Report for Python course for natural scientists and engineers. OOP task.

Konstantin Ibadullaev, 63072

17 October 2022

Contents

1 Annotation

This report serves as a description and a manual for the example of OOP and its implementation. This app imitates an Email address book, where each entry for a contact person contains following fields:

1. First name
2. Surname
3. Birthday
4. E-mail

The project is modularized and consists of 3 parts: class Person with aforementioned properties, graphical user interface(GUI), and the main script which invokes GUI and the class Person. The main function of this program is to store entries in .txt and .bin format and upload it whenever it is needed. It is also possible to edit entries and overwrite uploaded information after editing.

2 Class Person

This section is devoted to description of the class Person. The class Person takes several variables: first name, surname, birthday and email. Corresponding methods exist for each of them, which allow to set and return the value of the specific attribute. The method *def__str__* returns the description of the class. The full code is listed bellow:

```
"""
Creates class Person with following properties:
first name, surname, birthday, email
"""


class Person:
    def __init__(self, fname, surname, birthday, email):
        if fname != None and surname != None and birthday!= None and email!= None:
            self.__fname = fname
            self.__surname = surname
            self.__birthday = birthday
            self.__email = email
        else:
            raise ValueError('Check input data!')


    @property
    def Fname(self):
        return self.__fname

    @Fname.setter
    def Fname(self, fname):
        if fname != None:
            self.__fname = fname

    @property
    def Surname(self):
        return self.__surname

    @Surname.setter
    def Surname(self, surname):
        if surname != None:
            self.__fname = surname


    @property
    def Birthday(self):
        return self.__birthday

    @Birthday.setter
    def Birthday(self, birthday):
        if birthday != None:
            self.__birthday = birthday


    @property
    def Email(self):
        return self.__email

    @Email.setter
    def Email(self, email):
        if email != None:
            self.__email = email
```

```
def __str__(self):
    #rep ={
    #    'First name' :   self.__fname,
    #    'Surname' : self.__surname,
    #    'Birthday': self.__birthday,
    #    'Email' : self.__email
    #    }



    #rep = 'Person Object with id' + str(id(self)) +' \n'
    rep = 'First name : %s ' % self.__fname  +' \n'
    rep += 'Surname : %s ' % self.__surname  +' \n'
    rep += 'Birthday : %s ' % self.__birthday  +'\n'
    rep += 'Email : %s ' % self.__email
    return rep
```

This class is invoked via GUI each time when a new contact is added to the book. Each instance is stored in a list which is considered as global variable. Such an approach allows to edit the list of stored instances by methods of the GUI.

3 GUI

The graphical user interface for this application consists of the main window, entry fields for the input and corresponding labels, several action buttons and a treeview widget to display the data in the table format. One can add a new contact by pressing the button Add after filling out the

