Trabalho Cap 8

Gabriel Visentin Alenxadre 23000602-2

1-

(While)

```
i = 1
while i <= 10:
   print(i)
   i += 1
```

(For)

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

Resposta:

Entre as três versões, o goto funciona, mas espalha o fluxo com rótulos e saltos, dificultando a leitura. O while já deixa explícita a condição e mantém a lógica em um bloco. Para contagens conhecidas, o for é o mais claro: concentra início, limite e passo em uma linha. Portanto, para este caso (1 a 10), for > while > goto em legibilidade.

2-

(C)

```
#include <stdio.h>

unsigned long long fatorial(int n) {
   if (n < 0) return 0; // 0 = inválido (simples)
   unsigned long long fat = 1;
   for (int i = 2; i <= n; i++) fat *= i;
   return fat;
}

int main(void) {
   int opcao;
   while (1) {
```

```
        printf("\n=== MENU (C) ===\n");
        printf("1) Quadrado de um numero\n");
        printf("2) Fatorial de um numero\n");
        printf("3) Sair\n");
        printf("Opcao: ");
        if (scanf("%d", &opcao) != 1) return 0;

        switch (opcao) {
            case 1: {
                double x;
                printf("Digite um numero: ");
                if (scanf("%lf", &x) != 1) return 0;
                printf("Quadrado: %.2f\n", x * x);
                break;
            }
            case 2: {
                int n;
                printf("Digite um inteiro >= 0: ");
                if (scanf("%d", &n) != 1) return 0;
                if (n < 0) {
                    printf("Valor invalido.\n");
                } else {
                    printf("%d! = %llu\n", n, fatorial(n));
                }
                break;
            }
            case 3:
                printf("Saindo...\n");
                return 0;
            default:
                printf("Opcao invalida.\n");
        }
    }
}
```

(Python)

```
def fatorial(n: int):
    if n < 0:
        return None
    fat = 1
    for i in range(2, n+1):
        fat *= i
```

```
    return fat

while True:
    print("\n=== MENU (Python) ===")
    print("1) Quadrado de um numero")
    print("2) Fatorial de um numero")
    print("3) Sair")
    opcao = input("Opcao: ").strip()

    if opcao == "1":
        try:
            x = float(input("Digite um numero: "))
            print(f"Quadrado: {x * x:.2f}")
        except ValueError:
            print("Entrada invalida.")
    elif opcao == "2":
        try:
            n = int(input("Digite um inteiro >= 0: "))
            r = fatorial(n)
            print("Valor invalido." if r is None else f"{n}! = {r}")
        except ValueError:
            print("Entrada invalida.")
    elif opcao == "3":
        print("Saindo...")
        break
    else:
        print("Opcao invalida.")
```

Resposta:

O código com goto funciona, mas é mais difícil de entender porque tem saltos e rótulos. O while já mostra a condição do laço e fica mais claro. Para contar de 1 a 10, o for é o mais simples, pois já tem início, fim e passo na mesma linha. Ordem de legibilidade: for > while > goto.

Depois de executar os dois, em C com switch/case dá certo, mas precisa de mais linhas e cuidado com scanf/printf. Em Python com if/elif/else e input()/print() o código fica mais curto e mais fácil de ler. Comentário final: para esse menu, Python foi mais simples de implementar porque tem menos “cerimônia” e não precisa compilar; em C é mais verboso.

3-

```
from typing import Iterable

def processar(nums: Iterable[int]) -> int | None:
    for n in nums:
        if n == 0:
            break
        if n < 0:
            continue
        if n % 2 == 0:
            return n * 2
    return None
```

Resposta:

Com goto, você dependeria de rótulos e saltos manuais para simular “parar”, “pular” e “sair com retorno”. break/continue/return comunicam intenção de forma direta, reduzem “malabarismo de fluxo” e tornam o código mais legível e seguro.