Curso:
UpSkill – Digital Skills and Jobs

Unidade de Formação:
Programação
Formador:
Paulo Jorge Costa Nunes

Duração (horas):
96 Sala de Aula (SA)
96 Sessão Sincronia (SS)

Variáveis e Tipos de Dados: Strings
Coleções (Listas, Tuplas)
Tipos de Ficheiros texto e binários
Manipulação de Ficheiros CSV

R A S C U N H O

12 de dezembro de 2023

Lista de abreviações e símbolos

ASCII	American Standard Code for Information Interchange	iv
API	Application Program Interface	2
CSV	Comma Separated Values	1
JSON	JavaScript Object Notion	2
SOLID	Principles - Single responsibility - Open–closed - Liskov substitution - Interface segregation - Dependency inversion	2

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Capítulo 1

Plano

	Capítulo	Horas
1	Variáveis e Tipos de Dados	3
2	Estruturas Lógicas e Condicionais	3
3	Estruturas de Decisão e Repetição	6
4	Coleções (Listas, Tuplas, Sets, etc)	15
5	Funções e Expressões Lambdas	6
6	Tratamento de Erros	6
7	Módulos	3
8	Iteradores e Geradores	3
9	Decorators	3
10	Entrada e Saída de Dados (validação, regex, etc)	9
11	11. Orientação a Objetos (Herança e Polimorfismo) Conceitos Principles - Single responsibility - Open–closed - Liskov substitution - Interface segregation - Dependency inversion (SOLID) em contexto de Python	12
12	Manipulação de Ficheiros Comma Separated Values (CSV) e JavaScript Object Notion (JSON)	9
13	Ligação a Bases de Dados	9
14	Ligação a Web Application Program Interfaces (APIs)	9

Tabela 1.1: Plano de aulas

Capítulo 2

Variáveis e Tipos de Dados

2.1 Strings em Python

Cadeias de caracteres (ou sequências de caracteres) em Python são um tipo de dados denominado de strings (ou str). O tipo string possui várias operações úteis associadas a ele. Por exemplo transformar todas as letras em letra maiúsculas[1].

String em Python é uma coleção de caracteres:

1. Aspas simples: 'permitem aspas "duplas"internas'
2. Aspas duplas: "permitem aspas 'simples' internas"
3. Aspas triplas: "'Três aspas simples"', "Três aspas duplas"

Internamente o computador armazena caracteres manipulados como a combinação de 0 e 1. American Standard Code for Information Interchange (ASCII) é um padrão de codificação de caracteres para comunicação eletrónica. Os códigos ASCII representam texto em computadores, equipamentos de telecomunicações e outros dispositivos. Devido às limitações técnicas dos sistemas informáticos na altura em que foi inventado (1961), o ASCII tem apenas 128 pontos de código, dos quais apenas 95 são caracteres imprimíveis, o que limitou severamente o seu âmbito. Os sistemas de computador modernos evoluíram para usar Unicode (https://www.unicode.org/charts), que possui milhões de pontos de código, mas os primeiros 128 deles são iguais ao conjunto ASCII[2].

O padrão ASCII reserva os primeiros 32 pontos de código (números 0 a 31 decimais) e o último (número 127 decimal) para caracteres de controle. Por exemplo:

•

[1] https://algoritmosempython.com.br/cursos/programacao-python/strings
[2] https://en.wikipedia.org/wiki/ASCII

- 10 ou LF ou Ĵ ou \n: Line Feed: Mudança de linha
- 13 ou CR ou M̂ ou \r: Carriage Return: Mudança de linha
- 9 ou HT ou Î \t: Horizontal Tab: Avança na vertical (4 caracteres)
- 127 ou 7F ou DEL ou ^?: Delete: Apaga o caractere junto ao cursor.

Isso significa que as strings podem ser analisadas em caracteres individuais e que os caracteres individuais podem ser manipulados de várias maneiras. A Tabela 2.1 apresenta alguns caracteres da tabelas ASCII.

Binary	Oct	Dec	Hex	Glyph		
				1963	1965	1967
010 0000	040	32	20	space		
010 0001	041	33	21	!		
010 0010	042	34	22	"		
010 0011	043	35	23	#		
010 0100	044	36	24	$		
010 0101	045	37	25	%		
010 0110	046	38	26	&		
010 0111	047	39	27	'		
010 1000	050	40	28	(		
010 1001	051	41	29	)		
010 1010	052	42	2A	*		
010 1011	053	43	2B	+		
100 0001	101	65	41	A		
100 0010	102	66	42	B		
100 0011	103	67	43	C		
100 0100	104	68	44	D		
100 0101	105	69	45	E		
100 0110	106	70	46	F		
100 0111	107	71	47	G		

Binary	Oct	Dec	Hex	Glyph		
				1963	1965	1967
100 1000	110	72	48	H		
100 1001	111	73	49	I		
100 1010	112	74	4A	J		
100 1011	113	75	4B	K		
100 1100	114	76	4C	L		
100 1101	115	77	4D	M		
100 1110	116	78	4E	N		
100 1111	117	79	4F	O		
101 0000	120	80	50	P		
101 0001	121	81	51	Q		
101 0010	122	82	52	R		
101 0011	123	83	53	S		
101 0100	124	84	54	T		
101 0101	125	85	55	U		
101 0110	126	86	56	V		
101 0111	127	87	57	W		
101 1000	130	88	58	X		
101 1001	131	89	59	Y		
101 1010	132	90	5A	Z		

Figura 2.1: Carateres da tabela ASCII. Fonte: https://en.wikipedia.org/wiki/ASCII

O nome ANA em binário é representado pelos bits:

```
ANA
Três bytes (8 bits)
012345670123456701234567
010000010100111001000001
```

Listagem 2.1: ANA em binário.

e em decimal por:

```
ANA
Três numeros (3 digitos)
012012012
065116065
```

Listagem 2.2: ANA em décimal.

```
# Imprimindo uma string.
s = "Olá, mundo!"
print('1)', s)

# Tipo de uma string.
print('2)', type(s))

# É do tipo de uma string?
print('3)', isinstance(s, str))

# Tamanho de uma string.
print('4)', len(s))
```

Listagem 2.3: Código Python.

```
1) Olá, mundo!
2) <class 'str'>
3) True
4) 11
```

Listagem 2.4: Resultado.

```
# Concatenação
print('5)', "Meu Portugal " + "português")

# Substitui uma substring por alguma outra coisa.
s1 = s.replace("mundo", "meu lar")
print('6)', s1)

# A string s começa com "Olá"?
print('7)', s.startswith("Olá"))

# A string s termina com "mundo"?
print('8)', s.endswith("mundo"))

# Quantas ocorrências da palavra "abacate" a string s1 possui?
print(s1.count("lar"))
```

Listagem 2.5: Resultado.

```
5) Meu Portugal português
6) Olá, meu lar!
7) True
8) False
1
```

Listagem 2.6: Resultado.

2.2 Substrings em Python (Slicing)

Além das operações vistas acima, podemos acessar caracteres específicos de uma string em Python usando a notação `[]`. Neste esquema de acesso a caracteres de uma string, o primeiro caractere está no índice 0, o segundo no índice 1, e assim por diante, conforme ilustrado no exemplo abaixo.

