day2

January 11, 2025

1 Gradio Day!

Today we will build User Interfaces using the outrageously simple Gradio framework.

Prepare for joy!

Please note: your Gradio screens may appear in 'dark mode' or 'light mode' depending on your computer settings.

[1]:
```
# imports

import os
import requests
from bs4 import BeautifulSoup
from typing import List
from dotenv import load_dotenv
from openai import OpenAI
import google.generativeai
import anthropic
```

[2]:
```
import gradio as gr # oh yeah!
```

[3]:
```
# Load environment variables in a file called .env
# Print the key prefixes to help with any debugging

load_dotenv()
openai_api_key = os.getenv('OPENAI_API_KEY')
anthropic_api_key = os.getenv('ANTHROPIC_API_KEY')
google_api_key = os.getenv('GOOGLE_API_KEY')

if openai_api_key:
    print(f"OpenAI API Key exists and begins {openai_api_key[:8]}")
else:
    print("OpenAI API Key not set")

if anthropic_api_key:
    print(f"Anthropic API Key exists and begins {anthropic_api_key[:7]}")
else:
    print("Anthropic API Key not set")
```

```
if google_api_key:
    print(f"Google API Key exists and begins {google_api_key[:8]}")
else:
    print("Google API Key not set")
```

```
OpenAI API Key exists and begins sk-proj-
Anthropic API Key exists and begins sk-ant-
Google API Key exists and begins AIzaSyDn
```

[4]:
```
# Connect to OpenAI, Anthropic and Google; comment out the Claude or Google␣
↪lines if you're not using them

openai = OpenAI()

claude = anthropic.Anthropic()

google.generativeai.configure()
```

[5]:
```
# A generic system message - no more snarky adversarial AIs!

system_message = "You are a helpful assistant"
```

[6]:
```
# Let's wrap a call to GPT-4o-mini in a simple function

def message_gpt(prompt):
    messages = [
        {"role": "system", "content": system_message},
        {"role": "user", "content": prompt}
      ]
    completion = openai.chat.completions.create(
        model='gpt-4o-mini',
        messages=messages,
    )
    return completion.choices[0].message.content
```

[7]:
```
message_gpt("What is today's date?")
```

[7]:
```
"Today's date is October 31, 2023."
```

1.1 User Interface time!

[8]:
```
# here's a simple function

def shout(text):
    print(f"Shout has been called with input {text}")
    return text.upper()
```

[9]:
```
shout("hello")
```

```
Shout has been called with input hello
```

[9]:
```
'HELLO'
```

[10]:
```
# The simplicty of gradio. This might appear in "light mode" - I'll show you␣
↪how to make this in dark mode later.

gr.Interface(fn=shout, inputs="textbox", outputs="textbox").launch()
```

```
* Running on local URL:  http://127.0.0.1:7860

To create a public link, set `share=True` in `launch()`.
```

```
<IPython.core.display.HTML object>
```

[10]:

[11]:
```
# Adding share=True means that it can be accessed publically
# A more permanent hosting is available using a platform called Spaces from␣
↪HuggingFace, which we will touch on next week
# NOTE: Some Anti-virus software and Corporate Firewalls might not like you␣
↪using share=True. If you're at work on on a work network, I suggest skip␣
↪this test.

gr.Interface(fn=shout, inputs="textbox", outputs="textbox",␣
↪flagging_mode="never").launch(share=True)
```

```
* Running on local URL:  http://127.0.0.1:7861
* Running on public URL: https://1f2045e68993e1bf4d.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU
upgrades, run `gradio deploy` from the terminal in the working directory to
deploy to Hugging Face Spaces (https://huggingface.co/spaces)
```

```
<IPython.core.display.HTML object>
```

[11]:

```
Shout has been called with input Markku
Shout has been called with input Laine
Shout has been called with input Pekka
Shout has been called with input Markku
Shout has been called with input Laine
```

[12]:
```
# Adding inbrowser=True opens up a new browser window automatically

gr.Interface(fn=shout, inputs="textbox", outputs="textbox",␣
↪flagging_mode="never").launch(inbrowser=True)
```

```
* Running on local URL:  http://127.0.0.1:7862

To create a public link, set `share=True` in `launch()`.

<IPython.core.display.HTML object>
```

[12]:

```
Shout has been called with input Markku
```

1.2 Forcing dark mode

Gradio appears in light mode or dark mode depending on the settings of the browser and computer. There is a way to force gradio to appear in dark mode, but Gradio recommends against this as it should be a user preference (particularly for accessibility reasons). But if you wish to force dark mode for your screens, below is how to do it.

