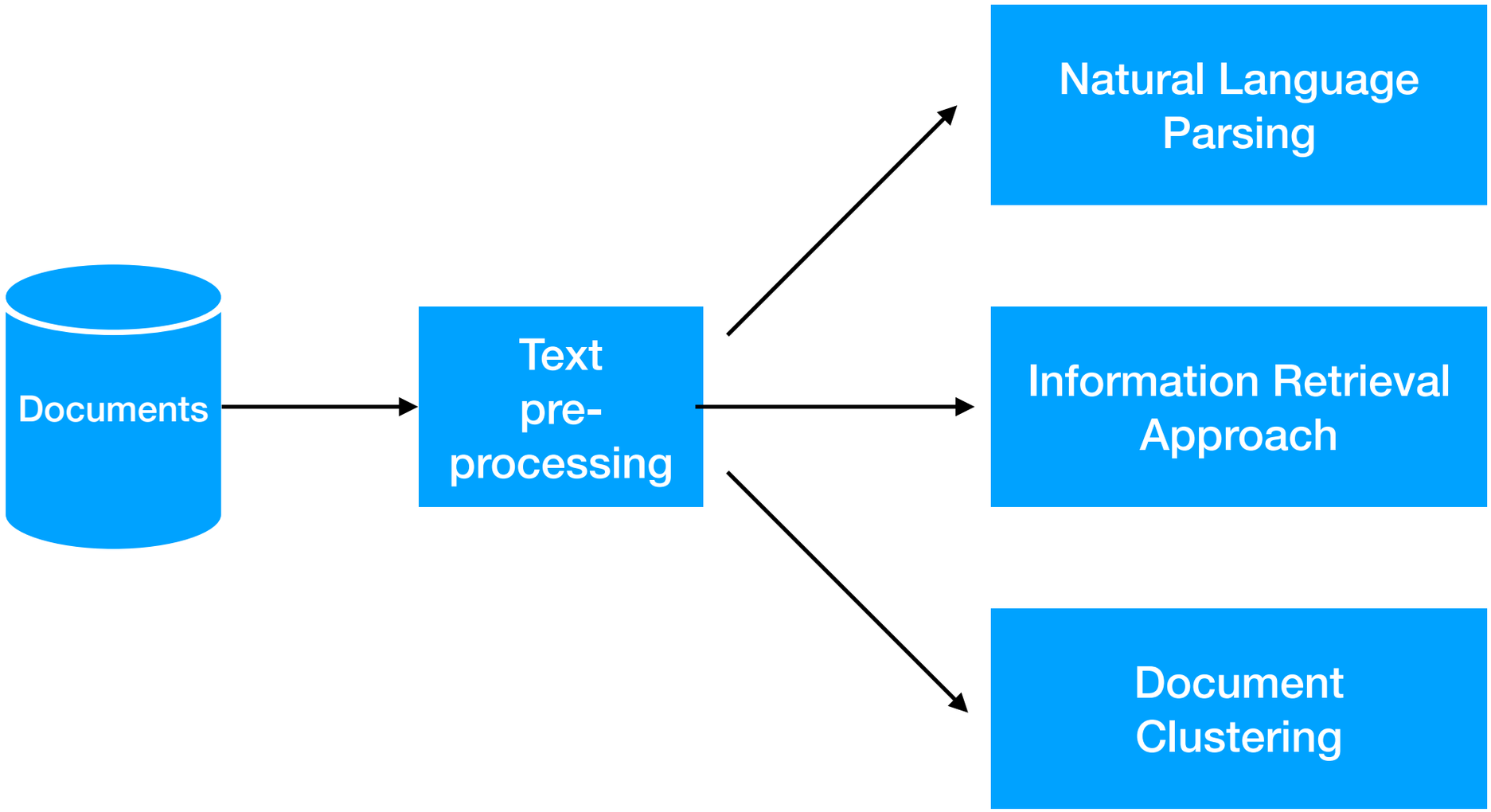
Text Preprocessing

What we will see here?

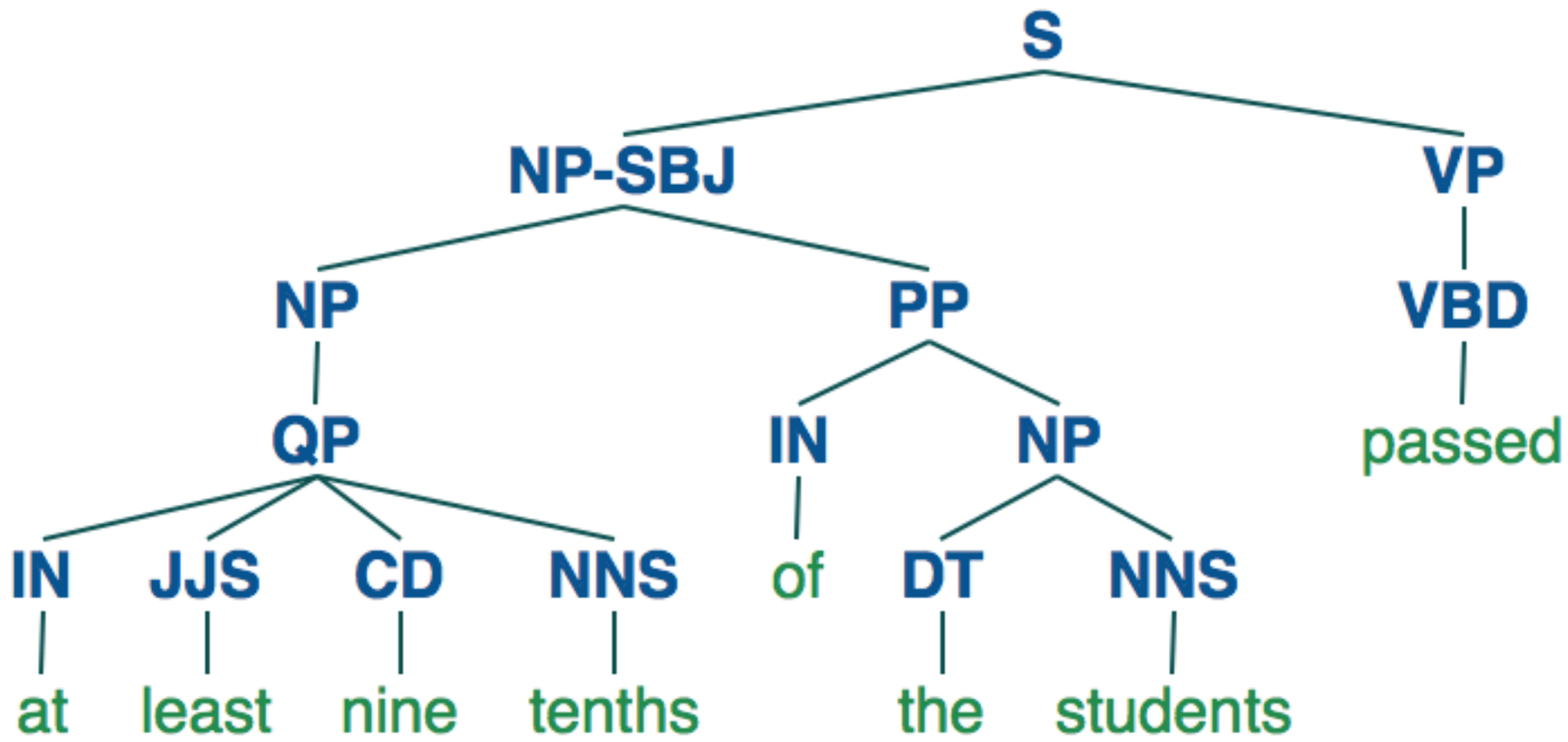
- Overview to a Natural Language Processing approach
- Role of textual processing
- Regular Expressions
- Tokenization, Sentence Splitting
- Stop word removal
- Stemming/lemmatization

