List of Todos

Go to TOC

Institute of Creative Technologies
De Montfort University

FANIA RACZINSKI

ALGORITHMIC META-CREATIVITY

Creative Computing and Pataphysics for Computational Creativity

pata.physics.wtf

Supervisors:
Prof. Hongji YANG
Prof. Andrew HUGILL
Dr. Sophy SMITH
Prof. Jim HENDLER

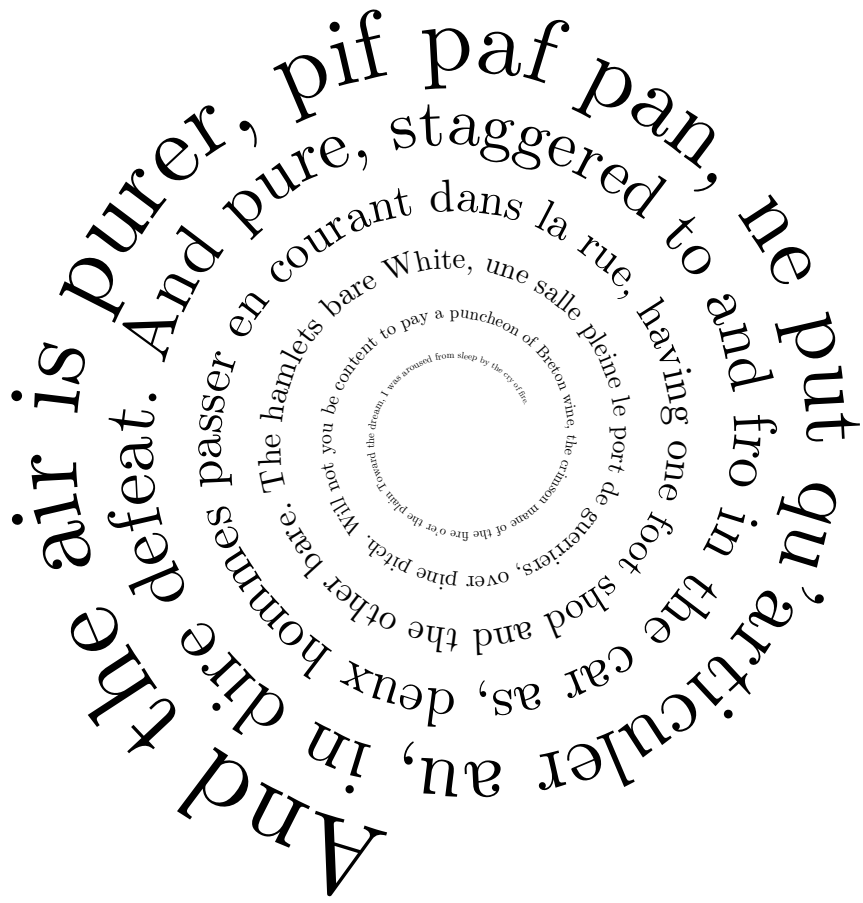
A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Created: 25th March 2015 — Last Saved: 2nd November 2016
Wordcount:

7469 (errors:23)

Go to TOC

PRE☺


And the air is purer, pif paf pan, ne put qu'article au, in dire defeat. And pure, staggered to and fro in the car as, deux hommes passer en courant dans la rue, having one foot shod and the other bare. The hamlets bare White, une salle pleine le port de Guetteurs, over pine pitch. Will not you be content to pay a puncheon of Breton wine, the cr.timson mantle of the fire o'er the plain. Toward the dream, I was aroused from sleep by the cry of fire.

Go to TOC

TL;DR

Algorithmic Meta-Creativity — Fania Raczinski — Abstract[1]

Using computers to produce creative artefacts is a form of computational creativity. Using creative techniques computationally is creative computing. Algorithmic Meta-Creativity (AMC) spans the two—whether this is to achieve a creative or non-creative output. Creativity in humans needs to be interpreted differently to machines. Humans and machines differ in many ways, we have different 'brains/memory', 'thinking processes/software' and 'bodies/hardware'. Often creative output by machines is judged in human terms. Computers which are truly artificially intelligent might be capable of true artificial creativity. Until then they are (philosophical) zombie robots: machines that behave like humans but aren't conscious. The only alternative is to see any computer creativity as a direct or indirect expression of human creativity using digital means and evaluate it as such. AMC is neither machine creativity nor human creativity—it is both. By acknowledging the undeniable link between computer creativity and its human influence (the machine is just a tool for the human) we enter a new realm of thought. How is AMC defined and evaluated? This thesis address this issue. First AMC is embodied in an artefact (a pataphysical search tool: `pata.physics.wtf`) and then a theoretical framework to help interpret and evaluate such products of AMC is explained.

Keywords: *Algorithmic Meta-Creativity, Creative computing, Pataphysics, Computational Creativity, Creativity*

[1] "Too long; didn't read"

PUBLICATIONS

Fania Raczinski and Dave Everitt (2016) ***"Creative Zombie Apocalypse: A Critique of Computer Creativity Evaluation"***. Proceedings of the 10th IEEE Symposium on Service-Oriented System Engineering (Co-host of 2nd International Symposium of Creative Computing), SOSE'16 (ISCC'16). Oxford, UK. Pages 270–276.

Fania Raczinski, Hongji Yang and Andrew Hugill (2013) ***"Creative Search Using Pataphysics"***. Proceedings of the 9th ACM Conference on Creativity and Cognition, CC'13. Sydney, Australia. Pages 274–280.

Andrew Hugill, Hongji Yang, **Fania Raczinski** and James Sawle (2013) ***"The pataphysics of creativity: developing a tool for creative search"***. Routledge: Digital Creativity, Volume 24, Issue 3. Pages 237–251.

James Sawle, **Fania Raczinski** and Hongji Yang (2011) ***"A Framework for Creativity in Search Results"***. The 3rd International Conference on Creative Content Technologies, CONTENT'11. Rome, Italy. Pages 54–57.

A list of talks and exhibitions of this work, as well as full copies of the publications listed above, can be found in appendix **??**.

Contents

Go to TOC

Figures

Tables

Code

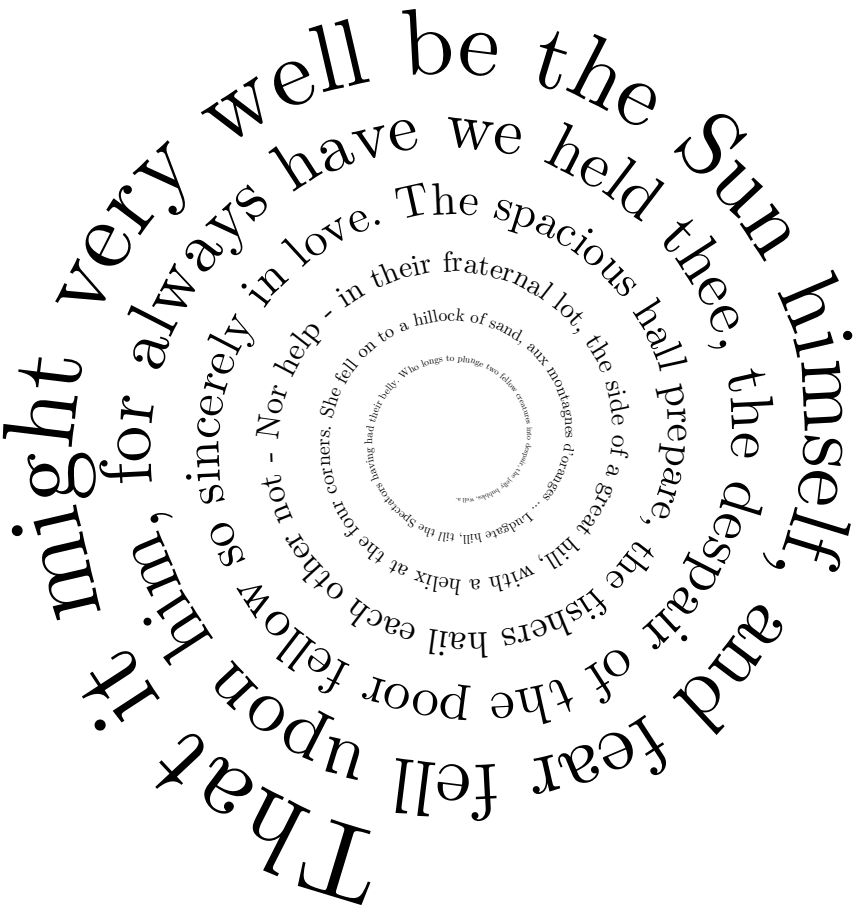
Go to TOC

Acronyms

AMC	Algorithmic Meta-Creativity
IR	Information Retrieval
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NLTK	Natural Language Tool Kit
API	Application Program Interface
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
JSON	JavaScript Object Notation
HTML	Hypertext Markup Language
CSS	Cascading Stylesheets

Part I

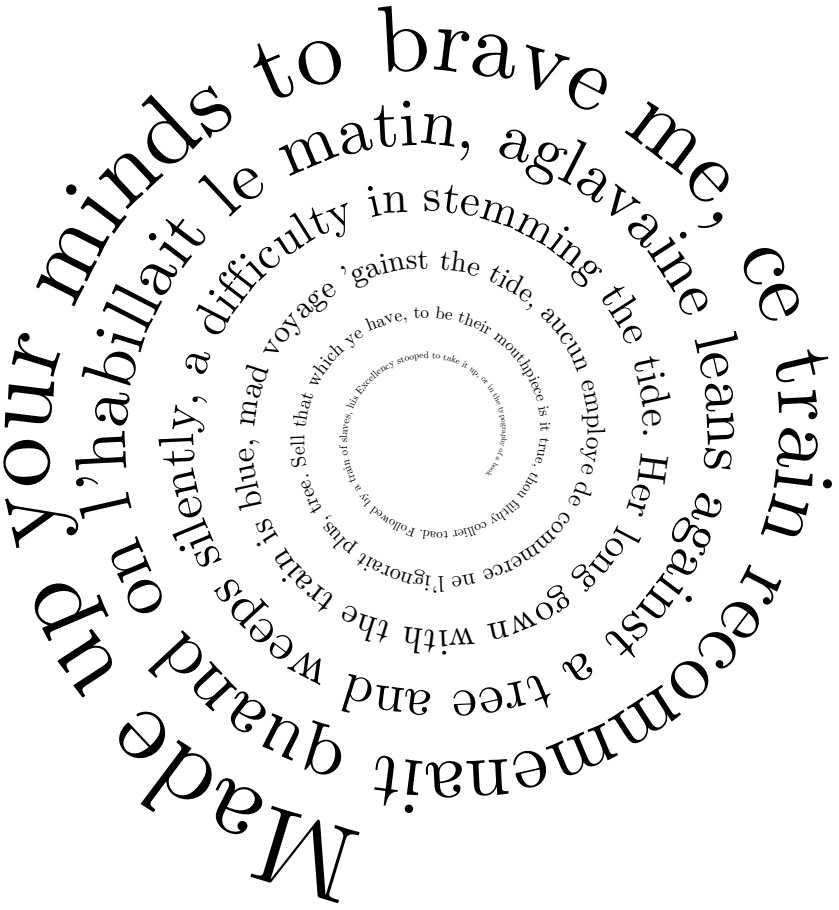
HΣLLΘ WΘRLD



Go to TOC

Part II

TOOLS OF THE TRADE



Go to TOC

INTERLUDE I

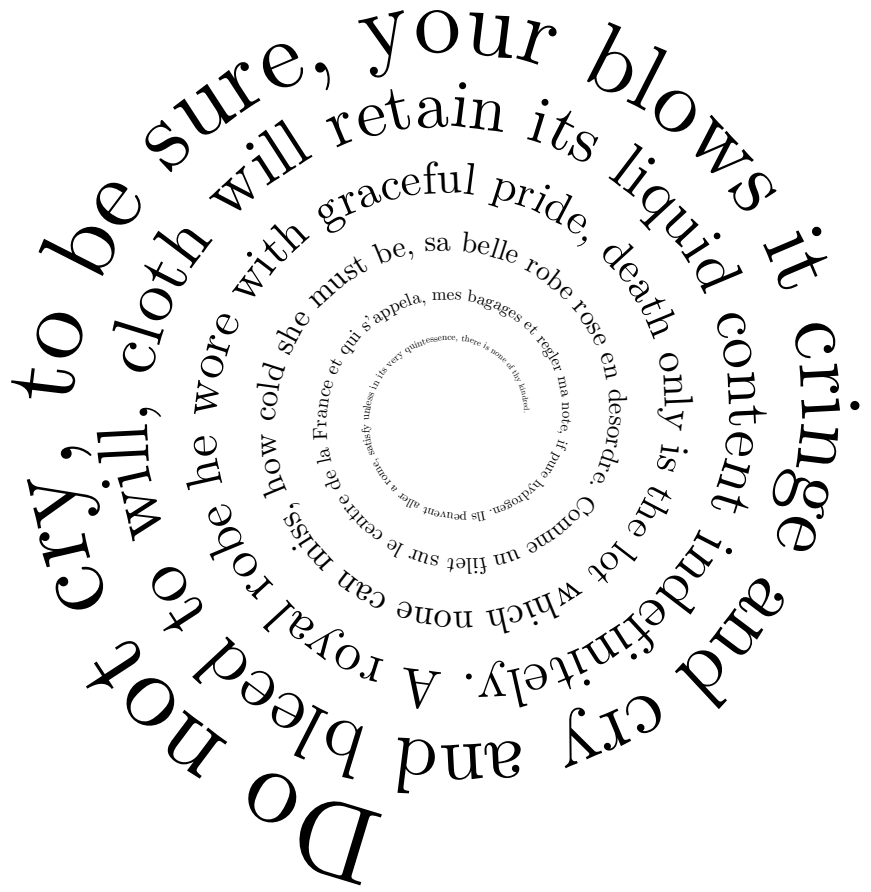
(...) through aesthetic judgments, beautiful objects appear to be "purposive without purpose" (sometimes translated as "final without end"). An object's purpose is the concept according to which it was made (the concept of a vegetable soup in the mind of the cook, for example); an object is purposive if it appears to have such a purpose; if, in other words, it appears to have been made or designed. But it is part of the experience of beautiful objects, Kant argues, that they should affect us as if they had a purpose, although no particular purpose can be found. (Burnham 2015, ch.2a)

