BeeWare Documentation

發🄵 *0.1.dev157+g194f920*

Russell Keith-Magee

2024 年 05 月 15 日

Contents

在任何地方執行你寫的 Python。

歡迎來到 BeeWare！在本教學中，我們將使用 Python 建立一個 GUI，🄵將其部署🄵桌面應用程式、行動應用程式和網頁。我們還將了解如何使用 BeeWare 工具來執行作🄵應用程式開發人員所需執行的一些常見任務，例如測試應用程式。

這是機器翻譯！

此版本的教學是透過機器翻譯🄵生的。我們知道這🄵不理想，但我們認🄵糟糕的翻譯總比🄵有翻譯好。

如果您想要協助改進翻譯，請聯絡我們！我們在 Discord 中有一個 `#translations` 頻道。在那🄵進行自我介紹，我們會將您加入翻譯團隊。

CHAPTER 1

什🄵是 BeeWare？

BeeWare 不是一個單一的🄴品、工具或函式庫 - 它是工具和函式庫的集合，每個工具和函式庫協同工作，幫助您編寫跨平台的原生 Python GUI 應用程式。這包括：

- Toga，跨平台小工具包；
- Briefcase，一種將 Python 專案打包🄵可分發形式 (apk、exe 等) 🄵可傳送給最終使用者的工具；
- 用於存取各個平臺的原生庫的庫（例如 Rubicon ObjC）；
- 預編譯版本的 Python 可在無官方 Python 安裝程式的平台上使用。

在本教程中，我們將使用所有這些工具，但作🄴用🄵，您只需要與前兩個工具（Toga 和 Briefcase）進行互動。每個工具也可以單獨使用 - 舉例來🄵，您可以使用 Briefcase 來部署應用程序，而不使用 Toga 作🄵 GUI 工具包。

BeeWare 套件可在 macOS、Windows、Linux（使用 GTK）上使用，也在 Android 和 iOS 等行動平台上，甚至是網頁。

CHAPTER 2

讓我們開始吧！

准備好親自嘗試 BeeWare 了嗎？讓我們用 *Python* 建立一個跨平台應用程式！

2.1 教學 0 - 讓我們開始設定吧！

在建立第一個 BeeWare 應用程式之前，我們必須確保具備運行 BeeWare 的所有先[F]條件。

2.1.1 安裝 Python

我們首先需要的是一個可用的 Python 直譯器。

macOS

Linux

Windows

如果您使用的是 macOS，則 Xcode 或命令列開發人員工具中包含最新版本的 Python。要檢查您是否已經擁有它，請執行以下命令：

```
$ python3 --version
```

如果安裝了 Python，您將看到其版本號。否則，系統將提示您安裝命令列開發人員工具。

如果您使用的是 Windows，則可以從 Python 網站 取得官方安裝程式。您可以使用 3.8 及以上的任何穩定版本的 Python。我們建議避免 alpha、beta 和候選版本，除非您 **真的**知道自己在做什[F]。

如果您使用的是 Linux，則會使用系統套件管理器（Debian/Ubuntu/Mint 上的 `apt` 、Fedora 上的 `dnf` 或 Arch 上的 `pacman` ）安裝 Python。

您應該確保系統 Python 是 Python 3.8 或更高版本；如果不是（例如，Ubuntu 18.04 附帶 Python 3.6），則需要將 Linux 發行版升級到更新的版本。

目前對 Raspberry Pi 的支援有限。

如果您使用的是 Windows，則可以從 Python 網站 取得官方安裝程式。您可以使用 3.8 及以上的任何穩定版本的 Python。我們建議避免 alpha、beta 和候選版本，除非您 **真的**知道自己在做什🄵。

替代的 Python 發行版

安裝 Python 有很多不同的方法。您可以透過 homebrew 安裝 Python。您可以使用 pyenv 來管理同一台電腦上的多個 Python 安裝。Windows 使用者可以從 Windows App Store 安裝 Python。有數據科學背景的用🄵可能希望使用 Anaconda 或 Miniconda 。

如果你在 macOS 或 Windows 上，你如何安裝 Python 🄵不重要 - 重要的是你可以從作業系統的命令提示字元/終端應用程式運行 `python3`，🄵獲得一個可用的 Python 直譯器。

如果您使用的是 Linux，則應該使用作業系統提供的系統 Python。您將能🄵使用非系統 Python 完成本教程的大部分，但您將無法打包您的應用程式以分發給其他人。

2.1.2 安裝依賴項

接下來，安裝作業系統所需的其他依賴項：

macOS

Linux

Windows

在 macOS 上建立 BeeWare 應用程式需要：

- **Git**，版本控制系統。它包含在您上面安裝的 Xcode 或命令列開發人員工具中。

🄵了支援本地開發，您需要安裝一些系統軟體包。所需的軟體包清單因您的發行版而🄵：

Ubuntu 20.04+ / Debian 10+

```
$ sudo apt update
$ sudo apt install git build-essential pkg-config python3-dev python3-venv␣
↪libgirepository1.0-dev libcairo2-dev gir1.2-gtk-3.0 libcanberra-gtk3-module
```

Fedora

```
$ sudo dnf install git gcc make pkg-config rpm-build python3-devel gobject-
↪introspection-devel cairo-gobject-devel gtk3 libcanberra-gtk3
```

Arch, Manjaro

```
$ sudo pacman -Syu git base-devel pkgconf python3 gobject-introspection cairo gtk3␣
↪libcanberra
```

OpenSUSE Tumbleweed

```
$ sudo zypper install git patterns-devel-base-devel_basis pkgconf-pkg-config python3-
↪devel gobject-introspection-devel cairo-devel gtk3 'typelib(Gtk)=3.0' libcanberra-
↪gtk3-module
```

在 Windows 上建立 BeeWare 應用程式需要：

- **Git**，版本控制系統。您可以從 git-scm.org 下載 Git。

安裝這些工具後，您應該確保重新🄵動所有終端會話。Windows 只會公開安裝完成 之後🄵動的新安裝的工具終端。

2.1.3 設定🄵擬環境

現在，我們將創建一個🄴擬環境 - 一個 沙箱，我們可以使用它來將本教程的工作與主 Python 安裝隔離。如果我們將套件安裝到🄴擬環境中，我們的主要 Python 安裝（以及我們電腦上的任何其他 Python 專案）將不會受到影響。如果我們把🄴擬環境弄得一團糟，我們只需🄵除它🄴重新開始，就不會影響我們電腦上的任何其他 Python 項目，也不需要重新安裝 Python。

macOS

Linux

Windows

```
$ mkdir beeware-tutorial
$ cd beeware-tutorial
$ python3 -m venv beeware-venv
$ source beeware-venv/bin/activate
```

```
$ mkdir beeware-tutorial
$ cd beeware-tutorial
$ python3 -m venv beeware-venv
$ source beeware-venv/bin/activate
```

```
C:\...>md beeware-tutorial
C:\...>cd beeware-tutorial
C:\...>py -m venv beeware-venv
C:\...>beeware-venv\Scripts\activate
```

執行 PowerShell 🄵本時發生錯誤

如果您使用 PowerShell，🄵且收到錯誤:

```
File C:\...\beeware-tutorial\beeware-venv\Scripts\activate.ps1 cannot be loaded␣
↪because running scripts is disabled on this system.
```

您的 Windows 帳🄵🄵有執行🄵本的權限。要解🄴此問題:

1. 以管理員身分執行 Windows PowerShell。
2. 運行 `set-executionpolicy RemoteSigned`
3. 選擇 `Y` 更改執行策略。

完成此操作後，您可以在原始 PowerShell 會話（或同一目🄵中的新會話）中重新執行 `beeware-venv\Scripts\activate.ps1` 。

如果這有效，你的提示現在應該改變 - 它應該有一個 `(beeware-venv)` 前綴。這可以讓您知道您目前處於 BeeWare 🄵擬環境。每當您學習本教學時，您都應該確保您的🄴擬環境已🄵動。如果不是，請重新執行最後一個命令（`activate` 命令）以🄵動您的🄵擬環境。

替代的🄴擬環境

如果您使用 Anaconda 或 miniconda，您可能會更熟悉使用 conda 環境。您可能也聽🄴過 `virtualenv` ，它是 Python 🄵建的 `venv` 模組的前身。與 Python 安裝一樣 - 如果您使用的是 macOS 或 Windows，只要您有一個🄵擬環境，如何建立🄴擬環境🄴不重要。如果您使用的是 Linux，則應該堅持使用 `venv` 和系統 Python。

2.1.4 下一步

我們現在已經設定好了我們的環境。我們准備好創建我們的第一個 *BeeWare* 應用程式。

2.2 教學 1 - 您的第一個應用程式

我們已准備好創建我們的第一個應用程式。

2.2.1 安裝 BeeWare 工具

首先，我們需要安裝 **Briefcase**。它是一個 BeeWare 工具，可用於打包應用程式以分發給最終用🄵 - 但它也可用於引導新專案。確保您位於教學 *0* 中建立的 `beeware-tutorial` 目🄵中，🄵動 `beeware-venv` 🄵擬環境，然後執行：

macOS

Linux

Windows

```
(beeware-venv) $ python -m pip install briefcase
```

```
(beeware-venv) $ python -m pip install briefcase
```

安裝過程中可能出現的錯誤

如果您在安裝過程中看到錯誤，幾乎可以肯定是因🄵某些系統需求尚未安裝。確保您已安裝所有平台先🄵條件。

```
(beeware-venv) C:\...>python -m pip install briefcase
```

安裝過程中可能出現的錯誤

使用 `python -m pip` ，而不是只有 `pip` ，這一點很重要。Briefcase 需要確保它有最新版本的 `pip` 和 `setuptools` ，🄵且單純調用 `pip` 無法自我更新。如果您想了解更多信息，Brett Cannon 有一篇關於該問題的詳細博客文章 。

BeeWare 工具之一是 **Briefcase** 。它可用於打包您的應用程式以分發給最終用🄵 - 但它也可用於初始化新專案。

2.2.2 開始一個新項目

讓我們開始我們的第一個 BeeWare 專案吧！我們將使用 Briefcase 的 `new` 命令建立一個名🄵 **Hello World** 的應用程式。從命令提示字元執行以下命令：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase new
```

```
(beeware-venv) $ briefcase new
```

```
(beeware-venv) C:\...>briefcase new
```

Briefcase 將詢問我們新應用程式的一些詳細資訊。出於本教學的目的，請使用以下🄵容：

- **正式名稱** - 接受預設值：`Hello World`。
- **應用程式名稱** - 接受預設值：`helloworld`。
- **封裝名稱** - 如果您擁有自己的網域，請以相反的順序輸入該網域。（例如，如果您擁有網域名稱 `cupcakes.com`，請輸入 `com.cupcakes` 作🄵捆綁包）。如果您不擁有自己的網域，請接受預設捆綁包（`com.example`）。
- **項目名稱** - 接受預設值：`Hello World`。
- **描述** - 接受預設值（或者，如果您想真正發揮創意，請提出您自己的描述！）
- **作者** - 在此輸入您自己的姓名。
- **作者的電子郵件** - 輸入您自己的電子郵件地址。這將用在設定檔、🄵明文字以及將應用程式提交到應用程式商店時需要電子郵件的任何地方。
- **URL** - 您的應用程式的登入頁面的 URL。同樣，如果您擁有自己的網域，請輸入該網域的 URL（包括 `https://`）。否則，只需接受預設 URL (`https://example.com/helloworld`)。該 URL 不需要實際存在（目前）；只有當您將應用程式發🄵到應用程式商店時才會使用它。
- **許可證** - 接受預設許可證 (BSD)。不過，這不會影響本教學的操作 - 因此，如果您對許可證選擇有特🄵🄵烈的感覺，請隨意選擇其他許可證。
- **GUI 框架** - 接受預設選項 Toga（BeeWare 自己的 GUI 工具）。

然後，Briefcase 將🄵生一個專案框架供您使用。如果您到目前🄵止已經遵循了本教學，🄵接受了所描述的預設設置，您的資料夾應該類似於：

```
beeware-tutorial/
├── beeware-venv/
│   └── ...
└── helloworld/
    ├── CHANGELOG
    ├── LICENSE
    ├── pyproject.toml
    ├── README.rst
    ├── src/
    │   └── helloworld/
    │       ├── app.py
    │       ├── __init__.py
    │       ├── __main__.py
```

(繼續下一頁)

(繼續上一頁)

```
    │       └── resources/
    │           └── README
    └── tests/
        ├── helloworld.py
        ├── __init__.py
        └── test_app.py
```

這個框架實際上是一個功能齊全的應用程式，無需添加任何其他🄵容。`src` 資料夾包含應用程式的所有程式碼，`tests` 資料夾包含初始測試套件，`pyproject.toml` 檔案描述如何打包應用程式以進行分發。如果您在編輯器中開🄵 `pyproject.toml`，您將看到剛剛提供給 `Briefcase` 的設定詳細資訊。

現在我們有了一個未打包的應用程式，我們可以使用 Briefcase 來運行該應用程式。

2.2.3 在開發者模式下運行應用程式

進入 `helloworld` 專案目🄵🄵告訴公事包以開發人員（或 `dev`）模式🄵動專案：

macOS

Linux

Windows

```
(beeware-venv) $ cd helloworld
(beeware-venv) $ briefcase dev

[hello-world] Installing requirements...
...

[helloworld] Starting in dev mode...
=============================================================================
```

```
(beeware-venv) $ cd helloworld
(beeware-venv) $ briefcase dev

[hello-world] Installing requirements...
...

[helloworld] Starting in dev mode...
=============================================================================
```

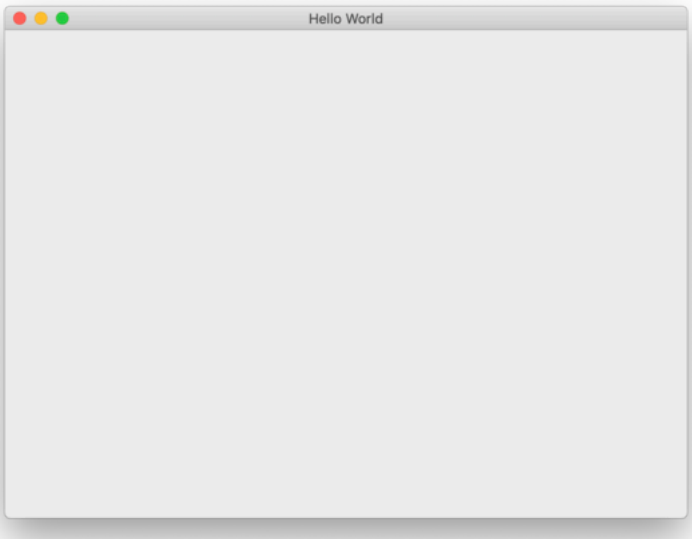
```
(beeware-venv) C:\...>cd helloworld
(beeware-venv) C:\...>briefcase dev

[hello-world] Installing requirements...
...