O nome completo de uma pessoa é composto pelos nomes próprios e pelos apelidos O nome pode conter no máximo seis vocábulos simples ou compostos, em regra, até dois nomes próprios e quatro apelidos[3]. De seguida são apresentados exemplo para efetuar operações com nomes de pessoas.

```
pos = '01234567890123456789012345678901234567890123456789'
nome = "RODOLFO PINHEIRO QUEIROS ARANTES"

print('0:', nome[0])
print('1:', nome[1])
print('2:', nome[2])
print("Posições.....:", pos)
print("Original.....:", nome)

# Podemos também ter acesso aos elementos em ordem reversa usando índices negativos. Neste
    esquema, o último caracter de uma string está no índice -1, o penúltimo no índice -2, e
    assim por diante, como mostrado no exemplo abaixo.

print('-1', nome[-1])
print('-2', nome[-2])
print('-3', nome[-3])
```

Listagem 2.7: Resultado.

```
0: R
1: O
2: D
Posições.....: 01234567890123456789012345678901234567890123456789
Original.....: RODOLFO PINHEIRO QUEIROS ARANTES
-1 S
-2 E
-3 T
```

Listagem 2.8: Resultado.

Podemos também ter acesso fatias ou "slices"de uma string ou lista em Python. Esta notação é muito concisa e poderosa, então é importante que o programador a entenda. Segundo essa notação, uma fatia de uma string, ou seja, uma substring, pode ser acedida se fornecermos os índices do começo e do final da fatia que desejamos analisar, como mostrado abaixo:

```
nome = "RODOLFO PINHEIRO QUEIROS ARANTES"
print('nome[:7]', nome[:7])
#
print('nome[9:]', nome[9:])
#
# Retorna os caracteres 1 e 2
print('nome[1:3]', nome[1:3])
```

[3] https://irn.justica.gov.pt/Servicos/Cidadao/Nascimento/Composicao-do-nome

```
#
# Retorna toda a string
print('nome[:]', nome[:])
```

Listagem 2.9: Resultado.

Note que, como mencionamos anteriormente, os índices de uma string começam do 0 e não do 1. Além disso, perceba que **o índice do final da fatia não é incluído nela**. No exemplo acima, o [1:3] nos retornou dois caracteres e não três. Foram retornados o caracter no índice 1 e o caracter no índice 2, mas não o caracter no índice 3. Se omitirmos o índice de ínicio da fatia ou o de final (ou ambos), o ínicio e o final da string serão considerados, respectivamente. Veja os exemplos:

```
nome[:7] ROD
nome[9:] FO PINHEIRO QUEIROS ARANTES
nome[1:3] OD
nome[:] RODOLFO PINHEIRO QUEIROS ARANTES
```

Listagem 2.10: Resultado.

É possível ainda especificar um parâmetro que indica quantos caracteres devem ser processados de cada vez. Por exemplo, se quisermos imprimir somente os caracteres nos índices pares ou ímpares de uma string, podemos fazer assim:

```
nome = "RODOLFO PINHEIRO QUEIROS ARANTES"
print('nome[::2]', nome[::2])  # Imprime os caracteres nos índices pares
print('nome[1::2]', nome[1::2])  # Imprime os caracteres nos índices ímpares
```

Listagem 2.11: Resultado.

```
nome[::2] RDLOPNER UIO RNE
nome[1::2] OOF IHIOQERSAATS
```

Listagem 2.12: Resultado.

Um outro exemplo útil do uso da técnica de slicing para manipulação de strings é **inverter uma palavra** ou frase usando somente operações de slicing:

```
nome = "RODOLFO PINHEIRO QUEIROS ARANTES"
print('nome[::-1]', nome[::-1])
```

Listagem 2.13: Resultado.

```
nome[::-1] SETNARA SORIEUQ ORIEHNIP OFLODOR
```

Listagem 2.14: Resultado.

No exemplo acima, usamos um terceiro parâmetro do recurso de slicing para indicar que retornamos toda a frase (os ::) e logo em seguida dizemos que faremos isso de trás para frente (por meio do -1 no final). Mais especificamente, o -1 indica que estamos saltando um caractere de cada vez, começando de trás para frente (o que é feito por meio do sinal de menos).

Então, para resumir, a sintaxe de slicing de strings é a seguinte `[início:fim:salto]`, onde:

- **início**: é o primeiro índice a ser considerado (o primeiro caracter da string é considerado caso este valor seja omitido);
- **fim - 1**: é o último índice a ser considerado (o último caracter da string é considerado caso este valor seja omitido); e
- **salto**: indica quantos caracteres devem ser saltados em cada etapa (o valor 1 é considerado por padrão, e um sinal de menos deve ser usado para percorrer a string em ordem reversa).

2.3 Alterar Maiúsculas e Minúsculas

```
nome = "RODOLFO PINHEIRO QUEIROS ARANTES"
print('# Nomes-Strings ' + ('-' * 20))
print("Posições.....:", pos)
print(".capitalize():", nome.capitalize())
print(".title().....:", nome.title())
print(".upper().....:", nome.upper())
print(".lower().....:", nome.lower())
#
import string
print(".capwords()..:", string.capwords(nome))
```

Listagem 2.15: Resultado.

```
# Nomes-Strings --------------------
Posições.....: 012345678901234567890123456789012345678 9
.capitalize(): Rodolfo pinheiro queiros arantes
.title().....: Rodolfo Pinheiro Queiros Arantes
.upper().....: RODOLFO PINHEIRO QUEIROS ARANTES
.lower().....: rodolfo pinheiro queiros arantes
.capwords()..: Rodolfo Pinheiro Queiros Arantes
```

Listagem 2.16: Resultado.

2.4 Partir Strings em Diversos Pedaços

a Função `str.split(sep=None, maxsplit=-1)`, retorna uma lista de palavras na string, usando sep como a string delimitadora.

- Se maxsplit é fornecido, no máximo maxsplit cortes são feitos (portando, a lista terá no máximo maxsplit+1 elementos).
- Se maxsplit não foi especificado ou -1 foi informado, então não existe limite no número de cortes (todos os cortes possíveis são realizados).

```
nome = "RODOLFO PINHEIRO QUEIROS ARANTES"
nome_lista = nome.split(' ')
print(nome_lista)
```

```
primeironome = nome_lista[0]  # primeiro
ultimonome = nome_lista[-1]  # último, -2 penúltimo
print('primeironome..:', primeironome)  #
print('ultimonome....:', ultimonome)

print("primeiro nome (nome próprio) e restantes nomes: nome.split(' ', 1)")
nome_lista = nome.split(' ', 1)
print('a)', nome_lista)

print("nomes + apelido: nome.rsplit(' ', 1)")
nome_lista = nome.rsplit(' ', 1)
print('b)', nome_lista)

print("Dois nomes próprios + apelidos: nome.split(' ', 2)")
nome_lista = nome.split(' ', 2)
print('c)', nome_lista)
```

Listagem 2.17: Resultado.

```
['RODOLFO', 'PINHEIRO', 'QUEIROS', 'ARANTES']
primeironome..: RODOLFO
ultimonome....: ARANTES
primeiro nome (nome próprio) e restantes nomes: nome.split(' ', 1)
a) ['RODOLFO', 'PINHEIRO QUEIROS ARANTES']
nomes + apelido: nome.rsplit(' ', 1)
b) ['RODOLFO PINHEIRO QUEIROS', 'ARANTES']
Dois nomes próprios + apelidos: nome.split(' ', 2)
c) ['RODOLFO', 'PINHEIRO', 'QUEIROS ARANTES']
```

Listagem 2.18: Resultado.

2.5 String Começa por uma Substring

A função `str.startswith(prefix[, start[, end]])`, retorne True se a String começar com o prefixo, caso contrário, retorna False.

