

# Funciones Dinámicas

## Práctica Funciones Dinámicas 1

Crea una función (`todos_positivos`) que reciba una lista de números como parámetro, y devuelva `True` si todos los valores de una lista son positivos, y `False` si al menos uno de los valores es negativo.

*Crea una lista llamada `lista_numeros` con valores positivos y negativos.*

---

## Práctica Funciones Dinámicas 2

Crea una función (`suma_menores`) que sume los números de una lista (almacenada en la variable `lista_numeros`) siempre y cuando sean mayores a 0 y menores a 1000, y devuelva el resultado de dicha suma.

---

## Práctica Funciones Dinámicas 3

Crea una función (`cantidad_pares`) que cuente la cantidad de números pares que existen en una lista (`lista_numeros`), y devuelva el resultado de dicha cuenta.

---

# Iteración entre Funciones

## Práctica sobre Interacción entre Funciones 1

Crea una función (`lanzar_dados`) que arroje dos dados al azar y devuelva sus resultados:

- La función debe retornar dos valores resultado, que se encuentren entre 1 y 6.

- No debe requerir argumentos para funcionar, sino que debe generar internamente los valores aleatorios.

Proporciona el resultado de estos dos dados a una función llamada `evaluar_jugada` (recibe dos argumentos) y que retorne, **sin imprimirlo**, un mensaje según la suma:

- Si la suma es menor o igual a 6:  
`"La suma de tus dados es {suma_dados}. Lamentable"`
- Si la suma es mayor a 6 y menor a 10:  
`"La suma de tus dados es {suma_dados}. Tienes buenas chances"`
- Si la suma es mayor o igual a 10:  
`"La suma de tus dados es {suma_dados}. Parece una jugada ganadora"`

**Pista:** Usa `random.choice` o `random.randint` para generar valores al azar entre 1 y 6.

---

## Práctica sobre Interacción entre Funciones

### 2

Crea una función llamada `reducir_lista()` que tome una lista como argumento (crea también la variable `lista_numeros`), y devuelva la misma lista, pero:

- eliminando duplicados (dejando uno solo),
- eliminando el valor más alto.

#### Ejemplo:

Si se proporciona `[1, 2, 15, 7, 2]`, debe devolver `[1, 2, 7]`.

Crea también una función `promedio()` que reciba como argumento la lista devuelta por `reducir_lista` y calcule el promedio de sus valores. Debe devolver el resultado, **sin imprimirlo**.

---

# Argumentos Indefinidos

## \*args

### Teoría

En Python, `*args` permite pasar un número variable de argumentos posicionales a una función.

Los argumentos se reciben como una **tupla**.

Se usa cuando **no sabes cuántos argumentos** se pasarán a la función.


```
1
2 def suma(*args):
3     resultado = 0
4     for numero in args:
5         resultado += numero
6     return resultado
7
8 # Ejemplos:
9 print(suma(1, 2))           # 3
10 print(suma(1, 2, 3, 4, 5)) # 15
11 print(suma())               # 0
12
```

---

### Práctica sobre \*args 1

Crea una función llamada `suma_cuadrados` que tome una cantidad indeterminada de argumentos numéricos, y retorne la suma de sus valores al cuadrado.

Salida esperada:



```
1
2 def suma_cuadrados(*args):
3     return sum(x**2 for x in args)
4
5 resultado = suma_cuadrados(1, 2, 3)
6 print(resultado) # Salida: 14 (1² + 2² + 3²)
7
```


---

## Práctica sobre \*args 2

Crea una función llamada `numeros_persona` que reciba como primer argumento un **nombre**, y a continuación una cantidad indefinida de **números**.

Debe devolver el mensaje:

`"{nombre}, la suma de tus números es {suma_numeros}"`



```
1 print(numeros_persona("Diego",1,2,3))
2 #salida esperada
3 #Hola Diego la suma de tus numeros es 6
```

---

## **\*\*kwargs**

### Teoría

En Python, **\*\*kwargs** permite pasar una cantidad variable de argumentos con nombre (clave-valor).

Se reciben como un **diccionario**.

Útil cuando **no sabes qué claves ni cuántas** se pasarán.

```
1
2 def mostrar_info(**kwargs):
3     for clave, valor in kwargs.items():
4         print(f"{clave}: {valor}")
5
6 # Ejemplo:
7 mostrar_info(nombre="Juan", edad=30, ciudad="Santiago")
8 #salida
9 # nombre: Juan
10 # edad: 30
11 # ciudad: Santiago
```

---

## Práctica sobre \*\*kwargs 1

Crea una función llamada `cantidad_atributos` que cuente la cantidad de parámetros que se entregan y devuelva esa cantidad como resultado.

Pista `len(nnnnnnnnnnnn(args))` o `kwargs`

```
1
2 print(cantidad_atributos(1, 2, 3)) # Salida: 3
3 print(cantidad_atributos(nombre="Juan", edad=30)) # Salida: 2
4 print(cantidad_atributos(1, 2, nombre="Juan")) # Salida: 3
5
```

---

## Práctica sobre \*\*kwargs 2

Crea una función llamada `lista_atributos` que devuelva en forma de lista los **valores** de los atributos entregados como palabras clave (`kwargs`).

Debe funcionar con cualquier cantidad de argumentos.

```
1
2 print(lista_atributos(nombre="Ana", edad=25, ciudad="Santiago"))
3 # Salida: ['Ana', 25, 'Santiago']
4
5 print(lista_atributos())
6 # Salida: []
```

---

## Práctica sobre \*\*kwargs 3

Crea una función llamada `describir_persona`, que tome como parámetro un nombre y luego una cantidad indeterminada de argumentos.

Ejemplo y salida:

```
1 ...
2 Ejemplo:
3 describir_persona("María", color_ojos="azules", color_pelo="rubio")
4
5 #salida esperada
6 Características de {nombre}:
7 {clave1}: {valor1}
8 {clave2}: {valor2}
```

## Reto:

### Programar el juego “el ahorcado”

El juego es sencillo y muy conocido, pero aquí va un repaso rápido:

- El programa elegirá al azar una palabra secreta de una lista.

- Mostrará al jugador una serie de guiones que representan las letras de esa palabra.
- El jugador deberá adivinar la palabra, letra por letra.
- Si acierta, se revelan todas las posiciones en las que aparece esa letra.
- Si falla, pierde una vida. Tiene un total de **6 vidas**.
- Si se queda sin vidas antes de adivinar la palabra completa, pierde. Si la adivina antes, gana.

## ¿Cómo empezar?

Recomendaciones:

1. **Importa** la función `choice` para seleccionar aleatoriamente una palabra de una lista.
2. **Crea una lista** de palabras posibles al inicio del programa.
3. **Desarrolla funciones** para:
  - Solicitar una letra al usuario.
  - Validar que el ingreso sea una letra válida.
  - Verificar si la letra está en la palabra.
  - Mostrar el estado actual del juego (letras acertadas y vidas restantes).
  - Determinar si el jugador ha ganado o perdido.

Recuerda: primero define las funciones y luego implementa el flujo principal del programa de forma ordenada.