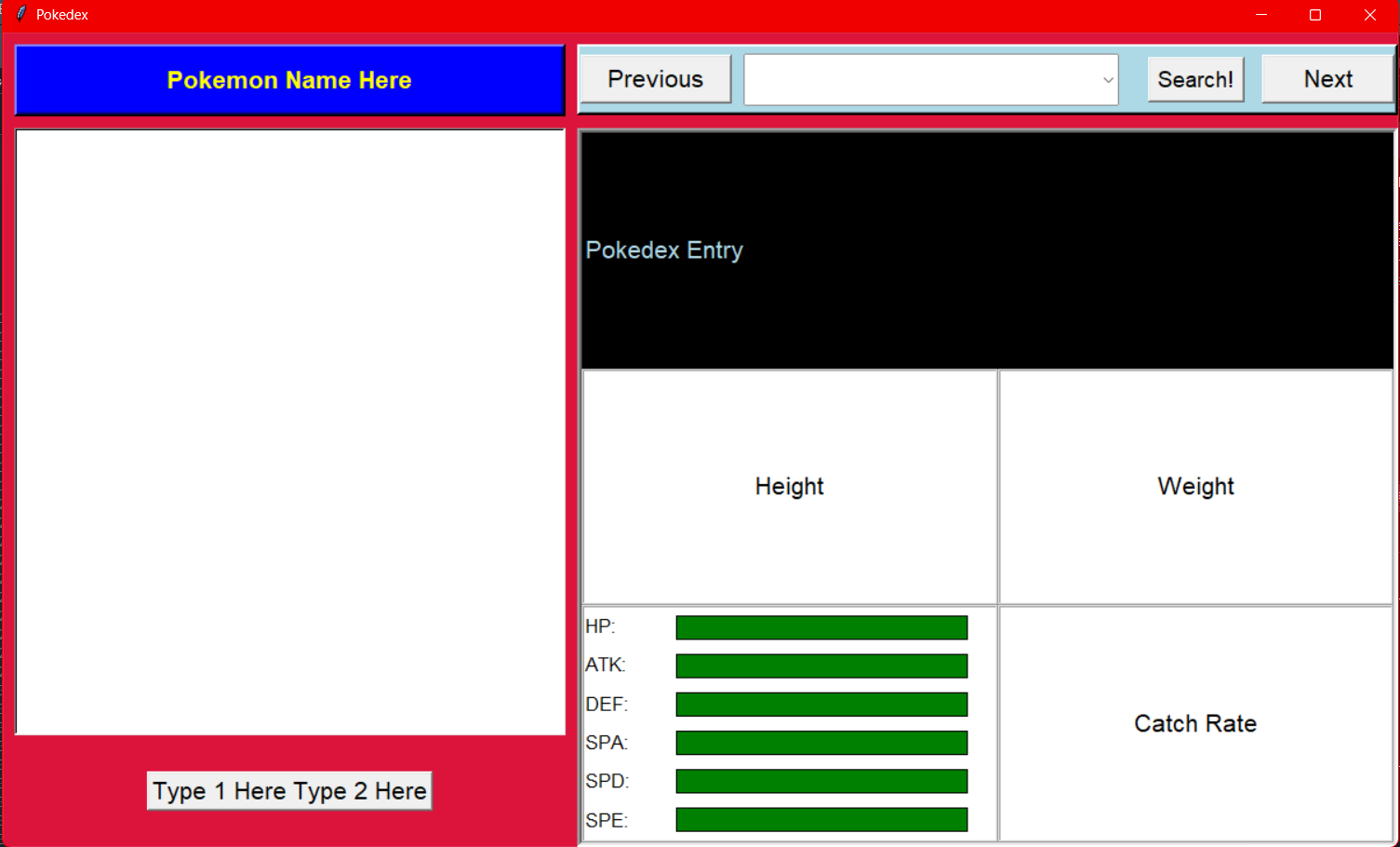
Pokedex Documentation

Abstract

I have made a Pokedex application using Tkinter and the Pokemon Api. This application can be used to search any pokemon registered in the Api and its registered types, stats, catch rate and bios. Its core functionality is to be a functioning digital encyclopedia of all registered pokemon in the API and help users find the pokemon they are intended to search.

