Errores y dudas comunes con variables

- En una asignación, la variable debe estar del lado izquierdo del operador :

```
45=edad   ✗
edad=45   ✓
```

- Usar identificadores descriptivos que indiquen qué contiene cada variable :

```
a=input()
x=4412                    ✗
for xyz in yyy:
```

- Inicializar variables si en su primer uso será necesario su valor :

```
producto=input("Descripción: ")
while producto!="":
    cantidad=cantidad+1   ✗
    producto=input("Descripción: ")
```

al calcular cantidad+1, la variable cantidad aún no tiene valor (en una asignación, primero se calcula la expresión a la derecha del operador)

- No inicializar una variable si su primer uso va a ser asignándole un valor :

```
nombre=input("Tu nombre: ")   ✗
edad=input("Tu edad: ")
while edad!=0:
    nombre=input("Tu nombre: ")
    print(nombre, edad)
    edad=input("Tu edad: ")
```

el valor anterior se pierde sin haberse usado

Errores y dudas comunes con estructuras de control

- La variable iteradora de un bucle **for** se incrementa automáticamente :

```
for i desde 0 hasta 10:
    i=i+1
```

altera "artificialmente" el rango de iteración y puede provocar comportamiento inesperado

- Tanto **while** como **if** ejecutan su bloque de código sólo si la condición es True :

```
if numero>3
```

depende del valor de la variable **numero** en cada ejecución

```
while n<100 and nombre!=""
```

las variables **n** y **nombre** deben tener algún valor antes de ejecutar el while

- **If** ejecuta su bloque sólo una vez (si la condición es True), mientras que **while** puede repetir su bloque (mientras la condición sea True) :

```
while venta>1000

if nombreValido(usuario)
```

- **If** y **while** aceptan como condición a cualquier expresión booleana o valor literal booleano :

```
if True
```

no tiene sentido, pero la sintaxis es válida

```
while estaActivo and not(esMoroso(socio))
```

estaActivo es una variable y esMoroso() es una función

Errores y dudas comunes con expresiones lógicas

- Cada parte debe ser una expresión completa por sí misma :

```
dia=="lunes" or "martes"      ✗
dia=="lunes" or dia=="martes"  ✓
```

- Una condición debe expresarse según en qué estructura se la utilice :

"Salir del bucle si el número no es par o es mayor que 50."

```
while numero%2==0 and numero<=50
```
← continúa si es par y menor o igual que 50

```
if numero%2!=0 or numero>50:
    break
```
← sale del bucle si no es par o es mayor que 50

- Las expresiones deben construirse sin contradicciones :

```
if numero<10 and numero>20   ✗
```
Un número nunca puede ser menor que 10 y mayor que 20 al mismo tiempo.

- Una expresión booleana tiene un valor True o False y puede usarse en cualquier otra expresión que admita un valor booleano :

```
while esMayorDeEdad(persona)

if estaActivo and not(esMoroso(socio))
```

si una función o una operación retorna un valor booleano, puede usarse directamente en una expresión

Errores y dudas comunes con funciones

- Una función es una expresión y su valor de retorno se usará en el lugar donde la función esté siendo invocada :

```
existe = existeEmpleado(lista, nombre)

if not existeEmpleado(lista, nombre)
```

- Para una buena modularidad, una función debe representar una operación atómica (una tarea concreta) evitando hacer más de una cosa :

```
def existeEmpleado(lista, nombre):
    for empleado in lista:
        if empleado[2]==nombre:
            print("Se encontró el empleado")
            return True
    return False
```

Imprimir datos es una tarea en sí misma y hace que la función sea menos reutilizable. Corresponde imprimir en la unidad invocadora.

- La invocación debe contener todos los datos que la función necesita recibir como parámetros :

```
if existeEmpleado(lista)

if existeEmpleado(lista, nombre)
```

A menos que el parámetro **nombre** tenga un valor por defecto, habrá un error.

- Si una función retorna un valor, posiblemente deba hacerse algo con él (aunque hay excepciones en las que no se desea hacer nada) :

```
existeEmpleado(lista, nombre)

existe = existeEmpleado(lista, nombre)

if existeEmpleado(lista, nombre)
```

La llamada a la función retorna un valor True o False pero se pierde.