Presentazione progetto Covid - 19

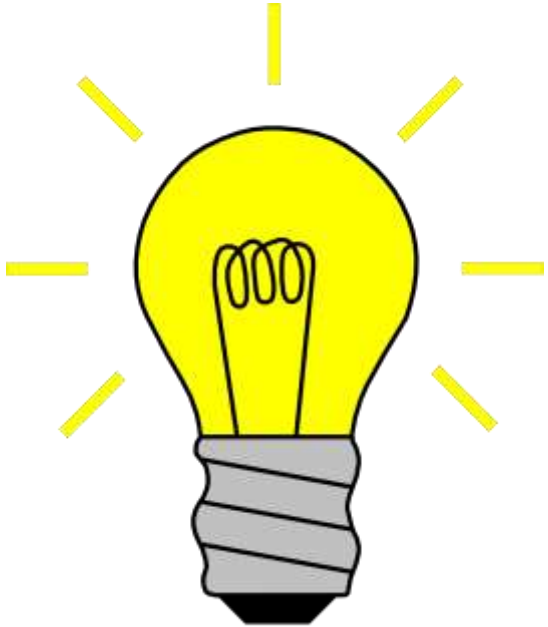
A cura di Massimo Pezzali

Problema

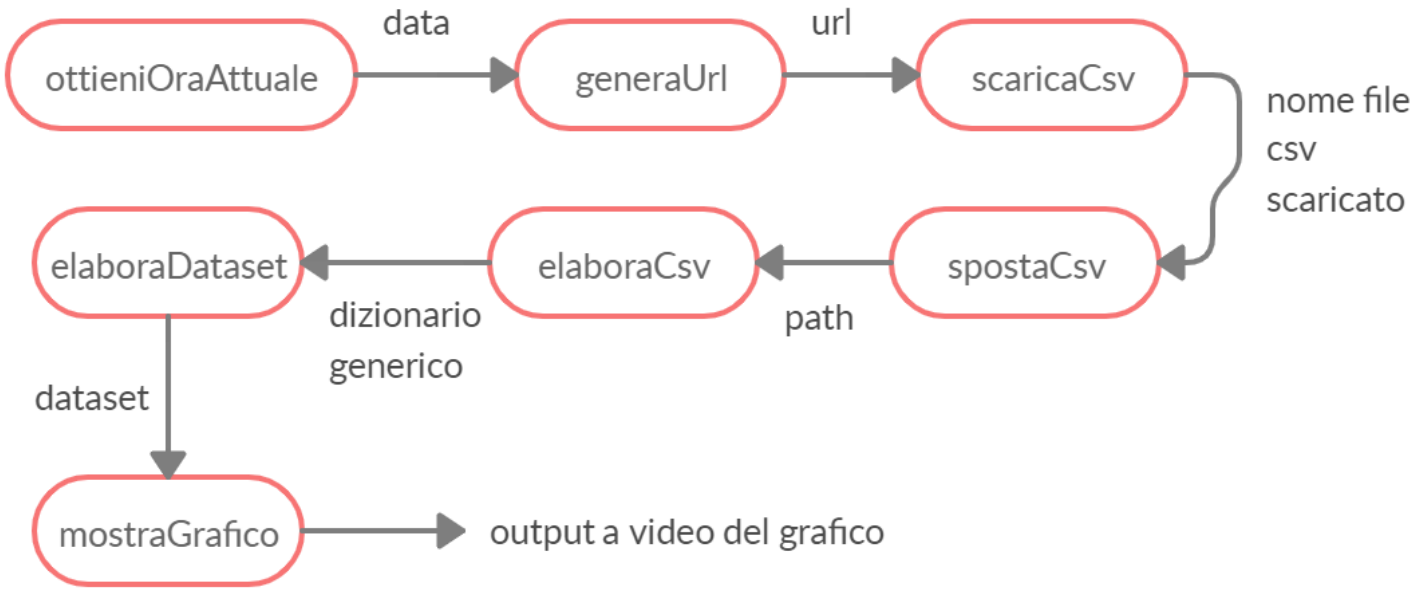
- Download file csv
- Organizzazione file csv nel file system
- Analisi e ottenimento dati dal file csv
- Creazione dataset
- Realizzazione grafico di trend

Soluzione

- Approccio OOP per la gui
- Struttura del main a funzioni
- Utilizzo di un client ntp



