

PYTHON – FOLHA DE CONSULTA

Valdemar W. Setzer

www.ime.usp.br/~vwsetzer (na seção [Recursos Educacionais](#))

Versão 15.8– 26/5/22

ÍNDICE

1. [Observações](#)
2. [Tipos, constantes, variáveis e matrizes](#)
3. [Operadores](#)
4. [Operadores lógicos \(booleanos\)](#)
5. [Funções nativas](#)
6. [Algumas funções e constantes matemáticas](#)
7. [Outras funções](#)
8. [Precedência \(ordem de execução\)](#)
9. [Declaração e uso de uma função](#)
10. [Notação lambda](#)
11. [Identificadores globais e locais](#)
12. [Classes](#)
13. [Comandos compostos](#)
14. [Palavras reservadas](#)
15. [Referências](#)
 - 15.1 [Tutoriais](#)
 - 15.2 [Outras referências](#)
16. [Instalação do Python e uso do interpretador IDLE](#)
17. [Curso](#)
18. [Textos, ambientes de programação, documentação e fóruns de programação](#)
19. [Agradecimentos](#)

1. OBSERVAÇÕES

1. Atenção: a grande parte das células das tabelas abaixo contém uma única linha; se houver muitas células com mais do que uma linha, aumente o tamanho da janela do navegador ou diminua o tamanho das letras (em geral, no PC com Ctrl –; para aumentar de volta, Ctrl +).
2. Todos os operadores, funções e comandos dos exemplos foram testados com o interpretador IDLE ("Integrated Development and Learning Environment") do Python 3.6.1, a menos de alguns itens onde foi usado o Jupyter do Azure (ver o item [Ambientes](#)). Testes com outras versões são anotadas com a versão, p. ex. {3.7.2}
3. Ver [página com informações sobre o IDLE](#).
4. Em todas as tabelas abaixo, supõe-se a execução prévia dos comandos de atribuição A=1; B=2; C=3; D=1.2; E=2.3; F=3.4; G='abc'; H='def'
5. Nas expressões usando operadores e funções, espaços em branco são ignorados. Assim, pode-se escrever A = 1; X= 2+ B; etc. Símbolos de operadores, nomes de funções, de variáveis e de comandos não podem conter espaços em branco. Assim, A b e + = não são considerados como a variável Ab e o operador +=, e sim como A e b, e + e = separados (resultam em erros de sintaxe).
6. Comentários: # define o resto da linha como comentário (isto é, ignorado pelo interpretador); "'''" ... "'''" define tudo entre os "'''" como comentário, inclusive várias linhas de código.
7. Nos exemplos, as palavras reservadas são grafadas em **negrito**.
8. Nos exemplos, após um comando para o IDLE em uma só linha, deve-se executá-lo digitando um Enter; o eventual resultado aparece neste texto depois de "→", p.ex. A+B → 3 (após dar-se o comando >>>A+B e Enter, aparece 3 na próxima linha). Atenção: ao usar copiar/colar podem ser coladas várias linhas, mas todas devem formar apenas um único comando, por exemplo um **if** ou **while** com várias linhas. No entanto, podem-se dar 2 ou mais comandos em uma só linha, separados por ";": X=1; Y=2; X,Y → (1,2)
9. Algumas funções exigem a adição de um módulo especial ao programa, o que é feito com o comando **import**. Para detalhes sobre ele, ver <https://docs.python.org/3/library/importlib.html>
10. Ver uma página com [exemplos de programas completos](#) testados no ambiente Azure da Microsoft. Ver também [um site com muitos exemplos](#); os códigos estão nas abas "Test Suite".
11. Comentários, sugestões e críticas são muito bem vindos!!!
12. A seção 12 contém agradecimentos com o código dos colaboradores entre chaves {...}.

2. TIPOS, CONSTANTES, VARIÁVEIS E MATRIZES

2.1 Tipos de variáveis

A linguagem Python é *fracamente tipada*, isto é, uma variável x fica com o tipo do valor que ela recebe. Assim, se for executada a atribuição $x = 1$, x fica com o tipo inteiro; se depois é executado $x = 1.5$ daí para diante x fica com o tipo float, até mudar de tipo.

2.2 Tipos numéricos

Inteiro (int): precisão ilimitada. Ex.: 1234567890123456789012345

Constantes. Binária. Ex.: 0b101 ou 0B101(valor decimal 5); **octal:** 0o127 ou 0O127 (87); **hexadecimal:** 0xA5B ou 0XA5B (2651). Constantes hexadecimais são comumente usada para represenar números binários em uma notação mais fácil de ser lida.

Ponto flutuante (float): usualmente implementado usando o tipo double da linguagem C. Ex:

.12345678901234567890 → 0.12345678901234568 (note-se o arredondamento).

Complexo: contem uma parte real, e uma imaginária indicada por um j . Exs.: constante $(1+2j)$ ou $(1+2J)$; variável $(A+D*1j)$

Para converter um tipo para inteiro usar as funções `int()`, para ponto flutuante `float()`, para complexo `complex()`. Essas funções devem ser usadas especialmente na entrada de dados, `input()` – que dá sempre um tipo str de cadeia de caracteres.

O módulo NumPy (ver as referências), que deve ser instalado, permite o uso de uma grande variedade de tipos numéricos. Para definir uma variável tipo ponto flutuante de 32 bits, basta dar, p. ex.

`x = numpy.float32(1.0)`

2.3 Tipo cadeia de caracteres (*string*)

Exemplo: `'tuv5xyz: '` → `'tuv5xyz: '`; `"tuv5xyz: "` → `'tuv5xyz: '`; `'tuv5x"yz: '` → `'tuv5x"yz: '`; Se é executado o comando `Cad='Esta é uma cadeia'` então `Cad` → `'Esta é uma cadeia'`

Cadeia vazia: `"` (dois apóstrofes) → `"` ou `""` → `"`. Atenção: `" "` → `' '` (um espaço em branco)

Indexação: o 1º índice indica o elemento inicial, começando em 0; o 2º o final, começando em 1: `CAD[0]` → `'E'`; `Cad[5:10]` → `'é uma'`; `Cad[10:]` → `'é uma cadeia'`

Concatenação de cadeias: `'tuv5xyz: ' + Cad` → `'tuv5xyz: Esta é uma cadeia'`; `2*Cad` → `'Esta é uma cadeiaEsta é uma cadeia'`.

2.4 Tipo n -pla ordenada (*tuple*)

Exemplo: `Tup = (A, 5, 3.14, 'bla')`; `Tup` → `(1, 5, 3.14, 'bla')`. (Para o valor de A ver o item 3 das observações.)

n -upla vazia: `()`

Indexação: `Tup[0]` → `1`; `Tup[1:3]` → `(5, 3.14)`; `Tup[2:]` → `((3.14, 'bla'))`; `2*Tup` → `(1, 5, 3.14, 'bla', 1, 5, 3.14, 'bla')`.

É *inválido* atribuir um valor a um elemento de uma tupla: `Tup[1] = 10`

2.5 Tipo lista (*list*)

Exemplo: `L = [A, 5, D, 'bla']`

Lista vazia: `[]`

Indexação: `L[2]` → `1.2`; `L[1:4]` → `[5, 1.2, 'bla']`. Note-se que os índices dos elementos vão de 0 a, digamos, n . `L[i:j]` indica os elementos começando no elemento de índice i e terminando no de índice $j-1$. `L[4]` → dá um erro pois não há elemento de índice 4. `L[4] = 9` → dá o mesmo erro. Para adicionar mais um elemento (sempre no fim da lista): `L = L + [9]`; `L` → `[1, 5, 1.2, 'bla', 9]`; `L = L + []` não altera `L`.