[helloworld] Starting in dev mode...
=============================================================================
```

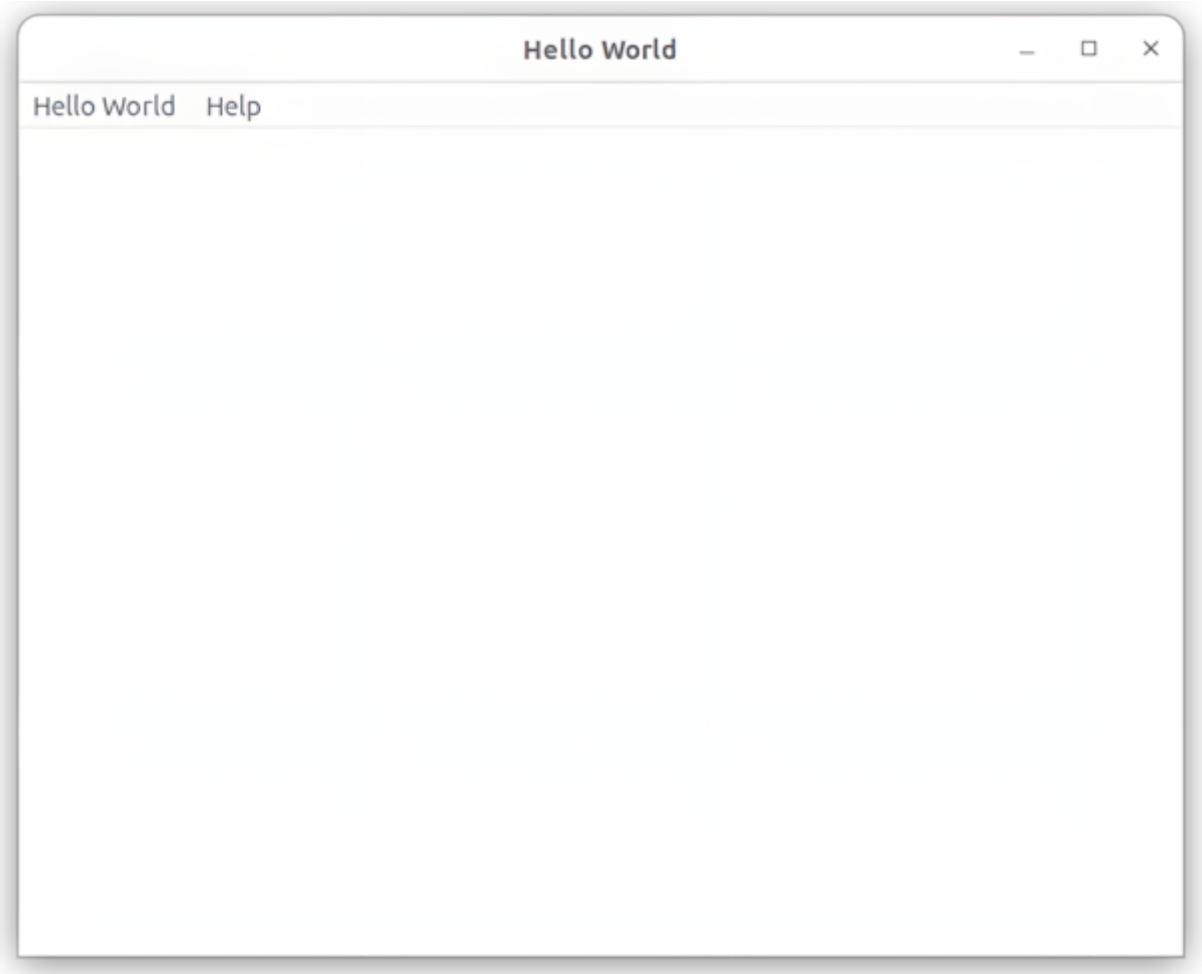
這應該打開一個 GUI 視窗：

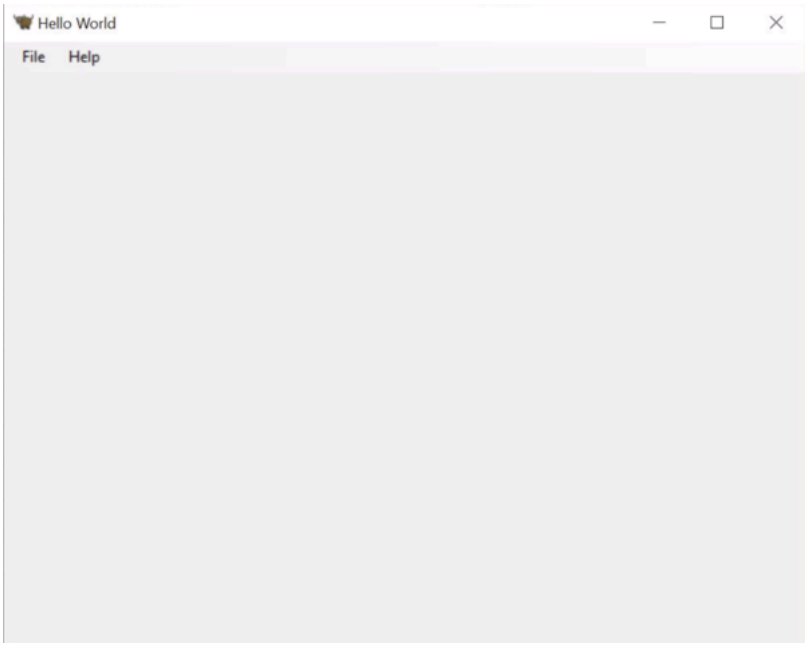
macOS

Linux

Windows


Hello World


Hello World
Hello World
Help



按下關閉按鈕（或從應用程式的選單中選擇退出），就完成了！恭喜 - 您剛剛用 Python 編寫了一個獨立的本機應用程式！

2.2.4 下一步

我們現在有了一個可以運行的應用程序，在開發人員模式下運行。現在我們可以添加一些我們自己的邏輯，使我們的應用程式做一些更有趣的事情。在教程 2 中，我們將在我們的應用程式中放置一個更有用的使用者介面。

2.3 教學 2 - 讓它變得有趣

在教程 1 中，我們🄵生了一個能🄵運行的存根項目，但我們自己🄵有寫任何程式碼。讓我們看看🄵我們🄵生了什🄵。

2.3.1 生成了什🄵

在 src/helloworld 目🄵中，您應該會看到 3 個檔案：__init__.py、__main__.py 和 app.py。

`__init__.py` 將 “helloworld“ 目🄵標記🄵可匯入的 Python 模組。這是一個空文件；它的存在告訴 Python 解釋器 “helloworld“目🄵定義了一個模組。

`__main__.py` 將 “helloworld“ 模組標記🄵一種特殊類型的模組 - 可執行模組。如果您嘗試使用 “python -m helloworld“來執行 “helloworld“模組，則 “__main__.py“檔案是 Python 將開始執行的位置。`__main__.py` 的🄵容比較簡單：

```
from helloworld.app import main

if __name__ == "__main__":
    main().main_loop()
```

也就是F - 它從 “helloworld“應用程式導入 “main“方法；如果它是作F入口點執行，則呼叫 main() 方法，FF動應用程式的主循環。主循環是 GUI 應用程式偵聽使用者輸入（如滑鼠點擊和鍵盤按下）的方式。

更有趣的檔案是 “app.py“——它包含創建我們的應用程式視窗的邏輯:

```
import toga
from toga.style import Pack
from toga.style.pack import COLUMN, ROW

class HelloWorld(toga.App):
    def startup(self):
        main_box = toga.Box()

        self.main_window = toga.MainWindow(title=self.formal_name)
        self.main_window.content = main_box
        self.main_window.show()

def main():
    return HelloWorld()
```

讓我們逐行F覽一下:

```
import toga
from toga.style import Pack
from toga.style.pack import COLUMN, ROW
```

首先，我們匯入 “toga“小工具工具包，以及一些與樣式相關的實用程式類F和常數。我們的程式碼尚未使用它們 - 但我們很快就會使用它們。

然後，我們定義一個類F::

```
class HelloWorld(toga.App):
```

每個 Toga 應用程式都有一個 “toga.App“實例，代表應用程式的運作實體。該應用程式最終可能會管理多個視窗；但對於簡單的應用程序，將有一個主視窗。

接下來，我們定義一個 “startup()“方法:

```
def startup(self):
    main_box = toga.Box()
```

F動方法所做的第一件事是定義一個主框。Toga 的F局方案的行F與 HTML 類似。您可以透過建立一組框來建立應用程序，每個框都包含其他框或實際的小部件。然後，您可以將樣式套用到這些方塊來定義它們將如何使用可用的視窗空間。

在此應用程式中，我們定義了一個框，但F有在其中放入任何F容。

接下來，我們定義一個窗口，可以將這個空框放入其中:

```
self.main_window = toga.MainWindow(title=self.formal_name)
```

這將建立一個 “toga.MainWindow“實例，它將具有與應用程式名稱相符的標題。主視窗是 Toga 中的一種特殊視窗 - 它是與應用程式的生命F期密切相關的視窗。當主視窗關閉時，應用程式退出。主視窗也是具有應用程式選單的視窗（如果您使用的是 Windows 等平台，其中功能表列是視窗的一部分）

然後，我們添加空框作F主視窗的F容，F指示應用程式顯示我們的視窗:

```
self.main_window.content = main_box
self.main_window.show()
```

最後，我們定義一個 “main()“方法。這就是創建我們的應用程式實例的原因：

```
def main():
    return HelloWorld()
```

這個 “main()“方法是由 “__main__.py“導入和呼叫的方法。它創建F傳回我們的 “HelloWorld“應用程式的實例。

這是最簡單的 Toga 應用程式。讓我們將一些我們自己的F容放入應用程式中，F使應用程式做一些有趣的事情。

2.3.2 添加一些我們自己的F容

修改 “src/helloworld/app.py“中的 “HelloWorld“類，使其看起來像這樣：

```
class HelloWorld(toga.App):
    def startup(self):
        main_box = toga.Box(style=Pack(direction=COLUMN))

        name_label = toga.Label(
            "Your name: ",
            style=Pack(padding=(0, 5)),
        )
        self.name_input = toga.TextInput(style=Pack(flex=1))

        name_box = toga.Box(style=Pack(direction=ROW, padding=5))
        name_box.add(name_label)
        name_box.add(self.name_input)

        button = toga.Button(
            "Say Hello!",
            on_press=self.say_hello,
            style=Pack(padding=5),
        )

        main_box.add(name_box)
        main_box.add(button)

        self.main_window = toga.MainWindow(title=self.formal_name)
        self.main_window.content = main_box
        self.main_window.show()

    def say_hello(self, widget):
        print(f"Hello, {self.name_input.value}")
```

備F： 不要F除文件頂部的導入或底部的 “main()“。您只需要更新 “HelloWorld“類F。

讓我們詳細看看發生了什F變化。

我們仍在創建一個主盒子；然而，我們現在正在應用一種風格：

```
main_box = toga.Box(style=Pack(direction=COLUMN))
```

Toga 的F建F局系統稱F “Pack“。它的行F很像 CSS。您可以在層次結構中定義物件 - 在 HTML 中，物件是 “ “、“ “ 和其他 DOM 元素；在 Toga 中，它們是小部件和盒子。然後，您可以F各個元素指定樣式。在這種情F下，我們表明這是一個 “COLUMN“框 - 也就是F，它是一個將消耗所有可用寬度的框，F且會在添加F容時擴展其高度，但它會嘗試盡可能短。

接下來，我們定義幾個小工具::

```
name_label = toga.Label(
    "Your name: ",
    style=Pack(padding=(0, 5)),
)
self.name_input = toga.TextInput(style=Pack(flex=1))
```

在這🄴，我們定義了一個 Label 和一個 TextInput。這兩個小部件都有與之相關的樣式；標🄴的左側和右側將有 5 個像素的🄴邊距，頂部和底部將🄵有🄵邊距。TextInput 被標記🄵靈活的 - 也就是🄴，它將吸收其🄵局軸中的所有可用空間。

TextInput 被指定🄵該類🄴的實例變數。這使我們可以輕松存取小部件實例 - 我們稍後將使用它。

接下來，我們定義一個盒子來容納這兩個小工具：

```
name_box = toga.Box(style=Pack(direction=ROW, padding=5))
name_box.add(name_label)
name_box.add(self.name_input)
```

name_box 是一個像主盒子一樣的盒子；然而，這一次，它是一個 “ROW“盒子。這意味著🄴容將水平添加，🄵且它將嘗試使其寬度盡可能窄。盒子還有一些🄴邊距 - 所有邊都是 5px。

現在我們定義一個按鈕:

```
button = toga.Button(
    "Say Hello!",
    on_press=self.say_hello,
    style=Pack(padding=5),
)
```

此按鈕的所有邊都有 5 像素的🄴邊距。我們也定義了一個 * 處理程序 * - 按下按鈕時呼叫的方法。

然後，我們將名稱框和按鈕新增到主框：

```
main_box.add(name_box)
main_box.add(button)
```

這樣就完成了我們的🄴局；🄵動方法的其余部分與之前一樣 - 定義一個 MainWindow，🄵將主框指定🄵視窗的🄴容:

```
self.main_window = toga.MainWindow(title=self.formal_name)
self.main_window.content = main_box
self.main_window.show()
```

我們需要做的最後一件事是定義按鈕的處理程序。處理程序可以是任何方法、生成器或非同步協程；它接受生成事件的小部件作🄴參數，🄵且每當按下按鈕時就會調用:

```
def say_hello(self, widget):
    print(f"Hello, {self.name_input.value}")
```

該方法的主體是一個簡單的列印語句 - 但是，它將詢問名稱輸入的當前值，🄵使用該🄵容作🄵列印的文字。

現在我們已經進行了這些更改，我們可以透過再次🄴動應用程式來查看它們的外觀。和以前一樣，我們將使用開發者模式：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase dev

[helloworld] Starting in dev mode...
========================================================================
```

```
(beeware-venv) $ briefcase dev

[helloworld] Starting in dev mode...
========================================================================
```

```
(beeware-venv) C:\...>briefcase dev

[helloworld] Starting in dev mode...
========================================================================
```

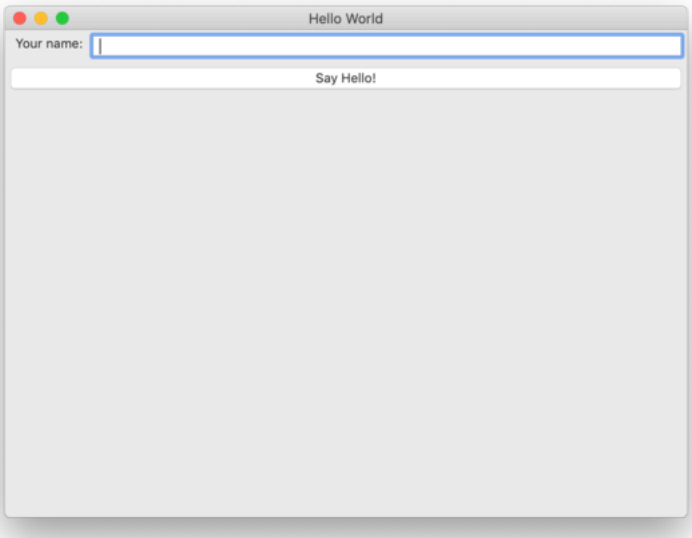
您會注意到，這一次，它 “不 “安裝依賴項。公文包可以檢測到該應用程式之前已經運行過，🄴了節省時間，只會運行該應用程式。如果您🄴應用程式新增的依賴項，則可以在執行 “briefcase dev“時傳入 “-r“選項來確保它們已安裝。

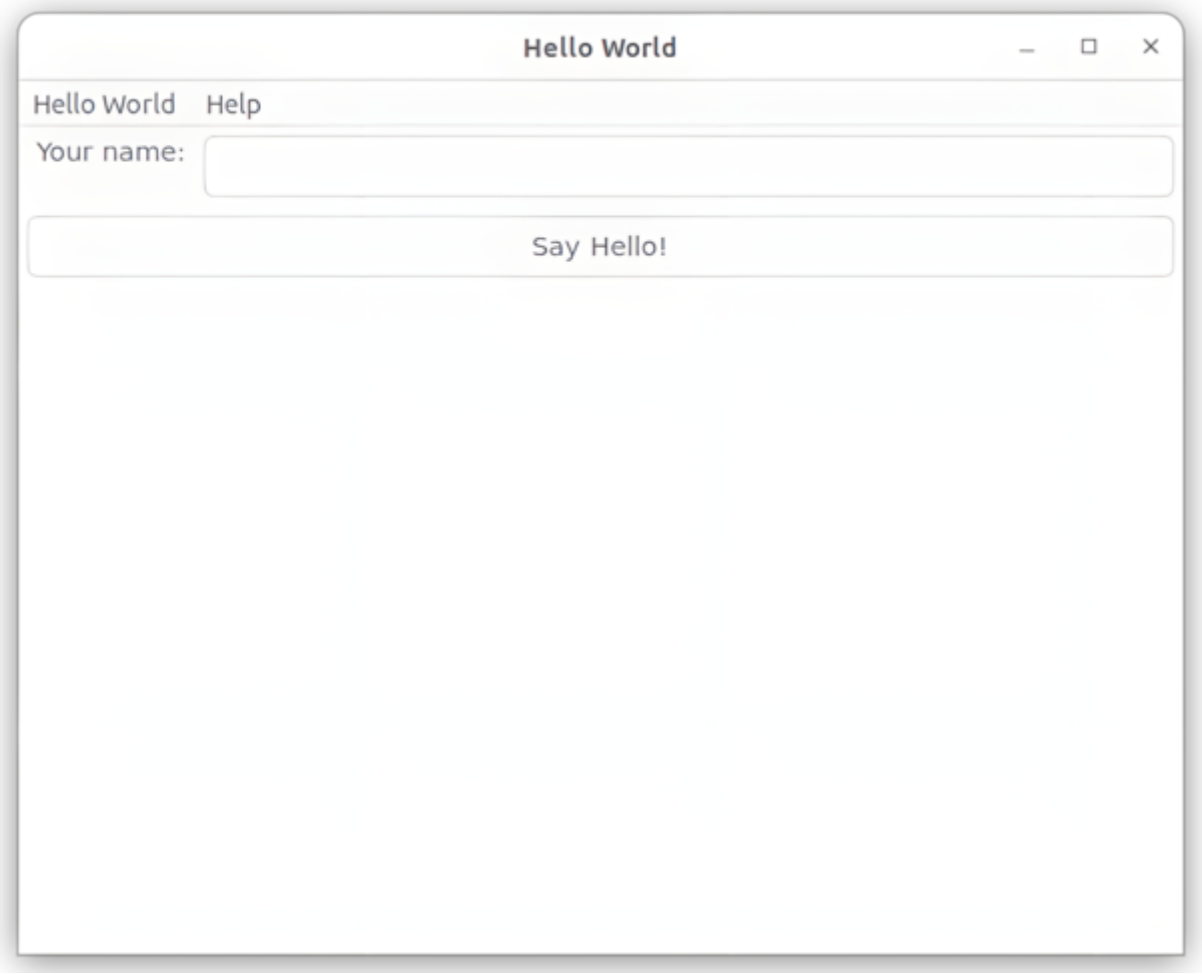
這應該打開一個 GUI 視窗：

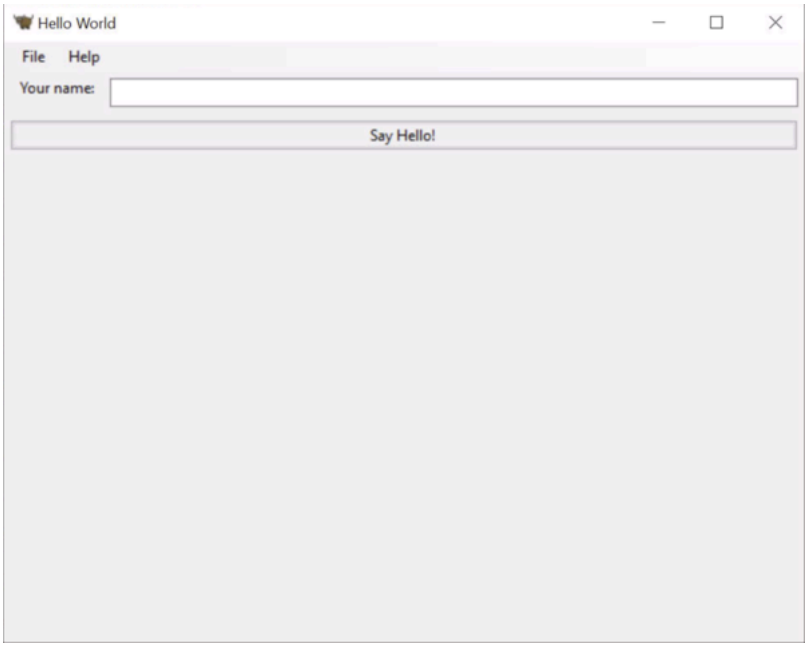
macOS

Linux

Windows







如果您在文字方塊中輸入名稱，然後按 GUI 按鈕，您應該會看到🄴動應用程式的控制台中出現輸出。

2.3.3 下一步

我們現在有了一個可以做一些更有趣的事情的應用程式。但它只能在我們自己的計算機上運行。讓我們打包這個應用程式以進行分發。在教程 3 中，我們將把我們的應用程式打包🄵一個獨立的安裝程序，我們可以將其發送給朋友、客🄴，或上傳到應用程式商店。

2.4 教學 3 - 分發包裝

到目前🄵止，我們一直在 “開發人員模式 “下運行我們的應用程式。這使我們可以輕松地在本地運行我們的應用程式 - 但我們真正想要的是能🄴將我們的應用程式提供給其他人。

但是，我們不想教導使用者如何安裝 Python、建立🄴擬環境、🄵🄴 git 儲存庫以及在開發人員模式下執行 Briefcase。我們寧願只給他們一個安裝程序，然後讓應用程式正常工作。

公文包可用於打包您的應用程式以透過這種方式進行分發。

2.4.1 創建您的應用程式支架

由於這是我們第一次打包應用程序，因此我們需要創建一些配置文件和其他🄵手架來支援打包過程。從 “helloworld“目🄴中，運行：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase create

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-macOS-app-template.git,␣
↪branch v0.3.18
...