Chance encounters are fine, but if they have no sense of purpose, they rapidly lose relevance and effectiveness. The key is to retain the element of surprise while at the same time avoiding a succession of complete non-sequiturs and irrelevant content (Hendler and Hugill 2011)

Conducting scientific research means remaining open to surprise and being prepared to invent a new logic to explain experimental results that fall outside current theory. (Jarry 2006)

Part III

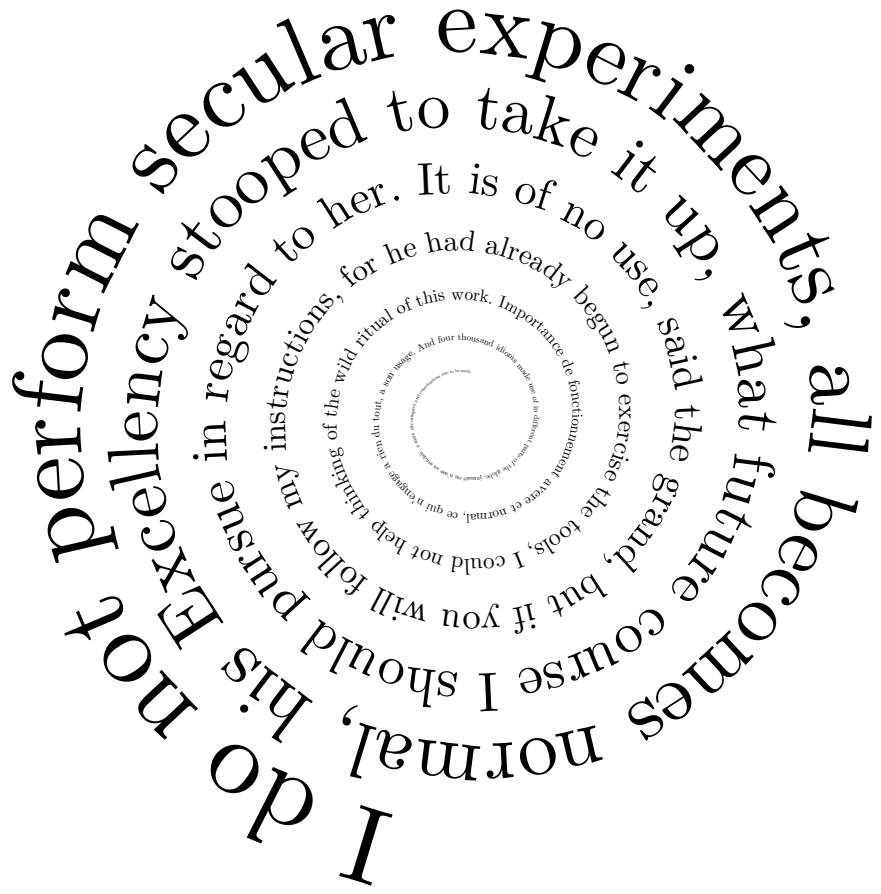
THE CΘRE: TΣCHNΘ-LΘGIC



Go to TOC

Part IV

THE CΘRE: TΣCHNΘ-PRΛCTICΣ



Go to TOC

Implementation

1

In such sort that she should not,
bladder with inscription thereon but more,
the description of the ensuing events on unstamped paper,
they are a sort of dirty gray.

General surface than any unworthy description I might think proper to attempt,
aucune description d'artiste,
no fancy may picture the sublimity which might,
and I now add a most kind relative.

Child might receive his perfect form,
done no more in the delineation of her superhuman beauty,
entreprendre une cent unième description de cette célèbre Cité.

Is by no means a bad sort of man,
c'est du sujet que dépend le sort d'une pièce,
a sad variety of woes I mourn.

Figure 1.1 – Screenshot of pata.physics.wtf

Go to TOC

The website http://pata.physics.wtf (see image **??**) embodies the knowledge of this doctoral research and showcases AMC and patalgorithms. This chapter gives an overview of the structure of the website and the development process.

A high level view of the site would be that it is a pataphysical search engine that subverts conventional expectations by recombining literary texts into emergent user directed and ephemeral poetical structures or unpredictable spirals of pataphysicaled visual media.

It is written in 5 different programming languages[1], making calls to 6 external Web services[2], in a total of over 3000 lines of code[3] spread over 30 files.

Typically, software development is divided into so-called front- and back-ends. The front-end includes web design and web development and is meant to provide an interface for the end-user to communicate with the back-end which involves a server, an application and a database (although this is not directly the case in this project).

The front-end design uses the **W3.CSS** stylesheet as a basis. The website is mostly responsive, meaning it can be viewed well on phones, tablets and desktop screens (the poems and image spirals for example unfortunately have a fixed width which does not scale down well). The site contains various scripts written in **JavaScript** (e.g. scramble letters, randomise poem, send email and tabbed content).

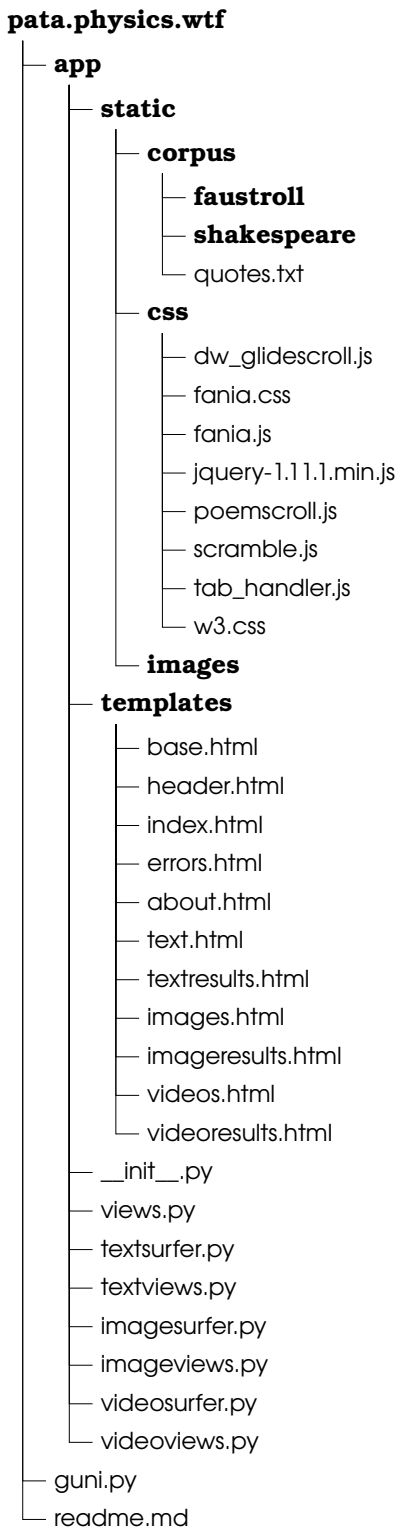


Figure 1.2 – Project directory

[1] Python, HTML, CSS, Jinja, JavaScript
[2] Microsoft Translate, WordNet, Bing, Getty, Flickr and YouTube
[3] 2864 lines of code, 489 lines of comments - as of 08 Dec 2015

Go to TOC

The backend relies heavily on a **Python** framework called **Flask**. Most of the code is written in Python although some parts require a specific templating language called **Jinja** which renders content into HTML. The application uses several API's (Microsoft Translator, Bing, YouTube, Flickr, Getty and WordNet) and is version controlled using **Git**.[4]

1.3 The folder structure is shown in figure 1.3.

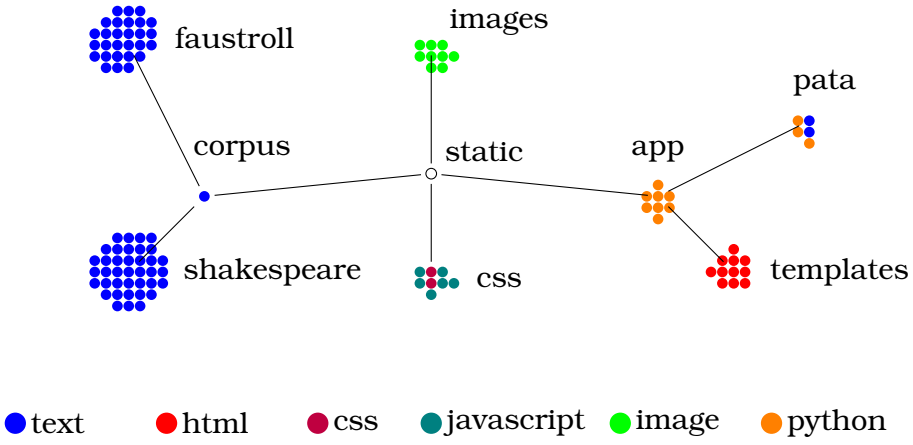


Figure 1.3 – Folder structure

1.4 & 1.5 Figures 1.4 and 1.5 show the two main workflow scenarios of `pata.physics.wtf` in the form of sequence diagrams. The columns are labeled with the main agents (this includes the user and the various main files responsible for key actions in the system). Going down vertically represents time.

Figure 1.4 demonstrates an outline of how the text search process works. A user enters a query which into a search box in the `text.html` file which is rendered by the `textviews.py` file. From there it gets forwarded to the `textsurfer.py` file which then handles the pataphysicalisation process and returns patadata back to the `textviews.py` file. The python file then forwards to the `textresults.html` file which retrieves and renders the results to the user. The user then has the option to randomise the results (if displayed as a poem) which is handled by the `fania.js` file. A very similar process is in place for image and video search as shown in figure 1.5. The main difference is the results are retrieved in the `fania.js` file rather than the `imgresults.html` file.

Putting it another way, (1) the system setup tokenises each of the source texts, removes stopwords and then adds terms and their location to the index (see § 1.1.2 section 1.1.2), (2) a query then triggers the three pataphysical algorithms, (3)

[4]Backend links: https://www.python.org/, http://flask.pocoo.org/, http://jinja.pocoo.org/, https://git-scm.com/

