

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
CR	Carriage Return	3
CRLF	Carriage Return Line Feed	4
CSV	Comma Separated Values	1
HT	Horizontal Tab	4
JSON	JavaScript Object Notion	2
LF	Line Feed	4
PNG	Portable Network Graphics	iv
SOLID	Single responsibility - Open-closed - Liskov substitution - Interface segregation - Dependency inversion	2
SVG	Scalable Vector Graphics	iv

Conteúdo

1	Plano	2
2	Variáveis e Tipos de Dados	3
2.1	Strings em Python	3
2.2	Substrings em Python (Slicing)	6
2.3	Alterar Maiúsculas e Minúsculas	8
2.4	Partir Strings em Diversos Pedacos	8
2.5	String Começa por uma Substring	9
2.6	String Termina por uma Substring	10
2.7	A String Contém uma Substring	11
2.8	Substituir Texto Em Strings	11
2.9	Remover no Início	12
2.10	Remover no Meio	12
2.11	Remover no Fim	12
2.12	Juntar Elementos de Strings e Vetores	13
2.13	Gerador de texto	13
3	Coleções (Listas, Tuplas, Sets, Dictionaries)	14
3.1	Listas	14
3.1.1	Criar Listas	14
3.1.2	Criar Tuplas	15
3.1.3	Adicionar Elementos em Listas	15
3.1.4	Alterar Elementos de Listas	15
3.1.5	Eliminar Elementos de Listas	16
3.2	Ordenação de Listas	16
3.3	Juntar Listas Paralelas	17
4	Manipulação de Ficheiros CSV e JSON	18
4.1	Introdução	18
4.2	Tipos de Ficheiros Informáticos	18

4.2.1	Ficheiros do Tipo Texto	19
4.2.2	Ficheiros do Tipo Binário	20
4.3	Extensões nos Nomes dos Ficheiros	20
4.4	Conceito de bit, byte e Representação de Números	21
4.5	Unidade de Medida de Memória e Armazenamento	22
4.6	Exemplo de Escrita de Dados em Ficheiro de Texto e Binário	24
4.7	Logotipo do UPskill em Diferentes Formatos de Ficheiro Binário de Imagens	27
4.8	Manipulação de Ficheiros CSV	28
4.8.1	Leitura e Escrita de Ficheiros CSV	29
4.9	Biblioteca csv to Python	31
4.9.1	Escrita de Ficheiros CSV	31
4.9.2	Leitura de Ficheiros CSV	32
4.10	Vinhos	32
4.11	Incêndios	33
4.12	Produtos IVA Zero	33
4.13	Acidentes	33

Lista de Figuras

2.1	Carateres da tabela American Standard Code for Information Interchange (ASCII). Fonte: https://theasciicode.com.ar/ascii-control-characters/delete-ascii-code-127.html##google_vignette	4
4.1	Conteúdo do ficheiro <code>dados.txt</code> visualizado com o editor de texto NotePad++.	26
4.2	Conteúdo do ficheiro <code>dados.bin</code> visualizado com o editor de texto NotePad++.	27
4.3	Logo UPskill no formato Portable Network Graphics (PNG). Resolução (1250, 563), tamanho: 41,078 bytes.	27
4.4	Logo UPskill no formato PNG. Zoom: 0.05.	28
4.5	Logo UPskill no formato Scalable Vector Graphics (SVG). A resolução pode ser infinita. tamanho: 56,554 bytes	28
4.6	Logo UPskill no formato SVG. Zoom: 0.05.	28

Lista de Tabelas

1.1	Plano de aulas	2
4.1	Byte composto por 8 bits (valores: 0,1). Contém o valor 65, letra A.	21
4.2	Exemplos de números escritos em decimal, binário e octal.	21
4.3	Exemplos de números escritos em decimal, binário e octal (cont).	22
4.4	Número de bits e bytes para representar números decimais em binário.	23
4.5	Unidades de medida: do kilobyte ao yottabyte.	24

