Fundamentos de Programação

Operadores e EXPRESSÕES

Relacionais e Lógicas

Aula 07

Post It da Aula Passada

Operadores Aritméticos

- Unários
  - Inversão de sinal
- Binários
  - Adição e subtração
  - Multiplicação e divisão
  - Potenciação e radiciação

Expressões Aritméticas

- Relacionamento
  - Variáveis, constantes e operadores
- Precedência de operadores
- Associatividade da expressão

Expressões Relacionais

E se, ao invés de um valor numérico como resultado, obtivéssemos um valor lógico?

EXPRESSÕES RELACIONAIS

Expressões relacionais se dedicam a comparar os resultados entre expressões aritméticas, variáveis e constantes, apresentando como resultado um dado do tipo lógico

Operadores Relacionais

Porém, antes de estudarmos as expressões relacionais em detalhes, precisamos conhecer os **operadores relacionais**:

Operador (símbolo)	Descrição
=	Igualdade
>	Maior que
<	Menor que
≥	Maior ou Igual que
≤	Menor ou Igual que
≠	Diferente de

Operadores Relacionais

expressão = expressão: retorna verdadeiro quando as expressões resultarem no mesmo valor; caso contrário retorna falso

10 = 9 (Falso)

7-2 = 4+1 (Verdadeiro)

expressão > expressão: retorna verdadeiro quando a primeira expressão resultar em um valor superior ao da segunda; caso contrário retorna falso

10 > 9 (Verdadeiro)

7-2 > 4+1 (Falso)

Operadores Relacionais

expressão < expressão: retorna verdadeiro quando a primeira expressão resultar em um valor inferior ao da segunda; caso contrário retorna falso

10 < 9 (Falso)

7-2 < 4+1 (Falso)

expressão ≥ expressão: retorna verdadeiro quando a primeira expressão resultar em um valor superior ou igual ao da segunda; caso contrário retorna falso

10 ≥ 9 (Verdadeiro)

7-2 ≥ 4+1 (Verdadeiro)

Operadores Relacionais

expressão ≤ expressão: retorna verdadeiro quando a primeira expressão resultar em um valor inferior ou igual ao da segunda; caso contrário retorna falso

10 ≤ 9 (Falso)

7-2 ≤ 4+1 (Verdadeiro)

expressão ≠ expressão: retorna verdadeiro quando a primeira expressão resultar em um valor diferente ao da segunda; caso contrário retorna falso

10 ≠ 9 (Verdadeiro)

7-2 ≠ 4+1 (Falso)

Operadores Relacionais

Em Python, expressões relacionais são nativas. Porém, a **notação utilizada difere** um pouco da apresentada até aqui:

Operador (símbolo)	Operador (Python)	Descrição
=	==	Igualdade
>	>	Maior que
<	<	Menor que
≥	>=	Maior ou Igual que
≤	<=	Menor ou Igual que
≠	!=	Diferente de

Expressões Relacionais

Assim sendo, qual o resultado das expressões relacionais apresentadas a seguir?

a) **25 < 30?**

b) **-10 >= -5?**

c) a = 10 + 5 * 2 - (2 * 2) ** 2
b = 5 - 4 ** 2 + (16 ** 0.5)
a >= b? a != b?

d) a = 2 ** 2 == 3 + 1
a?

Expressões Relacionais

Em Python, é possível criar expressões relacionais entre outros tipos de dado:

Tipo de Dado #1	Tipo de Dado #2
int	int
float	float
int	float
bool	bool
str	str
bytes	bytes

Expressões Relacionais

int vs. int

Foi como organizamos as expressões relacionais até este momento! A comparação é feita diretamente pelo valor numérico

5 == 5?

float vs. float

Similar à comparação entre valores inteiros, porém agora considera-se valores fracionários

1.0 != 1.0?

0.27 >= 0.26?

Expressões Relacionais

int vs. float

Novamente, a comparação é feita pelo valor numérico. Na prática, o valor inteiro é *castado* para real e então a comparação é feita como um float vs. float

1.0 == 1?

-1 >= 0.26?

bool vs. bool

A regra geral nesse caso é: True é maior (>) que False

False <= True?

True == True?

Expressões Relacionais

str vs. str

A comparação entre *strings* é feita considerando seus valores *unicode* (para os caracteres usuais, podemos utilizar a tabela *ASCII*). Ou seja, se uma *string* aparecer depois na ordem lexicográfica (de dicionário) em relação a uma segunda, então a primeira é maior (>) que a segunda

“ab” > “ac”? “oi” > “hi”?