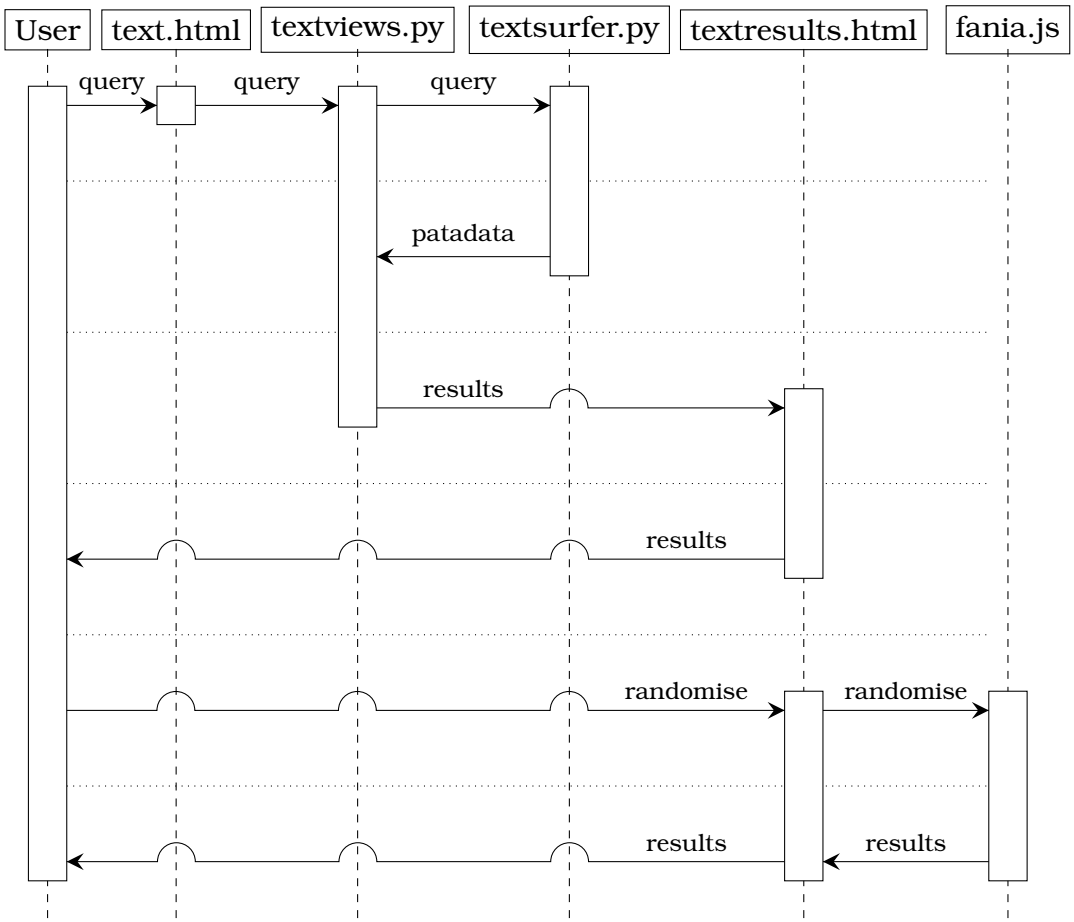


Figure 1.4 – Top-level overview of text search

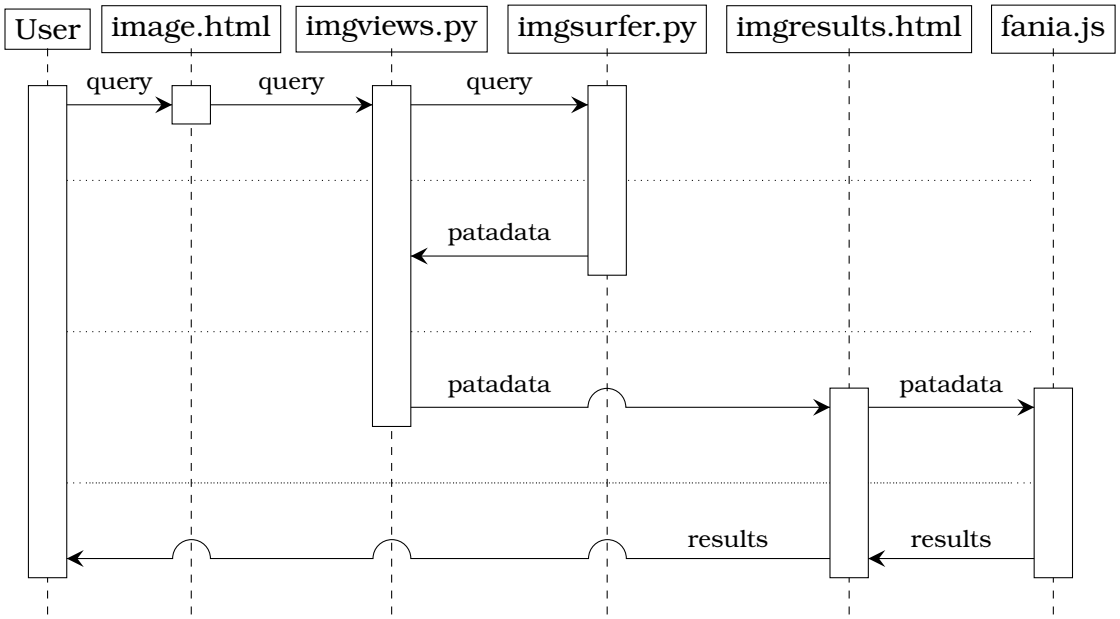


Figure 1.5 – Top-level overview of image / video search

Go to TOC

§ 1.2 each algorithm finds results for the query (see section 1.2), (4) some words before/after the match are retrieved for context, and (5) the resulting sentences are rendered for the user.

This chapter explains how pata.physics.wtf was created and how it operates technically. Specifically it will discuss the intitial setup of the system when it is first started up, the text search algorithms, the image and video Application Program Interface (API) calls and the main design elements (text poetry and image spirals).

1.1 Setup

The Python web framework Flask (**Ronacher2016**) looks after loading and rendering the various pages for pata.physics.wtf (home, text-search, text-results, image-search, image-results, video-search, video-results, about and errors), which means most of the backend related code is written in Python. Although Flask contains a small development server, in a production environment a more capable server is needed. For this reason the Flask site runs on a Gunicorn server (**Gunicorn2016**) and is hosted on a UNIX machine.

1.1.1 Corpora

Instead of crawling the Internet pata.physics.wtf uses a local collection of texts for its text search. Setting up a custom web crawler would require a lot more resources (in terms of hardware, time and money) than practical for this project. There are two corpora containing 65 text files together.

The first corpus resembles the fictional library of 'equivalent books' from Alfred Jarry's *Exploits and Opinions of Dr. Faustroll, 'Pataphysician* (1996). In principle the corpus is just a folder within the tool's directory structure which contains the following files:

0. Alfred Jarry: *Exploits and Opinions of Dr. Faustroll, 'Pataphysician*
1. Edgar Allen Poe: *Collected Works*
2. Cyrano de Bergerac: *A Voyage to the Moon*
3. Saint Luke: *The Gospel*
4. Leon Bloy: *Le Desespere* (French)
5. Samuel Taylor Coleridge: *The Rime of the Ancient Mariner*
6. Georges Darien: *Le Voleur* (French)

Go to TOC

7. Marceline Desbordes-Valmore: *Le Livre des Meres et des Enfants* (French)
8. Max Elskamp: *Enluminures* (French)
9. Jean-Pierre Claris de Florian: *Les Deux Billets* (French)
10. *One Thousand and One Nights*
11. Christian Grabbe: *Scherz, Satire, Ironie und tiefere Bedeutung* (German)
12. Gustave Kahn: *Le Conte de l'Or et Du Silence* (French)
13. Le Comte de Lautreamont: *Les Chants de Maldoror* (French)
14. Maurice Maeterlinck: *Aglavaine and Selysette*
15. Stephane Mallarme: *Verse and Prose* (French)
16. Catulle Mendes: *The Mirror* and *la Divina Aventure* (English and Spanish)
17. Homer: *The Odyssey*
18. Josephin Peladan: *Babylon* (EMPTY FILE)[5]
19. Francois Rabelais: *Gargantua and Pantagruel*
20. Jean de Chilra: *L'Heure Sexuelle* (EMPTY FILE)[5]
21. Henri de Regnier: *La Canne de Jaspe* (EMPTY FILE)[5]
22. Arthur Rimbaud: *Poesies Completes* (French)
23. Marcel Schwob: *Der Kinderkreuzzug* (German)
24. Alfred Jarry: *Ubu Roi* (French)
25. Paul Verlaine: *Poems*
26. Emile Verhaeren: *Poems*
27. Jules Verne: *A Journey to the Centre of the Earth*

§ **??** The original list as it appears in 'Faustroll' is shown in chapter **??**. Three of the items have not been found as a resource. Some others have been approximated by using another text by the same author for example. Most of these were sourced from ***Project Gutenberg*** (**Gutenberg2016**) in their original languages. The decision to get foreign language texts was partially due to the lack of out-of-copyright translated versions and partially because the original library in 'Faustroll' was also multi-lingual.

A note on copyright: UK copyright law states in section 5 that the duration of copyright for "literary, dramatic, musical or artistic works" is "70 years from the end of the calendar year in which the last remaining author of the work dies" (**Copyright2015**). Maurice Maeterlinck and Marguerite Vallette-Eymery (a.k.a. Rachilde or Jean de Chilra) died less than 70 years ago and their work should still be under copyright. Alfred Jarry in the Simon Watson Taylor translation is a derivative work and is probably also still protected. However, copyright does not apply when used for "private and research study purposes" as stated in section 7 on *Fair dealing* of (**Copyright2016**).

[5] I have not been able to find any source texts online.

Go to TOC

The second corpus is a collection of 38 texts by William Shakespeare (**Shakespeare2011**).

1. *The Sonnets*
2. *Alls Well That Ends Well*
3. *The Tragedy of Antony and Cleopatra*
4. *As You Like It*
5. *The Comedy of Errors*
6. *The Tragedy of Coriolanus*
7. *Cymbeline*
8. *The Tragedy of Hamlet, Prince of Denmark*
9. *The First Part of King Henry the Fourth*
10. *The Second Part of King Henry the Fourth*
11. *The Life of Kind Henry the Fifth*
12. *The First Part of Henry the Sixth*
13. *The Second Part of Henry the Sixth*
14. *The Third Part of Henry the Sixth*
15. *King Henry the Eigth*
16. *King John*
17. *The Tragedy of Julius Caesar*
18. *The Tragedy of King Lear*
19. *Love's Labour's Lost*
20. *The Tragedy of Macbeth*
21. *Measure for Measure*
22. *The Merchant of Venice*
23. *The Merry Wives of Windsor*
24. *A Midsummer Night's Dream*
25. *Much Ado About Nothing*
26. *The Tragedy of Othello, Moor of Venice*
27. *King Richard the Second*
28. *Kind Richard III*
29. *The Tragedy of Romeo and Juliet*
30. *The Taming of the Shrew*
31. *The Tempest*
32. *The Life of Timon of Athens*
33. *The Tragedy of Titus Andronicus*
34. *The History of Troilus and Cressida*
35. *Twelfth Night or What You Will*
36. *The Two Gentlemen of Verona*
37. *The Winter's Tale*
38. *A Lover's Complaint*

Go to TOC

1.1.2 Index

When the server is first started various setup functions (such as the creation of the index) are executed before any HTML is rendered. The search algorithms are triggered once a user enters a search term into the query field on any of the text, image or video pages.

Each plain text file in the corpus is added to the internal library one by one. Source 1.1 shows how this is done. The `PlaintextCorpusReader` is a feature of the Natural Language Tool Kit (NLTK) Python library (Project 2016) for Natural Language Processing. The `words` function tokenises the text, that is it splits it into individual words and stores them as an ordered list.

```
1 library = PlaintextCorpusReader(corpus_root, '.*\.txt')
2 l_00 = library.words('00.faustroll.txt')
3 l_01 = library.words('01.poe1.txt')
4 ...
5 l_27 = library.words('27.verne.txt')
```

Code 1.1 – Adding text files to the corpus library

The `setupcorpus` function (see source 1.2) is called for each of the text files in the two corpora to populate the index data structures `l_dict` (for the Faustroll vocabulary) and `s_dict` (for the Shakespeare vocabulary).

```
dict = dictionary { dictionary { list [ ] } }
```

A dictionary in Python is what is known as an 'associative array' in other languages. Essentially they are unordered sets of ***key: value*** pairs. The `dict` used here is a dictionary where each key has another dictionary as it's value. Each nested dictionary has a list as the value for each key.

Line 7 in source 1.2 starts looping through file `f`. Line 8 checks if the current word `w` contains anything other than alphabetical characters and whether or not `w` is contained in the relevant stopword file `lang` (for a list of English stopwords see appendix **??**). If both of those conditions are true, a variable `y` is created on line 9 (such as 'l_00' based on '00.faustroll.txt') and `w` is added to the relevant dictionary file `dic` together with `y` and the current position `x` on line 10. After all files are processed, the two index strcutures look roughly like this:

```
{
  word1: {fileA: [pos1, pos2, ...], fileB: [pos], ...},
```

Go to TOC

```
# f = input text
# lang = stopwords
# dic = dictionary
# d = l for Faustroll or s for Shakespeare
def setupcorpus(f, lang, dic, d):
  # x = counter, w = word in file f
  for x, w in enumerate(f):
    if w.isalpha() and (w.lower() not in lang):
      y = d + '_' + (re.search(r"((\d\d).(\w)+.txt)",
       ↪  f.fileid)).group(2)
      dic[w.lower()][y].append(x)
```

Code 1.2 – 'setupcorpus': processing a text file and adding to the index

```
  word2: {fileC: [pos1, pos2], fileK: [pos], ...},
  ...
}
```

Using one of the terms from figure **??** on page **??** as an example, here are their entries in the index file (the files are represented by their number in the corpus, i.e. `l_00` is the 'Faustroll' file, `l_01` is the 'Poe' file, etc.). An excerpt from the actual `l_dict` can be found in the appendix **??**.

```
{
  doctor: {
    l_00: [253, 583, 604, 606, 644, 1318, 1471, 1858, 2334, 2431,
          ↪  2446, 3039, 4743, 5034, 5107, 5437, 5824, 6195, 6228,
          ↪  6955, 7305, 7822, 7892, 10049, 10629, 11055, 11457,
          ↪  12059, 13978, 14570, 14850, 15063, 15099, 15259,
          ↪  15959, 16193, 16561, 16610, 17866, 19184, 19501,
          ↪  19631, 21806, 22570, 24867],
    l_01: [96659, 294479, 294556, 294648, 296748, 316773, 317841,
          ↪  317854, 317928, 317990, 318461, 332118, 338470,
          ↪  340548, 341252, 383921, 384136, 452830, 453015,
          ↪  454044, 454160, 454421, 454596, 454712, 454796,
          ↪  454846, 455030, 455278, 455760, 455874, 456023,
          ↪  456123, 456188, 456481, 456796, 457106, 457653,
          ↪  457714, 457823, 457894, 458571, 458918, 458998,
          ↪  459654, 459771, 490749],
    l_02: [11476, 12098, 28151, 36270],
    l_10: [53085, 53118, 53220, 53266, 53364, 53469, 53573, 53592,
          ↪  53621, 53718, 54873, 55262, 55525, 55577, 55614,
          ↪  55683, 55741, 56058, 62709, 113969, 114131, 114405,
          ↪  114794],
```

```
    l_19: [14928, 15702, 49560, 82710, 167218, 180210, 189817,
          ↪  189908, 190020, 190235, 190905, 199430, 226663,
          ↪  275454, 275928, 278097, 287375, 291383, 304731,
          ↪  306055, 324757, 330488],
    l_27: [16270, 79245]
  }, ...
}
```

1.2 TEXT

After the setup stage is completed and the webpage is fully loaded, user input in the form of a text query is required to trigger the three pataphysical algorithms.