Lista de Listagens

2.1	ANA em binário.	5
2.2	ANA em decimal.	5
2.3	Código Python.	5
2.4	Resultado.	5
2.5	Resultado.	5
2.6	Resultado.	5
2.7	Resultado.	6
2.8	Resultado.	6
2.9	Resultado.	7
2.10	Resultado.	7
2.11	Resultado.	7
2.12	Resultado.	7
2.13	Resultado.	7
2.14	Resultado.	7
2.15	Resultado.	8
2.16	Resultado.	8
2.17	Resultado.	9
2.18	Resultado.	9
2.19	Resultado.	9
2.20	Resultado.	10
2.21	Resultado.	10
2.22	Resultado.	10
2.23	Resultado.	11
2.24	Resultado.	11
2.25	Resultado.	11
2.26	Resultado.	11
2.27	Resultado.	12
2.28	Resultado.	12
2.29	Resultado.	12

2.30	Resultado.	12
2.31	Resultado.	13
2.32	Resultado.	13
2.33	Resultado.	13
2.34	Resultado.	13
3.1	Criar lista de frutas.	14
3.2	Criar tupla de frutas.	15
3.3	Adicionar elementos a uma lista.	15
3.4	Alterar elementos em listas.	15
3.5	Eliminar elementos em listas.	16
3.6	Ordenação de listas.	16
3.7	Ordenação de listas com função de comparação diferente dos elementos.	17
3.8	Exemplo juntar listas paralelas.	17
4.1	Exemplo de conteúdo de ficheiro de texto: Receita para rabanadas.	19
4.2	Script para gerar as tabelas 4.2 e 4.3	21
4.3	Script para gerar a tabela ??	22
4.4	Script para gerar a tabela ??	23
4.5	Atributos dos ficheiros <code>dados.txt</code> e <code>dados.bin</code>	25
4.6	Exemplo de conteúdo de ficheiro Comma Separated Values (CSV) com cabeçalho e separador o símbolo ;.	28
4.7	Exemplo de conteúdo de ficheiro CSV com o símbolo delimitador nos dados.	29
4.8	Exemplo de conteúdo de ficheiro CSV com o símbolo delimitador nos dados e aspas duplas.	29
4.9	Exemplo de conteúdo de ficheiro CSV com o símbolo delimitador 	29
4.10	Exemplo de conteúdo de ficheiro CSV com o símbolo delimitador 	29
4.11	Exemplo de ficheiro (<code>hipermercado.csv</code>) de com produtos de um hipermercado.	29
4.12	Leitura do ficheiro <code>hipermercado.csv</code>	30
4.13	Leitura do ficheiro <code>hipermercado.csv</code>	30
4.14	Leitura do ficheiro <code>hipermercado.csv</code>	31
4.15	Leitura do ficheiro <code>hipermercado.csv</code>	31
4.16	Função para escrever dados CSV.	31
4.17	Conteúdo ficheiro.	32
4.18	Função para ler dados CSV.	32
4.19	Resultado.	32

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 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 maiúsculas¹.

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](https://en.wikipedia.org/wiki/ASCII)) é um padrão de codificação de caracteres para comunicação eletrônica. Os códigos [ASCII](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/ASCII)².

O padrão [ASCII](https://en.wikipedia.org/wiki/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://algoritmoempython.com.br/cursos/programacao-python/strings>

²<https://en.wikipedia.org/wiki/ASCII>

- Carriage Return (**CR**) (Retorno de carro) - (\hat{M} , $\backslash r$, 0x0D em hexadecimal, 13 em decimal) - move o cursor para o início da linha sem avançar para a próxima linha.
- Line Feed (**LF**) (Avanço de linha) - (\hat{J} , $\backslash n$, 0x0A em hexadecimal, 10 em decimal) - move o cursor para baixo até a próxima linha sem retornar ao início da linha.
- Carriage Return Line Feed (**CRLF**) - Um **CR** imediatamente seguido por um **LF** (**CRLF**, $\backslash r \backslash n$ ou 0x0D0A) move o cursor para a próxima linha e depois para o início da linha.
- Horizontal Tab (**HT**) (9, \hat{I} , $\backslash t$) - Avança por omissão 4 caracteres na vertical (o número de caracteres de avanço pode ser mudado nas opções dos editores de texto. É comum usar avanço de 2 caracteres).

Os caracteres de controle **CR** e **LF** são caracteres de controle **ASCII** e Unicode, enquanto \hat{M} e \hat{J} são abstrações usadas em certas linguagens de programação.

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**.

