**wxPython foundation in research technical**

**wxPython căn bản**

* **Support Document**: https://www.geeksforgeeks.org/python-wxpython-module-introduction/
* **Tài liệu chính thức của wxFormBuilder**: https://github.com/wxFormBuilder/wxFormBuilder/wiki
* **GUI tool :** https://www.tutorialspoint.com/wxpython/wxpython\_gui\_builder\_tools.htm

A screenshot of a computer

Description automatically generated

A white text on a black background

Description automatically generated

How do I add objects and subwindows to a frame?

How can I use common dialogs?

Các lỗi thường gặp với đối tượng ứng dụng và khung giao diện trong wxPython

|  |  |  |  |
| --- | --- | --- | --- |
| No | Symptom | Cause | Solution |
| 1 | Error message at startup saying “unable to import module wx.” | Mô-đun wxPython không có trong đường dẫn PYTHONPATH. Điều này có nghĩa là wxPython có thể chưa được cài đặt đúng cách. Hoặc, nếu có nhiều phiên bản Python trên hệ thống, wxPython có thể đã được cài đặt cho phiên bản khác với phiên bản bạn đang sử dụng. | Xác định phiên bản Python nào đang được cài đặt trên máy bạn. Trên hệ thống Unix, lệnh which python sẽ cho bạn biết cài đặt mặc định. Trên hệ thống Windows, bạn có thể phải vào tùy chọn thư mục để xem ứng dụng nào đang được gán cho file .py. Nếu wxPython được cài đặt đúng cho phiên bản Python đó, nó sẽ nằm trong thư mục <python-home>/Lib/site-packages. Bạn có thể cần cài đặt hoặc cài đặt lại wxPython. |
| 2 | The application crashes immediately on startup, or a blank window is dis  played, followed immediately by an application crash.  -Ứng dụng bị treo ngay khi khởi động, hoặc một cửa sổ trống xuất hiện và ngay sau đó ứng dụng bị treo. | Một đối tượng wxPython được tạo hoặc sử dụng trước khi tạo đối tượng wx.App | Tạo đối tượng wx.App ngay khi bắt đầu chạy script của bạn. |
| 3 | My top-level windows are created and immediately close. The application  exits immediately -Các cửa sổ cấp cao nhất của tôi được tạo ra và ngay lập tức đóng lại. Ứng dụng kết thúc ngay lập tức. | Phương thức wx.App.MainLoop() chưa được gọi. | Gọi phương thức MainLoop() sau khi bạn hoàn thành tất cả các bước thiết lập. |
| 4 | My top-level windows are created and immediately close. The application  exits immediately, and I did call MainLoop() - Các cửa sổ cấp cao nhất của tôi được tạo ra và ngay lập tức đóng lại. Ứng dụng kết thúc ngay lập tức, và tôi đã gọi MainLoop(). | Có lỗi trong phương thức OnInit() của ứng dụng của bạn, hoặc một phương thức nào đó được gọi bởi nó (như phương thức \_\_init\_\_() của frame). | Một lỗi trước khi MainLoop() được gọi có thể kích hoạt một ngoại lệ khiến chương trình thoát. Nếu ứng dụng của bạn được thiết lập để chuyển hướng stdout và stderr sang các cửa sổ, thì các cửa sổ đó sẽ không hiển thị đủ lâu để bạn thấy thông báo lỗi. Tạo đối tượng ứng dụng của bạn với tùy chọn redirect=False để cho phép bạn xem thông báo lỗi. Xem phần “Redirecting Output” để biết thêm chi tiết. |

**Working in an event-driven environment ( Làm việc với EDA )**

Review về vấn đề so sánh hiệu quả của hệ thống EDA và DDD ? [**Here**](EDA_wxPythonandDDD.xlsx)

|  |
| --- |
| What terminology do I need to understand events? |
|  |

**Lập trình điều khiển sự kiện là gì?**

Lập trình điều khiển sự kiện (event-driven programming) là một kiểu cấu trúc chương trình mà trong đó chương trình chờ đợi và phản hồi lại các sự kiện. Cấu trúc của một chương trình wxPython (hoặc bất kỳ chương trình điều khiển sự kiện nào) khác cơ bản so với một kịch bản Python thông thường. Một kịch bản Python thông thường có điểm bắt đầu và điểm kết thúc cụ thể, và lập trình viên kiểm soát thứ tự thực thi bằng cách sử dụng các cấu trúc điều kiện, vòng lặp và hàm. Chương trình không phải là tuyến tính, nhưng thứ tự thực thi thường không phụ thuộc vào hành động của người dùng.

**Cách hoạt động của chương trình wxPython từ góc nhìn của người dùng**

Từ góc nhìn của người dùng, một chương trình wxPython thường dành nhiều thời gian để "không làm gì cả". Thông thường, chương trình sẽ ở trạng thái chờ cho đến khi người dùng hoặc hệ thống làm điều gì đó để kích hoạt chương trình wxPython hoạt động. Cấu trúc chương trình wxPython là một ví dụ của kiến trúc chương trình điều khiển sự kiện.

**Minh họa chu kỳ xử lý sự kiện**

**A diagram of a diagram

Description automatically generated**

Hình 3.1 Trong tài liệu gốc minh họa một sơ đồ đơn giản về các phần chính của một chương trình điều khiển sự kiện.

Hãy tưởng tượng vòng lặp chính của một hệ thống điều khiển sự kiện giống như một nhân viên trực tổng đài chăm sóc khách hàng. Khi không có cuộc gọi nào đến, nhân viên đang "chờ đợi". Cuối cùng, khi có sự kiện xảy ra, chẳng hạn như điện thoại reo, nhân viên bắt đầu quy trình phản hồi, bao gồm việc nói chuyện với khách hàng cho đến khi có đủ thông tin để chuyển khách hàng đến bộ phận phù hợp. Sau đó, nhân viên chờ đợi sự kiện tiếp theo.

**Các đặc điểm chính của cấu trúc chương trình điều khiển sự kiện**

Mặc dù mỗi hệ thống điều khiển sự kiện có chút khác nhau, nhưng chúng có nhiều điểm tương đồng. Các đặc điểm chính của cấu trúc chương trình điều khiển sự kiện như sau:

**Vòng lặp chờ đợi (idle loop)**

Sau khi thiết lập ban đầu, chương trình dành phần lớn thời gian trong một vòng lặp chờ đợi, nơi nó thực hiện ít hoặc không xử lý thông tin. Trong wxPython, vòng lặp này là phương thức wx.App.MainLoop(), và nó được gọi rõ ràng trong kịch bản của bạn. Vòng lặp chính tự động kết thúc khi tất cả các cửa sổ cấp cao nhất bị đóng.

**Sự kiện (events)**

Chương trình chứa các sự kiện tương ứng với những gì xảy ra trong môi trường chương trình. Sự kiện thường được kích hoạt bởi hoạt động của người dùng, nhưng cũng có thể là kết quả của hoạt động hệ thống hoặc mã tùy ý ở nơi khác trong chương trình. Trong wxPython, tất cả các sự kiện là các instance của lớp wx.Event hoặc một trong các lớp con của nó. Mỗi sự kiện có một thuộc tính loại sự kiện (event type) để phân biệt các loại sự kiện khác nhau. Ví dụ, sự kiện chuột lên và chuột xuống đều là instance của cùng một lớp, nhưng có loại sự kiện khác nhau.

**Kiểm tra sự kiện**

Là một phần của vòng lặp chờ đợi, chương trình định kỳ kiểm tra xem có điều gì đó cần phản hồi hay không. Có hai cơ chế để một hệ thống điều khiển sự kiện được thông báo về các sự kiện. Phương pháp phổ biến hơn, được sử dụng bởi wxPython, là đưa các sự kiện vào một hàng đợi trung tâm, cái này sẽ kích hoạt việc xử lý sự kiện đó. Các hệ thống điều khiển sự kiện khác sử dụng phương pháp thăm dò (polling), nơi các khả năng sinh ra sự kiện được truy vấn định kỳ bởi quy trình trung tâm và được hỏi xem họ có sự kiện nào đang chờ không.

**Xử lý sự kiện**

Khi một sự kiện xảy ra, hệ thống điều khiển sự kiện xử lý sự kiện đó để xác định mã nào, nếu có, nên được thực thi. Trong wxPython, các sự kiện hệ thống gốc được chuyển đổi thành các instance của wx.Event và sau đó được đưa đến phương thức wx.EvtHandler.ProcessEvent() để phân phối đến mã xử lý thích hợp. Các phần của cơ chế sự kiện bao gồm các đối tượng liên kết sự kiện (event binder) và các bộ xử lý sự kiện (event handler). Một liên kết sự kiện là một đối tượng wxPython được định nghĩa trước. Có một liên kết sự kiện riêng biệt cho mỗi loại sự kiện. Một bộ xử lý sự kiện là một hàm hoặc phương thức nhận một instance của sự kiện wxPython làm đối số. Một bộ xử lý sự kiện được gọi khi người dùng kích hoạt sự kiện tương ứng.

**Kết luận**

Vì vậy, lập trình điều khiển sự kiện là một kiểu cấu trúc chương trình mà trong đó chương trình chờ đợi các sự kiện xảy ra và sau đó phản hồi lại chúng. wxPython sử dụng phương thức MainLoop() để giữ cho chương trình hoạt động và xử lý các sự kiện khi chúng xảy ra.

**Coding event handlers - Viết các bộ xử lý sự kiện (Event Handlers) trong wxPython**

Trong wxPython, các sự kiện và bộ xử lý sự kiện được quản lý trên cơ sở từng widget (từng thành phần giao diện). Ví dụ, một sự kiện nhấp chuột vào nút sẽ được gửi đến bộ xử lý tương ứng dựa trên nút nào được nhấp. Để liên kết một sự kiện từ một widget cụ thể đến một phương thức xử lý cụ thể, bạn sử dụng một đối tượng liên kết (binder object) để quản lý kết nối này.

Ví dụ :

|  |
| --- |
| import wx  class MyFrame(wx.Frame):  def \_\_init\_\_(self, \*args, \*\*kw):  super(MyFrame, self).\_\_init\_\_(\*args, \*\*kw)    # Tạo một nút  aButton = wx.Button(self, label="Click Me")    # Liên kết sự kiện nhấp chuột của nút với phương thức xử lý sự kiện OnClick  self.Bind(wx.EVT\_BUTTON, self.OnClick, aButton)    # Thiết lập giao diện cơ bản  self.sizer = wx.BoxSizer(wx.VERTICAL)  self.sizer.Add(aButton, 0, wx.ALL | wx.CENTER, 5)  self.SetSizerAndFit(self.sizer)  def OnClick(self, event):  wx.MessageBox("Nút đã được nhấp!", "Thông báo", wx.OK | wx.ICON\_INFORMATION)  class MyApp(wx.App):  def OnInit(self):  self.frame = MyFrame(None, title="Event Handling Example")  self.frame.Show()  return True  app = MyApp(False)  app.MainLoop() |
| Trong ví dụ này:   * **wx.EVT\_BUTTON** là đối tượng liên kết sự kiện được định nghĩa sẵn trong wxPython. * **self.Bind(wx.EVT\_BUTTON, self.OnClick, aButton)** liên kết sự kiện nhấp chuột của nút **aButton** với phương thức **OnClick**. |

**Vòng lặp chính (Main Loop) trong wxPython**

Mặc dù chương trình wxPython của bạn dường như đang chờ đợi thụ động cho một sự kiện, thực tế nó vẫn đang thực hiện một công việc nào đó. Cụ thể, nó đang chạy phương thức wx.App.MainLoop(), về cơ bản là một vòng lặp vô hạn.

Phương thức MainLoop() có thể được biểu diễn bằng mã giả Python đơn giản như sau:

|  |
| --- |
| while True:  while not self.Pending():  self.ProcessIdle()  self.DoMessage() |

Nói cách khác, nếu không có thông điệp nào đang chờ xử lý, chương trình sẽ thực hiện một số xử lý không hoạt động (idle processing) cho đến khi một thông điệp đến, sau đó gửi thông điệp đó đến phương thức xử lý sự kiện thích hợp.

**Một số ví dụ thực tế khác**

Ví dụ 1: Xử lý sự kiện di chuyển chuột

|  |
| --- |
| import wx  class MyFrame(wx.Frame):  def \_\_init\_\_(self, \*args, \*\*kw):  super(MyFrame, self).\_\_init\_\_(\*args, \*\*kw)    # Liên kết sự kiện di chuyển chuột với phương thức xử lý sự kiện OnMouseMove  self.Bind(wx.EVT\_MOTION, self.OnMouseMove)    self.text = wx.StaticText(self, label="Di chuyển chuột trong cửa sổ này")  self.sizer = wx.BoxSizer(wx.VERTICAL)  self.sizer.Add(self.text, 0, wx.ALL | wx.CENTER, 5)  self.SetSizerAndFit(self.sizer)  def OnMouseMove(self, event):  pos = event.GetPosition()  self.text.SetLabel(f"Vị trí chuột: {pos}")  class MyApp(wx.App):  def OnInit(self):  self.frame = MyFrame(None, title="Mouse Move Event Example")  self.frame.Show()  return True  app = MyApp(False)  app.MainLoop() |
| Trong ví dụ này, sự kiện di chuyển chuột được xử lý bằng phương thức OnMouseMove, và vị trí của chuột được hiển thị trong cửa sổ. |

Ví dụ 2: Xử lý sự kiện thay đổi giá trị trong TextCtrl

|  |
| --- |
| import wx  class MyFrame(wx.Frame):  def \_\_init\_\_(self, \*args, \*\*kw):  super(MyFrame, self).\_\_init\_\_(\*args, \*\*kw)    self.textCtrl = wx.TextCtrl(self)    # Liên kết sự kiện thay đổi giá trị với phương thức xử lý sự kiện OnTextChange  self.Bind(wx.EVT\_TEXT, self.OnTextChange, self.textCtrl)    self.sizer = wx.BoxSizer(wx.VERTICAL)  self.sizer.Add(self.textCtrl, 0, wx.ALL | wx.EXPAND, 5)  self.SetSizerAndFit(self.sizer)  def OnTextChange(self, event):  newValue = self.textCtrl.GetValue()  self.SetTitle(f"TextCtrl Value: {newValue}")  class MyApp(wx.App):  def OnInit(self):  self.frame = MyFrame(None, title="Text Change Event Example")  self.frame.Show()  return True  app = MyApp(False)  app.MainLoop() |
| Trong ví dụ này, khi người dùng thay đổi giá trị trong TextCtrl, phương thức OnTextChange sẽ cập nhật tiêu đề của cửa sổ để hiển thị giá trị mới. |

**Kết luận**

Trong wxPython, việc xử lý sự kiện được thực hiện bằng cách liên kết các sự kiện cụ thể với các phương thức xử lý sự kiện tương ứng. Phương thức Bind() được sử dụng để tạo liên kết này. Chương trình wxPython sử dụng vòng lặp chính (MainLoop()) để chờ và xử lý các sự kiện khi chúng xảy ra.

**Thiết kế cho các chương trình điều khiển sự kiện**

**Lập trình điều khiển sự kiện (event-driven programming)** trong wxPython có một số hệ quả quan trọng đối với việc thiết kế và mã hóa. Vì không có giả định về thời điểm các sự kiện xảy ra, lập trình viên nhường nhiều quyền kiểm soát chương trình cho người dùng. Phần lớn mã trong chương trình wxPython của bạn được thực thi như là kết quả trực tiếp hoặc gián tiếp của hành động từ người dùng hoặc hệ thống. Ví dụ, việc lưu công việc trong chương trình của bạn xảy ra sau khi người dùng chọn một mục trong menu, nhấn một nút trên thanh công cụ, hoặc sử dụng một tổ hợp phím đặc biệt. Bất kỳ sự kiện nào trong số này có thể kích hoạt một bộ xử lý (handler) để lưu công việc của người dùng.

**Hậu quả của kiến trúc điều khiển sự kiện**

Một hệ quả khác của kiến trúc điều khiển sự kiện là kiến trúc này thường khá phân tán. Mã được gọi để phản hồi một sự kiện thường không được định nghĩa bởi widget đã kích hoạt sự kiện đó. Nói rõ hơn, không có gì trong bản chất của việc liên kết (binding) giữa một sự kiện và bộ xử lý của nó yêu cầu chúng phải có bất kỳ mối quan hệ nào với nhau. Ví dụ, mã được gọi để phản hồi một sự kiện nhấn nút không nhất thiết phải là một phần của định nghĩa của nút, mà có thể nằm trong khung chứa nút đó, hoặc bất kỳ vị trí nào khác.

Khi kết hợp với thiết kế hướng đối tượng (object-oriented design) tốt, kiến trúc này có thể dẫn đến mã lỏng lẻo và có thể tái sử dụng cao. Bạn sẽ thấy rằng bản chất linh hoạt của Python làm cho việc tái sử dụng các bộ xử lý sự kiện chung và cấu trúc giữa các ứng dụng wxPython khác nhau trở nên dễ dàng. Mặt khác, bản chất không liên kết chặt chẽ của một chương trình điều khiển sự kiện có thể khiến việc theo dõi và bảo trì trở nên khó khăn. Khi một sự kiện nhấp chuột xảy ra trong một nút được liên kết với một binder được liệt kê trong mã khung, và sự kiện này gọi một phương thức trong một lớp mô hình, việc theo dõi nó có thể trở nên khó khăn. (Ở một mức độ nào đó, vấn đề này là đúng với tất cả các lập trình hướng đối tượng).

Ví dụ minh họa

Ví dụ: Lưu công việc khi người dùng nhấn nút

|  |
| --- |
| import wx  class MyFrame(wx.Frame):  def \_\_init\_\_(self, \*args, \*\*kw):  super(MyFrame, self).\_\_init\_\_(\*args, \*\*kw)    # Tạo một nút lưu  saveButton = wx.Button(self, label="Save Work")    # Liên kết sự kiện nhấn nút với phương thức xử lý sự kiện OnSave  self.Bind(wx.EVT\_BUTTON, self.OnSave, saveButton)    self.sizer = wx.BoxSizer(wx.VERTICAL)  self.sizer.Add(saveButton, 0, wx.ALL | wx.CENTER, 5)  self.SetSizerAndFit(self.sizer)  def OnSave(self, event):  # Giả sử đây là phương thức lưu công việc  self.SaveWork()  def SaveWork(self):  # Mã để lưu công việc, ví dụ ghi một tệp  with open("work.txt", "w") as file:  file.write("Work saved!")  class MyApp(wx.App):  def OnInit(self):  self.frame = MyFrame(None, title="Event-Driven Design Example")  self.frame.Show()  return True  app = MyApp(False)  app.MainLoop() |
| * Một nút "Save Work" được tạo và sự kiện nhấn nút được liên kết với phương thức OnSave. * Khi người dùng nhấn nút, phương thức OnSave được gọi, phương thức này sau đó gọi SaveWork để lưu công việc. |

Lập trình điều khiển sự kiện trong wxPython yêu cầu lập trình viên phải thiết kế mã một cách lỏng lẻo và có thể tái sử dụng. Tuy nhiên, điều này cũng có thể làm cho việc theo dõi và bảo trì mã trở nên khó khăn hơn. Việc sử dụng tốt các nguyên tắc thiết kế hướng đối tượng và hiểu rõ cách thức hoạt động của các sự kiện và bộ xử lý trong wxPython sẽ giúp bạn xây dựng các ứng dụng hiệu quả và dễ bảo trì hơn.

**Kích hoạt sự kiện trong wxPython (Event triggers)**

Trong wxPython, hầu hết các widget sẽ kích hoạt các sự kiện cấp cao hơn để phản hồi các sự kiện cấp thấp hơn. Ví dụ, một lần nhấp chuột vào một nút (wx.Button) sẽ tạo ra sự kiện EVT\_BUTTON, là một loại sự kiện wx.CommandEvent cụ thể. Tương tự, khi bạn kéo chuột ở góc của cửa sổ, wxPython sẽ tự động tạo ra sự kiện wx.SizeEvent cho bạn. Lợi ích của các sự kiện cấp cao này là chúng giúp hệ thống của bạn dễ dàng tập trung vào các sự kiện quan trọng nhất, thay vì phải theo dõi từng lần nhấp chuột. Ví dụ, nói rằng một lần nhấp chuột là kích hoạt nút làm cho nó rõ ràng rằng nhấp chuột đó có ý nghĩa ngữ cảnh trong hệ thống, trong khi một lần nhấp chuột khác có thể không có ý nghĩa ngữ cảnh. Các sự kiện cấp cao cũng có thể bao gồm nhiều thông tin hữu ích hơn về sự kiện đó. Khi bạn tạo các widget tùy chỉnh của riêng mình, bạn cũng có thể định nghĩa các sự kiện tùy chỉnh của mình để quản lý quá trình này.

Các sự kiện trong wxPython được đại diện bởi các đối tượng. Cụ thể, các đối tượng sự kiện trong wxPython là các thể hiện của lớp wx.Event hoặc một trong các lớp con của nó. Lớp cha wx.Event là một lớp trừu tượng nhỏ bao gồm các phương thức lấy và đặt cho một vài thuộc tính chung cho tất cả các sự kiện, chẳng hạn như EventType, EventObject và Timestamp. Các lớp con khác nhau của wx.Event thêm thông tin chi tiết hơn. Ví dụ, wx.MouseEvent chứa thông tin về vị trí chính xác của chuột khi sự kiện xảy ra và thông tin về nút chuột nào đã được nhấp, nếu có.

Có nhiều lớp con khác nhau của wx.Event trong wxPython. Bảng 3.2 chứa danh sách một số lớp sự kiện mà bạn sẽ gặp thường xuyên nhất. Hãy nhớ rằng một lớp sự kiện có thể có nhiều loại sự kiện, mỗi loại tương ứng với một hành động khác nhau của người dùng.

|  |
| --- |
|  |

Thông thường, các đối tượng sự kiện tự chúng không làm nhiều việc, mà cần được chuyển đến các phương thức xử lý sự kiện liên quan bằng cách sử dụng một bộ liên kết sự kiện và hệ thống xử lý sự kiện.

