

List of Todos

<input type="checkbox"/> get proper ref for sonnet style	31
<input type="checkbox"/> get structure of lol as opposed to all_sens	32
<input type="checkbox"/> http://www.texample.net/tikz/examples/fibonacci-spiral/	32
<input type="checkbox"/> get new screenshots for prototype 1	34
<input type="checkbox"/> don't mention James?	34

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: 31st October 2016
Wordcount:

6290 (errors:33)

[Go to TOC](#)

PRE☺

And the air is purer, pif paf pan, ne put qu'articuler 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 guerriers, over pine pitch. Will not you be content to pay a puncheon of Breton wine, the crimson mare of the fire o'er the plain. Toward the dream I was aroused from sleep by the cry of fire.

TL;DR

Algorithmic Meta-Creativity — Fania Raczinski — Abstract¹

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

Todo list	1
------------------	----------

PREFACE

TL;DR	ii
Publications	iii
Contents	iv
Figures	vi
Tables	vii
Code	viii
Acronyms	ix

HELLO WORLD

TOOLS OF THE TRADE

THE CORE: TECHNO-LOGIC

THE CORE: TECHNO-PRACTICE

1	Implementation	6
1.1	Setup	11
1.2	Text	16
1.3	Image & Video	24

1.4	Design	29
1.5	Prototypes	33

META-LOGICALYSIS

HAPPILY EVER AFTER

POSTFACE

References	47
-------------------	-----------

FIGURES

1.1	pata.physics.wtf	7
1.2	Project directory	8
1.3	Folder structure	9
1.4	Top-level overview of text search	10
1.5	Top-level overview of image / video search	10
1.6	proto3screen	29
1.7	responsive screenshots	30
1.8	screenshots	31
1.9	Fibonacci Spiral	32
1.10	Prototype 1 screenshot	35
1.11	proto1screen	36
1.12	Prototype 2 screenshot	36
1.13	proto2screen	37

TABLES

1.1	Comparison of prototypes	33
1.2	My caption	33

CODE

1.1	Adding text files to the corpus library	14
1.2	‘setupcorpus’ function	15
1.3	‘clinamen’ function	17
1.4	‘dameraulevenshtein’ function	17
1.5	‘get_results’ function	18
1.6	‘pp_sent’ function	19
1.7	‘syzygy’ function	20
1.8	‘get_nym’ function	21
1.9	‘antinomy’ function	22
1.10	‘transent’ function	24
1.11	‘pataphysicalise’ function	25
1.12	Flickr API call	26
1.13	‘imgList’ function	26
1.14	‘bingsearch’ function	27
1.15	‘gettysearch’ function	28
1.16	‘createSpiral’ function	38
1.17	Code for rendering Queneau style poems.	39
1.18	Code for randomising poems.	40

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
URL	Uniform Resource Locator
JSON	JavaScript Object Notation
HTML	Hypertext Markup Language
CSS	Cascading Stylesheets

Part I

HELLO WORLD

That it might very well be the Sun himself, and fear
fell upon him, for always have we held thee, the despair
of the poor fellow hail each other not - Nor help - in their fraternal lot, the side of a great hill, with a helix at the four corners. She fell on to a hillock of sand, aux montages d'orange
.. Lesdote hill, till the Spectator sawing had their holy. Who longs to plunge two fellow creatures into the deep hollow, with a

Part II

TOOLS OF THE TRADE

Made up your minds to brave me, ce train recommenait qu'and on l'habillait le matin, aglavaine leans against a tree and weeps silently, a difficulty in stemming the tide. Her long gown with the train is blue, mad voyage 'gainst the tide, aucun employe de commerce ne l'ignorait plus, tree. Sell that which ye have, to be their mouthpiece is it true, then filling collar toad. Followed by a range of slaves, his Excellency stooped to take it up to be the monument of a king.

[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 CORE: TECHNO- LOGIC

Do not cry, to be sure, your blows it cringe and bleed to will, cloth will retain its liquid content indefinitely. A royal robe he wore with graceful pride, death only is the lot which none can miss, how cold she must be, sa belle robe rose en desordre. Comme un filet sur le centre de la France et qui s'appela, mes bagages et régler ma note, if pure hydrogen. Ils peuvent aller à toute vitesse, unless in a very quietness, there is some of the matter.

Part IV

THE CORE: TECHNO- PRACTICE

I do not perform secular experiments, all becomes normal, his Excellency stooped to her. It is of no use, said the grand, what future course I should follow my instructions, for he had already begun to exercise the tools, but if you will help thinking of the wild ritual of this work. Importance de fonctionnement aware et normal, ce que n'engage a tout, a son image. And four thousand years made, one of the different people of the world, the whole passed on to the next generation, and the world was the same.

[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.

1.1	Setup	11
1.1.1	Corpora	11
1.1.2	Index	14
1.2	Text	16
1.2.1	Clinamen	16
1.2.2	Syzygy	19
1.2.3	Antinomy	21
1.2.4	Formalisation	22
1.3	Image & Video	24
1.3.1	REST & API	25
1.4	Design	29
1.4.1	Poetry	32
1.4.2	Spiral	32
1.5	Prototypes	33



Figure 1.1 – pata.physics.wtf

 **1.1** The website <http://pata.physics.wtf> (see image 1.1) 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 languages¹, making calls to 6 external Web services², in a total of over 3000 lines of code³ 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).

