

**desde 0**

Guía paso a paso con ejemplos para desplegar tu prototipo de IA en internet usando Streamlit, la herramienta preferente en Saturdays.AI para ello

## 

[**Requisitos**](#_x7z1b0albor8) **1**

[**Instalación**](#_mcy2v875trw2) **2**

[**Despliegue**](#_8yb0tuaspuk2) **3**

[Despliegue local](#_6rtli6akpf6) 3

[Despliegue online](#_iv88allmjy4q) 6

[**Uso de Streamlit**](#_fjymgkeljzg3) **9**

[Funciones de texto:](#_43b1zu2v7ou7) 9

[Widgets:](#_rb1p22gcf8rm) 11

[Inputs](#_pcm84le5ai0e) 15

[Gráficos](#_zerrjh8vnb8e) 16

[Mapas](#_vhng389lra7a) 16

[Tamaño y posicionamiento](#_ecx5q7415wyd) 16

[Contenedores](#_oh0nuuaygq5a) 17

[Multimedia](#_5n2ypctv8ear) 19

[**Ejemplos**](#_lqk150qe6es5) **20**

[Streamlit y machine learning](#_8asad9xqjqlt) 20

[Código de entrenamiento de modelo](#_y3v3p94vp90m) 20

[Código Streamlit](#_3g0lqgupkwpk) 21

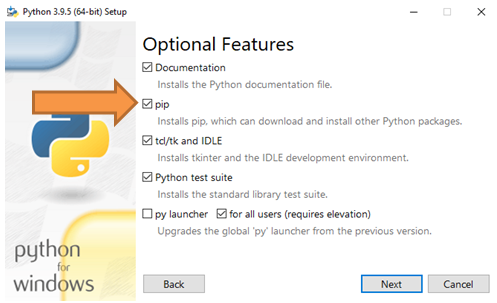
[Streamlit y el tratamiento de imágenes](#_5kekv8qu1e7n) 22

[Gráficos y mapas:](#_gyal99u64sdg) 23

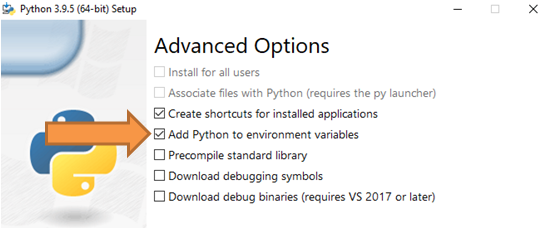
## 

## Requisitos (en local)

Para emplear Streamlit solo necesitaremos tener instalado python y pip (el gestor de archivos de python), pip se instala por defecto en todas las versiones actuales de python y por lo que si tienes una antigua te recomendamos actualizar.

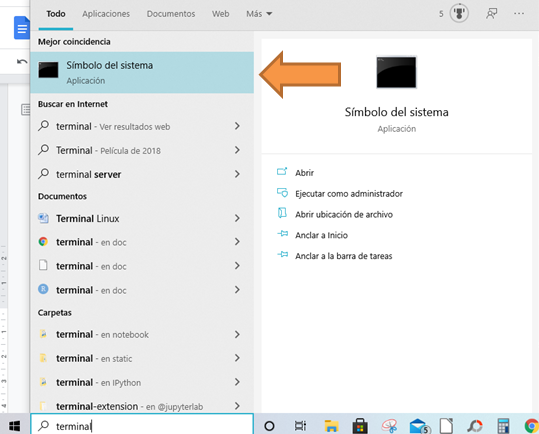


Respecto a la instalación de Python no entraremos en detalle puesto que es bastante sencilla, solo hay que entrar en la web oficial descargarlo e instalarlo (y si tenéis cualquier duda hay miles de tutoriales en internet). Sin embargo, es importante que Python se añada a las variables de entorno, sino el sistema operativo no será capaz de encontrarlo y en la práctica será como si no estuviese instalado, durante la instalación debería añadirse por defecto, sino es así tendréis que hacerlo manualmente, esto no es demasiado complicado y de nuevo podéis encontrar tutoriales online fácilmente.

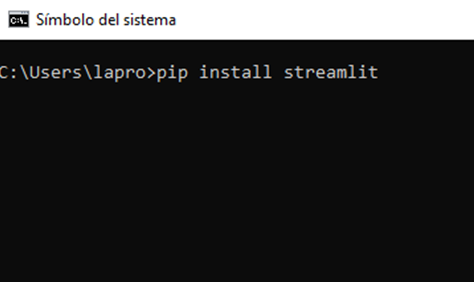


## Instalación

Con Python preparado solo hay que realizar dos pasos para usar streamlit, entrar en la pantalla de comandos:



Escribir “pip install streamlit” y pulsar enter:



## Despliegue

Puede parecer que analizar los métodos de despliegue antes de aprender a usar propiamente streamlit es empezar la casa por el tejado, pero he considerado que es mejor que veamos esto primero porque así comprobaremos que todo funciona correctamente y porque en realidad el despliegue con streamlit, sea local u online resulta bastante sencillo así que podemos quitarlo de enmedio para centrarnos en el código en sí.

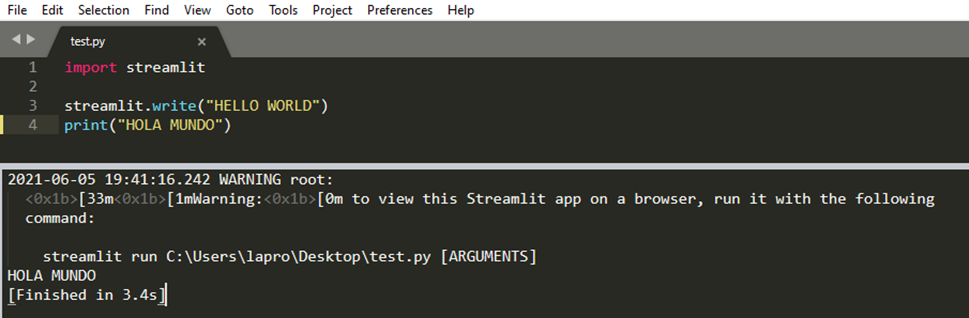
Para empezar crearemos un archivo minimo que vamos a desplegar, empleando un editor de código a vuestra elección o incluso el bloc de notas de windows cread un archivo llamado test (podéis usar cualquier otro nombre) con extensión py, este archivo contendrá el texto:

import streamlit

streamlit.write(“HELLO WORLD”)

### Despliegue local

Si intentas ejecutar un archivo con código streamlit de forma convencional, por ejemplo en el propio editor de código, verás que este se ejecuta de forma normal ignorando los comandos de streamlit.