Typical NLP approach



What is NL Parsing

Grammatical and logical analysis of text



NL Parsing and Pre-Processing

- Compared with text clustering and traditional Information Retrieval Methods, NL parsing requires a limited (or no) pre-processing

- Mainly sentence/paragraph split, and removal of non-text blocks (e.g., tables, source code contained in the textual document)

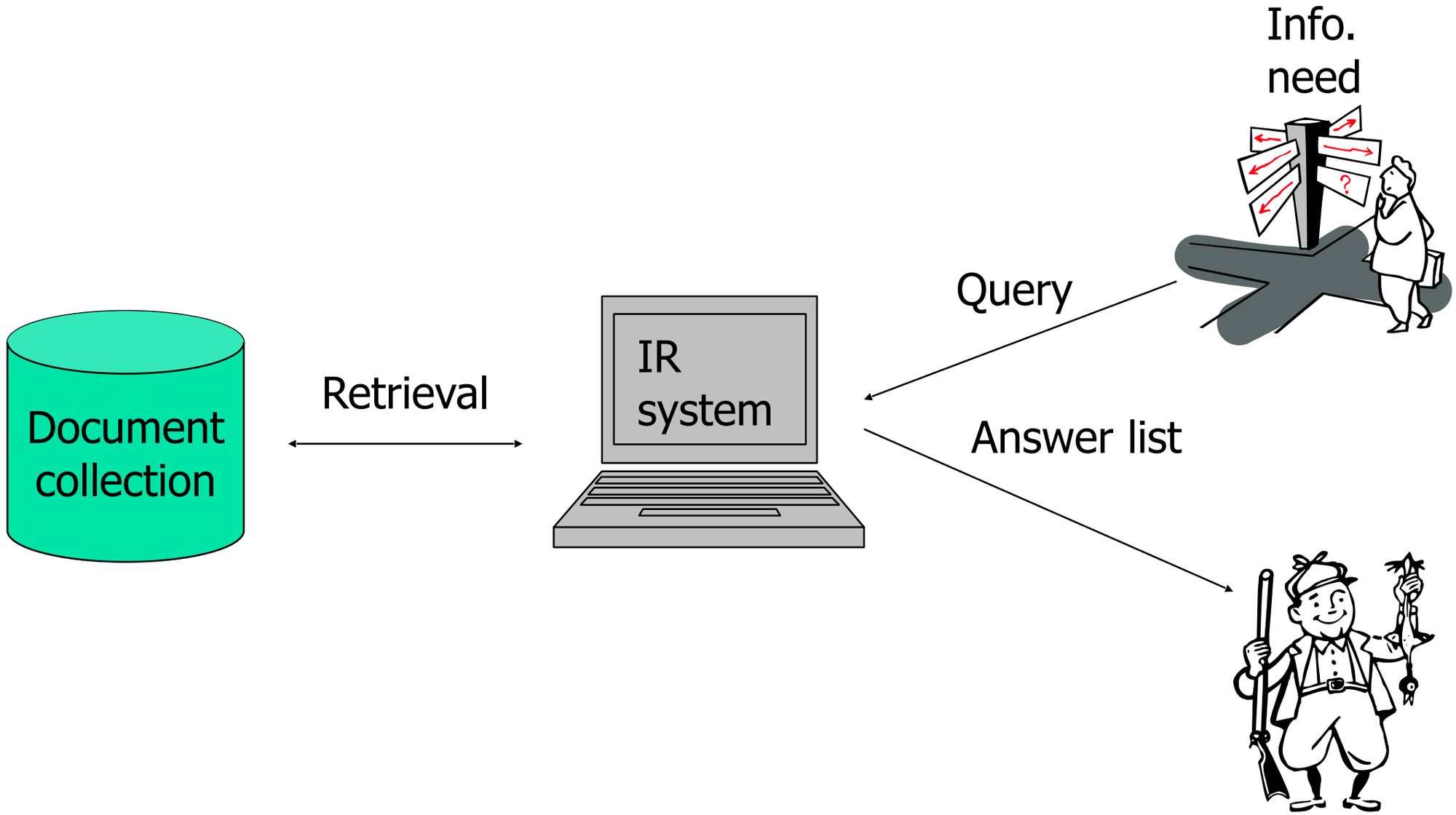
- Sometimes even sentence split is done by the approach automatically

What is Information Retrieval ?

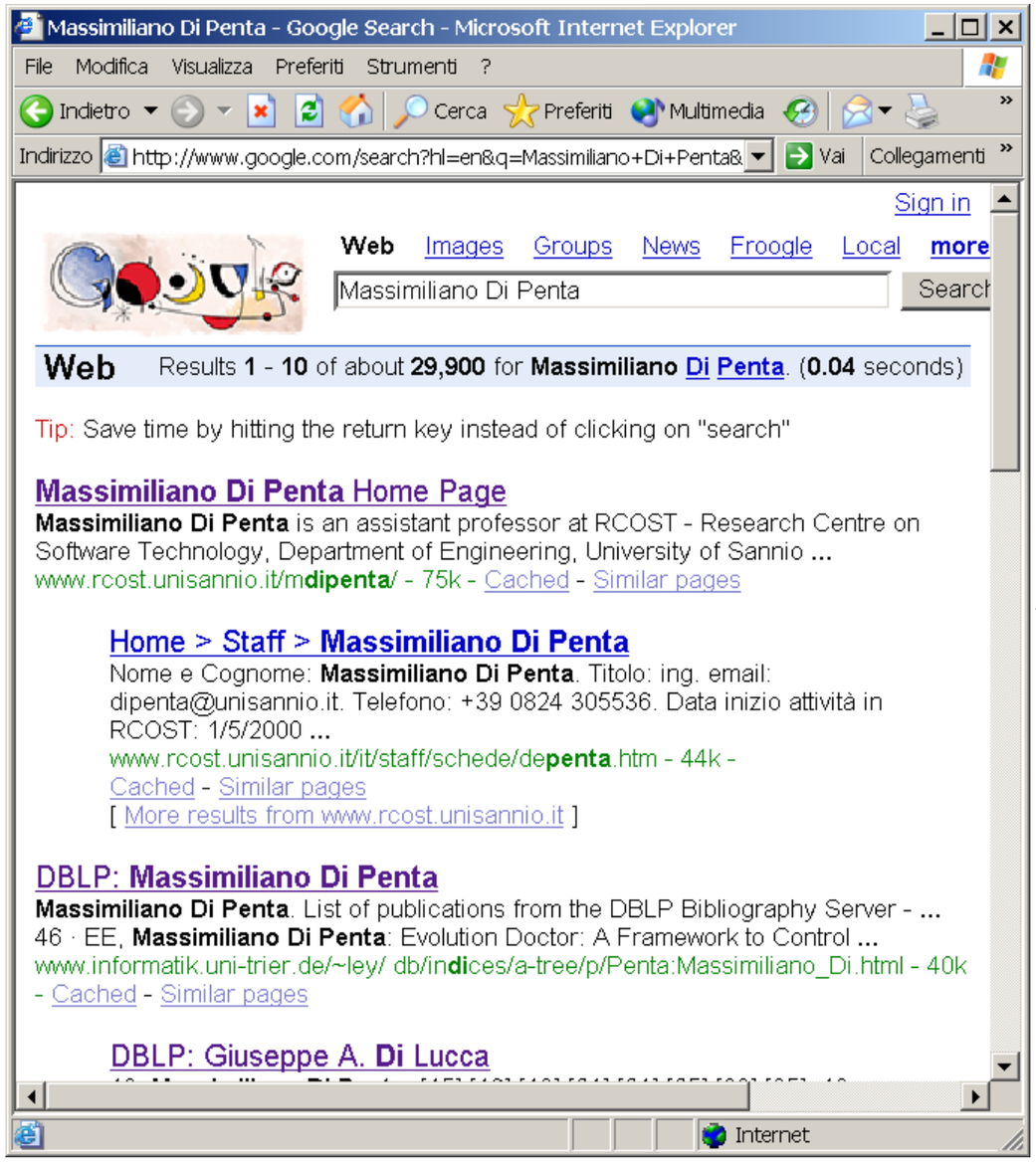
- The process of actively seeking out information relevant to a topic of interest [van Rijsbergen]
- Representation, storage, organization, and access to information items
  [Baeza-Yates, Ribeiro-Neto]
- Typically it refers to the automatic (rather than manual) retrieval of documents
  - Information Retrieval System (IRS)
- "Document" is the generic term for an information holder (book, chapter, article, webpage, etc)

