

ALGORITMOS E LÓGICA DE PROGRAMAÇÃO

AULA 2: CONSTANTES, VARIÁVEIS, OPERADORES, EXPRESSÕES, ENTRADA E SAÍDA

DO QUE VAMOS FALAR

- 1. Constantes
- 2. Operadores aritméticos
- 3. Operadores relacionais
- 4. Operadores lógicos
- 5. Comentários
- 6. Variáveis
- 7. Identificadores
- 8. Palavras reservadas
- 9. Python Tutor
- 10. Operadores de atribuição
- 11. Precedência e associatividade
- 12. Entrada
- 13. Saída



CONSTANTES

Símbolos que representam valores e não podem ser alterados. Geralmente são usados em expressões.

Exemplos: `7`, `-123`, `4.57`, `'adoro Python!'`, `"oi"`, `False`, `True`.

Exemplos de expressões em que todos os operandos são constantes:

`2 + 4 * 10.5`

`False and True == True`

`'O Pequeno' + ' ' + 'Príncipe'`

CONSTANTES

- **Números inteiros**: ausência de ponto decimal.

45, 12345, -65, 0, 0b110 (base 2), 0o7 (base 8), 0xF (base 16) etc.

- **Números reais**: presença de ponto decimal ou notação científica.

3.141593, 1045.99, 6.02e-23, 6.02E-23, 1e3 etc.

- **String**: delimitada por apóstrofes ('*meu texto*') ou aspas ("*meu texto*").

'tarde!', "123", 'True', '10+5', "6.02e-23" etc.

- **Booleanas**: as constantes booleanas são **True** e **False**, portanto, os seguintes valores **NÃO** são constantes booleanas:

'False', 'True', false, true, FALSE, TRUE, fAlSe, tRuE etc.

OPERADORES ARITMÉTICOS

OPERADOR	DESCRIÇÃO	EXEMPLO
$+$ (binário)	Soma o primeiro operando com o segundo.	$3 + 7 \rightarrow 10$
$+$ (unário)	Mantém o sinal do operando à direita. Corresponde à função identidade.	$+ (2) \rightarrow 2$ $+ (-2) \rightarrow -2$
$-$ (binário)	Subtrai o segundo operando do primeiro.	$8 - 2 \rightarrow 6$
$-$ (unário)	Inverte o sinal do operando à direita.	$- (2) \rightarrow -2$ $- (-2) \rightarrow +2$
$*$	Multiplica o primeiro operando com o segundo.	$5 * 3 \rightarrow 15$
$/$	Quociente da divisão real do primeiro operando pelo segundo.	$7 / 2 \rightarrow 3.5$ $7.0 / 2 \rightarrow 3.5$
$//$	Quociente da divisão inteira do primeiro operando pelo segundo.	$7 // 2 \rightarrow 3$ $7.0 // 2 \rightarrow 3.0$
$\%$	Resto da divisão inteira do primeiro operando pelo segundo.	$7 \% 2 \rightarrow 1$ $7.0 \% 2 \rightarrow 1.0$
$**$	Exponenciação: base $**$ expoente	$2 ** 4 \rightarrow 16$

OPERADORES ARITMÉTICOS

- Operações aritméticas que envolvam **apenas** operandos inteiros, resultam em números inteiros. Exceção: divisão real.
- Operações aritméticas que envolvam **algum** operando real, resultam em números reais.
- Teste no interpretador interativo as seguintes expressões aritméticas:

```
>>> 537//10
```

```
>>> 1234%10
```

```
>>> 1234//100
```

```
>>> 537/10
```

```
>>> 1234%100
```

```
>>> 1234//1000
```

```
>>> 15.0//2
```

```
>>> 1234%1000
```

```
>>> 123//1000
```

```
>>> 15.0/2
```

```
>>> 1234//10
```

```
>>> 123/1000
```

