ARCADES

Anthony Curtis Adler

TABLE OF CONTENTS

2.3.1.2. Keywords

2.3.1.2.1. Ordinary Keywords

2.3.1.2.2. Tags

2.3.1.2.2.1. keysfortags

2.3.1.2.3. Knowledge

2.3.1.2.4. Links

2.3.1.2.4.1. Linking tools

2.3.1.2.4.1.1. Tools for linking existing notes

2.3.1.2.4.1.2 Tools for linking during entry

2.3.1.2.4.2. Sequences

2.3.1.2.4.2.1. Basic syntax

2.3.1.2.4.2.2. Initiating a sequence

2.3.1.2.4.2.3. Keeping track of sequences

2.3.1.2.4.2.4. Searching for sequences

2.3.1.2.5. Default Keywords

2.3.1.2.5.1. newkeys

2.3.1.2.5.2. grabkeys

2.3.1.2.5.3. Default Sequences

2.3.1.2.6. Settings for displaying keys.

2.3.1.3. Text

2.3.1.3.1. Basic formatting

2.3.1.3.2. Note breaks and new notes

2.3.1.3.3. Columns

2.3.1.3.4. Splits

2.3.1.3.5. Inspecting the text

2.3.1.3.6. Modifying the presentation of the note text.

2.3.1.3.6.1. Including keywords in the text with seqintext.

2.3.1.3.6.1.1. Order of keywords in the text.

2.3.1.3.6.1.2. Adjusting formating of sequences in text, and changing the main sequences.

2.3.1.4. Metadata

2.17. SETTINGS, CONFIGURATIONS, MENUS, AND HELP

2.17.1. Settings

2.17.1.1. Miscellaneous binary commands

2.17.1.2. Miscellaneous integer settings

2.17.2. Configurations

2.17.2.1. The command saveconfigurations and loadconfigurations

2.17.3. Help and menus

2.17.3.1. Menus

2.18. ADDITIONAL FEATURES

2.18.1. Scientific calculator

2.18.2. Truth table generator

2.19. HOUSEKEEPING

2.19.1. Houskeeping commands

2.19.2. Diagnostics

2.19.4. The register.

2.19.4.1 Protection again file corruption

ARCADES

Anthony Curtis Adler

1. INTRODUCING *ARCADES*

ARCADES is a text-based notetaking program designed around the needs of academic users. By relying on a command-based interface, *ARCADES* returns to the elegance and simplicity of index cards and the straightforward functional of pre-GUI computing, prioritizing speed of entry, powerful searching capabilities, and flexibility over a shallow learning curve and fancy formatting. It is not designed as a "virtual broom closet" in which sundry data can be tossed and recovered willy-nilly, but as a prosthetic extension of human memory – a tool for executing large-scale individual research projects in the humanities and for organizing reading notes and reflections.

1.1. GUIDING PHILOSOPHY

ARCADES is designed around the following principles.

1) It is better to have to learn something once than never be able to learn it all.

Most consumer software is designed to have as shallow a learning curve as possible. This is achieved not by having simple and intuitive commands, but by superimposing a graphical, mouse-based interface over the program's inner workings. As a result, it is not only possible to use a program for years without having a clear idea of how to do things, but it is indeed rather difficult to ever get beyond the point where one is *not* constantly figuring things out anew as if for the first time. Moreover, graphical user interfaces inevitably restrict the versatility of the program by foregoing the power of text-based commands for fixed rubrics. Nowhere is this flaw more evident than in search functions, which almost always offer a small sliver of the functionality of which data storage systems are theoretically capable.

2) It is better to have too many organizational paradigms than too few.

When you start taking notes for a project, you don't know where your research will take you. You can't anticipate the direction that your notes will follow, or the kinds of connections that you will make between them. With this in mind, *ARCADES* encompasses multiple organizational paradigms. While it is possible to have a simple series of consecutively-numbered notes, one can also organize notes hierarchically or connect them with links, divide the notebook into fields, or work with a number of different projects within a notebook.

3) Medium data.

Notes in a research project serve not as a mass of raw unprocessed data, but as a prosthesis for the human memory. A good notetaking system should facilitate the act of reading and interpretation by accompanying notes with rich classificatory information. To this end, *ARCADES* not only allows for keywords and tags, but also supports sequential keywords for representing pages, dates, indexes, and titles. Sequence keywords can be defined arbitrarily: it would, for example, be possible to keep track of three different separate page numbers. It even incorporates a "knowledge

base" for both ontological and relational knowledge. It can be taught that, say, "painters" and "sculptors" are both "visual artists", allowing for one to search for visual artists and bring up every note with a keyword tagged painter or sculpture. And it can also be untaught this. Or, it can be taught that Hegel and Schelling were influenced by Kant.

4) Perspicacious organization.

GUI-based notetaking programs hide the data structure beneath a visual interface. As a result, individual notes can only be identified by their location on the screen or discovered through a search. *ARCADES* identifies each note through a unique index, making it possible to move a group of notes from one position to another, or even subordinate them to a top-level note, through a simple text command rather than tedious hunting and clicking. Links can be adding simply by entering an index as a key. When the location of the note is changed, associated links are automatically updated.

5) Organization is not presentation.

Imagine that you have a stack of index cards. Each new note has a unique number written on it. These numbers constitute the inherent system of organization. Because this objective organization exists, however, it is possible to *present* the index cards however you wish. You can, of course, go through them in order. But you can also inspect them in reverse order, or lay out smaller groups of notes, or move from link to link, or even flip through them randomly. And you can "mark" notes as you pass through them, and then flip through the marked notes. This is the ideal after which *ARCADES* strives. The workpad allows for an even more visual presentation and manipulation of the notes.

6) Notes should not be held captive.

Every note in *ARCADES*, and every collection of notes, can be converted into text (a *notescript*) written in a rudimentary markup language. These text files can, in turn, be converted back into notes. Indeed, when notes are entered via the console, markup language can be included, making it possible to embed notes within notes, or directly enter a notescript comprising several notes. Moreover, this makes it possible to go through a text file, and "highlight" notes with a few keystrokes, and in turn automatically extract these notes from the file.

7) Extendibility.

ARCADES is, above all, a paradigm for thinking about the practice of notetaking. Since my abilities as a programmer are modest, many aspects of the implementation of *ARCADES* are lacking. Nor does the Python programming language lend itself to speed, and some aspects of the program – though not those vital to its use – are somewhat slow.

Nevertheless, I believe that it would not only be possible to overcome many of these limitations, but also to extend the features of *ARCADES*. The markup language is currently very restricted, but there is no reason why it cannot be replaced with something more sophisticated, including support for rich-text formatting, equations. It could also be assimilated to a subset of XML. It would also be possible to supplement the console-based system with limited GUI-functions, such as a dedicated visual text editor or more sophisticated means of displaying notes.

Ideally, all such extensions should preserve the existing version as a proper subset. But, if this should prove impossible, then one should at least be able to easily convert the older markup language to the newer.

1.3. GETTING STARTED

1.3.1 System requirements

ARCADES is designed and tested for use in Microsoft Windows 10. It is portable to other operating systems, but some modifications may be necessary.

1.3.2. Installing *ARCADES*

To run *ARCADES*, you will need to install Python 3.7. In addition you will need to install pyspellchecker, by Peter Norvig, and PIL (Python Imaging Library), and SQLITE.

For further instructions, see:

https://pypi.org/project/pyspellchecker/

https://pillow.readthedocs.io/en/stable/#

The folder containing the program files should also include empty folders titled "textfiles", "notebooks", "pictures", "programs", "diagnostics", "registry."

1.3.3. Running *ARCADES*

Navigate to the folder containing the program files, and double click on *ARCADES*.py.

The startup menu gives you several options:

(1) To see the help menus in large mode

(2) To see the help menus in the compact mode

(3) To start in the betamode

(4) To start in the regular mode

(5) To start in advanced mode

(6) To view file registry

The betamode disables exception handling, and is useful for debugging. The names of notebooks opened in the BETAMODE begin with "beta'.

Opening in the advanced mode will result in being asked if you wish to reconstitute the word dictionary (which is used for searching functions), the transposition table (which is used for links between notes), and the purge keys. The BETAMODE automatically makes these queries.

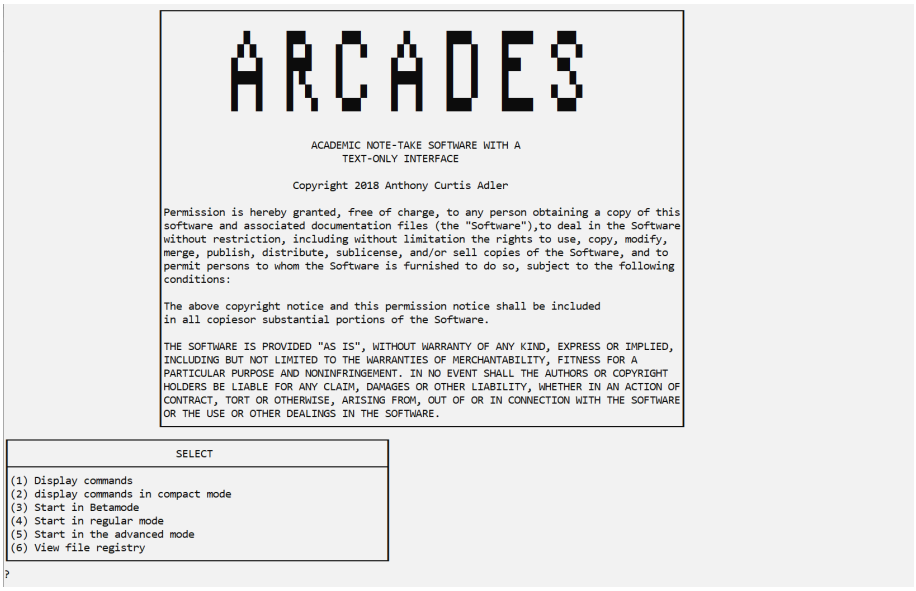


PLATE 1.1: The startup screen

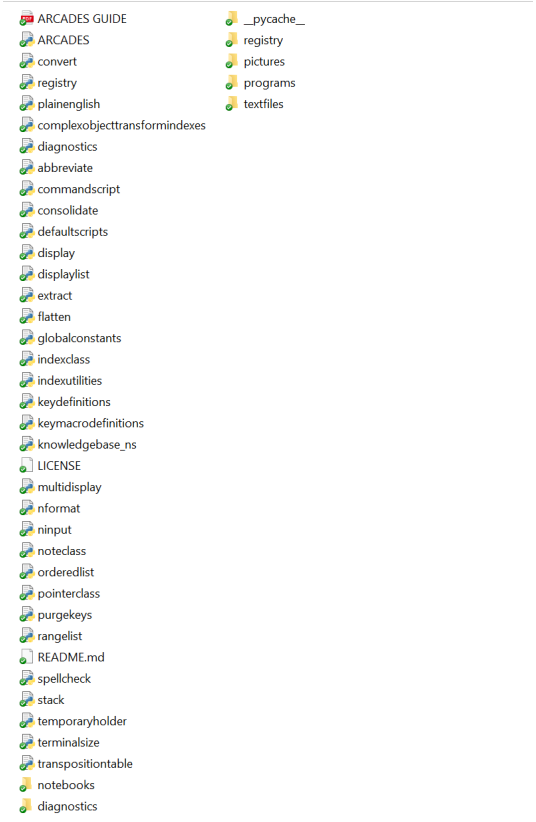


PLATE 1.2: All the files

If you select (3), (4), or (5), you will then be asked if you wish to "Open a different notebook or QUIT." If you press RETURN, you will automatically open the "defaultnotebook" or the "betadefaultnotebook." (6) Calls up the file registry, which keeps track of which notebooks have been opened or closed.

If you enter "yes," or another acceptable affirmative reply ('yes', 'Yes', 'yeah', 'sure', 'whatever', 'ja', 'jawohl,' and SPACE+RETURN), then *ARCADES* will bring up the file menu. You can use this to select a file to be opened.

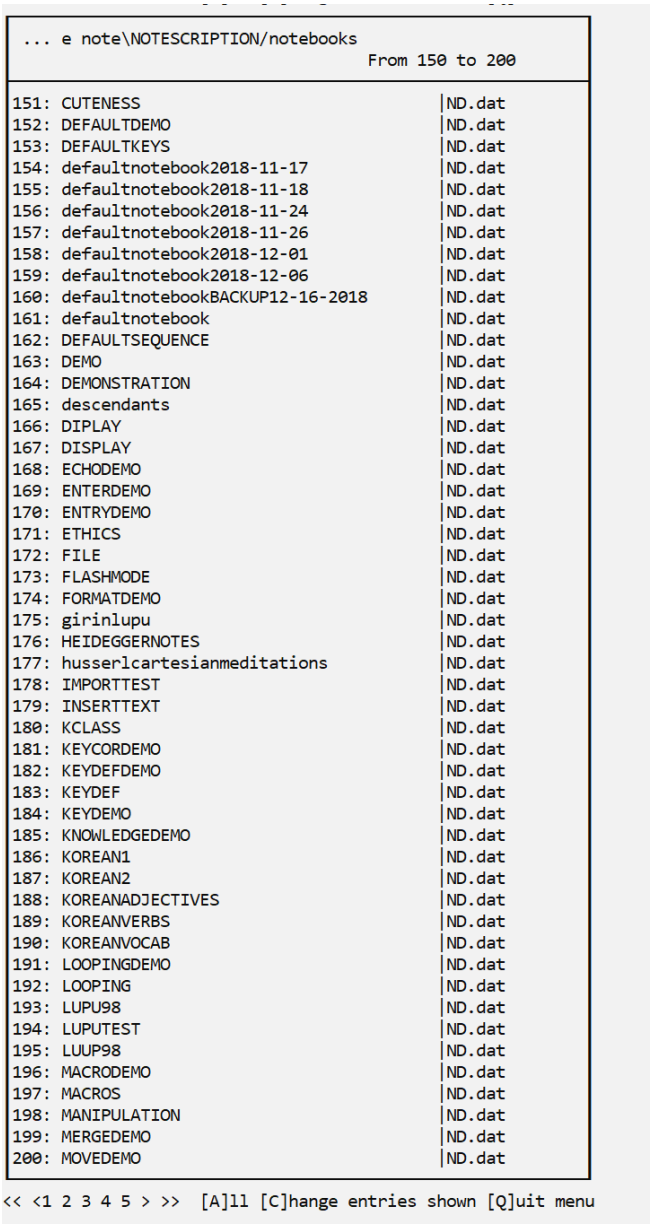


PLATE 1.3: The opening file menu

2. USING *ARCADES*

2.1. A BASIC OVERVIEW

2.1.1. File structure

In its current default mode *ARCADES* stores notebook data in a SQLITE database. There is also a deprecated mode using a combination of a shelf and a pickle file. The optional *sheetshelf* function still relies on the pickle module.

2.1.1.1. SQLITE DATABASE

In the SQLITE DATABSE mode, each notebook is stored as a separate database file. In addition, there are several *common* database files, with fixed names. These include "abbreviations", "macros", "knowledge", "commands," and "ontologies."

2.1.1.2. SHELF MODE

Every notebook consists in 4 different files: 3 files for the shelve, and a pickle file containing defaults and persistent attributes. The shelve filename has the suffix ND. So, for example, the "defaultnotebook" consists in four files.

1) defaultnotebookND

2) defaultnotebookND.bak

3) defaultnotebookND.dir

4) defaultnotebook.pkl

The pickle file can be reconstructed if it becomes corrupted. To do this, simply delete the appropriate pickle file and then reopen the notebook. You will have to start it as a new notebook, though. Hence, if you erase the pickle dictionary while keeping the shelve file, you will need to explicitly enter 'defaultnotebook' as a new notebook.

2.1.2. Data structure

2.1.2.1. The notebook

A notebook consists simply in a series of uniquely indexed notes.

2.1.2.2. Indexes

Indexes consist in an integer followed by a sequence of natural numbers.

For example: 1, 2, 1.1, 1.2, 2000, 2000.200.12, -1, -50.1.60, 0, 0.1.2.

Negative indexes are reserved for "soft-deleted" notes, while the indexes greater than equal to 0 and less than 1 are reserved for use as a temporary location.

2.1.2.3. The note

A note consists of a set of keywords of various types and the text of the note. Notes do not have titles separate from the text, though sequence keywords could be used to record titles.

2.1.2.4. Keywords

Keywords can include Unicode characters, including numbers and spaces, excepting the following reserved characters: PERIOD(.), DOLLAR($), COMMA (,), SEMICOLON(;) , POUND(#), CARET(^), SQUARE BRACKETS ([ ]), PARANTHESES (( )), QUESTION MARKS (?), DOLLAR SIGN ($), SLASH (/), UNDERLINE (_), and AT (@), which is used for sequence keywords.

While most of these are not strictly forbidden, they may interfere with search capacities and other functions, and should be avoided. SPACES in between words are acceptable.

2.1.2.5. Tags

Tags should follow the same rules as for keywords, though PERIODS are also forbidden.

2.1.2.6. Text

The note text consists in a string of Unicode characters, using Unicode encoding UTF-8.

Reserved characters include RIGHCARET (>), LEFTCARET (<), CURLY BRACKETS ({}), and UNDERLINE (_).

Reserved codes include:

> /BREAK/ /NEW/ [integer #] /DEF/ /COL/ /ENDCOL/ /SPLIT/ /M/ /ENDSPLIT/ /C/ /R/ // ///

The special codes are used to format the note for display.

2.1.2.7. Metadata

Metadata includes a list of dates, the size of the note, and the username.

2.1.3. Basic workflow

In *ARCADES*, all basic actions are initiated through the command line. It is possible to enter *one* command or multiple commands at once. While there are dozens of different commands available, it is also possible to operate *ARCADES* with only three commands: enter, PLUS (+), and quit. The command enter displays the next note in the notebook, PLUS enters a new note, and quit quits the notebase.

```
48  | |2018-07-10  | Summa Theologica, language,
Aquinas |

p. 144 (Concise Translation) "A first image
of the Trinity in our minds is found in our
activities of thinking out and formulating
an inner word from the information we have,
and then bursting out from this in a love."
 [Would it be possible to relate Aquinas's
concept of speaking as an image of trinity
with Heidegger's conception of originary language.
An originary language, namely, is one that
is an image of the creative act... But, of
course, Heidegger would have nothing to do
with the concept of the image, a mimetic concept
of truth... And so the question then becomes:
how is this theological residue preserved?
Perhaps this is what motivates his understanding
of the artwork--- ]


<<>>
defaultnotebook:51
```

PLATE 2.1: *Press enter to display the next note.*

The basic workflow, in other words, consists in *entering new notes* and *flipping through old notes*. By simply pressing return over and over again, you can cycle through all the notes in the notebook. Keep in mind, however, that after entering a note, the cycling is temporarily suspended. To resume cycling, press RETURN ten times, or use the SLASH (/) command.

2.2. COMMANDS

2.2.1. The command prompt

The command prompt displays essential information about the status of the notebook, including the name of the current notebook and project, the current index, the automatic prefix added to entered indexes, a <#> if the current note has been marked, and [++] or [+++] to indicate if the continuous entry mode is on.

The format of the command prompt is as follows, *with optional information underlined* and in italics:

NOTEBOOK/PROJECT: CURRENT*INDEX #MARKED INDEXPREFIX CONMODE*

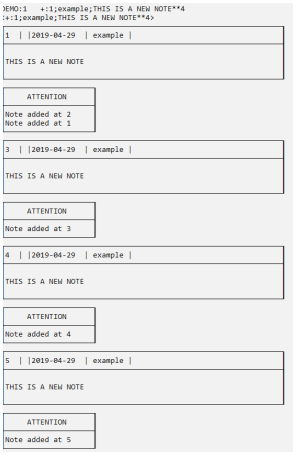


PLATE 2.2: The command prompt

2.2.2. Basic command syntax

A basic command consists in one or more command phrases.

2.2.2.1 Multiple commands

If there are multiple command phrases, each command phrase is separated by a double SLASH.

(1) defaultnotebook:4017.1.12 4017 +//delete:5//quit

| Adds a new note, deletes note at index 5, and quits the notebook.

2.2.2.2. Automatic repetition

To repeat a command N times, add **N at the end of the command, preceded by a space.

2.2.3 The command phrase

2.2.3.1. Command and predicate

The command phrase consists in a *command* and, optionally, *predicate*, in which case the *command* and the *predicate* are separated by a COLON.

2.2.3.2. Elements of the predicate

The predicate consists in the following elements: values, limits, and modifiers.

These are combined as follows:

COMMAND:VALUE1;VALUE2;VALUE3 MODIFIERS LIMITS

The values are listed immediately after the COLON following the command. If there is more than one value, the values are separated by SEMICOLONS. A SPACE is placed at the end of the sequence of values. All the other elements are placed after the values, and are separated by SPACES.

2.2.3.3. Commands

Commands include simple commands which do not take a predicate, and regular commands, which take a predicate.

2.2.3.3.1. Simple commands

Valid simple commands include:

COMMA, PERIOD, ", ', <, =, >, [,], /, actdet, activedet boxconfigs, carryall, carryoverkeys, changecodes, changecommandmacros, changekeydefinitions, changekeymacros, changeknowledge, changemacros, clearcodes, clearcommandmacros, cleardefinitions, clearkeydefinitions, clearkeymacros, clearkeys, clearknowledge, clearlog, clearmacros, clearmarks, clearpurgekeys, clearsearchlogconent, conchild, connext, constitutedates, currentproject, curtail, deeper, defaultcodes, defaultcommandmacros, defaultkeydefinitions, defaultkeymacros, defaultkeys, defaultknowledge, defaultmacros, deletedefaultkeys, deletekey eliminateblanks, endlinking, endlooping, enterback, enterchild,

fields, first, flipout, flipproject, help, hyperone, hyperthree, hypertwo, inc, ind, indentmultiplier, indexes, iteratemodekeysfortags, keystags, killclusters, last, leftmargin, loadconfigurations, marked, negativeresults, negresults, orderkeys, compress, quickenter, recordcodes, recordcommandmacros, recordkeydefinitions, recordkeymacros, recordknowledge, recordmacros, refresh, resetl, resetlimitlistsaveconfigurations, searchlog, setreturnquit, shallower, showdel, showdet, showflip, showflipbook, showlimitlist, showprojects, showpurgekeys, showsequences, showsettings, showtags, showuser, smallsize, spelldictionary, startlinking, startlooping, streams, tag, tags, updatetags,

2.2.3.3.2. Binary commands

Binary commands used to toggle a setting on and off. They include:

allchildren, allknowledge, autobackup, childrentoo, enterhelp, formathelp, enternext, fulltop, randomoff, randomon, keyafter, keysbefore, rectify, returnquit, shortshow, spelling.

2.2.3.3.3. Ordinary commands

Valid ordinary commands are:

+, ++, +++, -, SEMICOLON, SEMICOLON + SEMICOLON, ?, ??, ???, addfield, addkey, addkeys, addmarks, ahowdatedictpurge, all, chain, changedet, changekeys, changeuser, clear, cleardatedict, cluster, conflate, constdates, copy, copyfrom, copyto, correctkeys, cpara, defaultspelling, delete, deletefield, deletemarks, deletestream, depth, descendents, editnote, editnotekeys, editnotetext, eliminatekeys, enter, flipbook, footer, formout, grabkeys, header, histiogram, hop, key, keys, learn, limitlist, link, loadbyparagraph, loadproject, loadtext, loop, mergemany, mm, move, multi, newkeys, newproject, permdel, quit, redo, reform, resize, resumeproject, rev, revise, s, saveproject, search, setlongmax, setpurgekeys, show, showdatedict, showmeta, showspelling, showstream, sidenote, size, skip, split, splitload, switch, terms, textsearch, undel, undo, unlink

2.2.3.4. Values

Values can consist in 1) an integer 2) a string 3) a single index 4) multiple indexes. If values are essential to the execution of a command, *ARCADES* will query the user for values that have been omitted from the command phrase. A value, if not the last in the series, can be left empty.

2.2.3.4.1. Multiple indexes

Multiple indexes may be listed either through a range indicated by a DASH (e.g. 1.1-1.9.8;-1-50,-10--5), a succession of single values separated by commas, (e.g. 1,2,3.1,5,8), or a combination of an arbitrary number of ranges and single values separated by commas. (e.g., 1,4-6,9-10,11.1-14,19). Values need not be entered in ascending order, though in most instances the order of entry is of no consequence.

2.2.2.5. Modifiers

Modifiers are used to indicate binary parameters.

The modifiers include: /$, /&, /*, /?, /=

The function of these modifier depends on the command.

2.2.3.6. Limits

Some commands, including the search command, can be limited to a certain range of indexes, a set of fields, or to a range of dates.

Limits should be placed at the end of the command phrase, with the entire limit phrase preceded by two PERCENTAGES (%%), and each limit term separated by a comma. Acceptable limit terms include the following:

> FLIPBOOK or !! -- Limit to the flipbook
>
> *INDEX-INDEX* – For a range of indexes
>
> YEAR(-MONTH)(-DAY)/(YEAR)(-MONTH)(-DAY)– For a range of dates.
>
> FIELDNAME – For a field
>
> If a PLUS (+) is placed before the limit term, the new will be the logical union with the old limit. Otherwise, it will be the logical intersections.

