Python for Beginners

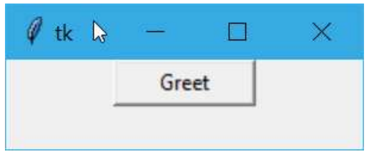
GUI Programming with TKinter

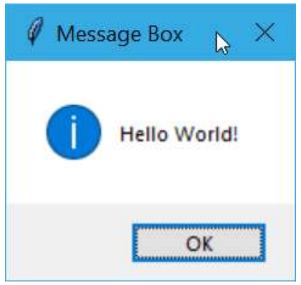
Programming GUI with Python

- Tkinter (tcl Tk interface)
  - Easy to learn
  - Simple and fast
  - Built in python (standard library)
  - Free to use (even commercial)
  - https://docs.python.org/3/library/tk.html
- PyQt5 (Python Qt Version 5)
  - Best choice for more sophisticated GUIs
  - Better event handling (signals, slots)
  - WYSIWYG interface builder available
  - Free for non commercial use
  - https://wiki.python.org/moin/PyQt

A simple Hello World

```
from tkinter import Tk, Button, messagebox


class MainWindow(Tk):

    def __init__(self):
        super().__init__()
        self.geometry("200x50")
        self.my_button = Button(self, text="Greet", command=self.greet)
        self.my_button.pack()

    def greet(self):
        messagebox.showinfo("Message Box", "Hello World!")


MainWindow().mainloop()
```





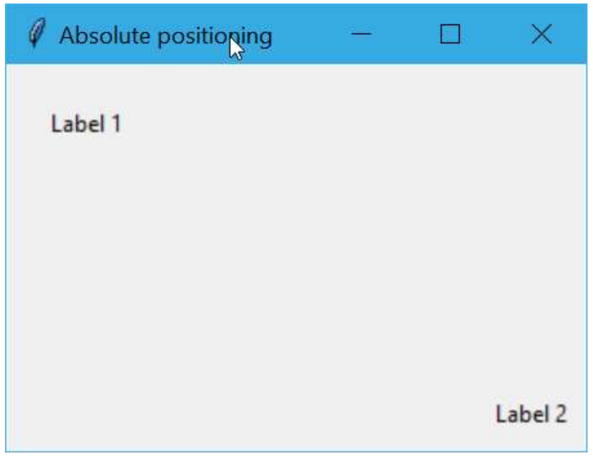
Hello World!

Layout Manager

- Three different managers:
  - Pack
    - Easy to use but restricted possibilities
    - Relative positioning in horizontal and vertical boxes
    - Best for simple layouts
  - Grid
    - two dimensional grid (rows, columns)
    - Automatic cell filling
    - Best for structured layouts
  - Place
    - Absolute positioning
    - Not flexible for different window sizes
    - For special layouts only
- Different managers must not be mixed in one window!

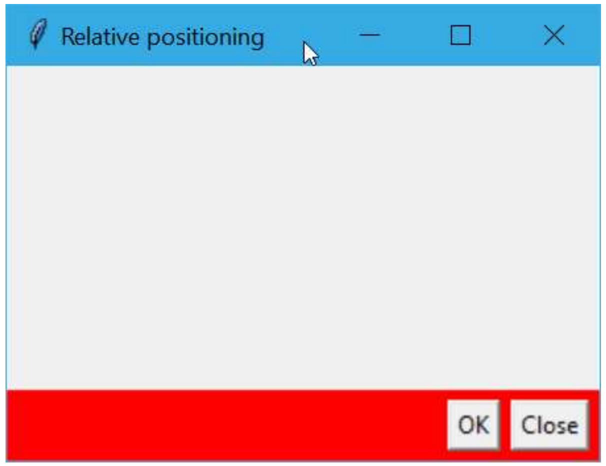
Place Manager

```
from tkinter import Tk, Label


class MainWindow(Tk):
    def __init__(self):
        super().__init__()

        self.title("Absolute positioning")
        self.geometry("300x200")

        label1 = Label(self, text="Label 1")
        label1.place(x=20, y=20)

        label2 = Label(self, text="Label 2")
        label2.place(x=250, y=170)

MainWindow().mainloop()
```



