20 ejercicios de Python para mejorar tu lógica de programación

Por Antonio Richaud

20 ejercicios de Python para mejorar tu lógica de programación

Publicado por Antonio Richaud

Introducción

Bienvenidos a esta guía diseñada con el objetivo de ayudar a quienes estan empezando o quieren repasar sus conocimientos en el lenguaje de programación Python. Cada ejercicio aquí presentado ha sido cuidadosamente seleccionado para desafiar tu lógica, mejorar tu comprensión de los conceptos básicos y sobre todo, para que disfrutes resolviendo los problemas que presento aqui.

Antes de comenzar, te invito a que enfrentes cada desafío con la mentalidad de resolverlo por tu cuenta. Sabemos que la tentación de mirar la respuesta es grande, pero te aseguro que la verdadera satisfacción está en esforzarte y experimentar esos momentos de "¡Eureka!" cuando finalmente encuentras la solución por ti mismo. Cada problema es una oportunidad para desarrollar tu intuición y habilidad como programador.

Esta guía ha sido creada esperando, de todo corazón, que sea una herramienta útil para ti, sin importar si estás empezando o si ya tienes experiencia. El aprendizaje es un viaje, y esta guía está aquí para acompañarte, paso a paso. Confía en tu capacidad, disfruta del proceso, y no olvides que cada error es un avance más hacia el éxito. :)

Ejercicio 1: Crear una lista de números del 1 al 10 y calcular la suma

Objetivo: Introducir la creación de listas, la iteración básica y el uso de funciones integradas como sum().

In [2]:

```
# Solicitar el número final de la lista
num = int(input("Introduce el número final de la lista: "))

# Crear una lista desde 1 hasta el número ingresado
lista = list(range(1, num + 1))

# Calcular la suma de los elementos de la lista
resultado = sum(lista)

# Mostrar el resultado
print(f"La suma de la lista hasta {num} es {resultado}")
```

```
Introduce el número final de la lista: 10
La suma de la lista hasta 10 es 55
```

Explicación:

1. **Usamos input () para pedir un número al usuario.**
2. **Con range (), creamos una secuencia de números desde el 1 hasta el valor que el usuario haya proporcionado.**
3. **La función sum () calcula la suma de todos los números en la lista.**
4. **Finalmente, mostramos el resultado con print ().**

Ejercicio 2: Crear una función que devuelva el cuadrado de un número

Objetivo: Introducir funciones, argumentos y valores de retorno.

In [3]:

```
# Definir la función para calcular el cuadrado de un número
def cuadrado_num(num):
    return num ** 2

# Solicitar el número al usuario
num = int(input("Introduce un número para calcular su cuadrado: "))

# Mostrar el resultado llamando a la función
print(f"El cuadrado de {num} es {cuadrado_num(num)}")
```

```
Introduce un número para calcular su cuadrado: 15
```

```
El cuadrado de 15 es 225
```

Explicación:

1. **Definimos una función llamada cuadrado_num () que toma un número como argumento y devuelve su cuadrado.**
2. **Utilizamos el operador ** para calcular el cuadrado del número.**
3. **Pedimos al usuario un número con input () y llamamos a la función para mostrar el resultado.**

Ejercicio 3: Crear un diccionario con tres pares clave-valor que representen un conjunto de herramientas de ciencia de datos (por ejemplo, 'Lenguaje': 'Python')

Objetivo: Aprender a crear y acceder a diccionarios.

In [4]:

```
# Crear un diccionario con herramientas de ciencia de datos
herramientas_ds = {
    "Lenguaje": "Python",
    "Biblioteca": "Pandas",
    "Visualización": "Seaborn"
}

# Mostrar el diccionario
print(f"Diccionario: {herramientas_ds}")
```

```
Diccionario: {'Lenguaje': 'Python', 'Biblioteca': 'Pandas', 'Visualización': 'Seaborn'}
```

Explicación:

1. **Un diccionario en Python se crea usando llaves {}, y contiene pares clave-valor.**
2. **En este ejemplo, hemos creado un diccionario con claves que representan aspectos de un toolkit de ciencia de datos.**
3. **Finalmente, mostramos el contenido del diccionario con print ().**

Ejercicio 4: Verificar si un número es positivo, negativo o cero

Objetivo: Introducir las declaraciones condicionales (if, elif, else).

In [5]:

```
# Solicitar un número al usuario
num = int(input("Introduce un número: "))

# Verificamos si el número es positivo, negativo o cero
if num > 0:
    print("El número es positivo")
elif num < 0:
    print("El número es negativo")
else:
    print("El número es 0")
```

```
Introduce un número: 10
El número es positivo
```

Explicación:

1. **Este código utiliza condicionales para verificar el valor del número ingresado por el usuario.**
2. **if verifica si el número es mayor que 0 (positivo).**
3. **elif verifica si es menor que 0 (negativo).**
4. **Si no cumple con ninguna de las anteriores, la condición else determina que el número es 0.**

Ejercicio 5: Crear una lista de números pares del 1 al 20 usando un bucle

Objetivo: Enseñar cómo usar bucles (for) y lógica condicional.

