

# **Pemula? Jangan Bingung! Ini Cara Mudah Bikin Aplikasi GUI di Python dengan Looping atau Event Handling**

## **I. Pendahuluan**

Graphical User Interface (GUI) adalah bagian penting dalam pengembangan aplikasi modern. Dengan GUI, kita bisa berinteraksi dengan aplikasi menggunakan elemen visual seperti tombol, menu, atau ikon. Ini membuat aplikasi jadi lebih mudah digunakan dan lebih intuitif dibandingkan dengan antarmuka berbasis teks atau Command Line Interface (CLI), yang hanya menampilkan perintah berupa tulisan (DomaiNesia, 2023).

Coba bayangkan kalau kamu membuka aplikasi favorit, tapi yang muncul hanya deretan teks tanpa ada tombol atau menu yang bisa kamu klik. Tentu saja itu akan membuat aplikasi jadi membingungkan, bukan? Nah, di sinilah GUI membuat segalanya jadi lebih sederhana dan lebih ramah bagi pengguna.

Dalam dunia pemrograman, Python sering menjadi pilihan banyak orang, baik pemula maupun profesional, untuk membuat GUI. Kenapa? Karena Python terkenal mudah dipelajari dengan sintaks yang sederhana, mirip dengan Bahasa sehari-hari. Tidak hanya itu, python juga memiliki berbagai library GUI yang kuat, seperti Tkinter, yang bisa langsung digunakan tanpa perlu menginstal apapun (Python Software Foundation, 2024). Tkinter adalah library bawaan Python yang memungkinkan kita untuk membuat antarmuka grafis (GUI) dengan cepat dan mudah (RevoU, 2024).

Pada artikel ini, kita akan membahas dua cara utama dalam membuat aplikasi GUI dengan Python: Looping dan Event Handling. Looping adalah metode tradisional yang sering digunakan dalam pemrograman untuk menjalankan tugas secara berulang. Di sisi lain, Event Handling adalah pendekatan yang lebih modern, di mana aplikasi merespons setiap tindakan pengguna, seperti ketika tombol ditekan atau saat mouse digerakkan (Van Rossum & Drake, 2011).

Disini, kita akan melihat kelebihan dan kekurangan dari kedua metode ini, dan kapan sebaiknya menggunakan salah satunya dalam pengembangan aplikasi GUI. Artikel ini sangat cocok buat kamu yang baru belajar Python atau baru mencoba membuat aplikasi GUI. Selain itu, ada contoh-contoh sederhana yang bisa kamu ikuti dan praktekan langsung (Grayson, 1999).

Dengan memahami perbedaan antara looping dan event handling, Kamu akan lebih percaya diri dalam membuat aplikasi GUI yang tidak hanya berfungsi, tapi juga efisien dan responsif. Mari kita mulai perjalanan kita untuk mempelajari lebih dalam tentang GUI di python dan bagaimana kedua pendekatan ini dapat membantumu menciptakan aplikasi yang mudah digunakan.

## **II Pengenalan Looping dan Event Handling**

### **II.A. Apa itu Looping?**

Looping adalah salah satu konsep dasar dalam pemrograman, di mana serangkaian intruksi dijalankan berulang-ulang sampai suatu kondisi terpenuhi. **Bayangkan** kamu meminta computer untuk melakukan tugas yang sama secara terus-menerus tanpa perlu kamu perintah ulang setiap kali. Dalam dunia pemrograman, inilah yang disebut **looping** (Grayson, 1999).

Misalnya, seperti **alarm** yang terus berbunyi sampai kamu mematikannya. Alarm akan terus berulang sesuai dengan pengaturan yang diberikan, tanpa perlu kamu setel ulang setiap kali.

Dalam konteks aplikasi GUI, **looping** berguna untuk tugas-tugas yang perlu dilakukan secara berulang tanpa perlu ada interaksi langsung dari pengguna. Beberapa contoh penggunaan looping dalam aplikasi GUI:

- **Memperbarui jam digital setiap detik** di layar.
- Memeriksa pesan masuk baru secara otomatis pada aplikasi chat.
- Menggerakkan karakter dalam permainan sederhana.

