OWASP
AppSec Europe
London 2nd-6th July 2018

Unicode: The hero or villain?

Input Validation of free-form Unicode text in Web Applications

Pawel Krawczyk

About

In application security since 90's - pentesting, security architecture, SSDLC, DevSecOps

Active developer Python, C, Java https://github.com/kravietz

OWASP - SAML, PL/SQL, authentication cheatsheets

WebCookies.org - web privacy and security scanner

Immusec.com - competitive pentesting & incident response in UK

Contact

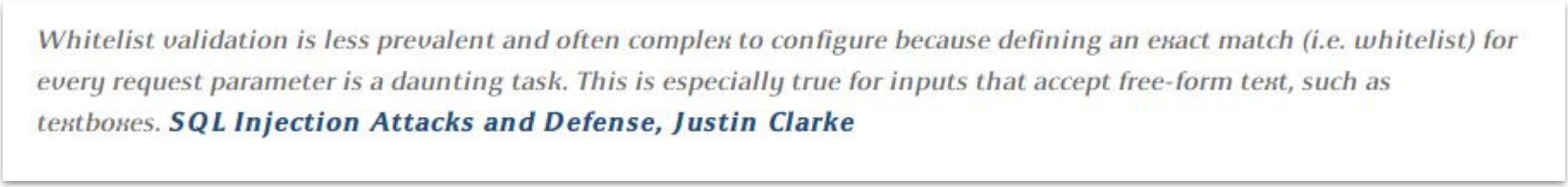
pkrawczyk@immusec.com

+44 7879 180015

https://www.linkedin.com/in/pawelkrawczyk

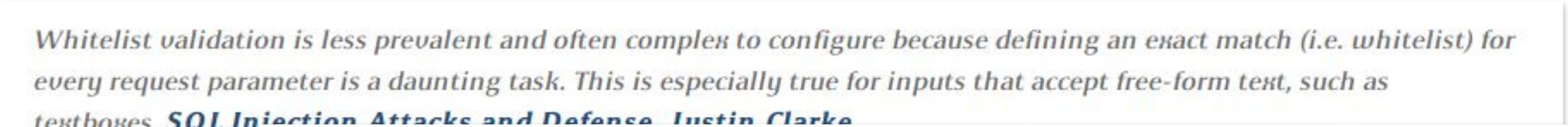
Definition of the problem

Free-form text validation

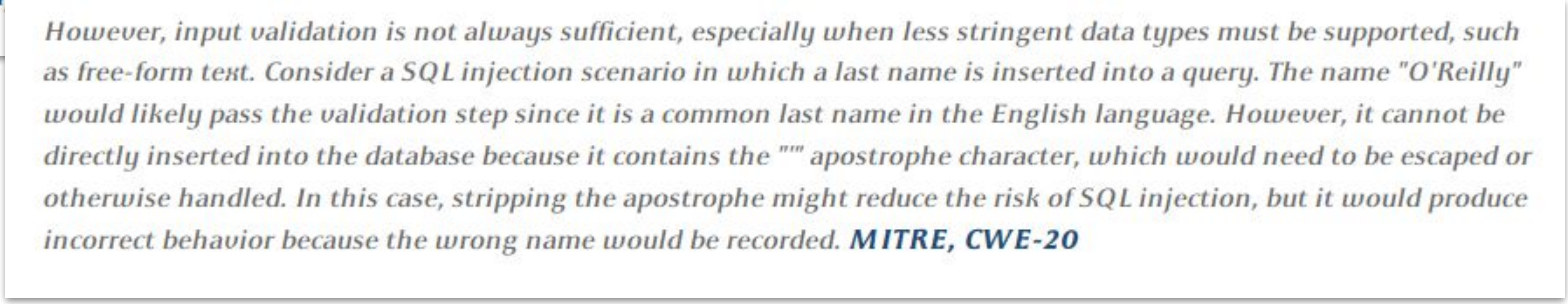


Whitelist validation is less prevalent and often complex to configure because defining an exact match (i.e. whitelist) for every request parameter is a daunting task. This is especially true for inputs that accept free-form text, such as textboxes. ***SQL Injection Attacks and Defense, Justin Clarke***

Free-form text validation

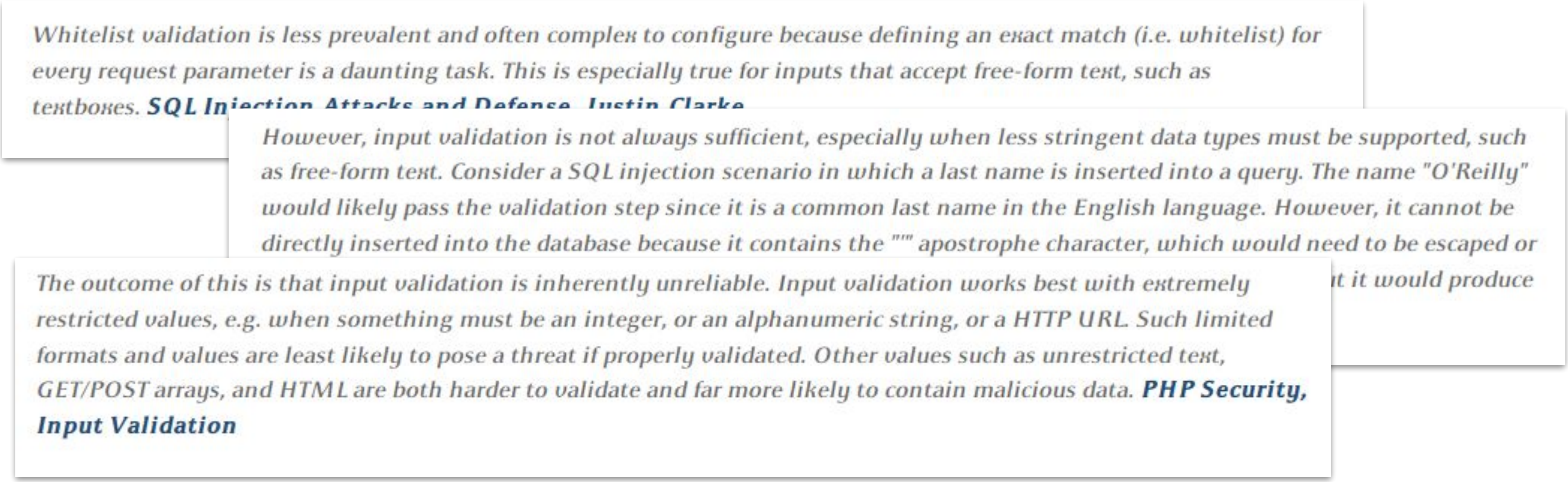


Whitelist validation is less prevalent and often complex to configure because defining an exact match (i.e. whitelist) for every request parameter is a daunting task. This is especially true for inputs that accept free-form text, such as textboxes. **SQL Injection Attacks and Defense, Justin Clarke**



However, input validation is not always sufficient, especially when less stringent data types must be supported, such as free-form text. Consider a SQL injection scenario in which a last name is inserted into a query. The name "O'Reilly" would likely pass the validation step since it is a common last name in the English language. However, it cannot be directly inserted into the database because it contains the "'" apostrophe character, which would need to be escaped or otherwise handled. In this case, stripping the apostrophe might reduce the risk of SQL injection, but it would produce incorrect behavior because the wrong name would be recorded. **MITRE, CWE-20**

Free-form text validation

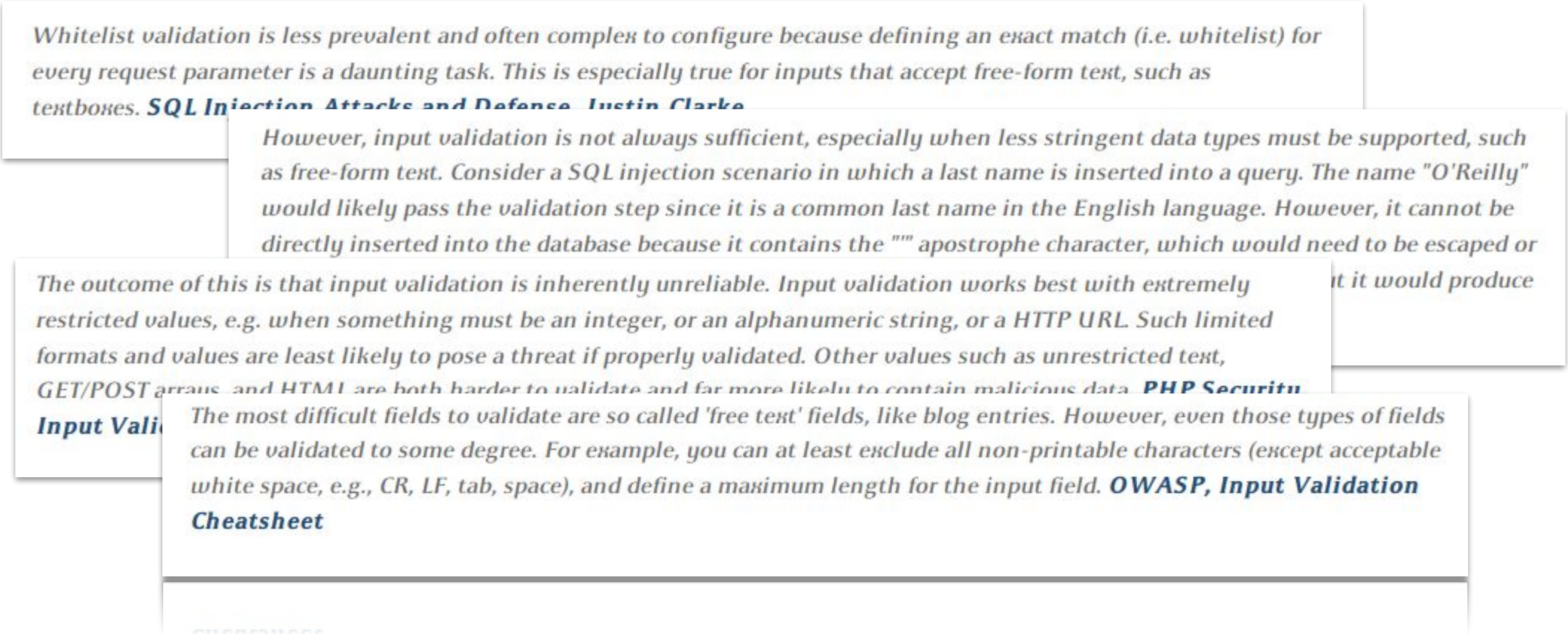


Whitelist validation is less prevalent and often complex to configure because defining an exact match (i.e. whitelist) for every request parameter is a daunting task. This is especially true for inputs that accept free-form text, such as textboxes. **SQL Injection Attacks and Defense, Justin Clarke**

However, input validation is not always sufficient, especially when less stringent data types must be supported, such as free-form text. Consider a SQL injection scenario in which a last name is inserted into a query. The name "O'Reilly" would likely pass the validation step since it is a common last name in the English language. However, it cannot be directly inserted into the database because it contains the "'" apostrophe character, which would need to be escaped or … *t it would produce*

The outcome of this is that input validation is inherently unreliable. Input validation works best with extremely restricted values, e.g. when something must be an integer, or an alphanumeric string, or a HTTP URL. Such limited formats and values are least likely to pose a threat if properly validated. Other values such as unrestricted text, GET/POST arrays, and HTML are both harder to validate and far more likely to contain malicious data. **PHP Security, Input Validation**

Free-form text validation



Whitelist validation is less prevalent and often complex to configure because defining an exact match (i.e. whitelist) for every request parameter is a daunting task. This is especially true for inputs that accept free-form text, such as textboxes. **SQL Injection Attacks and Defense, Justin Clarke**

However, input validation is not always sufficient, especially when less stringent data types must be supported, such as free-form text. Consider a SQL injection scenario in which a last name is inserted into a query. The name "O'Reilly" would likely pass the validation step since it is a common last name in the English language. However, it cannot be directly inserted into the database because it contains the "'" apostrophe character, which would need to be escaped or

t it would produce

The outcome of this is that input validation is inherently unreliable. Input validation works best with extremely restricted values, e.g. when something must be an integer, or an alphanumeric string, or a HTTP URL. Such limited formats and values are least likely to pose a threat if properly validated. Other values such as unrestricted text, GET/POST arrays, and HTML are both harder to validate and far more likely to contain malicious data. **PHP Security**

Input Vali

The most difficult fields to validate are so called 'free text' fields, like blog entries. However, even those types of fields can be validated to some degree. For example, you can at least exclude all non-printable characters (except acceptable white space, e.g., CR, LF, tab, space), and define a maximum length for the input field. **OWASP, Input Validation Cheatsheet**

Unicode Primer

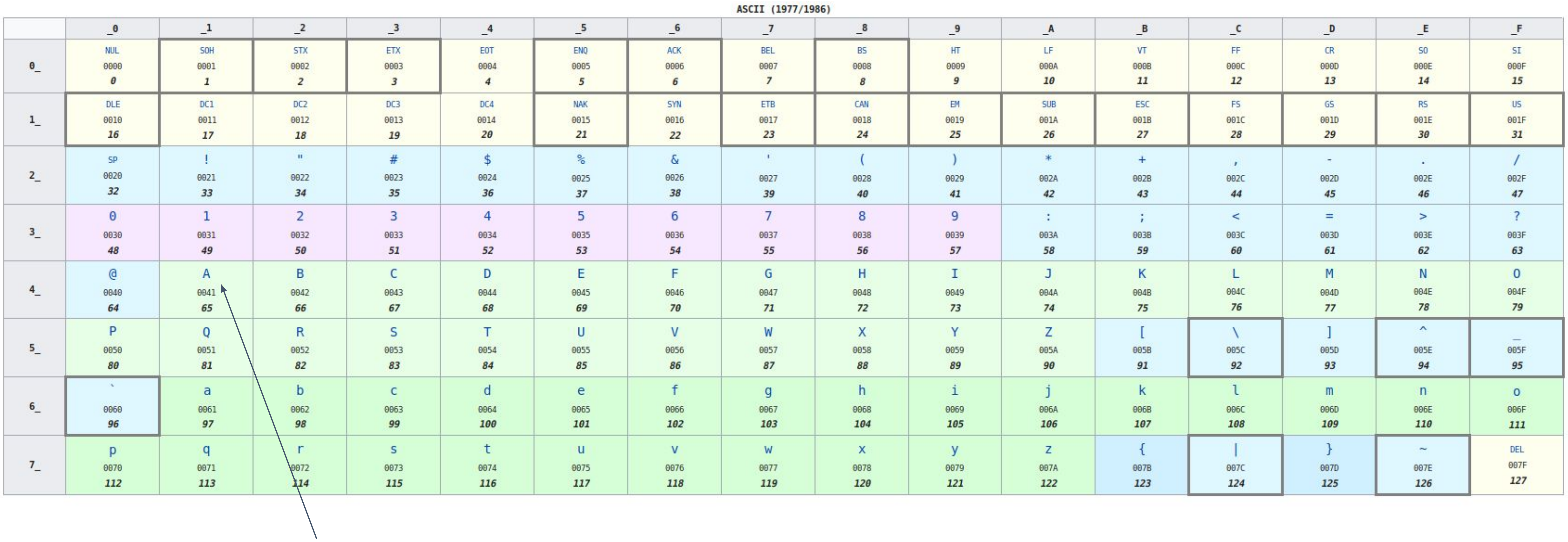
The rise and fall of letter “Ą”

Ą

Official name: A-OGONEK

“a letter in the Polish, Kashubian, Lithuanian, Creek, Navajo, Western Apache, Chiricahua, Osage, Hocąk, Mescalero, Gwich'in, Tutchone, and Elfdalian alphabets” (Wikipedia)

ASCII: just write “Ą” as “A”



ASCII (1977/1986)

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	NUL 0000 0	SOH 0001 1	STX 0002 2	ETX 0003 3	EOT 0004 4	ENQ 0005 5	ACK 0006 6	BEL 0007 7	BS 0008 8	HT 0009 9	LF 000A 10	VT 000B 11	FF 000C 12	CR 000D 13	SO 000E 14	SI 000F 15
1_	DLE 0010 16	DC1 0011 17	DC2 0012 18	DC3 0013 19	DC4 0014 20	NAK 0015 21	SYN 0016 22	ETB 0017 23	CAN 0018 24	EM 0019 25	SUB 001A 26	ESC 001B 27	FS 001C 28	GS 001D 29	RS 001E 30	US 001F 31
2_	SP 0020 32	! 0021 33	" 0022 34	# 0023 35	$ 0024 36	% 0025 37	& 0026 38	' 0027 39	(0028 40	) 0029 41	* 002A 42	+ 002B 43	, 002C 44	- 002D 45	. 002E 46	/ 002F 47
3_	0 0030 48	1 0031 49	2 0032 50	3 0033 51	4 0034 52	5 0035 53	6 0036 54	7 0037 55	8 0038 56	9 0039 57	: 003A 58	; 003B 59	< 003C 60	= 003D 61	> 003E 62	? 003F 63
4_	@ 0040 64	A 0041 65	B 0042 66	C 0043 67	D 0044 68	E 0045 69	F 0046 70	G 0047 71	H 0048 72	I 0049 73	J 004A 74	K 004B 75	L 004C 76	M 004D 77	N 004E 78	O 004F 79
5_	P 0050 80	Q 0051 81	R 0052 82	S 0053 83	T 0054 84	U 0055 85	V 0056 86	W 0057 87	X 0058 88	Y 0059 89	Z 005A 90	[005B 91	\ 005C 92	] 005D 93	^ 005E 94	_ 005F 95
6_	` 0060 96	a 0061 97	b 0062 98	c 0063 99	d 0064 100	e 0065 101	f 0066 102	g 0067 103	h 0068 104	i 0069 105	j 006A 106	k 006B 107	l 006C 108	m 006D 109	n 006E 110	o 006F 111
7_	p 0070 112	q 0071 113	r 0072 114	s 0073 115	t 0074 116	u 0075 117	v 0076 118	w 0077 119	x 0078 120	y 0079 121	z 007A 122	{ 007B 123	| 007C 124	} 007D 125	~ 007E 126	DEL 007F 127

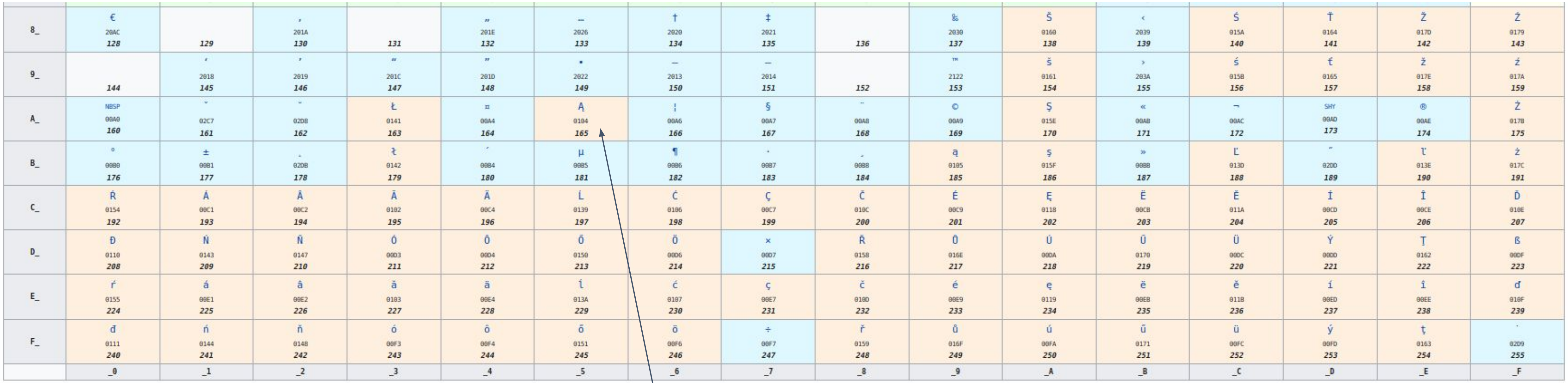
ASCII: just write “Ą” as “A”

(Pol.) KĄT = (Eng.) ANGLE

(Pol.) KAT = (Eng.) HANGMAN

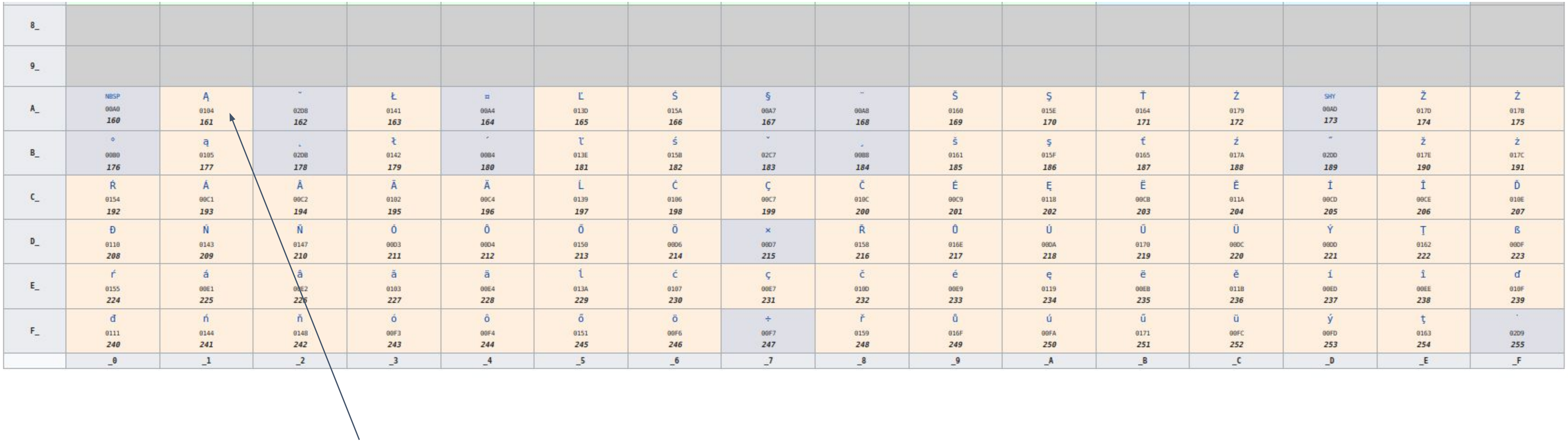
Contextual guessing, confusion, misunderstandings, we had lots of fun on IRC back in 90’s...