- O prefixo também pode ser uma tupla (ex: `('Ana', 'Carla')`) de prefixos a serem procurados.
- Com start opcional, a String de teste começa nessa posição.
- Com fim opcional, interrompe a comparação de String nessa posição.

```
#     0123456789012345678901234567890123456789
s1 = 'RODOLFO PINHEIRO QUEIROS ARANTES'
s2 = 'BRUNO AMADO ARANTES QUEIROS'
print("Pesquisa de texto em strings")
print("Começa por:")

print('1)', s1.startswith('RODOLFO'))
print('2)', s1.startswith('RODOLFO', 9))
print('3)', s1.startswith('PINHEIRO', 9))
print('4)', s2.startswith('ARANTES', 11, 18))
print('5)', s1.startswith(('RODOLFO', 'BRUNO')))
```

```
print('6)', s2.startswith(('RODOLFO', 'BRUNO')))
```

Listagem 2.19: Resultado.

```
01234567890123456789012345678901234567890123456789
Pesquisa de texto em strings
Começa por:
1) True
2) False
3) False
4) False
5) True
6) True
```

Listagem 2.20: Resultado.

2.6 String Termina por uma Substring

```
#     01234567890123456789012345678901234567890123456789
s1 = 'RODOLFO PINHEIRO QUEIROS ARANTES'
s2 = 'BRUNO AMADO ARANTES QUEIROS'
print("Pesquisa de texto em strings")
print("Termina por:")

print('1)', s1.endswith('ARANTES'))
print('2)', s1.endswith('RODOLFO', 9))
print('3)', s1.endswith('PINHEIRO', 9))
print('4)', s2.endswith('QUEIROS', 11, 18))
print('5)', s1.endswith(('ARANTES', 'QUEIROS')))
print('6)', s2.endswith(('ARANTES', 'QUEIROS')))
```

Listagem 2.21: Resultado.

```
01234567890123456789012345678901234567890123456789
Pesquisa de texto em strings
Termina por:
1) True
2) False
3) False
4) False
5) True
6) True
```

Listagem 2.22: Resultado.

2.7 A String Contém uma Substring

A função `str.find(sub[, start[, end]])`, retorna o índice mais baixo na string onde a substring sub é encontrado dentro da fatia s[start:end]. Argumentos opcionais como start e end são interpretados como na notação de fatiamento. Retorna -1 se sub não for localizado.

Nota O método find() deve ser usado apenas se precisarmos conhecer a posição de sub. Para verificar se sub é ou não uma substring, use o operador in: `'Py' in 'Python'`. Resultado: `True`.

```
#    01234567890123456789012345678901234567890123456789
m = 'Pedro José Tomé Monteiro'
print(f"1) {m.find('Tomé')}")
print(f"2) {'Tomé' in m}")
print(f"3) {m.find('Maria')}")
print(f"1) {m.find('Tomé', 15)}")
print(f"1) {m.find('Tomé', 11)}")
print(f"1) {m.find('José', 5, 12)}")
```

Listagem 2.23: Resultado.

```
1) 11
2) True
3) -1
1) -1
1) 11
1) 6
```

Listagem 2.24: Resultado.

2.8 Subtituir Texto Em Strings

```
h = 'Henrique Nogueira Pereira'
m1 = 'Lígia Isabel Mendes Belo'
m2 = 'Ana Augusto Soares Ferreira'

print('h + m1: adcionar apelido de f a m1 (não esquercer o espaço)')
m1h = m1 + ' ' + h.rsplit(' ', 1)[-1]
print(f'i) .{m1h}.')
print(f'\t.{m1}.')
print(f'\t.{h}.')

print('h + m2: subtituir apelido de f por apelido de m2')
m2h = m2.replace(m2.rsplit(' ', 1)[-1], h.rsplit(' ', 1)[-1])
print(f'ii) .{m2h}.')
print(f'\t.{m2}.')
print(f'\t.{h}.')
```

Listagem 2.25: Resultado.

```
h + m1: adcionar apelido de f a m1 (não esquercer o espaço)
i) .Lígia Isabel Mendes Belo Pereira.
  .Lígia Isabel Mendes Belo.
  .Henrique Nogueira Pereira.
h + m2: subtituir apelido de f por apelido de m2
ii) .Ana Augusto Soares Pereira.
  .Ana Augusto Soares Ferreira.
  .Henrique Nogueira Pereira.
```

Listagem 2.26: Resultado.

2.9 Remover no Início

Função `str.removepreffix(suffix, /)`. Se a string terminar com a string suffix e a suffix não estiver vazia, retorna string[:-len(suffix)]. Caso contrário, retorna uma cópia da string original:

```
m = 'João Filipe da Silva Matos'
m1 = m.removeprefix(m.split(' ', 1)[0] + ' ')
print(f'i) .{m1}.')
print(f'\t.{m}.')
```

Listagem 2.27: Resultado.

```
i) .Filipe da Silva Matos.
  .João Filipe da Silva Matos.
```

Listagem 2.28: Resultado.

2.10 Remover no Meio

```
m = 'João Filipe da Silva Matos'
m1 = m.replace(' da', '')
print(f'i) .{m1}.')
print(f'\t.{m}.')
```

Listagem 2.29: Resultado.

```
i) .João Filipe Silva Matos.
  .João Filipe da Silva Matos.
```

Listagem 2.30: Resultado.

2.11 Remover no Fim

Função `str.removesuffix(suffix, /)`. Se a string terminar com a string suffix e a suffix não estiver vazia, retorna string[:-len(suffix)]. Caso contrário, retorna uma cópia da string original:

```
m = 'Patrick Alexandre Pereira Batista'
m1 = m.removesuffix(' ' + m.rsplit(' ', 1)[-1])
print(f'i) .{m1}.')
print(f'\t.{m}.')
```

Listagem 2.31: Resultado.

```
i) .Patrick Alexandre Pereira.
  .Patrick Alexandre Pereira Batista.
```

Listagem 2.32: Resultado.

2.12 Juntar Elementos de Strings e Vetores

Função string `'<sepadador>'.join(<string | lista>)`. Retorna a string que é a concatenação das strings na lista (iterável).

Um erro to tipo `TypeError` será levantada se existirem quaisquer valores que não sejam strings no iterável. O separador entre elementos é a string que está fornecendo este método.

```
str = '-'.join('hello')
print(str)
list1 = ['U', 'P', 's', 'k', 'i', 'l', 'l']
print('i)', ''.join(list1))

nome = 'Miguel Paulo de Assunção Silva'
nome_lista = nome.split(' ')
nome_de_lista = ' '.join(nome_lista)
print('ii) Juntar elementos de um vetor:', nome_de_lista)
print('\t', nome)
print('\t', nome_lista)
```

Listagem 2.33: Resultado.

```
h-e-l-l-o
i) UPskill
ii) Juntar elementos de um vetor: Miguel Paulo de Assunção Silva
   Miguel Paulo de Assunção Silva
   ['Miguel', 'Paulo', 'de', 'Assunção', 'Silva']
```

Listagem 2.34: Resultado.

2.13 Gerador de texto

https://www.messletters.com/pt/characters/

Capítulo 3

Coleções (Listas, Tuplas, Sets, Dictionaries)

3.1 Listas

Em Python uma sequência de elementos, que podem ou não ser do mesmo tipo (números, strings) é denominada de lista (vetor). Em Python, listas de objetos são representadas pelo tipo *list*. As listas são um dos principais tipos de dados em Python que permitem simplificar os programas em Python[1].

É possível modificar elementos em uma lista. Dizemos que as listas são tipos de dados **mutáveis**. Este conceito ficará mais interessante quando estudarmos tuplas, que são tipos de dados https://algoritmosempython.com.br/cursos/programacao-python/listas que não se podem modificar.

3.1.1 Criar Listas

Sintaxe para criar uma lista: `lista = [elementos separados por vírgula]`

```
frutas = ['Banana', "Maça", "Morango", "Laranjas"]
print(frutas)
print(type(frutas))
# Resultado
['Banana', 'Maça', 'Morango', 'Laranjas']
<class 'list'>
```

Listagem 3.1: Criar lista de frutas.