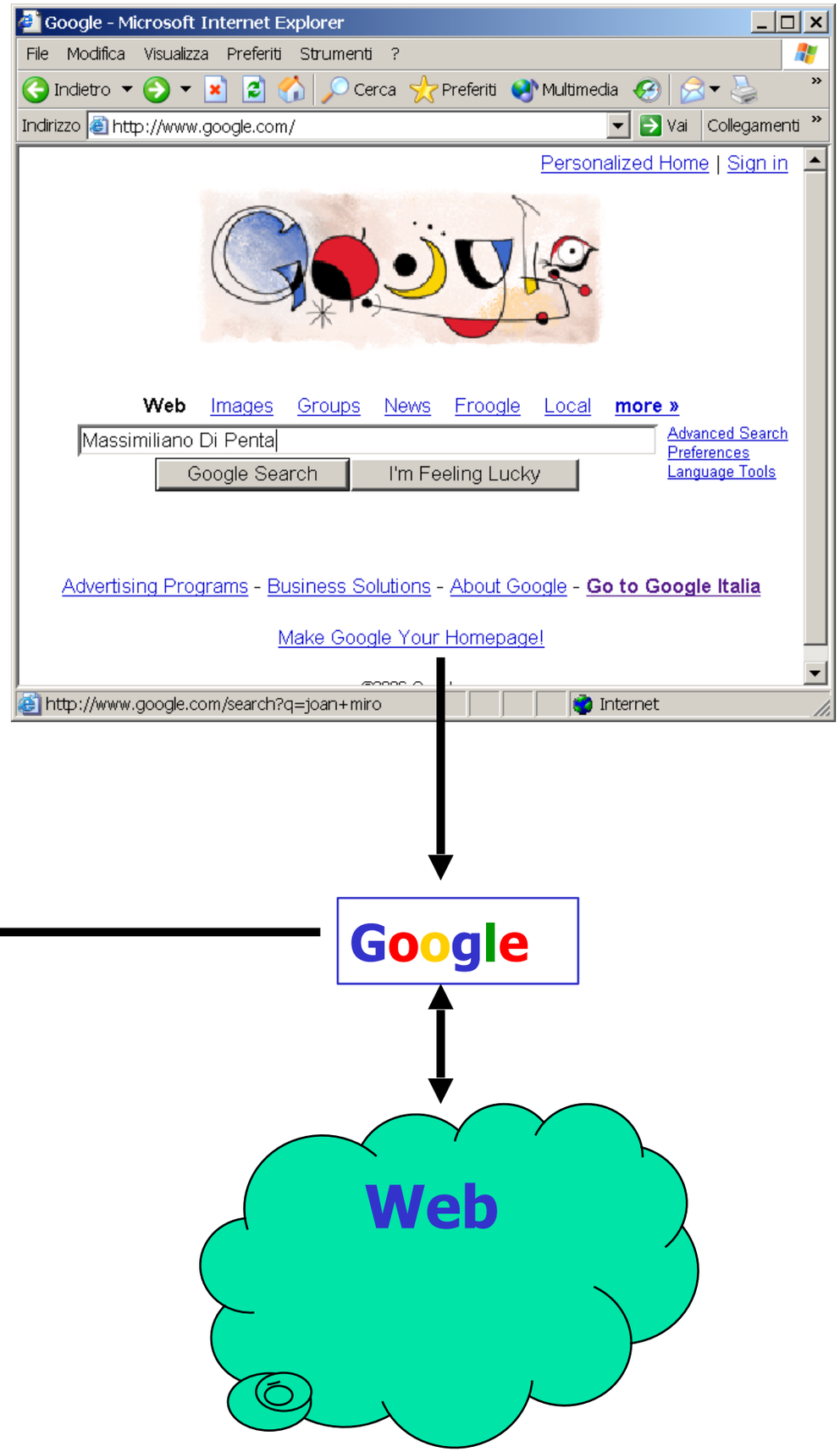
The problem of IR

Goal = find documents relevant to an information need from a large document set



Example

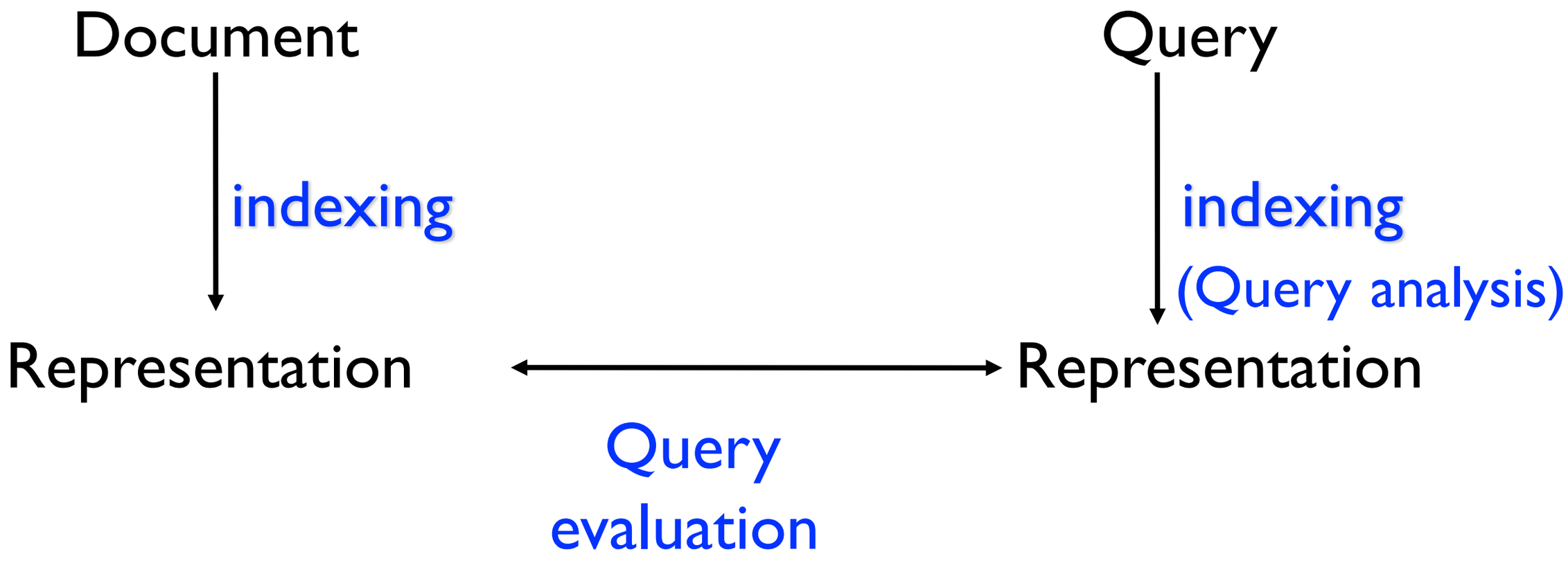

Massimiliano Di Penta - Google Search - Microsoft Internet Explorer
File Modifica Visualizza Preferiti Strumenti ?
Indietro Cerca Preferiti Multimedia
Indirizzo http://www.google.com/search?hl=en&q=Massimiliano+Di+Penta&
Vai Collegamenti
Sign in
Web Images Groups News Froogle Local more
Massimiliano Di Penta
Search
Web Results 1 - 10 of about 29,900 for Massimiliano Di Penta. (0.04 seconds)
Tip: Save time by hitting the return key instead of clicking on "search"
Massimiliano Di Penta Home Page
Massimiliano Di Penta is an assistant professor at RCOST - Research Centre on Software Technology, Department of Engineering, University of Sannio ...
www.rcost.unisannio.it/mdipenta/ - 75k - Cached - Similar pages
Home > Staff > Massimiliano Di Penta
Nome e Cognome: Massimiliano Di Penta. Titolo: ing. email: dipenta@unisannio.it. Telefono: +39 0824 305536. Data inizio attività in RCOST: 1/5/2000 ...
www.rcost.unisannio.it/it/staff/schede/depenta.htm - 44k - Cached - Similar pages
[More results from www.rcost.unisannio.it]
DBLP: Massimiliano Di Penta
