Clase 4 curso python para profesores 2026

El bucle `for`

¿Qué es un `for`?

Un **bucle `for`** sirve para **repetir código** recorriendo una colección de elementos (como una **lista o una tupla**) **uno por uno**.

Ejemplo:

```
numeros = [1, 2, 3, 4]

for n in numeros:
    print(n)
```

Qué ocurre:

- `n` toma el valor `1`, luego `2`, luego `3`, luego `4` que son los valores de la lista.
- El bloque indentado se ejecuta una vez por cada elemento

Estructura del `for`

```
for variable in coleccion:
    # código que se repite
```

- `variable`: recibe cada elemento
- `coleccion`: lista, tupla, string, `range`, etc.
- **La indentación es obligatoria**

`for` con listas

```
frutas = ["manzana", "pera", "plátano"]

for fruta in frutas:
    print(fruta)
```

Salida:

```
manzana
pera
plátano
```

`for` con tuplas

```
colores = ("rojo", "verde", "azul")

for color in colores:
    print(color)
```

`for` con strings

Un string es una colección de caracteres:

```
palabra = "Python"

for letra in palabra:
    print(letra)
```

`range()` y `for`

`range()` genera números.

```
for i in range(5):
    print(i)
```

Salida:

```
0
1
2
3
4
```

Variaciones:

```
range(inicio, fin)
range(inicio, fin, paso)
```

Ejemplo:

```
for i in range(1, 10, 2):
    print(i)
```

`for` + `if`

Podemos tomar decisiones dentro del bucle:

```
numeros = [1, 2, 3, 4, 5]

for n in numeros:
    if n % 2 == 0:
        print(n, "es par")
```

`enumerate()`

Sirve para obtener **índice y valor** al mismo tiempo.

```
frutas = ["manzana", "pera", "plátano"]

for indice, fruta in enumerate(frutas):
    print(indice, fruta)
```

Salida:

```
0 manzana
1 pera
2 plátano
```

`zip()`

Permite recorrer **varias listas a la vez**.

```
nombres = ["Ana", "Luis", "Marta"]
edades = [20, 25, 30]

for nombre, edad in zip(nombres, edades):
    print(nombre, edad)
```

Las listas pueden ser de distinto tamaño. En ese caso zip se limitará a unir los elementos de la lista más pequeña con el número de elementos igual a esta de la mayor. Realice el alumno la modificación pertinente en el ejemplo anterior.

`break` y `continue`

`break`: sale del bucle

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

`continue`: salta una iteración

```
for i in range(5):
    if i == 2:
        continue
    print(i)
```

20 EJERCICIOS CON SOLUCIÓN

---

1. Imprime los números del 1 al 10

```
for i in range(1, 11):
    print(i)
```

---

2. Imprime los elementos de una lista

```
lista = [3, 6, 9]
for n in lista:
    print(n)
```

---

3. Imprime solo números pares

```
for i in range(1, 11):
    if i % 2 == 0:
        print(i)
```

---

4. Suma todos los números de una lista

```
numeros = [1, 2, 3, 4]
suma = 0

for n in numeros:
    suma += n

print(suma)
```

---

5. Cuenta cuántos elementos tiene una lista

```
lista = [1, 2, 3]
contador = 0

for _ in lista:
    contador += 1

print(contador)
```

---

6. Imprime cada letra de un string

```
for letra in "Hola":
    print(letra)
```

---

7. Multiplica todos los números por 2

```
numeros = [1, 2, 3]
for n in numeros:
    print(n * 2)
```

---

8. Usa `enumerate`

```
colores = ["rojo", "verde"]
for i, color in enumerate(colores):
    print(i, color)
```

---

9. Usa `zip`

```
a = [1, 2]
b = [3, 4]

for x, y in zip(a, b):
    print(x + y)
```

---

10. Encuentra el número mayor

```
numeros = [3, 7, 2]
mayor = numeros[0]

for n in numeros:
    if n > mayor:
        mayor = n

print(mayor)
```

---

11. Imprime números del 10 al 1

```
for i in range(10, 0, -1):
    print(i)
```

---

12. Cuenta números pares

```
contador = 0
for i in range(10):
    if i % 2 == 0:
        contador += 1
```

```
print(contador)
```