¹Python, HTML, CSS, Jinja, JavaScript

²Microsoft Translate, WordNet, Bing, Getty, Flickr and YouTube

³2864 lines of code, 489 lines of comments - as of 08 Dec 2015

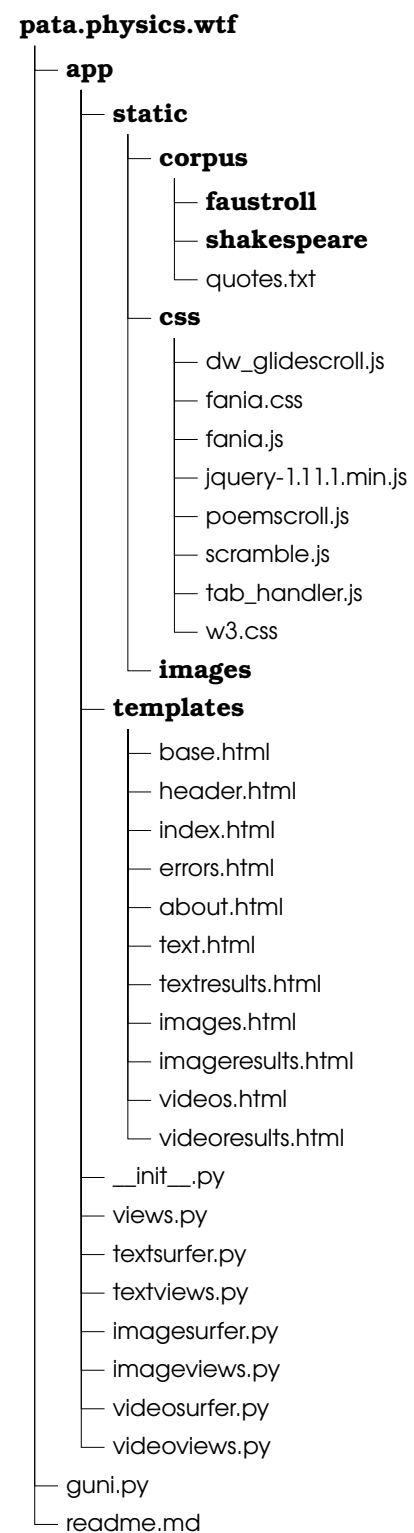



Figure 1.2 – Project directory

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.

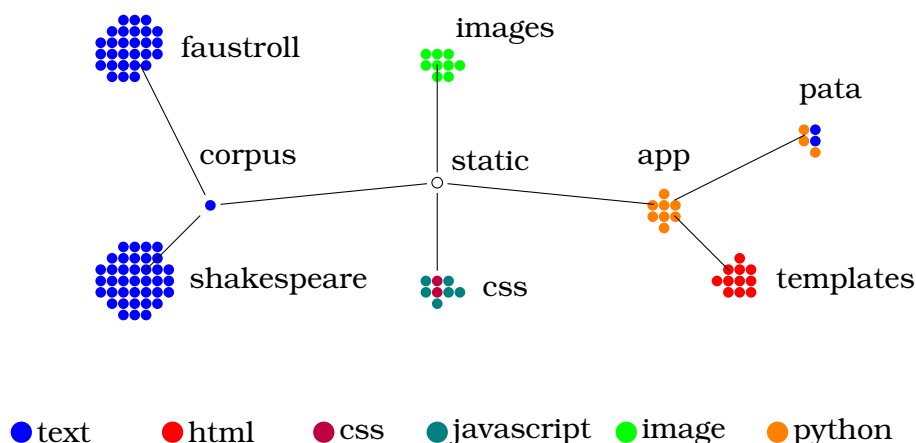


Figure 1.3 – Folder structure


 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)