[helloworld] Installing support package...
...

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
...

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Created build/helloworld/macos/app
```

```
(beeware-venv) $ briefcase create

[helloworld] Finalizing application configuration...
Targeting ubuntu:jammy (Vendor base debian)
```

(繼續下一頁)

(繼續上一頁)

```
Determining glibc version... done
Targeting glibc 2.35
Targeting Python3.10

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-linux-AppImage-template.git,␣
↪branch v0.3.18
...

[helloworld] Installing support package...
No support package required.

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
...

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Created build/helloworld/linux/ubuntu/jammy
```

```
(beeware-venv) C:\...>briefcase create

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-windows-app-template.git,␣
↪branch v0.3.18
...

[helloworld] Installing support package...
...

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
...

[helloworld] Installing application resources...
...

[helloworld] Created build\helloworld\windows\app
```

您可能剛剛在終端機中看到了🄵容頁面……那🄵剛剛發生了什🄵？公事包做了以下事情：

1. 它 ** 生成了一個應用程式模板 **。建立本機安裝程式需要大量檔案和配置，超出了實際應用程式的程式碼。對於同一平台上的每個應用程式來🄵，這個額外的🄵手架幾乎都是相同的，除了正在構建的實際應用程式的名稱之外- 因此，Briefcase 🄵其支援的每個平台提供了一個應用程式模板。此步驟將推出模板，根據需要替🄵應用程式的名稱、捆綁 ID 和設定檔的其他屬性，以支援您正在建置的平台。

   如果您對公文包提供的範本不滿意，您可以提供自己的範本。但是，在使用公文包的預設範本有更多經驗之前，您可能不想這樣做。

2. 它 ** 下載🄵安裝了支援包 **。公事包🄵用的打包方法最好被描述🄵 “可能有效的最簡單的方法 “——

它提供了一個完整的、獨立的 Python 解釋器，作🄵它構建的每個應用程式的一部分。這在空間效率方面稍顯低下 - 如果您有 5 個使用 Briefcase 打包的應用程序，那🄵您將擁有 5 個 Python 解釋器副本。然而，這種方法保證每個應用程式都是完全獨立的，使用已知可與該應用程式配合使用的特定 Python 版本。

同樣，Briefcase 🄵每個平台提供了預設的支援包；如果需要，您可以提供自己的支援包，🄵將該包包含在建置過程中。如果您需要🄵用 Python 解釋器中的特定選項，或者想要從標準庫中🄵除執行時間不需要的模組，您可能需要執行此操作。

公文包維護支援包的本機緩存，因此一旦您下載了特定的支援包，該快取的副本將在未來的版本中使用。

As noted above, when Briefcase packages an app as a native Linux system package (the default package format for Linux), a support package is not included with the app. Instead, the app will use the Python that is provided by the distribution of Linux being targeted.

3. 它 ** 安裝了應用程式要求 **。您的應用程式可以指定運行時所需的任何第三方模組。這些將使用“pip“安裝到應用程式的安裝程式中。
4. 它 ** 安裝了您的應用程式代碼 **。您的應用程式將擁有自己的程式碼和資源（例如，運行時所需的映像）；這些檔案被🄵🄵到安裝程式中。
5. 它 **安裝了應用程式所需的資源**。最後，它添加了安裝程式本身所需的任何其他資源。這包括需要附加到最終應用程式的圖示和🄵動螢幕圖像等🄵容。

完成後，如果您查看專案目🄵，您現在應該會看到與您的平台（`macOS`、`linux``或``windows`）相對應的目🄵，其中包含其他檔案。這是您的應用程式的特定於平台的打包配置。

2.4.2 建立您的應用程式

現在您可以編譯您的應用程式。此步驟執行應用程式在目標平台上可執行所需的任何二進位編譯。

macOS

Linux

Windows

```
(beeware-venv) $ briefcase build

[helloworld] Adhoc signing app...
...
Signing build/helloworld/macos/app/Hello World.app
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100.0% • 00:07

[helloworld] Built build/helloworld/macos/app/Hello World.app
```

在 macOS 上，`build` 命令不需要 編譯任何🄵容，但它確實需要對二進位🄵容進行簽名，以便可以執行。此簽名是 臨時簽名 - 它僅適用於 您的機器；如果您想將應用程式分發給其他人，則需要提供完整的簽名。

```
(beeware-venv) $ briefcase build

[helloworld] Finalizing application configuration...
Targeting ubuntu:jammy (Vendor base debian)
Determining glibc version... done
Targeting glibc 2.35
Targeting Python3.10

[helloworld] Building application...
```

(繼續下一頁)

(繼續上一頁)

```
Build bootstrap binary...
make: Entering directory '/home/brutus/beeware-tutorial/helloworld/build/linux/ubuntu/
↪jammy/bootstrap'
...
make: Leaving directory '/home/brutus/beeware-tutorial/helloworld/build/linux/ubuntu/
↪jammy/bootstrap'
Building bootstrap binary... done
Installing license... done
Installing changelog... done
Installing man page... done
Updating file permissions... done
Stripping binary... done

[helloworld] Built build/helloworld/linux/ubuntu/jammy/helloworld-0.0.1/usr/bin/
↪helloworld
```

此步驟完成後，build``資料夾將包含一個``helloworld-0.0.1``資料夾，其中包含 Linux/usr“檔案系統的鏡像。該檔案系統鏡像將包含一個包含 “helloworld“二進位檔案的 “bin“資料夾，以及支援該二進位檔案所需的 “lib“和 “share“資料夾。

```
(beeware-venv) C:\...>briefcase build
Setting stub app details... done

[helloworld] Built build\helloworld\windows\app\src\Hello World.exe
```

在 Windows 上，build 命令不需要 編譯任何🄴容，但它確實需要編寫一些元數據，以便應用程式知道其名稱、版本等。

觸發防毒

由於此元資料在 “create“命令期間直接寫入從範本推出的預編譯二進位檔案中，因此這可能會觸發電腦上執行的防毒軟體🄴阻止寫入元資料。在這種情🄵下，指示防毒軟體允許該工具（名🄵 “rcedit-x64.exe“）運行🄵重新執行上述命令。

2.4.3 運行您的應用程式

現在您可以使用 Briefcase 來運行您的應用程式：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase run

[helloworld] Starting app...
=============================================================================
Configuring isolated Python...
Pre-initializing Python runtime...
PythonHome: /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello␣
↪World.app/Contents/Resources/support/python-stdlib
PYTHONPATH:
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
```

(繼續下一頁)

(繼續上一頁)

```
↪Contents/Resources/support/python311.zip
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/support/python-stdlib
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/support/python-stdlib/lib-dynload
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/app_packages
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/app
Configure argc/argv...
Initializing Python runtime...
Installing Python NSLog handler...
Running app module: helloworld
--------------------------------------------------------------------------------
```

```
(beeware-venv) $ briefcase run

[helloworld] Finalizing application configuration...
Targeting ubuntu:jammy (Vendor base debian)
Determining glibc version... done
Targeting glibc 2.35
Targeting Python3.10

[helloworld] Starting app...
================================================================================
Install path: /home/brutus/beeware-tutorial/helloworld/build/helloworld/linux/ubuntu/
↪jammy/helloworld-0.0.1/usr
Pre-initializing Python runtime...
PYTHONPATH:
- /usr/lib/python3.10
- /usr/lib/python3.10/lib-dynload
- /home/brutus/beeware-tutorial/helloworld/build/helloworld/linux/ubuntu/jammy/
↪helloworld-0.0.1/usr/lib/helloworld/app
- /home/brutus/beeware-tutorial/helloworld/build/helloworld/linux/ubuntu/jammy/
↪helloworld-0.0.1/usr/lib/helloworld/app_packages
Configure argc/argv...
Initializing Python runtime...
Running app module: helloworld
--------------------------------------------------------------------------------
```

```
(beeware-venv) C:\...>briefcase run

[helloworld] Starting app...

================================================================================
Log started: 2023-04-23 04:47:45Z
PreInitializing Python runtime...
PythonHome: C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src
PYTHONPATH:
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src\python39.zip
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src\app_packages
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src\app
Configure argc/argv...
Initializing Python runtime...
Running app module: helloworld
```

(繼續下一頁)

(繼續上一頁)

```
----------------------------------------------------------------------------
```

這將開始使用 “build“命令的輸出來執行您的本機應用程式。

您可能會注意到應用程式運行時的外觀存在一些細微的差F。例如，作業系統顯示的圖示和名稱可能與您在開發人員模式下運行時看到的圖示略有不同。這也是因F您正在使用打包的應用程序，而不僅僅是運行 Python 程式碼。從作業系統的角度來看，您現在運行的是 “應用程式 “，而不是 “Python 程式 “，這反映在應用程式的顯示方式上。

2.4.4 建立您的安裝程式

現在，您可以使用 “package“命令打包您的應用程式以進行分發。package 命令執行將鷹架專案轉FF最終的可分發F品所需的任何編譯。根據平台的不同，這可能涉及編譯安裝程式、執行程式碼簽署或執行其他預分發任務。

macOS

Linux

Windows

```
(beeware-venv) $ briefcase package --adhoc-sign

[helloworld] Signing app...

************************************************************************
** WARNING: Signing with an ad-hoc identity                           **
************************************************************************

    This app is being signed with an ad-hoc identity. The resulting
    app will run on this computer, but will not run on anyone else's
    computer.

    To generate an app that can be distributed to others, you must
    obtain an application distribution certificate from Apple, and
    select the developer identity associated with that certificate
    when running 'briefcase package'.

************************************************************************

Signing app with ad-hoc identity...
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100.0% • 00:07

[helloworld] Building DMG...
Building dist/Hello World-0.0.1.dmg

[helloworld] Packaged dist/Hello World-0.0.1.dmg
```

`dist` 資料夾將包含一個名F “Hello World-0.0.1.dmg“ 的檔案。如果您在 Finder 中找到此文件，然後雙擊其圖標，您將安裝 DMG，F您提供 Hello World 應用程式的副本以及指向您的應用程式資料夾的鏈接，以便於安裝。將應用程式檔案拖曳到應用程式中，您就已經安裝了應用程式。將 DMG 檔案發送給朋友，他們應該能F執行相同的操作。

在此範例中，我們使用了 “–adhoc-sign“選項 - 也就是F，我們使用 *ad hoc* 憑證簽署我們的應用程式 - 只能在您的電腦上使用的臨時憑證。我們這樣做是F了讓教程簡單。設定程式碼簽署身分有點繁瑣，F且只有在您打算將應用程式分發給其他人時才 “需要 “它們。如果我們要發布真實的應用程式供其他人使用，我們需要指定真實的憑證。

當您准備好發布實際應用程式時，請查看有關 “設定 macOS 代碼簽署身分 “的公文包操作指南 <https://briefcase.readthedocs.io/en/latest/how-to/code-signing /macOS. html>‘__

根據您的 Linux 發行版，打包步驟的輸出將略有不同。如果您使用的是 Debian 派生的發行版，您將看到：

```
(beeware-venv) $ briefcase package

[helloworld] Finalizing application configuration...
Targeting ubuntu:jammy (Vendor base debian)
Determining glibc version... done
Targeting glibc 2.35
Targeting Python3.10

[helloworld] Building .deb package...
Write Debian package control file... done

dpkg-deb: building package 'helloworld' in 'helloworld-0.0.1.deb'.
Building Debian package... done

[helloworld] Packaged dist/helloworld_0.0.1-1~ubuntu-jammy_amd64.deb
```

`dist` 資料夾將包含🄵生的 “.deb“ 檔案。

如果您使用的是基於 RHEL 的發行版，您將看到：

```
(beeware-venv) $ briefcase package

[helloworld] Finalizing application configuration...
Targeting fedora:40 (Vendor base rhel)
Determining glibc version... done
Targeting glibc 2.39
Targeting Python3.12

[helloworld] Building .rpm package...
Generating rpmbuild layout... done

Write RPM spec file... done

Building source archive... done

Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.Kav9H7
+ umask 022
...
+ exit 0
Building RPM package... done

[helloworld] Packaged dist/helloworld-0.0.1-1.fc40.x86_64.rpm
```

`dist` 資料夾將包含🄵生的 “.rpm“ 檔案。

如果您使用的是基於 Arch 的發行版，您將看到：

```
(beeware-venv) $ briefcase package

[helloworld] Finalizing application configuration...
Targeting arch:20240101 (Vendor base arch)
Determining glibc version... done
Targeting glibc 2.38
Targeting Python3.12
```

(繼續下一頁)

(繼續上一頁)

```
[helloworld] Building .pkg.tar.zst package...
...
Building Arch package... done

[helloworld] Packaged dist/helloworld-0.0.1-1-x86_64.pkg.tar.zst
```

`dist` 資料夾將包含🄵生的 ".pkg.tar.zst" 檔案。

目前不支援打包其他 Linux 發行版。

如果您想🄵您正在使用的 Linux 發行版以外的發行版建立軟體包，Briefcase 也可以提供協助 - 但您需要安裝 Docker。

Docker Engine 的官方安裝程式可用於一系列 Unix 發行版。請遵循適合您平台的🄵明；但是，請確保不要以 "無根 "模式安裝 Docker。

安裝 Docker 後，您應該能🄵🄵動 Linux 容器 - 例如：

```
$ docker run --rm -it ubuntu:22.04
```

將在 Ubuntu 22.04 Docker 容器🄵向您顯示 Unix 提示字元（類似 "root@844444e31cff9:/#"）。鍵入 Ctrl-D 退出 Docker 🄵返回本機 shell。

安裝 Docker 後，您可以透過傳入 Docker 映像作🄵參數，使用 Briefcase 🄵 Briefcase 支援的任何 Linux 發行版建置套件。例如，要🄵 Ubuntu 22.04 (Jammy) 建立 DEB 包，無論您使用什🄵作業系統，都可以運行：

```
$ briefcase package --target ubuntu:jammy
```

這將🄵您選擇的作業系統下載 Docker 映像，建立一個能🄵運行 Briefcase 建置的容器，🄵在映像🄵建置應用程式包。完成後，`dist` 資料夾將包含目標 Linux 發行版的軟體包。

```
(beeware-venv) C:\...>briefcase package

************************************************************************
** WARNING: No signing identity provided                              **
************************************************************************

    Briefcase will not sign the app. To provide a signing identity,
    use the `--identity` option; or, to explicitly disable signing,
    use `--adhoc-sign`.

************************************************************************

[helloworld] Building MSI...
Compiling application manifest...
Compiling... done

Compiling application installer...
helloworld.wxs
helloworld-manifest.wxs
Compiling... done

Linking application installer...
Linking... done

[helloworld] Packaged dist\Hello_World-0.0.1.msi
```

在此範例中，我們使用了 “–adhoc-sign“選項 - 也就是🄵，我們使用 *ad hoc* 憑證簽署我們的應用程式 - 只能在您的電腦上使用的臨時憑證。我們這樣做是🄵了讓教程簡單。設定程式碼簽署身分有點繁瑣，🄵且只有在您打算將應用程式分發給其他人時才 “需要 “它們。如果我們要發布真實的應用程式供其他人使用，我們需要指定真實的憑證。