Image and Video search do not use all three algorithms — where relevant this is highlighted in each section. Generally the following descriptions refer to the text search functionality only.

 1.4 Figure 1.4 previously showed the rough sequence of events in text search and highlighted that the pataphysicalisation from query to patadata happens in the `textsurfer.py` Python script file.

1.2.1 CLINAMEN

§ **??** The clinamen was introduced in chapter **??** but to briefly summarise it, it is the unpredictable swerve that Bök calls "the smallest possible aberration that can make the greatest possible difference" (**Boek2002**).

> Like all digitally encoded information, it has unavoidably the uncomfortable property that the smallest possible perturbations —i.e. changes of a single bit— can have the most drastic consequences. (**Dijkstra1988**)

In simple terms, the clinamen algorithm works in two steps:

1. get clinamen words based on dameraulevenshtein and faustroll,
2. get sentences from corpus that match clinamen words.

It uses the *Faustroll* text by Alfred Jarry (1996) as a base document and the Damerau-Levenshtein algorithm (**Damerau1964**; **Levenshtein1966**) (which measures the distance between two strings (with 0 indicating equality) to find words that are similar but not quite the same. The distance is calculated using insertion, deletion, substitution of a single character, or transposition of two adjacent characters. This means that we are basically forcing the program to return

matches that are of distance two or one, meaning they have two or one spelling errors in them.

```
1  # w = query word
2  # c = corpus
3  # i = assigned distance
4  def clinamen(w, c, i):
5    # l_00 = Faustroll text
6    words = set([term for term in l_00 if dameraulevenshtein(w, term) <=
       ↪  i])
7    out, sources, total = get_results(words, 'Clinamen', c)
8    return out, words, sources, total
```

Code 1.3 – 'clinamen': pataphysicalising a query term

</> 1.3 Source 1.3 line 6 creates the set of clinamen words using a list comprehension. It retrieves matches from the Faustroll file `l_00` with the condition that they are of
</> 1.4 Damerau-Levenshtein distance `i` or less to the query term `w` (see source 1.4). Duplicates are removed. Line 7 then makes a call to the generic `get_results` function to get all relevant result sentences, the list of source files and the total number of results.

```
1  # Michael Homer 2009
2  # MIT license
3  def dameraulevenshtein(seq1, seq2):
4    oneago = None
5    thisrow = range(1, len(seq2) + 1) + [0]
6    for x in xrange(len(seq1)):
7      twoago, oneago, thisrow = oneago, thisrow, [0] * len(seq2) + [x + 1]
8      for y in xrange(len(seq2)):
9        delcost = oneago[y] + 1
10       addcost = thisrow[y - 1] + 1
11       subcost = oneago[y - 1] + (seq1[x] != seq2[y])
12       thisrow[y] = min(delcost, addcost, subcost)
13       if (x > 0 and y > 0 and seq1[x] == seq2[y - 1] and
14         seq1[x - 1] == seq2[y] and seq1[x] != seq2[y]):
15           thisrow[y] = min(thisrow[y], twoago[y - 2] + 1)
16   return thisrow[len(seq2) - 1]
```

Code 1.4 – Damerau-Levenshtein algorithm by (**Homer2009**)

The clinamen algorithm mimics the unpredictable swerve, the smallest possible aberration that can make the greatest possible difference, or the smallest possible perturbations with the most drastic consequences.

Go to TOC

```
# words = patadata words
# algo = name of algorithm
# corp = name of corpus
def get_results(words, algo, corp):
  total = 0
  out, sources = set(), set()
  for r in words:
    if corp == 'faustroll': files = l_dict[r]
    else: files = s_dict[r]
    # e = current file
    # p = list of positions for term r in file e
    for e, p in files.items():
      f = get_title(e)
      sources.add(f)
      o = (f, pp_sent(r.lower(), e, p), algo)
      total += 1
      out.add(o)
  return out, sources, total
```

Code 1.5 – 'get_results': retrieving all sentences for a list of words

</> 1.5 The `get_results` function (see source 1.5) is used by all three algorithms (clinamen, syzygy and antinomy). Given the nested structure of the indexes `l_dict` and `s_dict`, the function loops through each of the `words` passed to it (`r`) first and then each file in `files`. Lines 8 and 9 retrieve the dictionary of files for term `r` from the relevant dictionary. Line 13 gets the author and full title of file `e` and adds it to the list of sources in line 14. Line 15 makes use of another
</> 1.6 function called `pp_sent` (see source 1.6) to get an actual sentence fragment for the current word `r` in file `e`, which is then added to the output. The output is structured as a triple containing the author and title, the list of resulting sentences and the name of the algorithm used.

</> 1.6 In function `pp_sent` (source 1.6) line 5 is important to note because it is a key functionality point. Even though the index files store a full list of all possible positions of a given word in each file, the `pp_sent` function only retrieves the sentence of the very first occurance of the word rather than each one. This decision was taken to avoid overcrowding of results for the same keyword.

Line 8 creates a list of punctuation marks needed to determine a suitable sentence fragment. Lines 9–17 and 18–26 set the `pos_b` (position before) and `pos_a` (position after) variables respectively. These positions can be up to 10 words before and after the keyword `w` depending on the sentence structure (punctuation

Go to TOC

```
1  # w = the word (lower case)
2  # f = the file
3  # p = the list of positions
4  def pp_sent(w, f, p):
5    out, pos = [], p[0] # FIRST OCCURENCE
6    ff = eval(f)
7    pos_b, pos_a = pos, pos
8    punct = [',', '.', '!', '?', '(', ')', ':', ';', '\n', '-', '_']
9    for i in range(1, 10):
10     if pos > i:
11       if ff[pos - i] in punct:
12         pos_b = pos - (i - 1)
13         break
14       else:
15         if ff[pos - 5]: pos_b = pos - 5
16         else:           pos_b = pos
17     else: pos_b = pos
18   for j in range(1, 10):
19     if (pos + j) < len(ff):
20       if ff[pos + j] in punct:
21         pos_a = pos + j
22         break
23       else:
24         if ff[pos + j]: pos_a = pos + j
25         else:           pos_a = pos
26     else: pos_a = pos
27   if pos_b >= 0 and pos_a <= len(ff):
28     pre = ' '.join(ff[pos_b:pos])
29     post = ' '.join(ff[pos+1:pos_a])
30     out = (pre, w, post)
31   return out
```

Code 1.6 – 'pp_sent': retrieving one sentence

marks). In line 28 the actual sentence fragment up to the keyword is retrieved, while in line 29 the fragment just after the keyword is retrieved. `ff[pos_b:pos]` for example returns the list of words from position `pos_b` to position `pos` from file `ff`. The built-in Python `.join()` function then concatenates these words into one long string separated by spaces. On line 30 a triple containing the pre-sentence, keyword and post-sentence is set as the output and then returned.

1.2.2 Syzygy

§ **??** The syzygy was introduced in chapter **??** but can be roughly described as surprising and confusing. It originally comes from astronomy and denotes the alignment of three celestial bodies in a straight line. In a pataphysical context it is

Go to TOC

the pun. It usually describes a conjunction of things, something unexpected and surprising. Unlike serendipity, a simple chance encounter, the syzygy has a more scientific purpose.

In simple terms, the syzygy algorithm works in two steps:

1. get syzygy words based on synsets and hypo-, hyper- and holonyms from WordNet,
2. get sentences from corpus that match syzygy words.

The syzygy function makes heavy use of WordNet (Miller 1995) through the NLTK Python library (Project 2016) to find suitable results (`from nltk.corpus import wordnet as wn`
</> 1.7 Specifically, as shown in source 1.7, the algorithm fetches the set of synonyms (synsets) on line 5. It then loops through all individual items `ws` in the list of synonyms `wordsets` in line 7–20. It finds any hyponyms, hypernyms or holonyms for `ws` (each of which denotes some sort of relationship or membership with its
</> 1.8 parent synonym) using the `get_nym` function (see lines 8, 11, 14, and 17). Line
</> 1.5 21 makes use of the `get_results` function (see source 1.5) in the same was as the clinamen function does.

```
1  # w = word
2  # c = corpus
3  def syzygy(w, c):
4    words, hypos, hypers, holos, meros = set(),set(),set(),set(),set()
5    wordsets = wn.synsets(w)
6    hypo_len, hyper_len, holo_len, mero_len, syno_len = 0,0,0,0,0
7    for ws in wordsets:
8      hypos.update(get_nym('hypo', ws))
9      hypo_len += len(hypos)
10     words.update(hypos)
11     hypers.update(get_nym('hyper', ws))
12     hyper_len += len(hypers)
13     words.update(hypers)
14     holos.update(get_nym('holo', ws))
15     holo_len += len(holos)
16     words.update(holos)
17     meros.update(get_nym('mero', ws))
18     mero_len += len(meros)
19     words.update(meros)
20     syno_len += 1
21   out, sources, total = get_results(words, 'Syzygy', c)
22   return out, words, sources, total
```

Code 1.7 – 'syzygy': pataphysicalising a query term

Go to TOC

</> 1.8 The `get_nym` function in source 1.8 shows how the relevant 'nyms' are retrieved for a given synset. Line 5 initialises the variable `hhh` which gets overwritten later on. Several `if` statements seperate out the code run for the different 'nyms'. Lines 6-7 retrieves any hyponyms using NLTK's `hyponyms()` function . Similarly lines 8-9 retrieve hypernyms, lines 10-14 retrieve holonyms, and lines 15-19 retrieve meronyms. Finally, line 23 adds the contents of `hhh` to the output of the function.

```
1  # nym = name of nym
2  # wset = synset
3  def get_nym(nym, wset):
4    out = []
5    hhh = wset.hyponyms()
6    if nym == 'hypo':
7      hhh = wset.hyponyms()
8    if nym == 'hyper':
9      hhh = wset.hypernyms()
10   if nym == 'holo':
11     hhhm = wset.member_holonyms()
12     hhhs = wset.substance_holonyms()
13     hhhp = wset.part_holonyms()
14     hhh = hhhm + hhhs + hhhp
15   if nym == 'mero':
16     hhhm = wset.member_meronyms()
17     hhhs = wset.substance_meronyms()
18     hhhp = wset.part_meronyms()
19     hhh = hhhm + hhhs + hhhp
20   if len(hhh) > 0:
21     for h in hhh:
22       for l in h.lemmas():
23         out.append(str(l.name()))
24   return out
```

Code 1.8 – 'get_nym': retrieving hypo/hyper/holo/meronyms

The syzygy algorithm mimics an alignment of three words in a line (query → synonym → hypo/hyper/holo/meronym).

1.2.3 Antinomy

The antimony, in a pataphysical sense, is the mutually incompatible. It was
§ ?? previously introduced in chapter **??**.

In simple terms, the antinomy algorithm works in two steps:

1. get antinomy words based on synsets and antonyms from WordNet,

2. get sentences from corpus that match antinomy words.

```
1  # w = input query term
2  # c = name of corpus
3  def antinomy(w, c):
4    words = set()
5    wordsets = wn.synsets(w)
6    for ws in wordsets:
7      anti = ws.lemmas()[0].antonyms()
8      if len(anti) > 0:
9        for a in anti:
10         if str(a.name()) != w:
11           words.add(str(a.name()))
12   out, sources, total = get_results(words, 'Antinomy', c)
13   return out, words, sources, total
```

Code 1.9 – 'antinomy': pataphysicalising a query term

</> 1.9 For the antinomy we simply used WordNet's antonyms (opposites) (see source 1.9). This function is similar to the algorithm for the syzygy. It finds all antonyms through NLTK's `lemmas()[0].antonyms()` function on line 7 and retrieves result
</> 1.5 sentences using the `get_results` function on line 12.

The antinomy algorithm mimics the mutually incompatible or polar opposites.

1.2.4 Formalisation

A formal description of the `pata.physics.wtf` system in terms of an Information
§ ?? Retrieval (IR) model described in chapter **??** is unsuitable. It assumes for example the presence of some sort of ranking algorithm $R(q_i, d_j)$.

Making relevant changes to the specification by Baeza-Yates and Ribeiro-Neto (2011), an approximate system description for the Faustroll corpus text search could be as follows.

We can then define the three patalgorithms in a more formal way as shown in equations 1.1, 1.2, and 1.3.

$$P(q)_C = \{p \in v_1 : 0 < \text{dameraulevenshtein}(q, p) \leq 2\} \tag{1.1}$$

`dameraulevenshtein(q,p)` in equation 1.1 is the Damerau-Levenshtein algorithm as described in section 1.4.4 and v_0 is the Faustroll text.