Ví dụ minh họa

Ví dụ 1: Xử lý sự kiện nhấp nút

|  |
| --- |
| import wx  class MyFrame(wx.Frame):  def \_\_init\_\_(self, \*args, \*\*kw):  super(MyFrame, self).\_\_init\_\_(\*args, \*\*kw)    self.button = wx.Button(self, label="Click Me")    # Liên kết sự kiện nhấp nút với phương thức xử lý sự kiện OnButtonClick  self.Bind(wx.EVT\_BUTTON, self.OnButtonClick, self.button)    self.sizer = wx.BoxSizer(wx.VERTICAL)  self.sizer.Add(self.button, 0, wx.ALL | wx.CENTER, 5)  self.SetSizerAndFit(self.sizer)  def OnButtonClick(self, event):  wx.MessageBox("Nút đã được nhấp!", "Thông báo", wx.OK | wx.ICON\_INFORMATION)  class MyApp(wx.App):  def OnInit(self):  self.frame = MyFrame(None, title="Button Click Event Example")  self.frame.Show()  return True  app = MyApp(False)  app.MainLoop() |
| * Khi người dùng nhấp vào nút, sự kiện EVT\_BUTTON được tạo ra và liên kết với phương thức OnButtonClick. * Phương thức OnButtonClick hiển thị một hộp thoại thông báo rằng nút đã được nhấp. |

Ví dụ 2: Xử lý sự kiện thay đổi kích thước cửa sổ

|  |
| --- |
| import wx  class MyFrame(wx.Frame):  def \_\_init\_\_(self, \*args, \*\*kw):  super(MyFrame, self).\_\_init\_\_(\*args, \*\*kw)    self.Bind(wx.EVT\_SIZE, self.OnSize)    self.text = wx.StaticText(self, label="Thay đổi kích thước cửa sổ này")  self.sizer = wx.BoxSizer(wx.VERTICAL)  self.sizer.Add(self.text, 0, wx.ALL | wx.CENTER, 5)  self.SetSizerAndFit(self.sizer)  def OnSize(self, event):  size = event.GetSize()  self.text.SetLabel(f"Kích thước mới: {size}")  class MyApp(wx.App):  def OnInit(self):  self.frame = MyFrame(None, title="Window Size Event Example")  self.frame.Show()  return True  app = MyApp(False)  app.MainLoop() |
| * Khi người dùng thay đổi kích thước cửa sổ, sự kiện EVT\_SIZE được tạo ra và liên kết với phương thức OnSize. * Phương thức OnSize cập nhật nhãn của StaticText để hiển thị kích thước mới của cửa sổ. |

Trong wxPython, các widget sẽ kích hoạt các sự kiện cấp cao hơn để phản hồi các sự kiện cấp thấp hơn, giúp hệ thống dễ dàng tập trung vào các sự kiện quan trọng nhất. Các sự kiện này được đại diện bởi các đối tượng sự kiện, là các thể hiện của lớp wx.Event hoặc các lớp con của nó. Bằng cách sử dụng các bộ liên kết sự kiện và hệ thống xử lý sự kiện, bạn có thể dễ dàng quản lý và phản hồi các sự kiện trong ứng dụng của mình. Các ví dụ trên minh họa cách xử lý các sự kiện phổ biến như nhấp nút và thay đổi kích thước cửa sổ trong wxPython.

**Cách liên kết một sự kiện với một trình xử lý (handler) trong wxPython (How do I bind an event to a handler?)**

Trong wxPython, việc liên kết một sự kiện với một trình xử lý (handler) được thực hiện thông qua các đối tượng event binder. Các đối tượng này là các thể hiện của lớp wx.PyEventBinder. Một thể hiện của wx.PyEventBinder được cung cấp sẵn cho tất cả các loại sự kiện được hỗ trợ, và bạn cũng có thể tạo ra các event binder của riêng mình cho các loại sự kiện tùy chỉnh khi cần thiết.

**Các event binder và các loại sự kiện**

Mỗi loại sự kiện sẽ có một thể hiện event binder riêng. Ví dụ, lớp wx.MouseEvent có mười bốn loại sự kiện khác nhau, mỗi loại sử dụng cùng một thông tin cơ bản về trạng thái của chuột khi sự kiện đó được kích hoạt (ví dụ: nhấp chuột trái, nhấp chuột phải, nhấp đúp chuột).

Trong wxPython, tên của các event binder là toàn cục và bắt đầu bằng wx.EVT\_. Ví dụ, một số loại sự kiện của wx.MouseEvent bao gồm:

* wx.EVT\_LEFT\_DOWN: Nhấp chuột trái
* wx.EVT\_LEFT\_UP: Thả chuột trái
* wx.EVT\_LEFT\_DCLICK: Nhấp đúp chuột trái
* wx.EVT\_MOTION: Di chuyển chuột
* wx.EVT\_MOUSEWHEEL: Cuộn chuột

Bạn có thể liên kết tất cả các sự kiện chuột vào một hàm duy nhất bằng cách sử dụng loại sự kiện wx.EVT\_MOUSE\_EVENTS.

**Làm việc với các phương thức của wx.EvtHandler (Working with the wx.EvtHandler methods)**

Lớp wx.EvtHandler định nghĩa nhiều phương thức, trong đó phương thức thường được sử dụng nhất là Bind(). Phương thức này được sử dụng để tạo liên kết sự kiện giữa các đối tượng và các hàm xử lý sự kiện.

Cú pháp của phương thức Bind()

|  |
| --- |
| Bind(event, handler, source=None, id=wx.ID\_ANY, id2=wx.ID\_ANY) |
| * event: Tham số bắt buộc, là một thể hiện của wx.PyEventBinder đại diện cho loại sự kiện. * handler: Tham số bắt buộc, là một hàm hoặc phương thức trong Python có thể gọi được với một đối số duy nhất là đối tượng sự kiện. * source: Widget là nguồn của sự kiện (tùy chọn). * id và id2: Định danh của sự kiện (tùy chọn). |

Ví dụ cụ thể

Dưới đây là ví dụ minh họa việc liên kết sự kiện với và không có tham số nguồn:

|  |
| --- |
| import wx  class MyFrame(wx.Frame):  def \_\_init\_\_(self, parent, id):  wx.Frame.\_\_init\_\_(self, parent, id, 'Frame With Button', size=(300, 100))    panel = wx.Panel(self, -1)    # Tạo một nút bấm  button = wx.Button(panel, -1, "Close", pos=(130, 15), size=(40, 40))    # Liên kết sự kiện đóng khung hình với phương thức OnCloseWindow  self.Bind(wx.EVT\_CLOSE, self.OnCloseWindow)    # Liên kết sự kiện nhấn nút với phương thức OnCloseMe, sử dụng tham số nguồn là button  self.Bind(wx.EVT\_BUTTON, self.OnCloseMe, button)    def OnCloseMe(self, event):  self.Close(True)    def OnCloseWindow(self, event):  self.Destroy()  class MyApp(wx.App):  def OnInit(self):  frame = MyFrame(None, -1)  frame.Show(True)  return True  app = MyApp(0)  app.MainLoop() |

Trong ví dụ này:

**Liên kết sự kiện đóng khung hình:**

|  |
| --- |
| self.Bind(wx.EVT\_CLOSE, self.OnCloseWindow) |

Liên kết sự kiện EVT\_CLOSE (sự kiện đóng cửa sổ) với phương thức OnCloseWindow.

Không cần tham số nguồn vì sự kiện được kích hoạt và được xử lý bởi cùng một đối tượng là khung hình (Frame).

**Liên kết sự kiện nhấn nút:**

|  |
| --- |
| self.Bind(wx.EVT\_BUTTON, self.OnCloseMe, button) |

Liên kết sự kiện EVT\_BUTTON (sự kiện nhấn nút) với phương thức OnCloseMe.

Tham số nguồn button được sử dụng để xác định nút nào đã kích hoạt sự kiện, vì có thể có nhiều nút khác nhau trong khung hình.

Chi tiết thêm về các tham số

* event: Đại diện cho loại sự kiện, chẳng hạn wx.EVT\_BUTTON là sự kiện nhấn nút.
* handler: Hàm hoặc phương thức sẽ xử lý sự kiện. Trong ví dụ, đó là self.OnCloseMe và self.OnCloseWindow.
* source: Nguồn của sự kiện. Trong ví dụ, nút button là nguồn sự kiện cho EVT\_BUTTON.
* id và id2: Thường được sử dụng để xác định các sự kiện khi có nhiều nguồn sự kiện tương tự nhau.

Bạn cũng có thể sử dụng tham số source để xác định các đối tượng ngay cả khi đối tượng đó không phải là nguồn của sự kiện. Ví dụ, bạn có thể liên kết một sự kiện menu với trình xử lý sự kiện ngay cả khi sự kiện menu về mặt kỹ thuật được kích hoạt bởi khung hình. Ví dụ dưới đây minh họa cách liên kết một sự kiện menu.

**Ví dụ về liên kết một sự kiện menu**

|  |
| --- |
| #!/usr/bin/env python  import wx  class MenuEventFrame(wx.Frame):  def \_\_init\_\_(self, parent, id):  wx.Frame.\_\_init\_\_(self, parent, id, 'Menus', size=(300, 200))    # Tạo một thanh menu  menuBar = wx.MenuBar()    # Tạo một menu  menu1 = wx.Menu()    # Thêm một mục vào menu  menuItem = menu1.Append(-1, "&Exit...")    # Thêm menu vào thanh menu  menuBar.Append(menu1, "&File")    # Đặt thanh menu cho khung hình  self.SetMenuBar(menuBar)    # Liên kết sự kiện menu với trình xử lý sự kiện OnCloseMe  self.Bind(wx.EVT\_MENU, self.OnCloseMe, menuItem)    def OnCloseMe(self, event):  self.Close(True)  if \_\_name\_\_ == '\_\_main\_\_':  app = wx.App(False)  frame = MenuEventFrame(parent=None, id=-1)  frame.Show()  app.MainLoop() |
|  |

Tạo thanh menu và mục menu:

|  |
| --- |
| menuBar = wx.MenuBar()  menu1 = wx.Menu()  menuItem = menu1.Append(-1, "&Exit...")  menuBar.Append(menu1, "&File") |

* Tạo một thanh menu và thêm mục menu "Exit" vào thanh menu.

Ràng buộc sự kiện menu:

|  |
| --- |
| self.Bind(wx.EVT\_MENU, self.OnCloseMe, menuItem) |

* Ràng buộc sự kiện EVT\_MENU (sự kiện menu) với phương thức OnCloseMe.
* Tham số menuItem được sử dụng để xác định mục menu cụ thể nào đã kích hoạt sự kiện.

**Sử dụng các tham số id và id2**

Các tham số id và id2 của phương thức Bind() chỉ định nguồn của sự kiện bằng cách sử dụng số ID, thay vì widget. Thông thường, các tham số id và id2 không cần thiết, vì ID của nguồn sự kiện có thể được trích xuất từ tham số source. Tuy nhiên, đôi khi sử dụng ID trực tiếp có thể hợp lý hơn. Ví dụ, nếu bạn đang sử dụng các số ID được định trước cho một hộp thoại, sẽ dễ dàng hơn khi sử dụng số ID hơn là sử dụng widget.

Nếu bạn sử dụng cả hai số ID id và id2, bạn có thể ràng buộc toàn bộ phạm vi các widget với sự kiện có số ID nằm giữa hai ID đó. Điều này chỉ hữu ích nếu các ID của các widget bạn muốn ràng buộc là liên tiếp nhau.

**Kiểu ràng buộc sự kiện cũ**

Phương thức Bind() là mới trong wxPython 2.5. Trong các phiên bản wxPython trước đó, tên EVT\_\* được sử dụng như một đối tượng hàm, vì vậy một lời gọi ràng buộc sẽ xuất hiện như sau:

|  |
| --- |
| wx.EVT\_BUTTON(self, self.button.GetId(), self.OnClick) |

Nhược điểm của kiểu cũ là nó không trông giống hoặc hoạt động như một lời gọi phương thức hướng đối tượng. Tuy nhiên, kiểu cũ vẫn hoạt động trong wxPython 2.5 (vì các đối tượng wx.EVT\* vẫn có thể gọi được), nên bạn vẫn có thể thấy nó trong mã wxPython.

**Kết luận**

Phương thức Bind() trong wxPython cho phép bạn linh hoạt ràng buộc các sự kiện với các trình xử lý sự kiện một cách rõ ràng và dễ hiểu. Bạn có thể sử dụng các tham số như source, id, và id2 để kiểm soát chi tiết hơn nguồn gốc và phạm vi của các sự kiện. Điều này giúp mã nguồn của bạn trở nên linh hoạt và dễ bảo trì hơn.

|  |
| --- |
|  |

**Xử lý sự kiện trong wxPython (How are events processed by wxPython?)**

Một thành phần quan trọng của hệ thống dựa trên sự kiện là quy trình mà một sự kiện được gửi đến hệ thống và sau đó được chuyển tiếp đến đoạn mã sẽ được thực thi để phản hồi sự kiện đó. Trong phần này, chúng ta sẽ xem xét quy trình mà wxPython sử dụng khi xử lý một sự kiện đến. Chúng ta sẽ sử dụng một đoạn mã nhỏ làm ví dụ để theo dõi các bước trong quy trình này. Hình 3.2 hiển thị một cửa sổ mẫu với một nút bấm duy nhất, sẽ được sử dụng để tạo ra các sự kiện mẫu.

Ví dụ: Cửa sổ đơn giản với các sự kiện chuột

|  |
| --- |
| #!/usr/bin/env python  import wx  class MouseEventFrame(wx.Frame):  def \_\_init\_\_(self, parent, id):  wx.Frame.\_\_init\_\_(self, parent, id, 'Frame With Button', size=(300, 100))  self.panel = wx.Panel(self)  self.button = wx.Button(self.panel, label="Not Over", pos=(100, 15))    # Ràng buộc sự kiện nút bấm  self.Bind(wx.EVT\_BUTTON, self.OnButtonClick, self.button)    # Ràng buộc sự kiện di chuột vào nút  self.button.Bind(wx.EVT\_ENTER\_WINDOW, self.OnEnterWindow)    # Ràng buộc sự kiện rời chuột khỏi nút  self.button.Bind(wx.EVT\_LEAVE\_WINDOW, self.OnLeaveWindow)    def OnButtonClick(self, event):  self.panel.SetBackgroundColour('Green')  self.panel.Refresh()    def OnEnterWindow(self, event):  self.button.SetLabel("Over Me!")  event.Skip()    def OnLeaveWindow(self, event):  self.button.SetLabel("Not Over")  event.Skip()  if \_\_name\_\_ == '\_\_main\_\_':  app = wx.PySimpleApp()  frame = MouseEventFrame(parent=None, id=-1)  frame.Show()  app.MainLoop() |

Trong ví dụ này:

Ràng buộc sự kiện nút bấm:

|  |
| --- |
| self.Bind(wx.EVT\_BUTTON, self.OnButtonClick, self.button) |

* Sự kiện nhấn nút được ràng buộc với phương thức OnButtonClick trong khung (frame). Khi nút được nhấn, màu nền của khung sẽ đổi thành màu xanh lá cây.

Ràng buộc sự kiện di chuột vào nút:

|  |
| --- |
| self.button.Bind(wx.EVT\_ENTER\_WINDOW, self.OnEnterWindow) |

* Khi con trỏ chuột di chuyển vào nút, nhãn của nút sẽ thay đổi thành "Over Me!".

Ràng buộc sự kiện rời chuột khỏi nút:

|  |
| --- |
| self.button.Bind(wx.EVT\_LEAVE\_WINDOW, self.OnLeaveWindow) |

* Khi con trỏ chuột rời khỏi nút, nhãn của nút sẽ trở lại thành "Not Over".

**Quy trình xử lý sự kiện trong wxPython**

Khi một sự kiện xảy ra trong wxPython, quy trình xử lý sự kiện sẽ tìm kiếm trình xử lý sự kiện tương ứng với sự kiện đó. Các bước cụ thể như sau:

* Gửi sự kiện từ widget:

Khi sự kiện xảy ra, chẳng hạn như nhấn nút, widget (nút bấm trong ví dụ này) sẽ tạo ra một đối tượng sự kiện (event object).

* Tìm kiếm trình xử lý sự kiện:

wxPython bắt đầu tìm kiếm từ widget nơi sự kiện xảy ra. Nếu widget có ràng buộc sự kiện cho sự kiện đó, wxPython sẽ gọi phương thức xử lý sự kiện tương ứng.

* Tìm kiếm trong khung chứa widget:

Nếu widget không có ràng buộc sự kiện, wxPython sẽ tiếp tục tìm kiếm trong khung chứa widget (frame). Trong ví dụ, sự kiện nhấn nút được ràng buộc với phương thức OnButtonClick trong khung, vì vậy khi nút được nhấn, phương thức OnButtonClick trong khung sẽ được gọi.

* Xử lý sự kiện và bỏ qua sự kiện:

Nếu một sự kiện được xử lý nhưng không được bỏ qua (event.Skip()), wxPython sẽ dừng lại và không tìm kiếm tiếp. Nếu sự kiện được bỏ qua (event.Skip()), wxPython sẽ tiếp tục tìm kiếm trình xử lý sự kiện trong các đối tượng cha.

Trong wxPython, sự kiện được xử lý bằng cách tìm kiếm trình xử lý sự kiện từ widget xảy ra sự kiện và sau đó tìm kiếm lên các đối tượng cha. Bằng cách sử dụng phương thức Bind(), bạn có thể ràng buộc các sự kiện cụ thể với các phương thức xử lý sự kiện, cho phép ứng dụng của bạn phản hồi linh hoạt và chính xác các hành động của người dùng.

**Hiểu quy trình xử lý sự kiện(Understanding the event handling process)**

Quy trình xử lý sự kiện trong wxPython được thiết kế để giúp lập trình viên dễ dàng tạo các ràng buộc sự kiện ở những nơi hiển nhiên nhất, đồng thời bỏ qua các sự kiện không quan trọng. Mặc dù thiết kế đơn giản nhưng cơ chế bên dưới lại khá phức tạp. Tiếp theo, chúng ta sẽ xem xét quy trình xử lý sự kiện khi nhấn nút và khi di chuột vào nút.

Sơ đồ quy trình xử lý sự kiện

|  |
| --- |
|  |

Hình 3.3 hiển thị sơ đồ luồng cơ bản của quy trình xử lý sự kiện. Các hình chữ nhật biểu thị điểm bắt đầu và kết thúc của quy trình, các hình tròn biểu thị các đối tượng wxPython tham gia vào quy trình, các hình thoi biểu thị các điểm quyết định, và các hình chữ nhật với thanh ngang biểu thị các phương thức xử lý sự kiện thực tế.

**Quy trình xử lý sự kiện**

Quy trình sự kiện bắt đầu với đối tượng kích hoạt sự kiện. Thông thường, wxPython đầu tiên sẽ kiểm tra đối tượng kích hoạt để tìm một hàm xử lý sự kiện được ràng buộc với loại sự kiện đó. Nếu tìm thấy, phương thức sẽ được thực thi. Nếu không, wxPython sẽ kiểm tra xem sự kiện có truyền lên hệ thống phân cấp của các container hay không. Nếu có, widget cha sẽ được kiểm tra, tiếp tục lên hệ thống phân cấp cho đến khi wxPython tìm thấy một hàm xử lý hoặc đến được đối tượng cấp cao nhất. Nếu sự kiện không được truyền lên, wxPython vẫn kiểm tra đối tượng ứng dụng để tìm phương thức xử lý trước khi kết thúc. Khi một hàm xử lý sự kiện được chạy, quy trình thường kết thúc. Tuy nhiên, hàm có thể yêu cầu wxPython tiếp tục tìm kiếm các hàm xử lý khác.

