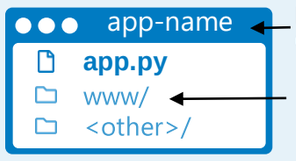
Shiny para Python: : GUÍA RÁPIDA

Crear una aplicación

Una aplicación Shiny es una página web interactiva (ui) impulsada por una sesión de Python en vivo ejecutada por un servidor (o por un navegador con Shinylive).

Los usuarios pueden manipular la interfaz de usuario, lo que hará que el servidor actualice las pantallas de la interfaz de usuario (mediante la ejecución de código Python).

Guarde la aplicación como app.py en un directorio con los archivos que usa.



El nombre del directorio es el nombre de la aplicación.

Coloque imágenes, CSS, etc. para compartir con el navegador en una carpeta llamada "www".

Funciones anidadas de Python para crear una interfaz HTML

Ejecute `shiny create .` en el terminal para generar una plantilla app.py archivo

Inicie aplicaciones con `shiny run app.py --reload`

Agregar entradas con funciones ui.input_*()

Agregar salidas con funciones ui.ouput_*()

Para cada salida, defina una función que genere la salida

Diseño de la interfaz de usuario con funciones de diseño

Especifique el tipo de salida con un @render. decorador

Llame a los valores de las entradas de la interfaz de usuario con input. <id>()

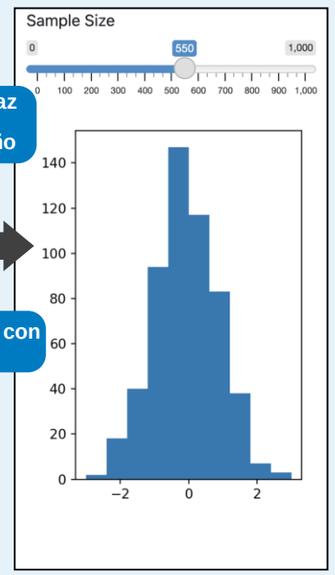
Llame a App() para combinar app_ui y servidor en una aplicación interactiva

```
from shiny import App, render, ui
import matplotlib.pyplot as plt
import numpy as np

app_ui = ui.page_fluid(
  ui.input_slider(
    "n", "Sample Size", 0, 1000, 20
  ),
  ui.output_plot("dist")
)

def server(input, output, session):
  @render.plot
  def dist():
    x = np.random.randn(input.n())
    plt.hist(x, range=[-3, 3])

app = App(app_ui, server)
```

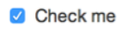


Entradas

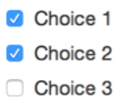
Use una función ui. para crear un widget de entrada que guarde un valor como <id>. Los valores de entrada son reactivos y deben llamarse como <id>().

ui.input_action_button(id, label, icon, width, …**)**

ui.input_action_link(id, label, icon, …**)**



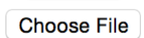
ui.input_checkbox(id, label, value, width**)**



ui.input_checkbox_group(id, label, choices, selected, inline, width**)**

ui.input_date(id, label, value, min, max, format, startview, weekstart, language, width, autoclose, datesdisabled, daysofweekdisabled**)**

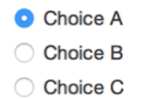
ui.input_date_range(id, label, start, end, min, max, format, startview, weekstart, language, separator, width, autoclose**)**



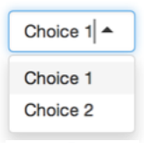
ui.input_file(id, label, multiple, accept, width, buttonLabel, placeholder, capture**)**

ui.input_numeric(id, label, value, min, max, step, width**)**

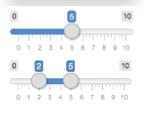
ui.input_password(id, label, value, width, placeholder**)**



ui.input_radio_buttons(id, label, choices, selected, inline, width**)**

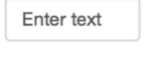


ui.input_select(id, label, choices, selected, multiple, selectize, width, size**)**
También **ui.input_selectize()**



ui.input_slider(id, label, min, max, value, step, ticks, animate, width, sep, pre, post, timeFormat, timezone, dragRange**)**

ui.input_switch(id, label, value, width**)**



ui.input_text(id, label, value, width, placeholder, autocomplete, spellcheck**)**
También **ui.input_text_area()**

Compartir

Comparte tu aplicación de tres maneras:

1. **Alójala en shinyapps.io**, un servicio basado en la nube de Posit. Para implementar aplicaciones Shiny:

   - Crea una cuenta profesional gratis en **shinyapps.io**
   - Usar el paquete reconnect-Python para publicar con **rsconnect deploy shiny <path to directory>**

2. **Compre Posit Connect**, una plataforma de publicación para R y Python. **posit.co/connect**

3. **Use opciones de despliegue de código abierto** **shiny.posit.co/py/docs/deploy.html**

Shinylive

Las aplicaciones Shinylive utilizan WebAssembly para ejecutarse completamente en un navegador, sin necesidad de un servidor especial para ejecutar Python.