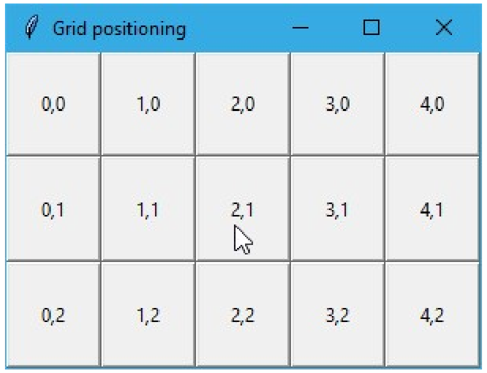
Pack Manager

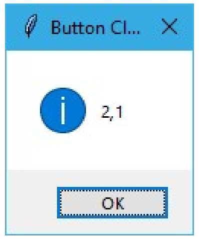
```
from tkinter import *


class MainWindow(Tk):
    def __init__(self):
        super().__init__()

        self.title("Relative positioning")
        self.geometry("300x200")

        frame_bottom = Frame(self, background="red")
        frame_bottom.pack(side=BOTTOM, fill=X)

        close_button = Button(frame_bottom, text="Close")
        close_button.pack(side=RIGHT, padx=5, pady=5)
        ok_button = Button(frame_bottom, text="OK")
        ok_button.pack(side=RIGHT)


MainWindow().mainloop()
```



Grid Manager

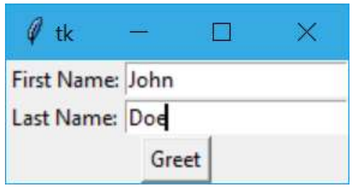
```
from tkinter import Tk, Button, messagebox, N, E, S, W

class MainWindow(Tk):
    def __init__(self):
        super().__init__()
        self.title(„Grid positioning")
        self.geometry("300x200")

        for column in range(0, 5, 1):
            self.columnconfigure(column, weight=1)
            for row in range(0, 3, 1):
                self.rowconfigure(row, weight=1)
                self.button = Button(self, text=f"{column},{row}", padx=5, pady=5)
                self.button.bind("<ButtonRelease>", self.button_click)
                self.button.bind("<KeyRelease-space>", self.button_click)
                self.button.grid(column=column, row=row, sticky=N+S+E+W)

    def button_click(self, event):
        messagebox.showinfo("Button Clicked", f'{event.widget["text"]}')

MainWindow().mainloop()
```

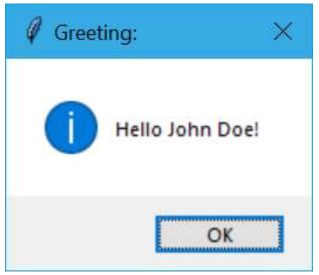




Entry

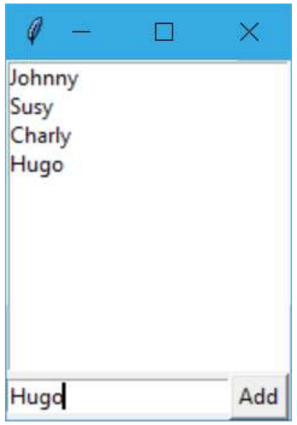
```
class MainWindow(Tk):

    def __init__(self):
        super().__init__()
        frame = Frame(self)
        Label(frame, text="First Name:").grid(row=0)
        Label(frame, text="Last Name:").grid(row=1)

        self.entry_firstname = Entry(frame)
        self.entry_firstname.grid(row=0, column=1)
        self.entry_lastname = Entry(frame)
        self.entry_lastname.grid(row=1, column=1)
        frame.pack(side=TOP)

        self.button_greet = Button(self, text="Greet", command=self.greet)
        self.button_greet.pack(side=BOTTOM)

    def greet(self):
        firstname = self.entry_firstname.get()
        lastname = self.entry_lastname.get()
        messagebox.showinfo("Greeting:", f"Hello {firstname} {lastname}!")
```