ASCII control characters	ASCII printable characters	Extended ASCII characters	ASCII 127
00 NULL (Null character)	32 space	128 Ç	
01 SOH (Start of Header)	33 !	129 ü	alt + 127 (Delete)
02 STX (Start of Text)	34 "	130 é	
03 ETX (End of Text)	35 #	131 â	
04 EOT (End of Trans.)	36 \$	132 ä	
05 ENQ (Enquiry)	37 %	133 à	
06 ACK (Acknowledgement)	38 &	134 á	
07 BEL (Bell)	39 '	135 ç	
08 BS (Backspace)	40 (136 ê	
09 HT (Horizontal Tab)	41)	137 ë	
10 LF (Line feed)	42 *	138 è	
11 VT (Vertical Tab)	43 +	139 ì	
12 FF (Form feed)	44 ,	140 í	
13 CR (Carriage return)	45 -	141 i	
14 SO (Shift Out)	46 .	142 Ä	
15 SI (Shift In)	47 /	143 Å	
16 DLE (Data link escape)	48 0	144 É	
17 DC1 (Device control 1)	49 1	145 æ	
18 DC2 (Device control 2)	50 2	146 Æ	
19 DC3 (Device control 3)	51 3	147 ð	
20 DC4 (Device control 4)	52 4	148 ò	
21 NAK (Negative acknowl.)	53 5	149 ó	
22 SYN (Synchronous idle)	54 6	150 û	
23 ETB (End of trans. block)	55 7	151 ü	
24 CAN (Cancel)	56 8	152 ý	
25 EM (End of medium)	57 9	153 Ò	
26 SUB (Substitute)	58 :	154 Ù	
27 ESC (Escape)	59 ;	155 ø	
28 FSC (File separator)	60 <	156 £	
29 GS (Group separator)	61 =	157 Ø	
30 RS (Record separator)	62 >	158 x	
31 US (Unit separator)	63 ?	159 f	
127 DEL (Delete)			

most consulted
ñ énye, n with tilde (alt + 164)
■ black square (alt + 254)
² superscript two, square (alt + 253)
° degree symbol (alt + 248)
' apostrophe, single quote (alt + 39)
μ letter Mu, micro, micron (alt + 230)
© copyright symbol (alt + 169)
® registered trademark (alt + 169)
³ superscript three, cube (alt + 252)
á a with acute accent (alt + 160)

frequently-used (spanish language)	vowels acute accent (spanish language)	vowels with diaries	mathematical symbols	commercial / trade symbols	quotes and parenthesis
ñ alt + 164	á alt + 160	ä alt + 132	½ alt + 171	\$ alt + 36	" alt + 34
Ñ alt + 165	é alt + 130	ë alt + 137	¼ alt + 172	£ alt + 156	' alt + 39
@ alt + 64	í alt + 161	ï alt + 139	¾ alt + 243	¥ alt + 190	(alt + 40
¿ alt + 168	ó alt + 162	ö alt + 148	¹ alt + 251	¢ alt + 189) alt + 41
? alt + 63	ú alt + 163	ü alt + 129	² alt + 252	¤ alt + 207	[alt + 91
¡ alt + 173	Á alt + 181	Ä alt + 142	³ alt + 253	© alt + 169] alt + 93
! alt + 33	É alt + 144	È alt + 211	f alt + 159	© alt + 184	{ alt + 123
: alt + 58	Í alt + 214	Ï alt + 216	± alt + 241	ª alt + 166	} alt + 125
/ alt + 47	Ó alt + 224	Ö alt + 153	x alt + 158	º alt + 167	« alt + 174
\ alt + 92	Ú alt + 233	Ü alt + 154	+ alt + 246	° alt + 248	» alt + 175

Figura 2.1: Carateres da tabela **ASCII**. Fonte: https://theasciicode.com.ar/ascii-control-characters/delete-ascii-code-127.html#google_vignette

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 decimal.

```
# 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)', type(s) is 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 apelidos³. De seguida são apresentados exemplos para efetuar operações com nomes de pessoas.

```
pos = '0123456789012345678901234567890123456789'
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 caractere 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.....: 0123456789012345678901234567890123456789
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:

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

```

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])
#
# 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 início da fatia ou o de final (ou ambos), o início 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 caractere da string é considerado caso este valor seja omitido);
- **fim - 1**: é o último índice a ser considerado (o último caractere 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.....: 0123456789012345678901234567890123456789
.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 Pedacos

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.

