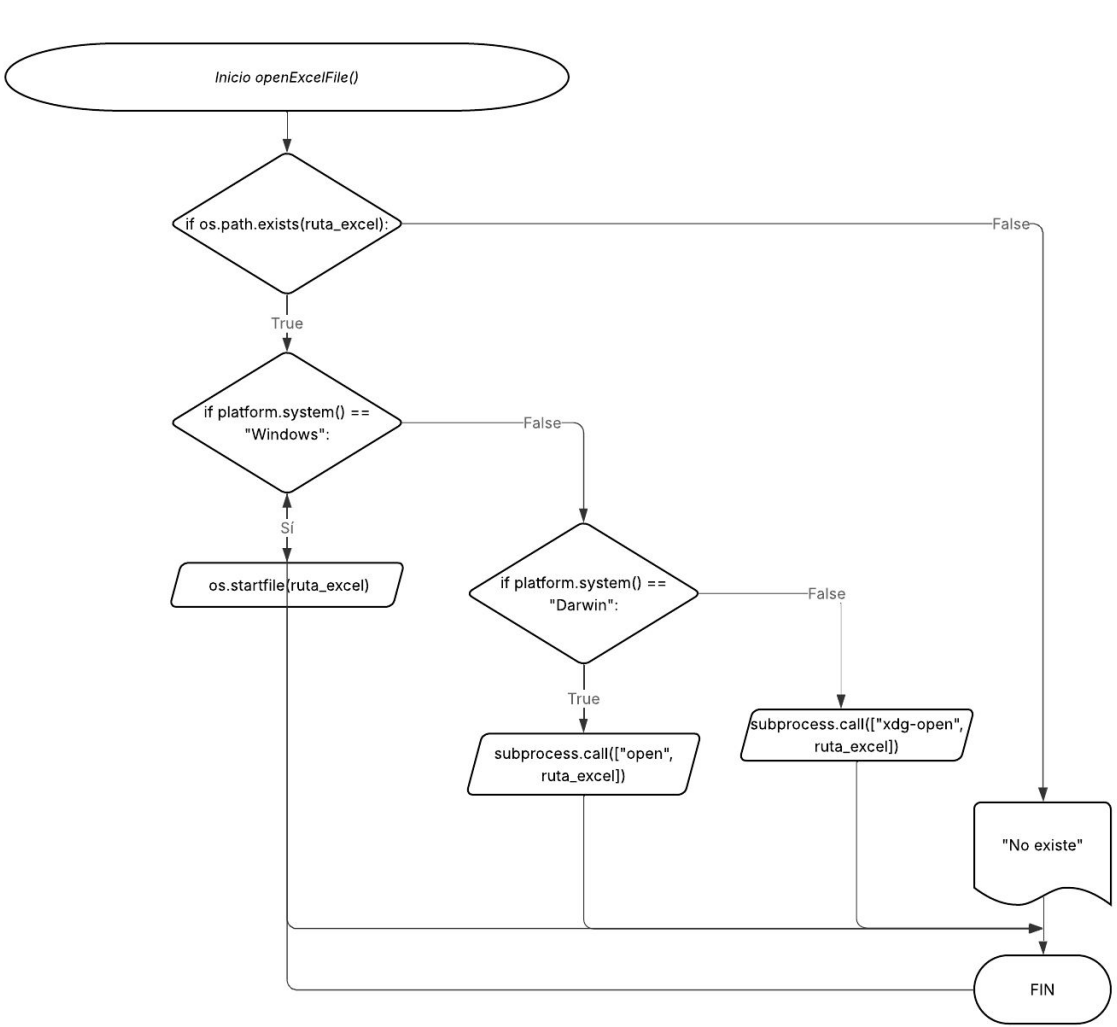

Inicio getPokemonFromAPI(pokedex_number)
pokemonData = requestPokemonGeneraDataJsonToApi(pokedex_number)
color = requestPokemonCustomizedDataJsonToApi(pokedex_number)['color']['name']
listStats = pokemonData['stats']
stats_values = [stat['base_stat'] for stat in listStats]
stats_names = [stat['stat']['name'] for stat in listStats]
stats = dict(zip(stats_names, stats_values))
pokemon = {
"id": pokemonData['id'],
"Nombre": pokemonData['name'],
"Tipo": " y ".join([t['type']['name'] for t in pokemonData['types']]),
"Movimientos": [m['move']['name'] for m in pokemonData['moves'][:5]],
"Base Experience:": pokemonData['base_experience'],
"Estadisticas" : stats,
"Altura": pokemonData['height'],
"Peso": pokemonData['weight'],
"Color": color,
updateFilePokemonList(pokemon)
FIN

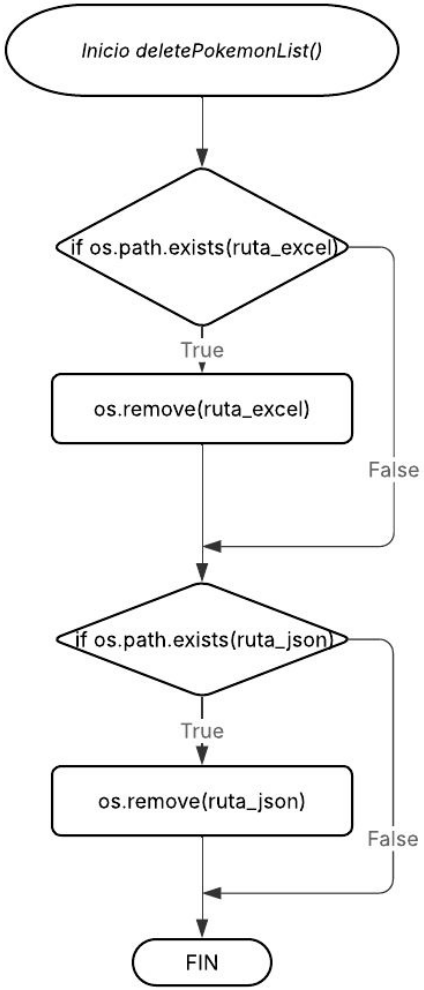

Inicio getPokemonFromFile(pokedex_number)
with open("pokemon_list.json", "r") as file:
pokemonList = json.load(file)
if isinstance(pokemonList, dict):
True
False
if str(pokemonList.get("id")) == str(pokedex_number):
True
False
PokemonList
for pokemon in pokemonList:
True
Flase
if str(pokemon.get("id")) == str(pokedex_number):
True
return pokemon
False
FIN

