

••• Proyecto Sobrevuelo

Fabian Simbaña, Jose Andino, Luis Perez, Kevin Paillacho

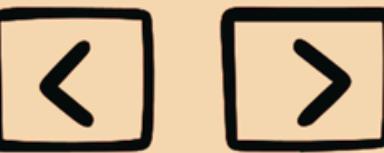


ooo

Un avión vuela con velocidad V a Ha metros sobre el nivel del mar en dirección a una montaña.

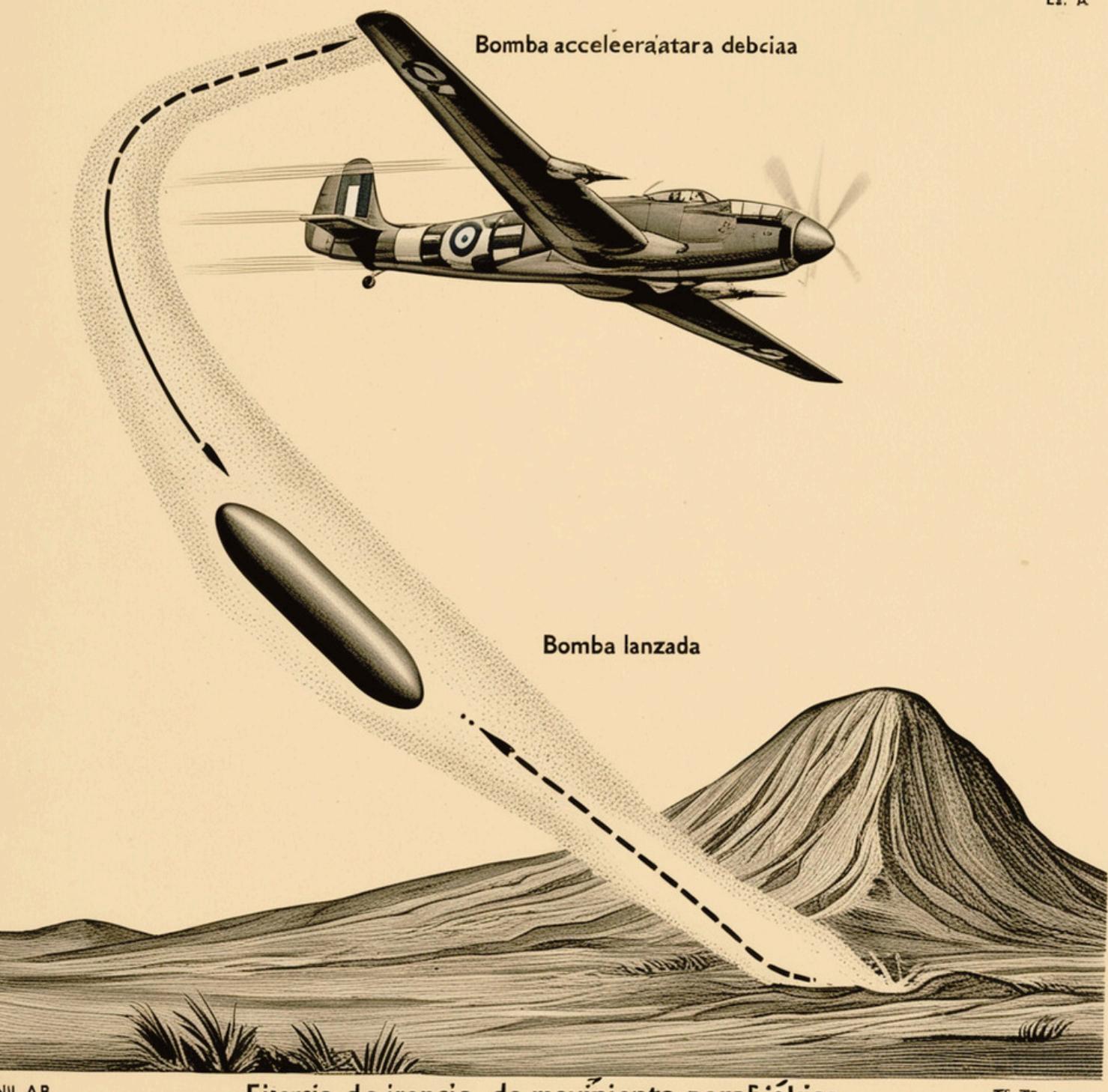
La montaña tiene forma de triángulo isósceles, con una altura Hm y un ángulo α con respecto al horizonte.

Cuando se encuentra a una distancia d del pico de la montaña, el avión suelta una bomba.



Bombela des
d. A.

