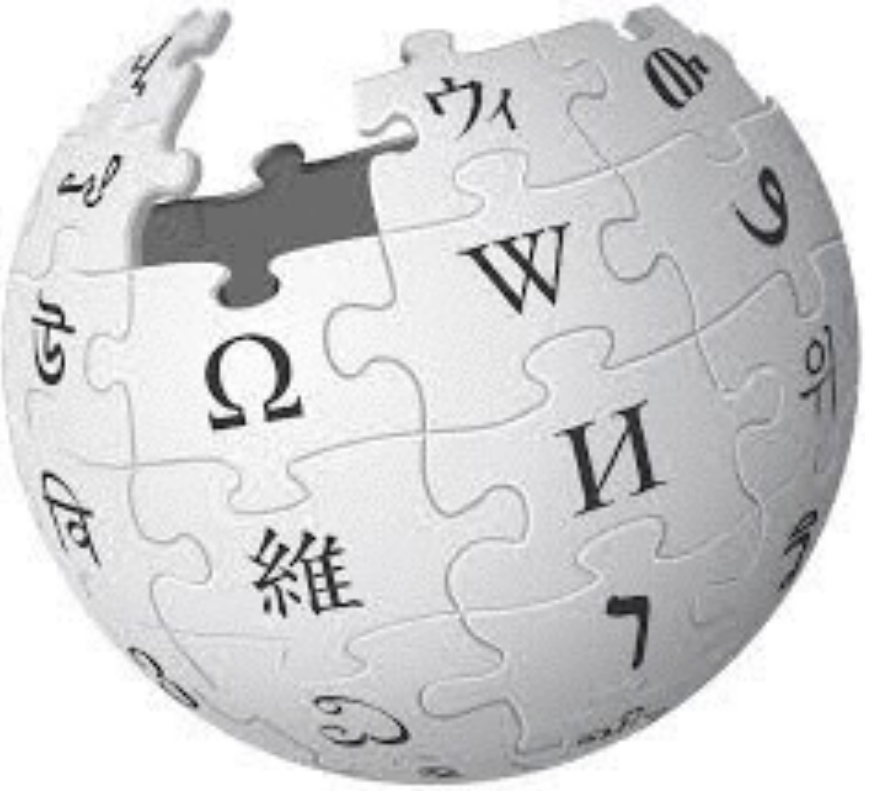
Applied Text Mining in Python

Introduction to Text Mining

Text is Everywhere!