Struttura


ottieniOraAttuale
data
generaUrl
url
scaricaCsv
nome file csv scaricato
spostaCsv
path
elaboraCsv
dizionario generico
elaboraDataset
dataset
mostraGrafico
output a video del grafico

ottieniOraAttuale

```
def ottieniOraAttuale():
    """
    questa funzione richiede l'orario al server ntp italiano e ritorna i dati sotto forma di lista
    """
    try:
        clientNtp = ntplib.NTPClient()  # creazione istanza classe client ntp
        response = clientNtp.request(
            "it.pool.ntp.org")  # richiesta orario a server ntp. e ricevimento del timestamp in response
        ora_data_formattati = datetime.datetime.fromtimestamp(response.tx_time)

        # formattazione timestamp ottenuto dal server
        return ora_data_formattati.strftime('%Y-%m-%d-%H-%M-%S').split("-")
    except:
        errore = tk.Tk()
        errore.withdraw()
        msg.showerror(title="Errore", message="Nessuna connessione ad internet o server non disponibile")
        exit()
```

generaUrl

```
def generaUrl(data_ora):
    path = os.getcwd() + "\\res\\"

    if len(os.listdir(path)) == 0:  # no csv scaricati fino ad ora
        print("Primo avvio")
        url = "https://raw.githubusercontent.com/pcm-dpc/COVID-19/master/dati-province/dpc-covid19-ita-province.csv"
        return url
    else:  # di sicuro c'è il file riassuntivo
        data_creazione_riassuntivo = datetime.datetime.fromtimestamp(
            os.path.getctime(path + "dpc-covid19-ita-province.csv")).strftime(
            '%Y-%m-%d-%H-%M-%S').split("-")  # per controllo data creazione

        if int(data_ora[3]) > 17:
            # se nuovo file disponibile (controllo se ora maggiore delle 17,il file è disponibile dalle 18 in poi)
            # e non già scaricato e file riassuntivo non è stato scaricato oggi oltre le 18 (non comprende dati nuovi di oggi)
            if not os.path.exists(path + "dpc-covid19-ita-province-2020" + data_ora[1] + data_ora[2] + ".csv"):
                if int(data_creazione_riassuntivo[2]) == int(data_ora[2]) and int(data_creazione_riassuntivo[3]) < 18 or int(data_creazione_riassuntivo[2]) != int(data_ora[2]):
                    # controllo che file riassuntivo non comprende anche i file del giorno in cui si sta usando il programma
                    print("Nuovo file disponibile, riassuntivo non comprende i dati di oggi")
                    url = "https://raw.githubusercontent.com/pcm-dpc/COVID-19/master/dati-province/dpc-covid19-ita-province-2020" + \
                          data_ora[1] + data_ora[2] + ".csv"
                    return url
                else:  # usa dati vecchi
                    print("Ultimo file scaricato, i dati sono aggiornati, li ripristino")
                    return False
            else:  # usa dati vecchi
                print("Nessun nuovo file rilasciato, ripristino i dati vecchi")
                return False
```

scaricaUrl

```
def scaricaCsv(url):
    payload_risposta = requests.get(url)

    nome_file = url[72:]

    with open(nome_file, 'a', newline='') as file:
        writer = csv.writer(file)
        for record in payload_risposta.iter_lines():
            print(record)
            writer.writerow(record.decode('utf-8').split(','))

    return nome_file
```

spostaCsv

```
def spostaCsv(nome_file):
    current_path = os.getcwd() + "\\" + nome_file
    dest_path = os.getcwd() + "\\res\\" + nome_file

    os.rename(current_path, dest_path)

    return dest_path
```

contaRigheCsv

```
def contaRigheCsv(path):
    file = open(path)
    reader = csv.reader(file)
    return len(list(reader))
```

elaboraCsv

```python
def elaboraCsv(path):  # output dizionario contenente i dati che si passeranno a maptoi

    # creazione dizionario con come chiave la data, mentre come valore il dizionario ci

    dizionario_date = {}
    dizionario_regioni = {}
    dizionario_province = {}

    with open(path) as csvfile:
        readCSV = csv.reader(csvfile, delimiter=',')

        for record in readCSV:
            data = record[0][
                   :10]

            if record[0][:10] not in dizionario_date:
                # print("Cambio di data!pulisco...")
                dizionario_regioni.clear()
                dizionario_province.clear()
            if record[3] not in dizionario_regioni:
                # print("cambio di regione!pulisco...")
                dizionario_province.clear()

            if record[9] != 'totale_casi':  # skip della primma riga
                dizionario_province[record[5]] = int(record[9])
                dizionario_regioni[record[3]] = copy.deepcopy(dizionario_province)
                dizionario_date[record[0][:10]] = copy.deepcopy(dizionario_regioni)
```