Atribuição a um elemento: é *válido* fazer `L[3]=10`; `L` → `[1, 5, 1.2, 10]`.

2.6 Tipo dicionário (*dictionary*)

Exemplo: `Dic = {5:10, 3:'bla', 'ble':'A', 'A':'8'}`; `Dic` → `{5: 10, 3: 'bla', 'ble': 'A', 'A': 8}`

Indexação: `Dic[5]` → `10`; `Dic[3]` → `'bla'`; `Dic['ble']` → `1`; `Dic[3]` → `'bla'`; `DIC['A']` → `'8'`

Note-se que cada elemento de um dicionário é da forma $x:y$, onde x é o índice e y é o valor associado a esse índice. Índices ou valores que são alfabéticos devem estar entre apóstrofes ou aspas.

Obtenção de **todos os índices** (*keys*): `Dic.keys()` → `dict_keys([1, 'ble', 3, 5])`

Obtenção de **todos os valores**: `Dic.values()` → `dict_values([8, 1, 'bla', 10])`

`dict_keys` e `dict_values` não podem ser indexadas. Para se trabalhar com todos os índices ou os valores, pode-se transformá-los em listas e depois trabalhar com elas {MDG}:

`L = [x for x in Dic.keys()]; L` → `[5, 3, 'ble', 1]`

`L = [x for x in Dic.values()]; L` → `[10, 'bla', 1, 8]`

2.7 Tipo conjunto (*set*)

Exemplo: $S = \{1, 'dois', 3, 'quatro'\}$; $S \rightarrow \{3, 1, 'dois', 'quatro'\}$ (Conjuntos não são ordenados e não podem ser indexados.)

Conjunto vazio: $\{\}$

Usos. Número de elementos (cardinalidade): $\text{len}(S) \rightarrow 4$. Pertinência: $'dois' \text{ in } S \rightarrow \text{True}$. União: $S \cup \{5\} \rightarrow \{1, 3, 'dois', 5, 'quatro'\}$ (S não mudou!). Interseção: $S \cap \{1, 'dois'\} \rightarrow \{1, 'dois'\}$. Complementação: $S - \{1, 'dois'\} \rightarrow \{3, 'quatro'\}$. Testa superconjunto próprio: $S > \{1, 'quatro'\} \rightarrow \text{True}$. Superconjunto: \geq . Subconjunto próprio: $<$. Subconjunto: \leq . União exclusiva (elimina os elementos comuns): $S \wedge \{'dois', 5\} \rightarrow \{1, 3, 5, 'quatro'\}$

Há dois tipos de conjuntos: *set*, em que seus elementos podem ser mudados, e *frozenset* que não pode ser mudado. A construção de um set é automática. Ex.: $x = \text{frozenset}(\{1, 2\})$; $x.\text{pop}() \rightarrow \text{erro}$. Conjuntos internos de conjuntos têm que ser do tipo *frozenset*.

2.8 Tipo lista de intervalo de inteiros (range)

Padrão: $\text{range}(m, n, k)$, com n e k opcionais ou só k opcional, gerando os inteiros de uma lista r tal que $r[i] = m + k*i$, onde $i \geq 0$ e $i < \text{abs}(n)$

Exemplos: $\text{list}(\text{range}(5)) \rightarrow [0, 1, 2, 3, 4]$; $\text{list}(\text{range}(1, 5)) \rightarrow [1, 2, 3, 4]$; $\text{list}(\text{range}(0, 8, 2)) \rightarrow [0, 2, 4, 6]$; $\text{list}(\text{range}(-1, -10, -2)) \rightarrow [-1, -3, -5, -7, -9]$

Uso em comandos **for**: ver o comando no item 11 abaixo e no 2.5 acima.

2.9. Vetores e matrizes (arrays)

Ao contrário de quase todas as outras linguagens de programação, Python não tem o tipo *array*, pois os valores de variáveis podem ocupar tamanhos diversos. Para representar matrizes, usam-se listas, já que as mesmas podem ser indexadas como se fossem matrizes. Em outras linguagens, a declaração de um array produz a inicialização de seus valores, isto é, a declaração do array faz com que todos seus elementos passem a existir. Mas o mesmo não ocorre com as listas de Python; assim, é preciso adicionar cada elemento novo, pela ordem.

Exemplos

1. Para definir um vetor de 5 elementos, inicializando com valores 0 (poderia ser outro qualquer, como 1 ou uma expressão): $V = [0 \text{ for } I \text{ in range}(4)]$; $V \rightarrow [0, 0, 0, 0]$; note-se que $V[4]$ não existe e não pode receber algum valor, como $V[4]=0$ ou $V[4]=[0]$. Para defini-lo: $V = V + [0]$. Para varrer todo um vetor de tamanho variável, use $\text{len}(V) \rightarrow 4$

2. Para definir uma matriz bidimensional, constrói-se uma lista de listas: $M = [[1, 2], [3, 4]]$; $M \rightarrow [[1, 2], [3, 4]]$; $M[1][1] \rightarrow 4$; $M[0][1] \rightarrow 2$; $M[1][0] \rightarrow 3$. Inicialização, com vários valores, de uma matriz com 3 linhas e 4 colunas:

$M = [[I+J+1 \text{ for } I \text{ in range}(2)] \text{ for } J \text{ in range}(3)]$; $M \rightarrow [[1, 2], [2, 3], [3, 4]]$; $M[2][1] \rightarrow 4$

Para dimensões maiores, basta estender as receitas.

<p>3. Para varrer todos elementos e exibir os seus valores:</p> <pre>M=[[1, 2], [3, 4]] for I in M: for E in I: print (E) → 1 2 3 4</pre> <p>5. Para obter linhas:</p> <pre>for I in M: I → [1, 2] [3, 4]</pre> <p>6. Para obter colunas:</p> <pre>J=0 J = [I[J] for I in M] J → [1, 3]</pre>	<p>4. Para gerar uma matriz com valores crescentes:</p> <pre>M=[] for I in range(2): M[I]=M[I]+[-1] M → [-1, -1] J=1 for I in range(2): M[I]=[J,J+1] J=J+2 M → [[1, 2], [3, 4]]</pre> <p>{PC} Chamou a atenção para a inicialização do J</p> <p>7. Para aplicar uma função, p.ex. sqrt a todos os elementos de M:</p> <pre>[[math.sqrt(x) for x in I] for I in M] → [[1.0, 1.4142135623730951], [1.7320508075688772, 2.0]]</pre>
--	--

8. Módulo para uso de matrizes

O módulo NumPy permite o uso de vetores e matrizes como se fosse nas linguagens tradicionais. Supondo que ele está instalado, para ativá-lo e dar-lhe o nome interno de np dá-se **import NumPy as np**. Ver as referências para detalhes como usá-lo.

