N_ALG PROG_A4 - Texto de apoio

Site: EAD Mackenzie
Tema: ALGORITMOS E PROGRAMAÇÃO I {TURMA 01D} 2023/1
Livro: N_ALG PROG_A4 - Texto de apoio

Impresso por: CAIO FRESSATTI PINHEIRO .
Data: domingo, 5 fev 2023, 02:02

Índice

1. ESTRUTURAS CONDICIONAIS ENCADEADAS

A linguagem Python nos fornece duas formas de estruturas com múltiplos possibilidades:

- Estrutura com múltiplas condições aninhadas ou encadeadas if-else
- Estrutura com uma única condição if e uso de múltiplas cláusulas elif

Vimos a primeira forma com múltiplas condições aninhadas com if-else. O Python apresenta uma alternativa ao problema dos múltiplos ifs aninhados. A cláusula elif substitui um par de else-if, mas sem criar outro nível de estrutura, evitando problemas de indentações.

Sintaxe:

if <condição 1>:

    <bloco de código if>

elif <condição 2>:

    <bloco de código elif>

else:

    <bloco de código else>

Se condição 1 for verdadeira, executa bloco de código if. Senão, se condição 2 for verdadeira, executa bloco de código elif. Caso contrário, executa bloco de código else.

A declaração elif só pode existir se existir uma declaração if.

A declaração do elif é opcional; pode existir if sem um elif (decisão simples) else também continua sendo opcional.

A declaração elif tem um teste lógico assim como if; ela é executada sempre que o teste do if resultar em False.

A indentação do elif deve ser a mesma da declaração if que está relacionada; if e seu elif devem ter a mesma quantidade de espaços à esquerda.

Na declaração if-elif-else, podem existir quantas declarações elif forem necessárias.

Exemplo:

Veja este código usando apenas if/else

```
if nota >= 7.5 and frequencia >=0.75:
    print("Aprovado direto")
else:
    if nota >= 6.0 and frequencia >=0.75:
        print("Aprovado com Exame")
    else:
        print("Reprovado")
```

Agora veja o mesmo código utilizando if/else/elif

```
if nota >=7.5 and frequencia >=0.75:
    print("Aprovado Direto")
elif nota >=6.0 and frequencia >=0.75:
    print("Aprovado com Exame")
else:
    print("Reprovado")
```

Observe que o else: if condição: do primeiro código foi substituído por elif condição: no segundo código.

2. ESTRUTURA CONDICIONAL COM INTERVALO DE VALORES

Nos programas que necessitam de vários testes condicionais para executar a saída esperada pelo usuário, dependendo da condição, deve-se levar em consideração que, quando uma condição escrita no elif for testado, isto é consequência da condição anterior ter retornado um resultado False, é comum programadores iniciantes criarem testes condicionais desnecessários, pois não levam esta questão em consideração.

Veja o seguinte problema:

Faça um programa que mostre a letra da grade correspondente dependendo do valor de entrada, conforme tabela abaixo:

VALOR DE ENTRADA	SAÍDA ESPERADA
Maior ou igual a 90	Grade A
Entre 80 e 89	Grade B
Entre 70 e 79	Grade C
Entre 69 e 60	Grade D
Abaixo de 60	Grade E

Um programador inexperiente poderia criar em seu código testes de condições desnecessárias, apesar de obter a saída esperada.

Veja o código abaixo:

```
grade = int(input('Digite o valor: '))
if grade>=90:
    print('Grade A')
elif grade<90 and grade>=80:
    print('Grade B')
elif grade<80 and grade>=70:
    print('Grade C')
elif grade<70 and grade>=60:
    print('Grade D')
elif grade <60:
    print('Grade E')
```

Observe que o código resolve corretamente o problema e que os intervalos descritos na tabela estão corretamente contemplados, porém há diversos testes desnecessários.

Observe: se o teste grade>=90 for False, significa que o valor armazenado na variável grade é menor que 90; dessa forma, é totalmente desnecessário no primeiro elif escrever a condição grade<90, pois já sabemos que isso é verdadeiro.

O mesmo acontece em cada um dos elifs abaixo, sendo inclusive totalmente desnecessário testar a última condição se a grade é menor que 60, pois – se a execução chegar nesse teste – é porque todas as condições acima foram falsas, portanto, o valor armazenado em grade será menor que 60.

Vamos resolver esse mesmo problema com um código mais enxuto?

```
grade = int(input('Digite o valor: '))
if grade>=90:
    print('Grade A')
elif grade>=80:
    print('Grade B')
elif grade>=70:
    print('Grade C')
elif grade>=60:
    print('Grade D')
else:
    print('Grade E')
```

Observe que o código acima tem a mesma saída de dados, porém ele é mais legível e com um desempenho melhor, uma vez que faz um número menor de testes de suas condições.

Dessa forma, concluímos que não basta resolver o problema, é preciso uma lógica apurada para criar uma boa solução para um problema: chamamos isso de código elegante.

Consegue perceber como nosso código ficou bem mais elegante?

Outra situação que merece reflexão é quando repetimos a mesma instrução, ou bloco de instruções, em cada uma das estruturas condicionais que são verdadeiras (True).

É comum encontrarmos um código que repete o mesmo bloco de instruções a cada condição; podemos verificar que esse código poderia ser menor, principalmente quando olhamos para sua leitura.