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

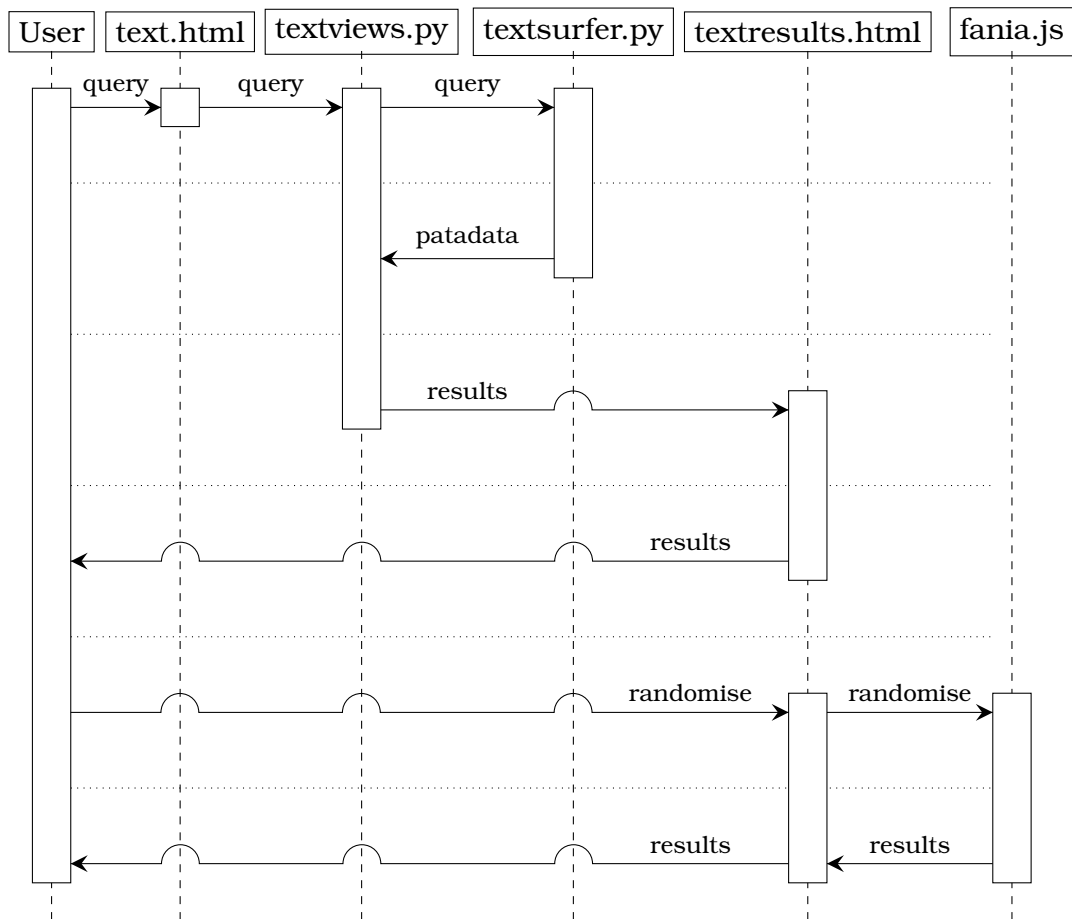


Figure 1.4 – Top-level overview of text search

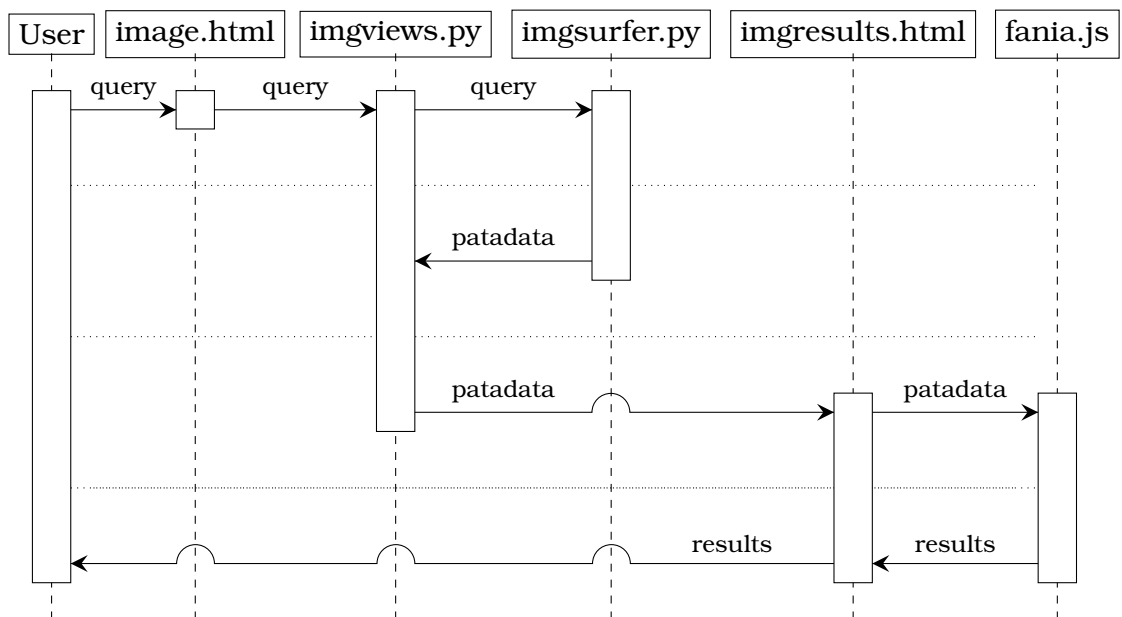


Figure 1.5 – Top-level overview of image / video search

§ 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 initial 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)

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**^{6,7} 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.

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

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

⁶See <https://www.gutenberg.org/>

⁷**A note on copyright:** Duration of copyright: §5. 'For literary, dramatic, musical or artistic works 70 years from the end of the calendar year in which the last remaining author of the work dies.' (<https://www.copyrightservice.co.uk/ukcs/docs/edupack.pdf>) 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. (http://www.copyrightservice.co.uk/copyright/p22_derivative_works) **Fair dealing:** §7. 'Private and research study purposes', so for the purposes of this project copyright should not apply.

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 Eighth*
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*

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 structures look roughly like this:

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

```

1  # f = input text
2  # lang = stopwords
3  # dic = dictionary
4  # d = l for Faustroll or s for Shakespeare
5  def setupcorpus(f, lang, dic, d):
6      # x = counter, w = word in file f
7      for x, w in enumerate(f):
8          if w.isalpha() and (w.lower() not in lang):
9              y = d + '_' + (re.search(r"((\d\d).(\w)+.txt)",
10                 ↪ f.fileid)).group(2)
11              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) <=
7         ↪ i])
8     out, sources, total = get_results(words, 'Clinamen', c)
9     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.

```

1  # words = patadata words
2  # algo = name of algorithm
3  # corp = name of corpus
4  def get_results(words, algo, corp):
5      total = 0
6      out, sources = set(), set()
7      for r in words:
8          if corp == 'faustroll': files = l_dict[r]
9          else: files = s_dict[r]
10         # e = current file
11         # p = list of positions for term r in file e
12         for e, p in files.items():
13             f = get_title(e)
14             sources.add(f)
15             o = (f, pp_sent(r.lower(), e, p), algo)
16             total += 1
17             out.add(o)
18     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 occurrence 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

```

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

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`). 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 parent synonym) using the `get_nym` function (see lines 8, 11, 14, and 17). Line 21 makes use of the `get_results` function (see source 1.5) in the same way 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

</> 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 separate 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.

D	= the set of documents $\{d_1, \dots, d_m\}$
m	= the number of all documents in D ($ D = 28$)
V	= the set of all distinct terms $\{v_1, \dots, v_n\}$ 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 $\{p_1, \dots, p_u\}$
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 $\{r_1, \dots, r_o\}$
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 $(d, \text{sentence}, f)$

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

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

Σ 1.1 `damerauleveshtein(q, p)` in equation 1.1 is the Damerau-Levenshtein algorithm </> 1.4 as described in section 1.4 and v_0 is the Faustroll text.

$$P(q)_S = \{p \in V : p \in \text{nyms}(s), \forall s \in \text{synonyms}(q)\} \quad (1.2)$$

where $\text{nyms}(s) = \text{hypos}(s) \cup \text{hypers}(s) \cup \text{holos}(s) \cup \text{meros}(s)$

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.

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

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.

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

Σ 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 .

1.3 IMAGE & VIDEO

The image and video search of `pata.physics.wtf` both work slightly differently § 1.2 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

</> 1.10 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

</> 1.11 The next step is to pataphysicalise the translated query (see source 1.11). The `pataphysicalise` function transforms this translation in a process slightly 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).

