

Unicode solutions in Python 2 and 3

	00A	00B	00C	00D	00E	00F
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	00A0	00A1	00A2	00A3	00A4	00A5
ä å æ ø ù ú û ü ý þ ÿ	00B0	00B1	00B2	00B3	00B4	00B5
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	00C0	00C1	00C2	00C3	00C4	00C5
ä å æ ø ù ú û ü ý þ ÿ	00D0	00D1	00D2	00D3	00D4	00D5
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	00E0	00E1	00E2	00E3	00E4	00E5
ä å æ ø ù ú û ü ý þ ÿ	00F0	00F1	00F2	00F3	00F4	00F5

1F61	1F62	1F63	1F64
😏	😬	😈	😺
1F610	1F620	1F630	1F640
😏	😬	😈	😺
1F611	1F621	1F631	1F641
😏	😬	😈	😺
1F612	1F622	1F632	1F642
😏	😬	😈	😺
1F613	1F623	1F633	1F643
😏	😬	😈	😺
1F614	1F624	1F634	1F644
😏	😬	😈	😺
1F615	1F625	1F635	1F645
😏	😬	😈	😺
1F616	1F626	1F636	1F646
😏	😬	😈	😺
1F617	1F627	1F637	1F647
😏	😬	😈	😺
1F618	1F628	1F638	1F648
😏	😬	😈	😺
1F619	1F629	1F639	1F649
😏	😬	😈	😺
1F61A	1F62A	1F63A	1F64A
😏	😬	😈	😺

053	054	055	056	057	058
𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0530	0540	0550	0560	0570	0580
𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0531	0541	0551	0561	0571	0581
𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0532	0542	0552	0562	0572	0582
𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0533	0543	0553	0563	0573	0583
𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0534	0544	0554	0564	0574	0584
𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0535	0545	0555	0565	0575	0585
𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0536	0546	0556	0566	0576	0586
𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0537	0547	0557	0567	0577	0587
𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0538	0548	0558	0568	0578	0588
𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0539	0549	0559	0569	0579	0589
𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
053A	054A	055A	056A	057A	058A
𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
053B	054B	055B	056B	057B	058B
𐌵	𐌶	𐌷	𐌸	𐌹	𐌺

1314	1315	1316	1317	1318	1319	131A	131B
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺	𐌻
13140	13150	13160	13170	13180	13190	131A0	131B0
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺	𐌻
13141	13151	13161	13171	13181	13191	131A1	131B1
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺	𐌻
13142	13152	13162	13172	13182	13192	131A2	131B2
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺	𐌻
13143	13153	13163	13173	13183	13193	131A3	131B3
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺	𐌻
13144	13154	13164	13174	13184	13194	131A4	131B4
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺	𐌻
13145	13155	13165	13175	13185	13195	131A5	131B5
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺	𐌻
13146	13156	13166	13176	13186	13196	131A6	131B6
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺	𐌻
13147	13157	13167	13177	13187	13197	131A7	131B7
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺	𐌻
13148	13158	13168	13178	13188	13198	131A8	131B8
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺	𐌻
13149	13159	13169	13179	13189	13199	131A9	131B9
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺	𐌻

0D0	0D1	0D2	0D3	0D4	0D5	0D6
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0D00	0D01	0D02	0D03	0D04	0D05	0D06
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0D07	0D08	0D09	0D0A	0D0B	0D0C	0D0D
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0D0E	0D0F	0D10	0D11	0D12	0D13	0D14
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0D15	0D16	0D17	0D18	0D19	0D1A	0D1B
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0D1C	0D1D	0D1E	0D1F	0D20	0D21	0D22
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0D23	0D24	0D25	0D26	0D27	0D28	0D29
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0D2A	0D2B	0D2C	0D2D	0D2E	0D2F	0D30
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0D31	0D32	0D33	0D34	0D35	0D36	0D37
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0D38	0D39	0D3A	0D3B	0D3C	0D3D	0D3E
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0D3F	0D40	0D41	0D42	0D43	0D44	0D45
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0D46	0D47	0D48	0D49	0D4A	0D4B	0D4C
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0D4D	0D4E	0D4F	0D50	0D51	0D52	0D53
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0D54	0D55	0D56	0D57	0D58	0D59	0D5A
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺
0D5B	0D5C	0D5D	0D5E	0D5F	0D60	0D61
𐌴	𐌵	𐌶	𐌷	𐌸	𐌹	𐌺

HEX	C	J	K	V
50D0	𐌴	𐌵	𐌶	𐌷
50D1	𐌴	𐌵	𐌶	𐌷
50D2	𐌴	𐌵	𐌶	𐌷
50D3	𐌴	𐌵	𐌶	𐌷
50D4	𐌴	𐌵	𐌶	𐌷
50D5	𐌴	𐌵	𐌶	𐌷
50D6	𐌴	𐌵	𐌶	𐌷
50D7	𐌴	𐌵	𐌶	𐌷
50D8	𐌴	𐌵	𐌶	𐌷
50D9	𐌴	𐌵	𐌶	𐌷
50DA	𐌴	𐌵	𐌶	𐌷
50DB	𐌴	𐌵	𐌶	𐌷
50DC	𐌴	𐌵	𐌶	𐌷
50DD	𐌴	𐌵	𐌶	𐌷
50DE	𐌴	𐌵	𐌶	𐌷