Massimiliano Di Penta. List of publications from the DBLP Bibliography Server - ...
46 · EE, Massimiliano Di Penta: Evolution Doctor: A Framework to Control ...
www.informatik.uni-trier.de/~ley/ db/indices/a-tree/p/Penta:Massimiliano_Di.html - 40k - Cached - Similar pages
DBLP: Giuseppe A. Di Lucca
Internet


Google - Microsoft Internet Explorer
File Modifica Visualizza Preferiti Strumenti ?
Indietro Cerca Preferiti Multimedia
Indirizzo http://www.google.com/
Vai Collegamenti
Personalized Home | Sign in
Web Images Groups News Froogle Local more »
Massimiliano Di Penta
Google Search I'm Feeling Lucky
Advanced Search
Preferences
Language Tools
Advertising Programs - Business Solutions - About Google - Go to Google Italia
Make Google Your Homepage!
http://www.google.com/search?q=joan+miro
Internet
Google
Web

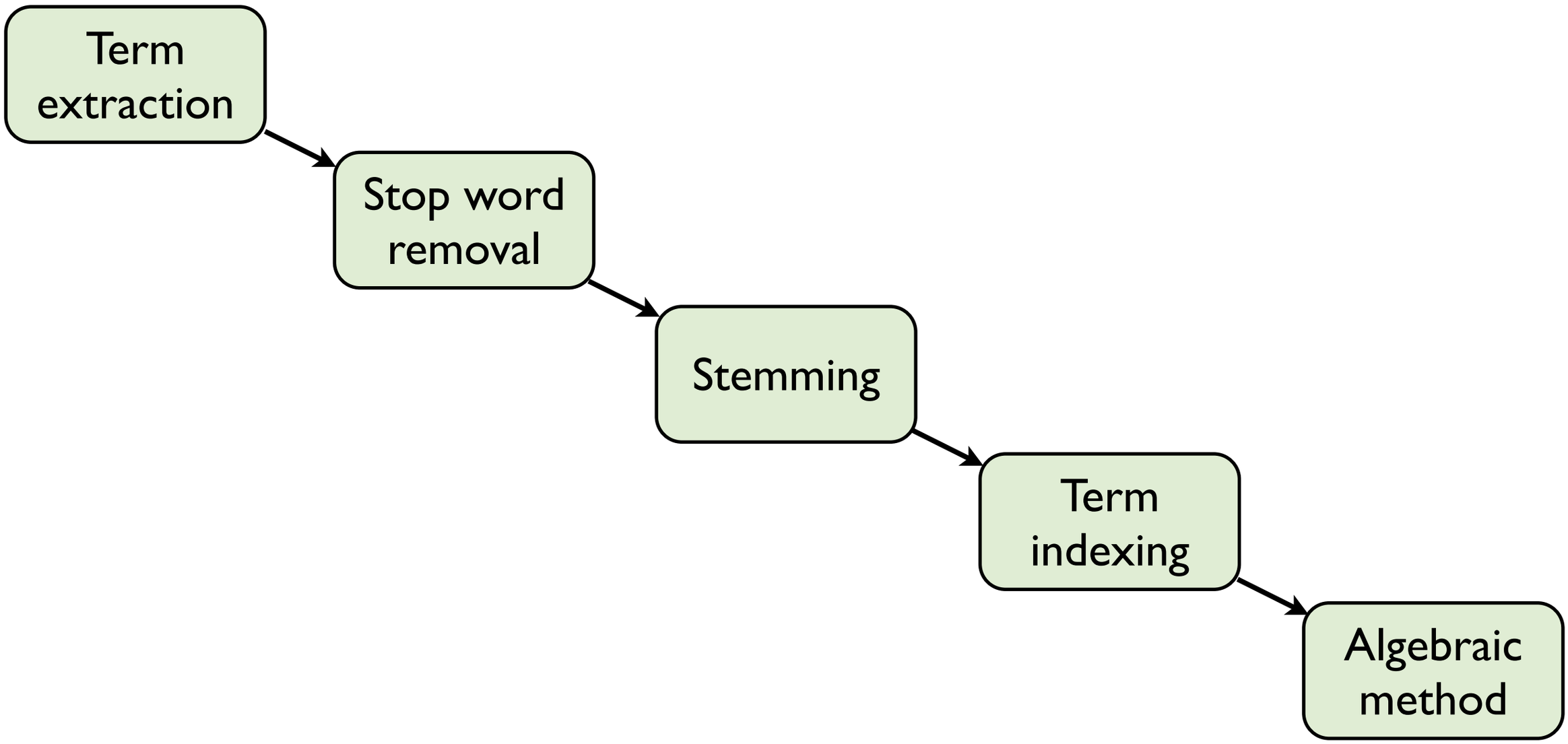
Note

- IR and text clustering often require a more heavyweight text processing
- This is because documents are just treated as “bag of words”
- Or maybe models are built on top of “bag of words”
- However, the syntax and semantics of the sentence does not contribute in the retrieval process

Indexing-based IR



Document indexing



The above process may vary a lot...