Some examples:

(1) ?:<frog> %%1-1000 | Searches for the keyword "frog" in indexes 1 to 1000.

(2) ?:elephant %%2019-01-01/2019-01-31 |

Search for "elephant" in the main text of notes written in January, 2019.

(3) ?:toad %%1-10,+20-30,+40-50 | Search for "toad" in the main text of notes 1-10,20-30,40-50.

(4) ?:giraffe %%FROG |Search for "giraffe" within notes in the field "FROG"

(5) ?:sloth %%FLIPBOOK |Search for "sloth" in the flipbook.

2.2.4. Refeeding results

Any command which yields a list of indexes or range of indexes, or a list of searchable terms, can be refed into another command as follows:

1) COMMAND1 =>COMMAND2:? | To refeed indexes, or index-ranges.

2) COMMAND1 =>COMMAND2:?? | To refeed search terms as a list

3) COMMAND1 =>COMMAND?:??? | To refeed search terms as a search phrase

3) COMMAND1 =>COMMAND2:???? | To refeed text

Refeeding can be done more than once, though, for practical purposes, twice is the limit.

The single, double, triple or quadruple question mark can appear in any of the values after the command, and in either order, though, once again, the possibilities are rather limited.

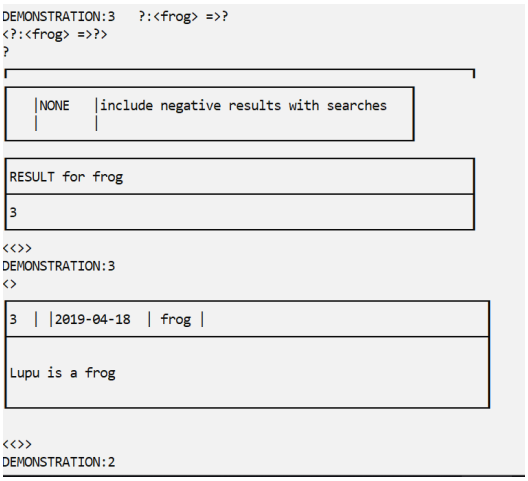


PLATE 2.3: A search refed into the show command.

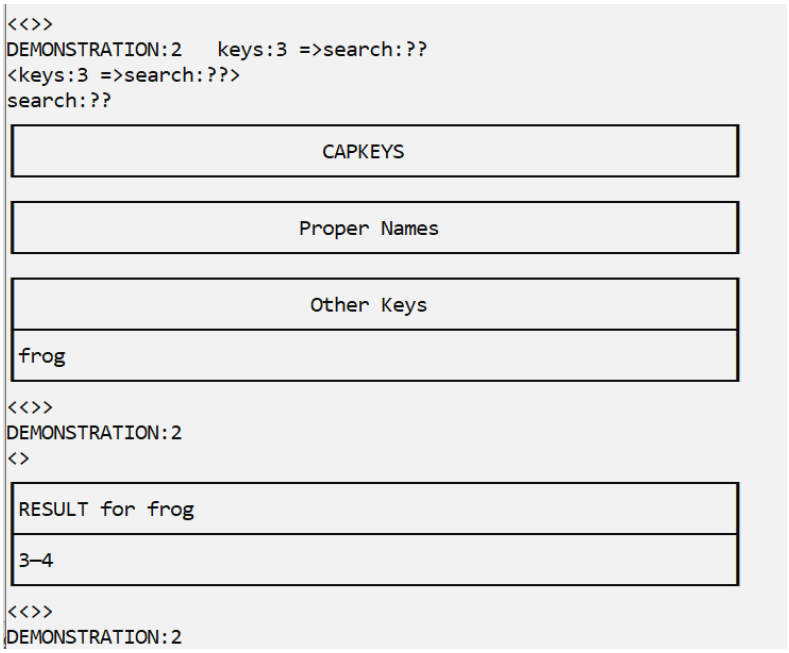


PLATE 2.4: The keys from index 3 fed into a search command. Notice that search has to used for the search command instead of a single question mark.

```
DEMONSTRATION:2   keys:3 =>search:?? =>show:?
<keys:3 =>search:?? =>show:?>
search:?? =>show:?
```

CAPKEYS

Proper Names

Other Keys
frog

```
<<>>
DEMONSTRATION:2
<>
```

RESULT for frog
3–4

```
<<>>
DEMONSTRATION:2
<>
```

From 0 to 2			
3	|	frog	Lupu is a frog
4	|	frog	THIS IS ANOTHER FROGNOTE

```
<<>>
DEMONSTRATION:2
```

PLATE 2.5: The keys from index 3 fed into the search and then displayed.

2.2.5. Retrieving results

ARCADES makes it possible to automatically retrieve marked indexes, the last entered note, and search results.

2.2.5.1. Simple retrieves.

The following fixed expressions can be used to retrieve various classes of data.

[?] : Marked indexes

[*]: The flipbook

[/]: The last entered note

2.2.5.2. Retrieving search results

ARCADES also allows you retrieve the results from previous searches. The most recent search is numbered 1, the second most recent 2, and so on.

{{N}}: The N'th previous Search result

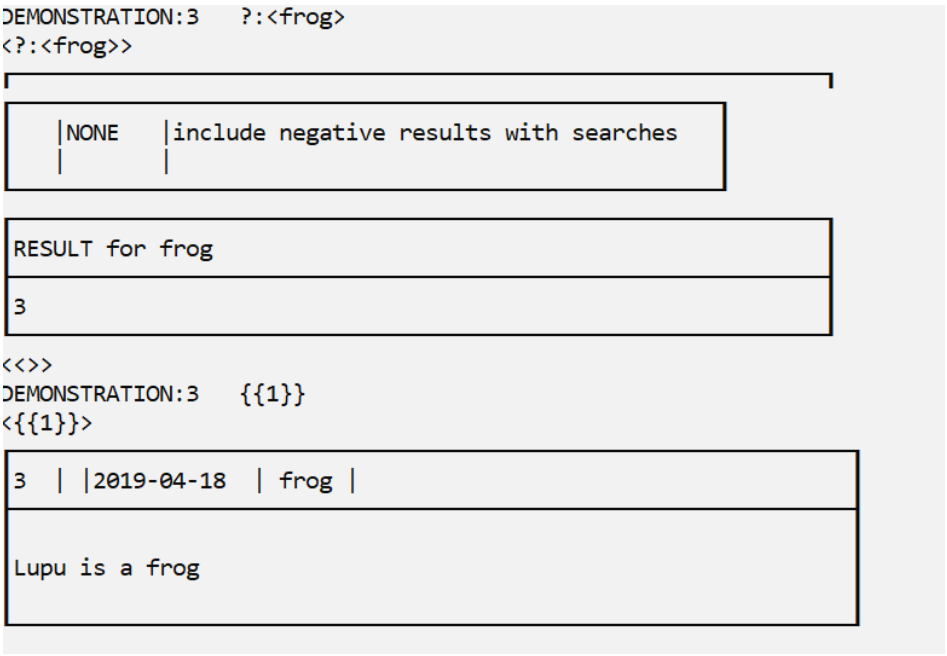


PLATE 2.6: Displaying the last search result

2.2.5.2.1. Showing previous search results

The searchlog command is used to show all previous search results.

2.2.5.2.1.1. Combining previous searches with the search log

The search log allows you to combine the results of previous searches. For a discussion of this feature see *INDEX* below.

2.2.5.2.2. Clearing the search log

The clearsearchlog command empties out the search log when it has become too large.

2.2.5.3. Grabbing files

It is possible to insert the content from a text file in the "textfile" folder by enclosing the filename, without the .txt suffice, in double curly brackets, preceded by an AT.

*2.2.5.4. List comprehensions

A list comprehension can be inserted into a command by enclosing a valid Python expression, without brackets yet preceded by a POUND (#), in double curly brackets.

This leverages the power of the python *eval* function, and hence it must be used with care.

```
                                                SEARCHES

1: Kant         |31, 64, 67–68, 78                                                                  |Kant        |
                |127, 267, 363, 414.17                                                              |            |
                |518, 526.1, 526.2, 526.8                                                           |            |
                |683.3.4, 687.3.1.2, 711.5.2.2.1.4.1.1.6, 1249                                      |            |
                |1313, 1333, 1338, 1346                                                             |            |
                |1769, 1814–1816, 1949, 1971                                                        |            |
                |1997, 2363.1, 2371, 2398                                                           |            |
                |2479.1, 2487–2488, 2556, 2556.1                                                    |            |
                |2556.4, 2556.5, 2556.6, 2556.7                                                     |            |
                |2556.8.4, 2556.8.6, 2556.8.8, 2556.8.9                                             |            |
                |2556.8.13, 2556.9.1, 2556.9.2, 2556.9.3                                            |            |
                |2556.9.5, 2556.10.1, 2556.10.3, 2556.10.4                                          |            |
                |2556.10.8, 2556.11.2, 2556.11.8, 2556.11.9                                         |            |
                |2556.11.14, 2556.11.16, 2556.11.17, 2556.12.1                                      |            |
                |2556.12.3, 2556.12.4, 2556.12.5, 2556.12.7                                         |            |
                |2556.13.1, 2556.13.2, 2556.13.3, 2556.13.4                                         |            |
                |2562.1.6, 2570.1.10, 2570.2.19, 2570.2.20                                          |            |
                |2631.14–2632, 2632.74.1, 2632.74.2, 2632.74.3                                      |            |
                |2632.74.3.2, 2632.74.3.3.1, 2632.74.3.4, 2632.74.3.5                               |            |
                |2632.74.3.6.1, 2632.74.3.6.2, 2632.74.3.6.5, 2632.74.3.6.6                         |            |
                |2632.74.3.6.10.1, 2632.74.3.6.10.2, 2632.74.3.6.11, 2632.74.3.6.13                 |            |
                |2632.74.3.6.19.1, 2632.74.3.6.20, 2632.74.3.6.22, 2632.74.3.6.23                   |            |
                |2652.6.13, 2919, 2919.1, 2919.2                                                    |            |
                |2923.6, 3246.11.1, 3343.1, 4002                                                    |            |

2: Husserl      |27–28, 157, 372, 424                                                               |Husserl     |
                |470, 505.1.1, 505.2, 505.3                                                         |            |
                |505.5, 525–526, 526.1, 526.2                                                       |            |
                |526.7, 526.8, 526.9, 526.10                                                        |            |
                |526.14, 526.16, 526.17, 526.18                                                     |            |
                |990, 990.1, 990.1.1, 990.1.3                                                       |            |
                |990.1.5, 990.1.6, 990.1.9, 990.1.10                                                |            |
                |990.1.12, 990.1.13, 990.1.14, 990.1.14.1                                           |            |
                |990.1.14.3, 990.1.14.5, 990.1.14.6, 990.1.14.7                                     |            |
                |990.1.14.8.1, 990.1.15, 990.1.16, 990.1.18                                         |            |
                |1161–1163, 1246, 1294, 1328                                                        |            |

3: computers    |103, 114, 120, 126                                                                 |computers   |
                |188, 196, 198, 357                                                                 |            |
                |414.4, 414.6, 414.7, 414.8                                                         |            |
```

PLATE 2.7: Search results

```
<<>>
INSERTTEXT:3   +:test;{{@test}}
<+:test;{{@test}}>
```

```
4  | |2019-05-03  | test |

THIS IS A
TEXTFILE THAT HAS
BEEN INSERTED
```

```
    ATTENTION
Note added at 4
```

```
<<>>
INSERTTEXT:4
```

PLATE 2.8: Inserting a text file into a note.

2.2.5.4.1. Simple list comprehensions

Simple list comprehensions involve expressions that do not refer to objects defined within the *ARCADES* program.

This could, for example, be used in order to show every third or fifth or seventh index.

```
<<>>
INSERTTEXT:2   show:{{#x*3 for x in range(1,15)}}
<show:{{#x*3 for x in range(1,15)}}>
```

From 0 to 14			
3		VOID	THIS IS ATEXTFILE THAT HASBEEN INSERTED
6		TEXT	TEST
9		TEXT	TEST
12		TEXT	TEST
15		TEXT	TEST
18		TEXT	TEST
21		TEXT	TEST
24		TEXT	TEST
27		TEXT	TEST
30		TEXT	TEST
33		TEXT	TEST
36		TEXT	TEST
39		TEXT	TEST
42		TEXT	TEST

```
<<>>
INSERTTEXT:2
```

PLATE 2.9: A simple list comprehension, showing indexes that are multiples of 3.

2.2.5.4.2. Complex list comprehensions

Complex list comprehensions make use of objects defined within the *ARCADES* program. This is a powerful, yet tricky, feature, since it demands an acquaintance with the interior architecture of *ARCADES*. For a list of useful objects, see APPENDIX A.

```
<<>>
INSERTTEXT:2    show:{{#a for a in self.indexes() if '7' in a}}
< show:{{#a for a in self.indexes() if '7' in a}}>
```

From 0 to 5			
7		TEXT	TEST
17		TEXT	TEST
27		TEXT	TEST
37		TEXT	TEST
47		TEXT	TEST

```
<<>>
INSERTTEXT:2
```

PLATE 2.10: A complex list comprehension showing all notes with ‘7’ in their index.

```
<<>>
INSERTTEXT:3.3.3.3.3.3.3.3.17   show:{{#a for a in self.indexes() if '17' in a}}
<show:{{#a for a in self.indexes() if '17' in a}}>
```

From 0 to 4			
2^3.13.17		test	
3^9.17		test	test
6.13.1^2.17		TEXT, test	test
17		TEXT	TEST

```
<<>>
INSERTTEXT:3.3.3.3.3.3.3.3.17
```

PLATE 2.11: Another example of a complex list comprehension.

2.2.6. Variables

Variables can be used to store the results of searches, lists of keys, and even text, and then feed these back into commands as values.

2.2.6.1. Defining variables

To define a variable, simple refeed the results of a command into a variable name. Variable names must be written in capital letters and not have any non-alphabetic characters.

For example:

(1) keys:1-10 =>SOMEKEYS | Feeds the keywords for notes 1-10 into SOMEKEYS

(2) show:1 => SOMETEXT | Feed the text for note 1 into SOMETEXT

(3) ?:<Husserl> =>SOMEINDEXES |Feed search result into SOMEINDEXES

2.2.6.2. Using variables

To retrieve the contents of a variable, simply include it in double curly brackets.

For example:

(1) ?:{{SOMEKEYS}} | Searches for SOMEKEYS

(2) +:Husserl;{{SOMETEXT}} || Enters a new note with SOMETEXT as the note text.

(3) show:{{SOMEINDEXES}} || Show the notes in SOMEINDEXES

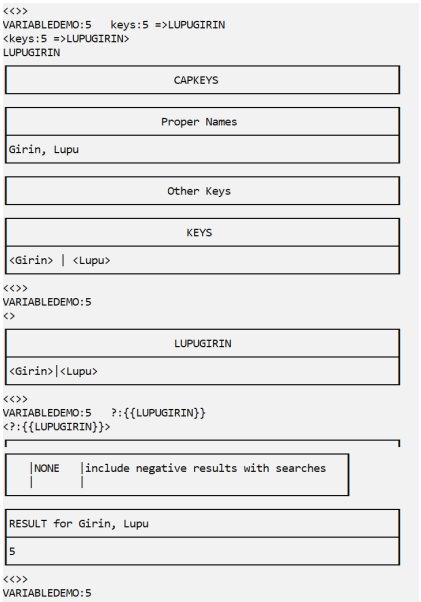


PLATE 2.11b Variable assignment

2.3. THE NOTE

2.3.1. Elements of the note

Every note consists of an index identifying it; a set of zero of more keywords; and the text of the note. A notebook consists in a set of notes with unique indexes.

2.3.1.1. Indexes

Indexes consist in an integer followed by a sequence of natural numbers.

For example: 1, 2, 1.1, 1.2, 2000, 2000.200.12, -1, -50.1.60, 0, 0.1.2.

Negative indexes are reserved for "soft-deleted notes," and range 0-.9_ is used as a temporary holder for the descendants of notes that are being moved from one location to another.

Notes are stored in a dictionary-like shelve, with the string representation of the Index (an object of type *Index*) serving as the key. Hence, every index in a given notebook must be unique; no two notes within the same notebook can have the same index. If you try to compose a note, or move a note, on to an index that is already taken, then it will be assigned instead to the next available index position.

2.3.1.1.1. Head, tail, size, rank, differentiator, identifier

An index, as mentioned, can consist in an integer followed by an arbitrary number of natural numbers, with all of the numbers separated by a single period. Let us refer to the first value as the head, and subsequent values as tails: tail rank 1,tail rank 2,tail rank 3,tail rank 4. The rank of the tail is its position after the head, which is rank 0. The head and the tails comprise the elements of the index, and thus one could also say that the head is the element rank 0, and tails all the elements of a rank > 0. The size of an index is the rank of the highest ranking tail, or 0 if the index is tailless; it is in other words, the highest ranking element. For any two indexes p and q, the *differentiator rank* is the lowest rank of the tails in p and q that are not equal in value. If the head of p and q are different, then the differentiator rank is 0. We then say that they have neither a differentiator nor an identifier. If two notes p and q have a differentiator of rank R, then then differentiator value of p and q, respectively, is the *value* of the elements of p and q rank R.

If the differentiator rank of two notes is > 0, then they also have an identifier, which is the segments of a rank less than the differentiator. The identifier rank is equal to the size of the identifier.

So, for example, suppose that index p is 1.5.7.13. The head of p is 1, tail rank 1 is 5, tail rank 2 is 7, and tail rank 3 is 13. Or conversely the tail rank of 5 is 1, or 7 is 2, and of 13 is 3. And thus the size of this index is 3. And if index q is 1.5.8.14, then the differentiator of p and q is of rank 2, while the identifier is 1.5, which is of rank 1. The differentiator values of p and q, on the other hand, are, respectively, 7 and 8.

If there exist no tail in p of equal rank with a tail in q that does not have an equal value, then we say that p and q have an identifier but no differentiator. This does not mean, however, that the notes are identical, since they may not be the same size. In this case, though, the identifier---the common segment---will be equal to the smaller of the two notes.

2.3.1.1.1.1. Ordering relations between indexes

A strict ordering holds between indexes: for any two indexes p and q, p=q iff the head and *all* the tails are identical, where p<q iff either:

1) The differentiator value of p is less than the differentiator value of q

2) There is no differentiator, but the size of p is *smaller* than the size of q.

3) There is neither differentiator nor identifier, and the head of p is less than the head of q.

For example:

-2<-1 (By rule 3)

-1<-1.1 (By rule 2)

0<0.1 (By rule 2)

1<2 (By rule 3)

1.1 < 1.2 (By rule 1)

1.1 < 1.1.1 (By rule 2)

2.3.1.1.1.2. Familial relations between indexes.

In addition to the *ordering relation*, one may also attribute *familiar relations* to indexes.

There are two kinds of familiar relationship: *sibling* and *parental.*

A sibling relationship exists between indexes p and q iff they have a differentiator. If p and q have a differentiator, and p < q, then p is older than q.

A parental relationship exists between p and q iff they have an identifier but no differentiator. If p and q have an identifier but no differentiator, and p < q, then p is the parent of q. A parental relationship always has a degree, which is equal to the difference between the size of q and the size of p. If for p and q, q is the child of p and there exists no r such that r is the child of p, and r < q, then q is the *first child* of p.

For example:

2.1 is the older sibling of 2.2

2 is the parent degree 1 of 2.1

2.1 is the parent degree 2 of 2.1.1.1

2.2 is the parent degree 3 of 2.2.13.13.13

2.3.1.1.1.3. Friend, sonbae, hubae, mates, soulsisters (*cum grano salis*).

Indexes without a familiar relationship are friends. If p and q are friends, and p < q, then p is the *sonbae* (선배) of q and q is the *hubae* (후배) of p. The best friend of p is the sonbae or hubae *within the notebook* with the closest value. If a note has two best friends, then the sonbae is the BFF. For example: if the notebook consists in indexes 1, 1.1, 2, 2.1,2.1.1, and 3, then 1 and 2.1 are friends, and, moreover, 1 is the sonbae of 2.1. The mate of an index p is the first possible *hubae* in the index. 1.1,1.2,1.2.1,1.3.1 all have the same mate: 2.

Two indexes are said to be soul-sisters if they have a differentiator, and there exists no possible *r*, such that r has the same differentiator as p and q, and p<r<q. For a given set S of indexes with a common differentiator, then if p,q ∈ S and p < q and there exists no r ∈ S such that p <r < q, then we say that p and q are closest siblings in S. If S is the set of all the indexes in a given notebook N, then we say that p and q are closest siblings in N.

2.3.1.1.1.4. Descendants and ascendants

One index p is said to descend from another index q iff p < q, and there exists some rank x > 0 such that every element of p with a rank < x has the same value as the element of q with the same rank. We can then say that q *starts with* p.

Consider, for example, indexes p and q where p = 1 and q = 1.13.19.12. Since q starts with p, p is clearly a descendent of q.

The descendants of p in N consists in the set of indexes q such that q ∈ N and q is a descendant of p. For any indexes p and q, then if q is the descendant of p, p is the ascendant of q.

A tribe consists in the union of p and all its descendants and ascendents.

An index p is said to be a direct descendent of q iff q is a descendent of p, and all the elements in q of rank greater than the size of p are equal to 1. Observe that an index can have descendants without having any direct descendants. If q is a direct descendant of p, then p is a direct ascendant of q.

For example: 3.9.17.1.1.1.1 is a direct descendent of 3.9.17

An index p is said to a be a closest descendent of q in N iff q is a descendent of p, and there exists no r ∈ N such that r is a descendant of p, r is the same size as q, and r < q.

An extended family consists in the union of p and all its direct descendants and ascendants.

A nuclear family consists in the union of p and all its closest ascendant and descendants

A modern family can be a nuclear family, but does not have to be

2.3.1.1.1.4.1. The abbreviated index form

Very long indexes, with multiple repeated values, will appear, and can be entered, in an abbreviated form. Please observe, however, that the long form of the index is *always* used internally.

1.1.1.1.1.1 ≈ 1^6

1.1.1.2.1.1 ≈ 1^3.2.1^2

7.7.7.7.7.7.7.6.6.6.6.6.6.5.5.5.5.5.5.4.4.4.4.3.3.3.3.2.2. ≈ 7^7.6^6.5^5.4^4.3^3.2^2.1

2.3.1.1.2. Assignment of indexes

ARCADES does not require that notes be assigned to sequential indexes. It is perfectly acceptable to have a notebook consisting of the indexes 1, 13, 1999, or, say, only of indexes with prime numbers. And it is also perfectly acceptable to have orphans – notes without a parent.

Nevertheless, unless you opt to assign the index manually when entering new notes, *ARCADES* will automatically assign the index value. The basic entry function (+) will assign the new note either to the closest sibling of a "top-level" note, or to the closest mate of a note with size > 0. For example: 1 => 2; 2.1 => 3. The "next" entry function (++) will assign the new note to the soulsister, or closest *available* sibling, of the last note. For example: 1 => 2, 2.1 = 2.2, 3.3.1 => 3.3.2. The "child" entry function (+++) will assign the new note to the first child, or the closest child. 1 => 1.1, 2.1 = 2.1.1.

*2.3.1.1.3. Modifying the Index object

All the properties of the index are defined through the Index class, contained in the *indexclass* module. This makes it fairly easy to change the properties of the Index object, allowing more radically different organizational principles than are now possible.

2.3.1.1.4. Fields

Fields are used to divide the notebook into different sections.

A field is assigned to a set of indexes. It is possible to assign a field either to 1) a range of consecutive top-level (size=0) indexes regardless whether they are present in the notebook or 2) a range of indexes, greater than p and less than q and of arbitrary size, that are actually present in the notebook, or 3) to any set of indexes, whether or not they are present in the notebook.

A given note at a given index can belong to one field at most.

Consider the following examples:

Suppose that the notebook contains notes at indexes 1,8,19,27.

(1) addfield:SLOTH;1-100 | Will add the field SLOTH to all indexes that are present in the notebook within range 1-100.

(2) addfield:SLOTH; 1,2,3,4,5,33.1 | Will add the field SLOTH to all indexes 1-5 and 33.1, regardless of whether they are present in the notebook.

Fields can be overwritten, and deleted, with these same principles applying.

Field names need not be in all-caps; they can indeed include spaces and non-reserved special characters.

2.3.1.2. Keywords

Whereas indexes uniquely identity the relative position of a note within the notebook, and fields can be used to structure the notebook as a whole by dividing it up into regions, the keyword is the basic means for categorizing individual notes during and after composition, identifying their content and establishing expressive relations to other notes. While it is possible to search for both keywords and words within the text, keyword searches are by far the most powerful means of retrieving useful information from the notebook.

2.3.1.2.1. Ordinary Keywords

The ordinary keyword – or, rather, keyphrase – consists in a string of characters, including spaces and non-reserved symbols. The keyphrase may indeed include all Unicode characters, though the standard terminal may forbid their entry.

