60h – Python

PROGRAMACAO EM PYTHON - INTERMÉDIO

julio.magalhaes@formandos.cinel.pt

INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL
ASSOCIAÇÃO PORTUGUESA DAS EMPRESAS DO SECTOR ELÉCTRICO E ELECTRÓNICO
NORTE2020
PORTUGAL 2020
UNIÃO EUROPEIA
Fundo Social Europeu

Objetivos

- Desenvolver pequenos scripts.
- Conhecer tipos de armazenamentos mais complexos
- Biblioteca gráfica

Python

- 2001 criada Python Software Foundation.
- Apoiada pela Microsoft, Google, …
- É incorporada em Sistemas Operativas BSD, MacOs, Linux
- Código aberto (Open Source)
- Tudo é encarado como sendo um objeto (até mesmo uma simples variável)
- Inúmeras bibliotecas com fins variados - versatilidade

Tipagem em Python

- Números inteiros
  - Números sem parte decimal (com ou sem sinal)
- Números reais
  - Números com parte decimal (com ou sem sinal)
  - Normalmente representados em notação científica (muito grandes ou muito pequenos)

Tipagem em Python

- Valores lógicos
  - São representados por True e False
- Cadeia de caracteres
  - Sequência de caracteres. São representadas por plicas (‘) ou aspas(“)
  - As plicas ou aspas não fazem parte da cadeia de carateres. Assim, o seu comprimentos diz respeito apenas aos caracteres que constituem essa sequência
  - ‘Bom dia’ é uma cadeia com 7 caracteres (espaço também conta)

Expressões compostas

- Para além de constantes, em Python existe um certo número de operações embutidas (ou pré-definidas), que ele conhece independentemente de qualquer indicação que seja fornecida por algum programa.
- Uma expressão composta é constituída por um operador e por um certo número de operandos.
- Operadores podem ser unários (not, - , ….) ou binários (+ , *, ….)

Operações sobre números inteiros

Operação	*Tipo dos argumentos*	*Valor*
$e_1 + e_2$	Inteiros	O resultado de somar e_1 com e_2.
$e_1 - e_2$	Inteiros	O resultado de subtrair e_2 a e_1.
$-e$	Inteiro	O simétrico de e.
$e_1 * e_2$	Inteiros	O resultado de multiplicar e_1 por e_2.
e_1 // e_2	Inteiros	O resultado da divisão inteira de e_1 por e_2.
e_1 % e_2	Inteiros	O resto da divisão inteira de e_1 por e_2.
abs(e)	Inteiro	O valor absoluto de e.

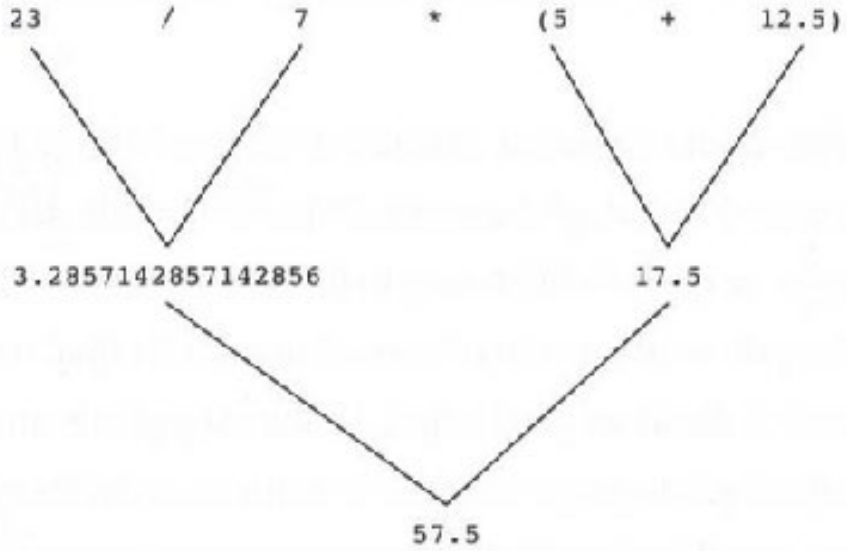
** - potência (n**y => n^y) ou utiliza função pow -> pow(n,y)

Operadores relacionais

Operação	*Tipo dos argumentos*	*Valor*
e_1 == e_2	Números	Tem o valor `True` se e só se os valores das expressões e_1 e e_2 são iguais.
e_1 != e_2	Números	Tem o valor `True` se e só se os valores das expressões e_1 e e_2 são diferentes.
e_1 > e_2	Números	Tem o valor `True` se e só se o valor da expressão e_1 é maior do que o valor da expressão e_2.
e_1 >= e_2	Números	Tem o valor `True` se e só se o valor da expressão e_1 é maior ou igual ao valor da expressão e_2.
e_1 < e_2	Números	Tem o valor `True` se e só se o valor da expressão e_1 é menor do que o valor da expressão e_2.
e_1 <= e_2	Números	Tem o valor `True` se e só se o valor da expressão e_1 é menor ou igual ao valor da expressão e_2.