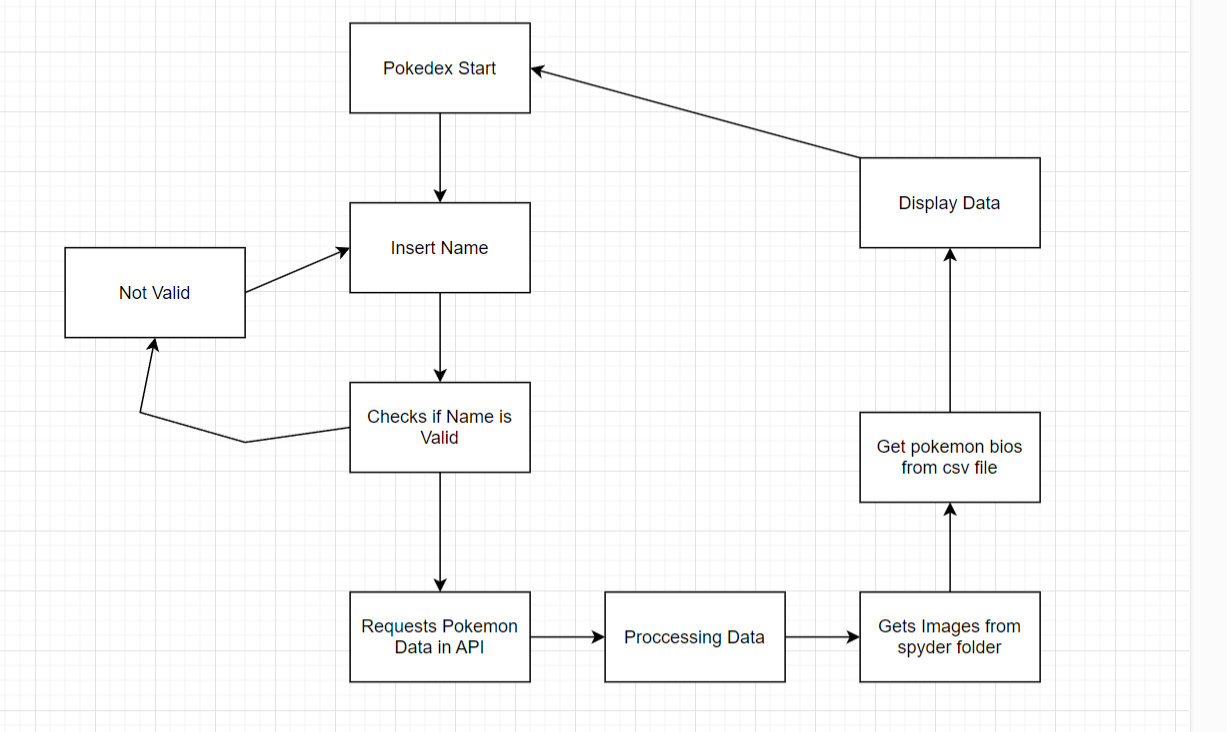
Project Plan

My plan is to tackle the process step by step. The first plan is the layout of the window in which the program resides. First I designed the window on how it will look when the user opens it.

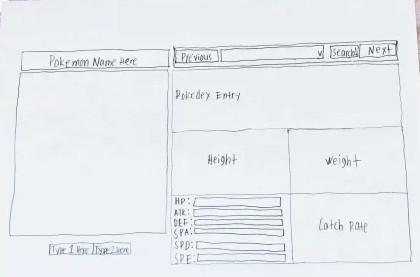


The Pokemon’s image will appear on the left side of this open window with their names displayed on the top right and their corresponding types underneath the image. The search function is on the right side and it has buttons so the user can pan left or right to see the next pokemon. The Pokemons bio will be shown in this black box and below it has the 4 corresponding information such as the Height and Weight of the pokemon, its catch rate and finally its base stats. After finishing up the design of the window it is time to code the program’s functionality. First is to implement the window design in python. After coding in the window, configuring the buttons and other essential things. Next is testing the software if it works. For the images to work I have to implement the program to read what’s in the spyder folder for which it contains both the images of the pokemon and its types. The descriptions for each pokemon are stored inside a csv file and it calls upon a specific description everytime a pokemon is displayed.

Evidence of Design



This is how the program works in a flowchart. With it it starts off with the user opening the application. Afterwards the user is prompted to insert the name of the pokemon. It checks whether the input the user types in the search bar is valid or not within the API. If it is not valid the program is prompted to ask the user for another input again. But if the input is valid, it will request for the data in the api and start processing it. It also gets images and data from specific files such as the pokemon’s images and their entries in the csv files. Afterwards, it displays the data on the window and the user can search for other pokemon again and the cycle repeats.



This is the wireframe of the application.

Pseudo-Code

Technical Description and Walkthrough

This Python code creates a graphical user interface (GUI) for a Pokémon data viewer using the tkinter library. Once a Pokémon dataset has been loaded from a CSV file, users can search for Pokémon, view detailed information about them, and browse through different entries. Included in the dataset are Pokémon’s names, stats (e.g., attack, HP), height, weight, types and catch rate.