OPERADORES RELACIONAIS

OPERADOR	DESCRIÇÃO	EXEMPLO
<code>==</code>	<code>True</code> se ambos operandos são iguais.	<code>3 == 2 → False</code>
<code>!=</code>	<code>True</code> se ambos operandos são diferentes.	<code>'ana' != 'análise' → True</code>
<code>></code>	<code>True</code> se o operando à esquerda é maior que o da direita.	<code>10 > 10 → False</code>
<code>>=</code>	<code>True</code> se o operando à esquerda é maior ou igual que o da direita.	<code>10 >= 10 → True</code>
<code><</code>	<code>True</code> se o operando à esquerda é menor que o da direita.	<code>'margô' < 'margarida' → False</code>
<code><=</code>	<code>True</code> se o operando à esquerda é menor ou igual que o da direita.	<code>(2*5 <= 20//2) → True</code>

OPERADORES LÓGICOS

OPERADOR	EXEMPLO	DESCRIÇÃO
and	<pre>True and True → True True and False → False False and True → False False and False → False</pre>	E lógico
or	<pre>True or True → True True or False → True False or True → True False or False → False</pre>	OU lógico
not	<pre>not True → False not False → True</pre>	NÃO lógico

OPERADORES LÓGICOS

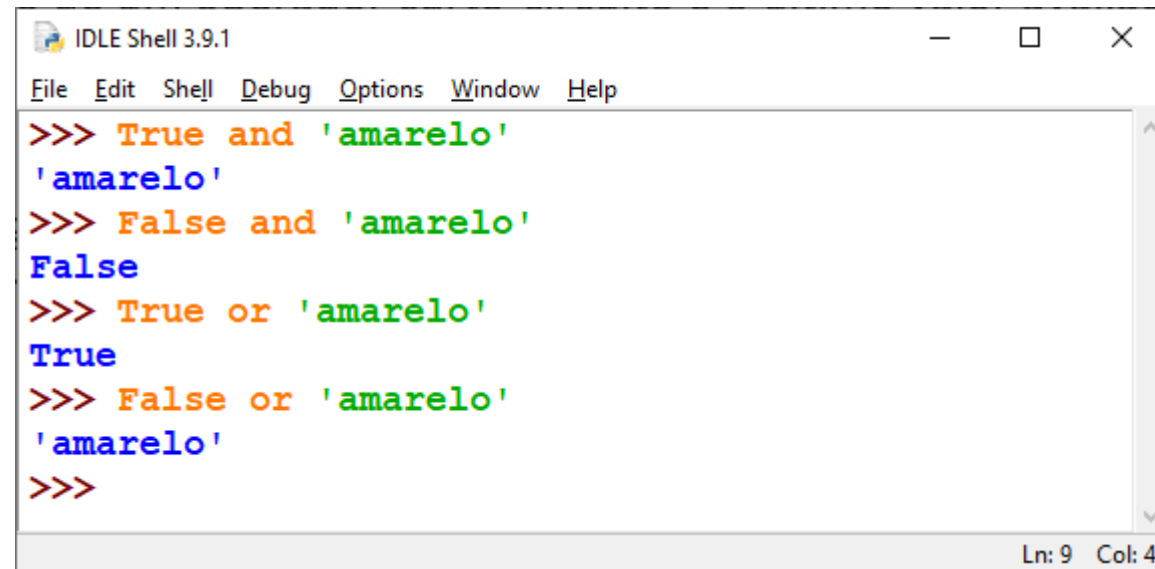
and e **or** são operadores curto-circuito, ou seja, seus operandos são avaliados da esquerda para a direita, e a avaliação encerra quando o resultado está determinado, algumas vezes sem precisar avaliar a expressão completa.

Por exemplo, se A e C são expressões que resultam em um valor interpretado como verdadeiro, mas B resulta em falso, então (A **and** B **and** C) não chega a avaliar a expressão C, pois o resultado já está determinado na primeira parte da expressão, será falso sem dúvidas.

Em geral, quando aplicados sobre valores genéricos, e não em booleanos, o valor do resultado de um operador curto-circuito é o último valor avaliado na expressão.

OPERADORES LÓGICOS

Em geral, quando aplicados sobre valores genéricos, e não com booleanos, o valor do resultado de um operador curto-circuito é o último valor avaliado na expressão.



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
>>> True and 'amarelo'
'amarelo'
>>> False and 'amarelo'
False
>>> True or 'amarelo'
True
>>> False or 'amarelo'
'amarelo'
>>>
```

Ln: 9 Col: 4

Em Python os seguintes valores são interpretados como falso quando avaliados em termos de **True** e **False**: **0**, **0.0**, **' '**, **[]**, **()**, **set()**, **{}**, **None**

COMENTÁRIOS

É possível inserir comentários nos arquivos de código-fonte Python. O uso de comentários pode ajudar no entendimento de determinados trechos da codificação e são ignorados na execução do programa.