1

```python
# 2 elaborazioni differenti-->una per il primo avvio, l'altra per un file giornaliero

if contaRigheCsv(path) < 150:  # righe <150, sono dati ultimo csv da aggiungere agli esistenti
    print("Rilevato file giornaliero, attacco i suoi dati a quelli già esistenti...")
    dizionarioDateRipristinato = ripristinaDizionarioDati()
    dizionarioDateRipristinato[data] = dizionario_date[data]

    # salvo nuovo diz, compreso di dati aggiornati
    salvaDizionarioDati(dizionarioDateRipristinato)

    return dizionarioDateRipristinato  # -->dizionario che comprende tutti i dati fino ad ora elaborati


else:  # altrimenti è un file riassuntivo, prima esecuzione, nulla da aggiungere a file esistenti
    print(
        "Rilevato file riassuntivo,nulla da aggiungere(primo avvio)...")
    salvaDizionarioDati(dizionario_date)  # primo salvataggio

    return dizionario_date
```

2

salva/ripristinaDizionario

```
def ripristinaDizionarioDati():
    with open(os.getcwd() + "\\cache\\dizionario_date.txt", "rb") as file:
        dizionario_ripristinato = pickle.load(file)
    return dizionario_ripristinato
```

```
def salvaDizionarioDati(dizionario_dati):
    with open(os.getcwd() + "\\cache\\dizionario_date.txt", "wb") as file:
        pickle.dump(dizionario_dati, file)
```

elaboraDataset

```
def elaboraDataset(dizionario_dati, provincia_selezionata,
                   regione_selezionata):  # dati ottenuti dai bottoni cliccati in gui
    dataset = []
    temp_date = []
    temp_casi = []

    for chiave in dizionario_dati.keys():
        temp_date.append(chiave[5:])
        temp_casi.append(dizionario_dati[chiave][regione_selezionata][provincia_selezionata])

    dataset.append(temp_date)
    dataset.append(temp_casi)
    dataset.append(regione_selezionata)
    dataset.append(provincia_selezionata)

    return dataset
```

mostraGrafico

```
def mostraGrafico(dataset):  # ogni volta ricalcolo grafico
    plt.rcParams['toolbar'] = 'None'
    # struttura dataset-->index 0 x=date,index 1 y=contagi,index 3 regione selezionata,index 4 provincia selezionata
    date = dataset[0]
    contagi = dataset[1]

    plt.figure(num='Grafico', figsize=(13, 10))
    plt.tick_params(labelsize=12, pad=3)
    plt.xticks(rotation=90)

    plt.title('Contagi da covid in {}'.format(dataset[3]))
    plt.xlabel('Giorni')
    plt.ylabel('Casi')

    plt.plot(date, contagi)

    plt.show()
```

main

1

```
# main()

regione_scelta_gui = ""
provincia_scelta_gui = ""

url = generaUrl(ottieniOraAttuale())
if url != False:
    nome_file = scaricaCsv(url)
    path = spostaCsv(nome_file)
    print(path)
    dizionario = elaboraCsv(path)
else:
    print("Ripristino dati vecchi...")
    dizionario = ripristinaDizionarioDati()

root = tk.Tk()
mainWindow = ParentWindow(root)
mainWindow.creaScenario("regioni_cropped.png")
root.mainloop()
```