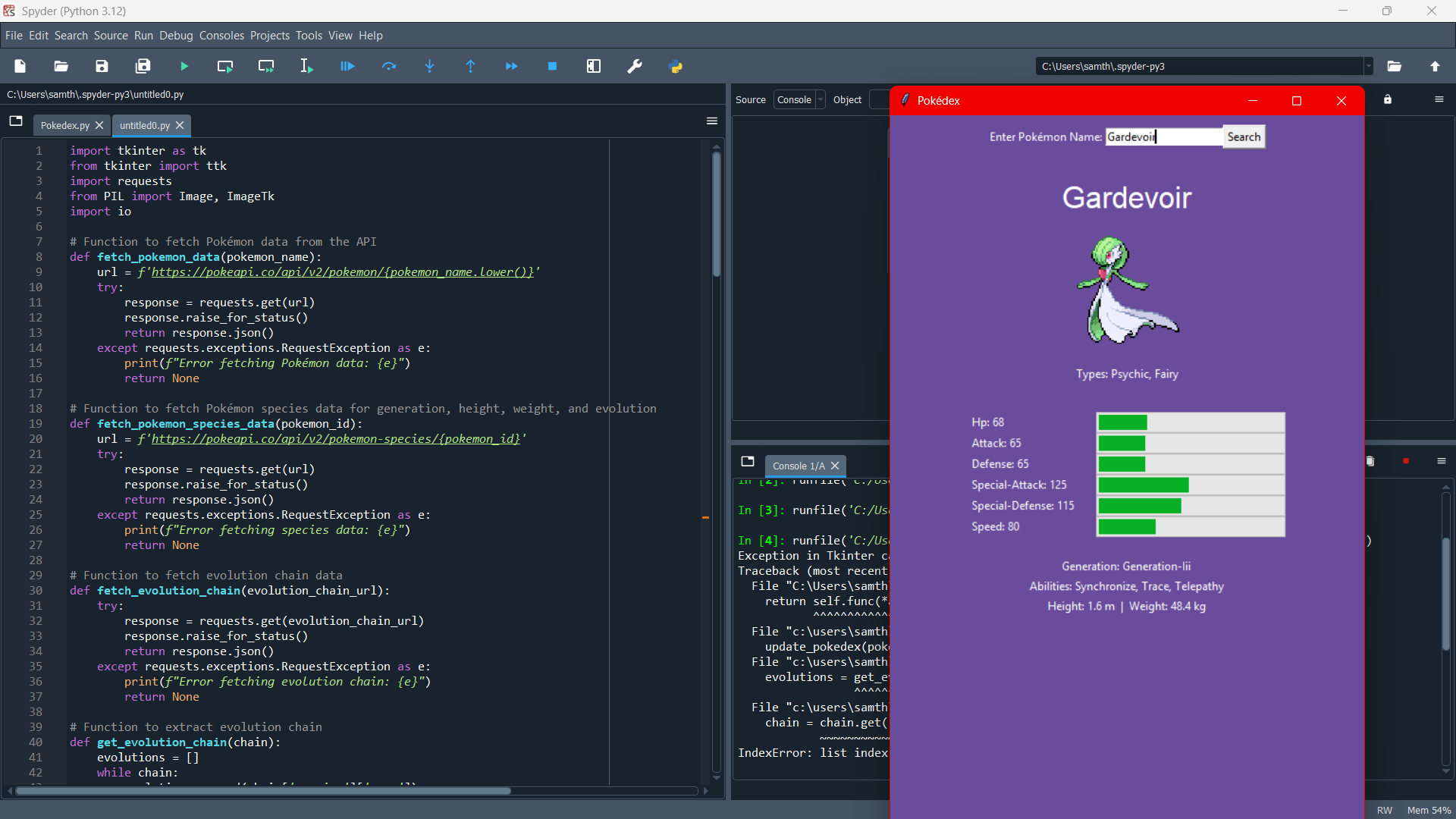
Users can use a search box to find Pokémon by name, and there are buttons to navigate through the list. When a Pokémon is selected, its details are displayed through the GUI. Name, height, weight, and catch rate are all shown in the text. Images of the Pokémon and their varieties are displayed using labels. Attack and HP are represented by color-coded bars, and the length of each bar reflects the Pokémon's stat values. The color of the bars varies according to the stat value; higher values are displayed in green, and lower values are displayed in red.

The app also features a search feature that ensures that only relevant results show up by filtering Pokémon names as you type. The select\_pokemon function changes all of the displayed data and visual elements based on the Pokémon that is selected.