3. OPERADORES

Op	Significado e tipos dos operandos	Exemplos
+	Operador binário (com dois argumentos): soma de int, float, ou complex; concatenação de cadeias de caracteres (<i>strings</i>), de listas e de <i>n</i> -plas	$A+B \rightarrow 3$, $D+E \rightarrow 3.5$; $A+D \rightarrow 2.2$; $(A+D*1j)+(E+C*1j) \rightarrow (3.3+4.2j)$; $G+H \rightarrow 'abcdef'$; $(123, 'xyz') + ('L', '3.14') \rightarrow (123, 'xyz', 'L', 3.14)$ $[1, 2] + [3] \rightarrow [1, 2, 3]$
+	Operador unário (com um argumento): sem efeito	$+A \rightarrow 1$
-	Operador binário de subtração de int, float ou complex; complementação de conjuntos	$C-B \rightarrow 1$; $F-D \rightarrow 2.2$; $F-A \rightarrow 2.4$
-	Operador unário de troca de sinal	$-A \rightarrow -1$
*	Multiplicação int, float ou complex	$B*C \rightarrow 6$; $D*E \rightarrow 2.76$; $B*D \rightarrow 2.4$; $(A+D*1j)+(E+C*1j) \rightarrow (3.3+4.2j)$
/	Divisão int, float, resulta float ou complex	$C/B \rightarrow 1.5$; $F/E \rightarrow 1.4782608695652175$; $C/D \rightarrow 2.5$; $(A+D*1j)/(E+C*1j) \rightarrow (0.41287613715885235-0.016794961511546552j)$
//	Divisão de int, resulta int, ou de float por int ou float só parte inteira	$C//B \rightarrow 1$; $F//D \rightarrow 2.0$; $F//B \rightarrow 1.0$;
%	Resto int da divisão de ints, parte inteira se divisão de floats	$C\%B \rightarrow 1$; $F//D \rightarrow 2.0$; $F//B \rightarrow 1.0$
**	Potenciação de int, float ou complex	$B**C \rightarrow 8$; $B**D \rightarrow 2.2973967099940698$; $D**B \rightarrow 1.44$; $D**E \rightarrow 1.5209567545525315$; $(1+1j)**2 \rightarrow 2j$; $27**(1/3) \rightarrow 3$ (raiz cúbica)
==	Testa igualdade de int, float complex ou string, resulta True (verdadeiro) ou False (ver tabela de operadores lógicos); com vários == em um só comando dá True somente se cada um dos pares consecutivos for igual	$B==A*2 \rightarrow \text{True}$; $A==B \rightarrow \text{False}$; $A==D \rightarrow \text{False}$; $A==\text{int}(D) \rightarrow \text{True}$; $(A+B*1j)==(1+2j) \rightarrow \text{True}$; $G=='abc' \rightarrow \text{True}$; $G==H \rightarrow \text{False}$; $1==1==1 \rightarrow \text{True}$; $1==1==2 \rightarrow \text{False}$;
!=	Diferente, idem	$A!=B \rightarrow \text{True}$; $A!=D \rightarrow \text{True}$; $A!=\text{int}(D) \rightarrow \text{False}$; $(1+2j)!= (2+2j) \rightarrow \text{True}$; $G!=H \rightarrow \text{True}$; $1!=2!=3 \rightarrow \text{True}$; $1!=1!=2 \rightarrow \text{False}$;
>	Maior do que, idem sem complex; testa superconjunto próprio	$B>A \rightarrow \text{True}$; $A>B \rightarrow \text{False}$; $D>A \rightarrow \text{True}$; $E>D \rightarrow \text{True}$; $H>G \rightarrow \text{True}$; $3>2>1 \rightarrow \text{True}$; $3>2>3 \rightarrow \text{False}$;
<	Menor do que, idem; subconjunto próprio	$A<B \rightarrow \text{True}$; $B<A \rightarrow \text{False}$; etc.
>=	Maior ou igual, idem; superconjunto	$B>=A \rightarrow \text{True}$; $B>=D \rightarrow \text{True}$; $H>=G \rightarrow \text{True}$; etc.
<=	Menor ou igual, idem; subconjunto	$B<=A \rightarrow \text{False}$; etc.
&	"e" bit a bit de valores binários, resultado decimal no IDLE; intersecção de conjuntos	$0b0101 \& 0b0001 \rightarrow 1$; $\text{bin}(0b1100 \& 0b1010) \rightarrow '0b1010'$ (na exibição corta os 0s à esquerda)
	"ou inclusivo" bit a bit, resultado decimal no IDLE; união de conjuntos	$\text{bin}(0b1100 0b1010) \rightarrow '0b1110'$
^	"ou exclusivo" bit a bit, resultado decimal no IDLE; união exclusiva de conjuntos	$\text{bin}(0b0110 \wedge 0b1010) \rightarrow '0b1100'$
>>	deslocamento bit a bit para a direita inserindo 0s à esquerda (equivale a divisão por $\text{pow}(2,n)$)	$J = 10$; $\text{bin}(J) \rightarrow '0b1010'$; $\text{bin}(J>>1) \rightarrow '0b101'$ (zeros à esquerda não são exibidos)
<<	deslocamento bit a bit para a esquerda inserindo 0s à direita (equivale a divisão por $\text{pow}(2,n)$)	$J = 10$; $\text{bin}(J) \rightarrow '0b1010'$; $\text{bin}(J<<2) \rightarrow '0b101000'$
is	Teste de identidade de objetos	
not	Teste de não identidade de objetos	

is		
=	Atribuição simples ou múltipla, lado direito int, float, complex ou string; ambos os lados podem ser uma <i>n</i> -pla sem "(" e ")"	A=1; A → 1; A=D; A → 1.2 (A mudou de int para float); J=(A+D*1j); J → (1+1.2j); J, I, K = 10, 20, B*30; I → 20; J, K → (10, 60) Troca de valores de variáveis: X, Y= 3, 4; X, Y= Y, X; X, Y → (4, 3)
+=	x += y equivale a x = x + y	J=1; J+=2; J → 3; J=(1+2j); J+=(2+3j); J → (3+5j)
-=	x -= y equivale a x = x - y, inclusive de conjuntos	J=3; J-=2; J → 1
*=	x *= y equivale a x = x * y	J=2; J*=3; J → 6
/=	x /= y equivale a x = x / y	J=6; J/=3; J → 2.0
//=	x //= y equivale a x = x // y	J=15; J//=4; J → 3; J=15.5; J//=3.7; J → 4.0
%=	x %= y equivale a x = x % y	J=6; J%=4; J → 2.0; J=6; J%=2; J → 0
=	x **= y equivale a x = x ** y	J=2; J=3; J → 8 (Não disponível na versão 2 do Python)
>>=	x >>= y equivale a x = x >> y	J=0b1010; J>>=2; bin(J) → '0b10' (zeros à esquerda não são exibidos)
<<=	x <<= y equivale a x = x << y	J=0b1010; J<<=2; bin(J) → '0b101000'
&=	x &= y equivale a x = x & y, inclusive de conjuntos	J=0b1100; K=0b1010; J&=K; bin(J) → '0b1000'
^=	x ^= y equivale a x = x ^ y, inclusive de conjuntos	J=0b1100; K=0b0110; J^=K; bin(J) → '0b1010'
=	x = y equivale a x = x y, inclusive de conjuntos	J=0b1100; K=0b1010; J =K; bin(J) → '0b1110'
if	Expressão condicional: x = Valor-se-True if Expressão-Lógica else Valor-se-False	J = 1 if 4>3 else 2 → 1 J = 5 * (1 if B<A else 3*A); → 15