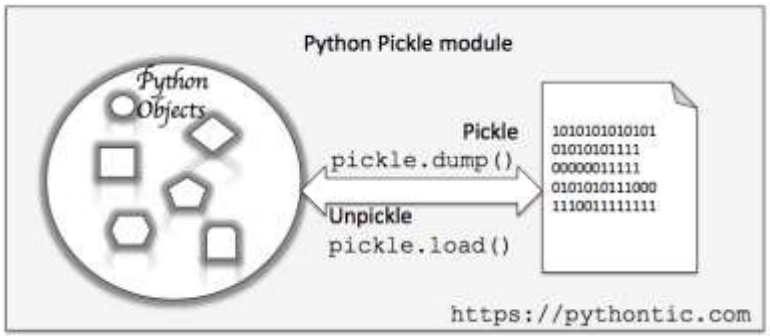
2

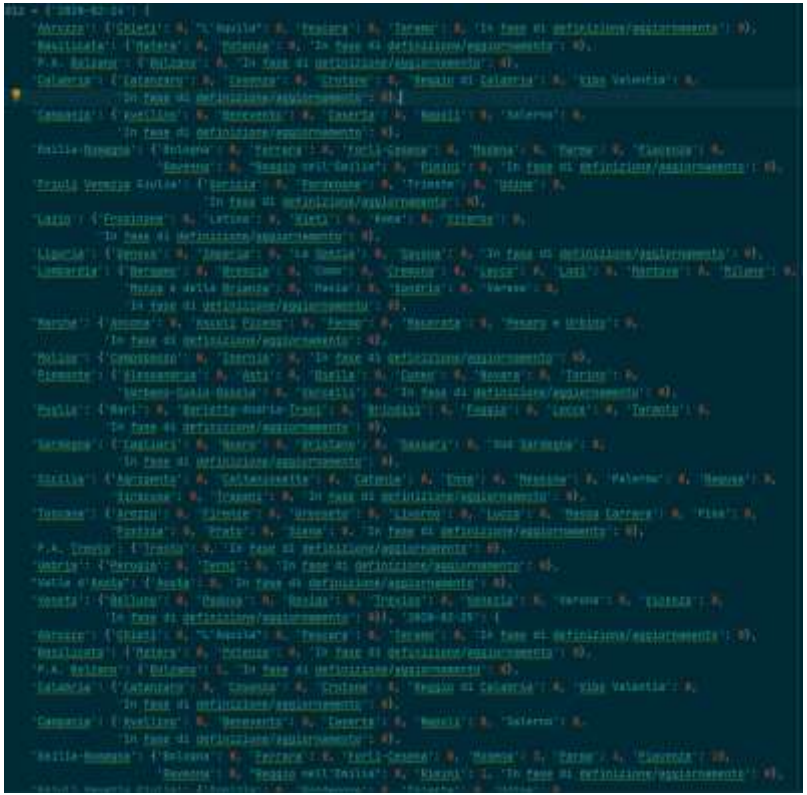
```
if regione_scelta_gui != "":  # se non è stata scelta alcune regione, non aprire sch
    child = tk.Tk()
    secondaryWindow = ChildWindow(child)
    secondaryWindow.tipo = "provincia"
    if regione_scelta_gui == "Valle d'Aosta":  # fix problema file non rinominabili
        secondaryWindow.creaScenario("Valle Aosta.png")
    else:
        if regione_scelta_gui == "Friuli":
            regione_scelta_gui = "Friuli Venezia Giulia"
        secondaryWindow.creaScenario(regione_scelta_gui + ".png")
    child.mainloop()

    if regione_scelta_gui == "Trentino":
        regione_scelta_gui = provincia_scelta_gui
        provincia_scelta_gui = regione_scelta_gui[5:]  # elimina P.A. dalla stringa

if provincia_scelta_gui != "":
    dataset = elaboraDataset(dizionario, provincia_scelta_gui, regione_scelta_gui)
    mostraGrafico(dataset)
```

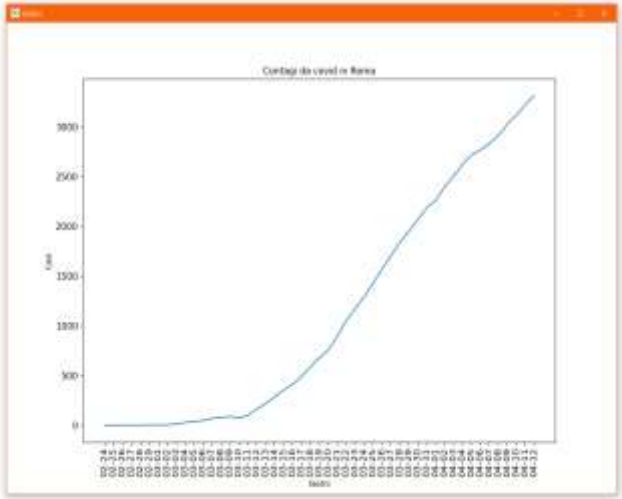
Sistema di caching





Librerie utilizzate

Aspetto della GUI



Classi della GUI

ParentWindow (superclasse)
path (delle immagini da usare) w (larghezza della finestra) h (altezza della finestra) tipo (della finestra) regioni (lista che contiene coordinate per disporre i bottoni delle regioni) canvas (che contiene lo scenario)
init (eseguito overriding dalla classe object) creaScenario caricaBottoni creaBottone salvaRegioneEsci (salva regione selezionata in main)