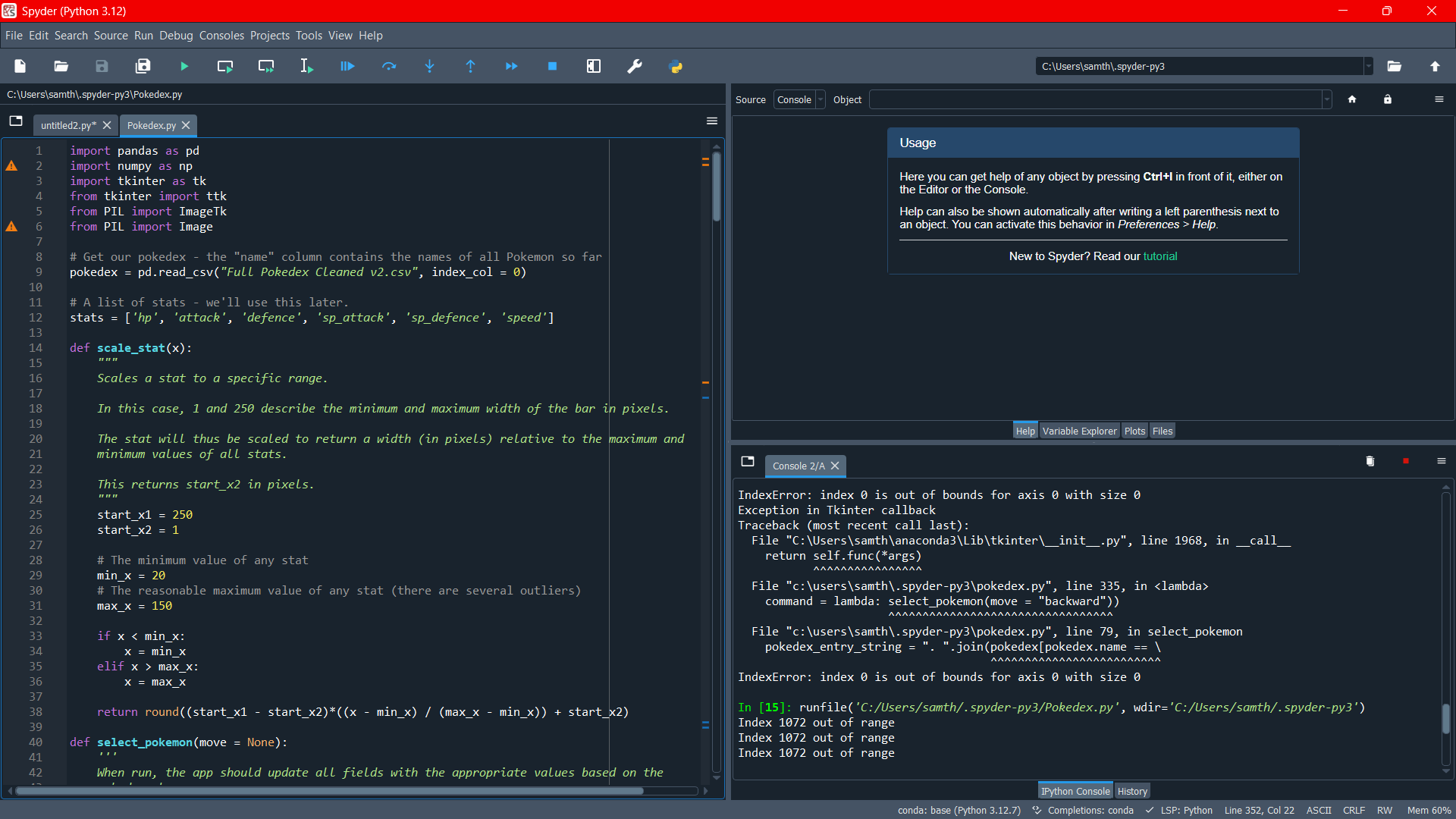
Walkthrough Video: <https://youtu.be/2rCk_oHUKAU?si=TJC1Yytv_TStzIaD>