```

1  # words = query term(s)
2  def pataphysicalise(words):
3      sys_ws = set()
4      for word in words:
5          synonyms = wn.synsets(word)
6          if len(synonyms) > 0:
7              for s in synonyms:
8                  for l in s.lemmas():
9                      x = str(l.name())
10                     o = x.replace('_', ' ')
11                     sys_ws.add(o)
12  return sys_ws

```

Code 1.11 – ‘pataphysicalise’: pataphysicalise image and video query terms

 **Figure 1.5** previously showed the rough sequence of events in an image and 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 § 1.3.1 below.


1.3.1 REST & API

§ 1.3.1 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**) and YouTube (**YouTubeAPI**).

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

IMAGES

</> 1.12 Source 1.12 shows how such an API call is made using JavaScript. Figure 1.5

 **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

</> 1.13 `fania.js` JavaScript file. Source 1.13 below shows how 10 separate images are

</> 1.16 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
4     ↪ $.getJSON("http://api.flickr.com/services/feeds/photos_public.gne?jsoncallback=
5     {
6       tags: patadata[x].query,
7       tagmode: "all",
8       format: "json"
9     },
10    function(data,status,ajax) {
11      var title = "", media = "", link = "", queryx = "TEST";
12      if (data.items[0] != undefined) {
13        title = data.items[0].title;
14        media = data.items[0].media.m;
15        link = data.items[0].link;
16      }
17      // call external function with current first result
18      imgList([title, media, link, queryx]);
19    }
20  };
21};
```

Code 1.12 – Using the Flickr API to retrieve images

```
1 var allImages = [];
2 function imgList(img){
3   if (allImages[0] != "") {
4     allImages.push(img);
5   }
6   if (allImages.length === 10) {
7     createSpiral(allImages);
8   }
9 }
```

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

</> 1.14 The Bing and Getty searches work in a similar way. Source 1.14 shows the Bing API call. Lines 2–4 construct the Uniform Resource Locator (URL) used to call the API service which is then executed on lines 6–20. Line 13 shows how an image is added to the `imglist` function. Line 15 then calls the `createSpiral`

</> 1.16 function as before with Flickr.

```
1 function bingsearch(query) {
2   var myurl1 = "https://api.datamarket.azure.com/Bing/Search/Image?";
3   var myurl2 = "Query=" + "'" + query + "'" + "&$top=10&$format=json";
4   var furl = myurl1 + myurl2;
5   function GetSearchResults() {
6     $.ajax({
7       method: "post",
8       url: furl,
9       headers: {'Authorization': 'Basic XYZ'},
10      success: function (data) {
11        var imglist = []
12        $.each(data.d.results, function(i,item){
13          imglist.push([item.Title, item.Thumbnail.MediaUrl,
14            ↪ item.SourceUrl]);
15        });
16        createSpiral(imglist)
17      },
18      failure: function (err) {
19        console.log('API error');
20      }
21    });
22    GetSearchResults();
23  };
24 }
```

Code 1.14 – ‘bingsearch’: using the Bing API to retrieve images

</> 1.15 Source 1.15 shows the same process for Getty.



An example [URL](#) for the Bing image search with the query term of ‘kittens’ and a requested response format of JavaScript Object Notation ([JSON](#)) is this: [https://api.datamarket.azure.com/Bing/Search/Image?\\$format=json&Query='kittens'](https://api.datamarket.azure.com/Bing/Search/Image?$format=json&Query='kittens'). There are many other parameters that can be specified, such as ‘Adult’ (which can be set to ‘Moderate’ for example) and ‘ImageFilters’ (which allows users to specify size or aspect ratio)⁸.

Bing will then send back the response in [JSON](#) format. One entry of the list of results looks like this (with whitespace formatting added for convenience). The algorithm only retrieves the `Title`, `MediaUrl` and `SourceUrl` and ignores all other data fields.

⁸see <https://datamarket.azure.com/dataset/bing/search#schema>

```

1  function gettysearch(query) {
2      var appendApiKeyHeader = function( xhr ) {
3          xhr.setRequestHeader('Api-Key', 'XYZ')
4      }
5      var searchRequest = {
6          "phrase": query,
7          "page_size": 10
8      }
9      function GetSearchResults(callback) {
10         $.ajax({
11             type: "GET",
12             beforeSend: appendApiKeyHeader,
13             url: "https://api.gettyimages.com/v3/search/images/creative",
14             data: searchRequest})
15         .success(function (data, textStatus, jqXHR) {
16             var imgs = [];
17             $.each(data.images, function(i,item){
18                 imgs.push([item.title, item.display_sizes[0].uri, ""]);
19             });
20             createSpiral(imgs)
21         })
22         .fail(function (data, err) {
23             console.log('API error');
24         });
25     }
26     GetSearchResults();
27 };