Unicode solutions in Python 2 and 3

Latin-1

Armenian

Malayalam

emoticons

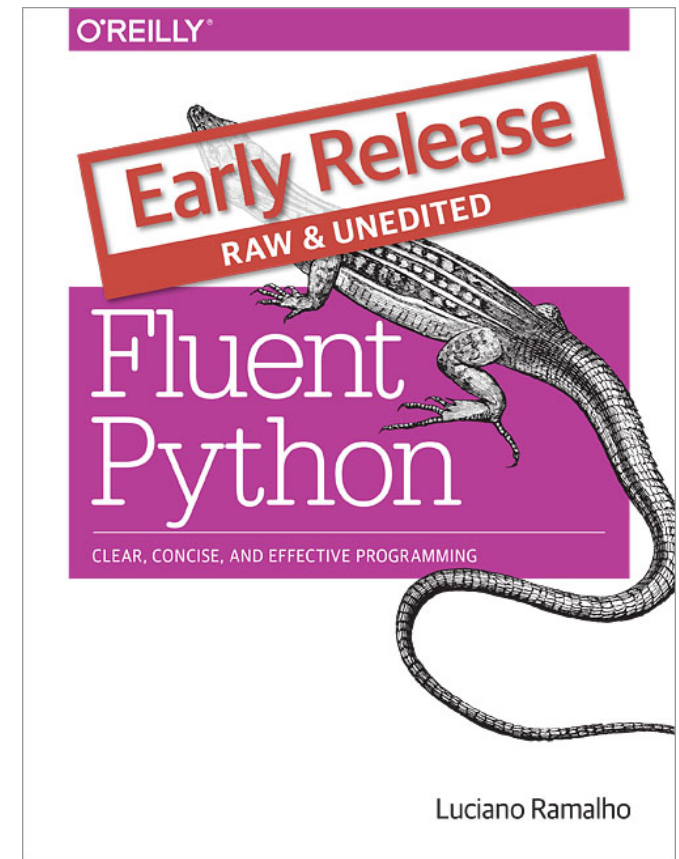
Egyptian
Hieroglyphs

CJK
Unified
Ideographs

About me: Luciano Ramalho

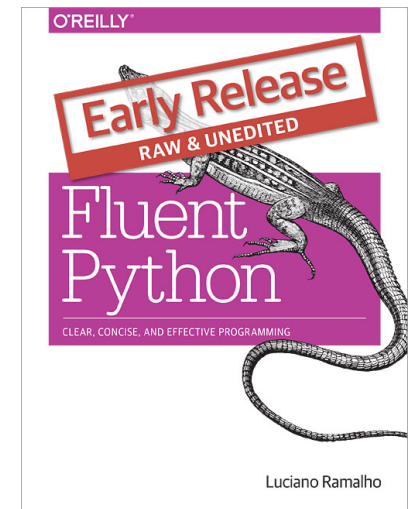
- Programming in Python since 1998
- Focus on content management (i.e. text wrangling)
- Teaching Python since 1999
- Speaker at PyCon US, OSCON, FISL, PythonBrasil, RuPy, QCon...
- Author of *Fluent Python*
- Twitter: [@ramalhoorg](#)
- Native language: Português
 - “ação”

4 non-ASCII
characters here



Resources

- All code, slides and images used in this talk:
 - <https://github.com/fluentpython/unicode-solutions>
- **Fluent Python**
 - <http://shop.oreilly.com/product/0636920032519.do>
 - Relevant content and examples:
 - Chapter 4: *Text versus Bytes*
 - all 39 pages
 - Chapter 18: *Concurrency with asyncio*
 - the *charfinder* examples



A bite of theory

	130E	130F	1310	1311	1312	1313	1314	1315	1316	1317	1318	1319	131A	131B
0	 130E0	 130F0	 13100	 13110	 13120	 13130	 13140	 13150	 13160	 13170	 13180	 13190	 131A0	 131B0
1	 130E1	 130F1	 13101	 13111	 13121	 13131	 13141	 13151	 13161	 13171	 13181	 13191	 131A1	 131B1
2	 130E2	 130F2	 13102	 13112	 13122	 13132	 13142	 13152	 13162	 13172	 13182	 13192	 131A2	 131B2
3	 130E3	 130F3	 13103	 13113	 13123	 13133	 13143	 13153	 13163	 13173	 13183	 13193	 131A3	 131B3
4	 130E4	 130F4	 13104	 13114	 13124	 13134	 13144	 13154	 13164	 13174	 13184	 13194	 131A4	 131B4
5	 130E5	 130F5	 13105	 13115	 13125	 13135	 13145	 13155	 13165	 13175	 13185	 13195	 131A5	 131B5
6	 130E6	 130F6	 13106	 13116	 13126	 13136	 13146	 13156	 13166	 13176	 13186	 13196	 131A6	 131B6
7	 130E7	 130F7	 13107	 13117	 13127	 13137	 13147	 13157	 13167	 13177	 13187	 13197	 131A7	 131B7
8	 130E8	 130F8	 13108	 13118	 13128	 13138	 13148	 13158	 13168	 13178	 13188	 13198	 131A8	 131B8
9	 130E9	 130F9	 13109	 13119	 13129	 13139	 13149	 13159	 13169	 13179	 13189	 13199	 131A9	 131B9
A	 130EA	 130FA	 1310A	 1311A	 1312A	 1313A	 1314A	 1315A	 1316A	 1317A	 1318A	 1319A	 131AA	 131BA
B	 130EB	 130FB	 1310B	 1311B	 1312B	 1313B	 1314B	 1315B	 1316B	 1317B	 1318B	 1319B	 131AB	 131BB
C	 130EC	 130FC	 1310C	 1311C	 1312C	 1313C	 1314C	 1315C	 1316C	 1317C	 1318C	 1319C	 131AC	 131BC
D	 130ED	 130FD	 1310D	 1311D	 1312D	 1313D	 1314D	 1315D	 1316D	 1317D	 1318D	 1319D	 131AD	 131BD
E	 130EE	 130FE	 1310E	 1311E	 1312E	 1313E	 1314E	 1315E	 1316E	 1317E	 1318E	 1319E	 131AE	 131BE
F	 130EF	 130FF	 1310F	 1311F	 1312F	 1313F	 1314F	 1315F	 1316F					

The single-byte codepage ballet

tk

_0 _1 _2 _3 _4 _5 _6 _7 _8 _9 _A _B _C _D _E _F

00 KOI8-R

10

20 ! " # \$ % & ' () * + , - . /

30 0 1 2 3 4 5 6 7 8 9 : ; < = > ?