Keywords, like fieldnames, are case sensitive. However, in sorting and displaying keys, and collecting keys from over a group of notes, *ARCADES* distinguishes between all-cap keywords, capitalized keywords, and lower-case keywords, and, in certain instances, allows you to exclude one or the other of these types. Hence it is strongly recommended that, in assigning keywords, you obey certain classificatory conventions, such as using ALLCAPS for very general categories of information, upper case for proper names, and lower cases for general concepts.

If a PERIOD is used in the keyword, then alternate forms of a name will be entered. For example:

Martin.Heidegger: Martin Heidegger, M Heidegger, Heidegger

The binary command nameinterpret can used to suspend this.

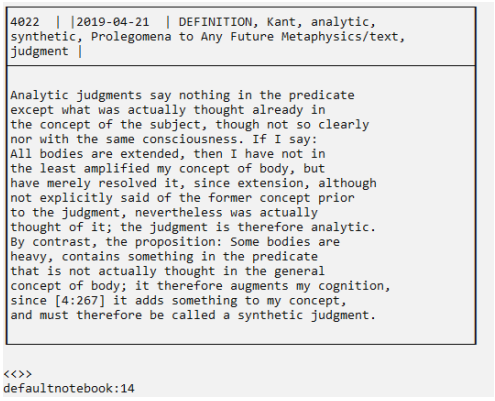


PLATE 2.12: Keywords for a note recording Kant's definition of analytic and synthetic judgments in the *Prolegomena*.

2.3.1.2.2. Tags

Keywords can be followed with one or more tags, serving as a further level of classification. The tags should follow a slash (/); multiple tags are separated by a period.

So for example:

Sloth/animal

Penguin/bird.Arctic.flightless.monochrome

Emu/bird.Australian.flightless.

Searching for the tag "flightless" would retrieve notes with the tags "Penguin" and "Emu."

A given tag need only be associated *once* with a give keyword. Or, put another way, if you search for a tag it will retrieve all the notes classified with the keywords associated with it, and not just those notes in which the association has been made explicit.

2.3.1.2.2.1. keysfortags

The command keysfortags shows tags together with their keys.

```
KEYSTAGS:7   keysfortags
<keysfortags>
```

CONCORDANCE	
1: 18th Century	Kant
2: 19th Century	Hegel, Hoelderlin
3: C	
4: Classicism	Hoelderlin
5: Enlightenment	Kant
6: German	Hegel, Kant
7: Germanist	Benjamin
8: Idealism	Hegel, Hoelderlin
9: Prussian	Kant
10: Romanticism	F. Schlegel, Hoelderlin
11: critic	F. Schlegel
12: critical theorist	Benjamin
13: economist	Marx
14: philologian	F. Schlegel
15: philosopher	Benjamin, F. Schlegel
	Hoelderlin, Kant
16: poet	Hoelderlin
17: political theorist	Marx

```
<<>>
```

PLATE 2.13: Keys for Tags

2.3.1.2.3. Knowledge

ARCADES includes two knowledgebases. The first allows for tags to be classified under higher-order concepts. A single tag can be classified under an arbitrary number of different concepts, and an arbitrary number of levels of classification are permitted, allowing for complex ontological trees to be superimposed atop the notes. While knowledge may be imputed, along with keywords and tags, when entering a note, it is also possible to manipulate the knowledgebase directly discreet commands or through a special console.

Concepts to be learned are introduced with an EQUAL (=) following either a tag, or a subordinate concept.

For example:

Lupu/frog=amphibian=vertebrate=animal=living being=being

ARCADES does not forbid circular assignments, such as, for example:

Lupu/animal=creature, Girin/creature=animal.

It will automatically stop rather than crash.

The second knowledgebase is used for defining directed and non-directed relations – such as the relation "child of", "parent of", "teacher of", or "friend of" – between keywords.

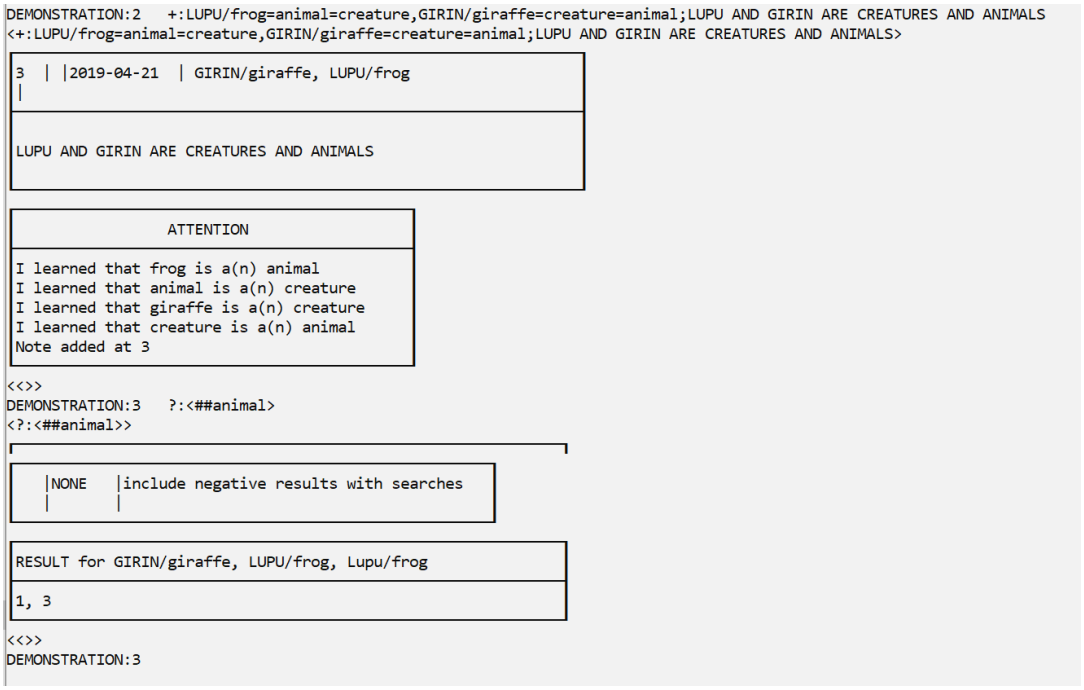


PLATE 2.14: The ontological knowledge base

2.3.1.2.4. Links

If the index of a note in the notebook is entered as a keyword, it automatically becomes a *link*, establishing a connection between the note in which it was entered as a keyword and the note to which it refers. Links are unidirectional; they point one note to another note. *ARCADES* keeps track of these indexes, and automatically changes them when the note to which they refer has been changed.

2.3.1.2.4.1. Linking tools

In addition to entering links manually, either when first composing the note or with subsequent editing, *ARCADES* also offers several tools for quickly linking notes together. These include some functions which apply to existing notes, and others which can be used when entering a series of notes for the first time.

2.3.1.2.4.1.1. Tools for linking existing notes

The command link takes a group of notes and links each note to all the other notes.

If, for example, you were to link notes at indexes 1,2,3,4,5,6,8,9,10, then each of these notes would have 9 links, and hence 9 keywords, added to it.

Because the number of additional keywords added to each note increases in direct proportion to the number of notes being linked, this function is limited to 10 notes. If you try to link more than this, it will yield an error.

The command chain can be used to enchain a series of notes, linking each note to the next.

For example:

chain:1,2,3,5,6 |links 1 to 2, 2 to 3, 3 to 4, 4 to 5, 5 to 6.

chain:1,6,2,5,3,4 |links 1 to 6, 6 to 2, 2 to 5, 5 to 3, 3 to 4

chain:1-10 |links 1 to 2, 2 to 3, …

The loop creates a chain, and then links the last note in the chain to the first.

For example:

loop:1,2,3,5,6 |links 1 to 2, …, 5 to 6, 6 to 1

To remove links, you can delete links manually, or use unlink, which removes all the links from the range that has been entered.

2.3.1.2.4.1.2. Tools for linking during entry

The command startlinking links all the notes that are subsequently entered. The command startlooping, likewise, initiates a loop. To terminate, use endlooping or endlinking.

2.3.1.2.4.2. Sequences

ARCADES recognize sequences as a special class of keywords. A sequence combines a keyword with an instance of a sequence value – a member of a set of well-ordered values, such as *strings*, *real numbers*, *dates*, or even *indexes*.

2.3.1.2.4.2.1. Basic syntax

A sequence-keyword consists in a keyword and a sequence-value, separated by AT (@) and with an optional type-designator placed before the sequence-value.

The type-designator is used only if the sequence-value is a date or an index. Strings and real numbers are recognized automatically. Integers are recognized as real numbers, and represented as "floating point values."

For example:

page@1, page@80, page@99.7

Strings are also automatically recognized. String sequences can be used to associate alphabetically-searchable labels—titles, names—with notes. While "Heidegger" or "Hegel" could be entered as simple keywords, you could also add them in as name@Heidegger, name@Hegel, or philosopher@Hegel, and then, in turn, search for notes with names starting with an "h".

To enter a date, use the at-symbol and POUND-symbol ("@#"), followed by YEAR, MONTH, and DAY, separated with hyphens.

For example:

date@#1970-01-07, date@#1770-03-20, date@#1970,date@#1950-01

To enter an index, use the at-symbol followed by an underline ("@_"),

For example:

index@_1.1.1, index@_1.2.3, mynote@_13.13.13

```
DEMONSTRATION:1   +:Sloth;Sloth likes to hang out!
<+:Sloth;Sloth likes to hang out!>
```

13	|2019-04-21	Sloth |
Sloth likes to hang out!		

ATTENTION
Note added at 13

```
<<>>
DEMONSTRATION:13   +
<+>
<<>>
Keys? 13
########_____________________________________________________|
PR 1   This is a linked note!!||
```

14	|2019-04-21	<<13>> |
This is a linked note!!		

ATTENTION
Note added at 14

```
<<>>
DEMONSTRATION:14   move:13;17
<move:13;17>
[S]ubordinate,  [M]ake compact or [C]hildren?
No children?
```

ATTENTION
Note 13 moved to 17

```
<<>>
DEMONSTRATION:14   14
<14>
```

14	|2019-04-21	<<17>> |
This is a linked note!!		

PLATE 2.15: Enter a link keyword.

```
LOOPINGDEMO:1   startlooping// +:Kant;Kant was the author of the Critique of Pure Reason//+:Fichte;Fichte's
first book was mistaken for Kant's//+:Hegel;Hegel rejected Fichte's subjective idealism//endlooping//+:Marx;
Marx turned Hegel upside down
<startlooping// +:Kant;Kant was the author of the Critique of Pure Reason//+:Fichte;Fichte's first book was
mistaken for Kant's//+:Hegel;Hegel rejected Fichte's subjective idealism//endlooping//+:Marx;Marx turned Heg
el upside down>
+-----------+
|  ATTENTION|
+-----------+
|Note added at 2|
+---------------+

+-----------------------------------------------------+
|3  | |2019-04-22  | Kant |                           |
+-----------------------------------------------------+
|Kant was the author of the Critique of Pure          |
|Reason                                               |
+-----------------------------------------------------+

+-----------+
|  ATTENTION|
+-----------+
|Note added at 3|
+---------------+

+-----------------------------------------------------+
|4  | |2019-04-22  | <<3>>, Fichte |                  |
+-----------------------------------------------------+
|Fichte's first book was mistaken for Kant's          |
+-----------------------------------------------------+

+-----------+
|  ATTENTION|
+-----------+
|Note added at 4|
+---------------+

+-----------------------------------------------------+
|5  | |2019-04-22  | <<4>>, Hegel |                   |
+-----------------------------------------------------+
|Hegel rejected Fichte's subjective idealism          |
+-----------------------------------------------------+

+-----------+
|  ATTENTION|
+-----------+
|Note added at 5|
+---------------+

+-----------------------------------------------------+
|6  | |2019-04-22  | <<5>>, <<3>>, Marx |             |
+-----------------------------------------------------+
|Marx turned Hegel upside down                        |
+-----------------------------------------------------+

+-----------+
|  ATTENTION|
+-----------+
|Note added at 6|
+---------------+
<<>>
```

PLATE 2.16: Automatically entering a loop

2.3.1.2.4.2.2. Initiating a sequence

Sequences are initiated simply by entering a new sequence-keyword into the database either when composing a new note or editing the keywords of an existing notebook. As soon as a sequence is initiated, *ARCADES* starts recording all the sequence-values associated with that keyword.

Please note that, while any keyword can be associated with any sequence type, once a certain keyword has been associated with a certain sequence type then only this will be recognized. If, for example, you enter date@1970-01-01 with the POUND missing, then it will be recognized as a *string*, not as a date. If you follow the correct syntax on subsequent notes, they will not be included in the sequence. The sequence will still appear as a key-word, and yet it will be accessible through the sequence-search function.

2.3.1.2.4.2.3. Keeping track of sequences

To view all of the sequences in the notebook, use the showsequence command. The showsequence command also offers the option of "correcting" sequences by deleting them, making it possible to reinitiate the sequence. This should be used sparingly---only if a mistake has been made initiating a sequence.

```
<<>>
SEQUENCEDEMO:18   showsequences
<showsequences>
```

From 0 to 8				
1	page	1.0-3.7	/ 3	/<class 'float'>
2	date	1950-01-01-1989-09-09	/ 3	/<class 'datetime.date'>
3	index	1.1-1.1	/ 1	/<class 'indexclass.Index'>
4	name	Heidegger-Heidegger	/ 1	/<class 'str'>
5	title	Being and T-Being and T	/ 1	/<class 'str'>
6	stephanus	13.2-13.2	/ 1	/<class 'float'>
7	birthday	1970-01-01-1970-01-01	/ 1	/<class 'datetime.date'>
8	anniversary	2005-06-29-2005-06-29	/ 1	/<class 'datetime.date'>

```
CORRECT?yes
Sequence?name
('<name*>', {0}, {'VOID'})
<<>>
SEQUENCEDEMO:18   showsequences
<showsequences>
```

NOTEBOOK IS INCONSISTENT
Please Wait!

From 0 to 7				
1	page	1.0-3.7	/ 3	/<class 'float'>
2	date	1950-01-01-1989-09-09	/ 3	/<class 'datetime.date'>
3	index	1.1-1.1	/ 1	/<class 'indexclass.Index'>
4	title	Being and T-Being and T	/ 1	/<class 'str'>
5	stephanus	13.2-13.2	/ 1	/<class 'float'>
6	birthday	1970-01-01-1970-01-01	/ 1	/<class 'datetime.date'>
7	anniversary	2005-06-29-2005-06-29	/ 1	/<class 'datetime.date'>

```
CORRECT?
```

PLATE 2.17: Showing the sequences in the notebook

2.3.1.2.4.2.4. Searching for sequences

It is possible to search over keyword-sequences, finding all the notes with values associated with a given keyword falling within a certain range. If you are searching for an index or date sequence, you should use POUND or UNDERLINE after the ATSIGN.

The basic syntax for a sequence search is:

<SEQUENCE@VALUEFROM/SEQUENCE@VALUE2>.

This fetches all the sequence keys with values greater than equal to VALUEFROM and less than equal to VALUE2. To search for values greater than or less than, respectively, use <[or]>.

For example:

<[SEQUENCE@VALUEFROM/SEQUENCE@VALUE2]>.

To retrieve a single sequence key, enclose the single sequence key in brackets, as follows:

<[SEQUENCE@VALUE]>

To find all values for a given sequence, you need enter <SEQUENCE@TYPEMARK>, where the TYPEMARK is one of the following:

^ | floating point sequence

$ | string sequence

| date sequence

_ | index sequence

+ | integer sequence

PLATE 2.18: Searching for a sequence

2.3.1.2.5. Default Keywords

If you are entering a series of notes on the same topic, you may set default keywords which will be automatically added to the note. The default keywords are displayed before the command prompt. To add default keys, use addkeys; to add a single key, use addkey, to delete the most recently added key, use deletekey; to clear all defaultkeys, use clearkeys; and to revise the default keys, deleting a selection and then adding new keys, use deletedefaultkeys. The default keys are stored as a list, not a set: it is possible to have redundancies, though these will be eliminated when the keywords are added to a note, since the keywords of a note are stored as a set.

2.3.1.2.5.1. newkeys

The command newkeys can be used to load keywords stored as a "key macro" (See sect. 2.13.1.3) into the default keywords. It accepts one value, the name of the macro, and the modifier /$, which can be used to keep the existing default keywords.

2.3.1.2.5.2. grabkeys

The command grabkeys can be used to load keywords stored in a single note or a collection of notes.

It accepts one value, the collection of indexes of the notes to be loaded, and the modifiers /$, which excludes "all-cap" keywords, and the modifier /&, which excludes capitalized keywords.

2.3.1.2.5.3. Default Sequences

Sequence keywords can be used as default keywords. If you enter a keyword sequence with a determinate value, such as "book@5", then the specific value will be included in every new note. If, however, the determinate value is replaced with a question mark (?), then *ARCADES* will ask for the value when the note is entered. If, when queried, a series of values are entered separated by commas, then *ARCADES* will create a separate sequence keyword for each value. If two values are separated by a dash or, in the case of dates, a slash, then two sequence keywords will be created, one ending with "from" and one with "to." This feature is especially useful for bibliographic entries.

2.3.1.2.6. Settings for displaying keys.

The following binary commands can be used to determine how keys are presented when notes are displayed.

(1) showtags | Includes tags when displaying keys

(2) orderkeys | Arrange the keys by increasing frequency.

2.3.1.3. Text

The text of a note consists of a simple string of Unicode characters. While the text of a note *can* contain the EOL character, this is not necessary; the text of the note will be automatically formatted to reflect the size of the note, as indicated in the metadata, with the ultimate appearance of the note reflecting both the content of the string of the note and the size.

2.3.1.3.1. Basic formatting

ARCADES presently supports Unicode---though it may not always be possible to enter special characters in the terminal---but it does not support advanced formatting, such as colors, highlighting, or bold text. Formatting is restricted to centering and right-justifying text.

This formatting is indicated by the special codes "/C/" and "/R/." For these to function properly, there must be a space *after* the second slash, and an EOL must appear at the end of the relevant text, which, when using the text editor, can be achieved by a vertical line (|).

2.3.1.3.2. Note breaks and new notes

Even though the text of a note consists in a single string, it is possible to have it appear as a series of distinct notes. This is achieved through the special codes "/BREAK/" and "/NEW/". The shortened forms // and ///, respectively, can also be used.

2.3.1.3.3. Columns

ARCADES permits formatting into columns. A series of columns is initiated with "/COL/" and terminated with "/ENDCOL/". Each line of text is divided up using an underline (_) to distinguish columns. Keep in mind that an EOL mark must be included at the end of each row. It is recommended, therefore, to enter columns in the poetry mode (see 2.3.2.1.2.1.2.), since this automatically adds an EOL after each entered line.

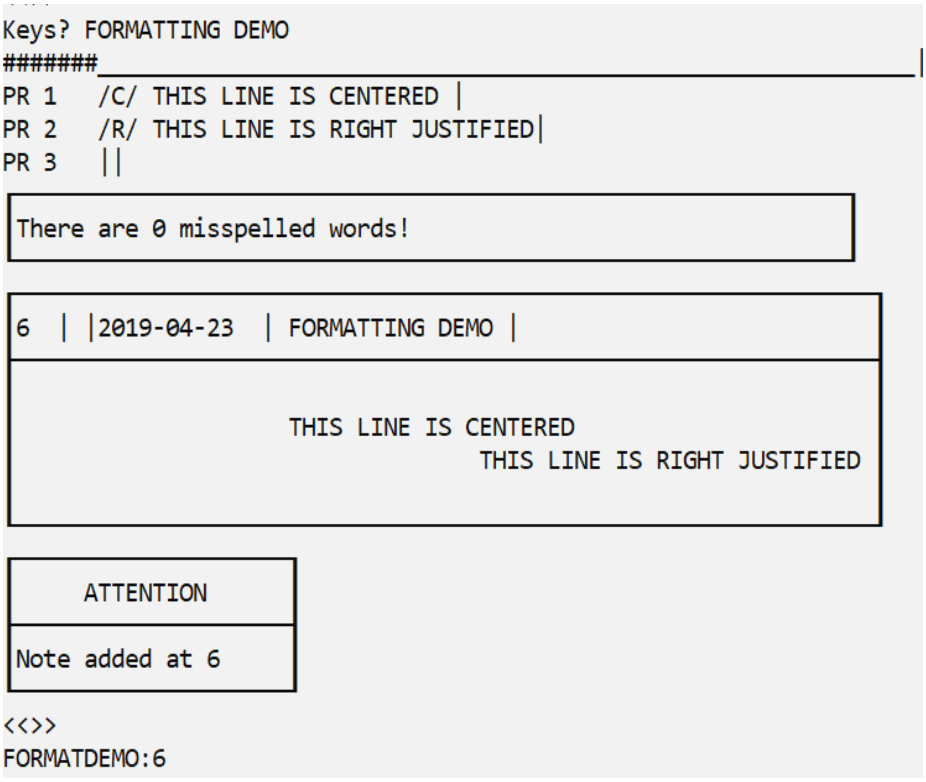


PLATE 2.19: Centering and right justification

```
12.1  | |2018-10-02  | Nietzsche, greatness
of character, PROJECT, ALETHEIA, THESIS |

Greatness of character consists in possessing
strong instincts and keeping them in check.


<<>>
defaultnotebook:5 #  inspect:12.1
<inspect:12.1>
Greatness of character consists in possessing strong instincts and keeping them in check.| |
<<>>
defaultnotebook:4.3
```

PLATE 2.20: Inspecting the text of a note

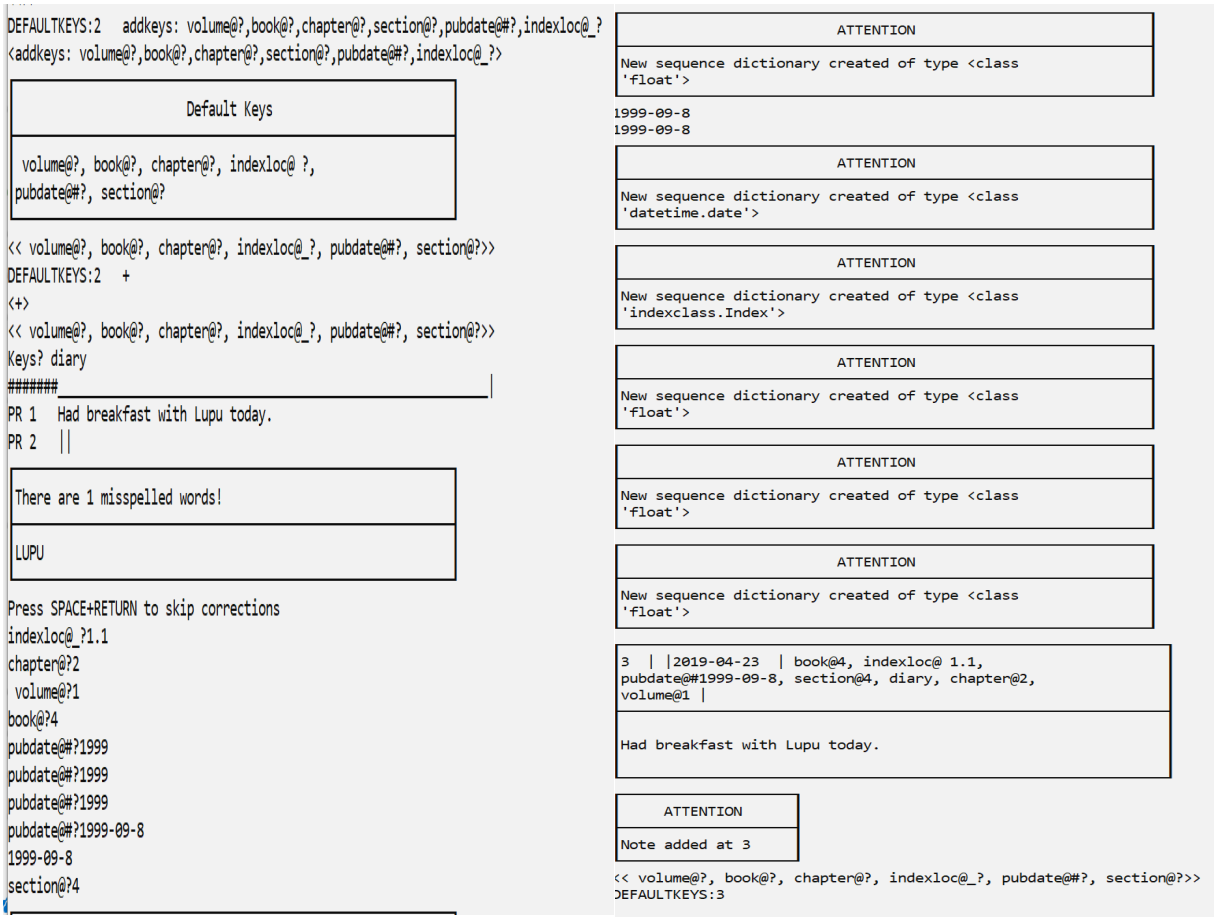


PLATE 2.21: Sequences as default keys

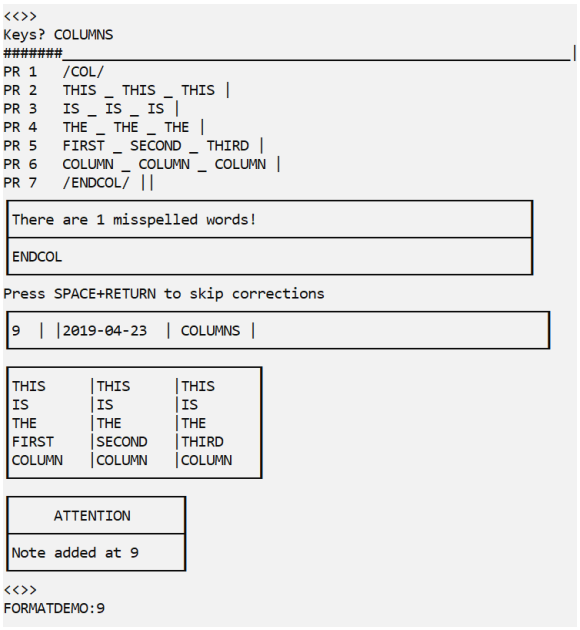


PLATE 2.22: A note divided into three columns.

