

ACTIVIDAD IA 802

Actividad 1: Programación en Python

A continuación, se presentan ejercicios para practicar estructuras de control (if, for, while, switch, funciones). Cada ejercicio incluye el código y su explicación para que puedas entender el **por qué** y **para qué** de su uso. Tu tarea es **replicar, modificar y comentar** cada fragmento de código.

Estructura condicional if

Ejercicio 1. Verifica si un número es positivo, negativo o cero.

```
1  edad = int(input("Introduce tu edad: "))
2
3  if edad < 13:
4      print("Eres un niño.")
5  elif edad < 18:
6      print("Eres un adolescente.")
7  elif edad < 60:
8      print("Eres un adulto.")
9  else:
10     print("Eres un adulto mayor.")
```

¿Por qué if?

La estructura if-elif-else permite clasificar casos mutuamente excluyentes. Aquí usamos rangos para decidir en qué etapa de la vida se encuentra una persona.

Preguntas de lógica:

1. ¿Qué sucedería si no incluyéramos el else?

R: Si no incluyéramos el else, el programa no imprimiría nada si la edad es mayor o igual a 60. El programa simplemente terminaría sin imprimir ningún mensaje.

2. ¿Cómo cambiarías el código para incluir una categoría "adulto joven" de 18 a 29 años?

R: Para incluir una categoría "adulto joven" de 18 a 29 años, podrías agregar un nuevo elif:

3. ¿Qué pasaría si alguien introduce una edad negativa?

R: Si alguien introduce una edad negativa, el programa imprimirá "Eres un niño.". Sin embargo, esto no es correcto, ya que una edad negativa no tiene sentido.

Para manejar este caso, podrías agregar una condición adicional para verificar si la edad es negativa.

```
EJERCICIO 1.py - C:/Users/Ana Ceja/Desktop/PYTHON/EJERCICIO 1.py (3.13.2)
File Edit Format Run Options Window Help
edad = int(input("Introduce tu edad: "))

if edad < 0:
    print("Edad inválida. Por favor, introduce una edad positiva.")
elif edad < 13:
    print("Eres un niño.")
elif edad < 18:
    print("Eres un adolescente.")
elif edad <= 30:
    print("Eres un adulto joven.")
elif edad < 60:
    print("Eres un adulto.")
else:
    print("Eres un adulto mayor.")
```

Ejercicio 2: Validación de contraseña con advertencia

```
1  usuario = input("Usuario: ")
2  password = input("Contraseña: ")
3
4  if usuario == "admin" and password == "secreto123":
5      print("Acceso concedido.")
6  elif usuario != "admin":
7      print("Usuario incorrecto.")
8  else:
9      print("Contraseña incorrecta.")
```

¿Por qué if?

Permite validar condiciones combinadas con operadores lógicos (and, or). Aquí se usa para diferenciar fallas de autenticación.

Preguntas de lógica:

1. ¿Qué operadores lógicos se usan en el código?

R: and: Se utiliza para combinar dos condiciones y evaluar si ambas son verdaderas.

!=: Se utiliza para evaluar si dos valores son diferentes.

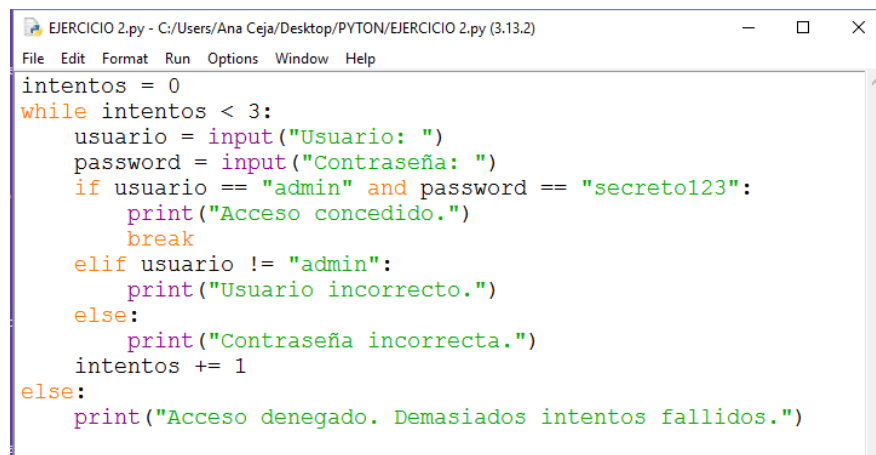
2. ¿Por qué es importante validar el usuario antes que la contraseña?