40 @ A B C D E F G H I J K L M N O

50 P Q R S T U V W X Y Z [\] ^ _

60 ` a b c d e f g h i j k l m n o

70 p q r s t u v w x y z { | } ~

80

90

A0

B0

C0

D0

E0

F0





í „ ... † ‡ €
“ ” • — —
J α Γ ! §
i γ μ ŷ

Video: <https://www.youtube.com/watch?v=J4qioAacrYo>

Source code: <http://bit.ly/10qt0MZ>

Why Unicode

- Too many incompatible byte encodings
- Separate concepts:
 - character identity: one **code point** for each abstract character
 - U+0041 → LATIN CAPITAL LETTER A
 - U+096C → DEVANAGARI DIGIT SIX
 - binary representation: multiple **encodings**

• U+0041 → 0x41		0x41 0x00
• U+096C → 0xE0 0xA5 0xAC		0x6C 0x09
		
		

A sample of encodings

char.	code point	ascii	latin1	cp1252	cp437	gb2312	utf-8	utf-16le
A	U+0041	41	41	41	41	41	41	41 00
¿	U+00BF	*	BF	BF	A8	*	C2 BF	BF 00
Ã	U+00C3	*	C3	C3	*	*	C3 83	C3 00
á	U+00E1	*	E1	E1	A0	A8 A2	C3 A1	E1 00
Ω	U+03A9	*	*	*	EA	A6 B8	CE A9	A9 03
ꣳ	U+06BF	*	*	*	*	*	DA BF	BF 06
“	U+201C	*	*	93	*	A1 B0	E2 80 9C	1C 20
€	U+20AC	*	*	80	*	*	E2 82 AC	AC 20
Г	U+250C	*	*	*	DA	A9 B0	E2 94 8C	0C 25
气	U+6C14	*	*	*	*	C6 F8	E6 B0 94	14 6C
氣	U+6C23	*	*	*	*	*	E6 B0 A3	23 6C
♫	U+1D11E	*	*	*	*	*	F0 9D 84 9E	34 D8 1E DD

Byte and text solutions

	008	009	00A	00B	00C	00D	00E	00F
0	XXX 0080	DCS 0090	NB SP 00A0	° 00B0	À 00C0	Đ 00D0	à 00E0	đ 00F0
1	XXX 0081	PU1 0091	¡ 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
2	BPH 0082	PU2 0092	¢ 00A2	² 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
3	NBH 0083	STS 0093	£ 00A3	³ 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
4	IND 0084	CCH 0094	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	NEL 0085	MW 0095	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	õ 00F5
6	SSA 0086	SPA 0096	¦ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
7	ESA 0087	EPA 0097	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
8	HTS 0088	SOS 0098	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
9	HTJ 0089	XXX 0099	© 00A9	¹ 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
A	VTJ 008A	SCI 009A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
B	PLD 008B	CSI 009B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
C	PLU 008C	ST 009C	¬ 00AC	¼ 00BC	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
D	RI 008D	OSC 009D	SHY 00AD	½ 00BD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
E	SS2 008E	PM 009E	® 00AE	¾ 00BE	Î 00CE	Þ 00DE	î 00EE	þ 00FE
F	SS3 008F	APC 009F	— 00AF	¿ 00BF	Ï 00CF	ß 00DF	ï 00EF	ÿ 00FF

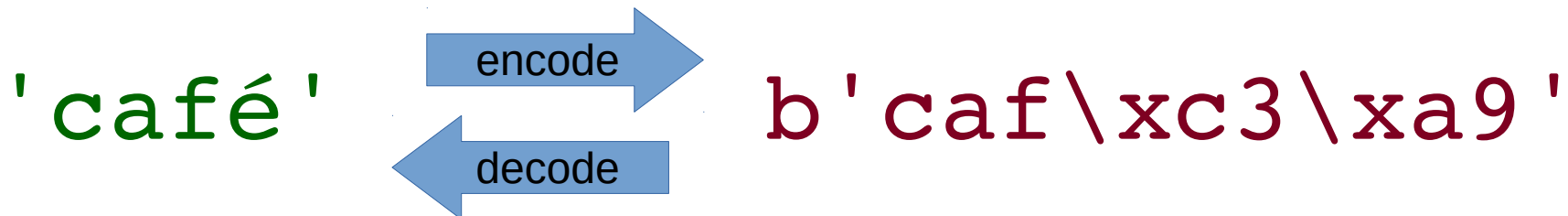
Data types for text or bytes



	Python 2.7	Python 3.4
Human text	unicode <code>u'café', u'caf\xe9'</code>	str <code>'café', u'café', 'caf\xe9'</code>
(immutable) Bytes	str <code>'café', 'caf\xe9', b'café'</code>	bytes <code>b'caf\xc3\xa9'</code>
(mutable) Bytes	bytearray <code>bytearray(b'caf\xc3\xa9')</code>	bytearray <code>bytearray(b'caf\xc3\xa9')</code>

.encode() v. .decode()

- “Humans use text. Computers speak bytes.”
 - Esther Nam and Travis Fischer in *Character encoding and Unicode in Python* (Pycon US 2014)



- Use **.encode()** to convert **human** text to **bytes**
- Use **.decode()** to convert **bytes** to **human** text

2.7 gotcha: methods `.encode()` and `.decode()` exist in both str and unicode types!