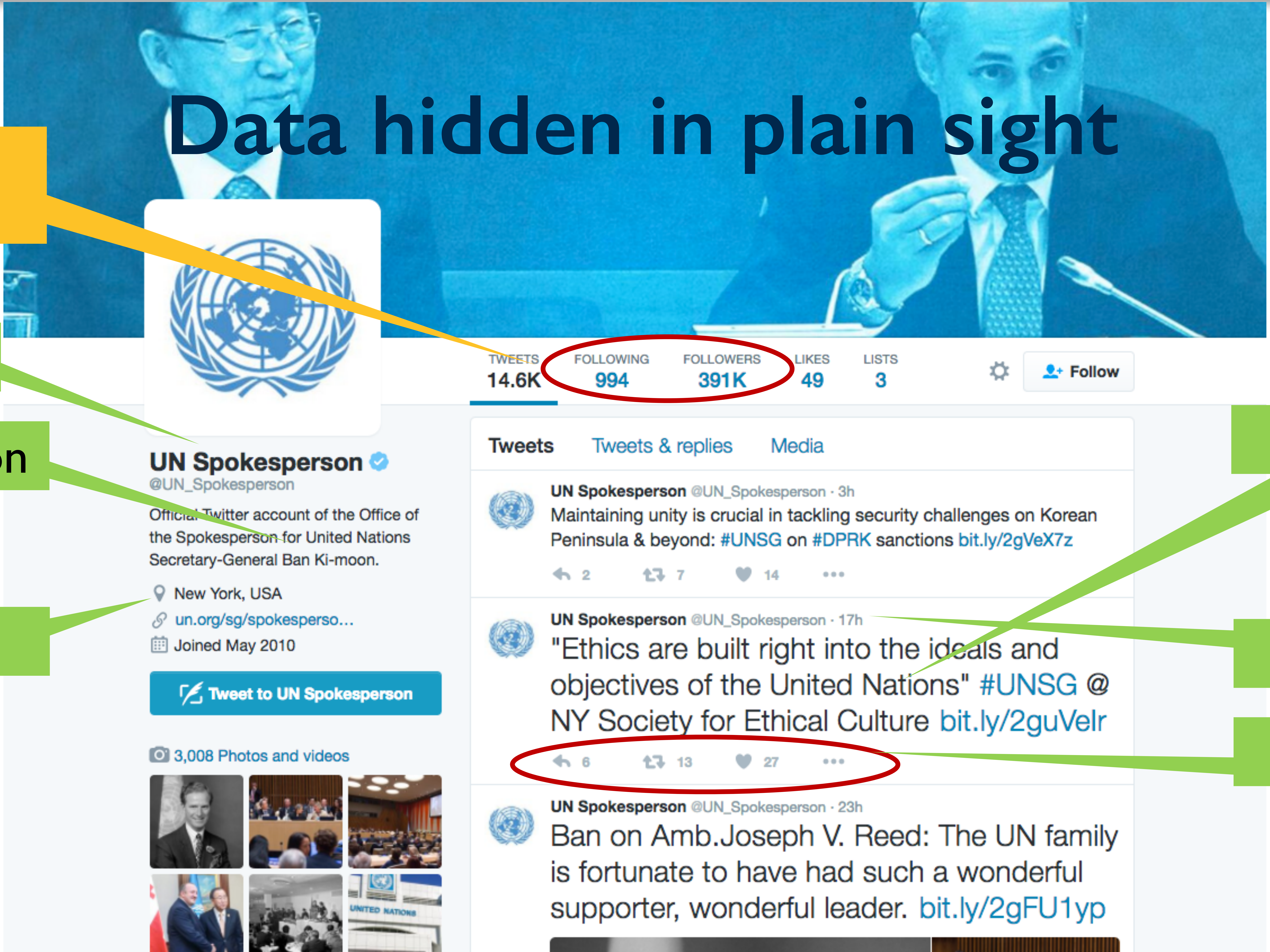
Dec 29, 2004 - The Inkscape developers have set a goal for the release of Inkscape 0.41 for mid-late January. There are some noteworthy new features that would be worth getting out to users, and with some major internal changes that will be taking place soon, a good stable release is needed prior to starting such an undertaking.



Text data is growing fast!

- **Data continues to grow exponentially**
  - Estimated to be 2.5 Exabytes (2.5 million TB) a day
  - Grow to 40 Zettabytes (40 billion TB) by 2020 (50-times that of 2010)
- **Approximately 80% of all data is estimated to be unstructured, text-rich data**
  - >40 million articles (5 million in English) in Wikipedia
  - >4.5 billion Web pages
  - >500 million tweets a day, 200 billion a year
  - >1.5 trillion queries / searches on Google a year

Data hidden in plain sight



TWEETS 14.6K | FOLLOWING 994 | FOLLOWERS 391K | LIKES 49 | LISTS 3

UN Spokesperson
@UN_Spokesperson

Official Twitter account of the Office of the Spokesperson for United Nations Secretary-General Ban Ki-moon.

New York, USA
un.org/sg/spokesperso...
Joined May 2010

UN Spokesperson @UN_Spokesperson · 3h
Maintaining unity is crucial in tackling security challenges on Korean Peninsula & beyond: #UNSG on #DPRK sanctions bit.ly/2gVeX7z
2 7 14

UN Spokesperson @UN_Spokesperson · 17h
"Ethics are built right into the ideals and objectives of the United Nations" #UNSG @ NY Society for Ethical Culture bit.ly/2guVelr
6 13 27

UN Spokesperson @UN_Spokesperson · 23h
Ban on Amb.Joseph V. Reed: The UN family is fortunate to have had such a wonderful supporter, wonderful leader. bit.ly/2gFU1yp

Social network

Author

Description

Location

Tweet
- Topic
- Sentiment

Time

Popularity

So, what can be done with text?