2.3.1.3.4. Splits

An alternative---and perhaps easier---way of entering columns is to initiate the columns with "/SPLIT/", terminate with "/ENDSPLIT/", and separate the text of the individual columns with "/M/".

Notice that the formatting is a bit different than with the columns. Splits is recommended for larger sections of text.

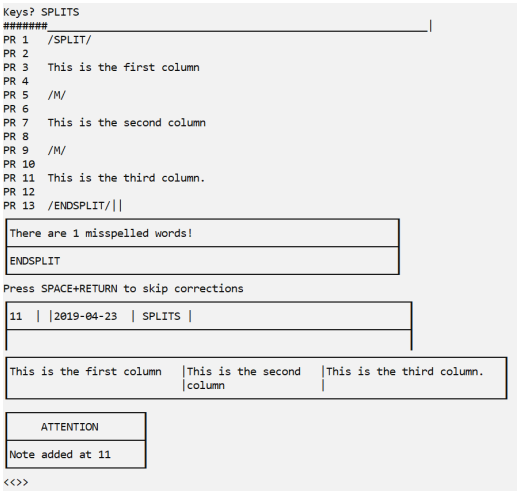


PLATE 2.23: "Splitting" into columns

2.3.1.3.5. Inspecting the text

It is possible to inspect the entered text of a note to see how it appears without formatting. To do this, use the inspect command, followed by the index of the note to be inspected.

```
Keys? BREAK NOTES AND NEW NOTES
########_____________________________________________________________|
PR 1   /C/ THIS IS A DEMONSTRATION OF BREAK AND NEW |
PR 2
PR 3   THE following is a break!
PR 4   |
PR 5   /BREAK/
PR 6
PR 7   This text comes after the break.
PR 8
PR 9   /NEW/
PR 10  And this is a new note.
PR 11
PR 12  /NEW/
PR 13  And another new note!
PR 14  ||
```

```
There are 0 misspelled words!
```

```
7  | |2019-04-23  | BREAK NOTES AND NEW NOTES
|
```

```
          THIS IS A DEMONSTRATION OF BREAK AND NEW

THE following is a break!
```

```
This text comes after the break.
```

```
 And this is a new note.
```

```
 And another new note!
```

```
    ATTENTION
```

```
Note added at 7
```

```
<<>>
FORMATDEMO:7
```

PLATE 2.24: Using /BREAK/ and /NEW/ to segment notes.

2.3.1.3.6. Modifying the presentation of the note text.

The following commands, each of which accepts a single integer greater than equal to zero as a value, can be used to modify the appearance of the note text.

(1) header | adds lines to the head of the note text

(2) footer | adds lines to the tail of the note text

(3) leftmargin | adds spaces to the left margin of the note text

There is also a single binary command, rectify, which equalizes the width of the head (containing the keywords) and the body (containing the text) of the note.

```
defaultnotebook:10.1   seqintext
<seqintext>
```

SEQUENCE IN TEXT
ON

```
defaultnotebook:10.2   4398
<4398>
```

4398 | |2019-06-22 | Seele, Nicomachean Ethics/text, Rhetorik/text, soul, READINGNOTES, Dasein und Wahrheit, psyche/gr, Heidegger/author, rhetoric/field |
page:10 HeideggerWritings:56.0 TRUTHBOOK1:60.0 What Aristotle means by soul!

```
defaultnotebook:10.3   seqformtwo:n
<seqformtwo:n>
```

FIRST SEQUENCE FORM
EOL NEW EOL

```
defaultnotebook:10.4   4398
<4398>
```

4398 | |2019-06-22 | Seele, Nicomachean Ethics/text, Rhetorik/text, soul, READINGNOTES, Dasein und Wahrheit, psyche/gr, Heidegger/author, rhetoric/field |
page:10 HeideggerWritings:56.0 TRUTHBOOK1:60.0

What Aristotle means by soul!

```
defaultnotebook:10.7
```

PLATE 2.25. seqformtwo used to change for formating of sequences displayed in the text.

2.3.1.3.6.1. Including keywords in the text with seqintext.

The binary command seqintext can be used to include sequence keywords in the note of the text, uncluttering the keywords while at the same time allowing for clearer presentation of especially pertinent information. It can even be used to include the equivalent of a title in each note, show at the the top of the text, simply by using title@TITLE as a keyword.

2.3.1.3.6.1.1. Order of keywords in the text.

The sequence keywords are included in the text in the following order

1) The main sequences, which can be user defined, but default to: title, author, date, datefrom, dateto, book, page, chapter, section.

2) Projects.

3) Other sequences

2.3.1.3.6.1.2. Adjusting formating of sequences in text, and changing the main sequences.

ARCADES provides two commands that can be used to adjust the formating of sequences when displayed within the text of the note: seqformone, which is used to adjust the character that appears after the first two sections, and seqformtwo, which determines the character that separates off the sequence keywords from the rest of the notetext. Both of these accept one value, - the breaking characters -- and include a variety of options. If entered without a value, they will call up a self-explanatory menu.

The command mainsequences can be used to change the main sequences. It accepts a single value: a list of sequences separated by commas. It is possible to restore the defaults by entering 'd'.

2.3.1.4. Metadata

Every note includes metadata: the size of the note expressed as a single integer indicating the width of the note, the name of the user who entered the note, and a list containing the timestamp of the creation of the note as well as every subsequent revision.

```
defaultnotebook:4.3   showmeta:12.1
<showmeta:12.1>
/C/Metadata for note #12.1
SIZE : 60
USER : USER
DATE : 2018-10-02 12:14:35.324381
<<>>
defaultnotebook:4.2 #
```

PLATE 2.26: Showing the metadata of a note.

2.3.1.4.1. Metadata commands

ARCADES offers the following commands for changing and viewing metadata.

2.3.1.4.1.1. resize

The command resize (alternate forms: size, rs) can be used to change the default size of note. It accepts a singer integer value, which corresponds to the width in characters of the note.

2.3.1.4.1.2. changeuser

The command changeuser changes the active user as listed in the metadata of the note.

2.3.1.4.1.3. showmeta

The command showmeta shows the metadata for the note at the index given as its value.

2.3.1.4.1.4. showuser

The command showuser displays the default name of the user which is included in the metadata when the note was entered.

2.3.2. ENTERING A NOTE

ARCADES allows rapid entry of new notes with a minimum number of keystrokes. It is possible, indeed, to initiate and terminate note entry only using the RETURN key. This comes at the cost of some degree of flexibility; rather than offering a full-fledged text editor, *ARCADES* uses a simplified note inputter, which forbids returning to previous lines and correcting them, or moving the cursor around beyond what is possible through the standard buffered text input function.

2.3.2.1. Basic note entry

The basic mode of note entry is through the command prompt.

2.3.2.1.1. Initiating note entry

In the regular entry mode, note entry is initiated either by entering a command or by listing a list of keywords.

2.3.2.1.1.1. Commands for initiating note entry

The following commands initiate note entry:

ent, enter, +	To enter a regular note in the next available position.
enternext, ++	To enter the sibling of the last note.
ent, enter, + /$	To enter a sibling of the last note
ent, enter, + /&	To enter a child of the last note
enterchild, +++	To enter the child of the last note
enterback, -	To enter a sibling of the parent of the last note.

If you wish to suppress the inclusion of the general default keys, and the default keys for active projects, you can add a vertical line (|) before the command.

2.3.2.1.1.1.1 Syntax of note entry

If the command is entered alone, then *ARCADES* will initiate the note inputter. It will *not* solicit an index value, but rather will automatically assign the index.

```
ENTRYDEMO:41   +
<+>
<<>>
Keys? Lupu
#######_____________________________________________________________|
PR 1   Lupu is an absolute amazing, perfectly green frog.
PR 2   ||
```

```
There are 1 misspelled words!
LUPU
```

```
Press SPACE+RETURN to skip corrections
```

```
42  | |2019-04-24  | Lupu |

Lupu is an absolute amazing, perfectly green
frog.
```

```
    ATTENTION
Note added at 42
```

```
<<>>
ENTRYDEMO:42
```

PLATE 2.27: Entering a note.

If an index is entered as the first value, *ARCADES* will initiate the note inputter, adding a note at the first available index, or the first available child or sibling of the index that had been entered. If a key or list of keys is entered in the first value, *ARCADES* will add an empty note—a note with an empty text-string---together with theses keys.

If a key or list of keys is entered in the first value, and a second value is also added, then *ARCADES* will add new note with the given keys and text. Use a vertical line (|) --- an EOL --- to mark the end of the line.

Finally: if an index is entered in the first value, key(s) in the second value, and text in the third, *ARCADES* will add a new note with the given keys and text at the index, or first available index position, as per the rules above.

2.3.2.1.1.1.1.1 Rules for finding the first available index

The rules governing the determination of the first available index are a bit tricky, since, in the case of a child note, *ARCADES* searches for the first available sibling of the kind of note requested, rather than the first available new child.

If, for example, you were to have notes at indexes 13, 13.1, 13.1.1, 13.1.1.1, 13.1.1.1.1, 13.1.1.1.1.1 and then add a child note at 13.1 with the command +++:13.1, the new note will appear at 13.1.2 – the first available sibling of the child of 13.1. If, however, you then add a sibling note at 13.1.1, the new note will appear at 13.1.3 – the first available sibling note of 13.1.1, since 13.1.2 is already occupied.

2.3.2.1.1.1.1.1.1 General modifiers for the enter commands

All of the entering commands permit the following 2 modifiers: /* and /+.

The first of these, /*, is used to repress the show function.

The second of these, /?, is used to suppress the default keys.

2.3.2.1.1.1.2. Initiating entry with a list of keywords

It is also possible to initiate entry of a note simply by listing more than one keyword, separated by a comma.

2.3.2.1.1.2. Keyword querying

If keywords have not been entered, *ARCADES* will query the user for keywords to add, either before, or after, or both. To activate or deactivate querying either before or after text input, use the binary commands keysbefore and keysafter. It is also possible, in this way, to suspend keyword querying altogether.

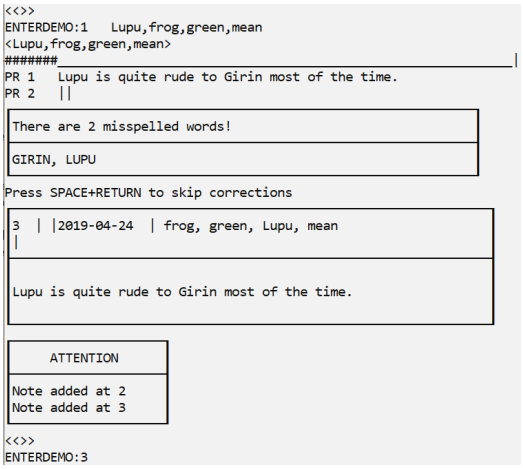


PLATE 2.28: Initiating note entry with a list of keywords.

2.3.2.1.2. Entering the note text

When note entry has been initiated, *ARCADES* will display the list of commands for the inputter and the formatting codes. If you do not wish to see the list commands and/or formatting codes each time, you can use the binary enterhelp and formathelp commands. If the querying for keywords before text entry has not yet been suspended, then it will ask for keywords. After entering a list of keywords, and pressing return, you will see seven POUNDS followed by a long line and terminating in a short vertical line. This indicates the size of the note, and will vary accordingly. Immediately below you will see PR 1, followed by a blinking cursor. "PR" indicates that you are in the prose-mode, which is the initial mode on activating the inputter, while the 1 indicates the line of text.

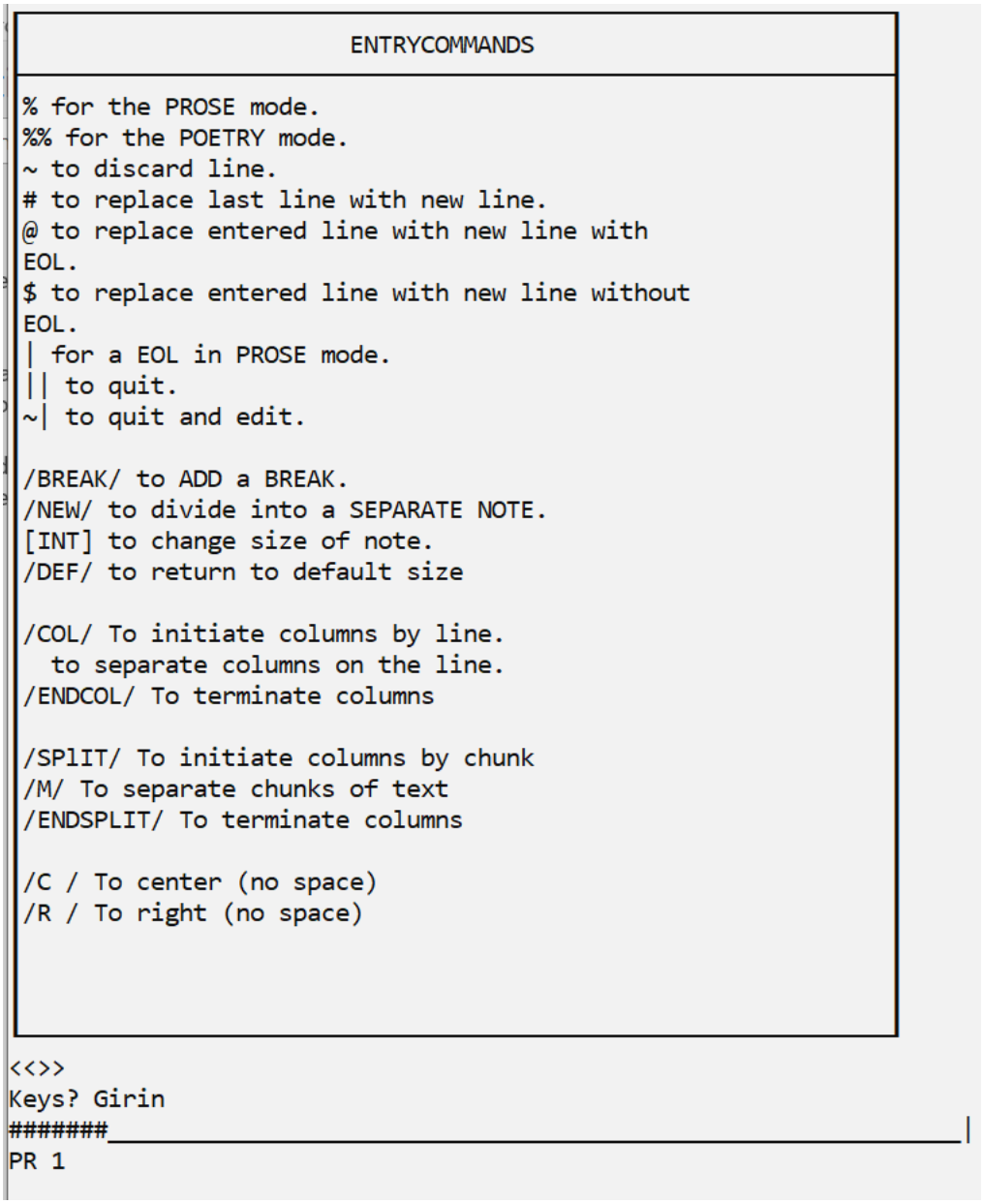


PLATE 2.29: Entering and formatting help

```
<<>>
Keys? Girin
#######_____________________________________________________|
PR 1   Girin is an awesome giraffe.
PR 2   He is quite cute, and has pretty ears.
PR 3   And he is also always calm and gentle,
PR 4   unless Lupu makes him very angry.
PR 5
```

PLATE 2.30: Basic note entry. The line number increases by one each time return is pressed.

2.3.2.1.2.1 Input modes

ARCADES features two text input modes: prose, and poetry. It is possible to switch back and forth between these by inputting PERCENTAGE (%) + RETURN, to enter the prose mode, or the PERCENTAGE + PERCENTAGE + RETURN, to enter the poetry mode.

```
<<>>
Keys?
#######_____________________________________________________|
PR 1   %%
PO 2   %
PR 2   %%
PO 3   %
PR 3   %%
PO 4   %
PR 4   %%
PO 5   %
PR 5
```

PLATE 2.31: Switching between prose and poetry modes.

2.3.2.1.2.1.1 The prose mode

The prose mode does not automatically insert an EOL character with each new line. It is still possible to insert a manual EOL character using one vertical stroke(|), or by inserting one or more spaces at the beginning of the line of text.

2.3.2.1.2.1.2. The poetry mode

The poetry mode automatically inserts an EOL character at the end of each line, thus preserving the integrity of the verse. Whereas in the prose mode, the size of the note is arbitrary, and determined in advanced, the poetry mode sizes the note based on the longest line of text. It is possible to combine poetry and prose in the same note, but, so long as prose is selected once, then the size of the note will be determined "poetically" rather than "prosaically."

2.3.2.1.2.2. Inclusions

2.3.2.1.2.2.1. Including keywords in the text

If words or phrases included in the text are enclosed in curly brackets, then *ARCADES* will give you the option of adding them to the keywords associated with the note.

2.3.2.1.2.2.2. Including text files

Text files can be inserted into the body of note by entering the name of the file in double curly brackets with the prefix AT (@) or STAR (*). Depending on whether the prefix is an AT or STAR,

ARCADES will seek the file in the folder "/textfiles" or "/attachments."

For example:

{{@textfile}}

{{*attachment}}

When the note is displayed, the text from the file will automatically be included.

To enable or disable the automatic display of the attached text, use the binary command showtext.

2.3.2.1.2.2.3. Including a JPEG image.

A jpeg (.jpg) file can be included in the body of the note by surrounding it with double curly brackets and using a CARET (^) as a prefix:

{{^picture}}

When the note is displayed, the jpeg file will be opened. It must be closed manually.

To enable or disable the automatic display of images, use the binary command showimages.

2.3.2.1.2.3. Terminating the note.

Two VERTICAL STROKES (|) placed at the end of the line will terminate the text input. It is also possible to terminate text input by pressing return a certain number of times in succession. Use the binary returnquit command to activate or deactivate this option, and use the setreturnquit command to set the number of returns needed to automatically quit.

By combining automatic initiation of note entry with automatic termination of the text inputter, it is possible to enter a note using only the return key.

2.3.2.1.2.4. Spelling

ARCADES includes a basic spell-checker, explained in section 2.16. If there are misspelled words, the spelling corrector will be activated automatically. To deactivate or activate the spelling feature, use the binary spelling command.

2.3.2.1.2.5. Cutting and pasting text into the text inputter

It is possible to quickly enter text by copying it from another source, and pasting it in to the text inputter at the curser prompt. If you wish to paste a poem, make sure to switch to the poetry mode; otherwise the lines will run into each other.

2.3.2.1.2.6. Advanced keyword entry modes.

ARCADES also offers two advanced entry modes, which make it possible to avoid the separate keyword entry, and entering the entire note as a single text, with all the keywords and sequences extracted from this.

```
ENTRYDEMO:110   +
<+>
<<>>
Keys? Pobble,toes,poem
#######_____________________________________|
PR 1   %%
PO 2   The Pobble who has no toes
PO 3         Had once as many as we;
PO 4   When they said, 'Some day you may lose them all;'--
PO 5         He replied, -- 'Fish fiddle de-dee!'
PO 6   And his Aunt Jobiska made him drink,
PO 7   Lavender water tinged with pink,
PO 8   For she said, 'The World in general knows
PO 9   There's nothing so good for a Pobble's toes!'
PO 10
PO 11  II
PO 12  The Pobble who has no toes,
PO 13        Swam across the Bristol Channel;
PO 14  But before he set out he wrapped his nose,
PO 15        In a piece of scarlet flannel.
PO 16  For his Aunt Jobiska said, 'No harm
PO 17  'Can come to his toes if his nose is warm;
PO 18  'And it's perfectly known that a Pobble's toes
PO 19  'Are safe, -- provided he minds his nose.'||
```

```
111  | |2019-04-25  | Pobble, poem, toes |
```

```
The Pobble who has no toes
      Had once as many as we;
When they said, 'Some day you may lose them all;'--
      He replied, -- 'Fish fiddle de-dee!'
And his Aunt Jobiska made him drink,
Lavender water tinged with pink,
For she said, 'The World in general knows
There's nothing so good for a Pobble's toes!'

II
The Pobble who has no toes,
      Swam across the Bristol Channel;
But before he set out he wrapped his nose,
      In a piece of scarlet flannel.
For his Aunt Jobiska said, 'No harm
'Can come to his toes if his nose is warm;
'And it's perfectly known that a Pobble's toes
'Are safe, -- provided he minds his nose.'
```

PLATE 2.32: Entering a poem in the poetry mode.

```
<<>>
ENTRYDEMO:13.1.1   +
<+>
<<>>
Keys? Baudelaire/poet,The Double Chamber/poem
#######_____________________________________|
PR 1   There the soul bathes itself in indolence made odorous with regret and desire. Ther
e is some sense of the twilight, of things tinged with blue and rose: a dream of delight d
uring an eclipse. The shape of the furniture is elongated, low, languishing; one would thi
nk it endowed with the somnambulistic vitality of plants and minerals.||

117  | |2019-04-25  | Baudelaire/poet,
The Double Chamber/poem |
-------------------------------------------

There the soul bathes itself
in indolence made odorous with
regret and desire. There is some
sense of the twilight, of things
tinged with blue and rose: a
dream of delight during an eclipse.
The shape of the furniture is
elongated, low, languishing;
one would think it endowed with
the somnambulistic vitality of
plants and minerals.

      ATTENTION
-------------------
Note added at 117
```

PLATE 2.33: The prose mode.

```
<<>>
ENTRYDEMO:117   resize:60
<resize:60>

                          SIZE
-------------------------------------------------------------
60

<<>>
ENTRYDEMO:117   +
<+>
<<>>
Keys? Baudelaire,The Double Chamber
#######_____________________________________________________|PR 1   There the soul
bathes itself in indolence made odorous with regret and desire. There is some sense of the
 twilight, of things tinged with blue and rose: a dream of delight during an eclipse. The
shape of the furniture is elongated, low, languishing; one would think it endowed with the
 somnambulistic vitality of plants and minerals.||

118  | |2019-04-25  | The Double Chamber, Baudelaire
|
-------------------------------------------

There the soul bathes itself in indolence made
odorous with regret and desire. There is some
sense of the twilight, of things tinged with
blue and rose: a dream of delight during an
eclipse. The shape of the furniture is elongated,
low, languishing; one would think it endowed
with the somnambulistic vitality of plants
and minerals.

      ATTENTION
-------------------
Note added at 118

<<>>
ENTRYDEMO:118
```

PLATE 2.34: Changing the size of a note.

2.3.2.1.2.6.1. The fromtext mode.

The fromtext command activates the fromtext mode, which automatically suspects the separate keyword prompt. The fromtext mode applies rules to parse the entered text, following the following rules.

(1) The text is divided into segments, each of which is surrounded by LMARK and RMARK.

(2) Each segment consists in an IDENTIFIER and a sequence of VALUES.

(3) The IDENTIFIER is separated from the VALUES by MARK2.

(4) Each value in the list of VALUES is separated by MARK2.

(5) The followed reserved IDENTIFIERS are used:

text | Used to indicate the main text of a note

keywords | Used for non-sequence keywords, which may include tags and knowledge.

(6) All other VALUES will be interpreted as sequences. But if date is included in its name, then it will define a date sequence, and, likewise, is index is in its name, an index sequence.

LMARK, RMARK, MARK1, MARK2 are preset to EOL, EOL, COLIN, and COMMA. It is possible, however, to define new presets. When EOL is used as a preset for LMARK/RMARK, then it is recommended that you enter the text in the poetry mode.