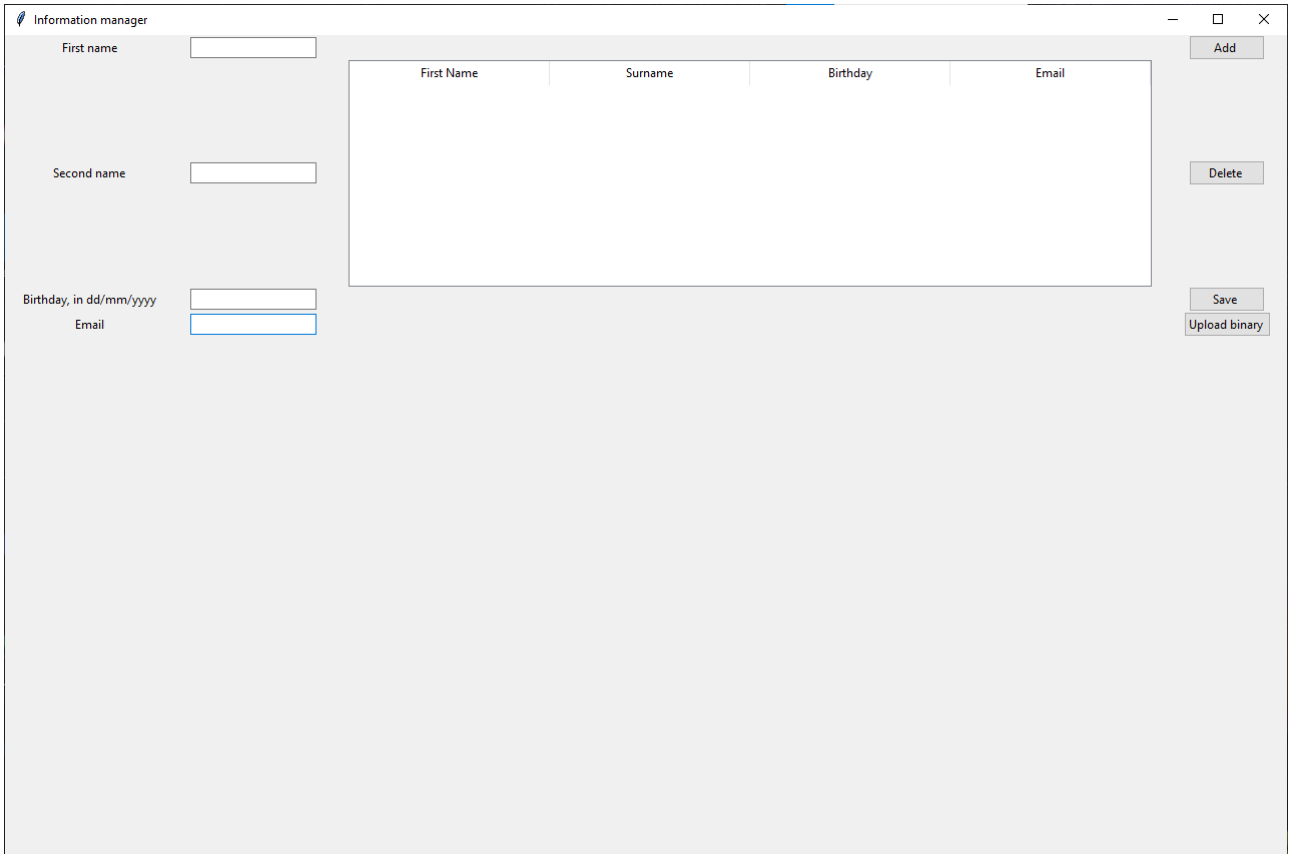

Figure 1: GUI

entry fields. For the demonstration purpose, 3 new contacts are added to the address book.

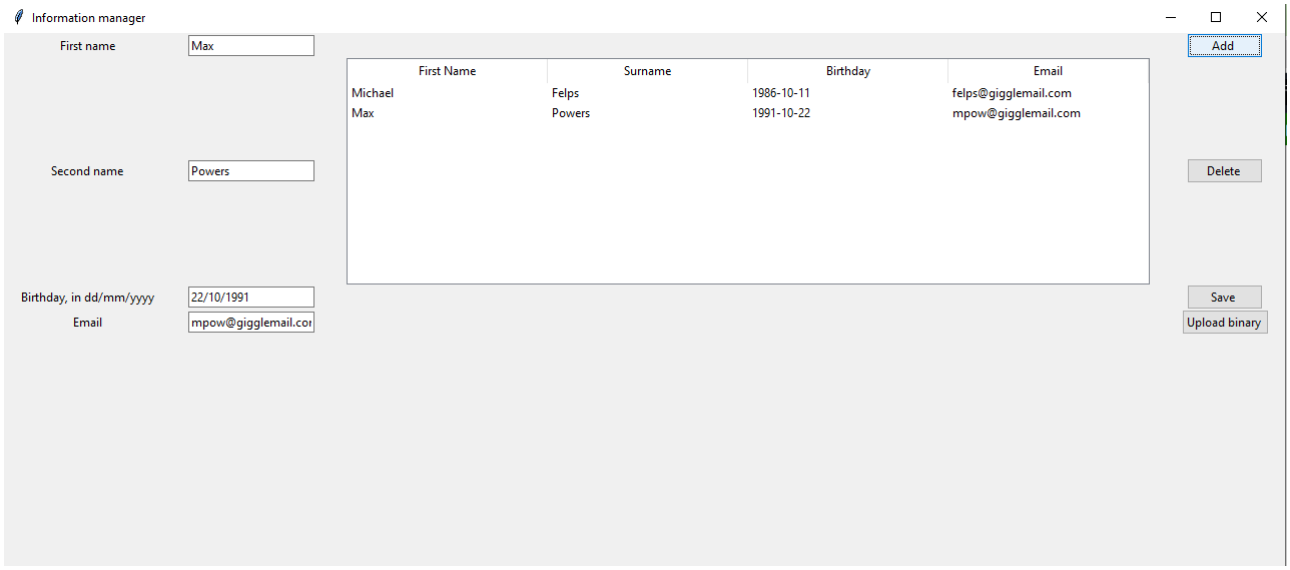

Figure 2: Add new contact.