- Parse text
- Find / Identify / Extract relevant information from text
- Classify text documents
- Search for relevant text documents
- Sentiment analysis
- Topic modeling
- …

Applied Text Mining in Python

Handling Text in Python

Primitive constructs in Text

- Sentences / input strings
- Words or Tokens
- Characters
- Document, larger files

And their properties …

Let’s try it out!

```
>>> text1 = "Ethics are built right into the ideals and objectives
of the United Nations "
>>> len(text1)
76
>>> text2 = text1.split(' ')
>>> len(text2)
13
>>> text2
['Ethics', 'are', 'built', 'right', 'into', 'the', 'ideals', 'and',
'objectives', 'of', 'the', 'United', 'Nations', '']
```

Finding specific words

- **Long words: Words that are most than 3 letters long**

```
>>> [w for w in text2 if len(w) > 3]
['Ethics', 'built', 'right', 'into', 'ideals', 'objectives', 'United',
'Nations']
```

- **Capitalized words**

```
>>> [w for w in text2 if w.istitle()]
['Ethics', 'United', 'Nations']
```

- **Words that end with s**

```
>>> [w for w in text2 if w.endswith(’s')]
['Ethics', 'ideals', 'objectives', 'Nations']
```

Finding unique words: using set()

```
>>> text3 = 'To be or not to be'
>>> text4 = text3.split(' ')
>>> len(text4)
6
>>> len(set(text4))
5
>>> set(text4)
set(['not', 'To', 'or', 'to', 'be'])
>>> len(set([w.lower() for w in text4]))
4
>>> set([w.lower() for w in text4])
set(['not', 'to', 'or', 'be']
```

Some word comparison functions …

- s.startswith(t)
- s.endswith(t)
- t in s
- s.isupper(); s.islower(); s.istitle()
- s.isalpha(); s.isdigit(); s.isalnum()

String Operations

- s.lower(); s.upper(); s.titlecase()
- s.split(t)
- s.splitlines()
- s.join(t)
- s.strip(); s.rstrip()
- s.find(t); s.rfind(t)
- s.replace(u, v)

From words to characters

```
>>> text5 = 'ouagadougou'
>>> text6 = text5.split('ou')
>>> text6
['', 'agad', 'g', '']
>>> 'ou'.join(text6)
'ouagadougou’
```

```
>>> text5.split('')
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
ValueError: empty separator
>>> list(text5)
['o', 'u', 'a', 'g', 'a', 'd',
'o', 'u', 'g', 'o', 'u']
>>> [c for c in text5]
['o', 'u', 'a', 'g', 'a', 'd',
'o', 'u', 'g', 'o', 'u']
```

Cleaning Text

```
>>> text8 = '    A quick brown fox jumped over the lazy dog. '
>>> text8.split(' ')
['', '', '\t', 'A', 'quick', 'brown', 'fox', 'jumped', 'over',
'the', 'lazy', 'dog.', '']
>>> text9 = text8.strip()
>>> text9.split(' ')
['A', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy',
'dog.']
```

Changing Text

- **Find and replace**

```
>>> text9
'A quick brown fox jumped over the lazy dog.'
>>> text9.find('o')
10
>>> text9.rfind('o')
40
>>> text9.replace('o', 'O')
'A quick brOwn fOx jumped Over the lazy dOg.'
```

Handling Larger Texts

- **Reading files line by line**

```
>>> f = open('UNDHR.txt', 'r')
>>> f.readline()
'Universal Declaration of Human Rights\n’
```

- **Reading the full file**

```
>>> f.seek(0)
>>> text12 = f.read()
>>> len(text12)
10891
>>> text13 = text12.splitlines()
>>> len(text13)
158
>>> text13[0]
'Universal Declaration of Human Rights'
```

File Operations

- f = open(*filename*, *mode*)
- f.readline(); f.read(); f.read(*n*)
- for *line* in f: doSomething(*line*)
- f.seek(*n*)
- f.write(*message*)
- f.close()
- f.closed

Issues with reading text files

```
>>> f = open('UNDHR.txt', 'r')
>>> text14 = f.readline()
'Universal Declaration of Human Rights\n'
```