```

0123456789012345678901234567890123456789
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

```

# 0123456789012345678901234567890123456789
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.

```

0123456789012345678901234567890123456789
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`.

```
#      0123456789012345678901234567890123456789
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"4) {m.find('Tomé', 15)}")
print(f"5) {m.find('Tomé', 11)}")
print(f"6) {m.find('José', 5, 12)}")
print(f"7) {'José' in m}")
```

Listagem 2.23: Resultado.

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

Listagem 2.24: Resultado.

2.8 Substituir Texto Em Strings

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

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

print('h + m2: substituir 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: adicionar apelido de f a m1 (não esquecer o espaço)
i) .Lígia Isabel Mendes Belo Pereira.
   .Lígia Isabel Mendes Belo.
   .Henrique Nogueira Pereira.
```

```
h + m2: substituir 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.removeprefix(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 do 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 Python¹.

É possível modificar elementos nas listas. Dizemos que as listas são tipos de dados **mutáveis**. Este conceito ficará mais interessante quando estudarmos tuplas, que são tipos de dados que não se podem modificar².

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://algoritosempython.com.br/cursos/programacao-python/listas>

²<https://algoritosempython.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 definidos³. 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']
```

³<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 decendente
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 várias 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 ficheiro¹:

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 eletrónico²:

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 ficheiros³:

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.

```
1 leite meio-gordo 800 ml
2 açúcar 100 g
3 casca de limão 1 unid.
4 pau de canela 2 unid.
5 pão cacete 8 fatia
6 ovo M3 unid.
7 óleo para fritar 500 ml
8 açúcar (para polvilhar) 2 c. de sopa
9 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, 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 específico capaz de interpretar/transformar/descodificar o seu conteúdo.

4.3 Extensões nos Nomes 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`.

4.4 Conceito de bit, byte e Representação de Números

Um bit é a unidade elementar de armazenamento em sistemas binários. Ou seja, têm apenas dois dígitos o zero (0) e o um (1). Um byte é um conjunto de oito bits. A Tabela 4.1 ilustra o conceito de bit e byte com um exemplo.

Valor	128	64	32	16	8	4	2	1
Bit N ^o	1	2	3	4	5	6	7	8
Bits	0	1	0	0	0	0	0	1

Tabela 4.1: Byte composto por 8 bits (valores: 0,1). Contém o valor 65, letra A.

A representação de um número decimal (base 10: 0123456789) em binário (base 2: 01) têm mais dígitos (bits) do que sua representação em decimal. Por outro lado, a representação em hexadecimal (base 16: 01234567890ABCDEF) tem menos dígitos. A Tabela 4.2 mostra diversos números escritos em diversas bases: decimal, binário e octal.

DEC	BIN	HEX	DEC	BIN	HEX
				1234567890123456	12345
0	0	0	00000	0000000000000000	00000
1	1	1	00001	0000000000000001	00001
2	10	2	00002	0000000000000010	00002
3	11	3	00003	0000000000000011	00003
4	100	4	00004	0000000000000100	00004
7	111	7	00007	0000000000000111	00007
10	1010	A	00010	0000000000001010	0000A
15	1111	F	00015	0000000000001111	0000F
16	10000	10	00016	0000000000010000	00010
64	1000000	40	00064	0000000001000000	00040
65	1000001	41	00065	0000000001000001	00041
127	1111111	7F	00127	0000000001111111	0007F
128	10000000	80	00128	0000000010000000	00080
255	11111111	FF	00255	0000000011111111	000FF
256	100000000	100	00256	0000000100000000	00100
789	1100010101	315	00789	0000001100010101	00315
1023	1111111111	3FF	01023	0000001111111111	003FF
1050	10000011010	41A	01050	0000010000011010	0041A
32767	11111111111111	7FFF	32767	0111111111111111	07FFF
65535	111111111111111	FFFF	65535	1111111111111111	0FFFF
65536	1000000000000000	10000	65536	1000000000000000	10000

Tabela 4.2: Exemplos de números escritos em decimal, binário e octal.

```

numeros = [0, 1, 2, 3, 4, 7, 10, 15, 16, 64, 65, 127, 128, 255, 256, 789, pow(2, 10) - 1,
            1050,
            pow(2, 15) - 1, pow(2, 16) - 1, pow(2, 16), pow(2, 31) - 1, pow(2, 32) - 1, pow(2,
            32),
            pow(2, 33) - 1, pow(2, 33)]

import math
for n in numeros:

```

DEC	BIN	HEX
	12345678901234567890123456789012	12345678
2147483647	00111111111111111111111111111111	07FFFFFFF
4294967295	01111111111111111111111111111111	0FFFFFFF
4294967296	10000000000000000000000000000000	100000000
8589934591	11111111111111111111111111111111	1FFFFFFF
8589934592	10000000000000000000000000000000	200000000

Tabela 4.3: Exemplos de números escritos em decimal, binário e octal (cont.).

```

if n <= 65536:
    x = 16
    print(f"{n:5d} &{n:12b} & {n:5X} &{n:05d} &{n:0{x}b} &{n:05X}\\\\\\")
else:
    x = 33
    print(f"{n:5d} &{n:0{x}b} &{n:09X}\\\\\\")

```

Listagem 4.2: Script para gerar as tabelas 4.2 e 4.3

Para calcular o número de bits necessários para representar números decimais em binário, pode-se usar $\log_2(N)$ arredondado para cima. Onde N é a quantidade de números diferentes que se pode escrever. Exemplo com oito bites podem-se escrever números de 0 a 255 (256 números diferentes). As fórmulas seguintes permitem calcular o número de bits e bytes:

$$nbits = \log_2(N) = \log_2(256) = 8$$

$$nbytes = \frac{nbits}{8} = \frac{8}{8} = 1$$

A Tabela 4.4 apresenta o número de bits e bytes necessários para escrever em binário diversos intervalos de números.

```

import math
for n2 in range(1, 34, 1):
    bits = 1
    hexs = 1
    n = pow(2, n2)
    if n > 0:
        bits = math.ceil(math.log2(n + 0.0))
        bytes = math.ceil(bits/8)
        # math.ceil(math.log2(n) / math.log2(16))
    print(f"{n2} &{n:12d} & {0}-{n-1:d}& {bits:2d}& {bytes:2d}\\\\\\")

```

Listagem 4.3: Script para gerar a tabela ??

4.5 Unidade de Medida de Memória e Armazenamento

A quantidade de bytes que um volume de dados contém pode ser gigantesca. Logo, para termos noções mais claras da capacidade de uma unidade de armazenamento, de um fluxo de dados transmitidos ou simplesmente do tamanho de um ficheiro, por exemplo, utilizamos

N	2^N	Intervalo	Bits	Bytes
1	2	0-1	1	1
2	4	0-3	2	1
3	8	0-7	3	1
4	16	0-15	4	1
5	32	0-31	5	1
6	64	0-63	6	1
7	128	0-127	7	1
8	256	0-255	8	1
9	512	0-511	9	2
10	1024	0-1023	10	2
11	2048	0-2047	11	2
12	4096	0-4095	12	2
13	8192	0-8191	13	2
14	16384	0-16383	14	2
15	32768	0-32767	15	2
16	65536	0-65535	16	2
17	131072	0-131071	17	3
18	262144	0-262143	18	3
19	524288	0-524287	19	3
20	1048576	0-1048575	20	3
21	2097152	0-2097151	21	3
22	4194304	0-4194303	22	3
23	8388608	0-8388607	23	3
24	16777216	0-16777215	24	3
25	33554432	0-33554431	25	4
26	67108864	0-67108863	26	4
27	134217728	0-134217727	27	4
28	268435456	0-268435455	28	4
29	536870912	0-536870911	29	4
30	1073741824	0-1073741823	30	4
31	2147483648	0-2147483647	31	4
32	4294967296	0-4294967295	32	4
33	8589934592	0-8589934591	33	5

Tabela 4.4: Número de bits e bytes para representar números decimais em binário.

medidas padronizadas. Um kilobyte consiste num conjunto de 1.024 bytes. Já 1 megabyte corresponde a 1.024 kilobytes e assim se segue, como se mostra na Tabela 4.5.

```

unidades = [('kilo', 'KB'), ('mega', 'MB'), ('giga', 'GB'),
            ('tera', 'TB'), ('peta', 'PB'), ('exa', 'EB'),
            ('zetta', 'ZB'), ('yotta', 'YB')]

k = 1024
ua, Ua = '', ''
for n in range(len(unidades)):
    u, U = unidades[n]
    if ua != '':
        ua + 'byte'
    print(f"1& {u}byte & {U} & 1024 & {ua}byte & {k:,}\\\\"")
    ua, Ua = u, U
    n = n + 1