Berikut contoh sederhana penggunaan **looping** aplikasi jam digital dalam aplikasi GUI Python menggunakan **Tkinter** (Summerfield, 2008):

```
import tkinter as tk
import time

def update_clock():
    current_time = time.strftime('%H:%M:%S')
    clock_label.config(text = current_time)
    # Memanggil fungsi ini lagi setelah 1 detik
    root.after(1000, update_clock)

root = tk.Tk()
root.title("Jam")

clock_label = tk.Label(root, font = ('calibri', 40, 'bold'))
clock_label.pack()

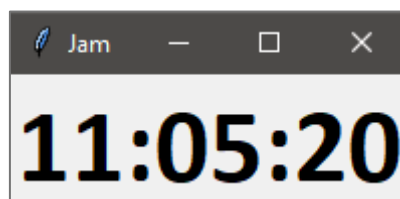
update_clock()
root.mainloop()
```

#### Penjelasan urutan langkah eksekusi kode dan penjelasan:

Kode	Penjelasan
import tkinter as tk	Mengimpor library Tkinter yang akan kita gunakan untuk membuat tampilan GUI
import time	Mengimpor library <b>time</b> untuk mengambil informasi waktu dari system

<code>def update_clock():</code>	Membuat sebuah fungsi yang nanti akan memperbarui waktu di aplikasi.
<code>current_time = time.strftime('%H:%M:%S')</code>	Mendapatkan waktu sekarang dalam format Jam:Menit, misalnya 14:23:45.
<code>clock_label.config(text=current_time)</code>	Mengganti teks di label (bagian yang menampilkan jam) dengan waktu terbaru.
<code>root.after(1000, update_clock)</code>	Menjalankan fungsi ini lagi setelah 1000ms = 1 detik, jadi jam terus diperbarui.
<code>root = tk.Tk()</code>	Membuat jendela aplikasi kita, tempat semua komponen (seperti label jam) akan ditampilkan.
<code>root.title('Jam')</code>	Mengatur judul jendela GUI menjadi 'Jam'
<code>clock_label = tk.Label(root, font=('calibri', 40, 'bold'))</code>	Membuat label untuk menampilkan jam, dengan font 'calibri', ukuran 40, dan tebal (bold).
<code>clock_label.pack()</code>	Menempatkan label jam di jendela. Fungsi pack() ini memastikan label ditampilkan dengan rapi di jendela.
<code>root.mainloop()</code>	Memulai aplikasi. Ini adalah loop utama yang menjaga agar jendela aplikasi tetap terbuka dan terus berjalan.

**Tampilan GUI setelah di eksekusi:**



Aplikasi jam digital yang ditampilkan di atas akan terus memperbarui waktu setiap detik secara otomatis. Angka yang ditampilkan menunjukkan waktu saat ini dalam format **jam:menit**. Aplikasi ini menggunakan looping otomatis di latar belakang untuk memastikan jam terus bertambah setiap detik, sehingga tampilan waktu akan selalu up-to-date sesuai dengan jam sistem pada komputer.

**Kapan Menggunakan Looping?**

Looping sangat berguna untuk tugas-tugas yang terus menerus perlu dijalankan tanpa menunggu Tindakan dari pengguna. Contohnya jika Kamu ingin aplikasi memantau secara otomatis data atau status tanpa harus meminta pengguna untuk melakukan sesuatu.

## II.B. Apa itu Event Handling?

**Event Handling** adalah pendekatan yang berbeda dari looping. Pada event handling, program akan menunggu sampai ada tindakan atau “event” yang dilakukan oleh pengguna atau system. Contohnya, ketika Kamu mengklik tombol, mengetik di keyboard, atau memindahkan mouse, aplikasi akan **merespons** Tindakan ini (Lutz, 2013).

**Analoginya**, bayangkan aplikasi sebagai seorang penjaga pintu. Ia tidak akan membuka pintu kecuali ada seseorang yang mengetuk atau memanggil. Ini adalah prinsip dasar dari event handling, di mana aplikasi **mendengarkan** apa yang dilakukan pengguna, lalu **bereaksi**.