By pressing the button Save one can store the entries in .txt and binary file.

It is also possible to delete particular contact by selecting the desired item and pressing the button Delete. For example, here one deletes 2 last contacts.

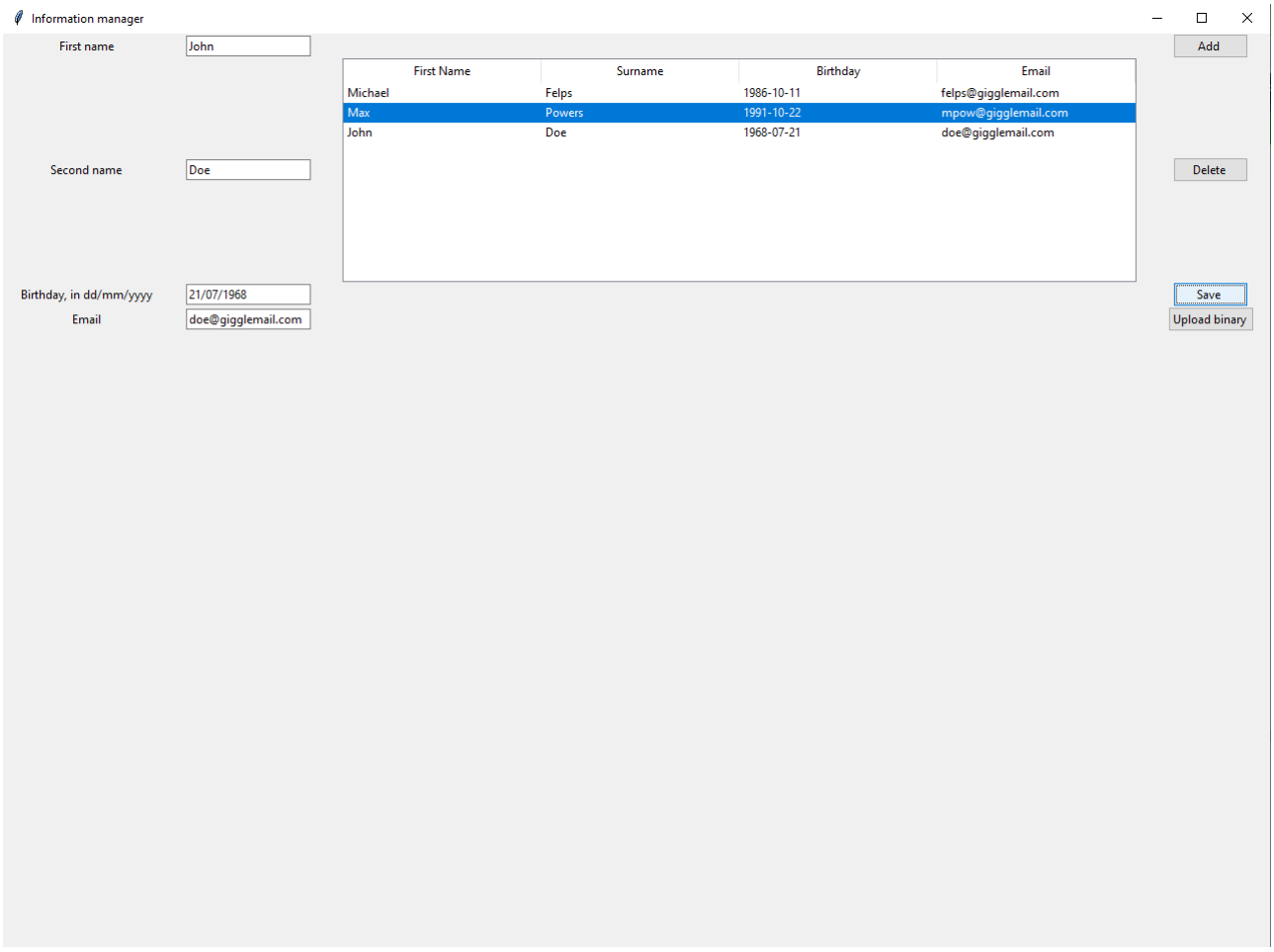

Figure 3: Save contacts.