```

Unidade			Bytes		
1	kilobyte	KB	1024	byte	1,024
1	megabyte	MB	1024	kilobyte	1,048,576
1	gigabyte	GB	1024	megabyte	1,073,741,824
1	terabyte	TB	1024	gigabyte	1,099,511,627,776
1	petabyte	PB	1024	terabyte	1,125,899,906,842,624
1	exabyte	EB	1024	petabyte	1,152,921,504,606,846,976
1	zettabyte	ZB	1024	exabyte	1,180,591,620,717,411,303,424
1	yottabyte	YB	1024	zettabyte	1,208,925,819,614,629,174,706,176

Tabela 4.5: Unidades de medida: do kilobyte ao yottabyte.

```
k = k * 1024
```

Listagem 4.4: Script para gerar a tabela ??

4.6 Exemplo de Escrita de Dados em Ficheiro de Texto e Binário

O exemplo da Listagem 4.6 ilustra um exemplo que consiste em armazenar os números de 0 a 255 ($\text{pow}(2,8)$) num ficheiro de texto e num ficheiro binário. No ficheiro de texto cada dígito dos números é armazenado com um byte. Por exemplo, o número 123 é armazenado com três bytes. No caso do ficheiro binário cada número é convertido para binário e armazenado em apenas um byte. Finalmente, são apresentados alguns dos atributos dos ficheiros e comparados os seus tamanhos. Neste caso o ficheiro binário (256 bytes) tem apenas 21.88% do tamanho do ficheiro de texto (256/1170 bytes).

```

import math

fic_txt = 'dados.txt'
fic_bin = 'dados.bin'
n = 8
N = pow(2, n)
num_bytes = math.ceil(n/8)
# Escreve números de 0 a 1023 num ficheiro de texto
f = open(fic_txt, 'wt', encoding='ascii')
for x in range(0, N):
    print(x, file=f)
f.close()

# Escreve números de 0 a 1023 num ficheiro binário (2 bytes/número)
f = open(fic_bin, 'wb')
for x in range(0, N):
    f.write(x.to_bytes(num_bytes, byteorder='big'))
    # file.write((i).to_bytes(24, byteorder='big', signed=False))
f.close()

def PropriedadesFicheiro(fic):
    import os
    from datetime import datetime
    print(f"Atributos do ficheiro: {fic}")
    # 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)
    # timestamp is number of seconds since 1970-01-01
    # timestamp: 1702317047.1003244 -> 2023-12-11 17:50:47.100324
    # convert the timestamp to a datetime object in the local timezone
    ts = os.path.getctime(fic)
    data_criacao = datetime.fromtimestamp(ts)
    # print the datetime object and its type
    N = os.path.getsize(fic)
    print("\tTamanho em bytes =", N)
    print("\tData criação =", data_criacao, f" timestamp: {ts}")
    print("\tTipo =", type(data_criacao))
    print("\tData alteração =", datetime.fromtimestamp(os.path.getmtime(fic)))
    return N

NT = PropriedadesFicheiro(fic_txt)
NB = PropriedadesFicheiro(fic_bin)
print(f"{NB}/{NT}*100 = {NB / NT * 100:.2f}%")

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

```

```

Atributos do ficheiro: dados.txt
Tamanho em bytes = 1170
Data criação = 2023-12-11 17:50:47.100324 timestamp: 1702317047.1003244
Tipo = <class 'datetime.datetime'>
Data alteração = 2023-12-12 18:44:05.732566
Atributos do ficheiro: dados.bin

```



```

Tamanho em bytes = 256
Data criação = 2023-12-11 17:50:47.094325   timestamp: 1702317047.0943253
Tipo = <class 'datetime.datetime'>
Data alteração = 2023-12-12 18:44:05.732566
256/1170x100 = 21.88%

```

Listagem 4.5: Atributos dos ficheiros dados.txt e dados.bin

A Figura 4.1 mostra parcialmente o conteúdo do ficheiro dados.bin. Como era de esperar cada número ocupa uma linha e cada linha tem o caractere de controlo fim de linha (CRLF).