R: - Seguridad: Si se valida la contraseña antes de validar el usuario, un atacante podría intentar adivinar la contraseña de un usuario existente.

- Eficiencia: Validar el usuario antes de validar la contraseña reduce el número de consultas a la base de datos o al sistema de autenticación.

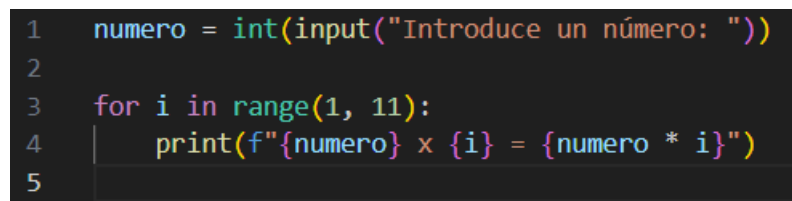
3. ¿Cómo implementarías un intento fallido limitado a 3 veces?

R: Para implementar un intento fallido limitado a 3 veces, se utiliza una variable para contar el número de intentos fallidos y un bucle para limitar el número de intentos.



```
EJERCICIO 2.py - C:/Users/Ana Ceja/Desktop/PYTHON/EJERCICIO 2.py (3.13.2)
File Edit Format Run Options Window Help
intentos = 0
while intentos < 3:
    usuario = input("Usuario: ")
    password = input("Contraseña: ")
    if usuario == "admin" and password == "secreto123":
        print("Acceso concedido.")
        break
    elif usuario != "admin":
        print("Usuario incorrecto.")
    else:
        print("Contraseña incorrecta.")
    intentos += 1
else:
    print("Acceso denegado. Demasiados intentos fallidos.")
```

Ejercicio 3: Mostrar tabla de multiplicar de un número



```
1  numero = int(input("Introduce un número: "))
2
3  for i in range(1, 11):
4      print(f"{numero} x {i} = {numero * i}")
5
```

¿Por qué for?

El for se usa cuando conoces de antemano el número de repeticiones. Aquí se usa para imprimir del 1 al 10.

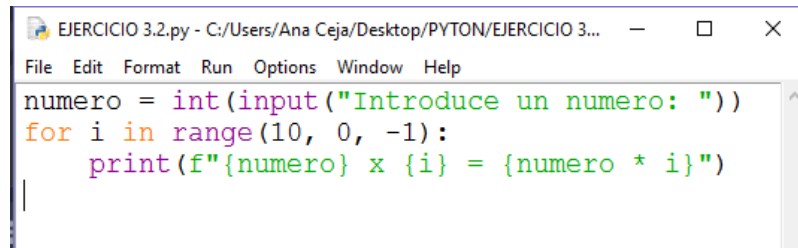
Preguntas de lógica:

- ¿Qué hace range(1, 11) exactamente? **R: range(1, 11)** es una función que genera un rango de números enteros desde 1 hasta 10. El primer parámetro (1) es el inicio del rango y el segundo parámetro (11) es el final del rango, pero no se incluye.

En otras palabras, `range(1, 11)` es equivalente a `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`.

2. ¿Cómo podrías modificar el código para imprimir la tabla al revés?