Operadores e precedências

Prioridade	*Operador*
Máxima	Aplicação de funções
	`not`, – (simétrico)
	*, /, //, %
	+, – (subtração)
	<, >, ==, >=, <=, !=
	`and`
Mínima	`or`



- / -> divisão de a por b
- // -> quociente (parte inteira) da divisão de a por b
- % -> resto da divisão de a por b

Utilização de operadores aritméticos com strings

- Em Python, podemos utilizar os operadores aritméticos com strings
- Concatenação:
  - >>>'Olá' + 'Mundo'
  - 'OláMundo'
- Duplicação:
  - >>>'Olá' * 5
  - 'OláOláOláOláOlá'
  - >>> print('Olá' * 5)
  - OláOláOláOláOlá

Variáveis reservadas

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

Atribuição de valores

- A instrução de atribuição de valores a variáveis tem importância fundamental nas linguagens de programação imperativas, como é o caso do python.
- A instrução recorre ao operador embutido '='. Este operador recebe 2 operandos. O operando do lado esquerdo será o nome da variável e o operando do lado direito o valor a atribuir à variável

  Ex: idade=30

  Nome_completo='Júlio Magalhães'

Atribuição múltipla

- Python, permite fazer várias atribuições em simultâneo (cuidado…)
- <nome$_1$>,<nome$_2$>,…,<nome$_n$>=<eprx$_1$>,<expr$_2$>,…,<expr$_n$>

Ex:

```
>>> nome, apelido='Julio', 'Magalhaes'
>>> print(nome)
Júlio
>>> print(apelido)
Magalhães
>>> print(nome,apelido)
Júlio Magalhães
```

Comentários

- # - tudo que escrever à frente do caracter #, por linha, será ignorado pelo interpretador
- ''' texto ''' – as linhas de texto entre a plica tripla será ignorado pelo interpretador

```
# comentário por linha

'''
sequencia de
comentários
'''
```

Tipos primitivos

- int -> 7 3 0 -3

```
a = int( input('insira 1 número: ‘))
b = int( input('insira outro número’))
s = a + b
print('soma é ', s)
```

- float –> 4.5 0.321 -5.0 3.14

- bool -> True False

- str -> ‘Olá’ ‘7.5’ ‘’ ‘ ‘

Descobrir tipo da variável

- Analise o seguinte código:

```
##descobrir o tipo da variável
n1 = input('Escreva 1 número: ')
print(type(n1))
n2 = int(input('Escreva outro número: '))
print(type(n2))
```

- Veja o tipo das variáveis para o mesmo input '7'

```
Escreva 1 número: 7
<class 'str'>
Escreva outro número: 7
<class 'int'>

Process finished with exit code 0
```

Formato de output em Python3

```
nome = input('Insira o seu nome: ')
print('O nome escrito foi {}.'.format(nome))
```

- O bloco {} será substituído pela formatação e conteúdo da variável nome.
- Abaixo, um exemplo de múltipla atribuição e output com 2 blocos de substituição. O sinal '+' significa que concatena o restante texto que se segue.

```
nome, apelido = input('Nome: '), input('Apelido: ')
print('O nome escrito foi {}'.format(nome) + ' e apelido foi {}'.format(apelido))
```

Continuando com output

- Analise os seguintes códigos:

```
a = int(input('insira 1 número: '))
b = int(input('insira outro número'))
s = a + b
print('soma entre ', a, ' e ', b,' é de ', s)
```

```
a = int(input('insira 1 número: '))
b = int(input('insira outro número'))
s = a + b
print('A soma entre {} e {} é {}'.format(a, b, s))
#print(f'A soma entre {a} e {b} é {s}')
```

Alinhamento no output

- Ocupar 15 posições de caracteres

```
nome = input('Escreva o seu nome: ')
print('O seu nome é {:15}!'.format(nome))
```

```
Escreva o seu nome: Júlio
O seu nome é Júlio          !
```

- Alinhar à direita

```
nome = input('Escreva o seu nome: ')
print('O seu nome é {:>15}!'.format(nome))
```

```
Escreva o seu nome: Júlio
O seu nome é          Júlio!
```

- Alinhar à esquerda

```
nome = input('Escreva o seu nome: ')
print('O seu nome é {:<15}!'.format(nome))
```

```
Escreva o seu nome: Júlio
O seu nome é Júlio          !
```

- Alinhar ao centro

```
nome = input('Escreva o seu nome: ')
print('O seu nome é {:^15}!'.format(nome))
```

```
Escreva o seu nome: Júlio
O seu nome é     Júlio     !
```

- Alinhar ao centro com espaços preenchidos com um símbolo '*'

```
nome = input('Escreva o seu nome: ')
print('O seu nome é {:*^15}!'.format(nome))
```

```
Escreva o seu nome: Júlio
O seu nome é *****Júlio*****!
```

Carateres especiais

Caráter escape	*Significado*
\\	Barra ao contrário (\)
\'	Plica (')
\"	Aspas (")
\b	Retrocesso de um espaço
\f	Salto de página
\n	Salto de linha
\r	"Return"
\t	Tabulação horizontal
\v	Tabulação vertical

Funções isnumeric(), isalpha(), ...

