Flet-Python
A cross-platform python framework

مستندات فارسی فریمورک Flet ارئه شده توسط گروه تلگرامی

Flet-Python

مقدمه

فلت چیست؟

فلت یک فریمورک است که امکان ساخت برنامه‌های وب، دسکتاپ و موبایل را در پایتون بدون نیاز به تجربه قبلی در توسعه فرانت‌اند فراهم می‌کند.

شما می‌توانید با استفاده از کنترل‌های فلت که بر پایه Flutter شرکت گوگل هستند، رابط کاربری (UI) برای برنامه خود بسازید. فلت فراتر از صرفاً بسته‌بندی ویجت‌های فلاتر عمل می‌کند. این فریمورک با ترکیب ویجت‌های کوچکتر، ساده‌سازی پیچیدگی‌ها، پیاده‌سازی بهترین روش‌های رابط کاربری و اعمال پیش‌فرض‌های منطقی،

ویژگی خاص خود را اضافه می‌کند. این ویژگی‌ها تضمین می‌کنند که برنامه‌های شما بدون نیاز به تلاش‌های اضافی در طراحی، شیک و حرفه‌ای به نظر برسند.

مثالی از یک برنامه فلت

در اینجا یک نمونه برنامه "شمارنده" را ایجاد می‌کنیم:

یک فایل به نام "counter.py" ایجاد می‌کنیم.

```
import flet as ft

def main(page: ft.Page):
    page.title = "Flet counter example"
    page.vertical_alignment = ft.MainAxisAlignment.CENTER

    txt_number = ft.TextField(value="0", text_align=ft.TextAlign.RIGHT,    width=100)

    def minus_click(e):
        txt_number.value = str(int(txt_number.value) - 1)
        page.update()

    def plus_click(e):
        txt_number.value = str(int(txt_number.value) + 1)
        page.update()

    page.add(
        ft.Row(
            [
                ft.IconButton(ft.icons.REMOVE, on_click=minus_click),
                txt_number,
                ft.IconButton(ft.icons.ADD, on_click=plus_click),
            ],
            alignment=ft.MainAxisAlignment.CENTER,
        )
    )

ft.app(main)
```

برای اجرای برنامه، ابتدا ماژول و پکیج فلت را نصب کنید (یک محیط جدید فلت ایجاد کنید):

```
pip install flet
```

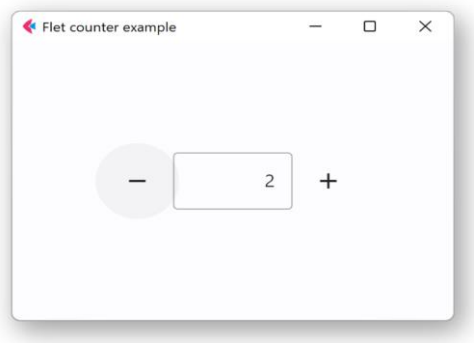
سپس برنامه را اجرا کنید:

```
flet run counter.py
```

این برنامه به صورت یک پنجره در سیستم‌عاملتان اجرا خواهد شد - چه جایگزین خوبی برای Electron!

macOS

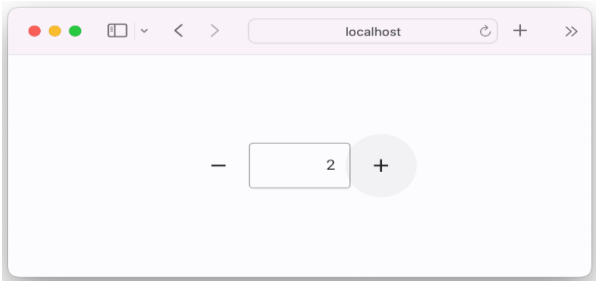
Windows



اکنون، برنامه خود را تحت یک برنامه وب اجرا کنید:

```
flet run --web counter.py
```

یک پنجره یا تب جدید مرورگر باز می شود:



بیایید شروع کنیم:

پیش از اینکه بتوانید اولین برنامه فلت خود را ایجاد کنید، باید محیط توسعه خود را راه‌اندازی کنید که نیاز به پایتون ۳.۸ یا بالاتر و پکیج `flet` دارد.