Windows-1250: “Ą” is 0xa5



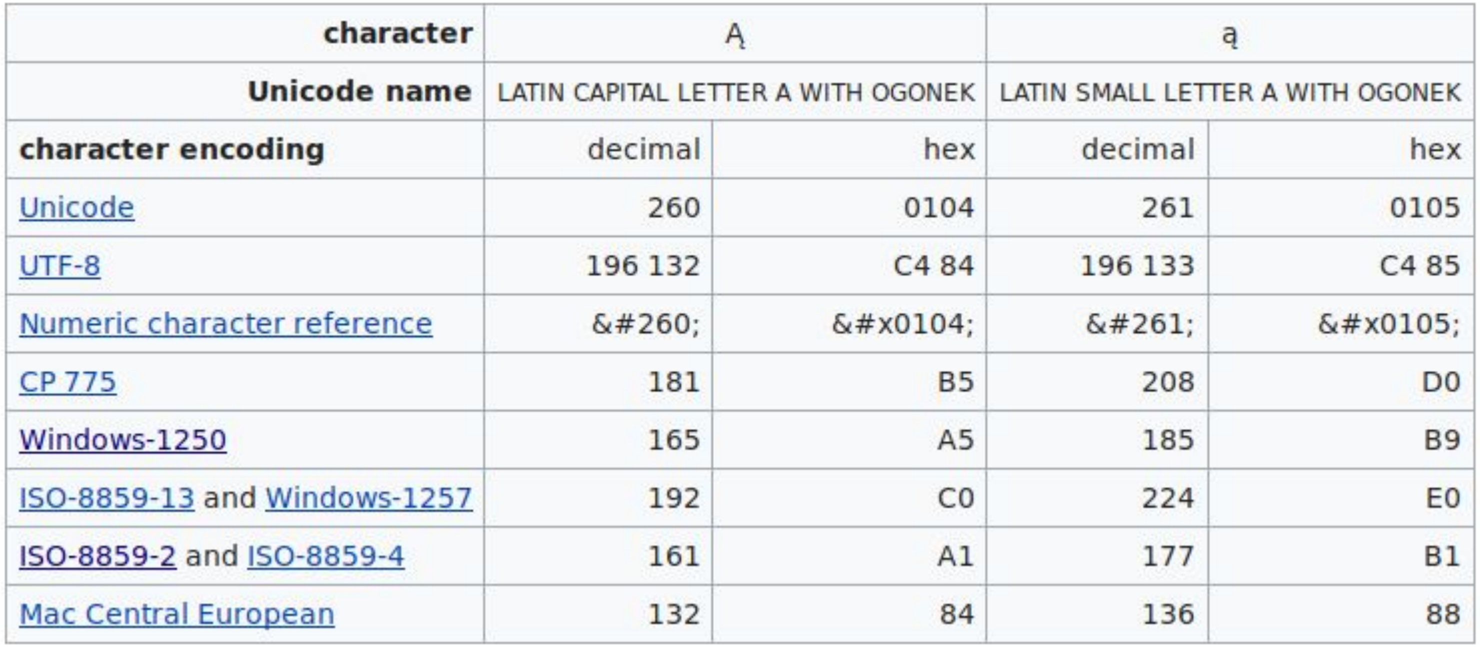
8_	€ 20AC 128	129	‚ 201A 130	131	„ 201E 132	… 2026 133	† 2020 134	‡ 2021 135	136	‰ 2030 137	Š 0160 138	‹ 2039 139	Ś 015A 140	Ť 0164 141	Ž 017D 142	Ź 0179 143
9_	144	‘ 2018 145	’ 2019 146	“ 201C 147	” 201D 148	• 2022 149	– 2013 150	— 2014 151	152	™ 2122 153	š 0161 154	› 203A 155	ś 015B 156	ť 0165 157	ž 017E 158	ź 017A 159
A_	NBSP 00A0 160	ˇ 02C7 161	˘ 02D8 162	Ł 0141 163	¤ 00A4 164	Ą 0104 165	¦ 00A6 166	§ 00A7 167	¨ 00A8 168	© 00A9 169	Ş 015E 170	« 00AB 171	¬ 00AC 172	SHY 00AD 173	® 00AE 174	Ż 017B 175
B_	° 00B0 176	± 00B1 177	˛ 02DB 178	ł 0142 179	´ 00B4 180	µ 00B5 181	¶ 00B6 182	· 00B7 183	¸ 00B8 184	ą 0105 185	ş 015F 186	» 00BB 187	Ľ 013D 188	˝ 02DD 189	ľ 013E 190	ż 017C 191
C_	Ŕ 0154 192	Á 00C1 193	Â 00C2 194	Ă 0102 195	Ä 00C4 196	Ĺ 0139 197	Ć 0106 198	Ç 00C7 199	Č 010C 200	É 00C9 201	Ę 0118 202	Ë 00CB 203	Ě 011A 204	Í 00CD 205	Î 00CE 206	Ď 010E 207
D_	Đ 0110 208	Ń 0143 209	Ň 0147 210	Ó 00D3 211	Ô 00D4 212	Ő 0150 213	Ö 00D6 214	× 00D7 215	Ř 0158 216	Ů 016E 217	Ú 00DA 218	Ű 0170 219	Ü 00DC 220	Ý 00DD 221	Ţ 0162 222	ß 00DF 223
E_	ŕ 0155 224	á 00E1 225	â 00E2 226	ă 0103 227	ä 00E4 228	ĺ 013A 229	ć 0107 230	ç 00E7 231	č 010D 232	é 00E9 233	ę 0119 234	ë 00EB 235	ě 011B 236	í 00ED 237	î 00EE 238	ď 010F 239
F_	đ 0111 240	ń 0144 241	ň 0148 242	ó 00F3 243	ô 00F4 244	ő 0151 245	ö 00F6 246	÷ 00F7 247	ř 0159 248	ů 016F 249	ú 00FA 250	ű 0171 251	ü 00FC 252	ý 00FD 253	ţ 0163 254	˙ 02D9 255
	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F

ISO-8859-2: “Ą” is 0xa1



8_																
9_																
A_	NBSP 00A0 160	Ą 0104 161	˘ 02D8 162	Ł 0141 163	¤ 00A4 164	Ľ 013D 165	Ś 015A 166	§ 00A7 167	¨ 00A8 168	Š 0160 169	Ş 015E 170	Ť 0164 171	Ź 0179 172	SHY 00AD 173	Ž 017D 174	Ż 017B 175
B_	° 00B0 176	ą 0105 177	˛ 02DB 178	ł 0142 179	´ 00B4 180	ľ 013E 181	ś 015B 182	ˇ 02C7 183	¸ 00B8 184	š 0161 185	ş 015F 186	ť 0165 187	ź 017A 188	˝ 02DD 189	ž 017E 190	ż 017C 191
C_	Ŕ 0154 192	Á 00C1 193	Â 00C2 194	Ă 0102 195	Ä 00C4 196	Ĺ 0139 197	Ć 0106 198	Ç 00C7 199	Č 010C 200	É 00C9 201	Ę 0118 202	Ë 00CB 203	Ě 011A 204	Í 00CD 205	Î 00CE 206	Ď 010E 207
D_	Đ 0110 208	Ń 0143 209	Ň 0147 210	Ó 00D3 211	Ô 00D4 212	Ő 0150 213	Ö 00D6 214	× 00D7 215	Ř 0158 216	Ů 016E 217	Ú 00DA 218	Ű 0170 219	Ü 00DC 220	Ý 00DD 221	Ţ 0162 222	ß 00DF 223
E_	ŕ 0155 224	á 00E1 225	â 00E2 226	ă 0103 227	ä 00E4 228	ĺ 013A 229	ć 0107 230	ç 00E7 231	č 010D 232	é 00E9 233	ę 0119 234	ë 00EB 235	ě 011B 236	í 00ED 237	î 00EE 238	ď 010F 239
F_	đ 0111 240	ń 0144 241	ň 0148 242	ó 00F3 243	ô 00F4 244	ő 0151 245	ö 00F6 246	÷ 00F7 247	ř 0159 248	ů 016F 249	ú 00FA 250	ű 0171 251	ü 00FC 252	ý 00FD 253	ţ 0163 254	˙ 02D9 255
	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F

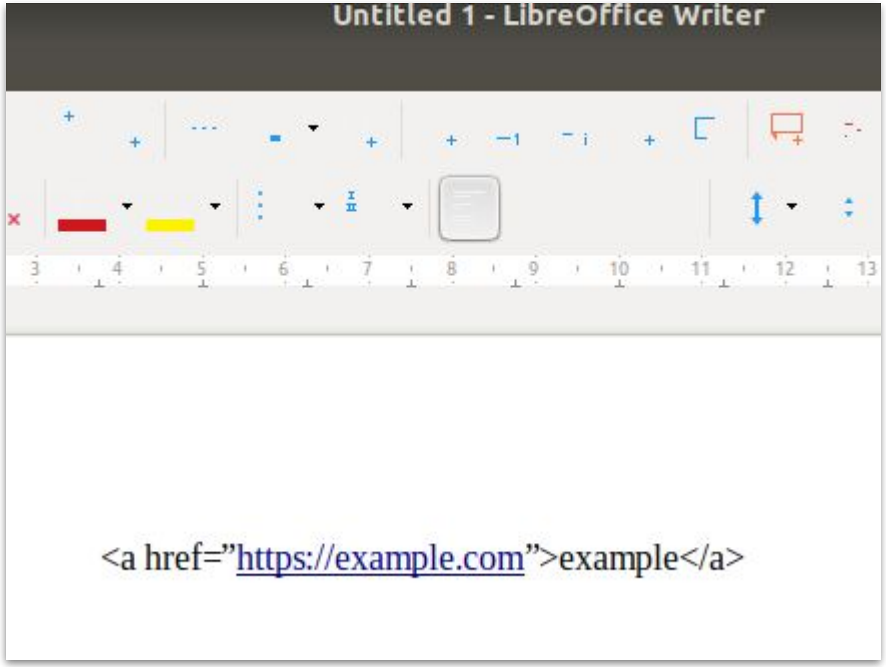
To moÅ¼liwe!



character	Ą		ą	
Unicode name	LATIN CAPITAL LETTER A WITH OGONEK		LATIN SMALL LETTER A WITH OGONEK	
character encoding	decimal	hex	decimal	hex
Unicode	260	0104	261	0105
UTF-8	196 132	C4 84	196 133	C4 85
Numeric character reference	&#260;	&#x0104;	&#261;	&#x0105;
CP 775	181	B5	208	D0
Windows-1250	165	A5	185	B9
ISO-8859-13 and Windows-1257	192	C0	224	E0
ISO-8859-2 and ISO-8859-4	161	A1	177	B1
Mac Central European	132	84	136	88

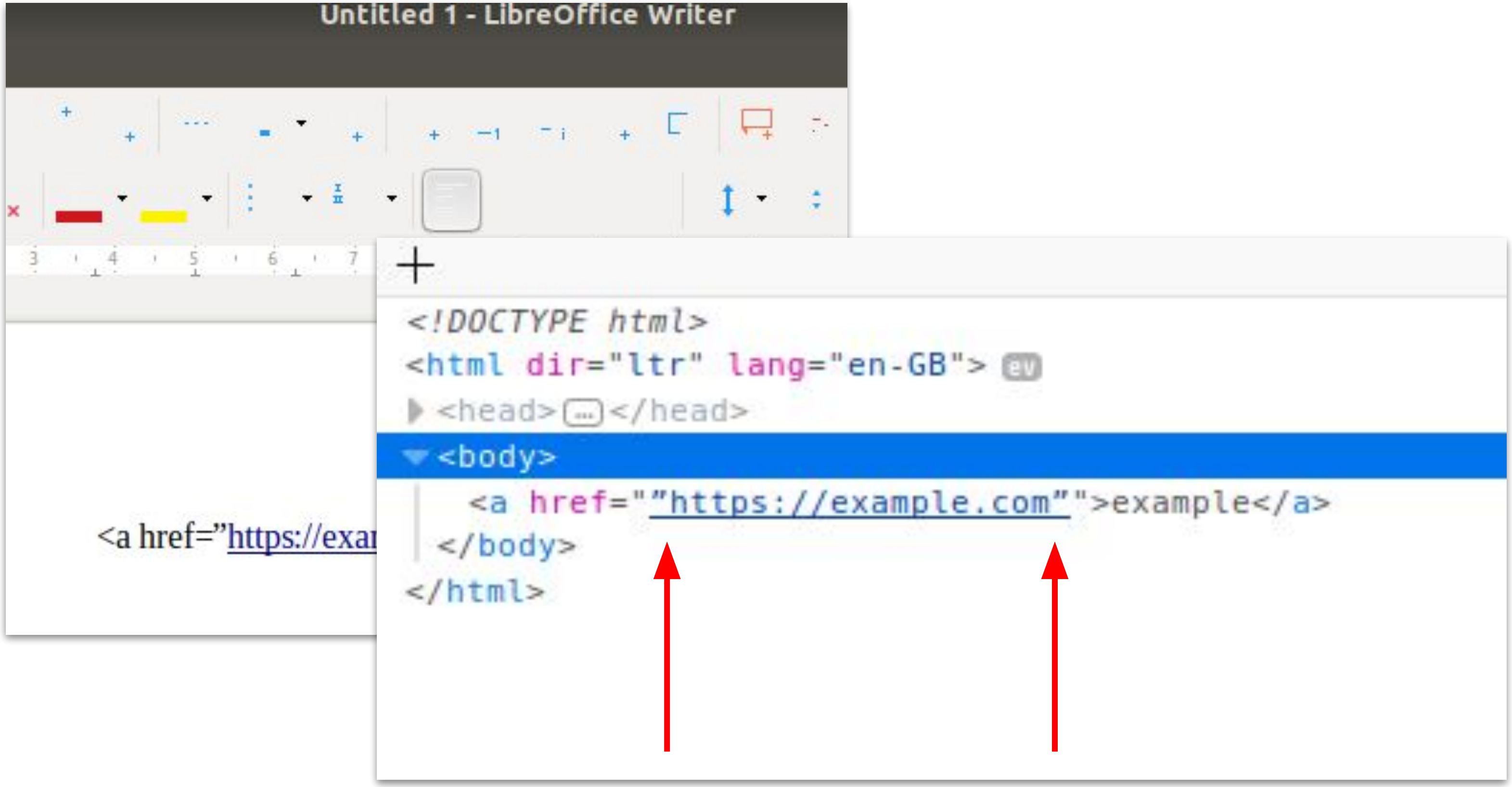
Source: Wikipedia

Confused beyond diacritics



<a href=”https://example.com”>example</a>

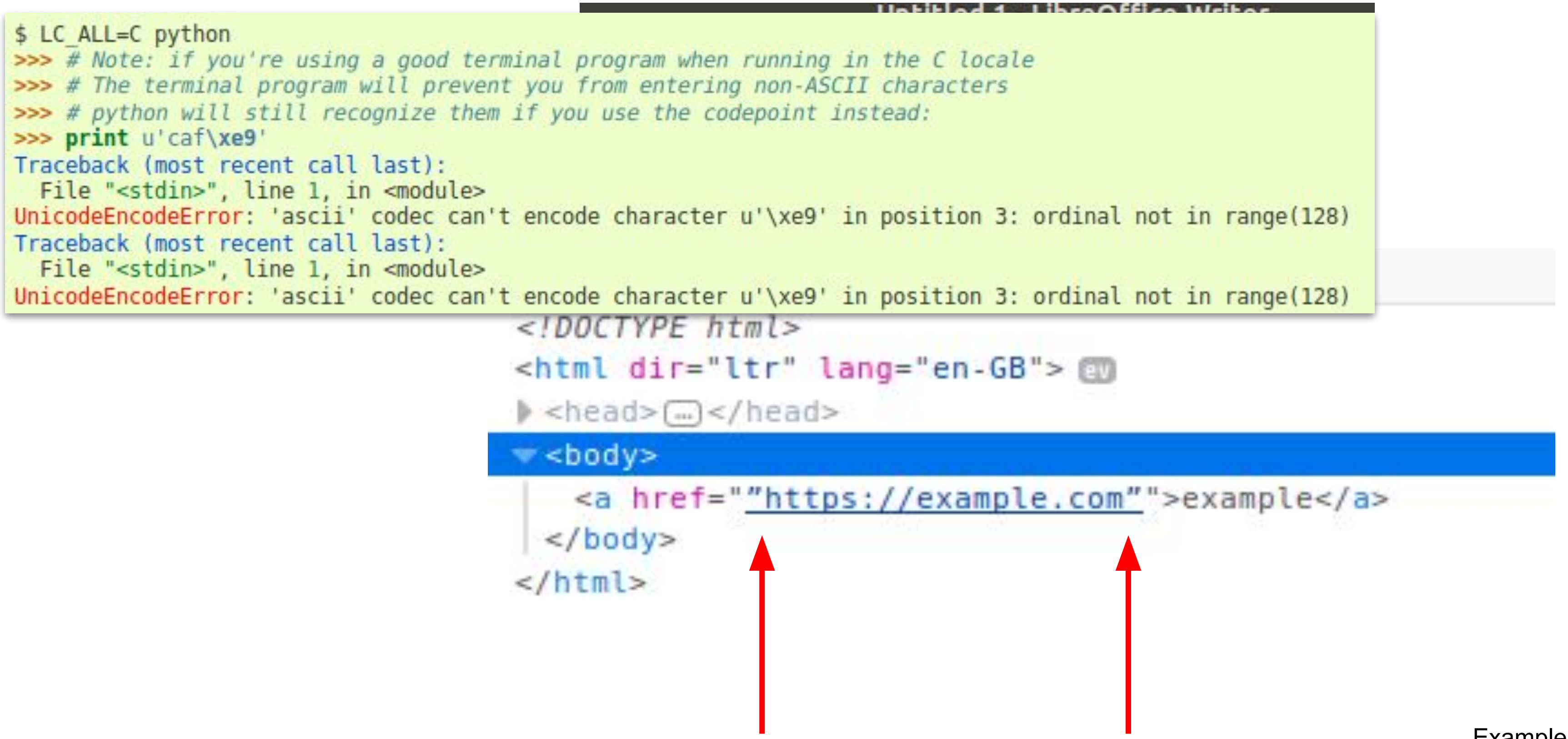
Confused beyond diacritics



<a href=”https://exar

```
<!DOCTYPE html>
<html dir="ltr" lang="en-GB">
▸ <head>…</head>
▾ <body>
    <a href="”https://example.com”">example</a>
  </body>
</html>
```

Confusing Unicode

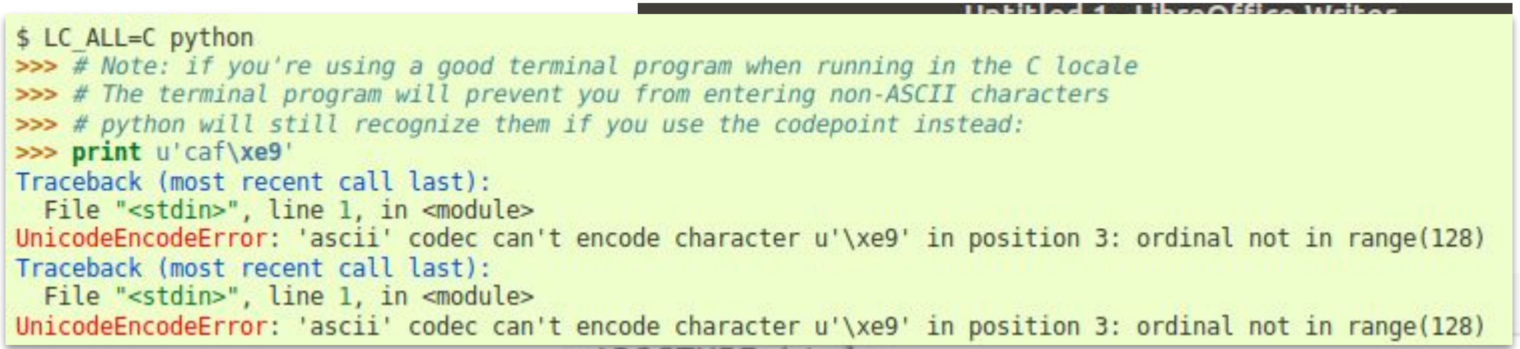


```
$ LC_ALL=C python
>>> # Note: if you're using a good terminal program when running in the C locale
>>> # The terminal program will prevent you from entering non-ASCII characters
>>> # python will still recognize them if you use the codepoint instead:
>>> print u'caf\xe9'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeEncodeError: 'ascii' codec can't encode character u'\xe9' in position 3: ordinal not in range(128)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeEncodeError: 'ascii' codec can't encode character u'\xe9' in position 3: ordinal not in range(128)
```

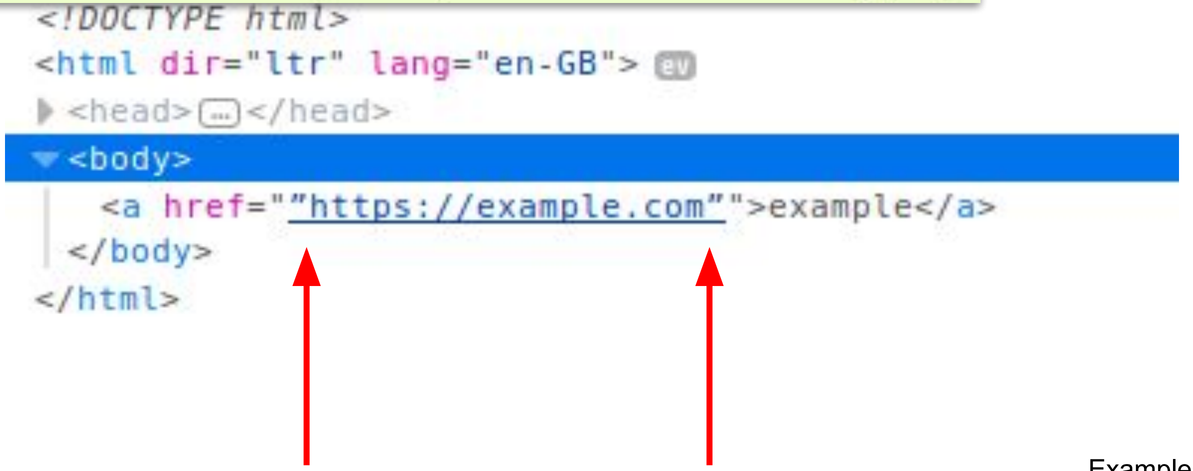
```
<!DOCTYPE html>
<html dir="ltr" lang="en-GB"> ev
▶ <head>…</head>
▼ <body>
    <a href="”https://example.com”">example</a>
  </body>
</html>
```

Examples: PythonHosted.org, Luciano Ramalho “Fluent Python”

