Name: _
Name: Miroslav ediv
Name: Miroslav ?edivý
Name: Miroslav Čedivá
Name: Miroslav Medivý
Name: Miroslav $\text{rediv}^2$
Name: Miroslav Čedivý
Name: Miroslav Šedivý
Name: Miroslav Šedivý

Your Name Is Invalid!
Miroslav Šedivý

[ˈmɪrɔslovaː ˈʃɛdɪvɪ]  
Miroslav Šedivý
Miroslav Šedivý

[ˈmiroːslav ˈʃɛdɪvːɨ]
Miroslav Šedivý
Your Name Is Invalid!
Your Name Is Invalid!

- Names in Python
  - strings/bytes
  - encoding
  - normalizing
  - case folding
  - sorting
  - regular expressions
Your Name Is Invalid!

- Names in Python
  - strings/bytes
  - encoding
  - normalizing
  - case folding
  - sorting
  - regular expressions
- Names on the web
  - prefix/first/middle/last/suffix names
  - allowed characters
strings/bytes in Python 3

- **str** — 1M+ code points: working memory
- **bytes** — 256 bytes: file/network
strings/bytes in Python 3

- **str** — 1M+ code points: working memory
- **bytes** — 256 bytes: file/network

```python
>>> "Chuck Norris".encode() == b"Chuck Norris"
>>> "Chuck Norris" == b"Chuck Norris".decode()
```
strings/bytes in Python 3

- **str** — 1M+ code points: working memory
- **bytes** — 256 bytes: file/network

```python
>>> "Chuck Norris".encode() == b"Chuck Norris"
>>> "Chuck Norris" == b"Chuck Norris".decode()
```

```python
>>> "Chuck Norris".encode() == b"Chuck Norris"
123456789012 123456789012
```
strings/bytes in Python 3

- **str** — 1M+ code points: working memory
- **bytes** — 256 bytes: file/network

```python
>>> "Chuck Norris".encode() == b"Chuck Norris"
True

>>> "Chuck Norris" == b"Chuck Norris".decode()
True

>>> "Müller".encode() == b"M\xc3\xbcller"
True
```

123456789012
123456789012

123456
12
3
4567
strings/bytes in Python 3

- `str` — 1M+ code points: working memory
- `bytes` — 256 bytes: file/network

```python
>>> "Chuck Norris".encode() == b"Chuck Norris"
>>> "Chuck Norris" == b"Chuck Norris".decode()

>>> "Chuck Norris".encode() == b"Chuck Norris"
123456789012 123456789012

>>> "Müller".encode() == b"M\xc3\xbcller"
123456 12 3 4567

>>> "你好".encode() == b"\xe4\xbd\xa0\xe5\xa5\xb"n1 2 1 2 3 4 5 6
>>> "Chuck Norris".encode("ascii") == b"Chuck Norris"
123456789012 123456789012
>>> "Chuck Norris".encode("ascii") == b"Chuck Norris"
123456789012
123456789012

>>> "Müller".encode("latin1") == b"M\xe4ller"
123456
12
3456
>>> "Chuck Norris".encode("ascii") == b"Chuck Norris"
123456789012
123456789012

>>> "Müller".encode("latin1") == b"M\xfcller"
123456
12
3456

>>> "Šedivý".encode("latin2") == b"\xa9ediv\xfdf"
123456
1
23456
Ruecksendung an:

Miroslav Šedivý
>>> "Šedivý".encode("latin2") == b"\xa9ediv\xfd"
123456  
1  23456
>>> "Šedivý".encode("latin2")
123456

== b"\xa9ediv\xf9"
1  23456

>>> "Šedivý".encode("latin1")
UnicodeEncodeError: 'latin-1' codec can't encode character '\u0160' in position 0: ordinal not in range(256)
>>> "Šedivý".encode("latin2")
  b"\xa9ediv\xfd"

>>> "Šedivý".encode("latin1")
UnicodeEncodeError: 'latin-1' codec can't encode character 'Š' in position 0: ordinal not in range(256)

>>> "Šedivý".encode("latin1", errors="replace")
  b"?ediv\xfd"
```python
>>> codecs.register_error("replace_randomly", lambda exc: (random.choice("1234567890"), exc.start + 1))
>>> "Šedivý".encode('latin1', errors='replace_randomly')
b'Sediv\xfd'
```
>>> codecs.register_error("replace_randomly", lambda exc: (random.choice("1234567890"), exc.start + 1))
>>> "Šedivý".encode('latin1', errors='replace_randomly')
b'5edivxfd'
>>> codecs.register_error("replace_randomly", lambda exc: (random.choice("1234567890"), exc.start + 1))
>>> "Šedivý".encode('latin1', errors='replace_randomly')
b'Šedivý'

Herr Miroslav edivy

Miroslav edivy
```python
codecs.register_error("replace_randomly", lambda exc: (random.choice("1234567890"), exc.start + 1))
>>> "Šedivý".encode('latin1', errors='replace_randomly')
b'5ediv\xfd'
```
“You can only enter letters in the «Adult's Last Name» field.”
What is a “letter”?

šedivý = "Šedivý"
What is a “letter”?

šedivý = "Šedivý"