Keuntungan utama dari **event handling**:

- **Efisiensi:** Program tidak perlu memeriksa secara terus-menerus apakah ada tindakan yang terjadi. Program hanya merespons saat pengguna melakukan sesuatu.
- **Responsivitas:** Karena tidak ada proses yang berjalan berulang tanpa tujuan, aplikasi tetap berjalan dengan ringan dan cepat. Aplikasi tidak akan menjadi lambat, karena hanya bekerja saat dibutuhkan, sehingga lebih **responsif**.

Berikut adalah contoh sederhana event handling dalam aplikasi GUI Python (Grayson, 1999):

```
import tkinter as tk

def button_click():
    label.config(text='Tombol diklik!')

root = tk.Tk()
button = tk.Button(root, text="Klik Saya", command=button_click)
button.pack()

label = tk.Label(root, text="Belum ada klik")
label.pack()

root.mainloop()
```

**Penjelasan urutan langkah eksekusi kode dan penjelasan:**

Baris Kode	Penjelasan
import tkinter as tk	Mengimpor modul Tkinter untuk membuat antarmuka GUI di Python dan disingkat menjadi tk.

<code>def button_click():</code>	Mendefinisikan fungsi <b>button_click()</b> yang akan dipanggil saat tombol diklik.
<code>label.config(text='Tombol diklik!')</code>	Begitu tombol diklik, teks di label akan berubah menjadi "Tombol diklik!".
<code>root = tk.Tk()</code>	Membuat jendela utama GUI. Kita menyebutnya sebagai root.
<code>button = tk.Button(root, text="Klik Saya", command=button_click)</code>	Ini adalah tombol yang akan ditampilkan di jendela. Kalau tombol ini diklik, fungsi <b>button_click()</b> akan dijalankan.
<code>button.pack()</code>	Ini untuk menempatkan tombol ke dalam jendela GUI agar bisa terlihat dan digunakan.
<code>label = tk.Label(root, text="Belum ada klik")</code>	Membuat label dengan teks awal " <b>Belum ada klik</b> ". Tulisan ini akan berubah setelah tombolnya diklik.
<code>label.pack()</code>	Menempatkan label di jendela utama GUI dengan metode pack() agar bisa dilihat.
<code>root.mainloop()</code>	Bagian ini menjalankan aplikasi. Jadi, jendela akan terus terbuka dan siap merespons kalau ada tombol yang diklik

#### Tampilan GUI setelah di eksekusi:

Tampilan Awal	Tampilan Setelah Tombol Diklik
	
Saat aplikasi pertama kali dijalankan, tombol di tengah jendela GUI menampilkan teks " <b>Klik Saya</b> ".	Setelah pengguna mengklik tombol " <b>Klik Saya</b> ", teks di label berubah menjadi " <b>Tombol diklik!</b> ".
Di bawah tombol, ada label yang menampilkan teks " <b>Belum ada klik</b> ", menunjukkan bahwa tombol belum ditekan oleh pengguna.	Ini menunjukkan bahwa aplikasi berhasil mendeteksi aksi klik pada tombol, dan fungsi yang merespons klik tersebut memperbarui teks label.

Tidak ada looping di sini. Sebaliknya, aplikasi **mendengarkan** atau **memberikan respons** dari aksi pengguna (dalam hal ini, klik tombol), dan hanya menjalankan kode saat pengguna melakukan ‘klik tombol’.

### Kapan Menggunakan Event Handling?

**Event handling** sangat cocok untuk aplikasi yang interaktif, di mana pengguna diharapkan untuk berinteraksi dengan aplikasi. Contohnya, aplikasi yang memerlukan input pengguna, seperti tombol, input teks, atau klik mouse.

## III. Studi Kasus: Aplikasi Penghitung Sederhana

Untuk memahami perbedaan antara looping dan event handling dalam pengembangan GUI dengan Python, mari kita lihat contoh sederhana berupa aplikasi penghitung (counter).