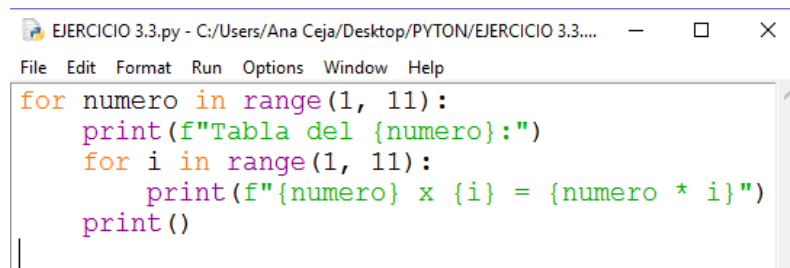
R: Para imprimir la tabla al revés, se modifica el rango para que comience en 10 y termine en 1, utilizando el parámetro `step` con un valor de -1.



```
EJERCICIO 3.2.py - C:/Users/Ana Ceja/Desktop/PYTHON/EJERCICIO 3...
File Edit Format Run Options Window Help
numero = int(input("Introduce un numero: "))
for i in range(10, 0, -1):
    print(f"{numero} x {i} = {numero * i}")
```

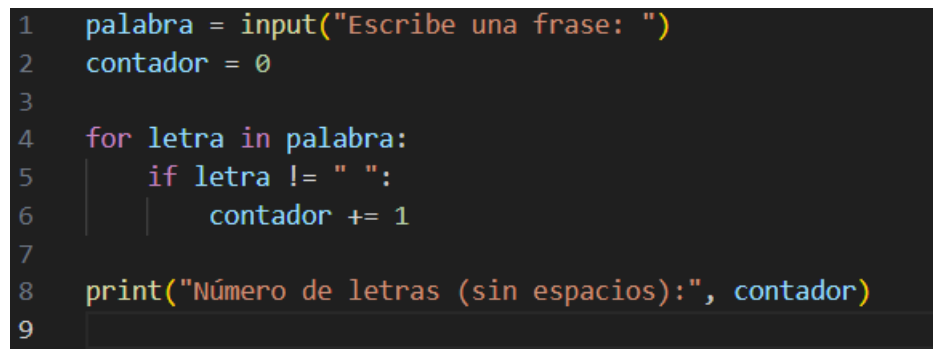
3. ¿Cómo harías que imprima todas las tablas del 1 al 10?

R: Para imprimir todas las tablas del 1 al 10, se utiliza un bucle anidado.



```
EJERCICIO 3.3.py - C:/Users/Ana Ceja/Desktop/PYTHON/EJERCICIO 3.3....
File Edit Format Run Options Window Help
for numero in range(1, 11):
    print(f"Tabla del {numero}:")
    for i in range(1, 11):
        print(f"{numero} x {i} = {numero * i}")
    print()
```

Ejercicio 4: Contar letras en una palabra (sin espacios)



```
1 palabra = input("Escribe una frase: ")
2 contador = 0
3
4 for letra in palabra:
5     if letra != " ":
6         contador += 1
7
8 print("Número de letras (sin espacios):", contador)
9
```

¿Por qué for con if?

El `for` recorre carácter por carácter, y el `if` filtra los espacios. Esto combina lógica de control con análisis de datos.

Preguntas de lógica:

1. ¿Qué otros caracteres podrías querer ignorar además de espacios?

R:

- Signos de puntuación (., !, ?, :, ;, etc.)
- Números
- Símbolos especiales (@, #, \$, etc.)
- Tabuladores (\t)
- Saltos de línea (\n)

```
EJERCICIO 4.1.py - C:/Users/Ana Ceja/Desktop/PYTHON/EJERCICIO 4.1.py (3.13.2)
File Edit Format Run Options Window Help
palabra = input("Escribe una frase: ")
contador = 0
for letra in palabra:
    if letra.isalpha(): # Solo cuenta letras
        contador += 1
print("Numero de letras:", contador)
```

2. ¿Cómo podrías contar solo vocales?

R: agregar una condición adicional al if.

```
EJERCICIO 4.2.py - C:/Users/Ana Ceja/Desktop/PYTHON/EJERCICIO 4.2.py (3.13.2)
File Edit Format Run Options Window Help
palabra = input("Escribe una frase: ")
contador = 0
vocales = "aeiouAEIOU"
for letra in palabra:
    if letra in vocales:
        contador += 1
print("Numero de vocales:", contador)
```

3. ¿Qué pasa si no usas if?