4. OPERADORES LÓGICOS (BOOLEANOS)

Op	Significado	Exemplos
	No que segue, supõe-se a execução prévia de L1T = True ; L2T = True ; L1F = False ; L2F = False	
True	Constante indicando "verdadeiro"	Atenção: true não é aceito, dá erro de variável não definida
False	Constante indicando "falso"	Idem para false; são considerados como False : None , 0, 0.0, 0j, '', (), [], {}; outros valores como True
not	Negação: muda True para False e vice-versa	not L1T → False; not L1F → True
x or y	"ou" inclusivo; dá falso somente se x e y forem falsos	L1T or L2T → True; L1T or L1F → True; L1F or L1T → True; L1F or L2F → False
x and y	"e"; dá verdadeiro somente se x e y forem verdadeiros	L1T and L2T → True; L1T and L1F → False; L1F and L1T → False; L1F and L2F → False
x in y	se x está na <i>string</i> , <i>n</i> -pla, lista ou conjunto y dá True, senão False	'a' in 'false' → True; 5 in (2, 5, 3) → True; 3 in [2, 5, 3] → True; 4 in [2, 5, 3] → False
x not in y	Contrário de in	

5. FUNÇÕES NATIVAS (tabela a ser complementada com mais significados e exemplos; [fonte](#))

Função	Significado	Exemplos
abs()	Valor absoluto; módulo no caso de complexo	abs(-1) → 1; abs(2) → 2; abs((1+2j)) → 2.23606797749979

x.add(y)	Inserir o elemento y no conjunto x	x = {1, 2, 'três'}; x.add('quatro'); x → {1, 2, 'quatro', 'três'}
all()		
any()		
basestring()		
bin()	Converte um int em binário	bin(B) → '0b10'; bin(20) → '0b10100'
x.bit_length()	Comprimento de bits significativos do binário x	0b101010.bit_length() → 6; 0b001010.bit_length() → 4
bool()	Retorna True se o argumento é verdadeiro, False se não há argumento ou ele é falso	bool(B>A) → True; bool(C>D) → False; bool() → False
bytearray()		
callable()		
chr()	Caractere correspondente ao código ASCII do argumento (entre 0 e 255)	chr(97) → 'a'; chr(150) → '\x96' (converteu para hexadecimal, pois não achou o caractere)
classmethod()		
clear(x)	Remove todos os elementos do conjunto x	x = {1, 2, 'três'}; x.clear(); x → set() ""indica conjunto vazio""; x = {9}; x → {9}
cmp(x,y)	Dá um int negativo se x<y, 0 se x==y, positivo se x>y	Não disponível no IDLE 3.6.1
compile()		
complex(re,im)	Converte em complexo com parte real re e imag. im	complex(1) → (1+0j); complex(2.5) → (2.5+0j)
x.conjugate()	Dá o conjugado do complexo x	x = (1+2j); x.conjugate() → (1-2j)
delattr()		
dict()		
dir()		
x.discard(y)	Remove o elemento y do conjunto x, se y está em x	x = {1, 2, 'três'}; x.discard('três'); x → {1, 2}
divmod(x,y)	Dá a dupla ordenada (x // y, x % y)	divmod(C,B) → (1,1), divmod(C,D) → (2.0, 0.6000000000000001)
enumerate()		
eval()		
execfile()		
file()		
filter(f, L)	Aplica a função f a cada elemento da lista L, e resulta nos elementos de L para os quais f for True	Ver exemplo no item "notação lambda"
float()	Converte para float	float(B) → 2.0;
float.as_integer_ratio()	Dá um par de inteiros cuja razão é o argumento	float.as_integer_ratio(1.5) → (3,2)
float.is_integer()	Dá True se o argumento for inteiro, False em caso contrário	float.is_integer(1.5) → False; float.is_integer(3.0) → True
format()		
frozenset()	Constroi um conjunto que não pode ser mudado	x = frozenset({1, 2}); x → {1, 2}; x.pop() dá erro
getattr()		

globals()		
hasattr()		
hash()		
help()	No modo interativo (IDLE), dá a documentação do argumento	help(int)
hex()	Converte um int para um hexadecimal	hex (8) → '0x8'; hex (50) → '0x32'; hex(C) → '0x3'
id()		
input()	Dá o valor da entrada de um dado em forma de <i>string</i> ; pode exibir uma mensagem	dia=input('Entre com o valor de dia →'); → "Entre com o valor de dia:"; 20; dia → 20 (como <i>string</i>); para calcular é preciso converter, p.ex. dia=int(input('Entre...'))
int()	Converte para int	int(D) → 1; int('123') → 123
x.isdisjoint(y)	True se o conjunto x é disjunto do conjunto y, False em caso contrário	{1, 2, 'três'}.isdisjoint({4, 'cinco'}) → True {1, 2, 'três'}.isdisjoint({2, 'cinco'}) → False
isinstance()		
issubclass()		
iter()		
s.join(x)	Método da classe string. Concatena as partes da lista, tupla ou conjunto x, separando-as com a <i>string</i> s.	SeparaComVirgulas = ','.join(['a', 'b', 'c']); SeparaComVÍrgulas → 'a,b,c'
len(x)	Dá o número de elementos de uma <i>string</i> , <i>lista</i> , <i>n-pla</i> ou conjunto (neste caso, a cardinalidade)	len(G) → 3; len((1,2,3,4)) → 4; len((A,B,G)) → 3; len([1,B,5,7]) → 4 len({1, 2, 'três'}) → 3
list()	Converte os elementos de uma <i>string</i> ou <i>n-pla</i> em lista; sem argumento dá a lista vazia	list(G) → ['a', 'b', 'c']; list((1, 2, 3)) → [1, 2, 3]
locals()		
long()		Não disponível no 3.61.1
lower()	Converte as letras uma <i>string</i> para minúsculas	'BLA3#'.lower() → 'bla3#'; nome = input('Digite seu nome:').lower(); print(nome)
s.lstrip('c')	Método da classe string. Elimina todos os caracteres c do começo da <i>string</i> s, ignorando brancos até c; se c é omitido, elimina os brancos no começo de s	s='x y z'; s.lstrip('x') → s=(' x y z '); s.lstrip('x') → ' y z'; s.lstrip() → 'x y z'; s=(' x y z '); s.lstrip() → 'x y z '; s.lstrip(' x,') → 'y z '; s.lstrip('x y') → 'z '
map(f,L)	Aplica a função f a cada elemento de uma lista L	list (map (abs, [2,-3,4,-5])) → [2, 3, 4, 5]
max()	Dá o maior dos elementos do argumento	max (1,2,3) → 3; max (['a', 'b', 'c']) → 'c'; max ('a', 'b', 'c') → 'c'; max (G) → 'c'
memoryview()		
min()	Como max, para o menor	
next()		
object()		
oct()	Converte para octal	oct(15) → '0o17'
open()		
ord()	Contrário de chr	ord ('a') → 97

