Mit Iconipy attraktive Pyton-Benutzeroberflächen gestalten

Was ist iconipy?

Iconipy ist ein Python-Paket, das für die Erstellung und Verwaltung von Icons direkt aus dem Programmcode entwickelt wurde.

Es unterstützt mehrere Icon-Sets und ermöglicht die nahtlose Integration mit verschiedenen GUI-Frameworks wie Qt (PyQt6 / PySide6), Tkinter, Kivy, PySimpleGUI, FreeSimpleGUI, NiceGUI, DearPyGUI und weiteren.

Mit Iconipy lassen sich die Eigenschaften von Icons wie Größe, Farbe und Hintergrundfarbe direkt im Code anpassen, was es zu einem praktiscen Werkzeug macht, um die visuelle Anmutung einer Python Anwendung zu optimieren.

Seine Benutzerfreundlichkeit und Flexibilität machen es zu einer beliebten Wahl sowohl für Anfänger als auch für erfahrene Entwickler

Eigenschaften und Vorteile

- **Große Icon-Auswahl**: Iconipy unterstützt mehrere beliebte Icon-Sets (Lucide Icons, Google Material Icons, Lineicons, Boxicons) und ermöglicht eine umfassende Anpassung der Icon-Eigenschaften wie z. B. Größe, Symbol-Farbe und Hintergrundfarbe. (Eigenen Icon-Sets können bei Bedarf hinzugefügt werden)
- **Framework-Kompatibilität:** Nahezu nahtlose Integration mit verschiedenen GUI-Frameworks wie Qt, Tkinter, FreeSimpleGUI, NiceGUI, DearPyGUI, usw.
- **Benutzerfreundlichkeit:** Einfaches und umfangreich dokumentiertes API-Design, das es sowohl Anfängern als auch erfahrenen Entwicklern leicht macht, Icons zu erstellen und zu verwalten**.**
- **Moderne Anmutung:** Hilft bei der Verbesserung der visuellen Attraktivität von Anwendungen durch die Bereitstellung hochwertiger, anpassbarer Symbole.
- **Effiziente Entwicklung:** Spart Zeit und Mühe, die bei der herkömmlichen Erstellung von Icons anfallen ein, so dass sich Entwickler mehr auf die Kernfunktionen konzentrieren können.
- **Open Source**: Das Paket wird unter der MIT-License veröffentlicht.

Wie siehen die Icons aus?

Hier einige Beispielanwendungen, die den Einsatz von mit Iconipy erzeugten Icons zeigen.

Installation

Um iconipy zu installieren, führst du folgenden Befehl im Terminal bzw. in der Kommandozeile aus:

```
pip install iconipy
```

Ist das Paket installiert, kannst du direkt mit der Erzeugung von Icons beginnen.

Ein einfaches Beispiel zur Erzeugung eines Icons als PIL-Image:

```
from iconipy import IconFactory


# Initialize an IconFactory with desired settings
create_icon = IconFactory(icon_size = 20)


# Create an icon
icon_home = create_icon.asPil('house')  # PIL Image
```

Das IconFactory Konzept

Das IconFactory Konzept ist ein praktisches Werkzeug, um das Design von Icons in Python Programmen zu konfigurieren und zu verwalten.

- **Zentralisiertes Design:** Indem eine IconFactory für jeden Stil erstellt wird, werden die Designeinstellungen an einer Stelle des Python-Codes zentral verwaltet. Das bedeutet, dass das Erscheinungsbild aller Icons aktualisiert werden kann, indem einfach die Einstellungen der IconFactory angepasst werden, anstatt Anpassungen für jedes Icon einzeln vorzunehmen.
- **Wiederverwendbarkeit des Codes:** Anstatt Code für jedes Symbol zu duplizieren, definieren man das Design in der jeweiligen IconFactory. Dies fördert die Wiederverwendbarkeit des Codes und reduziert Redundanz, wodurch die Codebasis sauberer und wartbarer wird.
- **Konsistenz:** Die Verwendung einer IconFactory stellt sicher, dass alle damit erstellten Symbole die gleichen Stilattribute wie Größe, Farben und Umrandung haben. Diese Konsistenz ist entscheidend für eine kohärente Benutzeroberfläche.
- **Flexibilität:** Es lassen sich problemlos mehrere IconFactory-Instanzen für verschiedene Zustände oder Themen erstellen. Zum Beispiel eine IconFactory für den normalen Schaltflächenstatus und eine andere für den Mouse-Over-Status.