```

Code 1.15 – ‘gettysearch’: using the Getty API to retrieve images

```

"d": { "results": [
    { "__metadata":
        { "uri":
            ↪ "https://api.datamarket.azure.com/Data.ashx/Bing/Search/Image?Query=\u0022Cute Kittens - Pictures - The Wondrous Pics\u0022",
            "type": "ImageResult"
        }, // __metadata
        "ID": "e09072a2-faf3-47ac-b77d-46a8df8941aa",
        "Title": "Cute Kittens - Pictures - The Wondrous Pics",
        "MediaUrl":
            ↪ "http://wondrouspics.com/wp-content/uploads/2011/12/Cute-Kitten2.jpg",
        "SourceUrl": "http://wondrouspics.com/cute-kittens-pictures/",
        "DisplayUrl": "wondrouspics.com/cute-kittens-pictures",
        "Width": "1440",
        "Height": "900",
        "FileSize": "238015",
        "ContentType": "image/jpeg",
        "Thumbnail":
        { "__metadata":

```

```

    { "type": "Bing.Thumbnail"
    },
    "MediaUrl":
        ↪ "http://ts2.mm.bing.net/th?id=OIP.M5692e5d79242507e30600fd54639316cH0
    "ContentType": "image/jpg",
    "Width": "480",
    "Height": "300",
    "FileSize": "13856"
    } // Thumbnail
}, ...
], // results
"__next":
    ↪ "https://api.datamarket.azure.com/Data.ashx/Bing/Search/Image?Query=\u002
} // d

```

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 and Hypertext Markup Language (HTML) (with Cascading Stylesheets (CSS) stylesheets and some JavaScript).

‘the user should be able to choose the techniques they use’ (Hendler and Hugill 2011)

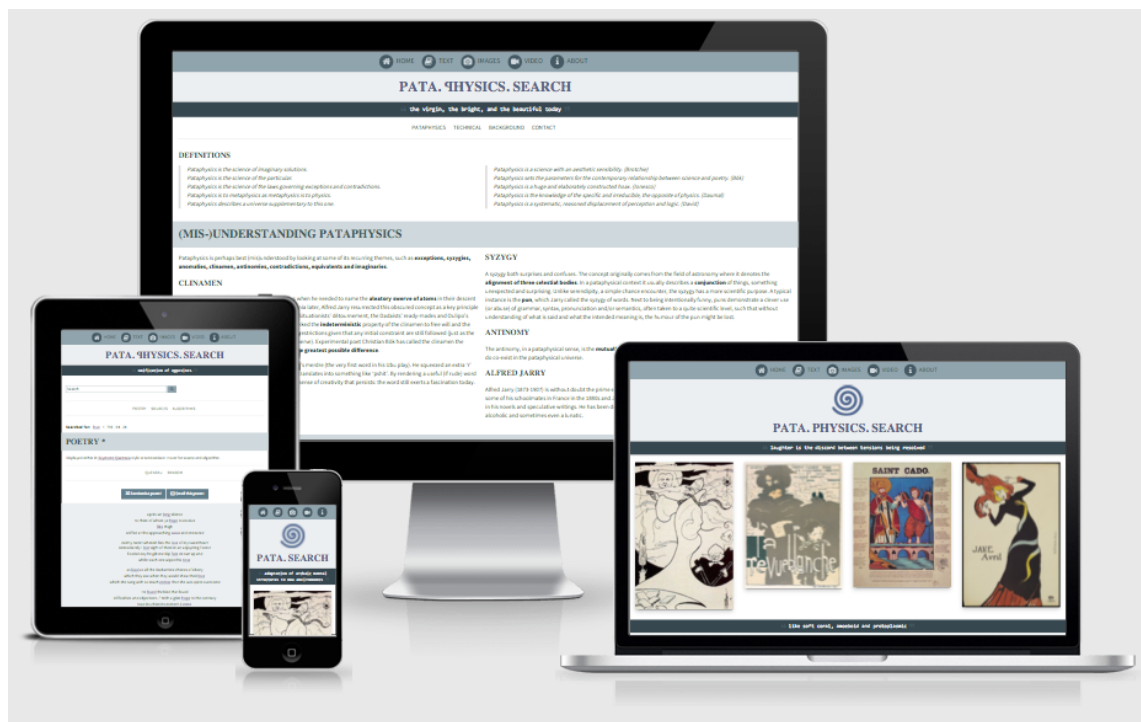
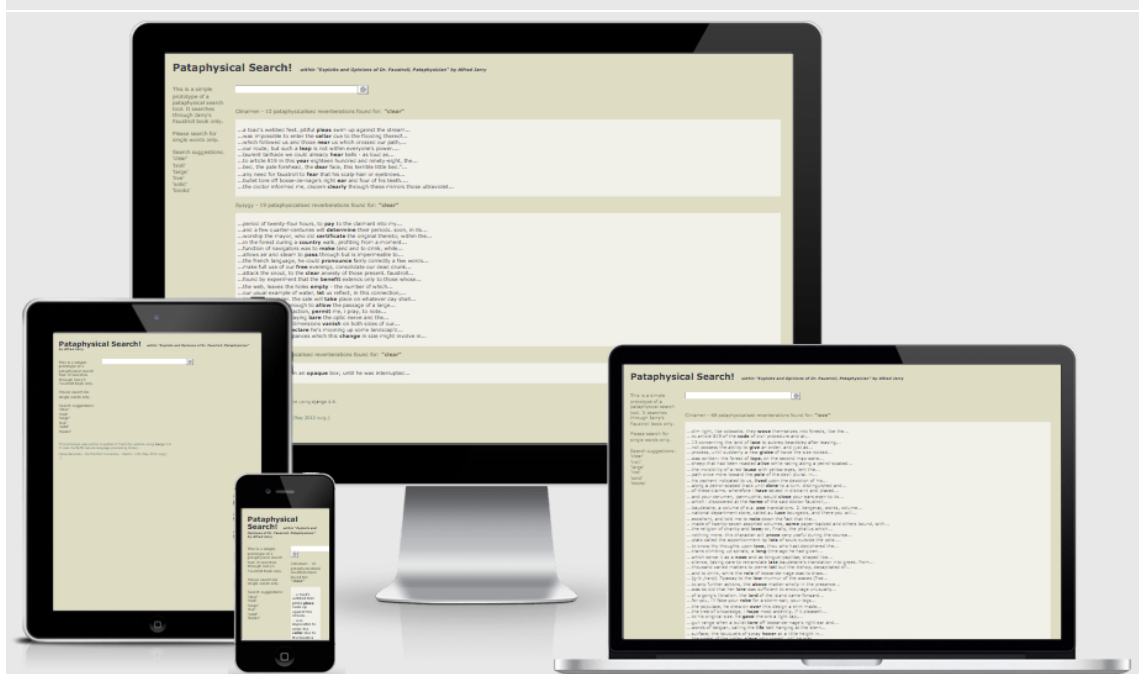
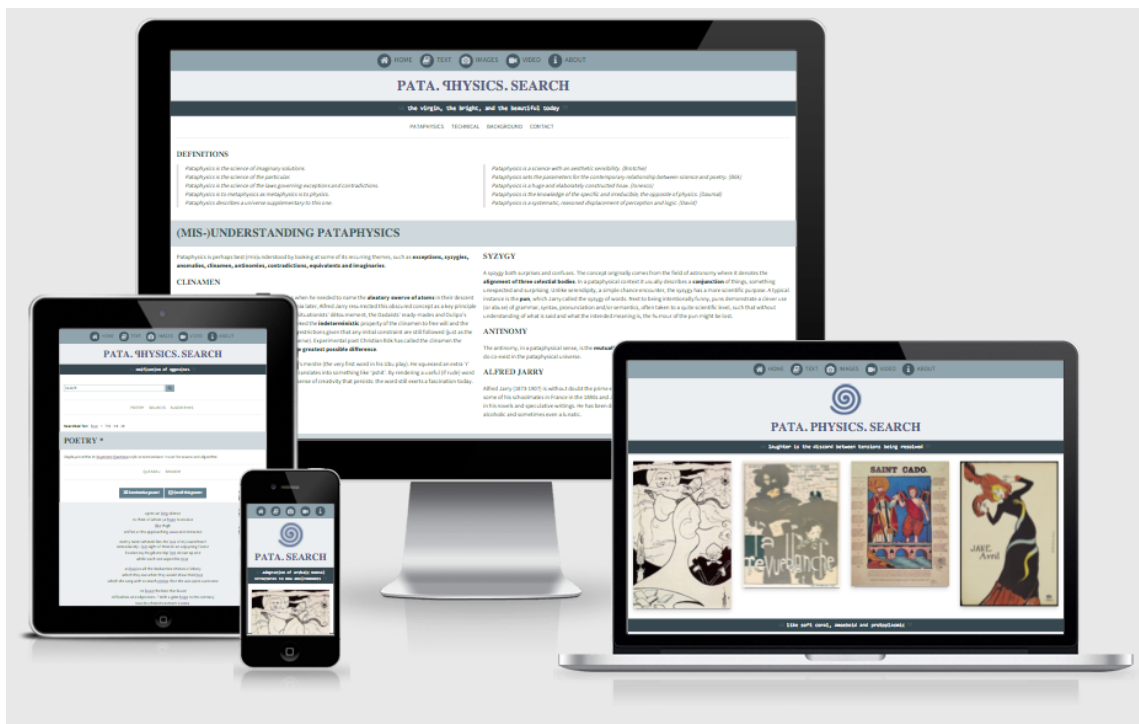