ChildWindow (sottoclasse)
Attributi sottoposti ad overriding: tipo (ora è a provincia) regioni (ora contiene le coordinate delle province, è un dizionario con a chiave le regioni) Attributi nuovi: regione (contiene la regione scelta dalla finestra delle regioni)
Dei metodi ereditati i seguenti sono stati sottoposti ad overriding: caricaBottoni (ora è adattato alla logica delle province) salvaregioneEsci (ora al posto di salvare in main regione, salva provincia)

ParentWindow

1

```
class ParentWindow(object):  # classe per la finestra delle regioni
    # attributi
    path = os.getcwd() + "\\assets\\"  # path directory script-->dinamica
    w = 0  # larghezza finestra
    h = 0  # altezza finestra
    tipo = "regione"  # per il print di aiuto all'utente
    regioni = ["Valle d'Aosta", 50, 100, "Piemonte", 57, 158, "Trentino", 220, 55, "Friuli", 300, 75, "Veneto",
               255, 125, "Lombardia", 150, 120, "Liguria", 110, 210, "Emilia-Romagna", 215, 185, "Toscana",
               215, 260, "Umbria", 275, 299, "Marche", 320, 265, "Lazio", 275, 350, "Abruzzo", 355, 335,
               "Molise", 380, 380, "Puglia", 435, 400, "Campania", 390, 425, "Basilicata", 460, 450,
               "Calabria", 490, 530, "Sicilia", 375, 650, "Sardegna", 185, 475]
    canvas = None

    # metodi
    def __init__(self, master):  # inizializzazione finestra
        print("Costruttore richiamato")
        self.master = master
        self.master.title("Analizzatore Covid-19")
        self.master.tk.call('wm', 'iconphoto', master._w, tk.PhotoImage(file=os.path.join(self.path + "favicon.png")))
        self.master.resizable(False, False)
        self.master.focus_set()
        self.master.bind("<Escape>", lambda e: (e.widget.destroy()))
        self.w = 585
        self.h = 734

        x = (master.winfo_screenwidth() / 2) - (self.w / 2)
        y = (master.winfo_screenheight() / 2) - (self.h / 2)

        master.geometry("%dx%d+%d+%d" % (self.w, self.h, x, y))

    def creaScenario(self, nomeImmagine):
        self.canvas = tk.Canvas(width=585, height=734, bg='white', relief="flat")
        self.canvas.pack(expand="yes", fill="both")
        bg = ImageTk.PhotoImage(file=self.path + nomeImmagine)
        self.canvas.create_image(self.w / 2, self.h / 2, image=bg, anchor="center")

        self.canvas.create_text(455, 100, fill="darkblue", font="Times 15 italic bold",
                                text="Seleziona una " + self.tipo)

        self.caricaBottoni(self.regioni)

        print("Scenario creato")

        self.canvas.mainloop()
```