The button Upload binary serves for uploading or restoring the contacts from the binary file. One can observe 3 stored contacts in the table.

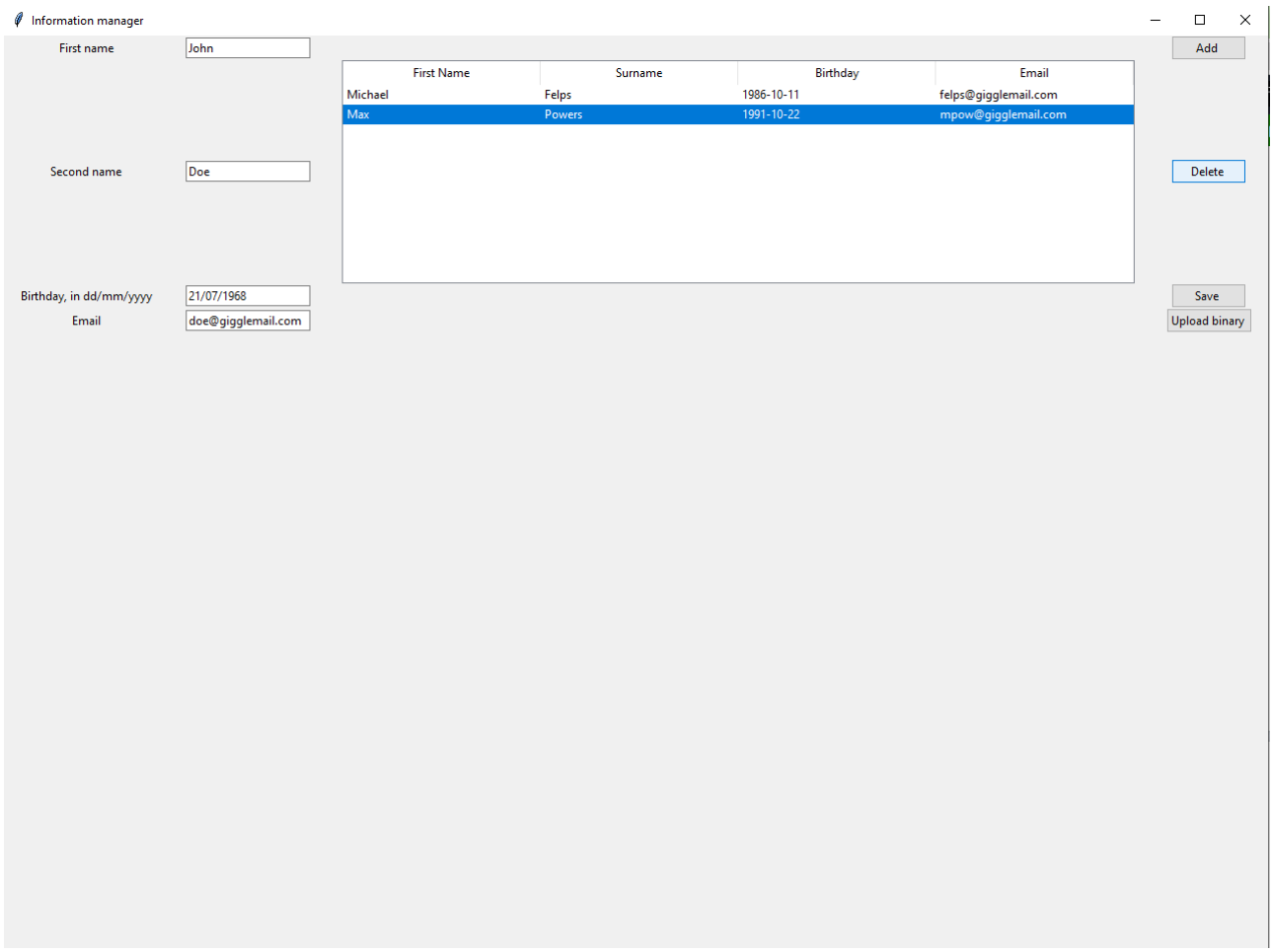

Figure 4: Delete last contact.