**Mô tả quy trình các bước xử lý ở hình 3.3**

|  |  |
| --- | --- |
|  | **Bước 1**: Tạo sự kiện  Quy trình bắt đầu khi sự kiện được tạo ra. Hầu hết các loại sự kiện có sẵn được tạo ra trong khuôn khổ wxPython để phản hồi các hành động cụ thể của người dùng hoặc thông báo của hệ thống. Ví dụ:   * Sự kiện di chuột vào một widget mới được kích hoạt khi wxPython nhận thấy di chuyển chuột vào trong ranh giới của một widget mới. * Sự kiện nhấn nút được tạo ra sau khi nhấn chuột trái xuống và nhả chuột trái lên cùng một nút.   Sự kiện đầu tiên được chuyển đến đối tượng chịu trách nhiệm tạo ra sự kiện đó. Đối với sự kiện nhấn nút, đối tượng này là nút bấm, và đối với sự kiện di chuột vào, đối tượng này là widget mà chuột vừa vào. |
|  | **Bước 2:** Xác định liệu đối tượng có được phép xử lý sự kiện hay không  Bước tiếp theo trong quy trình xử lý sự kiện là kiểm tra xem widget chịu trách nhiệm (đối tượng wx.EvtHandler) hiện có được phép xử lý sự kiện hay không.   * Một cửa sổ có thể được đặt để cho phép hoặc không cho phép xử lý sự kiện bằng cách gọi phương thức SetEvtHandlerEnabled(boolean) của wx.EvtHandler. * Nếu việc xử lý sự kiện không được phép, widget sẽ bị bỏ qua hoàn toàn trong quy trình sự kiện, các đối tượng ràng buộc liên quan đến widget đó sẽ không được tìm kiếm, và quy trình ở bước này sẽ đi theo nhánh "No".   Việc bật hoặc tắt một widget ở mức trình xử lý sự kiện không giống như tắt widget ở mức giao diện người dùng (UI):   * Tắt một widget ở mức UI được thực hiện bằng cách sử dụng các phương thức Disable() và Enable() của wx.Window. * Tắt một widget ở mức UI có nghĩa là người dùng không thể tương tác với widget đã tắt. Thông thường, widget đã tắt sẽ bị làm mờ trên màn hình để chỉ ra trạng thái của nó. * Một cửa sổ bị tắt ở mức UI sẽ không thể tạo ra bất kỳ sự kiện nào; tuy nhiên, nếu nó nằm trong hệ thống phân cấp container cho các sự kiện khác, nó vẫn xử lý các sự kiện mà nó nhận được.   Kiểm tra trạng thái bật/tắt của đối tượng khởi tạo xảy ra trong phương thức ProcessEvent() được gọi bởi hệ thống wxPython để bắt đầu và xử lý cơ chế phân phối sự kiện. Phương thức ProcessEvent() là phương thức trong lớp wx.EvtHandler thực hiện hầu hết các quy trình sự kiện được mô tả trong hình 3.3.   * Phương thức ProcessEvent() trả về True nếu việc xử lý sự kiện hoàn tất ở cuối phương thức. Việc xử lý được coi là hoàn tất nếu một hàm xử lý được tìm thấy cho tổ hợp instance và sự kiện đang được xử lý. * Hàm xử lý có thể yêu cầu tiếp tục xử lý bằng cách gọi phương thức Skip() của wx.Event. * Ngoài ra, nếu đối tượng khởi tạo là một lớp con của wx.Window, nó có thể lọc sự kiện bằng một đối tượng đặc biệt gọi là validator.   Validators sẽ được thảo luận chi tiết hơn trong phần sau |
|  | **Bước 3:** Xác định đối tượng ràng buộc  Phương thức ProcessEvent() sau đó tìm kiếm một đối tượng ràng buộc (binder) nhận ra sự liên kết giữa loại sự kiện và đối tượng hiện tại.   * Nếu không tìm thấy đối tượng ràng buộc cho chính đối tượng đó, quy trình sẽ đi lên theo hệ thống phân cấp lớp để tìm đối tượng ràng buộc được định nghĩa trong một lớp cha của đối tượng đó — điều này khác với việc đi lên theo hệ thống phân cấp container sẽ xảy ra trong bước tiếp theo. * Nếu tìm thấy một đối tượng ràng buộc, wxPython sẽ gọi hàm xử lý liên kết. Sau khi hàm xử lý được gọi, việc xử lý sự kiện dừng lại cho sự kiện đó, trừ khi hàm xử lý yêu cầu rõ ràng việc xử lý thêm.   Trong danh sách 3.3, sự kiện di chuột vào được bắt, và vì có sự liên kết được định nghĩa giữa đối tượng nút, đối tượng ràng buộc wx.EVT\_ENTER\_WINDOW, và phương thức liên kết OnEnterWindow(), phương thức này được gọi. Vì chúng ta không liên kết sự kiện nhấn nút chuột, wx.EVT\_LEFT\_DOWN, wxPython sẽ tiếp tục tìm kiếm trong trường hợp đó. |
|  | **Bước 4:** Xác định liệu có tiếp tục xử lý hay không  Sau khi gọi hàm xử lý sự kiện đầu tiên, wxPython kiểm tra xem có yêu cầu xử lý thêm không. Hàm xử lý sự kiện yêu cầu xử lý thêm bằng cách gọi phương thức Skip() của wx.Event. Nếu phương thức Skip() được gọi, việc xử lý tiếp tục và bất kỳ hàm xử lý nào được định nghĩa trong lớp cha sẽ được tìm và thực thi trong bước này. Phương thức Skip() có thể được gọi ở bất kỳ điểm nào trong hàm xử lý, hoặc bất kỳ mã nào được gọi bởi hàm xử lý. Phương thức Skip() đặt một cờ trong instance của sự kiện, mà wxPython kiểm tra sau khi phương thức xử lý hoàn tất.  Trong danh sách 3.3, OnButtonClick() không gọi Skip(), vì vậy trong trường hợp đó, quy trình xử lý sự kiện hoàn tất ở cuối phương thức xử lý. Hai hàm xử lý sự kiện khác gọi Skip() nên hệ thống sẽ tiếp tục tìm kiếm sự kiện phù hợp, cuối cùng sẽ gọi chức năng mặc định cho các sự kiện di chuột vào và rời khỏi của widget gốc, chẳng hạn như các sự kiện mouse-over. |
| **Bước 5: Xác định liệu sự kiện có tiếp tục được truyền lên hay không**  Cuối cùng, wxPython xác định liệu quy trình sự kiện có nên truyền lên hệ thống phân cấp container để tìm hàm xử lý sự kiện hay không. Hệ thống phân cấp container là đường dẫn từ một widget cụ thể lên đến frame cấp cao nhất, di chuyển từ mỗi widget lên container cha của nó, và tiếp tục lên trên.   * Nếu đối tượng hiện tại không có hàm xử lý cho sự kiện, hoặc nếu hàm xử lý đã gọi **Skip()**, wxPython xác định liệu sự kiện có nên truyền lên hệ thống phân cấp container hay không. Nếu câu trả lời là Không, quy trình sẽ tìm kiếm một hàm xử lý trong instance của wx.App và sau đó dừng lại. Nếu câu trả lời là Có, quy trình sự kiện bắt đầu lại bằng cách sử dụng container của cửa sổ hiện tại đang được tìm kiếm. Quy trình tiếp tục đi lên cho đến khi wxPython hoặc tìm thấy một liên kết phù hợp, đạt đến đối tượng frame cấp cao nhất không có cha, hoặc đạt đến đối tượng wx.Dialog (ngay cả khi dialog không phải là cấp cao nhất). Sự kiện được coi là đã tìm thấy một liên kết phù hợp nếu **ProcessEvent()** cho đối tượng đó trả về True, cho biết việc xử lý đã hoàn tất. Lý do dừng lại ở wx.Dialog là để ngăn chặn các sự kiện không liên quan và không mong muốn đến từ các hộp thoại ảnh hưởng đến frame cha. * Việc sự kiện có nên truyền lên hệ thống phân cấp container là một thuộc tính động của mỗi instance sự kiện, mặc dù trong thực tế các giá trị mặc định gần như luôn được sử dụng. Theo mặc định, chỉ có các instance của **wx.CommandEvent**, hoặc bất kỳ lớp con nào của nó, truyền lên hệ thống phân cấp container. Tất cả các sự kiện khác thì không.   Trong danh sách 3.3, đây là nơi sự kiện nhấp chuột được xử lý. Khi nhấp chuột vào **wx.Button** sẽ tạo ra một loại sự kiện lệnh **wx.EVT\_BUTTON**. Vì **wx.EVT\_BUTTON** là một **wx.CommandEvent**, sau khi wxPython không tìm thấy một liên kết trong đối tượng nút, nó tìm đến cha của nó, trong trường hợp này là panel. Vì không có liên kết phù hợp trong panel, cha của panel, tức frame, được kiểm tra tiếp theo. Vì frame có một liên kết phù hợp, **ProcessEvent()** gọi hàm thích hợp, trong trường hợp này là **OnButtonClick()**.  Bước 5 cũng giải thích tại sao các sự kiện di chuột vào và di chuột ra cần phải được liên kết với nút và không phải với frame. Vì các sự kiện chuột không phải là lớp con của **wx.CommandEvent**, các sự kiện di chuột vào và di chuột ra không truyền lên cha, do đó wxPython không thể tìm thấy một liên kết từ sự kiện di chuột vào của nút đến frame. Nếu có một sự kiện di chuột vào hoặc ra được liên kết với frame, sự kiện đó được kích hoạt bởi wxPython khi chuột di chuyển vào hoặc rời khỏi toàn bộ frame.  Các sự kiện lệnh được ưu tiên theo cách này vì chúng được coi là các sự kiện cấp cao hơn, cho thấy rằng người dùng đang làm gì đó trong không gian ứng dụng, thay vì trong hệ thống cửa sổ. Giả định là các sự kiện hệ thống cửa sổ chỉ quan tâm đến widget nhận ban đầu, trong khi các sự kiện cấp ứng dụng có thể quan tâm đến cấp cao hơn trong hệ thống phân cấp chứa.  Cuối cùng, nếu sự kiện không được xử lý sau khi đi qua hệ thống phân cấp chứa, **ProcessEvent()** được gọi trên đối tượng **wx.App** cho ứng dụng. Theo mặc định, điều này không làm gì, tuy nhiên, bạn có thể thêm các liên kết sự kiện vào **wx.App** của bạn để định tuyến các sự kiện theo cách không chuẩn. Ví dụ, nếu bạn đang viết một trình tạo GUI, bạn có thể muốn các sự kiện trong cửa sổ trình tạo của bạn truyền đến cửa sổ mã của bạn, ngay cả khi cả hai đều là các cửa sổ cấp cao nhất. Một cách để làm điều đó là bắt các sự kiện trong đối tượng ứng dụng và chuyển chúng đến cửa sổ mã. | |

**Sử dụng phương thức Skip() (Using the Skip() method)**

Hàm xử lý sự kiện đầu tiên được tìm thấy cho một sự kiện sẽ dừng việc xử lý sự kiện đó trừ khi phương thức Skip() của sự kiện được gọi trước khi hàm xử lý kết thúc. Gọi Skip() cho phép tìm kiếm các liên kết hàm xử lý bổ sung, theo các quy tắc đã mô tả ở bước 4 của phần 3.4.1. Điều này có nghĩa là các lớp cha và các cửa sổ cha sẽ được tìm kiếm như thể hàm xử lý đầu tiên không tồn tại. Trong một số trường hợp, bạn muốn sự kiện tiếp tục xử lý để cho phép hành vi mặc định trong widget gốc được thực thi cùng với hàm xử lý tùy chỉnh của bạn. Ví dụ sau đây hiển thị một ví dụ về Skip() cho phép chương trình phản hồi cả sự kiện nhấn nút chuột trái và sự kiện nhấp chuột trên cùng một nút.

Ví dụ : Phản hồi sự kiện nhấn chuột và sự kiện nhấp chuột đồng thời

|  |
| --- |
| #!/usr/bin/env python  import wx  class DoubleEventFrame(wx.Frame):  def \_\_init\_\_(self, parent, id):  wx.Frame.\_\_init\_\_(self, parent, id, 'Frame With Button', size=(300, 100))  self.panel = wx.Panel(self, -1)  self.button = wx.Button(self.panel, -1, "Click Me", pos=(100, 15))  self.Bind(wx.EVT\_BUTTON, self.OnButtonClick, self.button)  self.button.Bind(wx.EVT\_LEFT\_DOWN, self.OnMouseDown)    def OnButtonClick(self, event):  self.panel.SetBackgroundColour('Green')  self.panel.Refresh()    def OnMouseDown(self, event):  self.button.SetLabel("Again!")  event.Skip()  if \_\_name\_\_ == '\_\_main\_\_':  app = wx.App()  frame = DoubleEventFrame(parent=None, id=-1)  frame.Show()  app.MainLoop() |
| **Giải thích mã:**   1. **Liên kết sự kiện nhấp chuột**:  |  | | --- | | self.Bind(wx.EVT\_BUTTON, self.OnButtonClick, self.button) |   Dòng này liên kết sự kiện nhấp chuột vào nút với hàm xử lý **OnButtonClick()**, thay đổi màu nền của frame.   1. **Liên kết sự kiện nhấn nút chuột trái**:  |  | | --- | | self.button.Bind(wx.EVT\_LEFT\_DOWN, self.OnMouseDown) |   Dòng này liên kết sự kiện nhấn nút chuột trái với hàm xử lý **OnMouseDown()**, thay đổi nhãn của nút. Vì sự kiện nhấn nút chuột trái không phải là một sự kiện lệnh (**CommandEvent**), sự kiện này phải được liên kết trực tiếp với nút thay vì frame.   1. **Hàm OnButtonClick**:  |  | | --- | | def OnButtonClick(self, event): self.panel.SetBackgroundColour('Green') self.panel.Refresh() |   Hàm này thay đổi màu nền của panel khi sự kiện nhấp chuột được xử lý.   1. **Hàm OnMouseDown**:  |  | | --- | | def OnMouseDown(self, event): self.button.SetLabel("Again!") event.Skip() |   Hàm này thay đổi nhãn của nút khi sự kiện nhấn nút chuột trái được xử lý. Dòng **event.Skip()** đảm bảo rằng quá trình xử lý sự kiện tiếp tục để sự kiện nhấp chuột cũng được tạo ra. |

**Hoạt động của Skip():**

Khi người dùng nhấp chuột vào nút, sự kiện nhấn nút chuột trái được tạo ra đầu tiên, thông qua tương tác trực tiếp với hệ điều hành. Trong điều kiện bình thường, sự kiện nhấn nút chuột trái thay đổi trạng thái của nút sao cho sự kiện nhả nút chuột trái tạo ra một sự kiện nhấp chuột (**wx.EVT\_BUTTON**). DoubleEventFrame bảo toàn quá trình xử lý này nhưng chỉ nhờ vào câu lệnh **Skip()** trong dòng cuối cùng. Nếu không có câu lệnh **Skip()**, thuật toán xử lý sự kiện sẽ tìm thấy liên kết được tạo ra và dừng lại trước khi nút có thể tạo ra sự kiện nhấp chuột (**wx.EVT\_BUTTON**). Với lệnh gọi **Skip()**, quá trình xử lý sự kiện tiếp tục bình thường và sự kiện nhấp chuột được tạo ra.

Trong ví dụ này, không có gì đặc biệt về lựa chọn **wx.EVT\_LEFT\_DOWN** và **wx.EVT\_BUTTON**. Tình huống tương tự xảy ra bất cứ khi nào một hàm xử lý được tìm thấy trong cùng một quá trình sự kiện như một hàm khác. Ví dụ, thay đổi sự kiện **wx.EVT\_LEFT\_DOWN** thành một sự kiện **wx.EVT\_BUTTON** khác sẽ có cùng hiệu ứng. Một lệnh gọi **Skip()** vẫn cần thiết để cả hai hàm xử lý được xử lý.

**Các thuộc tính sự kiện khác chứa trong đối tượng ứng dụng**

Để quản lý vòng lặp sự kiện chính một cách trực tiếp hơn, bạn có thể điều chỉnh nó bằng cách sử dụng một số phương thức của **wx.App**. Ví dụ, bạn có thể muốn bắt đầu xử lý sự kiện tiếp theo theo lịch trình của riêng bạn, thay vì đợi wxPython bắt đầu xử lý. Tính năng này cần thiết nếu bạn đang bắt đầu một quy trình dài và không muốn giao diện người dùng (GUI) bị đơ.

Bạn sẽ không cần sử dụng các phương thức trong phần này thường xuyên, nhưng thỉnh thoảng cần có các khả năng này. Bảng 3.4 liệt kê các phương thức của **wx.App** mà bạn có thể sử dụng để điều chỉnh vòng lặp chính.

Bảng 3.4: Các phương thức vòng lặp sự kiện chính của wx.App

|  |  |
| --- | --- |
| Tên phương thức | Mô tả phương thức |
| Dispatch() | Buộc chương trình gửi sự kiện tiếp theo trong hàng đợi sự kiện. Được sử dụng bởi MainLoop(), ví dụ, hoặc trong các vòng lặp sự kiện tùy chỉnh. |
| Pending() | Trả về True nếu có các sự kiện đang chờ trong hàng đợi sự kiện của ứng dụng wxPython. |
| Yield(onlyIfNeeded=False) | Cho phép các sự kiện wxWidgets đang chờ được gửi trong quá trình dài có thể chặn hệ thống hiển thị hoặc cập nhật. Trả về True nếu có các sự kiện đang chờ được xử lý, ngược lại trả về False. Nếu tham số onlyIfNeeded là True, quá trình sẽ nhường nếu thực sự có các sự kiện đang chờ. Nếu tham số là False, thì việc gọi Yield đệ quy là một lỗi.  Cũng có một hàm toàn cục **wx.SafeYield()**, ngăn người dùng nhập dữ liệu trong suốt quá trình nhường (bằng cách tạm thời vô hiệu hóa các widget nhập liệu của người dùng). Điều này ngăn người dùng làm điều gì đó vi phạm trạng thái cần thiết bởi tác vụ đang nhường. |

Ví dụ minh họa:

Giả sử bạn đang viết một ứng dụng wxPython và bạn có một tác vụ dài như tải một tệp lớn. Bạn không muốn giao diện người dùng bị đơ trong quá trình này, nên bạn sử dụng Yield() để cho phép các sự kiện giao diện người dùng khác tiếp tục được xử lý.

|  |
| --- |
| import wx  import time  class MyFrame(wx.Frame):  def \_\_init\_\_(self):  super().\_\_init\_\_(parent=None, title="Yield Example")  panel = wx.Panel(self)  self.button = wx.Button(panel, label="Start Long Task", pos=(50, 50))  self.button.Bind(wx.EVT\_BUTTON, self.on\_button\_click)  self.Show()  def on\_button\_click(self, event):  self.button.Disable() # Vô hiệu hóa nút trong khi thực hiện tác vụ dài  for i in range(10):  print(f"Processing step {i+1}")  time.sleep(1) # Giả lập một tác vụ dài  wx.Yield() # Cho phép các sự kiện khác được xử lý  self.button.Enable() # Kích hoạt lại nút sau khi hoàn thành  if \_\_name\_\_ == "\_\_main\_\_":  app = wx.App(False)  frame = MyFrame()  app.MainLoop() |
| Trong ví dụ này:   * Khi người dùng nhấp vào nút "Start Long Task", nút sẽ bị vô hiệu hóa để ngăn việc nhấp thêm. * Vòng lặp giả lập một tác vụ dài bằng cách tạm dừng trong 1 giây mỗi bước. * Trong mỗi bước, **wx.Yield()** được gọi để cho phép các sự kiện khác (như vẽ lại cửa sổ) được xử lý. * Sau khi hoàn thành tác vụ, nút sẽ được kích hoạt lại.   Phương pháp này giúp đảm bảo rằng giao diện người dùng không bị đơ trong khi thực hiện các tác vụ dài. |

**Tạo sự kiện tùy chỉnh của riêng bạn (How can I create my own events?)**

Mặc dù đây là một chủ đề nâng cao hơn, nhưng việc tạo sự kiện tùy chỉnh là một phần quan trọng trong lập trình event-driven (hướng sự kiện) với wxPython. Đầu tiên, bạn có thể bỏ qua phần này và quay lại sau khi đã hiểu rõ hơn về các khái niệm cơ bản.

Ngoài các lớp sự kiện do wxPython cung cấp, bạn có thể tạo các sự kiện tùy chỉnh của riêng mình. Bạn có thể làm điều này để đáp ứng các cập nhật dữ liệu hoặc các thay đổi khác đặc thù cho ứng dụng của bạn, nơi mà các phiên bản sự kiện cần mang theo dữ liệu tùy chỉnh của bạn. Một lý do khác để tạo lớp sự kiện tùy chỉnh là để hỗ trợ một widget tùy chỉnh với loại sự kiện lệnh độc đáo của nó.

**Ví dụ về tạo sự kiện tùy chỉnh**

Trong phần này, chúng ta sẽ tạo một sự kiện tùy chỉnh và một widget tùy chỉnh để minh họa cách thực hiện điều này trong wxPython.

1. Tạo lớp sự kiện tùy chỉnh

Đầu tiên, chúng ta sẽ tạo một lớp sự kiện tùy chỉnh. Điều này bao gồm định nghĩa một lớp sự kiện mới, thừa kế từ **wx.PyCommandEvent**, và đăng ký một loại sự kiện mới.

|  |
| --- |
| import wx  # Định nghĩa một loại sự kiện mới  myEVT\_CUSTOM = wx.NewEventType()  # Tạo một lớp sự kiện mới, thừa kế từ wx.PyCommandEvent  class MyCustomEvent(wx.PyCommandEvent):  def \_\_init\_\_(self, etype, eid, data=None):  wx.PyCommandEvent.\_\_init\_\_(self, etype, eid)  self.data = data  def GetData(self):  return self.data  # Liên kết sự kiện với lớp sự kiện  EVT\_CUSTOM = wx.PyEventBinder(myEVT\_CUSTOM, 1) |

2. Tạo một widget tùy chỉnh phát ra sự kiện này

Tiếp theo, chúng ta sẽ tạo một widget tùy chỉnh phát ra sự kiện tùy chỉnh khi có một hành động nào đó.

|  |
| --- |
| class CustomButton(wx.Button):  def \_\_init\_\_(self, \*args, \*\*kwargs):  super(CustomButton, self).\_\_init\_\_(\*args, \*\*kwargs)  self.Bind(wx.EVT\_BUTTON, self.on\_click)    def on\_click(self, event):  # Phát ra sự kiện tùy chỉnh khi nút được nhấp  evt = MyCustomEvent(myEVT\_CUSTOM, self.GetId(), "Hello World!")  self.GetEventHandler().ProcessEvent(evt)  class MyFrame(wx.Frame):  def \_\_init\_\_(self):  super().\_\_init\_\_(parent=None, title="Custom Event Example")  panel = wx.Panel(self)  self.button = CustomButton(panel, label="Click Me", pos=(50, 50))  self.Bind(EVT\_CUSTOM, self.on\_custom\_event)  self.Show()  def on\_custom\_event(self, event):  wx.MessageBox(f"Received custom event with data: {event.GetData()}")  if \_\_name\_\_ == "\_\_main\_\_":  app = wx.App(False)  frame = MyFrame()  app.MainLoop() |

**Giải thích:**

1. **Định nghĩa sự kiện tùy chỉnh**:
   * **myEVT\_CUSTOM** là loại sự kiện tùy chỉnh mới.
   * **MyCustomEvent** là lớp sự kiện tùy chỉnh, chứa dữ liệu tùy chỉnh.
   * **EVT\_CUSTOM** liên kết loại sự kiện mới với lớp sự kiện.
2. **Tạo widget tùy chỉnh**:
   * **CustomButton** là một lớp nút tùy chỉnh, thừa kế từ **wx.Button**.
   * Khi nút được nhấp, một sự kiện tùy chỉnh được tạo và phát ra.
3. **Xử lý sự kiện tùy chỉnh**:
   * **MyFrame** là lớp khung chính của ứng dụng.
   * **on\_custom\_event** là phương thức xử lý sự kiện tùy chỉnh, hiển thị hộp thoại khi nhận được sự kiện.

Ví dụ này minh họa cách tạo và xử lý sự kiện tùy chỉnh trong wxPython, giúp bạn có thể mở rộng ứng dụng của mình với các hành vi cụ thể mà các sự kiện tiêu chuẩn không thể đáp ứng.

**Định nghĩa sự kiện tùy chỉnh cho một widget tùy chỉnh**

Chúng ta sẽ tạo một widget bao gồm một panel chứa hai nút bấm. Sự kiện tùy chỉnh, **TwoButtonEvent**, sẽ được kích hoạt chỉ sau khi người dùng đã nhấn cả hai nút. Sự kiện này sẽ chứa số lần người dùng đã nhấn vào widget. Mục đích là để minh họa cách tạo một sự kiện lệnh mới từ các sự kiện nhỏ hơn, trong trường hợp này là các sự kiện nhấn nút trái trên mỗi nút.

**Các bước tạo sự kiện tùy chỉnh:**

1. **Định nghĩa lớp sự kiện mới**: Lớp này là một subclass của **wx.PyEvent** hoặc **wx.PyCommandEvent**.
2. **Tạo kiểu sự kiện và đối tượng binder**: Để liên kết sự kiện với các đối tượng cụ thể.
3. **Viết mã để xây dựng các phiên bản của sự kiện mới**: Đưa các phiên bản này vào hệ thống xử lý sự kiện bằng cách sử dụng phương thức **ProcessEvent()**.