In [6]:

```
# Crear una lista vacía para los números pares
lista_pares = []

# Usar un bucle para iterar de 1 a 20
```

```
for num in range(1, 21):
    if num % 2 == 0:  # Verificar si el número es par
        lista_pares.append(num)

# Mostrar la lista de números pares
print(lista_pares)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Explicación:

1. **Usamos un bucle for que itera desde 1 hasta 20.**
2. **Dentro del bucle, usamos la condición num % 2 == 0 para verificar si el número es par (módulo igual a 0).**
3. **Si es par, lo añadimos a la lista con el método append ().**
4. **Al final, mostramos la lista de números pares.**

Ejercicio 6: Calcular el factorial de un número usando un bucle while

Objetivo: **Introducir bucles while y control básico de bucles.**

In [7]:

```
# Solicitar el número al usuario
num = int(input("Introduce un número: "))

# Inicializar variables
factorial = 1
i = 1

# Usar un bucle while para calcular el factorial
while i <= num:
    factorial *= i
    i += 1

# Mostrar el resultado
print(f"El factorial de {num} es {factorial}")
```

```
Introduce un número: 15
El factorial de 15 es 1307674368000
```

Ejercicio 7: Invertir una cadena de texto usando un bucle

Objetivo: **Enseñar la manipulación de cadenas y la iteración mediante bucles.**

In [8]:

```
# Solicitar una cadena de texto al usuario
cadena = input("Introduce una cadena de texto: ")

# Inicializar una cadena vacía para almacenar la reversa
reversa = ""

# Usar un bucle para invertir la cadena
for letra in cadena:
    reversa = letra + reversa

# Mostrar la cadena invertida
print(f"La cadena invertida es: {reversa}")
```

```
Introduce una cadena de texto: anitalavalatina
La cadena invertida es: anitalavalatina
```

Explicación:

1. **Solicitamos una cadena de texto al usuario.**
2. **Usamos un bucle for para iterar sobre cada carácter de la cadena.**
3. **En cada iteración, añadimos el carácter actual al inicio de la nueva cadena reversa, creando así una versión invertida.**
4. **Finalmente, mostramos la cadena invertida.**

Ejercicio 8: Contar el número de vocales en una cadena de texto

Objetivo: **Practicar bucles, condicionales y trabajar con cadenas.**

In [9]:

```
# Definir las vocales
vocales = "aeiouAEIOU"

# Solicitar una cadena de texto al usuario
cadena = input("Introduce una cadena de texto: ")

# Inicializar un contador para las vocales
contador_vocales = 0

# Usar un bucle para contar las vocales en la cadena
for letra in cadena:
    if letra in vocales:
        contador_vocales += 1

# Mostrar el número de vocales
print(f"El número de vocales en la cadena es: {contador_vocales}")
```

```
Introduce una cadena de texto: Hola, mi nombre es Antonio Richaud
El número de vocales en la cadena es: 13
```

Explicación:

1. **Definimos una cadena con las vocales en mayúsculas y minúsculas.**
2. **Usamos un bucle for para iterar sobre cada letra de la cadena ingresada por el usuario.**
3. **Si la letra es una vocal, incrementamos el contador de vocales.**
4. **Al final, mostramos la cantidad total de vocales.**

Ejercicio 9: Crear una función que verifique si una lista está vacía

Objetivo: **Introducir funciones, operaciones con listas y condicionales.**

In [15]:

```
# Definir la función para verificar si una lista está vacía
def es_lista_vacia(lista):
    return len(lista) == 0

# Crear una lista vacía
mi_lista = []

# Verificar si la lista está vacía
print(f"¿La lista está vacía? {es_lista_vacia(mi_lista)}")
```

```
¿La lista está vacía? True
```

Explicación:

1. **Definimos una función es_lista_vacia () que toma una lista como argumento.**
2. **Usamos la función len () para verificar si la lista tiene algún elemento. Si la longitud es 0, la lista está vacía.**
3. **Luego, creamos una lista vacía mi_lista y llamamos a la función para verificar si está vacía.**

Ejercicio 10: Encontrar el número máximo en una lista sin usar funciones incorporadas

Objetivo: **Enseñar cómo iterar a través de una lista y comparar valores.**

In [16]:

```
# Definir la lista de números
lista = [5, 6, 9, 1, 15, 19, 1, 2]

# Inicializar el valor máximo como el primer número de la lista
max_num = lista[0]

# Iterar a través de la lista para encontrar el número máximo
for num in lista:
    if num > max_num:
        max_num = num