x.pop()	Dá e remove um elemento arbitrário do conjunto x	x = {1, 2, 'três'}; x.pop(); x → 1; x → {2, 'três'}
pow(x,y)	Equivalente a x**y	pow(2,3) → 8; pow(4,0.5) → 2.0; pow (4,-2) → 0.625
print()	Saída de dados	print(A,D) → 1 1.2; print ('A =',A) → A = 1; print ('A*3 =',A*3,'\nD =',D) → A*3 = 3 D = 1.2
property()		
random	Uma classe. Exige importar o módulo random. Algumas funções dessa classe: 1. random.randint(a,b) retorna inteiro pseudo-aleatório entre a e b inclusive. 2. random.random() dá o próximo aleatório em ponto flutuante entre 0.0 e 1.0 3. random.uniform(a, b) idem entre a e b inclusive 4. random.choice(x) dá aleatoriamente um elemento da lista x não vazia 5. random.shuffle(x) ordena a lista x aleatoriamente	Os resultados obtidos com o IDLE podem ser outros, dependendo da "semente": 1. import random; random.randint(10, 100) → 12; random.randint(10, 100) → 67 2. random.random() → 0.05462293624556047; random.random() → 0.36903357168070205 3. random.uniform(2,5) → 3.983840861586745 4. random.choice([1,2,3,4,5])) → 3; random.choice([1,2,3,4,5])) → 4 5. x=[1,2,3,4]; random.shuffle(x); x → [3, 1, 4, 2]; random.shuffle(x); x → [4, 1, 2, 3]
range()	Cria virtualmente uma lista virtual	range(C): equivale a [0, 1, 2]; range(1, 5) a [1, 2, 3, 4, 5]; range(0, 10, 3) a [0, 3, 6, 9]; range(0, -4, -1) a [0, -1, -2, -3]
raw_input()		
reduce()		
reload()		
repr()		
reversed()		
s.rstrip(c)	Idem a lstrip, elimina a string c à direita da string s	
round(x,n)	x arredondado na n-ésima casa decimal; sem n arredonda para o inteiro	round(3.5566,3) → 3.557; round(4.5555,3) → 4.555; round(3.5555) → 4; round(3.4555) → 3
set()	Constroi um conjunto que pode ser mudado	x = {1, 2, 'três'}; x → {1, 2, 'três'}; x.pop() → {1}
setattr()		
slice()		
s.split(sep)	Método da classe string. Gera uma lista com os elementos da <i>string</i> s separados pela <i>string</i> sep; se sep for omitido, usa branco como separador	s='a,b,3'; s.split(',') → ['a', 'b', '3']; s='x y z'; s.split() → ['x', 'y', 'z']
sorted()	Ordena uma lista	sorted ([1,4,2]) → [1, 2, 4]; sorted([B,A]) → [1,2]
staticmethod()		
str()	Converte int ou float para string	str(C) → '3'; str(D) → '1.2'
c.strip(s)	Método da classe string. Dá a <i>string</i> s sem a <i>string</i> c no começo e no fim.	s = ' x y z '; s.strip() → 'x y z'; s = 'xyxyzzxyxyxy'; s.strip('xy') → 'zzz'

	Sem o c, considera brancos. Ver também lstrip e rstrip	
sum()	Soma os elementos de uma lista, n-pla ou conjunto	sum([A,B,C, 4, D]) → 11.2; sum((1,2,3)) → 6; sum ({1, 2, 3}) → 6
super()		
tuple()	Dá uma lista ordenada	tuple('abc') → ('a', 'b', 'c'); tuple([1, 2, 3]) → (1, 2, 3)
type()	Se o argumento for uma variável, dá seu tipo; se for um objeto, o tipo do mesmo	type (A) → <class 'int'>; type (D) → <class 'float'>; type (G) → <class 'str'>
upper()	Converte as letras de uma string para maiúsculas	'bla3#'.upper() → 'BLA3#'
unichr()		
unicode()		
vars()		
xrange()		
zip()		
__import__()	Importação de módulos. É o mesmo que o comando import.	Ver https://docs.python.org/3/library/importlib.html

6. ALGUMAS FUNÇÕES E CONSTANTES MATEMÁTICAS

Para usar essas funções, é necessário executar no IDEL ou inserir em um programa o comando **import** math, que ativa o módulo (*module*) math, e preceder cada função de math., p.ex. math.sqrt(4), math.e etc; os resultados são sempre do tipo float, a menos de observação em contrário. Para cálculos com números complexos, dar **import** cmath .

Função	Significado	Exemplos
atan(x)	Arcotangente, resultado em radianos	math.atan(2) → 1.1071487177940904;
ceil(x)	O menor inteiro >= x	math.ceil(4.7) → 5
cos(x)	Cosseno, x em radianos	math.cos(math.pi/2) → 6.123233995736766e-17 (devia ser zero; não é devido à aproximação); math.cos(math.pi) → -1.0
degrees(x)	Converte x em graus para radianos	math.degrees(math.pi) → 180.0
e	A constante e	math.e → 2.718281828459045
exp(x)	e**x	math.exp(1) → 2.718281828459045; math.exp(2) → 7.38905609893065
factorial(x)	Fatorial de x de tipo int, resultado int	math.factorial(5) → 120
floor(x)	Maior int <= x	math.floor(4.7) → 4
fsum	Somatória	Como a sum() da tabela Funções Nativas, mas arredondando
inf	A constante infinito (maior número em float representável)	math.inf → inf
log(x, base)	Logaritmo de x na base (opcional); sem base dá o log na base e	math.log(10) → 2.302585092994046; math.log(100,10) → 2.0
log10()	Logaritmo na base 10	math.log10(100) → 2.0; em geral mais precisa que math.log(x,10)
log2()	Logaritmo na base 2	math.log2(8) → 3.0

modf(x)	Dá a parte decimal e a inteira de x	math.modf(1.25) → (0.25, 1.0)
pi	O número pi	math.pi → 3.141592653589793 (ver exs. em sen, cos, tan)
radians(x)	Converte x em radianos para graus	math.radians(180) → 3.141592653589793
sin(x)	Seno, x em radianos	math.sin(math.pi/2) → 1.0; math.sin(math.pi) → 1.2246467991473532e-16 (devia ser zero; não é devido à aproximação)
sqrt()	Raiz quadrada	math.sqrt(4) → 2.0; math.sqrt(5.6) → 2.3664319132398464
tan(x)	Tangente, x em radianos	math.tan(math.pi) → 1.2246467991473532e-16 (devia ser zero); math.tan(math.pi/2) → 1.633123935319537e+16 (representação do infinito)
trunc(x)	Parte inteira de x	math.trunc(3.5) → 3

7. OUTRAS FUNÇÕES

Função	Significado	Exemplos
append()	Concatena uma lista a outra	L=[]; L → []; L.append('a'); L → ['a']; L.append('bc'); L → ['a', 'bc']
exit()	Encerra a execução de um programa	Necessita o módulo sys, incorporado com import sys ; uso da função: sys.exit() (Sugestão dada por Thiago Salgado scrimforever arroba_at gmail ponto com)

8. PRECEDÊNCIA (ORDEM DE EXECUÇÃO) ([fonte](#))

Ordem	Operador/função	Exemplos
1	(...)	
2	função()	abs(-5)+2 → 7
3	+ e - unários	-5-2 → -7; -(5-2) → -3
4	*, /, % e //	
5	+ e - binários (soma e subtração)	
6	& ("e" bit a bit)	
7	e ^	
8	<=, <, >, >=	
9	=, %=, /=, //= e -=	
10	+=, *= e **=	
11	in, not in	
12	not, or, and	

9. DECLARAÇÃO E USO DE UMA FUNÇÃO

Sintaxe	Exemplos no IDLE
# Declaração (atenção para o alinhamento vertical) def nome-da-função (parâmetro-1, ...,	>>> def soma(a,b): # declaração return a+b

parâmetro- <i>m</i>): comando-1 ... comando- <i>n</i> próximo-comando # Uso da função nome-da-função (argumento-1, ..., argumento- <i>m</i>)	<pre>>>> soma(1,2) → # ativação 3 >>> soma (A*3,D) → 4.2</pre>
--	--

Na sequência de um programa, a declaração de uma função deve sempre vir *antes* de sua ativação.