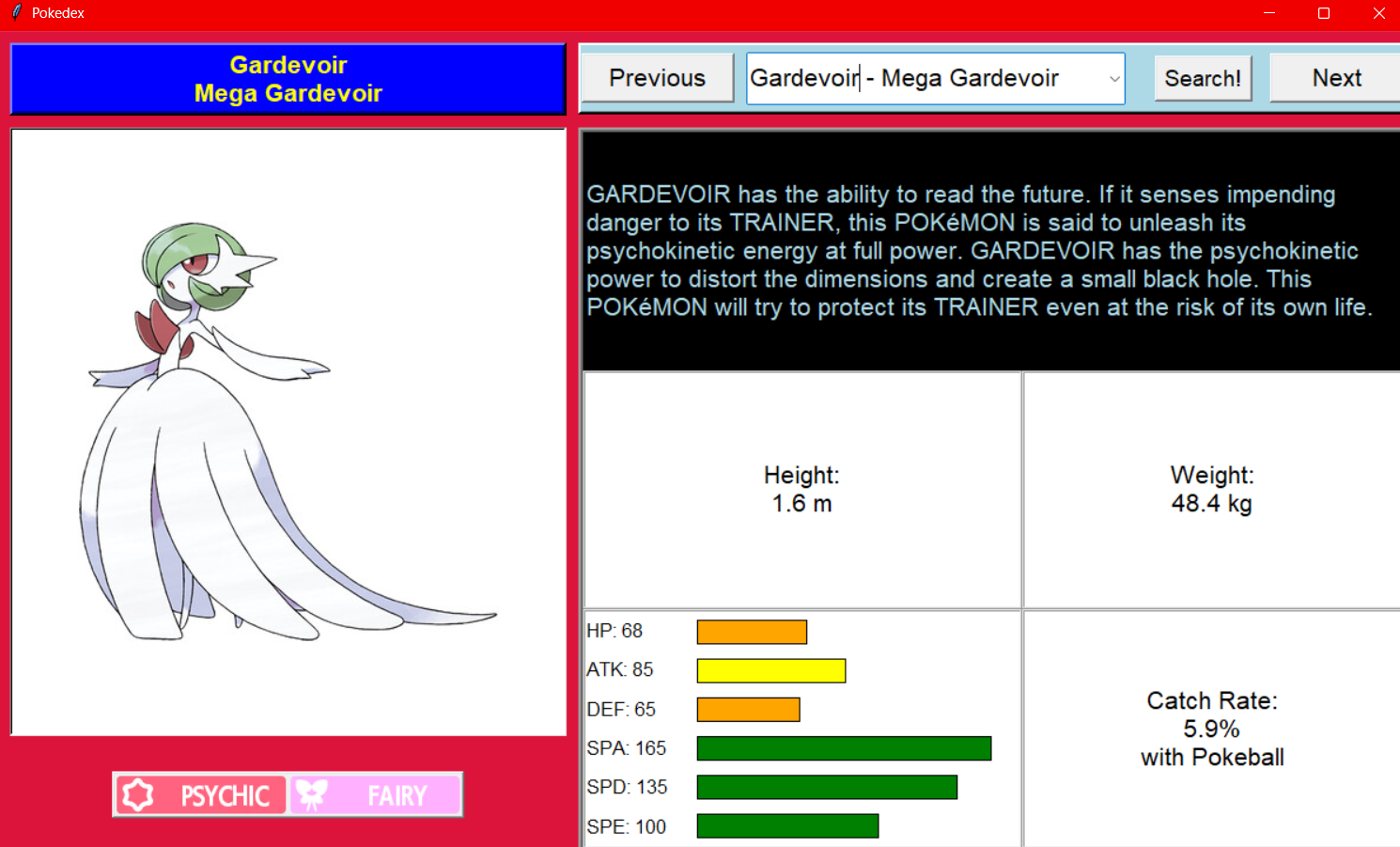
Testing

Beta Build



Modern Build





Critical Reflection

This code, which was created using Python's Tkinter and other libraries like Pandas and PIL, represents the graphical user interface (GUI) for a Pokedex application. The user is presented with an interactive display where they can examine various Pokémon, their stats, and other details after loading a CSV file containing Pokémon data. Despite the design's effectiveness, there are a few advantages and disadvantages to consider.

The smart choice to load and alter the Pokedex data using Pandas allows the application to manage complicated datasets with ease. The structure is well-organized, with specific roles for each task, like selecting Pokémon and scaling stats for visual representation. The reliance on "trial and error" to ascertain particular values, like bar coordinates, raises concerns about a lack of design precision, and these arbitrary constants may restrict the application's scalability or make it more challenging to maintain in the future. This could be reduced by adding more movement to the design or by defining layout attributes more systematically.

The interface itself is visually structured, with multiple frames (name, picture, type, information, etc.) logically arranged within the Tkinter window. The graphical elements, which include Pokémon images and type icons, provide users with an engaging experience. Despite being consistent, the color scheme could be further enhanced for accessibility, especially for color-blind users. The value-based color coding of stat bars, for example, may be confusing to those who cannot distinguish certain colors, which could restrict the application's inclusivity.

The management of errors is another area that requires improvement. Although some user input errors (such as going forward or backward past the Pokémon list) are taken into account by the code, the user is not provided with much feedback if something goes wrong. A more thorough approach would include clearer error messages or various methods for the user to correct errors.

The combination of text and graphical representations of Pokémon stats is an interesting feature, even though it is clear that the program's current stat bar rendering could be improved. The "scale\_stat" function's use of hardcoded boundaries (such as minimum and maximum stat values) can be problematic, especially when working with different datasets. Additionally, by doing away with the necessity of manually updating each component (such as the Pokémon's name, type, and stats) each time the user interacts with the GUI, a more modular approach might maximize maintainability.

Finally, the well-designed search function allows users to dynamically filter Pokémon based on their input. It could be further enhanced by handling scenarios in which no matches are found, improving the search results, or offering suggestions as the user types. At the moment, the user might not realize right away that no matches are available, especially if they type in a misspelled name.

In conclusion, the code demonstrates a strong foundation for a Pokedex application; however, it could be enhanced in terms of accessibility, scalability, error management, and design precision.