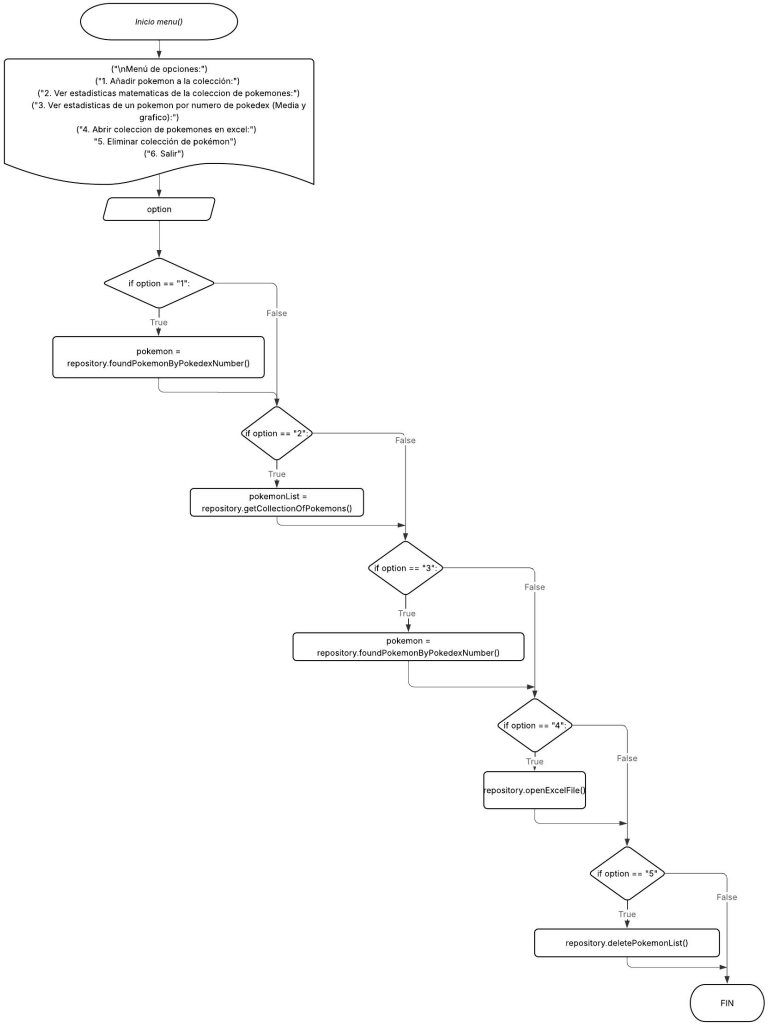

Inicio getPokemonFromFile(pokedex_number)
with open("pokemon_list.json", "r") as file:
pokemonList = json.load(file)
except (FileNotFoundError, json.JSONDecodeError):
True
pokemonList = []
False
if not is instance(pokemonList, list):
True
pokemonList = [pokemonList]
False
if not any(p['id'] == newPokemonInfo['id'] for p in pokemonList):
True
pokemonList.append(newPokemonInfo)
False
with open("pokemon_list.json", "w") as file:
json.dump(pokemonList, file, indent=4)
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
ruta_excel = os.path.join(BASE_DIR, "datos", "pokemon_list.xlsx")
FIN

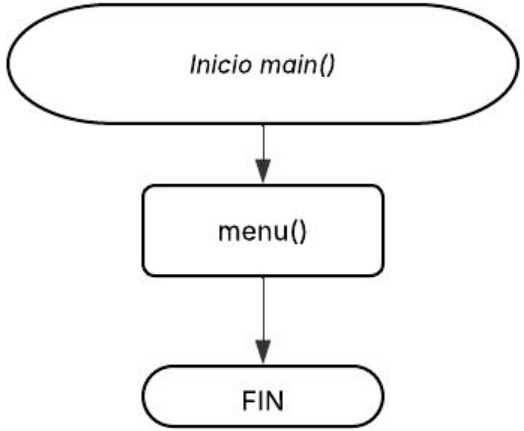

Inicio getCollectionOfPokemons()
with open("pokemon_list.json", "r") as file:
pokemonList = json.load(file)
if isinstance(pokemonList, dict)
True
return [pokemonList]
False
return pokemonList


Inicio addPokemonToExcel(pokemon, ruta_excel=ruta_excel)
if not os.path.exists(ruta_excel):
True
df = pd.DataFrame(columns=["id", "Nombre", "Tipo", "Movimientos", "Base Experience", "Estadisticas", "Altura", "Peso", "Color"])
df.to_excel(ruta_excel, index=False)
False
df_new = pd.DataFrame([pokemon])
df_existing = pd.read_excel(ruta_excel)
if not df_existing.empty and 'id' in df_existing.columns and (df_existing['id'] == pokemon['id']).any():
True
f"El Pokémon {pokemon['Nombre']} ya está en el archivo Excel."
False
df_combined = pd.concat([df_existing, df_new], ignore_index=True)
df_combined.to_excel(ruta_excel, index=False)
FIN


Inicio openExcelFile()
if os.path.exists(ruta_excel):
False
True
if platform.system() == "Windows":
False
Sí
os.startfile(ruta_excel)
if platform.system() == "Darwin":
False
True
subprocess.call(["open", ruta_excel])
subprocess.call(["xdg-open", ruta_excel])
"No existe"
FIN


Inicio deletePokemonList()
if os.path.exists(ruta_excel)
True
os.remove(ruta_excel)
False
if os.path.exists(ruta_json)
True
os.remove(ruta_json)
False
FIN


Inicio menu()
("\nMenú de opciones:")
("1. Añadir pokemon a la colección:")
("2. Ver estadisticas matematicas de la coleccion de pokemones:")
("3. Ver estadisticas de un pokemon por numero de pokedex (Media y grafico):")
("4. Abrir coleccion de pokemones en excel:")
"5. Eliminar colección de pokémon")
("6. Salir")
option
if option == "1":
True
False
pokemon = repository.foundPokemonByPokedexNumber()
if option == "2":
True
False
pokemonList = repository.getCollectionOfPokemons()
if option == "3":
True
False
pokemon = repository.foundPokemonByPokedexNumber()
if option == "4":
True
False
repository.openExcelFile()
if option == "5":
True
False
repository.deletePokemonList()
FIN