[1] https://algoritmosempython.com.br/cursos/programacao-python/listas

3.1.2 Criar Tuplas

Tupla é um tipo de estrutura de dados utilizada em Python que funciona de modo semelhante a uma lista, entretanto, com a característica principal de ser **imutável**. Isso significa que quando uma tupla é criada não é possível adicionar, alterar ou remover seus elementos. Geralmente, ela é utilizada para adicionar tipos diferentes de informações, porém, com a quantidade de elementos definidos[2]. Também é muito útil quando são passadas para funções desenvolvidas por outros programadores garantindo que não são alteradas internamente.

Sintaxe para criar uma tupla: `tupla = (elementos separados por vírgula)`. A sintaxe é semelhante à criação de listas utilizando os parênteses curvos '`()`' como delimitadores em vez do parênteses retos '`[]`'.

```
frutas = ('Banana', "Maça", "Morango", "Laranjas")
print(type(frutas))
print(frutas)
# Resultado
# <class 'tuple'>
# ['Banana', 'Maça', 'Morango', 'Laranjas', 'Pêra']
```

Listagem 3.2: Criar tupla de frutas.

3.1.3 Adicionar Elementos em Listas

Função append(): `lista.append(elemento`
Função extend(): `lista.append(lista2`

```
frutas = ['Banana', "Maça", "Morango", "Laranjas"]
frutas.append('Pêra')
print(frutas)
# Resultado
# ['Banana', 'Maça', 'Morango', 'Laranjas', 'Pêra']
# adicionar lista
frutas.extend(['Babaco', 'Pitanga'])
print(frutas)
# ['Banana', 'Maça', 'Morango', 'Laranjas', 'Pêra', 'Babaco', 'Pitanga']
```

Listagem 3.3: Adicionar elementos a uma lista.

3.1.4 Alterar Elementos de Listas

```
frutas = ['Banana', "Maça", "Morango", "Laranjas"]
frutas [3] = "Laranja"
print(frutas)
# Resultado
['Banana', 'Maça', 'Morango', 'Laranja']
```

[2] https://blog.betrybe.com/tecnologia/tuplas-em-python/#1

Listagem 3.4: Alterar elementos em listas.

3.1.5 Eliminar Elementos de Listas

```
# por índice
frutas = ['Banana', 'Maça', 'Morango', 'Laranjas', 'Pêra', 'Babaco', 'Pitanga']
frutas.pop(4)
print(frutas)

# por nome
frutas = ['Banana', 'Maça', 'Morango', 'Laranjas', 'Pêra', 'Babaco', 'Pitanga']
frutas.remove('Pêra')
print(frutas)
# Resultado
```

Listagem 3.5: Eliminar elementos em listas.

3.2 Ordenação de Listas

Função: `sorted(lista, key=None, reverse=False)`. Retorna uma nova lista ordenada dos elementos. Possui dois argumentos opcionais que devem ser especificados como argumentos nomeados.

Função: `lista.sort(key=None, reverse=False)`. Ordena fisicamente a lista. Ou seja não é criada uma nova lista. Esta função é importante quando estamos na presença de listas cuja dimensão ultrapassa a memória do computador.

O parâmetro `key` especifica a função de um argumento usado para extrair uma chave de comparação de cada elemento (por exemplo, `key=str.lower`). O valor padrão é `None` (compara os elementos diretamente).

```
# Ordenação. Criando uma nova lista
frutas = ['Banana', 'Maça', 'Morango', 'Laranjas', 'Pêra', 'Babaco', 'Pitanga']
frutas_ordenadas = sorted(frutas)
print(frutas_ordenadas)
# Resultado
# ['Babaco', 'Banana', 'Laranjas', 'Maça', 'Morango', 'Pitanga', 'Pêra']

# Ordenação da lista (on site)
frutas = ['Banana', 'Maça', 'Morango', 'Laranjas', 'Pêra', 'Babaco', 'Pitanga']
frutas.sort()
print(frutas)

# Ordenação da lista (on site) por ordem descendente
frutas = ['Banana', 'Maça', 'Morango', 'Laranjas', 'Pêra', 'Babaco', 'Pitanga']
frutas.sort(reverse=True)
print(frutas)
# Resultado
# ['Pêra', 'Pitanga', 'Morango', 'Maça', 'Laranjas', 'Banana', 'Babaco']
```

Listagem 3.6: Ordenação de listas.

```
# Ordenação da lista em função do tamanho do nome das frutas
frutas = ['Banana', 'Maça', 'Morango', 'Laranjas', 'Pêra', 'Babaco', 'Pitanga']
frutas.sort(key=lambda e: len(e))
print(frutas)
# Resultado
# ['Maça', 'Pêra', 'Banana', 'Babaco', 'Morango', 'Pitanga', 'Laranjas']

# Ordenação da lista em função do tamanho do nome das frutas + nome
frutas = ['Banana', 'Maça', 'Pitanga', 'Morango', 'Laranjas', 'Pêra', 'Babaco']
frutas.sort(key=lambda e: str(len(e)) + e)
print(frutas)
# Resultado
# ['Maça', 'Pêra', 'Babaco', 'Banana', 'Morango', 'Pitanga', 'Laranjas']
```

Listagem 3.7: Ordenação de listas com função de comparação diferente dos elementos.

3.3 Juntar Listas Paralelas

Função `zip(listas separadas por vírgula, strict=False)`. Itera sobre vários listas (iteráveis) em paralelo, produzindo tuplas com um item de cada um. Se as listas tiverem quantidades de elementos diferentes são apenas juntos os até ao mínimo de elementos das listas. O parâmetro `strict=True` obriga que as listas tenham a mesma dimensão. Se não tiverem é levantada uma exceção (erro).

```
# Juntar listas
codigos = [1, 2, 3, 4, 5, 6, 7]
frutas = ['Banana', 'Maça', 'Morango', 'Laranjas', 'Pêra', 'Babaco', 'Pitanga']
lista_zip = zip(codigos, frutas)
print("lista_zip: não é iterável", lista_zip)
lista = list(lista_zip)
print("Lista", lista)

# Juntar listas
codigos = [1, 2, 3, 4, 5]
frutas = ['Banana', 'Maça', 'Morango', 'Laranjas', 'Pêra', 'Babaco', 'Pitanga']
lista_zip = zip(codigos, frutas)
print("lista_zip: não é iterável", lista_zip)
lista = list(lista_zip)
print("Lista", lista)

print("zip, range")
for item in zip(range(1, len(frutas) + 1), frutas, strict=True):
    print(item)
```

Listagem 3.8: Exemplo juntar listas paralelas.

Capítulo 4

Manipulação de Ficheiros CSV e JSON

4.1 Introdução

Definição de ficheiro[1]**:**

1. caixa, gaveta ou pasta onde se guardam fichas
2. conjunto de fichas
3. catálogo
4. **INFORMÁTICA:** conjunto de informações, programas, etc., armazenado com um determinado nome na memória de um computador ou num suporte de informação

Ficheiro informático ou ficheiro eletrónico[2]**:**

Ficheiro guardado em qualquer tipo de memória eletrónica permanente e que está disponível para ser usado por programas informáticos.

Os ficheiros informáticos podem ser considerados a versão contemporânea dos documentos em papel guardados em ficheiros de escritório ou de biblioteca.