Term Extraction

- Separate words useful for the indexing from other elements
  - Punctuation, special characters
  - Numbers (do we need them?)
  - Elements we don’t want to consider (e.g., tables, blocks of source code)

Stop Word Removal

- Remove words that do not bring an informative content to the indexing process

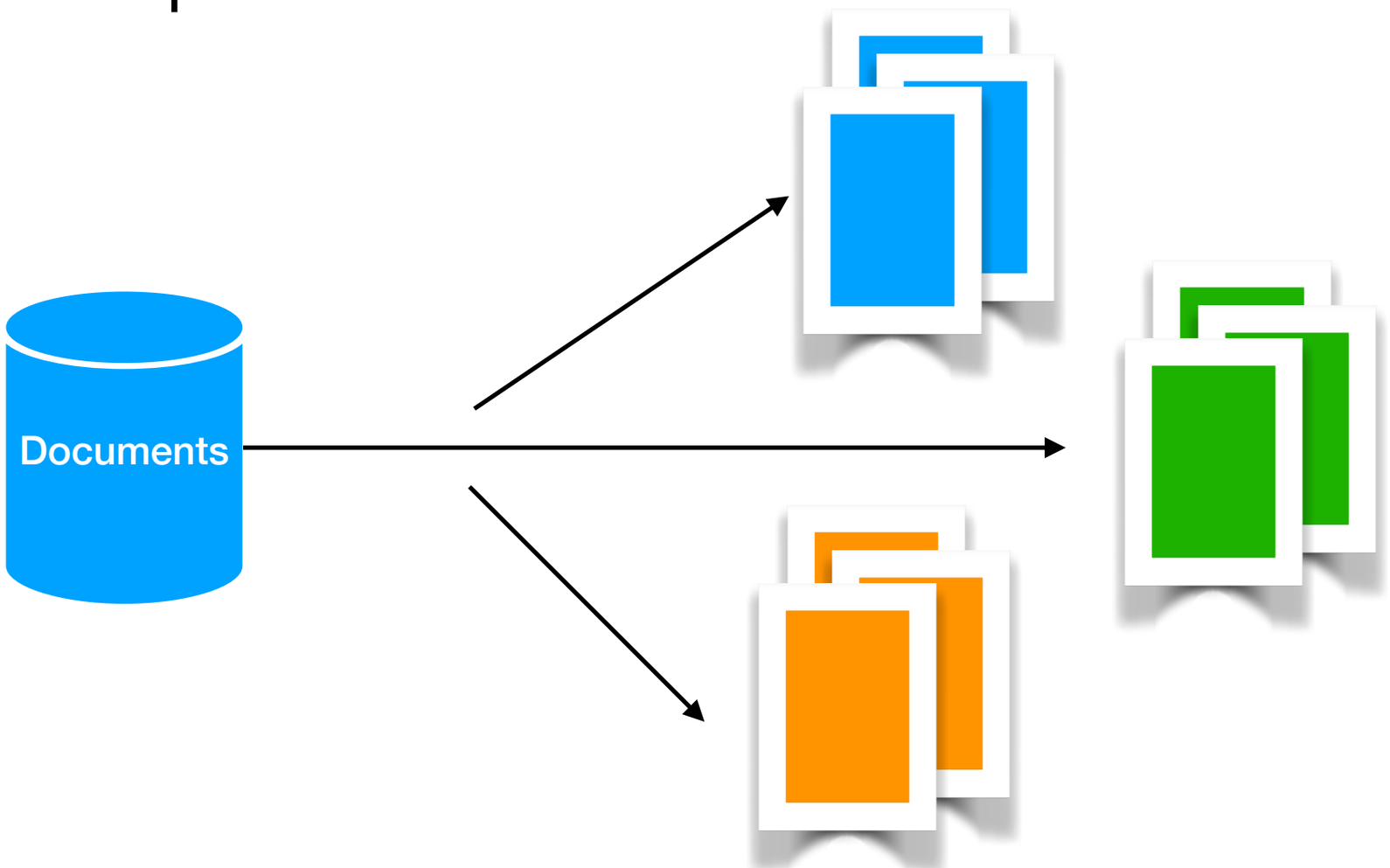
- E.g., articles, prepositions, some common verbs and adjectives

Stemming/Lemmatization

- Avoid that word differ between each other only by verb conjugation or singular/plural
  - cars → car (stemming enough)
  - computer → comput (stemming enough)
  - went → go (this requires lemmatization)

Document Clustering

Create cohesive groups of documents, e.g., articles on different topics



Note

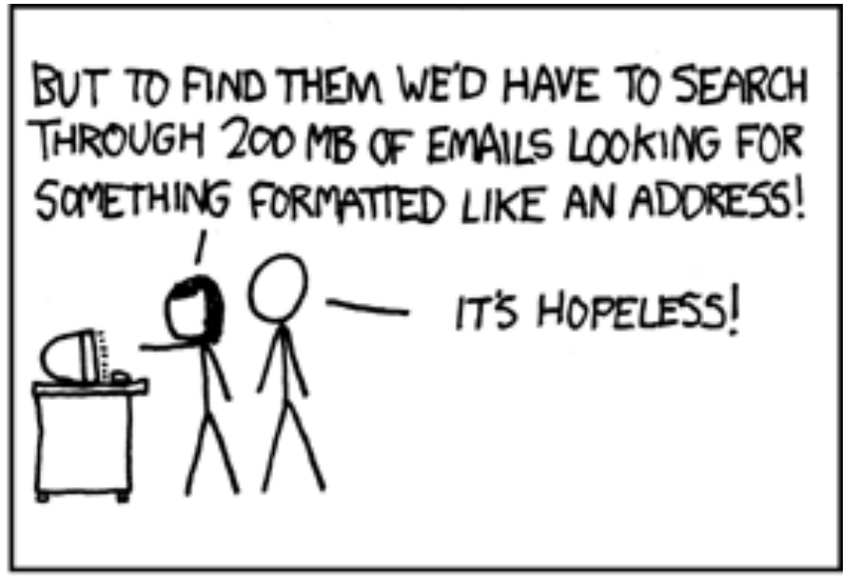
- The approaches being used build on the same concepts of IR

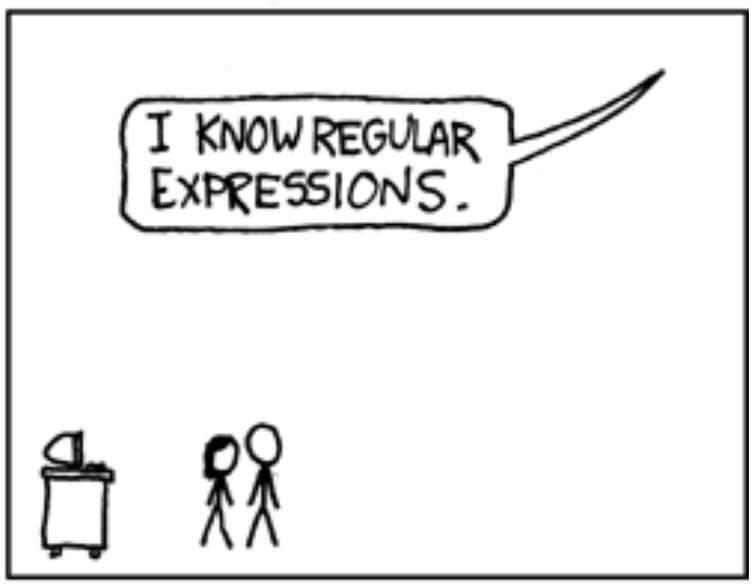
- Therefore, the text preprocessing to be performed is very similar

Regular Expressions

Our premier tool for text processing

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.