Text v. bytes

Py3

```
>>> text = 'café'
>>> type(text)
<class 'str'>
>>> len(text)
4
>>> list(text)
['c', 'a', 'f', 'é']
>>> print(text)
café
>>> octets = text.encode('utf8')
>>> type(octets)
<class 'bytes'>
>>> octets
b'caf\xc3\xa9'
>>> len(octets)
5
>>> list(octets)
[99, 97, 102, 195, 169]
>>> print(octets)
b'caf\xc3\xa9'
>>> octets.decode('utf8')
'café'
>>> str is bytes
False
```

- Items in unicode text are characters
- Items in byte sequences are bytes
 - integers 0...255
 - shown as ASCII sequences with a **b**" prefix for convenience

Text v. bytes

Py3

```
>>> text = 'café'
>>> type(text)
<class 'str'>
>>> len(text)
4
>>> list(text)
['c', 'a', 'f', 'é']
>>> print(text)
café
>>> octets = text.encode('utf8')
>>> type(octets)
<class 'bytes'>
>>> octets
b'caf\xc3\xa9'
>>> len(octets)
5
>>> list(octets)
[99, 97, 102, 195, 169]
>>> print(octets)
b'caf\xc3\xa9'
>>> octets.decode('utf8')
'café'
>>> str is bytes
False
```

Py2

```
>>> text = u'café'
>>> type(text)
<type 'unicode'>
>>> len(text)
4
>>> list(text)
[u'c', u'a', u'f', u'\xe9']
>>> print(text)
café
>>> octets = text.encode('utf8')
>>> type(octets)
<type 'str'>
>>> octets
'caf\xc3\xa9'
>>> len(octets)
5
>>> list(octets)
['c', 'a', 'f', '\xc3', '\xa9']
>>> print(octets)
café
>>> octets.decode('utf8')
u'caf\xe9'
>>> str is bytes
True
```


Bytes and bytearray

A blue starburst shape with the text 'Py3' in yellow.

```
>>> octets = bytes('café', encoding='utf_8')
>>> octets
b'caf\xc3\xa9'
>>> octets[0]
99
>>> octets[:1]
b'c'
>>> octet_arr = bytearray(octets)
>>> octet_arr
bytearray(b'caf\xc3\xa9')
>>> octet_arr[-1:]
bytearray(b'\xa9')
```

Bytes and bytearray

Py3

```
>>> octets = bytes('café', encoding='utf_8')
>>> octets
b'caf\xc3\xa9'
>>> octets[0]
99
>>> octets[:1]
b'c'
>>> octet_arr = bytearray(octets)
>>> octet_arr
bytearray(b'caf\xc3\xa9')
>>> octet_arr[-1:]
bytearray(b'\xa9')
```

Py2

```
>>> octets = bytes('café')
>>> octets
'caf\xc3\xa9'
>>> octets[0]
'c'
>>> octets[:1]
'c'
>>> octet_arr = bytearray(octets)
>>> octet_arr
bytearray(b'caf\xc3\xa9')
>>> octet_arr[-1:]
bytearray(b'\xa9')
```

Common codecs

- codec = encoder/decoder table or algorithm

```
>>> for codec in ['latin_1', 'utf_8', 'utf_16']:
...     print('{:8} {!r}'.format(codec, u'El Niño'.encode(codec)))
...
latin_1  b'El Ni\xf1o'
utf_8    b'El Ni\xc3\xb1o'
utf_16   b'\xff\xfeE\x0l\x00 \x00N\x00i\x00\xf1\x00o\x00'
```



Common codecs

- codec = encoder/decoder table or algorithm

```
>>> for codec in ['latin_1', 'utf_8', 'utf_16']:
...     print('{:8} {!r}'.format(codec, u'El Niño'.encode(codec)))
...
latin_1  b'El Ni\xf1o'
utf_8    b'El Ni\xc3\xb1o'
utf_16   b'\xff\xfeE\x0l\x00 \x0N\x0i\x00\xf1\x0o\x00'
```



```
>>> for codec in ['latin_1', 'utf_8', 'utf_16']:
...     print '{:8} {!r}'.format(codec, u'El Niño'.encode(codec))
...
latin_1  'El Ni\xf1o'
utf_8    'El Ni\xc3\xb1o'
utf_16   '\xff\xfeE\x0l\x00 \x0N\x0i\x00\xf1\x0o\x00'
```



Coping with Unicode Errors

- **SyntaxError**

- A **.py** file has source code in an unexpected encoding

- **UnicodeDecodeError**

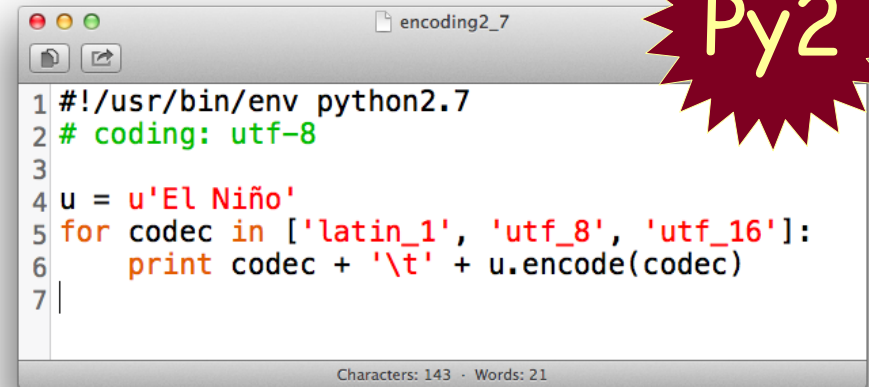
- A binary sequence is contains bytes that are not valid in the expected encoding

- **UnicodeEncodeError**

- A Unicode string contains codepoints that have no representation in the desired encoding

Coping with SyntaxError

- A **.py** file uses an unexpected encoding
 - The source file encoding is not the default, and no `# coding` comment was found.
 - The source file encoding is not the one declared in the `# coding` comment
- Default source encoding:
 - Python 2.7 → ASCII
 - Python 3.x → UTF-8



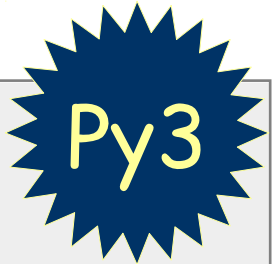
```
1#!/usr/bin/env python2.7
2# coding: utf-8
3
4u = u'El Niño'
5for codec in ['latin_1', 'utf_8', 'utf_16']:
6    print codec + '\t' + u.encode(codec)
7|
```