“ab” > “Ac”? “ oi” > “hi”?

A comparação **byte vs. byte** segue a mesma lógica da str vs. str!

Expressões Relacionais

Tentativas de **comparação entre tipos de dado “incompatíveis”**, em geral, resultam em erro

TypeError: '>' not supported between instances of
TypeError: '>=' not supported between instances of
TypeError: '<' not supported between instances of
TypeError: '<=' not supported between instances of

A exceção são os operadores de igualdade (==) e diferença (!=), estes funcionam entre quaisquer tipos de dado

Expressões Relacionais

Faça um algoritmo em Python que **atribua dois valores reais**, em variáveis *var01* e *var02*

Exiba na tela (*print*) os dois valores lidos e seus respectivos tipos (*type*)

Apresente o resultado das **expressões relacionais das variáveis intercaladas por todos os operadores relacionais disponíveis (==; >; >=; <; <=; !=)** com a *string* : “O resultado da expressão var01 *op* var02 é: ” e exiba (*print*) na tela

Expressões Lógicas

Expressões lógicas realizam uma ou mais operações lógicas.
Operações lógicas seguem as regras do cálculo proposicional

Uma proposição, nesse contexto, é uma **sentença que pode ser avaliada como verdadeira ou falsa**

Sendo assim, o resultado de uma expressão lógica, assim como nas expressões relacionais, é um **dado do tipo lógico**

Porém, seus operandos são sempre do tipo lógico!

Expressões Lógicas

P#n = Proposição (n)
C#n = Condição (n)
OL = Operador Lógico

[P#1] C#1 (OL) C#2

C#1 = 15 > 10 // **C#2** = 2 ** 2 <= 4 ** 0.5
[P#1] 15 > 10 **(OL)** 2 ** 2 <= 4 ** 0.5
[P#1] 15 > 10 **(OL)** 4 <= 2
[P#1] True **(OL)** False

P#1 será verdadeira ou falsa a depender do operador lógico (OL) definido!

Operadores Lógicos

Para compreendermos as expressões lógicas, precisamos inicialmente entender cada uma das **principais operadores lógicos disponíveis**:

Operador (símbolo)	Descrição	Tipo
∧ • &	Conjunção	Binário
∨ + ||	Disjunção inclusiva	Binário
¬ ~ !	Negação	Unário

Conjunção

Uma conjunção lógica **(∧)** é avaliada verdadeira quando todas as condições envolvidas na proposição também sejam avaliadas verdadeiras. A conjunção é tipicamente chamada de **E (AND) lógico**

Condição #1	Condição #2	Resultado (∧)
Verdadeira	Verdadeira	**Verdadeira**
Verdadeira	Falsa	**Falsa**
Falsa	Verdadeira	**Falsa**
Falsa	Falsa	**Falsa**

Conjunção

Aplicamos a conjunção lógica em vários aspectos da nossa vida cotidiana, porém de forma bastante natural e transparente

(Cenário) Ao voltar do almoço, passo na frente de uma confeitaria
(Proposição) Eu devo comer uma sobremesa?
(Condições) Eu tenho vontade de comer uma sobremesa?
Eu tenho tempo para comer uma sobremesa?
Eu tenho dinheiro para pagar uma sobremesa?

Conjunção

Eu tenho vontade de comer uma sobremesa? ∧ Eu tenho tempo para comer uma sobremesa? ∧ Eu tenho dinheiro para pagar uma sobremesa?

Sim ∧ Sim ∧ Sim (Eu vou comer uma sobremesa)

Sim ∧ Sim ∧ Não (Eu não vou comer uma sobremesa)

Sim ∧ Não {∧ Sim} (Eu não vou comer uma sobremesa)

Conjunção

Voltando ao nosso exemplo inicial:

[P#1] C#1 (OL) C#2
C#1 = 15 > 10 // **C#2** = 2 ** 2 <= 4 ** 0.5
[P#1] True **(OL)** False

Seja (OL) **uma conjunção (∧)**:

[P#1] True ∧ False
[P#1] False

Disjunção Inclusiva

Uma disjunção inclusiva **(∨)** é avaliada verdadeira quando alguma das condições envolvidas na proposição é avaliada como verdadeira. A disjunção inclusiva é tipicamente chamada de **OU (OR) lógico**

Condição #1	Condição #2	Resultado (∨)
Verdadeira	Verdadeira	**Verdadeira**
Verdadeira	Falsa	**Verdadeira**
Falsa	Verdadeira	**Verdadeira**
Falsa	Falsa	**Falsa**

Disjunção Inclusiva

Assim como a conjunção lógica, a disjunção inclusiva também é presente em diferentes momentos da nossa vida

(Cenário) Eu e um colega estamos nos preparando para uma reunião importante
(Proposição) Eu devo vestir um traje social?
(Condições) A convocação da reunião pede traje social?
Meu colega de trabalho irá de traje social?
Os convidados da reunião usarão traje social?