10. NOTAÇÃO LAMBDA

Essa notação permite que se declare uma função sem dar-lhe um nome, colocando-a em qualquer lugar em que uma função possa ser chamada.

Sintaxe	Exemplos no IDLE
lambda lista-de-argumentos: função desses argumentos	<pre>>>> y = lambda x: x**2 >>> y(8)→ 64 >>> min = lambda x,y: x if x < y else y >>> min (3,2) → 2 >>> itens = [1, 2, 3, 4, 5]; >>> list(map(lambda x: x**2, itens)) → [1, 4, 9, 16, 25] >>> lista_de_nos = range(-5, 10) >>> list(filter(lambda x: x < 0, number_list)) → [-5, -4, -3, -2, -1]</pre>

11. IDENTIFICADORES GLOBAIS E LOCAIS

Um identificador declarado dentro de uma função é somente local a ela (válido dentro dela); declarado fora dela, em um escopo (isto é, espaço de validade) englobando diretamente a função ele é global, pode ser usado tanto fora como dentro dela. O uso de um identificador local evita muitos erros, pois só a função onde ele está declarado pode modificar seu valor; esse identificador fica "encapsulado" na função. Nesse sentido, o correto é passar valores para a função e obtê-los dela por meio de parâmetros na sua declaração (argumentos na sua ativação).

<pre>>>> def F(): Loc = 1 # local a F print (Loc, Glob) >>> Glob = 2 # global a F >>> F() → 1 2 >>> Loc → Loc NameError: name 'Loc' is not defined</pre>	Para converter um identificador local em global: <pre>>>> def F(): global Glob print (Glob) >>> Glob=1 >>> F() → 1</pre>
--	--

Uma função F2 pode ser declarada dentro de uma outra função F1. Nesse caso, F2 torna-se local a F1 e não pode ser ativada fora de F1:

<pre>>>> def F1(): def F2(): # local a F1! LocF2 = 1 print (LocF2) >>> F2() → F2() NameError: name 'F2' is not defined</pre>	<pre>>>> def F1(): def F2(): LocF2 = 1 print (LocF2) F2() → # ativada dentro de F1 >>> F1() 1</pre>
--	---

Se a função F2 estiver declarada dentro de F1, a declaração **nonlocal** faz com que uma variável V declarada em F2 passe a ter o escopo de F1, mas não é válida fora de F1. V tem que ter um valor atribuído a ela em F1 *antes* da declaração **nonlocal**:

<pre>>>> def F1(): LocF2 = 1 # necessário! def F2(): nonlocal LocF2 LocF2 = "LocF2" print ("Na F1:", LocF2) F2() print ("Passou pela F2:", LocF2) >>> F1() → Na F1: 1 Passou pela F2: LocF2</pre>	<pre>>>> LocF2 = 1 >>> def F1(): def F2(): nonlocal LocF2 LocF2 = "LocF2" print ("Na F1:", LocF2) F2() print ("Passou pela F2:", LocF2) → SyntaxError: no binding for nonlocal 'LocF2' found</pre>
---	--

12. CLASSES

(Em construção.)

Classes podem ser conceitualmente encaradas como uma extensão das funções, e são usadas para se obter mais encapsulamento. Ao contrário das funções, as classes não contêm parâmetros; as classes podem conter declarações de variáveis, que se tornam locais a elas. Cada elemento declarado em uma classe é denominado "atributo" da classe, e é referenciado pelo nome da classe, um ponto e o nome do atributo. Como as funções, as classes devem ser definidas antes de serem usadas. As funções declaradas dentro de uma classe são denominadas *métodos*. Podem-se atribuir valores aos elementos de uma classe fora dela. Um classe pode ser atribuída a uma variável V; nesse caso V recebe uma instância da classe, um objeto com todas as propriedades da classe.

```
>>> class C:
    """ Esta é uma classe """ # Atributo implícito __doc__
    X = 1
    def FdeC (Y):
        return Y + 1

>>> C.X →
1
>>> C.FdeC(2) →
3
>>> C.X = 5
>>> C.X →
5
>>> C.__doc__ →
' Esta é uma classe '
>>> ObjC = C # Instanciação: criação de um objeto
>>> ObjC.X →
5
```

13. COMANDOS COMPOSTOS

Sintaxe	Exemplos (testados no Azure, V. Ambientes abaixo)
Bloco (de comandos): Comando; Comando; ... ; Comando #todos na mesma linha ou todos alinhados verticalmente à esquerda, ou em alguma coluna, se o	<pre>J = 2; K = 3; M = 4; J,K,M → (2,3,4) J=2 K=3 M=4 J,K,M → (2,3,4)</pre>

bloco estiver imerso em algum comando	
Comando Comando ... Comando	
Comando if de escolha lógica	J = 2; K = 3; L= 4 # válido para todos os exemplos seguintes
# atenção para o alinhamento vertical	
if Expressão	if J < K: print(J); print(K)→
Lógica:	2
Bloco	3
#qualquer coluna a partir da 2a. em relação ao if	if K>J : "" O ":" pode estar em qualquer coluna desta linha ""
elif Expressão	N = J+3
Lógica:	P = K+4
#opcional, alinhado na mesma	N, P → (5,7)
#coluna que o if	{PC} corrigiu o (5, 7)
Bloco #a partir de qualquer coluna	if K < J: N=5
elif Expressão	else: N=6
Lógica:	P=7
#opcional, a partir da mesma	N, P → (6,7)
#coluna que o if	
Bloco #a partir de qualquer coluna	
...	
elif Expressão	
Lógica:	
#opcional, a partir da mesma	
#coluna que o if	
Bloco #a partir de qualquer coluna	
else:	
""Opcional.	
No IDLE, o else deve começar na 1a. coluna; segue um só Comando ou (exclusivo)	
	if K < J: N=5
	elif L>K: N=6; P=9
	N, P → (6,9)
	if K<J: N=5
	elif L<K: N=6; N
	elif J>K:
	N=7
	N
	else: N=8
	N → 8

um Bloco com várias linhas começando na próxima a partir da 2ª coluna em relação ao **else**"""
Próximo-Comando
#alinhado na coluna do **if**

Comando while de repetição da execução (malha de execução, *loop*) de um bloco de comandos

while
Expressão Lógica:
Comando;
"""opcional: um só comando! Ou (exclusivo!)"""
Bloco
"""comandos alinhados verticalmente à direita do w"""
else: Bloco
numa só linha ou Bloco próximo comando

O comando **else** é executado quando a Expressão Lógica der valor **False**
O comando **break** interrompe a execução da malha de repetição e desvia para o próximo comando após o **while** com seu bloco. É conveniente usá-lo quando ocorre uma situação de exceção durante a