**Ví dụ về việc quản lý widget:**

|  |
| --- |
| import wx  # Bước 1: Định nghĩa lớp sự kiện mới  class TwoButtonEvent(wx.PyCommandEvent):  def \_\_init\_\_(self, evtType, id):  super().\_\_init\_\_(evtType, id)  self.clickCount = 0    def GetClickCount(self):  return self.clickCount    def SetClickCount(self, count):  self.clickCount = count  # Bước 2: Tạo kiểu sự kiện và đối tượng binder  myEVT\_TWO\_BUTTON = wx.NewEventType()  EVT\_TWO\_BUTTON = wx.PyEventBinder(myEVT\_TWO\_BUTTON, 1)  # Bước 3: Tạo widget tùy chỉnh  class TwoButtonPanel(wx.Panel):  def \_\_init\_\_(self, parent, id=-1, leftText="Left", rightText="Right"):  super().\_\_init\_\_(parent, id)  self.leftButton = wx.Button(self, label=leftText)  self.rightButton = wx.Button(self, label=rightText, pos=(100,0))  self.leftClick = False  self.rightClick = False  self.clickCount = 0  self.leftButton.Bind(wx.EVT\_LEFT\_DOWN, self.OnLeftClick)  self.rightButton.Bind(wx.EVT\_LEFT\_DOWN, self.OnRightClick)    def OnLeftClick(self, event):  self.leftClick = True  self.OnClick()  event.Skip()    def OnRightClick(self, event):  self.rightClick = True  self.OnClick()  event.Skip()    def OnClick(self):  self.clickCount += 1  if self.leftClick and self.rightClick:  self.leftClick = False  self.rightClick = False  evt = TwoButtonEvent(myEVT\_TWO\_BUTTON, self.GetId())  evt.SetClickCount(self.clickCount)  self.GetEventHandler().ProcessEvent(evt)  # Tạo khung chính và liên kết sự kiện tùy chỉnh  class CustomEventFrame(wx.Frame):  def \_\_init\_\_(self, parent, id):  super().\_\_init\_\_(parent, id, 'Click Count: 0', size=(300, 100))  panel = TwoButtonPanel(self)  self.Bind(EVT\_TWO\_BUTTON, self.OnTwoClick, panel)    def OnTwoClick(self, event):  self.SetTitle(f"Click Count: {event.GetClickCount()}")  if \_\_name\_\_ == '\_\_main\_\_':  app = wx.App(False)  frame = CustomEventFrame(parent=None, id=wx.ID\_ANY)  frame.Show()  app.MainLoop() |
| 1. **Định nghĩa sự kiện tùy chỉnh**:    * **TwoButtonEvent** là lớp sự kiện tùy chỉnh, chứa số lần nhấn nút.    * **myEVT\_TWO\_BUTTON** là loại sự kiện tùy chỉnh.    * **EVT\_TWO\_BUTTON** liên kết loại sự kiện với lớp sự kiện. 2. **Tạo widget tùy chỉnh**:    * **TwoButtonPanel** là một lớp panel tùy chỉnh, chứa hai nút bấm.    * Các sự kiện nhấn nút trái được liên kết với phương thức **OnLeftClick** và **OnRightClick**. 3. **Xử lý sự kiện tùy chỉnh**:    * **CustomEventFrame** là lớp khung chính của ứng dụng.    * **OnTwoClick** là phương thức xử lý sự kiện tùy chỉnh, thay đổi tiêu đề cửa sổ khi nhận được sự kiện.   Trong ví dụ này, sự kiện tùy chỉnh được tạo ra khi cả hai nút bấm được nhấn, và sự kiện này được xử lý để thay đổi tiêu đề của cửa sổ dựa trên số lần nhấn nút. Đây là cách tạo và xử lý sự kiện tùy chỉnh trong wxPython, giúp mở rộng khả năng của ứng dụng để đáp ứng các yêu cầu cụ thể. |

|  |
| --- |
| 1. **Khai báo lớp sự kiện tùy chỉnh**:    * **TwoButtonEvent** là lớp sự kiện tùy chỉnh, được khai báo như một lớp con của **wx.PyCommandEvent**. Sử dụng **wx.PyEvent** hoặc **wx.PyCommandEvent** giúp kết nối giữa các phương thức viết bằng Python và hệ thống C++ của wxWidgets.    * Nếu bạn sử dụng **wx.Event** trực tiếp, wxPython không thể thấy các phương thức mới của lớp con trong quá trình xử lý sự kiện, vì các trình xử lý sự kiện C++ không biết về lớp con Python. Khi sử dụng **wx.PyEvent**, một tham chiếu đến các instance Python được lưu và sau đó được truyền trực tiếp đến trình xử lý sự kiện, cho phép các phần của Python được sử dụng. 2. **Tạo loại sự kiện và đối tượng binder**:    * Hàm toàn cục **wx.NewEventType()** trả về một ID loại sự kiện đảm bảo duy nhất, tương tự như **wx.NewId()**. Giá trị này xác định duy nhất một loại sự kiện cho hệ thống xử lý sự kiện.    * Đối tượng binder được tạo bằng cách sử dụng loại sự kiện mới làm tham số. Tham số thứ hai từ 0 đến 2, đại diện cho số lượng ID wx được phương thức **wx.EvtHandler.Bind()** mong đợi để xác định đối tượng nào là nguồn của sự kiện. Trong trường hợp này, có một ID đại diện cho widget tạo ra sự kiện lệnh. 3. **Tạo sự kiện lệnh cấp cao mới**:    * Chương trình phải phản hồi các sự kiện người dùng cụ thể, ví dụ như nhấn chuột trái trên mỗi đối tượng nút.    * Tùy thuộc vào nút hiển thị được nhấp, các sự kiện được liên kết với các phương thức **OnLeftClick()** và **OnRightClick()**. Các trình xử lý này đặt một giá trị Boolean, chỉ ra rằng nút đã được nhấn. 4. **Gọi Skip()**:    * Lời gọi **Skip()** cho phép xử lý thêm sau khi trình xử lý sự kiện hoàn tất. Trong trường hợp này, sự kiện mới không yêu cầu lời gọi skip; nó được gửi đi trước khi phương thức trình xử lý hoàn tất. Tuy nhiên, tất cả các sự kiện nhấn chuột trái cần gọi **Skip()** để trình xử lý không chặn sự kiện nhấn nút cuối cùng. 5. **Xử lý sự kiện tùy chỉnh**:    * Gọi **ProcessEvent()** để đưa sự kiện mới vào hệ thống xử lý sự kiện.    * **GetEventHandler()** trả về một instance của **wx.EvtHandler**. |

**Tóm tắt**

**Kiến trúc ứng dụng wxPython**

* **Ứng dụng wxPython sử dụng luồng điều khiển dựa trên sự kiện**: Phần lớn thời gian của ứng dụng được dành cho vòng lặp chính, chờ đợi các sự kiện và gửi chúng tới các hàm xử lý thích hợp.
* **Tất cả các sự kiện wxPython là các lớp con của lớp wx.Event**: Các sự kiện cấp thấp, như nhấp chuột, được sử dụng để tạo các sự kiện cấp cao hơn, như nhấp nút hoặc chọn mục menu. Các sự kiện cấp cao này, xuất phát từ các widget wxPython, là các lớp con của lớp wx.CommandEvent. Phần lớn các lớp sự kiện được phân loại thêm bằng một trường loại sự kiện để phân biệt giữa các sự kiện có thể sử dụng cùng một tập dữ liệu.

**Xử lý và liên kết sự kiện**

* **wxPython sử dụng các instance của lớp wx.PyEventBinder để liên kết sự kiện với hàm xử lý**: Có nhiều instance được định nghĩa trước của lớp này, mỗi instance tương ứng với một loại sự kiện cụ thể. Mỗi widget wxPython là một lớp con của wx.EvtHandler. Lớp wx.EvtHandler có một phương thức Bind(), thường được gọi khi khởi tạo với một instance của event binder và một hàm xử lý làm tham số. Tùy thuộc vào loại sự kiện, các ID của đối tượng wxPython khác cũng có thể cần được truyền vào lời gọi Bind().
* **Các sự kiện thường được gửi đến đối tượng đã tạo ra chúng để tìm kiếm một đối tượng liên kết sự kiện với hàm xử lý**: Nếu sự kiện là một sự kiện lệnh (command event), sự kiện sẽ được truyền lên trên qua hệ thống các container cho đến khi tìm thấy một widget có hàm xử lý cho loại sự kiện đó. Khi tìm thấy một hàm xử lý sự kiện, quá trình xử lý sự kiện dừng lại, trừ khi hàm xử lý gọi phương thức Skip() của sự kiện. Bạn có thể sử dụng phương thức Skip() để cho phép nhiều hàm xử lý đáp ứng với một sự kiện duy nhất hoặc để đảm bảo rằng tất cả hành vi mặc định của sự kiện đều xảy ra. Một số khía cạnh của vòng lặp chính có thể được kiểm soát bằng cách sử dụng các phương thức của wx.App.

**Tạo sự kiện tùy chỉnh**

* **Có thể tạo sự kiện tùy chỉnh trong wxPython và phát ra chúng như một phần của hành vi của một widget tùy chỉnh**: Các sự kiện tùy chỉnh là các lớp con của wx.PyEvent, và các sự kiện lệnh tùy chỉnh là các lớp con của wx.PyCommandEvent. Để tạo một sự kiện tùy chỉnh, cần định nghĩa lớp mới và tạo một đối tượng binder cho mỗi loại sự kiện được quản lý bởi lớp mới. Cuối cùng, sự kiện phải được tạo ra ở đâu đó trong hệ thống bằng cách truyền một instance mới vào hệ thống xử lý sự kiện qua phương thức ProcessEvent().

**Kết luận**

* Trong chương này, chúng ta đã tìm hiểu về các đối tượng ứng dụng quan trọng nhất đối với ứng dụng wxPython của bạn.
* Trong chương tiếp theo, chúng ta sẽ giới thiệu một công cụ hữu ích, được viết bằng wxPython, sẽ hỗ trợ bạn trong công việc phát triển wxPython.

**Tạo bản vẽ thiết kế của bạn (Creating your blueprint)**

|  |
| --- |
| Abstract:  Refactoring and how it improves code.  ■ Separating the Model and View  ■ Using a model class  ■ Unit testing a GUI program  ■ Testing user events |

**Kiến Trúc và Bố Cục Code Giao Diện Người Dùng (GUI)**

Lập trình giao diện người dùng (GUI) thường bị cho là khó đọc, khó duy trì và thường như một "mớ bòng bong" - dài dòng và rối rắm. Có một module Python GUI nổi tiếng (không viết bằng wxPython) đã bao gồm một ghi chú trong phần nhận xét của nó: “Tại sao mã GUI luôn kết thúc nhìn như một mớ bòng bong, mặc dù có tất cả ý định tốt nhất để giữ nó gọn gàng?” Tuy nhiên, điều đó không phải lúc nào cũng đúng. Không có lý do cụ thể nào khiến mã giao diện người dùng phải khó viết hoặc quản lý hơn bất kỳ phần nào khác của chương trình.

Trong chương này, chúng ta sẽ thảo luận về ba kỹ thuật để làm cho mã GUI của bạn trở nên dễ quản lý hơn. Bởi vì mã bố cục giao diện dễ bị cấu trúc kém, chúng ta sẽ thảo luận về việc tái cấu trúc mã để làm cho nó dễ đọc, quản lý và duy trì hơn. Một lĩnh vực khác mà lập trình viên giao diện người dùng có thể gặp khó khăn là sự tương tác giữa mã hiển thị và các đối tượng nghiệp vụ cơ bản. Mẫu thiết kế Model/View/Controller (MVC) là một cấu trúc để giữ riêng biệt giữa hiển thị và dữ liệu để mỗi phần có thể thay đổi mà không ảnh hưởng đến phần kia. Cuối cùng, chúng ta sẽ thảo luận về các kỹ thuật để kiểm thử đơn vị mã wxPython của bạn.

Mặc dù tất cả các ví dụ trong chương này sẽ sử dụng wxPython, nhiều nguyên tắc có thể áp dụng cho bất kỳ bộ công cụ giao diện người dùng nào – mặc dù ngôn ngữ Python và bộ công cụ wxPython làm cho một số kỹ thuật trở nên đặc biệt thanh lịch. Thiết kế và kiến trúc của mã của bạn là bản thiết kế của hệ thống của bạn. Một bản thiết kế được suy nghĩ kỹ lưỡng sẽ làm cho ứng dụng của bạn dễ xây dựng hơn và dễ duy trì hơn. Các gợi ý trong chương này sẽ giúp bạn thiết kế một bản thiết kế vững chắc cho chương trình của bạn.

**Nguyên Tắc Tái Cấu Trúc**

Trên nguyên tắc, không khó để giữ mã giao diện người dùng trong tầm kiểm soát. Chìa khóa là tái cấu trúc, hoặc liên tục cải thiện thiết kế và cấu trúc của mã hiện có. Mục tiêu của tái cấu trúc là giữ mã ở trạng thái mà nó có thể dễ dàng đọc và duy trì trong tương lai. Bảng 5.1 chứa mô tả về một số nguyên tắc cần nhớ khi tái cấu trúc. Mục tiêu cơ bản nhất là nhớ rằng ai đó sẽ phải đọc và hiểu mã của bạn trong tương lai. Hãy cố gắng làm cho cuộc sống của người đó dễ dàng hơn – sau tất cả, đó có thể là bạn.

**Bảng 5.1: Các Nguyên Tắc Quan Trọng Trong Tái Cấu Trúc**

|  |  |
| --- | --- |
| **Nguyên Tắc** | **Mô Tả** |
| **Không trùng lặp** | Bạn nên tránh có nhiều đoạn mã có cùng chức năng. Điều này có thể gây ra vấn đề bảo trì khi cần thay đổi chức năng. |
| **Làm một việc một lúc** | Một phương thức nên làm một việc, và chỉ một việc duy nhất. Các việc riêng biệt nên được chuyển vào các phương thức riêng biệt. Các phương thức nên được giữ ngắn gọn. |
| **Xây dựng các mức nông** | Tránh lồng ghép mã quá hai hoặc ba mức. Mã lồng sâu cũng là ứng cử viên tốt cho một phương thức riêng biệt. |
| **Tránh các hằng số ma thuật** | Các hằng số chuỗi và số nên được giữ ở mức tối thiểu. Một cách tốt để quản lý điều này là tách dữ liệu hằng khỏi phần chính của mã và lưu trữ nó trong một danh sách hoặc từ điển. |

Một số nguyên tắc này đặc biệt quan trọng trong mã Python. Vì cú pháp dựa trên thụt lề của Python, các phương thức nhỏ gọn, ngắn rất dễ đọc. Tuy nhiên, các phương thức dài có thể khó giải mã hơn, đặc biệt nếu chúng không thể hiển thị trên một màn hình duy nhất. Tương tự, việc lồng sâu trong Python có thể làm cho việc truy vết đầu và cuối của các khối mã trở nên khó khăn. Tuy nhiên, Python là một ngôn ngữ đặc biệt tốt để tránh trùng lặp, đặc biệt là vì sự dễ dàng trong việc truyền các hàm và phương thức làm tham số.

**5.1.1 Ví dụ về Tái Cấu Trúc**

Để cho bạn thấy các nguyên tắc này hoạt động như thế nào trong thực tế, chúng ta sẽ cùng đi qua một ví dụ về tái cấu trúc. Hình 5.1 cho thấy một cửa sổ có thể được sử dụng làm giao diện phía trước cho một cơ sở dữ liệu giống như Microsoft Access. Bố cục này phức tạp hơn một chút so với những gì chúng ta đã thấy cho đến nay, nhưng theo tiêu chuẩn của các ứng dụng thực tế, nó vẫn khá đơn giản. Danh sách 5.1 cho thấy một cách cấu trúc kém để tạo ra Hình 5.1. Khi người ta nói về mã giao diện người dùng lộn xộn, đây là những gì họ muốn nói.

A screenshot of a computer

Description automatically generated

Danh sách 5.1: Một cách chưa tái cấu trúc để tạo ra Hình 5.1

|  |
| --- |
| #!/usr/bin/env python  import wx  class RefactorExample(wx.Frame):  def \_\_init\_\_(self, parent, id):  wx.Frame.\_\_init\_\_(self, parent, id, 'Refactor Example', size=(340, 200))  panel = wx.Panel(self, -1)  panel.SetBackgroundColour("White")  prevButton = wx.Button(panel, -1, "<< PREV", pos=(80, 0))  self.Bind(wx.EVT\_BUTTON, self.OnPrev, prevButton)  nextButton = wx.Button(panel, -1, "NEXT >>", pos=(160, 0))  self.Bind(wx.EVT\_BUTTON, self.OnNext, nextButton)  self.Bind(wx.EVT\_CLOSE, self.OnCloseWindow)  menuBar = wx.MenuBar()  menu1 = wx.Menu()  openMenuItem = menu1.Append(-1, "&Open", "Copy in status bar")  self.Bind(wx.EVT\_MENU, self.OnOpen, openMenuItem)  quitMenuItem = menu1.Append(-1, "&Quit", "Quit")  self.Bind(wx.EVT\_MENU, self.OnCloseWindow, quitMenuItem)  menuBar.Append(menu1, "&File")  menu2 = wx.Menu()  copyItem = menu2.Append(-1, "&Copy", "Copy")  self.Bind(wx.EVT\_MENU, self.OnCopy, copyItem)  cutItem = menu2.Append(-1, "C&ut", "Cut")  self.Bind(wx.EVT\_MENU, self.OnCut, cutItem)  pasteItem = menu2.Append(-1, "Paste", "Paste")  self.Bind(wx.EVT\_MENU, self.OnPaste, pasteItem)  menuBar.Append(menu2, "&Edit")    self.SetMenuBar(menuBar)  static = wx.StaticText(panel, wx.NewId(), "First Name", pos=(10, 50))  static.SetBackgroundColour("White")  text = wx.TextCtrl(panel, wx.NewId(), "", size=(100, -1), pos=(80, 50))  static2 = wx.StaticText(panel, wx.NewId(), "Last Name", pos=(10, 80))  static2.SetBackgroundColour("White")  text2 = wx.TextCtrl(panel, wx.NewId(), "", size=(100, -1), pos=(80, 80))  firstButton = wx.Button(panel, -1, "FIRST")  self.Bind(wx.EVT\_BUTTON, self.OnFirst, firstButton)  menu2.AppendSeparator()  optItem = menu2.Append(-1, "&Options...", "Display Options")  self.Bind(wx.EVT\_MENU, self.OnOptions, optItem)  lastButton = wx.Button(panel, -1, "LAST", pos=(240, 0))  self.Bind(wx.EVT\_BUTTON, self.OnLast, lastButton)  def OnPrev(self, event): pass  def OnNext(self, event): pass  def OnLast(self, event): pass  def OnFirst(self, event): pass  def OnOpen(self, event): pass  def OnCopy(self, event): pass  def OnCut(self, event): pass  def OnPaste(self, event): pass  def OnOptions(self, event): pass  def OnCloseWindow(self, event):  self.Destroy()  if \_\_name\_\_ == '\_\_main\_\_':  app = wx.PySimpleApp()  frame = RefactorExample(parent=None, id=-1)  frame.Show()  app.MainLoop() |

**Đánh giá theo các nguyên tắc trong bảng 5.1**

Khi phân tích mã ví dụ này theo các nguyên tắc trong bảng 5.1, chúng ta thấy rằng:

* **Không lồng sâu**: Không có lồng sâu trong mã này.
* **Không trùng lặp**: Có nhiều đoạn mã bị lặp lại, ví dụ như “thêm một nút và gán hành động cho nó,” “thêm một mục menu và gán hành động cho nó,” và “tạo một cặp chú thích/nhập liệu.”
* **Làm một việc một lần**: Mã này thực hiện nhiều việc khác nhau. Ngoài việc thiết lập khung cơ bản, nó còn tạo thanh menu, thêm các nút, và thêm các trường văn bản. Tệ hơn, ba chức năng này bị trộn lẫn trong mã, như thể các thay đổi muộn được thêm vào cuối phương thức.
* **Tránh các hằng số ma thuật**: Mỗi nút, mục menu và hộp văn bản đều có một chuỗi hằng và một điểm hằng trong constructor.