### Contoh dengan Looping

Pertama, kita akan membuat penghitung menggunakan metode looping:

```
import time
from tkinter import *

root = Tk()
label = Label(root, text="0")
label.pack()

for i in range(100):
    label.config(text=str(i))
    root.update() # Memperbarui GUI
    time.sleep(1)

root.mainloop()
```

Dalam contoh ini, kita menggunakan **loop for** untuk menghitung dari 0 hingga 99. Setiap kali angka bertambah, nilai pada label diperbarui dan GUI juga direfresh menggunakan **root.update()**. Walaupun terlihat sederhana, cara ini punya beberapa kekurangan, seperti:

1. **Aplikasi jadi tidak responsif** selama loop berjalan. Artinya, kamu tidak bisa melakukan hal lain di aplikasi sampai loop selesai. (Summerfield, 2008)
2. **Sulit untuk menghentikan atau mengatur ulang** perhitungan setelah proses sudah dimulai..
3. **Penggunaan CPU menjadi tinggi**, karena loop terus berjalan tanpa henti hingga selesai, yang bisa membuat komputer bekerja lebih keras..

### Penjelasan urutan langkah eksekusi kode dan penjelasan:

Kode	Penjelasan
------	------------

<code>import time</code>	Mengimpor modul <code>time</code> yang akan kita gunakan untuk menambahkan jeda waktu (seperti berhenti sejenak) di program kita.
<code>from tkinter import *</code>	Mengimpor semua yang ada di modul <code>Tkinter</code> . Modul ini akan kita pakai untuk membuat tampilan (GUI) di program kita.
<code>root = Tk()</code>	Membuat jendela utama aplikasi GUI, tempat kita menampilkan semua komponen seperti label, tombol, dll.
<code>label = Label(root, text='0')</code>	Membuat sebuah label di dalam jendela. Label ini awalnya menampilkan teks "0", yang akan kita ubah selama program berjalan.
<code>label.pack()</code>	Menempatkan label ke dalam jendela agar bisa terlihat. Fungsi <code>pack()</code> mengatur posisi label di dalam jendela.
<code>for i in range(100):</code>	Memulai perulangan (loop) yang akan berjalan 100 kali. Setiap kali loop dijalankan, angka <code>i</code> akan berubah.
<code>label.config(text=str(i))</code>	Memperbarui teks di label dengan angka <code>i</code> saat ini. Misalnya, pada iterasi pertama, label akan menampilkan "1", dan terus berubah di setiap iterasi.
<code>root.update()</code>	Memperbarui tampilan GUI agar perubahan pada label langsung terlihat oleh pengguna.
<code>time.sleep(1)</code>	Menjeda (menunggu) selama 1 detik sebelum melanjutkan ke perulangan berikutnya. Ini memberi waktu bagi pengguna untuk melihat perubahan.
<code>root.mainloop()</code>	Menjalankan aplikasi. Fungsi ini menjaga agar jendela tetap terbuka dan aplikasi terus berjalan hingga ditutup oleh pengguna.

#### - **Looping**

Di sini, kita menggunakan looping untuk memperbarui angka yang ditampilkan pada layar setiap 1 detik. Artinya, setiap detik, angka di label akan bertambah satu.

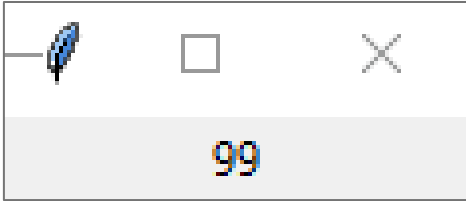
#### - **root.update()**

Fungsi ini memastikan bahwa setiap kali kita memperbarui angka di label, perubahan tersebut langsung terlihat di jendela aplikasi. Jadi, setiap kali angka berubah, pengguna bisa melihatnya secara real-time (langsung tanpa jeda).

#### - **time.sleep(1)**

Bagian ini menambahkan jeda 1 detik antara perubahan angka. Jadi, angka di label tidak berubah terlalu cepat, melainkan setiap detik, seolah-olah kita membuat sebuah penghitung (counter) yang berjalan perlahan.

### Tampilan GUI setelah di eksekusi:

Tampilan Awal GUI	Tampilan Akhir GUI
	
Ketika pertama kali menjalankan program, kamu akan melihat jendela kecil seperti pada gambar diatas namun muncul dengan sebuah angka yang dimulai dari "0" di tengahnya.	Setelah loop dijalankan sebanyak 100 kali, angka terakhir yang muncul di jendela adalah <b>99</b> (Lihat pada gambar diatas).
Jendela ini akan menampilkan angka 0 karena itu adalah nilai awal yang kita atur di program	Pada titik ini, aplikasi akan berhenti memperbarui angka, dan label akan menampilkan angka 99 secara permanen sampai kamu menutup jendela.

