Universidade Federal do Espírito Santo

Departamento de Computação e Eletrônica - CEUNES
Programação I / Programação Funcional
Prof. Oberlan Romão

Aula Prática 3

O objetivo dessa aula é colocar em prática tudo que já foi visto e, em especial, o uso de `if-elif-else`. Além disso, aprenderemos as definições de alguns tipos de variáveis que iremos utilizar no decorrer da disciplina.

Atenção: Alguns dos arquivos .py estão disponíveis no AVA, baixem-os para facilitar a implementação dos exercícios.

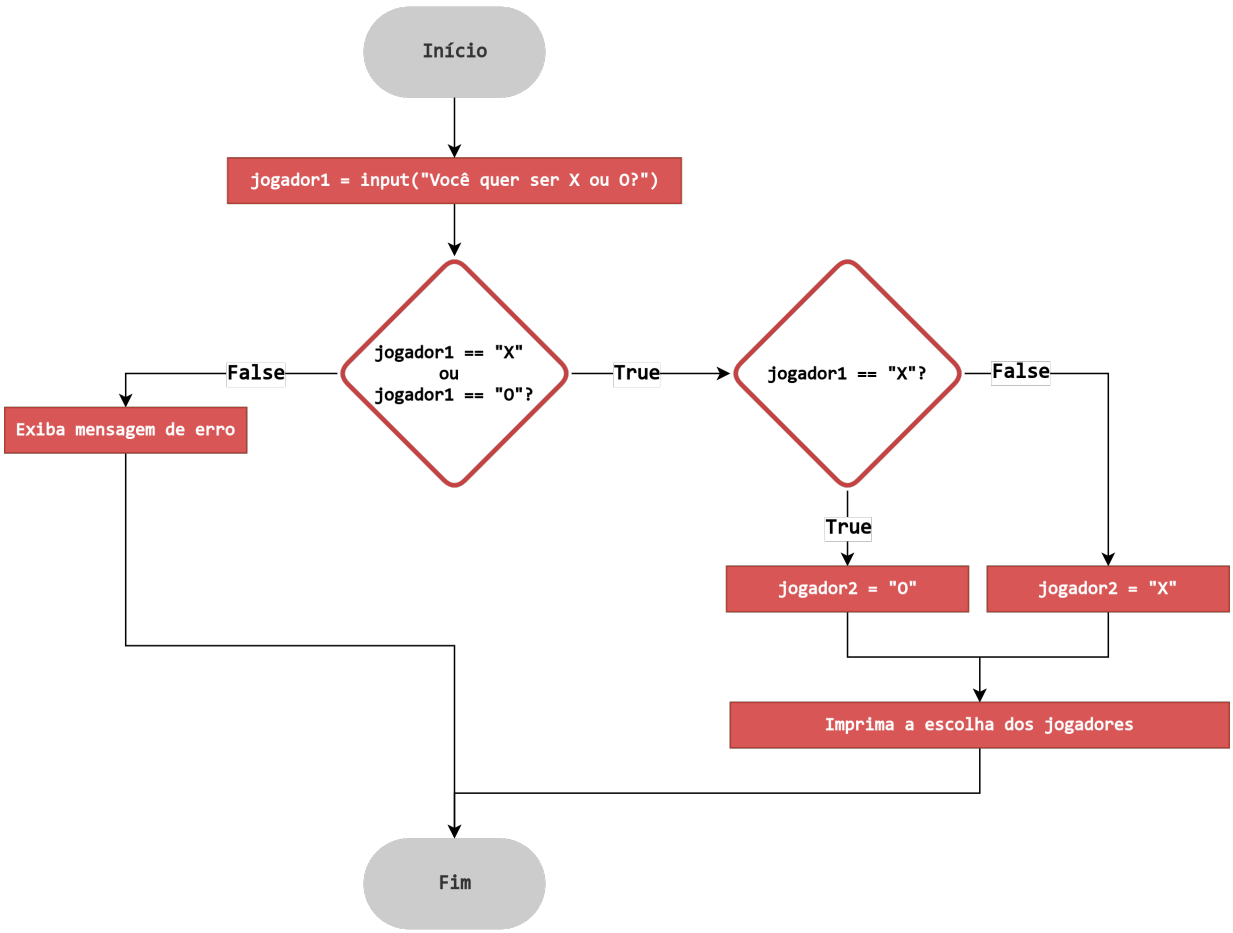
Vamos começar fazendo alguns exercícios:

1. (`ex1.py`) Reescreva o código abaixo usando as instruções `if-elif-else`. Além disso, caso o valor digitado para a média não esteja entre 0.0 e 10.0 exiba uma mensagem de erro.

```
media = float(input("Digite a média do aluno: "))
if media >= 9.0:
    print("A")
else:
    if media >= 8.0:
        print("B")
    else:
        if media >= 7.0:
            print("C")
        else:
            if media >= 6.0:
                print("D")
            else:
                print("R")
```

2. (`ex2.py`) O código presente no arquivo `ex2.py` contém parte de um sistema onde o usuário deve escolher um produto (pelo código) e o sistema deve exibir o nome e o preço do produto escolhido. Caso seja digitado um código diferente dos que estão listados, seu programa deve exibir uma mensagem de erro. Sua tarefa é completar o código.

3. (`ex3.py`) O jogo da velha é um jogo muito popular e com regras extremamente simples, que não traz grandes dificuldades para seus jogadores e é facilmente aprendido. O jogo é normalmente jogado com duas pessoas, onde um jogador é `X` e o outro jogador é `O`. Assim, se o `jogador1` escolher `X`, automaticamente, o `jogador2` será `O`, e vice-versa. O fluxograma abaixo apresenta uma estratégia para resolver esse problema.



a) Baseado nesse fluxograma, implemente um programa para definir o valor de `jogador2` de acordo com a escolha do `jogador1`.

b) Essa implementação, que segue a abordagem do fluxograma, possui uma deficiência: encerra o programa se o usuário digitar algo diferente de `X` ou `O`. Em uma aplicação/jogo real, isso não faz muito sentido. Por quê? Observe o fluxograma e **pense** no que deveria ser mudado **no fluxograma** para que o programa continue sendo executado até que seja escolhido `X` ou `O`, ou seja, enquanto o usuário não escolher `X` ou `O` o programa não avança.

4. (`ex4.py`) Um número natural é chamado de *ascendente* se cada um dos seus algarismos é estritamente maior do que qualquer um dos algarismos colocados à sua esquerda. Por exemplo, o número 358 ($3 < 5 < 8$) é ascendente. Faça um programa que leia **um número inteiro** n no intervalo $[100, 999]$ e diga se n é ou não ascendente. Além disso, caso o número esteja fora do intervalo $[100, 999]$, exiba uma mensagem de erro. Obs.: você deve utilizar operadores matemáticos, ou seja, você não pode considerar o valor como uma `string`. Pense em como separar cada dígito para fazer as comparações.

Operadores de atribuição

Como já sabemos, os operadores de atribuição são usados para atribuir valores a variáveis. Por exemplo, `a = 5` é um exemplo de utilização do operador de atribuição, que atribui o valor 5 à variável `a`. Além disso, em Python, é possível utilizar uma forma compacta para atualizar o valor de uma variável:

Operador	Exemplo	Equivalente a
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5

Variáveis contadoras e acumuladoras[1]

Em situações onde precisamos realizar contagens de ocorrências, ou somatórios/produtórios de valores dentro de um conjunto de dados, devemos utilizar variáveis específicas para fazer o armazenamento dos resultados. Chamamos de contadoras as variáveis que realizam a contagem de ocorrências de um determinado valor (ou evento/situação) e de acumuladoras as variáveis responsáveis por armazenar os resultados de somatórios e produtórios de valores.

Variáveis contadoras

As variáveis contadoras são normalmente inicializados com valor 0 e incrementados em 1 cada vez que uma nova ocorrência (ou evento) acontece. Por exemplo, considere que tenhamos um conjunto de dados com informações (idade e sexo) de 50 pessoas e desejemos saber quantas dessas pessoas são do sexo feminino e possuem 18 anos ou mais. Para isso, precisamos de um contador para armazenar a quantidade de ocorrências da condição definida no enunciado. Esse contador deve ser inicializado com 0 e incrementado em 1 sempre que o sexo de uma dada pessoa for feminino e sua idade for maior ou igual a 18.

Vamos a mais um exemplo, suponha que precisamos ler quatro número inteiros e contabilizar quantos desses número são pares. Nesse caso, podemos criar uma variável `contaPar` inicializada com 0 e, a cada leitura de dado, verificar se o valor lido é par ou não. Se for, incrementamos a variável `contaPar` em uma unidade, como mostra o código abaixo (`ex5.py`):

```
contaPar = 0
x = int(input("Digite um número: "))
if x%2 == 0:
    contaPar += 1 # contaPar = contaPar +1
x = int(input("Digite um numero: "))
if x%2 == 0:
    contaPar += 1
x = int(input("Digite um numero: "))
if x%2 == 0:
    contaPar += 1
x = int(input("Digite um numero: "))
if x%2 == 0:
    contaPar += 1
print(f"Foram digitados {contaPar} números pares")
```

[1]Fonte: http://www.cristiancechinel.pro.br/my_files/algorithms/bookhtml/node60.html

Pergunta: Faz sentido `contaPar` começar com um valor diferente de zero? Qual deve ser a saída do programa se todos os números lidos forem impares?
Exercício: Modifique o código anterior para que o mesmo contabilize também a quantidade de números impares lidos.