- isnumeric() é numérico?
- isalpha() é alfatético?
- isalnum() -> é alfanumérico? 'Palavra23'
- isspace() -> é espaço? ' '
- isupper() -> letras maiúsculas? 'PALAVRA'
- islower() -> letras minúsculas? 'palavra'
- istitle() -> palavra capitalizada? 'Palavra'

Identações - tab

- O código em python tem de estar muito bem estruturado para que o interpretador o consiga ler, caso contrário, poderá haver erros de semântica
- Conhece a expressão: "99% dos erros informáticos está entre o teclado e a cadeira"? Pois é, no desenvolvimento de código, os erros são do programador, quer por erros de sintaxe quer por erros de semântica.
- As operações que dizem respeito a determinado tipo de bloco têm de estar devidamente identados, usando as tabulações.

Comandos de seleção

- if condição :

  comandos

  else :

  comandos

-> parte do **else** pode ser opcional.

Comandos de seleção - encadeados

- if condição :

  comandos

  elif condição :

  comandos

  elif condição :

  comandos

  ....

  else :

  comandos

Escreva o seguinte código e experimente

```
n = float(input('Escreva uma nota: '))
if n <= 5 :
    print('Mau')
elif n < 10 :
    print('Negativo')
elif n < 18 :
    print('Positivo')
elif n <= 20 :
    print('Muito Bom')
else :
    print('Valor inválido')
```

Comandos de repetição

- while condição :

  comandos

```
n=10
while n>0 :
        print (n)
        n=n - 1
```

- for "variável" in "conjunto de elementos" :

  comandos

```
for n in range(10, 20) :
        print (n)
```

Bibliotecas e bibliotecas parciais

- Funcionalidades já implementadas e que podem ser usadas.
- Comando : import biblioteca (importa todas as funcionalidades implementadas na biblioteca)

Ex: **Biblioteca** math

Funcionalidades: ceil, arredonda para cima,

floor, arredonda para baixo

trunc, trunca o número

pow, potência

sqrt, raiz quadrada

factorial, fatorial de um número

round, arredonda um número a quantas casas decimais pretendemos

Comando: import math

Bibliotecas disponível em https://docs.python.org/3.10/py-modindex.html

Bibliotecas e bibliotecas parciais

Exemplo:

```
import math
n1 = int(input('Insira 1º número: '))
print('O valor da raiz quadrada de {} é {} '.format(n1 , math.sqrt(n1)))
```

- Bibliotecas parciais: **from biblioteca import funcionalidade** (importa apenas a funcionalidade especificada da respetiva biblioteca). Não precisa de escrever o evento biblioteca
- Comando: from math import sqrt, ceil

```
from math import sqrt
n1 = int(input('Insira 1º número: '))
print('O valor da raiz quadrada de {} é {} '.format(n1 , sqrt(n1)))
```

Funções

- Servem para melhor estruturar os programas
- Composta por um nome e possivelmente argumentos: f(x,y)
- Ex:

```
def quadrado (x):
    return x * x
```

O nome da função é quadrado

(x) É o argumento/parâmetro da função. É o valor dado no início da função. O "x" será reconhecido dentro da função.

return x * x é uma expressão que pertence à função. Neste caso, sai da função e responde a quem a chamou com um valor que é o resultado de x*x .

Se não existir return, a função apenas termina não devolvendo qualquer resposta a quem a chamou

Funções

- Programa para calcular o quadrado de x recorrendo a funções

```
##################################################
#Função quadrado: recebe um valor e devolve como resposta o quadrado desse número
def quadrado (x):
    resultado = x * x
    return resultado
##################################################


##Código principal do programa
print("Seja bem-vindo ao cálculo do quadrado de um número.\n")
valor = int( input("Introduza um valor: ") )
resposta = quadrado(valor)
print("O quadrado de {} é {}".format(valor,resposta))
```

Funções recursivas

- Funções que se chamam a elas mesmas com valores intermédios até chegarem ao cálculo mais básico para depois fazerem as sucessivas substituições.
- Ex

```
def fatorial(n):
        if n==0:
                return 1
        else:
                return n * fatorial(n-1)
```