Characters: 143 · Words: 21

2.7 gotcha:
default source
encoding is ASCII

UnicodeEncodeError

- A character in the Unicode text cannot be represented in the target byte encoding
 - happens with legacy encodings that cover only a small subset of Unicode



```
>>> city = 'São Paulo'
>>> city.encode('utf_8')
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')
b'S\xe3o Paulo'
>>> city.encode('cp437')
Traceback (most recent call last):
...
UnicodeEncodeError: 'charmap' codec can't encode character '\xe3' in position 1: character maps to <undefined>
>>> city.encode('cp437', errors='ignore')
b'So Paulo'
>>> city.encode('cp437', errors='replace')
b'S?o Paulo'
>>> city.encode('cp437', errors='xmlcharrefreplace')
b'S&#227;o Paulo'
```

UnicodeEncodeError

- A character in the Unicode text cannot be represented in the target byte encoding
 - happens with legacy encodings that cover only a small subset of Unicode

```
>>> city = u'São Paulo'
>>> city.encode('utf_8')
'S\xc3\xa3o Paulo'
>>> city.encode('utf_16')
'\xff\xfeS\x00\xe3\x00o\x00 \x00P\x00a\x00u\x00l\x00o\x00'
>>> city.encode('iso8859_1')
'S\xe3o Paulo'
>>> city.encode('cp437')
Traceback (most recent call last):
...
UnicodeEncodeError: 'charmap' codec can't encode character u'\xe3' in position 1: character maps to <undefined>
>>> city.encode('cp437', errors='ignore')
'So Paulo'
>>> city.encode('cp437', errors='replace')
'S?o Paulo'
>>> city.encode('cp437', errors='xmlcharrefreplace')
'S&#227;o Paulo'
```



UnicodeDecodeError

- Invalid byte in the source encoding
 - more common with the UTF encodings



```
>>> octets = b'Montr\xe9al'
>>> octets.decode('cp1252')
'Montréal'
>>> octets.decode('iso8859_7')
'Montrial'
>>> octets.decode('koi8_r')
'MontrIal'
>>> octets.decode('utf_8')
Traceback (most recent call last):
...
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe9 in position 5:
invalid continuation byte
>>> octets.decode('utf_8', errors='replace')
'Montr al'
```

UnicodeDecodeError

A red starburst shape containing the text "Py2" in yellow.

```
>>> octets = b'Montr\xe9al'
>>> octets.decode('cp1252')
u'Montr\xe9al'
>>> print _
Montréal
>>> octets.decode('iso8859_7') == u'Montrial'
True
>>> print octets.decode('iso8859_7')
Montrial
>>> octets.decode('koi8_r') == u'MontrMal'
True
>>> print octets.decode('koi8_r')
MontrMal
>>> octets.decode('utf_8')
Traceback (most recent call last):
...
UnicodeDecodeError: 'utf8' codec can't decode byte 0xe9 in position 5:
invalid continuation byte
>>> octets.decode('utf_8', errors='replace') == u'Montr al'
True
>>> print octets.decode('utf_8', errors='replace')
Montr al
```


Best practice to avoid errors

The Unicode sandwich



bytes → str

100% str

str → bytes

Decode bytes on input,

process text only,

encode text on output.

Figure 4-2 of *Fluent Python*, after Ned Batchelder's *Pragmatic Unicode* talk: <http://nedbatchelder.com/text/unipain.html>

How to implement the sandwich

- Always specify encoding when opening text files, so you send and receive text, and not bytes
 - in Python 2.7, remember to use `io.open()`

2.7 gotcha:
no way to specify encoding
with built-in `open(...)`.
Must use `io.open(...)`.

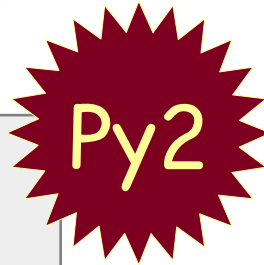
Text I/O

- **open()** built-in is Unicode-aware in Python 3
 - text mode default accepts **encoding** argument
 - **.write()** method only accepts Unicode text
 - **.read()** method returns Unicode text

```
>>> with open('cafe.txt', 'w', encoding='utf_8') as fp:
...     write_count = fp.write('café')
...
>>> write_count
4
>>> with open('cafe.txt', 'r', encoding='utf_8') as fp:
...     text = fp.read()
...
>>> text
'café'
```



Bytes or text I/O



```
>>> with open('cafe.txt', 'w') as fp:
...     fp.write('café')
...
>>> with open('cafe.txt', 'r') as fp:
...     octets = fp.read()
...
>>> octets
'caf\xc3\xa9'
```

- **open()** built-in only supports bytes in Python 2
 - even in “text mode” (deals with CR+LF...)
 - no **encoding** argument accepted
 - **.write()** implicitly converts **unicode** to **str** using ASCII codec
 - remember: 'café' is actually b'caf\xc3\xa9'
 - **.read()** method always returns bytes

Bytes or text I/O

Py2

```
>>> with open('cafe.txt', 'w') as fp:
...     fp.write('café')
...
>>> with open('cafe.txt', 'r') as fp:
...     octets = fp.read()
...
>>> octets
'caf\xc3\xa9'
```

```
>>> import io
>>> with io.open('cafe.txt', 'w', encoding='utf_8') as fp:
...     write_count = fp.write(u'café')
...
>>> write_count
4L
>>> with io.open('cafe.txt', 'r', encoding='utf_8') as fp:
...     text = fp.read()
...
>>> text
u'caf\xe9'
```

- **io.open()** is the Unicode-aware **open()** from Python 3 backported to Python 2.6+

Bytes or text I/O

Py2

```
>>> with open('cafe.txt', 'w') as fp:
...     fp.write('café')
...
>>> with open('cafe.txt', 'r') as fp:
...     octets = fp.read()
...
>>> octets
'caf\xc3\xa9'
```

```
>>> import io
>>> with io.open('cafe.txt', 'w', encoding='utf_8') as fp:
...     write_count = fp.write(u'café')
...
>>> write_count
4L
>>> with io.open('cafe.txt', 'r', encoding='utf_8') as fp:
...     text = fp.read()
...
>>> text
u'caf\xe9'
```

```
>>> with io.open('cafe.txt', 'wb') as fp:
...     write_count = fp.write('café')
...
>>> write_count
5L
>>> with io.open('cafe.txt', 'rb') as fp:
...     octets = fp.read()
...
>>> octets
'caf\xc3\xa9'
```

- **io.open()** also handles bytes
 - mode 'b'

FAQ: How to find out the encoding of a file?