Appendix

import pandas as pd

import numpy as np

import tkinter as tk

from tkinter import ttk

from PIL import ImageTk

from PIL import Image

# Get our pokedex - the "name" column contains the names of all Pokemon so far

pokedex = pd.read\_csv("Full Pokedex Cleaned v2.csv", index\_col = 0)

# A list of stats - we'll use this later.

stats = ['hp', 'attack', 'defence', 'sp\_attack', 'sp\_defence', 'speed']

def scale\_stat(x):

"""

Scales a stat to a specific range.

In this case, 1 and 250 describe the minimum and maximum width of the bar in pixels.

The stat will thus be scaled to return a width (in pixels) relative to the maximum and

minimum values of all stats.

This returns start\_x2 in pixels.

"""

start\_x1 = 250

start\_x2 = 1

# The minimum value of any stat

min\_x = 20

# The reasonable maximum value of any stat (there are several outliers)

max\_x = 150

if x < min\_x:

x = min\_x

elif x > max\_x:

x = max\_x

return round((start\_x1 - start\_x2)\*((x - min\_x) / (max\_x - min\_x)) + start\_x2)

def select\_pokemon(move = None):

'''

When run, the app should update all fields with the appropriate values based on the

pokedex above.

Note that this function is specifically designed for the dataframe generated

from the .csv file loaded above - your function may be different depending on

your data source.

'''

# If we pressed the right (forward) or left (backward) buttons,

# we want to change Pokemon.

try:

if move == 'forward':

# Move the current selection forward by one item.

search\_box.current(search\_box.current() + 1)

# Get the name of the Pokemon in the search\_box

selected = search\_box.get()

elif move == 'backward':

# Move the current selection forward by one item.

search\_box.current(search\_box.current() - 1)

selected = search\_box.get()

else:

selected = search\_box.get()

# If there's an error (generally related to pressing forward on the last pokemon

# or backward on the first pokemon), we'll just get the current Pokemon

except Exception as e:

# Print the error to see what went wrong

print(e)

selected = search\_box.get()

##### Fill in text-based information #####

# The name of the Pokemon

# note that we split the name of the Pokemon to a new line if its too long

name\_label["text"] = "\n".join(selected.split(" - "))

# The Pokedex entry

pokedex\_entry\_string = ". ".join(pokedex[pokedex.name == \

selected]["pokedex\_entry"].values[0].split(". ")[:4])

# Add the period to the end of the sentence if there isn't one.

if pokedex\_entry\_string[-1] != "." and pokedex\_entry\_string[-1] != "!":

pokedex\_entry\_string += "."

pokedex\_entry["text"] = pokedex\_entry\_string

# The height and weight in real units

height\_entry["text"] = "Height:\n" + str(pokedex[pokedex.name == \

selected]["height"].values[0]) + " m"

weight\_entry["text"] = "Weight:\n" + str(pokedex[pokedex.name == \

selected]["weight"].values[0]) + " kg"

# The catch rate

# Note that there are no "catch rates" for Pokemon Arceus - these are recorded as

# N/A in my database

if pokedex[pokedex.name == selected]["catch\_rate"].isna().all():

catch\_entry["text"] = "Catch Rate:\nNo known data"

else:

catch\_entry["text"] = "Catch Rate:\n" + str(pokedex[pokedex.name == \

selected]["catch\_rate"].values[0]) \

+ "%\nwith Pokeball"

##### Fill in Picture-based information #####

# Display the Pokemon's image

label\_image = ImageTk.PhotoImage(file = \

f"Pokemon Pictures/{pokedex[pokedex.name == selected].index[0] + 1}.png")

pic\_label.config(image = label\_image)

# Remember to keep a reference to the image or it won't display!

pic\_label.image = label\_image

# Get and display the appropriate image for the pokemon's type

type1label["image"] = type\_dict[pokedex[pokedex.name == selected]["type1"].values[0]]

# Note that many Pokemon have a "null" second typing. In such cases, we turn off the type2label

# using grid\_forget

if pokedex[pokedex.name == selected]["type2"].values[0] == "none":

type2label.grid\_forget()

else:

type2label["image"] = type\_dict[pokedex[pokedex.name == selected]["type2"].values[0]]

type2label.grid(row = 0, column = 1)

##### Fill in Stats #####

# Delete all existing bars in the canvas

for widget in stat\_frame.winfo\_children():

widget.destroy()

# Recreate the Canvas - a Canvas is basically an area where we can draw simple shapes

draw\_box = tk.Canvas(stat\_frame, height = 100, width = 275, bg = "white",

borderwidth = 0, highlightthickness = 0)

draw\_box.grid(row = 0, rowspan = 6, column = 2, sticky = "nsew")