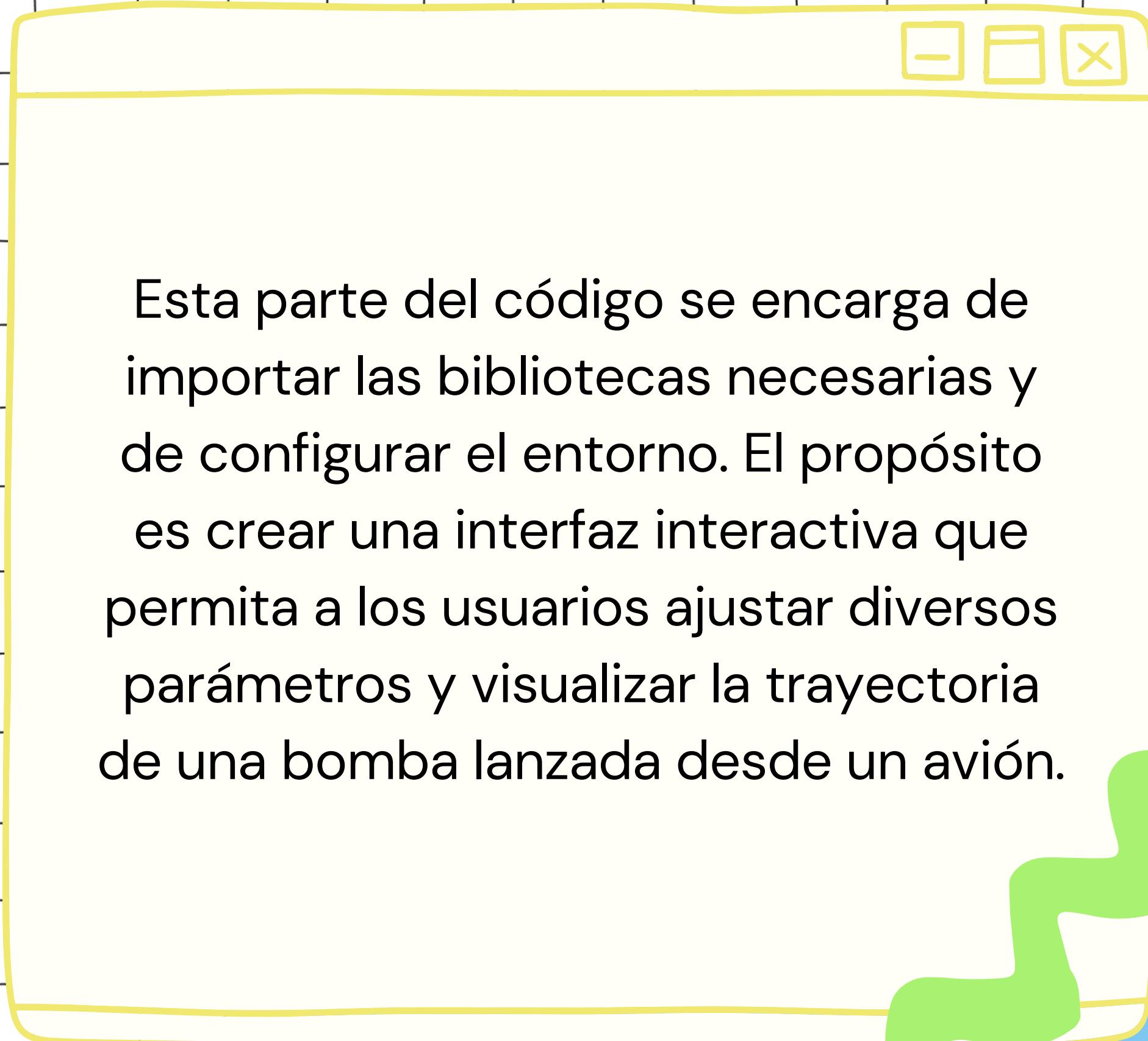
Bomba acceleeraatara debciaa



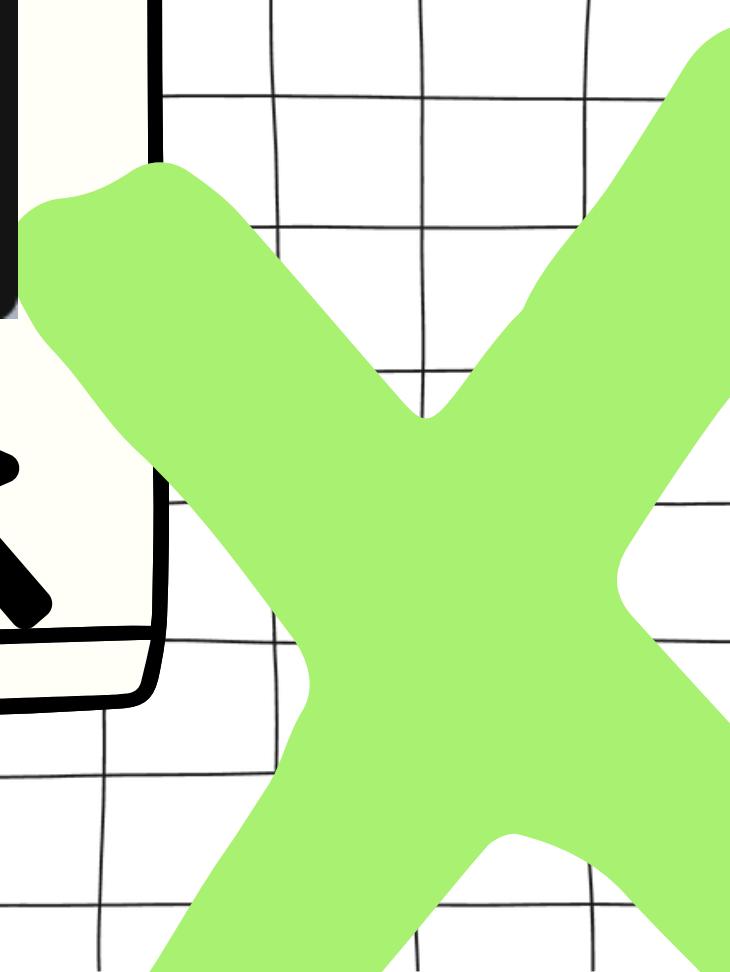
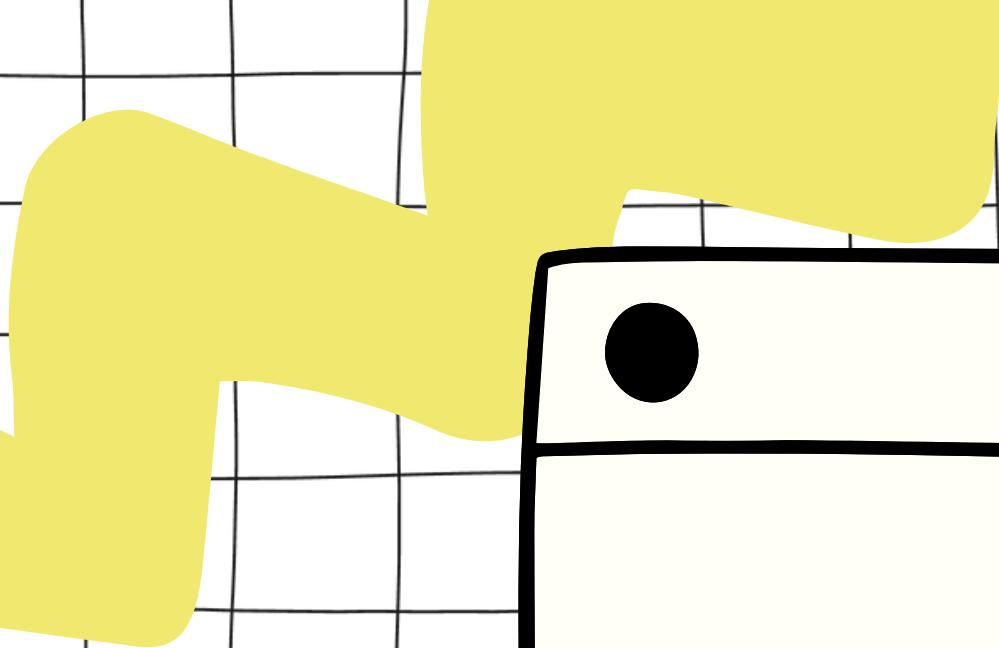
Ejercicio de irección de movimiento parabólico

Tf, Tf, Apga.

NIL A.B.



Esta parte del código se encarga de importar las bibliotecas necesarias y de configurar el entorno. El propósito es crear una interfaz interactiva que permita a los usuarios ajustar diversos parámetros y visualizar la trayectoria de una bomba lanzada desde un avión.



```
import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as widgets
from IPython.display import display, clear_output
```