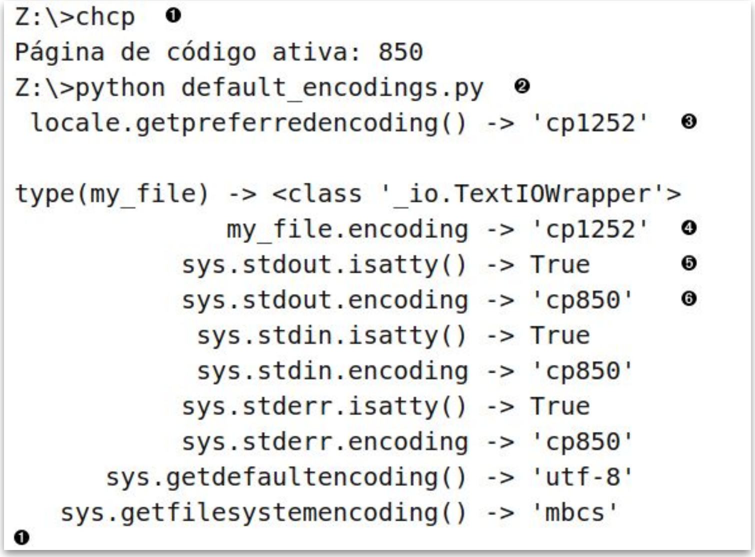
Confusing Unicode



```
$ LC_ALL=C python
>>> # Note: if you're using a good terminal program when running in the C locale
>>> # The terminal program will prevent you from entering non-ASCII characters
>>> # python will still recognize them if you use the codepoint instead:
>>> print u'caf\xe9'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeEncodeError: 'ascii' codec can't encode character u'\xe9' in position 3: ordinal not in range(128)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeEncodeError: 'ascii' codec can't encode character u'\xe9' in position 3: ordinal not in range(128)
```



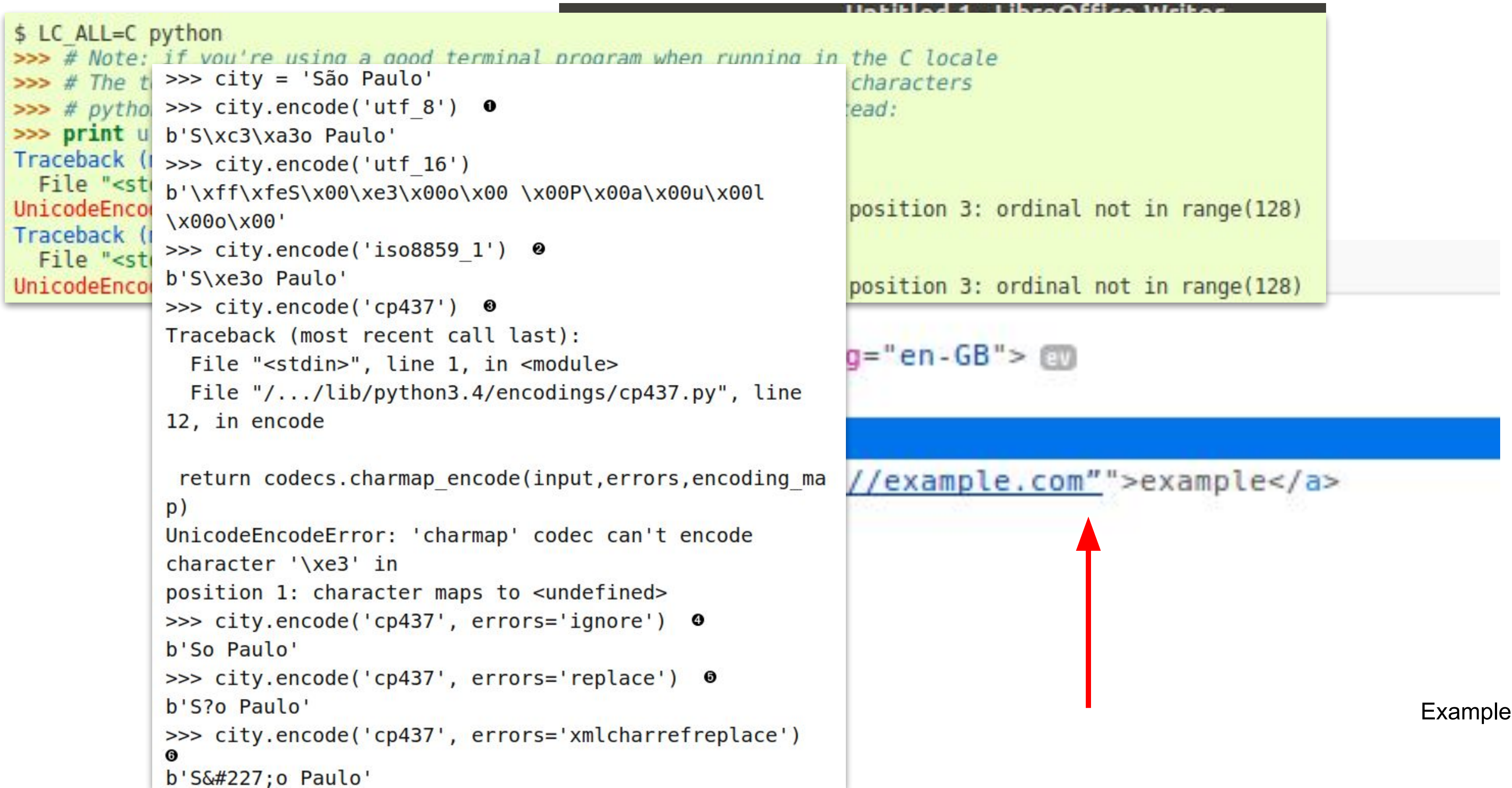
```
<!DOCTYPE html>
<html dir="ltr" lang="en-GB"> ev
▶ <head>…</head>
▼ <body>
    <a href=""https://example.com"">example</a>
  </body>
</html>
```



```
Z:\>chcp ❶
Página de código ativa: 850
Z:\>python default_encodings.py ❷
 locale.getpreferredencoding() -> 'cp1252' ❸

type(my_file) -> <class '_io.TextIOWrapper'>
              my_file.encoding -> 'cp1252' ❹
           sys.stdout.isatty() -> True     ❺
           sys.stdout.encoding -> 'cp850'  ❻
            sys.stdin.isatty() -> True
            sys.stdin.encoding -> 'cp850'
           sys.stderr.isatty() -> True
           sys.stderr.encoding -> 'cp850'
      sys.getdefaultencoding() -> 'utf-8'
   sys.getfilesystemencoding() -> 'mbcs'
❼
```

Examples: PythonHosted.org, Luciano Ramalho “Fluent Python”

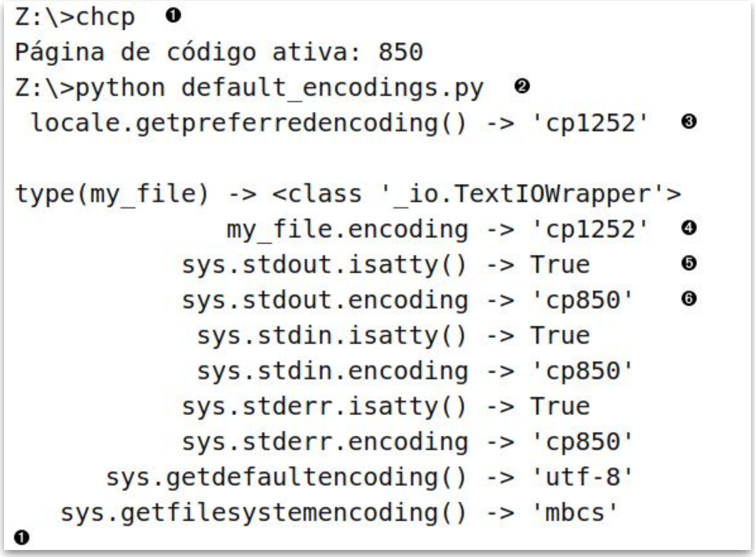
Confusing Unicode



```
$ LC_ALL=C python
>>> # Note: if you're using a good terminal program when running in the C locale
>>> # The t                                                           characters
>>> # pytho                                                           ead:
>>> print u
Traceback (
  File "<st
UnicodeEnco                                                           position 3: ordinal not in range(128)
Traceback (
  File "<st
UnicodeEnco                                                           position 3: ordinal not in range(128)
```

```
g="en-GB"> ev
//example.com”">example</a>
```

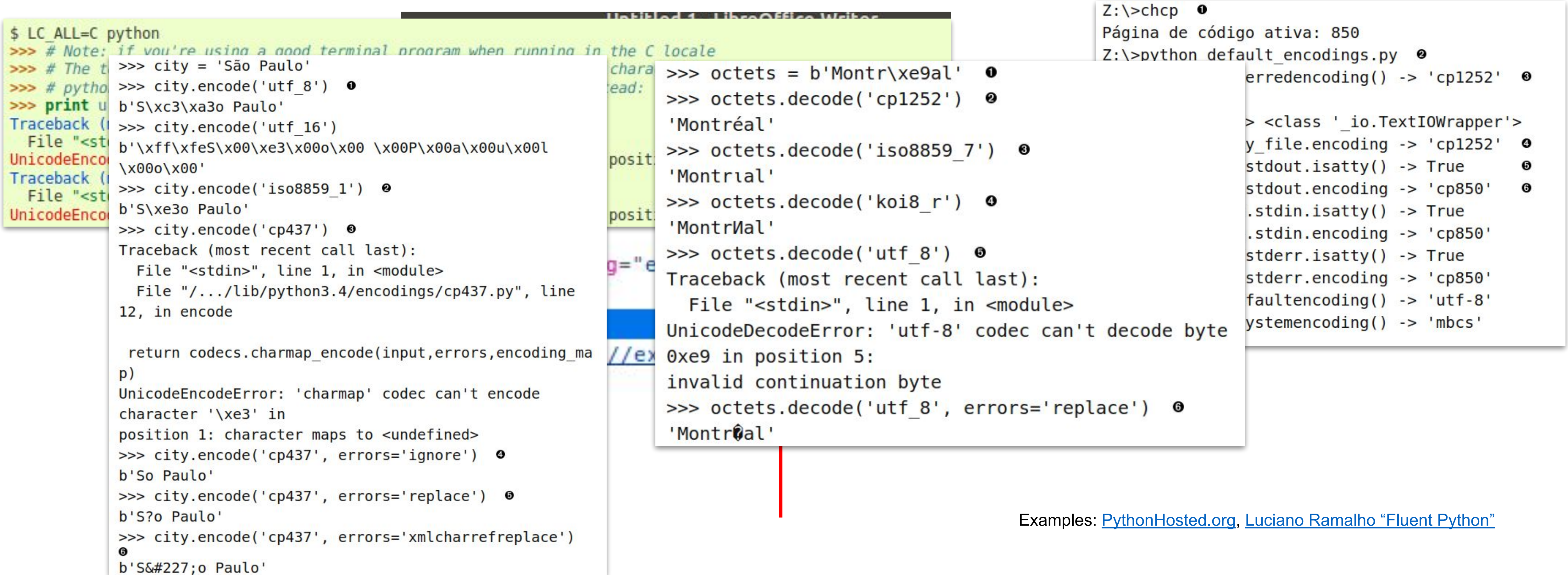
```
>>> city = 'São Paulo'
>>> city.encode('utf_8')  ❶
b'S\xc3\xa3o Paulo'
>>> city.encode('utf_16')
b'\xff\xfeS\x00\xe3\x00o\x00 \x00P\x00a\x00u\x00l
\x00o\x00'
>>> city.encode('iso8859_1')  ❷
b'S\xe3o Paulo'
>>> city.encode('cp437')  ❸
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/.../lib/python3.4/encodings/cp437.py", line
12, in encode

 return codecs.charmap_encode(input,errors,encoding_ma
p)
UnicodeEncodeError: 'charmap' codec can't encode
character '\xe3' in
position 1: character maps to <undefined>
>>> city.encode('cp437', errors='ignore')  ❹
b'So Paulo'
>>> city.encode('cp437', errors='replace')  ❺
b'S?o Paulo'
>>> city.encode('cp437', errors='xmlcharrefreplace')
❻
b'S&#227;o Paulo'
```



```
Z:\>chcp  ❶
Página de código ativa: 850
Z:\>python default_encodings.py  ❷
 locale.getpreferredencoding() -> 'cp1252'  ❸

type(my_file) -> <class '_io.TextIOWrapper'>
              my_file.encoding -> 'cp1252'  ❹
           sys.stdout.isatty() -> True      ❺
           sys.stdout.encoding -> 'cp850'   ❻
            sys.stdin.isatty() -> True
            sys.stdin.encoding -> 'cp850'
           sys.stderr.isatty() -> True
           sys.stderr.encoding -> 'cp850'
      sys.getdefaultencoding() -> 'utf-8'
   sys.getfilesystemencoding() -> 'mbcs'
❶
```

Examples: PythonHosted.org, Luciano Ramalho “Fluent Python”

Confusing Unicode



```
$ LC_ALL=C python
>>> # Note: if you're using a good terminal program when running in the C locale
>>> # The t                                                              chara
>>> # pytho                                                              ead:
>>> print u
Traceback (
  File "<st
UnicodeEnco                                                              posit
Traceback (
  File "<st
UnicodeEnco                                                              posit
                                                                         g="e
                                                                         //ex
```

```
>>> city = 'São Paulo'
>>> city.encode('utf_8')  ❶
b'S\xc3\xa3o Paulo'
>>> city.encode('utf_16')
b'\xff\xfeS\x00\xe3\x00o\x00 \x00P\x00a\x00u\x00l
\x00o\x00'
>>> city.encode('iso8859_1')  ❷
b'S\xe3o Paulo'
>>> city.encode('cp437')  ❸
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/.../lib/python3.4/encodings/cp437.py", line
12, in encode

 return codecs.charmap_encode(input,errors,encoding_ma
p)
UnicodeEncodeError: 'charmap' codec can't encode
character '\xe3' in
position 1: character maps to <undefined>
>>> city.encode('cp437', errors='ignore')  ❹
b'So Paulo'
>>> city.encode('cp437', errors='replace')  ❺
b'S?o Paulo'
>>> city.encode('cp437', errors='xmlcharrefreplace')
❻
b'S&#227;o Paulo'
```

```
>>> octets = b'Montr\xe9al'  ❶
>>> octets.decode('cp1252')  ❷
'Montréal'
>>> octets.decode('iso8859_7')  ❸
'Montrιal'
>>> octets.decode('koi8_r')  ❹
'MontrИal'
>>> octets.decode('utf_8')  ❺
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeDecodeError: 'utf-8' codec can't decode byte
0xe9 in position 5:
invalid continuation byte
>>> octets.decode('utf_8', errors='replace')  ❻
'Montr�al'
```

```
Z:\>chcp  ❶
Página de código ativa: 850
Z:\>python default_encodings.py  ❷
erredencoding() -> 'cp1252'  ❸

> <class '_io.TextIOWrapper'>
y_file.encoding -> 'cp1252'  ❹
stdout.isatty() -> True      ❺
stdout.encoding -> 'cp850'   ❻
.stdin.isatty() -> True
.stdin.encoding -> 'cp850'
stderr.isatty() -> True
stderr.encoding -> 'cp850'
faultencoding() -> 'utf-8'
ystemencoding() -> 'mbcs'
```

Examples: PythonHosted.org, Luciano Ramalho “Fluent Python”

Stay Calm and Unicode

Rule #1

Forget everything
you’ve learned
about pre-Unicode
characters and strings*

**including MBCS and UCS*

Ą

Unicode: The hero or villain?

Pawel Krawczyk

Ą

Character

Ą

Character

LATIN CAPITAL LETTER A WITH OGONEK

Ą

Character

<u>LATIN</u> CAPITAL **<u>LETTER</u>** A WITH OGONEK

Ą

Character

LATIN CAPITAL LETTER A WITH OGONEK

Code point

U+0104

Ą

Character

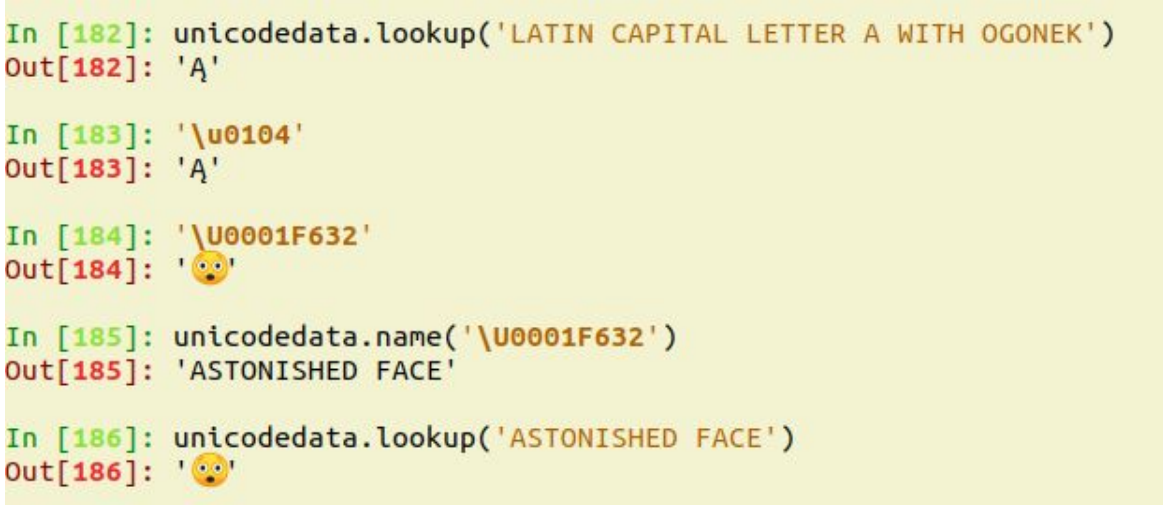
LATIN CAPITAL LETTER A WITH OGONEK

Code point

U+0104

“Character’s catalog number”

This is not encoding



```
In [182]: unicodedata.lookup('LATIN CAPITAL LETTER A WITH OGONEK')
Out[182]: 'Ą'

In [183]: '\u0104'
Out[183]: 'Ą'

In [184]: '\U0001F632'
Out[184]: '😲'

In [185]: unicodedata.name('\U0001F632')
Out[185]: 'ASTONISHED FACE'

In [186]: unicodedata.lookup('ASTONISHED FACE')
Out[186]: '😲'
```

Ą

Character

- LATIN CAPITAL LETTER A WITH OGONEK

Code point

- U+0104
- “Character’s catalog number”
- **This is not encoding**

Ą

Character

LATIN CAPITAL LETTER A WITH OGONEK

Code point

U+0104

“Character’s catalog number”

This is not encoding!!!

Ą

Character

LATIN CAPITAL LETTER A WITH OGONEK

Code point

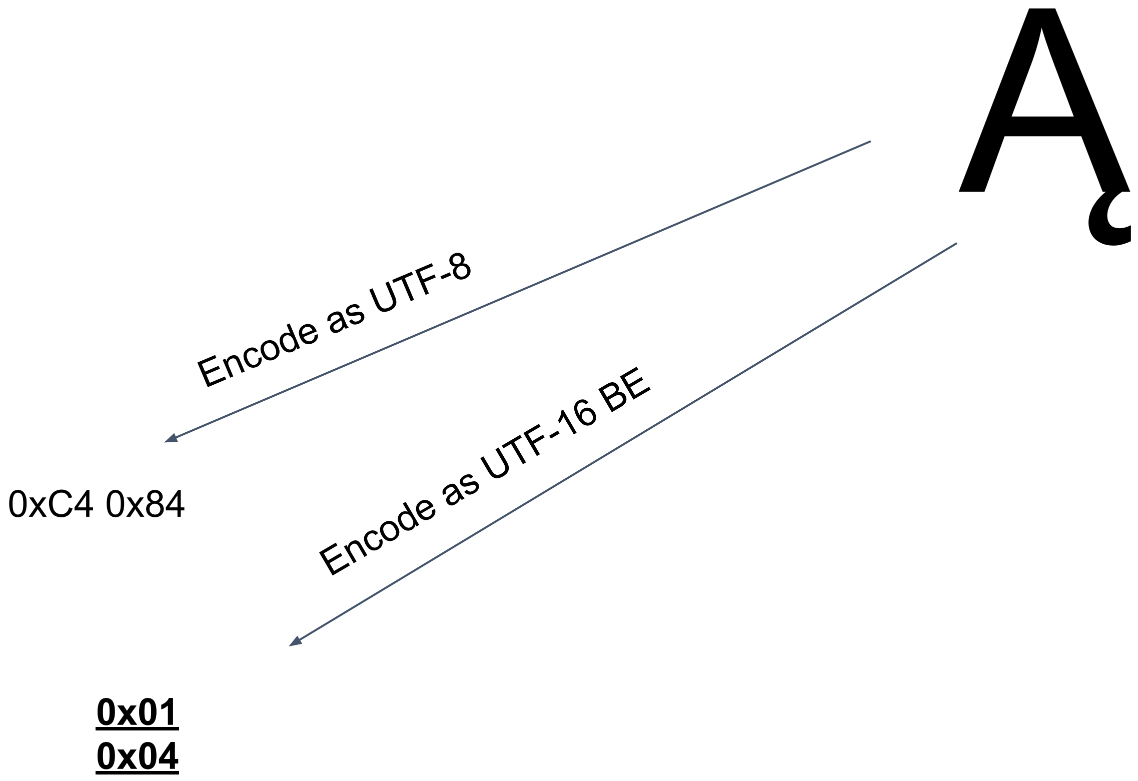
U+0104

“Character’s catalog number”