Manipulação de strings

- String significa uma cadeia de caracteres.
- Por exemplo, a frase "Curso de Python" é uma string.
- Python permite de uma forma tranquila, atribuir uma cadeia de caracteres a uma variável:
  - frase="Curso de Python"
- Podemos olhar para uma string como sendo uma variável indexada, ou seja, cada caracter tem um índice que inicia no 0.
- Assim, o índice da letra C é o 0 e o índice da letra y é o 10 (o espaço também é um caracter e ocupa uma posição na string)

String

- frase="Curso de Python"
- Para nos referirmos a um determinado caracter, utilizamos o nome da variável "frase" com o índice pretendido. Ex: frase[9] corresponde ao "P".
- letra = frase[0] #a variável letra assume a letra "C" – índice 0 da frase

String

- frase="Curso de Python"
- No python, podemos obter parte da string de uma forma muito linear. Por exemplo, se quisermos obter a sequência "Pyth" faríamos algo do género: sequencia=frase[9:13]
- A variável sequencia, terá os caracteres da frase que começam no índice 9 e vai até ao 12!!! O índice 13 não é considerado.

String

- frase="Curso de Python"
- E se quiséssemos uma sequência mas com saltos de 2 em 2 caracteres?
- Simples: sequencia=frase[9:14:2] ; o resultado seria Pto

String

- frase="Curso de Python"
- Uma das ações mais utilizadas na análise de string, é saber o comprimento de uma dada frase.
- x = len(frase) #a variável x assume o comprimento da variável frase
- Neste caso, x ficaria com o número 15, uma vez que os espaços também são caracteres.

String

- frase="Curso de Python"
- Uma ação muito utilizada, é saber quantas ocorrências de uma dada letra ocorre na string.
- x = frase.count('o') #a variável x assume a quantidade de ocorrências da letra "o" na frase
- Neste caso, x ficaria com o número 2 pois a letra 'o' aparece 2 vezes na frase

String

- frase="Curso de Python e outro de Python"
- Outra situação bastante recorrente, será procurar uma dada substring. Por exemplo, procurar "yth" na frase!
- frase.find("yth") irá devolver o índice 10 pois é neste índice que inicia a primeira ocorrência da sequência a procurar
- frase.find("Linguagem") irá devolver -1 pois a sequência "Linguagem" não ocorre vez nenhuma

String

- frase="Curso de Python e outro de Python"
- A instrução in é também muito utilizada. Permite-nos obter o resultado booleno (True ou False), caso uma sequência ocorra ou não na frase inicial.
- Ex:
  - "Curso" in frase -> daria resultado True
  - "curso" in frase -> daria resultado False

String

- frase="Curso de Python"
- As strings são imutáveis. Só através de atribuição é que podemos alterá-las.
- O que fará a instrução frase.replace("Python","Linguagem Python") ?
- Neste caso, a palavra a ser substituída é inferior ao que se pretende!
- O Python faz o aumento de índices automático.
- Se fizer um print ao conteúdo da variável, o que irá aparecer?

String

frase="Adoro Linguagem Python e também SQL"

- frase.upper() -> o que for maiúscula mantém, o restante transforma em maiúsculas. E se for um digito?
- frase.lower() – mantém minúscula e altera as maiúsculas para minúsculas
- frase.capitalize() -> todos carateres em minúsculas e só o 1º é capitalizado – maiúscula
- frase.title() -> Capitaliza todas as palavras. Só os 1º carateres são maiúsculas

String

- frase=" Curso de Python "
- frase.strip() -> remove espaços inúteis no inicio e no fim da frase. Reposiciona nos índices corretos do array.
  - E se tiver vários espaços no meio? (Feito de forma diferente…)
- frase.rstrip() – o r significa rigth. Apaga espaços à direita (fim da string)
- frase.lstrip() – o l significa left. Apaga espaços do início.
- **lista** = frase.split() – divisão da string, considerando os espaços. Faz uma lista de arrays… onde a lista é numerada a partir da posição 0
- '-'.join(**lista**) – junta os elementos de cada palavra separados por '-'
- '-'.join(frase) – o que aconteceria?

- Obter o número do carater
  - **ord**('R') -> 82
- Obter o caracter
  - **chr**(125) -> '}'

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]

Tuplos

- Tuplo (tuple) é uma sequência de elementos
- Correspondem à noção matemática de vetor (variável indexada)
- Exemplos de tuplos em python:
  - (), (1,2,3), (2, True), (5,)
- Nota: (5,) é um tuplo com apenas 1 elemento, o nº 5.
- Atribuição de um tuplo a uma variável:
  - notas = (13, 10, 9, 12)

Tuplos

- notas = (13, 10, 9, 12)
- notas[0] # valor do tuplo na posição 0 -> 13
- notas[-2] # 2º valor do tuplo a começar do fim -> 9
- Podemos trabalhar com variáveis na indexação
  - i=1
  - print(notas[i+1]) # escreve o que está no índice 1+1 -> 9
- Erro na indexação:
  - notas[i+10] #não existe índice 11, logo obtemos um erro
    #IndexError: tuple índex out of range