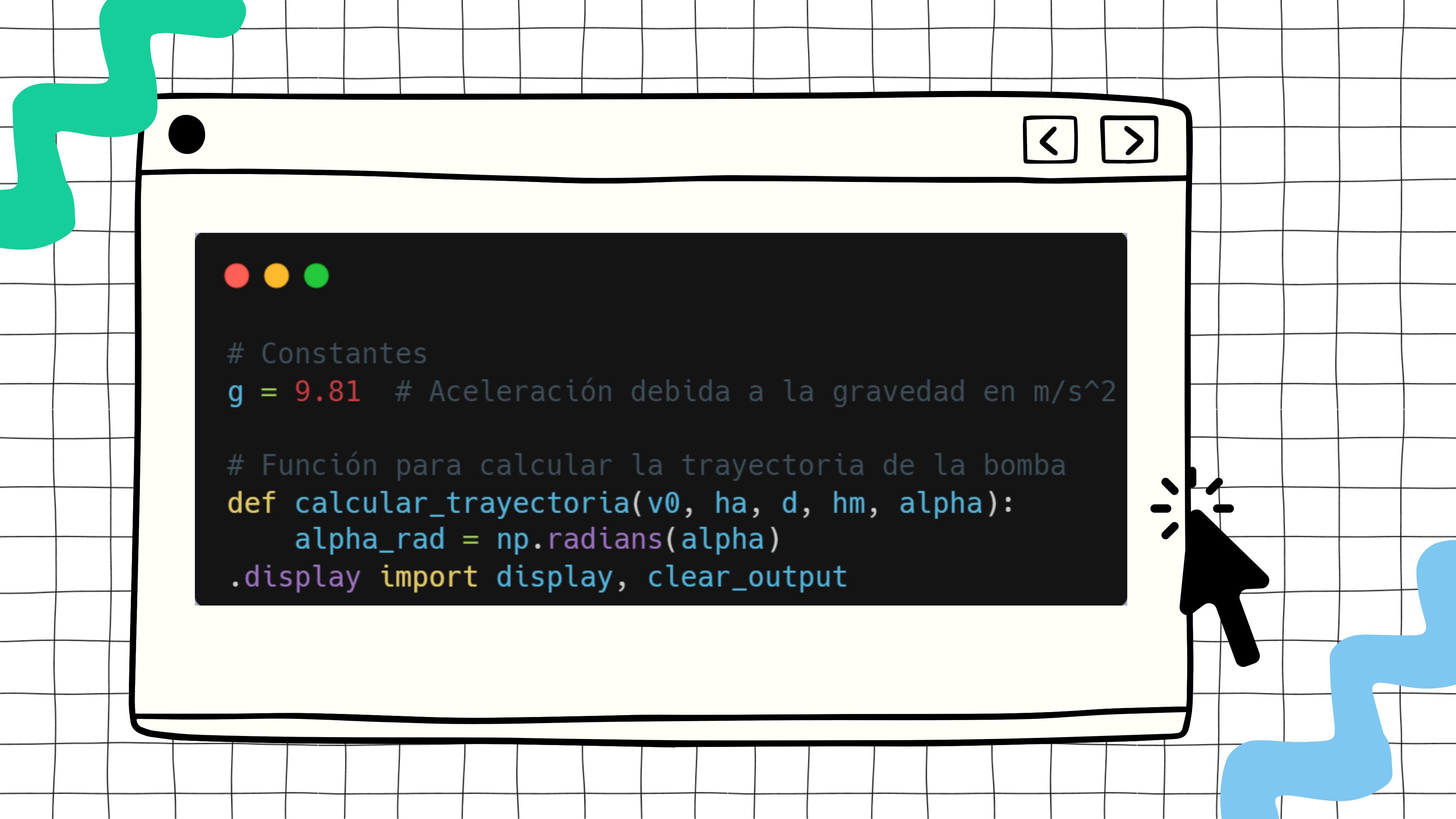


Función para calcular la trayectoria de la bomba

Esta parte del código define una función llamada `calcular_trayectoria` que se utiliza para calcular la trayectoria de una bomba lanzada desde un avión, y convierte el ángulo de lanzamiento de grados a radianes.

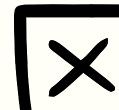
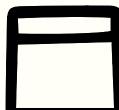
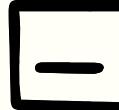
Parámetros:

- v_0 : Velocidad inicial del lanzamiento (en m/s).
- h_a : Altura inicial del avión (en metros).
- d : Distancia horizontal al pico de la montaña (en metros).
- h_m : Altura de la montaña (en metros).
- α : Ángulo de lanzamiento (en grados).



```
# Constantes
g = 9.81 # Aceleración debida a la gravedad en m/s^2

# Función para calcular la trayectoria de la bomba
def calcular_trayectoria(v0, ha, d, hm, alpha):
    alpha_rad = np.radians(alpha)
    .display import display, clear_output
```



Tiempo de impacto con el eje x

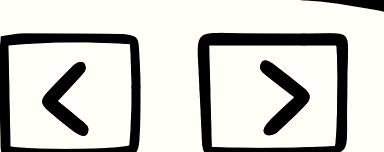
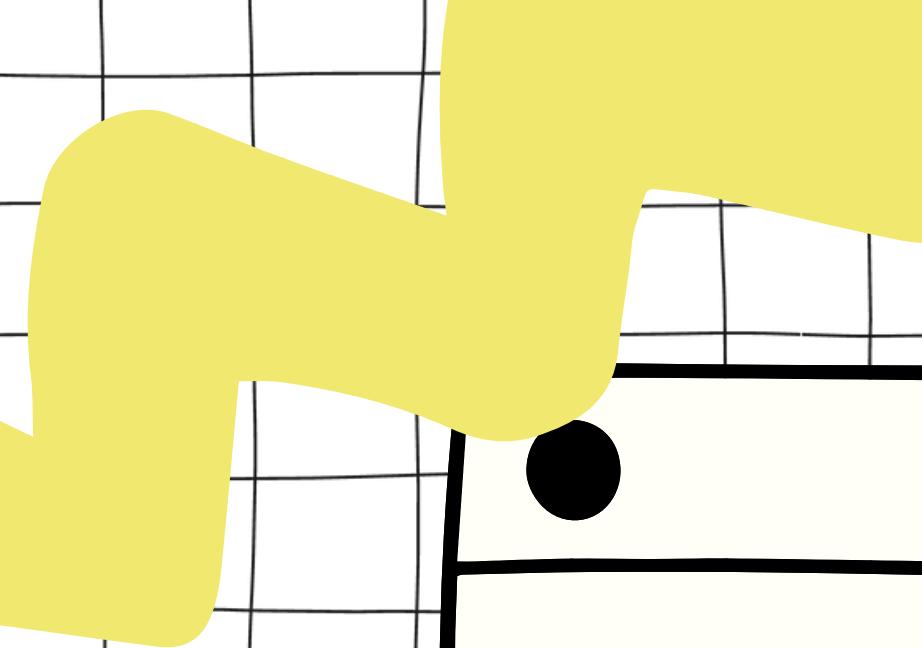
Esta parte del código calcula el tiempo y la distancia horizontales en el punto en que la bomba impacta el suelo en ausencia de cualquier obstáculo (es decir, sin considerar la montaña).