<u>This is encoding</u>

Encode as UTF-8

0xC4 0x84



Ą

Encode as UTF-8

0xC4 0x84

Encode as UTF-16 BE

0x01
0x04

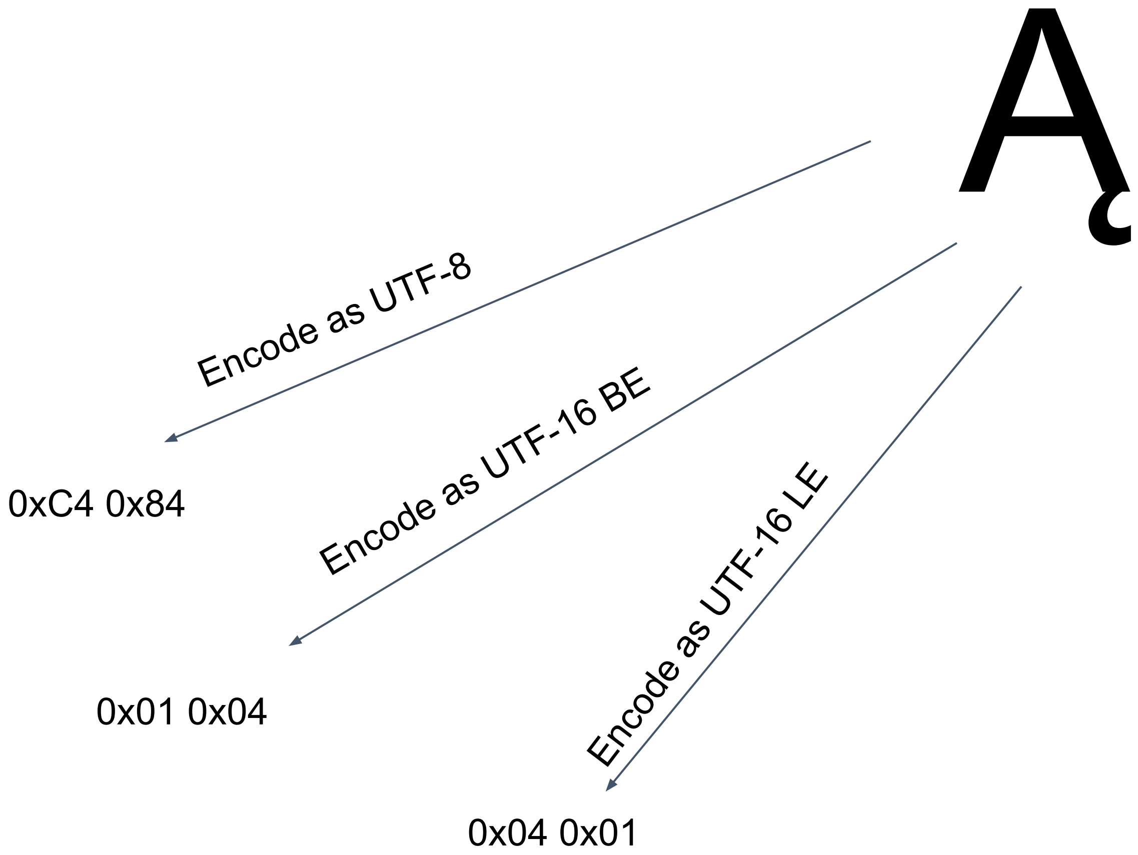
Character

LATIN CAPITAL LETTER A WITH OGONEK

Code point

U+**0104**

“Character’s catalog number”



Ą

Encode as UTF-8

0xC4 0x84

Encode as UTF-16 BE

0x01 0x04

Encode as UTF-16 LE

0x04 0x01

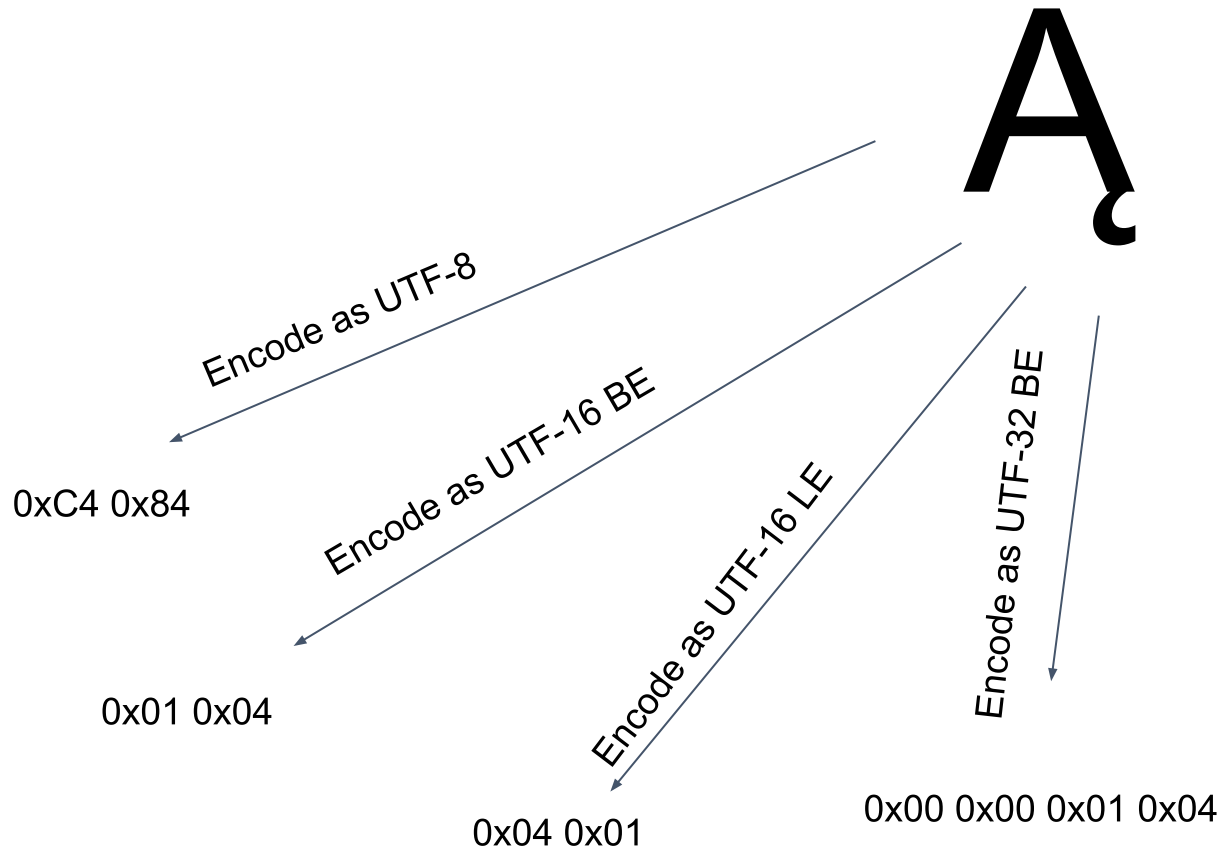
Character

LATIN CAPITAL LETTER A WITH OGONEK

Code point

U+0104

“Character’s catalog number”



Ą

Encode as UTF-8

0xC4 0x84

Encode as UTF-16 BE

0x01 0x04

Encode as UTF-16 LE

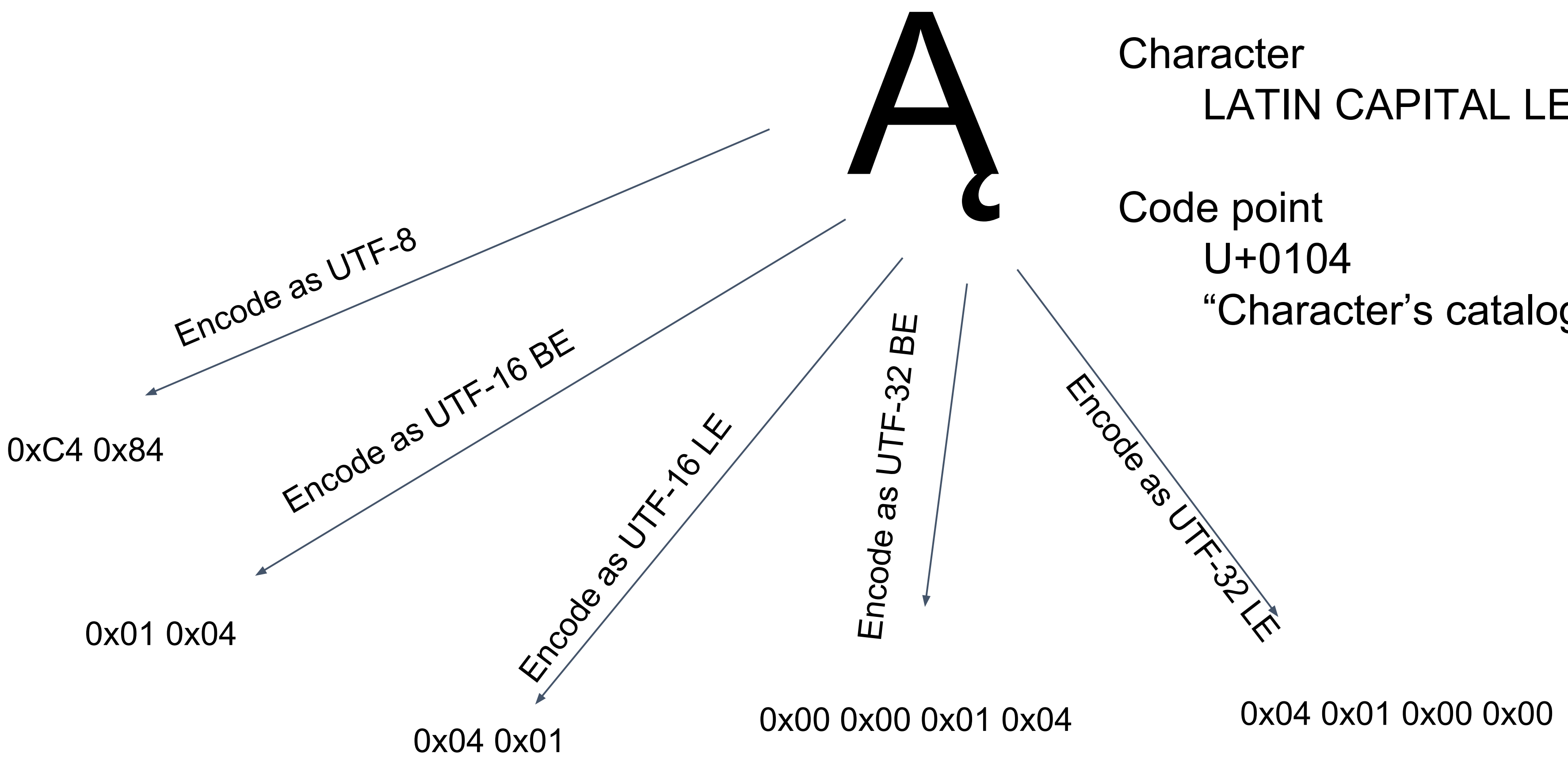
0x04 0x01

Encode as UTF-32 BE

0x00 0x00 0x01 0x04

Character
LATIN CAPITAL LETTER A WITH OGONEK

Code point
U+0104
“Character’s catalog number”



Ą

Encode as UTF-8
0xC4 0x84

Encode as UTF-16 BE
0x01 0x04

Encode as UTF-16 LE
0x04 0x01

Encode as UTF-32 BE
0x00 0x00 0x01 0x04

Encode as UTF-32 LE
0x04 0x01 0x00 0x00

Character
LATIN CAPITAL LETTER A WITH OGONEK

Code point
U+0104
“Character’s catalog number”

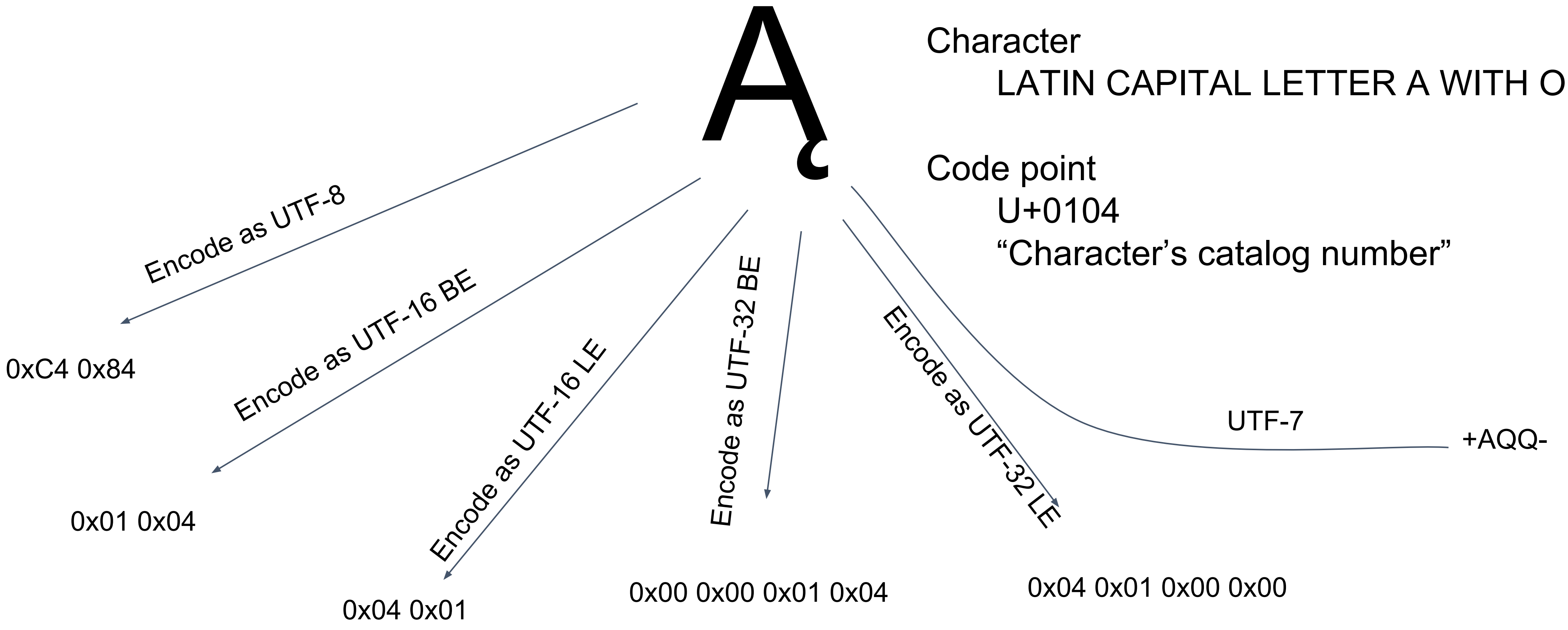
Character

LATIN CAPITAL LETTER A WITH OGONEK

Code point

U+0104

“Character’s catalog number”



Ą

Encode as UTF-8

0xC4 0x84

Encode as UTF-16 BE

0x01 0x04

Encode as UTF-16 LE

0x04 0x01

Encode as UTF-32 BE

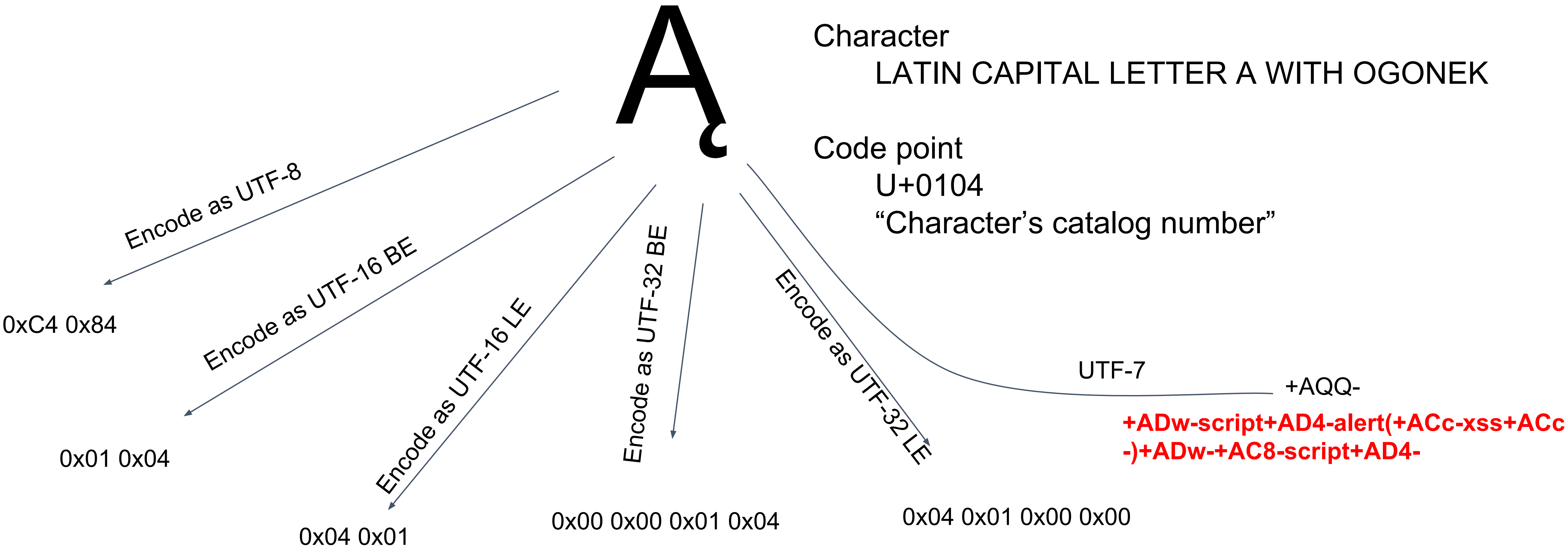
0x00 0x00 0x01 0x04

Encode as UTF-32 LE

0x04 0x01 0x00 0x00

UTF-7

+AQQ-



Ą

Character
LATIN CAPITAL LETTER A WITH OGONEK

Code point
U+0104
“Character’s catalog number”

Encode as UTF-8
0xC4 0x84

Encode as UTF-16 BE
0x01 0x04

Encode as UTF-16 LE
0x04 0x01

Encode as UTF-32 BE
0x00 0x00 0x01 0x04

Encode as UTF-32 LE
0x04 0x01 0x00 0x00

UTF-7
+AQQ-

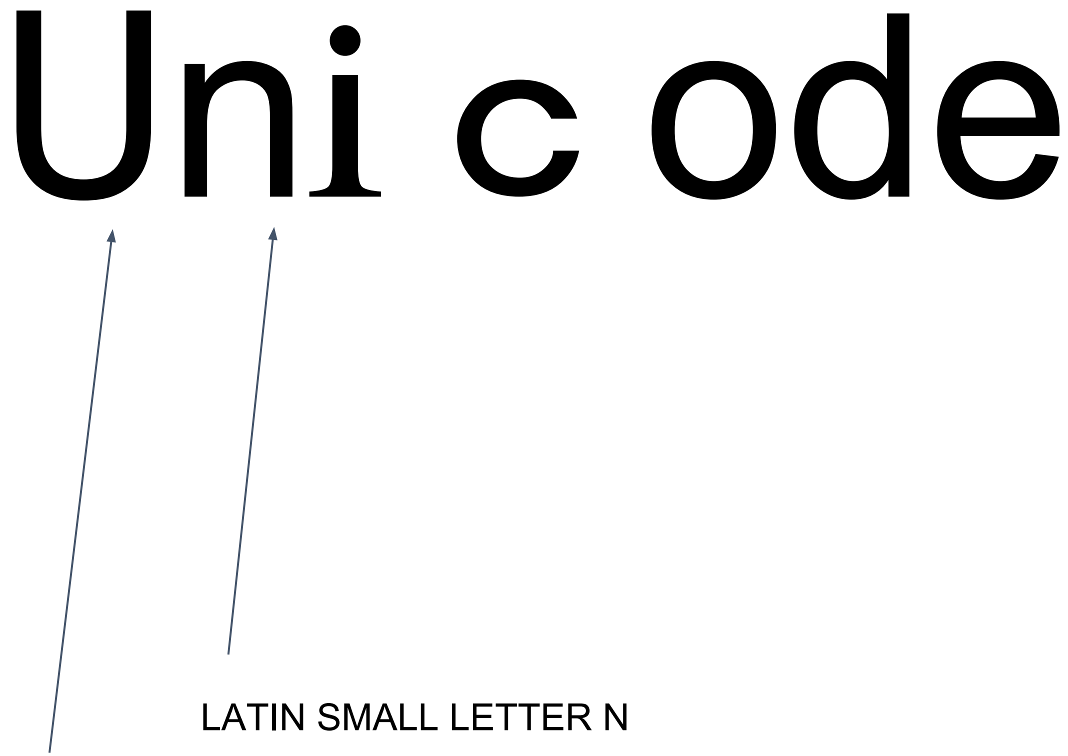
+ADw-script+AD4-alert(+ACc-xss+ACc-)+ADw-+AC8-script+AD4-