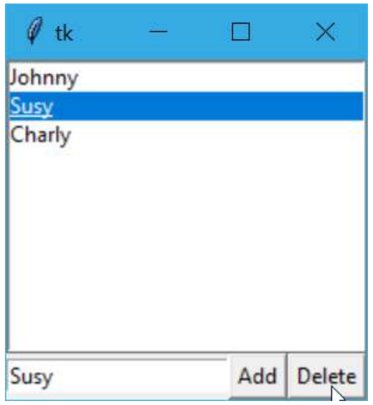
Listbox: set item source and add items

```
class MainWindow(Tk):

    def __init__(self):
        super().__init__()
        self.names = ["Johnny", "Susy", "Charly"]
        self.item_source = Variable(self,self.names)
        self.listbox_names = Listbox(self, listvariable=self.item_source)
        self.listbox_names.pack(fill=BOTH, expand=True)

        self.entry_name = Entry(self)
        self.entry_name.pack(side=LEFT)
        self.button_add = Button(self, text="Add", command=self.button_add_click)
        self.button_add.pack()

    def button_add_click(self):
        name = self.entry_name.get()
        self.names.append(name)
        self.item_source.set(self.names)
```



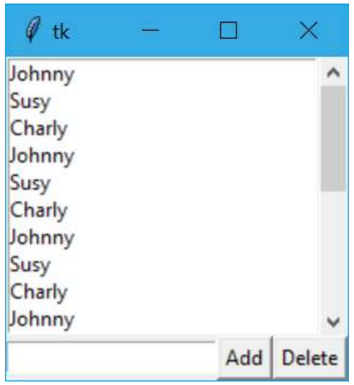
Listbox: get selection and remove items

```
# remember the selected name in a tk variable
# and start with None
self.selected_name = Variable(self,None)

# bind the select event to a method
self.listbox_names.bind("<<ListboxSelect>>", self.listbox_names_selection_changed)


def listbox_names_selection_changed(self, event):
    # get the selected index
    index = self.listbox_names.curselection()[0]
    # and remember the selected name
    self.selected_name.set(self.names[index])

def button_delete_click(self):
    # remove the selected name
    self.names.remove(self.selected_name.get())
    # and refresh the listbox
    self.item_source.set(self.names)
```



Listbox: add a Scrollbar

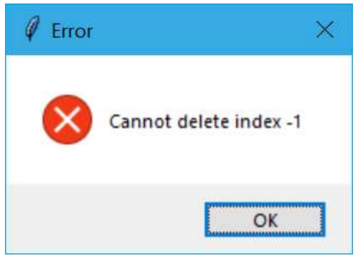
```
# create and pack a Frame to hold listbox and scrollbar together
self.main_frame = Frame(self)
self.main_frame.pack(fill=BOTH, expand=True)

# create and pack the scrollbar
self.scrollbar = Scrollbar(self.main_frame, orient=VERTICAL)
self.scrollbar.pack(side=RIGHT, fill=Y)

# connect the scrollbar with the listbox
self.listbox_names.config(yscrollcommand=self.scrollbar.set)
self.scrollbar.config(command=self.listbox_names.yview)
```



Dialogs: Messageboxes

- Info, Warning, Error

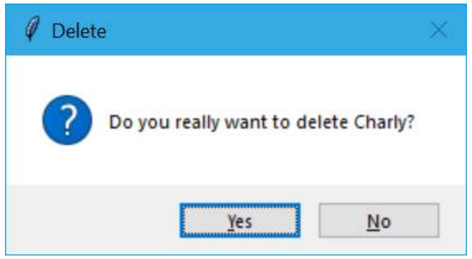
```
messagebox.showerror("Error", f"Cannot delete index {index}")
```


