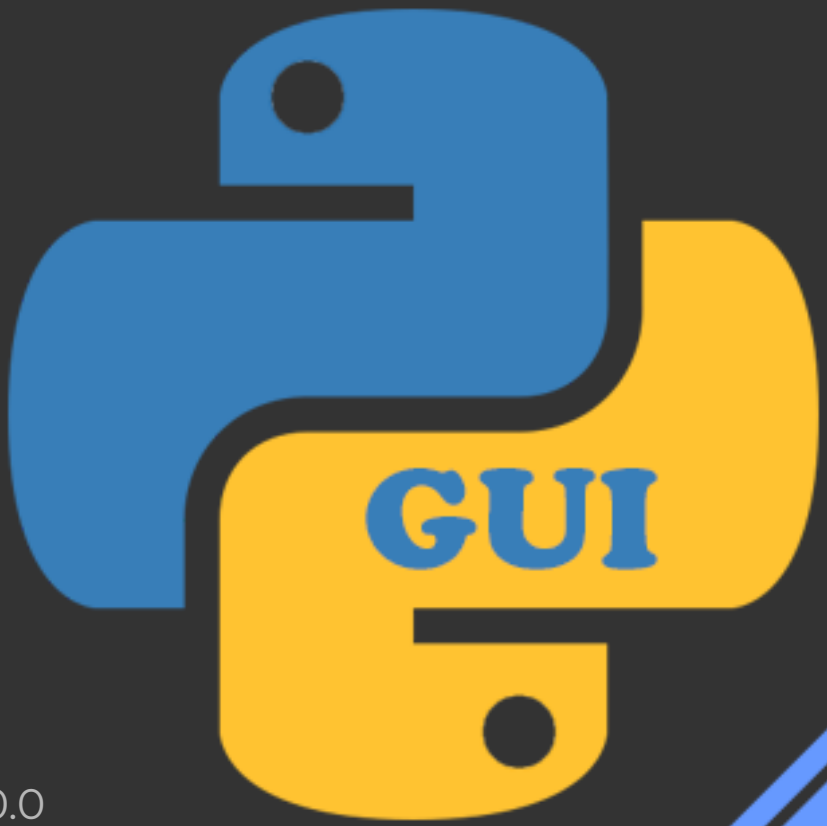
Mateusz Wiliński

Python GUI tkinter



--version: 1.0.0

1

Wbudowany interfejs graficzny tkinter w python

PIKADEMIA

Python GUI - tkinter

Programowanie GUI w Python (tkinter)

Ebook może być dowolnie rozpowszechniany
i używany w dowolnym celu.

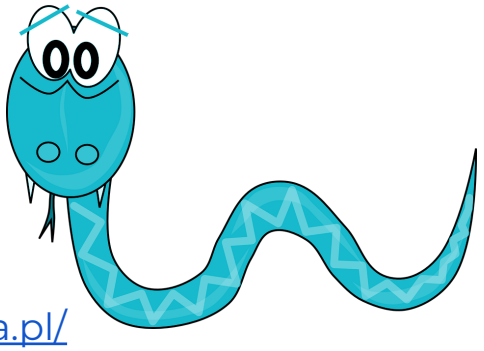
Zapraszamy do współpracy przy jego tworzeniu i ulepszaniu.

Kolorowanie składni kodu na podstawie VSC Light+

Link do aktualnej wersji:

Python GUI - tkinter

Autor: Mateusz Wiliński
Email: pikademia@gmail.com
Data ostatniej aktualizacji: 07.2022



Zapraszamy na stronę https://www.pikademia.pl/

PIKADEMIA

Spis zagadnień

Tkinter

Podstawy programu z tkinter

Tworzenie interfejsu graficznego z tkinter jest bardzo łatwe

1. Import modułu tkinter

```
from tkinter import *
```

2. Stworzenie okna tkinter

```
root = Tk()
```

3. Stworzenie loopa na koniec programu, który pozwoli utrzymać okno i korzystać z eventów

```
root.mainloop()
```

Okno główne programu - root

Okno główne posiada kilka interesujących metod i właściwości.