Homoglyphs

Unicode

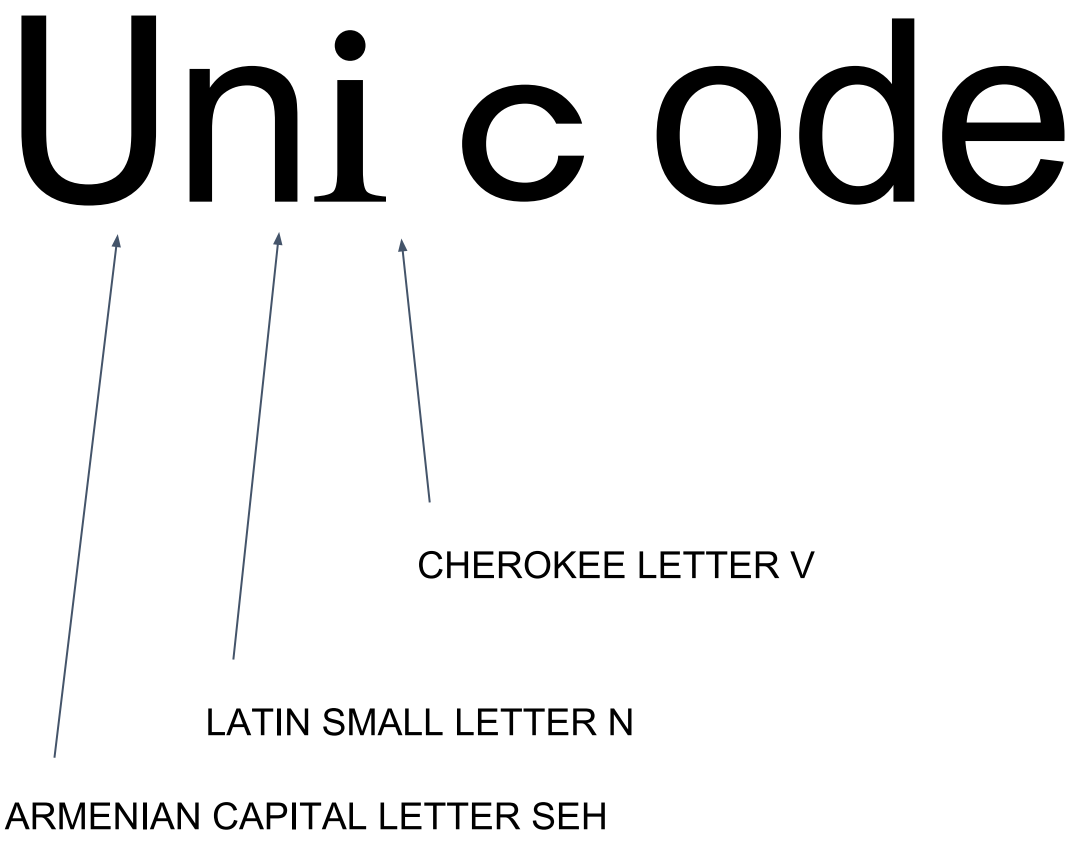
Սni c ode

ARMENIAN CAPITAL LETTER SEH



LATIN SMALL LETTER N

ARMENIAN CAPITAL LETTER SEH

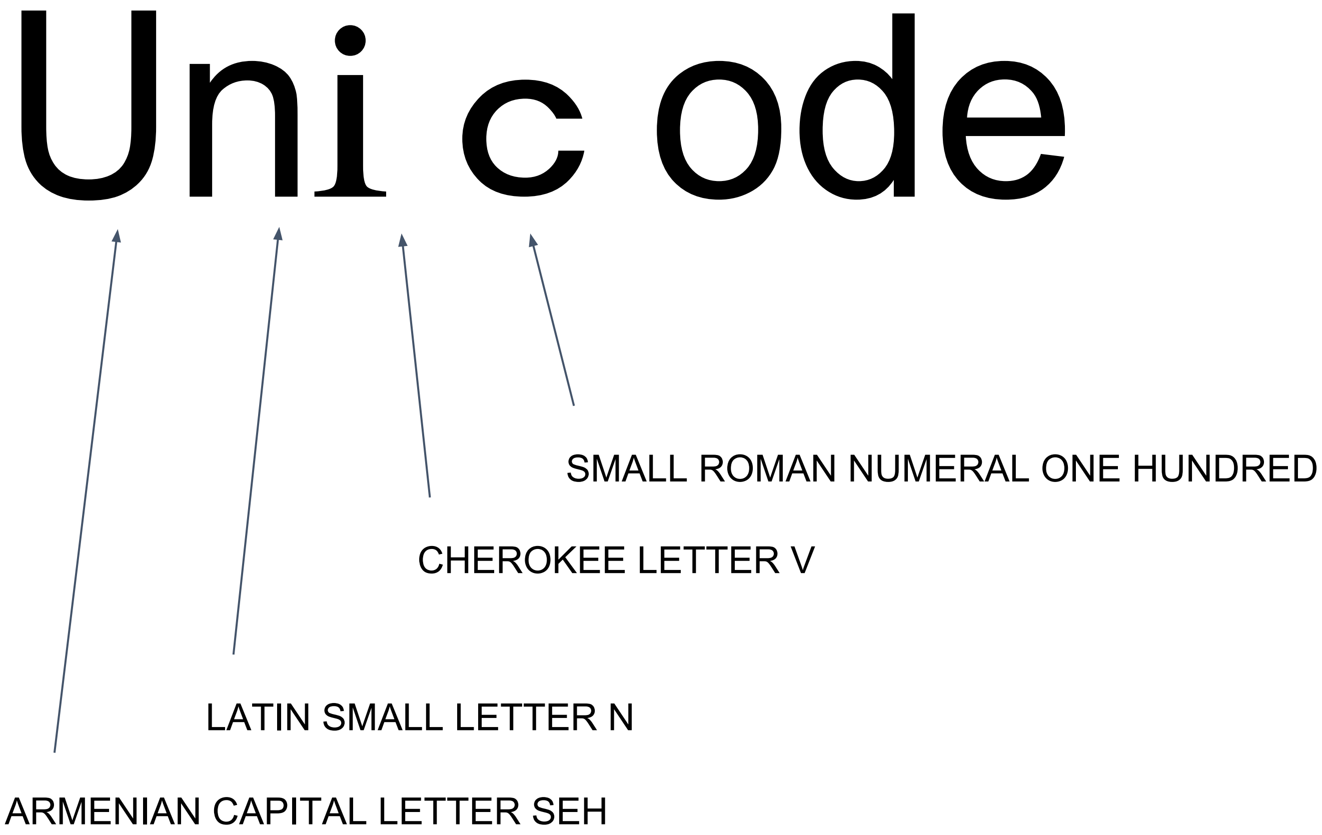


ՍnᎥcode

CHEROKEE LETTER V

LATIN SMALL LETTER N

ARMENIAN CAPITAL LETTER SEH



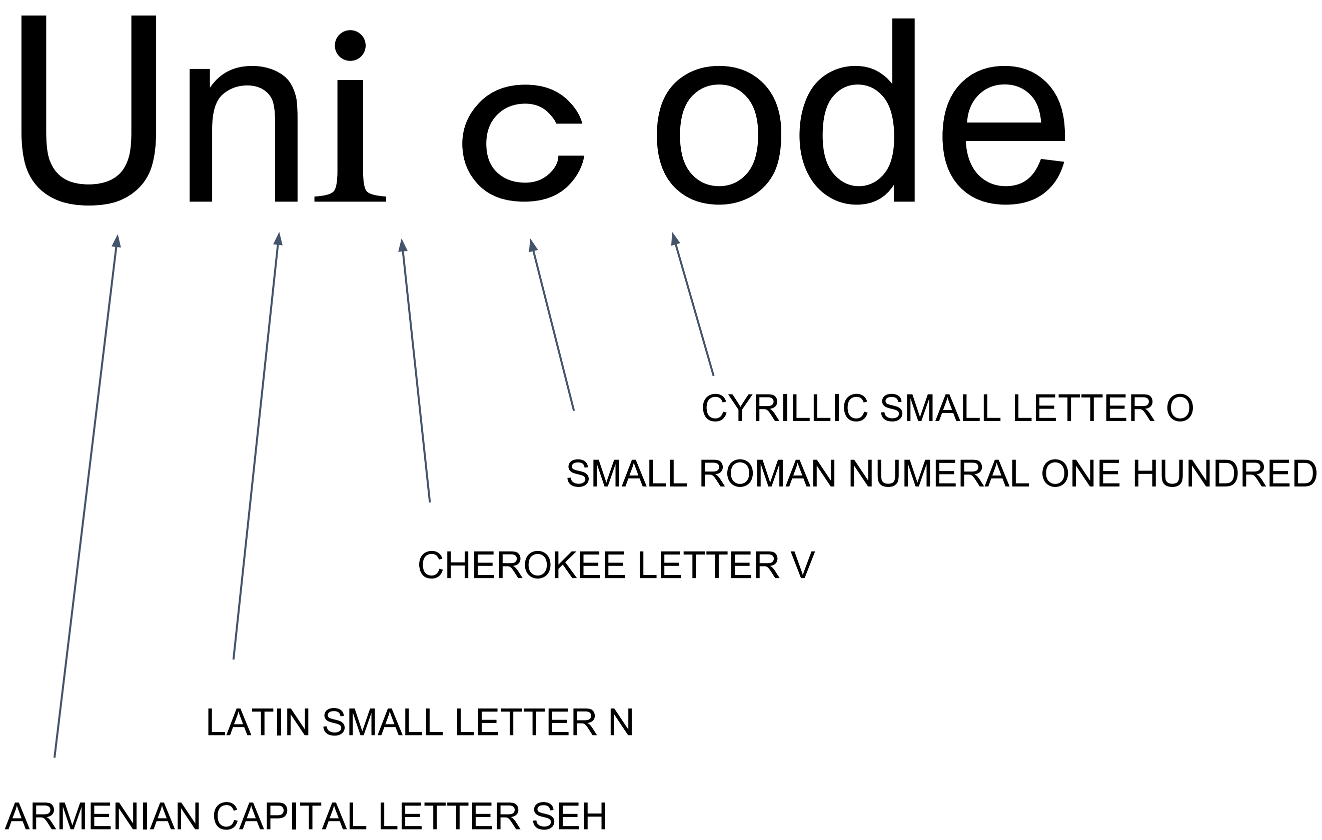
ՍnᎥ ⅽ ode

SMALL ROMAN NUMERAL ONE HUNDRED

CHEROKEE LETTER V

LATIN SMALL LETTER N

ARMENIAN CAPITAL LETTER SEH



ՍnᎥⅽоde

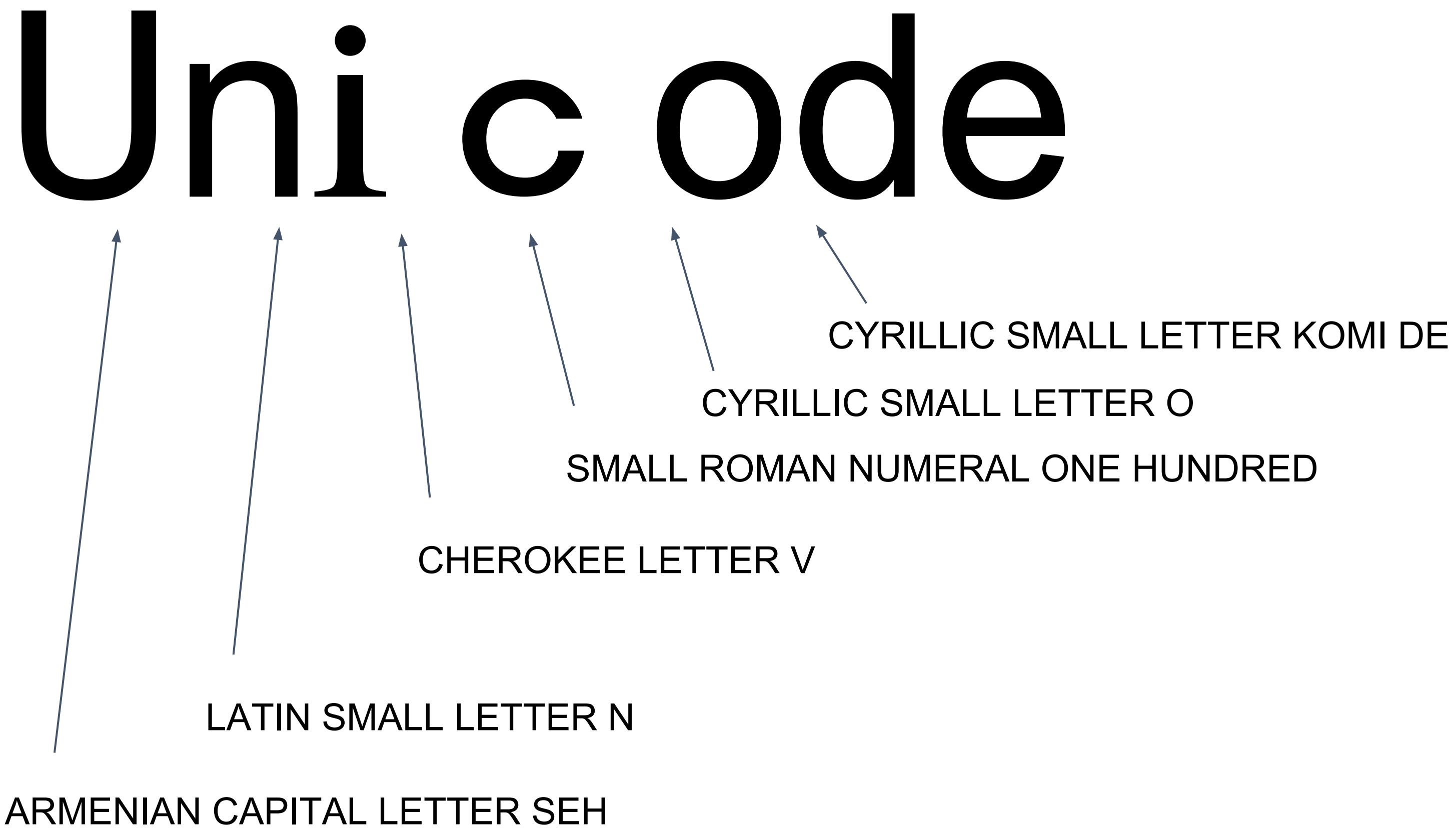
CYRILLIC SMALL LETTER O

SMALL ROMAN NUMERAL ONE HUNDRED

CHEROKEE LETTER V

LATIN SMALL LETTER N

ARMENIAN CAPITAL LETTER SEH



ՍnᎥⅽоԁe

CYRILLIC SMALL LETTER KOMI DE

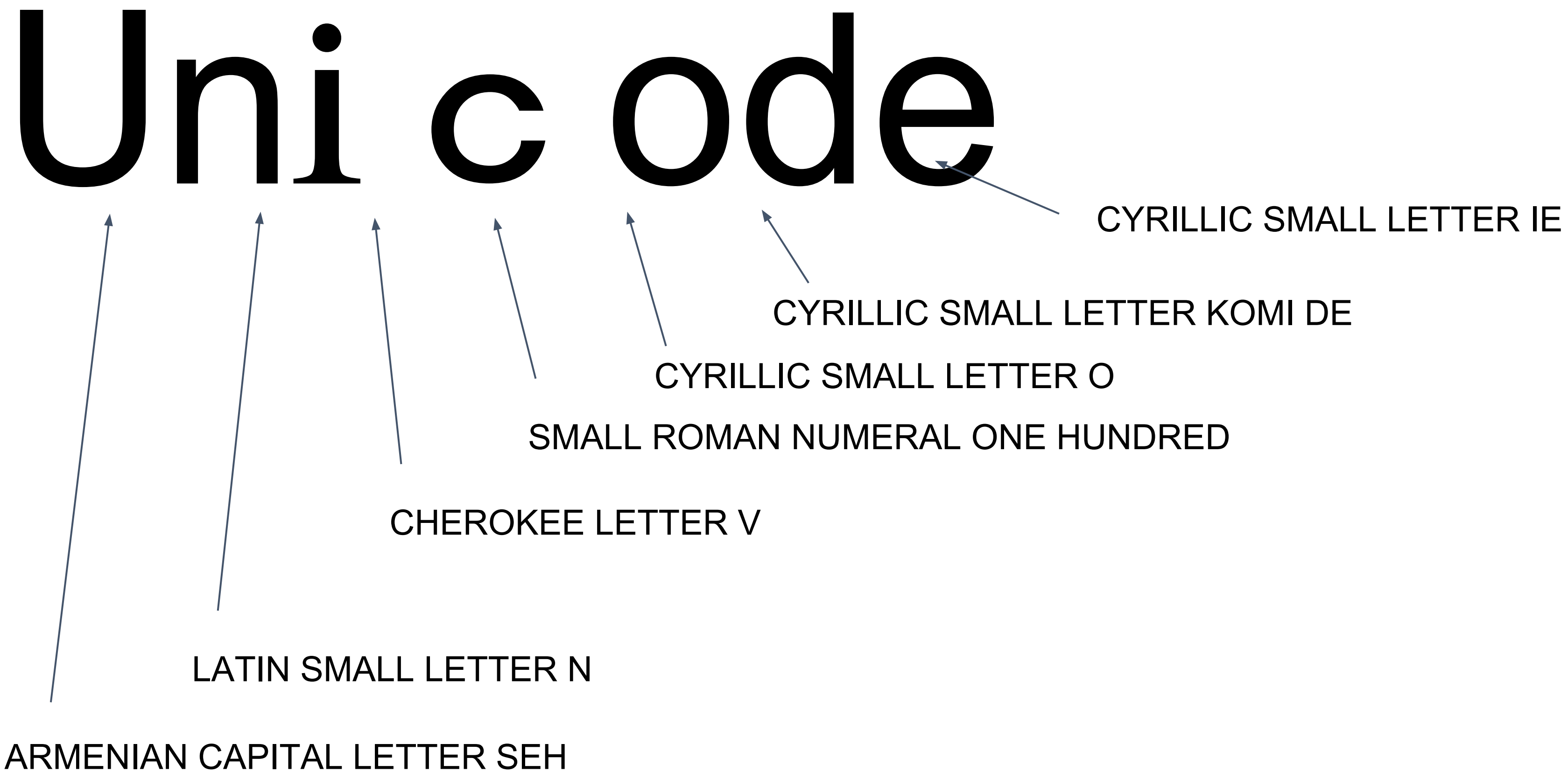
CYRILLIC SMALL LETTER O

SMALL ROMAN NUMERAL ONE HUNDRED

CHEROKEE LETTER V

LATIN SMALL LETTER N

ARMENIAN CAPITAL LETTER SEH



ՍnᎥⅽоԁе

CYRILLIC SMALL LETTER IE

CYRILLIC SMALL LETTER KOMI DE

CYRILLIC SMALL LETTER O

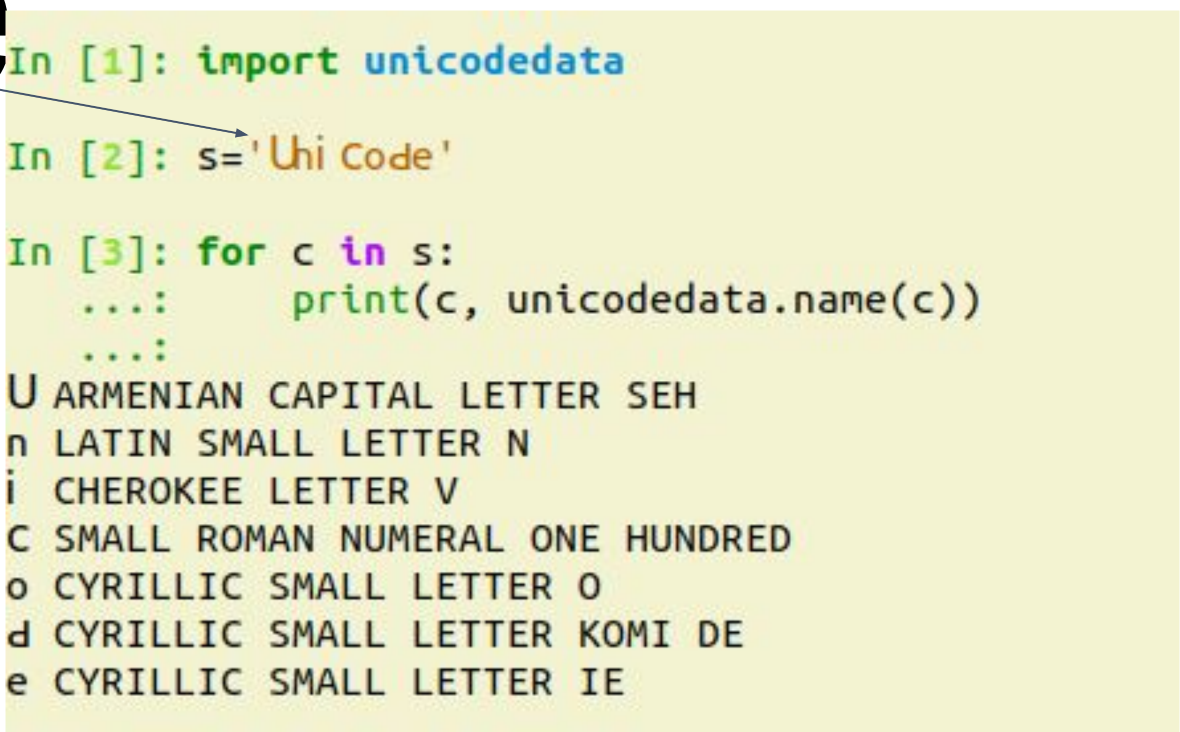
SMALL ROMAN NUMERAL ONE HUNDRED

CHEROKEE LETTER V

LATIN SMALL LETTER N

ARMENIAN CAPITAL LETTER SEH

ՍnᎥ ⅽ оԁе



```
In [1]: import unicodedata

In [2]: s='ՍnᎥⅽоԁе'

In [3]: for c in s:
   ...:     print(c, unicodedata.name(c))
   ...:
Ս ARMENIAN CAPITAL LETTER SEH
n LATIN SMALL LETTER N
Ꭵ CHEROKEE LETTER V
ⅽ SMALL ROMAN NUMERAL ONE HUNDRED
о CYRILLIC SMALL LETTER O
ԁ CYRILLIC SMALL LETTER KOMI DE
е CYRILLIC SMALL LETTER IE
```

Surviving in the Unicode world

Rule #2

Inside your application

think <u>text</u> composed of <u>characters</u>;

forget about bytes

Rule #3

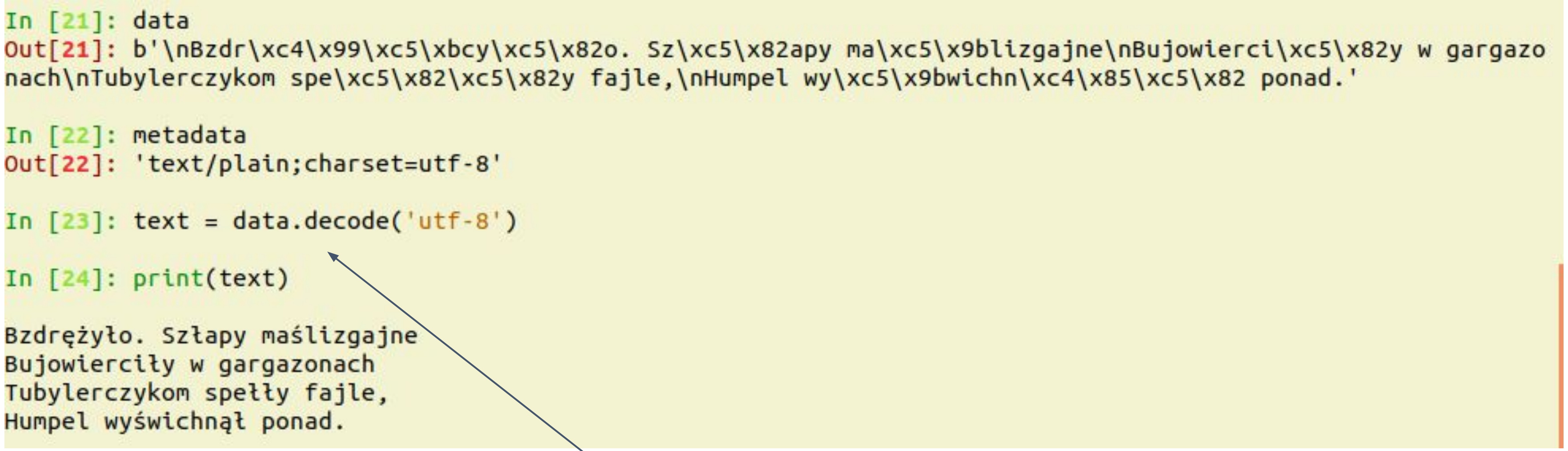
Decode bytes into text

on input

Encode text into bytes

on output

Input decoding example



```
In [21]: data
Out[21]: b'\nBzdr\xc4\x99\xc5\xbcy\xc5\x82o. Sz\xc5\x82apy ma\xc5\x9blizgajne\nBujowierci\xc5\x82y w gargazo
nach\nTubylerczykom spe\xc5\x82\xc5\x82y fajle,\nHumpel wy\xc5\x9bwichn\xc4\x85\xc5\x82 ponad.'

In [22]: metadata
Out[22]: 'text/plain;charset=utf-8'

In [23]: text = data.decode('utf-8')

In [24]: print(text)

Bzdrężyło. Szłapy maślizgajne
Bujowierciły w gargazonach
Tubylerczykom spełły fajle,
Humpel wyświchnął ponad.
```

Transport data

Transport metadata

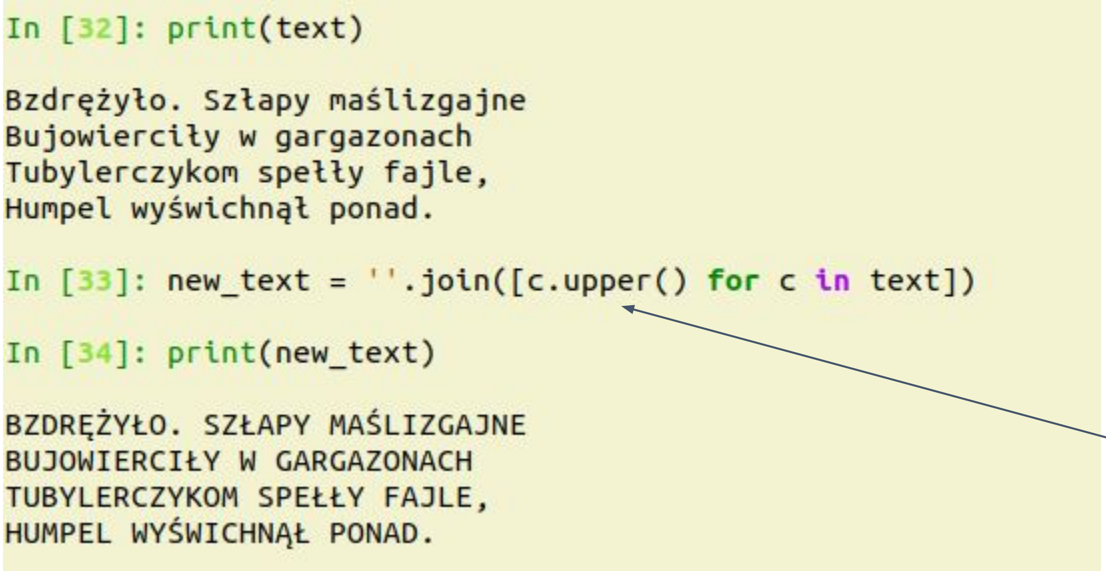
Decoding

Internal representation

Exceptions!