Atenção: ao usar o comando **while** no IDLE, ele é executado até o fim (até Expressão Lógica ficar com valor **False**) antes de se poder dar o próximo comando.
L = 4

```
M = 1
while M < L: M
+= 1;
    print(M) →
4
```

```
M = 1
while M < 4:
    print(M)
    M += 1 →
1
2
3
print(10) →
10
```

```
M = 1
while M < 4:
    print(M)
    M += 2
else:
    print(7); print(8) →
1
3
7
8
print(10) →
10
```

```
# Exemplo de malha de
# execução infinita e
# break
I = 1
L = []
while True:
    I = I + 1
    L.append(I)
    if I > 4:
        break
    print(I) →
2
3
4
print(L) →
[2, 3, 4, 5, 6]
```

execução da malha.			
--------------------	--	--	--

<p>Comando for de repetição</p> <p>for Lista de Variáveis in Lista de Expressões: Bloco else: #Opcional Bloco</p> <p>Os valores da Lista de Expressões são atribuídos às variáveis; para cada atribuição o Bloco do for é executado uma vez.</p> <p>Quando a lista é esgotada, é executado o bloco do else, se este existir.</p> <p>O comando break interrompe a execução da malha de repetição e vai para o próximo comando depois do for.</p>	<pre>for I in range(3): #começa em 0! print(I) → 0 1 2 for I in range(2,4): # começa em 2 I # e termina em 3 (4-1)! → 2 3 Frutas = ['caju', 'caqui', 'manga'] for fruta in Frutas: print ('Fruta da vez:', fruta) → Fruta da vez: caju Fruta da vez: caqui Fruta da vez: manga for I in range(3): if I==2: break print(I) → 0 1</pre>	<pre>for letra in 'xyz': print ('Letra da vez:', letter) → Letra da vez: x Letra da vez: y Letra da vez: z for I in range(3): """2 fors encaixados""" for J in [2,'xy']: # e uso do else print ('I =", I, 'J =", J] else: print ('Passou por aqui!') I = 0 J = 2 I = 0 J = xy Passou por aqui! → I = 1 J = 2 I = 1 J = xy Passou por aqui! I = 2 J = 2 I = 2 J = xy</pre>
--	--	--

Uso do **for** em Varredura de estruturas

<pre>for i in [1, 2, 3]: print (i) → 1 2 3 for i in (1, 2, 3): print (i) → 1 2 3</pre>	<pre>for car in "123": print (car) → 1 2 3 for indice in {'um':1, 'dois':2}: print (indice) → dois um</pre>
---	--

14. PALAVRAS RESERVADAS

As seguintes palavras não podem ser usadas como nomes de identificadores (variáveis e funções):

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

15. REFERÊNCIAS

15.1 Tutoriais

- Tutorial livre: www.educba.com/category/software-development/software-development-tutorials/python-tutorial/
- Tutorial em português da UFF (Python versão 2!): www.telecom.uff.br/pet/petws/downloads/tutoriais/python/tut_python_2k100127.pdf
- Tutorial "oficial": <https://docs.python.org/3.5/tutorial>

- Tutoriais sobre operadores: www.programiz.com/python-programming/operators; https://www.tutorialspoint.com/python/python_basic_operators.htm
- Tutorial sobre variáveis, constantes e tipos: www.tutorialspoint.com/python/python_variable_types.htm
- Tutorial sobre [uso de listas](#) com muitos exemplos e testes de conhecimento
- Tutorial sobre o uso de matrizes (*arrays*): www.i-programmer.info/programming/python/3942-arrays-in-python.html
- Tutorial sobre o uso do módulo NumPy para uso de matrizes: www.i-programmer.info/programming/python/5785-advanced-python-arrays-introducing-numpy.html?start=1
- Tutorial sobre classes: <https://docs.python.org/3/tutorial/classes.html>
- Tutorial sobre módulos: <https://docs.python.org/2/tutorial/modules.html>

15.2 Outras referências

- <https://docs.python.org/2/library/functions.html> (de onde foi tirada a tabela de funções nativas)
- <https://ddi.ifi.lmu.de/probestudium/2012/ws-i-3d-programmierung/tutorials/python-referenzkarte> (em inglês); essas folhas de consulta são chamadas de *Cheat Sheet* (cola); melhor seria *reference sheet*.
- *Cheat sheet* dirigida a administradores de redes, com lista de *libraries*, métodos e módulos, produzidas por PCWDSL: www.pcwld.com/python-cheat-sheet
- https://s3.amazonaws.com/assets.datacamp.com/blog_assets/PythonForDataScience.pdf
- Tipos: <https://docs.python.org/3/library/stdtypes.html>
- Funções matemáticas: <https://docs.python.org/2/library/math.html>
- Funções com números complexos: <https://docs.python.org/3/library/cmath.html#module-cmath>
- Precedências: www.tutorialspoint.com/python/operators_precedence_example.htm
- Manual de referência: <https://docs.python.org/3.5/>
- Site "oficial": <https://www.python.org/>
- Grupo de discussão de Python no Brasil: <https://python.org.br/>
- Livro: Nilo Ney Coutinho Menezes, *Introdução à Programação com Python: Algoritmos e lógica de programação para iniciantes*, 2ªed, Novate, 2018

16. INSTALAÇÃO DO PYTHON E USO DO INTERPRETADOR IDLE

1. Instalação

Para instalar o Python e usar o seu interpretador IDLE (*Integrated Develop and Learning*): <https://www.python.org/downloads>

2. Uso do IDLE no Windows

1. Em meu W7, em 14/2/19 o Python 3.7.2 foi instalado automaticamente no diretório (os diretórios iniciais serão pulados) ...\\AppData\\Local\\Python\\Python37-32, no programa python.exe. Para descobrir o caminho desse diretório, acione com o botão direito o ícone do Python se ele estiver na área de trabalho, ou na lista de programas preferidos do menu Iniciar, em seguida Propriedades, aba Atalho; o caminho está no campo Destino e pode ser copiado. Acionando o programa Python, aparece uma janela do IDLE, parecida com uma janela de *prompt* do Windows, aqui chamada "janela de comandos", com o caminho na primeira linha, o cabeçalho da janela. Depois da carga (pode levar algum tempo) aparece o *prompt* do Python, >>> e se podem dar comandos da linguagem. **2.** A edição de um comando na janela de comandos do Python segue o padrão do Windows e deve ser feita acionando o botão direito do mouse na linha de cabeçalho. Aparece um menu principal com as opções Restaurar, Mover etc. até Propriedades. Para editar os comandos, use a opção Editar, aparecendo um submenu Marcar, Copiar, Colar etc. **3.** Ativando-se a opção Marcar pode-se selecionar um trecho qualquer da janela de *prompt* na vertical, inclusive os espaços em branco. **4.** Para copiar um trecho selecionado, use a opção Copiar ou acione a tecla Enter; a seleção desaparece, o cursor vai para o último *prompt* (>>>); o trecho copiado vai para a área de descarte do Windows. **5.** Para colar, use a opção Colar, que aparece em letras com cor normal; se não há nada na área de descarte as letras ficam mais claras. A parte copiada está na área de descarte do Windows, ela pode ser copiada de qualquer janela do Windows ou colada em qualquer outro editor. **6.** Acionando-se Propriedades no menu principal, e depois Modo de edição rápida, não é preciso mais usar a opção Marcar do submenu para selecionar um trecho, bastando passar o cursor sobre ele, com a tecla esquerda pressionada. **7.** As teclas Ctrl+C não servem para colar. Na janela de *prompt*, provocam a execução do comando da última linha, a menos de algum trecho da janela de comandos estar selecionado, caso em que a seleção é apagada; as teclas Ctrl+V não servem para colar (produzem a inserção de ^V na linha de comando). **8.** Várias janelas de comandos podem ser abertas, ativando-se a Python mais de uma vez como descrito no item **1** acima, permitindo copiar e colar de uma para outra. **9.** No meu Windows, a tecla F5 insere na próxima linha de comando o último comando executado, depois o penúltimo e assim por diante; a tecla F7 abre uma janelinha com os últimos comandos executados, que podem ser copiados e colados na janela de comandos. **10.** Para fechar a janela de comandos, dê quit() ou, no início da próxima linha de comando, Ctrl+Z e Enter. **11.** É possível ativar o IDLE diretamente de uma janela normal de *prompt* do Windows. Para isso, nessa janela desvie (com o comando >cd *diretório*) para o diretório \\Python\\Python37-32 e dê o comando >python.