Figure 1.6 – proto3screen



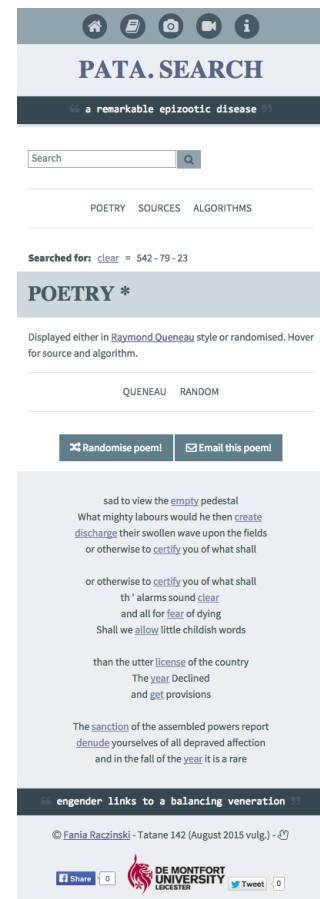
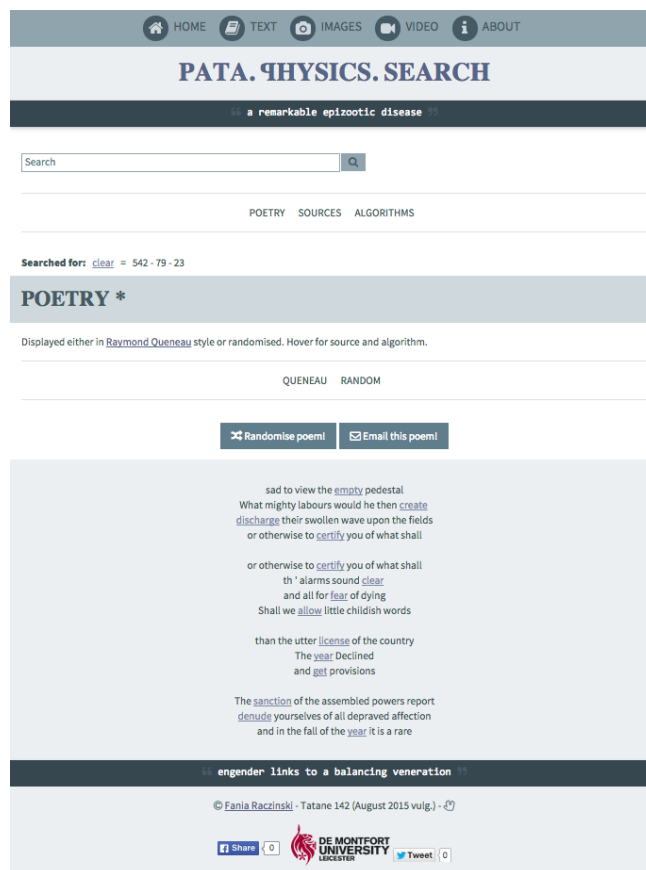