R: el contador incrementará por cada carácter en la palabra, incluyendo espacios, signos de puntuación, números, etc. El contador mostrará el número total de caracteres en la palabra, no solo las letras.

```
EJERCICIO 4.3.py - C:/Users/Ana Ceja/Desktop/PYTHON/EJERCICIO 4.3.py ...  
File Edit Format Run Options Window Help  
palabra = input("Escribe una frase: ")  
contador = 0  
for letra in palabra:  
    contador += 1  
print("Numero de caracteres:", contador)
```

Ejercicio 5: Validar entrada hasta que sea correcta

```
1  respuesta = ""  
2  
3  while respuesta != "python":  
4      respuesta = input("¿Cuál es el mejor lenguaje de programación?: ")  
5  
6  print("¡Correcto!")  
7
```

¿Por qué while?

Se utiliza cuando no sabes cuántas veces se va a repetir el ciclo. El programa sigue preguntando hasta que se cumpla la condición.

Preguntas de lógica:

1. ¿Qué pasaría si el usuario nunca escribe la palabra correcta?

R: Si el usuario nunca escribe la palabra correcta, el programa entrará en un bucle infinito, continuamente pidiendo al usuario que ingrese la respuesta.

2. ¿Cómo agregarías un número máximo de intentos?

R: Se utiliza una variable para contar el número de intentos y agregar una condición para salir del bucle cuando se alcance el máximo de intentos.

```
EJERCICIO 5.2.py - C:/Users/Ana Ceja/Desktop/PYTHON/EJERCICIO 5.2.py (3.13.2)
File Edit Format Run Options Window Help
respuesta = ""
intentos = 0
max_intentos = 3
while respuesta != "python" and intentos < max_intentos:
    respuesta = input("¿Cuál es el mejor lenguaje de programación?: ")
    intentos += 1
if respuesta == "python":
    print("¡Correcto!")
else:
    print("Lo siento, has agotado tus intentos.")|
```

3. ¿Puedes cambiar el programa para que no importe si escriben “Python” con mayúscula?

R: Ignoramos la mayúscula y minúscula, podríamos convertir la respuesta del usuario a minúscula antes de compararla con la respuesta correcta.

```
EJERCICIO 5.3.py - C:/Users/Ana Ceja/Desktop/PYTHON/EJERCICIO 5.3.py (3.13.2)
File Edit Format Run Options Window Help
respuesta = ""
intentos = 0
max_intentos = 3
while respuesta.lower() != "python" and intentos < max_intentos:
    respuesta = input("¿Cuál es el mejor lenguaje de programación?: ")
    intentos += 1
if respuesta.lower() == "python":
    print("¡Correcto!")
else:
    print("Lo siento, has agotado tus intentos.")|
```

Ejercicio 6: Cajero automático simulado

```
1  saldo = 1000
2  opcion = ""
3
4  while opcion != "salir":
5      print("\n1. Consultar saldo\n2. Retirar\n3. Salir")
6      opcion = input("Elige una opción: ")
7
8      if opcion == "1":
9          print("Saldo actual:", saldo)
10     elif opcion == "2":
11         cantidad = int(input("¿Cuánto deseas retirar?: "))
12         if cantidad <= saldo:
13             saldo -= cantidad
14             print("Retiro exitoso. Nuevo saldo:", saldo)
15         else:
16             print("Fondos insuficientes.")
17     elif opcion == "3":
18         opcion = "salir"
19     else:
20         print("Opción no válida.")
21
```

¿Por qué match-case?

Permite manejar múltiples casos claramente, como un switch. Es más legible que varios if.

Preguntas de lógica:

1. ¿Qué ventaja tiene match sobre if en este caso?

R: La instrucción match, que puede ser utilizada en lugar de if/elif/else en algunos casos. La ventaja de match en este caso es que permite una sintaxis más concisa y legible para comparar la variable opción con diferentes valores:

2. ¿Qué pasa si el usuario pone otro símbolo?

R: Si el usuario ingresa un símbolo o un valor que no es uno de los esperados ("1", "2", "3"), el programa imprimirá "Opción no válida."

3. ¿Qué deberías validar antes de dividir?