**Cải thiện mã**

Để khắc phục những vấn đề này, chúng ta sẽ tách mã tạo các nút thành một phương thức riêng. Dưới đây là một ví dụ về cách tái cấu trúc phần này:

|  |
| --- |
| #!/usr/bin/env python  import wx  class RefactorExample(wx.Frame):  def \_\_init\_\_(self, parent, id):  wx.Frame.\_\_init\_\_(self, parent, id, 'Refactor Example', size=(340, 200))  panel = wx.Panel(self, -1)  panel.SetBackgroundColour("White")  self.Bind(wx.EVT\_CLOSE, self.OnCloseWindow)  self.createMenuBar()  self.createButtonBar(panel)  self.createTextFields(panel)  def menuData(self):  return [  ("&File", [  ("&Open", "Open in status bar", self.OnOpen),  ("&Quit", "Quit", self.OnCloseWindow)  ]),  ("&Edit", [  ("&Copy", "Copy", self.OnCopy),  ("C&ut", "Cut", self.OnCut),  ("&Paste", "Paste", self.OnPaste),  ("", "", None),  ("&Options...", "Display Options", self.OnOptions)  ])  ]  def createMenuBar(self):  menuBar = wx.MenuBar()  for menuLabel, menuItems in self.menuData():  menuBar.Append(self.createMenu(menuItems), menuLabel)  self.SetMenuBar(menuBar)  def createMenu(self, menuData):  menu = wx.Menu()  for label, status, handler in menuData:  if not label:  menu.AppendSeparator()  else:  menuItem = menu.Append(-1, label, status)  if handler:  self.Bind(wx.EVT\_MENU, handler, menuItem)  return menu  def buttonData(self):  return [  ("First", self.OnFirst),  ("<< PREV", self.OnPrev),  ("NEXT >>", self.OnNext),  ("Last", self.OnLast)  ]  def createButtonBar(self, panel, yPos=0):  xPos = 0  for label, handler in self.buttonData():  button = self.buildOneButton(panel, label, handler, pos=(xPos, yPos))  xPos += button.GetSize().width  def buildOneButton(self, parent, label, handler, pos=(0, 0)):  button = wx.Button(parent, -1, label, pos)  self.Bind(wx.EVT\_BUTTON, handler, button)  return button  def textFieldData(self):  return [  ("First Name", (10, 50)),  ("Last Name", (10, 80))  ]  def createTextFields(self, panel):  for label, pos in self.textFieldData():  self.createCaptionedText(panel, label, pos)  def createCaptionedText(self, panel, label, pos):  static = wx.StaticText(panel, wx.NewId(), label, pos)  static.SetBackgroundColour("White")  textPos = (pos[0] + 75, pos[1])  wx.TextCtrl(panel, wx.NewId(), "", size=(100, -1), pos=textPos)  def OnPrev(self, event): pass  def OnNext(self, event): pass  def OnLast(self, event): pass  def OnFirst(self, event): pass  def OnOpen(self, event): pass  def OnCopy(self, event): pass  def OnCut(self, event): pass  def OnPaste(self, event): pass  def OnOptions(self, event): pass  def OnCloseWindow(self, event):  self.Destroy()  if \_\_name\_\_ == '\_\_main\_\_':  app = wx.App(False)  frame = RefactorExample(parent=None, id=-1)  frame.Show()  app.MainLoop() |
| **Giải thích:**   1. **Phương thức menuData**: Trả về dữ liệu của các menu dưới dạng danh sách các tuple. Mỗi tuple đại diện cho một menu và danh sách các mục trong menu đó. 2. **Phương thức createMenuBar**: Tạo thanh menu và thêm các menu vào thanh menu. 3. **Phương thức createMenu**: Tạo menu và thêm các mục vào menu. 4. **Phương thức buttonData**: Trả về dữ liệu của các nút dưới dạng danh sách các tuple. Mỗi tuple đại diện cho một nút và hàm xử lý sự kiện của nó. 5. **Phương thức createButtonBar**: Tạo thanh nút và đặt các nút vào thanh nút. 6. **Phương thức buildOneButton**: Tạo một nút và gán sự kiện cho nó. 7. **Phương thức textFieldData**: Trả về dữ liệu của các trường văn bản dưới dạng danh sách các tuple. Mỗi tuple đại diện cho một trường văn bản và vị trí của nó. 8. **Phương thức createTextFields**: Tạo các trường văn bản. 9. **Phương thức createCaptionedText**: Tạo một trường văn bản với nhãn.   Các phương thức sự kiện như **OnPrev**, **OnNext**, **OnLast**, **OnFirst**, **OnOpen**, **OnCopy**, **OnCut**, **OnPaste**, và **OnOptions** hiện tại đang trống, bạn có thể thêm logic vào các phương thức này dựa trên nhu cầu của bạn. |

**Tóm tắt**

Qua ví dụ này, chúng ta đã áp dụng các nguyên tắc tái cấu trúc để cải thiện mã gốc:

1. **Tách các chức năng riêng biệt**: Tạo các phương thức **create\_buttons**, **create\_menu\_bar**, và **create\_text\_fields** để tách rời mã thiết lập giao diện.
2. **Tránh trùng lặp**: Sử dụng vòng lặp để tạo các nút và các trường văn bản, giảm thiểu sự lặp lại trong mã.
3. **Tránh hằng số ma thuật**: Mã được cải thiện để tránh việc sử dụng quá nhiều hằng số trực tiếp, giúp mã dễ duy trì hơn.

Nhờ vào việc tái cấu trúc, mã trở nên dễ đọc, dễ bảo trì và dễ hiểu hơn nhiều.

**5.2 Làm thế nào để giữ Model và View tách biệt trong chương trình của tôi? (How do I keep the Model and View separate in my program?)**

MVC (Model-View-Controller) là một mẫu thiết kế cổ điển trong lập trình hướng đối tượng, xuất hiện từ cuối những năm 1970. Nó là tiêu chuẩn để cấu trúc các chương trình vừa xử lý vừa hiển thị thông tin. MVC giúp tách biệt các phần khác nhau của ứng dụng để dễ quản lý và bảo trì.

**5.2.1 Hệ thống Model-View-Controller là gì?**

Một hệ thống MVC gồm ba phần chính:

* **Model**: Chứa logic nghiệp vụ (business logic), tức là tất cả dữ liệu và thông tin mà hệ thống thao tác. Model thường chỉ công khai các API để các thành phần khác sử dụng và có thể bao gồm giao diện với các kho lưu trữ bên ngoài như cơ sở dữ liệu.
* **View**: Chứa mã hiển thị dữ liệu. Đây là các widget hiển thị thông tin cho người dùng. Trong wxPython, gần như bất kỳ đối tượng nào trong hệ thống phân cấp wx.Window đều thuộc về View.
* **Controller**: Chứa logic tương tác. Đây là mã nhận sự kiện từ người dùng và đảm bảo chúng được hệ thống xử lý. Trong wxPython, phần này được đại diện bởi hệ thống phân cấp wx.EvtHandler.

Bảng dưới đây tóm tắt các thành phần:

|  |  |
| --- | --- |
| **Thành phần** | **Mô tả** |
| **Model** | Logic nghiệp vụ. Chứa tất cả dữ liệu mà hệ thống thao tác. |
| **View** | Mã hiển thị. Các widget hiển thị thông tin cho người dùng. |
| **Controller** | Logic tương tác. Mã nhận sự kiện từ người dùng và đảm bảo chúng được xử lý. |

Trong nhiều bộ công cụ UI hiện đại, View và Controller thường bị đan xen vì các thành phần Controller cần được hiển thị trên màn hình và các widget hiển thị dữ liệu cũng cần phản hồi sự kiện của người dùng. Trong wxPython, mối quan hệ này được xác định bởi việc tất cả các đối tượng wx.Window cũng là các lớp con của wx.EvtHandler, tức là chúng hoạt động như cả phần tử View và phần tử Controller.

**Luồng dữ liệu trong kiến trúc MVC**

Hình dưới đây minh họa cách dữ liệu và thông tin được truyền trong một kiến trúc MVC.

|  |
| --- |
|  |

Một thông báo sự kiện được xử lý bởi hệ thống Controller, sau đó phân phối nó đến nơi thích hợp. Trong wxPython, cơ chế này được quản lý bằng phương thức wx.EvtHandler.ProcessEvent(). Trong thiết kế MVC nghiêm ngặt, các hàm xử lý sự kiện có thể được khai báo trong một đối tượng controller riêng biệt, thay vì trong lớp frame chính.

Khi sự kiện được xử lý, các đối tượng model có thể thực hiện một số xử lý trên dữ liệu ứng dụng. Khi việc xử lý xong, model sẽ gửi một thông báo cập nhật. Nếu có một đối tượng controller, thông báo thường được gửi lại cho controller và controller thông báo cho các đối tượng view thích hợp để cập nhật. Trong hệ thống nhỏ hơn hoặc kiến trúc đơn giản hơn, thông báo thường được nhận trực tiếp bởi các đối tượng view.

Trong wxPython, bản chất của cập nhật từ model tùy thuộc vào bạn. Các tùy chọn bao gồm:

* Tạo ra các sự kiện wxPython tùy chỉnh từ model hoặc controller.
* Để model duy trì danh sách các đối tượng nhận thông báo cập nhật.
* Cho phép các view tự đăng ký với model.

**Mục tiêu của thiết kế MVC thành công**

Mục tiêu của thiết kế MVC không phải là để mọi đối tượng biết về mọi đối tượng khác. Thay vào đó, một chương trình MVC thành công sẽ che giấu kiến thức về một phần của chương trình khỏi các phần khác. Hệ thống nên tương tác tối thiểu và qua một tập hợp các phương thức được xác định rõ ràng. Đặc biệt, thành phần Model nên hoàn toàn cách ly khỏi View và Controller. Bạn nên có thể thực hiện các thay đổi tùy ý đối với hai hệ thống đó mà không cần thay đổi các lớp Model.

Về phía View, bạn cũng nên có thể thực hiện các thay đổi tùy ý trong việc triển khai các đối tượng Model mà không cần thay đổi View hoặc Controller. Mặc dù View sẽ phụ thuộc vào sự tồn tại của các phương thức công khai nhất định, nó không nên thấy các thành phần nội bộ của Model.

Một cách để giúp thực thi điều này là tạo một lớp Model trừu tượng xác định API mà View có thể thấy. Các lớp con của Model có thể hoạt động như các đại diện cho một lớp nội bộ có thể thay đổi hoặc có thể chứa chính các hoạt động nội bộ. Tùy chọn đầu tiên có cấu trúc hơn, tùy chọn thứ hai dễ triển khai hơn.

Tiếp theo, chúng ta sẽ xem xét một trong những lớp Model được tích hợp sẵn trong wxPython, wx.grid.PyGridTableBase. Lớp này cho phép sử dụng điều khiển lưới trong một khung thiết kế MVC. Sau đó, chúng ta sẽ xem xét việc xây dựng và sử dụng một lớp model tùy chỉnh cho một widget tùy chỉnh.

**A wxPython Model: PyGridTableBase**

**Giới thiệu về wx.grid.Grid và PyGridTableBase**

Lớp **wx.grid.Grid** trong wxPython cung cấp một điều khiển giao diện người dùng kiểu bảng tính với các hàng và cột. Hình 5.3 minh họa cách một lưới **wxPython** trông như thế nào.

|  |
| --- |
|  |

Lưới **wx.grid.Grid** có nhiều tính năng thú vị, bao gồm khả năng tạo các bộ hiển thị và chỉnh sửa tùy chỉnh cho từng ô, cũng như kéo thả các hàng và cột. Chúng ta sẽ tập trung vào các tính năng cơ bản và cách sử dụng một mô hình để điền dữ liệu vào lưới.

**Cách điền dữ liệu vào lưới không sử dụng mô hình**

Danh sách 5.6 minh họa cách điền giá trị vào lưới mà không sử dụng mô hình. Trong ví dụ này, giá trị lưới là danh sách cầu thủ của đội bóng chày Chicago Cubs năm 1984.

|  |
| --- |
| import wx  import wx.grid  class SimpleGrid(wx.grid.Grid):  def \_\_init\_\_(self, parent):  wx.grid.Grid.\_\_init\_\_(self, parent, -1)  self.CreateGrid(9, 2)  self.SetColLabelValue(0, "First")  self.SetColLabelValue(1, "Last")  self.SetRowLabelValue(0, "CF")  self.SetCellValue(0, 0, "Bob")  self.SetCellValue(0, 1, "Dernier")  self.SetRowLabelValue(1, "2B")  self.SetCellValue(1, 0, "Ryne")  self.SetCellValue(1, 1, "Sandberg")  self.SetRowLabelValue(2, "LF")  self.SetCellValue(2, 0, "Gary")  self.SetCellValue(2, 1, "Matthews")  self.SetRowLabelValue(3, "1B")  self.SetCellValue(3, 0, "Leon")  self.SetCellValue(3, 1, "Durham")  self.SetRowLabelValue(4, "RF")  self.SetCellValue(4, 0, "Keith")  self.SetCellValue(4, 1, "Moreland")  self.SetRowLabelValue(5, "3B")  self.SetCellValue(5, 0, "Ron")  self.SetCellValue(5, 1, "Cey")  self.SetRowLabelValue(6, "C")  self.SetCellValue(6, 0, "Jody")  self.SetCellValue(6, 1, "Davis")  self.SetRowLabelValue(7, "SS")  self.SetCellValue(7, 0, "Larry")  self.SetCellValue(7, 1, "Bowa")  self.SetRowLabelValue(8, "P")  self.SetCellValue(8, 0, "Rick")  self.SetCellValue(8, 1, "Sutcliffe")  class TestFrame(wx.Frame):  def \_\_init\_\_(self, parent):  wx.Frame.\_\_init\_\_(self, parent, -1, "A Grid", size=(275, 275))  grid = SimpleGrid(self)  if \_\_name\_\_ == '\_\_main\_\_':  app = wx.PySimpleApp()  frame = TestFrame(None)  frame.Show(True)  app.MainLoop() |

**Sử dụng mô hình PyGridTableBase**

**wx.grid.PyGridTableBase** là một lớp mô hình cho lưới, cung cấp các phương thức mà đối tượng lưới có thể sử dụng để vẽ chính nó mà không cần biết về cấu trúc nội bộ của dữ liệu.

**Các phương thức cần thiết của PyGridTableBase**

Lớp **wx.grid.PyGridTableBase** có nhiều phương thức, nhưng bạn chỉ cần triển khai năm phương thức bắt buộc khi tạo một lớp con của **PyGridTableBase**. Bảng 5.4 mô tả các phương thức này.

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| GetNumberRows() | Trả về số nguyên chỉ ra số hàng trong lưới. |
| GetNumberCols() | Trả về số nguyên chỉ ra số cột trong lưới. |
| IsEmptyCell(row, col) | Trả về True nếu ô tại chỉ số (row, col) trống. |
| GetValue(row, col) | Trả về giá trị hiển thị tại ô (row, col). |
| SetValue(row, col, value) | Đặt giá trị cho ô (row, col). Nếu bạn muốn mô hình chỉ đọc, bạn vẫn phải bao gồm phương thức này, nhưng có thể để trống. |

**Sử dụng một lớp con PyGridTableBase cụ thể cho ứng dụng**

Danh sách 5.7 minh họa một ví dụ sử dụng lớp con cụ thể của **PyGridTableBase** làm mô hình cho ứng dụng của chúng ta. Chúng ta sẽ thiết lập dữ liệu thực tế trong một danh sách Python hai chiều và thiết lập các phương thức để đọc từ danh sách đó.

|  |
| --- |
| import wx  import wx  import wx.grid  class LineupTable(wx.grid.GridTableBase):  data = (("CF", "Bob", "Dernier"), ("2B", "Ryne", "Sandberg"),  ("LF", "Gary", "Matthews"), ("1B", "Leon", "Durham"),  ("RF", "Keith", "Moreland"), ("3B", "Ron", "Cey"),  ("C", "Jody", "Davis"), ("SS", "Larry", "Bowa"),  ("P", "Rick", "Sutcliffe"))  colLabels = ("First", "Last")    def \_\_init\_\_(self):  wx.grid.GridTableBase.\_\_init\_\_(self)    def GetNumberRows(self):  return len(self.data)    def GetNumberCols(self):  return len(self.data[0]) - 1    def GetColLabelValue(self, col):  return self.colLabels[col]    def GetRowLabelValue(self, row):  return self.data[row][0]    def IsEmptyCell(self, row, col):  return False    def GetValue(self, row, col):  return self.data[row][col + 1]    def SetValue(self, row, col, value):  pass  class SimpleGrid(wx.grid.Grid):  def \_\_init\_\_(self, parent):  wx.grid.Grid.\_\_init\_\_(self, parent, -1)  self.SetTable(LineupTable(), takeOwnership=True)  class TestFrame(wx.Frame):  def \_\_init\_\_(self, parent):  wx.Frame.\_\_init\_\_(self, parent, -1, "A Grid", size=(275, 275))  grid = SimpleGrid(self)  self.Show()  if \_\_name\_\_ == '\_\_main\_\_':  app = wx.App(False) # False to not redirect stdout/stderr  frame = TestFrame(None)  app.MainLoop() |
| Dưới đây là những thay đổi và điều chỉnh được thực hiện:   1. **Thay đổi thứ tự các cột dữ liệu**: Đổi **colLabels** từ **("Last", "First")** thành **("First", "Last")** để phù hợp với các giá trị được thiết lập trong bảng. 2. **Sửa lỗi khởi tạo wx.App()**: Sử dụng **wx.App(False)** thay vì **wx.PySimpleApp()** vì **wx.PySimpleApp()** đã bị loại bỏ trong các phiên bản wxPython mới hơn. 3. **Sử dụng takeOwnership=True khi gọi SetTable**: Để đảm bảo rằng lưới sẽ quản lý bộ nhớ của bảng dữ liệu. |

**Sử dụng PyGridTableBase: Ví dụ tổng quát**

Danh sách 5.8 minh họa một mô hình tổng quát cho các danh sách hai chiều.

|  |
| --- |
| import wx  import wx.grid  class GenericTable(wx.grid.GridTableBase):  def \_\_init\_\_(self, data, rowLabels=None, colLabels=None):  wx.grid.GridTableBase.\_\_init\_\_(self)  self.data = data  self.rowLabels = rowLabels  self.colLabels = colLabels    def GetNumberRows(self):  return len(self.data)    def GetNumberCols(self):  return len(self.data[0])    def GetColLabelValue(self, col):  if self.colLabels:  return self.colLabels[col]  return ""    def GetRowLabelValue(self, row):  if self.rowLabels:  return self.rowLabels[row]  return ""    def IsEmptyCell(self, row, col):  return False    def GetValue(self, row, col):  return self.data[row][col]    def SetValue(self, row, col, value):  pass  class SimpleGrid(wx.grid.Grid):  def \_\_init\_\_(self, parent):  wx.grid.Grid.\_\_init\_\_(self, parent, -1)  data = [  ["Bob", "Dernier"],  ["Ryne", "Sandberg"],  ["Gary", "Matthews"],  ["Leon", "Durham"],  ["Keith", "Moreland"],  ["Ron", "Cey"],  ["Jody", "Davis"],  ["Larry", "Bowa"],  ["Rick", "Sutcliffe"]  ]  colLabels = ["First", "Last"]  rowLabels = ["CF", "2B", "LF", "1B", "RF", "3B", "C", "SS", "P"]  table = GenericTable(data, rowLabels, colLabels)  self.SetTable(table, takeOwnership=True)  class TestFrame(wx.Frame):  def \_\_init\_\_(self, parent):  wx.Frame.\_\_init\_\_(self, parent, -1, "A Grid", size=(275, 275))  grid = SimpleGrid(self)  self.Show()  if \_\_name\_\_ == '\_\_main\_\_':  app = wx.App(False) # False to not redirect stdout/stderr  frame = TestFrame(None)  app.MainLoop() |
| **Các thay đổi chính:**   1. **Thay đổi lớp PyGridTableBase thành GridTableBase**:    * **PyGridTableBase** đã được thay thế bằng **GridTableBase** trong các phiên bản wxPython mới hơn. 2. **Khởi tạo wx.App**:    * Sử dụng **wx.App(False)** thay vì **wx.PySimpleApp()** vì **wx.PySimpleApp()** đã bị loại bỏ trong các phiên bản wxPython mới hơn. 3. **Thêm giá trị mặc định cho GetColLabelValue và GetRowLabelValue**:    * Nếu **colLabels** hoặc **rowLabels** không được cung cấp, trả về chuỗi rỗng để tránh lỗi. 4. **Sử dụng takeOwnership=True khi gọi SetTable**:    * Đảm bảo rằng lưới sẽ quản lý bộ nhớ của bảng dữ liệu.   Với các thay đổi này, mã sẽ chạy và tạo ra một cửa sổ chứa bảng dữ liệu. |

Lớp **GenericTable** nhận một danh sách hai chiều của dữ liệu và danh sách tùy chọn các nhãn hàng và/hoặc cột. Nó có thể được sử dụng trong bất kỳ chương trình wxPython nào.

**Sử dụng GridTableBase: Mô hình lớp dữ liệu riêng biệt**

Danh sách 5.10 minh họa cách sử dụng **GridTableBase** với một lớp dữ liệu riêng biệt.

|  |
| --- |
| import wx  import wx.grid  class LineupEntry:  def \_\_init\_\_(self, pos, first, last):  self.pos = pos  self.first = first  self.last = last  class LineupTable(wx.grid.GridTableBase):  colLabels = ("First", "Last")  colAttrs = ("first", "last")    def \_\_init\_\_(self, entries):  wx.grid.GridTableBase.\_\_init\_\_(self)  self.entries = entries    def GetNumberRows(self):  return len(self.entries)    def GetNumberCols(self):  return len(self.colAttrs)    def GetColLabelValue(self, col):  return self.colLabels[col]    def GetRowLabelValue(self, row):  return self.entries[row].pos    def IsEmptyCell(self, row, col):  return False    def GetValue(self, row, col):  entry = self.entries[row]  attr = self.colAttrs[col]  return getattr(entry, attr)    def SetValue(self, row, col, value):  pass  class SimpleGrid(wx.grid.Grid):  def \_\_init\_\_(self, parent):  wx.grid.Grid.\_\_init\_\_(self, parent, -1)  entries = [  LineupEntry("CF", "Bob", "Dernier"),  LineupEntry("2B", "Ryne", "Sandberg"),  LineupEntry("LF", "Gary", "Matthews"),  LineupEntry("1B", "Leon", "Durham"),  LineupEntry("RF", "Keith", "Moreland"),  LineupEntry("3B", "Ron", "Cey"),  LineupEntry("C", "Jody", "Davis"),  LineupEntry("SS", "Larry", "Bowa"),  LineupEntry("P", "Rick", "Sutcliffe")  ]  self.SetTable(LineupTable(entries), takeOwnership=True)  class TestFrame(wx.Frame):  def \_\_init\_\_(self, parent):  wx.Frame.\_\_init\_\_(self, parent, -1, "A Grid", size=(275, 275))  grid = SimpleGrid(self)  self.Show()  if \_\_name\_\_ == '\_\_main\_\_':  app = wx.App(False) # False để không chuyển hướng stdout/stderr  frame = TestFrame(None)  app.MainLoop() |
| **Các thay đổi chính:**   1. **Thay thế wx.grid.PyGridTableBase bằng wx.grid.GridTableBase**:    * **PyGridTableBase** đã được thay thế bằng **GridTableBase** trong các phiên bản wxPython mới hơn. 2. **Khởi tạo wx.App**:    * Sử dụng **wx.App(False)** thay vì **wx.PySimpleApp()** vì **wx.PySimpleApp()** đã bị loại bỏ trong các phiên bản wxPython mới hơn. 3. **Sử dụng takeOwnership=True khi gọi SetTable**:    * Đảm bảo rằng lưới sẽ quản lý bộ nhớ của bảng dữ liệu.   Với các thay đổi này, mã của bạn sẽ chạy và tạo ra một cửa sổ chứa bảng dữ liệu. |

Lớp **LineupEntry** mô tả một mục nhập trong bảng, và **LineupTable** sử dụng các mục này để cung cấp dữ liệu cho lưới.

**Kết luận**

wxPython cung cấp một cách mạnh mẽ để xử lý dữ liệu bảng trong các ứng dụng GUI. Sử dụng **GridTableBase**, bạn có thể tạo các mô hình dữ liệu tùy chỉnh, giúp việc quản lý và hiển thị dữ liệu dễ dàng hơn và linh hoạt hơn.

**Giải thích về MVC và ví dụ trong wxPython – Tùy chỉnh model (A custom model)**

**Mô hình MVC**

MVC (Model-View-Controller) là một mẫu thiết kế phần mềm giúp tách biệt logic nghiệp vụ của ứng dụng (Model), giao diện người dùng (View), và luồng điều khiển (Controller). Điều này giúp cho việc phát triển, bảo trì và mở rộng ứng dụng trở nên dễ dàng hơn.