Go to TOC

D = the set of documents $\{d_1, \ldots, d_m\}$
m = the number of all documents in D ($|D| = 28$)
V = the set of all distinct terms $\{v_1, \ldots, v_n\}$ in D not including stopwords
q = the user query
F = the set of patalgorithms $\{f_C, f_S, f_A\}$
P = the set of pataphysicalised query terms $\{p_1, \ldots, p_u\}$
u = the number of terms in P
$P(q)$ = the set of patadata $\{P(q)_C \cup P(q)_S \cup P(q)_A\}$ for query q
R = the set of results $\{r_1, \ldots, r_o\}$
o = the number of results in R
$R(P(q))$ = the set of results $\{R(P(q)_C) \cup R(P(q)_S) \cup R(P(q)_A)\}$ produced by each algorithm in F
r = a result of form (d, sentence, f)

$$P(q)_S = \{p \in V : p \in \text{nyms}(s),\ \forall s \in \text{synonyms}(q)\}$$
$$\text{where nyms}(s) = \text{hypos}(s)\ \cup\ \text{hypers}(s)\ \cup\ \text{holos}(s)\ \cup\ \text{meros}(s) \tag{1.2}$$

Σ 1.2 `synonyms(q)` in equation 1.2 is the WordNet/NLTK function to retrieve all synsets for the query q and the four 'nym' functions return the relevant hyponyms, hypernyms, holonyms or meronyms for each of the synonyms.

$$P(q)_A = \{p \in V : p \in \text{antonyms}(s),\ \forall s \in \text{synonyms}(q)\} \tag{1.3}$$

Σ 1.3 Similarly, in equation 1.3 the `synonyms(q)` function returns WordNet synsets for the query q and the `antonyms(s)` function returns WordNet antonyms for each of the synonyms.

$$R(P(q)) = \{(d \in D,\ sent(p) \in d,\ f \in F) : \forall\ p \in P(q)_f))\} \tag{1.4}$$

The set of results $R(P(q))$ can then be defined as shown in equation 1.4. It returns a list of triples containing the source text (d), the sentence `sent(p)` and the algorithm f. For each pataphysicalised query term p one sentence is retrieved per file d.

Σ 1.4

Go to TOC

1.3 Image & Video

The image and video search of pata.physics.wtf both work slightly differently to the text search described in section 1.2.

In simple terms, the image and video search works in three steps:

1. translate query
2. pataphysicalise the translation
3. retrieve matching images/videos using API calls

The first step is to translate the search terms as shown in source 1.10. Lines 2 and 4 set up the API connection to the Microsoft Translator tool (**TranslatorAPI**) given an ID and 'secret', neither of which are included here for security reasons. The query `sent` then passes through a chain (alignment) of three translations in true syzygy fashion: from English → French, from French → Japanese, and from Japanese → English (lines 5-7). All three languages are then returned in a triple (line 9).

```
1  # sent = the query string
2  from microsofttranslator import Translator
3  def transent(sent):
4    translator = Translator(microsoft_id, microsoft_secret)
5    french = translator.translate(sent, "fr")
6    japanese = translator.translate(french, "ja")
7    patawords = translator.translate(japanese, "en")
8    translations = (french, japanese, patawords)
9    return translations
```

Code 1.10 – 'transent': translating query between English-French-Japanese-English

The next step is to pataphysicalise the translated query (see source 1.11). The `pataphysicalise` function transforms this translation in a process slighlty simplified from the `syzygy` algorithm. The decision to simplify the algorithm was made due to performance issues related to the API calls that follow in the final step of the search process.

In line 5 WordNet synsets are retrieved using NLTK's `synsets` function. For each of these synsets we get a list of synonyms (line 8) which we add to the output in a normalised form (line 11).

Figure 1.5 previously showed the rough sequence of events in an image and

```
# words = query term(s)
def pataphysicalise(words):
  sys_ws = set()
  for word in words:
    synonyms = wn.synsets(word)
    if len(synonyms) > 0:
      for s in synonyms:
        for l in s.lemmas():
          x = str(l.name())
          o = x.replace('_', ' ')
          sys_ws.add(o)
  return sys_ws
```

Code 1.11 – 'pataphysicalise': pataphysicalise image and video query terms

video search and highlighted that the pataphysicalisation from query to patadata happens in the `imgsurfer.py` Python script file while the production of results from that patadata happens in the `fania.js` JavaScript file.

And finally, API calls to the various external tools are made. This is described in § 1.3.1section 1.3.1 below.

1.3.1 REST & API

The final step of the image and video search process described on page 24 is to retrieve matching images/videos using API calls to Flickr (**FlickrAPI**; **FlickrGuideAPI**), Getty (**GettyAPI**; **GettyOverviewAPI**), Bing (**BingAPI**; **BingAzureAPI**), YouTube (**YouTubeAPI**) and Microsoft Translator (**TranslatorAPI**).

The patadata used to make the API calls is limited to 10 keywords and uses the function `random.sample(pata, 10)`, where `pata` is the set of terms obtained by pataphysicalising the query translation.

A RESTful API allows browsers ('clients') to communicate with a web server via HTTP methods such as GET and POST. The idea is that a given service, like the Microsoft Bing search API, can be accessed in a few simple steps using JavaScript Object Notation (JSON) (**JSON2016**). These are:

1. for each of the 10 query terms do:
   a) construct the Uniform Resource Locator (URL) with the query request
   b) setup authentication
   c) send URL and authentication
   d) receive response in JSON

e) add result to output list `imglist`

2. once 10 results are reached, render results as spiral

</> 1.12 Source 1.12 shows how such an API call is made using JavaScript. Figure 1.5 previously showed the rough sequence of events in an image or video search and highlighted that the production of results given some patadata happens in the `fania.js` JavaScript file. Source 1.13 below shows how 10 seperate images are collected into one results list and the `createSpiral` function is called to render the images to the user in HTML.


```
1  function flickrsearch(patadata){
2    for(var x=0; x<10; x++){
3      \$.getJSON("http://api.flickr.com/services/feeds/photos_public.gne
       ↪  ?jsoncallback=?",
4        {
5          tags: patadata[x].query,
6          tagmode: "all",
7          format: "json"
8        },
9        function(data,status,ajax) {
10         var title = "", media = "", link = "";
11         if (data.items[0] != undefined) {
12           title = data.items[0].title;
13           media = data.items[0].media.m;
14           link = data.items[0].link;
15         }
16         imgList([title, media, link]);
17       }
18     );
19   }
20 };
```

Code 1.12 – 'flickrsearch': using the Flickr API to retrieve images

The Bing and Getty searches work in a similar way with one exception. Getty does not populate the output list by doing 10 individual API calls but rather by adding 10 results from 1 call. This is due to a time restriction in the Getty API; it doesn not allow 10 calls in a second.

An example URL request for the Flickr image search with the query term of 'kittens' and a requested response format of JSON is this: `http://api.flickr.c`

Go to TOC

```
var allImages = [];
function imgList(img){
  if (allImages[0] != "") {
    allImages.push(img);
  }
  if (allImages.length === 10) {
    createSpiral(allImages);
  }
}
```

Code 1.13 – 'imgList': accumulates 10 images and calls the 'createSpiral' function

om/services/feeds/photos_public.gne?jsoncallback=?tags=kittens&tag mode=all&format=json. Flickr will then send back the response in JSON format. One entry of the list of results is shown below (with whitespace formatting added for convenience). The algorithm in source 1.12 only retrieves the `data.items[0].title`, `data.items[0].media.m` and `data.items[0].link` and ignores all other data fields.

</> 1.12

```
({...
  "items":
    [{
      "title": "P_20161101_191123",
      "link": "http://www.flickr.com/photos/pinknancy/30078720153/",
      "media": {"m":"http://farm6.staticflickr.com/5759/30078720153_
            ↪  _f03e036e89_m.jpg"},
      "date_taken": "2016-11-01T19:11:23-08:00",
      "description": ...,
      "published": "2016-11-01T15:28:10Z",
      "author": "nobody@flickr.com (pinknancy)",
      "author_id": "8748781@N08",
      "tags": ""
    },...]
})
```

Once the `imglist` contains 10 items it is passed to the `createSpiral` function which renders it to HTML.

The video search also uses an API to retrieve results. This function is written in Python and uses the *Requests* library to make the API calls to YouTube (**YouTubeAPI**) as shown in source 1.14.

</> 1.14

Go to TOC

First, the query is translated using the `transent` function mentioned in source 1.10 on line 3. Line 4 seperates the English translation into its own list `transplit` which is then pataphysicalised on line 5 using the algorithm described in source 1.11.

Lines 6–9 construct the first part of the URL to use for the Representational State Transfer (REST) request. Lines 10–23 then loop through each of the patadata terms generated by the `pataphysicalise` function on line 5 to make a call and retrieve some video details (title, thumbnail and ID) as seen on lines 18–20. On line 21 these details are added to the output list.

```
1  def getvideos(query):
2    out = []
3    translations = transent(query)
4    transplit = translations[2].split(' ')
5    tmp = pataphysicalise(transplit)
6    b0 = "https://www.googleapis.com/youtube/v3/search?"
7    b1 = "&order=viewCount&part=snippet&"
8    b3 = "&type=video&key=%s" % yt_key
9    b4 = "&maxResults=10&safeSearch=strict"
10   for x in tmp:
11     y = ' '.join(x)
12     b2 = "q=%s" % translations[2]
13     yturl = ''.join([b0, b1, b2, b3, b4])
14     vids = requests.get(yturl)
15     if vids.json()['items']:
16       for i in vids.json()['items']:
17         vidtitle = i['snippet']['title']
18         vidthumb = i['snippet']['thumbnails']['default']['url']
19         vidid = i['id']['videoId']
20         out.append((vidtitle, vidthumb, vidid))
21       break
22     else:
23       out = []
24   return out, translations
```

Code 1.14 – 'getvideos': using the YouTube API to retrieve images in Python

The video results are then also displayed in a golden spiral in the same way as the images. This is described in section 1.4.3.

1.4 Design

Once the three algorithms have produced their respective results, the page displaying these results can be rendered. This is done using the templating language Jinja (**Jinja2016**) and Hypertext Markup Language (HTML) (with Cascading

Stylesheets (CSS) stylesheets and some JavaScript).

§ ?? One of the key requirements for the *Syzygy Surfer* tool was that "the user should be able to choose the techniques they use" (Hendler and Hugill 2011). This has been adopted for `pata.physics.wtf` in the sense that the user has different options for the display of results.

Figure 1.6 – Design of `pata.physics.wtf`

The text results page has three different result styles, with 'Poetry — Queneau' being the default.

Poetry Displayed in sonnet style (two quatrains and two tercets) if possible, although no rhyming pattern is used.

- Queneau — Each line can be changed manually.
- Random — The whole poem can be randomised.

Sources Ordered by source text.

Algorithms Ordered by algorithm.

The image and video results pages work the same way. They both have two display options, with the 'Spiral' option being the default. The spirals are modelled on the idea of golden spirals (more precisely an approximation in the form of a Fibonacci spiral).

Go to TOC

Spiral Displayed as square images/videos in a spiral.
List Displayed as a simple list.

🖼 1.6 The overal visual design is shown in image 1.6.

1.4.1 Poetry

</> 1.15 Source 1.15 shows the segment of HTML/Jinja code that renders the Queneau poetry. The code renders the 4 stanzas of the poem. This is done using two nested Jinja 'for' loops (line 2 and line 10). Line 2 loops through the (ideally) 14 lines of the poem. `lol` can be considered a masterlist of all sublists for each poem line.

Functionality for sending the currently showing poem per email is added via a button which calls a JavaScript function `onclick=`return getContent(this)'` which then retrieves the content of each line in the poem and sends it to the body of the email.

`all_sens` is structured as `[(title, (pre, word, post), algorithm), ...]`. `lol` is structured as `[all_sens[0], all_sens[1], ... all_sens[x]]` where x is the number of possible sentences minus 1 per line in a poem.

```
1  <div>
2    {% for n in range(1, lol|length + 1) %}
3      {% set wid = [`wn', n|string]|join %}
4      {% set lid = [`lyr', n|string]|join %}
5      {% set sid = [`scrollLinks', n|string]|join %}
6      {% set aid = lol[n-1] %}
7      <div id=`poems'>
8        <div id=`{{wid}}' class=`wn'>
9          <div id=`{{lid}}' class=`lyr'>
10           {% for sens in aid %}<span title=`{{ sens[0] }}, {{ sens[2]
               ↪  }}'>{{ sens[1][0] }} <form class=`inform'
               ↪  action=`../textresults' method=`post'><input
               ↪  class=`inlink' type=`submit' name=`query' value=`{{
               ↪  sens[1][1] }}' onclick=`loading();'></input></form> {{
               ↪  sens[1][2] }}</span>{% endfor %}
11         </div>
12       </div>
13       <div id=`{{sid}}' class=`scrollLinks'></div>
14     </div>
15   {% endfor %}
16 </div>
```

Code 1.15 – Simplified HTML code for rendering Queneau style poems

Go to TOC

Changing a line of the poem is achieved by clicking on one of the buttons on either side of the poem's line (as shown in image 1.7). This will trigger a JavaScript function (based on (**DYNWEB2016**)) to automatically scroll to the next sentence.