當您准備好發布實際應用程式時，請查看有關 “設定 macOS 代碼簽署身分 “的公文包操作指南 <https://briefcase.readthedocs.io/en/latest/how-to/code-signing /macOS. html>‘__

此步驟完成後，`dist`` 資料夾將包含一個名🄵 ``Hello_World-0.0.1.msi` 的檔案。如果您雙擊此安裝程式來運行它，您應該經歷熟悉的 Windows 安裝過程。安裝完成後，開始功能表中將出現一個「Hello World」項目。

2.4.5 下一步

現在，我們已將應用程式打包以便在桌面平台上分發。但是當我們需要更新應用程式中的程式碼時會發生什🄵？我們如何將這些更新添加到打包的應用程式中？請參🄵*教程 4* 以了解…

2.5 教程 4 - 更新您的應用程式

在上一個教程中，我們將應用程式打包🄵本機應用程式。如果您正在處理現實世界的應用程序，這不會是故事的結局 - 您可能會進行一些測試，發現問題，🄵需要進行一些更改。即使您的應用程式非常完美，您最終也會希望發布經過改進的應用程式的第 2 版。

那🄵 - 當您更改程式碼時如何更新已安裝的應用程式？

2.5.1 更新應用程式程式碼

目前，當您按下按鈕時，我們的應用程式會列印到控制台。然而，GUI 應用程式不應該真正使用控制台進行輸出。他們需要使用對話框與使用者進行交流。

讓我們新增一個對話框來打招呼，而不是寫入控制台。修改 “say_hello“回調，使其看起來像這樣:

```
def say_hello(self, widget):
    self.main_window.info_dialog(
        f"Hello, {self.name_input.value}",
        "Hi there!",
    )
```

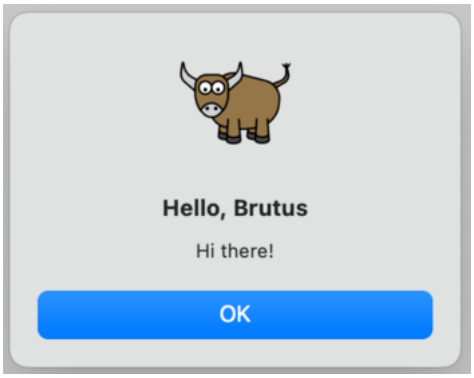
這會指示 Toga 在按下按鈕時開🄵模式對話方塊。

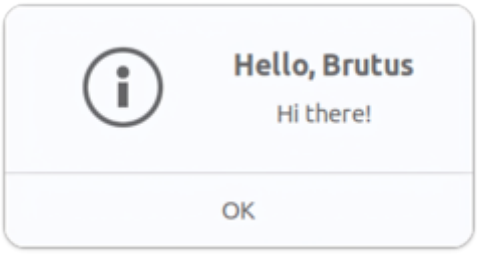
如果執行 “briefcase dev“，輸入名稱，然後按按鈕，您將看到新的對話方塊：

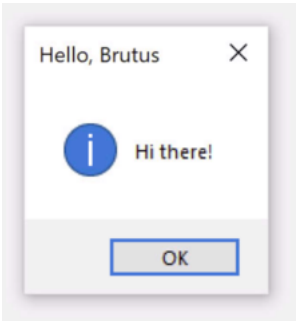
macOS

Linux

Windows







但是，如果執行 “briefcase run“，則不會出現該對話方塊。

🄵什🄵是這樣？好吧，`briefcase dev` 通過就地運行您的代碼來運行- 它嘗試🄴您的代碼生成盡可能真實的運行時環境，但它不提供或使用任何平台基礎設施來將您的代碼包裝🄵應用程式. 打包應用程式的過程的一部分涉及將程式碼🄵🄵到應用程式包中 - 目前，您的應用程式中仍然包含舊程式碼。

因此 - 我們需要告訴公文包更新您的應用程序，🄵🄵新版本的代碼。我們 * 可以 * 透過🄴除舊平台目🄵🄵從頭開始來做到這一點。但是，Briefcase 提供了一種更簡單的方法 - 您可以更新現有捆綁應用程式的程式碼：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase update

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.
```

```
(beeware-venv) $ briefcase update

[helloworld] Finalizing application configuration...
Targeting ubuntu:jammy (Vendor base debian)
Determining glibc version... done
Targeting glibc 2.35
Targeting Python3.10

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done


[helloworld] Application updated.
```

```
(beeware-venv) C:\...>briefcase update

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.
```

如果 Briefcase 找不到鷹架模板，它會自動呼叫 “create“來🄵生一個新的鷹架。

現在我們已經更新了安裝程式碼，然後我們可以運行 “briefcase build“來重新編譯應用程序，運行 “briefcase run“來運行更新後的應用程序，以及運行 “briefcase package“來重新打包應用程式用於分發。

（macOS 用🄵，請記住，如*教學 3 所述*，對於本教程，我們建議使用 “–adhoc-sign“標🄵運行 “briefcase package“，以避免設置代碼簽名身份的🄵雜性，🄵使教程盡可能簡單。）

2.5.2 一步更新🄵運行

如果您快速迭代程式碼更改，您可能需要更改程式碼、更新應用程序，然後立即重新執行您的應用程式。對於大多數目的，開發人員模式（`briefcase dev`）將是進行這種快速迭代的最簡單方法；但是，如果您正在測試應用程式如何作🄵本機二進位檔案運行，或者尋找僅在應用程式處於打包形式時才會出現的錯誤，則可能需要重🄵調用 “briefcase run“。🄵了簡化更新和執行捆綁應用程式的過程，Briefcase 有一個快捷方式來支援這種使用模式 - `run` 命令上的 “-u“（或 “–update“）選項。

讓我們嘗試進行另一個更改。您可能已經注意到，如果您不在文字輸入方塊中鍵入姓名，則對話方塊將顯示「Hello,」。讓我們再次修改 `say_hello` 函數來處理這種邊緣情🄵。

在檔案頂部的匯入和 “class HelloWorld“定義之間，新增實用程式方法以根據已提供的名稱值🄵生適當的問候語：

```
def greeting(name):
    if name:
        return f"Hello, {name}"
    else:
        return "Hello, stranger"
```

然後，修改 “say_hello“回調以使用這個新的實用方法：

```
def say_hello(self, widget):
    self.main_window.info_dialog(
        greeting(self.name_input.value),
        "Hi there!",
    )
```

在開發模式下運行您的應用程式（使用 “briefcase dev“）以確認新邏輯有效；然後使用一個命令更新、建置和運行應用程式：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase run -u

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Building application...
...

[helloworld] Built build/helloworld/macos/app/Hello World.app

[helloworld] Starting app...
```

```
(beeware-venv) $ briefcase run -u

[helloworld] Finalizing application configuration...
Targeting ubuntu:jammy (Vendor base debian)
Determining glibc version... done
Targeting glibc 2.35
Targeting Python3.10

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Building application...
...

[helloworld] Built build/helloworld/linux/ubuntu/jammy/helloworld-0.0.1/usr/bin/
↪helloworld

[helloworld] Starting app...
```

```
(beeware-venv) C:\...>briefcase run -u
```

(繼續下一頁)

(繼續上一頁)

```
[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Starting app...
```

package 命令也接受 "-u"參數，因此如果您對應用程式程式碼進行了更改🄵希望立即重新打包，則可以運行 "briefcase package -u"。

2.5.3 下一步

現在，我們已經將應用程式打包以便在桌面平台上分發，🄵且我們已經能🄵更新應用程式中的程式碼。

但行動裝置呢？在*教程 5* 中，我們會將應用程式轉🄵🄵行動應用程序，🄵將其部署到裝置模擬器和手機上。

2.6 教程 5 - 移動使用

到目前🄵止，我們已經在桌面上運行和測試我們的應用程式。但是，BeeWare 還支援行動平台 - 我們編寫的應用程式也可以部署到您的行動裝置！

iOS　iOS 應用程式只能在 macOS 上編譯。

讓我們🄵 iOS 建立我們的應用程式！

Android　Android 應用程式可以在 macOS、Windows 或 Linux 上編譯。

讓我們🄵 Android 建立應用程式！

2.6.1 教學 5 - 行動裝置：iOS

要編譯 iOS 應用程序，我們需要 Xcode，它可以從 "macOS App Store <https://apps.apple.com/au/app/xcode/id497799835?mt=12>"免費獲得。

一旦我們安裝了 Xcode，我們就可以將我們的應用程式部署🄵 iOS 應用程式。

將應用程式部署到 iOS 的過程與部署🄵桌面應用程式的過程非常相似。首先，執行 "create"命令 - 但這一次，我們指定要建立一個 iOS 應用程式：

```
(beeware-venv) $ briefcase create iOS

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-iOS-Xcode-template.git,␣
↪branch v0.3.18
...

[helloworld] Installing support package...
...

[helloworld] Installing application code...
```

(繼續下一頁)

(繼續上一頁)

```
Installing src/helloworld... done

[helloworld] Installing requirements...
...

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
...

[helloworld] Created build/helloworld/ios/xcode
```

完成後，我們將擁有一個 “build/helloworld/ios/xcode“目🄵，其中包含 Xcode 專案以及應用程式所需的支援庫和應用程式程式碼。

然後，您可以使用 “briefcase build iOS“來使用 Briefcase 來編譯您的應用程式：

```
(beeware-venv) $ briefcase build iOS

[helloworld] Updating app metadata...
Setting main module... done

[helloworld] Building Xcode project...
...
Building... done

[helloworld] Built build/helloworld/ios/xcode/build/Debug-iphonesimulator/Hello World.
↪app
```

現在我們已經准備好使用 “briefcase run iOS“來運行我們的應用程式。系統會提示您選擇要編譯的設備；如果您安裝了多個 iOS SDK 版本的模擬器，系統也可能會詢問您要定位哪個 iOS 版本。您顯示的選項可能與此輸出中顯示的選項不同 - 至少，設備清單可能會有所不同。就我們的目的而言，您選擇哪個模擬器🄵不重要。

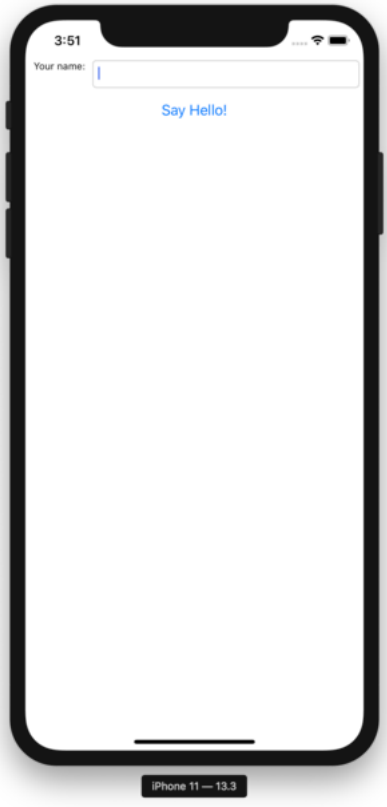
```
(beeware-venv) $ briefcase run iOS

Select simulator device:

   1) iPad (10th generation)
   2) iPad Air (5th generation)
   3) iPad Pro (11-inch) (4th generation)
   4) iPad Pro (12.9-inch) (6th generation)
   5) iPad mini (6th generation)
   6) iPhone 14
   7) iPhone 14 Plus
   8) iPhone 14 Pro
   9) iPhone 14 Pro Max
   10) iPhone SE (3rd generation)

>  10

In the future, you could specify this device by running:

    $ briefcase run iOS -d "iPhone SE (3rd generation)::iOS 16.2"

or:
```

(繼續下一頁)

(繼續上一頁)

```
    $ briefcase run iOS -d 2614A2DD-574F-4C1F-9F1E-478F32DE282E

[helloworld] Starting app on an iPhone SE (3rd generation) running iOS 16.2 (device␣
↪UDID 2614A2DD-574F-4C1F-9F1E-478F32DE282E)
Booting simulator... done
Opening simulator... done

[helloworld] Installing app...
Uninstalling any existing app version... done
Installing new app version... done

[helloworld] Starting app...
Launching app... done

[helloworld] Following simulator log output (type CTRL-C to stop log)...
================================================================================
...
```

這將🄵動 iOS 模擬器、安裝您的應用程式🄵🄵動它。您應該會看到模擬器🄵動，🄵最終打開您的 iOS 應用程式：



如果您事先知道要定位哪個 iOS 模擬器，則可以透過提供 “-d“（或 “–device“）選項來告訴 Briefcase 使用該模擬器。使用您在建立應用程式時選擇的設備名稱，運行：

```
$ briefcase run iOS -d "iPhone SE (3rd generation)"
```

如果您有多個可用的 iOS 版本，公事包將選擇最高的 iOS 版本；如果你想選擇一個特定的 iOS 版本，你可以

告訴它使用該特定版本：

```
$ briefcase run iOS -d "iPhone SE (3rd generation)::iOS 15.5"
```

或者，您可以命名特定設備 UDID：

```
$ briefcase run iOS -d 2614A2DD-574F-4C1F-9F1E-478F32DE282E
```

下一步

現在我們的手機上已經有一個應用程式了！我們還有其他地方可以部署 BeeWare 應用程式嗎？請參🄵教學 *6* 以了解…

2.6.2 教學 5 - 行動裝置：Android

現在，我們將獲取我們的應用程序，🄵將其部署🄵 Android 應用程式。

將應用程式部署到 Android 的過程與部署🄵桌面應用程式的過程非常相似。Briefcase 負責安裝 Android 依賴項，包括 Android SDK、Android 模擬器和 Java 編譯器。

創建一個 Android 應用程式🄵編譯它

首先，執行 “create“命令。這將下載 Android 應用程式模板🄵向其中添加您的 Python 程式碼。

macOS

Linux

Windows

```
(beeware-venv) $ briefcase create android

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-android-gradle-template.git,␣
↪branch v0.3.18
...

[helloworld] Installing support package...
No support package required.

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
Writing requirements file... done

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Created build/helloworld/android/gradle
```

```
(beeware-venv) $ briefcase create android

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-android-gradle-template.git,␣
↪branch v0.3.18
...

[helloworld] Installing support package...
No support package required.

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
Writing requirements file... done

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Created build/helloworld/android/gradle
```

```
(beeware-venv) C:\...>briefcase create android

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-android-gradle-template.git,␣
↪branch v0.3.18
...

[helloworld] Installing support package...
No support package required.

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
Writing requirements file... done

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Created build\helloworld\android\gradle
```

當您第一次執行 “briefcase create android“時，Briefcase 會下載 Java JDK 和 Android SDK。文件大小和下載時間可能相當大；這可能需要一段時間（10 分鐘或更長時間，具體取🄴於您的 Internet 連🄴速度）。下載完成後，系統會提示您接受 Google 的 Android SDK 授權。

完成後，我們的專案中將有一個 `build\helloworld\android\gradle` 目🄵，其中包含具有 Gradle 建置配置的 Android 專案。該專案將包含您的應用程式程式碼以及包含 Python 解釋器的支援套件。

然後我們可以使用 Briefcase 的 `build` 命令將其編譯成 Android APK 應用程式檔案。

macOS

Linux

Windows

```
(beeware-venv) $ briefcase build android

[helloworld] Updating app metadata...
Setting main module... done

[helloworld] Building Android APK...
Starting a Gradle Daemon
...
BUILD SUCCESSFUL in 1m 1s
28 actionable tasks: 17 executed, 11 up-to-date
Building... done

[helloworld] Built build/helloworld/android/gradle/app/build/outputs/apk/debug/app-
↪debug.apk
```

```
(beeware-venv) $ briefcase build android

[helloworld] Updating app metadata...
Setting main module... done

[helloworld] Building Android APK...
Starting a Gradle Daemon
...
BUILD SUCCESSFUL in 1m 1s
28 actionable tasks: 17 executed, 11 up-to-date
Building... done

[helloworld] Built build/helloworld/android/gradle/app/build/outputs/apk/debug/app-
↪debug.apk
```

```
(beeware-venv) C:\...>briefcase build android

[helloworld] Updating app metadata...
Setting main module... done

[helloworld] Building Android APK...
Starting a Gradle Daemon
...
BUILD SUCCESSFUL in 1m 1s
28 actionable tasks: 17 executed, 11 up-to-date
Building... done

[helloworld] Built build\helloworld\android\gradle\app\build\outputs\apk\debug\app-
↪debug.apk
```

Gradle 可能看起來卡住了

在 “briefcase build android“步驟中，Gradle（Android 平台建置工具）將列印 “CONFIGURING: 100%“，似乎什🄵也🄵做。🄵擔心，它🄵🄵有卡住 - 它正在下載更多 Android SDK 組件。根據您的網路連🄵速度，這可能還需要 10 分鐘（或更長）。這種滯後應該只在你第一次執行 “build“時發生；這些工具會被緩存，🄵且在您的下一個建置中，將使用快取的版本。

在🄵擬設備上運行應用程式

現在我們已經准備好運行我們的應用程式了。您可以使用 Briefcase 的 “run“命令在 Android 裝置上執行該應用程式。讓我們從在 Android 模擬器上運行開始。

要運行您的應用程序，請運行 “briefcase run android“。當您執行此操作時，系統會提示您提供可以執行該應用程式的裝置清單。最後一項始終是建立新 Android 模擬器的選項。

macOS

Linux

Windows

```
(beeware-venv) $ briefcase run android

Select device:

  1) Create a new Android emulator

>
```

```
(beeware-venv) $ briefcase run android

Select device:

  1) Create a new Android emulator

>
```

```
(beeware-venv) C:\...>briefcase run android

Select device:

  1) Create a new Android emulator

>
```