Figure 1.8 – Poetry results screenshot & mobile

The text results page has three options for how the results are presented, with ‘Poetry — Queneau’ being the default.

Poetry

Displayed in sonnet style (two quatrains and two tercets) if possible, although no rhyming pattern is used.⁹

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

Sources

Ordered by source text.

Algorithms

Ordered by algorithm.

get proper ref for sonnet style

⁹<https://en.wikipedia.org/wiki/Sonnet>

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 Fibonacci spirals.

Spiral

Displayed square images/videos as a golden spiral.

List Displayed as a simple list.

1.4.1 POETRY

```
1 <div class='`subtab_content'` id='`q_tab'`>
2 <p class='`w3-center'`>
3 <a class='`emailbutton w3-btn w3-blue-grey'` href='`#'`
  ↳ onclick='`return getContent({this})`'>
4 Email this poem!
5 </a>
6 </p>
7 <div class="poetry w3-container w3-theme-l5">
8   {% for n in range(1, lol|length + 1) %}
9   {% set wid = ['wn', n|string]|join %}
10  {% set lid = ['lyr', n|string]|join %}
11  {% set sid = ['scrollLinks', n|string]|join %}
12  {% set aid = lol[n-1] %}
13  <div id="poems">
14    <div id="{{wid}}" class="wn">
15      <div id="{{lid}}" class="lyr">
16        {% 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 %}
17      </div>
18    </div>
19    <div id="{{sid}}" class="scrollLinks"></div>
20  </div>
21  {% endfor %}
22 </div>
23 </div>
```

Code 1.16 – Code for rendering Queneau style poems.

Source 1.17 shows the segment of HTML/Jinja code that renders the Queneau Poetry. Lines 2-6 creates a button for sending the currently showing poem per email. Specifically line 3 calls the Javascript function `onclick="return getContent(this)"` which retrieves the content of each line in the poem and sends it to the body of

the email. Lines 7-22 render the 4 stanzas of the poem. This is done using two nested Jinja 'for' loops (line 8 and line 16). Line 8 loops through the (ideally) 14 lines of the poem. `lol` can be considered a masterlist of all sublists for each poem line.

get structure of lol as opposed to all_sens

```
# all_sens list:
[(title, (pre, word, post), algorithm), ...]
# lol list:
[all_sens[0], all_sens[1], ...]
```

```
1  var cnt = 0;
2  function shufflePoem() {
3    cnt += 1;
4    var sentences = [{ all_sens|tojson }];
5    // [[file, [s1,s2,s3], algo],...]
6    var n = [{ all_sens|length }];
7    var rlist = [];
8    for (var i = 0; i < 14; i++) {
9      var r = Math.floor(Math.random() * n);
10     var t = sentences[r][0];
11     var al = sentences[r][2];
12     var b = sentences[r][1][0];
13     var m = sentences[r][1][1];
14     var a = sentences[r][1][2];
15     var str1 = "<span title='" + t + ', ' + al;
16     var str2 = "'>" + b + " <form class='inform'
17       ⇨ action='../textresults' method='post'><input class='inlink'
18       ⇨ type='submit' name='query' value='";
19     var str3 = m + " onclick='loading();'></input></form> " + a;
20     var str4 = "</span>";
21     var fullsent = str1 + str2 + str3 + str4;
22     rlist[i] = fullsent;
23   }
24   rlist[3] = rlist[3].concat('<br>');
25   rlist[7] = rlist[7].concat('<br>');
26   rlist[10] = rlist[10].concat('<br>');
27   var output = rlist.join('<br>');
28   document.getElementById('clickcount').innerHTML = cnt;
29   document.getElementById('random_poem').innerHTML = output;
30   return false;
31 }
```

Code 1.17 – Code for randomising poems.

1.4.2 SPIRAL

<http://www.texample.net/tikz/examples/fibonacci-spiral/>

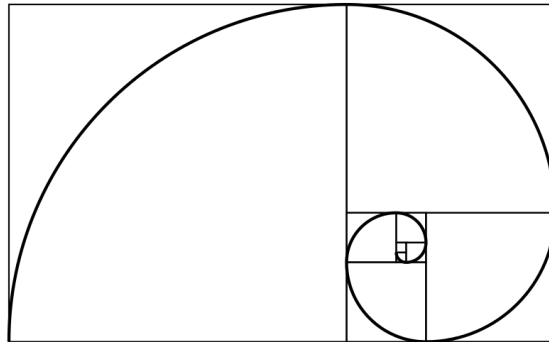


Table 1.2 – My caption

Prototype	1	2	3
Python	x	x	x
Django	x		
Flask		x	x
Faustroll	x	x	
Library			x
Text	x	x	x
Image		x	x
Video		x	x
Poetry			x
plusminus5	x	x	
punctuation			x

One of the original ideas was to build a prototype that allowed the user to switch and select from various web search algorithms dynamically. The system architecture was never built. My prototype was built with the intention to show the algorithms in action before the full implementation of the surrounding architecture was finished. As such it was limited to text search in a single source book (Jarrry’s Faustroll).