Error

Cannot delete index -1

- Questions

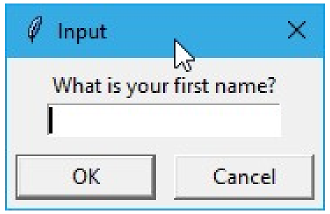
```
if messagebox.askyesno("Delete", f"Do you really want to delete {self.names[index]}?"):
    # remove the item at the index
    self.names.pop(index)
    # and refresh the listbox
    self.item_source.set(self.names)
```


Delete

Do you really want to delete Charly?

Dialogs: Simple Dialog

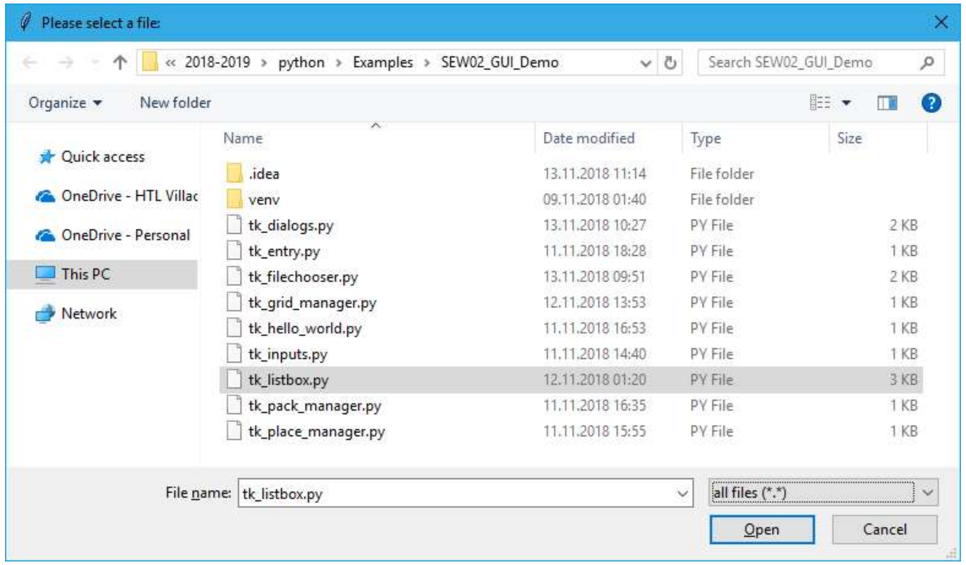
```
answer = simpledialog.askstring("Input", "What is your first name?", parent=self)
if answer is not None:
    messagebox.showinfo("Result", f"Your first name is '{answer}'")
```



You can use the methods: askstring(), askinteger(), askfloat()

Dialogs: File Dialog

```
my_filetypes = [('all files', '.*'), ('text files', '.txt')]
selected_file = filedialog.askopenfile(parent=self,
                                      initialdir=os.getcwd(),
                                      title="Please select a file:",
                                      filetypes=my_filetypes, mode="r")
if selected_file is not None:
    messagebox.showinfo("Result", f"Your selected '{str(selected_file)}'")
    # read and print the file content
    print(selected_file.read())
```



Name	Date modified	Type	Size
.idea	13.11.2018 11:14	File folder	
venv	09.11.2018 01:40	File folder	
tk_dialogs.py	13.11.2018 10:27	PY File	2 KB
tk_entry.py	11.11.2018 18:28	PY File	1 KB
tk_filechooser.py	13.11.2018 09:51	PY File	2 KB
tk_grid_manager.py	12.11.2018 13:53	PY File	1 KB
tk_hello_world.py	11.11.2018 16:53	PY File	1 KB
tk_inputs.py	11.11.2018 14:40	PY File	1 KB
tk_listbox.py	12.11.2018 01:20	PY File	3 KB
tk_pack_manager.py	11.11.2018 16:35	PY File	1 KB
tk_place_manager.py	11.11.2018 15:55	PY File	1 KB