[13]:
```
# Define this variable and then pass js=force_dark_mode when creating the␣
↪Interface

force_dark_mode = """
function refresh() {
    const url = new URL(window.location);
    if (url.searchParams.get('__theme') !== 'dark') {
        url.searchParams.set('__theme', 'dark');
        window.location.href = url.href;
    }
}
"""
gr.Interface(fn=shout, inputs="textbox", outputs="textbox",␣
↪flagging_mode="never", js=force_dark_mode).launch()
```

```
* Running on local URL:  http://127.0.0.1:7863

To create a public link, set `share=True` in `launch()`.

<IPython.core.display.HTML object>
```

[13]:

```
Shout has been called with input Markku
```

[14]:
```
# Inputs and Outputs

view = gr.Interface(
    fn=shout,
    inputs=[gr.Textbox(label="Your message:", lines=6)],
    outputs=[gr.Textbox(label="Response:", lines=8)],
    flagging_mode="never"
)
```

```
view.launch()
```

```
* Running on local URL:  http://127.0.0.1:7864

To create a public link, set `share=True` in `launch()`.
```

```
<IPython.core.display.HTML object>
```

[14]:

```
Shout has been called with input Markku
Laine
```

[15]:
```
# And now - changing the function from "shout" to "message_gpt"

view = gr.Interface(
    fn=message_gpt,
    inputs=[gr.Textbox(label="Your message:", lines=6)],
    outputs=[gr.Textbox(label="Response:", lines=8)],
    flagging_mode="never"
)
view.launch()
```

```
* Running on local URL:  http://127.0.0.1:7865

To create a public link, set `share=True` in `launch()`.
```

```
<IPython.core.display.HTML object>
```

[15]:

[16]:
```
# Let's use Markdown
# Are you wondering why it makes any difference to set system_message when it's␣
  ↪not referred to in the code below it?
# I'm taking advantage of system_message being a global variable, used back in␣
  ↪the message_gpt function (go take a look)
# Not a great software engineering practice, but quite sommon during Jupyter␣
  ↪Lab R&D!

system_message = "You are a helpful assistant that responds in markdown"

view = gr.Interface(
    fn=message_gpt,
    inputs=[gr.Textbox(label="Your message:")],
    outputs=[gr.Markdown(label="Response:")],
    flagging_mode="never"
)
view.launch()
```

```
* Running on local URL:  http://127.0.0.1:7866
```

```
To create a public link, set `share=True` in `launch()`.
```

```
<IPython.core.display.HTML object>
```

[16]:

[17]:
```python
# Let's create a call that streams back results
# If you'd like a refresher on Generators (the "yield" keyword),
# Please take a look at the Intermediate Python notebook in week1 folder.

def stream_gpt(prompt):
    messages = [
        {"role": "system", "content": system_message},
        {"role": "user", "content": prompt}
      ]
    stream = openai.chat.completions.create(
        model='gpt-4o-mini',
        messages=messages,
        stream=True
    )
    result = ""
    for chunk in stream:
        result += chunk.choices[0].delta.content or ""
        yield result
```

[18]:
```python
view = gr.Interface(
    fn=stream_gpt,
    inputs=[gr.Textbox(label="Your message:")],
    outputs=[gr.Markdown(label="Response:")],
    flagging_mode="never"
)
view.launch()
```

```
* Running on local URL:  http://127.0.0.1:7867

To create a public link, set `share=True` in `launch()`.
```

```
<IPython.core.display.HTML object>
```