```python
import unicodedata

for char in "aAÄÅßšš.☺":
    print(char,
          unicodedata.category(char),
          unicodedata.name(char))
```

What is a “letter”?

šedivý = "Šedivý"

```python
import unicodedata
for char in "aAäÄßšš.☺":
    print(char,
          unicodedata.category(char),
          unicodedata.name(char))
```

- a Ll LATIN SMALL LETTER A
- A Lu LATIN CAPITAL LETTER A
- ä Ll LATIN SMALL LETTER A WITH DIAERESIS
- Ä Lu LATIN CAPITAL LETTER A WITH DIAERESIS
- ß Ll LATIN SMALL LETTER SHARP S
- ß Æ Lu LATIN CAPITAL LETTER SHARP S
- š Ll LATIN SMALL LETTER S WITH CARON
- Š Lu LATIN CAPITAL LETTER S WITH CARON
- Zs SPACE
- . Po FULL STOP
- © So WHITE SMILING FACE
Cc  Control
Cf  Format
Co  Private Use
Cs  Surrrogate
Ll  Lowercase Letter
Lm  Modifier Letter
Lo  Other Letter
Lt  Titlecase Letter
Lu  Uppercase Letter
Mc  Spacing Mark
Me  Enclosing Mark
Mn  Nonspacing Mark
Nd  Decimal Number
Nl  Letter Number
No  Other Number
Pc  Connector Punctuation
Pd  Dash Punctuation
Pe  Close Punctuation
Pf  Final Punctuation
Pi  Initial Punctuation
Po  Other Punctuation
Ps  Open Punctuation
Sc  Currency Symbol
Sk  Modifier Symbol
Sm  Math Symbol
So  Other Symbol
Zl  Line
Zp  Paragraph
Zs  Space Separator
# Character maps

<table>
<thead>
<tr>
<th>Script</th>
<th>Character Table</th>
<th>Character Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kannada</td>
<td>Ꙗ Ꙏ ꙓ Ꙑ Ꙛ Ꙗ</td>
<td>Ꙗ Ꙏ Ꙓ Ꙑ Ꙛ Ꙗ</td>
</tr>
<tr>
<td>Katakana</td>
<td>𛄁 𛄂 𛄃 𛄄 𛄅 𛄆</td>
<td>𛄁 𛄂 𛄃 𛄄 𛄅 𛄆</td>
</tr>
<tr>
<td>Kayah Li</td>
<td>꯬ ꯦ ꯧ ꯨ ꯩ ꯪ</td>
<td>꯬ ꯦ ꯧ ꯨ ꯩ ꯪ</td>
</tr>
<tr>
<td>Kharoshthi</td>
<td>가 각 갂 갃 간 갅</td>
<td>가 각 갂 갃 간 갅</td>
</tr>
<tr>
<td>Khmer</td>
<td>១ ២ ៣ ៤ ៥ ៦</td>
<td>១ ២ ៣ ៤ ៥ ៦</td>
</tr>
<tr>
<td>Khojki</td>
<td>꜠ ꜡ Ꜣ ꜣ Ꜥ ꜥ</td>
<td>꜠ ꜡ Ꜣ ꜣ Ꜥ ꜥ</td>
</tr>
</tbody>
</table>

Text to copy: U+0160 LATIN CAPITAL LETTER S WITH CARON
Case folding

"a ä š".upper() == "A Ä Š"
"a ä š" == "A Ä Š".lower()
Case folding

"a ä š".upper() == "A Ä Š"
"a ä š" == "A Ä Š".lower()

"ß".upper() == "SS"
Case folding

"a ä š".upper() == "A Ä Š"
"a ä š" == "A Ä Š".lower()

"ẞ".upper() == "SS"
"ẞ" == "ẞ".lower()
Case folding