Tuplos

- Um elemento de um tuplo, pode ser também um tuplo:
  - a = ((1,2,3) , 4 , (5,6)) #tuplo "a" tem 3 elementos
- print(a[0]) # devolve o elemento do índice 0 -> (1,2,3)
- Como referir o elemento da posição 1 do elemento índice 0 do tuplo "a"?
  - a[0][1] # nº 2

Tuplos

- Tuplos em Python são imutáveis, ou seja, podemos manipular/operar os elementos mas não podemos redefinir os seus elementos.
- a = ((1,2,3) , 4 , (5,6))
- Se fizermos a[1] = 10, ou seja, em vez do 4 queremos substituir por 10, obtemos um erro. TypeError:'tuple' object does not support item assignment

Manipulação de tuplos

Operação	*Tipo dos argumentos*	*Valor*
$t_1 + t_2$	Tuplos	A concatenação dos tuplos t_1 e t_2.
$t * i$	Tuplo e inteiro	A repetição i vezes do tuplo t.
$t[i_1 : i_2]$	Tuplo e inteiros	O sub-tuplo de t entre os índices i_1 e $i_2 - 1$.
e `in` t	Universal e tuplo	`True` se o elemento e pertence ao tuplo t; `False` em caso contrário.
e `not in` t	Universal e tuplo	A negação do resultado da operação e `in` t.
`tuple(`a`)`	Lista ou dicionário ou cadeia de caracteres	Transforma o seu argumento num tuplo. Se não forem fornecidos argumentos, devolve o tuplo vazio.
`len(`t`)`	Tuplo	O número de elementos do tuplo t.

Listas

- É um novo tipo de estruturar os dados.
- É uma sequência de elementos de qualquer tipo.
- Muito semelhante aos tuplos
- Uma lista é um tipo de variável MUTÁVEL
- A ideia de lista é muito semelhante aos arrays e matrizes de outras linguagens.
- Exemplo de listas:
  - [1, 2, 3], [2, (1,2)], ['a'], [], [1, [2], [[3]]]
- Ao contrário dos tuplos, uma lista com 1 elemento não precisa da ','

Listas

Operação	*Tipo dos argumentos*	*Valor*
l_1 + l_2	Listas	A concatenação das listas l_1 e l_2.
$l * i$	Lista e inteiro	A repetição i vezes da lista l.
$l[i_1:i_2]$	Lista e inteiros	A sublista de l entre os índices i_1 e $i_2 - 1$.
`del(`*els*`)`	Lista e inteiro(s)	Em que *els* pode ser da forma $l[i]$ ou $l[i_1:i_2]$. Remove os elemento(s) especificado(s) da lista l.
e `in` l	Universal e lista	`True` se o elemento *e* pertence à lista l; `False` em caso contrário.
e `not in` l	Universal e lista	A negação do resultado da operação *e* `in` l.
`list(`*a*`)`	Tuplo ou dicionário ou cadeia de caracteres	Transforma o seu argumento numa lista. Se não forem fornecidos argumentos, o seu valor é a lista vazia.
`len(`l`)`	Lista	O número de elementos da lista l.

Listas

- lst1=[1,2,3] e lst2=[[4,5]]
- lst = lst1 + lst2 # resulta na concatenação dos elementos
- len(lst) # qual o tamanho da lista lst?
- lst[3] # qual o elemento do índice 3 da lista lst?
- Para referenciar um elemento de outro que é uma lista, utilizamos uma sintaxe idêntica aos tuplos. Por exemplo, para aceder ao 1º elemento da lista 'lst' que se encontra na 4ª posição (índice 3) utilizamos a referência lst[3][0] #4

Listas

- Mutabilidade:
  - lst=[1,2,3,[4,5]]
  - lst[2]='a' # resulta em [1,2,'a',[4,5]]
  - del(lst[1]) #apaga elemento do índice 1 -> lst=(1,'a',[4,5])
  - del(lst[1:]) # apaga todos os elementos do índice 1 até ao fim

Listas .... como copiar

- A atribuição de listas a listas é uma atribuição simples. Por exemplo, para lst1=[1,2,3,4], e pretendermos uma cópia desta lista basta fazer a atribuição lst2=lst1.
- Ao procedermos a este tipo de atribuição, **atenção**:
  - lst2[1]='a'
    - lst2 fica com os seguintes elementos -> lst2 = [1,'a',3,4]
    - Mas lst1 também fica alterada -> lst1 = [1,'a',3,4]

  A instrução lst2=lst1 é como criar um link para a mesma lista

  (lst1 e lst2 são "alias"/pseudónimos)

Listas

- Para fazer uma cópia do conteúdo de uma lista para outra:
  - l1 = [1,2,3]
  - l2 = l1.copy()
  - l1[0] = "X" # ['X,2,3']
  - print(l2) -> resultado seria [1,2,3]
- O método copy() copia o conteúdo de uma lista para outra e não o endereço de memória para onde aponta a primeira.

