Introdução ao comando de seleção e às expressões lógicas

[Expressão em Python | Expressão em Python | Operadores | Seleção | Seleções | Exemplo | Par ou ímpar]

Como ler este texto. Aqui procuro explicar os fundamentos do comando de seleção `if-else` para a linguagem *Python*, explicando *expressões lógicas* e também *aninhamento* de comandos `if-else`. Este texto foi produzido principalmente para *Python* versão 3, mas ao final tem uma explicação sobre a diferença do operador de divisão para o *Python 2*.

O que é um comando de seleção?

A existência de *comando de seleção* é essencial para a construção de *algoritmos flexíveis* no sentido de poder seguir uma sequência de instruções distintas de acordo com uma condição. Um exemplo simples poderia ser um algoritmo para computar a soma de todas as entradas de um extrato bancário correspondente aos créditos separadas dos lançamentos de débito. Desse modo, para cada entrada do extrato, deve-se verificar se é positiva, nesse caso acumula em uma variável para *crédito*, senão deve-se acumular em uma variável para débito. Ou seja, de acordo com o resultado da avaliação de uma **expressão lógica**, deve-se seguir determinado **fluxo de execução** (um **"caminho" de instruções**).
Qualquer *linguagem de programação* dispõe de um comando para *desvio de fluxo*, geralmente esse tipo de comando é demominado comando **`if-else`**. Outra característica comum das linguagens de programação é possibilitar definir **dois blocos** de comandos subordinados ao `if-else`, um deles devendo ser executado sempre que a condição resultar *verdadeira* e o outro, que é opcional, devendo ser executado sempre que resultar *falso*.

A figura 1 ilustra as duas possibilidades de comando `if-else`, a figura 1.a apresenta um comando de *seleção* apenas com o **bloco "*verdadeiro*"**, enquanto a 1.b ilustra um comando de *seleção* com os dois blocos.


Condição Verdadeira?
Não
Sim
Bloco de comandos a serem executados sob condição Verdadeira