# These are values that I have determined using trial and error to get the best looking

# bars. These values describe the coordinates of the top left and bottom right corners of

# the bar I want to draw.

start\_x1 = 0

start\_y1 = 7

start\_x2 = 250

start\_y2 = 27

bar\_height = 20 # height of each stat bar

bar\_step = 13 # distance between bars

for row, stat in enumerate(stat\_block):

# Get the value of the stat

stat\_value = pokedex[pokedex.name == selected][stats[row]].values[0]

# Recreate the text box containing the appropriate value

text\_box = tk.Label(stat\_frame,

# Add the value of the appropriate stat

text = stat + f" {stat\_value}",

font = ("Futura", 12),

anchor = "w",

bg = "white",

justify = tk.LEFT,

borderwidth = 0)

text\_box.grid(row = row, column = 0, sticky = "nsw")

# Higher stats should be green, lower stats should be red.

if stat\_value <= 50:

fill = "red"

elif 51 <= stat\_value < 70:

fill = "orange"

elif 70 <= stat\_value < 90:

fill = "yellow"

else:

fill = "green"

# Draw a bar of the appropriate length

draw\_box.create\_rectangle(start\_x1,

start\_y1,

# Scale the bar so that it has the correct length relative

# to the value of the stat

scale\_stat(stat\_value),

start\_y2,

fill = fill)

# Increment the value of y1 and y2 so that the next bar is shifted downwards

# and has the appropriate "thickness"

start\_y1 = start\_y2 + bar\_step

start\_y2 = start\_y1 + bar\_height

# The window!

window = tk.Tk()

window.title("Pokedex")

window.geometry("1200x700")

window.configure(background = "crimson")

# A list of all the Pokemon types.

type\_list = ['grass', 'fire', 'water', 'bug', 'normal', 'dark', 'poison',

'electric', 'ground', 'ice', 'fairy', 'steel', 'fighting',

'psychic', 'rock', 'ghost', 'dragon', 'flying']

# We have separate pictures for each Pokemon type, so we'll put them into a dictionary

# so that they won't get lost!

type\_dict = {i: ImageTk.PhotoImage(file = f"Type Pictures/{i.capitalize()}.png") for i in type\_list}

# # Note that many Pokemon have a "null" second typing. If you want to,

# you can include it in the dictionary and have the label be displayed as a empty value

# This is kind of okay, but you can see still see an empty label if you look closely.

# type\_dict['none'] = ''

# # No longer needed

# # A StringVar is a variable that can be linked to a widget. If assigned to textvariable,

# # whenever the text value of that widget changes, StringVar will change as well.

# selected = tk.StringVar()

# Configure 4 rows and 3 columns.

window.rowconfigure(0, weight = 0, pad = 20)

window.rowconfigure([i for i in range(1,4)], weight = 1)

window.columnconfigure([i for i in range(3)], minsize = 50, weight = 1)

### Name Frame

name\_frame = tk.Frame(window, relief = tk.RAISED, borderwidth = 4, bg = "blue1")

name\_label = tk.Label(name\_frame,

height = 2,

text = "Pokemon Name Here",

fg = "yellow",

bg = "blue1",

font = ("Futura", 16, "bold"))

# One of the few cases where you can mix pack() with grid().

name\_label.pack()

name\_frame.grid(row = 0, column = 0, padx = 10, sticky = "ew")

### Picture Frame

picture\_frame = tk.Frame(window,

relief = tk.SUNKEN,

borderwidth = 2,

height = 400,

width = 400,

bg = "white")

picture\_frame.rowconfigure(0, weight = 1)

picture\_frame.columnconfigure(0, weight = 1)

pic\_label = tk.Label(picture\_frame, bg = "white")

pic\_label.grid(row = 0, column = 0, sticky = "nsew")

picture\_frame.grid(row = 1, column = 0, rowspan = 2, sticky = "nsew", padx = 10)

picture\_frame.grid\_propagate(False)

### Type Frame

type\_frame = tk.Frame(window, relief = tk.RAISED, borderwidth = 2)

type1label = tk.Label(type\_frame, text = "Type 1 Here", font = ("Futura", 16))

type1label.grid(row = 0, column = 0)

type2label = tk.Label(type\_frame, text = "Type 2 Here", font = ("Futura", 16))

type2label.grid(row = 0, column = 1)

type\_frame.grid(row = 3, column = 0)

### Info Frame

info\_frame = tk.Frame(window, relief = tk.SUNKEN, borderwidth = 4)

# Calling rowconfigure and columnfigure within the info\_frame

# causes the contents of the frame (label) to be centered within it.

info\_frame.rowconfigure([0,1,2], weight = 2, minsize = 200)

# info\_frame.rowconfigure([1,2], weight = 1, minsize = 80)

info\_frame.columnconfigure([0,1], weight = 1, minsize = 300)