---

13. Detén el bucle al encontrar un número

```
for i in range(10):
    if i == 6:
        break
    print(i)
```

---

14. Salta el número 3

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

---

15. Suma solo números positivos

```
nums = [-1, 2, -3, 4]
suma = 0

for n in nums:
    if n > 0:
        suma += n

print(suma)
```

---

16. Recorre una tupla

```
t = (10, 20, 30)
for x in t:
    print(x)
```

---

17. Muestra índice y valor

```
lista = ["a", "b", "c"]
for i, v in enumerate(lista):
    print(i, v)
```

---

18. Dos listas al mismo tiempo

```
nombres = ["Ana", "Luis"]
notas = [8, 9]

for n, nota in zip(nombres, notas):
    print(n, nota)
```

---

19. Imprime cuadrados

```
for i in range(1, 6):
    print(i ** 2)
```

---

20. Comprueba si hay un número mayor que 10

```
nums = [3, 5, 12]

for n in nums:
    if n > 10:
        print("Hay un número mayor que 10")
        break
```

`for` anidados

Son bucles dentro de otros bucles. Son muy útiles cuando trabajamos con **listas de listas**, **tablas**, **matrices**, o queremos combinar todas las posibilidades de dos colecciones.

Concepto

```
for variable1 in coleccion1:
    for variable2 in coleccion2:
        # Bloque que depende de ambas variables
        print(variable1, variable2)
```

- El **bucle externo** recorre `coleccion1`.
- Por cada elemento de `coleccion1`, el **bucle interno** recorre `coleccion2`.
- Así se ejecuta el bloque interno muchas veces: `len(coleccion1) * len(coleccion2)` veces.

Ejemplo básico

```
colores = ["rojo", "verde"]
formas = ["círculo", "cuadrado"]

for color in colores:
    for forma in formas:
        print(color, forma)
```

Salida:

```
rojo círculo
rojo cuadrado
verde círculo
verde cuadrado
```

Matriz o tabla

Supongamos que queremos imprimir una **tabla de 3x3**:

```
for fila in range(1, 4):
    for columna in range(1, 4):
        print(f"Fila {fila}, Columna {columna}")
```

Salida:

```
Fila 1, Columna 1
Fila 1, Columna 2
Fila 1, Columna 3
Fila 2, Columna 1
Fila 2, Columna 2
Fila 2, Columna 3
Fila 3, Columna 1
Fila 3, Columna 2
Fila 3, Columna 3
```

Lista de listas (matriz)

```
matriz = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

for fila in matriz:
    for valor in fila:
        print(valor, end=" ")
    print()  # Salto de línea
```

Salida:

```
1 2 3
4 5 6
7 8 9
```

Combinaciones de listas

```
letras = ["A", "B"]
numeros = [1, 2, 3]

for letra in letras:
    for numero in numeros:
        print(letra + str(numero))
```

Salida:

```
A1
A2
A3
B1
B2
B3
```

Ejemplo con condición

Supongamos que queremos **imprimir solo pares de números cuya suma sea par**:

```
for i in range(1, 4):
    for j in range(1, 4):
        if (i + j) % 2 == 0:
            print(i, j)
```

Salida:

```
1 1
1 3
2 2
3 1
3 3
```

¡Perfecto! Vamos a preparar **10 ejercicios de bucles `for` anidados** con sus soluciones paso a paso. Voy a empezar de fácil a intermedio, para que puedas **practicar matrices, combinaciones y lógica dentro de bucles anidados**.

Ejercicios con `for` anidados

1. Imprimir una tabla de multiplicar del 1 al 3

Ejercicio:
Imprime todas las multiplicaciones del 1 al 3:

```
1 x 1 = 1
1 x 2 = 2
...
3 x 3 = 9
```

Solución:

```
for i in range(1, 4):
    for j in range(1, 4):
        print(f"{i} x {j} = {i*j}")
```

---

2. Matriz de números

Ejercicio:
Imprime una matriz 3x3 usando listas de listas:

Solución:

```
matriz = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

for fila in matriz:
    for valor in fila:
        print(valor, end=" ")
```

```
    print()
```

3. Combinaciones de letras y números

Ejercicio:
Combina `["A", "B"]` con `[1, 2, 3]` y muéstralo como `"A1", "A2", ...`