# Mostrar el número máximo
print(max_num)
```

Explicación:

1. **Inicializamos la variable max_num con el primer valor de la lista.**
2. **Usamos un bucle for para recorrer cada número de la lista y comparamos si es mayor que max_num. Si lo es, actualizamos max_num.**
3. **Al final, mostramos el número máximo.**

Ejercicio 11: Usar un bucle para imprimir un patrón de estrellas

Objetivo: Practicar bucles anidados y la creación de patrones.

In [19]:

```
# --- Patrón creciente ---
# Definir el número de líneas del patrón
n = 5

# Usar un bucle para imprimir el patrón de estrellas
for i in range(1, n + 1):
    for j in range(i):
        print("*", end="")
    print("")
```

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

In [20]:

```
# --- Patrón decreciente ---
# Definir el número de líneas del patrón
n = 5

# Usar un bucle para imprimir el patrón de estrellas en orden inverso
for i in range(n, 0, -1):
    for j in range(i):
        print("*", end="")
    print("")
```

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

Explicación:

1. **Usamos bucles anidados: el primer bucle for controla las líneas, y el segundo bucle for controla cuántas estrellas se imprimen por cada línea.**
2. **El patrón creciente imprime una cantidad de estrellas creciente en cada línea, mientras que el patrón decreciente imprime una cantidad de estrellas decreciente.**

Ejercicio 12: Crear una lista con los primeros 10 números cuadrados usando un bucle

Objetivo: Practicar la iteración de bucles y la manipulación de listas.

In [21]:

```
# Crear una lista vacía
lista_cuadrados = []

# Usar un bucle para generar los primeros 10 números cuadrados
for i in range(1, 11):
    lista_cuadrados.append(i ** 2)

# Mostrar la lista de números cuadrados
print(lista_cuadrados)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Explicación:

Explicación:

1. **Usamos un bucle for para iterar del 1 al 10.**
2. **Para cada número, calculamos su cuadrado usando i ** 2 y lo añadimos a la lista lista_cuadrados usando append ().**
3. **Al final, mostramos la lista de números cuadrados.**

Ejercicio 13: Encontrar la suma de todos los números impares entre 1 y 50

Objetivo: Usar bucles y condicionales para filtrar y sumar valores.

In [22]:

```
# Inicializar el contador de suma
suma_impares = 0

# Usar un bucle para iterar del 1 al 50
for i in range(1, 51):
    if i % 2 != 0:  # Verificar si el número es impar
        suma_impares += i

# Mostrar la suma total de los números impares
print(suma_impares)
```

```
625
```

Explicación:

1. **Usamos un bucle for para recorrer los números del 1 al 50.**
2. **Si el número es impar (i % 2 != 0), lo sumamos al contador suma_impares.**
3. **Finalmente, mostramos la suma total de los números impares.**

Ejercicio 14: Escribir una función para verificar si una cadena es un palíndromo

Objetivo: Introducir la manipulación de cadenas y la comparación de texto.

In [23]:

```
# Solicitar una cadena de texto al usuario
cadena = input("Introduce una cadena de texto: ")

# Invertir la cadena
reversa = cadena[::-1]

# Verificar si la cadena original es igual a la cadena invertida (ignorando mayúsculas)
if cadena.lower() == reversa.lower():
    print(f"{cadena} es un palíndromo")
else:
    print(f"{cadena} no es un palíndromo")
```

```
Introduce una cadena de texto: anilina
anilina es un palíndromo
```

Explicación:

1. **Usamos slicing [::-1] para invertir la cadena.**
2. **Comparamos la cadena original con la cadena invertida usando el método lower () para ignorar diferencias de mayúsculas y minúsculas.**
3. **Si ambas cadenas coinciden, es un palíndromo.**

Ejercicio 15: Encontrar la longitud de la palabra más larga en una lista

Objetivo: Enseñar la iteración sobre listas y el cálculo de la longitud de cadenas.

In [25]:

```
# Definir una lista de palabras
palabras = ["Data", "Science", "Python", "Machinelearning"]

# Inicializar la variable para la palabra más larga
palabra_larga = max(palabras, key=len)

# Mostrar la palabra más larga
```

```
print(f"La palabra más larga es: {palabra_larga}")
```

```
La palabra más larga es: Machinelearning
```

Explicación:

1. **Usamos la función max () con el argumento key=len para encontrar la palabra con mayor longitud en la lista.**
2. **Mostramos la palabra más larga.**

Ejercicio 16: Usar un bucle para imprimir la secuencia de Fibonacci hasta el décimo término

Objetivo: **Practicar la iteración con bucles y la generación de secuencias.**

In [26]:

```
# Inicializar los primeros dos términos de Fibonacci
n1, n2 = 0, 1

# Imprimir los primeros 10 términos de Fibonacci
print("Serie de Fibonacci para 10 números es:")
for i in range(10):
    print(n1, end=", ")
    num = n1 + n2
    n1 = n2
    n2 = num
```

```
Serie de Fibonacci para 10 números es:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
```