Tiempo de Impacto con el Eje X.

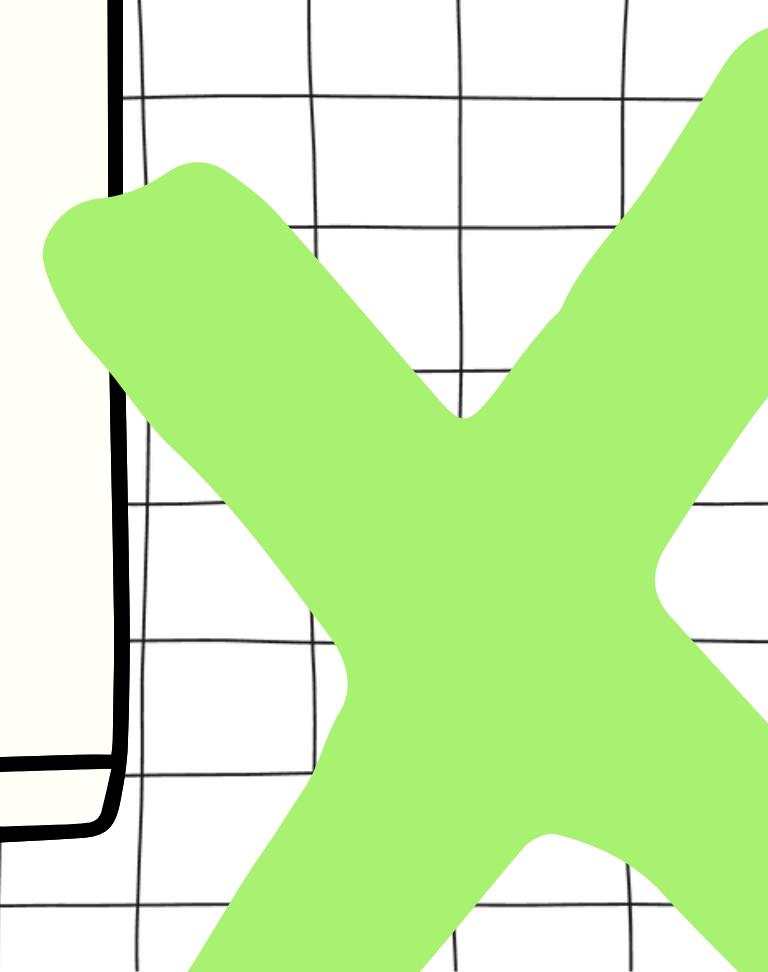
- np.sqrt es una función de numpy que calcula la raíz cuadrada.
- ha es la altura inicial del avión (en metros).
- g es la aceleración debida a la gravedad (9.81 m/s^2).

Distancia Horizontal en el Impacto.

- v0 es la velocidad inicial horizontal del avión (en m/s).
- t_impacto_x es el tiempo que tarda la bomba en caer al suelo.



```
# Tiempo de impacto con el eje x
t_impacto_x = np.sqrt(2 * ha / g)
x_impacto_x = v0 * t_impacto_x
```





Coordenadas de la Montaña.

Coordenadas en el Eje X.

- d: Distancia horizontal desde el punto de lanzamiento (donde está el avión) hasta el pico de la montaña.
- hm: Altura de la montaña.
- alpha_rad: Ángulo de la pendiente de la montaña en radianes.

Coordenadas en el Eje Y.

- O: Altura del punto donde la pendiente izquierda de la montaña toca el suelo.
- hm: Altura del pico de la montaña.
- O: Altura del punto donde la pendiente derecha de la montaña toca el suelo.



```
# Coordenadas de la montaña
montaña_x = [d - hm / np.tan(alpha_rad), d, d + hm / np.tan(alpha_rad)]
montaña_y = [0, hm, 0]
```



Función lineal de la montaña

La función montaña(x) define la altura de la montaña en cualquier punto x a lo largo del eje horizontal. Esta función utiliza las coordenadas montaña_x calculadas anteriormente para determinar si el punto x está en la pendiente izquierda, en el pico, o en la pendiente derecha de la montaña, y calcula la altura correspondiente.