Text: Jolanta Kozak “Dziaberlak” (“Jabberwocky” by Lewis Carroll)

Text processing, persistence* and fun

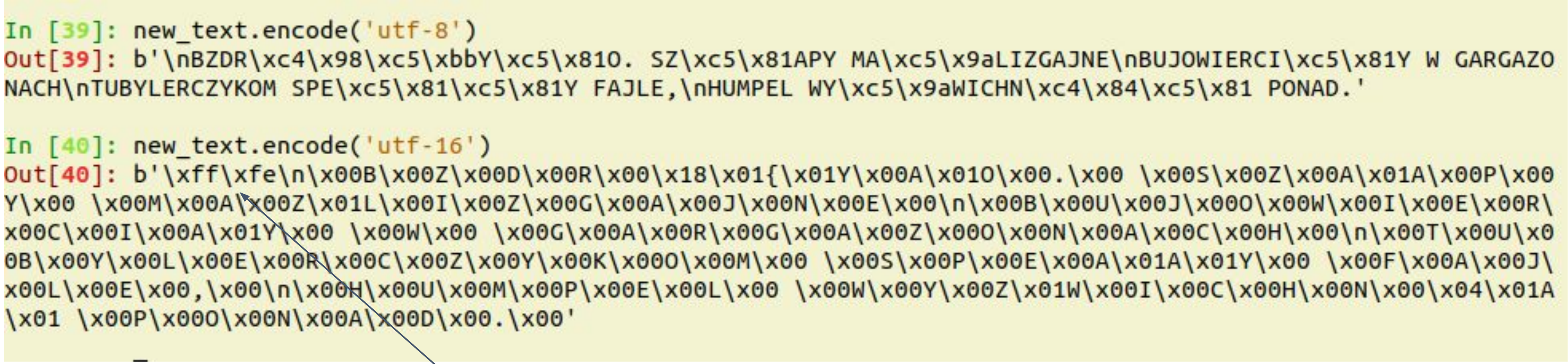


```
In [32]: print(text)

Bzdrężyło. Szłapy maślizgajne
Bujowierciły w gargazonach
Tubylerczykom spełły fajle,
Humpel wyświchnął ponad.

In [33]: new_text = ''.join([c.upper() for c in text])

In [34]: print(new_text)

BZDRĘŻYŁO. SZŁAPY MAŚLIZGAJNE
BUJOWIERCIŁY W GARGAZONACH
TUBYLERCZYKOM SPEŁŁY FAJLE,
HUMPEL WYŚWICHNĄŁ PONAD.
```

Example text processing

** do not persist before watching this presentation till the end*

Output encoding

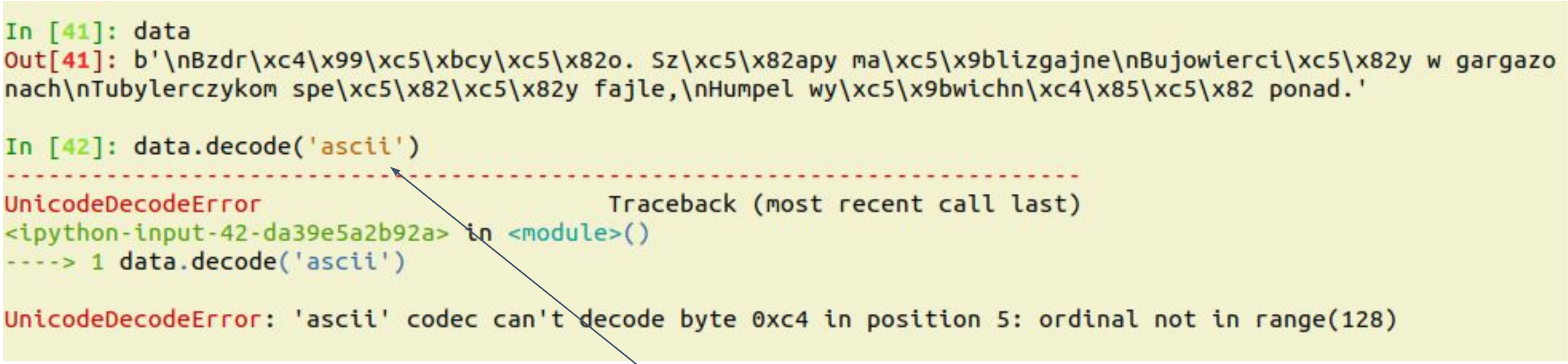


```
In [39]: new_text.encode('utf-8')
Out[39]: b'\nBZDR\xc4\x98\xc5\xbbY\xc5\x81O. SZ\xc5\x81APY MA\xc5\x9aLIZGAJNE\nBUJOWIERCI\xc5\x81Y W GARGAZO
NACH\nTUBYLERCZYKOM SPE\xc5\x81\xc5\x81Y FAJLE,\nHUMPEL WY\xc5\x9aWICHN\xc4\x84\xc5\x81 PONAD.'

In [40]: new_text.encode('utf-16')
Out[40]: b'\xff\xfe\n\x00B\x00Z\x00D\x00R\x00\x18\x01{\x01Y\x00A\x01O\x00.\x00 \x00S\x00Z\x00A\x01A\x00P\x00
Y\x00 \x00M\x00A\x00Z\x01L\x00I\x00Z\x00G\x00A\x00J\x00N\x00E\x00\n\x00B\x00U\x00J\x00O\x00W\x00I\x00E\x00R\
x00C\x00I\x00A\x01Y\x00 \x00W\x00 \x00G\x00A\x00R\x00G\x00A\x00Z\x00O\x00N\x00A\x00C\x00H\x00\n\x00T\x00U\x0
0B\x00Y\x00L\x00E\x00R\x00C\x00Z\x00Y\x00K\x00O\x00M\x00 \x00S\x00P\x00E\x00A\x01A\x01Y\x00 \x00F\x00A\x00J\
x00L\x00E\x00,\x00\n\x00H\x00U\x00M\x00P\x00E\x00L\x00 \x00W\x00Y\x00Z\x01W\x00I\x00C\x00H\x00N\x00\x04\x01A
\x01 \x00P\x00O\x00N\x00A\x00D\x00.\x00'
```

U+FEFF BYTE ORDER MARK (BOM)
“Unicode signature”

When things go south

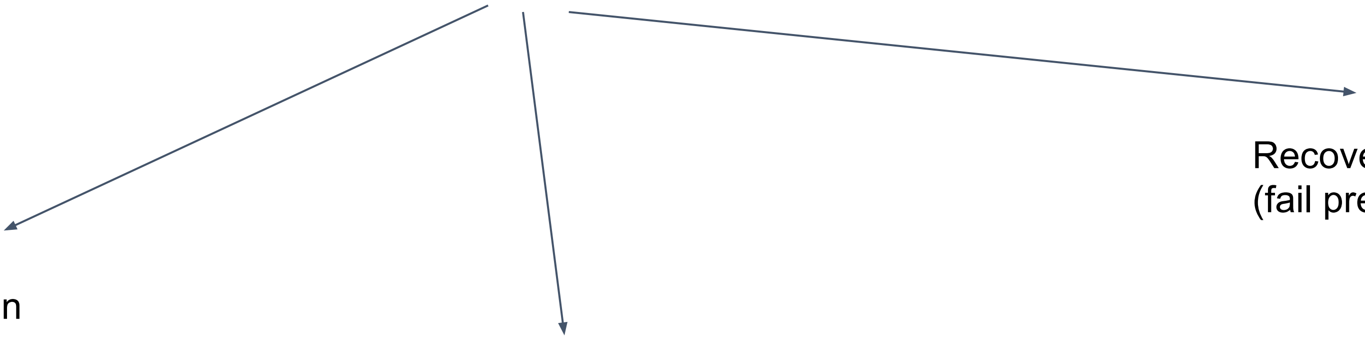
Wrong decoder



```
In [41]: data
Out[41]: b'\nBzdr\xc4\x99\xc5\xbcy\xc5\x82o. Sz\xc5\x82apy ma\xc5\x9blizgajne\nBujowierci\xc5\x82y w gargazo
nach\nTubylerczykom spe\xc5\x82\xc5\x82y fajle,\nHumpel wy\xc5\x9bwichn\xc4\x85\xc5\x82 ponad.'

In [42]: data.decode('ascii')
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
<ipython-input-42-da39e5a2b92a> in <module>()
----> 1 data.decode('ascii')

UnicodeDecodeError: 'ascii' codec can't decode byte 0xc4 in position 5: ordinal not in range(128)
```

- Client told us so
- Client told us nothing, so we assumed so
- Client told us ‘utf-8’, but we ignored it and assumed ‘ascii’ because we have been writing this software since 1986

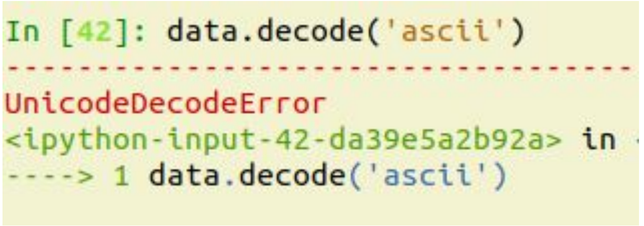
What to do - a policy decision!



Reject incorrect information
(fail closed)

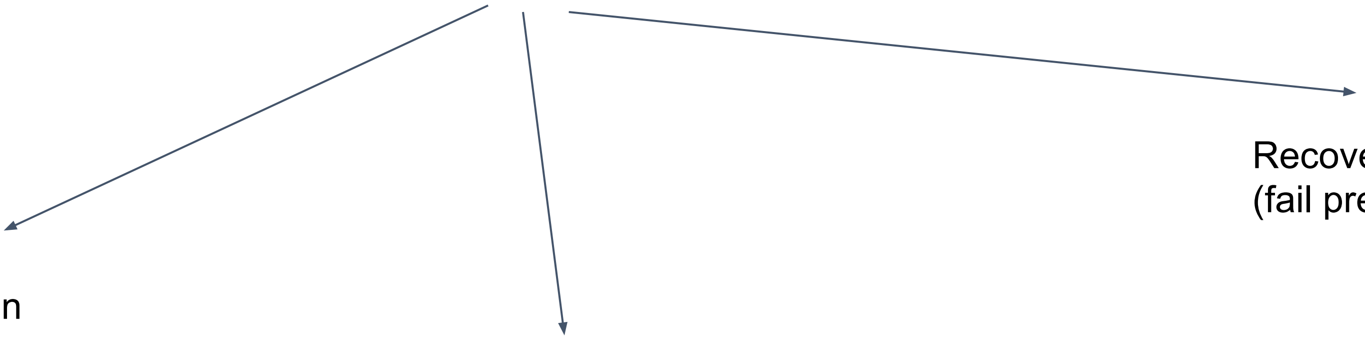
Partially lose information
(fail open)

Recover information
(fail pretending it’s fine)



```
In [42]: data.decode('ascii')
-----------------------------------
UnicodeDecodeError
<ipython-input-42-da39e5a2b92a> in
----> 1 data.decode('ascii')
```

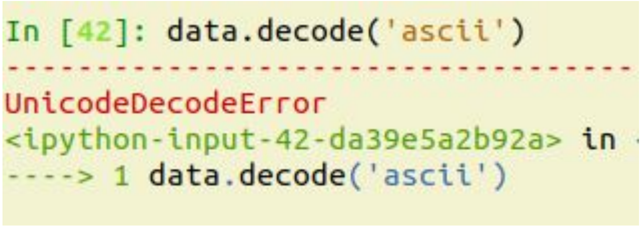
Policy decision



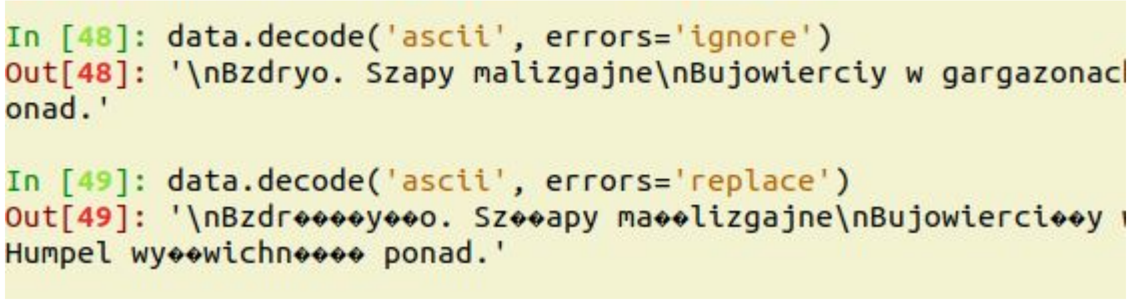
Reject incorrect information
(fail closed)

Partially lose information
(fail open)

Recover information
(fail pretending it’s fine)



```
In [42]: data.decode('ascii')
-------------------------------------
UnicodeDecodeError
<ipython-input-42-da39e5a2b92a> in
----> 1 data.decode('ascii')
```

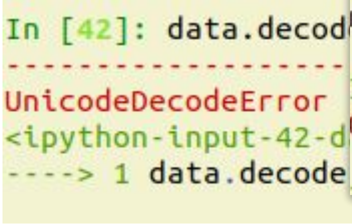


```
In [48]: data.decode('ascii', errors='ignore')
Out[48]: '\nBzdryo. Szapy malizgajne\nBujowierciy w gargazonac
onad.'

In [49]: data.decode('ascii', errors='replace')
Out[49]: '\nBzdr����y��o. Sz��apy ma��lizgajne\nBujowierci��y
Humpel wy��wichn���� ponad.'
```

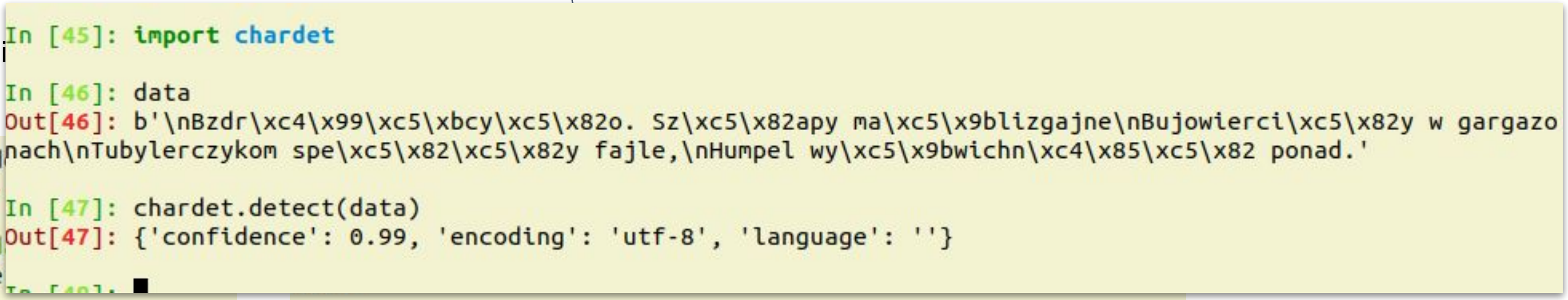
Policy decision

Recover information
(fail pretending it's fine)

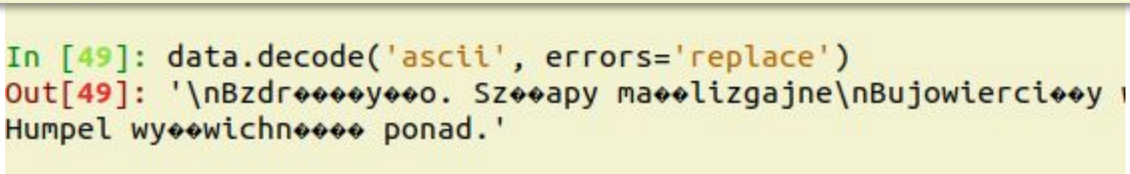
Reject incorrect i
(fail closed)



```
In [42]: data.decod
--------------------
UnicodeDecodeError
<ipython-input-42-d
----> 1 data.decode
```



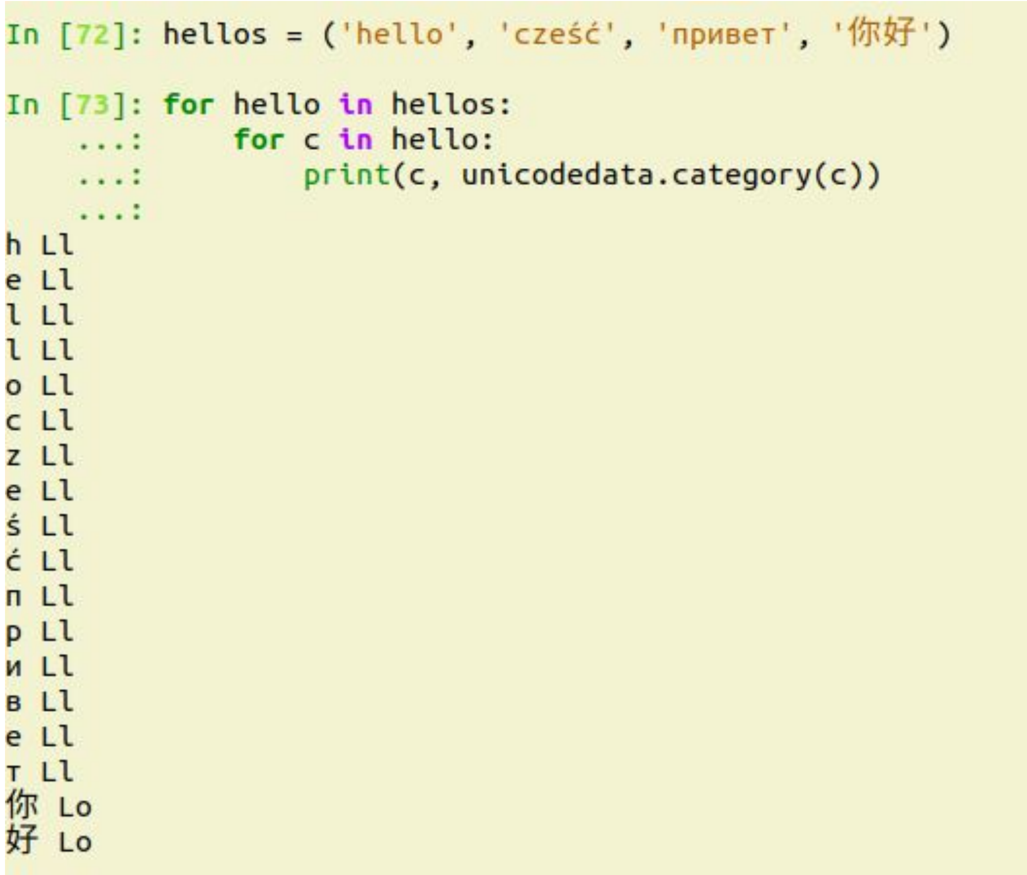
```
In [45]: import chardet

In [46]: data
Out[46]: b'\nBzdr\xc4\x99\xc5\xbcy\xc5\x82o. Sz\xc5\x82apy ma\xc5\x9blizgajne\nBujowierci\xc5\x82y w gargazo
nach\nTubylerczykom spe\xc5\x82\xc5\x82y fajle,\nHumpel wy\xc5\x9bwichn\xc4\x85\xc5\x82 ponad.'

In [47]: chardet.detect(data)
Out[47]: {'confidence': 0.99, 'encoding': 'utf-8', 'language': ''}

In [48]: █
```



```
In [49]: data.decode('ascii', errors='replace')
Out[49]: '\nBzdr����y��o. Sz��apy ma��lizgajne\nBujowierci��y w
Humpel wy��wichn���� ponad.'
```