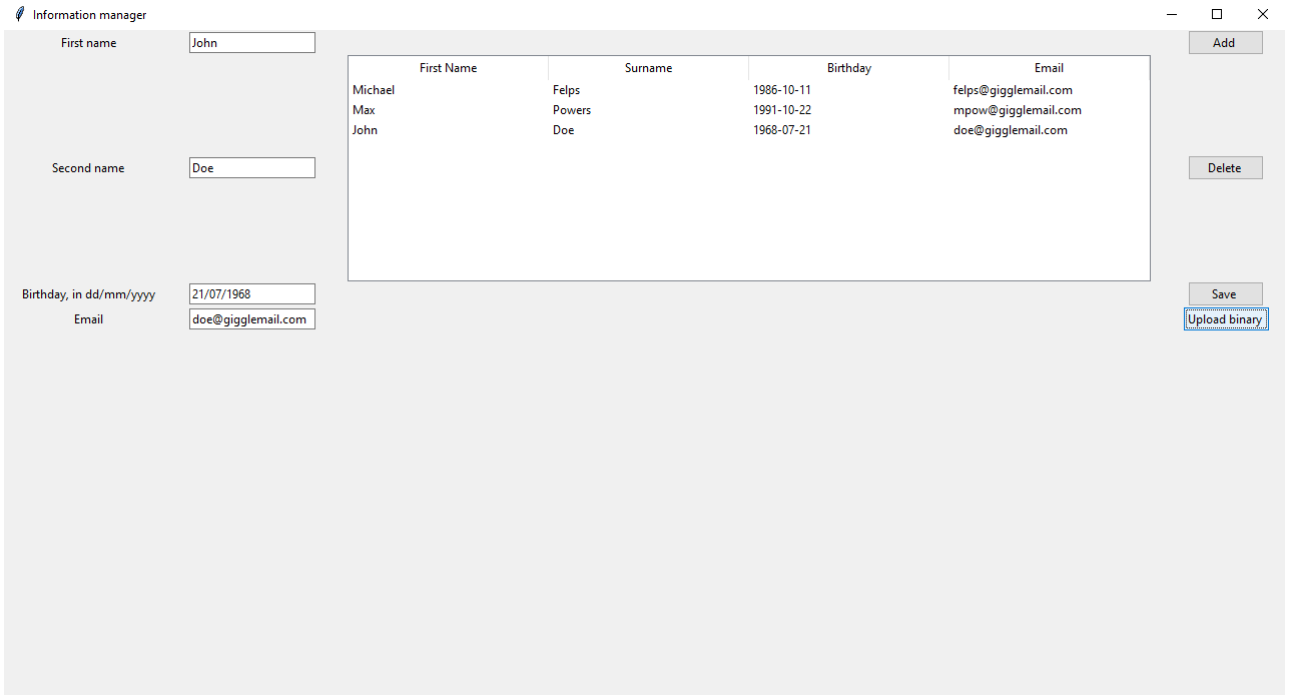

Figure 5: Upload contacts.

