User Interfaces in Python

gradio and textual

Approaches

- So far
  - Desktop application
    - tkinter (bundled, cross-platform)
    - GTK (Linux mostly)
    - PyQt (cross-platform)
    - kivy (cross-platform)
  - Web-based
    - Jupyter

But wait, there is more!

- Web-based
  - gradio: https://github.com/gradio-app/gradio
- Terminal
  - textual: https://github.com/Textualize/textual

gradio

- Use-case: build **demos** and **share** them
- Wrap **function** in a web-based interface (input parameters and display results)
- Documentation: https://gradio.app/docs/
- Input types:

  text, number, checkbox (+ groups), file, slider, dropdown, color picker, audio, image, video, dataframe, timeseries, 3D model
- Output types:
  - most of the above, barplot, lineplot, plot, gallery, json

Basics

- Install*

  pip install gradio
- Top-level class to build interface:

  gradio.Interface
- Wrapped function:
  - name referenced via **fn** parameter of Interface object
  - method inputs get mapped to **inputs** parameter of Interface object
  - method return value/tuple gets mapped to **outputs** parameter of Interface object
- Launch application **app.y** with:

  gradio app.py
- Open in browser:

  http://127.0.0.1:7860

* When using virtual env, the env must be activated - due to usage of uvicorn

Hello world

```
import gradio as gr

def greet(name):
    return "Hello " + name + "!"

demo = gr.Interface(fn=greet, inputs="text", outputs="text")

demo.launch()
```

Multiple I/O

```
import gradio as gr

def greet(name, is_morning, temperature):
    salutation = "Good morning" if is_morning else "Good evening"
    greeting = f"{salutation} {name}. It is {temperature} degrees today"
    celsius = (temperature - 32) * 5 / 9
    return greeting, round(celsius, 2)

demo = gr.Interface(
    fn=greet,
    inputs=["text", "checkbox", gr.Slider(0, 100)],
    outputs=["text", "number"],
)

if __name__ == "__main__":
    demo.launch()
```

More...

- Two ways of defining type:
  - textual: "text" or "image"
  - via object: gr.Text(...) or gr.Image(...)
- Reactive interfaces: live, streaming
- Blocks: low-level API for more flexible layouts and interfaces
- lots more...

textual

- Use-case: terminal application
- Adds interactivity to Rich (rich text formatting in terminal)
- Cross-platform: Linux, macOS, Windows
- Features: mouse support, 16.7mio colors, layout engine, reusable components

Basics

- Install

  pip install "textual[dev]"
- Demo

  python -m textual
- Modules for building an application:
  - textual.app: App, ComposeResult, ...
  - textual.widgets: Button, Header, Footer, Label, ...
  - textual.containers: Container, ...
- App/widget have **compose** method which returns an iterable of widgets (**ComposeResult**)
- Some events: on_key, on_input_changed, on_button_pressed, on_mount

Examples

- Let's look at some examples
  - pride flag
  - calculator
  - dictionary (uses https://api.dictionaryapi.dev/api/v2/entries/en/{word})
  - tree (built from JSON)
  - markdown browser