Tytuł programu

```
root.title("Pierwszy program w python GUI tkinter")
```

Wymiary okna i pozycja na ekranie

```
root.geometry('600x400+50+20')
```

Atrybuty, np. przezroczystość

```
root.attributes('-alpha',0.5)
```

Ikona programu

```
root.call('wm', 'iconphoto', root._w,
PhotoImage(file='icon.png'))
```

Wymiar ekranu

```
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
```

Konfiguracja okna / widgetu

```
root.config(background = "Light Blue")
```

Przykład umiejscowienia okna w środku ekranu

```
from tkinter import *
root = Tk()

window_width = 600
window_height = 400
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
center_x = int(screen_width/2 - window_width / 2)
center_y = int(screen_height/2 - window_height / 2)
root.geometry(f'{window_width}x{window_height}+{center_x}+{
center_y}')
root.mainloop()
```

Tworzenie i pozycjonowanie widgetów

Widgety tworzy się poprzez wywołanie konstruktora danej klasy.

Najczęściej przypisuje się taki obiekt do zmiennej,np:

```
my_label = Label(root, text="Hejka, to jest etykieta")

my_label.config(
    background = "#555",
    fg = "#ccc",
    font = ("Arial", 20),
    padx=20,
    pady=10
    )
```

Widgety tkinter - lista

button
canvas
checkbutton
combobox
entry
frame
label
labelframe
listbox
menu
menubutton
message
notebook
tk_optionMenu
panedwindow
progressbar
radiobutton
scale
scrollbar
separator
sizegrip
spinbox
text
treeview

Pozycjonowanie widgetów - 3 sposoby

Aby widget pojawił się wizualnie w programie, należy umieścić go wewnątrz głównego kontenera za pomocą jednego z 3 sposobów:

pack - układa widgety w kolejności od góry w dół

```
my_label.pack()
```

grid - układa widgety w matrycy wierszy i kolumn

```
my_label.grid(row=0, column=0)
```

place - pozycjonuje widgety w lokacji absolutnej

```
my_label.place(x=20,y=20,height=100,width=120)
```

Każdy z powyższych sposobów posiada szereg dostępnych właściwości, które można sprecyzować wewnątrz nawiasu okrągłego.

Pozycjonowanie: pack() - dostępne opcje

expand

Rozszerza widget na pełną dostępną przestrzeń rodzica

True

False

```
expand=True
```

fill

Określa w jaki sposób widget ma wypełnić dostępną przestrzeń.

NONE (default)

X (fill only horizontally)

Y (fill only vertically)

BOTH (fill both horizontally and vertically).

```
fill=BOTH
```

side

Wybór strony do której ma przylegać widget

TOP (default)

BOTTOM

LEFT

RIGHT.

```
side=LEFT
```

```
my_btn.pack(expand=True, fill=BOTH, side=LEFT)
```

Pozycjonowanie: grid() - dostępne opcje

row – Wiersz do którego zostanie dodany widget

column – Kolumna do której zostanie dodany widget, domyślnie 0

rowspan – Ile wierszy ma zajmować widget, domyślnie 1

columnspan – Ile kolumn ma zajmować widget, domyślnie 1

ipadx, ipady – Ile pixeli ma wynieść odsunięcie wewnątrz widgetu

padx, pady – Ile pixeli ma wynieść odsunięcie na zewnątrz widgetu

sticky – Zachowanie w przypadku, gdy komórka jest większa niż widget. Domyślnie sticky = '', widget jest wycentrowany. Można podać wartości by przysunąć widget do podanej krawędzi.
N, E, S, W, NE, NW, SE, and SW,

Pozycjonowanie: place() - dostępne opcje

anchor – The exact spot of widget other options refer to: may be N, E, S, W, NE, NW, SE, or SW, compass directions indicating the corners and sides of widget; default is NW (the upper left corner of widget)

bordermode – INSIDE (the default) to indicate that other options refer to the parent's inside (ignoring the parent's border); OUTSIDE otherwise.

height, width – Height and width in pixels.

relheight, relwidth – Height and width as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.

relx, rely – Horizontal and vertical offset as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.

x, y – Horizontal and vertical offset in pixels.