- **How do you remove the last newline character?**

```
>>> text14.rstrip()
'Universal Declaration of Human Rights’
```

  - Works also for DOS newlines (`^M`) that shows up as `'\r'` or `'\r\n'`

Take Home Concepts

- Handling text sentences
- Splitting sentences into words, words into characters
- Finding unique words
- Handling text from documents

Applied Text Mining in Python

Regular Expressions

Processing Free-text

```
>>> text10 = '"Ethics are built right into the ideals and
objectives of the United Nations" #UNSG @ NY Society for Ethical
Culture bit.ly/2guVelr @UN @UN_Women'
>>> text11 = text10.split(' ')
>>> text11
['"Ethics', 'are', 'built', 'right', 'into', 'the', 'ideals',
'and', 'objectives', 'of', 'the', 'United', 'Nations"', '#UNSG',
'@', 'NY', 'Society', 'for', 'Ethical', 'Culture', 'bit.ly/
2guVelr', '@UN', '@UN_Women']
```

- How do you find all Hashtags? Callouts?

Finding Specific Words

- **Hashtags**

```
>>> [w for w in text11 if w.startswith('#')]
['#UNSG']
```

- **Callouts**

```
>>> [w for w in text11 if w.startswith('@')]
['@', '@UN', '@UN_Women']
```

⚠

Finding patterns with regular expressions

- Callouts are more than just tokens beginning with '@'

@UN_Spokesperson @katyperry @coursera

- Match *something* after '@'
  - Alphabets
  - Numbers
  - Special symbols like '_'

@[A-Za-z0-9_]+

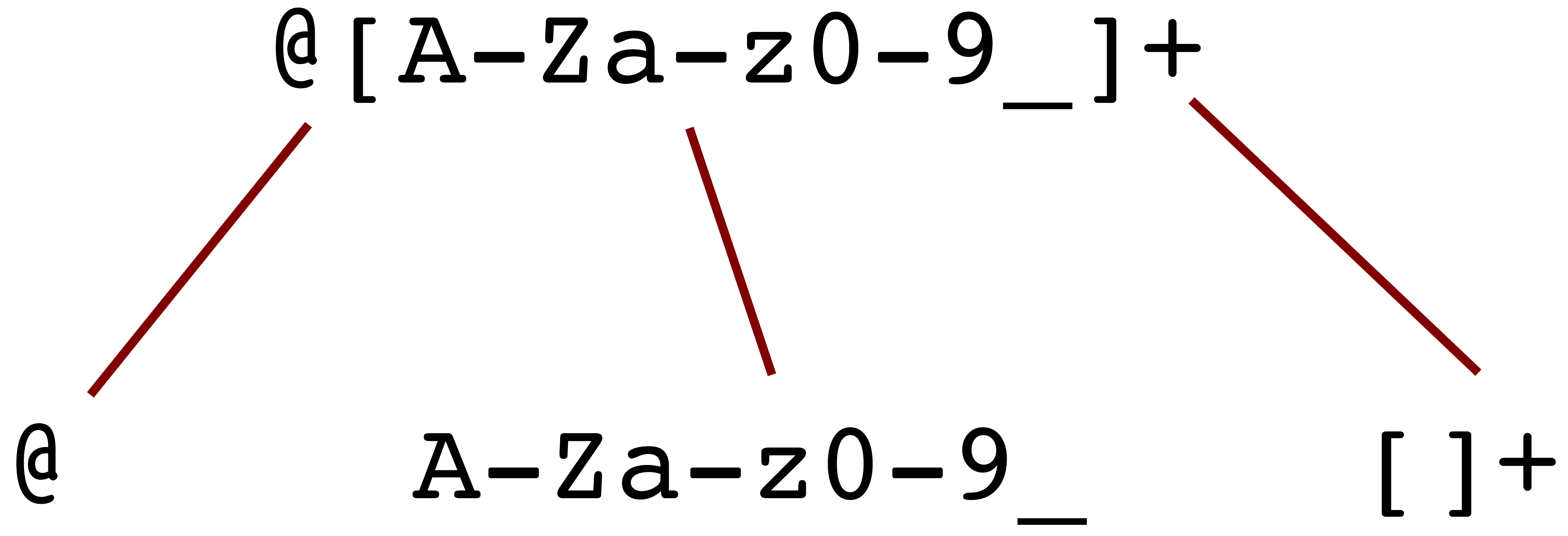
Let’s try it out!

```
>>> text10 = ‘”Ethics are built right into the ideals and objectives of the
United Nations” #UNSG @ NY Society for Ethical Culture bit.ly/2guVelr @UN
@UN_Women’
>>> text11 = text10.split(' ')
>>> [w for w in text11 if w.startswith('@')]
['@', '@UN', '@UN_Women']
```