"Some people, when confronted with a problem, think 'I know, I'll use regular expressions.' Now they have two problems."

-- Jamie Zawinski

http://www.jwz.org/

Regular expressions

- A formal language for specifying text strings

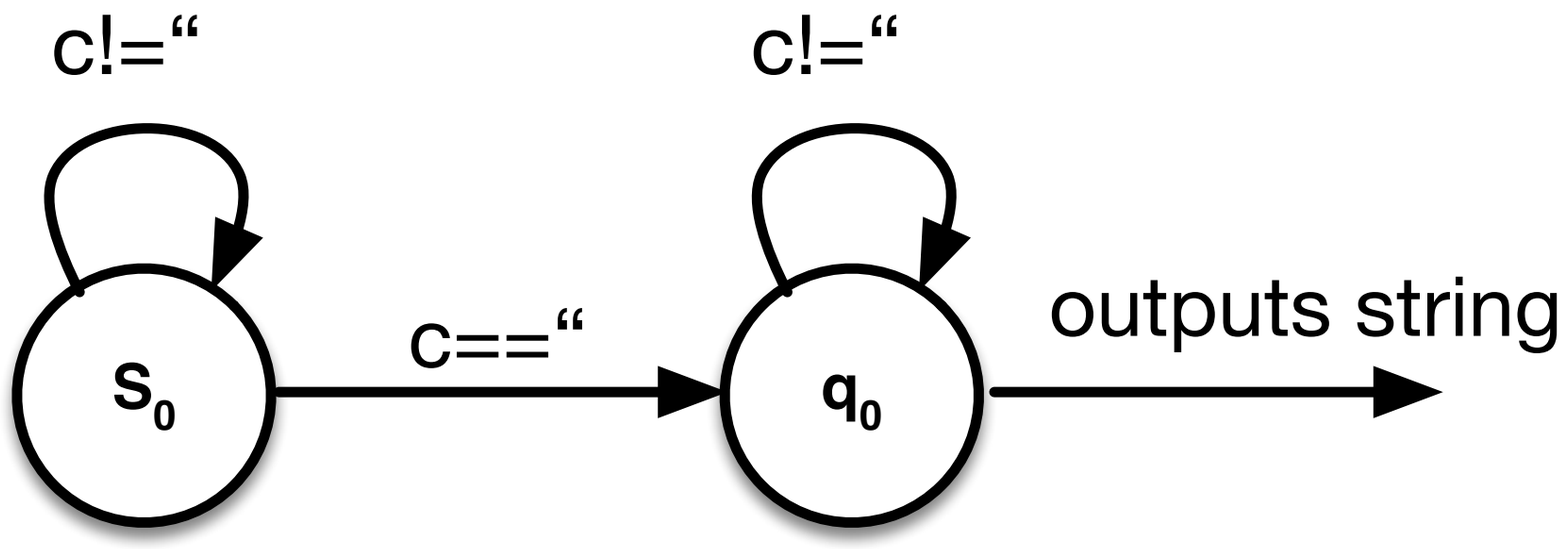
- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

Regular Expressions

- Regular expressions are a powerful string manipulation tool
- All modern languages have similar library packages for regular expressions
- Use regular expressions to:
  - Search a string (`search` and `match`)
  - Replace parts of a string (`sub`)
  - Break strings into smaller pieces (`split`)

How text matching works

- In essence, it is an automata
- Assume you want to capture a string literal in a program, the automata will look like:

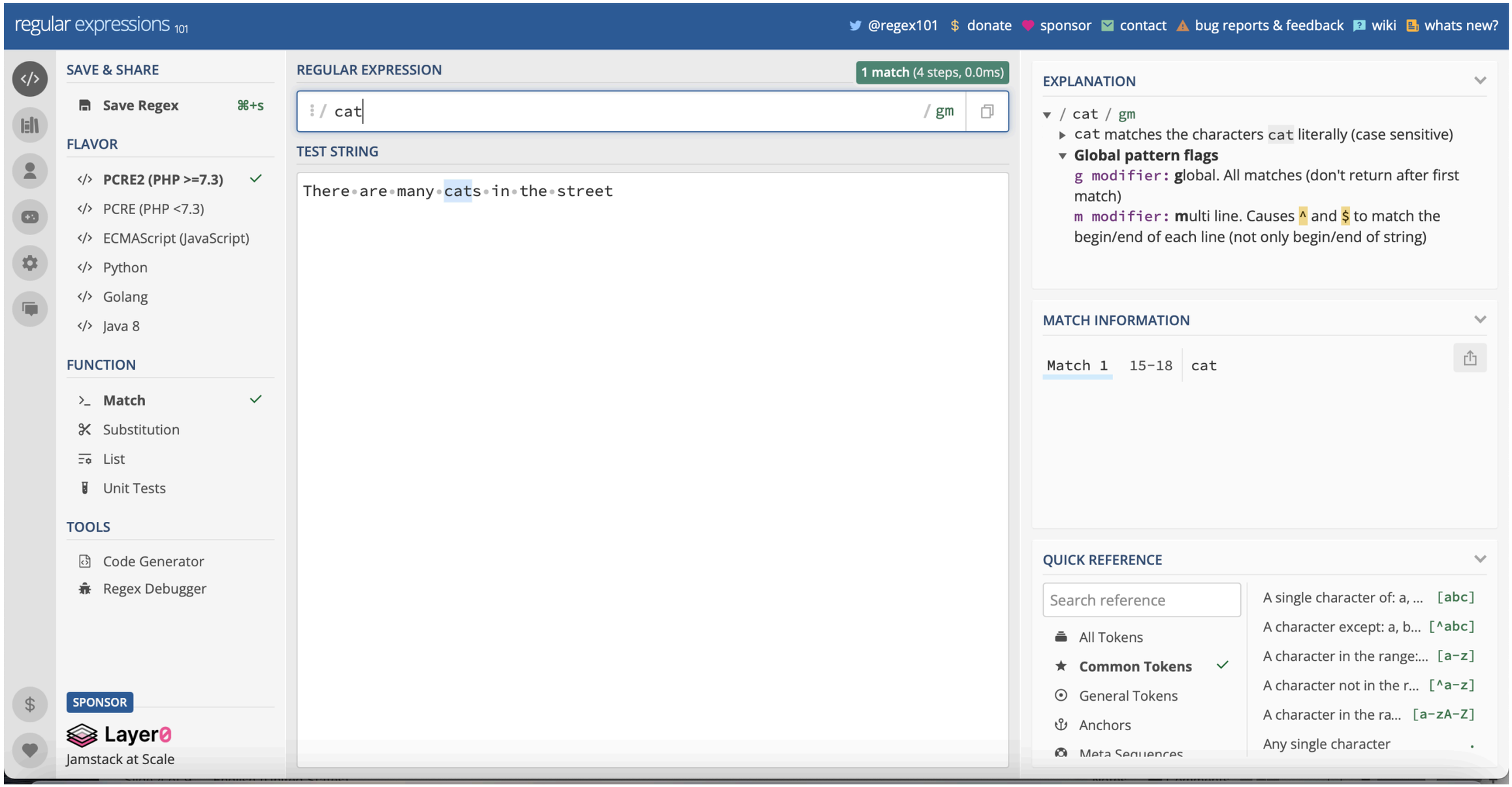


RE testing tool

- https://regex101.com
- Many other alternatives available

- We will first understand regular expressions using the debugger, and then will see them at work in python