Eventy

Click event za pomocą command

W wielu widgetach, możemy użyć właściwości **command**, która pozwoli nam przypisać funkcję, która wykona się po kliknięciu na danym widgetcie.

Przykład programu wyświetlającego aktualną datę po kliknięciu na przycisk:

```
from tkinter import *
import datetime

root = Tk()
root.geometry('600x400')

def get_date():
    data = datetime.datetime.now()
    data_label = Label(root, text=data, fg="#ff0000")
    data_label.pack()

btn = Button(root,
    text="Get date",
    background="#666",
    fg="#aaa",
    padx = 20,
    pady = 10,
    command=get_date)

btn.pack()
root.mainloop()
```

Przykład programu, wyświetlającego okno z podaną wiadomością (messagebox)

```
from tkinter import *
from tkinter import messagebox
root = Tk()
root.geometry('600x300')

def display_prompt():
    messagebox.showinfo(title="Event", message="Działa")

my_btn= Button(
    root, text="Get info", padx=20, pady=10,
    command=display_prompt)

my_btn.pack()
root.mainloop()
```

Jeśli chcemy dodać argument, do wywołania funkcji, należy zrobić to z użyciem wyrażenia lambda, np:

```
def display_prompt(msg):
    messagebox.showinfo(title="Event", message=msg)

my_btn= Button(
    root, text="Get info", padx=20, pady=10,
    command=lambda:display_prompt("Działa"))
```

Eventy za pomocą metody bind()

Eventy to wbudowane zachowania użytkownika, które mogą być nasłuchiwane przez system, np: kliknięcie myszy, wciśnięcie klawisza, ruch kursora, itp.
Możemy wywołać odpowiednie funkcje w chwili, kiedy taki event będzie miał miejsce na określonym widgetcie.
Nasłuchiwanie eventów dodajemy za pomocą metody bind(), gdzie podajemy nazwę eventu oraz funkcji do wywołania.

```
root.bind('<event>',my_func)
```

Przykład wywołania funkcji my_funct po kliknięciu myszy w oknie programu(root):

```
from tkinter import *
from tkinter import messagebox
root = Tk()
root.geometry('600x300+50+50')

def my_func(event):
    messagebox.showinfo(title= "Event", message = "Działa")
root.bind('<Button-1>',my_func)
root.mainloop()
```

Jeśli chcemy dodać więcej funkcji do danego eventa, należy dodać 3 argument add='+'
Bez tego, kolejne eventy nadpiszą te poprzednie, np:

```
root.bind('<Return>',my_func2, add='+')
```

Przykładowe typy eventów

Event	Opis
<Button-1>, <ButtonPress-1>, <1>	Przyciski myszy (Lewy-1, Środkowy-2, Prawy - 3
<B1-Motion>	Ruch myszy z wciśniętym przyciskiem (1,2,3)
<ButtonRelease-1>	Zwolnienie przycisku myszy
<Double-Button-1>	Podwójne kliknięcie przycisku
<Enter>	Wskaźnik myszy wszedł na widget
<Leave>	Wskaźnik myszy upuścił widget
<FocusIn>	Kursor został aktywowany na widgetcie lub jego dzieciach
<FocusOut>	Kursor opuścił widget
<Return>	Klawisz Enter
<Key> (<a>, <b>, ...)	Dowolny klawisz
<Shift-Up>	Łączenie klawisza Up z klawiszami specjalnymi (Shift, Alt, Control)
<Configure>	Widget zmienił rozmiar
<Deactivate>	Widget został deaktywowany
<Destroy>	Widget został zniszczony
<Expose>	Aplikacja lub widget staje się widoczny, jeśli wcześniej był zasłonięty przez inne okno
<KeyRelease>	Zwolnienie przycisku
<Map>	Widget staje się widzialny w aplikacji
<Motion>	Ruch kursora wewnątrz widgeta
<MouseWheel>	Pokrętło myszy
<Visibility>	Część aplikacji staje się widoczna na ekranie

Inputfield - wprowadzanie treści

Wprowadzanie treści jest możliwe za pomocą widgetu Entry(), pobieranie treści następuje za pomocą metody get()

```
from tkinter import *
root = Tk()
root.geometry('600x300+50+50')

entry_w = Entry(root, width=50)
entry_w.pack()

def get_input(event):
    my_label = Label(root, text=entry_w.get())
    my_label.pack()

entry_w.bind('<Return>', get_input)

root.mainloop()
```

Grafiki z modułem pillow

Zarządzanie grafikami za pomocą modułu zewnętrznego pillow.