```
"a ä š".upper() == "A Ä Š"
"a ä š" == "A Ä Š".lower()

"ß".upper() == "SS"

"ß" == "ß".lower()

"î".upper() == "Î"
```
Case folding

"a ä š".upper() == "A Ä Š"
"a ä š"   == "A Ä Š".lower()

"ß".upper() == "SS"
"ß"   == "ß".lower()

"ı".upper() == "İ"

i→İ, ı→I
Case folding

"a ä š".upper() == "A Ä Š"
"a ä š"   == "A Ä Š".lower()

"ß".upper()  == "SS"

"ß"        == "ẞ".lower()

"ı".upper()  == "İ"

ı → İ, ı → I

>>> import icu  # International Components for Unicode
>>> tr = icu.Locale('tr')
>>> str(icu.UnicodeString('ı').toStringUpper(tr))
'İ'
Normalize

```python
>>> word1 = 'Süß'
>>> word2 = unicodedata.normalize('NFD', word)
```
### Normalize

```python
globals().update(locals())

>>> word1 = 'Süß'
>>> word2 = unicodedata.normalize('NFD', word)
```

<table>
<thead>
<tr>
<th>Word 1</th>
<th>Code</th>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>83</td>
<td>S</td>
<td>LATIN CAPITAL LETTER S</td>
</tr>
<tr>
<td>ü</td>
<td>252</td>
<td>Ll</td>
<td>LATIN SMALL LETTER U WITH DIAERESIS</td>
</tr>
<tr>
<td>ß</td>
<td>223</td>
<td>Ll</td>
<td>LATIN SMALL LETTER SHARP S</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word 2</th>
<th>Code</th>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>83</td>
<td>S</td>
<td>LATIN CAPITAL LETTER S</td>
</tr>
<tr>
<td>u</td>
<td>117</td>
<td>Ll</td>
<td>LATIN SMALL LETTER U</td>
</tr>
<tr>
<td>¨</td>
<td>776</td>
<td>Mn</td>
<td>COMBINING DIAERESIS</td>
</tr>
<tr>
<td>ß</td>
<td>223</td>
<td>Ll</td>
<td>LATIN SMALL LETTER SHARP S</td>
</tr>
</tbody>
</table>
You can't parse [X]HTML with regex. Because HTML can't be parsed by regex. Regex is not a tool that can be used to correctly parse HTML. As I have answered in HTML-and-regex questions here so many times before, the use of regex will not allow you to consume HTML. Regular expressions are a tool that is insufficiently sophisticated to understand the constructs employed by HTML. HTML is not a regular language and hence cannot be parsed by regular expressions. Regexp queries are not equipped to breakdown HTML into its meaningful parts. so many times but it is not getting to me. Even enhanced irregular regular expressions as used by Perl are not up to the task of parsing HTML. You will never make me crack. HTML is a language of sufficient complexity that it cannot be parsed by regular expressions. Even Jon Skeet cannot parse HTML using regular expressions. Every time you attempt to parse HTML with regular expressions, the unholy child weeps the blood of virgins, and Russian hackers pwn your webapp. Pars HTML with regex summons tainted souls into the realm of the living. HTML and regex go together like love, marriage, and ritual infanticide. The 〈center〉 cannot hold it is too late. The force of regex and HTML together in the same conceptual space will destroy your mind like so much watery putty. If you parse HTML with regex you are giving in to them and their blasphemous ways which doom us all to inhuman toil for the One whose Name cannot be expressed in the Basic Multilingual Plane. he comes. HTML plus-regexp will liquify the nerves of the sentient whilst you observe, your psyche withering in the onslaught of horror. Regex-based HTML parsers are the cancer that is killing StackOverflow. it is too late it is too late we cannot be saved the transgression of a child ensures regex will consume all living tissue (except for HTML which it cannot, as previously prophesied) dear lord help us how can anyone survive this scourge using regex to parse HTML has doomed humanity to an eternity of dread torture and security holes using regex as a tool to process HTML establishes a breach between this world and the dread realm of corrupt entities (like SGML entities, but more corrupt) a mere glimpse of the world of reg...

ex parsers for HTML will instantly transport a programmer's consciousness into a world of ceaseless screaming. he comes. the pestilent stinky regex infection will devour your HTML parser. application and existence for all time like Visual Basic only worse he comes he comes do not fight he comes his unholy radiance destroying all enlightenment, HTML tags leaking from your eyes like liquid pain, the song of regular expression parsing will extinguish the voices of mortal man from the sphere I can see it can you see it is beautiful the final snuffing of the lies of Man ALL IS LOST ALL IS LOST the pony he comes he comes he comes his unholy grace permeates all MY FACE IN MY FACE in god no NO NOCO NO stop the anguish, the not real ZALGO TO THE PONY, HE COMES

Have you tried using an XML parser instead?
Sorting (“alphabetic”)
Sorting ("alphabetic")