Pense no seguinte problema:

Receba o saldo em conta corrente de um correntista, calcule e mostre o valor de crédito, dependendo do saldo informado conforme tabela abaixo:

SALDO	% DE CRÉDITO
Acima de R$ 4000,00	30% do saldo
Entre R$ 4000,00 e R$ 3000,00	25% do saldo
Entre R$ 2999,00 e R$ 2000,00	20% do saldo
Abaixo de R$ 2000,00	10% do saldo

Observe o código abaixo, no qual já usamos apenas os testes condicionais necessários, porém há uma repetição de instruções em cada bloco.

```
saldo = float(input('Saldo médio: '))
if saldo>4000:
    credito = saldo*0.30
    print(f'Crédito = {credito}')
elif saldo>=3000:
    credito = saldo*0.25
    print(f'Crédito = {credito}')
elif saldo>=2000:
    credito = saldo*0.20
    print(f'Crédito = {credito}')
else:
    credito = saldo*0.10
    print(f'Crédito = {credito}')
```

Esse código ficaria mais legível e mais fácil de fazer manutenção na versão abaixo, na qual a saída de dados – que é igual para todas as situações condicionais – foi colocada apenas no final do encadeamento das condições.

```
saldo = float(input('Saldo médio: '))
if saldo>4000:
    credito = saldo*0.30
elif saldo>=3000:
    credito = saldo*0.25
elif saldo>=2000:
    credito = saldo*0.20
else:
    credito = saldo*0.10
print(f'Crédito = {credito}')
```

Outras versões poderiam ser pensadas nesse contexto de legibilidade e manutenibilidade, dependendo do programador.

Com o cálculo fora da estrutura condicional encadeada:

File Edit Format Run Options Window Help

```
saldo = float(input('Saldo médio: '))
if saldo>4000:
    credito = 0.30
elif saldo>=3000:
    credito = 0.25
elif saldo>=2000:
    credito = 0.20
else:
    credito = 0.10
credito= saldo*credito
print(f'Crédito = {credito}')
```

Com o cálculo na saída de dados:

```
saldo = float(input('Saldo médio: '))
if saldo>4000:
    credito = 0.30
elif saldo>=3000:
    credito = 0.25
elif saldo>=2000:
    credito = 0.20
else:
    credito = 0.10
print(f'Crédito = {saldo*credito}')
```

Observe, nas versões acima, a importância da indentação, para que o compilador identifique quais são as instruções que serão executadas dentro do encadeamento de condições e quais instruções serão executadas ao sair desse encadeamento.

Há situações que, dentro de uma estrutura condicional encadeada, faz-se necessário o uso de uma estrutura condicional composta. Veja o problema a seguir:

Leia a nota e o número de faltas de um aluno e escreva seu conceito. De acordo com a tabela abaixo, quando o aluno tem mais de 20 faltas, ocorre uma redução de conceito:

NOTA	CONCEITO (ATÉ 20 FALTAS)	CONCEITO (MAIS DE 20 FALTAS)
9.0 até 10	A	B
7.5 até 8.9	B	C
5.0 até 7.4	C	D
4.0 até 4.9	D	E
0.0 até 3.9	E	E

Veja uma proposta de solução para esse problema:

```
nota = float(input('Nota:'))
falta = int(input('Faltas:'))
if nota>=0 and nota <=3.9:
    conceito = 'E'
elif nota <=4.9:
    if falta<=20:
        conceito = 'D'
    else:
        conceito = 'E'
elif nota<=7.4:
    if falta<=20:
        conceito = 'C'
    else:
        conceito = 'D'
elif nota<=8.9:
    if falta<=20:
        conceito = 'B'
    else:
        conceito = 'C'
else:
    if falta<=20:
        conceito = 'A'
    else:
        conceito = 'B'
print(conceito)
```

Perceba que a forma pela qual as estruturas condicionais simples, composta e encadeadas serão usadas depende da lógica que o programador usará para resolver o problema.

Certamente a solução que trouxer um número menor de testes condicionais será executada mais rapidamente, e a solução que não tiver instruções desnecessárias será mais fácil de entender para quem ler o código.

A escrita de códigos elegantes envolve lógica, experiência e domínio da linguagem.

Vejamos um último exemplo, agora usando a biblioteca random para gerar números aleatórios.

O jogo do par ou ímpar é usado quando duas pessoas precisam decidir um impasse. Par ou ímpar é um jogo entre duas pessoas cujo objetivo geralmente é resolver aleatoriamente um impasse. Os participantes apostam em par e ímpar. Depois disso, ambos mostram as mãos escondendo alguns dedos, contam-se os dedos e vence quem tiver acertado a paridade do número de dedos.

Aqui, jogaremos com a máquina: o computador escolherá um número aleatório entre 0 e 10 e o jogador aposta em par ou ímpar e informa um número correspondente a quantidade de dedos que ele lançaria.

Quem será o vencedor?

```
#Jogo par ou impar
#Jogador escolher par ou impar e apostar o número (10 dedos)
import random
escolha = input("Par ou impar?")
apostaJ = int (input("Quantos dedos vc vai lançar?"))
apostaC = random.randint(0,10)
print ("Computador : ", apostaC)
soma = apostaJ + apostaC
if soma % 2 == 0 and escolha == 'par':
    print ("Você ganhou")
elif soma % 2 !=0 and escolha == 'impar':
    print ("Você ganhou")
else:
    print ("Você perdeu")
```

3. REFERÊNCIAS

ASCENCIO, A. F. G.; CAMPOS, E. A. V. *Fundamentos da Programação de Computadores*: Algoritmos, Pascal, C/C++ e Java. 3. ed. São Paulo: Pearson Prentice Hall, 2012. ISBN 9788564574168

DIERBACH, C. *Introduction to Computer Science Using Python*: A Computational Problem Solving Focus. New York: Wiley, 2012.

MENEZES, N. N. C. *Introdução à Programação com Python*: algoritmos e lógica de programação para iniciantes. São Paulo: Novatec, 2014.