Inicio main()
menu()
FIN

ALGORITMOS

```
Algoritmo MenuPrincipal
Variables
   opcion: cadena
   pokemon: diccionario
   lista_pokemones: arreglo de diccionarios
    1.  Inicio
    2.      Repetir
    3.          Escribir("\nMenú de opciones:")
    4.          Escribir("1. Añadir pokemon a la colección")
    5.          Escribir("2. Ver estadísticas matemáticas de la colección")
    6.          Escribir("3. Ver estadísticas de un pokemon por número de pokedex")
    7.          Escribir("4. Abrir colección en Excel")
    8.          Escribir("5. Eliminar colección")
    9.          Escribir("6. Salir")
    10.
    11.         opcion <- Leer("Selecciona una opción: ")
    12.
    13.         Según opcion Hacer
    14.             Caso "1":
    15.                 pokemon <- foundPokemonByPokedexNumber()
    16.                 Si pokemon es NULO Entonces
    17.                     Escribir("Pokémon no encontrado.")
    18.                 Sino
    19.                     updateFilePokemonList(pokemon)
    20.                 Fin Si
    21.
    22.             Caso "2":
    23.                 lista_pokemones <- getCollectionOfPokemons()
    24.                 Si lista_pokemones es NULO Entonces
    25.                     Escribir("No hay pokémons en la colección.")
    26.                 Sino
    27.                     showPokemonCollectionStats(lista_pokemones)
    28.                 Fin Si
    29.
    30.             Caso "3":
    31.                 pokemon <- foundPokemonByPokedexNumber()
    32.                 Si pokemon es NULO Entonces
    33.                     Escribir("Pokémon no encontrado.")
    34.                 Sino
    35.                     showPokemonGraphicStats(pokemon)
    36.                 Fin Si
    37.
    38.             Caso "4":
    39.                 openExcelFile()
    40.                 Escribir("Abriendo archivo de Excel...")
    41.
    42.             Caso "5":
    43.                 Escribir("¿Estás seguro de eliminar la colección? (s/n): ")
```

```
            opcion_segura <- Leer()
            Si opcion_segura = "s" Entonces
               deletePokemonList()
               Escribir("Colección eliminada.")
            Sino
               Escribir("Operación cancelada.")
            Fin Si

         Caso "6":
            Escribir("Saliendo del programa...")
            Terminar

         Otro Caso:
            Escribir("Opción no válida.")
      Fin Según
   Hasta Que opcion = "6"
Fin
```

```
Algoritmo GestionPokemon
Variables
   lista_pokemones: arreglo de diccionarios
   promedio_stats: real
   moda_stats: real
   mediana_stats: real
   varianza_stats: real
   desviacion_stats: real
   Inicio
      lista_pokemones <- getCollectionOfPokemons()

      Si lista_pokemones es NULO Entonces
         Escribir("No hay Pokémones en la colección.")
         Terminar
      Fin Si

      promedio_stats <- getPromedyOfStatsList(lista_pokemones)
      moda_stats <- getModeOfPokemonAverageStats(lista_pokemones)
      mediana_stats <- getMeanOfCollectionOfPokemons(lista_pokemones)
      varianza_stats <- getVarianceOfCollectionOfPokemons(lista_pokemones)
      desviacion_stats <-
         getStandardDeviationOfCollectionOfPokemons(lista_pokemones)

      Escribir("Estadísticas de los Pokémones:")
      Escribir("Promedio de stats: ", promedio_stats)
      Escribir("Moda: ", moda_stats)
      Escribir("Mediana: ", mediana_stats)
      Escribir("Varianza: ", varianza_stats)
      Escribir("Desviación estándar: ", desviacion_stats)
   Fin
```

```
Algoritmo getPromedyOfStatsList
Variables
   lista_pokemones: arreglo de diccionarios
   promedios: arreglo de reales
   pokemon: diccionario
    1. Inicio
    2.     promedios <- [ ]
    3.
    4.     Para cada pokemon en lista_pokemones Hacer
    5.         promedio <- getPromedyOfStats(pokemon)
    6.         promedios.agregar(promedio)
    7.     Fin Para
    8.     Retornar promedios
    9. Fin


Algoritmo getPromedyOfStats
Variables
   pokemon: diccionario
   stats: diccionario
   valores_stats: arreglo de enteros
    1. Inicio
    2.     stats <- pokemon["Estadisticas"]
    3.     valores_stats <- valores(stats)  // Convertir valores del diccionario a lista
    4.     Retornar media(valores_stats)
    5. Fin


Algoritmo getModeOfPokemonAverageStats
Variables
listPokemons: arreglo de diccionarios
average_stats_values: arreglo de reales
    1. Inicio
    2. average_stats_values <- getPromedyOfStatsList(listPokemons)
    3. Retornar moda(average_stats_values)
    4. Fin

Algoritmo getMeanOfCollectionOfPokemons
Variables
listPokemons: arreglo de diccionarios
stats_values: arreglo de reales
    1. Inicio
    2. stats_values <- getPromedyOfStatsList(listPokemons)
    3. Retornar mediana(stats_values)
    4. Fin
```

```
Algoritmo getVarianceOfCollectionOfPokemons
Variables
listPokemons: arreglo de diccionarios
stats_values: arreglo de reales
    1. Inicio
    2. stats_values <- getPromedyOfStatsList(listPokemons)
    3. Retornar varianza(stats_values)
    4. Fin

Algoritmo getStandardDeviationOfCollectionOfPokemons
Variables
listPokemons: arreglo de diccionarios
stats_values: arreglo de reales
    1. Inicio
    2. stats_values <- getPromedyOfStatsList(listPokemons)
    3. Retornar desviacion_estandar(stats_values)
    4. Fin

Algoritmo media
Variables
   lista: arreglo de reales
   suma: real
   n: entero
    1. Inicio
    2.     suma <- 0
    3.     n <- longitud(lista)
    4.     Para cada valor en lista Hacer
    5.         suma <- suma + valor
    6.     Fin Para
    7.     Retornar suma / n
    8. Fin


Algoritmo mediana
   Variables
      lista: arreglo de reales
      lista_ordenada: arreglo de reales
      n: entero
    1.     Inicio
    2.         lista_ordenada <- ordenar(lista)
    3.         n <- longitud(lista_ordenada)
    4.         Si n % 2 = 1 Entonces
    5.             Retornar lista_ordenada[n//2]
    6.         Sino
    7.             Retornar (lista_ordenada[n//2 - 1] + lista_ordenada[n//2]) / 2
    8.         Fin Si
    9.     Fin
```

```
Algoritmo moda
   Varaibles
      lista: arreglo de reales
      max_repeticiones: entero
      moda: real
      repeticiones: entero
   1.    Inicio
   2.       max_repeticiones <- 0
   3.       moda <- NULO
   4.       Para i desde 0 hasta longitud(lista)-1 Hacer
   5.          repeticiones <- 0
   6.          Para j desde 0 hasta longitud(lista)-1 Hacer
   7.             Si lista[i] = lista[j] Entonces
   8.                repeticiones <- repeticiones + 1
   9.             Fin Si
   10.         Fin Para
   11.         Si repeticiones > max_repeticiones Entonces
   12.            max_repeticiones <- repeticiones
   13.            moda <- lista[i]
   14.         Fin Si
   15.      Fin Para
   16.      Retornar moda
   17.   Fin

Algoritmo varianza
   Variables
      lista: arreglo de reales
      varianza: real
      media: real
      suma_cuadrados: real
   1.    Inicio
   2.       varianza <- 0
   3.       media <- suma(lista) / longitud(lista)
   4.       suma_cuadrados <- 0
   5.       Para cada valor en lista Hacer
   6.          suma_cuadrados <- suma_cuadrados + (valor - media) ^ 2
   7.       Fin Para
   8.       Si longitud(lista) > 1 Entonces
   9.          varianza <- suma_cuadrados / (longitud(lista) - 1)
   10.      Sino
   11.         varianza <- 0
   12.      Fin Si
   13.      Retornar varianza
   14.   Fin
```

```
Algoritmo desviacion_estandar
  Variables
    lista: arreglo de reales
  1.    Inicio
  2.      Retornar varianza(lista) ^ 0.5
  3.    Fin

Algoritmo foundPokemonByPokedexNumber
  Variables
    pokedex_number: cadena
  1.    Inicio
  2.      Escribir("Introduce el número de pokedex: ")
  3.      pokedex_number <- Leer()
  4.      Retornar getPokemonByPokedexNumber(pokedex_number)
  5.    Fin

Algoritmo foundPokemonByPokedexNumber
  Variables
    pokedex_number: cadena
  1.    Inicio
  2.      Escribir("Introduce el número de pokedex: ")
  3.      pokedex_number <- Leer()
  4.      Retornar getPokemonByPokedexNumber(pokedex_number)
  5.    Fin

Algoritmo getPokemonByPokedexNumber
  Parámetros
    pokedex_number: cadena
  Variables
    pokemon: diccionario
  1.    Inicio
  2.      pokemon <- getPokemonFromFile(pokedex_number)
  3.      Si pokemon es NULO Entonces
  4.        pokemon <- getPokemonFromAPI(pokedex_number)
  5.      Fin Si
  6.      Retornar pokemon
  7.    Fin

Algoritmo requestPokemonGeneraDataJsonToApi
  Variables
    pokedex_number: cadena
    url: cadena
    data: diccionario
  1.    Inicio
  2.      url <- "https://pokeapi.co/api/v2/pokemon/" + pokedex_number
  3.      data <- RealizarPeticionHTTP(url)