Passe das Design deiner icon’s an!

Du kannst das Erscheinungsbild deiner Symbole ganz einfach anpassen, indem du für jeden benötigten Stil eine IconFactory erstellst. Zum Beispiel kannst du eine IconFactory für den „normalen“ Button-Zustand und eine andere für den Mouse-Over-Zustand erstellen. Dieser Ansatz ermöglicht es dir, das Design aller Symbole / Buttons schnell zu aktualisieren, indem du eine einzige Stelle in deinem Python-Code änderst.

```
from iconipy import IconFactory
create_button_icon = IconFactory(
                        icon_set = 'lucide',
                        icon_size = (128,64),
                        font_size = 38,
                        font_color = (0, 0, 0, 255), # black solid
                        outline_color = 'dimgrey',
                        outline_width = 6,
                        background_color = '#5500FF',
                        background_radius = 10
)
```

Icon-Sets

Wähle aus den in iconipy integrierten Icon-Sets:

- lucide
- boxicons
- lineicons
- material_icons_regular
- material_icons_round_regular
- material_icons_sharp_regular
- material_icons_outlined_regular

In der Dokumentation erfärst du, wie man weitere Icon-Sets hinzufügen kann.

```
from iconipy import IconFactory
create_icon = IconFactory(icon_set='lucide')
```

Icon-Namen

Um ein Icon erstellen zu können, benötigst du den Namen des Icons.

Mit folgendem Code lässt sich nach passenden Namen suchen:

```
print(create_icon.search('hand'))
```

Alternativ kannst du dir auch alle verfügbaren Namen eines Icon Sets ausgeben lassen:

```
print(create_icon.icon_names)
```

Eine Vorschau erhält man mit:

```
create_icon.show('house')
```

```
from iconipy import IconFactory
create_icon = IconFactory(icon_set='lucide')

print(create_button_icon.search('hand'))

print(create_button_icon.icon_names)
```

Bildformate

Nicht alle GUI-Frameworks unterstützen die gleichen Bildformate. Daher musst du die passende Funktion verwenden, um deine Symbole zu erstellen. Zum Beispiel können einige Frameworks PIL-Bilder direkt verarbeiten, während andere eigenen Formate verwenden (wie QPixmap oder PhotoImage) oder ausschließlich Pfade zu Dateien akzeptieren.

```
icon_home = create_button_icon.asPil('house') # CustomTkinter, wxPython, NiceGUI
icon_save = create_button_icon.asBytes('save') # PySimpleGUI, FreeSimpleGUI
icon_files = create_button_icon.asTkPhotoImage('files') # tkinter basiert
icon_folder = create_button_icon.asTkBitmapImage('folder') # tkinter basiert
icon_reload = create_button_icon.asQPixmap('refresh-cw') # PyQt, PySide
icon_exit_app = create_button_icon.asRawList('log-out') # DearPyGUI
icon_sticker = create_button_icon.asTempFile('sticker') # Kivy and ...
```

Dokumentation

Für einfach verständlichen Beispielcode und eine detaillierte Dokumentation zur Verwendung mit den beliebtesten GUI Toolkits, besuche:

GitHub: https://github.com/digidigital/iconipy

oder

Iconipy's official site: https://iconipy.digidigital.de

„Hello World“ Anwendungsbeispiele

In den nächsten Folien zeigen wir dir ein paar einfache Beispiele, wie du iconipy mit verschiedenen Frameworks nutzen kannst.