### Contoh dengan Event Handling

Sekarang, mari kita lihat implementasi yang sama menggunakan event handling:

```
from tkinter import *

root = Tk()
counter = 0
label = Label(root, text="0")
label.pack()

def increment():
    global counter
    counter += 1
    label.config(text=str(counter))

button = Button(root, text="Tambah", command=increment)
button.pack()

root.mainloop()
```


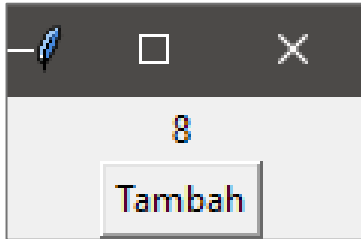
### Penjelasan urutan langkah eksekusi kode dan penjelasan:

Baris Kode	Penjelasan
from tkinter import *	Mengimpor semua fungsi dari Tkinter, yang akan kita gunakan untuk membuat tampilan GUI di Python.



<code>root = Tk()</code>	Membuat jendela utama untuk aplikasi GUI kita. Ini adalah tempat di mana semua elemen akan ditampilkan.
<code>counter = 0</code>	Membuat variabel bernama counter yang akan digunakan untuk menyimpan angka penghitung. Nilainya dimulai dari 0.
<code>label = Label(root, text="0")</code>	Membuat label di dalam jendela. Label ini awalnya menampilkan angka "0", yang nantinya akan berubah.
<code>label.pack()</code>	Menampilkan label di dalam jendela GUI. Fungsi pack() memastikan label muncul di jendela.
<code>def increment():</code>	Mendefinisikan sebuah fungsi bernama increment() yang akan dipanggil saat kita menekan tombol.
<code>global counter</code>	Menyatakan bahwa kita menggunakan variabel counter yang sudah dibuat di luar fungsi (agar bisa diakses di sini).
<code>counter += 1</code>	Setiap kali fungsi increment() dipanggil, angka di counter akan bertambah 1.
<code>label.config(text=str(counter))</code>	Memperbarui teks di label dengan nilai terbaru dari counter, sehingga angka di label berubah.
<code>button = Button(root, text="Tambah", command=increment)</code>	Membuat sebuah tombol dengan teks "Tambah". Saat tombol ini diklik, fungsi increment() akan dipanggil.
<code>button.pack()</code>	Menampilkan tombol Tambah di jendela GUI, sehingga bisa diklik oleh pengguna.
<code>root.mainloop()</code>	Menjalankan aplikasi. Ini menjaga jendela tetap terbuka dan merespons aksi pengguna, seperti menekan tombol.

#### Tampilan GUI setelah di eksekusi:

Tampilan Awal	Tampilan Setelah Tombol Diklik 8 kali
	
Saat aplikasi pertama kali dijalankan, label di dalam jendela GUI menampilkan angka "0". Ini	Setelah tombol " <b>Tambah</b> " diklik <b>8 kali</b> , angka yang ditampilkan di label berubah menjadi " <b>8</b> ".

adalah nilai awal dari variabel <b>counter</b> yang kita tetapkan di program.	
Di bawah angka <b>0</b> , ada tombol dengan teks " <b>Tambah</b> ". Tombol ini digunakan untuk menambah nilai angka pada label setiap kali diklik oleh pengguna.	Setiap kali tombol diklik, fungsi <code>increment()</code> dijalankan, yang menambah angka di variabel <code>counter</code> dan memperbarui angka yang ditampilkan di label.

## Analisis Perbandingan

### 1. Kecepatan Eksekusi:

Dalam contoh sederhana ini, **event handling** lebih cepat karena hanya bekerja saat ada aksi dari pengguna (misalnya ketika tombol diklik). Sedangkan, **looping** terus berjalan tanpa henti, bahkan ketika tidak ada yang dilakukan, jadi bisa sedikit lebih lambat.