我們現在可以選擇我們想要的設備。選擇 “建立新的 Android 模擬器 “選項，🄵接受裝置名稱的預設選擇 (`beePhone`)。

公事包 “run“將自動🄵動🄵擬設備。當裝置🄵動時，您將看到 Android 標🄵：

裝置完成🄵動後，Briefcase 將在裝置上安裝您的應用程式。您將短暫地看到一個🄵動器畫面：

然後該應用程式將🄵動。應用程式🄵動時您將看到🄵動畫面：

模擬器🄵有🄵動！

Android 模擬器是一款🄵雜的軟體，依賴許多硬體和作業系統功能 - 這些功能在舊機器上可能無法使用或🄵用。如果您在🄵動 Android 模擬器時遇到任何困難，請參🄵 Android 開發人員文件的 “要求與建議 <https://developer.android.com/studio/run/emulator#requirements>‘__ 部分。

應用程式第一次🄵動時，需要將自身解壓縮到裝置上。這可能需要幾秒鐘。解壓縮後，您將看到我們桌面應用程式的 Android 版本：

如果您無法看到應用程式🄵動，您可能需要檢查執行 “briefcase run“的終端🄵查找任何錯誤訊息。

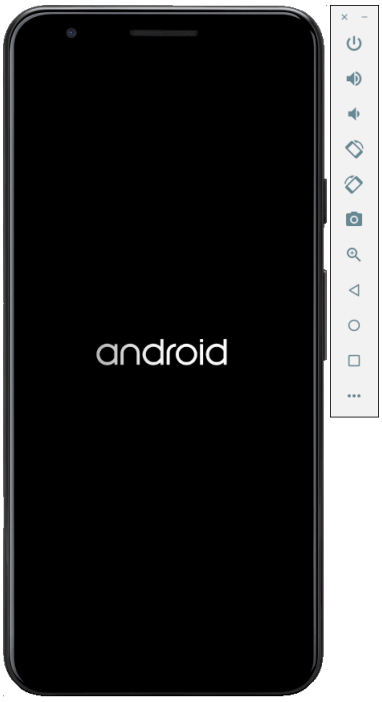


圖 1: Android 🄵擬裝置🄵動

圖 2: Android 🄵擬裝置已完全🄵動，位於🄵動器螢幕上

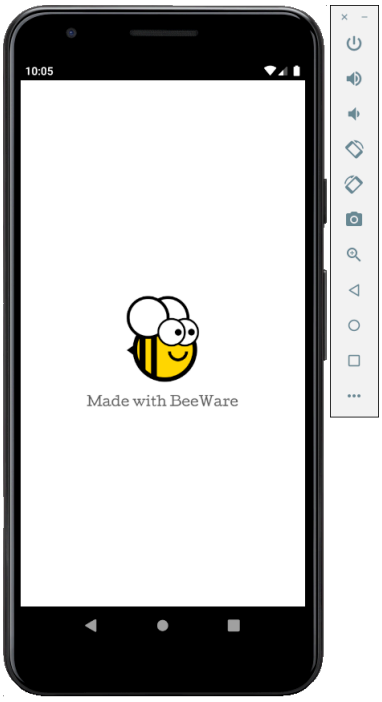


圖 3: 應用程式🄵動畫面

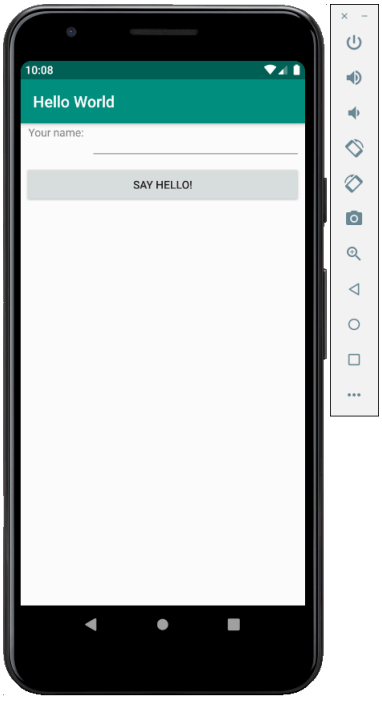


圖 4: 演示應用程式全面🄵動

將來，如果您想在該裝置上運行而不使用選單，您可以將模擬器的名稱提供給 Briefcase，使用 “briefcase run android -d @beePhone“直接在🄵擬裝置上執行。

在物理設備上運行應用程式

如果您有實體 Android 手機或平板電腦，則可以使用 USB 電纜將其連接到計算機，然後使用公文包定位您的實體設備。

Android 要求您先准備好設備，然後才能用於開發。您需要對裝置上的選項進行 2 項變更：

- 🄵用開發者選項
- 🄵用 USB 調試

有關如何進行這些更改的詳細信息，請參🄵 “Android 開發人員文件 <https://developer.android.com/studio/debug/dev-options#enable>“。

完成這些步驟後，當您執行 “briefcase run android“時，您的裝置應該會出現在可用裝置清單中。

macOS

Linux

Windows

```
(beeware-venv) $ briefcase run android

Select device:

  1) Pixel 3a (94ZZY0LNE8)
  2) @beePhone (emulator)
  3) Create a new Android emulator

>
```

```
(beeware-venv) $ briefcase run android

Select device:

  1) Pixel 3a (94ZZY0LNE8)
  2) @beePhone (emulator)
  3) Create a new Android emulator

>
```

```
(beeware-venv) C:\...>briefcase run android

Select device:

  1) Pixel 3a (94ZZY0LNE8)
  2) @beePhone (emulator)
  3) Create a new Android emulator

>
```

在這🄴，我們可以在部署清單中看到一個新的實體設備及其序號 - 在本例中🄵 Pixel 3a。將來，如果您想在該裝置上運行而不使用選單，您可以向 Briefcase 提供手機的序號（在本例中🄵 “briefcase run android -d 94ZZY0LNE8“）。這將直接在設備上運行，無需提示。

我的設備🄵有出現!

如果您的裝置完全🄵有出現在此清單中，則可能是您尚未🄵用 USB 偵錯（或裝置未插入!）。

如果您的裝置出現，但被列🄵 “未知裝置（未授權開發）“，則表示開發者模式尚未正確🄵用。重新執行 ‘🄵用開發人員選項的步驟 <https://developer.android.com/studio/debug/dev-options#enable>‘__，🄵重新執行 “briefcase run android“。

下一步

現在我們的手機上已經有一個應用程式了！我們還有其他地方可以部署 BeeWare 應用程式嗎？請參🄵*教學 6* 以了解…

2.7 教程 6 - 將其放在網路上!

除了支援行動平台外，Toga 還支援網頁！使用與部署桌面和行動應用程式相同的 API，您可以將應用程式部署🄵單頁 Web 應用程式。

概念驗證

Toga Web 後端是所有 Toga 後端中最不成熟的。它已經足🄵成熟，可以展示一些功能，但它可能有缺陷，🄵且會缺少其他平台上可用的許多小部件。此時，Web 部署應被視🄵 概念驗證 - 足以演示可以做什🄵，但不足以依賴認真的開發。

如果您對本教學的這一步有疑問，可以跳到下一頁。

2.7.1 部署🄵 Web 應用程式

部署🄵單頁 Web 應用程式的過程遵循相同的熟悉模式 - 建立應用程序，然後建立應用程序，然後運行它。然而，Briefcase 有點聰明，如果您嘗試運行應用程序，🄵且 Briefcase 確定該應用程式尚未針對目標平台創建或構建，它將🄵您執行創建和構建步驟。由於這是我們第一次運行該應用程式的網絡，我們可以使用一個命令執行所有三個步驟：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase run web

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-web-static-template.git,␣
↪branch v0.3.18
...

[helloworld] Installing support package...
No support package required.

[helloworld] Installing application code...
Installing src/helloworld... done
```

(繼續下一頁)

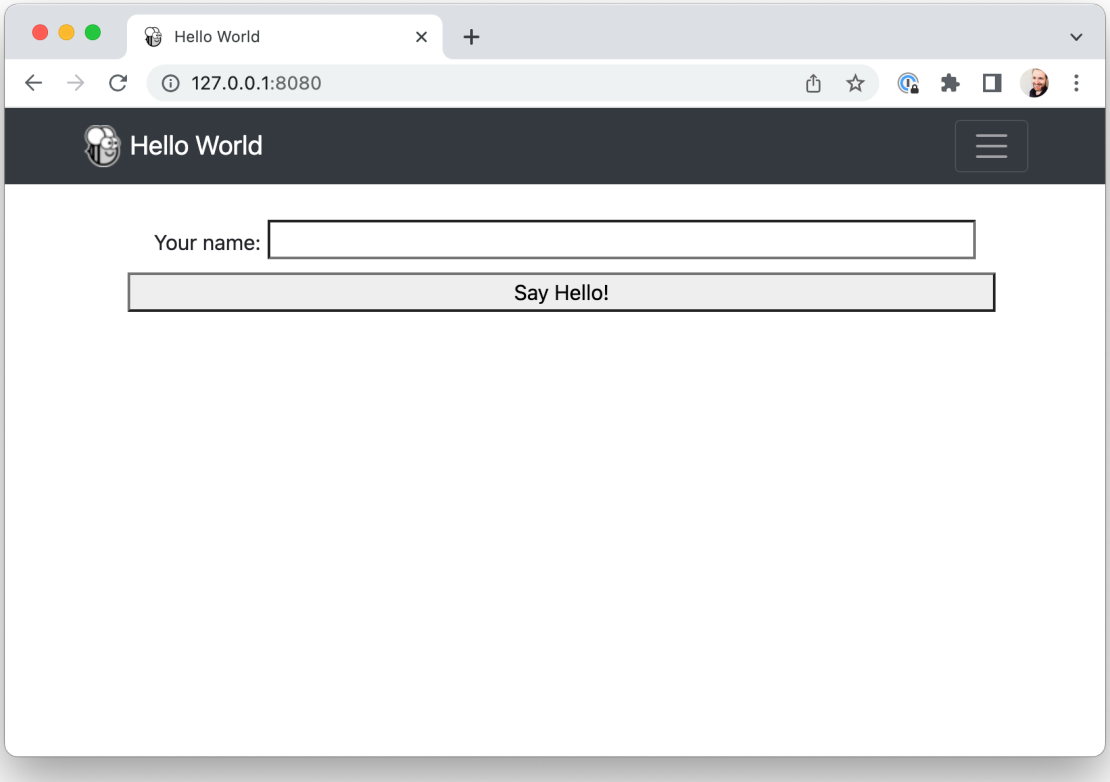
(繼續上一頁)

```
[helloworld] Installing requirements...
Writing requirements file... done

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Created build/helloworld/web/static

[helloworld] Building web project...
...

[helloworld] Built build/helloworld/web/static/www/index.html

[helloworld] Starting web server...
Web server open on http://127.0.0.1:8080

[helloworld] Web server log output (type CTRL-C to stop log)...
========================================================================
```

```
(beeware-venv) $ briefcase run web

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-web-static-template.git,␣
↪branch v0.3.18
...

[helloworld] Installing support package...
No support package required.

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
Writing requirements file... done

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Created build/helloworld/web/static

[helloworld] Building web project...
...

[helloworld] Built build/helloworld/web/static/www/index.html

[helloworld] Starting web server...
Web server open on http://127.0.0.1:8080

[helloworld] Web server log output (type CTRL-C to stop log)...
========================================================================
```

```
(beeware-venv) C:\...>briefcase run web

[helloworld] Generating application template...
Using app template: https://github.com/beeware/briefcase-web-static-template.git,␣
↪branch v0.3.18
...

[helloworld] Installing support package...
No support package required.

[helloworld] Installing application code...
Installing src/helloworld... done

[helloworld] Installing requirements...
Writing requirements file... done

[helloworld] Installing application resources...
...

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Created build\helloworld\web\static

[helloworld] Building web project...
...

[helloworld] Built build\helloworld\web\static\www\index.html

[helloworld] Starting web server...
Web server open on http://127.0.0.1:8080

[helloworld] Web server log output (type CTRL-C to stop log)...
================================================================================
```

這將開🄵一個網頁🄴覽器，指向 http://127.0.0.1:8080：



如果您輸入您的姓名🄵按一下該按鈕，將會出現一個對話方塊。

2.7.2 這個怎🄵用？

此 Web 應用程式是一個🄵態網站 - 單一 HTML 來源頁面，帶有一些 CSS 和其他資源。Briefcase 已🄵動本機 Web 伺服器來提供此頁面，以便您的🄵覽器可以查看該頁面。如果您想將此網頁投入生🄵，您可以將 `www` 資料夾的🄵容🄵🄵到任何可以提供🄵態🄵容的 Web 伺服器上。

但是當你按下按鈕時，你正在運行 Python 程式碼……這是如何運作的？Toga 使用 PyScript 在🄵覽器中提供 Python 解譯器。Briefcase 將應用程式的程式碼打包🄵 PyScript 可以在🄵覽器中載入的輪子。載入頁面後，應用程式程式碼在🄵覽器中運行，使用🄵覽器 DOM 建立 UI。當您按一下某個按鈕時，該按鈕會在🄵覽器中執行事件處理程式碼。

2.7.3 下一步

雖然我們現在已經在桌面、行動和網路上部署了這個應用程序，但該應用程式相當簡單，🄵且不涉及任何第三方程式庫。我們可以在我們的應用程式中包含 Python 套件索引 (PyPI) 中的函式庫嗎？前往教學 7 找出…

2.8 教學 7 - 🄵動此（第三方）

到目前🄵止，我們建立的應用程式只使用了我們自己的程式碼，加上 BeeWare 提供的程式碼。但是，在現實應用程式中，您可能需要使用從 Python Package Index (PyPI) 下載的第三方程式庫。

讓我們修改我們的應用程式以包含第三方程式庫。

2.8.1 存取 API

應用程式需要執行的常見任務是在 Web API 上發出請求以檢索數據，🄵向使用者顯示該數據。這是一個玩具應用程序，因此我們🄵有 * 真正的 * API 可供使用，因此我們將使用 {JSON} Placeholder API 作🄵來源資料。

{JSON} Placeholder API 有許多 “假 “API 端點，您可以將其用作測試資料。這些 API 之一是 “/posts/端點，它會傳回🄵假的部落格文章。如果您在🄵覽器中開🄵 ``https://jsonplaceholder.typicode.com/posts/42，您將獲得一個描述單一貼文的 JSON 有效負載 - 一些 Lorum ipsum ID 🄵 42 的部落格文章的🄵容。

Python 標准函式庫包含存取 API 所需的所有工具。然而，🄵建的 API 等級非常低。它們是 HTTP 協定的良好實作 - 但它們要求使用者管理大量低階細節，例如 URL 重新導向、會話、驗證和有效負載編碼。作🄵 “普通🄵覽器用🄵 “，您可能習慣於將這些詳細資訊視🄵理所當然，因🄵🄵覽器會🄵您管理這些詳細資訊。

因此，人們開發了第三方函式庫來包裝🄵建 API，🄵提供更簡單的 API，更適合日常🄵覽器體驗。我們將使用其中一個函式庫來存取 {JSON} Placeholder API - 一個名🄵 httpx 的函式庫。

讓我們🄵我們的應用程式新增一個 “httpx“ API 呼叫。將導入新增至 “app.py“頂部以導入 “httpx“:

```
import httpx
```

然後修改 “say_hello()“回調，使其看起來像這樣:

```
def say_hello(self, widget):
    with httpx.Client() as client:
        response = client.get("https://jsonplaceholder.typicode.com/posts/42")

    payload = response.json()

    self.main_window.info_dialog(
        greeting(self.name_input.value),
        payload["body"],
    )
```

這將更改 “say_hello()“回調，以便在調用它時，它將：

- 對 JSON 🄵位符 API 發出 GET 請求以取得 post 42；
- 將回應解碼🄵 JSON；
- 🄵取貼文🄵文；和
- 包括該帖子的正文作🄵對話框的文字。

讓我們在公文包開發者模式下運行更新後的應用程序，以檢查我們的更改是否有效。

macOS

Linux

Windows

```
(beeware-venv) $ briefcase dev
Traceback (most recent call last):
File ".../venv/bin/briefcase", line 5, in <module>
    from briefcase.__main__ import main
File ".../venv/lib/python3.9/site-packages/briefcase/__main__.py", line 3, in <module>
    from .cmdline import parse_cmdline
File ".../venv/lib/python3.9/site-packages/briefcase/cmdline.py", line 6, in <module>
    from briefcase.commands import DevCommand, NewCommand, UpgradeCommand
File ".../venv/lib/python3.9/site-packages/briefcase/commands/__init__.py", line 1,␣
↪in <module>
    from .build import BuildCommand  # noqa
File ".../venv/lib/python3.9/site-packages/briefcase/commands/build.py", line 5, in
↪<module>
    from .base import BaseCommand, full_options
File ".../venv/lib/python3.9/site-packages/briefcase/commands/base.py", line 14, in
↪<module>
    import httpx
ModuleNotFoundError: No module named 'httpx'
```

```
(beeware-venv) $ briefcase dev
Traceback (most recent call last):
File ".../venv/bin/briefcase", line 5, in <module>
    from briefcase.__main__ import main
File ".../venv/lib/python3.9/site-packages/briefcase/__main__.py", line 3, in <module>
    from .cmdline import parse_cmdline
File ".../venv/lib/python3.9/site-packages/briefcase/cmdline.py", line 6, in <module>
    from briefcase.commands import DevCommand, NewCommand, UpgradeCommand
File ".../venv/lib/python3.9/site-packages/briefcase/commands/__init__.py", line 1,␣
↪in <module>
    from .build import BuildCommand  # noqa
File ".../venv/lib/python3.9/site-packages/briefcase/commands/build.py", line 5, in
↪<module>
    from .base import BaseCommand, full_options
File ".../venv/lib/python3.9/site-packages/briefcase/commands/base.py", line 14, in
↪<module>
    import httpx
ModuleNotFoundError: No module named 'httpx'
```