Usage


regular expressions 101
@regex101
donate
sponsor
contact
bug reports & feedback
wiki
whats new?
SAVE & SHARE
Save Regex
⌘+s
FLAVOR
PCRE2 (PHP >=7.3)
PCRE (PHP <7.3)
ECMAScript (JavaScript)
Python
Golang
Java 8
FUNCTION
Match
Substitution
List
Unit Tests
TOOLS
Code Generator
Regex Debugger
SPONSOR
Layer0
Jamstack at Scale
REGULAR EXPRESSION
1 match (4 steps, 0.0ms)
/ cat
/ gm
TEST STRING
There are many cats in the street
EXPLANATION
/ cat / gm
cat matches the characters cat literally (case sensitive)
Global pattern flags
g modifier: global. All matches (don't return after first match)
m modifier: multi line. Causes ^ and $ to match the begin/end of each line (not only begin/end of string)
MATCH INFORMATION
Match 1
15-18
cat
QUICK REFERENCE
Search reference
All Tokens
Common Tokens
General Tokens
Anchors
Meta Sequences
A single character of: a, ...
[abc]
A character except: a, b...
[^abc]
A character in the range:...
[a-z]
A character not in the r...
[^a-z]
A character in the ra...
[a-zA-Z]
Any single character
.

Regular Expressions: Disjunctions

Letters inside square brackets []

Pattern	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Any digit

Ranges `[A-Z]`

Pattern	Matches	
`[A-Z]`	An upper case letter	`Drenched Blossoms`
`[a-z]`	A lower case letter	`my beans were impatient`
`[0-9]`	A single digit	`Chapter 1: Down the Rabbit Hole`

Regular Expressions: Negation in Disjunction

Pattern	Matches	
`[^A-Z]`	Not an upper case	`Oyfn pripetchik`
`[^Ss]`	Neither ‘S’ nor ‘s’	`I have no exquisite reason”`
`[^e^]`	Neither e nor ^	`Look here`
`[a^b]`	The pattern a carat b	`Look up a^b now`

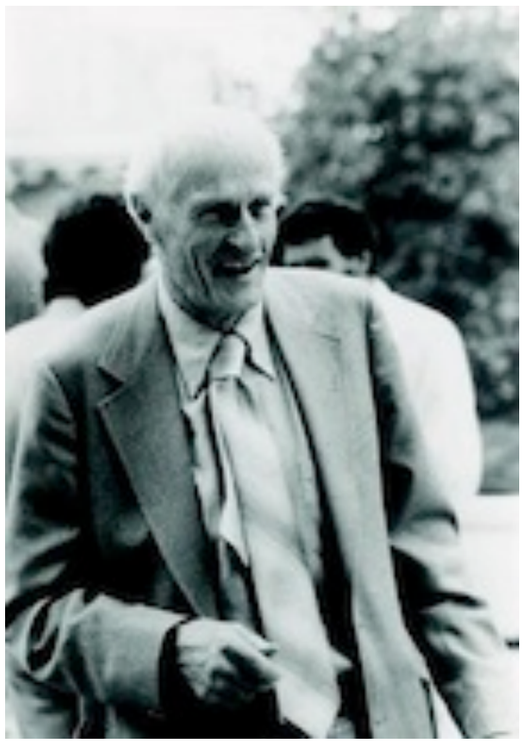
Regular Expressions: More Disjunction

Woodchuck is another name for groundhog!

Pattern	Matches
`groundhog|woodchuck`	`woodchuck`
`yours|mine`	`yours`
`a|b|c`	= [abc]
`[gG]roundhog|[Ww]oodchuck`	`Woodchuck`

Regular Expressions: ? *+.

Pattern	Matches	
`colou?r`	Optional previous char	color colour
`oo*h!`	0 or more of previous char	oh! ooh! oooh! ooooh!
`o+h!`	1 or more of previous char	oh! ooh! oooh! ooooh!
`baa+`		baa baaa baaaa baaaaa
`beg.n`	Any character	begin begun begun beg3n



Stephen C Kleene

Kleene *, Kleene +

Regular Expressions: Anchors ^ $

Pattern	Matches
^[A-Z]	Palo Alto
^[^A-Za-z]	1 “Hello”
\.$	The end.
.$	The end? The end!

Special control characters

- “\d” matches any digit; “\D” any non-digit
- “\s” matches any whitespace character; “\S” any non-whitespace character
- “\w” matches any alphanumeric character; “\W” any non-alphanumeric character
- “\b” matches a word boundary; “\B” matches a character that is not a word boundary

General note

If you want to match the appearance of a special character e.g. `($.^\[])` in a regular expression you need to precede it by escape

e.g.

`\$\d+(\.\d+)*` matches $10, $10.50 etc.

Word boundaries

- `\bcar\b` matches “car”, “.car”, “car go”, but not “caret”
- `car\b` matches “car”, “car.”, “supercar” but not “caret”
- `\Bcar` matches “supercar” but not “this is my car”

Other examples

- `/\b[a-zA-Z_][\w_]*\b/` matches any legal Python identifier, i.e., letter or underscore followed by (zero or more) alphanumeric or underscore

  - Matches Foo 99, 99

  - Does not match 99alpha

  - Does not even match the "alpha" after 99 because the sequence must be contained in a word boundary

- `\b\d\d:\d\d:\d\d\b` matches a time (note there is no control on the ranges)

Ranges

- {x,y} indicates that a pattern can be repeated from x to y times
- {x} indicates that a pattern must be repeated x times
- {x,} indicates that a pattern must be repeated at least x times
- Note: enclose the preceding pattern in parenthesis if composed of multiple characters

Examples

- `(go){2}` matches “gogo” but not “go”
- `(go){2,3}` matches “gogogo” or “gogo”
- `(go){2,}` matches “gogo”, “gogogo”, “gogogogo”, etc.

Regexp: issues

Find me all instances of the word “the” in a text.

`the`

Misses capitalized examples

`[tT]he`

Incorrectly returns `other` or `theology`

`[^a-zA-Z][tT]he[^a-zA-Z]`

Errors

The process we just went through was based on fixing two kinds of errors:

1. Matching strings that we should not have matched (there, then, other)
   False positives (Type I errors)

2. Not matching things that we should have matched (The)
   False negatives (Type II errors)

Errors cont.

- In NLP we are always dealing with these kinds of errors.
- Reducing the error rate for an application often involves two antagonistic efforts:
  - Increasing accuracy or precision (minimizing false positives)
  - Increasing coverage or recall (minimizing false negatives).