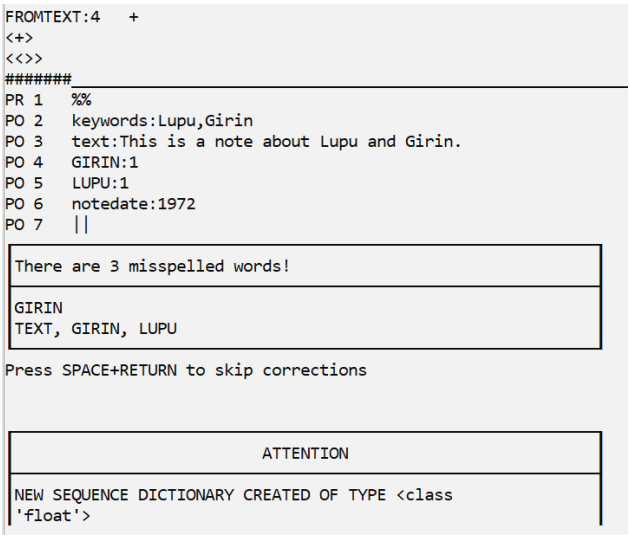


PLATE 2.35 Note entry in the fromtext mode.

```
FROMTEXT:5   5
<5>
┌──────────────────────────────────────────────────────────────────────┐
│5  | |2019-06-26  | LUPU@1, GIRIN@1, Lupu,                            │
│notedate@#1972, Girin |                                               │
├──────────────────────────────────────────────────────────────────────┤
│                                                                      │
│This is a note about Lupu and Girin.                                  │
│                                                                      │
└──────────────────────────────────────────────────────────────────────┘
FROMTEXT:5
```

PLATE 2.36 The result of fromtext entry.

2.3.2.1.2.6.1.1. Changing parsing presets

To create a new set of parsing presets, use newconvertmode. This will ask you for the name of the new mode. To define the presets, use convertdefinitions. To switch to an existing set of presets, use switchconvertmode. To show all available presets, use showallconvertmodes. When entering the divisor, you must add an UNDERLINE between the LMARK and the RMARK if the legnth of either term is greater than one character.

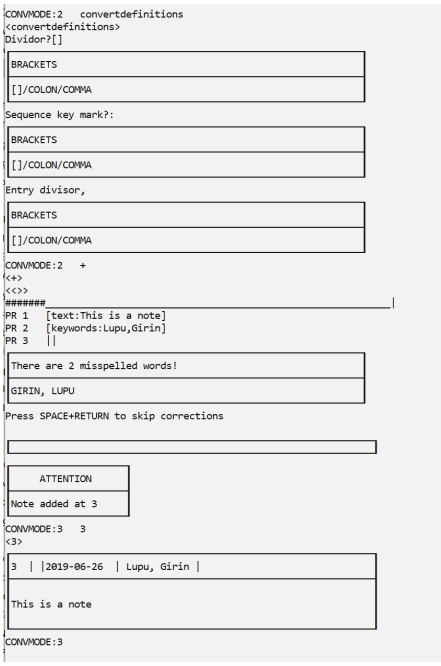


PLATE 2.37. Changing a convert definition.

2.3.2.1.2.6.2. convertbyline

The command convertbyline offers a simpler alternative, a hybrid between regular entry and converting entry. It extracts keywords and sequence keywords, following above the rules, during line entry, and hence can be combined with the regular mode of keyword entry.

2.3.2.1.2.6.2.1. Automatic activation of a convert mode during preset.

To activate a convert mode and apply it, simple include //MODENAME// in the body of the text, preferably at the head.

2.3.2.1.3, Editing a note

ARCADES does not presently include an integrated text inputter and editor, apart from the workpad module. Yet it is possible to edit, subsequent to entry, either the keys of a note, the text of the note, or both.

The commands for initiating editing are: editnote, editnotekeys, editnotetext. These are followed by an index or a collection of indexes. The command editnote accepts a single modifier /$, which is used to activate the "annotate" mode. Whereas editnote edits both the keywords and the text of a note, editnotekeys or editnotetext can be used to edit only the keys or only the text.

2.3.2.1.3.1. Editing the text of a note

When the text editing function is initiated, either by editnotetext or editnote, then *ARCADES* shows the instructions for editing the note, and displays "$1" followed by the first line of text in the original note, a vertical slash (|), and the blinking cursor.

The basic process of editing proceeds by pressing RETURN to skip over a line if you want to keep it as it was, retyping new text, or entering a single DASH to delete an entire line. If you enter a double DASH followed by RETURN, then the rest of the note, beginning with the given line, will be deleted.

(1) RETURN : keeps the current line

(2) *TEXT* : replaces the current line with *TEXT*.

(3) DASH : deletes the current line

(4) DASH + DASH: deletes all subsequent lines

It is also possible to add new lines into the note by using the following functions.

(1) PLUS + RETURN : inserts new lines prior to the current line.

(2) CARET +RETURN : replaces the current line with a series of new lines.

(3) DOLLAR + *TEXT* : adds text before the current line

(4) POUND + *TEXT* : adds text after the current line.

The PLUS and CARET commands call up the insert inputter, displaying "+1" at first, and incrementing the value for each subsequent line. To exit the insert editor, simply enter RETURN.

Note that pressing RETURN repeatedly, after initiating editing, will leave the note unchanged.

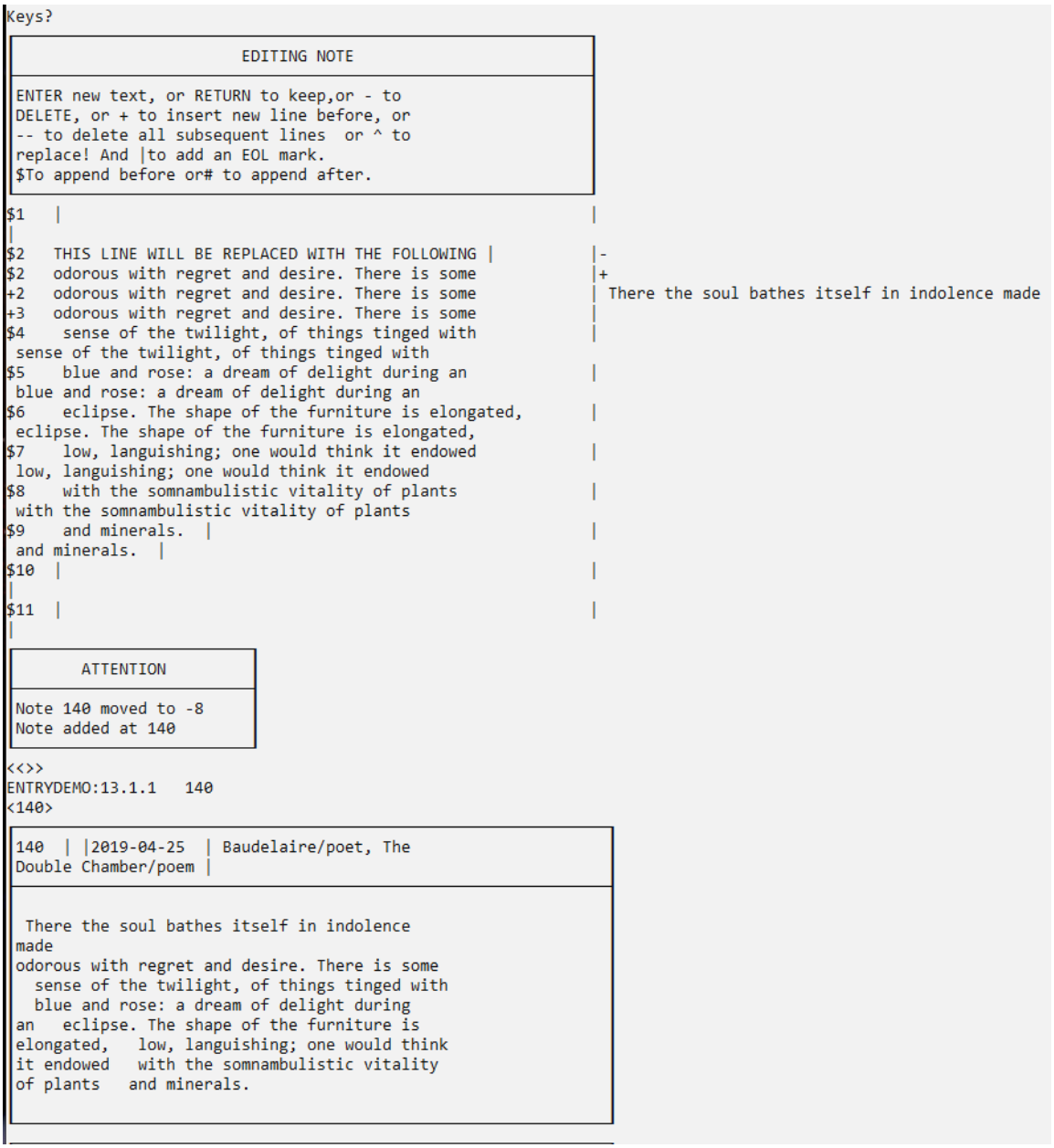


PLATE 2.38: Editing a note

2.3.2.1.3.1.1. editdelete

The binary command editdelete prevents *ARCADES* from keeping a copy of the deleted note.

2.3.2.1.3.2. The “annotate” mode

The “annotate” mode can be used to add an annotation on the side of a note. The “annotate” mode is activated by using the modifier /$ with editnote, and then placing an underline mark (_) before the comment to be added to the side.

```
Keys?
┌──────────────────────────────────────────────────────────────────┐
│                          EDITING NOTE                            │
├──────────────────────────────────────────────────────────────────┤
│ENTER new text, or RETURN to keep,or - to                         │
│DELETE, or + to insert new line before, or                        │
│-- to delete all subsequent lines  or ^ to                        │
│replace! And |to add an EOL mark.                                 │
│$To append before or# to append after.                            │
└──────────────────────────────────────────────────────────────────┘
$1   |                                                    |$HERE IT STARTS
$2   |                                                    |
|
$3    There the soul bathes itself in indolence made |    |
 There the soul bathes itself in indolence made |
$4   odorous with regret and desire. There is some       |
odorous with regret and desire. There is some
$5     sense of the twilight, of things tinged with      |
  sense of the twilight, of things tinged with
$6     blue and rose: a dream of delight during          |
  blue and rose: a dream of delight during
$7   an   eclipse. The shape of the furniture is         |
an   eclipse. The shape of the furniture is
$8   elongated,   low, languishing; one would think      |
elongated,   low, languishing; one would think
$9   it endowed   with the somnambulistic vitality       |
it endowed   with the somnambulistic vitality
$10  of plants   and minerals.   |                       |#HERE IT ENDS
$11  |                                                    |
|
$12  |                                                    |
|
$13  |                                                    |
|
┌──────────────────┐
│     ATTENTION    │
├──────────────────┤
│Note 140 moved to -9
│Note added at 140 │
└──────────────────┘
<<>>
ENTRYDEMO:13.1.1   140
<140>
┌──────────────────────────────────────────────────────────────────┐
│140  | |2019-04-25  | Baudelaire/poet, The                        │
│Double Chamber/poem |                                            │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│HERE IT STARTS                                                    │
│                                                                  │
│ There the soul bathes itself in indolence                        │
│made                                                              │
│odorous with regret and desire. There is some                     │
│   sense of the twilight, of things tinged                        │
│with    blue and rose: a dream of delight during                  │
│ an   eclipse. The shape of the furniture is                      │
│ elongated,   low, languishing; one would think                   │
│ it endowed   with the somnambulistic vitality                    │
│ of plants   and minerals.                                        │
│HERE IT ENDS                                                      │
```

PLATE 2.39: Appending text before and after while using the editor.

```
┌──────────────────────┐
│         Keys         │
├──────────────────────┤
│0 : The Double Chamber│
│1 : Baudelaire        │
└──────────────────────┘
Numbers of autokeys to keep,to delete (start list with $)or ALL to delete all autokeys ALL
┌──────────────────────────────────────────────────────┐
│                      Old Keys                        │
└──────────────────────────────────────────────────────┘
Keys? Double Chamber Parody,pirate version,Bro-delaire
┌──────────────────────────────────────────────────────┐
│                    EDITING NOTE                      │
├──────────────────────────────────────────────────────┤
│ENTER new text, or RETURN to keep,or - to             │
│DELETE, or + to insert new line before, or            │
│-- to delete all subsequent lines  or ^ to            │
│replace! And |to add an EOL mark.                     │
│$To append before or# to append after.                │
└──────────────────────────────────────────────────────┘
$1   There the soul bathes itself         |Thar the soul blows itself
$2   in indolence made odorous with       |in doledrums made undulous with
$3   regret and desire. There is some     |egrets and dis-ire. There is a sum
$4   sense of the twilight, of things     |of scents of Twilight, of flings
$5   tinged with blue and rose: a         |tinged with blue-rays: a scream
$6   dream of delight during an eclipse.  |of delight driving off the cliffs.
$7   The shape of the furniture is        |-
$7   elongated, low, languishing;         |-
$7   one would think it endowed with      |-
$7   the somnambulistic vitality of       |-
$7   plants and minerals. |               |-
$7   |                                    |
|
┌──────────────────────┐
│       ATTENTION      │
├──────────────────────┤
│Note 116 moved to -4  │
│Note added at 116     │
└──────────────────────┘
<<>>
ENTRYDEMO:13.1.1   116
<116>
┌──────────────────────────────────────────────────────┐
│116  | |2019-04-25  | Double                          │
│Chamber Parody, Bro-delaire,                          │
│pirate version |                                      │
├──────────────────────────────────────────────────────┤
│                                                      │
│Thar the soul blows itself in                         │
│doledrums made undulous with                          │
│ egrets and dis-ire. There is                         │
│a sum of scents of Twilight,                          │
│of flings tinged with blue-rays:                      │
│a scream   of delight driving                         │
│off the cliffs.                                       │
│                                                      │
└──────────────────────────────────────────────────────┘
```

PLATE 2.40: Deleting and changing lines with the note editor.

```
                        EDITING NOTE

ENTER new text, or RETURN to keep,or - to
DELETE, or + to insert new line before, or
-- to delete all subsequent lines  or ^ to
replace! And |to add an EOL mark.
$To append before or# to append after.

$1   /COL/
                                    |
'COL/
_|
$2   Im dunkeln Efeu saß ich, an der Pforte| |_Efeu=ivy
Im dunkeln Efeu saß ich, an der Pforte_Efeu=ivy|
$3   Des Waldes, eben, da der goldene Mittag,||_Mittag=midday
Des Waldes, eben, da der goldene Mittag,_Mittag=midday|
$4   Den Quell besuchend, herunterkam|       |
Den Quell besuchend, herunterkam_|
$5   Von Treppen des Alpengebirgs,|          |_Alpengebirg=alps
Von Treppen des Alpengebirgs,_Alpengebirg=alps|
$6   Das mir die göttlichgebaute,|           |
Das mir die göttlichgebaute,_|
$7   Die Burg der Himmlischen heißt|         |_Burg=castle
Die Burg der Himmlischen heißt_Burg=castle|
$8   Nach alter Meinung, wo aber|            |
Nach alter Meinung, wo aber_|
$9   Geheim noch manches entschieden|        |_Geheim=secretly
Geheim noch manches entschieden_Geheim=secretly|
$10  Zu Menschen gelanget; von da|           |
Zu Menschen gelanget; von da_|
$11  Vernahm ich ohne Vermuten|              |
Vernahm ich ohne Vermuten_|
$12  Ein Schicksal, denn noch kaum|          |_Schicksal=fate
Ein Schicksal, denn noch kaum_Schicksal=fate|
$13  War mir im warmen Schatten|             |
War mir im warmen Schatten_|
$14  Sich manches beredend, die Seele|       |_Seele=soul
Sich manches beredend, die Seele_Seele=soul|
$15  Italia zu geschweift|                   |
Italia zu geschweift_|
$16  Und fernhin an die Küsten Moreas.|      |
Und fernhin an die Küsten Moreas._|
$17  |                                       |
_|
```

PLATE 2.41: Adding annotations to the note with the note editor.

```
      ATTENTION
Note 9 moved to -2
Note added at 9

<<>>
SIDENOTE:9   9
<9>

9  | |2019-05-10  | VOID |

                                                |
Im dunkeln Efeu saß ich, an der Pforte          |Efeu=ivy
Des Waldes, eben, da der goldene Mittag,        |Mittag=midday
Den Quell besuchend, herunterkam                |
Von Treppen des Alpengebirgs,                   |Alpengebirg=alps
Das mir die göttlichgebaute,                    |
Die Burg der Himmlischen heißt                  |Burg=castle
Nach alter Meinung, wo aber                     |
Geheim noch manches entschieden                 |Geheim=secretly
Zu Menschen gelanget; von da                    |
Vernahm ich ohne Vermuten                       |
Ein Schicksal, denn noch kaum                   |Schicksal=fate
War mir im warmen Schatten                      |
Sich manches beredend, die Seele                |Seele=soul
Italia zu geschweift                            |
Und fernhin an die Küsten Moreas.               |
                                                |

<<>>
SIDENOTE:9
```

PLATE 2.42: Display of a note with annotations

2.3.2.2. Modes of note entry

Several different modes of note entry are available through the terminal. The most basic mode, described above, initiates either with an explicit command or with a list of keywords.

The second mode – called the "con-mode," automatically initiates a new note whenever RETURN is entered, if no other command or text proceeds it. This mode is activated by the connext or conchild command, and is deactivated by entering a double semicolon (;;). If you use the connext command, the next index will be a sibling of the current index. For example: 1=>2, 1.1=>1.2, 1.1.5=>1.1.6, The conchild command, in contrast, will produce a child of the current index; for example, 1=>1.1,1.1=>1.1.1,1.7.19=1.7.19.1.

The third mode, the "quick" mode, allows you to directly enter a note in the form of the command, with the list of keywords coming before the COLON and the text of the note coming after.

Neither the "quick" mode nor the "con" mode prevents you from entering regular commands.

2.3.2.3. Undo and redo

ARCADES keeps track of actions involves entering, deleting, and moving notes. To reverse an action that has been performed, use the command undo. To restore a command that has been undone, use redo.

The undo and redo commands are limited to actions that involve entering, moving, and deleting notes.

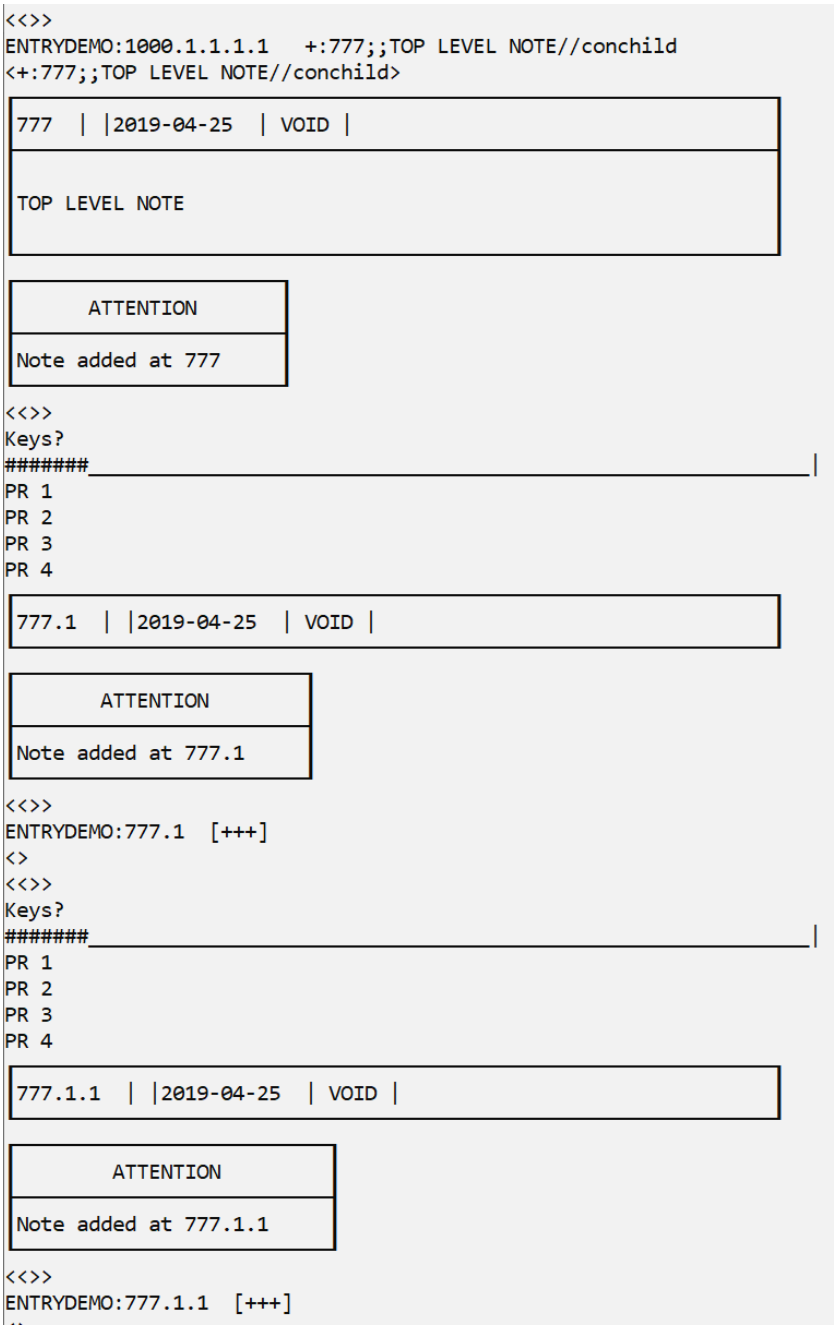


PLATE 2.43: The "conchild" entry mode.

```
ENTRYDEMO:777.1.1.1.1.1.1   Hegel,German Idealism:Hegel wrote the Phenomenology of Spirit.
<Hegel,German Idealism:Hegel wrote the Phenomenology of Spirit.>

┌──────────────────────┐
│      ATTENTION       │
├──────────────────────┤
│Note added at 1003    │
└──────────────────────┘

<<>>
ENTRYDEMO:777.1.1.1.1.1.1   Fichte,German Idealism:Fichte wrote the Commercial State.
<Fichte,German Idealism:Fichte wrote the Commercial State.>

┌──────────────────────┐
│      ATTENTION       │
├──────────────────────┤
│Note added at 1004    │
└──────────────────────┘

<<>>
ENTRYDEMO:777.1.1.1.1.1.1   show:1003-1004
<show:1003-1004>

┌──────────────────────────────────────────────────────────────────────────────┐
│From 0 to 2                                                                   │
├──────────────────────────────────────────────────────────────────────────────┤
│1003                  |   | Hegel, German Idealism         | Hegel wrote the Phenomenology of Spirit.
│1004                  |   | German Idealism, Fichte        | Fichte wrote the Commercial State.
└──────────────────────────────────────────────────────────────────────────────┘
```

PLATE 2.44: The "quick" entry mode.

2.3.3. NOTE AND NOTESCRIPT

The most fundamental principle of *ARCADES* is the convertability of *note* and *notescript*. Every note, or collection of notes, can be exported as a user-readable *notescript*, and these can in turn be imported and converted back into notes. This not only makes it possible to exchange notes with other users, but also makes the process of notetaking independent from the software. It is possible, indeed, by marking up an existing text, to convert it into notes. And it is also possible to record notes using a simple text editor or word processor, or even to automate the generation of notes. Finally, the notescript codes can be included in the text of notes entered through the system, allowing, in effect, for the embedding of one note within another.

2.3.4. THE WORKPAD

The *ARCADES* workpad enables a more visually oriented mode of note display and note entry without departing from the text-based non-GUI operation of *ARCADES*. It was designed around the Python CURSES module, which is not included in the standard Windows distribution of Python. The workpad cannot be used if *ARCADES* is run from *IDLE*.

Notes from the regular notebook can be sent to the workpad, and then placed on the window area, which can be scrolled vertically and horizontally, and manipulated in various ways. In addition, the workpad allows for drawing with Unicode characters, and typing directly on the workpad. Finally, new notes can be composed in the workpad, which are then added to back to the notebook on leaving the workpad mode.

2.3.4.1. Initiating a workpad.

To initiate a workpad, use the command createworkpad. This command accepts, as an optional value, the name of the pad to be initiated. If no padname is given, it will activate the default workpad.

2.3.4.2. Adding notes to the workpad from the notebook.

To add notes to the workpad buffer, use the command addtopad, which accepts two values: the range of indexes of notes to be added, and the name of the pad you wish to use. If no padname is given, the notes will be added to the default pad.

2.3.4.3. Switching to the workpad.

The command padshow or showpad switches to the workpad. The command showpad accepts as a value the name of the pad that you wish to open. [*padshow* is not fully implemented.]

2.3.4.4. Emptying the workpad.

Use the command renewpad, with the name of a workpad as an optional value, to empty out the stack of a workpad.

2.3.4.5. Changing the current workpad.

Use the command switchpad:PADNAME to change the current workpad.

Use the command currentpad to display the current workpad, and allpads to display all the workpads.