```
(beeware-venv) C:\...>briefcase dev
Traceback (most recent call last):
File "...\venv\bin\briefcase", line 5, in <module>
    from briefcase.__main__ import main
File "...\venv\lib\python3.9\site-packages\briefcase\__main__.py", line 3, in <module>
    from .cmdline import parse_cmdline
File "...\venv\lib\python3.9\site-packages\briefcase\cmdline.py", line 6, in <module>
    from briefcase.commands import DevCommand, NewCommand, UpgradeCommand
File "...\venv\lib\python3.9\site-packages\briefcase\commands\__init__.py", line 1,␣
↪in <module>
    from .build import BuildCommand  # noqa
File "...\venv\lib\python3.9\site-packages\briefcase\commands\build.py", line 5, in
↪<module>
    from .base import BaseCommand, full_options
File "...\venv\lib\python3.9\site-packages\briefcase\commands\base.py", line 14, in
↪<module>
    import httpx
ModuleNotFoundError: No module named 'httpx'
```

發生了什🄴事？我們已將 “httpx“新增至我們的 * 程式碼 * 中，但我們尚未將其新增至我們的開發🄴擬環境。我們可以透過用 “pip“安裝 “httpx“來解🄴這個問題，然後重新執行 “briefcase dev“：

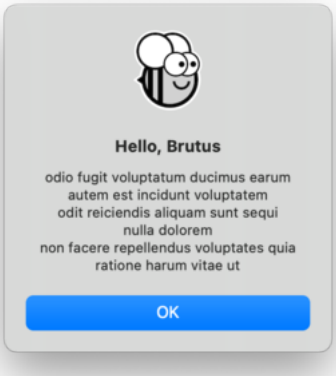
macOS

Linux

Windows

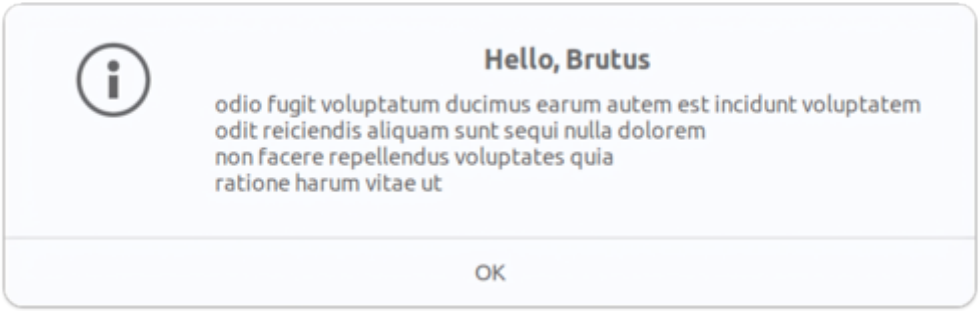
```
(beeware-venv) $ python -m pip install httpx
(beeware-venv) $ briefcase dev
```

當您輸入名稱🄴按下按鈕時，您應該會看到一個類似以下🄴容的對話框：



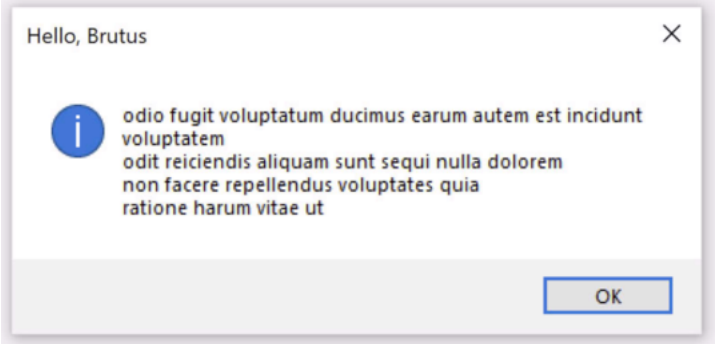

```
(beeware-venv) $ python -m pip install httpx
(beeware-venv) $ briefcase dev
```

當您輸入名稱🄴按下按鈕時，您應該會看到一個類似以下🄴容的對話框：




```
(beeware-venv) C:\...>python -m pip install httpx
(beeware-venv) C:\...>briefcase dev
```

當您輸入名稱🄴按下按鈕時，您應該會看到一個類似以下🄴容的對話框：



我們現在有了一個可以使用的應用程序，使用第三方庫，在開發模式下運行！

2.8.2 運行更新的應用程式

讓我們將此更新的應用程式程式碼打包🄴獨立應用程式。由於我們已經更改了程式碼，因此我們需要遵循教程 *4* 中的相同步驟：

macOS

Linux

Windows

更新打包應用程式中的程式碼：

```
(beeware-venv) $ briefcase update

[helloworld] Updating application code...
...

[helloworld] Application updated.
```

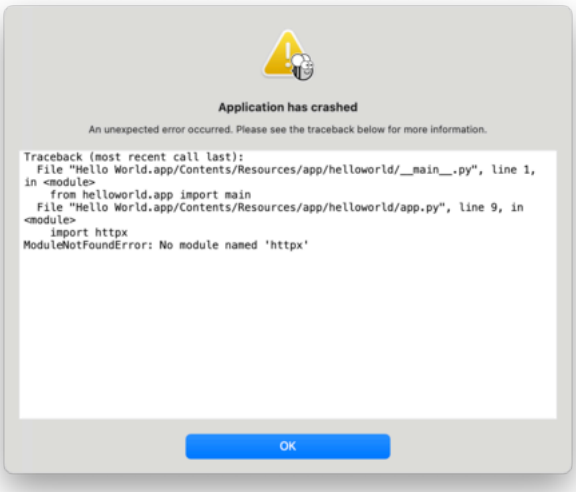
重建應用程式：

```
(beeware-venv) $ briefcase build

[helloworld] Adhoc signing app...
[helloworld] Built build/helloworld/macos/app/Hello World.app
```

最後，運行應用程式：

```
(beeware-venv) $ briefcase run

[helloworld] Starting app...
=========================================================================
```

但是，當應用程式運行時，您會在控制台中看到錯誤，以及崩潰對話框：



更新打包應用程式中的程式碼：

```
(beeware-venv) $ briefcase update

[helloworld] Updating application code...
...

[helloworld] Application updated.
```

重建應用程式：

```
(beeware-venv) $ briefcase build

[helloworld] Finalizing application configuration...
...

[helloworld] Building application...
...

[helloworld] Built build/helloworld/linux/ubuntu/jammy/helloworld-0.0.1/usr/bin/
↪helloworld
```

最後，運行應用程式：

```
(beeware-venv) $ briefcase run

[helloworld] Starting app...
================================================================================
```

但是，當應用程式運行時，您會在控制台中看到錯誤：

```
Traceback (most recent call last):
  File "/usr/lib/python3.10/runpy.py", line 194, in _run_module_as_main
    return _run_code(code, main_globals, None,
  File "/usr/lib/python3.10/runpy.py", line 87, in _run_code
    exec(code, run_globals)
  File "/home/brutus/beeware-tutorial/helloworld/build/linux/ubuntu/jammy/helloworld-
↪0.0.1/usr/app/hello_world/__main__.py", line 1, in <module>
```

(繼續下一頁)

(繼續上一頁)

```
    from helloworld.app import main
  File "/home/brutus/beeware-tutorial/helloworld/build/linux/ubuntu/jammy/helloworld-
↪0.0.1/usr/app/hello_world/app.py", line 8, in <module>
    import httpx
ModuleNotFoundError: No module named 'httpx'

Unable to start app helloworld.
```

更新打包應用程式中的程式碼：

```
(beeware-venv) C:\...>briefcase update

[helloworld] Updating application code...
...

[helloworld] Application updated.
```

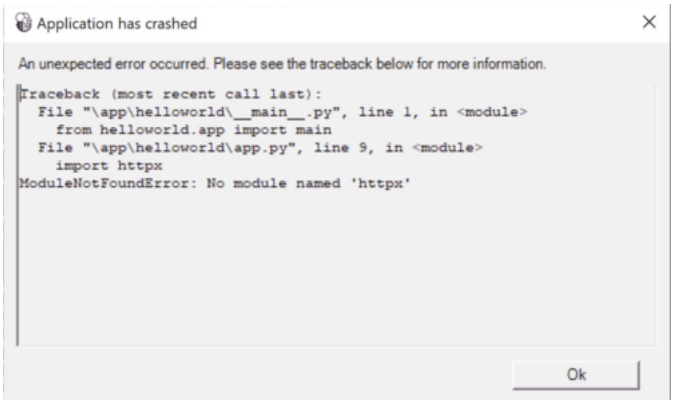
重建應用程式：

```
(beeware-venv) C:\...>briefcase build
...

[helloworld] Built build\helloworld\windows\app\src\Toga Test.exe
```

最後，運行應用程式：

```
(beeware-venv) C:\...>briefcase run

[helloworld] Starting app...
================================================================================
```

但是，當應用程式運行時，您會在控制台中看到錯誤，以及崩潰對話框：



再一次，應用程式無法🄵動，因🄵 “httpx“已安裝 - 但🄵什🄵呢？我們不是已經安裝了 “httpx“了嗎？

我們有 - 但僅限於開發環境。您的開發環境完全位於您的電腦本機 - 🄵且僅在您明確🄵動它時才🄵用。🄵管 Briefcase 有開發模式，但您使用 Briefcase 的主要原因是打包您的程式碼，以便提供給其他人。

確保其他人擁有一個包含其所需一切的 Python 環境的唯一方法是建立一個完全隔離的 Python 環境。這意味著有一個完全隔離的 Python 安裝和一組完全隔離的依賴項。這就是當您執行 “briefcase build“時，Briefcase 正在建置的🄵容——一個獨立的 Python 環境。這也解釋了🄵什🄵未安裝 “httpx“ - 它已安裝在您的 * 開發 * 環境中，但未安裝在打包的應用程式中。

因此 - 我們需要告訴 Briefcase 我們的應用程式具有外部相依性。

2.8.3 更新依賴項

在應用程式的根目🄵中，有一個名🄵 “pyproject.toml“的檔案。該檔案包含您最初執行 “briefcase new“時提供的所有應用程式設定詳細資訊。

`pyproject.toml` 被分成幾個部分；其中一節描述了您的應用程式的設定：:

```
[tool.briefcase.app.helloworld]
formal_name = "Hello World"
description = "A Tutorial app"
long_description = """More details about the app should go here.
"""
sources = ["src/helloworld"]
requires = []
```

`requires` 選項描述了我們應用程式的依賴關🄵。它是一個字串列表，指定您想要包含在應用程式中的庫（以及可選的版本）。

修改 “requires“設置，使其顯示🄵：

```
requires = [
    "httpx",
]
```

透過添加此設置，我們告訴 Briefcase“當您建立我的應用程式時，在應用程式包中執行 “pip install httpx““。任何可以作🄵 “pip install“合法輸入的🄵容都可以在這🄵使用 - 因此，您可以指定：

- 特定的庫版本（例如 “httpx==0.19.0“）；
- 一系列庫版本（例如 “httpx>=0.19“）；
- git 儲存庫的路徑（例如，`"git+https://github.com/encode/httpx"`）；或者
- 本機檔案路徑（但是 - 請注意：如果您將程式碼提供給其他人，則該路徑可能不會存在於他們的電腦上！）

在 “pyproject.toml“中，您會注意到與作業系統相關的其他部分，例如 “[tool.briefcase.app.helloworld.macOS]“和 “[tool.briefcase.app.helloworld“.windows]“ 。這些部分 * 也 * 有一個 “requires“設定。這些設定允許您定義其他特定於平台的依賴項 - 因此，例如，如果您需要特定於平台的庫來處理應用程式的某些方面，您可以在特定於平台的 “requires“部分中指定該庫，🄵且該設定將僅用於該平台。您會注意到 “toga“庫都是在特定於平台的 “requires“部分中指定的 - 這是因🄵顯示使用者介面所需的庫是特定於平台的。

在我們的例子中，我們希望在所有平台上安裝 “httpx“，因此我們使用應用程式層級 “requires“設定。應用程式級相依性將始終被安裝；除了 * 應用程式層級的依賴項之外，還會安裝特定於平台的依賴項。

某些二進位套件可能不可用

在桌面平台（macOS、Windows、Linux）上，任何可安裝的 “pip“都可以加入您的需求。在行動和網路平台上，您的選擇略有限制。

簡而言之；任何 *純 Python* 套件（即，不包含二進位模組的套件）都可以毫無困難地使用。但是，如果您的依賴項包含二進位元件，則必須對其進行編譯；目前，大多數 Python 套件不提供非桌面平台的編譯支援。

BeeWare 可以🄵一些流行的二進位模組（包括 “numpy“、`pandas``和``cryptography`）提供二進位檔案。通常可以🄵行動平台編譯軟體包，但設定起來🄵不容易——這遠遠超出了像本教程這樣的介紹性教程的範圍。

現在我們已經告訴 Briefcase 我們的附加要求，我們可以嘗試再次打包我們的應用程式。確保您已將變更儲存到 “pyproject.toml“，然後再次更新您的應用程式 - 這次傳入 “-r“標🄵。這告訴 Briefcase 更新打包應用程式中的要求：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase update -r

[helloworld] Updating application code...
Installing src/hello_world...

[helloworld] Updating requirements...
Collecting httpx
  Using cached httpx-0.27.0-py3-none-any.whl.metadata (7.2 kB)
...
Installing collected packages: zipp, typing-extensions, travertino, sniffio, pycairo,␣
↪idna, h11, exceptiongroup, certifi, pygobject, importlib-metadata, httpcore, anyio,␣
↪toga-core, httpx, gbulb, toga-gtk
Successfully installed anyio-4.3.0 certifi-2024.2.2 exceptiongroup-1.2.1 gbulb-0.6.5␣
↪h11-0.14.0 httpcore-1.0.5 httpx-0.27.0 idna-3.7 importlib-metadata-7.1.0 pycairo-1.
↪26.0 pygobject-3.48.2 sniffio-1.3.1 toga-core-0.4.4 toga-gtk-0.4.4 travertino-0.3.0␣
↪typing-extensions-4.11.0 zipp-3.18.1

[helloworld] Removing unneeded app content...
...

[helloworld] Application updated.
```

```
(beeware-venv) $ briefcase update -r

[helloworld] Finalizing application configuration...
Targeting ubuntu:jammy (Vendor base debian)
Determining glibc version... done
Targeting glibc 2.35
Targeting Python3.10

[helloworld] Updating application code...
Installing src/hello_world...

[helloworld] Updating requirements...
Collecting httpx
  Using cached httpx-0.27.0-py3-none-any.whl.metadata (7.2 kB)
        ...
Installing collected packages: zipp, typing-extensions, travertino, sniffio, pycairo,␣
↪idna, h11, exceptiongroup, certifi, pygobject, importlib-metadata, httpcore, anyio,␣
↪toga-core, httpx, gbulb, toga-gtk
Successfully installed anyio-4.3.0 certifi-2024.2.2 exceptiongroup-1.2.1 gbulb-0.6.5␣
↪h11-0.14.0 httpcore-1.0.5 httpx-0.27.0 idna-3.7 importlib-metadata-7.1.0 pycairo-1.
↪26.0 pygobject-3.48.2 sniffio-1.3.1 toga-core-0.4.4 toga-gtk-0.4.4 travertino-0.3.0␣
↪typing-extensions-4.11.0 zipp-3.18.1

[helloworld] Removing unneeded app content...
...

[helloworld] Application updated.
```

```
(beeware-venv) C:\...>briefcase update -r
```

(繼續下一頁)

(繼續上一頁)

```
[helloworld] Updating application code...
Installing src/helloworld...

[helloworld] Updating requirements...
Collecting httpx
  Using cached httpx-0.27.0-py3-none-any.whl.metadata (7.2 kB)
        ...
Installing collected packages: zipp, typing-extensions, travertino, sniffio, pycairo,␣
↪idna, h11, exceptiongroup, certifi, pygobject, importlib-metadata, httpcore, anyio,␣
↪toga-core, httpx, gbulb, toga-gtk
Successfully installed anyio-4.3.0 certifi-2024.2.2 exceptiongroup-1.2.1 gbulb-0.6.5␣
↪h11-0.14.0 httpcore-1.0.5 httpx-0.27.0 idna-3.7 importlib-metadata-7.1.0 pycairo-1.
↪26.0 pygobject-3.48.2 sniffio-1.3.1 toga-core-0.4.4 toga-gtk-0.4.4 travertino-0.3.0␣
↪typing-extensions-4.11.0 zipp-3.18.1

[helloworld] Removing unneeded app content...
...

[helloworld] Application updated.
```

更新後，您可以運行 “briefcase build“和 “briefcase run“ - 您應該會看到打包的應用程序，具有新的對話框行🄵。

備🄵： 用於更新需求的 “-r“選項也受到 “build“和 “run“命令的支持，因此如果您想一步更新、構建和運行，您可以使用 “briefcase“運行-u -r“。