```python
>>> ''.join(sorted('aAoaouÅÄöÖüÜßẞšŠ'))
'AouaouÅÄßßšš'

>>> import locale
>>> locale.setlocale(locale.LC_ALL, 'de_DE.UTF-8')
>>> ''.join(sorted('abABäÄ', key=locale.strxfrm))
'aAäÄbB'
```
Sorting (“alphabetic”)

```python
>>> ''.join(sorted('aAoouÄöüüßẞš'))
'AOUaouÄÜßÖÜŠšẞ'

>>> import locale
>>> locale.setlocale(locale.LC_ALL, 'de_DE.UTF-8')
>>> ''.join(sorted('abABäÄ', key=locale.strxfrm))
'aAäAbB'

>>> locale.setlocale(locale.LC_ALL, 'sv_SE.UTF-8')
>>> ''.join(sorted('abABäÄ', key=locale.strxfrm))
'aAbBäÄ'
```
Sorting ("alphabetic")

```python
>>> ''.join(sorted('aAouUäÖüßẞšŠ'))
'AOUaouÄÜßöüŠšẞ'

>>> import locale
>>> locale.setlocale(locale.LC_ALL, 'de_DE.UTF-8')
>>> ''.join(sorted('abABäÄ', key=locale.strxfrm))
'aAäAbB'

>>> locale.setlocale(locale.LC_ALL, 'sv_SE.UTF-8')
>>> ''.join(sorted('abABäÄ', key=locale.strxfrm))
'aAbBäÄ'
```

- Hungarian: cékla, cvikli, csipős
Sorting ("alphabetic")

```python
global string as 'aAoOuÅÄöÖüÜßẞšŠ'

- Join sorted characters:
  ```python
global string as 'aAoOuÅÄöÖüÜßẞšŠ'
    ''.join(sorted(global string))
  ```
  ```
  'AOUaouÄÖÜßäöüŠšẞ'
  ```

- Import locale:
  ```
  import locale
  ```

- Set locale:
  ```
  locale.setlocale(locale.LC_ALL, 'de_DE.UTF-8')
  ```

- Sort string with locale:
  ```
  ''.join(sorted('abABäÄ', key=locale.strxfrm))
  ```
  ```
  'aAäAbB'
  ```

- Additional locale:
  ```
  locale.setlocale(locale.LC_ALL, 'sv_SE.UTF-8')
  ```

- Sort string with locale:
  ```
  ''.join(sorted('abABäÄ', key=locale.strxfrm))
  ```
  ```
  'aAbBäÄ'
  ```