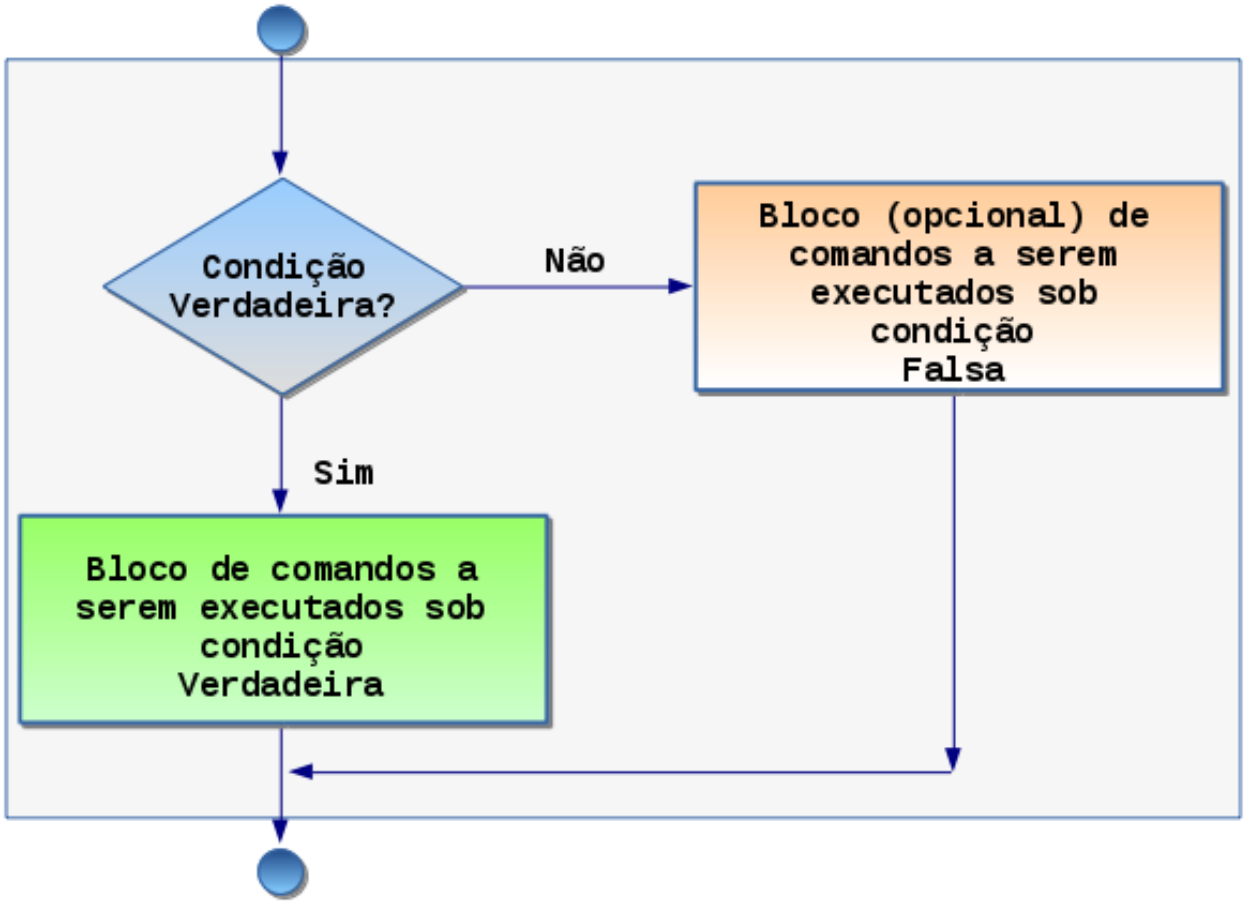

Fig. 1. Diagrama de fluxo de execução (a) sem o ramo de senão e (b) com o ramo de senão ..

Na figura 1.a, se o resultado da **condição lógica** for **verdadeiro**, segue o caminho indicado pela seta com rótulo **sim** e se **falso** segue o caminho da seta com rótulo **não**, que é a próxima instrução após o *bloco "verdadeiro"*.
Na figura 1.b, está o esquema do `if-else` com os dois blocos: se o resultado da **condição lógica** for **verdadeiro**, segue o caminho para o *bloco "verdadeiro"* (verde), mas se resultar **falso**, segue para o *bloco "falso"* (laranja).

De modo esquemático, o comando é do tipo `if (EXPRLOG) blocoV else blocoF`. Assim, devemos destacar que nos dois blocos do comando `if-else` pode haver outros comando `if-else`, como examinado na seção sobre comandos `if` aninhados.

Expressão lógica em *Python*

Na linguagem *Python* existe um tipo específico para resultados de *expressões lógicas*, que o tipo booleano. Assim, uma variável *boleana* pode ter apenas um dos valores dois seguintes valores: *True* ou *False*. Então, para definir uma variável *boleana*, basta usar como *lado direito* da atribuição uma *expressão lógica*. Desse modo, a expressão lógica mais simples é o valor `True`, ou o valor `False`.

Já uma *expressão lógica* não trivial deve envolver duas *expressões aritméticas* conectadas por um *operador relacional* (`EL = EA1 <= EA2`) ou duas outras *expressões lógicas* conectadas por um *operador lógico* (`EL = EL1 and EL2`). Os operadores *relacionais* e *lógicos* são explicados na seção sobre operadores relacionais e lógicos.

```
if (i!=N) : printf("%d == %d\n" % (i, N)); while (i < N) : printf("i = %d\n" % i); i += 1;
```

Cód. 1. Exemplos de expressão lógica em Python.

No código 1 existem duas expressões lógicas, uma subordinada a um comando `if-else` e a outra subordinada a um comando `while`.

Mas vale a pena notar que a linguagem *Python* dispõe de um tipo específico para resultados de *expressões lógicas*, examinado na seção expressão e valor lógico.