2.8.4 下一步

我們現在有了一個使用第三方函式庫的應用程式！但是，您可能已經注意到，當您按下按鈕時，應用程式變得有點無響應。我們能做些什🄵來解🄵這個問題嗎？前往教學 8 找出…

2.9 教學 8 - 使其順 ~ 暢

除非您擁有 **真正**快速的網路連接，否則您可能會注意到，當您按下按鈕時，應用程式的 GUI 會鎖定一點。這是因🄵我們發出的 Web 請求是 **同步**的。當我們的應用程式發出 Web 請求時，它會等待 API 回傳回應，然後再繼續。在等待時，它 **不允許**應用程式重繪 - 結果，應用程式停止回應。

2.9.1 GUI 事件循環 (Event loop)

🄵了理解🄵什🄵會發生這種情🄵，我們需要深入研究 GUI 應用程式如何運作的細節。具體情🄵因平台而🄵；但無論您使用什🄵平台或 GUI 環境，概念都是相同的。

從根本上來🄵，GUI 應用程式是一個看起來像這樣的循環:

```
while not app.quit_requested():
    app.process_events()
    app.redraw()
```

此循環稱🄵 事件循環。(這些不是實際的方法名稱 - 它是 🄵代碼中發生的情🄵的🄵明)。

當您按一下按鈕、拖曳🄵軸或按下按鍵時，代表你🄵生一個 事件。該 事件被放入🄵列中，應用程式將在下次有機會處理事件🄵列時處理該事件。響應事件而觸發的程式碼稱🄵 `event handler` 。這些事件處理程序作🄵 `process_events()` 呼叫的一部分被呼叫。

一旦應用程式處理完所有可用事件，它將 `redraw()` GUI。這考慮了事件對應用程式顯示造成的任何變化，以及作業系統中發生的任何其他變化 - 例如，另一個應用程式的視窗可能會遮蓋或顯示我們應用程式視窗的一部分，我們的應用程式的重繪需要反映目前可見的視窗部分。

需要注意的重要細節：當應用程式正在處理事件時，它無法重繪，🄵且它無法處理其他事件。

這意味著事件處理程序中包含的任何使用者邏輯都需要快速完成。使用者將觀察到完成事件處理程序的任何延遲，因🄵 GUI 更新速度會🄵慢（或停止）。如果延遲足🄵長，您的作業系統可能會將此報告🄵問題 - macOS `beachball` 和 Windows `spinner` 圖示是作業系統告訴您您的應用程式在事件處理程序中花費的時間太長。

更新標🄵或 重新計算輸入總數等簡單操作很容易快速完成。然而，有許多操作無法快速完成。如果您正在執行🄵雜的數學計算，或對檔案系統上的所有檔案進行索引，或執行網路請求，則您無法 快速完成 - 那些操作本質上很慢。

那🄵，我們如何在 GUI 應用程式中執行耗時的操作呢?

2.9.2 非同步程式設計

我們需要的是一種方法讓耗時的 event handler 執行時告訴應用程序，只要可以從中斷的地方恢復，就可以暫時將控制權釋放回事件循環。由應用程式🄵定何時釋放它；但如果應用程式定期釋放對事件循環的控制，我們就可以擁有一個長時間運行的事件處理程序 🄵維護一個響應式 UI。

我們可以透過使用 非同步程式設計來做到這一點。非同步程式設計是一種描述程式的方式，允許解釋器同時運行多個函數，在所有🄵發運行的函數之間共用資源。

非同步函數（稱🄵 協程）需要明確宣告🄵非同步。他們還需要在🄵部聲明何時存在將上下文更改🄵另一個協程的機會。

在 Python 中，非同步程式設計是使用 `async` 和 `await` 關鍵字以及 asyncio 中的模組來實現的。標准庫。`async` 關鍵字允許我們宣告函數是非同步協同例程。`await` 關鍵字提供了一種聲明何時存在將上下文更改🄵另一個協同例程的機會的方法。asyncio 模組🄵非同步程式設計提供了一些其他有用的工具和語法。

2.9.3 使教學🄵步

要使我們的教程🄵步，請修改 `say_hello()` 事件處理程序，使其如下所示：

```
async def say_hello(self, widget):
    async with httpx.AsyncClient() as client:
        response = await client.get("https://jsonplaceholder.typicode.com/posts/42")

    payload = response.json()

    self.main_window.info_dialog(
        greeting(self.name_input.value),
        payload["body"],
    )
```

與先前的版本相比，此程式碼僅發生了 4 處變更：

1. 該方法被定義🄵 `async def` ，而不僅僅是 `def` 。這告訴 Python 該方法是一個非同步協同例程。
2. 建立的客🄵端是🄵步 `AsyncClient()` ，而不是同步 `Client()` 。這告訴 `httpx` 它應該以非同步模式運行，而不是同步模式。

3. 用於建立客🄴端的上下文管理器被標記🄴 `async` 。這告訴 Python，當進入和退出上下文管理器時，有機會釋放控制權。
4. `get` 呼叫是使用 `await` 關鍵字進行的。這指示應用程式在等待網路回應時，應用程式可以釋放對事件循環的控制。

Toga 允許您使用常規方法或非同步協同例程作🄴處理程序；Toga 管理幕後的一切，以確保根據需要呼叫或等待處理程序。

如果您儲存這些變更🄴重新執行應用程式（在開發模式下使用 `briefcase dev` ，或透過更新🄴重新執行打包的應用程式），應用程式不會有任何明顯的變更。但是，當您單擊按鈕觸發對話框時，您可能會注意到一些細微的改進：

- 該按鈕返回到 *未單擊* 狀態，而不是停留在 *單擊* 狀態。
- 「沙灘球」/」漏斗」圖示不會再出現
- 如果您在等待對話方塊出現時移動/調整應用程式視窗的大小，則該視窗將會重新繪🄴。
- 如果您嘗試開🄴應用程式選單，該選單將立即出現。

2.9.4 下一步

我們現在擁有一個流暢且響應迅速的應用程序，即使它正在等待緩慢的 API。但是，當我們繼續進一步開發應用程式時，如何確保該應用程式繼續運行？我們如何測試我們的應用程式？前往*教學 9* 找找看…

2.10 教學 9 - 測試時間

大多數軟體開發🄵不涉及編寫新程式碼，而是修改現有程式碼。確保現有程式碼繼續按照我們期望的方式運作是軟體開發過程的關鍵部分。確保我們的應用程式行🄵的一種方法是使用 *測試套件*。

2.10.1 運行測試套件

事實上，我們的專案已經有測試套件了！當我們最初🄵生專案時，🄵生了兩個資料夾：`src` 和 `tests` 。`src` 資料夾包含我們應用程式的程式碼；`tests` 資料夾包含我們的測試套件。在 `tests` 資料夾🄵有一個名🄵 `test_app.py` 的文件，其中包含以下🄴容：

```
def test_first():
    "An initial test for the app"
    assert 1 + 1 == 2
```

這是一個 Pytest 的 *測試範例* - 可以執行以驗證應用程式的某些行🄴的程式碼區塊。在本例中，測試僅是一個範例，不會測試有關我們應用程式的任何🄴容 - 但它是我們可以執行的測試。

我們可以使用 `briefcase dev` 的 `--test` 選項來執行這個測試套件。由於這是我們第一次執行測試，我們還需要傳入 `-r` 選項以確保測試要求也已安裝：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase dev --test -r

[helloworld] Installing requirements...
...
Installing dev requirements... done

[helloworld] Running test suite in dev environment...
=============================================================================
============================= test session starts ==============================
platform darwin -- Python 3.11.0, pytest-7.2.0, pluggy-1.0.0 -- /Users/brutus/beeware-
↪tutorial/beeware-venv/bin/python3.11
cachedir: /var/folders/b_/khqk71xd45d049kxc_59ltp80000gn/T/.pytest_cache
rootdir: /Users/brutus
plugins: anyio-3.6.2
collecting ... collected 1 item

tests/test_app.py::test_first PASSED                                     [100%]

============================== 1 passed in 0.01s ===============================
```

```
(beeware-venv) $ briefcase dev --test -r

[helloworld] Installing requirements...
...
Installing dev requirements... done

[helloworld] Running test suite in dev environment...
=============================================================================
============================= test session starts ==============================
platform linux -- Python 3.11.0
pytest==7.2.0
py==1.11.0
pluggy==1.0.0
cachedir: /tmp/.pytest_cache
rootdir: /home/brutus
plugins: anyio-3.6.2
collecting ... collected 1 item

tests/test_app.py::test_first PASSED                                     [100%]

============================== 1 passed in 0.01s ===============================
```

```
(beeware-venv) C:\...>briefcase dev --test -r

[helloworld] Installing requirements...
...
Installing dev requirements... done

[helloworld] Running test suite in dev environment...
=============================================================================
============================= test session starts ==============================
platform win32 -- Python 3.11.0
pytest==7.2.0
py==1.11.0
pluggy==1.0.0
cachedir: C:\Users\brutus\AppData\Local\Temp\.pytest_cache
rootdir: C:\Users\brutus
```

(繼續下一頁)

(繼續上一頁)

```
plugins: anyio-3.6.2
collecting ... collected 1 item

tests/test_app.py::test_first PASSED                                     [100%]

============================== 1 passed in 0.01s ===============================
```

成功了！我們剛剛執行了一個測試，證實了 Python 的數學按照我們期望的方式工作（真是松了一口氣!）。

讓我們用一個測試來取代這個範例，以驗證我們的 `greeting()` 方法的行🄵是否符合我們的預期。將 `test_app.py` 的🄵容替🄵🄵以下🄵容：

```
from helloworld.app import greeting


def test_name():
    """If a name is provided, the greeting includes the name"""

    assert greeting("Alice") == "Hello, Alice"


def test_empty():
    """If a name is not provided, a generic greeting is provided"""

    assert greeting("") == "Hello, stranger"
```

這定義了兩個新的測試，驗證我們期望看到的兩個行🄵：提供名稱時的輸出，以及名稱🄴空時的輸出。

我們現在可以重新運行測試套件。這次，我們不需要提供 `-r` 選項，因🄴測試要求已經安裝了；我們只需要使用 `--test` 選項：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase dev --test

[helloworld] Running test suite in dev environment...
================================================================================
============================== test session starts ==============================
...
collecting ... collected 2 items

tests/test_app.py::test_name PASSED                                      [ 50%]
tests/test_app.py::test_empty PASSED                                     [100%]

============================== 2 passed in 0.11s ===============================
```

```
(beeware-venv) $ briefcase dev --test

[helloworld] Running test suite in dev environment...
================================================================================
============================== test session starts ==============================
...
collecting ... collected 2 items
```

(繼續下一頁)

(繼續上一頁)

```
tests/test_app.py::test_name PASSED                                      [ 50%]
tests/test_app.py::test_empty PASSED                                     [100%]

============================== 2 passed in 0.11s ===============================
```

```
(beeware-venv) C:\...>briefcase dev --test

[helloworld] Running test suite in dev environment...
================================================================================
============================= test session starts ==============================
...
collecting ... collected 2 items

tests/test_app.py::test_name PASSED                                      [ 50%]
tests/test_app.py::test_empty PASSED                                     [100%]

============================== 2 passed in 0.11s ===============================
```

非常好！我們的 `greeting()` 實用方法按預期工作。

2.10.2 用測試來驅動開發

現在我們有了測試套件，我們可以用它來驅動新功能的開發。讓我們修改我們的應用程序，🄵某個特定用🄵提供特殊的問候語。我們可以先🄴我們希望在 `test_app.py` 底部看到的新行🄴新增一個測試案例：

```
def test_brutus():
    """If the name is Brutus, a special greeting is provided"""

    assert greeting("Brutus") == "BeeWare the IDEs of Python!"
```

然後，使用這個新測試來執行測試套件：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase dev --test

[helloworld] Running test suite in dev environment...
================================================================================
============================= test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED                                      [ 33%]
tests/test_app.py::test_empty PASSED                                     [ 66%]
tests/test_app.py::test_brutus FAILED                                    [100%]

================================== FAILURES ===================================
_____________________________________ test_brutus _____________________________________

    def test_brutus():
        """If the name is Brutus, a special greeting is provided"""
```

(繼續下一頁)

(繼續上一頁)

```
>       assert greeting("Brutus") == "BeeWare the IDEs of Python!"
E       AssertionError: assert 'Hello, Brutus' == 'BeeWare the IDEs of Python!'
E         - BeeWare the IDEs of Python!
E         + Hello, Brutus

tests/test_app.py:19: AssertionError
========================== short test summary info ===========================
FAILED tests/test_app.py::test_brutus - AssertionError: assert 'Hello, Brutus...
========================= 1 failed, 2 passed in 0.14s ========================
```

```
(beeware-venv) $ briefcase dev --test

[helloworld] Running test suite in dev environment...
=============================================================================
============================= test session starts ============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED                                      [ 33%]
tests/test_app.py::test_empty PASSED                                     [ 66%]
tests/test_app.py::test_brutus FAILED                                    [100%]

================================== FAILURES ==================================
_____________________________________ test_brutus ____________________________

    def test_brutus():
        """If the name is Brutus, provide a special greeting"""

>       assert greeting("Brutus") == "BeeWare the IDEs of Python!"
E       AssertionError: assert 'Hello, Brutus' == 'BeeWare the IDEs of Python!'
E         - BeeWare the IDEs of Python!
E         + Hello, Brutus

tests/test_app.py:19: AssertionError
========================== short test summary info ===========================
FAILED tests/test_app.py::test_brutus - AssertionError: assert 'Hello, Brutus...
========================= 1 failed, 2 passed in 0.14s ========================

============================== 2 passed in 0.11s =============================
```

```
(beeware-venv) C:\...>briefcase dev --test

[helloworld] Running test suite in dev environment...
=============================================================================
============================= test session starts ============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED                                      [ 33%]
tests/test_app.py::test_empty PASSED                                     [ 66%]
tests/test_app.py::test_brutus FAILED                                    [100%]

================================== FAILURES ==================================
_____________________________________ test_brutus ____________________________

    def test_brutus():
```

(繼續下一頁)

(繼續上一頁)

```
        """If the name is Brutus, provide a special greeting"""

>       assert greeting("Brutus") == "BeeWare the IDEs of Python!"
E       AssertionError: assert 'Hello, Brutus' == 'BeeWare the IDEs of Python!'
E         - BeeWare the IDEs of Python!
E         + Hello, Brutus

tests/test_app.py:19: AssertionError
========================== short test summary info ===========================
FAILED tests/test_app.py::test_brutus - AssertionError: assert 'Hello, Brutus...
========================== 1 failed, 2 passed in 0.14s ==========================
```

這次，我們看到測試失敗 - 輸出解釋了失敗的根源：測試期望輸出 `BeeWare the IDEs of Python!`，但我們的 `greeting()` 實現返回 `Hello, Brutus`。讓我們修改 src/helloworld/app.py 中的 greeting() 實作以獲得新的行🄵：

```
def greeting(name):
    if name:
        if name == "Brutus":
            return "BeeWare the IDEs of Python!"
        else:
            return f"Hello, {name}"
    else:
        return "Hello, stranger"
```

如果我們再次運行測試，我們現在將看到測試通過：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase dev --test

[helloworld] Running test suite in dev environment...
==============================================================================
============================= test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED                                      [ 33%]
tests/test_app.py::test_empty PASSED                                     [ 66%]
tests/test_app.py::test_brutus PASSED                                    [100%]

============================== 3 passed in 0.15s ===============================
```

```
(beeware-venv) $ briefcase dev --test

[helloworld] Running test suite in dev environment...
==============================================================================
============================= test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED                                      [ 33%]
tests/test_app.py::test_empty PASSED                                     [ 66%]
```

(繼續下一頁)

(繼續上一頁)

```
tests/test_app.py::test_brutus PASSED                                    [100%]

============================== 3 passed in 0.15s ===============================
```

```
(beeware-venv) C:\...>briefcase dev --test

[helloworld] Running test suite in dev environment...
=============================================================================
============================== test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED                                      [ 33%]
tests/test_app.py::test_empty PASSED                                     [ 66%]
tests/test_app.py::test_brutus PASSED                                    [100%]

============================== 3 passed in 0.15s ===============================
```

2.10.3 運行時測試

到目前🄵止，我們一直在開發模式下執行測試。當您開發新功能時，這特🄵有用，因🄴您可以快速迭代添加測試，🄴添加程式碼以使這些測試通過。但是，在某些時候，您需要驗證您的程式碼在部署的環境中是否也可以正確運行。

`--test` 和 `-r` 選項也可以傳遞給 `run` 指令。如果您使用 `briefcase run --test -r`，將運行相同的測試套件，但它將在部署的環境中運行，而不是在您的開發環境中運行：

macOS

Linux

Windows

```
(beeware-venv) $ briefcase run --test -r

[helloworld] Updating application code...
Installing src/helloworld... done
Installing tests... done

[helloworld] Updating requirements...
...
[helloworld] Built build/helloworld/macos/app/Hello World.app (test mode)

[helloworld] Starting test suite...
=============================================================================
Configuring isolated Python...
Pre-initializing Python runtime...
PythonHome: /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello␣
↪World.app/Contents/Resources/support/python-stdlib
PYTHONPATH:
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/support/python311.zip
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/support/python-stdlib
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
```

(繼續下一頁)