  4.      Retornar data
  5.    Fin
```

```
Algoritmo requestPokemonCustomizedDataJsonToApi
  Variables
    pokedex_number: cadena
    url: cadena
1.        data: diccionario
2.    Inicio
3.        url <- "https://pokeapi.co/api/v2/pokemon-species/" + pokedex_number
4.        data <- RealizarPeticionHTTP(url)
5.        Retornar data
6.    Fin
Algoritmo getPokemonFromAPI
  Variables
    pokedex_number: cadena
    pokemonData: diccionario
    color: cadena
    listStats: arreglo de diccionarios
    stats_values: arreglo de enteros
    stats_names: arreglo de cadenas
    stats: diccionario
    pokemon: diccionario
1.    Inicio
2.        pokemonData <- requestPokemonGeneraDataJsonToApi(pokedex_number)
3.        color <-
      requestPokemonCustomizedDataJsonToApi(pokedex_number)['color']['name']
4.        listStats <- pokemonData['stats']
5.        stats_values <- [stat['base_stat'] para stat en listStats]
6.        stats_names <- [stat['stat']['name'] para stat en listStats]
7.        stats <- crearDiccionario(stats_names, stats_values)
8.        pokemon <- {
9.            "id": pokemonData['id'],
10.           "Nombre": pokemonData['name'],
11.           "Tipo": unir([t['type']['name'] para t en pokemonData['types']], " y "),
12.           "Movimientos": [m['move']['name'] para m en pokemonData['moves'][:5]],
13.           "Base Experience": pokemonData['base_experience'],
14.           "Estadisticas": stats,
15.           "Altura": pokemonData['height'],
16.           "Peso": pokemonData['weight'],
17.           "Color": color
18.       }
19.       updateFilePokemonList(pokemon)
20.       addPokemonToExcel(pokemon)
21.       Retornar pokemon
22.   Fin
```

```
Algoritmo getPokemonFromFile
   Variables
      pokedex_number: cadena
      pokemonList: arreglo de diccionarios
    1.    Inicio
    2.       Intentar
    3.          Abrir archivo ruta_json en modo lectura como file
    4.          pokemonList <- json.cargar(file)
    5.          Si esDiccionario(pokemonList) Entonces
    6.             Si cadena(pokemonList['id']) = pokedex_number Entonces
    7.                Retornar pokemonList
    8.             Sino
    9.                Retornar NULO
    10.            Fin Si
    11.         Fin Si
    12.         Para cada pokemon en pokemonList Hacer
    13.            Si cadena(pokemon['id']) = pokedex_number Entonces
    14.               Retornar pokemon
    15.            Fin Si
    16.         Fin Para
    17.      Manejar FileNotFoundError:
    18.         Escribir("Buscando en la api...")
    19.      Manejar JSONDecodeError:
    20.         Escribir("Error al decodificar el JSON.")
    21.      Retornar NULO
    22.   Fin

Algoritmo getCollectionOfPokemons
   Variables
      pokemonList: arreglo de diccionarios
    1.    Inicio
    2.       Intentar
    3.          Abrir archivo ruta_json en modo lectura como file
    4.          pokemonList <- json.cargar(file)
    5.          Si esDiccionario(pokemonList) Entonces
    6.             Retornar [pokemonList]
    7.          Sino
    8.             Retornar pokemonList
    9.          Fin Si
    10.      Manejar FileNotFoundError:
    11.         Escribir("Archivo no encontrado.")
    12.         Retornar NULO
    13.      Manejar JSONDecodeError:
    14.         Escribir("Error al decodificar el JSON.")
    15.         Retornar NULO
    16.   Fin
```

```
Algoritmo updateFilePokemonList
   Variables
      newPokemonInfo: diccionario
      pokemonList: arreglo de diccionarios
    1.      Inicio
    2.          Intentar
    3.              Intentar
    4.                  Abrir archivo ruta_json en modo lectura como file
    5.                  pokemonList <- json.cargar(file)
    6.                  Si no esArreglo(pokemonList) Entonces
    7.                      pokemonList <- [pokemonList]
    8.                  Fin Si
    9.              Manejar (FileNotFoundError, JSONDecodeError):
    10.                 pokemonList <- []
    11.             Fin Intentar
    12.             Si no existe p en pokemonList donde p['id'] = newPokemonInfo['id'] Entonces
    13.                 pokemonList.agregar(newPokemonInfo)
    14.             Fin Si
    15.             Abrir archivo ruta_json en modo escritura como file
    16.             json.guardar(pokemonList, file, indentar=4)
    17.         Manejar Error como e:
    18.             Escribir("Error al guardar el archivo: ", e)
    19.     Fin


Algoritmo addPokemonToExcel
   Variables
      pokemon: diccionario
      ruta_excel: cadena (opcional)
      df: DataFrame
      df_new: DataFrame
      df_existing: DataFrame
      df_combined: DataFrame
    1.      Inicio
    2.          Intentar
    3.              Si no existeArchivo(ruta_excel) Entonces
    4.                  df <- crearDataFrame(columnas=["id", "Nombre", "Tipo", "Movimientos",
        "Base Experience", "Estadisticas", "Altura", "Peso", "Color"])
    5.                  df.guardarExcel(ruta_excel)
    6.              Fin Si
    7.              df_new <- crearDataFrame([pokemon])
    8.              df_existing <- leerExcel(ruta_excel)
    9.              Si no df_existing.estaVacio() Y 'id' en df_existing.columnas Y (df_existing['id']
        = pokemon['id']).cualquiera() Entonces
    10.                 Escribir("El Pokémon ", pokemon['Nombre'], " ya está en el archivo Excel.")
    11.                 Retornar
    12.             Fin Si
    13.             df_combined <- concatenar(df_existing, df_new)
```

```
            df_combined.guardarExcel(ruta_excel)
        Manejar Error como e:
            Escribir("Error al manipular el archivo Excel: ", e)
    Fin
```

```
Algoritmo openExcelFile
    Inicio
        Si existeArchivo(ruta_excel) Entonces
            Según sistemaOperativo() Hacer
                Caso "Windows":
                    ejecutar(os.startfile(ruta_excel))
                Caso "Darwin":
                    ejecutar(["open", ruta_excel])
                Otro Caso:
                    ejecutar(["xdg-open", ruta_excel])
            Fin Según
        Sino
            Escribir("El archivo Excel no existe.")
        Fin Si
    Fin
```

```
Algoritmo deletePokemonList
    Inicio
        Si existeArchivo(ruta_excel) Entonces
            eliminarArchivo(ruta_excel)
            Escribir("Archivo Excel eliminado correctamente.")
        Fin Si
        Si existeArchivo(ruta_json) Entonces
            eliminarArchivo(ruta_json)
            Escribir("Archivo Json eliminado.")
        Fin Si
    Fin
```

```
Algoritmo getCollectionSize
  Variables
    coleccion: arreglo de diccionarios
    Inicio
        coleccion <- getCollectionOfPokemons()
        Si coleccion es NULO Entonces
            Retornar 0
        Sino
            Retornar longitud(coleccion)
        Fin Si
    Fin
```

```
Algoritmo showPokemonCollectionStats
Variables
    lista_pokemones: arreglo de diccionarios
    nombres: arreglo de cadenas
    colores: arreglo de cadenas
    promedios: arreglo de reales
    moda, mediana, varianza, desviacion: real
    1.  Inicio
    2.      Para cada pokemon en lista_pokemones Hacer
    3.          nombres.agregar(pokemon["Nombre"])
    4.          colores.agregar(pokemon["Color"])
    5.      Fin Para
    6.
    7.      promedios <- getPromedyOfStatsList(lista_pokemones)
    8.
    9.      moda <- getModeOfPokemonAverageStats(lista_pokemones)
    10.     mediana <- getMeanOfCollectionOfPokemons(lista_pokemones)
    11.     varianza <- getVarianceOfCollectionOfPokemons(lista_pokemones)
    12.     desviacion <- getStandardDeviationOfCollectionOfPokemons(lista_pokemones)
    13.
    14.     Crear figura con ejes (8x6)
    15.     Dibujar barras con (nombres, promedios, colores)
    16.     EscribirEnFigura("Moda: " & moda, posición=(0.1, 0.10))
    17.     EscribirEnFigura("Mediana: " & mediana, posición=(0.1, 0.06))
    18.     EscribirEnFigura("Varianza: " & varianza, posición=(0.5, 0.10))
    19.     EscribirEnFigura("Desviación estándar: " & desviacion, posición=(0.5, 0.06))
    20.
    21.     Mostrar gráfico
    22. Fin

Algoritmo showPokemonGraphicStats
Variables
    pokemon: diccionario
    stats_nombres: arreglo de cadenas
    stats_valores: arreglo de enteros
    angulos: arreglo de reales
    1.  Inicio
    2.      Si pokemon es NULO Entonces
    3.          Retornar
    4.      Fin Si
    5.
    6.      stats_nombres <- claves(pokemon["Estadisticas"])
    7.      stats_valores <- valores(pokemon["Estadisticas"])
    8.
    9.      angulos <- linspace(0, 2*pi, longitud(stats_nombres))
    10.     stats_valores += stats_valores[0]  // Cerrar el polígono
    11.     angulos += angulos[0]
    12.
```