## Various Information

# The Pokedex entry is long, so we'll let it take up two columns

# Use wraplength to tell the app to wrap text in this label.

pokedex\_entry = tk.Label(info\_frame, text = "Pokedex Entry", font = ("Futura", 16),

wraplength = 700, anchor = "w", bg = "black", fg = "light blue",

justify=tk.LEFT)

pokedex\_entry.grid(row = 0, column = 0, columnspan = 2, sticky = "nsew")

height\_entry = tk.Label(info\_frame, text = "Height", font = ("Futura", 16), borderwidth = 2,

relief = tk.RIDGE, bg = "white")

height\_entry.grid(row = 1, column = 0, sticky = "nsew")

weight\_entry = tk.Label(info\_frame, text = "Weight", font = ("Futura", 16), borderwidth = 2,

relief = tk.RIDGE, bg = "white")

weight\_entry.grid(row = 1, column = 1, sticky = "nsew")

catch\_entry = tk.Label(info\_frame, text = "Catch Rate", font = ("Futura", 16), borderwidth = 2,

relief = tk.RIDGE, bg = "white")

catch\_entry.grid(row = 2, column = 1, sticky = "nsew")

# Create a special frame for the stats

stat\_frame = tk.Frame(info\_frame, borderwidth = 2, relief = tk.RIDGE, bg = "white")

stat\_frame.rowconfigure([i for i in range(6)], weight = 1)

stat\_frame.columnconfigure(1, weight = 1)

stat\_block = ["HP:", "ATK:", "DEF:", "SPA:", "SPD:", "SPE:"]

draw\_box = tk.Canvas(stat\_frame, height = 100, width = 275, bg = "white",

borderwidth = 0, highlightthickness = 0)

draw\_box.grid(row = 0, rowspan = 6, column = 2, sticky = "nsew")

# These values were selected by trial and error - see select\_pokemon()

start\_x1 = 0

start\_y1 = 7

start\_x2 = 250

start\_y2 = 27

bar\_height = 20 # height of each stat bar

bar\_step = 13 # distance between bars

for row, stat in enumerate(stat\_block):

text\_box = tk.Label(stat\_frame,

text = stat,

font = ("Futura", 12),

anchor = "w",

bg = "white",

justify = tk.LEFT,

borderwidth = 0)

text\_box.grid(row = row, column = 0, sticky = "nsw")

draw\_box.create\_rectangle(start\_x1, start\_y1, start\_x2, start\_y2, fill = "green")

start\_y1 = start\_y2 + bar\_step

start\_y2 = start\_y1 + bar\_height

# draw\_box.pack(side = tk.LEFT)

stat\_frame.grid(row = 2, column = 0, sticky = "nsew")

info\_frame.grid(row = 1, rowspan = 3, column = 1, columnspan = 2, sticky = "nsew")

# Grid propagate tells the app not to resize the frame - doesn't always work!

info\_frame.grid\_propagate(False)

### Search Frame

search\_frame = tk.Frame(window, relief = tk.RAISED, borderwidth = 3, bg = "light blue")

# Calling rowconfigure and columnfigure within the info\_frame

# causes the contents of the frame (label) to be centered within it.

search\_frame.columnconfigure([0,1,2,3,4], weight = 1)

search\_frame.rowconfigure(0, weight = 1, minsize = 55)

# Turn the buttons into actual buttons

left\_button = tk.Button(search\_frame, text = "Previous", font = ("Futura", 16),

command = lambda: select\_pokemon(move = "backward"))

left\_button.grid(row = 0, column = 0, sticky = "ew")

submit\_button = tk.Button(search\_frame, text = "Search!", font = ("Futura", 14),

command = select\_pokemon)

submit\_button.grid(row = 0, column = 3)

right\_button = tk.Button(search\_frame, text = " Next ", font = ("Futura", 16),

command = lambda: select\_pokemon(move = "forward"))

right\_button.grid(row = 0, column = 4, sticky = "ew")

def search\_pokemon():

"""

This function runs with every keystroke made in the search box

It will filter the list of Pokemon such that all remaining options match the string

in the search box

"""

current\_text = search\_box.get()

if current\_text == "" or current\_text in pokedex.name.tolist():

search\_box.config(values = pokedex.name.tolist())

else:

values = []

for name in pokedex.name.tolist():

if current\_text.lower() in name.lower():

values.append(name)

search\_box.config(values = values)

# Adjust to make a combobox

# Needs a values argument that requires a list of Pokemon

search\_box = ttk.Combobox(search\_frame,

values = pokedex.name.tolist(),

font = ("Futura", 16),

postcommand = search\_pokemon)

search\_box.grid(row = 0, column = 1, columnspan = 2, sticky = "nsew", pady = 5, padx = 10)

search\_frame.grid(row = 0, column = 1, columnspan = 2, sticky = "ew")

# Run the App

window.mainloop()