Adicionar/remover

- Podemos adicionar um novo valor a uma lista recorrendo ao método "append" ou através da concatenação de listas.
- Desta forma, a inserção no final da lista será do tipo:

  lista = ["Preto","Magenta","Ciano"]

  lista.append("Amarelo") ou lista = lista + ["Amarelo"]
- O resultado final seria a lista com os valores:

  ["Preto", "Magenta", "Ciano", "Amarelo"]
- Para remover uma ocorrência de um valor numa lista poderemos utilizar o método "remove":

  lista.remove("Preto")

Proposta

- Ficha de exercícios nº 1

Ficheiros

- Um conjunto de dados conexo, por exemplo, um documento de texto, uma música, uma imagem, …
- Identificado por um caminho(absoluto/relativo)
- Ao contrário de valores de variáveis, os ficheiros são persistentes
- Para o sistema operativo um ficheiro é sempre uma sequência de bytes
- Suportes físicos: discos ou fitas magnéticas, memórias flash, CD ...

Ficheiros

- Abertura de ficheiro -> função open("caminho", "modo")
- Fecho de ficheiro -> função ficheiro.close()
- Modos de abertura de um ficheiro:

r – Modo de leitura

w – Modo de escrita

a – Modo de acrescentar

r+ - Modo de leitura e escrita sem truncamento

w+ - Modo de leitura e escrita com truncamento

Quando o modo é "w+", significa que elimina o ficheiro e todo o seu conteúdo e cria novamente com o mesmo nome e com o conteúdo que iremos lá colocar. Com "r+" irá sobrepor o novo conteúdo

https://pt.stackoverflow.com/questions/372717/qual-a-diferen%C3%A7a-entre-os-modos-r-e-w-em-python

Ficheiros – modos de abertura

- Abertura do ficheiro "C:\TurmaA\alunos.txt"

```
ficheiro = open("C:\TurmaA\alunos.txt", "r")
```

- Se o ficheiro alunos.txt estiver na mesma localização do executável, então podemos dar o nome do ficheiro:

```
ficheiro = open("alunos.txt", "r")
```

- Podemos na abertura de um ficheiro, forçar o tipo de codificação:

```
ficheiro = open("alunos.txt", "r", encoding="UTF-8")
```

Ficheiros – opções de leitura

- Modos de leitura:

read()	Lê o ficheiro de uma só vez
read(N)	Lê N Bytes
readline()	Lê a próxima linha do ficheiro (incluindo o \n)
readlines()	Lê e guarda como consequência de linhas (cada linha com \n)

Leitura de ficheiros

- Ficheiro texto.txt

Linguagem Python é muito versátil.

Trabalhar com ficheiros em python é bastante intuitivo...

Os modos de trabalhar com ficheiros são: leitura, escrita, acrescentar, leitura e escrita sem truncamento e com truncamento

- Script:

```
fich = open("python.txt","r", encoding="UTF-8") ##fich contém informação sobre a ligação ao ficheiro físico
##Ex: <_io.TextIOWrapper name='python.txt' mode='r' encoding='cp1252'>
```

```
##Leitura de uma só vez
conteudo = fich.read()
fich.close()
print (conteudo)
```

```
##Cria lista em que cada elemento é uma frase
cont = fich.readlines()
print(cont)
```

```
##Le linha a linha ate fim  do ficheiro
## contabiliza nº de linhas que leu
nl=0
for linha in fich:
    print("linha:",linha)
    nl +=1
print("O ficheiro tem", nl,"linhas")
```

Ficheiros – modos de escrita

- Modos de escrita:

write("string") escreve a partir da posição do indicador de escrita. Esta função retorna nº de carateres escritos no ficheiro.

writelines("Sequência") escreve 1 tuplo ou lista cujos elementos são linhas de texto. Escreve a partir da posição do indicador de escrita cada um dos elementos. Não representa qualquer indicação do final de um elemento para o inicio do outro. Esta função não retorna qualquer valor.

Ficheiros – iterando com as linhas

- Modos de iteração (movendo o cursor de leitura):

  seek() fich.seek(pos) – Movimenta para posição 'pos'

  tell() fich.tell() – Indica a posição relativamente ao início

Recorrer ao Sistema Operativo

```
import os
fich = "ola.tx"
os.remove(fich)     #remove o ficheiro
listagem = os.listdir("c:\Python\Exercícios\")  # obtém listagem da diretoria
```

Nota: https://docs.python.org/pt-br/3/library/os.html#module-os

Bibliotecas personalizadas

- Podemos desenvolver pequenas bibliotecas, à nossa medida, e utilizá-las noutros scripts que possamos desenvolver
- Precisam de ser criadas com a extensão ".py"
- Quando precisarmos carregá-las, apenas importamos o nome do ficheiro (sem a extensão).
- Criar exemplo…. Uma biblioteca (quad) com a função quadrado definida em que dado um valor, devolve o seu quadrado. Utilize esta biblioteca no seu script.