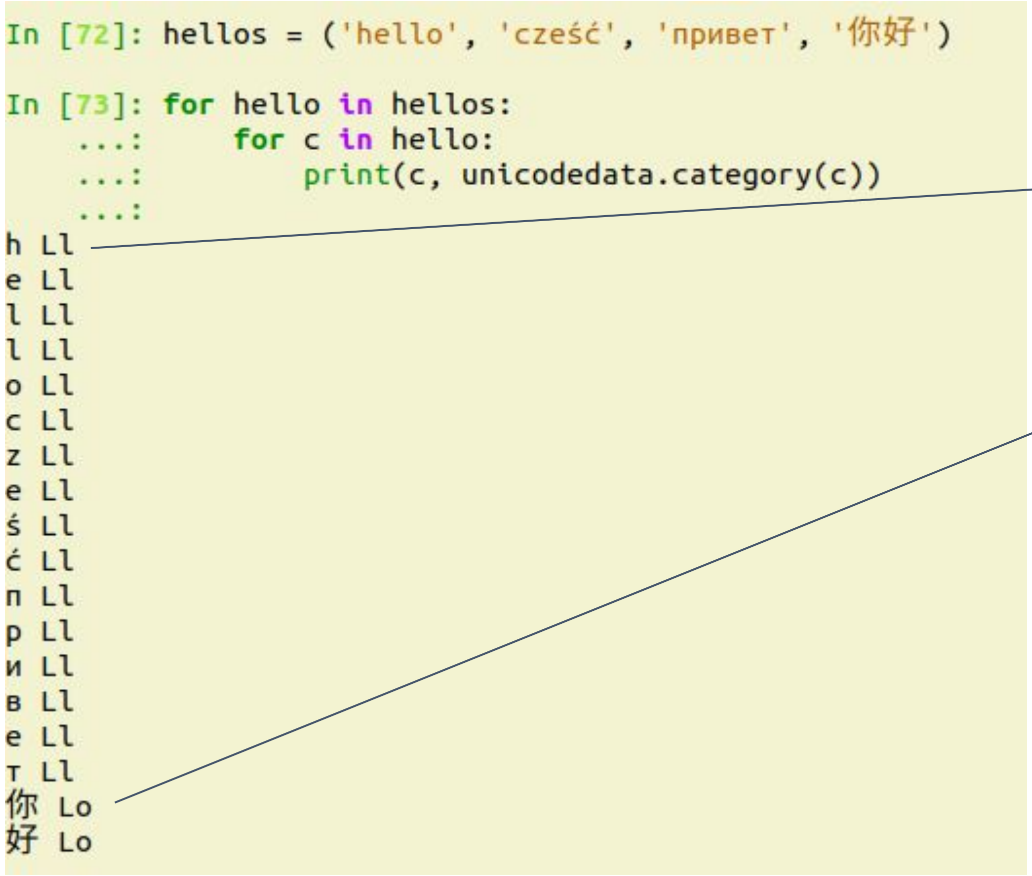
Validation techniques

Character category enforcement



```
In [72]: hellos = ('hello', 'cześć', 'привет', '你好')

In [73]: for hello in hellos:
    ...:     for c in hello:
    ...:         print(c, unicodedata.category(c))
    ...:
h Ll
e Ll
l Ll
l Ll
o Ll
c Ll
z Ll
e Ll
ś Ll
ć Ll
п Ll
р Ll
и Ll
в Ll
е Ll
т Ll
你 Lo
好 Lo
```

Character category enforcement



```
In [72]: hellos = ('hello', 'cześć', 'привет', '你好')

In [73]: for hello in hellos:
    ...:     for c in hello:
    ...:         print(c, unicodedata.category(c))
    ...:
h Ll
e Ll
l Ll
l Ll
o Ll
c Ll
z Ll
e Ll
ś Ll
ć Ll
п Ll
р Ll
и Ll
в Ll
е Ll
т Ll
你 Lo
好 Lo
```

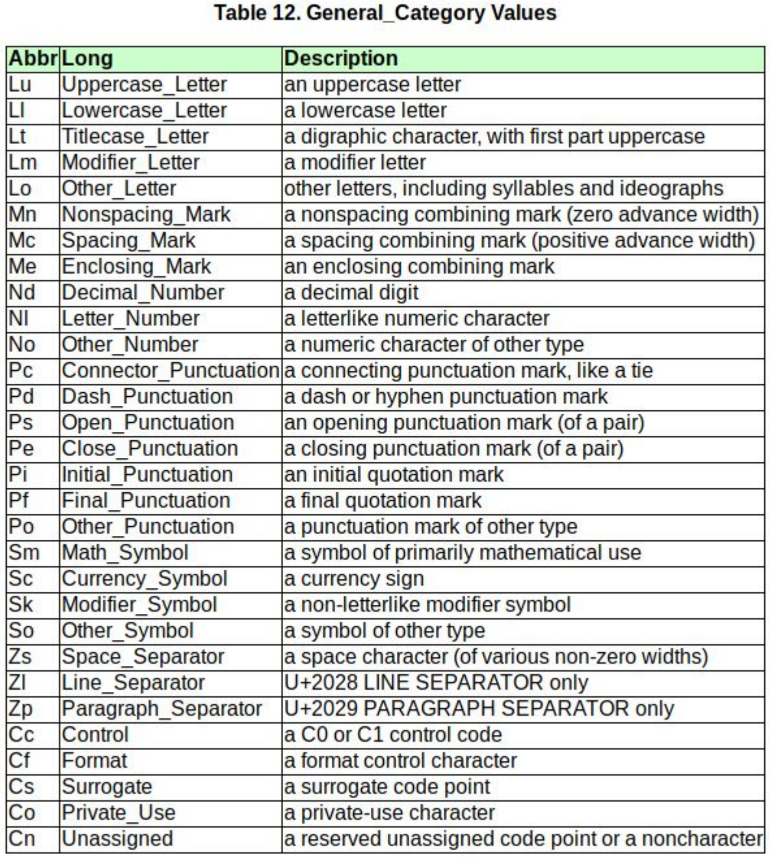


Table 12. General_Category Values

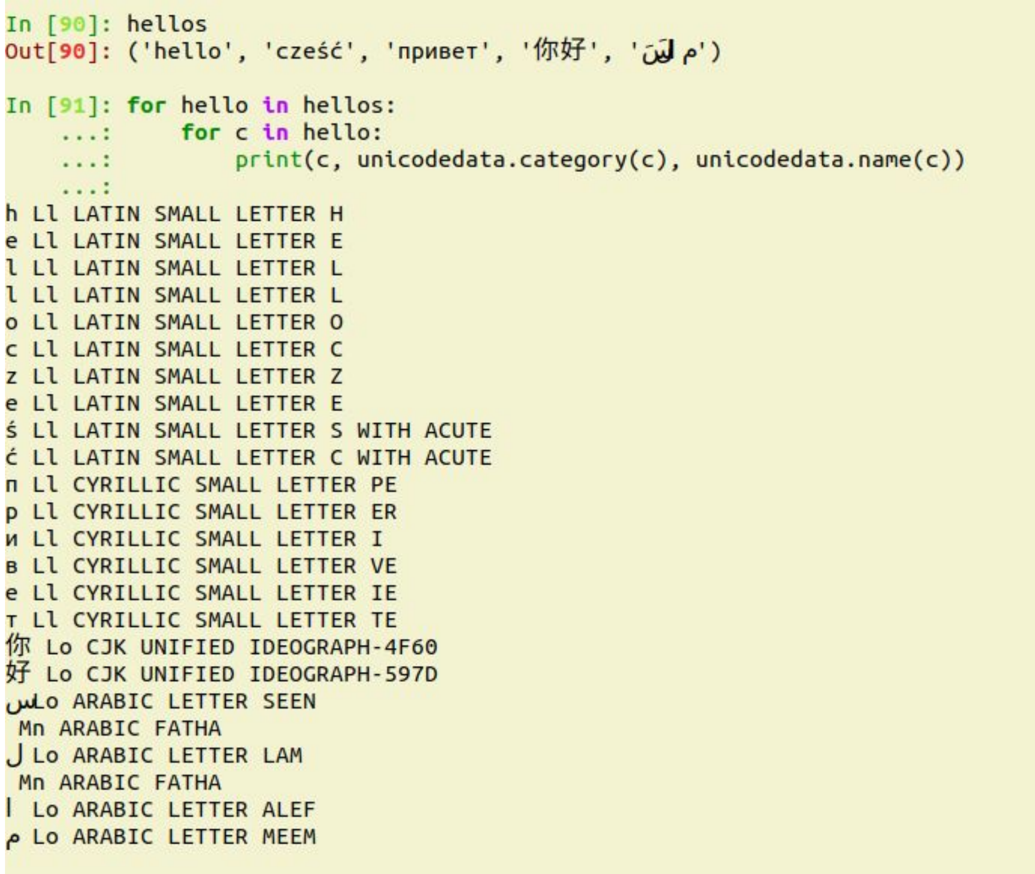
Abbr	Long	Description
Lu	Uppercase_Letter	an uppercase letter
Ll	Lowercase_Letter	a lowercase letter
Lt	Titlecase_Letter	a digraphic character, with first part uppercase
Lm	Modifier_Letter	a modifier letter
Lo	Other_Letter	other letters, including syllables and ideographs
Mn	Nonspacing_Mark	a nonspacing combining mark (zero advance width)
Mc	Spacing_Mark	a spacing combining mark (positive advance width)
Me	Enclosing_Mark	an enclosing combining mark
Nd	Decimal_Number	a decimal digit
Nl	Letter_Number	a letterlike numeric character
No	Other_Number	a numeric character of other type
Pc	Connector_Punctuation	a connecting punctuation mark, like a tie
Pd	Dash_Punctuation	a dash or hyphen punctuation mark
Ps	Open_Punctuation	an opening punctuation mark (of a pair)
Pe	Close_Punctuation	a closing punctuation mark (of a pair)
Pi	Initial_Punctuation	an initial quotation mark
Pf	Final_Punctuation	a final quotation mark
Po	Other_Punctuation	a punctuation mark of other type
Sm	Math_Symbol	a symbol of primarily mathematical use
Sc	Currency_Symbol	a currency sign
Sk	Modifier_Symbol	a non-letterlike modifier symbol
So	Other_Symbol	a symbol of other type
Zs	Space_Separator	a space character (of various non-zero widths)
Zl	Line_Separator	U+2028 LINE SEPARATOR only
Zp	Paragraph_Separator	U+2029 PARAGRAPH SEPARATOR only
Cc	Control	a C0 or C1 control code
Cf	Format	a format control character
Cs	Surrogate	a surrogate code point
Co	Private_Use	a private-use character
Cn	Unassigned	a reserved unassigned code point or a noncharacter

Source: Unicode Standard Annex #44

Rule #5

Enforce Unicode categories

Character script enforcement



```
In [90]: hellos
Out[90]: ('hello', 'cześć', 'привет', '你好', 'سَلَام')

In [91]: for hello in hellos:
    ...:     for c in hello:
    ...:         print(c, unicodedata.category(c), unicodedata.name(c))
    ...:
h Ll LATIN SMALL LETTER H
e Ll LATIN SMALL LETTER E
l Ll LATIN SMALL LETTER L
l Ll LATIN SMALL LETTER L
o Ll LATIN SMALL LETTER O
c Ll LATIN SMALL LETTER C
z Ll LATIN SMALL LETTER Z
e Ll LATIN SMALL LETTER E
ś Ll LATIN SMALL LETTER S WITH ACUTE
ć Ll LATIN SMALL LETTER C WITH ACUTE
п Ll CYRILLIC SMALL LETTER PE
р Ll CYRILLIC SMALL LETTER ER
и Ll CYRILLIC SMALL LETTER I
в Ll CYRILLIC SMALL LETTER VE
е Ll CYRILLIC SMALL LETTER IE
т Ll CYRILLIC SMALL LETTER TE
你 Lo CJK UNIFIED IDEOGRAPH-4F60
好 Lo CJK UNIFIED IDEOGRAPH-597D
س Lo ARABIC LETTER SEEN
 َ Mn ARABIC FATHA
ل Lo ARABIC LETTER LAM
 َ Mn ARABIC FATHA
ا Lo ARABIC LETTER ALEF
م Lo ARABIC LETTER MEEM
```

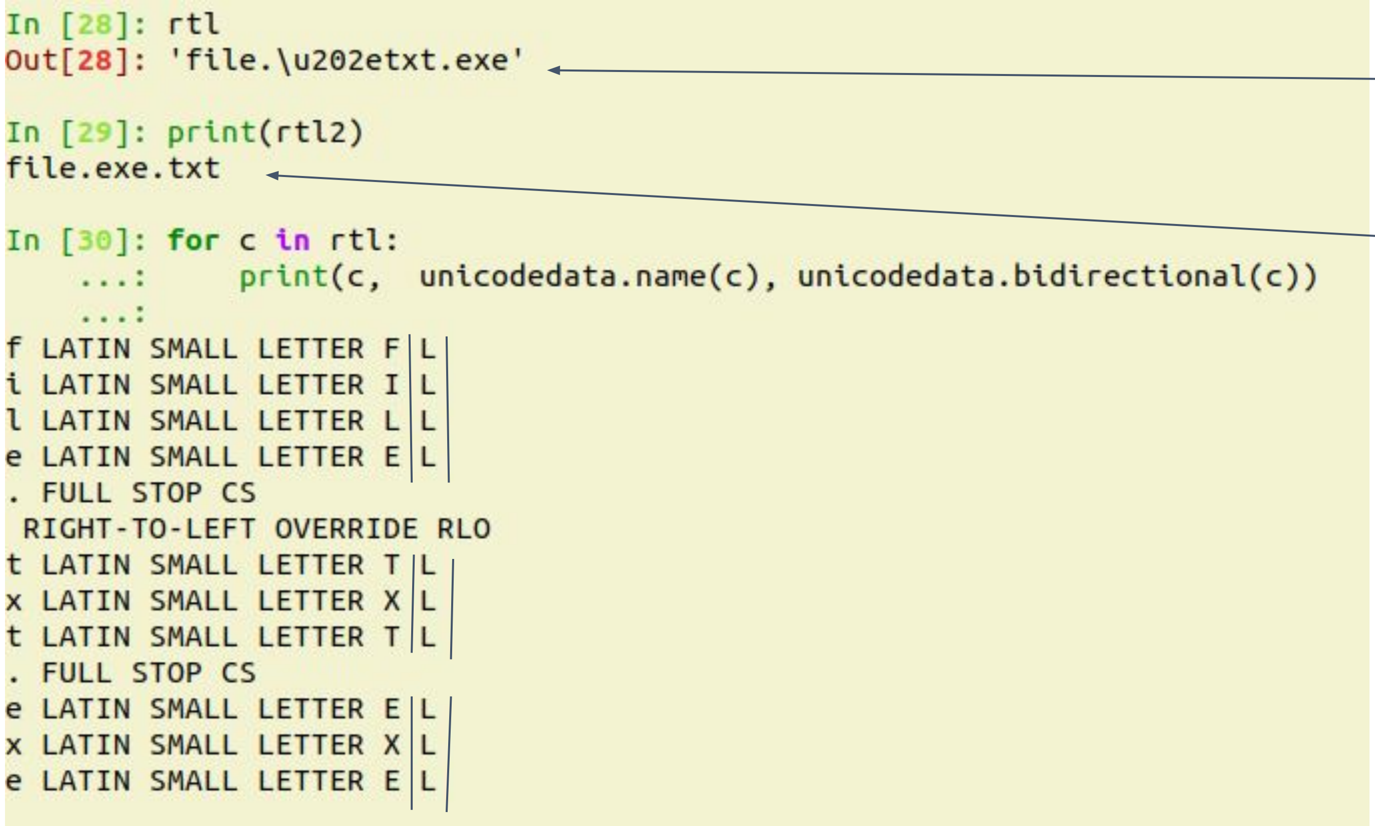
Scripts used in the example:

- LATIN
- CYRILLIC
- CJK
- ARABIC

Rule #6

Enforce Unicode scripts

Text direction enforcement

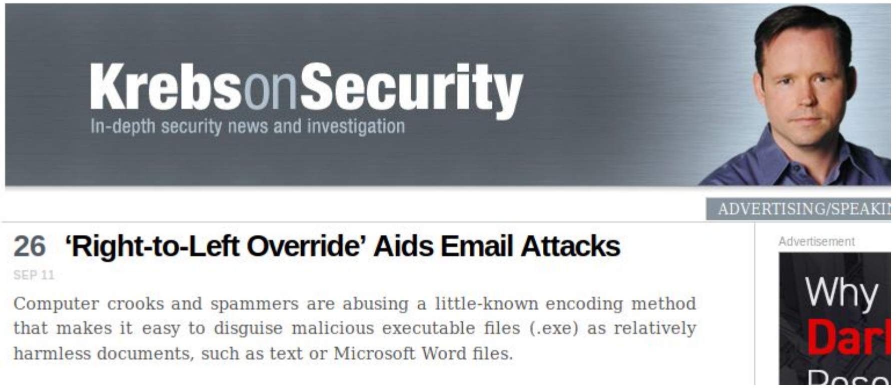


```
In [28]: rtl
Out[28]: 'file.\u202etxt.exe'

In [29]: print(rtl2)
file.exe.txt

In [30]: for c in rtl:
    ...:     print(c,  unicodedata.name(c), unicodedata.bidirectional(c))
    ...:
f LATIN SMALL LETTER F L
i LATIN SMALL LETTER I L
l LATIN SMALL LETTER L L
e LATIN SMALL LETTER E L
. FULL STOP CS
 RIGHT-TO-LEFT OVERRIDE RLO
t LATIN SMALL LETTER T L
x LATIN SMALL LETTER X L
t LATIN SMALL LETTER T L
. FULL STOP CS
e LATIN SMALL LETTER E L
x LATIN SMALL LETTER X L
e LATIN SMALL LETTER E L
```

RIGHT-TO-LEFT OVERRIDE U+202E

Visual spoof*

**now prevented by many client programs*



KrebsonSecurity

In-depth security news and investigation

ADVERTISING/SPEAKI

26 **'Right-to-Left Override' Aids Email Attacks**

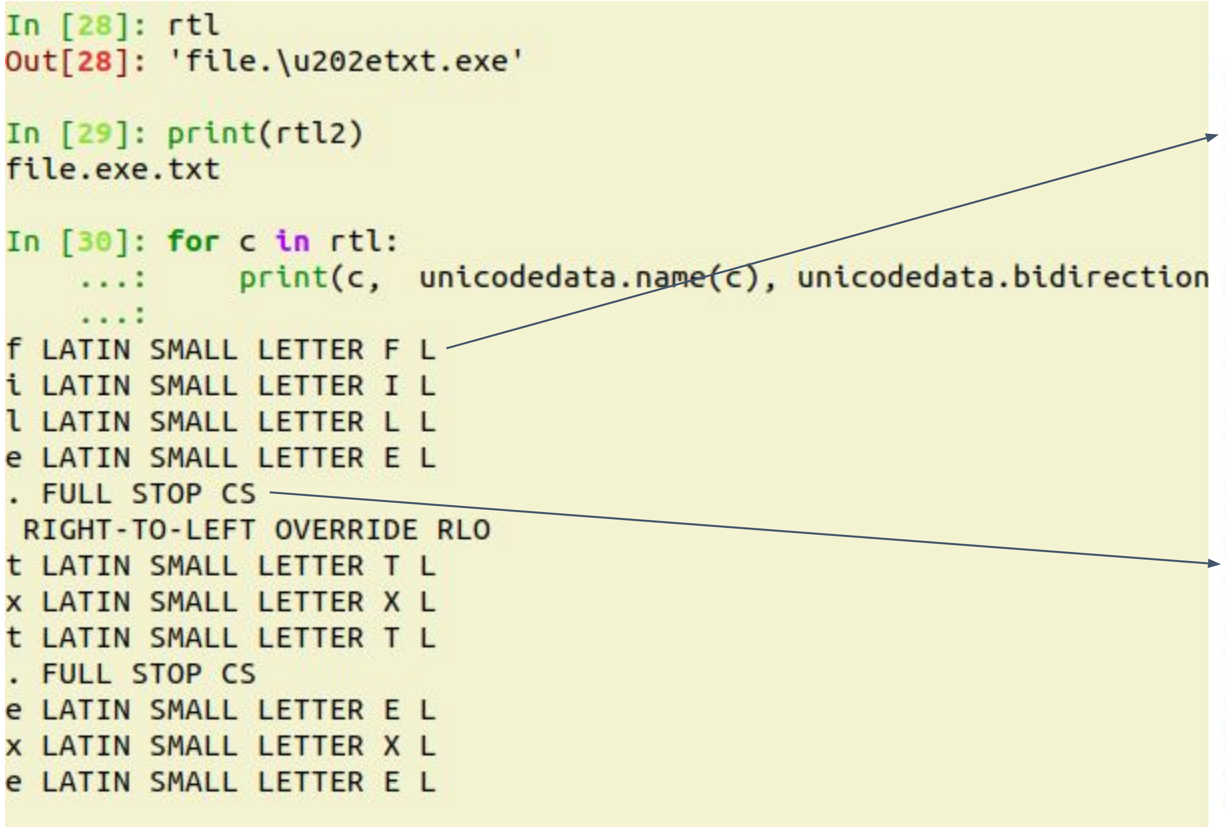
SEP 11

Computer crooks and spammers are abusing a little-known encoding method that makes it easy to disguise malicious executable files (.exe) as relatively harmless documents, such as text or Microsoft Word files.