* **Model**: Chứa dữ liệu và logic nghiệp vụ của ứng dụng. Nó không quan tâm đến cách dữ liệu được hiển thị.
* **View**: Chịu trách nhiệm hiển thị dữ liệu từ Model cho người dùng.
* **Controller**: Xử lý các đầu vào từ người dùng, cập nhật Model và View.

**Mã ví dụ và giải thích**

Để minh họa cho mô hình MVC, chúng ta sẽ tạo một lớp **AbstractModel** để quản lý việc thông báo cho các view khi có sự thay đổi trong model.

**Lớp AbstractModel**:

* **\_\_init\_\_**: Khởi tạo danh sách các listeners (các hàm sẽ được gọi khi model thay đổi).
* **addListener**: Thêm một listener vào danh sách.
* **removeListener**: Loại bỏ một listener khỏi danh sách.
* **update**: Gọi tất cả các listener trong danh sách, truyền đối tượng model hiện tại cho chúng.

**Lớp SimpleName kế thừa từ AbstractModel**:

* **\_\_init\_\_**: Khởi tạo đối tượng với tên đầu tiên và tên cuối cùng, sau đó cập nhật trạng thái.
* **set**: Cập nhật tên đầu tiên và tên cuối cùng, và thông báo cho các listener về sự thay đổi.

**Lớp ModelExample** là phần view và controller:

* **\_\_init\_\_**: Tạo giao diện người dùng, kết nối các thành phần và đăng ký phương thức **OnUpdate** làm listener cho model.
* **createButtonBar**: Tạo các nút bấm cho giao diện.
* **createTextFields**: Tạo các ô văn bản cho giao diện.
* **OnUpdate**: Cập nhật các ô văn bản khi model thay đổi.
* **Các phương thức như OnFred, OnBarney**: Thay đổi giá trị của model khi nút tương ứng được bấm.

**Mã ví dụ**

Dưới đây là mã ví dụ đã được sửa đổi và đầy đủ cho phiên bản wxPython mới:

|  |
| --- |
| import wx  class AbstractModel(object):  def \_\_init\_\_(self):  self.listeners = []  def addListener(self, listenerFunc):  self.listeners.append(listenerFunc)  def removeListener(self, listenerFunc):  self.listeners.remove(listenerFunc)  def update(self):  for eachFunc in self.listeners:  eachFunc(self)  class SimpleName(AbstractModel):  def \_\_init\_\_(self, first="", last=""):  super(SimpleName, self).\_\_init\_\_()  self.set(first, last)  def set(self, first, last):  self.first = first  self.last = last  self.update()  class ModelExample(wx.Frame):  def \_\_init\_\_(self, parent, id):  wx.Frame.\_\_init\_\_(self, parent, id, 'Flintstones', size=(340, 200))  panel = wx.Panel(self)  panel.SetBackgroundColour("White")  self.Bind(wx.EVT\_CLOSE, self.OnCloseWindow)  self.textFields = {}  self.createTextFields(panel)  self.model = SimpleName()  self.model.addListener(self.OnUpdate)  self.createButtonBar(panel)  def buttonData(self):  return (("Fredify", self.OnFred),  ("Wilmafy", self.OnWilma),  ("Barnify", self.OnBarney),  ("Bettify", self.OnBetty))  def createButtonBar(self, panel, yPos = 0):  xPos = 0  for eachLabel, eachHandler in self.buttonData():  pos = (xPos, yPos)  button = self.buildOneButton(panel, eachLabel, eachHandler, pos)  xPos += button.GetSize().width  def buildOneButton(self, parent, label, handler, pos=(0,0)):  button = wx.Button(parent, -1, label, pos)  self.Bind(wx.EVT\_BUTTON, handler, button)  return button  def textFieldData(self):  return (("First Name", (10, 50)),  ("Last Name", (10, 80)))  def createTextFields(self, panel):  for eachLabel, eachPos in self.textFieldData():  self.createCaptionedText(panel, eachLabel, eachPos)  def createCaptionedText(self, panel, label, pos):  static = wx.StaticText(panel, wx.NewId(), label, pos)  static.SetBackgroundColour("White")  textPos = (pos[0] + 75, pos[1])  self.textFields[label] = wx.TextCtrl(panel, wx.NewId(), "", size=(100, -1), pos=textPos, style=wx.TE\_READONLY)  def OnUpdate(self, model):  self.textFields["First Name"].SetValue(model.first)  self.textFields["Last Name"].SetValue(model.last)  def OnFred(self, event):  self.model.set("Fred", "Flintstone")  def OnBarney(self, event):  self.model.set("Barney", "Rubble")  def OnWilma(self, event):  self.model.set("Wilma", "Flintstone")  def OnBetty(self, event):  self.model.set("Betty", "Rubble")  def OnCloseWindow(self, event):  self.Destroy()  if \_\_name\_\_ == '\_\_main\_\_':  app = wx.App()  frame = ModelExample(parent=None, id=-1)  frame.Show()  app.MainLoop() |
| **Giải thích mã ví dụ**   * **AbstractModel**: Quản lý các listener để thông báo khi model thay đổi. * **SimpleName**: Lớp model cụ thể, chứa tên đầu tiên và tên cuối cùng, kế thừa từ **AbstractModel**. * **ModelExample**: Khởi tạo giao diện người dùng và kết nối các thành phần của MVC. Giao diện bao gồm các ô văn bản và nút bấm để cập nhật model. Khi model thay đổi, các ô văn bản được cập nhật tự động.   Khi một nút được bấm, model sẽ thay đổi và thông báo cho view qua phương thức **OnUpdate**, cập nhật giao diện người dùng với giá trị mới từ model. |

**Làm thế nào để kiểm thử đơn vị một chương trình GUI? ( How do you unit-test a GUI program?)**

Một lợi thế quan trọng của việc tái cấu trúc và sử dụng mô hình MVC là giúp dễ dàng kiểm tra hiệu suất của chương trình bằng các bài kiểm thử đơn vị. Kiểm thử đơn vị là kiểm thử cho một chức năng cụ thể của chương trình. Vì tái cấu trúc và sử dụng mô hình MVC thường chia chương trình thành các phần nhỏ hơn, điều này giúp dễ dàng viết các bài kiểm thử đơn vị nhắm mục tiêu vào các phần riêng lẻ của chương trình. Kiểm thử đơn vị là một công cụ hữu ích khi kết hợp với tái cấu trúc, vì một bộ kiểm thử đơn vị đầy đủ cho phép bạn xác minh rằng bạn không gây ra lỗi khi di chuyển mã xung quanh.

**5.3.1 Mô-đun unittest**

Khi viết kiểm thử đơn vị, việc sử dụng một engine kiểm thử có sẵn rất hữu ích để tránh phải viết mã lặp đi lặp lại. Python có mô-đun **unittest** để hỗ trợ việc này từ phiên bản 2.1. Mô-đun **unittest** triển khai một framework kiểm thử gọi là PyUnit. Một mô-đun PyUnit bao gồm các kiểm thử, test cases (trường hợp kiểm thử), và test suites (bộ kiểm thử). Bảng 5.5 định nghĩa ba nhóm này.

**Bảng 5.5: Ba mức độ trừu tượng trong mô-đun unittest**

|  |  |
| --- | --- |
| **Mục** | **Định nghĩa** |
| **Test** | Một phương thức riêng lẻ được gọi bởi engine PyUnit. Theo quy ước, tên của phương thức kiểm thử bắt đầu bằng **test**. Một phương thức kiểm thử thường thực thi một số mã, sau đó thực hiện một hoặc nhiều lệnh assert để kiểm tra kết quả có như mong đợi hay không. |
| **TestCase** | Một lớp định nghĩa một hoặc nhiều kiểm thử cá nhân chia sẻ một cài đặt chung. Lớp này được định nghĩa trong PyUnit để quản lý một nhóm các kiểm thử như vậy. Lớp **TestCase** có hỗ trợ cho việc thiết lập chung trước và tháo dỡ sau mỗi kiểm thử, đảm bảo rằng mỗi kiểm thử chạy riêng biệt với các kiểm thử khác. Lớp **TestCase** cũng định nghĩa một số phương thức assert đặc biệt, chẳng hạn như **assertEqual**. |
| **TestSuite** | Một hoặc nhiều phương thức kiểm thử hoặc các đối tượng **TestCase** được nhóm lại với nhau nhằm mục đích chạy cùng lúc. Khi bạn thực sự yêu cầu PyUnit chạy kiểm thử, bạn sẽ truyền một đối tượng **TestSuite** để chạy. |

Một kiểm thử PyUnit đơn lẻ có thể có ba kết quả: thành công, thất bại hoặc lỗi. Thành công chỉ ra rằng phương thức kiểm thử chạy hoàn thành, tất cả các assert đều đúng, và không có lỗi nào được kích hoạt. Thất bại và lỗi chỉ ra các vấn đề khác nhau với mã. Kết quả thất bại có nghĩa là một trong các assert của bạn trả về false, cho thấy mã chạy thành công nhưng không làm điều bạn mong đợi. Kết quả lỗi có nghĩa là một ngoại lệ Python được kích hoạt ở đâu đó trong quá trình thực thi kiểm thử, cho thấy rằng mã của bạn không chạy thành công. Lỗi hoặc thất bại đầu tiên trong một kiểm thử đơn lẻ sẽ kết thúc việc thực thi kiểm thử đó, ngay cả khi còn nhiều assert khác để kiểm tra trong mã, và runner kiểm thử sẽ chuyển sang kiểm thử tiếp theo.

**5.3.2 Ví dụ về unittest**

Dưới đây là một ví dụ về mô-đun **unittest** cho ví dụ mô hình trong Listing 5.12.

|  |
| --- |
| import unittest  import wx  from modelExample import ModelExample  class TestExample(unittest.TestCase):    def setUp(self):  self.app = wx.App()  self.frame = ModelExample(parent=None, id=-1)  def tearDown(self):  self.frame.Destroy()  def testModel(self):  self.frame.OnBarney(None)  self.assertEqual("Barney", self.frame.model.first, msg="First is wrong")  self.assertEqual("Rubble", self.frame.model.last)  def suite():  suite = unittest.TestSuite()  suite.addTest(unittest.makeSuite(TestExample))  return suite  if \_\_name\_\_ == '\_\_main\_\_':  unittest.TextTestRunner().run(suite()) |

**Giải thích các phần của unittest:**

* **TestCase**: Lớp kiểm thử là một subclass của **unittest.TestCase**. Runner kiểm thử tạo một instance của lớp cho mỗi kiểm thử, để đảm bảo rằng các kiểm thử độc lập với nhau.
* **setUp()**: Phương thức này được gọi trước mỗi kiểm thử. Điều này cho phép đảm bảo rằng mỗi kiểm thử bắt đầu với ứng dụng ở cùng một trạng thái. Trong trường hợp này, chúng ta tạo một instance của frame mà chúng ta đang kiểm thử.
* **tearDown()**: Phương thức này được gọi sau mỗi kiểm thử. Điều này cho phép làm sạch và đảm bảo rằng trạng thái hệ thống vẫn nhất quán từ kiểm thử này sang kiểm thử khác. Trong trường hợp này, chúng ta gọi **Destroy()** trên frame để buộc wxWidgets thoát, và giữ hệ thống ở trạng thái tốt cho kiểm thử tiếp theo.
* **testModel()**: Phương thức kiểm thử này bắt đầu bằng cách gọi rõ ràng phương thức handler sự kiện **OnBarney** để kiểm tra hành vi. Sau đó, nó sử dụng **assertEqual()** để kiểm tra rằng đối tượng model đã được thay đổi chính xác. Phương thức **assertEqual()** nhận hai đối số, và kiểm thử sẽ thất bại nếu chúng không bằng nhau. Tất cả các phương thức assert của PyUnit đều nhận một đối số tùy chọn **msg** được hiển thị nếu assert thất bại.

**5.3.3 Kiểm thử sự kiện người dùng**

Kiểm thử này không phải là kiểm thử hoàn chỉnh của hệ thống. Chúng ta cũng có thể kiểm thử rằng các **TextField** trong frame đã được cập nhật với các giá trị sau khi model được cập nhật. Một kiểm thử khác có thể là tạo sự kiện click nút tự động và đảm bảo rằng handler phù hợp được gọi. Kiểm thử này ít đơn giản hơn. Dưới đây là một ví dụ:

|  |
| --- |
| def testEvent(self):  panel = self.frame.GetChildren()[0]  wilma = None  for each in panel.GetChildren():  if each.GetLabel() == "Wilmafy":  wilma = each  break  event = wx.CommandEvent(wx.EVT\_BUTTON.typeId, wilma.GetId())  wilma.GetEventHandler().ProcessEvent(event)  self.assertEqual("Wilma", self.frame.model.first)  self.assertEqual("Flintstone", self.frame.model.last) |

**Giải thích:**

1. **Tìm button "Wilmafy"**: Đoạn mã này tìm nút "Wilmafy" trong danh sách các con của panel.
2. **Tạo sự kiện**: Tạo một **wx.CommandEvent** với loại sự kiện **wx.EVT\_BUTTON** và ID của button "Wilmafy".
3. **Xử lý sự kiện**: Gọi **ProcessEvent()** để gửi sự kiện vào hệ thống. Điều này sẽ kích hoạt handler của button "Wilmafy".
4. **Kiểm tra kết quả**: Sử dụng **assertEqual()** để kiểm tra rằng các giá trị trong model đã được thay đổi đúng.

Việc tạo các sự kiện giúp kiểm tra tính phản hồi của hệ thống từ đầu đến cuối. Trong lý thuyết, bạn có thể tạo ra sự kiện mouse-down và mouse-up trong button để đảm bảo rằng sự kiện click button được tạo ra như một phản hồi. Tuy nhiên, điều này không hoạt động với các widget gốc vì các sự kiện cấp thấp của wx không được dịch lại thành các sự kiện hệ thống gốc và gửi tới widget gốc. Tuy nhiên, một quy trình tương tự có thể hữu ích khi kiểm thử các widget tùy chỉnh.

**Kết quả kiểm thử mô phỏng**

Kết quả của bài kiểm thử PyUnit này, chạy từ cửa sổ console, như sau:

|  |
| --- |
| . ----------------------------------------------------------------------  Ran 1 test in 0.190s  OK |

Đây là một kiểm thử thành công. Dòng trên cùng, với dấu chấm, chỉ ra rằng một kiểm thử đã chạy thành công. Mỗi kiểm thử sẽ có một ký tự trong hiển thị, **.** chỉ ra thành công, **F** chỉ ra thất bại, và **E** chỉ ra lỗi. Sau đó là danh sách đơn giản về số lượng kiểm thử và tổng thời gian đã trôi qua, và một **OK** chỉ ra rằng tất cả các kiểm thử đều thành công.

Khi có thất bại hoặc lỗi, bạn sẽ nhận được một stack trace cho thấy cách Python đã đến điểm lỗi. Nếu chúng ta thay đổi kiểm tra họ cuối cùng thành "Fife", chẳng hạn, chúng ta sẽ nhận được kết quả sau:

|  |
| --- |
| F  ======================================================================  FAIL: testModel (\_\_main\_\_.TestExample)  ----------------------------------------------------------------------  Traceback (most recent call last):  File "C:\wxPyBook\book\tests\testExample.py", line 14, in testModel  self.assertEqual("Fife", self.frame.model.last)  AssertionError: 'Fife' != 'Rubble'  ----------------------------------------------------------------------  Ran 1 test in 0.174s  FAILED (failures=1) |

Lỗi trên xảy ra do assert cuối cùng không khớp. Bạn nhận được một traceback Python chuẩn, cho phép bạn nhanh chóng xác định nguồn gốc của vấn đề

**5.3.3 Kiểm tra các sự kiện người dùng**

Việc kiểm tra này không phải là một bài kiểm tra hoàn chỉnh của hệ thống. Chúng ta cũng có thể kiểm tra xem các **TextField** trong khung có được cập nhật với các giá trị sau khi mô hình được cập nhật hay không. Kiểm tra này sẽ khá đơn giản. Một kiểm tra khác bạn có thể muốn thực hiện là tự động tạo sự kiện nhấn nút và đảm bảo rằng hàm xử lý phù hợp được gọi. Kiểm tra này ít đơn giản hơn. Ví dụ 5.14 dưới đây cho thấy một ví dụ:

**Ví dụ 5.14 Một ví dụ kiểm tra bằng cách tạo một sự kiện người dùng**

|  |
| --- |
| def testEvent(self):  panel = self.frame.GetChildren()[0]  for each in panel.GetChildren():  if each.GetLabel() == "Wilmafy":  wilma = each  break  event = wx.CommandEvent(wx.wxEVT\_COMMAND\_BUTTON\_CLICKED, wilma.GetId())  wilma.GetEventHandler().ProcessEvent(event)  self.assertEqual("Wilma", self.frame.model.first)  self.assertEqual("Flintstone", self.frame.model.last) |

Những dòng đầu tiên của ví dụ này tìm nút phù hợp (trong trường hợp này, nút “Wilmafy”). Vì chúng ta không lưu trữ các nút dưới dạng các biến instance của Python, chúng ta chỉ cần duyệt qua danh sách con của panel cho đến khi tìm thấy nút đúng. (Bạn cũng có thể làm điều này bằng cách sử dụng list comprehension của Python nếu muốn). Hai dòng tiếp theo tạo **wx.CommandEvent** để gửi từ nút. Tham số duy nhất cho hàm tạo là **wx.wxEVT\_COMMAND\_BUTTON\_CLICKED**, một hằng số cho loại sự kiện số nguyên thực sự được gắn với đối tượng **EVT\_BUTTON**. (Bạn có thể tìm các hằng số số nguyên trong tệp nguồn **wx.py** của wxPython). Sau đó, chúng ta đặt ID của sự kiện thành ID của nút Wilmafy. Tại thời điểm này, sự kiện có tất cả các đặc điểm liên quan của sự kiện thực sự như nó sẽ được tạo ra bởi wxPython. Vì vậy, chúng ta gọi **ProcessEvent()** để gửi nó vào hệ thống. Nếu mã hoạt động như mong đợi, thì tên đầu tiên và họ của mô hình sẽ được thay đổi thành “Wilma” và “Flintstone.”

Bằng cách tạo các sự kiện, bạn có thể kiểm tra tính đáp ứng của hệ thống từ đầu đến cuối. Trong lý thuyết, bạn có thể tạo ra sự kiện nhấn chuột xuống và nhả chuột trong nút của bạn để đảm bảo rằng sự kiện nhấn nút được tạo ra như một phản ứng. Trong thực tế, điều này sẽ không hoạt động với các widget gốc vì các sự kiện wx cấp thấp không được dịch lại thành các sự kiện hệ thống gốc và gửi đến widget gốc. Tuy nhiên, một quá trình tương tự có thể hữu ích khi kiểm tra các widget tùy chỉnh (như điều khiển hai nút trong chương 3). Loại kiểm tra đơn vị này có thể mang lại cho bạn sự tin tưởng vào tính đáp ứng của ứng dụng của bạn.

**5.4 Tóm tắt**

* **Mã GUI có tiếng xấu vì bừa bộn và khó bảo trì.** Điều này có thể khắc phục với một chút nỗ lực bổ sung, điều này sẽ mang lại lợi ích khi bạn cần thay đổi mã của mình.
* **Tái cấu trúc là cải thiện mã hiện có.** Một số mục tiêu của tái cấu trúc là loại bỏ sự trùng lặp, loại bỏ các hằng số phép màu và tạo ra các phương thức ngắn chỉ làm một việc. Liên tục phấn đấu cho những mục tiêu đó sẽ làm cho mã của bạn dễ đọc và dễ hiểu hơn. Ngoài ra, tái cấu trúc tốt cũng làm giảm khả năng xảy ra một số loại lỗi (như lỗi sao chép và dán).
* **Tách dữ liệu của bạn khỏi mã bố cục làm cho cả dữ liệu và bố cục dễ làm việc hơn.** Cơ chế tiêu chuẩn để quản lý sự tách biệt này là cơ chế MVC. Trong các thuật ngữ của wxPython, View là các đối tượng wx.Window hiển thị dữ liệu của bạn, Controller là các đối tượng wx.EvtHandler phân phối các sự kiện và Model là mã riêng của bạn chứa thông tin để hiển thị.
* **Có lẽ ví dụ rõ ràng nhất của cấu trúc MVC trong các lớp cốt lõi của wxPython là wx.grid.GridTableBase,** được sử dụng để mô hình hóa dữ liệu để hiển thị trong một điều khiển wx.grid.Grid. Dữ liệu trong bảng có thể đến từ chính lớp đó hoặc lớp có thể tham chiếu đến một đối tượng khác chứa dữ liệu liên quan.
* **Bạn có thể tạo cấu hình MVC của riêng mình với một cơ chế đơn giản để thông báo cho view khi model đã được cập nhật.** Cũng có các mô-đun hiện có trong wxPython sẽ giúp bạn làm điều này.
* **Kiểm tra đơn vị là một cách hữu ích để xác minh tính hợp lệ của chương trình của bạn.** Trong Python, mô-đun unittest là một trong những cách tiêu chuẩn để thực thi các kiểm tra đơn vị. Trong một số gói, việc kiểm tra đơn vị của một GUI là khó khăn, nhưng wxPython làm cho nó tương đối dễ dàng để lập trình tạo ra các sự kiện. Điều này cho phép bạn kiểm tra hành vi xử lý sự kiện của ứng dụng từ đầu đến cuối.

Trong chương tiếp theo, chúng ta sẽ chỉ cho bạn cách xây dựng một ứng dụng nhỏ và cách thực hiện một số việc sẽ phổ biến với nhiều ứng dụng wxPython mà bạn sẽ xây dựng.