- Some files have an encoding header
 - HTML, XML, *some* database dumps
- Otherwise, you must be told. Ask!
- If you can't ask, try the **Chardet** package
 - not 100% safe, but pretty smart
 - uses statistics and heuristics
 - includes a **chardetect** command-line tool

Unicode solutions

	0D0	0D1	0D2	0D3	0D4	0D5	0D6	0D7
0	ശ്വേ	ര	ര	ീ		ജ	ധ	
1	ഠ	ഡ	റ	ു		ആ	ന	
2	ം	ഒ	ല	ു		ഈ	ൺ	
3	ഃ	ഓ	ണ	ള	ു	ഈ	ൺ	
4		ഔ	ത	ഴ	ു		ൽ	
5	അ	ക	ഥ	വ			ൺ	
6	ആ	ഖ	ഭ	ശ	ൈ	ഠ		
7	ഇ	ഗ	ധ	ഷ	ഌ	ൌ	ഘ	
8	ഈ	ഘ	ന	സ	ൈ	ഠ		
9	ഉ	ഒ		ഹ			ന	ൺ
A	ഈ	ച	പ		ഌ	ൌ	ർ	ൺ
B	ഋ	ൠ	ഫ		ഌ	ൌ	ർ	ൺ
C	ഠ	ജ	ബ		ൌ		ന	ർ
D		ത്വ	ഭ	്	്		ഈ	ൽ
E	എ	ണ	മ	ാ			വ	ശ
F	ഏ	ട	യ	ി			ൺ	ക

Combining characters

- Latin character accents and other diacritical marks can be written as separate characters

```
>>> s1 = 'café'
>>> s2 = 'cafe\u0301'
>>> import unicodedata
>>> unicodedata.name(s2[-1])
'COMBINING ACUTE ACCENT'
>>> print(s1, s2)
café café
>>> len(s1), len(s2)
(4, 5)
>>> list(s1), list(s2)
(['c', 'a', 'f', 'é'], ['c', 'a', 'f', 'e', ''])
>>> s1 == s2
False
```



Normalization

- Composing or decomposing all characters
 - Optional: replacing compatibility characters
- Normalization forms: NFC, NFKC, NFD, NFKD

```
>>> from unicodedata import normalize
>>> def nfc_equal(str1, str2):
...     return normalize('NFC', str1) == normalize('NFC', str2)
...
>>> s1 = 'café'
>>> s2 = 'cafe\u0301'
>>> s1 == s2
False
>>> nfc_equal(s1, s2)
True
```

A blue starburst shape with the text 'Py3' in yellow.

Case folding

- Standard character substitutions

```
>>> def fold_equal(str1, str2):  
...     return (normalize('NFC', str1).casefold() ==  
...             normalize('NFC', str2).casefold())  
...  
>>> s3 = 'Straße'  
>>> s4 = 'strasse'  
>>> s3 == s4  
False  
>>> nfc_equal(s3, s4)  
False  
>>> fold_equal(s3, s4)  
True  
>>> fold_equal(s1, s2)  
True  
>>> fold_equal('A', 'a')  
True
```

A blue starburst shape with the text "Py3" in yellow.

2.7 gotcha:
unicode.casefold()
not implemented

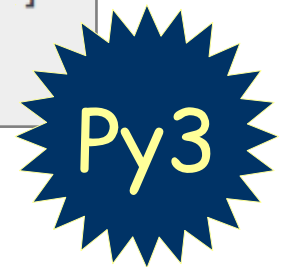
Unicode sorting

- By default, text ordering uses the code point values of the characters
 - this is not what humans expect
 - in Portuguese, accents and diacritics are tiebreakers only

```
>>> fruits = ['caju', 'atemoia', 'cajá', 'açaí', 'acerola']  
>>> sorted(fruits)  
['acerola', 'atemoia', 'açaí', 'caju', 'cajá']
```

wrong: açaí
should be first

wrong: caju
should be last



Unicode sorting

- The standard library solution requires use of the **locale** module and a suitable locale available in the OS
 - only main program should set locale, and only at start-up
 - desired locale is not always available...

```
>>> import locale
>>> locale.setlocale(locale.LC_COLLATE, 'pt_BR.UTF-8')
'pt_BR.UTF-8'
>>> fruits = ['caju', 'atemoia', 'cajá', 'açaí', 'acerola']
>>> sorted_fruits = sorted(fruits, key=locale.strxfrm)
>>> sorted_fruits
['açaí', 'acerola', 'atemoia', 'cajá', 'caju']
```



Unicode sorting

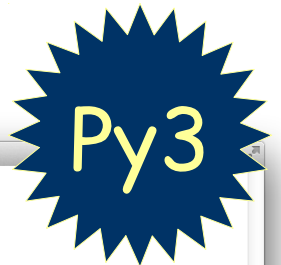
- James Tauber's PyUCA is a Python 3 implementation of UCA (Unicode Collation Algorithm)
 - locale independent!
 - designed to work with many languages
 - <https://pypi.python.org/pypi/pyuca/>

```
>>> import pyuca
>>> coll = pyuca.Collator()
>>> fruits = ['caju', 'atemoia', 'cajá', 'açaí', 'acerola']
>>> sorted_fruits = sorted(fruits, key=coll.sort_key)
>>> sorted_fruits
['açaí', 'acerola', 'atemoia', 'cajá', 'caju']
```