Import regular expressions first!

```
>>> import re
>>> [w for w in text11 if re.search('@[A-Za-z0-9_]+', w)]
['@UN', '@UN_Women']
```

Parsing the callout regular expression



`@[A-Za-z0-9_]+`

`@`   `A-Za-z0-9_`   `[]+`

- starts with `@`
- followed by any alphabet (upper or lower case), digit, or underscore
- that repeats at least once, but any number of times

Meta-characters: Character matches

. : wildcard, matches a single character

^ : start of a string

$: end of a string

[ ] : matches one of the set of characters within [ ]

[a-z] : matches one of the range of characters a, b, …, z

[^abc] : matches a character that is not a, b, or, c

a|b : matches either a or b, where a and b are strings

( ) : Scoping for operators

\ : Escape character for special characters (\t, \n, \b)

Meta-characters: Character symbols

`\b` : Matches word boundary

`\d` : Any digit, equivalent to `[0-9]`

`\D` : Any non-digit, equivalent to `[^0-9]`

`\s` : Any whitespace, equivalent to `[ \t\n\r\f\v]`

`\S` : Any non-whitespace, equivalent to `[^ \t\n\r\f\v]`

`\w` : Alphanumeric character, equivalent to `[a-zA-Z0-9_]`

`\W` : Non-alphanumeric, equivalent to `[^a-zA-Z0-9_]`

Meta-characters: Repetitions

`*` : matches zero or more occurrences

`+` : matches one or more occurrences

`?` : matches zero or one occurrences

`{n}` : exactly `n` repetitions, $n \geq 0$

`{n,}` : at least `n` repetitions

`{,n}` : at most `n` repetitions

`{m,n}` : at least `m` and at most `n` repetitions

Recall the callout regular expression

```
>>> text10 = ‘”Ethics are built right into the ideals and
objectives of the United Nations” #UNSG @ NY Society for Ethical
Culture bit.ly/2guVelr @UN @UN_Women’
>>> text11 = text10.split(' ')

>>> [w for w in text11 if re.search('@[A-Za-z0-9_]+', w)]
['@UN', '@UN_Women']

>>> [w for w in text11 if re.search('@\w+', w)]
['@UN', '@UN_Women']
```

Let’s look at some more examples!

- Finding specific characters

```
>>> text12 = ‘ouagadougou’

>>> re.findall(r'[aeiou]', text12)
['o', 'u', 'a', 'a', 'o', 'u', 'o', 'u']

>>> re.findall(r'[^aeiou]', text12)
['g', 'd', 'g’]
```

Case study: Regular expression for Dates

- Date variations for 23rd October 2002

```
23-10-2002
23/10/2002
23/10/02
10/23/2002
23 Oct 2002
23 October 2002
Oct 23, 2002
October 23, 2002
```

```
\d{2}[/-]\d{2}[/-]\d{4}
```

Regular Expression for Dates (contd.)

```
>>> dateStr = '23-10-2002\n23/10/2002\n23/10/02\n10/23/2002\n23 Oct 2002\n23
October 2002\nOct 23, 2002\nOctober 23, 2002\n’

>>> re.findall(r'\d{2}[/-]\d{2}[/-]\d{4}', dateStr)
['23-10-2002', '23/10/2002', '10/23/2002']

>>> re.findall(r'\d{2}[/-]\d{2}[/-]\d{2,4}', dateStr)
['23-10-2002', '23/10/2002', '23/10/02', '10/23/2002']

>>> re.findall(r'\d{1,2}[/-]\d{1,2}[/-]\d{2,4}', dateStr)
['23-10-2002', '23/10/2002', '23/10/02', '10/23/2002']
```

```
23-10-2002
23/10/2002
23/10/02
10/23/2002
```

