

Operações Lógicas

1. Operações booleanas

Quando estudamos variáveis, vimos que existem alguns tipos primitivos: str (texto), int (número inteiro), float (número real) e bool (lógico). Vimos diversas operações aritméticas também, como a soma, a divisão ou a potência, cujos resultados são int ou float. Porém, podemos ter também operações cujo resultado é bool: são operações lógicas.

1.1. Comparações

Algumas das operações lógicas mais conhecidas são as comparações:

```
comparacao1 = 5 > 3
print(comparacao1)
comparacao2 = 5 < 3
print(comparacao2)
```

Se executarmos o código acima, a saída que teremos na tela será:

```
True
False
```

Isso ocorre porque 5 é maior que 3. Portanto, comparacao1 recebeu uma expressão cujo valor lógico é verdadeiro, portanto seu resultado foi True, e o oposto ocorreu para comparacao2. O Python possui 6 operadores de comparação:

Maior que: >
Maior ou igual: >=
Menor que: <
Menor ou igual: <=
Igual: ==
Diferente: !=

Note que o operador para comparar se 2 valores são iguais é ==, e não =. Isso ocorre porque o operador = é o nosso operador de atribuição: ele diz que a variável à sua esquerda deve receber o valor da expressão à direita. O operador de == irá testar se o valor à sua esquerda é igual ao valor à sua direita e irá responder True ou False, como todos os outros operadores de comparação.

1.2 Negação lógica

Outra operação lógica bastante importante é a negação. Ela inverte o resultado de uma expressão lógica. Caso a expressão resulte em True, a sua negação irá resultar em False, e vice-versa.

A negação em Python é representada pela palavra not. Vamos modificar o exemplo anterior:

```
comparacao1 = not 5 > 3
print(comparacao1)
comparacao2 = not 5 < 3
print(comparacao2)
O resultado será:
```

False

True

Podemos resumir o funcionamento do not utilizando uma tabela-verdade. Nela testamos os diferentes valores possíveis para a entrada e anotamos o resultado para cada conjunto de valores:

A	not A
True	False
False	True

1.2. Conjunção lógica

Em alguns casos precisamos testar se duas ou mais condições são verdadeiras.

Imagine, por exemplo, que o critério de aprovação em uma escola seja a média superior a 6.0 e presença superior a 75%. Neste caso, o aluno precisa atender a ambos os critérios para ser aprovado. Se ele tirou uma ótima nota, mas faltou demais, será reprovado. Se ele compareceu a todas as aulas, mas teve notas baixas, idem.

A conjunção lógica, também conhecida como e lógico, é representada em Python pela palavra and.

O código abaixo testa se é verdade que o aluno foi aprovado:

```
media = float(input('Digite a média do aluno: '))
```

```
presenca = float(input('Digite as presenças do aluno: '))
```

```
aprovado = media >= 6.0 and presenca >= 0.75
```

```
print('O aluno foi aprovado?', aprovado)
```

Execute o código acima e teste algumas combinações diferentes de valores. Note que basta uma das condições ser falsa para que o resultado total seja False.

A tabela-verdade para o e lógico entre duas entradas A e B é:

A	B	A and B
False	False	False
False	True	False
True	False	False
True	True	True

1.3. Disjunção lógica

Nem sempre precisamos que ambas as condições sejam verdadeiras. Vários de nós já nos deparamos com promoções de queima de estoque anunciadas da seguinte maneira:

"promoção válida até o dia 15 deste mês ou enquanto durarem os estoques".

Neste caso, para a promoção acabar, não é necessário que ambas as coisas ocorram (atingir o dia 15 e zerar o estoque). Se ainda temos 10 itens no estoque, mas hoje é dia 16, a promoção acabou. Se hoje é dia 5, mas o estoque está zerado, a promoção acabou.

A disjunção lógica, também chamada de ou lógico, é representada em Python pela palavra or.

O programa abaixo testa se a promoção acabou:

```
dia_final = int(input('Digite o dia do mês para encerrar a promoção: '))
dia_atual = int(input('Digite o dia do mês atual: '))
estoque = int(input('Digite a quantidade de itens no estoque: '))
```

```
acabou = dia_atual > dia_final or estoque == 0
print(acabou)
```

Faça alguns testes com o programa acima e note que basta uma condição ser verdadeira para seu resultado ser True.

A tabela-verdade para o ou lógico é:

A	B	A and B
False	False	False
False	True	True
True	False	True
True	True	True

Resumo:

not: inverte a expressão original

and: verdadeiro apenas se ambas as condições forem verdadeiras

or: falso apenas se ambas as condições forem falsas

2. Valores truthy e falsy

Valores não-booleanos em Python, como inteiros ou strings, podem ser convertidos para booleanos utilizando a função bool, da mesma maneira que utilizamos int e float em exemplos anteriores para converter entradas de string para número.

Certos valores serão convertidos para True, enquanto outros serão convertidos para False. Em certos contextos, como expressões condicionais - que serão estudadas muito em breve - essa conversão ocorre de maneira implícita. Quando um valor pode ser interpretado como True, dizemos que ele é um valor truthy, e quando ele pode ser interpretado como False, ele é conhecido como um valor falsy.

Valores Falsy comuns são:

O valor inteiro 0

O valor real 0.0

Strings vazias (strings com 0 caracteres)

Coleções vazias (listas, tuplas, dicionários etc. com 0 elementos)

Valores Truthy comuns são:

Inteiros diferentes de 0

Reais diferentes de 0.0

Strings contendo ao menos 1 caractere

Coleções (listas, tuplas, dicionários etc.) com pelo menos 1 elemento

A constante None, que representa uma variável "vazia"

Não se preocupe se não estiver familiarizado com todos os dados exemplificados aqui.
Estudaremos cada um deles em outros momentos.