13.
14. Crear figura polar (6x6)
15. Dibujar línea polar (angulos, stats_valores, color=pokemon["Color"])
16. Rellenar área (angulos, stats_valores, color=pokemon["Color"], opacidad=0.3)
17.
18.
19. EscribirTitulo(f"Estadísticas de {pokemon['Nombre']}")
20. EscribirLeyenda(f"Media: {getPromedyOfStats(pokemon)}")
21.
22. Mostrar gráfico
23. Fin

RESUMEN

DOCUMENTACIÓN PIA

Planteamiento del Problema y Análisis de Datos sobre la PokéAPI

Planteamiento del problema:
Pokémon es una de las franquicias más famosas del mundo, con videojuegos, series, películas y hasta cartas coleccionables. Detrás de todo esto hay muchos datos interesantes sobre los Pokémon: sus nombres, tipos, ataques y número en la Pokédex. En este proyecto, queremos analizar esa información para entender mejor cómo funcionan los Pokémon. Esto puede servir tanto para fans curiosos como para desarrolladores de juegos o incluso investigadores. Al organizar y estudiar estos datos, podemos descubrir patrones útiles o simplemente aprender más sobre el pokémon que desees.

Relevancia:
Analizar los datos de los Pokémon es útil porque ayuda a entender mejor el juego y a tomar decisiones más acertadas, como elegir qué Pokémon usar en una batalla. También puede servir para crear apps o herramientas que mejoren la experiencia de los jugadores. Conocer datos como el nombre, tipo, número en la Pokédex y ataques permite a los fans y desarrolladores aprovechar mejor la información que ofrece el mundo Pokémon.

Datos a obtener:
Los datos que vamos a utilizar son:

- Nombre del Pokémon: Es el identificador principal de cada Pokémon
- Número en la Pokédex: El número en la Pokédex es un identificador único que asigna un valor específico a cada Pokémon dentro de la enciclopedia digital de la franquicia.
- Tipo de Pokémon: Los Pokémon se dividen en diferentes tipos (por ejemplo, agua, fuego, tierra, eléctrico, etc.), y el tipo tiene una gran influencia en el rendimiento de los Pokémon durante las batallas.
- Ataques o Movimientos: Cada Pokémon tiene una lista de ataques o movimientos que puede aprender y usar en combate. Estos ataques pueden variar en poder y efectividad dependiendo del tipo de Pokémon y la situación en la que se encuentre.
- Altura: Sirve para saber qué tan grande es el Pokémon. Es un dato curioso pero también útil para compararlos.
- Peso: Nos muestra cuánto pesa el Pokémon, lo cual puede influir en ciertas habilidades o ataques dentro del juego.

Descripción API:
Hemos elegido utilizar la PokéAPI, una API gratuita y de código abierto que ofrece un acceso completo y detallado a la base de datos de Pokémon. PokéAPI se ha convertido en una de las fuentes más confiables para obtener información sobre los Pokémon, y su facilidad de uso la hace accesible para desarrolladores y aficionados por igual.