2

```
def caricaBottoni(self, lista):
    for i in range(0, len(lista), 3):  # caricamento bottoni sulla scenario
        self.creaBottone(lista[i], lista[i + 1], lista[i + 2])

def creaBottone(self, regione, x, y):
    bottone = self.canvas.create_text(x, y, fill="black", font="Times 11", text=regione)
    self.canvas.tag_bind(bottone, '<Button-1>', lambda event, arg=regione: self.salvaRegioneEsci(event, arg))

def salvaRegioneEsci(self, event, regione):
    global regione_scelta_gui
    print(regione)
    regione_scelta_gui = regione
    self.canvas.delete("all")
    self.master.destroy()
```

```
class ChildWindow(ParentWindow):  # classe per la finestre delle province
    tipo = "provincia"
    regione = ""
    regioni = {'Abruzzo': {'Chieti': [350, 350], "L'Aquila": [235, 190], 'Pescara': [401, 325], 'Teramo': [250, 280]},
               'Basilicata': {'Matera': [370, 330], 'Potenza': [230, 350]},
               'Calabria': {'Catanzaro': [500, 370], 'Cosenza': [200, 300], 'Crotone': [360, 350],
                            'Reggio Calabria': [250, 500],
                            'ViboValentia': [200, 420]},
               'Campania': {'Avellino': [330, 340], 'Benevento': [280, 380], 'Caserta': [200, 280],
                            'Napoli': [230, 390],
                            'Salerno': [320, 370]},
               'Emilia-Romagna': {'Bologna': [325, 365], 'Ferrara': [365, 325], 'Forlì-Cesena': [390, 415],
                                  'Modena': [280, 380],
                                  'Parma': [225, 340], 'Piacenza': [175, 320], 'Ravenna': [400, 365],
                                  'Reggio nell'Emilia': [250, 355],
                                  'Rimini': [400, 380]},
               'Friuli Venezia Giulia': {'Gorizia': [375, 420], 'Pordenone': [210, 406], 'Trieste': [420, 400],
                                         'Udine': [500, 375]},
               'Lazio': {'Frosinone': [370, 410], 'Latina': [320, 440], 'Rieti': [300, 320], 'Roma': [200, 380],
                         'Viterbo': [220, 320]},
               'Liguria': {'Genova': [315, 320], 'Imperia': [200, 400], 'La Spezia': [400, 360], 'Savona': [250, 350]},
               'Lombardia': {'Bergamo': [270, 350], 'Brescia': [330, 340], 'Como': [210, 350], 'Cremona': [300, 425],
                             'Lecco': [240, 330], 'Lodi': [250, 420], 'Mantova': [370, 435], 'Milano': [210, 395],
                             'Monza e della Brianza': [235, 375], 'Pavia': [215, 435], 'Sondrio': [275, 300],
                             'Varese': [170, 330]},
               'Marche': {'Ancona': [335, 325], 'Ascoli Piceno': [370, 435], 'Parma': [380, 410],
                          'Macerata': [325, 390],
                          'Pesaro e Urbino': [250, 290]},
               'Molise': {'Campobasso': [350, 370], 'Isernia': [330, 380]},
               'Piemonte': {'Alessandria': [390, 450], 'Asti': [320, 430], 'Biella': [315, 320], 'Cuneo': [250, 475],
                            'Novara': [375, 330], 'Torino': [250, 390], 'Verbano-Cusio-Ossola': [350, 235],
                            'Vercelli': [330, 505]},
               'Puglia': {'Bari': [380, 360], 'Barletta-Andria-Trani': [230, 330], 'Brindisi': [370, 390],
                          'Foggia': [280, 500],
                          'Lecce': [400, 430], 'Taranto': [320, 400]},
               'Sardegna': {'Cagliari': [330, 500], 'Nuoro': [350, 325], 'Oristano': [250, 370], 'Sassari': [250, 270],
                            'Sud Sardegna': [200, 450]},
               'Sicilia': {'Agrigento': [280, 390], 'Caltanissetta': [320, 370], 'Catania': [380, 360],
                           'Enna': [340, 350],
                           'Messina': [420, 300], 'Palermo': [280, 330], 'Ragusa': [380, 435], 'Siracusa': [480, 420],
                           'Trapani': [190, 350]},
               'Toscana': {'Arezzo': [300, 350], 'Firenze': [330, 315], 'Grosseto': [320, 450], 'Livorno': [230, 390],
                           'Lucca': [230, 280], 'Massa Carrara': [280, 250], 'Pisa': [260, 335], 'Pistoia': [280, 280],
                           'Prato': [300, 300], 'Siena': [320, 370]},
               "Trentino": {'P.A. Bolzano': [380, 290], 'P.A. Trento': [250, 420]},
               'Umbria': {'Perugia': [260, 350], 'Terni': [315, 470]},
               "Valle d'Aosta": {'Aosta': [380, 365]},
```

1

ChildWindow

```
def caricaBottoni(self, lista):  # al posto di una lista si utiliz
    global regione_scelta_gui
    for provincia in lista[regione_scelta_gui].keys():
        coordinate = lista[regione_scelta_gui][provincia]
        self.creaBottone(provincia, coordinate[0], coordinate[1])

def salvaRegioneEsci(self, event, regione):
    global provincia_scelta_gui
    print(regione)
    provincia_scelta_gui = regione
    self.canvas.delete("all")
    self.master.destroy()
```

2