Dicionário

- Dicionário é designado por dict (dictionary)
- É mutável e constituído por um conjunto de pares. O 1º elemento de cada par dá-se o nome de "chave" e o 2º elemento o nome de "valor" (chave***1***:valor***1***, chave***2***:valor***2***, …, chave***n***:valor***n***)
- Também apelidamos o dicionário de Lista Associativa.
- Os seus elementos de cada par pode ter tipos de valores distintos.

  Ex: dici = {'a':1, 'b':2, 'c':(3,'a',1)}
- Dicionário não pode ter chaves repetidas.

  dici2 = {'a':1, 'b':2, 'a':(3,'a',1)}. Neste caso, o valor a ser considerado é o último que identa a chave, ou seja, o valor tuplo (3,'a',1)

Dicionário

- Os elementos de um dicionário, ao contrário dos tuplos e listas, não são referenciados pelos seus índices mas sim pelas suas chaves.

```
>>> dici2 = {'a':1, 'b':2, 'a':(3,'a',1)}
>>> dici2['a']
(3, 'a', 1)
```

Dicionário

- Exemplos:
  - dici = {'a':1, 'b':2, 'c’:3}
    - >>>dici[‘a’]
    - 1
    - >>> dici['x']=77 #acrescentar elementos…
    - >>> dici
    - {'a': 1, 'b': 2, 'c': (3, 'a', 1), 'x': 77}

Dicionário

- Quando a definição de dicionário tem chaves repetidas…

```
>>> dici2 = {'a' : 1, 'b' : 2, 'a' : (3,'a',1)}
>>> dici2
{'a' : (3, 'a', 1), 'b' : 2}
```

- Elimina todos os pares cuja chave é repetida e assume apenas a última.

Dicionário

- Quando referencia uma chave que não existe, obtém um erro:

```
>>> dici = {'a':1, 'b':2, 'c':(3,'a',1)}
>>> dici['b']
2
>>> dici['x']
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    dici['x']
KeyError: 'x'
```

Dicionário

- Operações com elementos de um dicionário são possíveis:

```
>>> dici = {'a':1, 'b':2, 'c':(3,'a',1)}
>>> dici['a’] = dici['a'] + 5
>>> dici
{'a': 6, 'b': 2, 'c': (3, 'a', 1)}
```

- Verificar se uma chave existe num dicionário?

```
>>> 'z' in dici
False
```

Operações sobre Dicionários

Operação	*Tipos de argumentos*	*Valor*
del(d[c])	Elemento de dicionário	Remove do dicionário d o elemento com a chave c.
c in d	Chave e dicionário	True se a chave c pertence ao dicionário d; False em caso contrário
c not in d	Chave e dicionário	A negação do resultado da operação c in d.
len(d)	Dicionário	O número de elementos do dicionário d.

Concatenar dicionários

- Não podemos usar o operador '+' para concatenar elementos de 2 dicionários, ao contrário de tuplos e listas, por exemplo.
- Utilizamos o método update():
  - dicionario1.update(dicionario2)
- Exemplo:
  - d1={'a': 1, 'b': 'd'}
  - d2={'c':"ola", 'd':"mundo"}
  - d1.update(d2) -> d1 resulta em {'a': 1, 'b': 'd', 'c':"ola", 'd':"mundo"}

Operações em Dicionários

```
>>> poetas = {'Fernando Pessoa':'Autopsicografia', 'Florbela Espanca':'A mulher',
'Eugénio de Andrade':'É urgente o amor'}

>>> for i in poetas:
        print(i, 'escreveu', poetas[i])
Fernando Pessoa escreveu Autopsicografia
Florbela Espanca escreveu A mulher
Eugénio de Andrade escreveu É urgente o amor
```

Dicionários - exemplo

No exemplo que se segue, temos um string 'poema' com um poema.

Cada verso encontra-se antes do caracter '\n' e cada quadra antes de 2 mudanças de linha '\n\n'.

```
>>> poema='O poeta é um fingidor.\nFinge tão completamente\nQue chega a fingir que é dor\nA dor que deveras sente.\n\nE os que leem o que escreve,\nNa dor lida sentem bem,\nNão as duas que ele teve,\nMas só a que eles não têm.\n\nE assim nas calhas de roda\nGira, a entreter a razão,\nEsse comboio de corda\nQue se chama coração.'
```

Dicionários - exemplo

Pretendemos agora criar um dicionário com a quantidade de vezes que ocorre cada um dos caracteres.

```
ocorre = {} ##dicionario vazio
for car in poema:  ##para cada carater no poema...
    if car not in ocorre:
        ocorre[car] = 1 ##inicializa a contagem do carater com uma ocorrência
    else:
        ocorre[car] = ocorre[car] + 1

print (ocorre)
```