The tree view widget is used to display the contact information in the table format. The listing for the code is presented below.

```
import tkinter as tk
from tkinter import ttk
from tkinter.ttk import Label, Entry
from datetime import datetime
from tkinter import Tk, Text
from tkinter import *
from Person import Person
import pickle
import os

class App(tk.Tk):
    def __init__(self):
        super().__init__()

        # configure the root window
        self.title('Information manager')
        self.geometry('1280x1080')

        # configure the grid
        self.columnconfigure(0, weight=1)
        self.columnconfigure(1, weight=1)
        self.columnconfigure(2, weight=1)
        self.columnconfigure(3, weight=1)


        # label first name
        self.fname_label = ttk.Label(self, text='First name')
        self.fname_label.grid(column=0, row=0)

        # label second name
        self.sname_label = ttk.Label(self, text='Second name')
        self.sname_label.grid(column=0, row=1)

        # label  bday
        self.bday_label = ttk.Label(self, text='Birthday, in dd/mm/yyyy')
        self.bday_label.grid(column=0, row=2)

        # label email
        self.email_label = ttk.Label(self, text='Email')
        self.email_label.grid(column=0, row=3)


        # entry first name
        self.fn = tk.StringVar()
        self.fname_entry = ttk.Entry(self, textvariable=self.fn)
        self.fname_entry.grid(column=1, row=0,)

        # entry surname
        self.sn = tk.StringVar()
        self.sname_entry = ttk.Entry(self, textvariable=self.sn)
        self.sname_entry.grid(column=1, row=1)

        # entry  bd
        self.bd = tk.StringVar()
        self.bday_entry = ttk.Entry(self, textvariable=self.bd)
        self.bday_entry.grid(column=1, row=2)

        # label email
        self.em = tk.StringVar()
        self.email_entry = ttk.Entry(self, textvariable=self.em)
        self.email_entry.grid(column=1, row=3)
```

```
        # Add new person
        self.add_button = ttk.Button(self, text='Add ')
        self.add_button['command'] = self.Add_Dude
        self.add_button.grid(column=3, row=0)


        # Delete last person
        self.del_button = ttk.Button(self, text='Delete ')
        self.del_button['command'] = self.Del_Dude
        self.del_button.grid(column=3, row=1)

        # Save all
        self.save_button = ttk.Button(self, text='Save ')
        self.save_button['command'] =  self.Save_Dudes
        self.save_button.grid(column=3, row=2)

        # Upload binary
        self.upl_button = ttk.Button(self, text='Upload binary ')
        self.upl_button['command'] =  self.Upload_Dudes
        self.upl_button.grid(column=3, row=3 )

        # Create a treeview
        # define columns
        columns = ('First_name', 'Surname', 'Birthday','Email')

        self.tree = ttk.Treeview(self, columns=columns, show='headings')
        # define headings
        self.tree.heading('First_name', text='First Name')
        self.tree.heading('Surname', text='Surname')
        self.tree.heading('Birthday', text='Birthday')
        self.tree.heading('Email', text='Email')
        self.tree.grid(column=2, row=1, sticky=E)


    def Add_Dude(self):
        """Append a new entry"""
        global dudes
        # Create a new instance
        dude = Person(
            self.fname_entry.get(),
            self.sname_entry.get(),
            datetime.strptime(self.bday_entry.get(), '%d/%m/%Y').date(),
            self.email_entry.get()
            )
        # append to list
        dudes.append(dude)

        # Display the text
        self.tree.update()
        self.tree.insert('', tk.END, values=(dude.Fname, dude.Surname, dude.Birthday,dude.Email ))
        self.tree.update()



    def Del_Dude(self):
       """Delete the last entry"""
       global dudes
```

```
        selected_item = self.tree.selection()[0] ## get selected item

        dudes.pop(self.tree.index(selected_item))
        self.tree.delete(selected_item)
        self.tree.update()


    def Save_Dudes(self):
        """Save results"""
        global dudes

        # save txt
        with open('data.txt', 'w') as f:
                  for dude in dudes:
                        f.write(str(dude))
                        f.write('\n')
        # save binary
        with open('dudes.bin', 'wb') as fid:
           pickle.dump(dudes, fid)

    def Upload_Dudes(self):
        """ Upload the saved binary  """
        #check if binary exists
        if os.path.isfile('dudes.bin'):
            global dudes
            with open('dudes.bin', 'rb') as fid:
                    databin = pickle.load(fid)
            dudes = databin
            # Display the text
            self.tree.delete(*self.tree.get_children())
            self.tree.update()
            for i in range(len(databin)):
                self.tree.insert('', tk.END, values=(databin[i].Fname,
                databin[i].Surname, databin[i].Birthday,databin[i].Email ))
            self.tree.update()


#  Create an empty list to store the data(global)
dudes = []


# Uncomment lines below to call App within this script.

#if __name__ == "__main__":
  #Create an empty list
 # dudes = []
 # app = App()
  #app.mainloop()
```

4 Main script

This script serves only to initialize the program. It is common and considered as good practice to separate project in several parts depending on the functionality. Here, for example, GUI and class Person are modularized. Such an approach allows to reuse code in future projects and identify errors much faster than if it was a single file.

```
# main script
from Person import Person
from PGUI import App


if __name__ == "__main__":
    # call app
     app = App()
     app.mainloop()
```