# SCC 124 - Introdução à Programação para Engenharias

## Expressões

python

Professor: André C. P. L. F. de Carvalho, ICMC-USP
Pos-doutorando: Isvani Frias-Blanco
Monitor: Henrique Bonini de Britto Menezes



## Aula de hoje

- Expressões
- Operadores
  - Aritméticos
  - Relacionais
  - *Bitwise*
  - Atribuição
- Mistura de tipos



## Expressões

- Frequentemente usadas em programas escritos em Python
- São compostas por operandos e operadores
  - Ex.: *x = a + 5*;
  - Operando
    - Constante literal, variável, expressão ou função
      - Ex.: a, 2, soma(x),...
  - Operadores
    - Representados por símbolos e palavras chave especiais, manipulam operandos



## Expressões

- Construídas de acordo com a sintaxe da linguagem
- Calculam valores que podem:
  - Ser atribuídos a variáveis
  - Controlar fluxo de execução de um programa
- Valor e tipo do resultado
  - Dependem do operador e do tipo dos operandos utilizados



## Exemplos

```
>>> 3 + 4
7
>>> 3 * 4
12
>>> x = 4 * 3
>>> x
12
>>> x = - 7 + 5 * 3 / 4 - (8 * 3 + 2)
```



## Programa em Python

```
# Cálculo do valor de uma expressão

val = 0
while (val < 16):
    val = val * 3 - 8 + 14
print ('Valor da expressão eh: %d\n' %(val))
```

## Mesma programa em C

```
/* Calculo do valor de uma expressao*/
# include <stdio.h>

main (){
   int valor;

   valor = 0;
   while (valor < 16){
      valor = valor + 4;
   }
   printf ("Valor da expressao eh: %d\n", valor);
}
```



## Operadores

- Aritméticos
- Relacionais
- *Bitwise*
- Atribuição



## Operadores

- Ordem como uma expressão é avaliada é importante
  - Ex.: *x + y / 100*
  - Ordem de avaliação pode ser definida explicitamente
    - Ex.: *(x + y) / 100*
  - Operadores da linguagem Python possuem uma ordem de precedência



## Operadores

- Precedência
  - Informa ordem de aplicação de operadores na ausência de parênteses
    - Operador com precedência mais alta é aplicado antes
    - Ex.: *x = 4 * 2 - 3*



## Precedência de operadores

Menor

Maior

| Operador | Descrição |
|---|---|
| lambda | Expressão Lambda |
| if – else | Expressão Condicional |
| or | Booleano OR |
| and | Booleano AND |
| not x | Booleano NOT |
| in, not in, is, is not, <, <=, >, >=, !=, == | Comparações, incluindo testes de pertinência e testes de identidade |
| \| | Bitwise OR |
| ^ | Bitwise XOR |
| & | Bitwise AND |
| <<, >> | Shifts |
| +, - | Adição e subtração |
| *, @, /, //, % | Multiplicação, multiplicação de matrizes divisão, resto |
| +x, -x, ~x | Positivo, negativo, bitwise NOT |
| ** | Exponenciação |
| await x | Expressão await |
| x[index], x[index:index], x(arguments...), x.attribute | Subscrito, fatiamento (slicing), chamada, referência a atributo |
| (expressions...), [expressions...], {key: value...}, {expressions...} | Exibir binding ou tuple, exibir lista, exibir dicionário, exibir conjunto |



## Operadores

- Operadores na mesma linha têm a mesma precedência
- Associatividade
  - Operadores binários são avaliados da esquerda para a direita
    - Exceto os de atribuição
      - Avaliados da direita para a esquerda

## Exemplo

- Expressão em Python para a fórmula:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

- *x = (-b + math.sqrt (b*b - 4 *a*c)) / (2*a)*



## Exercício

- Escrever programa em Python para definir valor das expressões abaixo:
  a) *y = 7 + 4 % 2 - 5 * 4*
  b) *x = 7*
  *y = (x + 3) * 2 - 7 % 2 * (5 - 2)*



## Operadores aritméticos

- Operadores binários

| Operador | Uso | Descrição |
|---|---|---|
| + | *op1 + op2* | *Adiciona op1 e op2* |
| - | *op1 - op2* | *Subtrai op2 de op1* |
| * | *op1 * op2* | *Multiplica op1 por op2* |
| / | *op1 / op2* | *Divide op1 por op2* |
| // | *op1 // op2* | *Divide op1 por op2 e trunca* |
| % | *op1 % op2* | *Calcula o resto op1 / op2* |
| @ | *op1 @ op2* | *Multiplica op1 por mop2** |

* Ainda não está implementado na linguagem Python



## Operadores de divisão

- Python tem dois operadores de divisão
  - Divisão /
    - Resultado é um valor real
      - Não importa se a divisão for exata
      - Exemplos:
        - 9/4 = 2.5
        - 8/4 = 2.0
  - Divisão e truncamento //