- Multilingual examples:
  - Hungarian: cékla, cvikli, csipős
  - Czech & Slovak: c, č, d, ..., h, ch, i, ..., s, š, t
Sorting (“alphabetic”)

```python
>>> ''.join(sorted('aAoOuÄööüüßẞš'))
'AOUaouÄÜßäöüŠšẞ'

>>> import locale

>>> locale.setlocale(locale.LC_ALL, 'de_DE.UTF-8')

>>> ''.join(sorted('abABäÄ', key=locale.strxfrm))
'aAäAbB'

>>> locale.setlocale(locale.LC_ALL, 'sv_SE.UTF-8')

>>> ''.join(sorted('abABäÄ', key=locale.strxfrm))
'aAbBäÄ'

- Hungarian: cékla, cvikli, csipős
- Czech & Slovak: c, č, d, ..., h, ch, i, ..., s, š, t
- French: cote, côté, coté, côté
```
Locale is connected to the process 😊

```python
import locale
locale.setlocale(locale.LC_ALL, 'de_DE.UTF-8')
sorted(words, key=locale.strxfrm))
```
Locale is connected to the process 😞

```python
import locale
locale.setlocale(locale.LC_ALL, 'de_DE.UTF-8')
sorted(words, key=locale.strxfrm))

import icu
collator = icu.Collator.createInstance(icu.Locale('de_DE.UTF-8'))
sorted(words, key=collator.getSortKey)
```
Locale is connected to the process 😞

```python
import locale
locale.setlocale(locale.LC_ALL, 'de_DE.UTF-8')
sorted(words, key=locale.strxfrm))

import icu
collator = icu.Collator.createInstance(icu.Locale('de_DE.UTF-8'))
sorted(words, key=collator.getSortKey)

import pyuca
collator = pyuca.Collator()
sorted(words, key=collator.sort_key)
```
Unicode regular expressions

```python
>>> text = 'München 123'
```
Unicode regular expressions

```python
>>> text = 'München 123'

>>> import re
>>> re.findall(r'[a-zA-Z]+', text)
['M', 'nchen']
```
Unicode regular expressions

```python
>>> text = 'München 123'

>>> import re
>>> re.findall(r'[a-zA-Z]+', text)
['M', 'nchen']

>>> re.findall(r'[\w]+', text)
['München', '123']
```
Unicode regular expressions

```python
>>> text = 'München 123'

>>> import re
>>> re.findall(r'[a-zA-Z]+', text)
['M', 'nchen']

>>> re.findall(r'\w+', text)
['München', '123']

>>> import regex
>>> regex.findall(r'\p{L}+', text)
['München']
```
I came for Python, stayed for the names
I came for Python, stayed for the names

First name: _
Last name: _
I came for Python, stayed for the names

First name: 
Last name: 

First name: 
Middle name: 
Last name: 

eumiro
I came for Python, stayed for the names

First name: 
Last name: 

First name: 
Middle name: 
Last name: 

First name: 
Patronymic surname: 
Matronymic surname: 

I came for Python, stayed for the names

First name: _
Last name: _

First name: _
Middle name: _
Last name: _

First name: _
Patronymic surname: _
Matronymic surname: _

Last name: _
First name: _
I came for Python, stayed for the names

First name:  
Last name:  

First name:  
Middle name:  
Last name:  

First name:  
Patronymic surname:  
Matronymic surname:  

Last name:  
First name:  

Name:  
Number:  
I came for Python, stayed for the names

First name:  
Last name:  

First name:  
Middle name:  
Last name:  

First name:  
Patronymic surname:  
Matronymic surname:  

Last name:  
First name:  

Name:  
Number:  

Name:  

I came for Python, stayed for the names

First name: _
Last name: _

First name: _
Middle name: _
Last name: _

First name: _
Patronymic surname: _
Matronymic surname: _

Last name: _
First name: _

Name: _
Number: _

von und zu, de/d', van/de/ter, z/ze, di/de/da/degli/dalla, of, …
I came for Python, stayed for the names

First name: _
Last name: _

First name: _
Middle name: _
Last name: _

First name: _
Patronymic surname: _
Matronymic surname: _

Last name: _
First name: _

Name: _
Number: _

Name: _

von und zu, de/d', van/de/ter, z/ze, di/de/da/dagli/dalla, of, …

Dr., PhD.,
Use one field for the full name

Full name:
How should we call you:
Use one field for the full name

Full name:              _
How should we call you: _

First name: Mister; middle name: period; last name T.