Summary

- Regular expressions play a surprisingly large role
  - Sophisticated sequences of regular expressions are often the first model for any text processing text

- For hard tasks, we use machine learning classifiers
  - But regular expressions are still used for pre-processing, or as features in the classifiers
  - Can be very useful in capturing generalizations

Substitutions

Substitution in Python and UNIX commands:

```
s/regexp1/pattern/
```

e.g.:

```
s/colour/color/
```

Capture Groups

Say we want to put angles around all numbers:

the 35 boxes → the <35> boxes

Use parens () to "capture" a pattern into a numbered register (1, 2, 3…)

Use \1 to refer to the contents of the register

```
s/([0-9]+)/<\1>/
```

Note: \1 is the Python syntax. Use $1 in the regexp matching tool or in Perl

Capture groups: multiple registers

```
/the (.*)er they (.*), the \1er we \2/
```

Matches

the faster they ran, the faster we ran

But not

the faster they ran, the faster we ate

But suppose we don't want to capture?

Parentheses have a double function: grouping terms, and capturing

Non-capturing groups: add a ?: after paren:

This means that such a group will not count as \1, \2 etc.. and will be skipped when counting groups

```
/(?:some|a few) (people|cats) like some \1/
```

matches

```
some cats like some cats
```

but not

```
some cats like some some
```

Lookahead assertions

`(?= pattern)` is true if pattern matches, but is **zero-width; doesn't advance character pointer**

`(?! pattern)` true if a pattern does not match

How to match, at the beginning of a line, any single word that doesn’t start with “Volcano”:

```
/^(?!Volcano)[A-Za-z]+/
```

Simple Application: ELIZA

Early NLP system that imitated a Rogerian psychotherapist Joseph Weizenbaum, 1966.

Uses pattern matching to match, e.g.,:

`“I need X”`

and translates them into, e.g.

`“What would it mean to you if you got X?`

Simple Application: ELIZA

Men are all alike.
IN WHAT WAY

They're always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE

Well, my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED

How ELIZA works

```
s/.* I'M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/

s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/

s/.* all .*/IN WHAT WAY?/

s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/
```

Regular Expressions in Python

Search and Match

- The two basic functions are **re.search** and **re.match**
  - Search looks for a pattern anywhere in a string
  - Match looks for a match starting at the beginning
- Both return *None* (logical false) if the pattern isn't found and a "match object" instance if it is

```
>>> import re
>>> pat = "a*b"
>>> re.search(pat,"fooaaabcde")
<_sre.SRE_Match object at 0x809c0>
>>> re.match(pat,"fooaaabcde")
>>>
```

Q: What's a match object?

A: an instance of the match class with the details of the match result

```
>>> r1 = re.search("a*b","fooaaabcde")
>>> r1.group()  # group returns string matched
'aaab'
>>> r1.start()  # index of the match start
3
>>> r1.end()    # index of the match end
7
>>> r1.span()   # tuple of (start, end)
(3, 7)
```

What got matched?

Here's a pattern to match simple email addresses
\w+@(\w+\.)+(com|org|net|edu|it)

```
>>> pat1 = "\w+@(\w+\.)(com|org|net|edu|it)"
>>> r1 = re.match(pat,"dipenta@unisannio.it")
>>> r1.group()
'dipenta@unisannio.it'
```

- We might want to extract the pattern parts, like the email name and host

What got matched?

- We can put parentheses around groups we want to be able to reference

```
>>> pat2 = "(\w+)@((\w+\.)+(com|org|net|edu|it))"
>>> r2 = re.match(pat2,”dipenta@cs.unisannio.it”)
>>> r2.group(1)
'dipenta'
>>> r2.group(2)
'cs'
>>> r2.groups()
r2.groups()
('dipenta', ‘cs.unisannio.it', ’unisannio.’, 'it’)
```

- Note that the ‘groups’ are numbered in a preorder traversal of the forest

What got matched?

- We can ‘label’ the groups as well…

```
>>> pat3 ="(?P<name>\w+)@(?P<host>(\w+\.)+(com|
  org|net|edu|it))"
>>> r3 = re.match(pat3,”dipenta@unisannio.it”)
>>> r3.group('name')
'dipenta'
>>> r3.group('host')
‘unisannio.it’
```

- And reference the matching parts by the labels

Pre match and Post match

- Useful if we want to continue processing what's before a pattern, or what's after a pattern

- We have seen already the meaning of match.group, match.start, match.end

- PREMATCH: `match.string[:match.start()]`

- MATCH: `match.group()`

- POSTMATCH: `match.string[match.end():]`

Example

```
import re
s="today John is going to school"
match=re.search("John",s)

print(match.string[:match.start()])
print(match.string[match.end():])
```

More re functions

- re.split() is like split but can use patterns

```
>>> re.split("\W+", “This... is a test,
   short and sweet, of split().”)
['This', 'is', 'a', 'test', 'short’,
   'and', 'sweet', 'of', 'split’, ‘’]
```

- re.sub substitutes one string for a pattern

```
>>> re.sub('(blue|white|red)', 'black', 'blue
   socks and red shoes')
'black socks and black shoes’
```

- re.findall() finds all matches

```
>>> re.findall("\d+”,"12 dogs,11 cats, 1 egg")
['12', '11', ’1’]
```

Compiling regular expressions

- If you plan to use a re pattern more than once, compile it to a re object
- Python produces a special data structure that speeds up matching

```
>>> capt3 = re.compile(pat3)
>>> cpat3
<_sre.SRE_Pattern object at 0x2d9c0>
>>> r3 = cpat3.search(“dipenta@unisannio.it”)
>>> r3
<_sre.SRE_Match object at 0x895a0>
>>> r3.group()
‘dipenta@unisannio.it’
```

Pattern object methods

Apply to all the re functions (e.g., match, search, split, findall, sub), e.g.:

```
>>> p1 = re.compile("\w+@\w+\.+com|org|net|edu")
>>> p1.match("steve@apple.com").group(0)
'steve@apple.com'

>>> p1.search(”Email steve@apple.com today.").group(0)
'steve@apple.com’

>>> p1.findall("Email steve@apple.com and bill@msft.com now.")
['steve@apple.com', 'bill@msft.com’]

>>> p2 = re.compile("[.?!]+\s+")
>>> p2.split("Tired? Go to bed!   Now!! ")
['Tired', 'Go to bed', 'Now', ’ ']
```

Matching options

- re functions have a third argument named flag
- Some values (others are more to control the character encoding):
  - re.IGNORECASE makes the matching case insensitive
  - re.MULTILINE is useful to process multiline strings
    - “^” matches the beginning of the string and each character following the newline (\n)
    - “$” matches the end of a string or of a line

Examples

- `re.search(“bug",b,re.IGNORECASE)` b is matches if it is equal to “Bug”, “BUG”, “bug” etc.

- `re.findall("^max","max is going\nmax is coming”,re.MULTILINE)`

  ['max', ‘max']

- `re.findall("^max","max is going\nmax is coming”)`

  ['max']

Greedy vs. Lazy Regexp

In a greedy a Kleene closure tries to match the longest possible sequence

Example: assume you have the regexp

```
\"(.+)\"
```

if you match the text:
John says “I’m happy today” and “I don’t want to go to school” while walking

The group will take the longest sequence between “”, i.e.
“I’m happy today” and “I don’t want to go to school”

Making the regexp lazy

- This allows matching until the first “ found only
- To achieve this, insert a ? after the Kleene closure

```
\”(.+?)\”
```

In this case if you match the text
John says “I’m happy today” and “I don’t want to go to school” while walking

The group will contain
“I’m happy today”

Further details…

Read the documentation!

https://docs.python.org/3/library/re.html