2.3.4.6. Using the workpad.

2.3.4.6.1. The workpad display.

The top left of the workpad displays the number of notes in the workpad and in the buffer. and, directly below this, the coordinates of the left and topmost position of the window together with the total size of the workpad, expressed in the following format: Xleftmost/Xtotal : Ytopmost/Ytotal.

Also displayed on the top ribbon is the index of note lying directly under the cursor, and the indexes of the first notes being displayed on the workpad.

The left corner of the bottom ribbon displays the following information: the line drawing mode, the hex value of the foreground and background color, the drawing character, the drawing symmetry form, the speed, and the character display mode.

2.3.4.6.2. Basic operations

The workpad is controlled with keyboard commands. The command menu can be called up by pressing SHIFT F12.

The basic operation of the workpad is determined by the mode which is selected by pressing F3. The mode is displayed at the bottom left side of the workpad, unless the regular mode has been selected. The available modes include:

REGULAR	| Use the arrow keys to move window
TYPING	| Type directly onto the workpad
CURSOR	| Use the arrow keys to move the cursor

EXTENDING	Use the left and up arrows to enlarge the workpad
CONTRACTING	Use the left and up arrows to contract the workpad
MOVING OBJECTS	Use the arrows keys to move one or more selected objects
MOVING OBJECTS MOVING SCREEN	
	Moves objects while also moving the window at the same time
DRAWING	For line drawing

2.3.4.6.3. Placing indexes into the workpad.

To place all notes from the stack into the workpad, press F11. This will call up a menu, which will let you change the mode of the placement (square or peripheral) and the spacing. Pressing F11 a second time will complete the action. [*The peripheral mode is still buggy*]

You can also place notes one by one from the stack at the cursor location by pressing INSERT, or return them to the stack by pressing DELETE.

2.3.4.6.4. Other commands

When not in the drawing mode, press SHIFT F12 to call up the workpad command menu. In the drawing mode, the drawing command menu will be displayed.

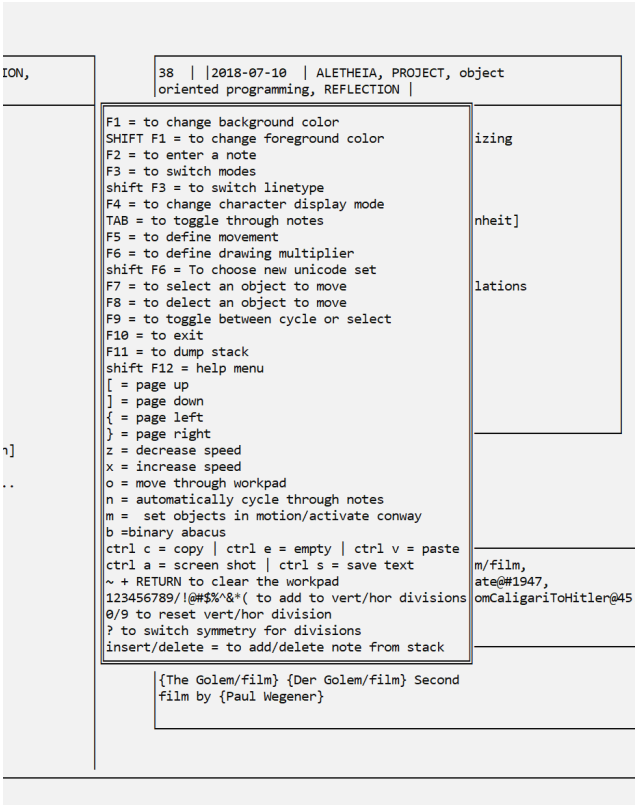


PLATE 2.45 *The* workpad *command menu*

2.3.4.6.5. The *drawing mode*

The drawing mode allows you to draw on the screen using Unicode characters. Several different line types are available, as well as other Unicode characters, and the line character changes to maintain continuity. Other features include a "symmetry" mode, which allows you to draw symmetrically repeating patterns, a fill-mode, and the capacity to select objects, and then stretch, contract, flip, cut, paste, and move them.

PLATE 2.46 An example of a drawing.

```
ARROW KEYS = draw object
FUNC+ARROW = move without drawing
.  contract horizontally
% set on conway mode
TAB next drawing character
shift F11 next drawing palette
* fill out space calling up menu
& fill out space with no menu
BACKSPACE previous drawing character
> stretch horizontally
, contract vertically
< stretch vertically
| flip object on vertical axis
_ flip object on horizontal axis
+ rotate object
= atomize object
```

PLATE 2.47 Commands for the drawing mode.

2.3.4.6.6. Conway's Game of Life

John Conway's game of life can be activated by pressing % in the drawing mode, and then m outside the drawing mode. Various modifications are possible, including introducing degrees of randomness and mutation. The options menu is called up automatically for a few iterations when you use the arrow keys to change settings.

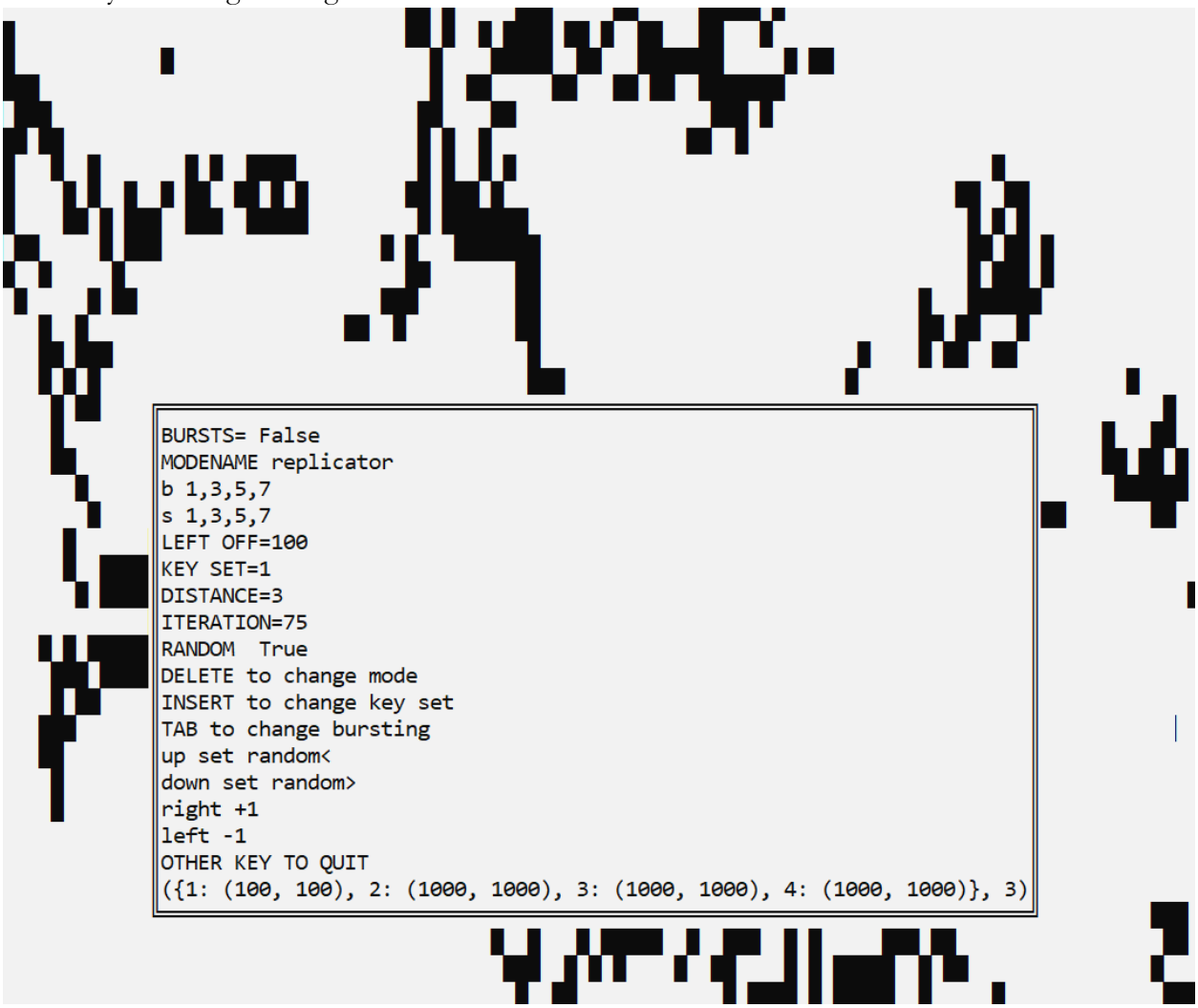


PLATE 2.48 The Conway mode with options menu.

2.3.4.6.7. Creating new notes.

Press F2 to create a new note. The new note will be placed in the topmost corner. Use the arrow keys to adjust the size of the note. The cursor will appear in the top section of the note. Enter keywords, separated by spaces, and then press F1 to begin the main body of the note, and press F1 to conclude the note. A menu of note entry commands appears automatically.

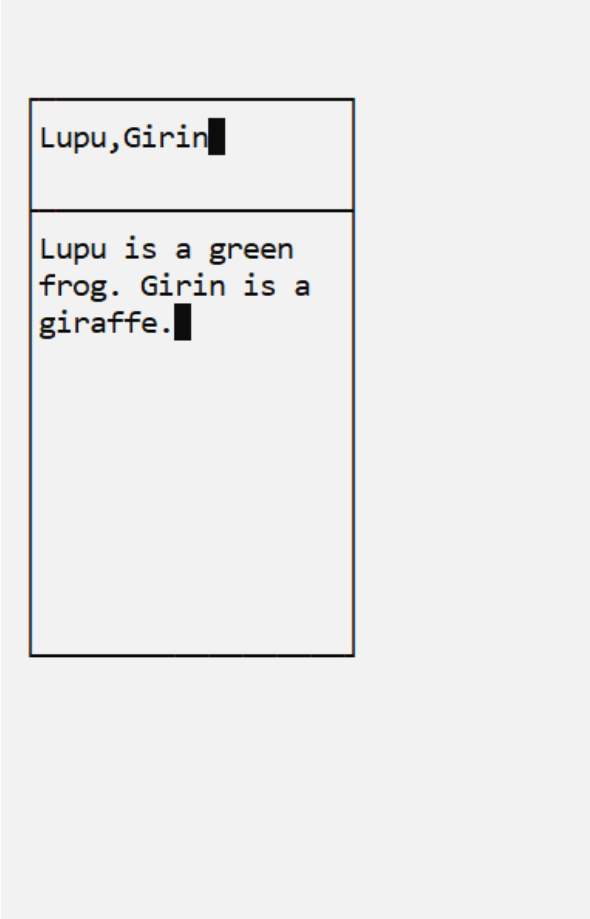


PLATE 2.49 Entering a note

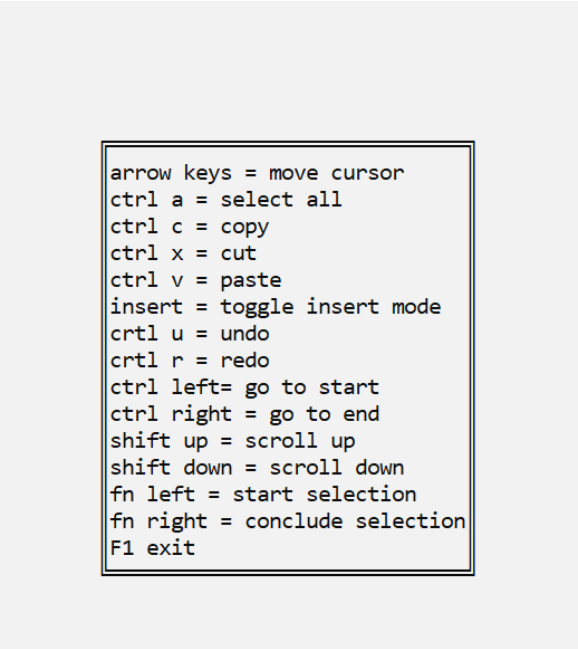


PLATE 2.50 Note entry commands.

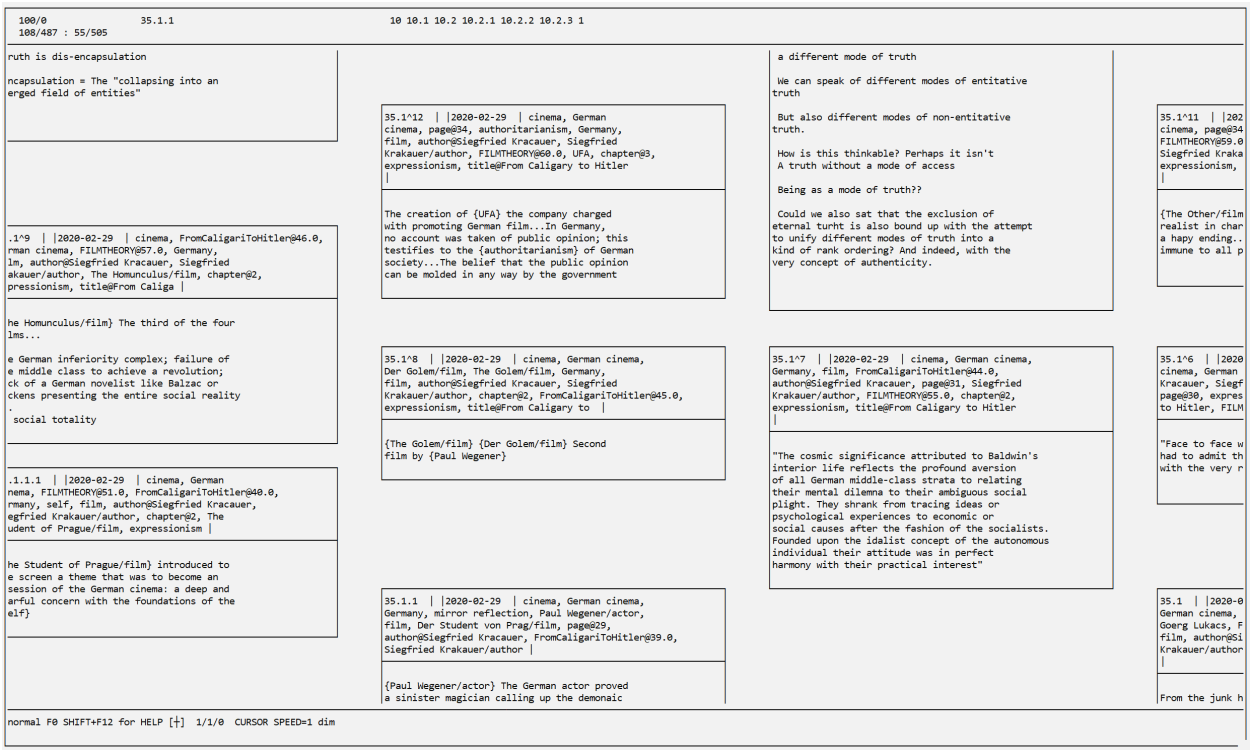


PLATE 2.51 *The* workpad

2.3.4.6.8. The binary abacus.

PLATE 2.52 *The binary abacus.*

Press b to activate the "binary abacus," which offers a relaxing visual display of the multiplication of the binary representations of increasingly large numbers by following simple transformative operations.

2.3.4.7. Saving workpads to the sheetshelf.

The sheetshelf allows you to save workpads and share them between different documents.

2.3.4.7.1. Saving the current workpad to the sheetshelf.

To save the current workpad to the sheetshelf, use to command tosheetshelf, which accepts as values the name of the pad which you want to save as well as the name that it should be saved as in the sheetshelf. The first value defaults to the current pad, and the second to the name of the pad. The name of the notebook is automatically added to the padname in the sheetshelf.

2.3.4.7.2. Selecting a sheet from the sheetshelf.

The command selectsheet selects a sheet from the shelf, and activates it as the current pad. It accepts as a value the name of the pad that this will be opened as, defaulting to "default."

2.3.4.7.3. Closing the sheetshelf.

The command closesheetshelf closes the sheetshelf.

2.3.4.7.4. Updating the sheetshelf when exiting a workpad.

When you exit the workpad, you will be given the option of updating the sheetshelf with the changes that have been made to the workpad.

```
DBsheetshelf:1   selectsheet
<selectsheet>
start
0:defaultnotebookpig
?0
```

NEW PAD CREATED!
default

ALL PADS
default

```
DBsheetshelf:3   showpad
<showpad>
```

ACTIVATING
default

```
Do you want to reclassify uploaded objects?no
Do you want to update sheetshelf?yes
defaultnotebookpig
<bound method Mapping.keys of <shelve.DbfilenameShelf object at 0x03AF0070>>
```


```
DBsheetshelf:4
```

PLATE 2.52.1 Updating the sheetshelf when exiting a workpad.

2.3.3.1. Notescript formatting codes

When *ARCADES* reads a notescript, it extracts all the elements of the text that begin with a left arrow bracket (<) and ends with a right arrow bracket (>).

These extracted elements constitute notephrases. Notephrases are interpreted according to the initial element immediately following the bracket.

If the notephrase begins with DOLLAR ($), PLUS (+), DASH (-), then it modifies the keywords.

Dollar, followed by a list of keys, resets the keywords.

PLUS, followed by a list of keys, adds new keywords.

DASH deletes the last keyword from the list of keywords.

If the notephrase begins with AT+ *INDEX* + AT, a DOUBLE APOSTROPHE (“), a SINGLE APOSTROPHER (‘), a CARET (^), a SEMICOLON (;), or no special symbol, then it adds a note with the given keywords, and with the text following these special marks.

AT+ *INDEX* + AT| Adds *TEXT* at *INDEX*

SINGLE APOSTROPHE + *TEXT* | Adds *TEXT* at a sibling index.

DOUBLE APOSTROPHE + *TEXT* | Adds *TEXT* at a child index.

SEMICOLONE + *TEXT* | Adds *TEXT* at the previous letter.

Finally, a note beginning with STAR (*), adds a complete note, with keywords and text. The keywords and the text are separated by a semicolon.

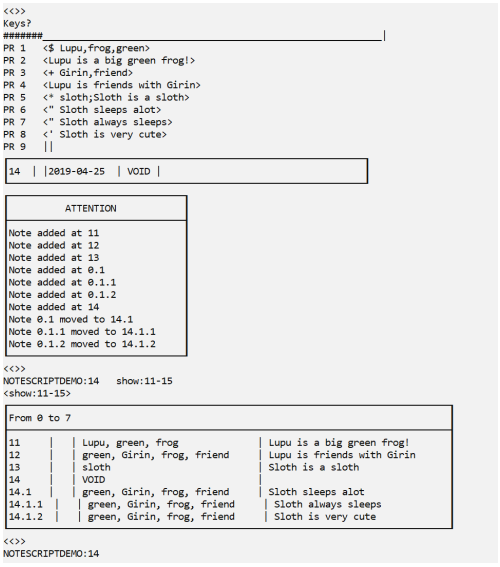


PLATE 2.53: Entering embedded notes.

2.3.3.2. Embedded notes

A *Notescript* can be embedded in the text entered through the text inputter. While a reasonably long notescript can be entered this way, it is not recommended. This feature can also be used to input an impromptu note while composing another note.

2.3.3.3. Reading in notescripts with loadtext

For longer notescripts, it is recommended to use the loadtext command. If no file is entered after the colon, then it will call up the menu.

The command loadtext accepts one value, and three modifies: /$, /&, and /*

The value identifies the name of the file to be loaded, whereas the modifier /$ is used to suppress the inclusion of default keys, /& is used to include a project, and /* to only load the project and not the notes.

```
girinlupu:0.2.3   loadtext
<loadtext>
 ... e note\NOTESCRIPTION/textfiles

1: backup             |.txt
2: betabackup         |.txt
3: girinandlupu       |.txt
4: heideggerfrage     |.txt
5: other text files   |DIR

Enter the number of file to open, or name of new file, or
(B)ack to return to initial directory.3

Are you sure you want to open: girinandlupu? yes

LOADING FILE

girinandlupu

          ATTENTION

Note 0.2.2 moved to 8.2.2
Note 0.2.3 moved to 8.2.3
Note 0.2.4 moved to 8.2.4
Note added at 9
Note added at 10
Note added at 11
Note added at 0.1
Note added at 0.1.1
Note added at 0.1.2
Note 0.1 moved to 0.2.2
Note 0.1.1 moved to 0.2.3
Note 0.1.2 moved to 0.2.4

<<>>
girinlupu:0.2.4
```

PLATE 2.54: Automatically loading and parsing text.

2.3.3.3.1. Outputting notescripts with formout

The command formout generates and saves a notescript.

It accepts four values: the collection of indexes to be converted, the filename, and Boolean values to indicate if the metadata and indexes are to be included. These last values can also be indicated, respectively, with the modifiers /$ and /&. You will also be asked if you wish to include the "projects" in output.

2.4. THE NOTEBOOK

A notebook consists of a collection of notes with unique indexes. It is impossible to assign two different notes to the same index, but beyond this there are no restrictions on the indexes that are permitted. They could, for example, all have values between 1 and 2, or start with 20000. It is also not necessary that any of the notes be of size 0, or that parents exist for children, or older siblings for younger.

*2.4.1 The data structure of the notebook

2.4.2. Basic organizational principles

2.4.2.1. Fields

Fields, explained above (2.3.1.1.1) may be used to structure the notebook as a whole by dividing it up into different regions.

2.4.2.2. Marked notes

ARCADES keeps track of marked notes, and allows the indexes of marked notes to be used as the value of command.

To mark an individual note at the current location, use the command left bracket ([]. To unmark an individual note, use the command right bracket (]).

It is also possible to mark notes as a group by using the command addmarks, unmark notes using deletemarks, clear all the marks with clearmarks, and, finally, marked to display all the indexes of marked notes.

2.5. MOVING THROUGH THE NOTES

When the con-mode is not activated, *ARCADES* will move through the notes whenever you press RETURN. This function, however, is automatically suspended whenever you enter a note, either until you press RETURN, or enter a command, ten times in succession, or enter a backslash (/).

There are two *basic* modes of moving through the notes: *iteration* and *branching*.

Iteration involves moving according to a rule through the indexes contained within the scope of iteration.

Branching involves moving to a note that is related to the current note.

The flashmode allows you to flip through a stack of multi-sided flashcards.

2.5.1. Iteration mode

The behavior of *ARCADES* in the iteration mode depends on two things:

1) The "rule" determining movement. (The iteration-rule)

2) The content of the iterator-sequence. (The iteration-content)

ARCADES defaults to the *iteration* mode, with the iteration-content comprising all positive-valued indexes, regardless of their size, and the iterator-rule consisting in moving to the note whose index is the "closest" index greater than the current index.

```
girinlupu:8.2.2   show:1-20
<show:1-20>
```

From 0 to 16			
2	|	/C	Welcome to NOTESCRIPTION!
3	|	frog, Lupu, green	Lupu is a big green frog!
4	|	friend, Lupu, frog, Girin	Lupu is friends with Girin
5	|	sloth	Sloth is a sloth
5.2.2	|	friend, Lupu, frog, Girin	Sloth sleeps alot
5.2.3	|	friend, Lupu, frog, Girin	Sloth always sleeps
5.2.4	|	friend, Lupu, frog, Girin	Sloth is very cute
6	|	frog, Lupu, green	Lupu is a big green frog!
7	|	friend, Lupu, frog, Girin	Lupu is friends with Girin
8	|	sloth	Sloth is a sloth
8.2.2	|	friend, Lupu, frog, Girin	Sloth sleeps alot
8.2.3	|	friend, Lupu, frog, Girin	Sloth always sleeps
8.2.4	|	friend, Lupu, frog, Girin	Sloth is very cute
9	|	frog, Lupu, green	Lupu is a big green frog!
10	|	friend, Lupu, frog, Girin	Lupu is friends with Girin
11	|	sloth	Sloth is a sloth

```
<<>>
girinlupu:8.2.2
<>
```

8.2.4 | | friend, Lupu, frog, Girin, green |
Sloth is very cute

```
<<>>
girinlupu:8.2.4
<>
```

9 | | frog, Lupu, green |
Lupu is a big green frog!

```
<<>>
girinlupu:9
<>
```

10 | | friend, Lupu, frog, Girin, green |
Lupu is friends with Girin

```
<<>>
girinlupu:10
```

PLATE 2.55: Moving through notes in the iteration mode.

2.5.1.1. The iteration rule

2.5.1.1.1. Regular iteration

In the regular mode of iteration, *ARCADES* advances either backwards or forwards, and by either one index or more than one index. The behavior of the regular mode is thus determined by the *direction* and *depth*. The direction defaults to forward, moving to a greater index value, and the speed defaults to one, advancing to the first closest index. The *tilt* determines whether it advances to parents/children, older or young siblings, or is neutral, and defaults to the neutral mode.The fourth factor determining the behavior of the regular iteration mode is the depth, which defaults to 0.

2.5.1.1.1.1 Changing direction

The commands for changing direction are the left arrow bracket (<) to move backwards, and right arrow bracket (>) to move forwards.

*2.5.1.1.1.2. Changing speed

This feature does not yet exist, but may be added.

2.5.1.1.1.3. Changing depth

If the depth is 0, then the iterator iterates over notes with indexes of any size; if it is greater than 0, then it only iterates over notes whose indexes are less than the depth.