Explicación:

1. **Inicializamos los primeros dos términos de la secuencia de Fibonacci (n1 y n2).**
2. **Usamos un bucle for para calcular e imprimir los primeros 10 términos de la secuencia.**
3. **En cada iteración, sumamos n1 y n2 para generar el siguiente número de la secuencia.**

Ejercicio 17: Crear un diccionario para contar la cantidad de veces que aparece cada carácter en una cadena

Objetivo: **Enseñar el uso de diccionarios y la iteración sobre cadenas.**

In [27]:

```
# Definir la cadena de texto
texto = "Antonio Richaud"

# Crear un diccionario vacío para almacenar los conteos
conteos = {}

# Iterar sobre cada carácter en la cadena
for char in texto:
    if char in conteos:
        conteos[char] += 1
    else:
        conteos[char] = 1

# Mostrar el diccionario con los conteos
print(conteos)
```

```
{'A': 1, 'n': 2, 't': 1, 'o': 2, 'i': 2, ' ': 1, 'R': 1, 'c': 1, 'h': 1, 'a': 1, 'u': 1, 'd': 1}
```

Explicación:

1. **Creamos un diccionario vacío llamado conteos.**
2. **Iteramos sobre cada carácter de la cadena texto. Si el carácter ya existe como clave en el diccionario, aumentamos su valor en 1. Si no existe, lo agregamos con un valor inicial de 1.**
3. **Finalmente, mostramos el diccionario con los conteos de cada carácter.**

Ejercicio 18: Escribe un programa que imprima los números del 1 al 30, pero para múltiplos de 3 imprima “Tres”, para múltiplos de 5 imprima “Cinco” y para múltiplos de ambos imprima “Tres y Cinco”

In [28]:

```
# Iterar sobre los números del 1 al 30
for i in range(1, 31):
    if i % 3 == 0 and i % 5 == 0:
        print("Tres y Cinco")
    elif i % 3 == 0:
        print("Tres")
    elif i % 5 == 0:
        print("Cinco")
    else:
        print(i)
```

```
1
2
Tres
4
Cinco
Tres
7
8
Tres
Cinco
11
Tres
13
14
Tres y Cinco
16
17
Tres
19
Cinco
Tres
22
23
Tres
Cinco
26
Tres
28
29
Tres y Cinco
```

Explicación:

1. **Iteramos sobre los números del 1 al 30.**
2. **Si el número es divisible por 3 y 5, imprimimos "Tres y Cinco".**
3. **Si es divisible solo por 3, imprimimos "Tres". Si es divisible solo por 5, imprimimos "Cinco".**
4. **Si no cumple ninguna de estas condiciones, imprimimos el número.**

Ejercicio 19: Crear un bucle anidado para imprimir una tabla de multiplicar para los números del 1 al 5

Objetivo: **Enseñar el uso de bucles anidados y operaciones aritméticas básicas.**

In [29]:

```
# Usar bucles anidados para imprimir la tabla de multiplicar del 1 al 5
for i in range(1, 6):
    for j in range(1, 6):
        print(f"{i} x {j} = {i * j}", end="  ")
    print("")
```

```
1 x 1 = 1  1 x 2 = 2  1 x 3 = 3  1 x 4 = 4  1 x 5 = 5
2 x 1 = 2  2 x 2 = 4  2 x 3 = 6  2 x 4 = 8  2 x 5 = 10
3 x 1 = 3  3 x 2 = 6  3 x 3 = 9  3 x 4 = 12  3 x 5 = 15
4 x 1 = 4  4 x 2 = 8  4 x 3 = 12  4 x 4 = 16  4 x 5 = 20
5 x 1 = 5  5 x 2 = 10  5 x 3 = 15  5 x 4 = 20  5 x 5 = 25
```

Explicación:

1. **Usamos un bucle anidado donde i representa las filas y j las columnas.**
2. **En cada iteración, multiplicamos i por j y mostramos el resultado en formato de tabla de multiplicar.**
3. **Después de completar cada fila, imprimimos una nueva línea.**

Ejercicio 20: Escribir un programa que encuentre el segundo número más grande en una lista

Objetivo: Introducir la ordenación de listas y el uso de índices.

In [30]:

```
# Definir la lista de números
lista = [5, 6, 9, 1, 15, 19, 1, 2]

# Ordenar la lista
lista.sort()

# Mostrar el segundo número más grande
print(lista[-2])
```

```
15
```

Explicación:

1. Usamos el método sort () para ordenar la lista de números en orden ascendente.
2. Después de ordenar la lista, accedemos al segundo número más grande utilizando el índice -2, que representa el penúltimo elemento.