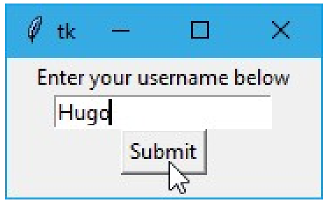
File name: tk_listbox.py

all files (*.*)

You can use the methods:
askopenfile(), askopenfiles(),
askfopenfilename(), askopenfilenames(),
asksaveasfile(), asksaveasfilename()

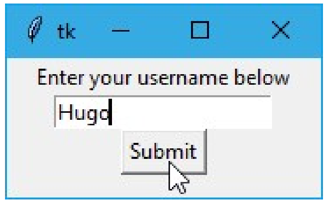
Dialogs: Custom Dialog (1)

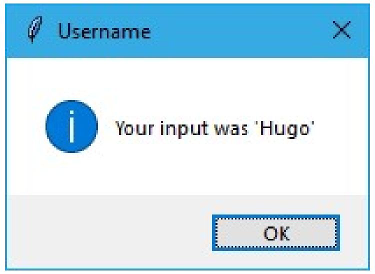
```python
# A Dialog Window should inherit from Toplevel and not from Tk
class MyDialogWindow(Toplevel):

    # pass the parent window to the dialog window
    def __init__(self, parent):
        # initialize the dialog window with the parent window
        super().__init__(parent)
        # create a very simple layout
        Label(self, text='Enter your username below').pack()
        self.myEntry = Entry(self)
        self.myEntry.pack()
        self.mySubmitButton = Button(self, text='Submit', command=self.on_save_click)
        self.mySubmitButton.pack()
        # this variable holds the entered username value
        self.username = ""

    def on_save_click(self):
        # read the entered username and store it
        self.username = self.myEntry.get()
        # destroy the dialog window
        self.destroy()
```



Dialogs: Custom Dialog (2)

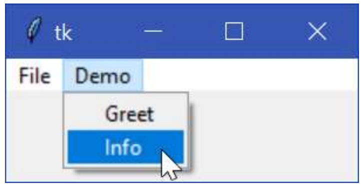
```
#create the dialog window with self as parent
input_dialog = MyDialogWindow(self)
# show the dialog window (wait_window = disable the parent window)
self.wait_window(input_dialog)
# after closing the dialog window go on in the parent
# and read the entered username from its public field
messagebox.showinfo("Username", f"Your input was '{input_dialog.username}'")
```





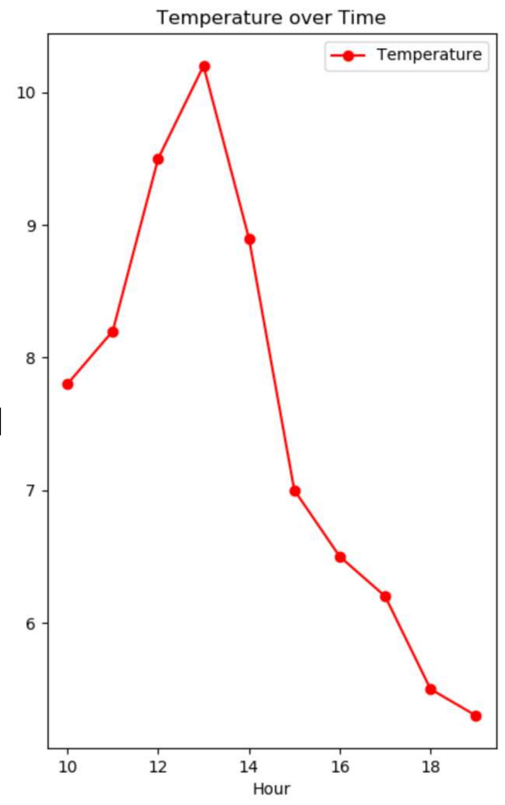
Menus

```
menu = Menu(self)
self.config(menu=menu)

fileMenu = Menu(menu, tearoff=False)
fileMenu.add_command(label="Exit", command=self.exit)
demoMenu = Menu(menu, tearoff=False)
demoMenu.add_command(label="Greet", command=self.greet)
demoMenu.add_command(label="Info", command=self.show_info)

menu.add_cascade(label="File", menu=fileMenu)
menu.add_cascade(label="Demo", menu=demoMenu)
```