## Divisão e truncamento (//)

- Parte fracionária é descartada (truncamento)
  - Se os dois operandos forem do tipo inteiro
    - Resultado é do tipo inteiro
    - Ex.: 9 // 4 = 2
  - Se pelo menos um dos operandos for do tipo ponto flutuante
    - Os argumentos são primeiro convertidos para o tipo de maior precisão (neste caso, *float*)
    - Resultado é do tipo ponto flutuante
    - Ex. 9.0 // 4 = 9 // 4.0 = 9.0 // 4.0 = 2.0



## Operador de resto (%)

- Resto da divisão de um valor por outro
  - Exemplos:
    - 9 % 4 =
    - 9.0 % 4 =
    - 7 % -4 =
    - -7 % 4 =
  - Útil para testar se um número é divisível por um outro (resto da divisão é igual a zero)
  - Resultado possui mesmo sinal do segundo operando
    - Valor absoluto do resultado é menor que o valor absoluto do segundo operando

## Operador de resto (%)

- Resto da divisão de um valor por outro
  - Exemplos:
    - 9 % 4 = 1
    - 9.0 % 4 = 1.0
    - 7 % -4 = -1
    - -7 % 4 = 1
  - Útil para testar se um número é divisível por um outro (resto da divisão é igual a zero)
  - Resultado possui mesmo sinal do segundo operando
    - Valor absoluto do resultado é menor que o valor absoluto do segundo operando



## Operador de resto (%)

- Pode ser usado para extrair os N valores menos significativos
  - Exemplos:
    - Base 10
      - x = y % 10
      - x = y % 100
    - Base 2
      - x = y % 0b10
      - x = y % 0b100



## Operadores relacionais

- Comparam dois valores e retornam a relação entre eles
  - Valor Booleano

| Operador | Uso | Descrição |
|---|---|---|
| == | *op1 == op2* | *op1 é igual a op2* |
| != | *op1 != op2* | *op1 é diferente de op2* |
| > | *op1 > op2* | *op1 é maior que op2* |
| < | *op1 < op2* | *op1 é menor que op2* |
| >= | *op1 >= op2* | *op1 é maior ou igual a op2* |
| <= | *op1 <= op2* | *op1 é menor ou igual a op2* |



## Operadores relacionais

- Retornam dois valores
  - *True* ou *False*
    - Qualquer valor ≠ 0 é interpretado como *True*
      - Pode ser útil, mas deve ser evitado
  - Valores especiais do tipo *bool* (não são *strings*)
- Não confundir = (atribuição) com == (igual)
  - Um dos erros mais comuns
  - Interpretador Python geralmente não detecta esses erros



## Operadores lógicos

- Utilizam operandos Booleanos
  - Retornam resultado Booleano

Precedência

| Operador | Uso | Retorna **True** se |
|---|---|---|
| not | not op1 | op1 é falso |
| and | op1 and op2 | op1 e op2 são ambos verdade, avalia condicionalmente op2 |
| or | op1 or op2 | op1 ou op2 é verdade, avalia condicionalmente op2 |

Maior

↓

Menor



## Operadores lógicos

- Como funcionam

| Operandos | | Operadores | | |
|---|---|---|---|---|
| op1 | op2 | not (op2) | and | or |
| *False* | *False* | *True* | *False* | *False* |
| *False* | *True* | *False* | *False* | *True* |
| *True* | *False* | *True* | *False* | *True* |
| *True* | *True* | *False* | *True* | *True* |

## Operadores lógicos

- Avaliação preguiçosa
  - Operandos de um operador lógico são avaliados da esquerda para a direita
  - Avaliação termina assim que a resposta puder ser determinada
    - op1 *and* op2
    - op1 *or* op2



## Operadores bitwise

- Permite manipular bits dos dados

| Operador | Uso | Descrição |
|---|---|---|
| >> | *op1 >> op2* | *move op2 posições p/ direita os bits de op1* |
| << | *op1 << op2* | *move op2 posições p/ esquerda os bits de op1* |
| & | *op1 & op2* | *bitwise and* |
| \| | *op1 \| op2* | *bitwise or* |
| ^ | *op1 ^ op2* | *bitwise xor (ou exclusivo)* |
| ~ | *~ op2* | *complemento a 1* |



## Operadores bitwise

- Como funcionam bit a bit

| Operandos | | Operadores | | | |
|---|---|---|---|---|---|
| op1 | op2 | ~op2 | & | \| | ^ |
| *0* | *0* | *1* | *0* | *0* | *0* |
| *0* | *1* | *0* | *0* | *1* | *1* |
| *1* | *0* | *1* | *0* | *1* | *1* |
| *1* | *1* | *0* | *1* | *1* | *0* |