R: En este código, no hay divisiones. Sin embargo, si estuviéramos realizando divisiones, deberíamos validar que el divisor no sea cero antes de realizar la operación, para evitar un error de división por cero.

if divisor != 0:

resultado = dividendo / divisor

else:

print("Error: no se puede dividir por cero.")

Ejercicio 7: Menú de operaciones matemáticas

```
1  op = input("Elige una operación (+, -, *, /): ")
2  a = int(input("Primer número: "))
3  b = int(input("Segundo número: "))
4
5  match op:
6      case "+":
7          print("Resultado:", a + b)
8      case "-":
9          print("Resultado:", a - b)
10     case "*":
11         print("Resultado:", a * b)
12     case "/":
13         print("Resultado:", a / b if b != 0 else "No se puede dividir entre cero")
14     case _:
15         print("Operación no válida.")
16
```


¿Por qué match-case?

Permite manejar múltiples casos claramente, como un switch. Es más legible que varios if.

Preguntas de lógica:

1. ¿Qué ventaja tiene match sobre if en este caso?

R: - Permite una sintaxis más concisa y legible.

- Evita la necesidad de utilizar **elif** repetidamente.

- Proporciona una forma más elegante de manejar los casos en los que la variable **op** no coincide con ninguno de los valores esperados.

2. ¿Qué pasa si el usuario pone otro símbolo?

R: Si el usuario ingresa un símbolo o un valor que no es uno de los esperados ("+", "-", "*", "/"), el programa imprimirá "operación no valida."

3. ¿Qué deberías validar antes de dividir?

R: Antes de dividir, deberías validar que el divisor (b) no sea cero. En el código, esto ya se está haciendo:

case "/": print("resultado:", a / b if b != 0 else " no se puede dividir entre cero")

Sin embargo, en lugar de imprimir un mensaje de error, se considera lo siguiente:

```
EJERCICIO 7.py - C:/Users/Ana Ceja/Desktop/PYTHON/EJERCICIO 7.py (3.13.2)
File Edit Format Run Options Window Help

op = input("Elige una operacion (+, -, *, /): ")
a = int(input("Primer numero: "))
b = int(input("Segundo numero: "))

match op:
    case "+":
        print("resultado:", a + b)
    case "-":
        print("resultado:", a - b)
    case "*":
        print("resultado:", a * b)
    case "/":
        print("resultado:", a / b if b != 0 else " no se puede dividir entre cero")
    case _:
        if b == 0:
            raise ValueError("No se puede dividir entre cero")
        print("resultado:", a / b)
    case _:
        print("operacion no valida.")
```

Ejercicio 8: Función que devuelve si un número es primo

```
1 def es_primo(n):
2     if n < 2:
3         return False
4     for i in range(2, int(n**0.5) + 1):
5         if n % i == 0:
6             return False
7     return True
8
9 print(es_primo(13)) # True
10
```

¿Por qué función?

Porque encapsulamos una tarea reutilizable. Y se usa la raíz cuadrada como optimización.

Preguntas de lógica:

1. ¿Qué hace exactamente el bucle for dentro de la función?

R: El bucle for dentro de la función `es_primo(n)` verifica si el número `n` es divisible por cualquier número entero entre 2 y la raíz cuadrada de `n`. El bucle for comprueba si `n` tiene algún factor distinto de 1 y de sí mismo.

2. ¿Por qué se empieza desde 2 y no desde 1?

R: 1 no es un número primo y cualquier número es divisible por 1, por lo que no aportaría información útil para determinar si `n` es primo.

3. ¿Qué pasaría si no usáramos `return` dentro del `if`?

R: Si no se utiliza `return` dentro del `if`, el programa continuará ejecutando el resto de la función, incluso si se encuentra un divisor.

En este caso específico, si se encuentra un divisor, el programa imprimiría `False` (debido al `return False` fuera del bucle), pero si no se encuentra ningún divisor, el programa también imprimiría `False`, lo cual es incorrecto.

Por lo tanto, es importante utilizar `return` dentro del `if` para asegurarse de que la función termine correctamente en cuanto se encuentra un divisor.