Unicode database

- Metadata about each Unicode character
 - name, numeric value, category etc.
 - standard library: **unicodedata** (Python 2 and Python 3)



```
$ python3 numerics_demo.py
U+0031  1      re_dig isdig  isnum  1.00  DIGIT ONE
U+00bc  ¼      -      -      isnum  0.25  VULGAR FRACTION ONE QUARTER
U+00b2  ²      -      isdig  isnum  2.00  SUPERSCRIPT TWO
U+0969  ३      re_dig isdig  isnum  3.00  DEVANAGARI DIGIT THREE
U+136b  ፫      -      isdig  isnum  3.00  ETHIOPIC DIGIT THREE
U+216b  XII     -      -      isnum  12.00  ROMAN NUMERAL TWELVE
U+2466  ⑦      -      isdig  isnum  7.00  CIRCLED DIGIT SEVEN
U+2480  ⑬      -      -      isnum  13.00  PARENTHEZIZED NUMBER THIRTEEN
U+3285  ⑥      -      -      isnum  6.00  CIRCLED IDEOGRAPH SIX
$
```

numeric values!

Unicode database

```
$ python3 numerics_demo.py
U+0031      1      re_dig isdig  isnum  1.00  DIGIT ONE
U+00bc      ¼      -      -      isnum  0.25  VULGAR FRACTION ONE QUARTER
U+00b2      ²      -      isdig  isnum  2.00  SUPERSCRIPT TWO
U+0969      ३      re_dig isdig  isnum  3.00  DEVANAGARI DIGIT THREE
U+136b      ፫      -      isdig  isnum  3.00  ETHIOPIC DIGIT THREE
U+216b      XII    -      -      isnum  12.00 ROMAN NUMERAL TWELVE
U+2466      ⑦      -      isdig  isnum  7.00  CIRCLED DIGIT SEVEN
U+2480      (13)   -      -      -
U+3285      ⑥      -      -      -
$
```

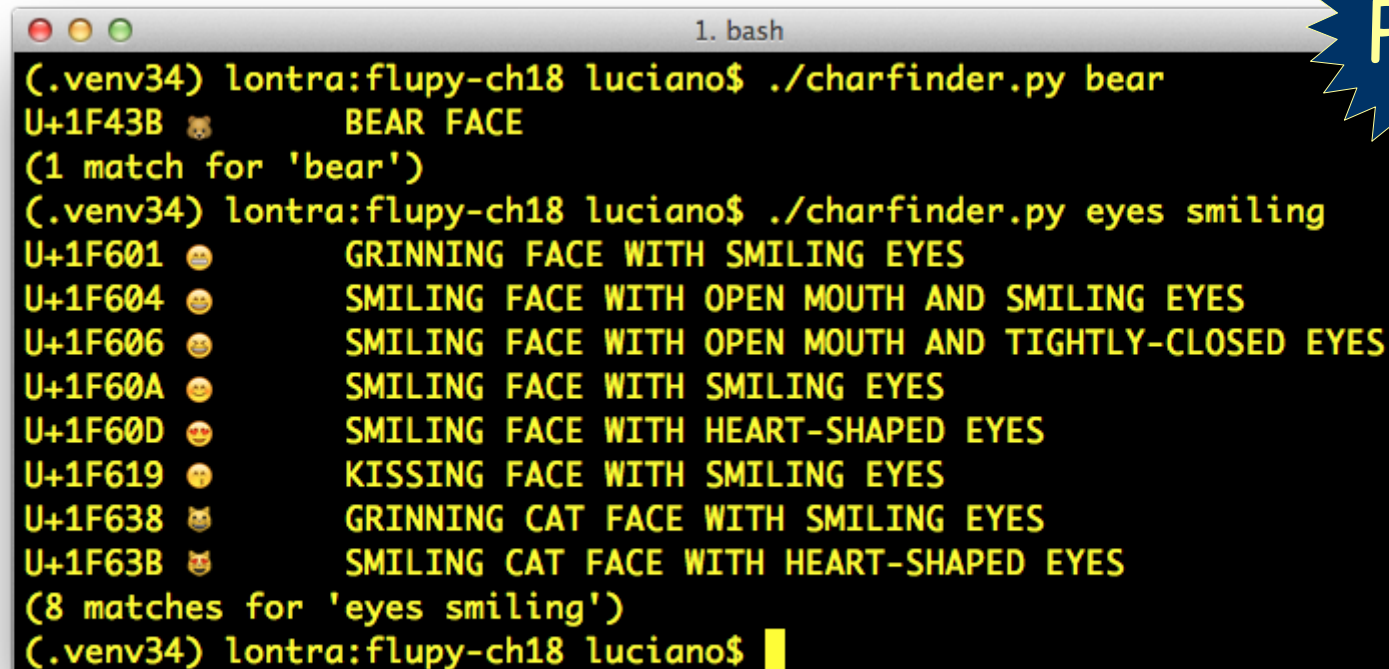
```
1 import unicodedata
2 import re
3
4 re_digit = re.compile(r'\d')
5
6 sample = '1\xbc\xb2\u0969\u136b\u216b\u2466\u2480\u3285'
7
8 for char in sample:
9     print('U+%04x' % ord(char),                # <A>
10         char.center(6),                        # <B>
11         're_digit' if re_digit.match(char) else '-', # <C>
12         'isdig' if char.isdigit() else '-',      # <D>
13         'isnum' if char.isnumeric() else '-',    # <E>
14         format(unicodedata.numeric(char), '5.2f'), # <F>
15         unicodedata.name(char),                  # <G>
16         sep='\t')
17
```

Py3

Characters: 578 - Words: 57

flupy-ch18/charfinder.py

- Command-line utility to search for characters by words in the official name
 - e.g. “cat face”, “black chess”...