3. Execução de um programa armazenado

Para ativar no IDLE um programa de nome prog.py armazenado (a extensão py é essencial), se ele estiver no mesmo diretório do python.exe basta dar na janela de *prompt* do Windows (atenção, não é o *prompt* do IDLE!) o comando >python prog.py, senão é preciso especificar o caminho do programa, p.ex. >python c:\dir-python\prog.py. Para mudar o diretório ativo do *prompt* do Windows, use o comando cd e dê o caminho que pode ser copiado como descrito em 1. do item anterior. Essa maneira de ativar um programa é muito útil pois é possível armazená-lo localmente, tendo-o editado previamente antes de o executar, inclusive testando alterações. Para editar um programa, use o Bloco de Notas do Windows, no campo Tipo escolha Todos os arquivos, em Codificação escolha UTF-8, coloque no nome a extensão .py e salve o programa. Uma outra possibilidade é salvar como .txt e depois mudar a extensão para .py. Para isso, use por exemplo o programa Total Commander, que eu uso em lugar do Windows Explorer, pois é muito mais prático, já que apresenta duas janelas, podendo-se mover ou copiar de uma para a outra, podem-se fazer buscas por nomes de arquivos ou diretórios, mudar extensões dos nomes etc. Ao salvar um programa, passe para a janela do *prompt* do Windows e, se já tiver ativado o programa uma vez, use Shift+F3 (repete o último comando) para que o caminho e o nome do programa seja inserido na linha.

4. Uso do depurador do IDLE (debugger)

Para detectar erros em um programa armazenado localmente, pode-se usar o debugger do IDLE, que nos comandos de um programa é abreviado por Pdb (de Python debugger).

1. Em um programa armazenado por exemplo com o nome prog.py, como exemplificado acima, deve-se no começo dele inserir o comando **import pdb** para incluir o módulo do Pdb. **2.** Para iniciar a execução do Pdb em algum ponto do programa, deve-se inserir no programa um ponto de interrupção (*breakpoint*) com o comando **pdb.set_trace()**. **3.** Quando o programa for executado, como exemplificado no item 3 acima, ao ele atingir o comando pdb.set_trace, vai aparecer na janela do IDLE uma nova linha com o comando seguinte (ainda não executado) do programa precedido de ->; com isso sabe-se em que linha do programa se está. Em seguida aparece em uma nova linha o seguinte: <Pdb>. **4.** Depois desse <Pdb> deve-se dar um comando para o Pdb, como **n** ("vá para a próxima linha do programa", de *new line*), quando o comando atual é interpretado e é exibido um -> com a próxima linha do programa; isso pode ser repetido, executando-se o programa passo a passo (comando a comando). **5.** Para exibir o valor de uma variável, depois de um <Pdb> deve-se dar o comando **p** (de "print") seguido do nome de uma variável, por exemplo p A para exibir o valor de A naquele ponto da execução. **6.** O comando **l** (letra ele minúscula, de *list*) do Pdb produz a exibição de alguns comandos do programa acima e abaixo do ponto em que a execução está parada; esse ponto é marcado com um -> na frente do próximo comando do programa a ser executado; isso é útil quando um programa tem várias cópias de um mesmo comando. **7.** Apertando-se a tecla **Enter** é repetido o último comando do Pdb que foi dado. **8.** Se o próximo comando a ser executado for uma função definida no programa (com o comando **def**), dando-se o comando **s** do Pdb ele entra dentro do código da função executando-o passo a passo. **9.** Dando-se o comando **r** do Pdb é executado sem interrupção todo o código da função sendo depurada, até o seu comando **return**. **9.** O comando **c** do Pdb encerra a execução passo a passo do Pdb até ela atingir o próximo ponto de interrupção (ver o item 2 acima). **10.** O comando **q** (de "quit") do Pdb encerra totalmente a execução do depurador e o programa é executado sem parar. **11.** Pode-se disparar a execução do depurador ao ativar um programa, como por exemplo >python -m prog.py; é necessário importar o módulo pdb dentro do programa (ver o item 1 acima). Para mais detalhes, ver <https://www.machinelearningplus.com/python/python-debugging/>

17. CURSO

- Curso de Python em português no Coursera (grátis sem certificado), por Fábio Kon (IME-USP): <https://www.coursera.org/learn/ciencia-computacao-python-conceito>
- **Atenção:** nesse curso o Prof. Kon dá exemplos nos quais ele edita um programa em um editor de textos, depois armazena-o por exemplo no arquivo de nome programa.py. Em seguida, ele ativa o interpretador da Python usando o comando ou algo parecido ...> python35 programa.py. Ocorre que o IDLE usa a codificação de caracteres UTF-8, de modo que o texto do programa deve ser armazenado localmente nessa codificação, que em alguns editores de texto pode ser selecionada. No Windows, que normalmente usa a codificação Latin-1 (não aceita pelo IDLE), o Bloco de Notas tem essa opção, na opção Salvar como → Codificação e seleciona-se UTF-8. Idem para os programas que o curso exige serem enviados para o Coursera para serem executados e avaliados nesse sistema. Se isso não for feito, ao executar o programa aparece uma mensagem de erro de caractere não reconhecido.

18. TEXTOS, AMBIENTES DE PROGRAMAÇÃO, DOCUMENTAÇÃO E FÓRUMS DE PROGRAMAÇÃO

- [Livro de Luciano Ramalho](#)
- [Jupyter](#): permite programar, guardar e executar programas na nuvem do sistema usando um navegador (não é preciso instalar o IDLE), e escrever documentos
- [Azure](#): ambiente da Microsoft; inclui o Jupyter e provê recursos adicionais. Programas ficam em notebooks. Para criar um, entre em sua conta, acione Notebooks (no menu à esquerda) e depois + New (no canto inferior esquerdo). Em cada célula de um notebook é possível colocar um programa e modificá-

lo; para executá-lo, selecione-o e dê Shift Enter. Uma variável ou função declarada em uma célula já executada é válida nas outras células. Aparentemente, não inclui o depurador (*debugger*) do IDL.

- [GitHub](#) é um forum, uma comunidade de programadores, onde os participantes inserem programas. Lá se encontra na íntegra o livro *Python Data Science Handbook* de Jake VanderPlas. {LF}
- [PyCharm](#), um ambiente de programação com versão *open source* (sem custo). {LF}
- [Anaconda](#), um ambiente *open source* de programação incluindo sistemas para *machine learning*.
- Lista de [grupos de discussão](#) de Python, por estado.
- [Eventos sobre Python](#).

19. AGRADECIMENTOS

- Colaborações (fazer uma busca na página com as abreviaturas): {LF} Luís Felipe Carvalho; {MDG} Marco Dimas Gubitoso; {PC} Paulo César Zandona Vieira achou 2 errinhos.