{'O': 1, ' ': 52, 'p': 2, 'o': 19, 'e': 39, 't': 10, 'a': 24, 'é': 2, 'u': 10, 'm': 10, 'f': 2, 'i': 9, 'n': 9, 'g': 4, 'd': 11, 'r': 14, '.': 4, '\n': 13, 'F': 1, 'ã': 5, 'c': 8, 'l': 6, 'Q': 2, 'h': 3, 'q': 6, 'A': 1, 'v': 3, 's': 17, 'E': 3, ',': 5, 'N': 2, 'b': 2, 'M': 1, 'ó': 1, 'ê': 1, 'G': 1, 'z': 1, 'ç': 1}

Dicionários - exemplo

- Pretendemos escrever agora os primeiros 40 caracteres do poema…

```
>>> poema[0:40]
'O poeta é um fingidor.\nFinge tão complet’
```

- Quantos carateres contém o poema?

```
>>>num_car = len(poema)
>>>print("O poema tem {} caracteres (incluindo ".format(num_car), end=‘’)
>>>print("mudanças de linha e outras pontuações)")
O poema tem 305 caracteres (incluindo mudanças de linha e outras pontuações)
```

Dicionários - exemplo

- Quantos carateres contém o poema sem espaços, virgulas, mudanças de linha e outros caracteres de pontuação?

- Uma sugestão será criar uma lista de carateres especiais:

```
especiais = ['!','"','(',')','{','}','[',']','?','\','«','»', ',' ,';','-','º','ª','+', \
             '*',' ','\n','<','>']
```

Dicionários - exemplo

- Depois, aplicar o mesmo código mas apenas se o carater não fizer parte dos carateres especiais

```
ocorre = {} ##dicionario vazio
for car in poema:  ##para cada carater no poema...
    if car not in especiais:
        if car not in ocorre:
            ocorre[car] = 1 ##inicializa a contagem do carater com uma ocorrência
        else:
            ocorre[car] = ocorre[car] + 1
```

Dicionários - exemplo

```
poema='O poeta é um fingidor.\nFinge tão completamente\nQue chega …'
especiais = ['!','"','(',')','{','}','[',']','?','\'','«','»', ',',';','-','º','ª','+','*',' ','\n','<','>']

ocorre = {} ##dicionario vazio
for car in poema:  ##para cada carater no poema...
    if car not in especiais:
        if car not in ocorre:
            ocorre[car] = 1 ##inicializa a contagem do carater com uma ocorrência
        else:
            ocorre[car] = ocorre[car] + 1

num_car_com_pont = len(poema) ##qt de carateres que existe na string
num_car_sem_pont = len(ocorre) ##qts letras diferentes existem
##somar as quantidades de caracteres
num_car=0
for car in ocorre:
    num_car = num_car + ocorre[car]

print("O poema tem {} caracteres (incluindo mudanças de linha e outras pontuações ".format(num_car_com_pont), end='')
print("O poema tem {} caracteres distintos(sem pontuações) ".format(num_car_sem_pont))
print("O poema tem {} caracteres (sem pontuações) ".format(num_car))

qtt_quadras = poema.count('\n\n') + 1 ##1ª quadra não começa com '\n\n'
print("O poema tem {} quadras".format(qtt_quadras))
```

Exemplo para determinar a quantidade de carateres no poema, quantidade de carateres alfabéticos e quantas quadras existem.

Dicionários – Funções associadas

```
>>>poetas.items()
dict_items([('Fernando Pessoa', 'Autopsicografia'), ('Florbela Espanca', 'A mulher'), ('Eugénio de Andrade', 'É urgente o amor')])

>>> poetas.keys()
dict_keys(['Fernando Pessoa', 'Florbela Espanca', 'Eugénio de Andrade'])

>>> poetas.values()
dict_values(['Autopsicografia', 'A mulher', 'É urgente o amor'])
```

Dicionários – ações

- Obter um valor do dicionário:
  - x = poetas.get('Eugénio de Andrade') #Resposta: 'É urgente o amor'
  - O dicionário ficaria intacto
- Obter um valor do dicionário e removê-lo
  - x = poetas.pop('Eugénio de Andrade') # Resposta: 'É urgente o amor'
  - O dicionário ficaria: {'Fernando Pessoa':'Autopsicografia',
    'Florbela Espanca':'A mulher'}
- Verificar se um dado valor existe no dicionário sem saber qual a sua chave:
  - "A mulher" in poetas.values() #resposta: True

Obter "key" dado um "value"

```
poetas = {'Fernando Pessoa':'Autopsicografia', 'Florbela Espanca':'A mulher',
'Eugénio de Andrade':'É urgente o amor'}


valor = "'Autopsicografia"


for key, value in poetas.items(): # items() devolve lista onde cada elemento é 1 tuplo
        if valor == value:
                print(key)
```