- Edite y/o aloje aplicaciones Shinylive at **shinylive.io**
- Crear una versión de Shinylive de una aplicación para implementarlacon

```
shinylive export myapp site
```

A continuación, despliegue en un sitio de alojamiento como Github o Netifly

- Incruste aplicaciones Shinylive en sitios, blogs, etc. de Quarto.

```
---
filters:
  - shinylive
---

An embedded Shinylive app:

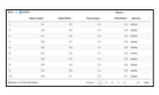
```{shinylive-python}
#| standalone: true

# [App.py code here…]
```
```

Para incrustar una aplicación Shinylive en un documento de Quarto, incluya la sintaxis en negrita.

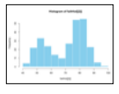
Salidas

Haga coincidir las funciones ui.output_* con los decoradores @render.* para vincular la salida de Python a la interfaz de usuario.



ui.output_data_frame(id**)**
@render.data_frame

ui.output_image(id, width, height, click, dblclick, hover, brush, inline**)**
@render.image

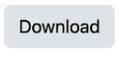


ui.output_plot(id, width, height, click, dblclick, hover, brush, inline**)**
@render.plot

ui.output_table(id**)**
@render.table

ui.output_text_verbatim(id, …**)**
ui.output_text(id, container, inline**)**
@render.text

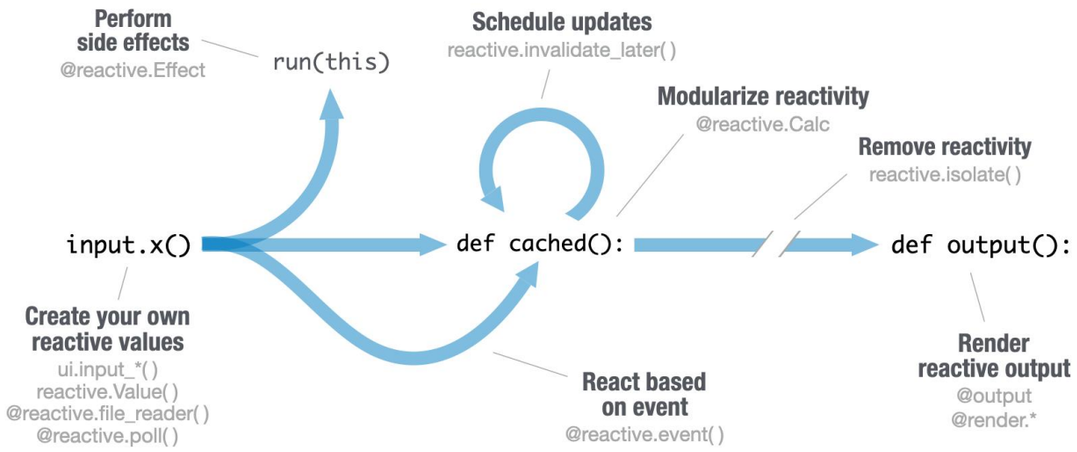
ui.output_ui(id, inline, container, …**)**
ui.output_html(id, inline, container, …**)**
@render.ui



ui.download_button(id, label, icon, …**)**
@session.download

Reactividad

Los valores reactivos funcionan junto con las funciones reactivas. Llame a un valor reactivo desde dentro de los argumentos de una de estas funciones para evitar el error **No current reactive context.**



CREA TUS PROPIOS VALORES REACTIVOS

```
# …
app_ui = ui.page_fluid(
  ui.input_text("a", "A")
)

def server(
  input, output, session
):
  rv = reactive.Value()
  rv.set(5)
# …
```

ui.input_*() crea un widget de entrada que guarda un valor reactivo como entrada. <id>().

reactive.value( )
Crea un objeto cuyo valor se puede establecer.

MOSTRAR SALIDA REACTIVA

```
app_ui = ui.page_fluid(
  ui.input_text("a", "A"),
  ui.output_text("b"),
)
def server(
  input, output, session
):
  @render.text
  def b():
    return input.a()
```

ui.output_*()
agrega un elemento de salida a la interfaz de usuario.

@render.*
Decorador para identificar y renderizar salidas

def <id>(): Código para generar la salida

CREACIÓN DE EXPRESIONES REACTIVAS

```
# …
def server(
  input, output, session
):
  @reactive.calc
  def re():
    return input.a()  +
input.b()
# …
```

@reactive.calc
Convierte una función en una expresión reactiva. Shiny notifica a las funciones que usan la expresión cuando se invalida, lo que desencadena un nuevo cálculo. Shiny almacena en caché el valor de la expresión mientras es válida para evitar cálculos innecesarios.