Regex for Dates (contd.)

```
23 Oct 2002
23 October 2002
Oct 23, 2002
October 23, 2002
```

```
>>> re.findall(r'\d{2} (Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)
\d{4}', dateStr)
['Oct']
```

```
>>> re.findall(r'\d{2} (?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)
\d{4}', dateStr)
['23 Oct 2002']
```

```
>>> re.findall(r'\d{2} (?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)[a-
z]* \d{4}', dateStr)
['23 Oct 2002', '23 October 2002']
```

Regex for Dates (contd.)

```
23 Oct 2002
23 October 2002
Oct 23, 2002
October 23, 2002
```

```
>>> re.findall(r'(?:\d{2} )?(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|
Dec)[a-z]* (?:\d{2}, )?\d{4}', dateStr)
['23 Oct 2002', '23 October 2002', 'Oct 23, 2002', 'October 23, 2002']

>>> re.findall(r'(?:\d{1,2} )?(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|
Dec)[a-z]* (?:\d{1,2}, )?\d{4}', dateStr)
['23 Oct 2002', '23 October 2002', 'Oct 23, 2002', 'October 23, 2002']
```

Take Home Concepts

- What are regular expressions?
- Regular expression meta-characters
- Building a regular expression to identify dates

Applied Text Mining in Python

Internationalization

World of Languages

English and ASCII

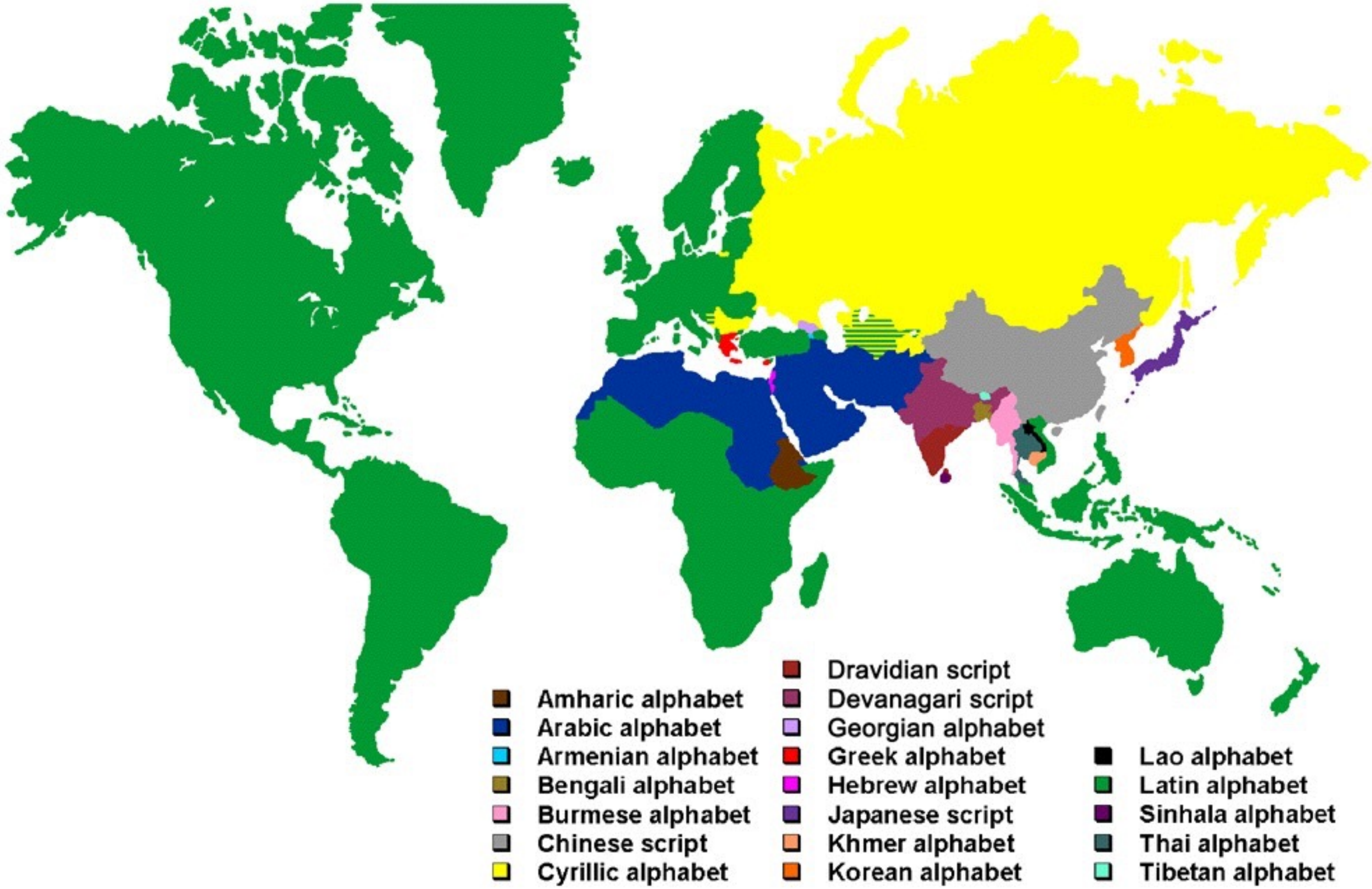
- **ASCII: American Standard Code for Information Interchange**
  - 7-bit character encoding standard: 128 valid codes
  - Range: 0x00 – 0x7F [$(0000\ 0000)_2$ to $(0111\ 1111)_2$]
  - Includes alphabets (upper and lower cases), digits, punctuations, common symbols, control characters
  - Worked (relatively) well for English typewriting