### 2. Penggunaan Memori:

**Event handling** biasanya menggunakan lebih sedikit memori karena tidak perlu menjalankan loop terus menerus. Looping membutuhkan lebih banyak memori untuk terus berjalan dan menyimpan statusnya.

### 3. Responsivitas:

Aplikasi yang menggunakan **event handling** lebih responsif karena tidak membuat GUI menjadi "lambat" atau "macet". Event handling hanya bereaksi saat ada aksi dari pengguna, sedangkan looping bisa membuat aplikasi terasa tidak responsif karena terus berjalan di latar belakang.

### 4. Kemudahan Pengembangan:

Untuk aplikasi GUI yang interaktif, **event handling** lebih mudah untuk dibuat dan dipelihara. Kamu hanya perlu menambahkan aksi pada elemen yang diperlukan (misalnya tombol), tanpa perlu mengelola loop yang terus berulang.

## IV. Perbandingan: Kapan Sebaiknya Menggunakan Looping vs Event Handling?

Saat membuat aplikasi GUI, penting untuk memilih antara **Looping** dan **Event Handling** agar aplikasi kita tetap **efisien** dan **responsif**. Mari kita bahas kapan sebaiknya menggunakan masing-masing metode ini.

### Kapan Sebaiknya Menggunakan Looping?

Looping cocok digunakan saat aplikasi harus terus-menerus memperbarui data atau melakukan sesuatu secara otomatis tanpa perlu campur tangan pengguna. Beberapa contoh di mana **looping** sangat bermanfaat adalah:

1. **Pemantauan Real-time:** Ketika aplikasi perlu memperbarui data atau tampilan secara terus-menerus tanpa menunggu input dari pengguna. Misalnya, aplikasi yang menampilkan **grafik pergerakan saham secara real-time** (Summerfield, 2008)
2. **Proses Batch:** Untuk tugas-tugas yang memerlukan pemrosesan banyak data secara berurutan. Misalnya, saat **menganalisis file log** atau melakukan perhitungan berat seperti menghitung hasil dari kumpulan data besar (Lutz, 2013).
3. **Simulasi:** Dalam aplikasi simulasi, di mana perubahan terjadi secara otomatis berdasarkan waktu atau kondisi tertentu. Contohnya, aplikasi untuk **simulasi pertumbuhan populasi** atau simulasi pergerakan planet (Van Rossum & Drake, 2011).
4. **Animasi Otomatis:** Ketika kita ingin membuat **animasi yang berjalan terus-menerus** tanpa pengguna melakukan apapun. Contohnya, membuat **screensaver** atau animasi latar belakang yang bergerak otomatis (Grayson, 1999)

### Kapan Sebaiknya Menggunakan Event Handling?

Event Handling lebih cocok digunakan saat aplikasi perlu merespons tindakan langsung dari pengguna. Ini sangat penting untuk aplikasi yang membutuhkan interaksi pengguna yang signifikan. Berikut beberapa contoh di mana **event handling** lebih efektif:

1. **Aplikasi Formulir:**

Ketika pengguna perlu memasukkan data. Misalnya, pada **formulir pendaftaran** atau **kalkulator**, di mana aplikasi harus merespons setiap kali pengguna memasukkan atau mengubah data.

2. **Game Interaktif:**

Dalam game di mana tindakan pemain harus memicu respons dari aplikasi, misalnya game **puzzle** atau **game arcade sederhana**. Setiap kali pemain menekan tombol atau menggerakkan karakter, aplikasi meresponsnya (Sweigart, 2012).

3. **Antarmuka Pengguna dengan Banyak Elemen:**

Jika aplikasi memiliki banyak komponen interaktif, seperti **menu, tombol, atau slider**, maka **event handling** sangat penting agar aplikasi merespons cepat setiap kali pengguna berinteraksi dengan salah satu elemen. (Phillips, 2018).

4. **Aplikasi Produktivitas:**

Dalam aplikasi seperti **editor teks** atau **aplikasi desain grafis**, setiap tindakan pengguna (misalnya mengetik atau mengedit gambar) harus langsung diproses dan ditampilkan (Summerfield, 2015).