ما توصیه می‌کنیم که فلت را در یک محیط مجازی نصب کنید که می‌توان این کار را به چندین روش مختلف انجام داد.

پیشنیازها

لینوکس

اگر قصد دارید فلت را روی لینوکس نصب کنید، پیش‌نیازهای اضافی وجود دارد که باید رعایت کنید.

WSL (زیرسیستم ویندوز برای لینوکس)

برنامه‌های فلت را می‌توان بر روی WSL 2 (Windows Subsystem for Linux 2) اجرا کرد. اگر با خطای `cannot open display` مواجه شدید، می‌توانید از این راهنما برای رفع مشکل استفاده کنید.

ماژول venv پایتون

شما می‌توانید با اجرای دستورات زیر در ترمینال خود یک محیط مجازی ایجاد کنید:

Windows

```
md first-flet-app
cd first-flet-app
python -m venv .venv
.venv\Scripts\activate
```

macOS/Linux

```
mkdir first-flet-app
cd first-flet-app
python3 -m venv .venv
source .venv/bin/activate
```

پس از فعال کردن محیط مجازی، خواهید دید که پیشوند `(.venv)` در ابتدای خط فرمان شما نمایش داده می‌شود.

اکنون می‌توانید آخرین نسخه فلت را در محیط مجازی `.venv` نصب کنید:

```
pip install flet
```

برای بررسی نسخه نصب شده فلت، می‌توانید از دستور زیر استفاده کنید:

```
flet --version
```

می‌توانید اطلاعات بیشتری درباره ماژول `venv` پایتون در اینجا بخوانید.

اکنون شما آماده‌اید تا اولین برنامه فلت خود را ایجاد کنید.

Poetry

راه دیگر برای راه‌اندازی یک محیط مجازی برای پروژه فلت شما استفاده از Poetry است.

پس از نصب Poetry، دستور زیر را در ترمینال خود اجرا کنید:

```
poetry new first-flet-app
```

این دستور یک دایرکتوری جدید به نام `first-flet-app` با ساختار زیر ایجاد می‌کند:

```
first-flet-app/
├── pyproject.toml
├── README.md
├── first-flet-app/
│   └── __init__.py
└── tests/
    └── __init__.py
```

اکنون می‌توانید وابستگی(دیپندنسی) فلت را به پروژه خود اضافه کنید:

```
cd first-flet-app
poetry add flet
```

برای بررسی نسخه نصب شده فلت، می‌توانید از دستور زیر استفاده کنید:

```
poetry run flet --version
```

اکنون شما آماده‌اید تا اولین برنامه فلت خود را ایجاد کنید.

نکته:

هنگام ایجاد و اجرای برنامه فلت با استفاده از Poetry،
باید قبل از هر دستوری از `poetry run` استفاده کنید!

ایجاد یک برنامه جدید فلت

برای ایجاد یک برنامه "ساده" فلت، دستور زیر را اجرا کنید:

```
flet create <project-name>
```

برای مثال:

```
flet create my_flet_app
```

`<project-name>` به عنوان نام دایرکتوری خروجی استفاده خواهد شد.

فلت یک دایرکتوری به نام `<project-name>` ایجاد می‌کند و فایل `main.py` با محتوای زیر در آن قرار می‌دهد:

```
import flet as ft

def main(page: ft.Page):
    page.add(ft.SafeArea(ft.Text("Hello, Flet!")))

ft.app(main)
```

نکته:

برای ایجاد برنامه فلت خود در دایرکتوری فعلی، دستور زیر را اجرا کنید:

```
flet create .
```

برنامه فلت دارای تابع `main()` است که در آن عناصر رابط کاربری (کنترل‌ها) را به صفحه یا پنجره اضافه می‌کنید. برنامه با اجرای تابع `ft.app()` که برنامه فلت را مقداردهی اولیه کرده و `main()` را اجرا می‌کند، به پایان می‌رسد.

برای ایجاد یک برنامه فلت از قالب "شمارنده"، دستور زیر را اجرا کنید:

```
flet create --template counter <project-name>
```

یا برای ایجاد برنامه فلت از قالب شمارنده در دایرکتوری فعلی خود، از این دستور استفاده کنید:

```
flet create --template counter .
```

می‌توانید اطلاعات بیشتری درباره دستور `flet create` از اینجا بیابید.

اکنون بیایید فلت را در عمل مشاهده کنیم و برنامه را اجرا کنیم!

اجرای برنامه فلت:

برنامه فلت را می‌توان به عنوان یک برنامه تحت دسکتاپ یا وب با استفاده از دستور ساده `flet run` اجرا کرد.

اجرا تحت یک برنامه دسکتاپی:

برای اجرای برنامه فلت به عنوان یک برنامه دسکتاپ، از دستور زیر استفاده کنید:

```
flet run
```

این دستور فایل `main.py` موجود در دایرکتوری فعلی را اجرا می‌کند.

اگر نیاز به اجرای یک فایل دیگر دارید، از دستور زیر استفاده کنید:

```
flet run [script]
```

برای اجرای `main.py` که در یک دایرکتوری دیگر قرار دارد، مسیر مطلق یا نسبی دایرکتوری که فایل در آن قرار دارد را ارائه دهید. برای مثال:

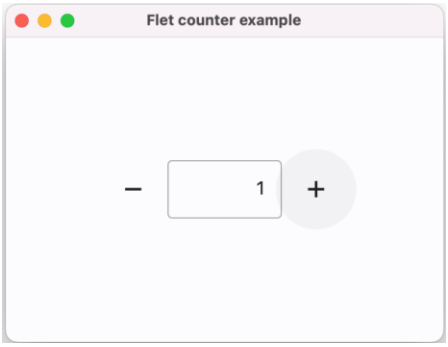
```
flet run /Users/JohnSmith/Documents/projects/flet-app
```

برای اجرای یک اسکریپت با نامی غیر از `main.py`، مسیر مطلق یا نسبی فایل را ارائه دهید. برای مثال:

```
flet run counter.py
```

برنامه در یک پنجره سیستم‌عاملتان اجرا خواهد شد:

macOS



Windows

اجرا یک برنامه تحت وب:

برای اجرای برنامه فلت به عنوان یک برنامه وب، از دستور زیر استفاده کنید:

```
flet run --web [script]
```

یک پنجره یا تب جدید در مرورگر باز خواهد شد و برنامه از پورت TCP به صورت تصادفی استفاده خواهد کرد:

برای اجرای برنامه فلت به عنوان یک برنامه وب روی یک پورت ثابت، از گزینه `--port` یا `-p` استفاده کنید. برای مثال:

```
flet run --web --port 8000 app.py
```

بارگذاری خودکار (Hot reload)

به طور پیش‌فرض، فلت فایل اسکریپتی که اجرا شده است را مشاهده می‌کند و هر زمان که فایل تغییر و ذخیره شود، برنامه را مجدداً بارگذاری می‌کند. اما فلت تغییرات در سایر فایل‌ها را مشاهده نمی‌کند.

برای مشاهده تمام فایل‌های موجود در همان دایرکتوری، از دستور زیر استفاده کنید:

```
poetry run flet run -d [script]
```

برای مشاهده دایرکتوری اسکریپت و تمام زیردایرکتوری‌ها به صورت بازگشتی، از این دستور استفاده کنید:

```
poetry run flet run -d -r [script]
```

شما می‌توانید اطلاعات بیشتری درباره دستور `flet run` اینجا بیابید.

کنترل های فلت:

رابط کاربری از **کنترل‌ها** (یا ویجت‌ها) ساخته شده است. برای اینکه کنترل‌ها برای کاربر قابل مشاهده باشند، باید به یک صفحه `Page` یا داخل کنترل‌های دیگر اضافه شوند. صفحه بالاترین کنترل است. قرار دادن کنترل‌ها درون یکدیگر می‌تواند به عنوان یک درخت با صفحه به عنوان ریشه نمایش داده شود.

کنترل‌ها کلاس‌های معمولی پایتون هستند. برای ایجاد نمونه‌های کنترل، از سازنده‌ها با پارامترهایی که با ویژگی‌های آن‌ها مطابقت دارند استفاده کنید. برای مثال:

```
    t = ft.Text(value="Hello, world!", color="green")
```

برای نمایش کنترل در یک صفحه، آن را به لیست کنترل‌ها `controls` صفحه اضافه کرده و سپس با استفاده از `page.update()` تغییرات صفحه را به مرورگر یا کلاینت دسکتاپ ارسال کنید:

```
import flet as ft

def main(page: ft.Page):
    t = ft.Text(value="Hello, world!", color="green")
    page.controls.append(t)
    page.update()

ft.app(target=main)
```

در این مثال، یک کنترل متنی با متن "Hello, world!" و رنگ سبز ایجاد شده و به صفحه اضافه می‌شود. سپس `page.update()` برای ارسال تغییرات به مرورگر یا کلاینت دسکتاپ فراخوانی می‌شود.

نکته:

در مثال‌های بعدی، فقط محتوای تابع `main` نمایش داده خواهد شد.

می‌توانید ویژگی‌های کنترل‌ها را تغییر دهید و رابط کاربری در `page.update()` بعدی به‌روزرسانی خواهد شد:

```
    t = ft.Text()

    for i in range(10):
        t.value = f"Step {i}"
        page.update()
        time.sleep(1)
```

برخی از کنترل‌ها "کنترل‌های محفظه" (container controls) هستند (مانند Page) که می‌توانند شامل کنترل‌های دیگر باشند. به عنوان مثال، کنترل `Row` اجازه می‌دهد که کنترل‌های دیگر را به صورت ردیفی بچینید:

```
page.add(
    ft.Row(controls=[
        ft.Text("A"),
        ft.Text("B"),
        ft.Text("C")
        ])
    )
```

یا قرار دادن `TextField` و `ElevatedButton` در کنار یکدیگر:

```
page.add(
        ft.Row(controls=[
            ft.TextField(label="Your name"),
            ft.ElevatedButton(text="Say my name!")
            ])
    )
```

دستور `page.update()` به اندازه کافی هوشمند است که فقط تغییرات انجام‌شده از آخرین فراخوانی را ارسال کند، بنابراین می‌توانید تعدادی کنترل جدید به صفحه اضافه کنید، برخی از آن‌ها را حذف کنید، ویژگی‌های سایر کنترل‌ها را تغییر دهید و سپس `page.update()` را برای انجام یک به‌روزرسانی گروهی فراخوانی کنید. برای مثال:

```
for i in range(10):
    page.controls.append(ft.Text(f"Line {i}"))
    if i > 4:
        page.controls.pop(0)
    page.update()
    time.sleep(0.3)
```

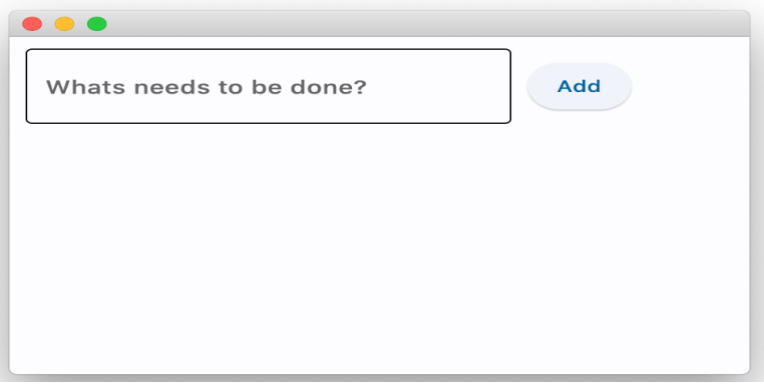
برخی از کنترل‌ها، مانند دکمه‌ها، می‌توانند دارای هندلرهای رویداد باشند که به ورودی کاربر واکنش نشان می‌دهند. برای مثال، `ElevatedButton.on_click`:

```
def button_clicked(e):
        page.add(ft.Text("Clicked!"))

    page.add(ft.ElevatedButton(text="Click me", on_click=button_clicked))
```

و مثال پیشرفته‌تر برای یک برنامه ساده To-Do:

```
import flet as ft

def main(page):
    def add_clicked(e):
        page.add(ft.Checkbox(label=new_task.value))
        new_task.value = ""
        new_task.focus()
        new_task.update()

    new_task = ft.TextField(hint_text="What's needs to be done?", width=300)
    page.add(ft.Row([new_task, ft.ElevatedButton("Add", on_click=add_clicked)]))

ft.app(target=main)
```

در این مثال، وقتی دکمه "Add" کلیک می‌شود، یک چک‌باکس جدید با برچسب وظیفه جدید به صفحه اضافه می‌شود.



اطلاعات بیشتر:

فلت یک مدل رابط کاربری دستوری (imperative) را پیاده‌سازی می‌کند، جایی که شما به‌طور "دستی" رابط کاربری برنامه را با کنترل‌های حالت‌دار (stateful) می‌سازید و سپس با به‌روزرسانی ویژگی‌های کنترل، آن را تغییر می‌دهید. در مقابل، فلاتر مدل اعلامی (declarative) را پیاده‌سازی می‌کند، جایی که رابط کاربری به‌طور خودکار بر اساس تغییرات داده‌های برنامه بازسازی می‌شود. مدیریت وضعیت برنامه در برنامه‌های مدرن فرانت‌اند ذاتاً پیچیده است و رویکرد "قدیمی‌مدرن" فلت می‌تواند برای برنامه‌نویسانی که تجربه‌ای در فرانت‌اند ندارند، جذاب‌تر باشد.

ویژگی visible:

هر کنترل دارای ویژگی `visible` است که به‌طور پیش‌فرض `true` است؛ یعنی کنترل بر روی صفحه نمایش داده می‌شود. تنظیم `visible` به `false` به‌طور کامل مانع از نمایش کنترل (و تمام فرزندان آن، در صورت وجود) بر روی بوم صفحه می‌شود. کنترل‌های پنهان نمی‌توانند با صفحه‌کلید یا ماوس فوکوس یا انتخاب شوند و رویدادهایی را منتشر نمی‌کنند.

ویژگی disabled:

هر کنترل دارای ویژگی `disabled` است که به‌طور پیش‌فرض `false` است؛ یعنی کنترل و تمام فرزندان آن فعال هستند. ویژگی `disabled` بیشتر برای کنترل‌های ورود داده مانند `TextField`، `Dropdown`، `Checkbox` و دکمه‌ها استفاده می‌شود. با این حال، ویژگی `disabled` می‌تواند به کنترل والدین نیز تنظیم شود و مقدار آن به‌طور بازگشتی به تمام فرزندان آن منتقل می‌شود.

برای مثال، اگر یک فرم با چندین کنترل ورودی دارید، می‌توانید ویژگی `disabled` را به‌طور جداگانه برای هر کنترل تنظیم کنید:

```
  first_name = ft.TextField()
    last_name = ft.TextField()
    first_name.disabled = True
    last_name.disabled = True
    page.add(first_name, last_name)
```

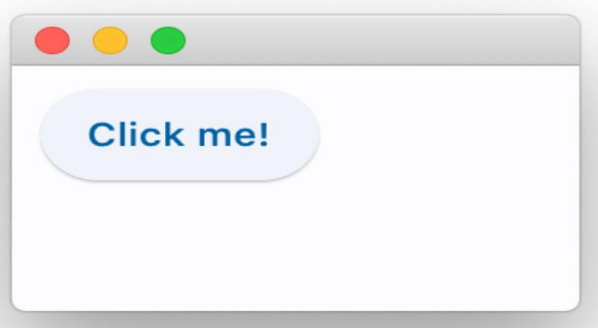
یا می‌توانید کنترل‌های فرم را در یک محفظه، مانند `Column`، قرار دهید و سپس ویژگی `disabled` را برای ستون تنظیم کنید:

```
    first_name = ft.TextField()
    last_name = ft.TextField()
    c = ft.Column(controls=[
        first_name,
        last_name
    ])
    c.disabled = True
    page.add(c)
```

دکمه ها(Buttons):

`Button` یکی از مهم‌ترین کنترل‌های ورودی است که هنگام فشار دادن، رویداد کلیک (click event) تولید می‌کند:

```
btn = ft.ElevatedButton("Click me!")
page.add(btn)
```



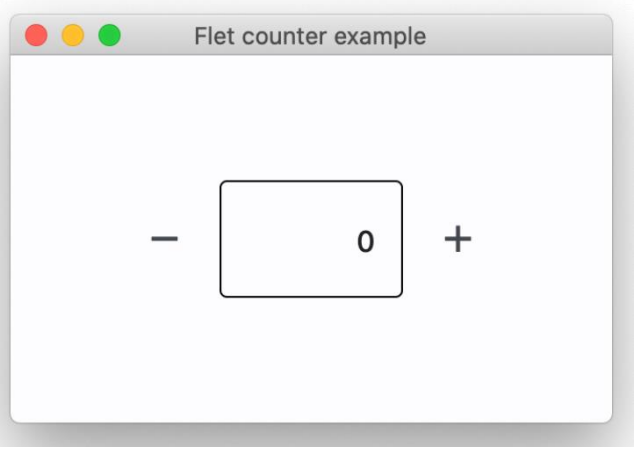
تمام رویدادهایی که توسط کنترل‌ها روی یک صفحه وب ایجاد می‌شوند، به طور مداوم به اسکریپت شما ارسال می‌شوند. اما چگونه می‌توانید به کلیک یک دکمه پاسخ دهید؟

هشدارهای رویداد (Event handler)

در مثال زیر، برنامه "Counter" با دکمه‌هایی که رویداد تولید می‌کنند، نشان داده شده است:

```
import flet as ft

def main(page: ft.Page):
    page.title = "Flet counter example"
    page.vertical_alignment = ft.MainAxisAlignment.CENTER

    txt_number = ft.TextField(value="0", text_align="right", width=100)

    def minus_click(e):
        txt_number.value = str(int(txt_number.value) - 1)
        page.update()

    def plus_click(e):
        txt_number.value = str(int(txt_number.value) + 1)
        page.update()

    page.add(
        ft.Row(
            [
                ft.IconButton(ft.icons.REMOVE, on_click=minus_click),
                txt_number,
                ft.IconButton(ft.icons.ADD, on_click=plus_click),
            ],
```

```
            alignment=ft.MainAxisAlignment.CENTER,
        )
    )

ft.app(target=main)
```



در این مثال:

- یک `TextField` برای نمایش مقدار فعلی شمارنده ایجاد شده است.

- دو دکمه (`IconButton`) یکی برای کاهش (`minus_click`) و دیگری برای افزایش (`plus_click`) مقدار شمارنده ایجاد شده‌اند.

- هر دکمه یک هندلر رویداد (`on_click`) دارد که با کلیک کاربر، مقدار `TextField` را تغییر می‌دهد و سپس صفحه را با `page.update()` به‌روزرسانی می‌کند.

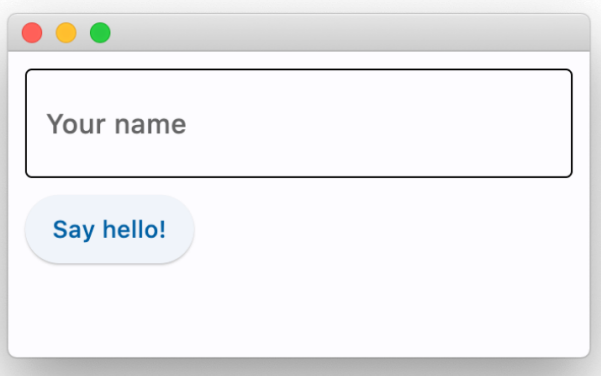
با این کار، وقتی کاربر روی یکی از دکمه‌ها کلیک می‌کند، مقدار شمارنده تغییر کرده و بلافاصله در رابط کاربری نمایش داده می‌شود.

کادرمتن (Textbox)

فلت مجموعه‌ای از کنترل‌ها را برای ساخت فرم‌ها ارائه می‌دهد که شامل "TextField"، "Checkbox"، "Dropdown" و "ElevatedButton" است.

در مثال زیر، از کاربر نامی درخواست می‌شود:

```
import flet as ft

def main(page):
    def btn_click(e):
        if not txt_name.value:
            txt_name.error_text = "Please enter your name"
            page.update()
        else:
            name = txt_name.value
            page.clean()
            page.add(ft.Text(f"Hello, {name}!"))

    txt_name = ft.TextField(label="Your name")

    page.add(txt_name, ft.ElevatedButton("Say hello!", on_click=btn_click))

ft.app(target=main)
```



در این مثال:

- یک `TextField` با برچسب "Your name" برای ورود نام کاربر ایجاد شده است.

- یک دکمه `ElevatedButton` با متن "Say hello!" برای ارسال ورودی کاربر اضافه شده است.

- اگر کاربر نامی وارد نکرده باشد و روی دکمه کلیک کند، پیام خطا "Please enter your name" نمایش داده می‌شود.

- اگر نامی وارد شده باشد، صفحه پاک شده و پیامی شامل نام کاربر، مانند "Hello, [Name]!" نمایش داده می‌شود.

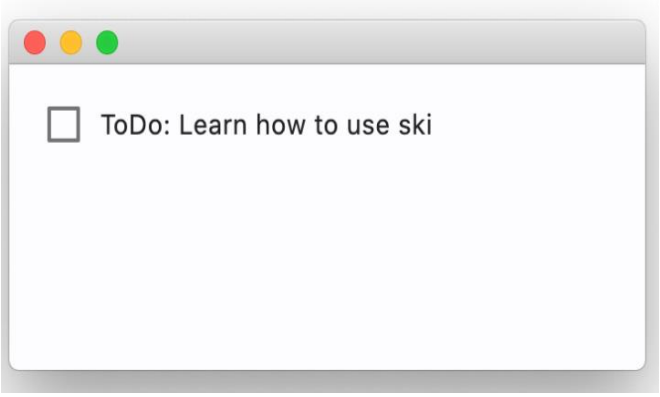
این مثال نشان می‌دهد که چگونه می‌توان از کادر متن و دکمه‌ها برای ایجاد یک فرم ساده در فلت استفاده کرد و با ورودی‌های کاربر تعامل کرد.

چک باکس (Checkbox)

کنترل `Checkbox` در فلت ویژگی‌ها و رویدادهای متنوعی را برای استفاده آسان فراهم می‌کند.

در مثال زیر، یک چک‌باکس برای انجام یک کار ساده ایجاد می‌شود:

```
import flet as ft

def main(page):
    def checkbox_changed(e):
        output_text.value = (
            f"You have learned how to ski: {todo_check.value}."
        )
        page.update()

    output_text = ft.Text()
    todo_check = ft.Checkbox(label="ToDo: Learn how to ski", value=False,
on_change=checkbox_changed)
    page.add(todo_check, output_text)

ft.app(target=main)
```



در این مثال:

- یک چک‌باکس با برچسب "ToDo: Learn how to ski" و مقدار اولیه `False` (یعنی تیک‌نخورده) ایجاد شده است.

- یک متن خروجی (`output_text`) برای نمایش نتیجه انتخاب چک‌باکس اضافه شده است.

- وقتی کاربر چک‌باکس را تغییر می‌دهد (تیک بزند یا تیک را بردارد)، رویداد `on_change` فراخوانی شده و تابع `checkbox_changed` اجرا می‌شود.

- در تابع `checkbox_changed`، مقدار چک‌باکس (یعنی `True` یا `False`) به روز می‌شود و این مقدار در متن خروجی نمایش داده می‌شود.

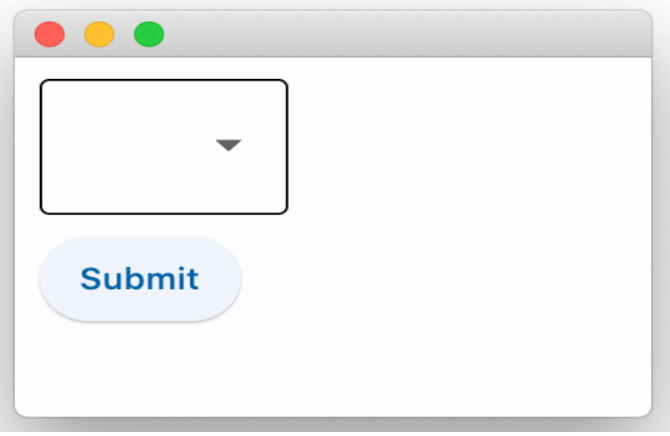
این کد به شما نشان می‌دهد که چگونه می‌توانید از چک‌باکس برای ایجاد یک لیست کارهای انجام‌دادنی ساده استفاده کنید و وضعیت آن را پیگیری کنید.

منوی کشویی (Dropdown)

در فلت، کنترل "Dropdown" به شما اجازه می‌دهد تا لیستی از گزینه‌ها را برای انتخاب کاربر ایجاد کنید.

در مثال زیر، یک منوی کشویی ساده ایجاد شده که کاربر می‌تواند از بین رنگ‌ها یکی را انتخاب کند و سپس با کلیک بر روی دکمه، انتخاب خود را ارسال کند:

```
import flet as ft

def main(page):
    def checkbox_changed(e):
        output_text.value = (
            f"You have learned how to ski: {todo_check.value}."
        )
        page.update()

    output_text = ft.Text()
    todo_check = ft.Checkbox(label="ToDo: Learn how to ski", value=False,
on_change=checkbox_changed)
    page.add(todo_check, output_text)

ft.app(target=main)
```



در این مثال:

- یک منوی کشویی (`Dropdown`) با سه گزینه: ""Red"، "Green"" و "Blue" ایجاد شده است.

- یک دکمه `ElevatedButton` با متن "Submit" وجود دارد که وقتی کاربر بر روی آن کلیک می‌کند، رویداد `on_click` فراخوانی می‌شود.

- تابع `button_clicked` مقدار انتخاب‌شده در منوی کشویی را در متن خروجی (`output_text`) نمایش می‌دهد.

این کد به شما نشان می‌دهد که چگونه می‌توانید از منوی کشویی برای دریافت ورودی از کاربر و نمایش آن استفاده کنید.