Resume vs. Résumé

- Diacritics
  - résumé :: resume
  - naïve :: naive
  - café :: cafe
  - Québec
  - Zürich
  - Fédération Internationale de Football Association (FIFA)

- International languages
  - 基本上 सहायक ασπασθ универсальной
  - ♪ ♬ ♩
  - ☺ ☹

Written Scripts

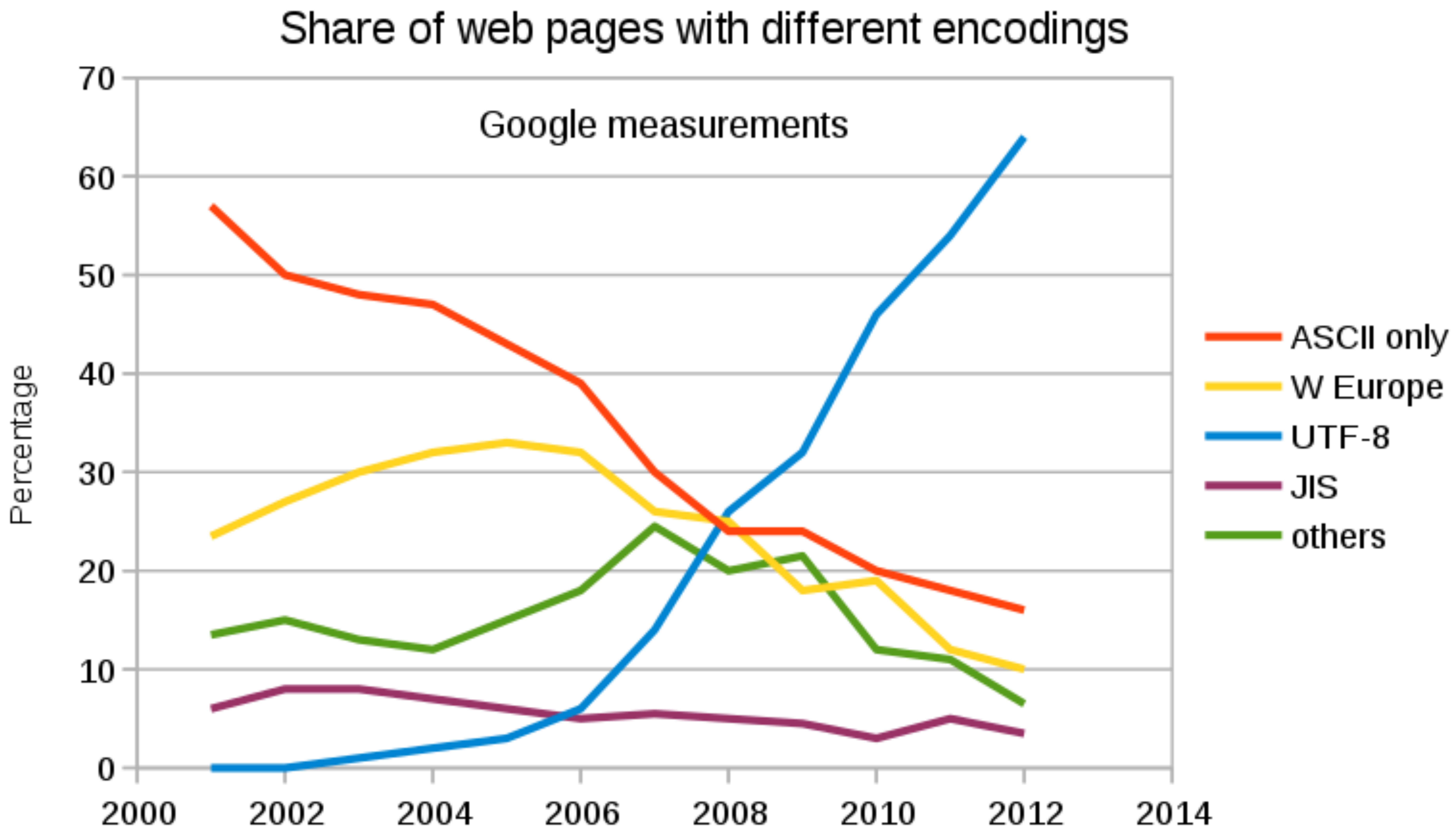


- Latin: 36% (2.6B people)
- Chinese: 18% (1.3B)
- Devanagari: 14% (1B)
- Arabic: 14% (1B)
- Cyrillic: 4% (0.3B)
- Dravidian: 3.5% (0.25B)

Other Character Encodings

- IBM EBCDIC
- Latin-1
- JIS: Japanese Industrial Standards
- CCCII: Chinese Character Code for Information Interchange
- EUC: Extended Unix Code
- Numerous other national standards

- Unicode and UTF-8


Share of web pages with different encodings
Google measurements
Percentage
70
60
50
40
30
20
10
0
2000
2002
2004
2006
2008
2010
2012
2014
ASCII only
W Europe
UTF-8
JIS
others

Unicode

- Industry standard for encoding and representing text
- Over 128,000 characters from 130+ scripts and symbol sets