## V. Kesimpulan

Selama kita mempelajari cara membuat aplikasi GUI di Python, kita sudah mengenal dua cara utama: **Looping** dan **Event Handling**. Keduanya sama-sama penting dan membantu kita dalam membuat aplikasi yang berjalan dengan baik dan merespons pengguna dengan cepat.

### Ringkasan

**Looping** dan **Event Handling** adalah dua teknik dasar yang sering digunakan saat membuat aplikasi GUI. Masing-masing memiliki kelebihan dan waktu yang tepat untuk digunakan:

No	Aspek	Looping	Event Handling
1.	<b>Kapan digunakan?</b>	Digunakan untuk tugas-tugas yang berjalan otomatis terus menerus, tanpa butuh interaksi dari pengguna.	Digunakan ketika aplikasi perlu merespons input dari pengguna, seperti klik atau mengetik.
2.	<b>Apa yang dilakukan?</b>	Cocok untuk aplikasi yang harus memperbarui data secara teratur atau memantau sesuatu tanpa henti.	Cocok untuk aplikasi yang butuh merespons pengguna, misalnya saat mereka menekan tombol atau mengisi form.
3.	<b>Contoh Penggunaan</b>	Memperbarui jam digital setiap detik.	Aplikasi pendaftaran dengan form
		Membuat animasi otomatis berjalan.	Game interaktif.
		Pemantauan sensor suhu secara otomatis.	Aplikasi dengan banyak tombol atau input.
4.	<b>Kelebihan</b>	Baik untuk tugas yang tidak memerlukan interaksi pengguna.	Membuat aplikasi lebih interaktif dan responsif terhadap tindakan pengguna.
		Menangani tugas berulang secara otomatis.	Sangat cocok untuk aplikasi yang perlu banyak aksi dari pengguna (aplikasi interaktif).
5.	<b>Keterbatasan</b>	Bisa membuat aplikasi kurang responsif jika digunakan untuk menangani banyak tugas sekaligus.	Tidak cocok untuk tugas yang memerlukan pemantauan atau pemrosesan berkelanjutan.

**Penting untuk diingat**, memilih antara Looping atau Event Handling tergantung pada apa yang aplikasi kamu butuhkan. Setiap aplikasi punya kebutuhan yang berbeda, jadi pilihlah cara yang paling cocok.

Dalam aplikasi yang lebih rumit, sering kali kita perlu menggabungkan **Looping** dan **Event Handling** agar aplikasi bisa bekerja dengan baik dan tetap responsif. Dengan menggabungkan keduanya, kita bisa mendapatkan hasil yang lebih optimal.

## Referensi

- RevoU. (2024). *10 Contoh Program Python Tkinter untuk Membangun GUI 2024* | RevoU. <https://revou.co/panduan-teknis/python-tkinter>
- DomaiNesia. (2023). *Apa itu GUI? Pahami Definisi dan Cara Kerja GUI*. <https://www.domainesia.com/berita/gui-adalah/>
- Python Software Foundation. (2024). *Graphical User Interfaces with Tk — Python 3.12.6 documentation*. <https://docs.python.org/3/library/tk.html>
- Grayson, J. (1999). *Python and Tkinter Programming*. Manning. <https://books.google.co.id/books?id=1f1lQgAACAAJ>
- Lutz, M. (2013). *Learning python: Powerful object-oriented programming*.
- Phillips, D. (2018). Python 3 Object-Oriented Programming Third Edition Build robust and maintainable software with object-oriented design patterns in Python 3.8. In *Packt Publishing Ltd*.
- Summerfield, M. (2008). Rapid Gui Programming with Python and Qt: The Definite Guid to PyQt Programming. In *Rapid GUI Programming with Python and Qt*: (Vol. 54, Issue 2). <https://doi.org/10.1093/infdis/jit776>
- Summerfield, M. (2015). *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming*. Prentice Hall. <https://books.google.co.id/books?id=0cUGswEACAAJ>
- Sweigart, A. (2012). Making Games with Python & Pygame. In *Albert Sweigart*.
- Van Rossum, G., & Drake, F. L. (2011). *The Python Language Reference Manual: For Python Version 3.2*. Network Theory Limited. <https://books.google.co.id/books?id=Ut4BuQAACAAJ>