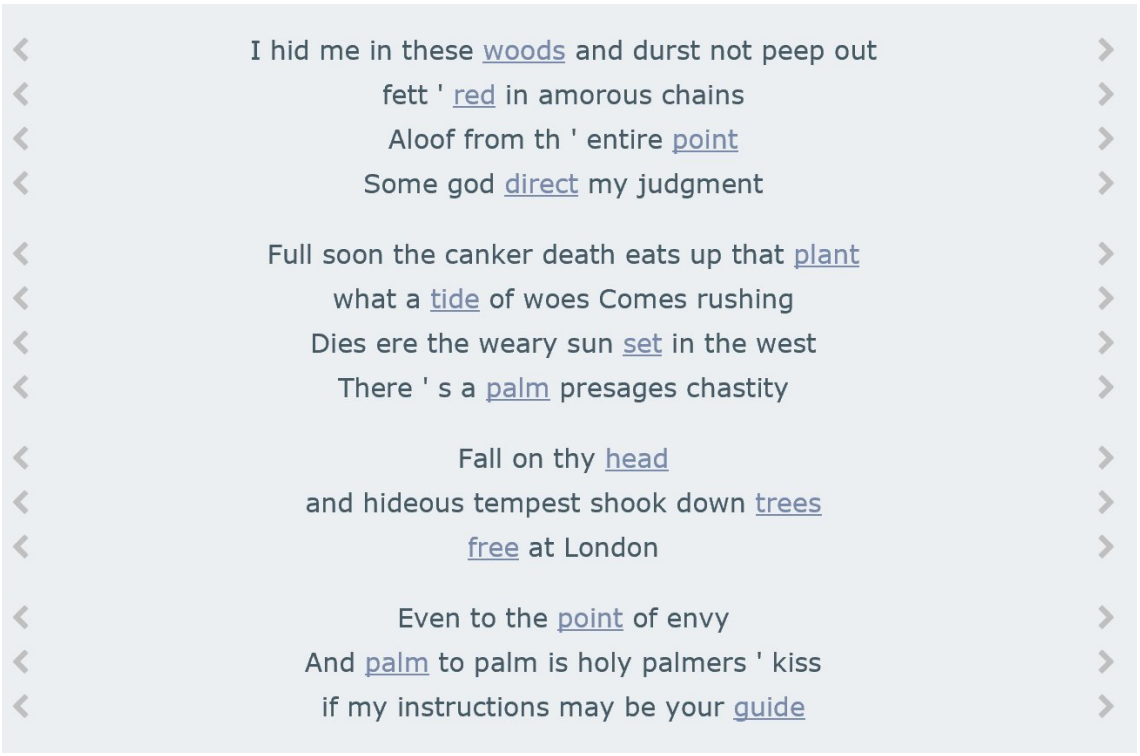


Figure 1.7 – Example Queneau poem for query 'tree'

Non-Queneau poems have a slightly different functionality. It is not possible to change the poem line by line but rather the whole poem can be randomised on demand. This relies on a random number generator in JavaScript. A function `shufflePoem()` creates a random variable `r` as `Math.floor(Math.random() * n)`, which can then be used to generate a new list of 14 lines for the poem randomly selected from the pool of sentences `all_sens`.

1.4.2 Lists

The two other ways to display text results are as a list ordered by source or by patalgorithm which works in a similar way to what is described in source 1.16. The code is wrapped in an HTML unordered list tag `<ul>`. A Jinja `for` loop generates the individual `<li>` tags on line 4.

A `sens` in `all_sens` is structured as `(title, (pre, word, post), algorithm)`. This means that to access the name of the algorithm we need to call the Jinja template `{{ sens[2] }}`, to get the first half of the sentence we need `{{ sens[1][0] }}`, the

Go to TOC

middle keyword (i.e. the patadata term) `{{ sens[1][1] }}` and the second half of the sentence `{{ sens[1][2] }}`.

```
<ul>
  {% for sens in all_sens %}
    {% if file == sens[0] %}
      <li title=`{{ sens[2] }}'>...{{ sens[1][0] }} <form class=`inform'
       ↪  action=`../textresults' method=`post'><input class=`w3-hide'
       ↪  type=`radio' name=`corpus' value=`{{ corpus }}'
       ↪  checked><input class=`inlink' type=`submit' name=`query'
       ↪  value=`{{ sens[1][1] }}' onclick=`loading();'></input></form>
       ↪  {{ sens[1][2] }}...</li>
    {% endif %}
  {% endfor %}
</ul>
```

Code 1.16 – Simplified HTML code for rendering a list of text results by source

1.8 Image 1.8 shows a shortened example set of results for query 'tree' ordered by source, that is, ordered by original file.

1.9 Image 1.9 shows a shortened example set of results for query 'tree' ordered by patalgorithm, that is, ordered by the algorithm which produced the patadata.

1.4.3 Spiral

§ ?? The image and video spirals are constructed in complicated nested HTML components. The code for generating an image spiral is shown in appendix **??**. The video spiral is constructed in a similar way but directly in the HTML file as opposed to in the JavaScript file.

§ ?? Generally, the idea was taken from the pataphysical ***grand gidouille*** (see chapter **??**) and represented as a Fibonacci spiral.

1.10 Figure 1.10 shows a spiral created using the Flickr image search for query 'blue mountains' overlaid with a white Fibonacci spiral to highlight the structure.

1.5 Prototypes

The final website `pata.physics.wtf` went through several iterations of development since it was first conceived in 2012. This included 3 major technical updates since the first prototype and 2 new visual re-designs.

Table 1.1 shows the main differences and similarities between the versions.

Go to TOC

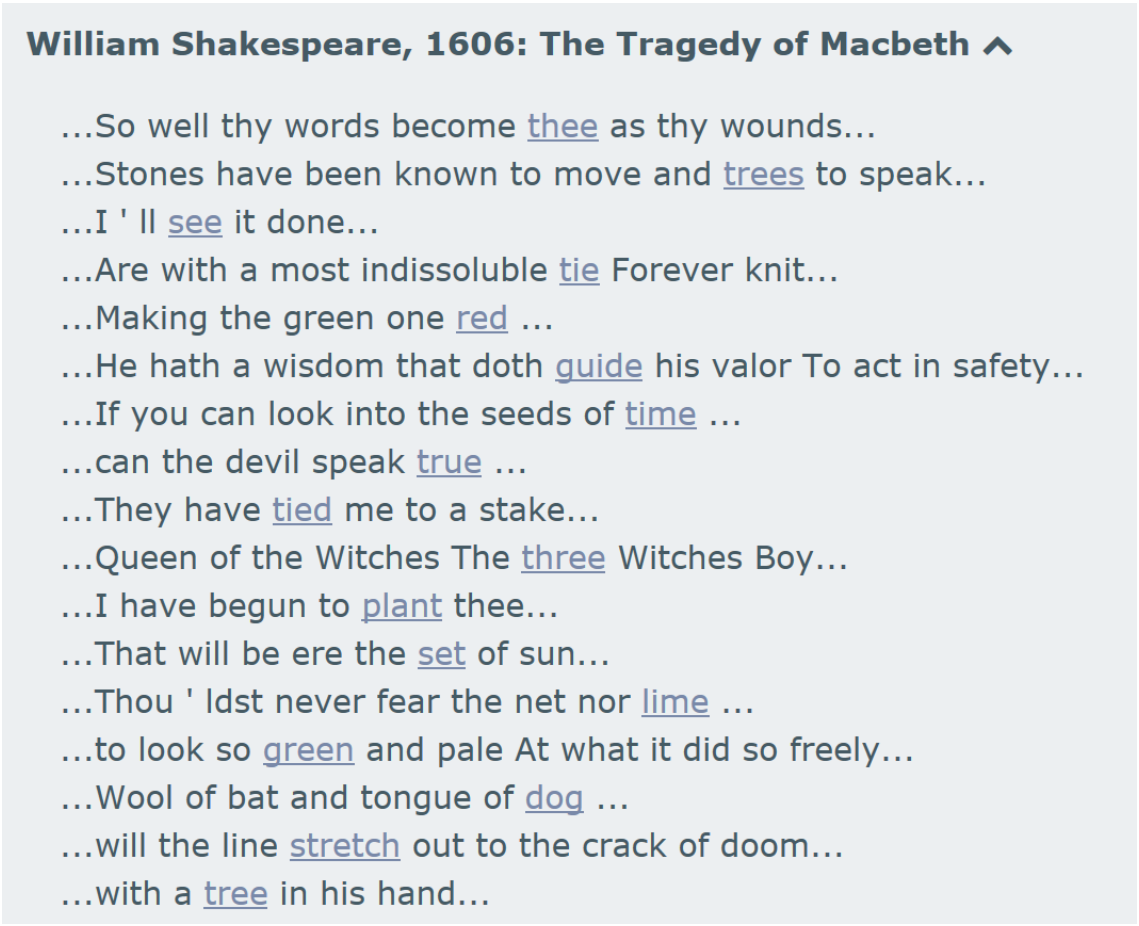


Figure 1.8 – Example results for query ‘tree’ ordered by source

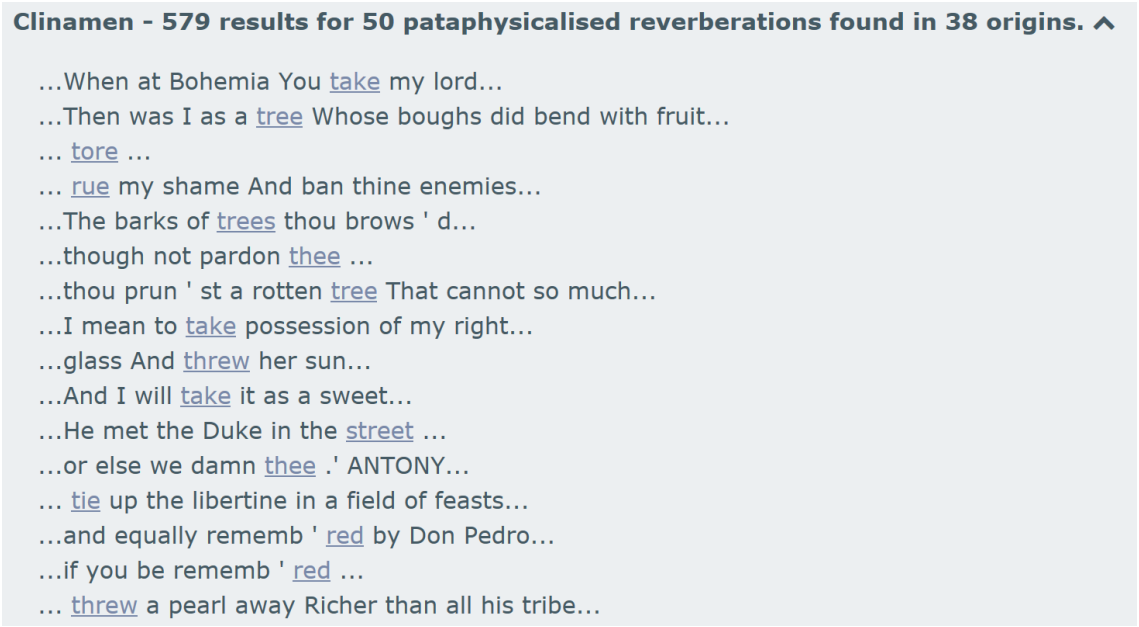


Figure 1.9 – Example results for query ‘tree’ ordered by patalgorithm

Go to TOC

Figure 1.10 – Fibonacci spiral overlaid onto an image results for query 'blue mountains' using Flickr

Table 1.1 – Comparison of different versions of `pata.physics.wtf`

	Version 1	**Version 2**	**Version 3**	**Version 4**
Language(s)	Python, Django	Python, Flask	Python, Flask	Python, Flask, JavaScript
Server	Django, Heroku	Flask, Mnemosyne	Flask, Gunicorn, Mnemosyne	Flask, Gunicorn, OVH
Features	Text	Text, Image, Video	Text, Image, Video	Text, Image, Video
Corpus	Faustroll text	Faustroll text	Faustroll's library	Faustroll's library and Shakespeare
API's	WordNet	WordNet, Flickr, Bing, YouTube, Microsoft Translator	WordNet, Bing, YouTube, Microsoft Translator	WordNet, Flickr, Getty, Bing, YouTube, Microsoft Translator
Design	Algorithms	Algorithms, Spiral	Algorithms, Source, Poetry, Spiral, List	Algorithms, Source, Poetry, Spiral, List
Responsive	No	Yes	Yes	Yes

Go to TOC

Images 1.11, 1.12 and 1.6 show the 3 main visual designs.