- Can be implemented by different character endings
  - UTF-8: One byte to up to four bytes
  - UTF-16: One or two 16-bit code units
  - UTF-32: One 32-bit code unit

UTF-8

- Unicode Transformational Format – 8-bits
- Variable length encoding: One to four bytes
- Backward compatible with ASCII
  - One byte codes same as ASCII
- Dominant character encoding for the Web

- How to handle in Python?
  - Default in Python 3
  - In Python 2:

```
# -*- coding: utf-8 -*-
```

Let’s see an example: Résumé

Python 3

```
>>> text1="Résumé"
>>> len(text1)
6
>>> text1
'Résumé'

>>> [c for c in text1]
['R', 'é', 's', 'u', 'm', 'é']
```

Python 2

```
>>> text1="Résumé"
>>> len(text1)
8
>>> text1
‘R\xc3\xa9sum\xc3\xa9'

>>> [c for c in text1]
['R', '\xc3', '\xa9', 's', 'u', 'm', '\xc3', ‘\xa9']

>>> text2=u'Résumé'
>>> len(text2)
6
>>> text2
u'R\xe9sum\xe9'
>>> [c for c in text2]
[u'R', u'\xe9', u's', u'u', u'm', u'\xe9']
```

Take Home Concepts

- Diversity in Text
- ASCII and other character encodings
- Handling text in UTF-8