# 📌 LECCIÓN 7: APLICACIONES SENCILLAS CON PYTHON

**🎯 Objetivo:**

Construir un juego sencillo en Python usando únicamente la biblioteca estándar, pasando de pseudocódigo a diagrama de flujo (Draw.io) y a una implementación incremental en VS Code con puntos de control; reforzando validación de entradas, uso de listas/sets, manejo de cadenas, condicionales y bucles, como cierre integrador de lo visto en las guías anteriores.

**🧩 Situación problema**

Vas a diseñar un juego por consola donde una persona intenta **adivinar una palabra secreta** letra por letra. El programa mostrará el avance con guiones en las letras no descubiertas, llevará registro de letras ya usadas y tendrá **6 intentos** como máximo. No usaremos librerías externas.

**💪 Primer reto: estructurar una idea general de lo que vamos a hacer con ayuda de pseudocódigo.**

**Requisitos**

- El programa **elige** una palabra secreta de una lista interna.
- En cada turno el jugador **ingresa una sola letra** (a–z).
- Si la letra **ya se usó**, se muestra un mensaje y el turno avanza sin descontar intento.
- Si la letra **está** en la palabra, se **revela en todas sus posiciones**.
- Si la letra **no está**, se **resta 1 intento**.
- En cada turno se **muestra**: el progreso (guiones/letras), los **intentos restantes** y las **letras usadas**.
- El juego **termina** cuando:
  - el jugador **descubre toda** la palabra (gana), o
  - se **agotan los 6 intentos** (pierde).

Ahora, con base en estas reglas escribe tu propio pseudocódigo siguiendo estos requisitos. Recuerda mantener el lenguaje sencillo y un orden lógico en las acciones. Toma como referencia el algoritmo para jugar al ahorcado en una hoja de papel, omite utilizar el dibujo del ahorcado ya que utilizaras los intentos para marcar el final del juego.

**Ejemplo de respuesta (pseudocódigo)**

1) INICIO
2) Dar bienvenida al usuario.
3) Elegir una palabra secreta de una lista.
4) Mostrar la palabra como guiones (una marca por cada letra).

5) Definir 6 intentos disponibles.
6) Mostrar intentos restantes y letras usadas (vacías al inicio).
7) Mientras queden intentos y la palabra no esté completa:
8) Pedir al usuario una letra (en minúscula).
9) Si la letra ya se usó:
    - Mostrar mensaje de “letra repetida”.
    - Pasar al siguiente turno.
10) Si la letra está en la palabra:
    - Revelar esa letra en todas sus posiciones.
11) Si la letra no está en la palabra:
    - Restar 1 intento.
    - Mostrar mensaje de “letra incorrecta”.
12) Mostrar de nuevo el avance de la palabra, intentos restantes y letras usadas.
13) Si la palabra ya quedó completa:
    - Mostrar mensaje de victoria y terminar.
14) Si los intentos llegan a 0 y la palabra no se completó:
    - Mostrar mensaje de derrota con la palabra real.
15) FIN

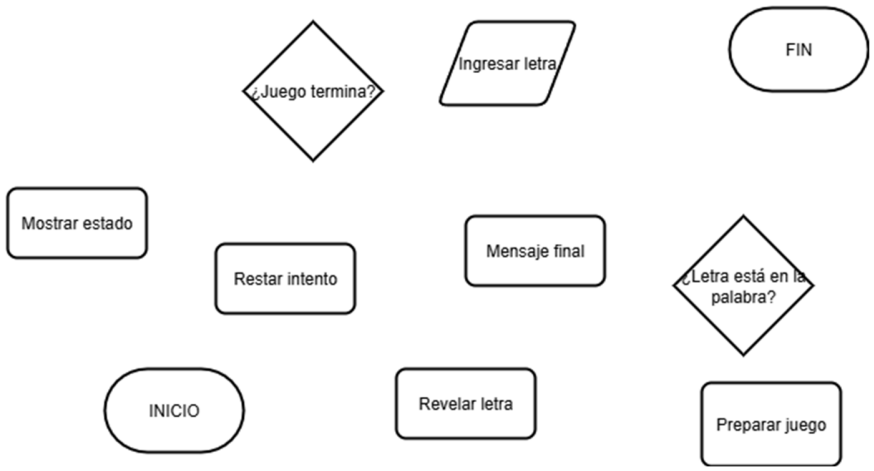
¡Muy bien! Ya tenemos la idea general del juego con el pseudocódigo. Ahora usaremos esa información para crear un diagrama de flujo que nos ayude a estructurar el código en Python.

---

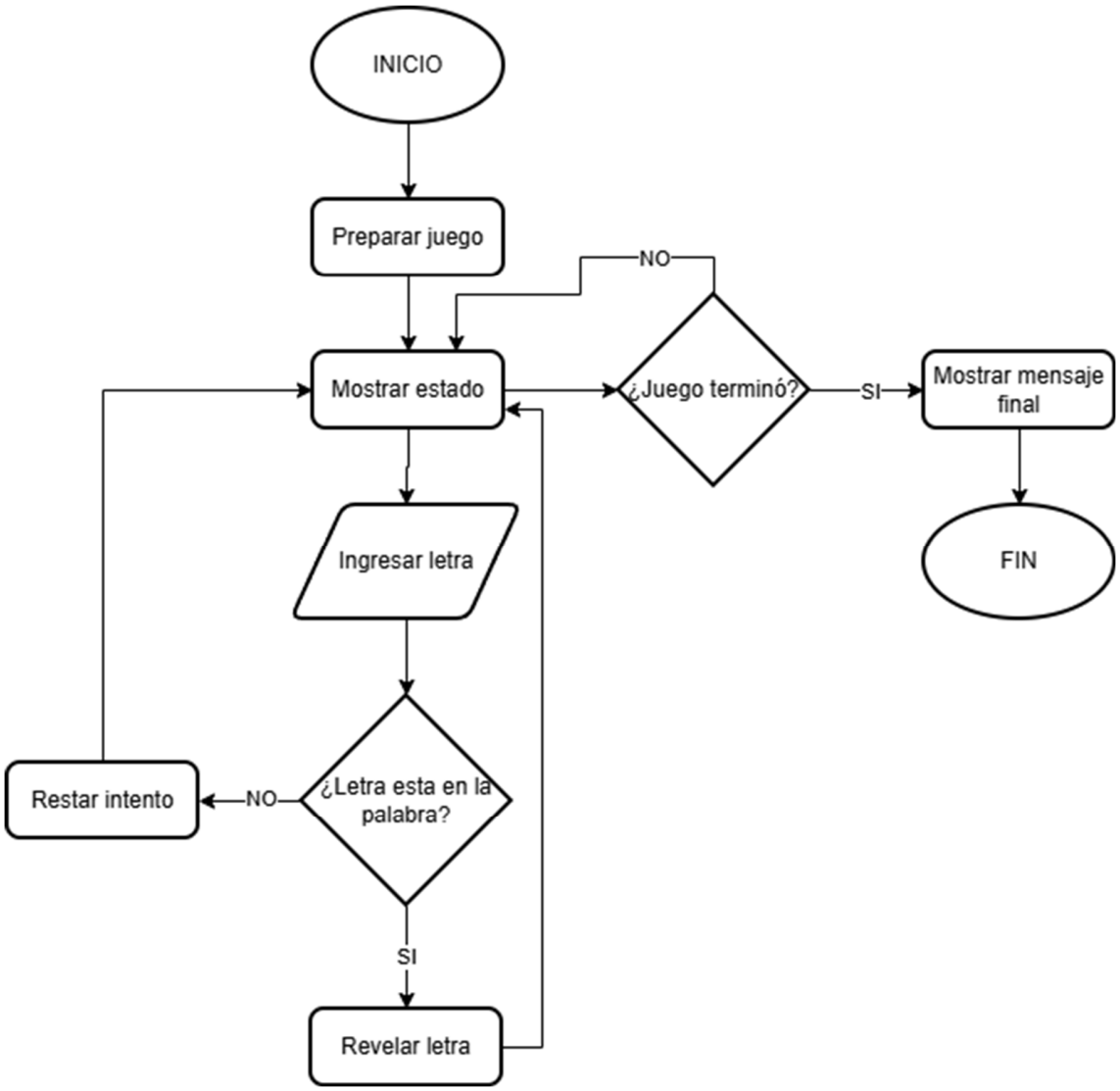
💪 **Segundo reto: realizar un diagrama de flujo partiendo del pseudocódigo.**

Utilizando como base el pseudocódigo del primer reto, ahora realiza un diagrama de flujo en Draw.io. Asegúrate de que la lógica se pueda entender claramente y evita utilizar pasos excesivos.

Si aun tienes problemas para visualizar correctamente lo que debes hacer, tal vez estas piezas de información puedan ayudarte:



Ahora utiliza flechas para unir de manera lógica las piezas del diagrama de flujo. Una posible respuesta al segundo reto se muestra a continuación:




Con un camino claro a seguir como el que nos muestra el diagrama de flujo, podemos comenzar a programar paso a paso el código de nuestro juego en Python. ¡Manos a la obra!

---

Antes de iniciar con nuestro siguiente reto, vamos a generar una función que nos permitirá modularizar nuestro juego. Crea un nuevo archivo llamado ahorcado.py y copia el siguiente fragmento de código:

```
def main():
    print("CHECKPOINT 1: esqueleto listo")
    # aquí luego irán: preparar juego, mostrar estado, ingresar letra, etc.
```

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

Si ves CHECKPOINT 1 en consola, nuestro paso inicial estará listo y podremos continuar. Comenta el **print** del checkpoint con CTRL + K → CTRL + C para inhabilitarlo.

---

💪 **Tercer reto: preparar el juego y mostrar el estado inicial.**

Tus instrucciones son las siguientes:

1. Utiliza **import random** en la línea 1 para iniciar tu código.

2. Genera una ***función*** llamada *seleccionar_palabra_aleatoria* que contenga una ***lista*** de palabras, utiliza ***return*** para devolver aleatoriamente alguna de estas palabras con ***random.choice.***

3. Genera una ***función*** llamada *mostrar_estado* que muestre el estado, sus parámetros serán *palabra* y las *letras_acertadas*. Debajo de esta función añadiremos el siguiente fragmento de código que permitirá identificar el estado de nuestro juego (asegúrate de dejar un espacio en las comillas antes del join):

   ```
   estado = " ".join([letra if letra in letras_acertadas else "_" for letra in palabra])
   print("Palabra:", estado)
   ```

¿Qué es lo que crees que este fragmento de código hace?

4. Dentro de la ***función*** main() crea las siguientes variables:

| | | |
|---|---|---|
| palabra | → | seleccionar_palabra_aleatoria() |
| intentos | → | 6 |
| letras_usadas | → | [] |
| letras_acertadas | → | [] |

¿Por qué se dejan paréntesis y corchetes vacíos?

5. Cierra la **función *main()*** con *mostrar_estado* y los parámetros *palabra* y *letras_acertadas*. Genera un **print** que muestre los *intentos* y las *letras_usadas*.

6. Genera una variable *letra* al final de la función *main()*, con la indentación adecuada y donde pidas al usuario que ingrese una letra. Al final debe quedar el fragmento *if __name__ == "__main__":* por fuera de la función *main()*.



**Ejemplo de respuesta (Python)**

```
import random

def seleccionar_palabra_aleatoria():
    palabras = ["python", "programa", "ahorcado", "juego", "logica"]
    return random.choice(palabras)

def mostrar_estado(palabra, letras_acertadas):
    estado = " ".join([letra if letra in letras_acertadas else "_" for letra in palabra])
    print("Palabra:", estado)

def main():
    palabra = seleccionar_palabra_aleatoria()
    intentos = 6
    letras_usadas = []
    letras_acertadas = []

    mostrar_estado(palabra, letras_acertadas)
    print("Intentos:", intentos, "| Usadas:", letras_usadas)

    letra = input("Ingresa una letra: ").strip.lower()

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

Si al ejecutar el código, te solicita “Ingresar una letra” y verifica el estado del juego, habrás terminado esta parte del reto.

---

Perfecto, ahora sí construyamos el 💪 **Cuarto reto** con la misma estructura que usaste en el Tercer reto: orientaciones paso a paso, preguntas intercaladas y ejemplo de respuesta al final.

---

💪 **Cuarto reto: construir el bucle del juego**

**Tus instrucciones son las siguientes:**

1. Dentro de la función main(), **debajo** de mostrar_estado(...), del print("Intentos...") y de la letra = ..., escribe la línea que dará inicio al bucle:

```
while intentos > 0 and not all(letra in letras_acertadas for letra in palabra):
```

¿Qué crees que significa esta condición?

---

2. Dentro del bucle, pide una letra al usuario:

```
letra = input("Ingresa una letra: ").strip().lower()
```

¿Por qué crees que usamos .strip().lower() aquí?

---

3. Verifica si la letra ya se usó.

   - Si sí, muestra el mensaje **"Letra repetida"** y usa ***continue*** para pasar al siguiente turno.
   - Si no, añade la letra a la lista letras_usadas.

¿Qué pasaría si no usáramos continue en este caso?

---

4. Comprueba si la letra está en la palabra.

   - Si está, agrégala a letras_acertadas y muestra **"¡Acierto!"**.
   - Si no está, resta 1 a intentos y muestra **"Incorrecta. Intentos: ..."**.

---

5. Al final de cada turno, vuelve a mostrar el estado de la palabra con mostrar_estado(...) y la lista de letras usadas.

¿Cómo se verá la palabra si tiene letras repetidas, como en "programa", cuando adivinas la "a"?

---

6. Cuando el bucle termine, agrega un bloque **después del while** que muestre el mensaje final:

   - Si se adivinó la palabra: **"¡Ganaste! La palabra era ..."**
   - Si se acabaron los intentos: **"Fin del juego. La palabra era ..."**

---

**Ejemplo de respuesta (Python)**

```
def main():
    palabra = seleccionar_palabra_aleatoria()
    intentos = 6
    letras_usadas = []
    letras_acertadas = []

    mostrar_estado(palabra, letras_acertadas)
    print("Intentos:", intentos, "| Usadas:", letras_usadas)

    while intentos > 0 and not all(letra in letras_acertadas for letra in palabra):
        letra = input("Ingresa una letra: ").strip().lower()

        if letra in letras_usadas:
            print("Letra repetida.")
            continue
        else:
```

```
        letras_usadas.append(letra)

    if letra in palabra:
        if letra not in letras_acertadas:
            letras_acertadas.append(letra)
        print("¡Acierto!")
    else:
        intentos -= 1
        print("Incorrecta. Intentos:", intentos)

    mostrar_estado(palabra, letras_acertadas)
    usadas = ", ".join(letras_usadas) if letras_usadas else "(ninguna)"
    print("Usadas:", usadas)

  if all(letra in letras_acertadas for letra in palabra):
    print("¡Ganaste! La palabra era:", palabra)
  else:
    print("Fin del juego. La palabra era:", palabra)
```

---

**Reto 5 (Opcional): validar la entrada del usuario**

**Objetivo:** asegurarnos de que el usuario ingrese **solo una letra** (a–z).

**Tus instrucciones son las siguientes:**

1. Crea una función llamada pedir_letra() (arriba de main()), que:
    - Pida una letra al usuario.
    - Quite espacios y convierta a minúscula.
    - Verifique que sea **una sola** letra (a–z).
    - Si no es válido, muestre un mensaje y vuelva a pedir.
2. En el bucle del juego (dentro de main()), **reemplaza** la línea donde pedías la letra por:
3. letra = pedir_letra()

**Preguntas:**

- ¿Qué ventaja tiene pedir y validar la letra en **una función aparte**?
- Si el usuario pulsa Enter vacío o escribe “33”, ¿qué debería pasar ahora?

**Ejemplo de respuesta (Python):**

```
def pedir_letra():
  while True:
    letra = input("Ingresa una letra: ").strip().lower()
    # Versión simple: restringimos a a–z
```



```
    if len(letra) == 1 and letra in "abcdefghijklmnopqrstuvwxyz":
        return letra
    print("Entrada inválida. Ingresa UNA sola letra (a–z).")

# Dentro del while de main(), reemplaza:
# letra = input("Ingresa una letra: ").strip().lower()
# por:
# letra = pedir_letra()
```

Nota para docentes: este reto introduce la idea de **validación**. Si prefieres, puedes dejarlo como lectura o demostración breve.

---

### 💡 Reto 6 (Opcional): pequeñas mejoras de usabilidad

**Objetivo:** hacer el juego más claro de leer, sin cambiar la lógica.

**Tus instrucciones son las siguientes:**

1. En mostrar_estado, muestra las letras separadas por un espacio para que se vea la longitud:
2. estado = " ".join([letra if letra in letras_acertadas else "_" for letra in palabra])
3. Al imprimir **letras usadas**, si la lista está vacía, muestra “(ninguna)” en lugar de [].
4. Ajusta los mensajes finales para que sean claros y bonitos.

**Preguntas:**

- ¿Por qué ayuda ver espacios entre guiones y letras?
- ¿Qué información mínima debe aparecer siempre en pantalla tras cada turno?

**Ejemplo de respuesta (Python):**

```
def mostrar_estado(palabra, letras_acertadas):
    estado = " ".join([letra if letra in letras_acertadas else "_" for letra in palabra])
    print("Palabra:", estado)

# Al final de cada turno, al mostrar usadas:
usadas = ", ".join(letras_usadas) if letras_usadas else "(ninguna)"
print("Usadas:", usadas)

# Mensajes finales (después del while en main)
if all(letra in letras_acertadas for letra in palabra):
    print("\n¡Ganaste! La palabra era:", palabra)
else:
    print("\nFin del juego. La palabra era:", palabra)
```

---

**¡Excelente trabajo!** Acabas de crear tu primera aplicación funcional con Python. ¡Felicitaciones! 🚀😊

---

🖊️ **Autor:** L. Nova
📅 **Fecha de creación:** 26 de enero de 2026