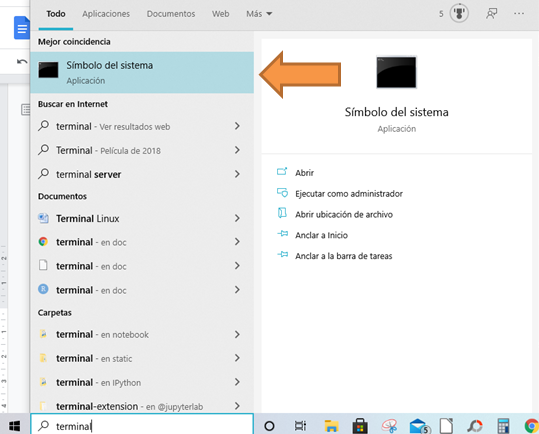


De hecho, como vemos en la imagen, aparece un warning indicando lo que debemos hacer en realidad, para ejecutar localmente un archivo de Streamlit tendremos que acceder a la pantalla de comandos y usar streamlit run para ejecutar el archivo.

DESPLAZAMIENTO POR LA PANTALLA DE COMANDOS

Si tienes experiencia con la pantalla de comandos puedes saltarte este apartado, en caso contrario no te preocupes te lo explicaremos paso a paso.

Para empezar la abrimos:

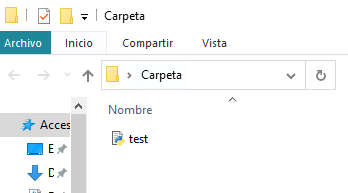


Una vez dentro aparecemos en nuestra carpeta de usuario para desplazarnos desde aquí usaremos dos comandos cd y dir.

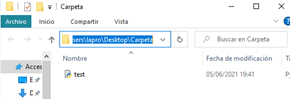
**dir** muestra por escrito lo que hay en la carpeta actual, si lo ejecutas desde la carpeta de tu usuario verás aparecer muchos archivos de los que no tienes que preocuparte ahora mismo.

**cd** te permite desplazarte a otra carpeta.

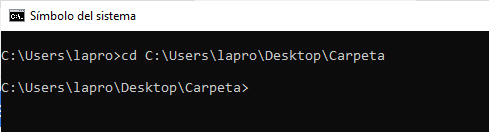
Empezaremos usando cd Escritorio o cd Desktop para acceder al escritorio (puedes emplear dir para comprobar si aparece como Escritorio o Desktop pero también puedes probarlo directamente, (si te equivocas solo aparecerá un mensaje diciendo que la carpeta no existe), después sigue usando cd hasta llegar a la carpeta donde hayas guardado test.py y una vez ahí ejecuta streamlit run test.py. También puedes acceder a la carpeta directamente copiando su ruta y ejecutando cd a ella:



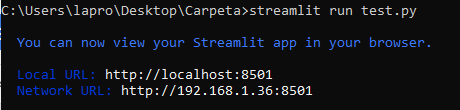
Al clicar en el nombre de la carpeta aparece su ruta:



Accedemos con cd:



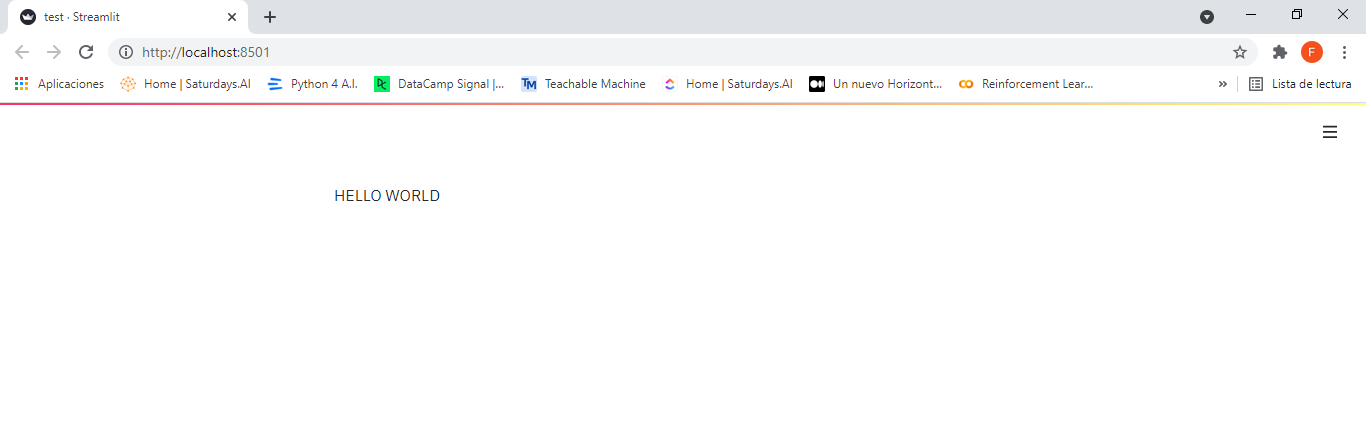
Y usamos streamlit run:



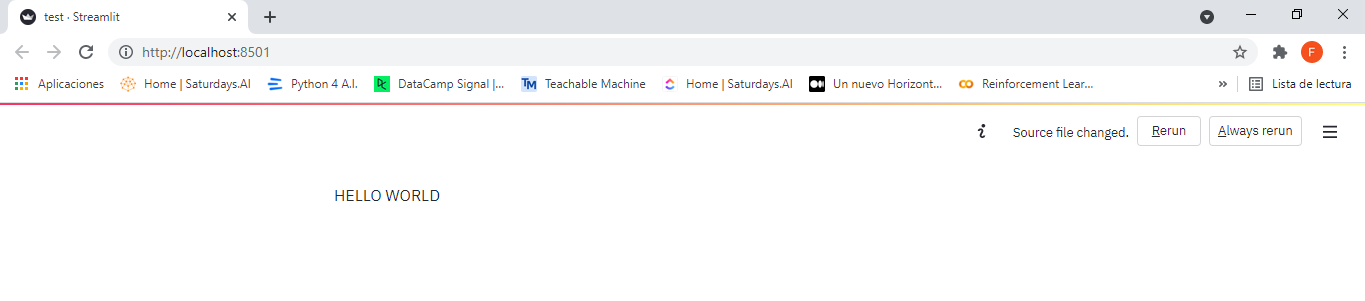
**Atención:**

* Al usar streamlit run debemos indicar la extensión py, sino no encontrara el archivo.
* Streamlit solo funciona con archivos de python, ten en cuenta que aunque jupyter notebook y los archivos de google colab estén escritos con python tienen formato ipynb por lo que no son compatibles, si quieres ejecutar código de uno de ellos con streamlit copialos a un archivo con extensión py.

Una vez hayas ejecutado streamlit run se abrirá el archivo test en una pestaña nueva en tu navegador y la pantalla del terminal quedará congelada.



Ahora si haces cualquier cambio en el archivo test (y lo guardas) aparecerá un aviso para que recargues la página:



Entra en test.py y añade al final:

streamlit.checkbox("Checkbox")

Guarda los cambios, vuelve al navegador y presiona en rerun o recarga la página, verás como a aparecido un checkbox completamente funcional.

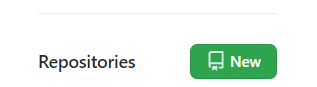
Y ya está, el despliegue local no tiene mayor dificultad, se ejecuta con streamlit run y se recarga después de hacer cambios en el archivo fuente. Cuando hayas acabado abre el terminal, que seguirá congelado, y pulsa control c, la ejecución finalizará y ya no será posible interactuar con la página (no podrás marcar el checkbox).

### Despliegue online

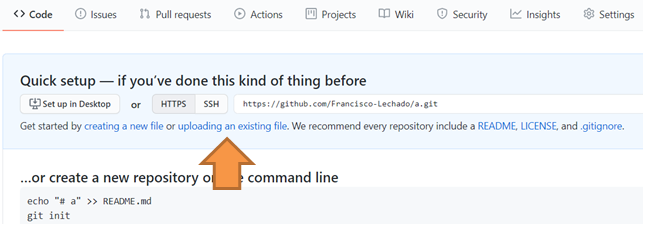
Streamlit te permite desplegar tus proyectos online con la misma facilidad que si lo hicieras localmente y de forma gratuita (con un máximo de tres proyectos simultáneos).

Esto requiere una cuenta de Github y una cuenta de Streamlit, ambas se pueden obtener de forma sencilla registrándose en sus respectivas webs.

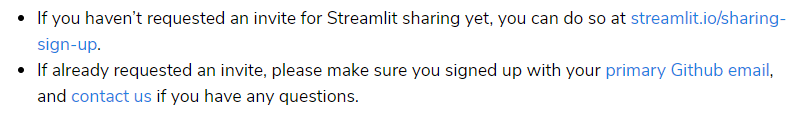
Primero crea tu cuenta de Github, entra en ella y sube tu archivo test.py a un repositorio, Github puede parecer complejo de usar en un principio, de nuevo puedes buscar un tutorial extendido en internet, pero para nuestros propósitos basta con que crees un repositorio (si no tienes ya uno) pulsando en repositores new en la parte superior izquierda de la página:



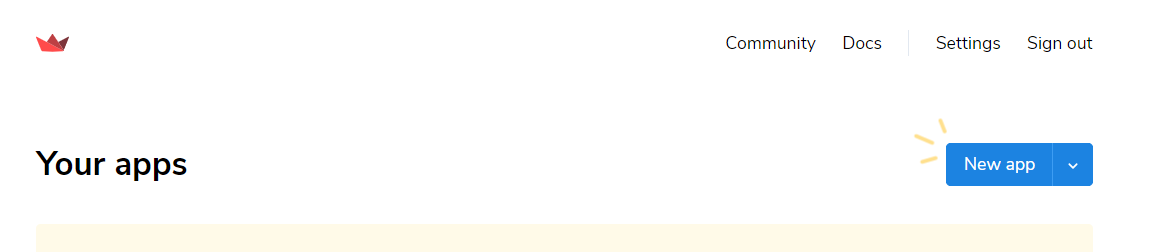
Elijas un nombre y pulsa en Create repository. Una vez hecho selecciona uploading an existing file:



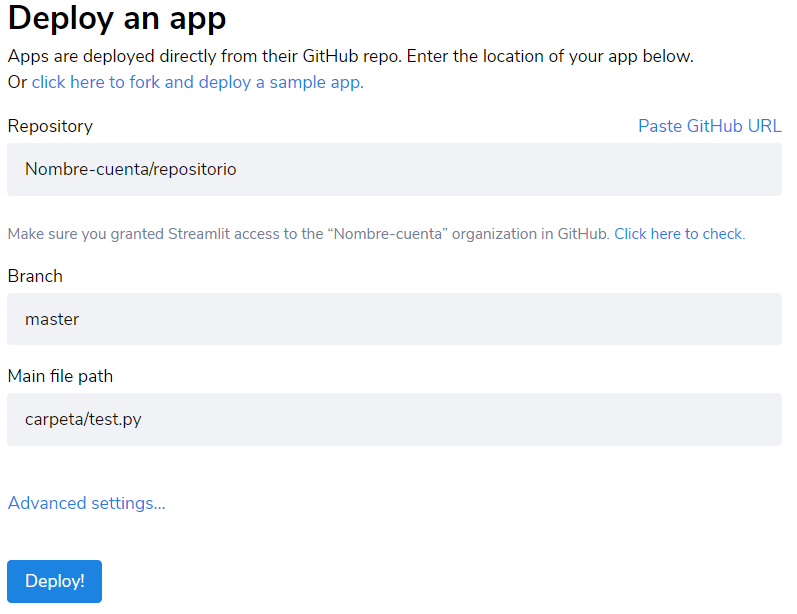
Arrastra test.py dentro de la página, baja y presiona commit changes. Cabe mencionar de nuevo que Github tiene mucha más profundidad que esto y puedes encontrar la información que necesites online, pero por ahora es suficiente, ya puedes entrar en Streamlit, selecciona sign in, continue with GitHub y sigue el enlace para solicitar una invitación.



Introduce tu nombre apellidos y email y finaliza la solicitud, ahora tendrás que esperar unos días hasta que la acepten, mientras tanto puedes ir preparando tu web localmente. En los próximos días recibirás una notificación en tu email, indicando que tu cuenta está preparada, entra en streamlit.io con tu cuenta de Github y presiona New app



Selecciona el archivo que quieres desplegar y pulsa deploy:



Ya estás online, comparte el enlace con quien quieras.

**Atención:** Hay librerías de Python que requieren instalarse para poder usarse (normalmente mediante pip install librería), de la misma forma para poder acceder a dichas librerías desde el despliegue online, debemos notificar a github de que las necesitamos, para esto solo necesitas añadir a tu repositorio un archivo, llamado específicamente requirements.txt, que contenga estas librerías e idealmente la versión específica de ellas que se usa en nuestro proyecto (solo debe estar presente en el repositorio no es necesario hacer nada con él), no todas las librerías necesitan estar listadas, ni es necesario indicar la versión, pero es lo más recomendable para evitar problemas futuros.

Nuestro archivo requirements.txt puede contener, por ejemplo:

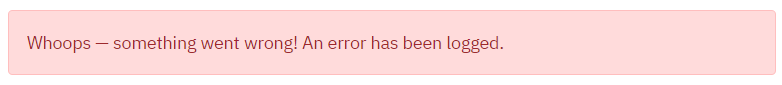
streamlit==0.81.0

pandas==1.2.3

Preparar el archivo requirements puede resultar tedioso y es fácil cometer errores con las versiones de cada librería, por suerte esto se puede automatizar, para ello abrimos la pantalla de comandos una vez más, ejecutamos pip install pipreqs, nos trasladamos a la carpeta donde se encuentre nuestro proyecto y ejecutamos pipreqs --encoding utf8

O cualquier otro formato que estemos utilizando (utf8 es el más habitual) y se generará automáticamente el archivo requirements.txt.

Si accedemos al proyecto sin indicar alguna de las librerías necesarias en un archivo requirements.txt veremos un mensaje de error:



Si tenemos abierta la cuenta de github propietaria del archivo aparecerá un botón en la parte inferior derecha de la pantalla:



Pulsalo para desplegarlo y entre otras cosas verás un mensaje tipo no module named ‘nombre’ (los cambios en requirements.txt pueden tardar unos segundos en actualizarse mientras se instalan las librerías adecuadas, cuando esto sucede aparecen notificaciones en este espacio).

## Uso de Streamlit

Y por fin hemos llegado al código, a partir de aquí asumimos que tienes unos conocimientos básicos de Python, aunque sean lo mínimos, saber que es una variable, que es una función y cómo se usan.

De entrada, cabe mencionar que para usar Streamlit debemos importarlo y que habitualmente se usa la abreviatura st para referirse a él, por tanto, antes de usarlo añadiremos al archivo:

import streamlit as st

Y que para ver en profundidad todas las opciones de Streamlit podéis consultar la api

<https://docs.streamlit.io/en/stable/api.html>

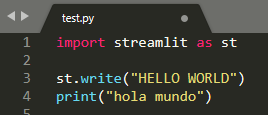
Dicho esto, podemos empezar.

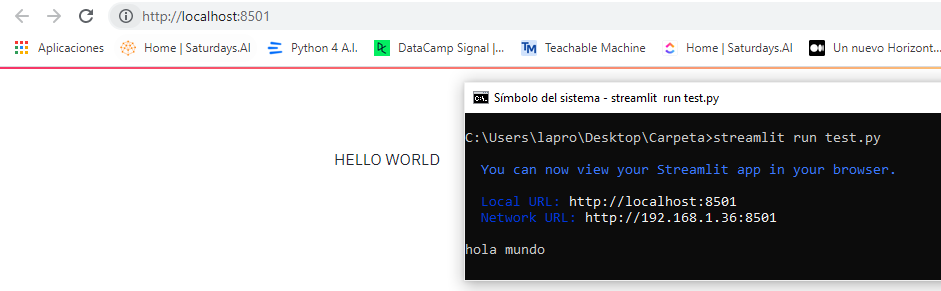
### Funciones de texto:

Las funciones más simples de streamlit son probablemente las de texto, empecemos por write.

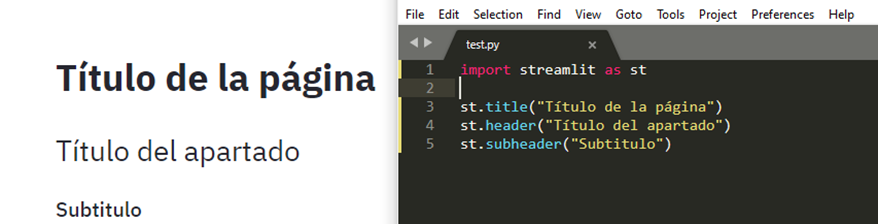
**st.write(“texto”)**

Escribirá dicho texto en la página, es importante recordar que no podemos usar print con streamlit, cualquier texto que intentes imprimir con print aparecerá en la pantalla de comando.