4.2 Tipos de Ficheiros Informáticos

Existem basicamente dos tipos de ficheiros[3]:

1. Ficheiros de texto;

[1] https://www.infopedia.pt/dicionarios/lingua-portuguesa-aao/ficheiro
[2] https://apdsi.pt/glossario/f/ficheiro-informatico
[3] https://blog.pantuza.com/artigos/diferencas-entre-arquivos-texto-e-binario

2. Ficheiros binários.

4.2.1 Ficheiros do Tipo Texto

Nos ficheiros de texto os dados são representados linha à linha. Na prática é uma sequência de bytes (1 byte são 8 bits. Um bit pode ter o valor 1 ou 0) representando caracteres. As linhas são representadas pelo caracter de quebra de linha, por exemplo: '\n'. Esse caracter vai variar de sistema operativo para sistema operativo. Todos os dados são armazenados como caracteres. Utilizando a tabela ASCII, cada caractere necessita de um byte em memória para ser armazenado. Por exemplo, na Listagem 4.1 o número '100' utilizaria um bytes para cada dígito (3) do número.

```
leite meio-gordo 800 ml
açúcar 100 g
casca de limão 1 unid.
pau de canela 2 unid.
pão cacete 8 fatia
ovo M3 unid.
óleo para fritar 500 ml
açúcar (para polvilhar) 2 c. de sopa
canela (para polvilhar) 1 q.b.
```

Listagem 4.1: Exemplo de conteúdo de ficheiro de texto: Receita para rabanadas.

Aplicações com arquivos texto:

- Formatos de dados: csv, yaml, txt
- Linguagens de marcação: HTML, Markdown
- Formatos de mensagens: JSON, XML, SVG
- Formatos de código de programas: py, c, cc, bas, bas, php, aspx, cs

Uma das grandes vantagens dos ficheiros de texto é ser apenas necessário um editor de texto para os criar, visualizar e alterar. Um editor de texto para efetuar as operações básicas de criar e editar ficheiros de texto é muito simples. No entanto, existem editores de texto mais sofisticados que facilitam a escrita do conteúdo dos ficheiros. Algumas das funções mais comuns são corretor ortográfico, auto-completar palavras, etc. Os editores para escrever programas (Ex: PyCharm, Visual Code) facilitam a escrita de programas porque o utilizador ao digitar letras, são apresentadas todas as palavras começadas por essas letras. O utilizador pode escolher um delas. É comum também utilizar cores para as diferentes instruções das linguagens de programação.

4.2.2 Ficheiros do Tipo Binário

Os ficheiros binários são representados por uma sequência de bytes sem o conceito de quebra de linha. Ele armazena o dado literal, ou seja, não são caracteres. Pode imaginar uma fita sequencial cheia de dados. O número 100 ocuparia apenas 1 byte cujo valor em binário seria `01100100`.

Aplicações com arquivos binários:

- Codificadores de vídeo: Xvid, x264, MPEG
- Codificadores de áudio: FLAC, LAME
- Compressores de arquivos: gzip, 7z, xz
- Codificadores de imagem: PNG, JPEG, GIF, WebP

Para visualizar o conteúdo de ficheiro binários é necessário um programa especifico capaz de interpretar/transformar/descodificar o seu conteúdo.

4.3 Extensões no Nome dos Ficheiros

Para facilitar o trabalho dos utilizadores de computadores e outros dispositivos eletrónicos foram definidas extensões nos nomes dos ficheiros de acordo com o tipo de conteúdo de armazena. Assim, com um simples clique no ficheiro, o computador escolhe o programa adequado para abrir e visualizar o conteúdo do ficheiros. Por exemplo, um clique num ficheiro com o nome `carta.docx`, abre o abre o programa Microsoft Word (ou semelhante) para visualizar o ficheiro.

- Ficheiros do Microsoft Office: .doc, .docx, .xls, .xlsx, .ppt (só de leitura), .pptx (só de leitura). Saiba mais sobre como ver e editar documentos do Office.
- Multimédia: .3gp, .avi, .mov, .mp4, .m4v, .m4a, .mp3, .mkv, .ogv, .ogm, .ogg, .oga, .webm, .wav.
- Imagens: .bmp, .gif, .jpg, .jpeg, .png, .webp. Ficheiros comprimidos: .zip, .rar.
- Texto: .txt, .html, .xml, .json
- Outros: .pdf.

```
fic_txt = 'dados.txt'
fic_bin = 'dados.bin'

f = open(fic_txt, 'wt', encoding='ascii')
for x in range(0, 1023+1):
    print(x, file=f)
f.close()

f = open(fic_bin, 'wb')
for x in range(0, 1023+1):
    f.write(x.to_bytes(2))
    # file.write((i).to_bytes(24, byteorder='big', signed=False))
f.close()

import os
from datetime import datetime

print(os.path.getsize(fic_txt))
print(os.path.getmtime(fic_txt))
print(os.path.getctime(fic_txt))

# os.path.getsize() returns the size of the file
# os.path.getmtime() returns the file last modified date
# os.path.getctime() returns the file creation date (equals to last modified date in Unix
    systems like macOS)

def PropriedadesFicheirio(fic):
    print(fic)
    # timestamp is number of seconds since 1970-01-01
    # 1702312270.2769284
    # convert the timestamp to a datetime object in the local timezone
    data_criacao = datetime.fromtimestamp(os.path.getmtime(fic))
    # print the datetime object and its type
    print("Data criação =", data_criacao)
    print("Tipo =", type(data_criacao))
    data_alteracao = datetime.fromtimestamp(os.path.getmtime(fic))
    print("Data alteração = ", data_alteracao)

PropriedadesFicheirio(fic_txt)
PropriedadesFicheirio(fic_bin)

# Open the binary file
file = open(fic_bin, "rb")
# Reading the first three bytes from the binary file
bytes = file.read(2)
# Printing data by iterating with while loop
while bytes:
    inteiro = int.from_bytes(bytes)
    print(inteiro)
    bytes = file.read(2)
# Close the binary file
file.close()

#os.system(fic_txt)
```

```
0
1
2
3
4
5
6
7
```

```
8
9
10
...
1023
```

```

```

```
....
```

Figura 4.1

Figura 4.2

Figura 4.3

4.4 Ficheiros TXT

4.4.1 Abrir Ficheiros de Texto para Leitura

```
open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True,
    opener=None)
```

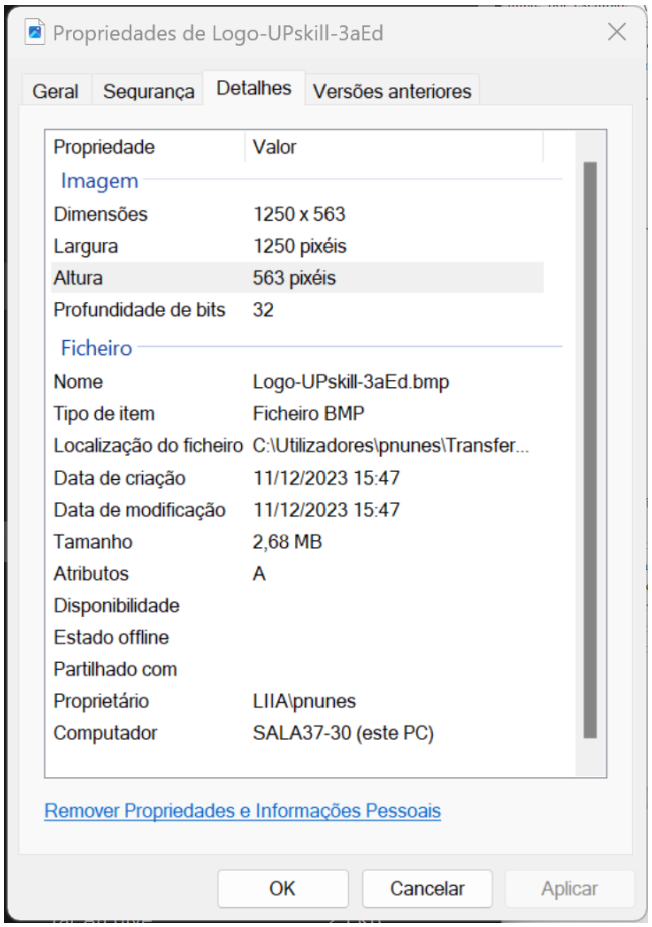
Propriedades de Logo-UPskill-3aEd
Geral
Segurança
Detalhes
Versões anteriores
Propriedade
Valor
Imagem
Dimensões
1250 x 563
Largura
1250 pixéis
Altura
563 pixéis
Profundidade de bits
32
Ficheiro
Nome
Logo-UPskill-3aEd.bmp
Tipo de item
Ficheiro BMP
Localização do ficheiro
C:\Utilizadores\pnunes\Transfer...
Data de criação
11/12/2023 15:47
Data de modificação
11/12/2023 15:47
Tamanho
2,68 MB
Atributos
A
Disponibilidade
Estado offline
Partilhado com
Proprietário
LIIA\pnunes
Computador
SALA37-30 (este PC)
Remover Propriedades e Informações Pessoais
OK
Cancelar
Aplicar