Solución:

```
letras = ["A", "B"]
numeros = [1, 2, 3]

for letra in letras:
    for numero in numeros:
        print(letra + str(numero))
```

4. Imprimir un triángulo de asteriscos

Ejercicio:
Imprime un triángulo de 4 filas:

```
*
**
***
****
```

Solución:

```
for i in range(1, 5):
    for j in range(i):
        print("*", end="")
    print()
```

---

5. Suma de todos los elementos de una matriz

Ejercicio:
Calcula la suma de todos los números en la matriz:

```
matriz = [
    [1, 2],
    [3, 4]
]
```

Solución:

```
suma = 0
for fila in matriz:
    for valor in fila:
        suma += valor
print(suma)
```

---

6. Solo números pares en una matriz

Ejercicio:
Imprime solo los números pares de esta matriz:

```
matriz = [
    [1, 2, 3],
    [4, 5, 6]
]
```

Solución:

```
for fila in matriz:
    for valor in fila:
        if valor % 2 == 0:
            print(valor)
```

7. Contar combinaciones

Ejercicio:
Cuenta cuántas combinaciones se pueden formar con `["X", "Y"]` y `[1,2,3]`.

Solución:

```
letras = ["X", "Y"]
numeros = [1, 2, 3]
contador = 0

for letra in letras:
    for numero in numeros:
        contador += 1

print("Total combinaciones:", contador)
```

8. Matriz de multiplicaciones

Ejercicio:
Crea una tabla 3x3 de multiplicaciones en forma de **lista de listas**:

Solución:

```
tabla = []

for i in range(1, 4):
    fila = []
    for j in range(1, 4):
        fila.append(i*j)
    tabla.append(fila)

print(tabla)
```

9. Mostrar solo las diagonales de una matriz 3x3

Ejercicio:
Imprime solo los elementos donde el índice de fila y columna es igual.

Solución:

```
matriz = [
```

```
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

for i in range(len(matriz)):
    for j in range(len(matriz[i])):
        if i == j:
            print(matriz[i][j])
```

10. Imprimir todas las combinaciones posibles de dos listas

Ejercicio:
Dadas estas listas:

```
animales = ["perro", "gato"]
colores = ["negro", "blanco"]
```

Imprime todas las combinaciones: `"perro-negro", "gato-blanco", ..."`

Solución:

```
animales = ["perro", "gato"]
colores = ["negro", "blanco"]

for animal in animales:
    for color in colores:
        print(f"{animal}-{color}")
```

Vamos a analizar ahora el código de disparo

Previamente vamos a explicar, por fin, cómo trabajar con rectángulos en pygame ya que nuestra forma de trabajar hasta ahora con una variable para cada x e y lo hará inviable.

En Pygame, un `Rect` representa un **rectángulo** y se usa para **dibujar, mover y detectar colisiones**.

Se puede crear así:

```
rect = pygame.Rect(100, 150, 50, 50)  # x, y, ancho, alto
```

- `x` y `y` representan la **posición del rectángulo**, normalmente la esquina **superior izquierda**.
- `width` y `height` son **ancho y alto**.

Simplemente con esa línea hemos creado un montón de “propiedades” que se irán modificando según se vaya moviendo el rectángulo sin que sea necesaria nuestra intervención como programadores. Te dejo un resumen de las propiedades más importantes

Propiedades importantes de Rect

Propiedad	Qué representa
`rect.x`	Coordenada x de la esquina superior izquierda
`rect.y`	Coordenada y de la esquina superior izquierda
`rect.top`	Coordenada y del borde superior
`rect.bottom`	Coordenada y del borde inferior
`rect.left`	Coordenada x del borde izquierdo
`rect.right`	Coordenada x del borde derecho
`rect.centerx`	Coordenada x del centro
`rect.centery`	Coordenada y del centro
`rect.width`	Ancho del rectángulo
`rect.height`	Alto del rectángulo
`rect.topleft`	Tupla (x, y) de la esquina superior izquierda
`rect.bottomright`	Tupla (x, y) de la esquina inferior derecha

Modificando estas propiedades mueves el rectángulo.
Ejemplo:

```
rect.x += 5      # mover a la derecha
rect.y -= 3      # mover hacia arriba
rect.top = 100   # colocar borde superior en y=100
```