Negação em *Python*. Pode-se usar o operador `not` (não) para *negar* o resultado da expressão lógica (que esteja à sua direita). Por exemplo, os seguintes trechos de código são equivalentes em *Python*:

```
if (a==b) : printf("OK");
else : printf("NAO");
```

```
if (not(a-b)) : printf("OK");
else : printf("NAO");
```

Cód. 2. Exemplos de uso de negação em expressão lógica em Pyton.

Podemos explorar equivalências para entender a razão dos dois códigos serem equivalentes. Observe que

!(a-b) é a mesma coisa que **negar** que *a-b* seja **não nulo**, ou seja, *!(a-b != 0)* e,
!(a-b != 0) é a mesma coisa que afirmar que *a-b == 0*, e
a-b == 0 é a mesma coisa que *a == b*.

Expressão e valor lógicos em *Python*

Em *Python* existem um particular tipo de variável que é *booleano*, sendo que uma variável boleana pode ter apenas um dos valores: *True* ou *False*. Portanto, é possível definir variáveis que recebam o resultado de uma *expressão lógica*, que deve ser *True* ou *False*.

Assim, se em determinado código for necessário manter o resultado de uma expressão lógica e depois usar esse resultado em um comando `if-else`, seria possível fazer de três formas distintas, sendo a mais à esquerda a mais eficiente:

```
varlog = a == b # ou:
varlog = (a == b)
if (varlog) : printf("
OK")
else : printf("NAO")
```

```
varlog = (a ==
b) # ou: varlog =
a == b
if (varlog ==
True) : printf("O
K")
else : printf("NA
O")
```

```
varlog = (a == b) #
ou: varlog = a == b
if (not (varlog ==
False)) : printf("O
K")
else : printf("NAO"
)
```

Cód. 3. Exemplos de uso de negação em expressão lógica em Python.

Note que no exemplo acima mais à direita usamos `not (varlog == False)` que equivale a `varlog == True`, que portanto também equivale a `varlog` (pois está essa armazena valor *booleano*).

Operadores relacionais e lógicos em *Python*

Os **operadores relacionais** (*relações entre expressões aritméticas*) em *Python* são os seguinte:

Tab. 1. Tabela dos operadores relacionais.

Operação	Operador	Exemplo
iguais	==	n1 == 4
diferentes	!=	n1 != 4
maior	>	n1 > n2
menor	<	n2 < n1
maior ou igual	>=	n1 >= 3
menor ou igual	<=	n1 <= 33

Os **operadores lógicos** principais em *Python* são

Tab. 2. Tabela dos operadores lógicos em Python.

Operação	Operador	Exemplo
conjunção (`e`)	and	(n1 > 1 and n1 < 10)
disjunção (`ou`)	or	(n1 > 1) or (n1 < 10)
negação (`não`)	not	not(n1 > 1 or n1 < 10)

Comando de seleção `if-else` em *Python*

O **comando de seleção** é essencial para qualquer *software*, é ele que permite que o programa desvie seu fluxo de execução de acordo com os valores dos dados (as já citadas *variáveis*). Ele está presente em qualquer linguagem de programação e está baseado no cômputo de uma *expressão lógica* (**EXPLOG**), se o resultado dela for *verdadeiro*, o fluxo segue por um "caminho", senão segue por outro. Essa estrutura está ilustrada na figura abaixo:

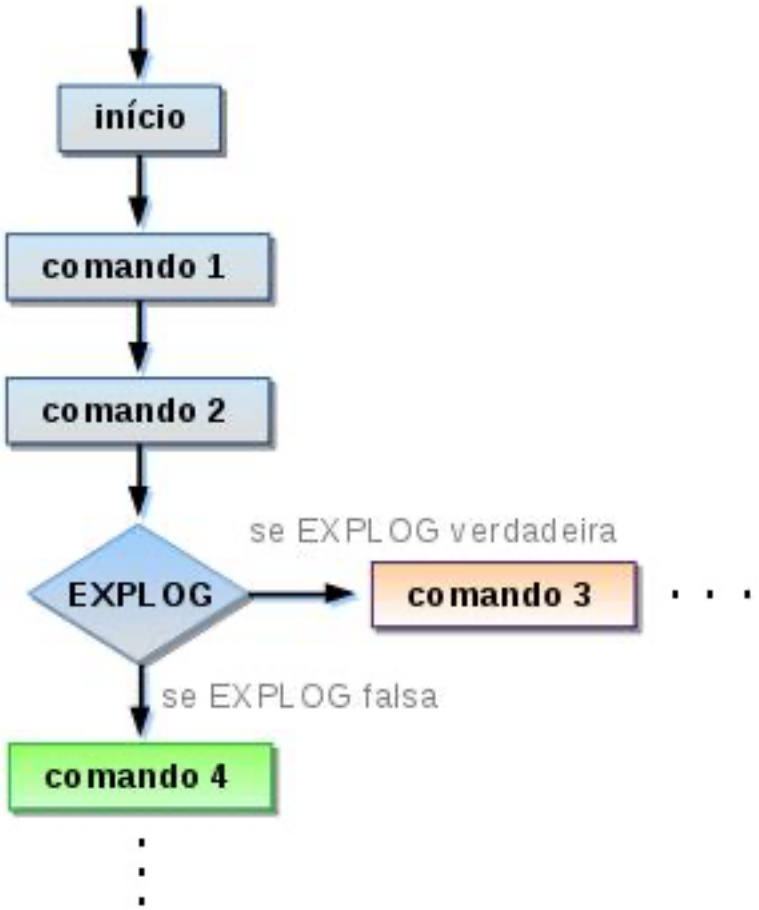

Fig. 2. Diagrama de fluxo de execucação do comando de seleção.

Existem outras duas características de comando de seleção (em qualquer linguagem) que são:

1. permitir a execução de comandos mutuamente exclusivo, ou seja, se a expressão lógica resultar verdadeira executa-se determinado comando, senão executa-se outro comando ("comando alternativo") - mas a existência do "senão" é opcional;
2. possível subordinar mais de uma instrução ao comando "se" ou ao comando "senão", mas cada linguagem tem a sua sintaxe particular para isso (alinhamento em *Python*).

A linguagem *Python*, assim como várias outras, usam as masmas *palavras reservadas* (palavras que *não* podem ser usados como nomes para variáveis): as palavras `if` e o `else`. Sua sintaxe é ilustrada na tabela abaixo.

Além disso, as linguagens permitem que vários comandos (**bloco de comandos**) sejam subordinados à uma seleção. Em *Python* isso é feito com o deslocamento horizontal de linhas, ou seja, com "**indentação**".

A tabela a seguir ilustra a sintaxe do comando de seleção em *Python*:

Tab. 3. Comandos de seleção em Python.

Apenas 1 comando	Vários comandos

`if (EXPR) : comando1;`
`else : comando2;`
	`if (EXPR) :`
	`  comando1;`
	`  comando2;`
	`else :`
	`  comando3;`
	`  comando4;`

Vale destacar que o comando `else` é opcional (como já citado), ou seja, não existindo ação a ser executada caso a condição seja falsa, não é necessário usar um `else` com algum comando. Assim é correto em *Python* escrever: `if (1==1) : print("ok")`

Em *Python* os parênteses não são obrigatórios em expressões, quer dizer é válido redigir o comando `if n1==n2 : comando1` (assim como `if (n1==n2) : comando1`). Deste modo para o interpretador *Python* conseguir traduzir corretamente o comando, lançou-se mão dos dois pontos para indicar final de expressão (`if EXPR :`) e no `else`, por uniformidade, também exige-se `:` (ou seja, deve-se usar `else :`).