— Mr. T —

AZ QUOTES
“Please enter characters from the European character set only.”
“Please enter characters from the European character set only.”

“Please enter a full valid name.”
Don't assume anything
Don't assume anything

- don't put random limit on their length
Don't assume anything

- don't put random limit on their length
  - *Pippilotta Delicatessa Windowshade Mackrelmint Ephraim's Daughter Longstocking*
Don't assume anything

- don't put random limit on their length
  - *Pippilotta Delicatessa Windowshade Mackrelmint Ephraim's Daughter Longstocking*
  - *Karl-Theodor Maria Nikolaus Johann Jacob Philipp Franz Joseph Sylvester Buhl-Freiherr von und zu Guttenberg*
Don't assume anything

- don't put random limit on their length
  - Pippilotta Delicatessa Windowshade Mackrelmint Ephraim's Daughter Longstocking
  - Karl-Theodor Maria Nikolaus Johann Jacob Philipp Franz Joseph Sylvester Buhl-Freiherr von und zu Guttenberg
- don't use stop words
Don't assume anything

- don't put random limit on their length
  - *Pippilotta Delicatessa Windowshade Mackrelmint Ephraim's Daughter Longstocking*
  - *Karl-Theodor Maria Nikolaus Johann Jacob Philipp Franz Joseph Sylvester Buhl-Freiherr von und zu Guttenberg*
- don't use stop words
- family members don't have necessarily the same family name (*Šedivý / Šedivá*)
Don't assume anything (II)
Don't assume anything (II)

- different transcription from non-latin alphabets:
Don't assume anything (II)

- different transcription from non-latin alphabets:
  - Чехов → Čechov, Tschechow, Chekhov, Čehov, Tjekhov, Tchekhov, Csehov, Tsjechov, Czechow, ...
Don't assume anything (II)

- different transcription from non-latin alphabets:
  - Чехов → Čechov, Tschechow, Chekhov, Čehov, Tjekhov, Tchekhov, Csehov, Tsjechov, Czechow, ...
  - 毛泽东 → Mao Zedong, Mao Tse-tung, Mao Ce-tung
Don't assume anything (II)

- different transcription from non-latin alphabets:
  - Чехов → Čechov, Tschechow, Chekhov, Čehov, Tjekhov, Tchekhov, Csehov, Tsjechov, Czechow, ...
  - 毛泽东 → Mao Zedong, Mao Tse-tung, Mao Ce-tung

- men change their family names too ("maiden name" / “née”?)
Don't assume anything (II)

- different transcription from non-latin alphabets:
  - Чехов → Čechov, Tschechow, Chekhov, Čehov, Tjekhov, Tchekhov, Csehov, Tsjechov, Czechow, ...
  - 毛泽东 → Mao Zedong, Mao Tse-tung, Mao Ce-tung

- men change their family names too ("maiden name" / "née")?
- one-letter name is probably not an initial (Benoît B. Mandelbrot)
Don't assume anything (II)

- different transcription from non-latin alphabets:
  - Чехов → Čechov, Tschechow, Chekhov, Čehov, Tjekhov, Tchekhov, Csehov, Tsjechov, Czechow, ...
  - 毛泽东 → Mao Zedong, Mao Tse-tung, Mao Ce-tung

- men change their family names too (“maiden name” / “née”?)

- one-letter name is probably not an initial (*Benoît B. Mandelbrot*)
- all printable (codepoint > 32) are probably fine, 😃
Christopher Null: Hello, I'm Mr. Null. My Name Makes Me Invisible to Computers
Christopher Null: Hello, I'm Mr. Null. My Name Makes Me Invisible to Computers
xkcd.com/327
Addresses: streets, cities
Addresses: streets, cities

Einbahnstraße
Addresses: streets, cities

Hauptstraße != Hauptstraße
Hint question*

What is your mother's maiden name

Answer (at least 6 characters)*

Smith
Grzegorz Brzęczyszczykiewicz.
Chrząszczycyrzewoszczyce.
Powiat Łękołody.
Country: Iceland
City: Búðardalur
Name: a horse farm with an Icelandic/Danish couple and 3 kids and a lot of sheep!

The Danish woman works in a supermarket in Búðardalur.