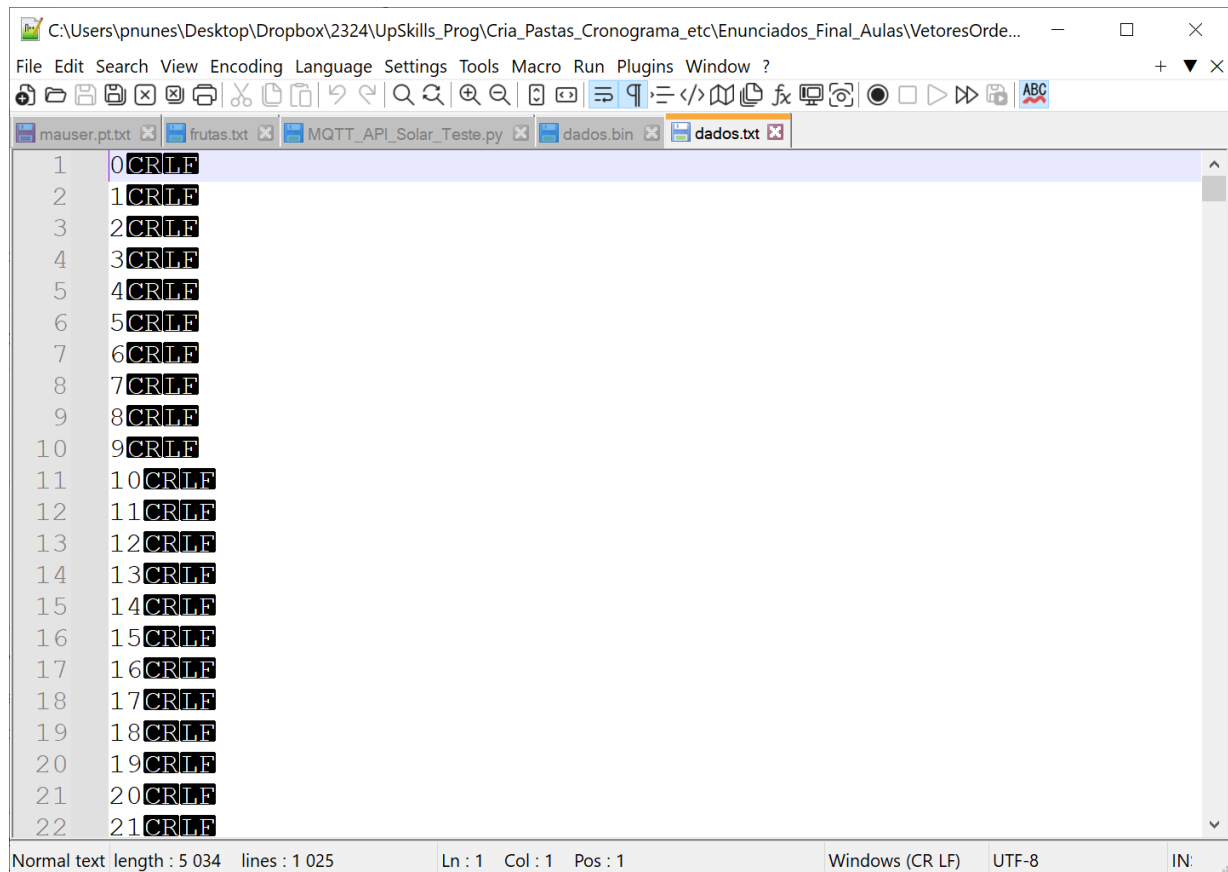


Figura 4.1: Conteúdo do ficheiro dados.txt visualizado com o editor de texto Notepad++.