Diese Beispiele sind nur ein kleiner Vorgeschmack auf die vielen Anpassungsmöglichkeiten für deine Icons.

Für ausführlichere und realistischere Szenarien, schau dir die Links am Ende jeder Folie an. Dort findest du detaillierte Beispiele und weiterführende Dokumentationen.

Buttons mit icons in tkinter

```
import tkinter as tk

from iconipy import IconFactory

# Initialize IconFactory (default settings except for a custom icon size of 20)
create_icon = IconFactory(icon_size=20)

# Create the main application window
window = tk.Tk()

# Generate an icon of a globe
world_icon = create_icon.asTkPhotoImage('globe')

# Create a button with text and an icon, then add it to the window
button = tk.Button(window, text='Hello World!', image=world_icon, compound=tk.LEFT)
button.grid(row=0, column=0, padx=10, pady=10)

# Run the application's event loop
window.mainloop()
```

Weitere Beispiele https://github.com/digidigital/iconipy/tree/main/demo_programs/tkinter

PySimpleGUI und FreeSimpleGUI Icon Button

```
try:
    import FreeSimpleGUI as sg
except:
    import PySimpleGUI as sg

from iconipy import IconFactory

create_icon = IconFactory(icon_size=20) # Initialize IconFactory
icon_bytes = create_icon.asBytes('globe') # Create the icon as Bytes

layout = [
    [sg.Text('Hello World Button Demo')],
    [sg.Button('', image_data=icon_bytes, button_color=(sg.theme_background_color(), sg.theme_background_color()),
border_width=0, key='Exit')]
]

window = sg.Window('Hello world button!', layout)

while True:
    event, values = window.read()
    if event in (sg.WIN_CLOSED, 'Exit'):
        Break
```

Komplexere Beispiele findest du hier: https://github.com/digidigital/iconipy/tree/main/demo_programs/FreeSimpleGUI_PySimpleGUI

Buttons mit Icons in ttk

```
import tkinter as tk
from tkinter import ttk
from iconipy import IconFactory

# Initialize IconFactory (default settings except for a custom icon size of 20)
create_icon = IconFactory(icon_size=20)

# Create the main application window
window = tk.Tk()

# Generate an icon of a globe
world_icon = create_icon.asTkPhotoImage('globe')

# Create a button with text and an icon, then add it to the window
button = ttk.Button(window, text='Hello World!', image=world_icon, compound=tk.LEFT)
button.grid(row=0, column=0, padx=10, pady=10)

# Run the application's event loop
window.mainloop()
```

Weiter Anwendungsbeispiele hier: https://github.com/digidigital/iconipy/tree/main/demo_programs/ttk

Buttons mit Symbolen in ttkbootstrap

```
import tkinter as tk
import ttkbootstrap as ttk
from iconipy import IconFactory

# Initialize IconFactory (default settings except for a custom icon size of 20)
create_icon = IconFactory(icon_size=20)

# Create the main application window
window = tk.Tk()

# Apply the ttkbootstrap style
style = ttk.Style(theme="cosmo")

# Generate an icon of a globe
world_icon = create_icon.asTkPhotoImage('globe')

# Create a button with text and an icon, then add it to the window
button = ttk.Button(window, text='Hello World!', image=world_icon, compound=tk.LEFT)
button.grid(row=0, column=0, padx=10, pady=10)

# Run the application's event loop
window.mainloop()
```

Weiterer Code: https://github.com/digidigital/iconipy/tree/main/demo_programs/ttkbootstrap

CTkButton mit icon / image

```
import customtkinter as ctk
from iconipy import IconFactory

# Initialize IconFactory (default settings except for a custom icon size of 20 and font color 'white')
create_icon = IconFactory(icon_size=20, font_color='white')

# Create the main application window
window = ctk.CTk()

# Generate an icon of a globe
world_icon = create_icon.asTkPhotoImage('globe')

# Create a button with text and an icon, then add it to the window
button = ctk.CTkButton(window, text='Hello World!', image=world_icon, compound='left')
button.grid(row=0, column=0, padx=10, pady=10)

# Run the application's event loop
window.mainloop()
```