Na początku instalujemy moduł pillow w konsoli

```
pip install pillow
```

Importujemy moduł:

```
from PIL import ImageTk, Image
```

Zdjęcie powinno być załadowane, dodane do widgetu, a następnie wyświetlone jako widget, np:

```
img = ImageTk.PhotoImage(Image.open("1.jpg"))
my_label = Label(image=img)
my_label.pack()
```

Przykład prostej przeglądarki zdjęć.

```
from tkinter import *
from PIL import ImageTk, Image

root = Tk()
root.title("Image app")
image_number  = 0
my_label = Label()
```

```
images = [
    ImageTk.PhotoImage(Image.open("1.jpg")),
    ImageTk.PhotoImage(Image.open("icon.png")),
    ImageTk.PhotoImage(Image.open("3.jpg"))
    ]

def change_image(dir):
    global image_number
    if(image_number < len(images)-1 and dir == 1):
        image_number += dir
    elif(image_number > 0 and dir == -1):
        image_number += dir
    else:
        return
    load_image(image_number)

def load_image(img_num):
    my_label.config(image=images[img_num])
    my_label.grid(row = 0, column = 0, columnspan=2)

load_image(image_number)
btn_prev = Button(root, text="Prev",
command=lambda:change_image(-1))
btn_prev.grid(row = 1, column = 0)
btn_next = Button(root, text="Next",
command=lambda:change_image(1))
btn_next.grid(row = 1, column = 1)
root.mainloop()
```

Filedialog - otwieranie okna dialogowego

```
from tkinter import *
from tkinter import filedialog
from PIL import ImageTk, Image

root = Tk()
root.geometry('800x500')
img = None

def open_file():
    filename = filedialog.askopenfilename(
        #initialdir="D:\python trial",
        filetypes=(
            ('Images', ('*.png', '*.jpg')),
            ('All files', '*.*')
        )
    )
    global img
    img = ImageTk.PhotoImage(Image.open(filename))
    my_label = Label(image=img).pack()

open_btn = Button(root, text="Open file", command =
open_file)
open_btn.pack()

root.mainloop()
```

Przeglądarka zdjęć ładowanych z okna dialogowego

```
from tkinter import *
from PIL import ImageTk, Image
from tkinter import filedialog
root = Tk()
root.title("Image app")
image_number  = 0
my_label = Label()
images = []

def open_file():
    global image_number
    image_number  = 0
    images.clear()
    filenames = filedialog.askopenfilenames(
        filetypes=(
            ('Images', ('*.png', '*.jpg')),
            ('All files', '*.*')
        )
    )
    for file in filenames:
        images.append(ImageTk.PhotoImage(Image.open(file)))
    load_image(image_number)
```

```
def change_image(dir):
    global image_number
    if(image_number < len(images)-1 and dir == 1):
        image_number += dir
    elif(image_number > 0 and dir == -1):
        image_number += dir
    else:
        return
    load_image(image_number)

def load_image(img_num):
    global my_label
    my_label.config(image=images[img_num])
    my_label.grid(row = 0, column = 0, columnspan=5)

btn_open = Button(root, text="Open images",
command=open_file)
btn_open.grid(row = 1, column = 2)
btn_prev = Button(root, text="Prev",
command=lambda:change_image(-1))
btn_prev.grid(row = 1, column = 1, )
btn_next = Button(root, text="Next",
command=lambda:change_image(1))
btn_next.grid(row = 1, column = 3)

root.mainloop()
```