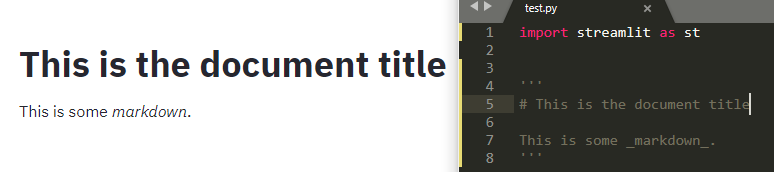
De forma similar tenemos las funciones **title**, **header** y **subheader**:



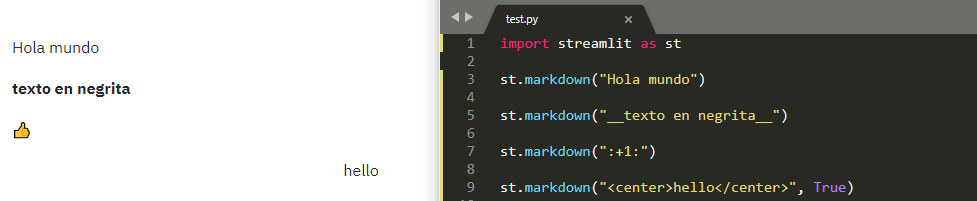
Y finalmente, **st.markdown()**, esta función permite usar markdown para añadir texto con formato y emoticonos.

<https://raw.githubusercontent.com/omnidan/node-emoji/master/lib/emoji.json> <https://katex.org/docs/supported.html>.

La notación magic commands que consiste en poner el texto entre tres pares de comillas permite usar markdown directamente.



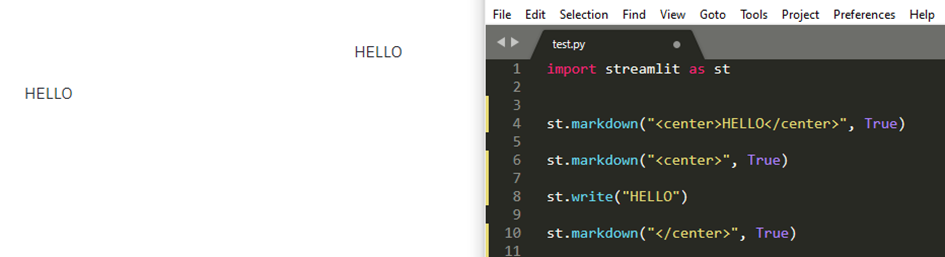
Pero posiblemente lo más importante es que tanto write como markdown (sin incluir magic commands) tienen un parámetro secundario llamado unsafe\_allow\_html que, con el valor True asignado, nos permitirá usar código html dentro de nuestro proyecto, lo cual por supuesto incluye la posibilidad de manipular css y javascript de forma convencional.



(en el caso de write tienes que especificar el nombre del parámetro para que funcione unsafe\_allow\_html = True)

No obstante el uso de html está contraindicado por el equipo de Streamlit que advierte que esta opción puede ser eliminada en cualquier momento, como se especifica en la api y en este hilo de conversación <https://github.com/streamlit/streamlit/issues/152>.

Por último, cabe destacar que cada elemento de Streamlit funciona como un iframe html, es decir se genera independientemente y luego se inserta en la página, lo implica que no es posible, por ejemplo usar el comando anterior dejando abierta una etiqueta <center> para luego añadir un widget (de los que veremos a continuación) y cerrar la etiqueta después de modo que este quede centrado, el primer write (o markdown) se generará aparte, por lo que la etiqueta se cerrará automáticamente o se producirá un error.



### Widgets:

Las funciones que veremos a continuación insertan un widget en la página y devuelven su valor actual, todas ellas comparten tres parámetros:

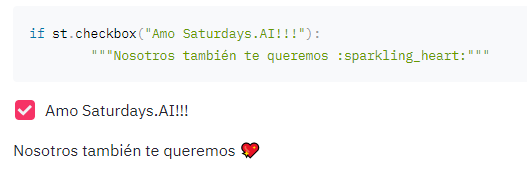
* **Label** el primer parámetro de todos los widgets, es un string para presentar el widget al usuario.
* **key** parámetro opcional que identifica internamente a los widgets, debe ser único, es un string que normalmente se genera de forma automática a partir de los parámetros del widget y, por tanto, si por algún motivo tienes que añadir multiples widgets iguales verás un mensaje de error DuplicatedWidgetID, basta con asignarles keys manualmente para evitar el error.
* **help** este parámetro opcional permite añadir información adicional, asignale un string y aparecerá un icono que lo mostrara cuando el usuario pase el ratón por encima:



**st.checkbox**

Inserta un checkbox en la página y devuelve True (está marcado) o False (no lo está)

Por ejemplo:



Así de sencillo, también contiene un parámetro opcional value, al que se le puede asignar el valor True para hacer que el checkbox aparezca marcado por defecto.

**st.radio**

Inserta una lista de strings con botones radio y devuelve el valor del que este seleccionado. Sus parámetros son las opciones (en forma de lista, tupla, numpy.ndarray, pandas.Series, o pandas.DataFrame) y opcionalmente:

* index (entero que decide el valor marcado por defecto).
* format\_func función que al recibir los valores de las opciones como parámetro devuelva el string que se mostrará al usuario (en lugar del valor real).

**st.selectbox**

Idéntico al anterior con un desplegable de opciones en lugar de botones radio.

**st.multiselect**

un selectbox que permite seleccionar más de una opción, index queda sustituido por una lista de strings (que, lógicamente, deben estar entre las opciones dadas) y, por supuesto, en lugar de un valor devuelve un lista.