Métodos de colisión

Pygame incluye métodos muy útiles para detectar choques entre rectángulos. Nosotros nos vamos a centrar en colliderect y utlizaremos un for para las colisiones entre distintas listas de rectángulos, complica la programación, pero nos permitirá introducir for dentro de nuestros códigos pygame:

`colliderect()`

Comprueba si un rectángulo **choca con otro**:

```
jugador = pygame.Rect(100, 100, 50, 50)
enemigo = pygame.Rect(120, 120, 50, 50)

if jugador.colliderect(enemigo):
    print("¡Choque!")
```

En ocasiones nos interesará comprobar si el rectángulo del jugador se ha “chocado” con un enemigo y se pierdan vidas. Aquí es el momento en el que añadiremos un for que recorrerá todo la lista de enemigos, o disparos, y eliminará vidas al jugador:

```
for e in enemigos:
      if e.colliderect(jugador):
      vidas -= 1
```

Vamos a explicar el código de disparo:

El juego consiste en un **rectángulo que se mueve** y **dispara proyectiles** hacia arriba. Los `for` se usan principalmente para **recorrer listas de disparos** y **procesar eventos**.

---

1. Capturar eventos:

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        sys.exit()

    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_SPACE:
            nuevo_disparo = pygame.Rect(jugador.centerx - 5, jugador.top - 10, 10, 20)
            disparos.append(nuevo_disparo)
```

Qué pasa aquí:

- `pygame.event.get()` devuelve **una lista de eventos** que han ocurrido desde el último ciclo.
- Cada evento se asigna a la variable `event` en el `for`.
- Dentro del bucle, revisamos **qué tipo de evento es**:
    - Si es `QUIT`, cerramos el juego.
    - Si es una tecla pulsada (`KEYDOWN`) y es **ESPACIO**, **creamos un nuevo disparo** y lo añadimos a la lista `disparos`.
    - Observamos que utilizamos las propiedades centerx y top del jugador para darle las coordenadas iniciales al rectángulo disparo que se generará.

2. Actualizar la posición de los disparos: `for d in disparos:`

```
for d in disparos:
    d.y -= vel_disparo  # mover hacia arriba
```

Qué pasa aquí:

- `disparos` es una lista que contiene **rectángulos**.
- Cada disparo se asigna a `d` y **su coordenada `y` se reduce**, porque en Pygame, `y=0` está arriba de la pantalla.
- La coordenada y del disparo se actualizará en cada pasada del bucle generando la imagen de subida del disparo.

3. Eliminar disparos que salen de la pantalla

```
for d in disparos:
    if d.top < 10:
        disparos.remove(d)
```

Qué pasa aquí:

- Recorremos la lista `disparos`.
- Si un disparo ha subido demasiado (`top < 10`), lo eliminamos.
- Nota: **eliminar elementos mientras recorremos la lista puede ser riesgoso**, porque puede saltarse elementos. Por eso se suele usar **list comprehension**, como en la línea comentada:

```
# disparos = [d for d in disparos if d.bottom > 0]
```

4. Dibujar los disparos