Experimente! Construa um exemplo com 2 comandos de "impressão" subordinados ao `if`, depois experimente remover as "indentações" dos 2 comandos e tente rodar o programa. Você receberá uma mensagem de erro.

Comandos de seleção `if-else` aninhados em *Python*

Em geral, toda linguagem de programação de alto-nível permite que os comandos `if-else` sejam utilizados de modo **aninhado**, ou seja, qualquer dos comandos acima (`comando1`... `comando4`) podem ser um novo `if-else`. Por exemplo:

Tab. 4. Comandos de seleção aninhados em Python.

Linguagem	if aninhado

Python	`if (a<b) : # este exemplo nao trata todos os casos...`
	`  if (b<c) : print("O menor e': %d" % a);`
	`elif (b>=c) : # usar "elif" para economizar "indentacao"`
	`  print("O maior e': %d" % a);`

O comando elif é a junção de um `else` seguido de um `if`, ou seja, as duas formas na tabela abaixo produzem o mesmo resultado (mas com `elif` fica mais compacta a descrição).

Tab. 5. Reescrevendo comando `elif` com `else-if`.

Com `elif`	Com `else-if`
`if (a!=b) :` `  print("diferem");`	`if (a!=b) :` `  print("diferem");`

```
elif (c==d) :
  print("tudo igual")
```

```
else :
  if (c==d) :
    print("tudo igual");
```

Note que não deve existir `else` sem um `if` e a associação entre eles é definida pela indentação. Por exemplo, na tabela 4, o comando `elif` (que é um `else` seguido de um `if`) está associado ao primeiro `if` e não ao segundo (ou seja, se `if (a<b)` resultar falso, o comando `if (b < c)` é ignorado e o comando `elif (b >= c)` é executado).

Um exemplo para decidir se uma variável inteira armazena um valor par ou ímpar

Utilizando a divisão inteira (quer dizer `7/2` resultando `3`) pode-se conseguir determinar se um inteiro é ou não par. Por exemplo, dispondo de uma variável inteira `n1`, pode-se fazer a divisão inteira por *2* e multiplicar seu resultado também por *2*, se recuperar o valor integral de `n1`, então esse é um valor par (e.g. *7/2* é *3* que multiplicado por *2* resulta "apenas" *6*, concluindo daí que *7* não é par - vide explicação estendida).

Tab. 6. Exemplo de expressão aritmética em expressão lógica para decidir se valor é par em Python.

Código *Python 3*
if (n1//2*2==n1) : ** printf(n1, " eh par\n")** **else :** ** printf(n1, " eh impar\n")**

Mas pode-se obter o mesmo resultado utilizando o operador binário `%` que é "resto da divisão inteira". Este operador devolve o resto da divisão, por exemplo, `7%2` resulta `1` (o quociente é `3`: `3 * 2 + 1 = 7`). Assim o código fica:

Tab. 7. Exemplos de comandos para decidir se valor é ímpar ou par em Python.

Código *Python*

if (n1%2==0) :
  print("%d eh par" % n1)
else :
  print("%d eh impar" % n1)

Mas por que razão "n1//2*2==n1" é suficiente para decidir se n1 é ou não par?

Bem, isso é um *truque computacional* baseado na divisão inteira (quociente), como examinamos ao final da seção sobre variáveis e expressões.

Mas vale destacar que as versões 2 e 3 do *Python* tem comportamento distinto quanto ao operador de divisão. No *Python 2*, `7/2` e `7//2` produzem o mesmo resultado `3`), entretanto,

no *Python 3*, o primeiro caso sempre como divisão "real" e o segundo é o quociente da divisão inteira, ou seja, no *Python 3*: `7/2` resulta *3.5* enquanto `7//2` resulta *3*.
Experimento em seu ambiente *Python* o código abaixo (o formatador `%s` é para imprimir texto/*string*):

```
print("7/2=%s, 7//2=%s" % (7/2, 7//2))
```

Leônidas de Oliveira Brandão
http://line.ime.usp.br

Alterações