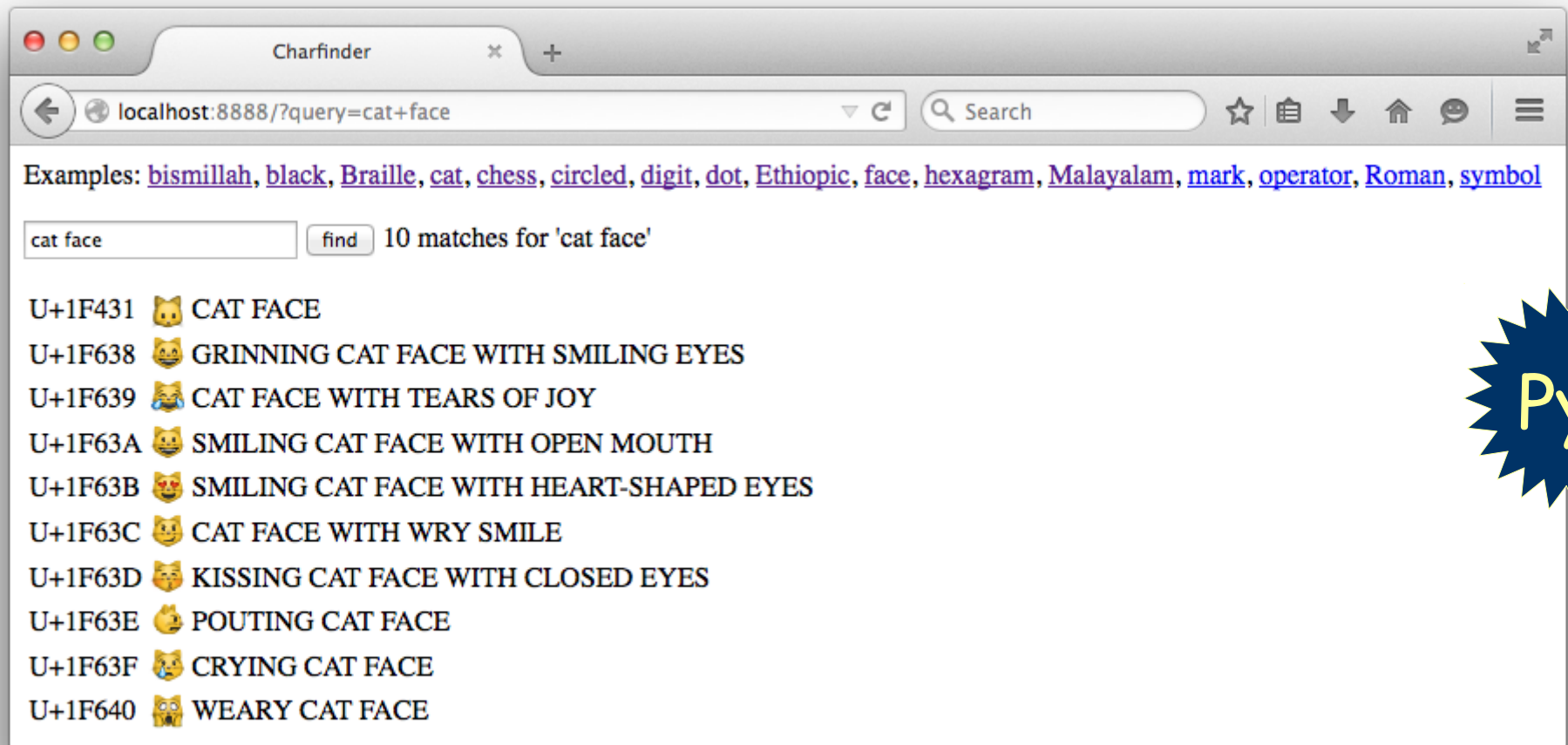


```
1. bash
(.venv34) lontra:flupy-ch18 luciano$ ./charfinder.py bear
U+1F43B 🐻 BEAR FACE
(1 match for 'bear')
(.venv34) lontra:flupy-ch18 luciano$ ./charfinder.py eyes smiling
U+1F601 😄 GRINNING FACE WITH SMILING EYES
U+1F604 😁 SMILING FACE WITH OPEN MOUTH AND SMILING EYES
U+1F606 😊 SMILING FACE WITH OPEN MOUTH AND TIGHTLY-CLOSED EYES
U+1F60A 😇 SMILING FACE WITH SMILING EYES
U+1F60D 🥰 SMILING FACE WITH HEART-SHAPED EYES
U+1F619 😗 KISSING FACE WITH SMILING EYES
U+1F638 😸 GRINNING CAT FACE WITH SMILING EYES
U+1F63B 😺 SMILING CAT FACE WITH HEART-SHAPED EYES
(8 matches for 'eyes smiling')
(.venv34) lontra:flupy-ch18 luciano$
```

Py3

flupy-ch18/http_charfinder.py

- HTTP and Telnet servers used to illustrate asyncio programming (*Fluent Python*, ch. 18)



¿Preguntas?

- More answers:
 - Python Unicode HOWTO
 - for Python 2
 - for Python 3
 - *Fluent Python*, chapter 4
 - Twitter: @ramalhoorg

	1F60	1F61	1F62	1F63	1F64
0	 1F600	 1F601	 1F602	 1F603	 1F604
1	 1F601	 1F601	 1F601	 1F601	 1F601
2	 1F602	 1F602	 1F602	 1F602	 1F602
3	 1F603	 1F603	 1F603	 1F603	 1F603
4	 1F604	 1F604	 1F604	 1F604	 1F604
5	 1F605	 1F605	 1F605	 1F605	 1F605
6	 1F606	 1F606	 1F606	 1F606	 1F606
7	 1F607	 1F607	 1F607	 1F607	 1F607
8	 1F608	 1F608	 1F608	 1F608	 1F608
9	 1F609	 1F609	 1F609	 1F609	 1F609
A	 1F60A	 1F60A	 1F60A	 1F60A	 1F60A
B	 1F60B	 1F60B	 1F60B	 1F60B	 1F60B
C	 1F60C	 1F60C	 1F60C	 1F60C	 1F60C
D	 1F60D	 1F60D	 1F60D	 1F60D	 1F60D
E	 1F60E	 1F60E	 1F60E	 1F60E	 1F60E
F	 1F60F	 1F60F	 1F60F	 1F60F	 1F60F