Advertisement

Why Dar

Text direction enforcement



```
In [28]: rtl
Out[28]: 'file.\u202etxt.exe'

In [29]: print(rtl2)
file.exe.txt

In [30]: for c in rtl:
    ...:     print(c,  unicodedata.name(c), unicodedata.bidirection
    ...:
f LATIN SMALL LETTER F L
i LATIN SMALL LETTER I L
l LATIN SMALL LETTER L L
e LATIN SMALL LETTER E L
. FULL STOP CS
 RIGHT-TO-LEFT OVERRIDE RLO
t LATIN SMALL LETTER T L
x LATIN SMALL LETTER X L
t LATIN SMALL LETTER T L
. FULL STOP CS
e LATIN SMALL LETTER E L
x LATIN SMALL LETTER X L
e LATIN SMALL LETTER E L
```

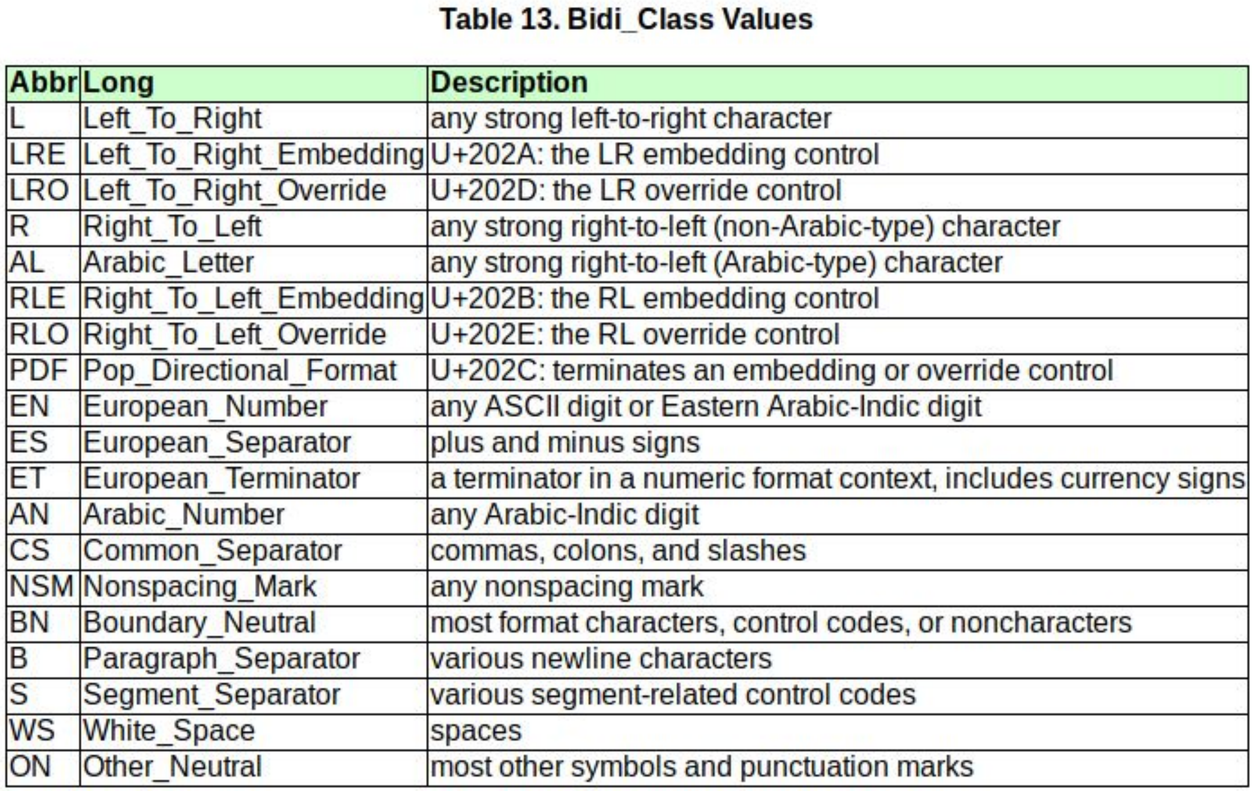


Table 13. Bidi_Class Values

Abbr	Long	Description
L	Left_To_Right	any strong left-to-right character
LRE	Left_To_Right_Embedding	U+202A: the LR embedding control
LRO	Left_To_Right_Override	U+202D: the LR override control
R	Right_To_Left	any strong right-to-left (non-Arabic-type) character
AL	Arabic_Letter	any strong right-to-left (Arabic-type) character
RLE	Right_To_Left_Embedding	U+202B: the RL embedding control
RLO	Right_To_Left_Override	U+202E: the RL override control
PDF	Pop_Directional_Format	U+202C: terminates an embedding or override control
EN	European_Number	any ASCII digit or Eastern Arabic-Indic digit
ES	European_Separator	plus and minus signs
ET	European_Terminator	a terminator in a numeric format context, includes currency signs
AN	Arabic_Number	any Arabic-Indic digit
CS	Common_Separator	commas, colons, and slashes
NSM	Nonspacing_Mark	any nonspacing mark
BN	Boundary_Neutral	most format characters, control codes, or noncharacters
B	Paragraph_Separator	various newline characters
S	Segment_Separator	various segment-related control codes
WS	White_Space	spaces
ON	Other_Neutral	most other symbols and punctuation marks

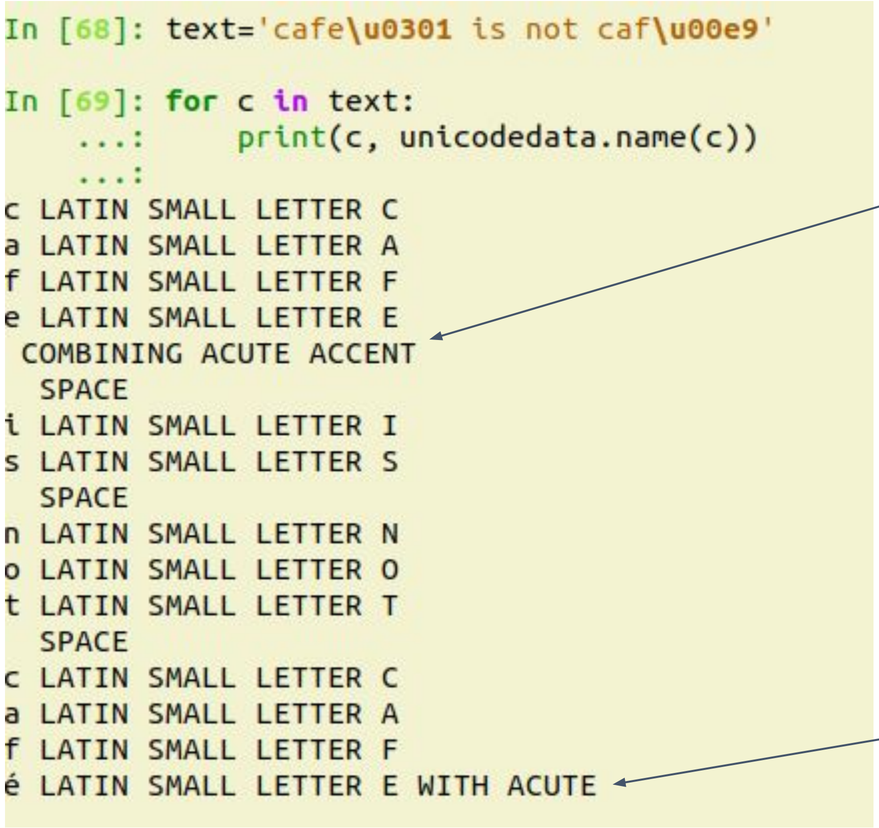
Source: Unicode Standard Annex #44

Rule #7

Enforce consistent text direction

Normalization

When café is not café?

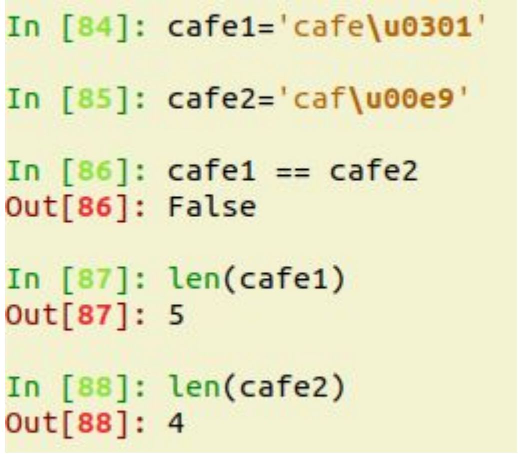


```
In [68]: text='cafe\u0301 is not caf\u00e9'

In [69]: for c in text:
    ...:     print(c, unicodedata.name(c))
    ...:
c LATIN SMALL LETTER C
a LATIN SMALL LETTER A
f LATIN SMALL LETTER F
e LATIN SMALL LETTER E
 COMBINING ACUTE ACCENT
  SPACE
i LATIN SMALL LETTER I
s LATIN SMALL LETTER S
  SPACE
n LATIN SMALL LETTER N
o LATIN SMALL LETTER O
t LATIN SMALL LETTER T
  SPACE
c LATIN SMALL LETTER C
a LATIN SMALL LETTER A
f LATIN SMALL LETTER F
é LATIN SMALL LETTER E WITH ACUTE
```

Two U+000E and U+0301 characters combined on display

Single character U+00E9

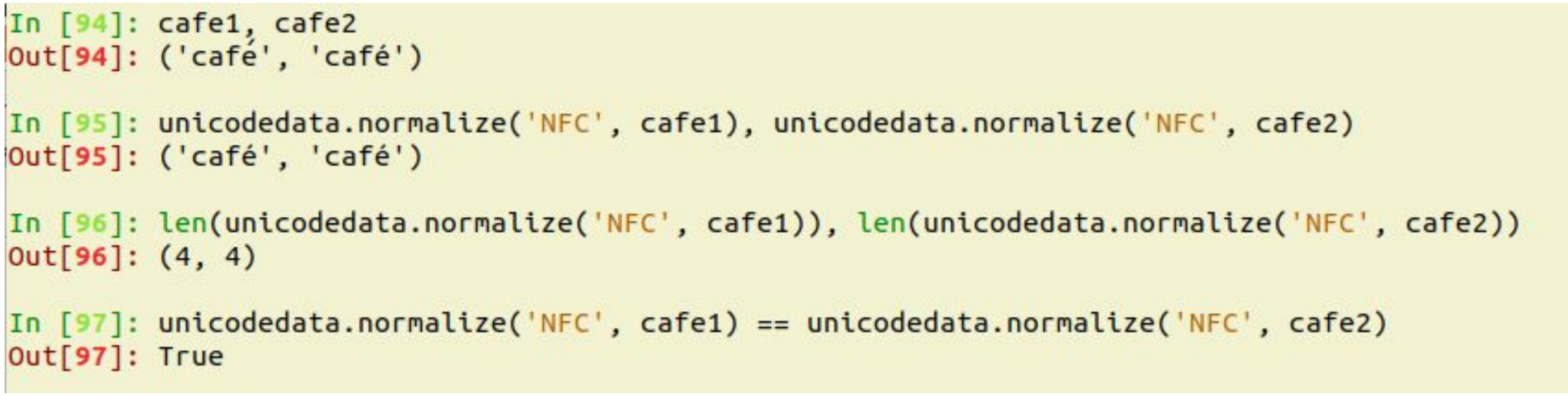
When it is a problem?



```
In [84]: cafe1='cafe\u0301'

In [85]: cafe2='caf\u00e9'

In [86]: cafe1 == cafe2
Out[86]: False

In [87]: len(cafe1)
Out[87]: 5

In [88]: len(cafe2)
Out[88]: 4
```

- Collation
- Sorting
- Comparison
- Persistence of data with different composition

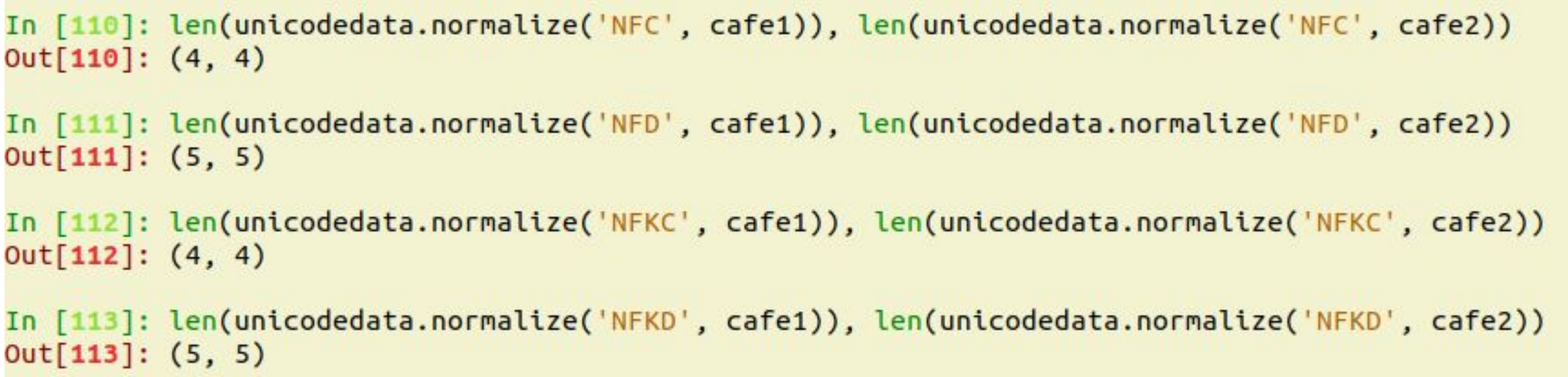
Unicode normalization



```
In [94]: cafe1, cafe2
Out[94]: ('café', 'café')

In [95]: unicodedata.normalize('NFC', cafe1), unicodedata.normalize('NFC', cafe2)
Out[95]: ('café', 'café')

In [96]: len(unicodedata.normalize('NFC', cafe1)), len(unicodedata.normalize('NFC', cafe2))
Out[96]: (4, 4)

In [97]: unicodedata.normalize('NFC', cafe1) == unicodedata.normalize('NFC', cafe2)
Out[97]: True
```

- NFC = Normalization Form “C”
- Converts Unicode characters to a single, consistent form
- U+000E U+0301 and U+00E9 are Unicode “canonical equivalents”

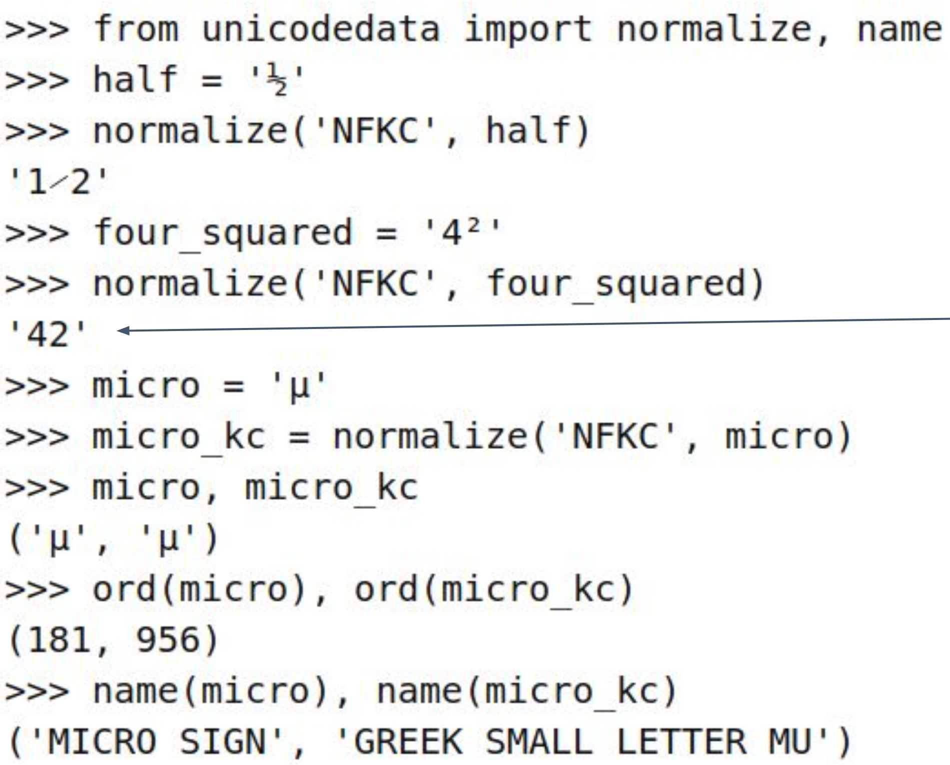
NFC, NFD, NFKC, NFKD?😱



```
In [110]: len(unicodedata.normalize('NFC', cafe1)), len(unicodedata.normalize('NFC', cafe2))
Out[110]: (4, 4)

In [111]: len(unicodedata.normalize('NFD', cafe1)), len(unicodedata.normalize('NFD', cafe2))
Out[111]: (5, 5)

In [112]: len(unicodedata.normalize('NFKC', cafe1)), len(unicodedata.normalize('NFKC', cafe2))
Out[112]: (4, 4)

In [113]: len(unicodedata.normalize('NFKD', cafe1)), len(unicodedata.normalize('NFKD', cafe2))
Out[113]: (5, 5)
```

- NFC will compose, make shorter - **é** becomes U+00E9
- NFD will decompose, make longer - **é** becomes U+000E U+0301
- NFKC and NFKD will also replace “compatibility characters”
  - **Possible loss of information!**

NFKC, NFKD



```
>>> from unicodedata import normalize, name
>>> half = '½'
>>> normalize('NFKC', half)
'1⁄2'
>>> four_squared = '4²'
>>> normalize('NFKC', four_squared)
'42'
>>> micro = 'µ'
>>> micro_kc = normalize('NFKC', micro)
>>> micro, micro_kc
('µ', 'μ')
>>> ord(micro), ord(micro_kc)
(181, 956)
>>> name(micro), name(micro_kc)
('MICRO SIGN', 'GREEK SMALL LETTER MU')
```

This is a significant loss of information!

Source: Luciano Ramalho "Fluent Python"

Compatibility normalization

Ⅻ

- Precomposed Roman numerals Ⅻ U+216B

Compatibility normalization

Ⅻ

- Precomposed Roman numerals Ⅻ U+216B
- Ⅹ U+2169 Ⅰ U+2160 Ⅰ U+2160 (Roman numerals)

Compatibility normalization

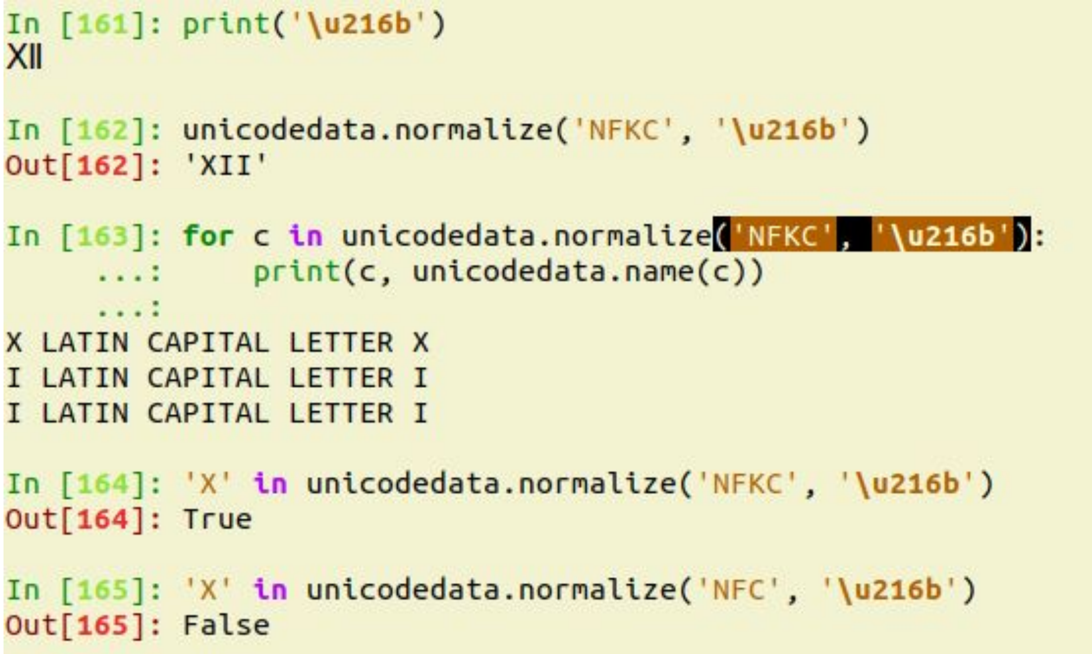
Ⅻ

- Precomposed Roman numerals Ⅻ U+216B
- Ⅹ U+2169 Ⅰ U+2160 Ⅰ U+2160 (Roman numerals)
- **X** U+0058 **I** U+0049 **I** U+0049 (Latin letters)

Another typical example:

- ¼ U+00BC → **1** U+0031 ⁄ U+2044 **4** U+0034

NFKC, NFKD



```
In [161]: print('\u216b')
Ⅻ

In [162]: unicodedata.normalize('NFKC', '\u216b')
Out[162]: 'XII'

In [163]: for c in unicodedata.normalize('NFKC', '\u216b'):
     ...:     print(c, unicodedata.name(c))
     ...:
X LATIN CAPITAL LETTER X
I LATIN CAPITAL LETTER I
I LATIN CAPITAL LETTER I

In [164]: 'X' in unicodedata.normalize('NFKC', '\u216b')
Out[164]: True

In [165]: 'X' in unicodedata.normalize('NFC', '\u216b')
Out[165]: False
```

NKFC replaces *single* Ⅻ U+216B ROMAN NUMERAL TWELVE character by *three* Roman digits represented by Latin letters X and I

Useful for search and comparison

- is there “X” in “XII”?
- is there “f” in “ffi”?

Many typesetting programs will replace popular “compatibility sequences” by appropriate Unicode characters:

--> replaced by → U+2192 RIGHTWARDS ARROW

Rule #8

Normalize Unicode text

Validation strategies

Policy decisions

Let’s talk about your user base for a moment…

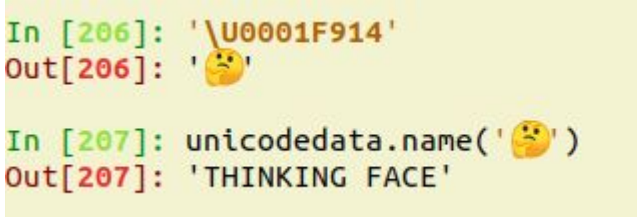
- Are they expected to communicate in Chinese, English, Polish, Arabic…?
- Do we expect text in Linear-B, Ugaritic, Klingon?
  - Can you process data in any of these languages?
- What kind of text is expected where?
- E.g. names - are they composed of letters only (“Portia Sutcliffe”)
  - Or maybe we also expect digits and punctuation (“Cynthia O'Keefe”, “Howard Upperton-Wildingham III”)
- Define what is *valid* text
- Define appropriate *valid* categories, scripts and text directions
- Normalize Unicode input prior to validation

Policy decisions

Let’s talk about your user base for a moment…

- Are they expected to communicate in Chinese, English, Polish, Arabic…?
- Do we expect text in Linear-B, Ugaritic, Klingon?
  - Can you process data in any of these languages?
- What kind of text is expected where?
- E.g. names - are they composed of letters only (“Portia Sutcliffe”)
  - Or maybe we also expect digits and punctuation (“Cynthia O'Keefe”, “Howard Upperton-Wildingham III”)
- Define what is valid text
- Define appropriate valid categories, scripts and text directions
- Normalize Unicode input prior to validation
- Do we have free-text fields?
  - If yes, how *free* is the “free text”?
  - Letters, digits, punctuation?
  - Symbols?
    - Because if someone explains “I clicked File > Properties > General” it takes symbols
- Is all this part of localization for given region?
  - Along with database collation, currency, numbers...

Questions? □



```
In [206]: '\U0001F914'
Out[206]: '🤔'

In [207]: unicodedata.name('🤔')
Out[207]: 'THINKING FACE'
```

pawel.krawczyk@hush.com
+44 7879 180015
https://www.linkedin.com/in/pawelkrawczyk/