Figura 4.4

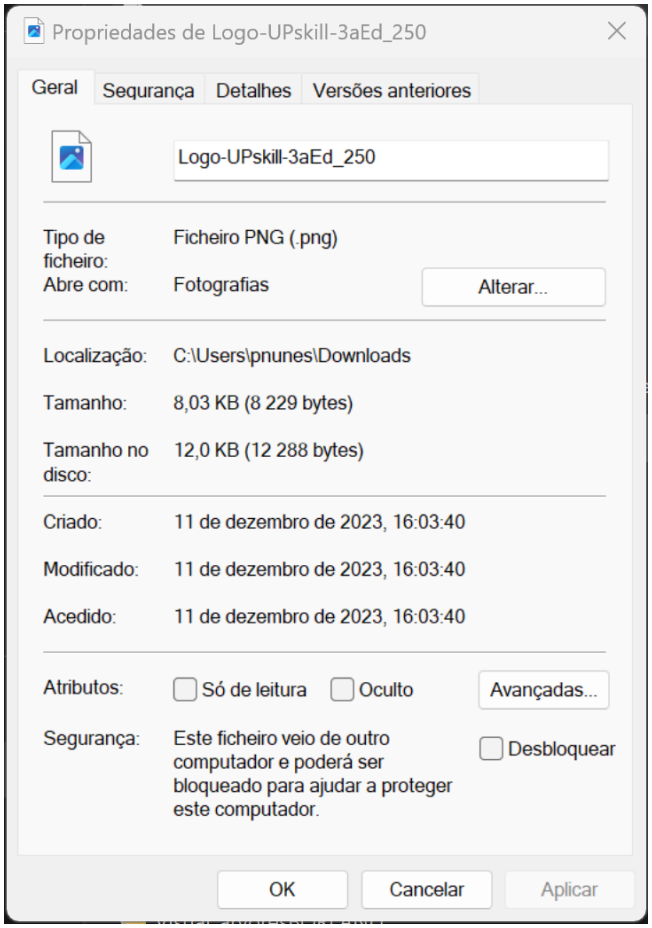
Propriedades de Logo-UPskill-3aEd_250
Geral
Segurança
Detalhes
Versões anteriores
Logo-UPskill-3aEd_250
Tipo de ficheiro:
Ficheiro PNG (.png)
Abre com:
Fotografias
Alterar...
Localização:
C:\Users\pnunes\Downloads
Tamanho:
8,03 KB (8 229 bytes)
Tamanho no disco:
12,0 KB (12 288 bytes)
Criado:
11 de dezembro de 2023, 16:03:40
Modificado:
11 de dezembro de 2023, 16:03:40
Acedido:
11 de dezembro de 2023, 16:03:40
Atributos:
Só de leitura
Oculto
Avançadas...
Segurança:
Este ficheiro veio de outro computador e poderá ser bloqueado para ajudar a proteger este computador.
Desbloquear
OK
Cancelar
Aplicar

Figura 4.5

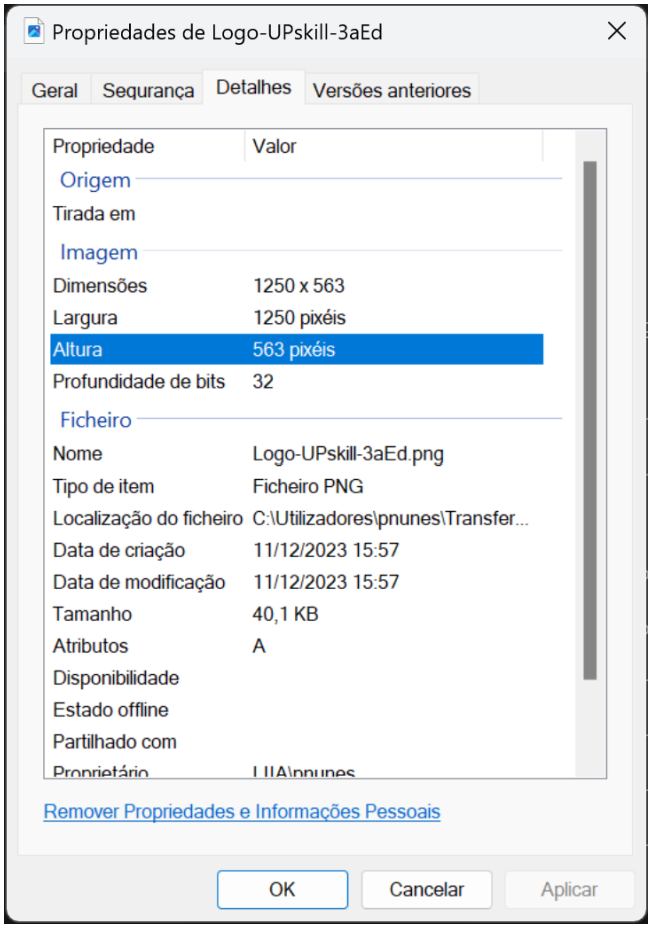
Propriedades de Logo-UPskill-3aEd
Geral
Segurança
Detalhes
Versões anteriores
Propriedade
Valor
Origem
Tirada em
Imagem
Dimensões
1250 x 563
Largura
1250 pixéis
Altura
563 pixéis
Profundidade de bits
32
Ficheiro
Nome
Logo-UPskill-3aEd.png
Tipo de item
Ficheiro PNG
Localização do ficheiro
C:\Utilizadores\pnunes\Transfer...
Data de criação
11/12/2023 15:57
Data de modificação
11/12/2023 15:57
Tamanho
40,1 KB
Atributos
A
Disponibilidade
Estado offline
Partilhado com
Proprietário
LIA\pnunes
Remover Propriedades e Informações Pessoais
OK
Cancelar
Aplicar