**st.slider**

Inserta un slider, por defecto entre 0 y 100, devuelve el valor o valores seleccionados. Parámetros opcionales:

* min\_value Mínimo.
* max\_value Máximo.
* value Valor por defecto o lista/tupla de dos valores (en ese caso el slider selecciona un rango).
* step Intervalo de cambio (es decir si es 5 al mover el slider el valor cambiará de 5 en 5)
* format el formato para int/float soporta: %d %e %f %g %i el de date/time/datetime usa notación Moment.js: <https://momentjs.com/docs/#/displaying/format/>

Todos los valores deben ser del mismo tipo, es decir si se quiere un mínimo de 1 y un máximo de 10 con un intervalo de 0,5 habrá que indicar 1.0 y 10.0 o se producirá un error.

**st.select\_slider**

Un slider al que se le asignan opciones (en forma de lista, tupla, numpy.ndarray, pandas.Series, o pandas.DataFrame) siendo string en vez de números.



#### Inputs

**st.text/number/date/time\_input** permiten capturar susodichos inputs de los usuarios.

también tienen los parámetros opcionales:

* value (valor por defecto)
* Para text max\_chars y type (default o password)
* Para number min\_value, max\_value, step (intervalo de salto) y formato ( %d %e %f %g %i %u)
* Para date min\_value, max\_value.

**st.color\_picker** parámetro opcional value (valor por defecto)

**st.text\_area** opcional height y max\_chars.

**st.file\_uploader**

Para que los usuarios suban archivos.

Parámetros opcionales:

* type string o lista de extensiones admitidas
* accept\_multiple\_files (True/False)

**st.button**

Introduce un botón y devuelve True si ha sido pulsado, pero espera, ¿cómo que si ha sido? ¿No debería ser un listener que se active cuando se pulsa? No, esto no es posible porque:

**Atención:** cada vez que que se modifica el valor de un componente de Streamlit la página se recarga desde 0, por ejemplo si pulsa un botón la página se vuelve a cargar con este devolviendo el valor True y si después se cambia el valor de un slider la página vuelve a cargar con el valor del slider actualizado y el botón devolviendo False.

Esto supone que si alguna parte del código requiere unos segundos para cargar (un gráfico, por ejemplo) cada vez que se modifique cualquier valor habrá que esperar de nuevo por lo que es recomendable que estos elementos solo se presenten cuando está todo preparado (cosa que podemos determinar usando un botón, también podemos usar st.stop() para detener la ejecución antes de cargarlo o un formulario que veremos con los contenedores)

### Gráficos

Cabe destacar que para mostrar gráficos en streamlit se puede usar st.write y un gráfico de una librería compatible (Matplotlib, Plotly...) o una de las funciones chart (line\_chart, area\_chart, bar\_chart…) la lista es extensa y, por supuesto, la encontrarás detallada en la api, cabe destacar se puede alterar el tamaño de los gráficos con los paramentros height y width, pero que este último será ignorado si no se establece use\_container\_width = False.

### Mapas

**st.map**

Se puede llamar sin parámetros para mostrar un mapa del mundo.

Opcional:

* data: un dataframe que contenga las columnas lat/latitude y lon/longitude, se marcaran en el mapa dicho puntos.
* zoom: a traves de un entero
* use\_container\_width: True/False

### Tamaño y posicionamiento

Habrás notado que los elementos de streamlit se apilan verticalmente y que muchos de los widgets se extienden horizontalmente por toda la pantalla, como podemos controlar esto:

El aspecto general de la página está dictado por el **Theme**, que contiene el color principal (el rosa que aparece en los elementos que insertamos), el de fondo, un color de fondo secundario y el del texto y su fuente. Para cambiar el Theme de un proyecto crea en la misma carpeta que lo contenga una carpeta llamada .streamlit y un archivo config.toml dentro de ella, este debe contener:

[theme]

primaryColor="#col"

backgroundColor="#col"

secondaryBackgroundColor="#col"

textColor="#col"

font="fuente"

Donde col es el color que hayas escogido (6 dígitos hexadecimales), puedes probar distintas configuraciones desde tu página desplegada localmente pulsando en el menú (esquina superior derecha), Settings, edit active theme, cuando te guste el aspecto pulsa Copy theme to clipboard y pegalo en config.toml, a partir de ahora localmente este aparecerá cuando tengas seleccionado custom setting, online copia .streamlit en la carpeta de github y quedará seleccionado por defecto.

Si te parece que los márgenes son demasiado amplios puedes usar **st.set\_page\_config(layout="wide")** por desgracia no existen otras opciones de leyout a demás de centered (opción por defecto) debe ser la primera línea de streamlit que se ejecute en el archivo, también permite establecer un título (page\_title) y un icono (page\_icon) para la página.

### Contenedores

Hay distintos tipos de contenedores en streamlit, estos tienen en común que se pueden usar con notación with o como variables, es decir:

with contenedor():

st.write(“texto”)

st.slider(“texto”)...

o

c = contenedor

c.write(“texto”)

c.slider(“texto”)...

Los contenedores son:

**st.form**

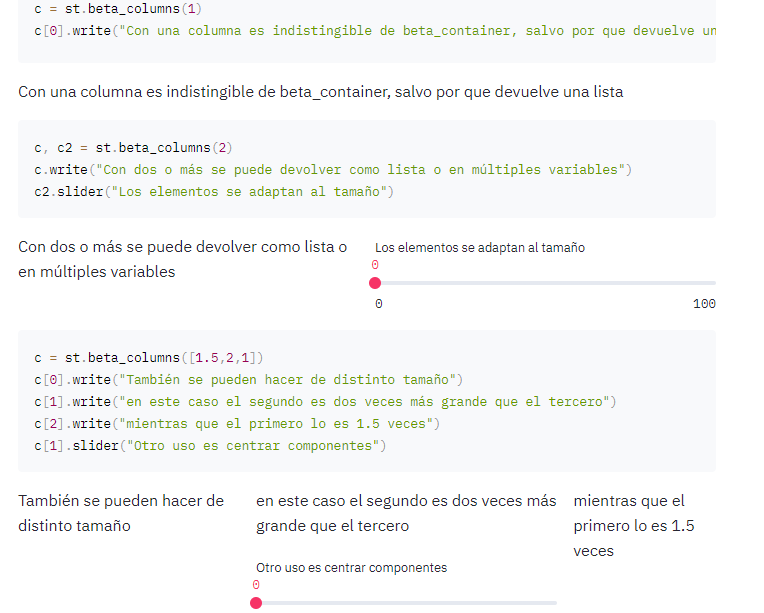
Formulario, se le debe asignar manualmente una key, sus elementos debe ir seguidos y debe finalizar con st.form\_submit\_button() con label opcional, la página no se recargará al modificar el valor de sus elementos, solo al pulsar el botón, las keys de sus elementos se pueden repetir (mientras no sea dentro del mismo formulario), la key del propio formulario debe ser única.