The depth can be set directly with the command depth. It can also be increased or decreased with deeper or shallower.

*2.5.1.1.1.4. Changing tilt

This function needs to be developed.

2.5.1.1.2 Random iteration

In the random mode, *ARCADES* advances randomly (pseudo-randomly, that is) through the indexes in the iterator-sequence until it has passed through all of them. Use the randomon to activate the random mode, and randomoff to deactivate it.

2.5.1.2. The iteration-sequence

The iteration-sequence defaults to all the notes with positive values, but it is also possible to iterate over a subset of the notebook. This is done by redefining the flipbook; whenever the flipbook is redefined, then the iterator-sequence is automatically reset.

2.5.1.2.1. “Flipping out” search results

The binary flipout command activates the flipout-mode. When then flipout-mode is activated, then the results of searches are sent to the flipbook, which is reset accordingly.

2.5.1.2.2. Manually changing the flipbook

The flipbook can also be manually set by using the flipbook command. The flipbook command can accept as a value either a collection of indexes, or a list of fields. If entered as a simple command, without any value, then flipbook resets the flipbook to all the positive indexes in the notebook.

2.5.1.2.3. Refeeding to the flipbook

It is possible to feed the results of other commands to the flipbook, provided that they yield a collection of indexes.

For example:

(1) marked =>flipbook:? | Feeds the marked notes into the flipbook.
| Equivalent to flipbook:[?]

(2) search:VALUE=>flipbook:? | Feeds a search result into the flipbook.
| Equivalent of "flipping out" a search
| Also equivalent to flipbook.

2.5.1.2.4 Descendants

Because *ARCADES* allows for hierarchical structures, you may wish to reset the sequence-iterator not to a group of notes defined by a range of indexes, but to all the descendants of a given index. This can be achieved with the descendants command. When this command is used, it not only resets the flipbook, but it also indicates the value that has been entered after the command prompt. It will now be possible to enter this value, which all the indexes in the flipbook start with, simply by using a single period.

To restore the sequence-iterator to all the indexes in the notebook, enter descendants without a value.

2.5.1.2.5. Advanced functions

There are two additional flipbook functions which will be discussed in the context of advanced features. These include: flipproject (see section 2.9), which sends the indexes in the current project to the sequence-iterator, and the cluster function (see section 2.8.4.)

2.5.1.2.6. Manual operation of the iterator

In the iterator-mode, *ARCADES* always advances to the "next" note according to the direction, speed, and rule, and the content of the iterator sequence itself. It is, however, also possible to navigate manually through the iterator-sequence with the following commands.

(1) period | moves forward one index

(2) comma | moves back one index

(3) period*N | moves forward by N indexes

e.g. ……. | moves forward by 7 indexes

(4) comma*N | moves back N by indexes

e.g. ,,,, | moves back by 4 indexes

(5) first | goes to the first note in the iterator sequence

(6) last | goes to the last note in the iterator sequence

(7) hop:N | moves forward by N indexes

e.g. hop:5 | moves forward by 5 indexes

```
<<>>
CHILDRENDEMO:1   all /$
<all /$>
```

From 0 to 17			
2		/C	Welcome to NOTESCRIPTION!
3		PARENT	
3.1		PARENT, ALL MY CHILDREN	
3.1.1		PARENT, ALL MY CHILDREN	
3.1.1.1		PARENT, ALL MY CHILDREN	
3.1.1.1.1		PARENT, ALL MY CHILDREN	
3.1^5		PARENT, ALL MY CHILDREN	
3.1^6		PARENT, ALL MY CHILDREN	
3.1^7		PARENT, ALL MY CHILDREN	
3.1^8		PARENT, ALL MY CHILDREN	
3.1^9		PARENT, ALL MY CHILDREN	
3.1^10		PARENT, ALL MY CHILDREN	
3.1^11		PARENT, ALL MY CHILDREN	
3.1^12		PARENT, ALL MY CHILDREN	
3.1^13		PARENT, ALL MY CHILDREN	
3.1^14		PARENT, ALL MY CHILDREN	
3.1^15		PARENT, ALL MY CHILDREN	

```
<<>>
CHILDRENDEMO:1   descendents:3.1.1
<descendents:3.1.1>
```

CLUSTERS
Cluster #1 : <<3.1^10>>, <<3.1^11>>, <<3.1^12>>, <<3.1^13>>, <<3.1^14>>, <<3.1^15>>, <<3.1^2>>, <<3.1^3>>, <<3.1^4>>, <<3.1^5>>, <<3.1^6>>, <<3.1^7>>, <<3.1^8>>, <<3.1^9>>

Iterator reset
3.1.1–3.1.1.1.1.1.1.1.1.1.1.1.1.1.1

```
<<>>
CHILDRENDEMO:3 3.1.1  .1.1
<.1.1>
3.1.1.1.1
```

3.1.1.1.1	2019-05-19	PARENT, ALL MY CHILDREN

```
<<>>
CHILDRENDEMO:3.1 3.1.1
```

PLATE 2.56: Jumping down a note with descendents.

```
<<>>
girinlupu:3   randomon
<randomon>
<<>>
girinlupu:3
<>
```

5.2.2 | | friend, Lupu, frog, Girin, green
Sloth sleeps alot

```
<<>>
girinlupu:5.2.2
<>
```

8.2.3 | | friend, Lupu, frog, Girin, green
Sloth always sleeps

```
<<>>
girinlupu:8.2.3   randomoff
<randomoff>
<<>>
girinlupu:8.2.3
<>
```

5.2.2 | | friend, Lupu, frog, Girin, green
Sloth sleeps alot

```
<<>>
girinlupu:5.2.2
<>
```

5.2.3 | | friend, Lupu, frog, Girin, green
Sloth always sleeps

```
<<>>
girinlupu:5.2.3
```

PLATE 2.57: Moving through the notes in the “random” mode.

```
girinlupu:8   flipout
<flipout>
```

FLIPOUT
ON

```
<<>>
girinlupu:5   ?:sloth
<?:sloth>
```

|NONE |include negative results with searches |	

Iterator reset
5–8

RESULT for sloth
5, 8, 11

```
<<>>
girinlupu:11
<>
```

11		sloth	
Sloth is a sloth

```
<<>>
girinlupu:11
<>
```

8		sloth	
Sloth is a sloth

```
<<>>
girinlupu:8
<>
```

5		sloth	
Sloth is a sloth

```
<<>>
girinlupu:5
```

PLATE 2.58: Flipping out a search.

2.5.2. Branching mode

Those of us who grew up with personal computers in the 80s may remember a simple text-based game where you chase a monster through a labyrinth of caves connected to each other through passageways. This game, a simple application of graph-theory, indicates a powerful way of thinking of the organization of a notebook. Up until this point we've been considering a notebook primarily as a collection of notes identified through indexes, where the relations of these notes to one another is entirely a function of the relation of their indexes. As different as the various modes of iteration are from one another, they all share this way of conceiving what a notebook is. The iterator-sequence is a collection of indexes, and moving through the notes means moving through the indexes. But it is also possible to think of notes as relating to other notes through their own content; this content includes every attribute of the note – its keywords, its text, its metadata, and also its index value. From such a perspective, indeed, the iterating mode may itself be seen as a special case of this second, and more general organizational logic. Because each note has *only one index*, and because *all indexes form a well-ordered series*, it becomes possible, by abstracting away from all other attributes of the note, to precisely define a collection of indexes and a rule (even if only the "rule" of randomness) governing the movement through them. But this is in turn limiting: not least of all, it restricts the organizing principle to what has been established, in advanced, by the user.

Once you consider notes as related by their content, it then becomes possible to conceive of having a multitude of different *affinities*. These include affinities which are:

1) based on keywords, including links and other keywords.

2) based on common words in the text.

3) based on the date of composition.

The branching mode implements an organizational logic based on these different kinds of affinities. It allows you, in other words, to move from one note to another note chosen from among a set of the notes to which it has affinities according to a well-defined rule. We might indeed distinguish between *affine notes* and *kindred notes*. Kindred notes are related by their respective indexes– and indeed, as explained above, all notes within a notebook have some kind of kinship between them. Affine notes are related by their respective attributes. Strictly affine notes involve an affinity that is not simple kinship – that doesn't involve just the index value. If notes are strictly affine, then their affinity remains the same even if they are moved to different locations. Hence, *links*, even though expressed through indexes, in fact involve a strict affinity, since the *link* is actually merely a "pointer" to the transposition table.

2.5.2.1. Using the branching mode

To activate or deactivate the branching mode, use the binary iteratemode command.

Use the commands branchone, branchtwo, and branchthree to select between the different branching modes.

2.5.2.1.1. The first branching mode

The first branching mode moves randomly to notes that have related keywords.

2.5.2.1.2. The second branching mode

The second branching mode moves randomly to related links.

2.5.2.1.3. The third branching mode

The third branching mode allows you to move by choice through links.

*2.5.2.1.4. Other possible branching modes

There are many other possible branching modes which might be implemented. These include:

1) Distinguishing between keyword affinities according to keyword frequency, with greater value assigned to less frequent keywords.

2) Identifying affinities in the text.

2.5.3. The flashmode

Unlike the previous two modes, the flashmode only works on a special type of note: a "flashcard" note. The flashmode, which is available only in the iteration mode, allows you to flip over the sides of such flashcard notes.

2.5.4. Flashcards

Flashcards are made by entering in an ordinary note, using /FC/ to separate the sides. If /FC/ appears twice in the note, for example, then the note will have three sides.

For example:

<man /FC/ der Mensch /FC/ l'homme /FC/ il uomo>

| Defines a 4 sided flashcard with the English, German, French, and Italian words for "man"

2.5.5. Entering the flashmode

The binary command flashmode is used to enter or exit the flashmode. In addition to this, you will also need to set the number of sides of the flashcard by using the command setsides, which defaults to two. To adjust when the flashcards advance, and which card they start with, use setflipat.

When the flashmode is activated, *ARCADES* will advance through the sides of each note before moving on to the text note in the iterator.

```
<<>>
FLASHMODE:2
<>
```

3 [1] | | VOID |
man

```
<<>>
FLASHMODE:3
<>
```

3 [2] | | VOID |
der Mensch

```
<<>>
FLASHMODE:3
<>
```

4 [1] | | VOID |
house

```
<<>>
FLASHMODE:4
<>
```

4 [2] | | VOID |
das Haus

```
<<>>
FLASHMODE:4
<>
```

5 [1] | | VOID |
woman

```
<<>>
FLASHMODE:5
<>
```

5 [2] | | VOID |
die Frau

```
<<>>
```

PLATE 2.59: Flashcards

The flashmode operates within the iterator mode, and all the regular features of the iterator mode remain the same as before. The flashmode, indeed, does nothing more than add an additional requirement that must be fulfilled before the iterator advances.

2.5.6. Using flashcards outside the flashmode

The use of the following commands do not require that the flashmode is activated.

(1) noflash | disables the display of flashcards

(2) flashforward, ff | advances to the next side

(3) flashback, fb | goes back to previous side

(4) flashreset, fr | returns to the first side

(5) flashto, ft | goes to a specific side

The numeration of sides begins with 0.

2.5.7 Interaction of iteration and note entry

The interaction between the iterator and note entry is subtle, and may take some getting used to. It is important to keep in mind, first of all, that while the iteration is confined to the notes whose indexes are included in the iterator-sequence, notes from throughout the notebook can be displayed, and accessed, and notes can be added anywhere. Ordinarily, the index position at which a new note will be added is determined by the iterator as it passes through the iterator sequence, but this is not always the case.

2.5.7.1 Displaying a single index

Any index can be entered as a command. If a note with the index is present in the notebook, then this note will be displayed. Doing this, however, will not affect either the position of the iterator, or the position at which new note will be added.

2.5.7.2. The effect of adding notes on the iterator-sequence

If you add new notes to the notebook, these will be automatically added to the iterator-sequence.

2.5.7.3. Skipping

By entering the command skip:*INDEX*, you can skip to the note at *INDEX*. This *INDEX* will the become the position for entering a new note.

*There are some oddities in the behavior of this function which need to be resolved.

2.6. DISPLAYING

ARCADES offers three different modes for displaying the notes in the notebook. The first, called the "sequential" mode, displays individual notes--or kindred groups of notes-- individually as you move through the notebook. The second, the "list" mode, displays multiple notes in a list allowing you to scroll backwards or forwards. The third, the "multi" mode, displays many notes simultaneously as a tiled sheet.

```
<<>>
navigation:3.1   flipbook:1-5
<flipbook:1-5>
```

FLIPBOOK changed to
2–3, 3.1, 3.1.1, 3.1.1.1, 3.1.1.1.1, 3.1.1.1.1.1–4.1.1.1.1.1, 4.1.1.1.1.1.1, 4.1.1.1.1.1.2, 4.1.1.1.1.1.3, 4.1.1.1.1.1.4, 4.1.1.1.1.1.5, 4.1.1.1.1.1.5.1–5

Iterator reset
2–5

```
<<>>
navigation:3.1.1   10001
<10001>
```

10001 | |2019-04-27 | VOID |
One More!

```
<<>>
navigation:3.1.1.1
<>
```

3.1 | | VOID |

3 | |2019-04-27 | VOID |

3.1 | |2019-04-27 | VOID |

3.1.1 | |2019-04-27 | VOID |

3.1.1.1 | |2019-04-27 | VOID |

3.1.1.1.1 | |2019-04-27 | VOID |

3.1.1.1.1.1 | |2019-04-27 | VOID |

```
<<>>
navigation:3.1
```

PLATE 2.60: Temporarily exiting the iterator sequence.

```
                                    INDEXES
1: 2                |2019-04-27   |/C     |Welcome to NOTESCRIPTION!
2: 3                |2019-04-27   |VOID   |
3: 3.1              |2019-04-27   |VOID   |
4: 3.1.1            |2019-04-27   |VOID   |
5: 3.1.1.1          |2019-04-27   |VOID   |
6: 3.1.1.1.1        |2019-04-27   |VOID   |
7: 3.1.1.1.1.1      |2019-04-27   |VOID   |
8: 4                |2019-04-27   |VOID   |
9: 4.1              |2019-04-27   |VOID   |
10: 4.1.1           |2019-04-27   |VOID   |
11: 4.1.1.1         |2019-04-27   |VOID   |
12: 4.1.1.1.1       |2019-04-27   |VOID   |
13: 4.1.1.1.1.1     |2019-04-27   |VOID   |
14: 4.1.1.1.1.1.1   |2019-04-27   |VOID   |
15: 4.1.1.1.1.1.2   |2019-04-27   |VOID   |
16: 4.1.1.1.1.1.3   |2019-04-27   |VOID   |
17: 4.1.1.1.1.1.4   |2019-04-27   |VOID   |
18: 4.1.1.1.1.1.5   |2019-04-27   |VOID   |
19: 5               |2019-04-27   |VOID   |
20: 5.1             |2019-04-27   |VOID   |
21: 5.1.1           |2019-04-27   |VOID   |
22: 5.1.1.1         |2019-04-27   |VOID   |
23: 5.1.1.1.1       |2019-04-27   |VOID   |
<<>>
NAVIGATION:4.1.1.1.1.1   flipbook:1-6
<flipbook:1-6>
FLIPBOOK changed to
2–3, 3.1, 3.1.1, 3.1.1.1, 3.1.1.1.1, 3.1.1.1.1.1–4.1.1.1.1.1,
4.1.1.1.1.1.1, 4.1.1.1.1.1.2, 4.1.1.1.1.1.3,
4.1.1.1.1.1.4, 4.1.1.1.1.1.5, 4.1.1.1.1.1.5.1–5.1.1.1.1
Iterator reset
2–5.1.1.1.1
<<>>
NAVIGATION:4.1.1.1.1.1.1   +:10000;;THIS NOTE WAS OUTISDE THE IRETATION SEQUENCE, BUT WILL BE ADDED TO IT.
<+:10000;;THIS NOTE WAS OUTISDE THE IRETATION SEQUENCE, BUT WILL BE ADDED TO IT.>
10000  | |2019-04-27  | VOID |
THIS NOTE WAS OUTISDE THE IRETATION SEQUENCE,
BUT WILL BE ADDED TO IT.
      ATTENTION
Note added at 10000
<<>>
NAVIGATION:10000
```

PLATE 2.61: Newly entered note added to the iterator sequence.

2.6.1. The sequential display method

As you either progress through the indexes of the notebook, or enter in indexes directly as commands, *ARCADES* displays them, either alone, or together with the cluster of notes that belong to the tribe of the top level note.

The behavior of the sequential display mode is determined with the following commands: childrentoo, fulltop, and automulti.

2.6.1.1. Modifying the sequential display method: childrentoo

In its default "childrentoo" mode, *ARCADES* displays the current note together with the entire tribe of notes belonging to the top-level note with which it is associated. The binary childrentoo command can be used to have *ARCADES* no longer show the tribe together with the note.

*The interaction of depth, deeper, and shallower with the childrentoo command is a bit quirky. When the depth is set at 1, the tribe is repressed, but otherwise the complete tribe is displayed, despite the fact that only numbers of size less than the depth will be iterated over.

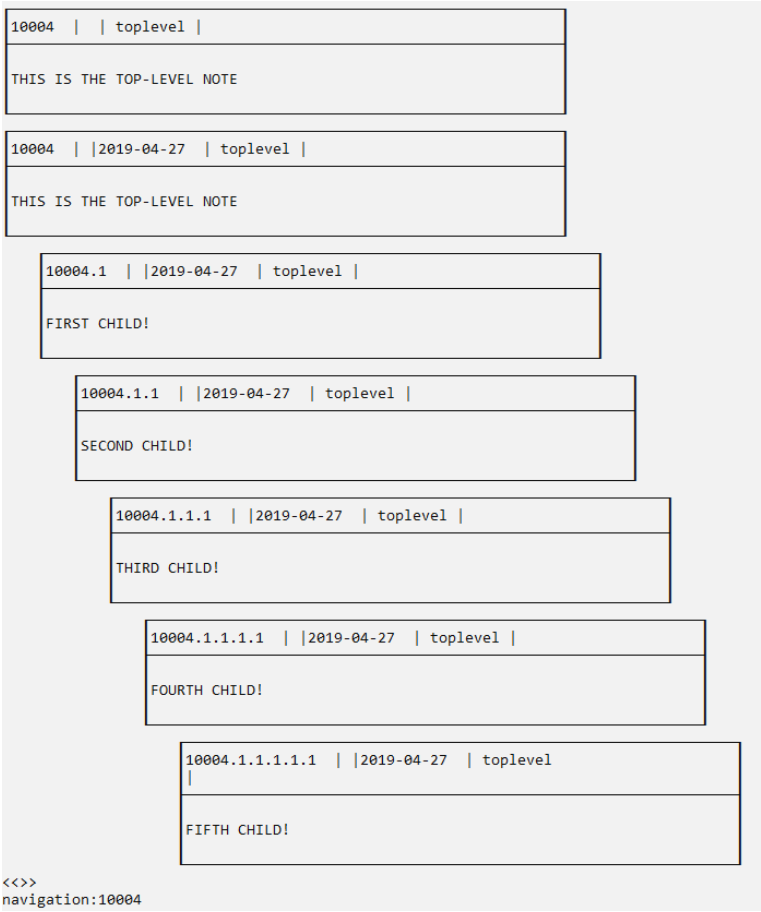


PLATE 2.62: Displaying a note with its children.

```
navigation:10004
<>
        10004.1  |  | toplevel |
        FIRST CHILD!
<<>>
navigation:10004.1
<>
            10004.1.1  |  | toplevel |
            SECOND CHILD!
<<>>
navigation:10004.1.1
<>
                10004.1.1.1  |  | toplevel |
                THIRD CHILD!
<<>>
navigation:10004.1.1.1
<>
                    10004.1.1.1.1  |  | toplevel |
                    FOURTH CHILD!
<<>>
navigation:10004.1.1.1.1
<>
                        10004.1.1.1.1.1  |  | toplevel |
                        FIFTH CHILD!
<<>>
navigation:10004.1.1.1.1.1
```

PLATE 2.63: Displaying the children notes separately.

2.6.1.2. Modifying the sequential display method: fulltop

When the "childrentoo" mode is deactivated, then, if the "fulltop" mode is activated, *ARCADES* will only display the entire tribe for top-level notes. Other notes with be displayed alone.

2.6.1.3. Modifying the sequential display method: automulti

When the "automulti" mode is activated together with the "childrentoo" or "fulltop"-modes, then *Norescription* will display the tribe as a tiled sheet.

10004 | | toplevel |	10004.1 | | toplevel |
THIS IS THE TOP-LEVEL NOTE	FIRST CHILD!
10004.1.1 | | toplevel |	**10004.1.1.1 | | toplevel |**
SECOND CHILD!	THIRD CHILD!
10004.1.1.1.1 | | toplevel |	**10004.1.1.1.1.1 | | toplevel |**
FOURTH CHILD!	FIFTH CHILD!

PLATE 2.64: "Automulti" mode.

2.6.1.4. The representation of hierarchy through the variable left margin

When displayed either individually or as part of a tribe, so long as the "automulti"-mode is suspended, notes are indented in order to indicate the size of their index.

The degree of identation can be adjusted using the command indentmultiplier.

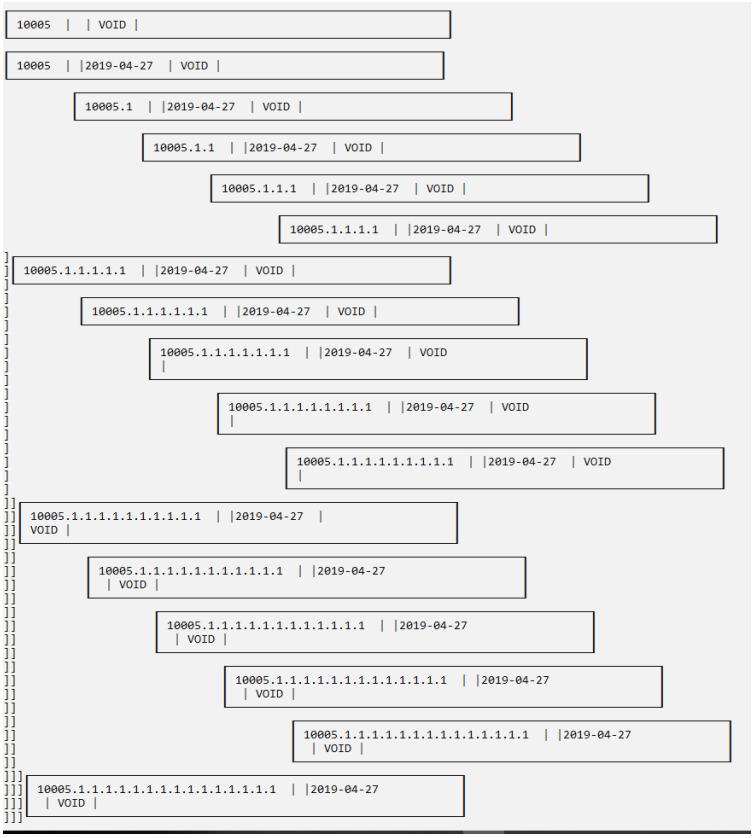


PLATE 2.65: The vertical lines on the left indicate depth by multiples of 5

2.6.2. The notebook display mode

Various commands are available to display a large number of notes either as a list of full-length notes or of notes in the compressed "short"-form.

2.6.2.1. The all command

The all command can be used to display all the notes in the notebook. If your notebook contains many notes, it may take a bit of time to run the first time it is used, but it saves the results to a buffer which may be retrieved subsequently.

The command all, in addition to being restricted to a certain index range, accepts a single value and several modifiers. The optional value indicates the *size* of the indexes to be listed; all indexes must be less than this value. If no value is listed, then notes with indexes are all sizes will be displayed.

Modifiers include: /& /? /=

1) /& Excludes non top-level notes.

2) /* Excludes square characters around notes – allowing format to be outputted in a form that can be copied into other programs.

3) /? Suspends the automatic short mode

4) /= Includes dates with the notes

2.6.2.1.1. Displaying the "all"-buffer

The "display"-buffer, to which the "regular"-mode outputs, can be presented by entering a double DOLLAR ($$).qq

2.6.2.1.2. Displaying the "display"-buffer

The "display"-buffer, to which the "regular"-mode outputs, can be presented by entering a single DOLLAR ($).