REALIZAR EFECTOS SECUNDARIOS

```
# …
def server(
  input, output, session
):
  @reactive.effect
  @reactive.event(input.a)
    def print():
      print("Hi")
# …
```

@reactive.effect
Activar de forma reactiva una función con un efecto secundario. Llame a un valor reactivo o use @reactive.event para especificar cuándo se volverá a ejecutar la función.

REACCIONAR EN FUNCIÓN DEL EVENTO

```
# …
def server(
  input, output, session
):
  @reactive.Calc
  @reactive.event(input.a)
  def re():
    return input.b()
# …
```

@reactive.event( )
Hace que una función reaccione solo cuando se invalida un valor especificado, aquí input.a.

ELIMINAR LA REACTIVIDAD

```
# …def server(
  input, output, session
):
  @render.text
  def a():
    with reactive.isolate():
      return input.a()
# …
```

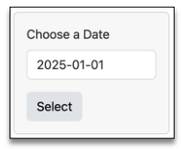
reactive.isolate()
Cree un contexto no reactivo dentro de una función reactiva. Llamar a un valor reactivo dentro de este contexto no hará que la función de llamada se vuelva a ejecutar en caso de que el valor deje de ser válido.

Diseños

Combine varios elementos en un "solo elemento" que tenga sus propias propiedades con una función de panel:

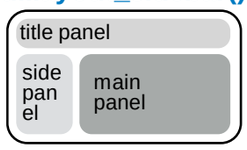
ui.panel_absolute()
ui.panel_conditional()
ui.panel_fixed()
ui.panel_main()
ui.panel_sidebar()
ui.panel_title()
ui.panel_well()
ui.row() / ui.column()

```
ui.panel_well(
  ui.input_date(…),
  ui.input_action_button(…)
)
```



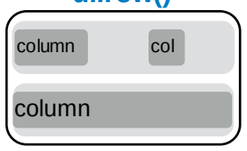
Paneles de diseño con una función de diseño. Agregue elementos como argumentos de las funciones de diseño.

ui.layout_sidebar()




```
app_ui = ui.page_fluid(
  ui.panel_title( ),
  ui.layout_sidebar(
    ui.panel_sidebar( ),
    ui.panel_main( ),
  ))
```

ui.row()




```
app_ui = ui.page_fluid(
  ui.row(
    ui.column(width = 4),
    ui.column(width = 2, offset = 3),
  ),
  ui.row(ui.column(width = 12)))
```

Capa ui.nav( )s una encima de la otra y navega entre ellos, con:


```
ui.page_fluid(ui.navset_tab(
    ui.nav("tab 1", "contents"),
    ui.nav("tab 2", "contents"),
    ui.nav("tab 3", "contents")))
```

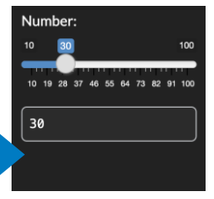

```
ui.page_fluid(ui.navset_pill_list(
    ui.nav("tab 1", "contents"),
    ui.nav("tab 2", "contents"),
    ui.nav("tab 3", "contents")))
```


```
ui.page_navbar(
    ui.nav("tab 1", "contents"),
    ui.nav("tab 2", "contents"),
    ui.nav("tab 3", "contents"),
    title = "Page")
```

Temas

Use el paquete shinyswatch para agregar temas de arranque existentes a la interfaz de usuario de la aplicación Shiny.

```
import shinyswatch
app_ui = ui.page_fluid(
  shinyswatch.theme.darkly(),
  …
)
```



Shiny para R Comparación

Shiny para Python es bastante similar a Shiny para R con unas pocas importantes diferencias:

1. Llama a las entradas como **input.<id>()**	input$x	input.x()
2. Definir salidas como funciones decoradas **def <id>():**	output$y <- renderText(z())	@renderText def y(): return z()

3. Para crear una expresión reactiva, utilice **@reactive.calc**	z <- reactive({
	  input$x + 1
	})
		@reactive.calc
		def z():
		  return input.x()+1

4. Para crear un observador, utilice **@reactive.effect**	a <- observe({
	  print(input$x)
	})
		@reactive.effect
		def a():
		  print(input.x())

5. Combinar estos con **@reactive.event**	b <-
	  eventReactive(
	  input$goCue,
	  {input$x + 1}
	)
		@reactive.calc
		@reactive.event(
		  input.go_cue
		)
		def b():
		  return input.x()+1

6. Usa **reactive.value()** en lugar de reactiveVal()	reactiveVal(1)	reactive.value(1)
7. Usa **nav_*()** en lugar de *Tab()	insertTab() appendTab() etc.	nav_insert() nav_append() etc.
8. Las funciones se organizan de forma intuitiva en submódulos	dateInput() textInput() etc.	ui.input_date() ui.input_text() etc.