Takk fyrir!
Patrick McKenzie: Falsehoods Programmers Believe About Names


- People have exactly one canonical full name.
- People have exactly one full name which they go by.
- People have, at this point in time, exactly one canonical full name.
- People have, at this point in time, one full name which they go by.
- People have exactly N names, for any value of N.
- People’s names fit within a certain defined amount of space.
- People’s names do not change.
- People’s names change, but only at a certain enumerated set of events.
Patrick McKenzie: *Falsehoods Programmers Believe About Names*


- People’s names are written in ASCII.
- People’s names are written in any single character set.
- People’s names are all mapped in Unicode code points.
- People’s names are case sensitive.
- People’s names are case insensitive.
- People’s names sometimes have prefixes or suffixes, but you can safely ignore those.
- People’s names do not contain numbers.
- People’s names are not written in ALL CAPS.
People’s names are not written in all lower case letters.
People’s names have an order to them. Picking any ordering scheme will automatically result in consistent ordering among all systems, as long as both use the same ordering scheme for the same name.
People’s first names and last names are, by necessity, different.
People have last names, family names, or anything else which is shared by folks recognized as their relatives.
People’s names are globally unique.
People’s names are almost globally unique.
Alright alright but surely people’s names are diverse enough such that no million people share the same name.
My system will never have to deal with names from China.
Patrick McKenzie: Falsehoods Programmers Believe About Names


- Or Japan.
- Or Korea.
- Or Ireland, the United Kingdom, the United States, Spain, Mexico, Brazil, Peru, Russia, Sweden, Botswana, South Africa, Trinidad, Haiti, France, or the Klingon Empire, all of which have “weird” naming schemes in common use.
- That Klingon Empire thing was a joke, right?
- Confound your cultural relativism! People in my society, at least, agree on one commonly accepted standard for names.
- There exists an algorithm which transforms names and can be reversed losslessly. (Yes, yes, you can do it if your algorithm returns the input. You get a gold star.)
- I can safely assume that this dictionary of bad words contains no people’s names in it.
- People’s names are assigned at birth.
OK, maybe not at birth, but at least pretty close to birth.
Alright, alright, within a year or so of birth.
Five years?
You’re kidding me, right?
Two different systems containing data about the same person will use the same name for that person.
Two different data entry operators, given a person’s name, will by necessity enter bitwise equivalent strings on any single system, if the system is well-designed.
People whose names break my system are weird outliers. They should have had solid, acceptable names, like 田中太郎.
People have names.
Your Name Is Invalid!
Your Name Is Invalid!

- respect your users' names
Your Name Is Invalid!

- respect your users' names
- don't break the locale: import icu
Your Name Is Invalid!

- respect your users' names
- don't break the locale: import icu
- bytes $\rightarrow$ str (as soon as possible), str $\rightarrow$ bytes (as late as possible)
Your Name Is Invalid!

- respect your users' names
- don't break the locale: `import icu`
- `bytes → str` (as soon as possible), `str → bytes` (as late as possible)
- UTF-8 is cool, Python 3 is cool, be cool too: use Python 3 and UTF-8
Your Name Is Invalid!

- respect your users' names
- don't break the locale: `import icu`
- bytes → str (as soon as possible), str → bytes (as late as possible)
- UTF-8 is cool, Python 3 is cool, be cool too: use Python 3 and UTF-8
- telling the user “Your Name Is Invalid!” is NOT cool → yournameisvalid
Your Name Is Invalid!

- respect your users' names
- don't break the locale: `import icu`
- `bytes → str` (as soon as possible), `str → bytes` (as late as possible)
- UTF-8 is cool, Python 3 is cool, be cool too: use Python 3 and UTF-8
- telling the user “Your Name Is Invalid!” is NOT cool → `yournameisvalid`
- be nice!
Your Name Is Invalid!

- respect your users' names
- don't break the locale: `import icu`
- bytes → str (as soon as possible), str → bytes (as late as possible)
- UTF-8 is cool, Python 3 is cool, be cool too: use Python 3 and UTF-8
- telling the user “Your Name Is Invalid!” is NOT cool → yournameisvalid
- be nice!

Miroslav Šedivý