A Figura 4.2 mostra o conteúdo do ficheiro dados.bin. Como era de esperar, contem caracteres de controlo (Ex: NULL, EOT). Na prática com tem todos os caracteres da tabela ASCII.

```

1 NULSOHSTXETXEOTENOACKBELBS>LF
2 VTFFCR
3 SOSIDLEDC1DC2DC3DC4NAKSYNETBCANEMSUBESC
  ES GS RS US
  !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJ
  KLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
  { | } ~ DEL € , f , , , , t ‡ ^ % Š < Ć Ž ' ' " " • — ™ Š > æ ž Ÿ ;
  ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ ® ¯ ° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿ À Á Â Ã Ä Å Æ Ç È É Ê Ë
  Ì Í Î Ï Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß à á â ã ä å æ ç è é ê ë ì í î ï ð ñ ò ó ô õ
  ö ÷ ø ù ú û ü ý þ ÿ

```

length : 256 lines : 3 Ln : 3 Col : 243 Pos : 257 Unix (LF) ANSI IN:

Figura 4.2: Conteúdo do ficheiro dados.bin visualizado com o editor de texto NotePad++.

4.7 Logotipo do UPskill em Diferentes Formatos de Ficheiro Binário de Imagens

As figuras 4.5 e 4.5 podem ser aumentadas com zoom do visualizador de PDF sem perder qualidade, por estão num formato vetorial. Neste formato as imagens são definidas por equações matemáticas em vez de pixels. As imagens PNG são compostas de pixels, que são pequenos quadrados coloridos que se unem para formar a imagem. As imagens PNG têm um tamanho fixo, o que significa que podem perder qualidade quando aumentadas ou diminuídas, tornando-as dependentes da resolução⁴.



Figura 4.3: Logo UPskill no formato PNG. Resolução (1250, 563), tamanho: 41,078 bytes.

⁴<https://cloudinary.com/guides/image-formats/svg-vs-png-4-key-differences-and-how-to-choose>



Figura 4.4: Logo UPskill no formato PNG. Zoom: 0.05.



Figura 4.5: Logo UPskill no formato SVG. A resolução pode ser infinita. tamanho: 56,554 bytes



Figura 4.6: Logo UPskill no formato SVG. Zoom: 0.05.

4.8 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 imprimíveis⁵.

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 4.7 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 4.6: 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. 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**.

⁵<https://realpython.com/python-csv>

```

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 4.7: 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 4.8 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 4.8: 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 caractere |.

```

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 4.9: 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 4.10 ilustra um exemplo de ficheiro de texto de comprimento fixo. O seu tamanho no disco são 370 bytes enquanto o ficheiro da Listagem 4.9. tem 270 bytes em disco, que corresponde a mais 37%.

```

1 123456789012345678901234567890123451234567890123456789012345678901234567890123456789012345678
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 4.10: Exemplo de conteúdo de ficheiro CSV com o símbolo delimitador |.

4.8.1 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;Cereiac;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;Lactícínios;4.92;6%
11;Natas;Lactícínios;0.79;6%
12;Ovos;Lactícínios;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;Oregãos;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;Lactícínios;2.59;6%

```

Listagem 4.11: 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 4.12: Leitura do ficheiro hipermercado.csv.

Cada linha do ficheiro é um elemento do 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 ['1;Ketchup;Mercearia Salgado;1.59;23%\n', '2;Atum;Mercearia Salgado;3.38;6%\n', '3; ...]
2 1;Ketchup;Mercearia Salgado;1.59;23%
3
4 2;Atum;Mercearia Salgado;3.38;6%
5
6 3;Cogumelos;Mercearia Salgado;1.98;23%
7
8 4;Bolachas Cacau;Mercearia Doce;1.39;23%
9
10 5;Cerelec;Mercearia Doce;5.65;6%
11
12 6;Heineken 12*25CL;Bebidas;8.99;23%
13 ...

```

Listagem 4.13: Leitura do ficheiro hipermercado.csv.

4.9 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](#)⁶.

```
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 4.14: 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', 'Cerealac', 'Mercearia Doce', '5.65', '6%']
['6', 'Heineken 12*25CL', 'Bebidas', '8.99', '23%']
['7', 'Gel Dove', 'Higiene e Beleza', '5.99', '23%']
...
```

Listagem 4.15: Leitura do ficheiro `hipermercado.csv`.

4.9.1 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)
```

⁶<https://realpython.com/python-csv>

Listagem 4.16: 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 4.17: Conteúdo ficheiro.

4.9.2 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 4.18: 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 4.19: 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 caractere

4.10 Vinhos

TODO

4.11 Incêndios

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

4.12 Produtos IVA Zero

<https://www.publico.pt/interactivos/evolucao-cabaz-iva-zero-ficou-mais-car-o-mais-barato>

4.13 Acidentes

pordata.pt

Bibliografia