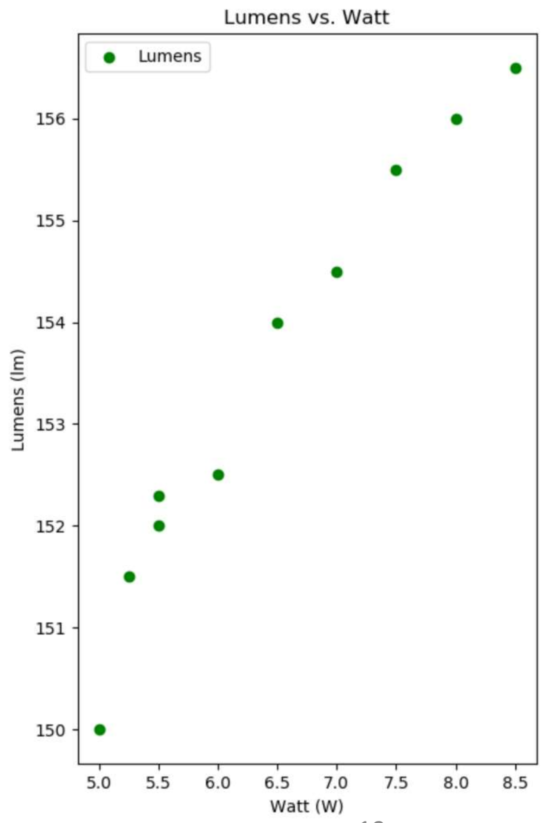
Charts: line chart

```
temperature_data = {
    'Hour': [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
    'Temperature': [7.8, 8.2, 9.5, 10.2, 8.9, 7, 6.5, 6.2, 5.5, 5.3]
}


line_chart = self.get_line_chart(temperature_data)
line_chart.pack(side=LEFT, fill=BOTH)

def get_line_chart(self, data):
    dataframe = DataFrame(data, columns=['Hour', 'Temperature'])
    dataframe = dataframe[['Hour', 'Temperature']].groupby('Hour').sum()
    figure = Figure(figsize=(5, 4), dpi=100)
    axes = figure.add_subplot(111)
    dataframe.plot(kind='line', legend=True, ax=axes, color='r', marker='o', fontsize=10)
    axes.set_title('Temperature over Time')
    return FigureCanvasTkAgg(figure, self).get_tk_widget()
```