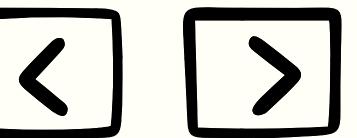
```
● ● ●  
# Función lineal de la montaña  
def montaña(x):  
    if montaña_x[0] <= x <= montaña_x[1]:  
        return (hm / (d - montaña_x[0])) * (x - montaña_x[0])  
    elif montaña_x[1] < x <= montaña_x[2]:  
        return hm - (hm / (montaña_x[2] - d)) * (x - d)  
    else:  
        return -1 # Valor que nunca se alcanzará si está fuera de los límites de la montaña
```



Encontrar la intersección con la montaña

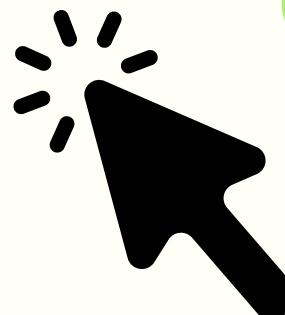
Esta parte del código calcula la trayectoria de la bomba y verifica si en algún punto de esa trayectoria la bomba intersecta la montaña.

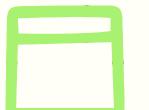
- Se generan 500 instantes de tiempo desde el lanzamiento hasta el impacto teórico.
- Se calculan las coordenadas horizontales (x) y verticales (y) de la bomba para cada uno de esos instantes.
- Se verifica si, en algún instante, la bomba intersecta la montaña (su altura es menor o igual a la altura de la montaña en esa coordenada x).
- Si se encuentra una intersección, se actualizan las variables `interseccion_x` y `interseccion_y` y se termina el bucle.



```
# Encontrar la intersección con la montaña
t_vals = np.linspace(0, t_impacto_x, num=500)
x_vals = v0 * t_vals
y_vals = ha - 0.5 * g * t_vals**2

interseccion_x = x_impacto_x
interseccion_y = 0
for t, x, y in zip(t_vals, x_vals, y_vals):
    if y <= montaña(x):
        interseccion_x = x
        interseccion_y = y
        break
```





Ajustar la trayectoria hasta la intersección

Esta parte del código ajusta la trayectoria calculada de la bomba para que termine en el punto de intersección con la montaña, si tal intersección existe.

- Se ajusta el tiempo de impacto ($t_{impacto}$) para que coincida con el punto de intersección con la montaña, en lugar del tiempo teórico de impacto sin obstáculos.
- Se generan nuevas coordenadas de la trayectoria (x y y) hasta el punto de intersección real con la montaña.
- Se devuelve la trayectoria ajustada junto con las coordenadas de intersección y las coordenadas de la montaña.

```
● ● ●
```

```
# Ajustar la trayectoria hasta la intersección
t_impacto = t_vals[np.where(x_vals == interseccion_x)][0]
t = np.linspace(0, t_impacto, num=500)
x = v0 * t
y = ha - 0.5 * g * t**2

return x, y, interseccion_x, interseccion_y, montaña_x, montaña_y
```

Función para graficar la montaña y la trayectoria de la bomba

La función graficar_trayectoria se encarga de visualizar gráficamente la trayectoria de una bomba y su intersección con una montaña.

Funcionamiento General

- Calculo de Trayectoria: Utiliza la función calcular_trayectoria para obtener las coordenadas de la trayectoria de la bomba y el punto de intersección con la montaña.
- Gráfico: Dibuja la montaña, la trayectoria de la bomba y el punto de intersección en un solo gráfico utilizando matplotlib.

```
● ● ●  
# Función para graficar la montaña y la trayectoria de la bomba  
def graficar_trayectoria(v0, ha, d, hm, alpha):  
    x, y, interseccion_x, interseccion_y, montaña_x, montaña_y = calcular_trayectoria(v0, ha, d, hm, alpha)  
  
    plt.figure(figsize=(10, 6))  
    plt.plot(montaña_x, montaña_y, label='Montaña')  
    plt.plot(x, y, label='Trayectoria de la bomba')  
    plt.scatter([interseccion_x], [interseccion_y], color='red', zorder=5, label='Punto de intersección')  
    plt.xlabel('Distancia horizontal (m)')  
    plt.ylabel('Altura (m)')  
    plt.legend()  
    plt.grid()  
    plt.show()
```



Crear sliders para los parámetros

Estos sliders permiten ajustar dinámicamente los parámetros de entrada (v_0 , ha , d , hm , α) que se utilizan para calcular y graficar la trayectoria de una bomba y su intersección con una montaña. Cuando se mueve alguno de los sliders, se actualiza automáticamente la gráfica correspondiente con la nueva configuración de parámetros, mostrando cómo cambia la trayectoria de la bomba en función de esos ajustes.

En conjunto con la función `graficar_trayectoria` que mencionamos anteriormente, estos sliders permiten una exploración interactiva de cómo diferentes valores de velocidad inicial, altura de lanzamiento, distancia y altura de la montaña, y ángulo de inclinación afectan la trayectoria y el punto de impacto de la bomba.