[18]:

[19]:
```python
def stream_claude(prompt):
    result = claude.messages.stream(
        model="claude-3-haiku-20240307",
        max_tokens=1000,
        temperature=0.7,
        system=system_message,
        messages=[
            {"role": "user", "content": prompt},
        ],
    )
    response = ""
    with result as stream:
        for text in stream.text_stream:
            response += text or ""
            yield response
```

[20]:
```
view = gr.Interface(
    fn=stream_claude,
    inputs=[gr.Textbox(label="Your message:")],
    outputs=[gr.Markdown(label="Response:")],
    flagging_mode="never"
)
view.launch()
```

```
* Running on local URL:  http://127.0.0.1:7868

To create a public link, set `share=True` in `launch()`.

<IPython.core.display.HTML object>
```

[20]:

1.3 Minor improvement

I've made a small improvement to this code.

Previously, it had these lines:

```
for chunk in result:
  yield chunk
```

There's actually a more elegant way to achieve this (which Python people might call more 'Pythonic'):

```
yield from result
```

I cover this in more detail in the Intermediate Python notebook in the week1 folder - take a look if you'd like more.

[21]:
```
def stream_model(prompt, model):
    if model=="GPT":
        result = stream_gpt(prompt)
    elif model=="Claude":
        result = stream_claude(prompt)
    else:
        raise ValueError("Unknown model")
    yield from result
```

[22]:
```
view = gr.Interface(
    fn=stream_model,
    inputs=[gr.Textbox(label="Your message:"), gr.Dropdown(["GPT", "Claude"],␣
 ↪label="Select model", value="GPT")],
    outputs=[gr.Markdown(label="Response:")],
    flagging_mode="never"
)
view.launch()
```

```
* Running on local URL:  http://127.0.0.1:7869

To create a public link, set `share=True` in `launch()`.

<IPython.core.display.HTML object>
```

[22]:

2 Building a company brochure generator

Now you know how - it's simple!

Before you read the next few cells

Try to do this yourself - go back to the company brochure in week1, day5 and add a Gradio UI to the end. Then come and look at the solution.

[23]:
```
# A class to represent a Webpage

class Website:
    url: str
    title: str
    text: str

    def __init__(self, url):
        self.url = url
        response = requests.get(url)
        self.body = response.content
        soup = BeautifulSoup(self.body, 'html.parser')
        self.title = soup.title.string if soup.title else "No title found"
        for irrelevant in soup.body(["script", "style", "img", "input"]):
            irrelevant.decompose()
        self.text = soup.body.get_text(separator="\n", strip=True)

    def get_contents(self):
        return f"Webpage Title:\n{self.title}\nWebpage Contents:\n{self.
 ↪text}\n\n"
```

[24]:
```
# With massive thanks to Bill G. who noticed that a prior version of this had a␣
 ↪bug! Now fixed.
```

```
system_message = "You are an assistant that analyzes the contents of a company␣
↪website landing page \
and creates a short brochure about the company for prospective customers,␣
↪investors and recruits. Respond in markdown."
```

[25]:
```
def stream_brochure(company_name, url, model):
    prompt = f"Please generate a company brochure for {company_name}. Here is␣
↪their landing page:\n"
    prompt += Website(url).get_contents()
    if model=="GPT":
        result = stream_gpt(prompt)
    elif model=="Claude":
        result = stream_claude(prompt)
    else:
        raise ValueError("Unknown model")
    yield from result
```

[26]:
```
view = gr.Interface(
    fn=stream_brochure,
    inputs=[
        gr.Textbox(label="Company name:"),
        gr.Textbox(label="Landing page URL including http:// or https://"),
        gr.Dropdown(["GPT", "Claude"], label="Select model")],
    outputs=[gr.Markdown(label="Brochure:")],
    flagging_mode="never"
)
view.launch()
```

```
* Running on local URL:  http://127.0.0.1:7870

To create a public link, set `share=True` in `launch()`.

<IPython.core.display.HTML object>
```

[26]:

[ ]: