



Flet-Python

A cross-platform python framework

مستندات فارسی فریمورک Flet ارائه 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):

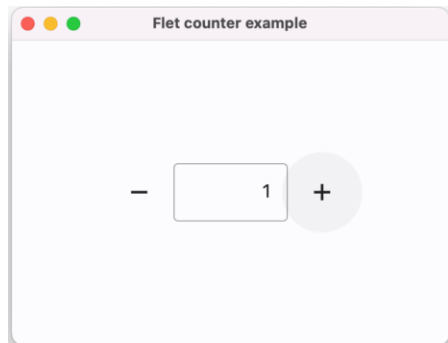
```
pip install flet
```

سپس برنامه را اجرا کنید:

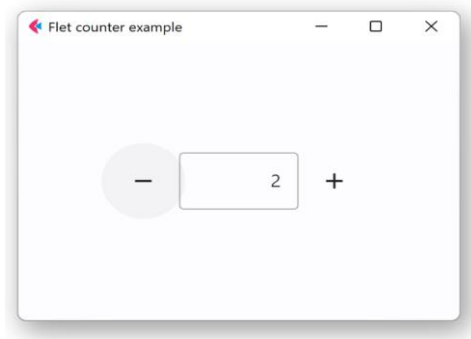
```
flet run counter.py
```

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

macOS



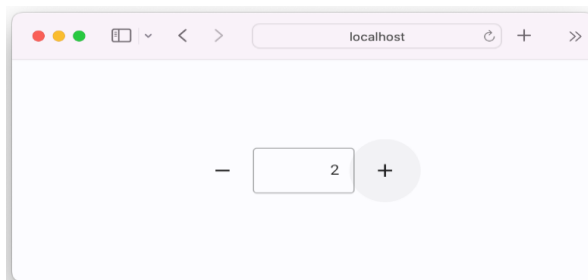
Windows



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

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

یک پنجره یا تب جدید مرورگر باز می شود:



بیا یاد شروع کنیم:

پیش از اینکه بتوانید اولین برنامه فلت خود را ایجاد کنید، باید محیط توسعه خود را راه اندازی 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:

macOS/Linux

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

Windows

```
md first-flet-app
cd first-flet-app
python -m venv .venv
.venv\Scripts\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 (دپندنسی) 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:

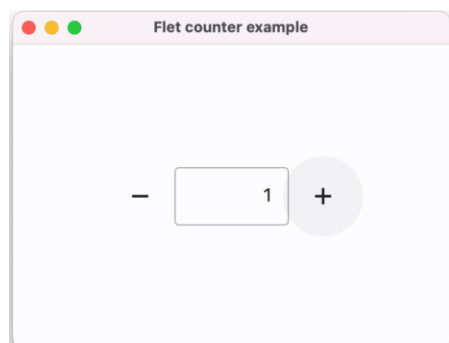
```
flet run /Users/JohnSmith/Documents/projects/flet-app
```

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

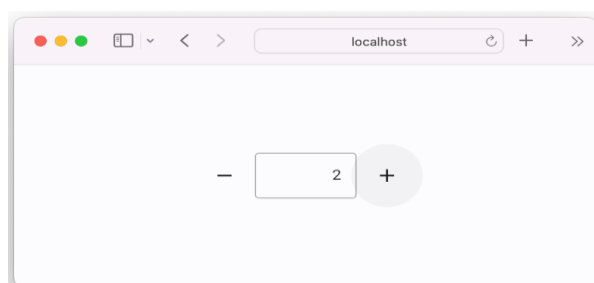


اجرای یک برنامه تحت وب:

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 پیش فرض، فلت فایل اسکریپتی که اجرا شده است را مشاهده می کند و هر زمان که فایل تغییر و ذخیره شود، برنامه را مجدداً بارگذاری می کند. اما فلت تغییرات در سایر فایل ها را مشاهده نمی کند.

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

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

برای مشاهده دایرکتوری اسکریپت و تمام زیردایرکتوری ها 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.

کنترل‌های 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")
```

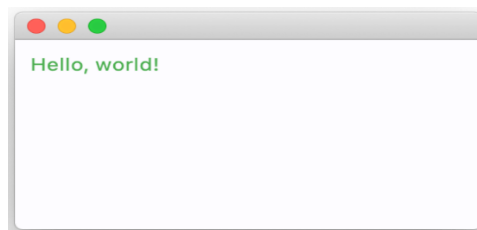
برای نمایش کنترل در یک صفحه، آن را به لیست کنترل‌ها `controls` صفحه اضافه کرده و سپس با استفاده از `page.update()` تغییرات صفحه را به مرورگر یا کلاینت دسکتاپ ارسال کنید:

```
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

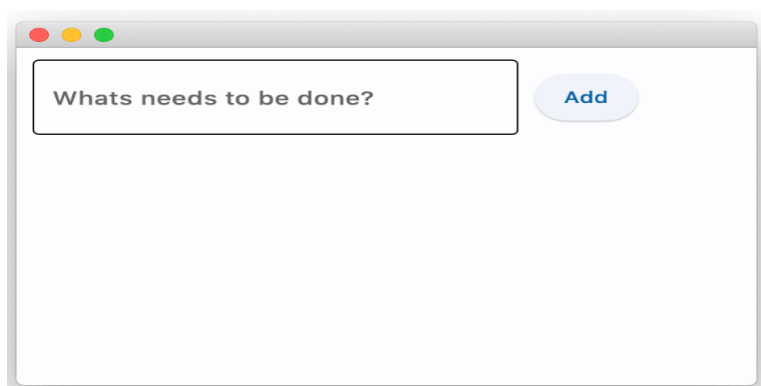
def main(page):
    def add_clicked(e):
        page.add(ft.Checkbox(label=new_task.value))
        new_task.value = ""
        new_task.focus()
        new_task.update()

    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 به طور پیش فرض 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 به طور بازگشتی به تمام فرزندان آن منتقل می‌شود.

برای مثال، اگر یک فرم با چندین کنترل ورودی دارید، می‌توانید ویژگی disabled را به طور 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)
```

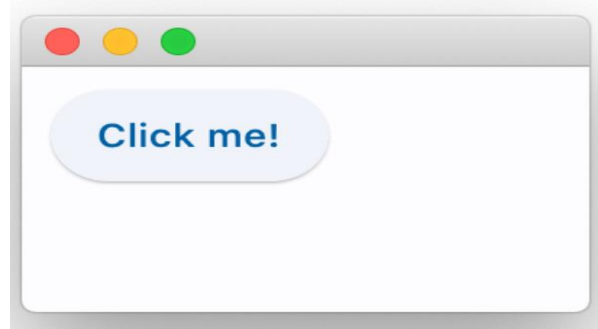
یا می‌توانید کنترل‌های فرم را در یک محفظه، مانند Column، قرار دهید و سپس ویژگی disabled را برای ستون تنظیم کنید:

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

دکمه‌ها (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()

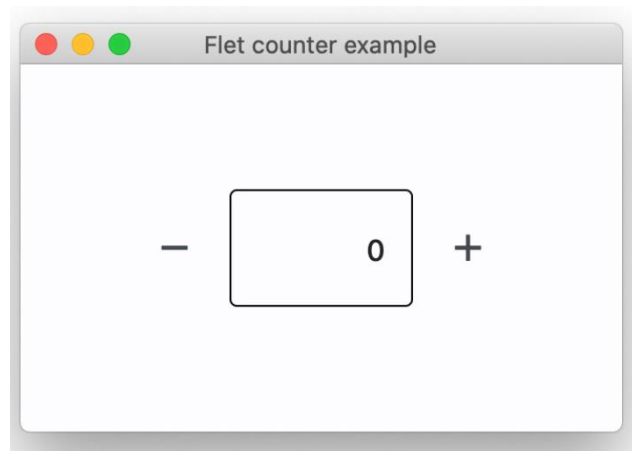
    def plus_click(e):
        txt_number.value = str(int(txt_number.value) + 1)
        page.update()

    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)

فالت 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:

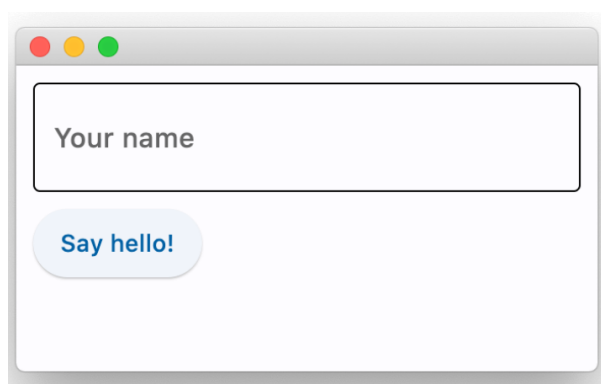
```
import flet as ft

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.

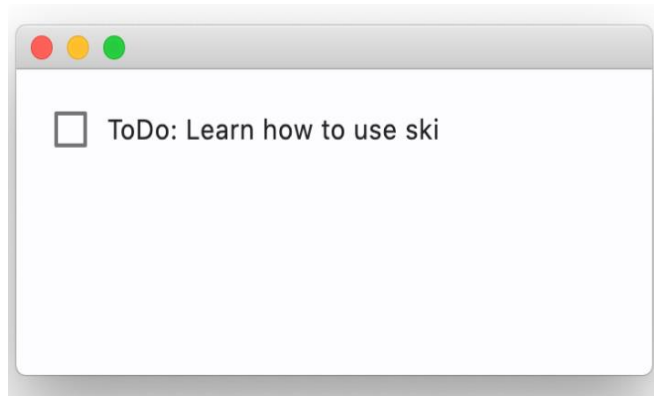
در مثال زیر، یک چک‌باکس برای انجام یک کار ساده ایجاد می‌شود:

```
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 تیک نخورده) 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 انجام‌دانی 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.

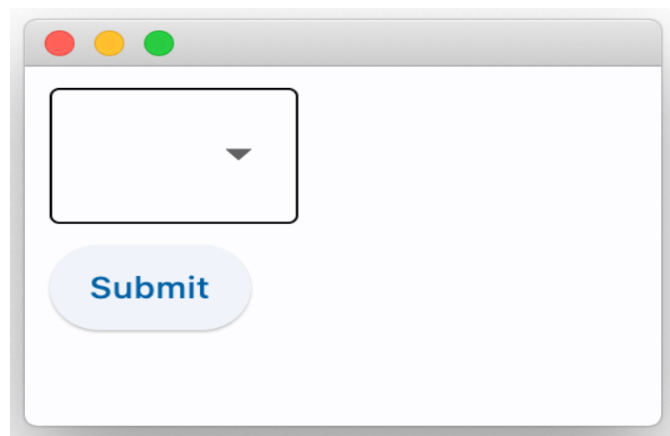
در مثال زیر، یک منوی کشویی ساده ایجاد شده که کاربر می‌تواند از بین رنگ‌ها یکی را انتخاب کند و سپس با کلیک بر روی دکمه، انتخاب خود را ارسال کند:

```
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: "Blue"، "Green"، و "Red" 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 انتخاب شده 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.

در فریمورک فلت، علاوه بر کنترل‌های از پیش ساخته شده‌ای که بیش از ۱۰۰ عدد هستند و می‌توانند به تنهایی استفاده شوند، زیبایی واقعی برنامه‌نویسی با فلت در این است که تمام این کنترل‌ها را می‌توان برای ایجاد کامپوننت‌های UI قابل استفاده مجدد با استفاده از مفاهیم برنامه‌نویسی شی‌گرا در پایتون به کار برد.

کنترل‌های سفارشی

کنترل‌های سفارشی به شما این امکان را می‌دهند که کنترل‌های خود را با استفاده از کنترل‌های موجود در فلت و همچنین با سبک‌دهی و یا ترکیب آن‌ها ایجاد کنید.

کنترل‌های استایل‌دار

ساده‌ترین نوع کنترل سفارشی که می‌توانید ایجاد کنید، یک کنترل استایل‌دار است، به عنوان مثال، یک دکمه با رنگ و رفتار خاص که در طول برنامه شما چندین بار استفاده می‌شود.

برای ایجاد یک کنترل استایل‌دار، شما باید یک کلاس جدید در پایتون ایجاد کنید که از کنترل فلت مورد نظر شما ارث‌بری کند، در اینجا `ElevatedButton`:

```
class MyButton(ft.ElevatedButton):
    def __init__(self, text):
        super().__init__()
        self.bgcolor = ft.colors.ORANGE_300
        self.color = ft.colors.GREEN_800
        self.text = text
```

توضیح کد:

- تعریف کلاس `MyButton`: این کلاس از `ElevatedButton` فلت ارث‌بری می‌کند. `ElevatedButton` یک دکمه با طراحی برجسته است که از قبل در فلت وجود دارد.

- متد `__init__`: این متد سازنده کلاس است. در اینجا، شما می‌توانید خصوصیات و رفتارهای جدیدی به کنترل اضافه کنید.

- `super().__init__()`: این خط از متد سازنده کلاس والد (`ElevatedButton`) فراخوانی می‌شود تا ویژگی‌ها و متدهای آن کلاس در دسترس قرار گیرد.

- `self.bgcolor`: رنگ پس‌زمینه دکمه را به رنگ نارنجی (`ORANGE_300`) تغییر می‌دهد.

- `self.color`: رنگ متن دکمه را به رنگ سبز (`GREEN_800`) تغییر می‌دهد.

- `self.text`: متن دکمه را تنظیم می‌کند بر اساس پارامتر ورودی `text`.

حالا می‌توانید از این کنترل جدید در برنامه خود استفاده کنید:

```
import flet as ft

def main(page: ft.Page):

    page.add(MyButton(text="OK"), MyButton(text="Cancel"))

ft.app(target=main)
```

OK

Cancel

توضیح کد:

- تعریف تابع `main`: تابع اصلی برنامه است که در آن صفحه (`page``) را دریافت می‌کند و کنترل‌های جدید را به آن اضافه می‌کند.

- `page.add`: این متد کنترل‌های جدید (`MyButton``) را به صفحه اضافه می‌کند.

- اجرا کردن برنامه: `ft.app(target=main)` برنامه را اجرا می‌کند و تابع `main` را به عنوان نقطه شروع تعریف می‌کند.

این کد یک اپلیکیشن ساده ایجاد می‌کند که شامل دو دکمه با متن "OK" و "Cancel" با استایل‌های خاصی است که شما تعیین کرده‌اید.

در بخش استفاده از کنترل‌های استایل‌دار در اپلیکیشن ماشین حساب به شما نشان داده می‌شود که چگونه می‌توانید از کنترل‌های استایل‌دار در اپلیکیشن‌های خود استفاده کنید.

مدیریت رویدادها

مشابه با ویژگی‌ها، می‌توانید هندلرهای رویداد را به عنوان پارامترها به سازنده کلاس کنترل سفارشی خود پاس دهید:

```
import flet as ft

class MyButton(ft.ElevatedButton):
    def __init__(self, text, on_click):
        super().__init__()
        self.bgcolor = ft.colors.ORANGE_300
        self.color = ft.colors.GREEN_800
        self.text = text
        self.on_click = on_click

def main(page: ft.Page):

    def ok_clicked(e):
        print("OK clicked")
```

```
def cancel_clicked(e):
    print("Cancel clicked")

page.add(
    MyButton(text="OK", on_click=ok_clicked),
    MyButton(text="Cancel", on_click=cancel_clicked),
)

ft.app(target=main)
```

توضیح کد:

- کلاس `MyButton`:

- سازنده `__init__`: علاوه بر ویژگی‌های بصری، یک پارامتر جدید `on_click` برای هندلر رویداد کلیک دریافت می‌کند.

- ویژگی `self.on_click`: این ویژگی را به سازنده ارسال شده اختصاص می‌دهد.

- توابع `ok_clicked` و `cancel_clicked(e)`:

- `ok_clicked(e)`: تابعی است که وقتی دکمه "OK" کلیک می‌شود، پیغام "OK clicked" را چاپ می‌کند.

- `cancel_clicked(e)`: تابعی است که وقتی دکمه "Cancel" کلیک می‌شود، پیغام "Cancel clicked" را چاپ می‌کند.

- تابع `main`:

- `page.add`: این متد دو دکمه جدید (`MyButton`) را به صفحه اضافه می‌کند و برای هر کدام هندلر رویداد مربوط به کلیک را تنظیم می‌کند.

این کد یک اپلیکیشن ساده ایجاد می‌کند که شامل دو دکمه با متن‌های "OK" و "Cancel" است و وقتی بر روی آن‌ها کلیک می‌شود، پیغام‌های مشخص شده در کنسول چاپ می‌شود.

کنترل‌های ترکیبی

کنترل‌های ترکیبی از کنترل‌های Container مانند Column ، Row ، Stack یا حتی View ارث‌بری می‌کنند تا کنترل‌های مختلف فلت را ترکیب کنند. مثال زیر یک کنترل Task است که می‌تواند در یک اپلیکیشن To-Do استفاده شود:

```
import flet as ft
class Task(ft.Row):
    def __init__(self, text):
        super().__init__()
        self.text_view = ft.Text(text)
        self.text_edit = ft.TextField(text, visible=False)
        self.edit_button = ft.IconButton(icon=ft.icons.EDIT, on_click=self.edit)
        self.save_button = ft.IconButton(
            visible=False, icon=ft.icons.SAVE, on_click=self.save
        )
        self.controls = [
            ft.Checkbox(),
            self.text_view,
            self.text_edit,
            self.edit_button,
            self.save_button,
        ]

    def edit(self, e):
        self.edit_button.visible = False
        self.save_button.visible = True
        self.text_view.visible = False
        self.text_edit.visible = True
        self.update()

    def save(self, e):
        self.edit_button.visible = True
        self.save_button.visible = False
        self.text_view.visible = True
        self.text_edit.visible = False
        self.text_view.value = self.text_edit.value
        self.update()

def main(page: ft.Page):
    page.add(
        Task(text="Do laundry"),
```

```
Task(text="Cook dinner"),
)

ft.app(target=main)
```

دیدن خروجی

توضیح کد:

- کلاس `Task`:

- سازنده `__init__`:

- `self.text_view`: نمای متنی که وظیفه نمایش متن را دارد.

- `self.text_edit`: فیلد متنی برای ویرایش که به صورت پیش فرض پنهان است.

- `self.edit_button`: دکمه ویرایش که هنگام کلیک کردن تابع `'edit'` را فراخوانی می کند.

- `self.save_button`: دکمه ذخیره که هنگام کلیک کردن تابع `'save'` را فراخوانی می کند و به صورت

پیش فرض پنهان است.

- `self.controls`: لیست کنترل هایی که در `'Row'` قرار می گیرند.

- متدهای `edit` و `save`:

- `edit(self, e)`: این تابع وضعیت دکمه های ویرایش و ذخیره و همچنین نمای متنی و فیلد ویرایش را

تغییر می دهد.

- `save(self, e)`: این تابع وضعیت دکمه های ویرایش و ذخیره را به حالت اولیه باز می گرداند و مقدار فیلد

ویرایش را به نمای متنی انتقال می دهد.

- تابع `main`:

- `page.add`: این متد دو وظیفه (`Task`) را به صفحه اضافه می کند.

این کد یک اپلیکیشن ساده To-Do با وظایف قابل ویرایش و ذخیره ایجاد می‌کند که شامل دکمه‌های ویرایش و ذخیره و همچنین چک‌باکس و متن است.

برای دیدن مثال‌های بیشتر به ریپازیتوری‌های [flet-contrib](#) and [community examples](#) مراجعه کنید.

متدهای چرخه‌حیات (Life-cycle)

در فریمورک فلت، کنترل‌های سفارشی متدهای "hook" برای چرخه‌زندگی ارائه می‌دهند که ممکن است برای سناریوهای مختلف در اپلیکیشن شما مورد نیاز باشند. این متدها به شما امکان می‌دهند تا منطق خاصی را در مراحل مختلف عمر کنترل پیاده‌سازی کنید.

build()

- توضیحات: این متد زمانی که کنترل در حال ایجاد و تخصیص به `self.page` است، فراخوانی می‌شود.

- زمان استفاده: اگر نیاز به اجرای منطق خاصی دارید که نمی‌تواند در سازنده کنترل انجام شود و به `self.page` نیاز دارد، این متد را `Override` کنید. برای مثال، انتخاب آیکون مناسب بسته به `self.page.platform` برای [اپلیکیشن‌های سازگار](#).

- مثال:

```
def build(self):
    if self.page.platform == "web":
        self.icon = ft.icons.WEB
    else:
        self.icon = ft.icons.MOBILE
```

did_mount()

- توضیحات: این متد پس از افزودن کنترل به صفحه و اختصاص `uid` موقتی فراخوانی می‌شود.

- زمان استفاده: اگر نیاز به اجرای منطق خاصی دارید که باید پس از اضافه شدن کنترل به صفحه انجام شود، این متد را `Override` کنید. برای مثال، یک [ویجت آب‌وهوا](#) که هر دقیقه با `API Open Weather` برای به‌روزرسانی وضعیت آب‌وهوا تماس می‌گیرد.

- مثال:

```
def did_mount(self):
    self.start_weather_update()

def start_weather_update(self):
    # شروع به روزرسانی وضعیت آب و هوا
    pass
```

will_unmount()

- توضیحات: این متد قبل از حذف کنترل از صفحه فراخوانی می‌شود.

- زمان استفاده: برای اجرای کدهای پاک‌سازی قبل از حذف کنترل از صفحه، این متد را Override کنید.

- مثال:

```
def will_unmount(self):
    self.stop_weather_update() # متوقف کردن به روزرسانی وضعیت آب و هوا
```

before_update()

- توضیحات: این متد هر بار که کنترل در حال به‌روزرسانی است، فراخوانی می‌شود.

- زمان استفاده: مطمئن شوید که درون این متد به متد `update()` فراخوانی نکنید.

- مثال:

```
def before_update(self):
    # اجرای منطق قبل از به‌روزرسانی
    pass
```

کنترل‌های منزوی شده (Isolated controls)

کنترل‌های سفارشی دارای ویژگی `is_isolated` هستند که به طور پیش فرض برابر با `False` است.

- توضیحات: اگر `is_isolated` را به `True` تنظیم کنید، کنترل شما از چیدمان بیرونی ایزوله خواهد شد. به این معنی که وقتی متد `update()` برای کنترل والد فراخوانی می‌شود، کنترل‌های فرزند تغییرات آن‌ها در دلایل به‌روزرسانی لحاظ نمی‌شود. کنترل‌های ایزوله باید متد `self.update()` را برای ارسال تغییرات خود به صفحه فلت فراخوانی کنند.

- عملکرد: به عنوان بهترین عمل، هر کنترل سفارشی که درون متدهای کلاس خود `self.update()` را فراخوانی می‌کند، باید ایزوله باشد.

مثال:

```
class Task(ft.Row):
    def __init__(self, text):
        super().__init__()
        self.isolated = True
```

توضیحات:

- `self.isolated = True`: این تنظیم باعث می‌شود که تغییرات درون این کنترل به طور مستقل از تغییرات کنترل‌های والدین پردازش شوند و کنترل باید خود به‌روزرسانی‌هایش را مدیریت کند.

با استفاده از این متدهای چرخه‌زندگی و ویژگی‌های کنترل‌های ایزوله‌شده، می‌توانید کنترل‌های سفارشی قدرتمند و انعطاف‌پذیر ایجاد کنید که به طور خاص با نیازهای اپلیکیشن شما سازگار باشند.

اپلیکیشن‌های سازگار

فریمورک فلت به شما این امکان را می‌دهد که اپلیکیشن‌های سازگاری ایجاد کنید که با یک کدبیس، ظاهری متفاوت در دستگاه‌های مختلف، بسته به پلتفرم آن‌ها (iOS یا Android)، ارائه دهند.

در ادامه یک مثال ساده از یک اپلیکیشن که در پلتفرم‌های iOS و Android ظاهر متفاوتی دارد، آورده شده است:

```

import flet as ft

def main(page):

    page.adaptive = True

    page.appbar = ft.AppBar(
        leading=ft.TextButton("New", style=ft.ButtonStyle(padding=0)),
        title=ft.Text("Adaptive AppBar"),
        actions=[
            ft.IconButton(ft.cupertino_icons.ADD, style=ft.ButtonStyle(padding=0))
        ],
        bgcolor=ft.colors.with_opacity(0.04, ft.cupertino_colors.SYSTEM_BACKGROUND),
    )

    page.navigation_bar = ft.NavigationBar(
        destinations=[
            ft.NavigationBarDestination(icon=ft.icons.EXPLORE, label="Explore"),
            ft.NavigationBarDestination(icon=ft.icons.COMMUTE, label="Commute"),
            ft.NavigationBarDestination(
                icon=ft.icons.BOOKMARK_BORDER,
                selected_icon=ft.icons.BOOKMARK,
                label="Bookmark",
            ),
        ],
        border=ft.Border(
            top=ft.BorderSide(color=ft.cupertino_colors.SYSTEM_GREY2, width=0)
        ),
    )

    page.add(
        ft.SafeArea(
            ft.Column(
                [
                    ft.Checkbox(value=False, label="Dark Mode"),
                    ft.Text("First field:"),
                    ft.TextField(keyboard_type=ft.KeyboardType.TEXT),
                    ft.Text("Second field:"),
                    ft.TextField(keyboard_type=ft.KeyboardType.TEXT),
                    ft.Switch(label="A switch"),
                    ft.FilledButton(content=ft.Text("Adaptive button")),
                    ft.Text("Text line 1"),
                    ft.Text("Text line 2"),
                ]
            )
        )
    )

```

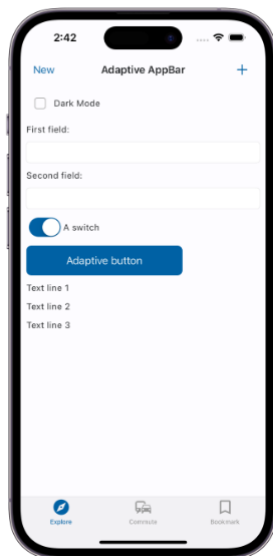
```

        ft.Text("Text line 3"),
      ],
    ),
  ),
)

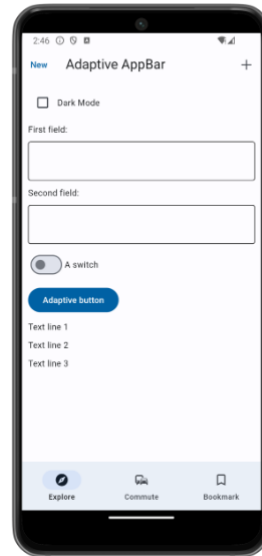
ft.app(target=main)

```

iPhone



Android



توضیحات کد:

فعال کردن حالت سازگار:

با تنظیم `page.adaptive = True`، اپلیکیشن شما به صورت خودکار با توجه به پلتفرم دستگاه (iOS یا Android) ظاهر خود را تنظیم می‌کند.

نوار ابزار (AppBar):

در این بخش، نوار ابزار تعریف شده است که در آن دکمه‌ها و متن‌ها با توجه به پلتفرم تنظیم می‌شوند. از آیکون‌های خاص پلتفرم‌ها استفاده شده است، به عنوان مثال، آیکون `ft.cupertino_icons.ADD` برای iOS.

نوار ناوبری (NavigationBar):

نوار ناوبری با آیکون‌ها و مقاصدی مانند "Commuter", "Explore" و "Bookmark" تعریف شده است. این نوار نیز با توجه به پلتفرم دستگاه تنظیم شده است.

کنترل‌های Cupertino و Material

بیشتر کنترل‌های موجود در فریمورک فلت (Flet) بر اساس [طراحی Material](#) ساخته شده‌اند. اما علاوه بر آن‌ها، فلت تعدادی کنترل نیز ارائه می‌دهد که به سبک iOS طراحی شده‌اند و به عنوان کنترل‌های Cupertino شناخته می‌شوند.

کنترل‌های Adaptive

کنترل‌های Cupertino معمولاً معادل کنترل‌های Material هستند که دارای ویژگی‌ای به نام [adaptive](#) می‌باشند. ویژگی adaptive به صورت پیش‌فرض مقدار False دارد. با استفاده از کنترل‌های Material که ویژگی adaptive آن‌ها به True تنظیم شده باشد، کنترل متفاوتی بسته به پلتفرم مورد استفاده ایجاد می‌شود. به عنوان مثال:

```
ft.Checkbox(adaptive=True, value=True, label="Adaptive Checkbox")
```

فلت مقدار ویژگی [page.platform](#) را بررسی می‌کند. اگر مقدار آن `ft.PagePlatform.IOS` یا `ft.PagePlatform.MACOS` باشد، یک کنترل Cupertino ایجاد می‌شود. در غیر این صورت، کنترل Material ساخته خواهد شد.

نکات مهم

- تنظیم ویژگی `adaptive`: این ویژگی می‌تواند برای یک کنترل فردی یا یک کنترل کانتینر مانند `Row`، `Column` یا هر کنترل دیگری که دارای ویژگی‌های `content` یا `controls` است تنظیم شود. اگر کنترل کانتینر `adaptive` باشد، تمام کنترل‌های فرزند آن نیز `adaptive` خواهند بود، مگر اینکه ویژگی `adaptive` به طور خاص برای یک کنترل فرزند به ``False`` تنظیم شده باشد.

لیست کنترل‌های Adaptive

کنترل‌های Adaptive در فلت شامل موارد زیر هستند که معادل‌های Cupertino خود را دارند:

- ``Checkbox: `ft.Checkbox(adaptive=True)`

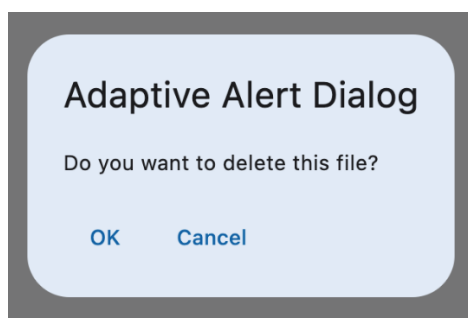
- ``Switch: `ft.Switch(adaptive=True)`

- ``TextField: `ft.TextField(adaptive=True)`

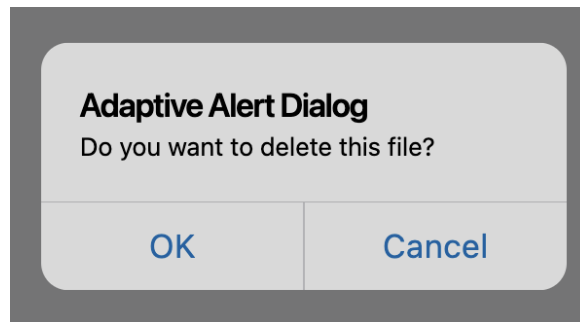
- ``Button: `ft.FilledButton(adaptive=True)`

با استفاده از این کنترل‌ها، می‌توانید برنامه‌هایی بسازید که به صورت خودکار با پلتفرم‌های مختلف سازگار شوند و تجربه کاربری مطلوبی را در دستگاه‌های iOS و Android فراهم آورند

[AlertDialog](#)



[CupertinoAlertDialog](#)



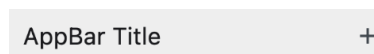
[Any button in Dialog actions](#)



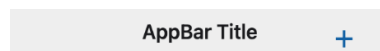
[CupertinoDialogAction](#)



[AppBar](#)



[CupertinoAppBar](#)



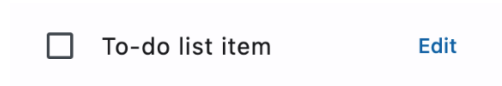
[NavigationBar](#)



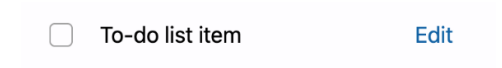
[CupertinoNavigationBar](#)



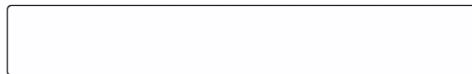
[ListTile](#)



[CupertinoListTile](#)



[TextField](#)



[CupertinoTextField](#)



[Checkbox](#)

[CupertinoCheckbox](#)

[Slider](#)



[CupertinoSlider](#)



[Switch](#)



[CupertinoSwitch](#)



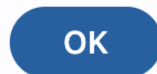
[Radio](#)



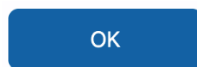
[CupertinoRadio](#)



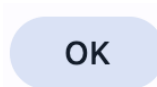
[FilledButton](#)



[CupertinoFilledButton](#)



[FilledTonalButton](#)



CupertinoButton



IconButton



CupertinoButton



ElevatedButton



CupertinoButton



OutlinedButton



TextButton