**Working with the basic building blocks**

**Ví dụ về menu bar và status bar**

|  |
| --- |
| import wx  class MyFrame(wx.Frame):  def \_\_init\_\_(self, \*args, \*\*kw):  super(MyFrame, self).\_\_init\_\_(\*args, \*\*kw)  # Thiết lập GUI  self.InitUI()    def InitUI(self):  # Tạo menu bar  menubar = wx.MenuBar()  # Tạo menu File  fileMenu = wx.Menu()  new\_item = fileMenu.Append(wx.ID\_NEW, '&New\tCtrl+N', 'New file')  open\_item = fileMenu.Append(wx.ID\_OPEN, '&Open\tCtrl+O', 'Open file')  save\_item = fileMenu.Append(wx.ID\_SAVE, '&Save\tCtrl+S', 'Save file')  fileMenu.AppendSeparator()  exit\_item = fileMenu.Append(wx.ID\_EXIT, 'E&xit\tCtrl+Q', 'Exit application')  menubar.Append(fileMenu, '&File')  # Tạo menu Help  helpMenu = wx.Menu()  about\_item = helpMenu.Append(wx.ID\_ABOUT, '&About\tF1', 'About application')  menubar.Append(helpMenu, '&Help')  # Thiết lập menu bar cho frame  self.SetMenuBar(menubar)  # Tạo status bar  self.CreateStatusBar()  self.SetStatusText('Ready')  # Kết nối các sự kiện  self.Bind(wx.EVT\_MENU, self.OnNew, new\_item)  self.Bind(wx.EVT\_MENU, self.OnOpen, open\_item)  self.Bind(wx.EVT\_MENU, self.OnSave, save\_item)  self.Bind(wx.EVT\_MENU, self.OnExit, exit\_item)  self.Bind(wx.EVT\_MENU, self.OnAbout, about\_item)  # Thiết lập kích thước và tiêu đề cho frame  self.SetSize((800, 600))  self.SetTitle('wxPython Menu and Status Bar Example')  self.Centre()    def OnNew(self, event):  self.SetStatusText('New file selected')  def OnOpen(self, event):  self.SetStatusText('Open file selected')  def OnSave(self, event):  self.SetStatusText('Save file selected')  def OnExit(self, event):  self.Close(True)  def OnAbout(self, event):  wx.MessageBox('This is a wxPython example', 'About', wx.OK | wx.ICON\_INFORMATION)  if \_\_name\_\_ == "\_\_main\_\_":  app = wx.App(False)  frame = MyFrame(None)  frame.Show()  app.MainLoop() |
| **Giải thích mã nguồn:**   1. **Khởi tạo thư viện wxPython**:    * **import wx**: Import thư viện wxPython. 2. **Lớp MyFrame**:    * **MyFrame** kế thừa từ **wx.Frame**, tạo cửa sổ chính của ứng dụng.    * **\_\_init\_\_** khởi tạo các thành phần của GUI bằng cách gọi **InitUI**. 3. **Phương thức InitUI**:    * **Tạo Menu Bar**:      + **menubar = wx.MenuBar()**: Tạo một thanh menu mới.      + **fileMenu = wx.Menu()**: Tạo menu File với các mục New, Open, Save và Exit.      + **helpMenu = wx.Menu()**: Tạo menu Help với mục About.      + **menubar.Append(fileMenu, '&File')** và **menubar.Append(helpMenu, '&Help')**: Thêm các menu vào thanh menu.      + **self.SetMenuBar(menubar)**: Đặt thanh menu cho cửa sổ chính.    * **Tạo Status Bar**:      + **self.CreateStatusBar()**: Tạo thanh trạng thái.      + **self.SetStatusText('Ready')**: Đặt văn bản mặc định cho thanh trạng thái.    * **Kết nối sự kiện**:      + **self.Bind(wx.EVT\_MENU, self.OnNew, new\_item)**: Kết nối sự kiện chọn mục menu với phương thức xử lý sự kiện tương ứng.    * **Thiết lập kích thước và tiêu đề**:      + **self.SetSize((800, 600))**: Đặt kích thước cho cửa sổ.      + **self.SetTitle('wxPython Menu and Status Bar Example')**: Đặt tiêu đề cho cửa sổ.      + **self.Centre()**: Căn giữa cửa sổ trên màn hình. 4. **Các phương thức xử lý sự kiện**:    * **OnNew**, **OnOpen**, **OnSave**, **OnExit**, **OnAbout**: Các phương thức này xử lý sự kiện khi các mục menu được chọn. Ví dụ, **OnExit** sẽ đóng cửa sổ, **OnAbout** sẽ hiển thị một hộp thoại thông tin. 5. **Khởi tạo ứng dụng**:    * **app = wx.App(False)**: Khởi tạo ứng dụng wxPython.    * **frame = MyFrame(None)**: Tạo cửa sổ chính.    * **frame.Show()**: Hiển thị cửa sổ chính.    * **app.MainLoop()**: Bắt đầu vòng lặp sự kiện của ứng dụng. |

**Ví dụ về Thanh Trạng Thái với Nhiều Trường Trạng Thái**

|  |
| --- |
| import wx  class MyFrame(wx.Frame):  def \_\_init\_\_(self, \*args, \*\*kw):  super(MyFrame, self).\_\_init\_\_(\*args, \*\*kw)  # Thiết lập GUI  self.InitUI()    def InitUI(self):  # Tạo menu bar  menubar = wx.MenuBar()  # Tạo menu File  fileMenu = wx.Menu()  new\_item = fileMenu.Append(wx.ID\_NEW, '&New\tCtrl+N', 'New file')  open\_item = fileMenu.Append(wx.ID\_OPEN, '&Open\tCtrl+O', 'Open file')  save\_item = fileMenu.Append(wx.ID\_SAVE, '&Save\tCtrl+S', 'Save file')  fileMenu.AppendSeparator()  exit\_item = fileMenu.Append(wx.ID\_EXIT, 'E&xit\tCtrl+Q', 'Exit application')  menubar.Append(fileMenu, '&File')  # Tạo menu Help  helpMenu = wx.Menu()  about\_item = helpMenu.Append(wx.ID\_ABOUT, '&About\tF1', 'About application')  menubar.Append(helpMenu, '&Help')  # Thiết lập menu bar cho frame  self.SetMenuBar(menubar)  # Tạo status bar với 3 trường  self.CreateStatusBar(3)  self.SetStatusWidths([-2, -1, -1])  self.SetStatusText('Ready', 0)  self.SetStatusText('Welcome to wxPython!', 1)  self.SetStatusText('Version 1.0', 2)  # Kết nối các sự kiện  self.Bind(wx.EVT\_MENU, self.OnNew, new\_item)  self.Bind(wx.EVT\_MENU, self.OnOpen, open\_item)  self.Bind(wx.EVT\_MENU, self.OnSave, save\_item)  self.Bind(wx.EVT\_MENU, self.OnExit, exit\_item)  self.Bind(wx.EVT\_MENU, self.OnAbout, about\_item)  # Thiết lập kích thước và tiêu đề cho frame  self.SetSize((800, 600))  self.SetTitle('wxPython Menu and Status Bar Example')  self.Centre()    def OnNew(self, event):  self.SetStatusText('New file selected', 0)  self.SetStatusText('Creating new file...', 1)  def OnOpen(self, event):  self.SetStatusText('Open file selected', 0)  self.SetStatusText('Opening file...', 1)  def OnSave(self, event):  self.SetStatusText('Save file selected', 0)  self.SetStatusText('Saving file...', 1)  def OnExit(self, event):  self.Close(True)  def OnAbout(self, event):  wx.MessageBox('This is a wxPython example', 'About', wx.OK | wx.ICON\_INFORMATION)  if \_\_name\_\_ == "\_\_main\_\_":  app = wx.App(False)  frame = MyFrame(None)  frame.Show()  app.MainLoop() |
| 1. **Khởi tạo thanh trạng thái với nhiều trường**:    * **self.CreateStatusBar(3)**: Tạo thanh trạng thái với 3 trường.    * **self.SetStatusWidths([-2, -1, -1])**: Thiết lập kích thước cho mỗi trường trong thanh trạng thái. Các giá trị âm (-2, -1, -1) cho phép các trường tự động điều chỉnh kích thước theo tỷ lệ.    * **self.SetStatusText('Ready', 0)**, **self.SetStatusText('Welcome to wxPython!', 1)**, **self.SetStatusText('Version 1.0', 2)**: Thiết lập văn bản mặc định cho từng trường trong thanh trạng thái. 2. **Cập nhật thanh trạng thái trong các phương thức xử lý sự kiện**:    * **self.SetStatusText('New file selected', 0)** và **self.SetStatusText('Creating new file...', 1)**: Cập nhật văn bản của các trường trong thanh trạng thái khi một mục menu được chọn. Trong phương thức **OnNew**, trường đầu tiên được cập nhật để hiển thị thông báo rằng tệp mới đã được chọn, và trường thứ hai hiển thị trạng thái tạo tệp mới.    * Tương tự, các phương thức **OnOpen**, **OnSave**, và **OnExit** cập nhật các trường trạng thái khi các mục menu tương ứng được chọn. 3. **Các phương thức xử lý sự kiện khác**:    * **OnExit** sẽ đóng cửa sổ.    * **OnAbout** sẽ hiển thị một hộp thoại thông tin. 4. **Khởi tạo ứng dụng**:    * **app = wx.App(False)**: Khởi tạo ứng dụng wxPython.    * **frame = MyFrame(None)**: Tạo cửa sổ chính.    * **frame.Show()**: Hiển thị cửa sổ chính.    * **app.MainLoop()**: Bắt đầu vòng lặp sự kiện của ứng dụng. |

**Bảng: Các phương thức của wx.StatusBar**

|  |  |
| --- | --- |
| **Chức năng** | **Mô tả** |
| **GetFieldsCount()** | Thuộc tính để lấy số lượng trường trong thanh trạng thái |
| **SetFieldsCount(count)** | Thuộc tính để thiết lập số lượng trường trong thanh trạng thái |
| **GetStatusText(field=0)** | Thuộc tính để lấy văn bản hiển thị trong trường trạng thái được chỉ định. Chỉ số 0 là mặc định và đại diện cho trường bên trái nhất |
| **SetStatusText(text, field=0)** | Thuộc tính để thiết lập văn bản hiển thị trong trường trạng thái được chỉ định. Chỉ số 0 là mặc định và đại diện cho trường bên trái nhất |
| **PopStatusText(field=0)** | Lấy văn bản từ ngăn xếp văn bản của trường trạng thái được chỉ định, thay đổi văn bản của trường đó thành giá trị đã được lấy ra |
| **PushStatusText(text, field=0)** | Thay đổi hiển thị của trường trạng thái được chỉ định thành văn bản được cung cấp và đẩy giá trị đó lên đỉnh của ngăn xếp cho trường đó |
| **SetStatusWidths(widths)** | Nhận một danh sách các số nguyên trong Python và chỉ định độ rộng của các trường trạng thái. Một số dương biểu thị độ rộng cố định bằng pixel, và số âm biểu thị tỉ lệ động của độ rộng theo tỷ lệ với giá trị tuyệt đối của số. |

**Bảng: Các kiểu của lớp wx.ToolBar**

|  |  |
| --- | --- |
| **Kiểu** | **Mô tả** |
| **wx.TB\_3DBUTTONS** | Làm cho các công cụ hiển thị với giao diện 3D. |
| **wx.TB\_HORIZONTAL** | Kiểu mặc định, sắp xếp thanh công cụ theo chiều ngang. |
| **wx.TB\_NOICONS** | Không hiển thị các biểu tượng bitmap cho mỗi công cụ. |
| **wx.TB\_TEXT** | Thanh công cụ sẽ hiển thị văn bản trợ giúp ngắn cùng với các biểu tượng bitmap mặc định. |
| **wx.TB\_VERTICAL** | Sắp xếp thanh công cụ theo chiều dọc. |

**Bảng: Các phương thức thường dùng của wx.ToolBar**

|  |  |
| --- | --- |
| **Chức năng** | **Mô tả** |
| **AddControl(control)** | Thêm một điều khiển (control) wxPython tùy ý vào thanh công cụ. Xem thêm phương thức liên quan **InsertControl()**. |
| **AddSeparator()** | Thêm khoảng trống giữa các công cụ. |
| **AddSimpleTool(id, bitmap, shortHelpString="", kind=wx.ITEM\_NORMAL)** | Thêm một nút công cụ đơn giản vào thanh công cụ với bitmap đã cho. **shortHelpString** được hiển thị như một tooltip. Loại công cụ có thể là **wx.ITEM\_NORMAL**, **wx.ITEM\_CHECKBOX**, hoặc **wx.ITEM\_RADIO**. |
| **AddTool(id, bitmap, bitmap2=wx.NullBitmap, kind=wx.ITEM\_NORMAL, shortHelpString="", longHelpString="", clientData=None)** | Các tham số bổ sung cho công cụ đơn giản. **bitmap2** được hiển thị khi công cụ được nhấn. **longHelpString** được hiển thị trong thanh trạng thái khi con trỏ ở trên công cụ, và **clientData** có thể được dùng để liên kết một mảnh dữ liệu tùy ý với công cụ. Có một phương thức liên quan là **InsertTool()**. |
| **AddCheckTool(...)** | Thêm một công cụ checkbox toggle với các tham số giống như **AddTool()**. |
| **AddRadioTool(...)** | Thêm một công cụ radio toggle với các tham số giống như **AddTool()**. Một dãy liên tiếp, không gián đoạn của các công cụ radio được xem như là một nhóm để toggle. |
| **DeleteTool(toolId)** | Xóa công cụ với **id** được cho. |
| **DeleteToolByPosition(x, y)** | Xóa công cụ hiển thị tại vị trí được cho. |
| **FindControl(toolId)** | Tìm và trả về điều khiển (tool) với **id** được cho. |
| **FindToolForPosition(x, y)** | Tìm và trả về công cụ được hiển thị tại vị trí được cho. |
| **ToggleTool(toolId, toggle)** | Nếu công cụ với **id** được chỉ định là radio hoặc checkbox, thiết lập toggle của công cụ đó dựa trên đối số Boolean **toggle**. |

**Ví dụ tổng hợp:**

|  |
| --- |
| import wx  class MyFrame(wx.Frame):  def \_\_init\_\_(self, \*args, \*\*kw):  super(MyFrame, self).\_\_init\_\_(\*args, \*\*kw)  # Thiết lập GUI  self.InitUI()  def InitUI(self):  # Tạo menu bar  menubar = wx.MenuBar()  # Tạo menu File  fileMenu = wx.Menu()  new\_item = fileMenu.Append(wx.ID\_NEW, '&New\tCtrl+N', 'New file')  open\_item = fileMenu.Append(wx.ID\_OPEN, '&Open\tCtrl+O', 'Open file')  save\_item = fileMenu.Append(wx.ID\_SAVE, '&Save\tCtrl+S', 'Save file')  fileMenu.AppendSeparator()  exit\_item = fileMenu.Append(wx.ID\_EXIT, 'E&xit\tCtrl+Q', 'Exit application')  menubar.Append(fileMenu, '&File')  # Tạo menu Help  helpMenu = wx.Menu()  about\_item = helpMenu.Append(wx.ID\_ABOUT, '&About\tF1', 'About application')  menubar.Append(helpMenu, '&Help')  # Thiết lập menu bar cho frame  self.SetMenuBar(menubar)  # Tạo status bar với 3 trường  self.CreateStatusBar(3)  self.SetStatusWidths([-2, -1, -1])  self.SetStatusText('Ready', 0)  self.SetStatusText('Welcome to wxPython!', 1)  self.SetStatusText('Version 1.0', 2)  # Tạo toolbar  toolbar = self.CreateToolBar(wx.TB\_HORIZONTAL | wx.NO\_BORDER | wx.TB\_TEXT)  new\_tool = toolbar.AddTool(wx.ID\_NEW, 'New', wx.ArtProvider.GetBitmap(wx.ART\_NEW, wx.ART\_TOOLBAR), 'New file')  open\_tool = toolbar.AddTool(wx.ID\_OPEN, 'Open', wx.ArtProvider.GetBitmap(wx.ART\_FILE\_OPEN, wx.ART\_TOOLBAR), 'Open file')  save\_tool = toolbar.AddTool(wx.ID\_SAVE, 'Save', wx.ArtProvider.GetBitmap(wx.ART\_FILE\_SAVE, wx.ART\_TOOLBAR), 'Save file')  toolbar.AddSeparator()  exit\_tool = toolbar.AddTool(wx.ID\_EXIT, 'Exit', wx.ArtProvider.GetBitmap(wx.ART\_QUIT, wx.ART\_TOOLBAR), 'Exit application')  toolbar.Realize()  # Kết nối các sự kiện  self.Bind(wx.EVT\_MENU, self.OnNew, new\_item)  self.Bind(wx.EVT\_MENU, self.OnOpen, open\_item)  self.Bind(wx.EVT\_MENU, self.OnSave, save\_item)  self.Bind(wx.EVT\_MENU, self.OnExit, exit\_item)  self.Bind(wx.EVT\_MENU, self.OnAbout, about\_item)  self.Bind(wx.EVT\_TOOL, self.OnNew, new\_tool)  self.Bind(wx.EVT\_TOOL, self.OnOpen, open\_tool)  self.Bind(wx.EVT\_TOOL, self.OnSave, save\_tool)  self.Bind(wx.EVT\_TOOL, self.OnExit, exit\_tool)  # Thiết lập kích thước và tiêu đề cho frame  self.SetSize((800, 600))  self.SetTitle('wxPython ToolBar and StatusBar Example')  self.Centre()  def OnNew(self, event):  self.SetStatusText('New file selected', 0)  self.SetStatusText('Creating new file...', 1)  def OnOpen(self, event):  self.SetStatusText('Open file selected', 0)  self.SetStatusText('Opening file...', 1)  def OnSave(self, event):  self.SetStatusText('Save file selected', 0)  self.SetStatusText('Saving file...', 1)  def OnExit(self, event):  self.Close(True)  def OnAbout(self, event):  wx.MessageBox('This is a wxPython example', 'About', wx.OK | wx.ICON\_INFORMATION)  if \_\_name\_\_ == "\_\_main\_\_":  app = wx.App(False)  frame = MyFrame(None)  frame.Show()  app.MainLoop() |
| 1. **Khởi tạo thư viện wxPython**:    * **import wx**: Import thư viện wxPython. 2. **Lớp MyFrame**:    * **MyFrame** kế thừa từ **wx.Frame**, tạo cửa sổ chính của ứng dụng.    * **\_\_init\_\_** khởi tạo các thành phần của GUI bằng cách gọi **InitUI**. 3. **Phương thức InitUI**:    * **Tạo Menu Bar**:      + **menubar = wx.MenuBar()**: Tạo một thanh menu mới.      + **fileMenu = wx.Menu()**: Tạo menu File với các mục New, Open, Save và Exit.      + **helpMenu = wx.Menu()**: Tạo menu Help với mục About.      + **menubar.Append(fileMenu, '&File')** và **menubar.Append(helpMenu, '&Help')**: Thêm các menu vào thanh menu.      + **self.SetMenuBar(menubar)**: Đặt thanh menu cho cửa sổ chính.    * **Tạo Status Bar với Nhiều Trường**:      + **self.CreateStatusBar(3)**: Tạo thanh trạng thái với 3 trường.      + **self.SetStatusWidths([-2, -1, -1])**: Thiết lập kích thước cho mỗi trường trong thanh trạng thái. Các giá trị âm (-2, -1, -1) cho phép các trường tự động điều chỉnh kích thước theo tỷ lệ.      + **self.SetStatusText('Ready', 0)**, **self.SetStatusText('Welcome to wxPython!', 1)**, **self.SetStatusText('Version 1.0', 2)**: Thiết lập văn bản mặc định cho từng trường trong thanh trạng thái.    * **Tạo ToolBar**:      + **toolbar = self.CreateToolBar(wx.TB\_HORIZONTAL | wx.NO\_BORDER | wx.TB\_TEXT)**: Tạo thanh công cụ với các thuộc tính ngang, không viền và hiển thị văn bản.      + **toolbar.AddTool()**: Thêm các công cụ vào thanh công cụ, bao gồm New, Open, Save, và Exit.      + **toolbar.AddSeparator()**: Thêm một khoảng trống giữa các công cụ.      + **toolbar.Realize()**: Hiển thị thanh công cụ sau khi thêm các công cụ. 4. **Kết nối sự kiện**:    * **self.Bind(wx.EVT\_MENU, self.OnNew, new\_item)**: Kết nối sự kiện chọn mục menu với phương thức xử lý sự kiện tương ứng.    * **self.Bind(wx.EVT\_TOOL, self.OnNew, new\_tool)**: Kết nối sự kiện chọn công cụ với phương thức xử lý sự kiện tương ứng. 5. **Các phương thức xử lý sự kiện**:    * **OnNew**, **OnOpen**, **OnSave**, **OnExit**, **OnAbout**: Các phương thức này xử lý sự kiện khi các mục menu hoặc công cụ được chọn. Ví dụ, **OnExit** sẽ đóng cửa sổ, **OnAbout** sẽ hiển thị một hộp thoại thông tin. 6. **Khởi tạo ứng dụng**:    * **app = wx.App(False)**: Khởi tạo ứng dụng wxPython.    * **frame = MyFrame(None)**: Tạo cửa sổ chính.    * **frame.Show()**: Hiển thị cửa sổ chính.    * **app.MainLoop()**: Bắt đầu vòng lặp sự kiện của ứng dụng. |

**Thu thập thông tin tiêu chuẩn (Getting standard information)**

Ứng dụng của bạn thường cần thông tin cơ bản từ người dùng, thông thường thông qua các hộp thoại (dialog boxes). Trong phần này, chúng ta sẽ nói về việc sử dụng các hộp thoại tệp tin và màu tiêu chuẩn để lấy thông tin từ người dùng.

**6.3.1 Làm thế nào để sử dụng hộp thoại tệp tin tiêu chuẩn? (How do I use standard file dialogs?)**

Hầu hết các ứng dụng GUI phải lưu và tải dữ liệu dưới một hình thức nào đó. Để tiện lợi cho cả bạn và người dùng, việc có một cơ chế nhất quán để chọn tệp tin là điều mong muốn. May mắn thay, wxPython cung cấp hộp thoại tiêu chuẩn **wx.FileDialog** để chèn vào ứng dụng của bạn cho mục đích này. Trên hệ điều hành Windows, lớp này là một lớp bọc quanh hộp thoại tệp tin tiêu chuẩn của Windows. Trên hệ thống X Window, đây là một hộp thoại tùy chỉnh tương tự. Hình 6.5 hiển thị hộp thoại tệp tin cho ứng dụng phác thảo (sketch application).