```
# Este é um comentário!  
# Este é outro comentário!  
# Este é o terceiro comentário!  
# Você já entendeu... este é o 4º comentário.
```

OBSERVAÇÃO

Comentários são iniciados com `#` e têm apenas uma linha. Caso seja necessário comentar mais linhas, cada uma terá que começar com `#`.

VARIÁVEIS

Espaço de memória associado a um identificador e usado para **guardar valores** que o programa poderá acessar e modificar. O identificador da variável permite referenciá-la sem ambiguidade, por isso deve ser único.

Em Python uma variável é criada no momento em que um **valor é atribuído** a um identificador válido. Veja o exemplo:

minha_idade = 28



Identificador da variável
(também chamado de nome da variável)



Operador de
atribuição



Valor atribuído
à variável

VARIÁVEIS

- Como o próprio nome indica, uma mesma variável pode "variar" em relação ao valor que está atribuído a ela. Ou seja, **valores diferentes** podem ser atribuídos à mesma variável, inclusive, valores de tipos diferentes.

```
>>> x = False # x guarda um dado do tipo booleano (boolean).  
>>> x = 3.14   # x guarda um dado do tipo real (float).  
>>> x = 100    # x guarda um dado do tipo inteiro (int).  
>>> x = 'abc'  # x guarda um dado do tipo cadeia de caracteres (string).
```

- Lembre-se que em Python o tipo do dado está **associado ao valor atribuído** e não à variável. Comprove isso usando a função `type(variável)`.
- Uma variável guarda só um valor por vez, afinal "dois corpos não ocupam o mesmo espaço ao mesmo tempo".

IDENTIFICADORES

Os identificadores são formados por uma sequência de um ou mais caracteres. A recomendação é que sejam claros, concisos e significativos para quem lerá o código-fonte. Há **regras para definição** de identificadores:

- Podem conter somente letras, dígitos e *underscores* (sublinhado);
- Não podem iniciar com dígito;
- Não podem conter caracteres especiais, por exemplo, espaço;
- Não podem ser palavras reservadas da linguagem.

Um bom identificador deve ser conciso, porém descritivo (`cpf` é melhor que `c`, `tamanho_nome` é melhor que `tamanho_do_nome_da_pessoa`).

IDENTIFICADORES

Python é uma linguagem de programação *case-sensitive*, ou seja, é sensível a diferença entre letras minúsculas e maiúsculas. Veja o exemplos:

meu_nome	True	False	and
≠	≠	≠	≠
Meu_nome	true	fAlSe	AND

Portanto, tome cuidado para escrever as instruções considerando essa característica. Existem linguagens que não são *case-sensitive*.

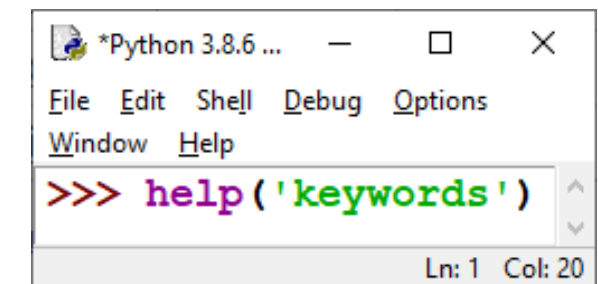
PALAVRAS RESERVADAS

O Python 3.8 possui 35 **keywords**, que são palavras reservadas e, portanto, **não podem** ser usadas como identificadores.

<u>False</u>	<u>await</u>	<u>else</u>	<u>import</u>	<u>pass</u>
<u>None</u>	<u>break</u>	<u>except</u>	<u>in</u>	<u>raise</u>
<u>True</u>	<u>class</u>	<u>finally</u>	<u>is</u>	<u>return</u>
<u>and</u>	<u>continue</u>	<u>for</u>	<u>lambda</u>	<u>try</u>
<u>as</u>	<u>def</u>	<u>from</u>	<u>nonlocal</u>	<u>while</u>
<u>assert</u>	<u>del</u>	<u>global</u>	<u>not</u>	<u>with</u>
<u>async</u>	<u>elif</u>	<u>if</u>	<u>or</u>	<u>yield</u>

OBSERVAÇÃO

Para obter todas as *keywords* de sua versão do Python use a seguinte instrução:



```
*Python 3.8.6 ...  
File Edit Shell Debug Options  
Window Help  
>>> help('keywords')  
Ln: 1 Col: 20
```

Fonte: <https://realpython.com/python-keywords/>

PYTHON TUTOR

Ambiente que elabora e exibe **representações gráficas** do que ocorre na **memória** e na **saída** após a execução de cada instrução do código-fonte.

The screenshot shows the Python Tutor interface in a web browser. The address bar indicates the URL `pythontutor.com/visualize.html#mode=display`. The main content area displays the following Python code:

```
Python 3.6
(known limitations)

1 n1 = 10
2 n2 = 35
→ 3 soma = n1 + n2
```

Below the code, a legend indicates that the green arrow points to the line that just executed, and the red arrow points to the next line to execute. A progress bar and navigation buttons (<< First, < Prev, Next >, Last >>) are visible. The status bar at the bottom shows "Done running (3 steps)" and a link to "Customize visualization (NEW!)".

On the right side, the "Frames" and "Objects" panels are shown. The "Global frame" table contains the following data:

Variable	Value
n1	10
n2	35
soma	45

Fonte: pythontutor.com

OPERADORES DE ATRIBUIÇÃO

OPERADOR	EXEMPLO	DESCRIÇÃO
<code>=</code>	<code>variável = expressão</code>	Atribui o resultado da expressão à variável.
<code>+=</code>	<code>variável += expressão</code> <code>variável = variável + (expressão)</code>	Atribui o resultado da adição à variável.
<code>-=</code>	<code>variável -= expressão</code> <code>variável = variável - (expressão)</code>	Atribui o resultado da subtração à variável.
<code>*=</code>	<code>variável *= expressão</code> <code>variável = variável * (expressão)</code>	Atribui o resultado do produto à variável.
<code>/=</code>	<code>variável /= expressão</code> <code>variável = variável / (expressão)</code>	Atribui o quociente da divisão real à variável.
<code>//=</code>	<code>variável //= expressão</code> <code>variável = variável // (expressão)</code>	Atribui o quociente da divisão inteira à variável.
<code>%=</code>	<code>variável %= expressão</code> <code>variável = variável % (expressão)</code>	Atribui o resto da divisão inteira à variável.
<code>**=</code>	<code>variável **= expressão</code> <code>variável = variável ** (expressão)</code>	Atribui o resultado da exponenciação à variável.

PRECEDÊNCIA E ASSOCIATIVIDADE DE OPERADORES

Precedência: indica a prioridade que um operador possui em uma expressão em relação aos outros operadores, ou seja, aquele com maior precedência será avaliado antes daqueles com menor. A precedência de um operador pode ser alterada com uso de parênteses.

- **Exemplo (1):** $5 + 3 * 2$ resulta em 11, pois a multiplicação tem precedência em relação à adição.
- **Exemplo (2):** $(5 + 3) * 2$ resulta em 16, pois o par de parênteses alterou a precedência da adição, aumentando sua prioridade.

Em caso de expressões com pares de **parênteses aninhados**, a precedência é do mais aninhado para o menos aninhado.

PRECEDÊNCIA E ASSOCIATIVIDADE DE OPERADORES

Associatividade: é uma propriedade que define como operadores de mesma precedência são agrupados em uma expressão na ausência de parênteses.

Por exemplo, imagine uma expressão em que um operando é precedido e sucedido por operadores de mesma precedência, como $2 + 5 - 4$, neste caso o operando **5** pode ser associado com a adição, $2 + 5$, ou com a subtração, $5 - 4$.

Com base na regra de associatividade dos operadores de adição e subtração, é possível concluir, **sem ambiguidade**, qual será o resultado da expressão. Pelas regras do Python, no exemplo anterior, o operando **5** será associado à adição.

Os operadores podem ser associativos **à esquerda**, **à direita**, ou serem **não associativos**.

PRECEDÊNCIA E ASSOCIATIVIDADE DE OPERADORES

- **Associativos à esquerda:** as operações são agrupadas da esquerda para a direita. Ou seja, o operando será associado ao operador à esquerda.
- **Associativos à direita:** as operações são agrupadas da direita para a esquerda. Ou seja, o operando será associado ao operador à direita.
- **Não associativos:** as operações não podem ser encadeadas (porque possuem comportamento indefinido, o que pode gerar um erro) ou o encadeamento pode representar algo "especial" (uma interpretação diferenciada na linguagem de programação).

Cada **linguagem de programação** têm suas próprias regras de precedência e associatividade para seus operadores.

PRECEDÊNCIA E ASSOCIATIVIDADE DE OPERADORES

OPERADOR	DESCRIÇÃO	ASSOCIATIVIDADE
**	Exponenciação.	À direita
+ operando, - operando	Identidade e inversão de sinal.	À esquerda
* , / , // , %	Multiplicação, divisão real, divisão inteira e resto da divisão.	
+ , -	Adição e subtração.	
== , != , < , <= , > , >=	Operadores relacionais.	Não associativo
not	NÃO lógico.	À esquerda
and	E lógico.	
or	OU lógico.	
= , += , -= , *= , /= , //= , %= , **=	Atribuições.	Não associativo

PRECEDÊNCIA E ASSOCIATIVIDADE DE OPERADORES

Em Python os **operadores relacionais são não associativos**, ainda que sejam avaliados da esquerda para a direita nas expressões.

Isso ocorre porque Python interpreta uma expressão com encadeamento de operadores relacionais do mesmo modo que a **matemática**. Veja o exemplo:

- A expressão $7 > 5 > 4$ em Python será interpretada como $(7 > 5) \text{ and } (5 > 4)$. O que resultada em `True`.
- Em algumas linguagens a expressão $7 > 5 > 4$ pode ser interpretada como $(7 > 5) > 4$, por causa da associatividade à esquerda. Isso pode parecer estranho ao programador, pois equivale a avaliar `True > 4`.

PRECEDÊNCIA E ASSOCIATIVIDADE DE OPERADORES

Teste as expressões no IDLE e verifique se resultam no presumido por você.