Charts: scatter chart

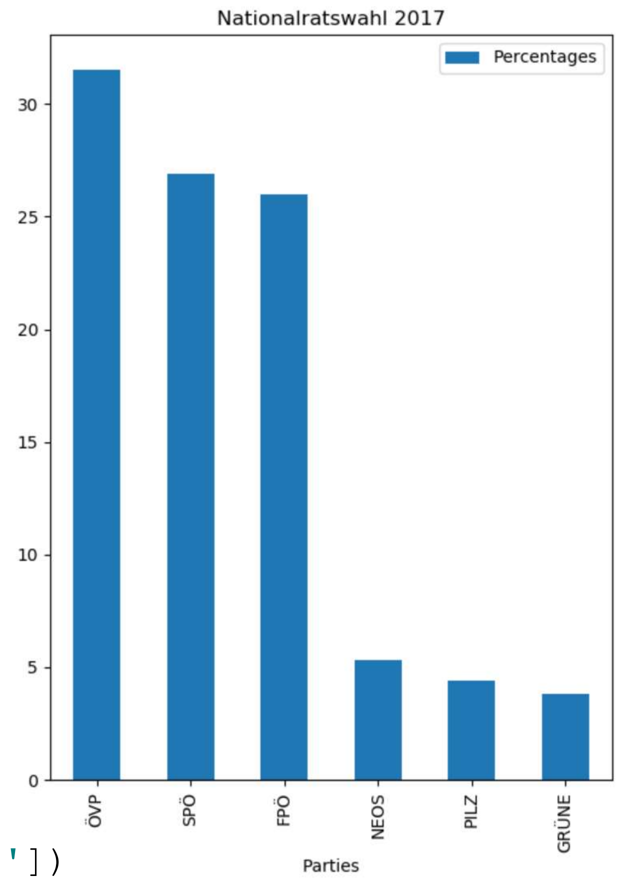
```
light_data = {
    'Watt': [5, 5.5, 6, 5.5, 5.25, 6.5, 7, 8, 7.5, 8.5],
    'Lumens': [150.0, 152.0, 152.5, 152.3, 151.5, 154.0, 154.5, 156.0, 155.5, 156.5]
}


scatter_chart = self.get_scatter_chart(light_data)
scatter_chart.pack(side=LEFT, fill=BOTH)


def get_scatter_chart(self, data):
    dataframe = DataFrame(data, columns=['Watt', 'Lumens'])
    figure = Figure(figsize=(5, 4), dpi=100)
    axes = figure.add_subplot(111)
    axes.scatter(dataframe['Watt'], dataframe['Lumens'], color='g')
    axes.legend()
    axes.set_xlabel('Watt (W)')
    axes.set_ylabel('Lumens (lm)')
    axes.set_title('Lumens vs. Watt')
    return FigureCanvasTkAgg(figure, self).get_tk_widget()
```



Charts: bar charts

```
election_data = {
    'Parties': ['SPÖ', 'ÖVP', 'PILZ', 'FPÖ', 'NEOS', 'GRÜNE'],
    'Percentages': [26.9, 31.5, 4.4, 26.0, 5.3, 3.8]
}
```

```
bar_chart = self.get_bar_chart(election_data)
bar_chart.pack(side=LEFT, fill=BOTH)
```



```
def get_bar_chart(self, data):
    dataframe = DataFrame(data, columns=['Parties', 'Percentages'])
    dataframe = dataframe[['Parties', 'Percentages']].groupby('Parties').sum()
    dataframe.sort_values('Percentages', inplace=True, ascending=False)
    figure = Figure(figsize=(6,8), dpi=100)
    axes = figure.add_subplot(111)
    dataframe.plot(kind='bar', legend=True, ax=axes)
    axes.set_title('Nationalratswahl 2017')
    return FigureCanvasTkAgg(figure, self).get_tk_widget()
```

Timer

- Delay the next command (e.g. method animate) after a specific time period (e.g. 500ms)
- The after-method must be called at a Tk object or widget

```
self.after(500, self.animate)
```

- Example: a blinking button

Timer (2): turn the blinking on or off

```
# create the button that should blink
self.my_button = Button(self, text="Start", command=self.switch_animation)

# remember the origin background color of the button
self.origin_color = self.my_button.cget("background")
# start without blinking
self.my_button_blinking = False

def switch_animation(self):
    # change the blinking status
    self.my_button_blinking = not self.my_button_blinking
    # change the text of the button
    self.my_button.configure(text="Stop")
    # and start the animation
    self.animate()
```

Timer (3): animate with a delayed callback

```
def animate(self):
    # if the button should blink
    if self.my_button_blinking:
        # switch the background color
        color = self.my_button.cget("background")
        if color != "red":
            self.my_button.configure(background="red")
        else:
            self.my_button.configure(background="green")
        # wait 500ms to call the animate method again
        self.after(500, self.animate)
    else:
        # stop blinking
        self.my_button.configure(background=self.origin_color)
        self.my_button.configure(text="Start")
```