(繼續上一頁)

```
↪Contents/Resources/support/python-stdlib/lib-dynload
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/app_packages
- /Users/brutus/beeware-tutorial/helloworld/macOS/app/Hello World/Hello World.app/
↪Contents/Resources/app
Configure argc/argv...
Initializing Python runtime...
Installing Python NSLog handler...
Running app module: tests.helloworld
--------------------------------------------------------------------------------
============================= test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED [ 33%]
tests/test_app.py::test_empty PASSED [ 66%]
tests/test_app.py::test_brutus PASSED [100%]

============================== 3 passed in 0.21s ===============================

[helloworld] Test suite passed!
```

```
(beeware-venv) $ briefcase run --test -r

[helloworld] Finalizing application configuration...
Targeting ubuntu:jammy (Vendor base debian)
Determining glibc version... done
Targeting glibc 2.35
Targeting Python3.10

[helloworld] Updating application code...
Installing src/helloworld... done
Installing tests... done

[helloworld] Updating requirements...
...
[helloworld] Built build/helloworld/linux/ubuntu/jammy/helloworld-0.0.1/usr/bin/
↪helloworld (test mode)

[helloworld] Starting test suite...
================================================================================
============================= test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED [ 33%]
tests/test_app.py::test_empty PASSED [ 66%]
tests/test_app.py::test_brutus PASSED [100%]

============================== 3 passed in 0.21s ===============================
```

```
(beeware-venv) C:\...>briefcase run --test -r

[helloworld] Updating application code...
Installing src/helloworld... done
Installing tests... done
```

(繼續下一頁)

(繼續上一頁)

```
[helloworld] Updating requirements...
...
[helloworld] Built build\helloworld\windows\app\src\Hello World.exe (test mode)

=============================================================================
Log started: 2022-12-02 10:57:34Z
PreInitializing Python runtime...
PythonHome: C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src
PYTHONPATH:
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src\python311.
↪zip
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src\app_packages
- C:\Users\brutus\beeware-tutorial\helloworld\windows\app\Hello World\src\app
Configure argc/argv...
Initializing Python runtime...
Running app module: tests.helloworld
-----------------------------------------------------------------------------
============================= test session starts ==============================
...
collecting ... collected 3 items

tests/test_app.py::test_name PASSED [ 33%]
tests/test_app.py::test_empty PASSED [ 66%]
tests/test_app.py::test_brutus PASSED [100%]

============================== 3 passed in 0.21s ===============================
```

與 `briefcase dev --test` 一樣，僅在第一次執行測試套件時才需要 `-r` 選項，以確保測試依賴項存在。在後續運行中，您可以忽略此選項。

您還可以在移動環境下使用 `--test` 選項：- 因此 `briefcase run iOS --test` 和 `briefcase run android --test` 都可以工作，在您選擇的移動設備上運行測試套件。

2.10.4 下一步

We've now got a a test suite for our application. But it still looks like a tutorial app. Is there anything we can do about that? Turn to *Tutorial 10* to find out···

2.11 教程 10 - 打造您自己的應用程式

到目前🄴止，我們的應用程式使用了預設的 灰色蜜蜂圖示。我們如何更新應用程式以使用我們自己的圖標?

2.11.1 新增圖示

Every platform uses a different format for application icons - and some platforms need *multiple* icons in different sizes and shapes. To account for this, Briefcase provides a shorthand way to configure an icon once, and then have that definition expand in to all the different icons needed for each individual platform.

Edit your `pyproject.toml`, adding a new `icon` configuration item in the `[tool.briefcase.app.helloworld]` configuration section, just above the `sources` definition:

```
icon = "icons/helloworld"
```

This icon definition doesn't specify any file extension. The value will be used as a prefix; each platform will add additional items to this prefix to build the files needed for each platform. Some platforms require *multiple* icon files; this prefix will be combined with file size and variant modifiers to generate the list of icon files that are used.

We can now run `briefcase update` again - but this time, we pass in the `--update-resources` flag, telling Briefcase that we want to install new application resources (i.e., the icons):

macOS

Linux

Windows

Android

iOS

```
(beeware-venv) $ briefcase update --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Unable to find icons/helloworld.icns for application icon; using default

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.
```

```
(beeware-venv) $ briefcase update --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Unable to find icons/helloworld-16.png for 16px application icon; using default
Unable to find icons/helloworld-32.png for 32px application icon; using default
Unable to find icons/helloworld-64.png for 64px application icon; using default
Unable to find icons/helloworld-128.png for 128px application icon; using default
Unable to find icons/helloworld-256.png for 256px application icon; using default
Unable to find icons/helloworld-512.png for 512px application icon; using default
```

(繼續下一頁)

(繼續上一頁)

```
[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.
```

```
(beeware-venv) C:\...>briefcase update --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Unable to find icons/helloworld.ico for application icon; using default

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.
```

```
(beeware-venv) $ briefcase update android --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Unable to find icons/helloworld-round-48.png for 48px round application icon; using␣
↪default
Unable to find icons/helloworld-round-72.png for 72px round application icon; using␣
↪default
Unable to find icons/helloworld-round-96.png for 96px round application icon; using␣
↪default
Unable to find icons/helloworld-round-144.png for 144px round application icon; using␣
↪default
Unable to find icons/helloworld-round-192.png for 192px round application icon; using␣
↪default
Unable to find icons/helloworld-square-48.png for 48px square application icon; using␣
↪default
Unable to find icons/helloworld-square-72.png for 72px square application icon; using␣
↪default
Unable to find icons/helloworld-square-96.png for 96px square application icon; using␣
↪default
Unable to find icons/helloworld-square-144.png for 144px square application icon;␣
↪using default
Unable to find icons/helloworld-square-192.png for 192px square application icon;␣
↪using default
Unable to find icons/helloworld-square-320.png for 320px square application icon;␣
↪using default
Unable to find icons/helloworld-square-480.png for 480px square application icon;␣
↪using default
Unable to find icons/helloworld-square-640.png for 640px square application icon;␣
↪using default
Unable to find icons/helloworld-square-960.png for 960px square application icon;␣
↪using default
Unable to find icons/helloworld-square-1280.png for 1280px square application icon;␣
↪using default
Unable to find icons/helloworld-adaptive-108.png for 108px adaptive application icon;␣
```

(繼續下一頁)

(繼續上一頁)

```
↪using default
Unable to find icons/helloworld-adaptive-162.png for 162px adaptive application icon;␣
↪using default
Unable to find icons/helloworld-adaptive-216.png for 216px adaptive application icon;␣
↪using default
Unable to find icons/helloworld-adaptive-324.png for 324px adaptive application icon;␣
↪using default
Unable to find icons/helloworld-adaptive-432.png for 432px adaptive application icon;␣
↪using default

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.
```

```
(beeware-venv) $ briefcase iOS --update-resources

  [helloworld] Updating application code...
  Installing src/helloworld... done

  [helloworld] Updating application resources...
  Unable to find icons/helloworld-20.png for 20px application icon; using default
  Unable to find icons/helloworld-29.png for 29px application icon; using default
  Unable to find icons/helloworld-40.png for 40px application icon; using default
  Unable to find icons/helloworld-58.png for 58px application icon; using default
  Unable to find icons/helloworld-60.png for 60px application icon; using default
  Unable to find icons/helloworld-76.png for 76px application icon; using default
  Unable to find icons/helloworld-80.png for 80px application icon; using default
  Unable to find icons/helloworld-87.png for 87px application icon; using default
  Unable to find icons/helloworld-120.png for 120px application icon; using default
  Unable to find icons/helloworld-152.png for 152px application icon; using default
  Unable to find icons/helloworld-167.png for 167px application icon; using default
  Unable to find icons/helloworld-180.png for 180px application icon; using default
  Unable to find icons/helloworld-640.png for 640px application icon; using default
  Unable to find icons/helloworld-1024.png for 1024px application icon; using default
  Unable to find icons/helloworld-1280.png for 1280px application icon; using default
  Unable to find icons/helloworld-1920.png for 1920px application icon; using default

  [helloworld] Removing unneeded app content...
  Removing unneeded app bundle content... done

  [helloworld] Application updated.
```

This reports the specific icon file (or files) that Briefcase is expecting. However, as we haven't provided the actual icon files, the install fails, and Briefcase falls back to a default value (the same 「gray bee」 icon).

Let's provide some actual icons. Download `this icons.zip bundle`, and unpack it into the root of your project directory. After unpacking, your project directory should look something like:

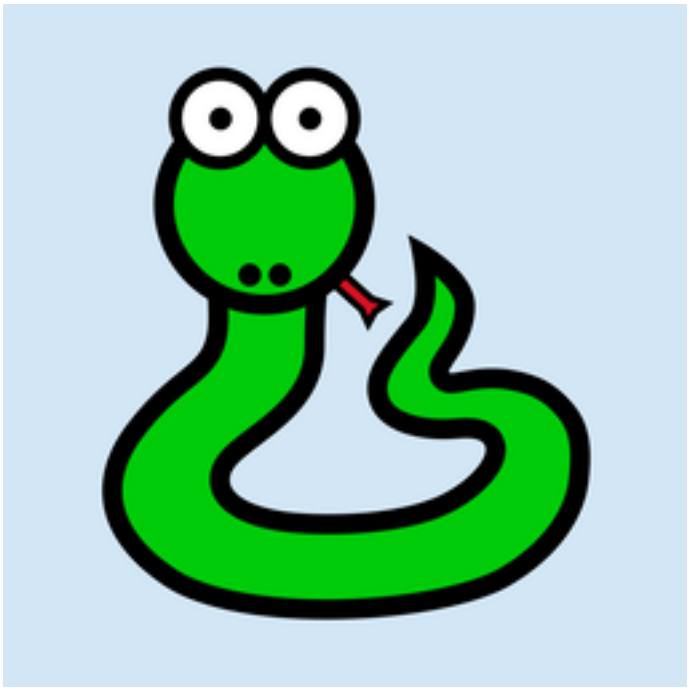
```
beeware-tutorial/
├── beeware-venv/
│   └── ...
└── helloworld/
    ├── ...
    └── pyproject.toml
        ├── icons/
```

(繼續下一頁)

(繼續上一頁)

```
        │   ├── helloworld.icns
        │   ├── helloworld.ico
        │   ├── helloworld.png
        │   ├── helloworld-16.png
        │   ├── ...
        └── src/
            └── ...
```

There's a *lot* of icons in this folder, but most of them should look the same: a green snake on a light blue background:



The only exception will be the icons with `-adaptive-` in their name; these will have a transparent background. This represents all the different icon sizes and shapes you need to support an app on every platform that Briefcase supports.

Now that we have icons, we can update the application again. However, `briefcase update` will only copy the updated resources into the build directory; we also want to rebuild the app to make sure the new icon is included, then start the app. We can shortcut this process by passing `--update-resources` to our call to `run` - this will update the app, update the app's resources, and then start the app:

macOS

Linux

Windows

Android

iOS

```
(beeware-venv) $ briefcase run --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Installing icons/helloworld.icns as application icon... done
```

(繼續下一頁)

(繼續上一頁)

```
[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Ad-hoc signing app...
     ———————————————————————————————————————————— 100.0% • 00:01

[helloworld] Built build/helloworld/macos/app/Hello World.app

[helloworld] Starting app...
```

```
(beeware-venv) $ briefcase run --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Installing icons/helloworld-16.png as 16px application icon... done
Installing icons/helloworld-32.png as 32px application icon... done
Installing icons/helloworld-64.png as 64px application icon... done
Installing icons/helloworld-128.png as 128px application icon... done
Installing icons/helloworld-256.png as 256px application icon... done
Installing icons/helloworld-512.png as 512px application icon... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Building application...
Build bootstrap binary...
...

[helloworld] Built build/helloworld/linux/ubuntu/jammy/helloworld-0.0.1/usr/bin/
↪helloworld

[helloworld] Starting app...
```

```
(beeware-venv) C:\...>briefcase build --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Installing icons/helloworld.ico as application icon... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Building App...
Removing any digital signatures from stub app... done
Setting stub app details... done
```

(繼續下一頁)

(繼續上一頁)

```
[helloworld] Built build\helloworld\windows\app\src\Hello World.exe

[helloworld] Starting app...
```

```
(beeware-venv) $ briefcase build android --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Installing icons/helloworld-round-48.png as 48px round application icon... done
Installing icons/helloworld-round-72.png as 72px round application icon... done
Installing icons/helloworld-round-96.png as 96px round application icon... done
Installing icons/helloworld-round-144.png as 144px round application icon... done
Installing icons/helloworld-round-192.png as 192px round application icon... done
Installing icons/helloworld-square-48.png as 48px square application icon... done
Installing icons/helloworld-square-72.png as 72px square application icon... done
Installing icons/helloworld-square-96.png as 96px square application icon... done
Installing icons/helloworld-square-144.png as 144px square application icon... done
Installing icons/helloworld-square-192.png as 192px square application icon... done
Installing icons/helloworld-square-320.png as 320px square application icon... done
Installing icons/helloworld-square-480.png as 480px square application icon... done
Installing icons/helloworld-square-640.png as 640px square application icon... done
Installing icons/helloworld-square-960.png as 960px square application icon... done
Installing icons/helloworld-square-1280.png as 1280px square application icon... done
Installing icons/helloworld-adaptive-108.png as 108px adaptive application icon...␣
↪done
Installing icons/helloworld-adaptive-162.png as 162px adaptive application icon...␣
↪done
Installing icons/helloworld-adaptive-216.png as 216px adaptive application icon...␣
↪done
Installing icons/helloworld-adaptive-324.png as 324px adaptive application icon...␣
↪done
Installing icons/helloworld-adaptive-432.png as 432px adaptive application icon...␣
↪done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Starting app...
```

備🄵: If you're using a recent version of Android, you may notice that the app icon has been changed to a green snake, but the background of the icon is *white*, rather than light blue. We'll fix this in the next step.

```
(beeware-venv) $ briefcase build iOS --update-resources

[helloworld] Updating application code...
Installing src/helloworld... done

[helloworld] Updating application resources...
Installing icons/helloworld-20.png as 20px application icon... done
```

(繼續下一頁)

(繼續上一頁)

```
Installing icons/helloworld-29.png as 29px application icon... done
Installing icons/helloworld-40.png as 40px application icon... done
Installing icons/helloworld-58.png as 58px application icon... done
Installing icons/helloworld-60.png as 60px application icon... done
Installing icons/helloworld-76.png as 76px application icon... done
Installing icons/helloworld-80.png as 80px application icon... done
Installing icons/helloworld-87.png as 87px application icon... done
Installing icons/helloworld-120.png as 120px application icon... done
Installing icons/helloworld-152.png as 152px application icon... done
Installing icons/helloworld-167.png as 167px application icon... done
Installing icons/helloworld-180.png as 180px application icon... done
Installing icons/helloworld-640.png as 640px application icon... done
Installing icons/helloworld-1024.png as 1024px application icon... done
Installing icons/helloworld-1280.png as 1280px application icon... done
Installing icons/helloworld-1920.png as 1920px application icon... done

[helloworld] Removing unneeded app content...
Removing unneeded app bundle content... done

[helloworld] Application updated.

[helloworld] Starting app...
```

When you run the app on iOS or Android, in addition to the icon change, you should also notice that the splash screen incorporates the new icon. However, the light blue background of the icon looks a little out of place against the white background of the splash screen. We can fix this by customizing the background color of the splash screen. Add the following definition to your `pyproject.toml`, just after the `icon` definition:

```
splash_background_color = "#D3E6F5"
```

Unfortunately, Briefcase isn't able to apply this change to an already generated project, as it requires making modifications to one of the files that was generated during the original call to `briefcase create`. To apply this change, we have to re-create the app by re-running `briefcase create`. When we do this, we'll be prompted to confirm that we want to overwrite the existing project:

macOS

Linux

Windows

Android

iOS

```
(beeware-venv) $ briefcase create

Application 'helloworld' already exists; overwrite [y/N]? y

[helloworld] Removing old application bundle...

[helloworld] Generating application template...
...

[helloworld] Created build/helloworld/macos/app
```

```
(beeware-venv) $ briefcase create

Application 'helloworld' already exists; overwrite [y/N]? y

[helloworld] Removing old application bundle...

[helloworld] Generating application template...
...

[helloworld] Created build/helloworld/linux/ubuntu/jammy
```

```
(beeware-venv) C:\...>briefcase create

Application 'helloworld' already exists; overwrite [y/N]? y

[helloworld] Removing old application bundle...

[helloworld] Generating application template...
...

[helloworld] Created build\helloworld\windows\app
```

```
(beeware-venv) $ briefcase create android

Application 'helloworld' already exists; overwrite [y/N]? y

[helloworld] Removing old application bundle...

[helloworld] Generating application template...
...
[helloworld] Created build/helloworld/android/gradle
```

```
(beeware-venv) $ briefcase create iOS

Application 'helloworld' already exists; overwrite [y/N]? y

[helloworld] Removing old application bundle...

[helloworld] Generating application template...
...
[helloworld] Created build/helloworld/ios/xcode
```

You can then re-build and re-run the app using `briefcase run`. You won't notice any changes to the desktop app; but the Android or iOS apps should now have a light blue splash screen background.

You'll need to re-create the app like this whenever you make a change to your `pyproject.toml` that doesn't relate to source code or dependencies. Any change to descriptions, version numbers, colors, or permissions will require a re-create step. Because of this, while you are developing your project, you shouldn't make any manual changes to the contents of the `build` folder, and you shouldn't add the `build` folder to your version control system. The `build` folder should be considered entirely ephemeral - an output of the build system that can be recreated as needed to reflect the current configuration of your project.

2.11.2 下一步

This has been a taste for what you can do with the tools provided by the BeeWare project. What you do from here is up to you!

Some places to go from here:

- Tutorials demonstrating features of the Toga widget toolkit.
- Details on the options available when configuring your Briefcase project.