```
>>> 3*2/2*3
```

```
>>> 3*(2/2*3)
```

```
>>> 3*2/(2*3)
```

```
>>> 2**1**3
```

```
>>> 2**(1**3)
```

```
>>> (2**1)**3
```

```
>>> --7
```

```
>>> ++123
```

```
>>> -+-10.4
```

```
>>> 10<20<30
```

```
>>> 10<20 and 20<30
```

```
>>> 30>20>10
```

```
>>> 30>20 and 20>10
```

```
>>> (10<20)<30
```

```
>>> (30>20)>10
```

```
>>> 10<(20<30)
```

```
>>> 30>(20>10)
```

```
>>> 'a'!='A'==True
```

```
>>> not False and True
```

```
>>> False or not True
```

```
>>> not not not False
```

```
>>> 2+2 == 4 == 8/2
```

```
>>> (2+2 == 4) == 8/2
```

```
>>> a = b = c = 10
```

```
>>> a = b = (c = 10)
```

```
>>> a += b += 1
```

```
>>> and != or
```


ENTRADA DE DADOS

Para coletar dados externos ao programa, podemos usar a função `input()`. Geralmente essa função é usada para permitir que o usuário personalize as entradas do programa que são comumente dadas por meio de um teclado.

```
nome = input('Seu nome: ')           # nome receberá uma string.
idade = int(input('Sua idade: '))    # idade receberá um inteiro.
salario = float(input('Seu salário: ')) # salário receberá um real.
```

- A função `input(string)` pode conter como argumento uma *string* que será exibida na saída;
- Os dados lidos com `input()` sempre serão interpretados como uma *string*, por isso, caso seja necessário, os dados deverão ser convertidos para outro tipo apropriado.

SAÍDA DE DADOS

Uma função para saída de dados é `print()`. A função aceita quantidade variável de argumentos que serão exibidos, todos separados por vírgulas.

```
>>> print('Hello World!')
>>> print('Minha idade é:', 29)
>>> nome = 'Megan'
>>> print('Olá! Sou', nome, 'prazer!')
>>> a = int(input())
>>> b = int(input())
>>> print('A soma é', a+b)
>>> print('A soma é %d' % (a+b))
```