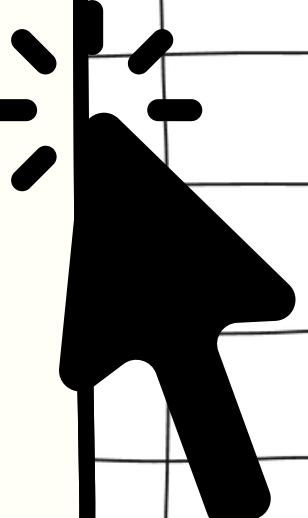
```
# Crear sliders para los parámetros
v0_slider = widgets.FloatSlider(value=100, min=10, max=500, step=10, description='Vel. avión (m/s):')
ha_slider = widgets.FloatSlider(value=80, min=10, max=500, step=10, description='Alt. avión (m):')
d_slider = widgets.FloatSlider(value=50, min=10, max=500, step=10, description='Dist al pico (m):')
hm_slider = widgets.FloatSlider(value=75, min=10, max=500, step=10, description='Alt. montaña (m):')
alpha_slider = widgets.FloatSlider(value=60, min=1, max=89, step=1, description='Ángulo (grados):')
```



Función que se llama cuando se actualiza cualquier slider

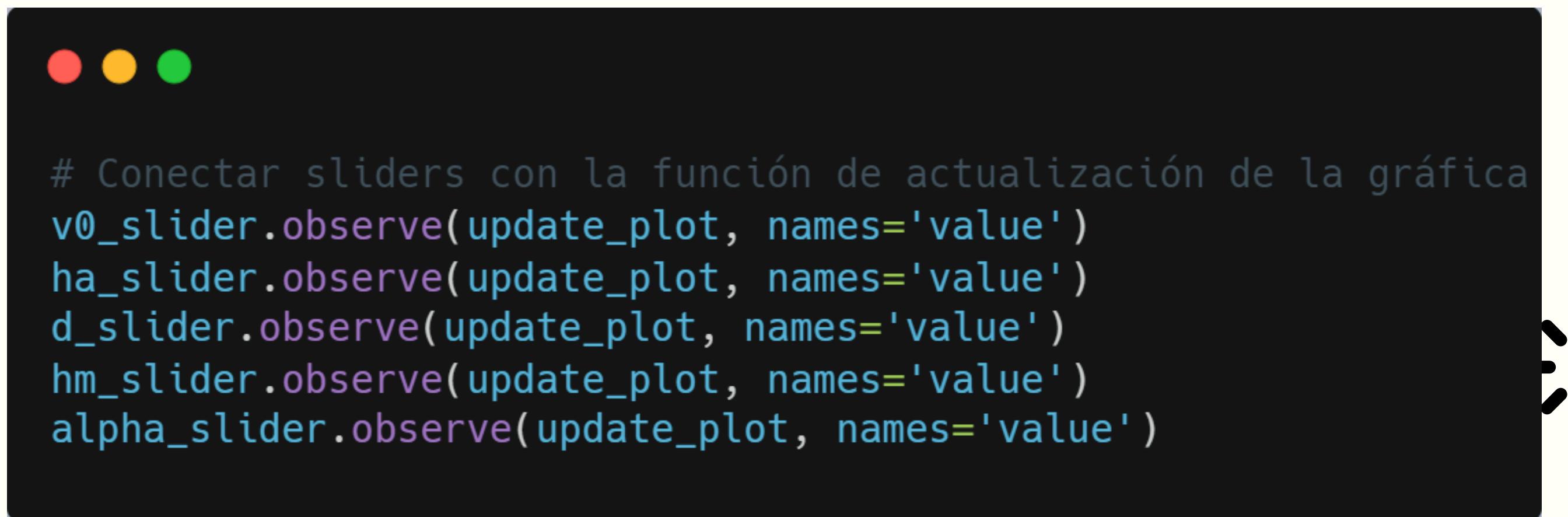
Esta parte del código define una función `update_plot(change)` que se utiliza para actualizar el gráfico interactivo cada vez que se cambia el valor de alguno de los sliders (`v0_slider`, `ha_slider`, `d_slider`, `hm_slider`, `alpha_slider`).

```
# Función que se llama cuando se actualiza cualquier slider
def update_plot(change):
    v0 = v0_slider.value
    ha = ha_slider.value
    d = d_slider.value
    hm = hm_slider.value
    alpha = alpha_slider.value
    clear_output(wait=True)
    display(v0_slider, ha_slider, d_slider, hm_slider, alpha_slider)
    graficar_trayectoria(v0, ha, d, hm, alpha)
```

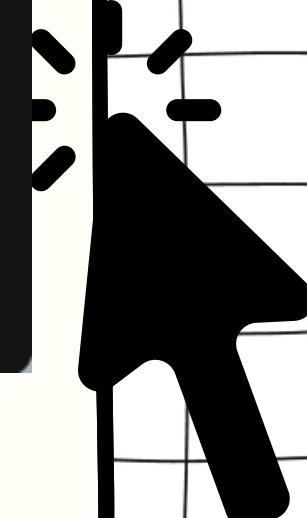


Conectar sliders con la función de actualización de la gráfica

Esta parte del código establece observadores para cada uno de los sliders (`v0_slider`, `ha_slider`, `d_slider`, `hm_slider`, `alpha_slider`) que se han definido previamente. Estos observadores están configurados para llamar a la función `update_plot` cada vez que el valor de cualquiera de estos sliders cambie.



```
# Conectar sliders con la función de actualización de la gráfica
v0_slider.observe(update_plot, names='value')
ha_slider.observe(update_plot, names='value')
d_slider.observe(update_plot, names='value')
hm_slider.observe(update_plot, names='value')
alpha_slider.observe(update_plot, names='value')
```