Zusätzliche Beispiele: https://github.com/digidigital/iconipy/tree/main/demo_programs/customtkinter

PyQt buttons mit Icons

```
from PyQt6.QtWidgets import QApplication, QMainWindow, QPushButton, QVBoxLayout, Qwidget
from PyQt6.QtGui import Qicon
from PyQt6.QtCore import Qt
from iconipy import IconFactory
import sys

create_icon = IconFactory(icon_size=18, font_color='dimgrey') # Initialize IconFactory

app = QApplication(sys.argv) # Initialize the application
window = QMainWindow() # Create the main window
central_widget = QWidget() # Create a central widget and set a layout
layout = QVBoxLayout(central_widget)

globe_icon=create_icon.asQPixmap('globe') # Create icon as Qpixmap

button = QPushButton("Hello World!") # Create a button with text and an icon
button.setIcon(QIcon(globe_icon))layout.addWidget(button) # Add the button to the layout

window.setCentralWidget(central_widget) # Set the central widget
window.show() # Show the main window

sys.exit(app.exec()) # Run the application's event loop
```

Weiterer Beispielcode: https://github.com/digidigital/iconipy/tree/main/demo_programs/PyQt6

PySide

```
from PySide6.QtWidgets import QApplication, QMainWindow, QPushButton, QVBoxLayout, Qwidget
from PySide6.QtGui import Qicon
from PySide6.QtCore import Qt
from iconipy import IconFactory
import sys

create_icon = IconFactory(icon_size=18, font_color='dimgrey') # Initialize IconFactory

app = QApplication(sys.argv) # Initialize the application
window = QMainWindow() # Create the main window
central_widget = QWidget() # Create a central widget and set a layout
layout = QVBoxLayout(central_widget)

globe_icon=create_icon.asQPixmap('globe') # Create icon as Qpixmap

button = QPushButton("Hello World!") # Create a button with text and an icon
button.setIcon(QIcon(globe_icon))layout.addWidget(button) # Add the button to the layout

window.setCentralWidget(central_widget) # Set the central widget
window.show() # Show the main window

sys.exit(app.exec()) # Run the application's event loop
```

Mehr: https://github.com/digidigital/iconipy/tree/main/demo_programs/PySide6

wxPython BitmapButton

```
import wx
from iconipy import IconFactory

app = wx.App()
frame = wx.Frame(None, -1, 'wxPython Hello World! Button')
frame.SetDimensions(0, 0, 200, 70)

panel = wx.Panel(frame, wx.ID_ANY)

# Initialize IconFactory for a simple icon with transparent background
create_icon = IconFactory(icon_set = 'lucide',
                icon_size = 18,
                font_color = 'grey') # (reg, green, blue. alpha)

# Create the icon as temp file and load it
# You can replace 'globe' with any valid icon name for the icon set
bmp = wx.Bitmap(create_icon.asTempFile('globe'), wx.BITMAP_TYPE_ANY)

# Create a BitmapButton with the loaded image
button = wx.BitmapButton(panel, id=wx.ID_ANY, bitmap=bmp, size=(64, 32))

frame.Show()
app.MainLoop()
```

Siehe auch: https://github.com/digidigital/iconipy/tree/main/demo_programs/wxPython

Weitere „Hello World“ Code-Beispiele

Das ist noch nicht alles!

Iconipy unterstützt noch mehr Frameworks als in dieser Präsentation erwähnt.

Wenn du wissen möchtest, wie du iconipy mit anderen Toolkits nutzen kannst, besuche diese Websites:

Kivy → https://github.com/digidigital/iconipy/tree/main/demo_programs/kivy

NiceGUI → https://github.com/digidigital/iconipy/tree/main/demo_programs/NiceGUI

DearPyGui → https://github.com/digidigital/iconipy/tree/main/demo_programs/DearPyGUI

appJar → https://github.com/digidigital/iconipy/tree/main/demo_programs/appJar