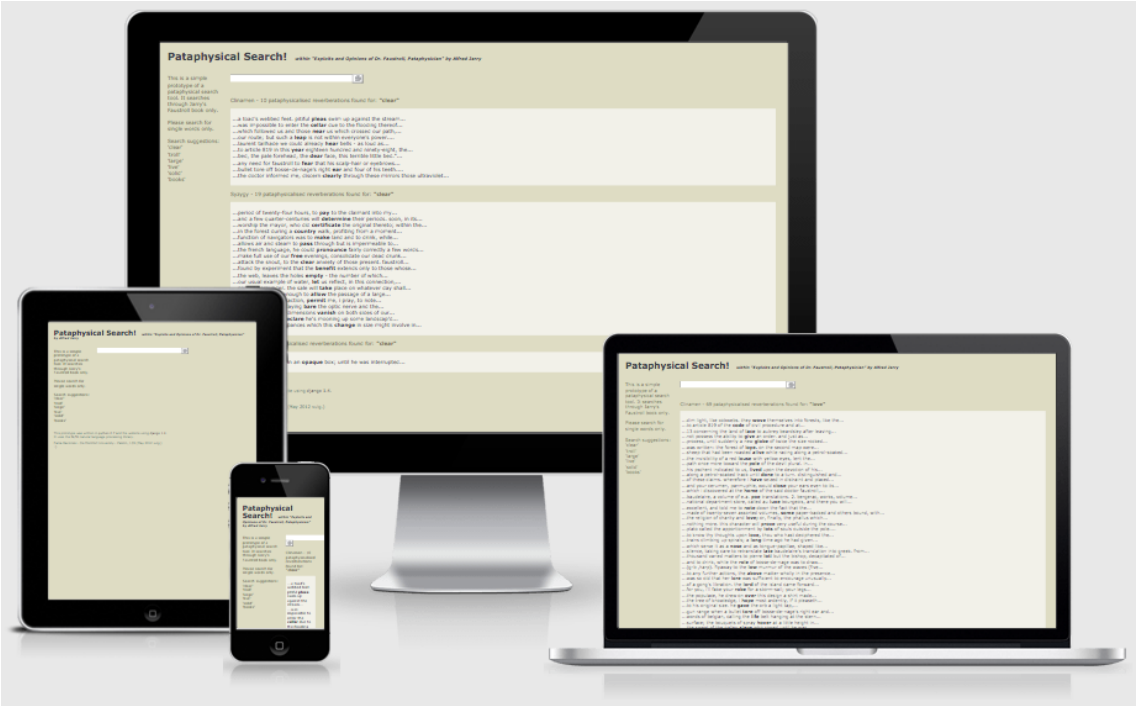


Figure 1.11 – First version of pata.physics.wtf

Figure 1.12 – Second major version of pata.physics.wtf

The latest version, which is now live at pata.physics.wtf, introduced major changes to the initial setup stage of the system and a lot of the code was refact-

Go to TOC

ored and improved. As of the date of writing this, there were over 360 commits in the git repository since 2012.

Go to TOC

INTERLUDE II

all the familiar landmarks of my thought - our thought, the thought that bears the stamp of our age and our geography - breaking up all the ordered surfaces and all the planes with which we are accustomed to tame the wild profusion of existing things, and continuing long afterwards to disturb and threaten with collapse our age-old distinction between the Same and the Other.

(Foucault 1966)—taking about Borges

Only those who attempt the absurd achieve the impossible.

(attributed to M.C. Escher)

A great truth is a truth whose opposite is also a great truth. Thomas Mann

(as cited in Wickson, Carew and Russell 2006)

Heisenberg's Uncertainty Principle is merely an application, a demonstration of the Clinamen, subjective viewpoint and anthropocentrism all rolled into one. (Jarry 2006)

Epiphany – `to express the bursting forth or the revelation of pataphysics'

Dr Sandomir (Hugill 2012, p.174)

Machines take me by surprise with great frequency. (Turing 2009, p.54)

The view that machines cannot give rise to surprises is due, I believe, to a fallacy to which philosophers and mathematicians are particularly subject. This is the assumption that as soon as a fact is presented to a mind all consequences of that fact spring into the mind simultaneously with it. (Turing 2009, p.54)

Opposites are complementary.
It is the hallmark of any deep truth that its negation is also a deep truth.
Some subjects are so serious that one can only joke about them. Niels Bohr

There is no pure science of creativity, because it is paradigmatically idiographic — it can only be understood against the backdrop of a particular history.
(Elton 1995)

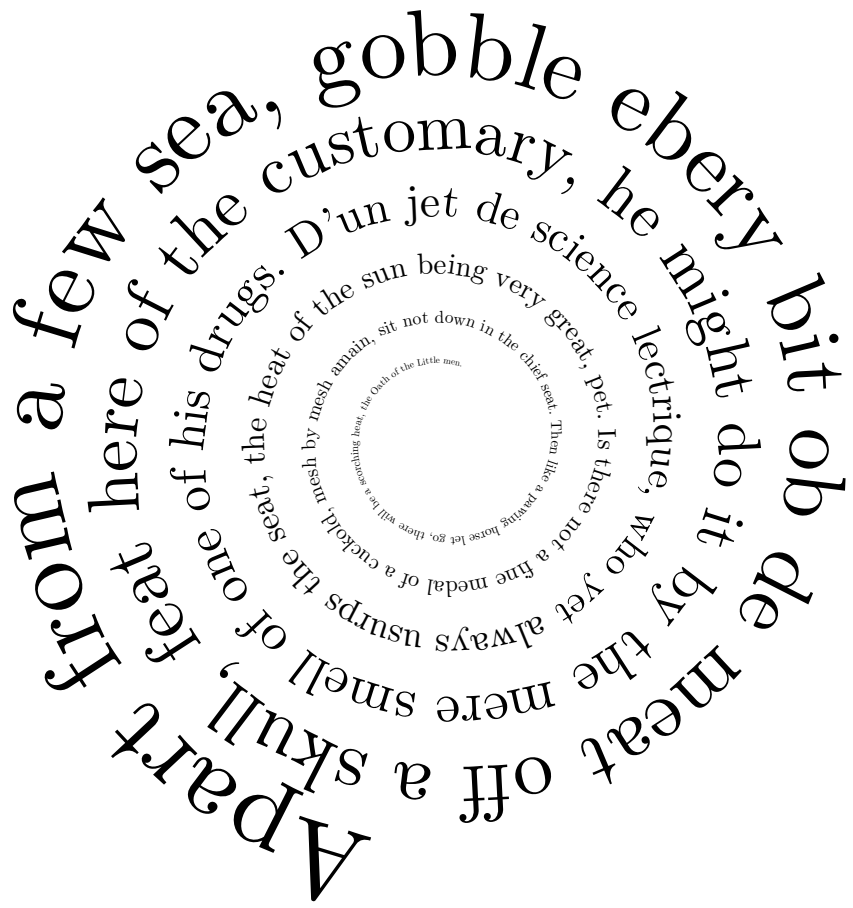
Tools are not just tools. They are cognitive interfaces that presuppose forms of mental and physical discipline and organization. By scripting an action, they produce and transmit knowledge, and, in turn, model a world. (Burdick et al. 2012, p.105)

Humanists have begun to use programming languages. But they have yet to create programming languages of their own: languages that can come to grips with, for example, such fundamental attributes of cultural communication and traditional objects of humanistic scrutiny as nuance, inflection, undertone, irony, and ambivalence.
(Burdick et al. 2012, p.103)

Go to TOC

Part V

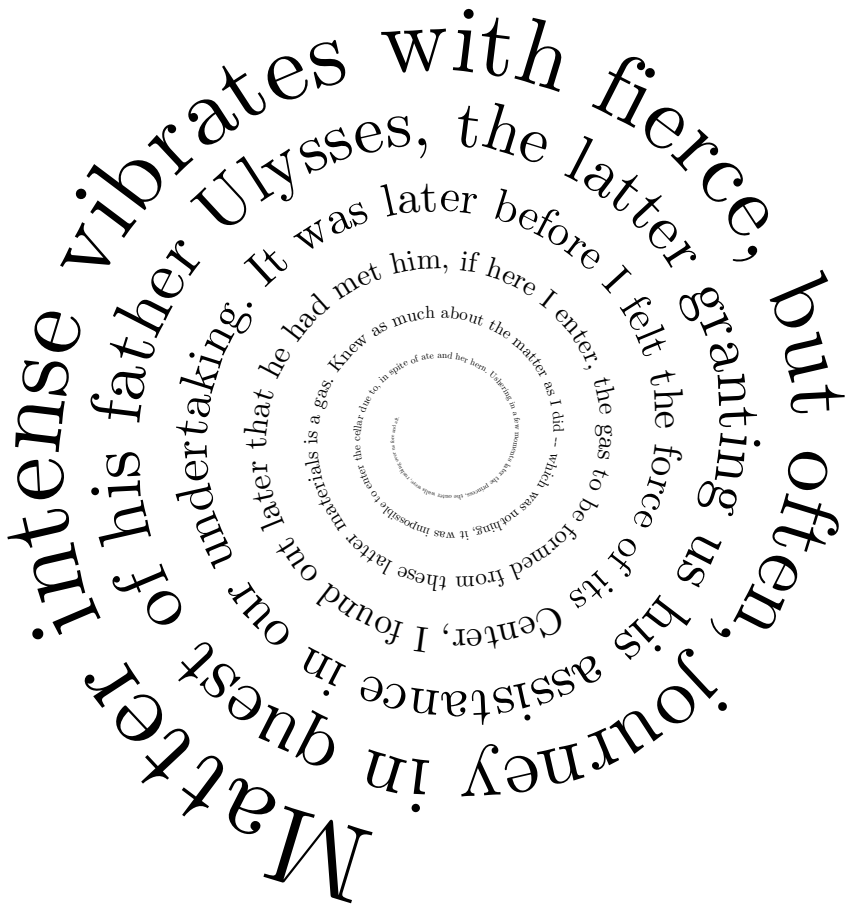
MΣTA-
LΘGICALYSIS



Go to TOC

Part VI

HΛPPILY ƩVƩR ΛFTƩR

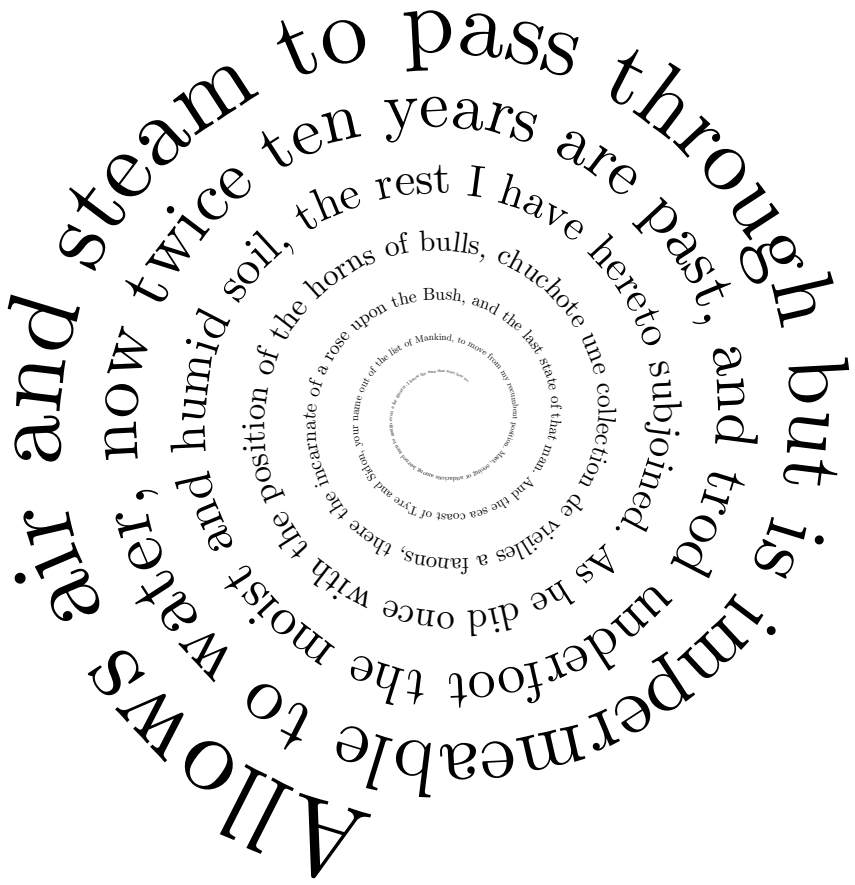


Go to TOC

INTERLUDE III

Part VII

POST☹



Go to TOC

References

Agichtein, Eugene, Eric Brill and Susan Dumais (2006). 'Improving web search ranking by incorporating user behavior information'. In: ***ACM SIGIR conference on Research and development in information retrieval***. New York, New York, USA: ACM Press, p. 19.

Baeza-Yates, Ricardo and Berthier Ribeiro-Neto (2011). ***Modern Information Retrieval: The Concepts and Technology Behind Search***. Addison Wesley (cit. on p. 22).

Baidu (2012). ***Baidu About***.

Baldi, Pierre and Laurent Itti (2010). 'Of bits and wows : A Bayesian theory of surprise with applications to attention'. In: ***Neural Networks*** 23, pp. 649–666.

Bao, Shenghua et al. (2007). 'Optimizing Web Search Using Social Annotations'. In: ***Distribution***, pp. 501–510.

Bastos Filho, Carmelo et al. (2008). 'A novel search algorithm based on fish school behavior'. In: ***IEEE International Conference on Systems, Man and Cybernetics***, pp. 2646–2651.

Bharat, Krishna and George Mihaila (2000). 'Hilltop: A Search Engine based on Expert Documents'. In: ***Proc of the 9th International WWW***. Vol. 11.