Figura 4.6

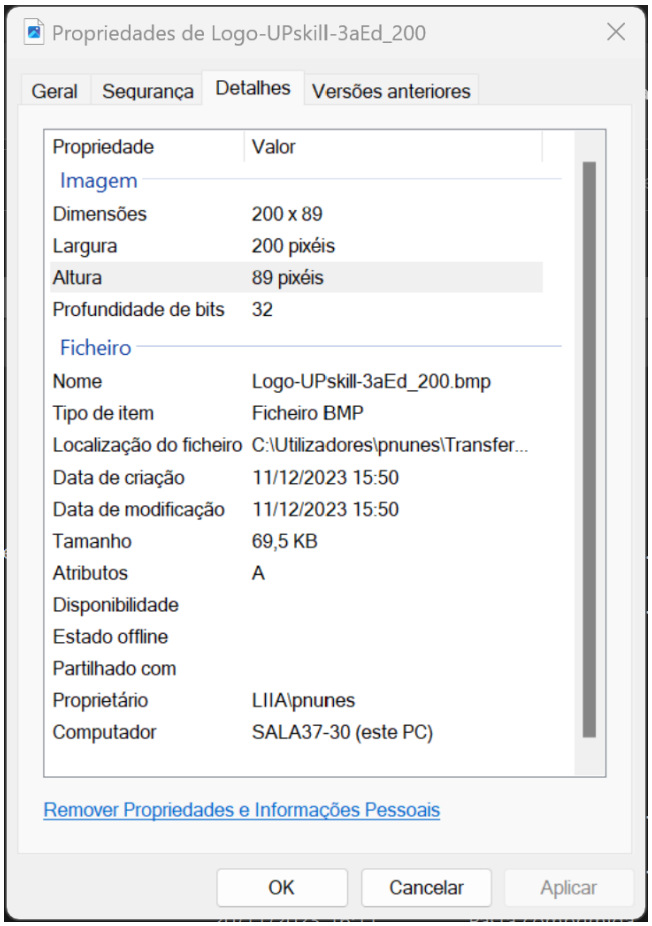

Figura 4.7

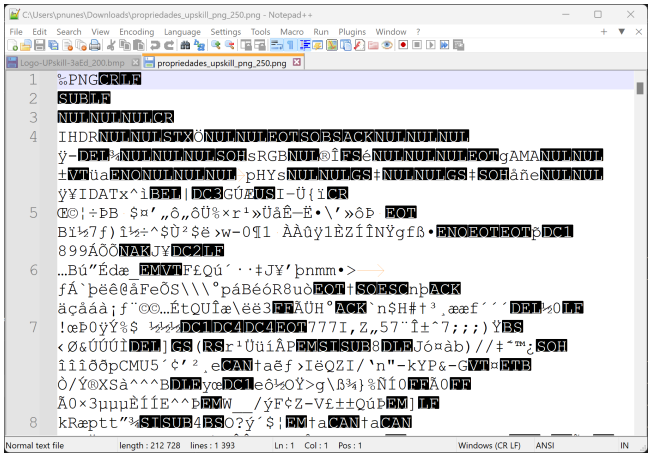

Figura 4.8

Abre file e retorna um objeto arquivo correspondente. Se o arquivo não puder ser aberto, uma `OSError` é levantada.

Caractere	Significado
'r'	abre para leitura (padrão)
'w'	abre para escrita, truncando o arquivo primeiro (removendo tudo o que estiver contido no mesmo)
'x'	abre para criação exclusiva, falhando caso o arquivo exista
'a'	abre para escrita, anexando ao final do arquivo caso o mesmo exista
'+'	aberto para atualização (leitura e escrita)
'b'	binary mode
't'	modo texto (padrão)

Tabela 4.1

4.5 Manipulação de Ficheiros CSV

Um ficheiro CSV (ficheiro de valores separados por vírgula) é um tipo de ficheiro de texto simples que usa estruturação específica para organizar dados tabulares. Por ser um ficheiro de texto simples, ele pode conter apenas dados de texto reais – em outras palavras, caracteres ASCII ou Unicode imprimíveis[4].

O chamado formato CSV (Valores Separados por Vírgula) é o formato de importação e exportação mais comum para folhas de dados e bases de dados. Os dados são armazenados em ficheiros de texto na forma tabular. Cada linha contém dados acerca da mesma entidade (pessoa, animal, objeto, conceito) separada por **vírgula** (`,`). Cada coluna contém dados do mesmo tipo. Por exemplo nomes ou preços. O ficheiro pode conter na primeira linha o significado das colunas.

A Listagem 4.3 ilustra um exemplo de ficheiro CSV, que tem na primeira linha os nomes das colunas e nas restantes linhas dados acerca de pessoas.

```
1 IDUTENTE;NOME;NUMERODISPOSITIVOELETRONICO
2 1002115;LEONOR CLEMENTINO CUNHA LEAL LOBATO;10698-595
3 1004899;CAMILO CARMONA AMORIM FERREIRA EANES;10147-132
4 1006661;RITA FERRO CLEMENTINO REGO ARANTES CARVALHEIRA ROSADO;19954-561
5 ...
```

Listagem 4.2: Exemplo de conteúdo de ficheiro CSV com cabeçalho e separador o símbolo `;`.

De facto, o separador mais comum é o ponto e vírgula pelo facto de ser mais raro ter dados contendo o caractere (`;`) do que contendo o caractere (`,`). O formato de texto CSV tem a limitação de os dados não poderem conter o caractere delimitador. Isso faria alterar o número de colunas dos dados. No exemplo abaixo as linhas 2 e 4 têm quatro colunas em vez de três.

[4] https://realpython.com/python-csv

Por consequência, os dados não seriam lidos de forma correta. Por exemplo, o código postal para a linha 2 seria **50** em vez de **6300-559 Guarda**.

```
1 Nome;Morada;Codigo Postal
2 Instituto Politécnico da Guarda,Avenida Dr. Francisco Sá Carneiro, 50,6300-559 Guarda
3 Município da Guarda,Praça do Município,6301-854 GUARDA
4 Finanças de Guarda,Av. Monsenhor Mendes do Carmo, 13 r/c,6300-586 Guarda
```

Listagem 4.3: Exemplo de conteúdo de ficheiro CSV com o símbolo delimitador nos dados.

Para contornar o problema é comum colocar os dados entre aspas simples (') ou duplas ("), contornando o problema. A Listagem 4.4 é um exemplo com aspas duplas (").

```
1 "Nome","Morada","Codigo Postal"
2 "Instituto Politécnico da Guarda","Avenida Dr. Francisco Sá Carneiro, 50","6300-559 Guarda"
3 "Município da Guarda,Praça do Município","6301-854 GUARDA"
4 "Finanças de Guarda","Av. Monsenhor Mendes do Carmo, 13 r/c","6300-586 Guarda"
```

Listagem 4.4: Exemplo de conteúdo de ficheiro CSV com o símbolo delimitador nos dados e aspas duplas.

No entanto, não resolve os casos em que os dados contenham aspas. Para resolver o problema é comum usar como delimitador caracteres que não estejam contidos nos dados. Por exemplo, o caracetre |.

```
1 |Nome|,|Morada|,|Codigo Postal|
2 |Instituto Politécnico da Guarda|,|Avenida Dr. Francisco Sá Carneiro, 50|,|6300-559 Guarda|
3 |Município da Guarda,Praça do Município|,|6301-854 GUARDA|
4 |Finanças de Guarda|,|Av. Monsenhor Mendes do Carmo, 13 r/c|,|6300-586 Guarda|
```

Listagem 4.5: Exemplo de conteúdo de ficheiro CSV com o símbolo delimitador |.

Para resolver os problemas dos delimitadores e aspas é comum usar ficheiros de texto com comprimento fixo para as colunas. Isto obriga a saber o tamanho máximo para cada um dos dados. Por outro lado é necessário mais espaço em disco para armazenar os dados. Por exemplo: 'Município da Guarda' tem 20 caracteres com corrimento fixo são necessários 35 (20 + 15 espaços). A Listagem 4.6 ilustra um exemplo de ficheiro de texto de comprimento fixo. O seu tamanho no disco são 370 bytes enquanto o ficheiro da Listagem 4.5. tem 270 bytes em disco, que corresponde a mais 37%.

```
1 12345678901234567890123456789012345123456789012345678901234567890123456789012345678901234567890123456789012345678
2 Nome                              Morada                                  Codigo Postal
3 Instituto Politécnico da Guarda   Avenida Dr. Francisco Sá Carneiro, 50   6300-559 Guarda
4 Município da Guarda               Praça do Município                      6301-854 GUARDA
5 Finanças de Guarda                Av. Monsenhor Mendes do Carmo, 13 r/c   6300-586 Guarda
```

Listagem 4.6: Exemplo de conteúdo de ficheiro CSV com o símbolo delimitador |.