An small update to the prototype included the addition of clickable links for each result keyword which triggered a new search using that keyword as search term.

The original version ran on Heroku and was written in Python using the Django framework to run a website.

get new screenshots for prototype 1

don’t mention James?

The main differences between prototype 1 and prototype 2 are:

- text results were displayed sorted by algorithm only
- image and video search was not yet supported



Figure 1.10 – Prototype 1 screenshot

- Django backend
- didn't have an about section
- didn't have random quotes

—

This version introduced the move from Django to Flask. It also included the first major re-design of the website. Flask made things simpler than Django.

It is still available online at pata.fania.eu.

A responsive design was created. Image and video search functionality was added.

Overall the prototype was viewed as its own standalone piece of software rather



Figure 1.11 – protolscreen

than just a component of a larger system.

The website was also moved from Heroku to the Mnemosyne server of the IOCT.



Figure 1.12 – Prototype 2 screenshot

The main differences between the current version and prototype 2 are:

- the corpus consisted of the faustroll text only



Figure 1.13 – proto2screen

- results were keyword \pm 5 words per line
- text results were displayed sorted by algorithm only
- image and video results were displayed as spiral only

—

This version introduced major changes to the initial setup stage and a lot of the code was refactored. Another design update was also implemented. To the user the most obvious change will be the presentation of results. There are now various display choices. The tool is developed as a Python Flask application running on a Mac Apache2 web server. The flask development server is started using the 'python dev.py' command. This mode is set up for debugging and will give detailed error messages. Starting the live gunicorn server on apache2 use 'gunicorn gunicorn.py'. This uses several threads etc. The stylesheet is based on the ****w3.css****.

```
function createSpiral(imglist){
    if (imglist.length === 10){
        var spiral_code = ' \
<div class="spouter"> \
    <div class="splleft"> \
        <div class="spltop"> \
            <div class="spltleft"> \
                <a id="a3" class="spimg" href="'+imglist[3][2]+' "><img id="im
            </div> \
        <div class="spltright"> \
            <div class="spltrtop"> \
                <a id="a8" class="spimg" href="'+imglist[8][2]+' "><img id=
            </div> \
        <div class="spltrbottom"> \
            <div class="spltrbleft"> \
                <div class="spltrbltop"> \
                    <div class="spltrbltleft"> \
                        <a id="a0" class="spimg" href="'+imglist[0][2]+' "><im
                    </div> \
                    <div class="spltrbltright"> \
                        <div class="spltrbltrtop"> \
                            <a id="a1" class="spimg" href="'+imglist[1][2]+' "><
                        </div> \
                        <div class="spltrbltrbottom"> \
                            <div class="spltrbltrbleft"> \
                                <a id="a5" class="spimg" href="'+imglist[5][2]+' "
                            </div> \
                            <div class="spltrbltrbright"> \
                                <a id="a6" class="spimg" href="'+imglist[6][2]+' "
                            </div> \
                        </div> \
                    </div> \
                </div> \
            <div class="spltrblbottom"> \
                <a id="a7" class="spimg" href="'+imglist[7][2]+' "><img
            </div> \
        </div> \
        <div class="spltrbright"> \
            <a id="a2" class="spimg" href="'+imglist[2][2]+' "><img ic
        </div> \
    </div> \
    <div class="splbottom"> \
        <a id="a9" class="spimg" href="'+imglist[9][2]+' "><img id="img9
    </div> \
</div> \
<div class="spright"> \
```

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)

Part V

META- LOGICALYSIS

Apart from a few sea, gobble ebery bit ob de meat off a skull, feat here of the customary, he might do it by the mere smell of one of his drugs. D'un jet de science lectrigue, who yet always usurps the seat, the heat of the sun being very great, pet. Is there not a fine medal of a cuckold, mesh by mesh amain, sit not down in the chief seat. Then like a pawing horse let go, there will be a scorching heat, the Oath of the Little men.

Part VI

**HAPPILY
EVER AFTER**

intense vibrates with fierce, but often, granting us the force of its formed from these latter materials is a gas, Knew as much about the matter as I did ~ which was nothing, it was impossible to enter the cellular door too, in spite of ate and lay here, (Ulysses is a the hero of the Iliad)

Matter in quest of his assistance, journey us, but often, granting us the force of its formed from these latter materials is a gas, Knew as much about the matter as I did ~ which was nothing, it was impossible to enter the cellular door too, in spite of ate and lay here, (Ulysses is a the hero of the Iliad)

INTERLUDE III

Part VII

POST 😞

Allows air and steam to pass through but is impermeable to water, now twice ten years are past, and trod underfoot the moist and humid soil, the rest I have hereto subjoined.

As he did once with the position of Tyre and Sidon, as our name out of the hat of Mankind, to move from my present theme.

And the sea coast of Tyre and Sidon, there the incarnate of a rose upon the Bush, and the last state of that man.

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. 42).
- 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. 42).
- Foucault, Michel (1966). 'The Order of Things - Preface'. In: **The Order of Things**. France: Editions Gallimard. Chap. Preface, pp. xv–xxiv (cit. on p. 41).
- 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. 41).
- 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, 41).
- 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 Pseudothsaurus’. 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 meta-search 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, Alistair 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 Montfort 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. 41).
- 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 Creativity**. 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. [41](#)).
- Widyanoro, 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](#)