Algunas de sus caracteristicas son que la API proporciona información sobre más de 1,000 Pokémon, abarcando desde los Pokémon de la primera generación hasta los más recientes, no solo ofrece datos básicos, sino también información detallada, como estadísticas, movimientos, evoluciones y mucho más. Esto incluye aspectos como el poder de cada ataque, la efectividad de los movimientos contra otros tipos de Pokémon y las habilidades especiales que cada Pokémon posee, La API también incluye imágenes oficiales (sprites) de cada Pokémon, da información sobre los tipos de Pokémon y cómo estos interactúan con otros tipos, ofrece información detallada sobre movimientos que puede aprender a medida que sube de nivel o a través de técnicas especiales, habilidades y efectos y por último proporciona información sobre las cadenas evolutivas, es decir, cómo un Pokémon puede evolucionar a otro, y en algunos casos, cómo puede haber múltiples formas de evolución dependiendo de condiciones especiales.

Descripción de la estructura de datos utilizada:
Para este proyecto utilizamos las estructuras de diccionarios y listas. Las listas se usaron para almacenar múltiples elementos, como todos los Pokémon (todos_pokemon), sus habilidades (todas_habilidades), y los datos individuales como tipos, experiencia, altura y peso. Cada Pokémon y cada habilidad se representaron mediante diccionarios, ya que esta estructura permite almacenar pares clave-valor que describen sus características (por ejemplo, nombre, tipo, movimientos, etc.). También usamos las listas dentro de los diccionarios, como en el caso de los movimientos del Pokémon, que son varios por cada uno. Esto hace que sea más fácil buscar información del API y/o filtrar con expresiones regulares.

Justificación del tratamiento de datos aplicados:
Usamos listas y diccionarios en este código porque son fáciles de usar y nos permiten tener los datos bien ordenados. Con los diccionarios pudimos guardar toda la información de cada Pokémon o habilidad en un solo lugar. Las listas nos ayudaron a juntar muchos Pokémon y también a hacer cálculos como la moda, media, mediana, varianza y desviación estándar. Además, al tener los datos organizados en estas estructuras, fue más sencillo guardarlos en archivos JSON y trabajar con ellos para sacar conclusiones.

GUIÓN

Guión

Rebeca: Hola, hoy mi compañero Angel y yo Rebeca les presentaremos el proyecto Pokémon. Es una aplicación que permite crear y gestionar una colección personalizada de Pokémon, usando datos reales obtenidos desde internet. A través de esta herramienta podemos visualizar estadísticas y gráficos para analizar el rendimiento de los Pokémon guardados.

Rebeca: Para comenzar, veamos qué funciones principales ofrece esta aplicación. Permite buscar Pokémon por su número de Pokédex, guardar sus datos localmente, verlos en un archivo de Excel y generar visualizaciones con estadísticas importantes como la media, moda, mediana, varianza y desviación estándar. Además, todo esto se maneja desde un menú interactivo muy sencillo para el usuario.

Rebeca: Ahora que entendemos lo que hace el programa, es importante conocer su estructura. Está dividido en dos partes principales: el archivo 'PIA_script.py', que contiene la lógica de la interfaz y los gráficos, y 'PIA_modulo.py', donde se encuentra toda la lógica del programa, incluyendo el acceso a la API, guardado de datos y cálculos estadísticos.

Rebeca: Empezamos con la función central del sistema: `menu()`. Esta función muestra las opciones disponibles para el usuario y se encarga de ejecutar lo que éste elija, como buscar un Pokémon, generar gráficos o abrir el archivo Excel. En resumen, controla el flujo completo

Rebeca: Una vez dentro del menú, si el usuario elige añadir un Pokémon, se llama a la función `foundPokemonByPokedexNumber()`. Esta función solicita el número de Pokédex, intenta buscar ese Pokémon en el archivo local, y si no lo encuentra, lo busca directamente en la PokéAPI. Luego muestra los datos en pantalla y guarda el Pokémon en nuestra colección.

Angel: Después de tener varios Pokémon en la colección, es útil poder analizarlos. La función `showPokemonCollectionStats()` genera una gráfica de barras con el promedio de estadísticas de cada Pokémon. También calcula y muestra medidas como moda, mediana, varianza y desviación estándar. Todo esto con ayuda de funciones matemáticas definidas en el módulo.

Angel: En caso de que queramos ver con más detalle las estadísticas de un solo Pokémon, utilizamos la función `showPokemonGraphicStats()`. Esta función crea un gráfico tipo radar que permite comparar las distintas estadísticas base del Pokémon visualmente. Es una forma muy intuitiva de identificar fortalezas y debilidades.

Angel: Toda esta información no sería útil si no pudiéramos guardarla. Para eso usamos `updateFilePokemonList()` y `addPokemonToExcel()` para almacenar los datos en archivos JSON y Excel, respectivamente. Más adelante, si queremos trabajar con esa colección, podemos usar `getCollectionOfPokemons()` para cargar todos los Pokémon guardados.

Angel: Para realizar los análisis estadísticos, se usan funciones como `getPromedyOfStats()`, `media()`, `moda()`, `mediana()`, `varianza()` y `desviacion_estandar()`. Estas se encargan de procesar las estadísticas base de los Pokémon y permiten crear los gráficos y cálculos que ya vimos anteriormente.

Angel: Finalmente, hay algunas funciones complementarias que mejoran la experiencia del usuario. `openExcelFile()` permite abrir el archivo con la colección directamente desde el programa. `deletePokemonList()` elimina por completo la colección, y `getCollectionSize()` simplemente nos indica cuántos Pokémon hemos guardado hasta ahora. Estas funciones son pequeñas pero muy prácticas.

Proyecto Pokémon

Manejo de colección de Pokémon y análisis estadístico

¿Qué hace el programa?

- Permite buscar Pokémon por número de Pokédex

- Guarda datos en Excel

- Muestra gráficas de estadísticas

- Calcula media, moda, mediana, varianza y desviación estándar

Estructura del código

PIA_SCRIPT.PY:

INTERFAZ DE USUARIO Y GRÁFICOS

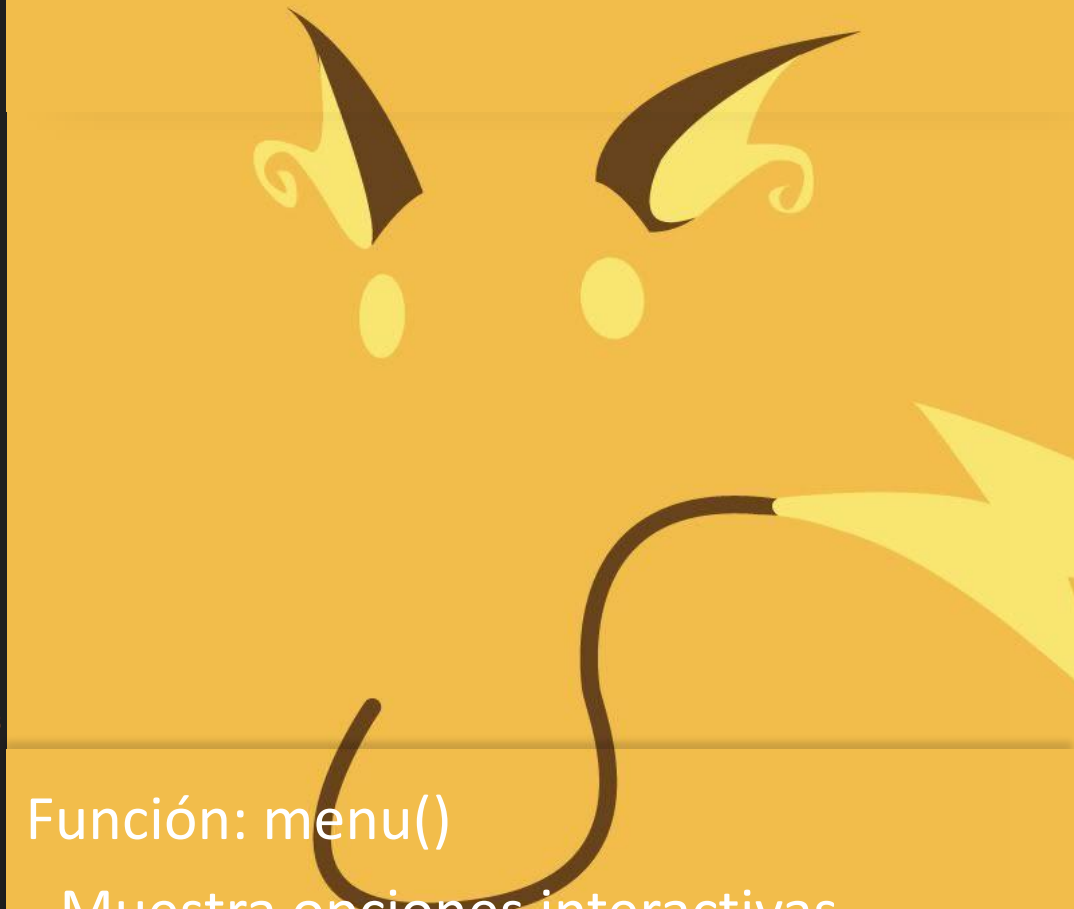
PIA_MODULO.PY:

LÓGICA DEL PROGRAMA Y MANEJO DE ARCHIVOS

```python
def menu():
    while True:
        print("\nMenú de opciones:")
        print("1. Añadir pokemon a la colección:")
        print("2. Ver estadisticas matematicas de la coleccion d
        print("3. Ver estadisticas de un pokemon por numero de p
        print("4. Abrir coleccion de pokemones en excel:")
        print("5. Eliminar colección de pokémon")
        print("6. Salir")
        option = str(input("Selecciona una opción: "))

        if option == "1":
            pokemon = repository.foundPokemonByPokedexNumber()
            if pokemon is None:
                print("Pokémon no encontrado.")
                continue
            repository.updateFilePokemonList(pokemon)
        elif option == "2":
            pokemonList = repository.getCollectionOfPokemons()
            if pokemonList is None:
                print("No hay pokémons en la colección.")
                continue
            showPokemonCollectionStats(pokemonList)
        elif option == "3":
            pokemon = repository.foundPokemonByPokedexNumber()
            if pokemon is None:
                print("Pokémon no encontrado.")
                continue
            showPokemonGraphicStats(pokemon)
        elif option == "4":
            repository.openExcelFile()
            print("Abriendo archivo de Excel...")
        elif option == "5":
            safeOption = str(input("¿Estás seguro de que quieres
            if safeOption.lower() != "s":
                print("Operación cancelada.")
                continue
            print("Eliminando colección de pokémon...")
            repository.deletePokemonList()
        elif option == "6":
            print("Saliendo del programa...")
            break
        else:
            print("Opción no válida")
```

Menú del programa



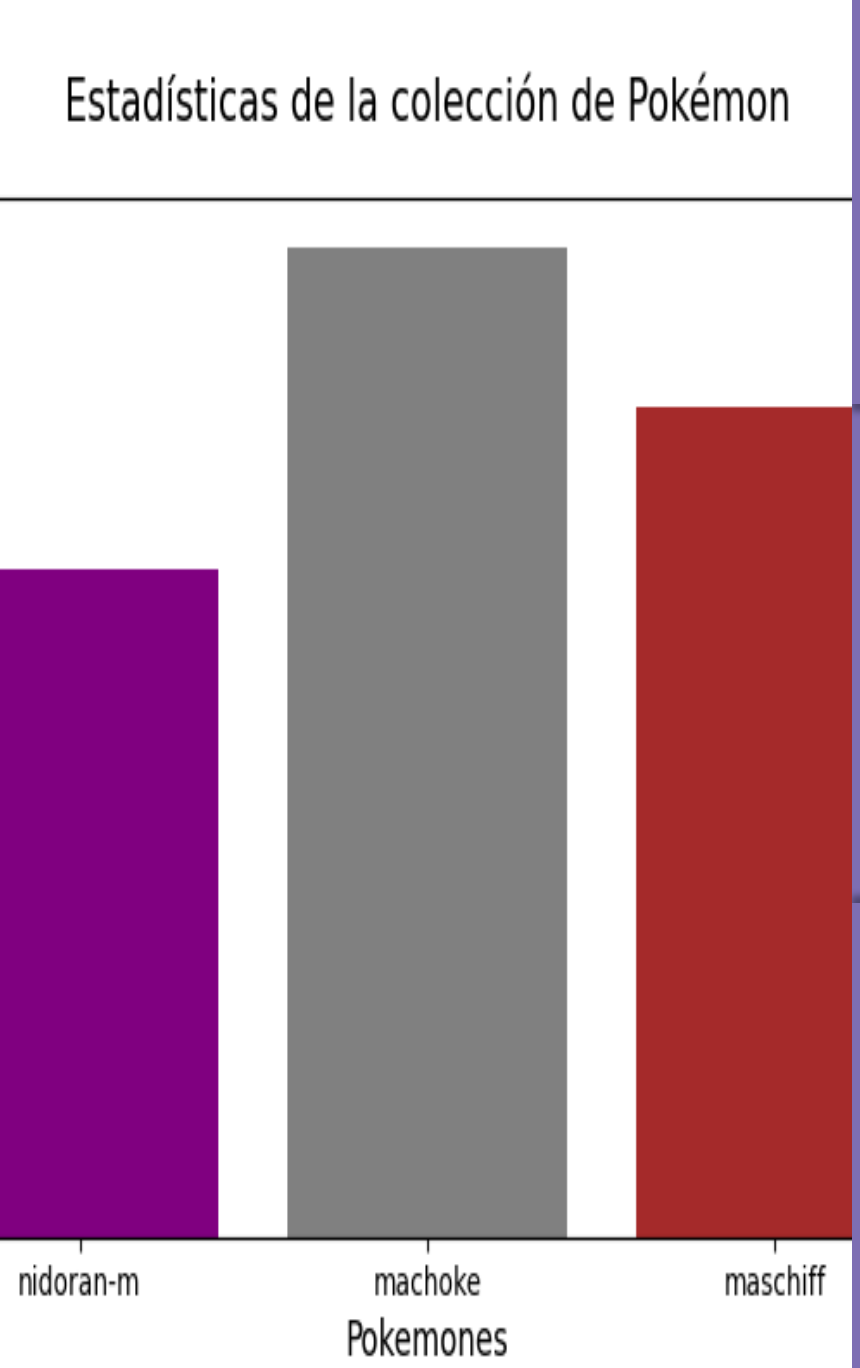
Función: menu()