4.5.1 Leitura e Escrita de Ficheiros CSV

```
1;Ketchup;Mercearia Salgado;1.59;23%
2;Atum;Mercearia Salgado;3.38;6%
```

```
3;Cogumelos;Mercearia Salgado;1.98;23%
4;Bolachas Cacau;Mercearia Doce;1.39;23%
5;Cerelac;Mercearia Doce;5.65;6%
6;Heineken 12*25CL;Bebidas;8.99;23%
7;Gel Dove;Higiene e Beleza;5.99;23%
8;Lenços de Bolso;Limpeza do Lar;1.59;23%
9;Jardineira;Congelados;1.99;6%
10;Leite 6x1L;Lacticinios;4.92;6%
11;Natas;Lacticinios;0.79;6%
12;Ovos;Lacticinios;0.94;6%
13;Espadarte Posta;Peixaria;6.49;6%
14;Queijo Limiano;Charcutaria;3.89;6%
15;Laranjas;Frutas e Legumes;1.449;6%
16;Ameixa Seca;Frutas e Legumes;2.00;6%
17;Tabuleiro;Casa;7.00;23%
18;Pensar Python;Cultura;17.0;6%
19;Limpa Vidros;Bricolage;3.69;23%
20;Oregaos;Mercearia Salgado;1.42;6%
21;Bolachas Belvita;Mercearia Doce;2.39;23%
22;Pedras Salgadas;Bebidas;2.46;13%
23;Skip 40D;Limpeza do Lar;12.99;23%
24;Pão Aveia;Padaria;1.99;6%
25;Iogurte Grego;Lacticinios;2.59;6%
```

Listagem 4.7: Exemplo de ficheiro (hipermercado.csv) de com produtos de um hipermercado.

```
#Leitura de ficheiro de texto sem conjsidrar a sua estrutura/formato

f = open('hipermercado.txt', 'rt', encoding='utf-8')
linhas = f.readlines()  # lista em que cada elemento contém uma linha do ficheiro
f.close()
print(linhas)
for p in linhas:
    print(p)
```

Listagem 4.8: Leitura do ficheiro `hipermercado.csv`.

Cada linha do ficheiro é um elemento to tipo string da lista. A string inclui o caractere de controlo (\n) de fim de linha (primeira linha). Ao imprimir cada uma das linhas resulta uma linha em branco entre as linhas porque o caractere (\n) está a ser escrito no ecrã mudando de linha.

```
['1;Ketchup;Mercearia Salgado;1.59;23%\n', '2;Atum;Mercearia Salgado;3.38;6%\n', '3; ....]
1;Ketchup;Mercearia Salgado;1.59;23%

2;Atum;Mercearia Salgado;3.38;6%

3;Cogumelos;Mercearia Salgado;1.98;23%

4;Bolachas Cacau;Mercearia Doce;1.39;23%

5;Cerelac;Mercearia Doce;5.65;6%

6;Heineken 12*25CL;Bebidas;8.99;23%
...
```

Listagem 4.9: Leitura do ficheiro `hipermercado.csv`.

4.6 Biblioteca csv to Python

A biblioteca `csv` fornece funcionalidade para ler e gravar ficheiros CSV. Projetado para funcionar imediatamente com ficheiros CSV gerados pelo Excel, é facilmente adaptado para funcionar com uma variedade de formatos CSV. A biblioteca csv contém objetos e outros códigos para ler, gravar e processar dados de e para arquivos CSV[5].

```
import csv
lista = []
with open('hipermercado.txt', 'rt', newline='',  encoding='utf-8') as f:
    reader = csv.reader(f, delimiter=';')
    for r in reader:
        lista.append(r)
for p in lista:
    print(p)
```

Listagem 4.10: Leitura do ficheiro `hipermercado.csv`.

Cada linha da lista é uma lista com os dados de um produto.

```
['1', 'Ketchup', 'Mercearia Salgado', '1.59', '23%']
['2', 'Atum', 'Mercearia Salgado', '3.38', '6%']
['3', 'Cogumelos', 'Mercearia Salgado', '1.98', '23%']
['4', 'Bolachas Cacau', 'Mercearia Doce', '1.39', '23%']
['5', 'Cerelac', 'Mercearia Doce', '5.65', '6%']
['6', 'Heineken 12*25CL', 'Bebidas', '8.99', '23%']
['7', 'Gel Dove', 'Higiene e Beleza', '5.99', '23%']
...
```

Listagem 4.11: Leitura do ficheiro `hipermercado.csv`.

4.6.1 Escrita de Ficheiros CSV

```
import csv

def EscreverFicheiroCSV(nome_ficheiro, cabecalhos, lista, delimitador=';', aspas="'",
    tipo_aspas=csv.QUOTE_MINIMAL):
    # csv.writer(csvfile, dialect='excel', **fmtparams)
    with open(nome_ficheiro, 'wt', newline='', encoding='utf-8') as f:
        writer = csv.writer(f, delimiter=delimitador, quotechar=aspas, quoting=tipo_aspas)
        writer.writerow(cabecalhos)
        writer.writerows(lista)
        # for r in lista:
        #     writer.writerow(r)


lista = [
    ('Instituto Politécnico da Guarda', 'Avenida Dr. Francisco Sá Carneiro, 50',
     '6300-559 Guarda'),
    ('Município da Guarda,Praça do Município', '6301-854 GUARDA'),
    ('Finanças de Guarda', 'Av. Monsenhor Mendes do Carmo, 13 r/c', '6300-586 Guarda')]

EscreverFicheiroCSV('moradas_aspas_esc.csv',
                    ['Nome', 'Morada', 'Codigo Postal'], lista)
```

[5] https://realpython.com/python-csv

Listagem 4.12: Função para escrever dados CSV.

```
Nome;Morada;Codigo Postal
Instituto Politécnico da Guarda;Avenida Dr. Francisco Sá Carneiro, 50;6300-559 Guarda
Município da Guarda,Praça do Município;6301-854 GUARDA
Finanças de Guarda;Av. Monsenhor Mendes do Carmo, 13 r/c;6300-586 Guarda
```

Listagem 4.13: Conteúdo ficheiro.

4.6.2 Leitura de Ficheiros CSV

```
import csv
# Leitura de ficheiro \ac{CSV}
def LerFicheiroCSV(nome_ficheiro, delimitador):
    import csv
    lista = []
    with open(nome_ficheiro, 'rt', newline='',  encoding='utf-8') as f:
        reader = csv.reader(f, delimiter=delimitador)
        cab = reader.__next__()
        for r in reader:
            lista.append(r)
    return cab, lista

cab, lista = LerFicheiroCSV("moradas_aspas.csv", delimitador=',')
print(cab)
for m in lista:
    print(m)
```

Listagem 4.14: Função para ler dados CSV.

```
['Nome', 'Morada', 'Codigo Postal']
['Instituto Politécnico da Guarda', 'Avenida Dr. Francisco Sá Carneiro, 50', '6300-559 Guarda
    ']
['Município da Guarda,Praça do Município', '6301-854 GUARDA']
['Finanças de Guarda', 'Av. Monsenhor Mendes do Carmo, 13 r/c', '6300-586 Guarda']
```

Listagem 4.15: Resultado.

Parece evidente que o algoritmo de utilizar caracteres como separador e aspas (que pode ser qualquer caractere) para delimitar os dados das colunas não é suficiente. Porque pode sempre existir nos dados um caracertre

4.7 Vinhos

TODO

4.8 Incêndios

https://raw.githubusercontent.com/centraldedados/incendios/master/data/incendios2015.csv

4.9 Produtos IVA Zero

https://www.publico.pt/interactivos/evolucao-cabaz-iva-zero-ficou-mais-caro-mais-barato

4.10 Acidentes

pordata.pt

Bibliografia