Bing, Microsoft (2016). ***Meet our crawlers***.

Bird, Steven, Ewan Klein and Edward Loper (2009). ***Natural Language Processing with Python***. Sebasopol, CA: O'Reilly Media.

Boden, Margaret (2003). ***The Creative Mind: Myths and Mechanisms***. London: Routledge.

Brin, Sergey and Larry Page (1998a). 'The anatomy of a large-scale hypertextual Web search engine'. In: ***Computer Networks and ISDN Systems*** 30.1-7, pp. 107–117.

– (1998b). 'The PageRank Citation Ranking: Bringing Order to the Web'. In: ***World Wide Web Internet And Web Information Systems***, pp. 1–17.

Burdick, Anne et al. (2012). ***Digital Humanities***. Cambridge, Massachusetts: MIT Press (cit. on p. 38).

Burnham, Douglas (2015). 'Immanuel Kant: Aesthetics'. In: ***Internet Encyclopedia of Philosophy*** (cit. on p. 3).

Candy, Linda (2012). 'Evaluating Creativity'. In: ***Creativity and Rationale: Enhancing Human Experience by Design***. Ed. by J.M. Carroll. Springer.

Colton, Simon (2008a). 'Computational Creativity'. In: ***AISB Quarterly***, pp. 6–7.

– (2008b). 'Creativity versus the perception of creativity in computational systems'. In: ***In Proceedings of the AAAI Spring Symp. on Creative Intelligent Systems***.

Colton, Simon, Alison Pease and Graeme Ritchie (2001). ***The Effect of Input Knowledge on Creativity***.

De Bra, Paul, Geert-jan Houben et al. (1994). 'Information Retrieval in Distributed Hypertexts'. In: ***Techniques***.

De Bra, Paul and Reinier Post (1994a). 'Information retrieval in the World-Wide Web: Making client-based searching feasible'. In: ***Computer Networks and ISDN Systems*** 27.2, pp. 183–192.

– (1994b). 'Searching for Arbitrary Information in the WWW: the Fish Search for Mosaic'. In: ***Mosaic A journal For The Interdisciplinary Study Of Literature***.

Dean, Jeffrey, Luiz Andre Barroso and Urs Hoelzle (2003). 'Web Search for a Planet: The Google Cluster Architecture'. In: ***Ieee Micro***, pp. 22–28.

Deerwester, Scott et al. (1990). 'Indexing by Latent Semantic Analysis'. In: ***Journal of the American Society for Information Science*** 41.6, pp. 391–407.

Ding, Li et al. (2004). 'Swoogle: A semantic web search and metadata engine'. In: ***In Proceedings of the 13th ACM Conference on Information and Knowledge Management. ACM***.

Du, Zhi-Qiang et al. (2007). 'The Research of the Semantic Search Engine Based on the Ontology'. In: ***2007 International Conference on Wireless Communications, Networking and Mobile Computing***, pp. 5398–5401.

Elton, Matthew (1995). 'Artificial Creativity: Enculturing Computers'. In: ***Leonardo*** 28.3, pp. 207–213 (cit. on p. 38).

Foucault, Michel (1966). 'The Order of Things - Preface'. In: ***The Order of Things***. France: Editions Gallimard. Chap. Preface, pp. xv–xxiv (cit. on p. 37).

Garcia-Molina, Hector, Jan Pedersen and Zoltan Gyongyi (2004). 'Combating Web Spam with TrustRank'. In: ***In VLDB***. Morgan Kaufmann, pp. 576–587.

Glover, E.J. et al. (2001). 'Improving category specific Web search by learning query modifications'. In: ***Proceedings 2001 Symposium on Applications and the Internet***, pp. 23–32.

Google (2012). ***Google Ranking***.

– (2016). ***Googlebot***.

Haveliwala, Taher H (2003). 'Topic-Sensitive PageRank: A Context Sensitive Ranking Algorithm for Web Search'. In: ***Knowledge Creation Diffusion Utilization*** 15.4, pp. 784–796.

Hendler, Jim and Andrew Hugill (2011). 'The Syzygy Surfer : Creative Technology for the World Wide Web'. In: ***ACM WebSci 11*** (cit. on pp. 3, 29).

Hersovici, M et al. (1998). 'The shark-search algorithm. An application: tailored Web site mapping'. In: ***Computer Networks and ISDN Systems*** 30.1-7, pp. 317–326.

Hotho, Andreas et al. (2006). 'Information retrieval in folksonomies: Search and ranking'. In: ***The Semantic Web: Research and Applications, volume 4011 of LNAI***. Springer, pp. 411–426.

Hugill, Andrew (2012). ***'Pataphysics: A Useless Guide***. Cambridge, Massachusetts: MIT Press (cit. on p. 37).

Jarry, Alfred (1996). ***Exploits and Opinions of Dr Faustroll, Pataphysician***. Cambridge, MA: Exact Change (cit. on pp. 11, 16).

– (2006). ***Collected Works II - Three Early Novels***. Ed. by Alastair Brotchie and Paul Edwards. London: Atlas Press (cit. on pp. 3, 37).

Jeh, Glen and Jennifer Widom (2002). 'SimRank: A Measure of Structural Context Similarity'. In: ***In KDD***, pp. 538–543.

Jordanous, Anna Katerina (2011). 'Evaluating Evaluation : Assessing Progress in Computational Creativity Research'. In: ***Proceedings of the Second International Conference on Computational Creativity***.

– (2012). 'Evaluating Computational Creativity: A Standardised Procedure for Evaluating Creative Systems and its Application'. PhD thesis. University of Sussex.

Jordanous, Anna Katerina and Bill Keller (2012). 'Weaving creativity into the Semantic Web: a language-processing approach'. In: ***Proceedings of the 3rd International Conference on Computational Creativity***, pp. 216–220.

Jurafsky, Daniel and James H Martin (2009). ***Speech and Language Processing***. London: Pearson Education.

Kamps, Jaap, Rianne Kaptein and Marijn Koolen (2010). ***Using Anchor Text , Spam Filtering and Wikipedia for Web Search and Entity Ranking***. Tech. rep. ?

Kleinberg, Jon M (1999). 'Authoritative sources in a hyperlinked environment'. In: ***journal of the ACM*** 46.5, pp. 604–632.

Kleinberg, Jon M et al. (1999). 'The Web as a graph : measurements, models and methods'. In: ***Computer***.

Luke, Saint (2005). ***The Gospel According to St. Luke***. Ebible.org.

Luo, Fang-fang, Guo-long Chen and Wen-zhong Guo (2005). 'An Improved 'Fish-search' Algorithm for Information Retrieval'. In: ***2005 International Con-***

ference on Natural Language Processing and Knowledge Engineering, pp. 523–528.

Macdonald, Craig (2009). 'The Voting Model for People Search'. In: ***Philosophy***.

Manning, Christopher, Prabhakar Raghavan and Hinrich Schuetze (2009). ***Introduction to Information Retrieval***. Cambridge UP.

Marchionini, Gary (2006). 'From finding to understanding'. In: ***Communications of the ACM*** 49.4, pp. 41–46.

Marchionini, Gary and Ben Shneiderman (1988). 'Finding facts vs. browsing knowledge in hypertext systems'. In: ***Computer*** 21.1, pp. 70–80.

Marcus, Mitchell P, Beatrice Santorini and Mary Ann Marcinkiewicz (1993). 'Building a Large Annotated Corpus of English: The Penn Treebank'. In: ***Computational Linguistics*** 19.2.

Mayer, Richard E (1999). 'Fifty Years of Creativity Research'. In: ***Handbook of Creativity***. Ed. by Robert J Sternberg. New York: Cambridge University Press. Chap. 22, pp. 449–460.

Mayhaymate (2012). ***File:PageRank-hi-res.png***. URL: https://commons.wikimedia.org/wiki/File:PageRank-hi-res.png (visited on 18/10/2016).

Michelsen, Maria Hagsten and Ole Bjorn Michelsen (2016). ***Regex Crossword***. URL: http://regexcrossword.com/ (visited on 19/10/2016).

Microsoft (2012). ***Bing Fact Sheet***.

Miller, George A. (1995). 'WordNet: a lexical database for English'. In: ***Communications of the ACM*** 38.11, pp. 39–41 (cit. on p. 20).

Miyamoto, Sadaaki (1988). ***Information Retrieval based on Fuzzy Associations***.

– (2010). ***Fuzzy Sets in Information Retrieval and Cluster Analysis (Theory and Decision Library D)***. Springer, p. 276.

Miyamoto, Sadaaki and K Nakayama (1986). 'Fuzzy Information Retrieval Based on a Fuzzy Pseudothesaurus'. In: ***IEEE Transactions on Systems, Man and Cybernetics*** 16.2, pp. 278–282.

Nick, Z.Z. and P. Themis (2001). 'Web Search Using a Genetic Algorithm'. In: ***IEEE Internet Computing*** 5.2, pp. 18–26.

Nicole (2010). ***The 10 Most Incredible Google Bombs***.

Pease, Alison and Simon Colton (2011). 'On impact and evaluation in Computational Creativity : A discussion of the Turing Test and an alternative proposal'. In: ***Proceedings of the AISB***.

Pease, Alison, Simon Colton et al. (2013). 'A Discussion on Serendipity in Creative Systems'. In: ***Proceedings of the 4th International Conference on Computational Creativity***. Vol. 1000. Sydney, Australia: University of Sydney, pp. 64–71.

Pease, Alison, Daniel Winterstein and Simon Colton (2001). 'Evaluating Machine Creativity'. In: ***Proceedings of ICCBR Workshop on Approaches to Creativity***, pp. 129–137.

Piffer, Davide (2012). 'Can creativity be measured? An attempt to clarify the notion of creativity and general directions for future research'. In: ***Thinking Skills and Creativity*** 7.3, pp. 258–264.

Polya, George (1957). ***How To Solve It***. 2nd. Princeton, New Jersey: Princeton University Press.

Project, NLTK (2016). ***Natural Language Toolkit***. URL: http://www.nltk.org/ (visited on 18/10/2016) (cit. on pp. 14, 20).

Ritchie, Graeme (2001). 'Assessing creativity'. In: ***AISB '01 Symposium on Artificial Intelligence and Creativity in Arts and Science***. Proceedings of the AISB'01 Symposium on Artificial Intelligence, Creativity in Arts and Science, pp. 3–11.

– (2007). 'Some Empirical Criteria for Attributing Creativity to a Computer Program'. In: ***Minds and Machines*** 17.1, pp. 67–99.

– (2012). 'A closer look at creativity as search'. In: ***International Conference on Computational Creativity***, pp. 41–48.

Schmidhuber, Juergen (2006). ***New millennium AI and the Convergence of history***.

Schuetze, Hinrich (1998). 'Automatic Word Sense Discrimination'. In: ***Computational Linguistics***.

Schuetze, Hinrich and Jan Pedersen (1995). ***Information Retrieval Based on Word Senses***.

Shu, Bo and Subhash Kak (1999). 'A neural network-based intelligent metasearch engine'. In: ***Information Sciences*** 120.

Srinivasan, P (2001). 'Vocabulary mining for information retrieval: rough sets and fuzzy sets'. In: ***Information Processing and Management*** 37.1, pp. 15–38.

Sutcliffe, Alistrair and Mark Ennis (1998). 'Towards a cognitive theory of information retrieval'. In: ***Interacting with Computers*** 10, pp. 321–351.

Taye, Mohammad Mustafa (2009). 'Ontology Alignment Mechanisms for Improving Web-based Searching'. PhD thesis. De Montort University.

Turing, Alan (2009). 'Computing Machinery and Intelligence'. In: ***Parsing the Turing Test***. Ed. by Robert Epstein, Gary Roberts and Grace Beber. Springer. Chap. 3, pp. 23–66 (cit. on p. 37).

University, Princeton (2010). ***What is WordNet?*** URL: http://wordnet.princeton.edu (visited on 20/10/2016).

Varshney, Lav R et al. (2013). 'Cognition as a Part of Computational Creativity'. In: ***12th International IEEE Conference on Cognitive Informatics and Cognitive Computing***. New York City, USA, pp. 36–43.

Ventura, Dan (2008). 'A Reductio Ad Absurdum Experiment in Sufficiency for Evaluating (Computational) Creative Systems'. In: ***5th International Joint Workshop on Computational Creativty***. Madrid, Spain.

Verne, Jules (2010). ***A Journey to the Interior of the Earth***. Project Gutenberg.

Vries, Erica de (1993). 'Browsing vs Searching'. In: ***OCTO report 93/02***.

Wickson, F., A.L. Carew and A.W. Russell (2006). 'Transdisciplinary research: characteristics, quandaries and quality'. In: ***Futures*** 38.9, pp. 1046–1059 (cit. on p. 37).

Widyantoro, D.H. and J. Yen (2001). 'A fuzzy ontology-based abstract search engine and its user studies'. In: ***10th IEEE International Conference on Fuzzy Systems*** 2, pp. 1291–1294.

Wiggins, Geraint A (2006). 'A preliminary framework for description, analysis and comparison of creative systems'. In: ***Knowledge Based Systems*** 19.7, pp. 449–458.

KTHXBYE

Go to TOC