- Muestra opciones interactivas
- Controla todo el flujo del programa

Añadir Pokémon

Función:
foundPokemonByPokedexNumber()

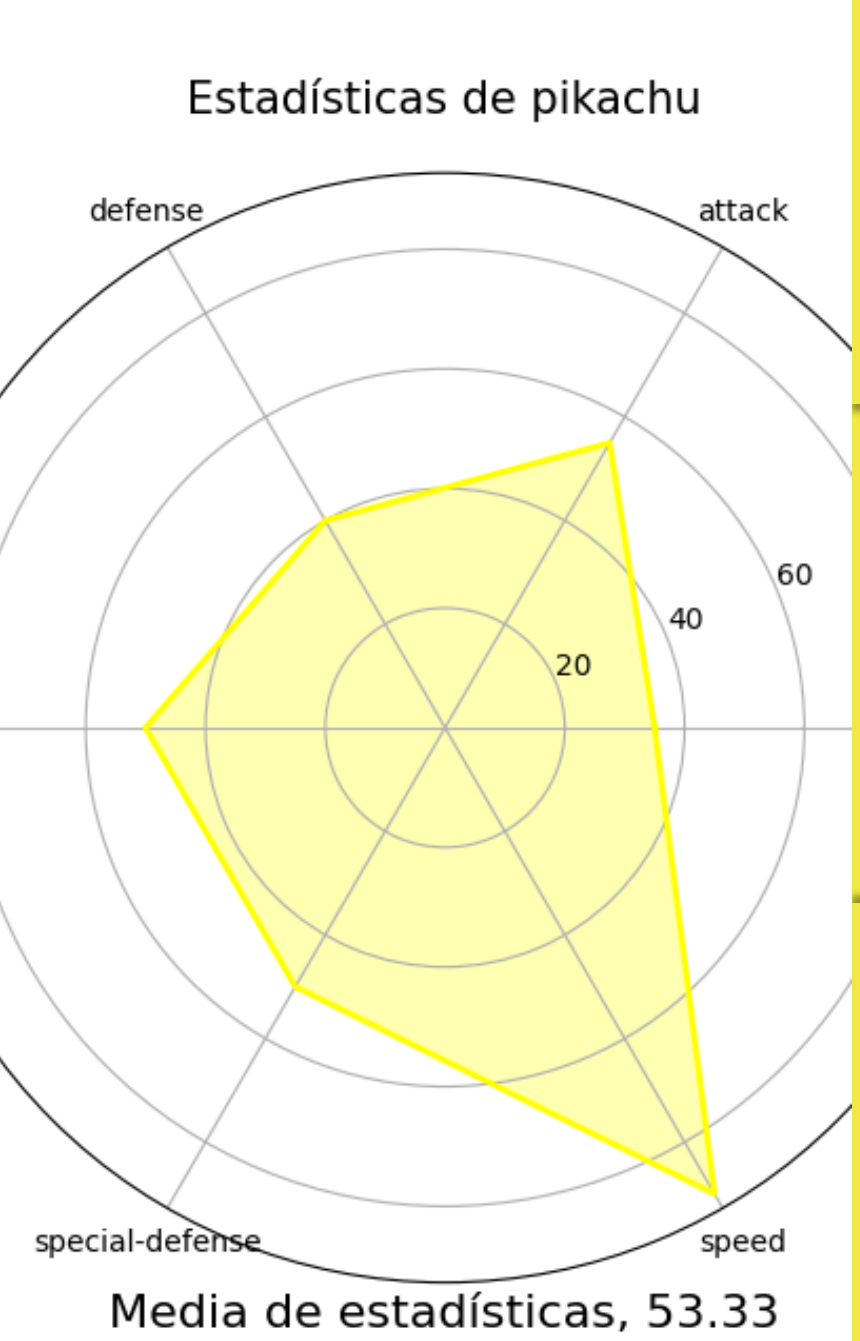
- Solicita número de Pokédex
- Busca en archivo o API
- Muestra y guarda los datos del Pokémon



Estadísticas de la colección

Función:
showPokemonCollectionStats()

- Gráfica de barras por promedio de poder
- Muestra moda, mediana, varianza y desviación



Estadísticas de un Pokémon

Función: showPokemonGraphicStats()

- Gráfico tipo radar con estadísticas base
- Visualiza en qué destaca un Pokémon

Guardado y lectura de datos

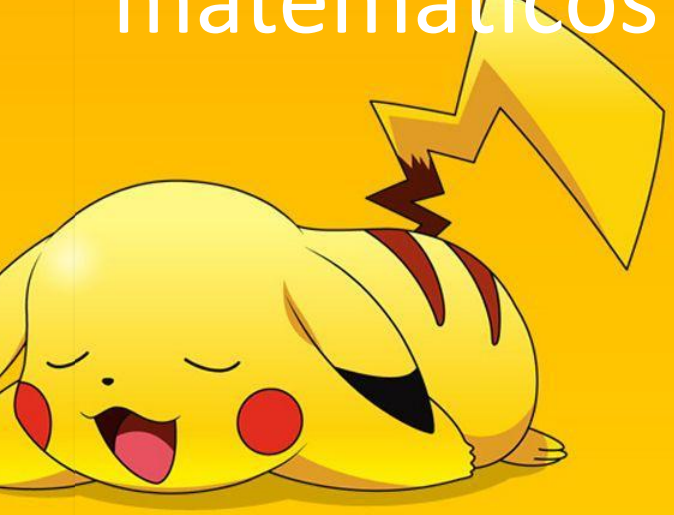
Funciones:

- updateFilePokemonList(): guarda en JSON

- addPokemonToExcel(): guarda en Excel

- getCollectionOfPokemons(): lee la colección

Cálculos matemáticos



Funciones:

- getPromedyOfStats(), media(), moda(), mediana()

- varianza(), desviacion_estandar()

- Calculan las estadísticas clave de los Pokémon

Extras útiles

Funciones:

- openExcelFile(): abre el archivo
- deletePokemonList(): elimina la colección
- getCollectionSize(): cuenta cuántos Pokémon hay