```
for d in disparos:
    pygame.draw.rect(screen, color_disparo, d)
```

Qué pasa aquí:

- Recorremos cada disparo en la lista.
- Dibujamos un rectángulo en la pantalla con `pygame.draw.rect`.
- Otro ejemplo de **recorrer todos los elementos de una lista** para **realizar una acción con cada uno**.

---

5. Resumen de patrones de `for` en este juego

1. **Recorrer listas de eventos:**

   ```
   for event in pygame.event.get():
       # procesar cada evento
   ```

2. **Recorrer lista de disparos para actualizar posiciones:**

   ```
   for d in disparos:
       d.y -= vel_disparo
   ```

3. **Recorrer lista de disparos para eliminar elementos:**

   ```
   for d in disparos:
       if d.top < 10:
           disparos.remove(d)
   ```

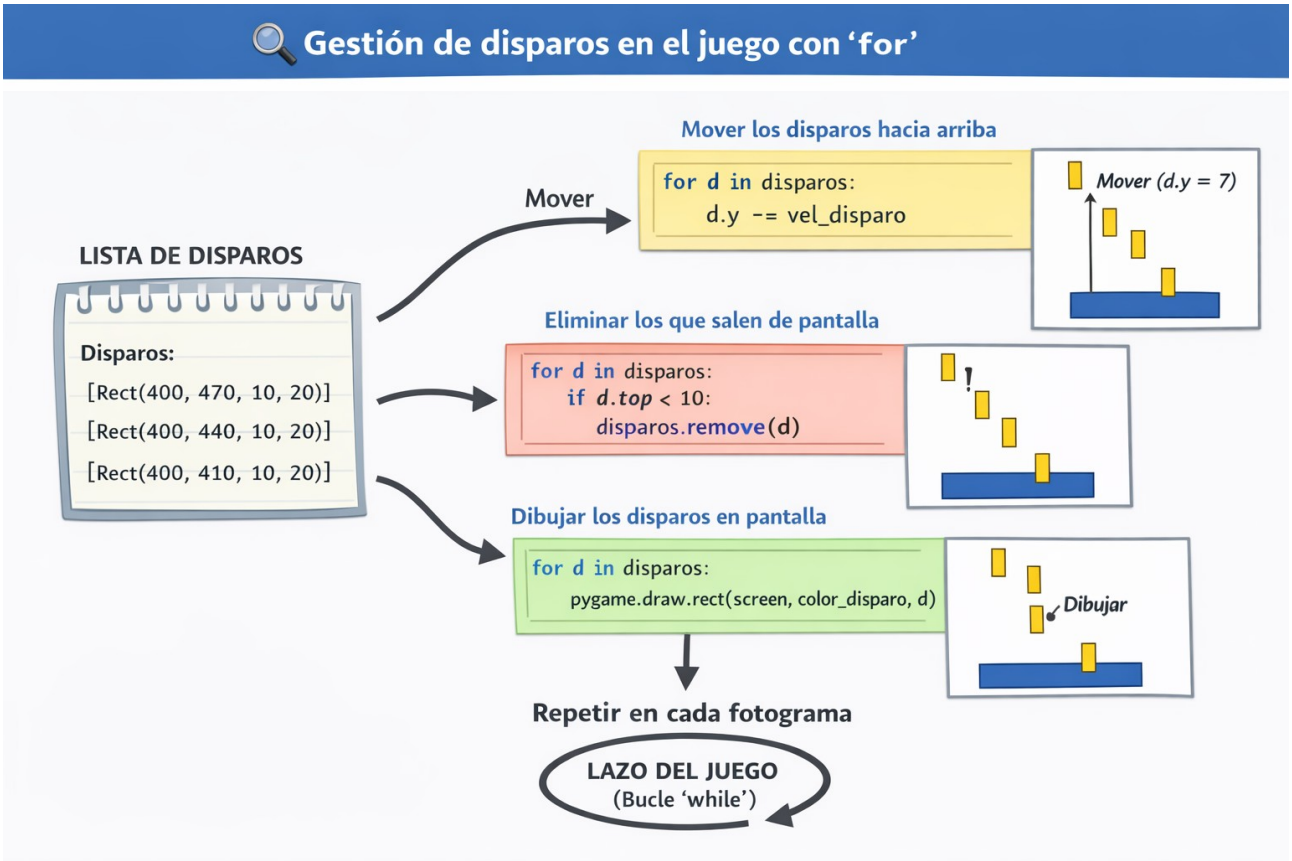
   o usando list comprehension:

   ```
   disparos = [d for d in disparos if d.bottom > 0]
   ```

4. **Recorrer lista de disparos para dibujar:**

   ```
   for d in disparos:
   ```

```
pygame.draw.rect(screen, color_disparo, d)
```



Reto 1: Vamos a crear enemigos. Solo se crearán y se mostrarán en pantalla bajando. A

Reto2 : Los enemigos, al volver a salir por la parte de arriba de la pantalla tendrán posiciones diferentes.

Reto3: Al chocar con un disparo serán eliminados y aumentaremos un contador. Al eliminar a todos los enemigos se terminará el juego.

Reto 4: Si los enemigos chocan con el jugador este perderá una vida. El juego terminará cuando el jugador haya perdido todas sus vidas o se hayan eliminado todos los enemigos.

Reto 5: Al finalizar, se debe mostrar una pantalla con el total de enenmigos eliminados y el total de vidas restantes del jugador.

Reto 6: Se mostará un marcador indicando el número de enemigos eliminados y la vida del jugador.