Phương thức quan trọng nhất để sử dụng **wx.FileDialog** là hàm khởi tạo (constructor), có chữ ký như sau:

|  |
| --- |
| wx.FileDialog(parent, message="Choose a file", defaultDir="", defaultFile="", wildcard="\*.\*", style=0) |

Ví dụ :

|  |
| --- |
| import wx  class MyFrame(wx.Frame):  def \_\_init\_\_(self, \*args, \*\*kw):  super(MyFrame, self).\_\_init\_\_(\*args, \*\*kw)  # Thiết lập GUI  self.InitUI()    def InitUI(self):  # Tạo menu bar  menubar = wx.MenuBar()  # Tạo menu File  fileMenu = wx.Menu()  open\_item = fileMenu.Append(wx.ID\_OPEN, '&Open\tCtrl+O', 'Open file')  save\_item = fileMenu.Append(wx.ID\_SAVE, '&Save\tCtrl+S', 'Save file')  fileMenu.AppendSeparator()  exit\_item = fileMenu.Append(wx.ID\_EXIT, 'E&xit\tCtrl+Q', 'Exit application')  menubar.Append(fileMenu, '&File')  # Thiết lập menu bar cho frame  self.SetMenuBar(menubar)  # Tạo status bar  self.CreateStatusBar()  self.SetStatusText('Ready')  # Kết nối các sự kiện  self.Bind(wx.EVT\_MENU, self.OnOpen, open\_item)  self.Bind(wx.EVT\_MENU, self.OnSave, save\_item)  self.Bind(wx.EVT\_MENU, self.OnExit, exit\_item)  # Thiết lập kích thước và tiêu đề cho frame  self.SetSize((800, 600))  self.SetTitle('wxPython File Dialog Example')  self.Centre()    def OnOpen(self, event):  with wx.FileDialog(self, "Choose a file to open", wildcard="All files (\*.\*)|\*.\*", style=wx.FD\_OPEN | wx.FD\_FILE\_MUST\_EXIST) as fileDialog:  if fileDialog.ShowModal() == wx.ID\_CANCEL:  return # Người dùng đã hủy  # Lấy đường dẫn tệp tin được chọn  path = fileDialog.GetPath()  self.SetStatusText(f'File chosen: {path}')  def OnSave(self, event):  with wx.FileDialog(self, "Choose a file to save", wildcard="All files (\*.\*)|\*.\*", style=wx.FD\_SAVE | wx.FD\_OVERWRITE\_PROMPT) as fileDialog:  if fileDialog.ShowModal() == wx.ID\_CANCEL:  return # Người dùng đã hủy  # Lấy đường dẫn tệp tin được chọn  path = fileDialog.GetPath()  self.SetStatusText(f'File saved: {path}')  def OnExit(self, event):  self.Close(True)  if \_\_name\_\_ == "\_\_main\_\_":  app = wx.App(False)  frame = MyFrame(None)  frame.Show()  app.MainLoop() |
| 1. **Khởi tạo thư viện wxPython**:    * **import wx**: Import thư viện wxPython. 2. **Lớp MyFrame**:    * **MyFrame** kế thừa từ **wx.Frame**, tạo cửa sổ chính của ứng dụng.    * **\_\_init\_\_** khởi tạo các thành phần của GUI bằng cách gọi **InitUI**. 3. **Phương thức InitUI**:    * **Tạo Menu Bar**:      + **menubar = wx.MenuBar()**: Tạo một thanh menu mới.      + **fileMenu = wx.Menu()**: Tạo menu File với các mục Open, Save và Exit.      + **menubar.Append(fileMenu, '&File')**: Thêm menu File vào thanh menu.      + **self.SetMenuBar(menubar)**: Đặt thanh menu cho cửa sổ chính.    * **Tạo Status Bar**:      + **self.CreateStatusBar()**: Tạo thanh trạng thái.      + **self.SetStatusText('Ready')**: Đặt văn bản mặc định cho thanh trạng thái.    * **Kết nối sự kiện**:      + **self.Bind(wx.EVT\_MENU, self.OnOpen, open\_item)**: Kết nối sự kiện chọn mục menu Open với phương thức **OnOpen**.      + **self.Bind(wx.EVT\_MENU, self.OnSave, save\_item)**: Kết nối sự kiện chọn mục menu Save với phương thức **OnSave**.      + **self.Bind(wx.EVT\_MENU, self.OnExit, exit\_item)**: Kết nối sự kiện chọn mục menu Exit với phương thức **OnExit**.    * **Thiết lập kích thước và tiêu đề**:      + **self.SetSize((800, 600))**: Đặt kích thước cho cửa sổ.      + **self.SetTitle('wxPython File Dialog Example')**: Đặt tiêu đề cho cửa sổ.      + **self.Centre()**: Căn giữa cửa sổ trên màn hình. 4. **Phương thức xử lý sự kiện**:    * **OnOpen(self, event)**: Hiển thị hộp thoại tệp tin để chọn tệp mở. Nếu người dùng chọn một tệp, đường dẫn tệp sẽ được hiển thị trong thanh trạng thái.    * **OnSave(self, event)**: Hiển thị hộp thoại tệp tin để chọn tệp lưu. Nếu người dùng chọn một tệp, đường dẫn tệp sẽ được hiển thị trong thanh trạng thái.    * **OnExit(self, event)**: Đóng cửa sổ. 5. **Khởi tạo ứng dụng**:    * **app = wx.App(False)**: Khởi tạo ứng dụng wxPython.    * **frame = MyFrame(None)**: Tạo cửa sổ chính.    * **frame.Show()**: Hiển thị cửa sổ chính.    * **app.MainLoop()**: Bắt đầu vòng lặp sự kiện của ứng dụng. |

**Bảng: Các tham số của hàm khởi tạo wx.FileDialog**

|  |  |
| --- | --- |
| **Tham số** | **Mô tả** |
| **parent** | Cửa sổ cha cho hộp thoại, hoặc **None** nếu không có cửa sổ cha. |
| **message** | Thông báo được hiển thị trong thanh tiêu đề của hộp thoại. |
| **defaultDir** | Thư mục mà hộp thoại nên bắt đầu. Nếu để trống, hộp thoại sẽ bắt đầu ở thư mục làm việc hiện tại. |
| **defaultFile** | Tệp tin được chọn khi hộp thoại mở ra. Nếu để trống, không có tệp tin nào được chọn. |
| **wildcard** | Các tùy chọn bộ lọc cho phép người dùng giới hạn hiển thị các loại tệp tin đã chọn. Định dạng là `<hiển thị> |
| **style** | Kiểu bitmask. Các kiểu được liệt kê trong bảng 6.7. |

**Bảng: Các tùy chọn kiểu cho wx.FileDialog**

|  |  |
| --- | --- |
| **Kiểu** | **Mô tả** |
| **wx.CHANGE\_DIR** | Sau khi người dùng chọn một tệp, thư mục làm việc hiện tại sẽ được thay đổi thành thư mục của tệp đó. |
| **wx.MULTIPLE** | Chỉ áp dụng cho hộp thoại mở, kiểu này cho phép người dùng chọn nhiều tệp. |
| **wx.OPEN** | Kiểu này được sử dụng để mở một tệp. |
| **wx.OVERWRITE\_PROMPT** | Chỉ áp dụng cho hộp thoại lưu, kiểu này hiển thị một lời nhắc để xác nhận lựa chọn nếu một tệp hiện có được chọn để ghi đè. |
| **wx.SAVE** | Kiểu này được sử dụng để lưu một tệp. |

**Làm thế nào để tạo hộp thoại "About"? (How do I build an about box?)**

Hộp thoại "About" là một ví dụ điển hình về hộp thoại hiển thị thông tin phức tạp hơn so với hộp thoại thông báo đơn giản, nhưng không yêu cầu các chức năng khác. Trong trường hợp này, bạn có thể sử dụng **wx.html.HtmlWindow** như một cơ chế đơn giản để hiển thị văn bản có định dạng. Thực ra, **wx.html.HtmlWindow** mạnh mẽ hơn rất nhiều so với những gì chúng ta sẽ trình bày ở đây, và bao gồm các phương thức để quản lý tương tác người dùng và hiển thị chi tiết. Chương 16 sẽ trình bày các tính năng của **wx.html.HtmlWindow**.

Dưới đây là một lớp tạo hộp thoại "About" sử dụng trình hiển thị HTML:

|  |
| --- |
| import wx  import wx.html  class SketchAbout(wx.Dialog):  text = '''  <!DOCTYPE html>  <html>  <head>  <style>  body {  background-color: #f8f9fa;  font-family: Arial, sans-serif;  }  .container {  max-width: 600px;  margin: auto;  padding: 20px;  background-color: #ffffff;  border: 1px solid #dee2e6;  border-radius: 5px;  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);  }  h1 {  color: #343a40;  text-align: center;  }  p {  color: #6c757d;  }  .footer {  text-align: center;  margin-top: 20px;  font-size: 12px;  color: #adb5bd;  }  </style>  </head>  <body>  <div class="container">  <h1>Sketch!</h1>  <p><b>Sketch</b> is a demonstration program for <b>wxPython In Action</b> Chapter 6. It is based on the SuperDoodle demo included with wxPython, available at <a href="http://www.wxpython.org/">http://www.wxpython.org/</a></p>  <p><b>SuperDoodle</b> and <b>wxPython</b> are brought to you by <b>Robin Dunn</b> and <b>Total Control Software</b>, Copyright &copy; 1997-2006.</p>  <div class="footer">  &copy; 2023 Sketch Application  </div>  </div>  </body>  </html>  '''  def \_\_init\_\_(self, parent):  wx.Dialog.\_\_init\_\_(self, parent, -1, 'About Sketch', size=(460, 400))  html = wx.html.HtmlWindow(self)  html.SetPage(self.text)  button = wx.Button(self, wx.ID\_OK, "Okay")  sizer = wx.BoxSizer(wx.VERTICAL)  sizer.Add(html, 1, wx.EXPAND|wx.ALL, 5)  sizer.Add(button, 0, wx.ALIGN\_CENTER|wx.ALL, 5)  self.SetSizer(sizer)  self.Layout()  class MyFrame(wx.Frame):  def \_\_init\_\_(self, \*args, \*\*kw):  super(MyFrame, self).\_\_init\_\_(\*args, \*\*kw)    self.InitUI()    def InitUI(self):  menubar = wx.MenuBar()  helpMenu = wx.Menu()  about\_item = helpMenu.Append(wx.ID\_ABOUT, '&About\tF1', 'About application')  menubar.Append(helpMenu, '&Help')  self.SetMenuBar(menubar)  self.Bind(wx.EVT\_MENU, self.OnAbout, about\_item)  self.CreateStatusBar()  self.SetStatusText('Ready')  self.SetSize((800, 600))  self.SetTitle('wxPython About Dialog Example')  self.Centre()  def OnAbout(self, event):  dlg = SketchAbout(self)  dlg.ShowModal()  dlg.Destroy()  if \_\_name\_\_ == "\_\_main\_\_":  app = wx.App(False)  frame = MyFrame(None)  frame.Show()  app.MainLoop() |
| 1. **Sử dụng CSS để tạo kiểu**:    * Sử dụng thẻ **<style>** trong phần **<head>** để áp dụng CSS cho trang HTML.    * Thiết lập kiểu nền, font chữ và màu sắc cho phần thân trang.    * Sử dụng các lớp CSS để định dạng phần tử HTML, như **.container**, **h1**, **p**, và **.footer**. 2. **Cập nhật nội dung HTML**:    * Tạo phần tử **<div class="container">** để chứa nội dung chính, áp dụng các kiểu CSS đã định nghĩa.    * Thiết lập màu sắc và căn giữa tiêu đề **<h1>**.    * Thêm đoạn văn bản với các thông tin về ứng dụng, bao gồm liên kết đến trang web wxPython.    * Thêm phần tử **<div class="footer">** để hiển thị thông tin bản quyền. 3. **Tạo và hiển thị hộp thoại "About"**:    * Sử dụng **wx.html.HtmlWindow** để hiển thị nội dung HTML.    * Thêm nút "Okay" để đóng hộp thoại. 4. **Tạo ứng dụng wxPython**:    * Lớp **MyFrame** tạo cửa sổ chính với menu Help có mục "About".    * Kết nối sự kiện **EVT\_MENU** với phương thức **OnAbout** để hiển thị hộp thoại "About". 5. **Khởi chạy ứng dụng**:    * Tạo một thể hiện của **wx.App**.    * Tạo và hiển thị cửa sổ chính **MyFrame**.    * Bắt đầu vòng lặp sự kiện của ứng dụng. |

Hầu hết mã này là chuỗi HTML, chứa các thẻ bố cục và phông chữ. Hộp thoại là sự kết hợp của **wx.html.HtmlWindow** và một nút với ID **wx.ID\_OK**. Khi nhấn nút sẽ tự động đóng cửa sổ, giống như các hộp thoại khác. Một sizer dạng hộp dọc được sử dụng để quản lý bố cục.

**Làm thế nào để tạo màn hình chờ (splash screen)?**

Hiển thị một màn hình chờ tuyệt vời khi ứng dụng của bạn khởi động sẽ mang lại vẻ chuyên nghiệp cho người dùng. Nó cũng có thể làm người dùng phân tâm trong khi ứng dụng hoàn thành việc thiết lập tốn thời gian. Trong wxPython, rất dễ để tạo một màn hình chờ từ bất kỳ bitmap nào bằng cách sử dụng lớp **wx.SplashScreen**. Màn hình chờ có thể được hiển thị trong một khoảng thời gian cụ thể, và cho dù thời gian đã được đặt hay chưa, màn hình sẽ luôn đóng khi người dùng nhấp vào nó. Lớp này chủ yếu bao gồm hàm khởi tạo của nó như sau:

|  |
| --- |
| wx.SplashScreen(bitmap, splashStyle, milliseconds, parent, id,  pos=wx.DefaultPosition, size=wx.DefaultSize,  style=wx.SIMPLE\_BORDER|wx.FRAME\_NO\_TASKBAR|wx.STAY\_ON\_TOP) |

**Bảng: Các tham số của hàm khởi tạo wx.SplashScreen**

|  |  |
| --- | --- |
| **Tham số** | **Mô tả** |
| **bitmap** | Một **wx.Bitmap**, đây là những gì hiển thị trên màn hình. |
| **splashStyle** | Kiểu bitmap khác, có thể là bất kỳ kết hợp nào của các giá trị sau: **wx.SPLASH\_CENTRE\_ON\_PARENT**, **wx.SPLASH\_CENTRE\_ON\_SCREEN**, **wx.SPLASH\_NO\_CENTRE**, **wx.SPLASH\_TIMEOUT**, **wx.SPLASH\_NO\_TIMEOUT**. |
| **milliseconds** | Nếu **wx.SPLASH\_TIMEOUT** được chỉ định là **splashStyle**, đây là số mili giây trước khi hết thời gian. |
| **parent** | Cửa sổ cha. Thông thường là **None**. |
| **id** | ID của cửa sổ, thường là -1. |
| **pos** | Vị trí trên màn hình nếu **wx.SPLASH\_NO\_CENTER** là **splashStyle**. |
| **size** | Kích thước. Thông thường không cần chỉ định vì kích thước của bitmap sẽ được sử dụng. |
| **style** | Kiểu khung wxPython thông thường, mặc định là những gì bạn cần. |

Dưới đây là mã cho màn hình chờ:

|  |
| --- |
| import wx  import wx.adv  class SplashScreenFrame(wx.adv.SplashScreen):      def \_\_init\_\_(self):          # Load the bitmap for the splash screen          bitmap = wx.Image("splash.bmp", wx.BITMAP\_TYPE\_BMP).Scale(300, 200).ConvertToBitmap() # Resize bitmap            # Create the splash screen          wx.adv.SplashScreen.\_\_init\_\_(self, bitmap,                                  wx.adv.SPLASH\_CENTRE\_ON\_SCREEN | wx.adv.SPLASH\_TIMEOUT,                                  3000,  # Display for 3000 milliseconds (3 seconds)                                  None, -1,                                  style=wx.SIMPLE\_BORDER | wx.FRAME\_NO\_TASKBAR | wx.STAY\_ON\_TOP)            # Bind the event to close the splash screen when it is clicked          self.Bind(wx.EVT\_CLOSE, self.OnExit)          # Call Yield to allow the splash screen to be shown          wx.Yield()          # Set up a timer to automatically close the splash screen after the timeout          self.timer = wx.Timer(self)          self.Bind(wx.EVT\_TIMER, self.OnExit, self.timer)          self.timer.Start(3000, oneShot=True)      def OnExit(self, event):          self.timer.Stop()          self.Destroy()          main\_frame = MainFrame(None)          main\_frame.Show()          wx.GetApp().SetTopWindow(main\_frame)  class MainFrame(wx.Frame):      def \_\_init\_\_(self, \*args, \*\*kw):          super(MainFrame, self).\_\_init\_\_(\*args, \*\*kw)          self.InitUI()      def InitUI(self):          menubar = wx.MenuBar()          fileMenu = wx.Menu()          about\_item = fileMenu.Append(wx.ID\_ABOUT, '&About\tF1', 'About application')          fileMenu.AppendSeparator()          exit\_item = fileMenu.Append(wx.ID\_EXIT, 'E&xit\tCtrl+Q', 'Exit application')          menubar.Append(fileMenu, '&File')          self.SetMenuBar(menubar)          self.Bind(wx.EVT\_MENU, self.OnAbout, about\_item)          self.Bind(wx.EVT\_MENU, self.OnExit, exit\_item)          self.CreateStatusBar()          self.SetStatusText('Ready')          self.SetSize((800, 600))          self.SetTitle('wxPython Splash Screen Example')          self.Centre()      def OnAbout(self, event):          wx.MessageBox('This is a wxPython example', 'About', wx.OK | wx.ICON\_INFORMATION)      def OnExit(self, event):          self.Close(True)  class MyApp(wx.App):      def OnInit(self):          splash = SplashScreenFrame()          splash.Show()          return True  if \_\_name\_\_ == "\_\_main\_\_":      app = MyApp(False)      app.MainLoop() |
| 1. **Import Thư viện**:    * **import wx**: Import thư viện wxPython.    * **import wx.adv**: Import các phần mở rộng của wxPython, trong đó có **wx.adv.SplashScreen**. 2. **Lớp SplashScreenFrame**:    * Kế thừa từ **wx.adv.SplashScreen**, lớp này tạo và hiển thị màn hình chờ khi ứng dụng khởi động.  |  | | --- | | class SplashScreenFrame(wx.adv.SplashScreen):  def \_\_init\_\_(self): |     Tải hình ảnh bitmap từ tệp **splash.bmp** và chuyển đổi nó thành **wx.Bitmap**:   |  | | --- | | bitmap = wx.Image("splash.bmp", wx.BITMAP\_TYPE\_BMP).ConvertToBitmap() |   Khởi tạo **wx.adv.SplashScreen** với các tham số:   * **bitmap**: Hình ảnh hiển thị trên màn hình chờ. * **wx.adv.SPLASH\_CENTRE\_ON\_SCREEN | wx.adv.SPLASH\_TIMEOUT**: Hiển thị ở giữa màn hình và tự động tắt sau thời gian quy định. * **3000**: Thời gian hiển thị màn hình chờ (3000 mili giây = 3 giây). * **None, -1**: Tham số cha và ID của cửa sổ. * **style=wx.SIMPLE\_BORDER | wx.FRAME\_NO\_TASKBAR | wx.STAY\_ON\_TOP**: Kiểu của khung cửa sổ.  |  | | --- | | wx.adv.SplashScreen.\_\_init\_\_(self, bitmap,  wx.adv.SPLASH\_CENTRE\_ON\_SCREEN | wx.adv.SPLASH\_TIMEOUT,  3000, # Display for 3000 milliseconds (3 seconds)  None, -1,  style=wx.SIMPLE\_BORDER | wx.FRAME\_NO\_TASKBAR | wx.STAY\_ON\_TOP) |   Kết nối sự kiện đóng cửa sổ với phương thức **OnExit** để đóng màn hình chờ khi người dùng nhấp vào:   |  | | --- | | self.Bind(wx.EVT\_CLOSE, self.OnExit) |   Gọi **wx.Yield()** để cho phép màn hình chờ hiển thị ngay lập tức:   |  | | --- | | wx.Yield() |   Tạo bộ đếm thời gian (**wx.Timer**) để tự động đóng màn hình chờ sau 3 giây:   |  | | --- | | self.timer = wx.Timer(self)  self.Bind(wx.EVT\_TIMER, self.OnExit, self.timer)  self.timer.Start(3000, oneShot=True) |   Phương thức **OnExit** dừng bộ đếm thời gian, hủy màn hình chờ và hiển thị khung chính của ứng dụng:   |  | | --- | | def OnExit(self, event):  self.timer.Stop()  self.Destroy()  main\_frame = MainFrame(None)  main\_frame.Show()  wx.GetApp().SetTopWindow(main\_frame) |  1. **Lớp MainFrame**:  * Tạo cửa sổ chính của ứng dụng với menu bar và thanh trạng thái.  |  | | --- | | class MainFrame(wx.Frame):  def \_\_init\_\_(self, \*args, \*\*kw):  super(MainFrame, self).\_\_init\_\_(\*args, \*\*kw)  self.InitUI() |   Phương thức **InitUI** tạo menu bar và các mục menu "About" và "Exit", kết nối sự kiện với các phương thức xử lý tương ứng:   |  | | --- | | def InitUI(self):  menubar = wx.MenuBar()  fileMenu = wx.Menu()  about\_item = fileMenu.Append(wx.ID\_ABOUT, '&About\tF1', 'About application')  fileMenu.AppendSeparator()  exit\_item = fileMenu.Append(wx.ID\_EXIT, 'E&xit\tCtrl+Q', 'Exit application')  menubar.Append(fileMenu, '&File')  self.SetMenuBar(menubar)  self.Bind(wx.EVT\_MENU, self.OnAbout, about\_item)  self.Bind(wx.EVT\_MENU, self.OnExit, exit\_item)  self.CreateStatusBar()  self.SetStatusText('Ready')  self.SetSize((800, 600))  self.SetTitle('wxPython Splash Screen Example')  self.Centre() |   Phương thức **OnAbout** hiển thị thông báo "About":   |  | | --- | | def OnAbout(self, event):  wx.MessageBox('This is a wxPython example', 'About', wx.OK | wx.ICON\_INFORMATION) |   Phương thức **OnExit** đóng ứng dụng:   |  | | --- | | def OnExit(self, event):  self.Close(True) |  1. **Lớp MyApp**:  * Khởi tạo ứng dụng wxPython và hiển thị màn hình chờ khi khởi động:  |  | | --- | | class MyApp(wx.App):  def OnInit(self):  splash = SplashScreenFrame()  splash.Show()  return True |  1. **Khởi chạy ứng dụng**:  * Tạo một thể hiện của **MyApp** và bắt đầu vòng lặp sự kiện của ứng dụng:  |  | | --- | | if \_\_name\_\_ == "\_\_main\_\_":  app = MyApp(False)  app.MainLoop() | |