**st.beta\_container**

Contenedor genérico para agrupar elementos:

**st.beta\_columns**

Devuelve contenedores agrupados horizontalmente:



**beta\_expander**

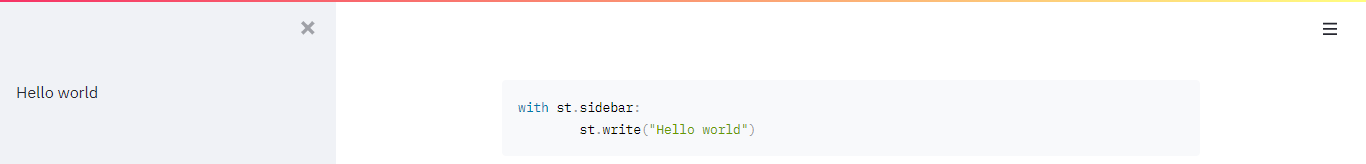
Es un botón que al clicarse se expande para mostrar el contenido tiene un label y un parámetro opcional expanded que permite hacer que se despliegue por defecto.

**empty**

Funciona beta\_container con una simple diferencia, los elementos nuevos sustituyen a los anteriores en lugar de añadirse a continuación, lo que permite crear o transformar elementos dinámicamente.

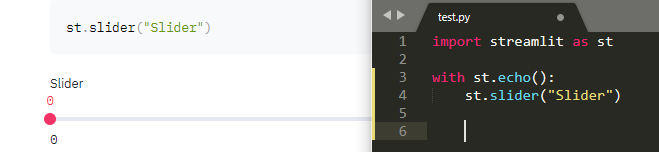
**side\_bar**

Es un contenedor desplegable lateral.



Mostrar código

Por último cabe mencionar que el código mostrado con las funciones aparece gracias a la función echo:



### Multimedia

Ya sabemos que se puede insertar contenido multimedia a través de st.write y st.markdown, mediante html, pero existe también un forma nativa.

**st.Image**

Como su nombre indica permite insertar imágenes, tiene multitud de opciones que podéis consultar en la api, la forma más simple de usarla es indicando como parámetro la url de una imagen, cabe mencionar que tiene el parámetro opcional width.

**st.audio y st.video**

También aceptan el archivo de distintas formas que podéis consultar (incluyendo una url), pero sus parámetros opcionales son más sencillos, format y start\_time.

## Ejemplos

### Streamlit y machine learning

Aunque es perfectamente posible trabajar con streamlit y modelos de aprendizaje, no es algo muy lógico, puesto que los modelos tardan en prepararse y streamlit tiene tendencia a recargar la página, streamlit se ocupa de la presentación al usuario final y los modelos son parte del backend y además streamlit no es compatible con el formato ipynb que facilita tanto el desarrollo de estos proyectos (al permitir ejecutar sólo partes específicas del código), lo que además impide aprovechar recursos como google colab, implica esto que streamlit no tiene espacio en el área de la IA, por supuesto que no, todos los problemas que hemos repasado no están causados por incompatibilidad de streamlit con los modelos, sino por la incompatibilidad de entrenar un modelo con el front end en general, los modelos deben prepararse por separado y unirse a streamlit cuando están preparados para llegar al público, el primer paso será entrenar un modelo y guardar el resultado (para esto podemos usar librerías como pickle o joblib), después lo cargaremos el modelo ya entrenado en un archivo de Python y usaremos streamlit para facilitar al usuario la introducción los datos y para mostrar el resultado, por ejemplo:

Si tienes alguna experiencia con machine learning es probable que te hayas encontrado alguna vez con el dataset Iris, dado que por su simpleza se utiliza mucho con propósitos educativos, contiene datos para clasificar variedades una flor y no necesita ningún tratamiento previo, copia el siguiente código en un archivo de python, ejecutalo y generará tres archivos en la misma carpeta, iris, con el modelo entrenado, max una lista de los valores máximos que puede tomar cada columna y min, su equivalente para los mínimos.

#### Código de entrenamiento de modelo

# Importamos las librerias:

from sklearn import datasets

from sklearn import tree

import joblib

import pandas as pd

# Cargamos los datos, iris contendra data (los parametros) y target (la clase).

iris = datasets.load\_iris()

# Comvertimos los parametros en un pandas.DataFrame.

df\_x = pd.DataFrame(iris.data, columns=["longitud\_sepalo", "anchura\_sepalo", "longitud\_petalo", "anchura\_petalo"])

# Guardamos el máximo y mínimo de cada columna en una lista y está en un archivo independiente.

max\_values = []

min\_values = []

for i in df\_x:

max\_values.append(max(df\_x[i]))

min\_values.append(min(df\_x[i]))

joblib.dump(max\_values,"max.pkl")

joblib.dump(min\_values,"min.pkl")

# Aquí preparariamos el modelo, en este ejemplo hemos escogido un dataset sencillo que no necesita preparacion asi que entrenamos el modelo directamente.

tree = tree.DecisionTreeClassifier()

model = tree.fit(iris.data, iris.target)