## Operadores bitwise

- Como funcionam para um byte

| Operador | Exemplo | Resultado |
|---|---|---|
| >> | *11110011 >> 2* | |
| << | *11110011 << 1* | |
| & | *10001001 & 00000101* | |
| \| | *10001001 \| 00000101* | |
| ^ | *10001001 ^ 00000101* | |
| ~ | *~11110000* | |



## Operadores bitwise

- Como funcionam para um byte

| Operador | Exemplo | Resultado |
|---|---|---|
| >> | *11110011 >> 2* | *00111100* |
| << | *11110011 << 1* | *11100110* |
| & | *10001001 & 00000101* | *00000001* |
| \| | *10001001 \| 00000101* | *10001101* |
| ^ | *10001001 ^ 00000101* | *10001100* |
| ~ | *~11110000* | *00001111* |



## Operadores bitwise

- Exemplo
  - Calcular o valor da expressão:
    - *13 >> 1; /** Usar codificação com 8 bits */*

## Operadores bitwise

- Exemplo
  - Calcular o valor da expressão:
    - *13 >> 1; /** Usar codificação com 8 bits */*
  - Resposta:
    - Representação binária do valor 13: 00001101
    - Movendo 00001101 uma posição para a direita: 00000110 = 6
    - Equivale a divisão truncada por 2



## Operador (comando) de atribuição

- Atribuição de valores a variáveis é construída na forma de uma expressão
  - Tipo da variável será o tipo do operando
  - Ex.: resultado = 4
    - Variável será do tipo *int*

```
>>> n = 3
>>> n
3
>>> n = 2.0
>>> n
2.0
```



## Operador de atribuição

- Permite atribuir valores a mais de uma variável simultaneamente

```
>>> a, b, c = 3, 4, 5
>>> a

>>> b

>>> c
```

```
>>> x = y = z = 0
>>> x

>>> y

>>> z
```



## Operador de atribuição

- Permite atribuir valores a mais de uma variável simultaneamente

```
>>> a, b, c = 3, 4, 5
>>> a
3
>>> b
4
>>> c
5
```

```
>>> x = y = z = 0
>>> x
0
>>> y
0
>>> z
0
```



## Operador de atribuição

- Python permite a combinação da atribuição com operadores binários
  - Atribuição com atalho
    *total += 3 ⇔ total = total + 3*
  - Exemplos:
    - *valor+= 3*
    - *res -= 2*
    - *x /= 10*
    - *y //= 8*
    - *z *= 2*



## Mistura de tipos

- Expressão pode incluir valores de tipos numéricos diferentes
  - Operandos numéricos são convertidos para um tipo comum
    - Maior precisão
    - Resultado será do tipo de maior precisão
    - Garante que o resultado da operação seja tão preciso quando possível

## Exemplo

```
>>> 3 * 3.75 / 1.5
7.5
>>> 7.0 / 2
3.5
```

- Operadores com operandos de tipos diferentes geram resultado com o tipo de maior precisão
  - Ex.: inteiro para ponto flutuante



## Exemplo

- Mistura de tipos

```
>>> n = 3
>>> x = 2.0
>>> r1 = n + 1
>>> r2 = n + 1.0
>>> r3 = x + 1
>>> r4 = x + 2.0
>>> r1

>>> r2

>>> r3

>>> r4
```



## Exemplo

- Mistura de tipos

```
>>> n = 3
>>> x = 2.0
>>> r1 = n + 1
>>> r2 = n + 1.0
>>> r3 = x + 1
>>> r4 = x + 2.0
>>> r1
4
>>> r2
4.0
>>> r3
3.0
>>> r4
4.0
```



## Conversão de Valores

- Python distingue entre número e *strings*
  - Mistura desses tipos em uma expressão provoca erro
- Python possui operadores para converter números em *strings*
  - E vice-versa



## Conversão de Valores

- Conversão de um valor numérico no *string* correspondente
  - *str ()*
- Converção de um *string* de números no valor numérico correspondente
  - *int ()*
  - *float ()*
  - *eval ()*
    - Avalia string como uma expressão, não da para saber tipo do resultado (evitar)



## Conversão de Valores

```
>>> 2 + '3.8'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: unsupported operand type(s) for +: int' and 'str'
>>> 2 + eval ('3.8')
5.8
>>> str (2 + eval ('3.8'))
'5.8'
>>> n = int ('20')
>>> n
20
>>> a = float ('4.25')
>>> a
4.25
```

## Exercício

- Escreva um programa em Python que verifica se um número é primo



## Exercício

- Escrever um programa em Python para calcular a média final do aluno utilizando os critérios discutidos para esta disciplina



## Conclusão

- Expressões
- Operadores
  - Aritméticos
  - Booleanos
  - Bitwise
  - Atribuição
- Conversão de tipos



## Perguntas