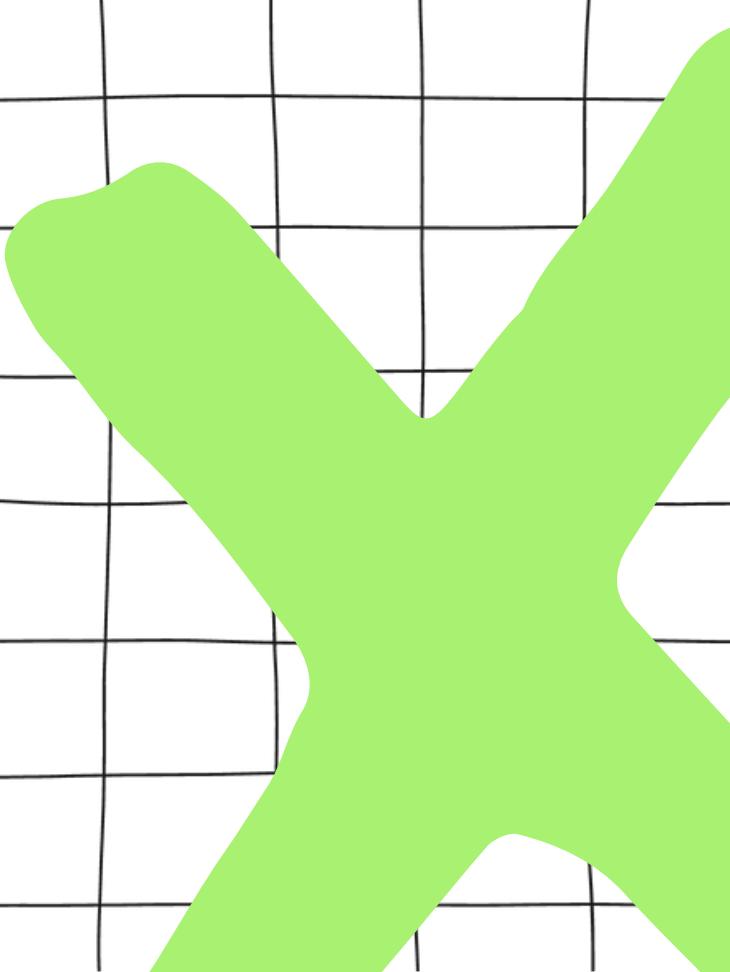
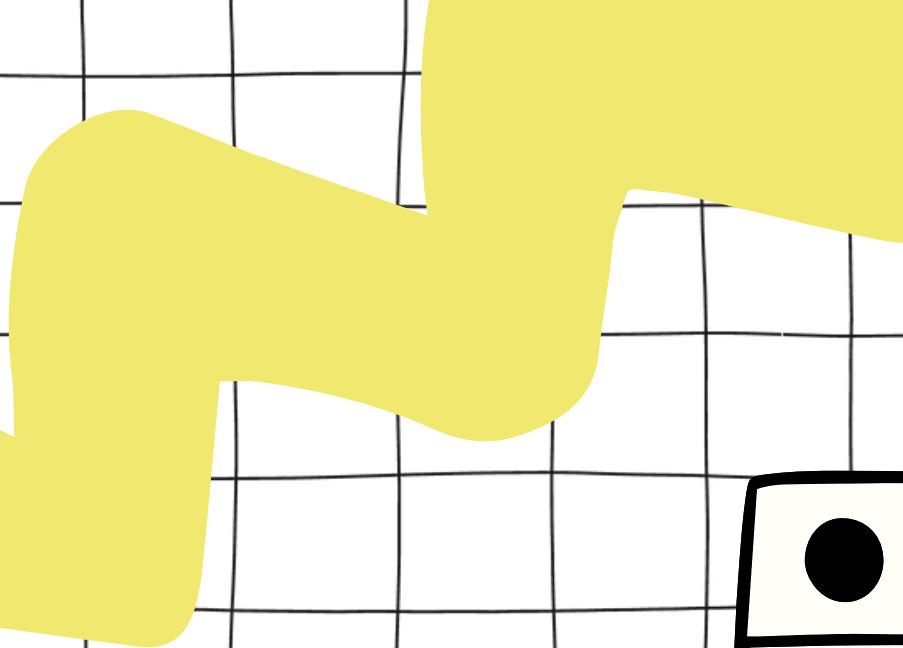
Mostrar sliders y la gráfica inicial

Esta parte del código tiene como objetivo mostrar inicialmente los sliders y el gráfico de la trayectoria de la bomba con los valores iniciales de los parámetros definidos por los sliders.

• • •



```
# Mostrar sliders y la gráfica inicial  
display(v0_slider, ha_slider, d_slider, hm_slider, alpha_slider)  
update_plot(None)
```



Thank you

www.reallygreatsite.com