# Guardamos el modelo entrenado en un archivo independiente.

joblib.dump(model, "iris.pkl")

Después copia este código en otro archivo que esté en la misma carpeta que los generados por el código anterior y pruebalo (las longitudes tienen más pesos que las anchuras):

#### Código Streamlit

import joblib

import streamlit as st

iris = joblib.load("iris.pkl")

max\_val = joblib.load("max.pkl")

min\_val = joblib.load("min.pkl")

st.title("Iris Classifier")

left, center, right = st.beta\_columns([5,1,5])

# En realidad no usaremos center, sólo está para crear un espacio entre las columnas

pred = [

left.slider("Longitud del sépalo", min\_val[0], max\_val[0]),

right.number\_input("Anchura del sépalo", min\_val[1], max\_val[1], step = 0.3),

left.slider("Longitud del pétalo", min\_val[2], max\_val[2]),

right.number\_input("Anchura del pétalo", min\_val[3], max\_val[3], step = 0.3)

]

if st.button("Predecir"):

val = iris.predict([pred])

if val == 0:

name = "Setosa :hibiscus:"

elif val == 1:

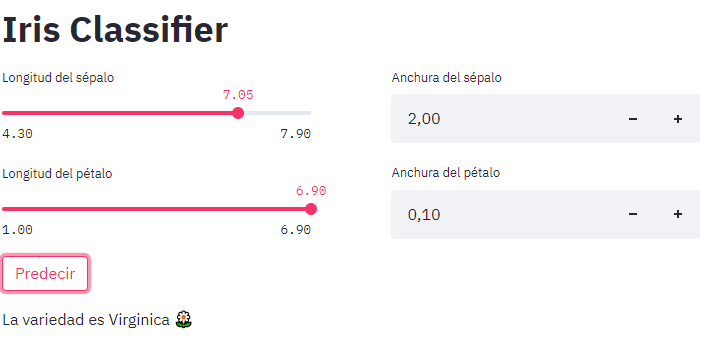
name = "Versicolor :tulip:"

else:

name = "Virginica :blossom:"

st.markdown("La variedad es "+ name)

Con esto ya tienes una pequeña muestra de lo que se puede hacer con Streamlit, generando con un par de líneas de código una web muy intuitiva para cualquier usuario.



### Streamlit y el tratamiento de imágenes

En este ejemplo veréis que con dos líneas de código podemos hacer que Streamlit permita a los usuarios subir imágenes y se las muestre, entre tanto, por supuesto, podemos tratarlas con otras librerías o métodos propios:

import streamlit as st

from PIL import Image

from skimage import filters

# Permitimos al usuario subir un archivo JPG o NPG.

pic = st.file\_uploader("Introduce tu imagen", ["JPG", "PNG"])

# Si lo hace:

if pic:

# Cambiamos su formato a un formato de imagen más convencional, lo que no es necesario para trabajar con Streamlit pero sí con otras librerías.

pic = Image.open(pic)

# Tratamos la imagen.

pic = filters.sobel(pic)

# Y se la mostramos al usuario con el tamaño que el escoja.

size = st.slider("Tamaño de la imagen", min\_value = 100, max\_value = 300, step = 10)

st.image(pic, width = size)



### Gráficos y mapas:

import streamlit as st

import pandas as pd

show = st.beta\_container()

with st.sidebar:

opt = st.radio("Opción", ["Gráfico", "Mapa"])

if opt == "Gráfico":

width = st.slider("Ancho", min\_value = 300, max\_value = 900, step = 10, value = 600)

height = st.slider("Alto", min\_value = 100, max\_value = 300, step = 10, value = 200)

gtype = st.selectbox("Tipo de grafico",

("area\_chart", "bar\_chart", "line\_chart"),

# Recuerda, el parámetro format\_func sirve para diferenciar lo que queremos recibir

# (area\_char, bar\_chart...) de lo que queremos mostrarle al usuario (Gráfico de barras...).

format\_func = lambda t : "Area" if t == "area\_chart" else "Barras" if t == "bar\_chart" else "Lineas"

)

show.title("Gráficos simples")

show.write("Como ves no es difícil preparar un entorno con gráficos generados dinámicamente a través de streamlit:")

# Datos aleatorios.

data = pd.DataFrame([[1,3,2],[3,2,2],[4,5,2]], columns = ["C1","C2","C3"])

# La función getattr permite llamar a una función a través de su librería y su nombre en forma de string.

getattr(show, gtype) (data, width = width, height = height, use\_container\_width = False)

# Recuerda que los contenedores en streamlit se usan como llamadas a la propia librería, así que en la función

# anterior show es equivalente a st salvo por la ubicación en la que aparecen los resultados.

else:

show.title("Donde estamos")

show.write("Tampoco es difícil generar un mapa, aquí puedes ver la mayoría de las ciudades en las puedes encontrar una comunidad Saturdays:")

# Estas son las coordenadas de las ciudades.

locations = pd.DataFrame([

[41.3879, 2.16992],

[20.6736, -103.344],

[40.4167, -3.70325],

[-2.19616, -79.88621],

[37.38283, -5.97317],

[39.46975, -0.37739],

[-0.225219, -78.5248],

[25.6714, -100.309],

[25.2646, 55.3077],

[9.748917, -83.753428],

[38.3452, -0.481006],

[36.83814, -2.45974],

[43.36029, -5.84476],

[43.26271, -2.92528],

[37.89155, -4.77275],

[51.5072, -0.1275],

[37.98704, -1.13004],

[39.5695100, 2.6474500],

[41.119, 1.24546],

[39.46975, -0.37739],

[19.4194, -99.1455],

[-17.414, -66.1653],

[-27.33056, -55.86667],

[29.0892, -110.961],

[-16.4897, -68.1193],

[-12.0453, -77.0311],

[-0.225219, -78.5248],

[32.22174, -110.92648],

[-20.348404, 57.552152]

],

columns = ["lat", "lon"])

# Mostrar coordenadas en el mapa.

show.map(locations)