Variáveis acumuladoras

Como comentado anteriormente, as variáveis acumuladoras são utilizadas em dois tipos de situações, para a realização de somatórios e de produtórios. No caso dos somatórios, a variável é normalmente inicializada com o valor 0 e incrementado no valor de um outro termo qualquer, dependendo do problema em questão.

Considere que no problema anterior, além calcular a quantidade de números pares, também desejamos calcular a soma desses números. Nesse caso, precisamos inserir no código um acumulador, que deve ser inicializado em 0 e incrementado no valor do número lido. Note que, dessa forma, podemos facilmente calcular a média dos valores pares lidos. Veja o código abaixo (`ex6.py`):

```
contaPar = 0
somaPar = 0
x = int(input("Digite um numero: "))
if x%2 == 0:
  contaPar += 1 #contPar = contPar + 1
  somaPar += x #somaPar = somaPar + x
x = int(input("Digite um numero: "))
if x%2 == 0:
  contaPar += 1
  somaPar += x
x = int(input("Digite um numero: "))
if x%2 == 0:
  contaPar += 1
  somaPar += x
x = int(input("Digite um numero: "))
if x%2 == 0:
  contaPar += 1
  somaPar += x
print(f"Foram digitados {contaPar} números pares")
print(f"Soma dos números pares: {somaPar}")
if contaPar > 0:
    print(f"Média dos números pares: {somaPar/contaPar}")
```

Exercício: Modifique o código anterior para que o mesmo contabilize também a quantidade e a média dos números impares lidos.

No caso de utilizarmos variáveis acumuladoras para armazenar produtórios é necessário a inicialização da mesma com o valor neutro da multiplicação (o número 1). A cada iteração a variável é então multiplicado por um outro valor qualquer, dependendo do problema em questão.
Exercício: Modifique novamente o código anterior para que seja impresso também a multiplicação dos valores pares e impares lidos.

Variáveis *flag*

Em algumas situações, precisamos usar uma variável para armazenar um estado do programa. Por exemplo, podemos criar uma variável para indicar se um sistema está funcionando corretamente (ou se apresentou alguma falha) ou se foi digitado, em todas as entradas, valores válidos. Normalmente inicializamos esta variável com um valor padrão (por exemplo, `True`) e atualizamos a variável caso uma mudança de estado ocorra (trocando o valor, por exemplo, para `False`).

Este tipo de variável, que serve para sinalizar uma situação específica, é chamada de *flag*. Uma variável *flag* pode simplificar significativamente a escrita, manutenção e o entendimento de um programa. Vamos a um exemplo hipotético, onde devemos verificar se um programa está funcionando corretamente, ou seja, se nenhuma falha ocorreu. O código abaixo exibe uma solução sem o uso de uma variável *flag*.

```
1  ...
2  if <condição1>:
3      print("Falha do tipo 1")
4  if <condição2>:
5      print("Falha do tipo 2")
6  if <condição3>:
7      print("Falha do tipo 3")
8  ...
9  if <condição100>:
10     print("Falha do tipo 100")
11 ...
12 if not(<condição1>) and not(<condição2>) ... not(<condição100>):
13     print("Sistema funcionando normalmente")
```

Observe que para sabermos se o sistema está funcionando corretamente, o último `if` (linha 12) ficou extremamente grande e a chance de cometermos algum erro (por exemplo, esquecermos de chegar alguma condição) é muito alta. Vamos ver como podemos simplificar esse `if` usando uma variável *flag*.

```
1  semErro = True
2  ...
3  if <condição1>:
4      print("Falha do tipo 1")
5      semErro = False
6  if <condição2>:
7      print("Falha do tipo 2")
8      semErro = False
9  ...
10 if <condição100>:
11     print("Falha do tipo 100")
12     semErro = False
13 ...
14 if semErro:
15     print("Sistema funcionando normalmente")
```

Exercício: Refaça o exercício 2 (`ex2.py`), para que o mesmo utilize uma variável *flag* para verificar se o usuário digitou um código válido.