Disjunção Inclusiva

A convocação da reunião pede traje social? ∨ Meu colega de trabalho irá de traje social? ∨ Os convidados da reunião usarão traje social?

Não ∨ Não ∨ Não (Eu não irei vestir traje social)

Não ∨ Sim {∨ Sim} (Eu irei vestir traje social)

Sim {∨ Sim ∨ Sim} (Eu irei vestir traje social)

Disjunção Inclusiva

Voltando ao nosso exemplo inicial:

[P#1] C#1 (OL) C#2
C#1 = 15 > 10 // **C#2** = 2 ** 2 <= 4 ** 0.5
[P#1] True **(OL)** False

Seja (OL) **uma disjunção inclusiva (∨)**:

[P#1] True ∨ False
[P#1] True

Negação

Uma negação **(¬)** é avaliada verdadeira quando uma condição envolvida na proposição for falsa. A negação é tipicamente chamada **NÃO (NOT) lógico**

Condição	Resultado (¬)
Verdadeira	**Falsa**
Falsa	**Verdadeira**

Negação

A negação lógica é bastante usada no dia a dia, estando presente em diversos cenários recorrentes

(Cenário) O dia está “cinza” e eu vou visitar um amigo
(Proposição) Se não chover, irei até a sua casa
(Condições) Choveu?

Negação

¬ Choveu?

¬ Sim (Falso, não irei até a casa do meu amigo)

¬ Não (Verdadeiro, irei até a casa do meu amigo)

Negação

Por ser um operador lógico unário, a negação não se aplica ao nosso exemplo inicial

Porém, podemos **aplicar a negação individualmente às condições** presentes na proposição do exemplo:

¬ 15 > 10
¬ True
False

¬ 2 ** 2 <= 4 ** 0.5
¬ False
True

Expressões Lógicas

Em Python, a **notação dos operadores lógicos e a precedência** dos mesmos são apresentadas na tabela a seguir:

Operador (símbolo)	Operador (Python)	Precedência
¬ ~ !	**not**	1 (maior)
∧ • &	**and**	2
∨ + ||	**or**	3

Além disso, cabe ressaltar que a **associatividade é à esquerda**

Expressões Lógicas

Ainda, podemos utilizar **parênteses para modificar a ordem natural de processamento** de uma expressão lógica:

True and True or True and False
True or False
True

(True and True or True) and False
True and False
False

Expressões Lógicas

Construa um algoritmo em Python que **processe as seguintes expressões lógicas e exiba (*print*) os seus resultados**:

a) False and False or True or True
b) not True and False or True and not False
c) not not True and False or False

Em seguida, **apenas adicionando parênteses às expressões**, faça as mesmas resultarem na negação do valor lógico originalmente obtido

Nota de Rodapé

Além da precedência entre operadores referentes a cada categoria de expressão estudada até então (aritmética, relacional e lógica), existe uma **precedência entre as próprias expressões**:

aritmética > relacional > lógica

5 + 1 >= **3 * 2** and **10 + 4** == **14**
6 >= 6 and **14 == 14**
True and True
True

Exercício #07

Faça um algoritmo em Python que **guarda o raio de uma circunferência (float) em centímetros** em uma variável *r*.

Calcule o diâmetro da circunferência em polegadas (1 polegada = 2,54 centímetros).

Crie as expressões relacionais e lógicas que respondam às seguintes questões e exiba (*print*) o resultado das mesmas na tela:

a) O diâmetro é maior ou igual que 13 e menor ou igual que 17?
b) O diâmetro é maior que 17 e menor ou igual que 24?
c) O diâmetro é maior que 24 e menor ou igual que 30?
d) O diâmetro é menor que 13 ou maior que 30?

Fundamentos de Programação

Aula 07

Obrigado e ATÉ A PRÓXIMA!