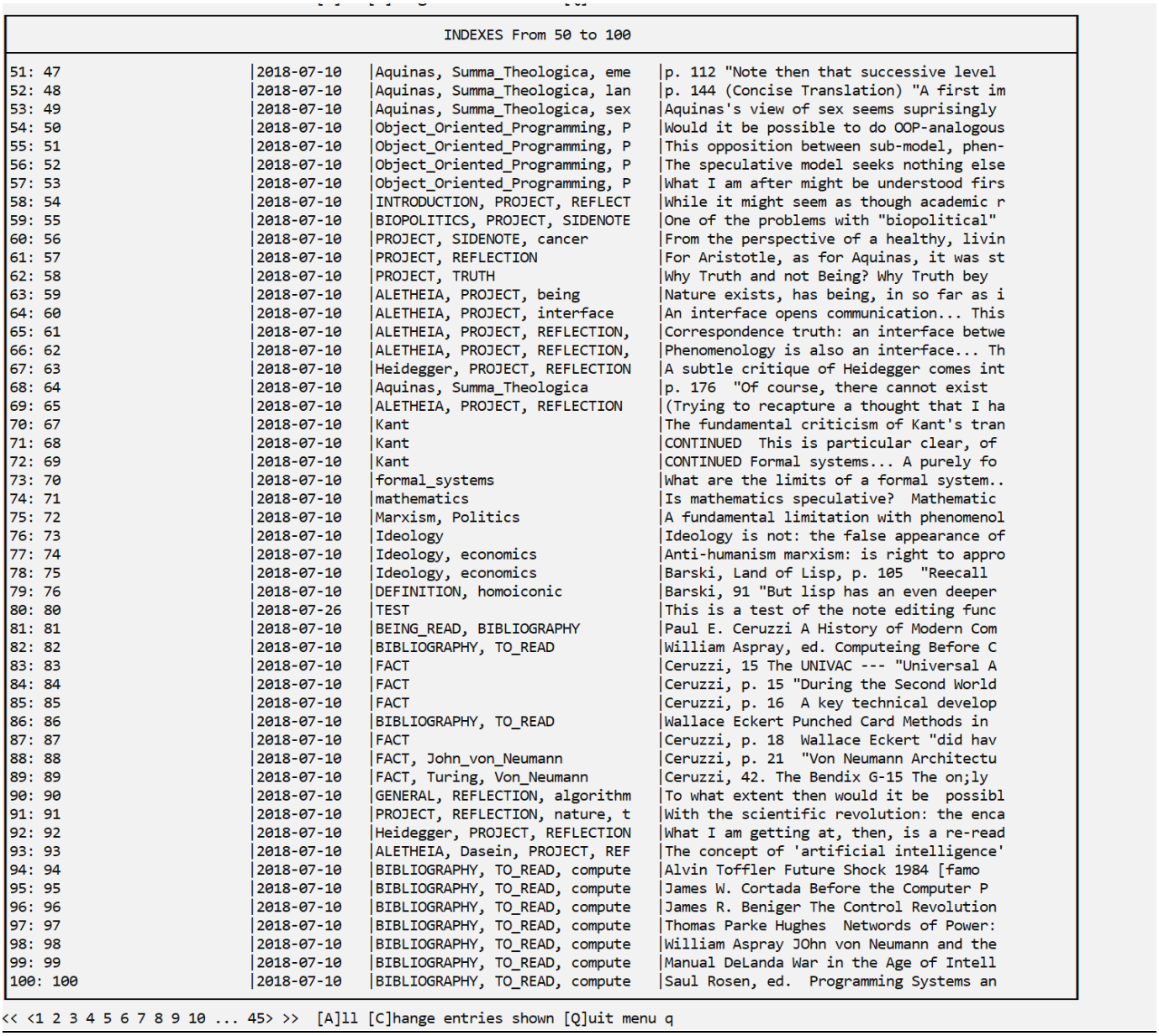
2.6.2.1.3. Scrolling through the notes

If many notes are listed, they will not be shown all at once, but will be presented through a menu that allows you to scroll back and forth between them.

If you do want to list all notes, press A[ll], whereas to change the number of entries shown, press C[hange]; the arrow keys advance forward and backward, while double arrows go to the first or last page of entries. [Q]uit returns to the command prompt.

2.6.2.2. The show command

The show command is identical to the all command, except that it does not have a "quick"-mode. It accepts one value: the collection of indexes to be shown.



INDEXES From 50 to 100

51: 47	2018-07-10	Aquinas, Summa_Theologica, eme	p. 112 "Note then that successive level
52: 48	2018-07-10	Aquinas, Summa_Theologica, lan	p. 144 (Concise Translation) "A first im
53: 49	2018-07-10	Aquinas, Summa_Theologica, sex	Aquinas's view of sex seems suprisingly
54: 50	2018-07-10	Object_Oriented_Programming, P	Would it be possible to do OOP-analogous
55: 51	2018-07-10	Object_Oriented_Programming, P	This opposition between sub-model, phen-
56: 52	2018-07-10	Object_Oriented_Programming, P	The speculative model seeks nothing else
57: 53	2018-07-10	Object_Oriented_Programming, P	What I am after might be understood firs
58: 54	2018-07-10	INTRODUCTION, PROJECT, REFLECT	While it might seem as though academic r
59: 55	2018-07-10	BIOPOLITICS, PROJECT, SIDENOTE	One of the problems with "biopolitical"
60: 56	2018-07-10	PROJECT, SIDENOTE, cancer	From the perspective of a healthy, livin
61: 57	2018-07-10	PROJECT, REFLECTION	For Aristotle, as for Aquinas, it was st
62: 58	2018-07-10	PROJECT, TRUTH	Why Truth and not Being? Why Truth bey
63: 59	2018-07-10	ALETHEIA, PROJECT, being	Nature exists, has being, in so far as i
64: 60	2018-07-10	ALETHEIA, PROJECT, interface	An interface opens communication... This
65: 61	2018-07-10	ALETHEIA, PROJECT, REFLECTION,	Correspondence truth: an interface betwe
66: 62	2018-07-10	ALETHEIA, PROJECT, REFLECTION,	Phenomenology is also an interface... Th
67: 63	2018-07-10	Heidegger, PROJECT, REFLECTION	A subtle critique of Heidegger comes int
68: 64	2018-07-10	Aquinas, Summa_Theologica	p. 176 "Of course, there cannot exist
69: 65	2018-07-10	ALETHEIA, PROJECT, REFLECTION	(Trying to recapture a thought that I ha
70: 67	2018-07-10	Kant	The fundamental criticism of Kant's tran
71: 68	2018-07-10	Kant	CONTINUED This is particular clear, of
72: 69	2018-07-10	Kant	CONTINUED Formal systems... A purely fo
73: 70	2018-07-10	formal_systems	What are the limits of a formal system..
74: 71	2018-07-10	mathematics	Is mathematics speculative? Mathematic
75: 72	2018-07-10	Marxism, Politics	A fundamental limitation with phenomenol
76: 73	2018-07-10	Ideology	Ideology is not: the false appearance of
77: 74	2018-07-10	Ideology, economics	Anti-humanism marxism: is right to appro
78: 75	2018-07-10	Ideology, economics	Barski, Land of Lisp, p. 105 "Reecall
79: 76	2018-07-10	DEFINITION, homoiconic	Barski, 91 "But lisp has an even deeper
80: 80	2018-07-26	TEST	This is a test of the note editing func
81: 81	2018-07-10	BEING_READ, BIBLIOGRAPHY	Paul E. Ceruzzi A History of Modern Com
82: 82	2018-07-10	BIBLIOGRAPHY, TO_READ	William Aspray, ed. Computeing Before C
83: 83	2018-07-10	FACT	Ceruzzi, 15 The UNIVAC --- "Universal A
84: 84	2018-07-10	FACT	Ceruzzi, p. 15 "During the Second World
85: 85	2018-07-10	FACT	Ceruzzi, p. 16 A key technical develop
86: 86	2018-07-10	BIBLIOGRAPHY, TO_READ	Wallace Eckert Punched Card Methods in
87: 87	2018-07-10	FACT	Ceruzzi, p. 18 Wallace Eckert "did hav
88: 88	2018-07-10	FACT, John_von_Neumann	Ceruzzi, p. 21 "Von Neumann Architectu
89: 89	2018-07-10	FACT, Turing, Von_Neumann	Ceruzzi, 42. The Bendix G-15 The on;ly
90: 90	2018-07-10	GENERAL, REFLECTION, algorithm	To what extent then would it be possibl
91: 91	2018-07-10	PROJECT, REFLECTION, nature, t	With the scientific revolution: the enca
92: 92	2018-07-10	Heidegger, PROJECT, REFLECTION	What I am getting at, then, is a re-read
93: 93	2018-07-10	ALETHEIA, Dasein, PROJECT, REF	The concept of 'artificial intelligence'
94: 94	2018-07-10	BIBLIOGRAPHY, TO_READ, compute	Alvin Toffler Future Shock 1984 [famo
95: 95	2018-07-10	BIBLIOGRAPHY, TO_READ, compute	James W. Cortada Before the Computer P
96: 96	2018-07-10	BIBLIOGRAPHY, TO_READ, compute	James R. Beniger The Control Revolution
97: 97	2018-07-10	BIBLIOGRAPHY, TO_READ, compute	Thomas Parke Hughes Networds of Power:
98: 98	2018-07-10	BIBLIOGRAPHY, TO_READ, compute	William Aspray JOhn von Neumann and the
99: 99	2018-07-10	BIBLIOGRAPHY, TO_READ, compute	Manual DeLanda War in the Age of Intell
100: 100	2018-07-10	BIBLIOGRAPHY, TO_READ, compute	Saul Rosen, ed. Programming Systems an

<< <1 2 3 4 5 6 7 8 9 10 ... 45> >> [A]ll [C]hange entries shown [Q]uit menu q

PLATE 2.66. A list of indexes

2.5.2.3. The inc command

The inc command shows the notes in the immediate vicinity of the current index. While inc is a bit slower than presenting an already-constituted list of indexes, it is much quicker than constituting all the indexes from scratch, and it also automatically reflects all modifications to the notebook.

2.6.3. The multi display mode

The “multi” mode fits notes together into a “sheet,” according to their width. The notes are placed in a stack, with the largest possible note that can fit in the free area withdrawn from the stack. The output of the multi display is stored under a user-defined name, and can also be saved as a text file.

```
defaultnotebook:4   inc
<inc>
```

1.18–11			
1.18		HEGELRIGHT@34.0, Hegel	Mit der Gottseligkeit und der Bibel aber hat es sich die höchste Berec
2		PROJECT, ontological neutralit	The ontological neutrality of language. Perhaps it cannot be a qu
3		ALETHEIA, PROJECT, THESIS	The aim of this book project is to present an account of truth as irre
4		ALETHEIA, PROJECT, THESIS	Being --- what 'is', the 'isness' of the Beingness of what is --- the
4.2#		ALETHEIA, PROJECT, THESIS	Or rather: he was sentenced to death, but then managed to escape..
4.3		Hegel, ALETHEIA, HEGELHIST@40.	
5#		Hegel, ALETHEIA, PROJECT	Ontology in this sense --- the tradition that runs from Parmenides to
6#		PROJECT, Heidegger, ALETHEIA	Heidegger seeks to come up with a terminology to undestand the non-c
7		PROJECT, ALETHEIA, <<8>>	The question becomes: how can we begin to get a sense for the exceess
8		ALETHEIA, PROJECT, ORGANIZATIO	TRUTH BEYOND BEING (TITLE) INTRODUCTION Truth beyond Being Th
9		ALETHEIA, PROJECT, STRATEGY	It is necessary to ask: will my appraoch be historical or thematic or
10		ALETHEIA, PROJECT, ORGANIZATIO	INTRODUCTION --- THE basic problematic PART I/ Showing the inadequ
10.1		Benjamin/philosopher	(st) (starting) distraction vs. contemplation The paintin
11		ALETHEIA, PROJECT, ORGANIZATIO	Ontology in Heidegger is the limit of a representationalist mode of th

```
[, >,<, ] (Q)uit >
```

4–18			
4		ALETHEIA, PROJECT, THESIS	Being --- what 'is', the 'isness' of the Beingness of what is --- the
4.2#		ALETHEIA, PROJECT, THESIS	Or rather: he was sentenced to death, but then managed to escape..
4.3		Hegel, ALETHEIA, HEGELHIST@40.	
5#		Hegel, ALETHEIA, PROJECT	Ontology in this sense --- the tradition that runs from Parmenides to
6#		PROJECT, Heidegger, ALETHEIA	Heidegger seeks to come up with a terminology to undestand the non-c
7		PROJECT, ALETHEIA, <<8>>	The question becomes: how can we begin to get a sense for the exceess
8		ALETHEIA, PROJECT, ORGANIZATIO	TRUTH BEYOND BEING (TITLE) INTRODUCTION Truth beyond Being Th
9		ALETHEIA, PROJECT, STRATEGY	It is necessary to ask: will my appraoch be historical or thematic or
10		ALETHEIA, PROJECT, ORGANIZATIO	INTRODUCTION --- THE basic problematic PART I/ Showing the inadequ
10.1		Benjamin/philosopher	(st) (starting) distraction vs. contemplation The paintin
11		ALETHEIA, PROJECT, ORGANIZATIO	Ontology in Heidegger is the limit of a representationalist mode of th
11.1		late poetry, Schmidt/author	THE "NEBENEINANDER" of both cyclic and linear history. Das N
12		ALETHEIA, PROJECT, THESIS	GUIDING PROPOSITIONS I. Truth is beyond being. A given ontology al
12.1		ALETHEIA, PROJECT, greatness o	Greatness of character consists in possessing strong instincts and kee
13		field, ALETHEIA, REFLECTION	What is the basis for suppsing as our starting point these two 'fields
14		ALETHEIA, REFLECTION	A guiding principle... A theory of truth cannot serve as a starti
15		ALETHEIA, STRATEGY, PROJECT	TRUTH BOOK I'm beginning to think that I should concentrate on He
16		ALETHEIA, PROJECT, philosophy	To reduce philosophy to therapy is to act as though wer were always
17		another God, Heidegger	Clearly, Feenberg does not like Heidegger's idea that we must wait fo
18		ALETHEIA, REFLECTION	The fundamental thought --- the truth is not disclosed onto-logically.

```
[, >,<, ] (Q)uit
```

PLATE 2.67: Incremental display of notes.

2.6.3.1. The multi command

The syntax consists in four values, and three modifiers:

multi:*INDEX COLLECTION*;*DISPLAY STREAM*;*WIDTH*;*SAVE STREAM* /* /? /=

The first value indicates the indexes to be displayed; the second the name of the "stream" to which display will be outputted, and the third the width, in characters, of the display.

The /$ modifier shows notes in a uniformly small size, as defined using the command smallsize.

The /* modifier is used to activate the "vary"-mode, adjusting the size of the notes according to the length of the text string.

The /? pauses the display after each line.

The /= saves the output to a textfile, using either the name of the display stream or name entered as value 4.

2.6.3.2. Streams

To list all the active streams, enters streams.

To display an active stream, enter showstream:*STREAM*

The /? modifier can be used to pause.

To delete an active stream: enter deletestream:*STREAM*.

2.7. MANIPULATING THE NOTEBOOK

A notebook, as mentioned, consists of a collection of notes with unique indexes. It is possible, using relatively simple commands, to reorganize the notes, move them to different index locations, create copies of them, merge notes together, and delete them.

*2.7.1. The basic program structure

The program architecture of *ARCADES* has been designed in such a way that all basic operations on the notes are handled by two methods: *addnew* and *move*. Notes are "deleted" by moving them to the negative indexes, though it is also possible to permanently delete motes through a third core method, *delete*, this need not be used.

Although this organization principle makes it easy to expand *ARCADES* while preventing subtle bugs from emerging, it comes at the cost of speed. Moving many notes to another location can take some time, and is not recommended.

2.7.2. Overview of commands for manipulating the notebook

Commands for manipulating the notebook may be divided into three categories:

1) Those which only affect the relation of the note to the index.

| These "index" commands include: delete, move, copy, clear, purge, rehome,

| eliminateblanks.

2) Those which change the "content" and "properties" of notes without changing indexes.

| These "content" commands include: correctkeys, reform,

3) Those which change both the "content" of notes – the keywords and text –as well as the relation of this content to indexes.

| These "note" commands include: revise, mergemany, conflate, split, sidenote

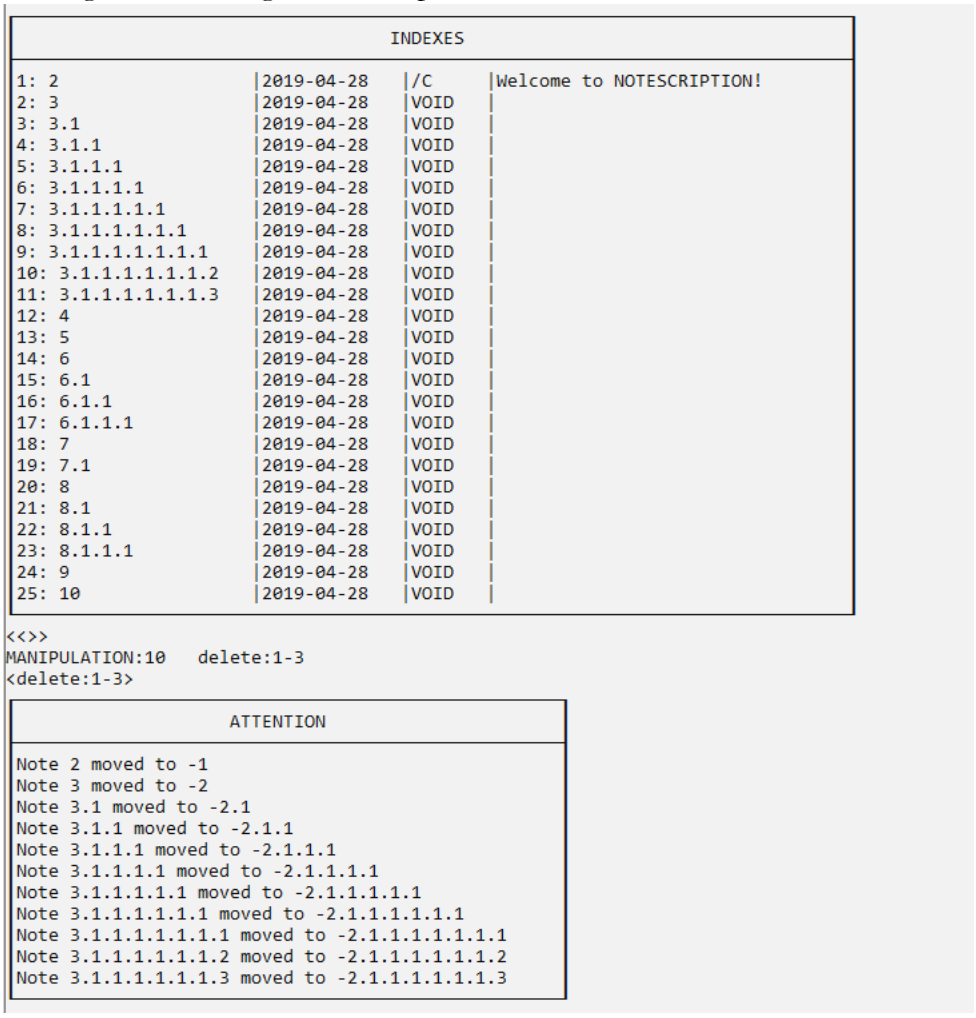
PLATE 2.68: Multiple display with smaller notes.

PLATE 2.69: Multiple display with larger notes.

2.7.2.1. “Index” commands

2.7.2.1.1. delete

Entering delete, or alternatively del, with a collection of notes as a value, deletes these notes by moving them to a negative index positions.



```
                                INDEXES
1: 2                  |2019-04-28   |/C     |Welcome to NOTESCRIPTION!
2: 3                  |2019-04-28   |VOID   |
3: 3.1                |2019-04-28   |VOID   |
4: 3.1.1              |2019-04-28   |VOID   |
5: 3.1.1.1            |2019-04-28   |VOID   |
6: 3.1.1.1.1          |2019-04-28   |VOID   |
7: 3.1.1.1.1.1        |2019-04-28   |VOID   |
8: 3.1.1.1.1.1.1      |2019-04-28   |VOID   |
9: 3.1.1.1.1.1.1.1    |2019-04-28   |VOID   |
10: 3.1.1.1.1.1.1.2   |2019-04-28   |VOID   |
11: 3.1.1.1.1.1.1.3   |2019-04-28   |VOID   |
12: 4                 |2019-04-28   |VOID   |
13: 5                 |2019-04-28   |VOID   |
14: 6                 |2019-04-28   |VOID   |
15: 6.1               |2019-04-28   |VOID   |
16: 6.1.1             |2019-04-28   |VOID   |
17: 6.1.1.1           |2019-04-28   |VOID   |
18: 7                 |2019-04-28   |VOID   |
19: 7.1               |2019-04-28   |VOID   |
20: 8                 |2019-04-28   |VOID   |
21: 8.1               |2019-04-28   |VOID   |
22: 8.1.1             |2019-04-28   |VOID   |
23: 8.1.1.1           |2019-04-28   |VOID   |
24: 9                 |2019-04-28   |VOID   |
25: 10                |2019-04-28   |VOID   |
<<>>
MANIPULATION:10   delete:1-3
<delete:1-3>
                    ATTENTION
Note 2 moved to -1
Note 3 moved to -2
Note 3.1 moved to -2.1
Note 3.1.1 moved to -2.1.1
Note 3.1.1.1 moved to -2.1.1.1
Note 3.1.1.1.1 moved to -2.1.1.1.1
Note 3.1.1.1.1.1 moved to -2.1.1.1.1.1
Note 3.1.1.1.1.1.1 moved to -2.1.1.1.1.1.1
Note 3.1.1.1.1.1.1.1 moved to -2.1.1.1.1.1.1.1
Note 3.1.1.1.1.1.1.2 moved to -2.1.1.1.1.1.1.2
Note 3.1.1.1.1.1.1.3 moved to -2.1.1.1.1.1.1.3
```

PLATE 2.70: Deleting a range of indexes.

2.7.2.1.2. clear

The clear command is used to delete all the notes from the notebook by moving them to negative index locations.

Because it is rather dangerous to use, it queries you each at step to see if you want to continue.

There is little reason for using clear except when working with very small notebooks

```
MANIPULATION:10   clear
<clear>
Are you sure?yes
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
Go on?
              ATTENTION

Note 3.1.1.1.1.1.1 moved to -3
Note 3.1.1.1.1.1.1.1 moved to -4
Note 3.1.1.1.1.1.1.2 moved to -5
Note 3.1.1.1.1.1.1.3 moved to -6
Note 4 moved to -7
Note 5 moved to -8
Note 6 moved to -9
Note 6.1 moved to -10
Note 6.1.1 moved to -11
Note 6.1.1.1 moved to -12
Note 7 moved to -13
Note 7.1 moved to -14
Note 8 moved to -15
Note 8.1 moved to -16
Note 8.1.1 moved to -17
Note 8.1.1.1 moved to -18
Note 9 moved to -19
Note 10 moved to -20

<<>>
MANIPULATION:10
```

PLATE 2.71: Clearing deleted notes.

2.7.2.1.2.1. The undel command

The command undel is used to “undelete” the notes with negative indexes, moving them back into positive registers.

2.7.2.1.3. compress

The compress command “compresses” the notebook by moving all the notes to contiguous index positions.

```
DISPLAY:17   all /$
<all /$>
```

```
From 0 to 20
2  |   | /C                              | Welcome to NOTESCRIPTION!
3  |   | Dasein/de, The Question Concer  | Im folgenden fragen wir nach der {Technik/de}. Das Fragen baut an eine
4  |   | The Question Concerning Techno  | Die {Technik/de} ist nicht das gleiche wie das {Wesen der Technik}.
5  |   | The Question Concerning Techno  | Die gängige Vorstellung von der Technik, wonach sie ein Mittel ist und
6  |   | moderne Technik/de, Kraftwerk/  | Die instrumentale Bestimmung der Technik ist sogar so unheimlich richt
7  |   | The Question Concerning Techno  | {Wahrheit/de} {Richtigkeit/de} {Wille/de}Gesetzt nun aber, die Technik
8  |   | The Question Concerning Techno  | Man pflegt seit langem die Ursache als das Bewirkende vorzustellen. Wi
9  |   | The Question Concerning Techno  | {ETYMOLOGY} {causality} Causa, casus, gehört zum Zeitwort cadere, fall
16 |   | The Question Concerning Techno  | So ist denn das Ge-stell als ein Geschick der Entbergung zwar das We
17 |   | The Question Concerning Techno  | {ETYMOLOGY} {Wesen/de} Schon wenn wir »Hauswesen«, »Staatswesen« sagen
18 |   | The Question Concerning Techno  | Alles Wesende währt. Aber ist das Währende nur d+as Fortwährende?
19 |   | The Question Concerning Techno  | Allein, wenn dieses Geschick, das Ge-stell, die äußerste Gefahr ist, n
20 |   | The Question Concerning Techno  | Darum liegt alles daran, daß wir den Aufgang bedenken und andenkend hü
21 |   | The Question Concerning Techno  | {ambiguity} Einmal fordert das {Ge-stell/de} in das Rasende des Bestel
22 |   | The Question Concerning Techno  | Das Unaufhaltsame des Bestellens und das Verhaltene des Rettenden zieh
23 |   | The Question Concerning Techno  | Blicken wir in das zweideutige Wesen der Technik, dann erblicken wir d
24 |   | προμος, Zwiesprache             | {Kunst/de} Am Beginn des abendländischen Geschickes stiegen in Grieche
25 |   | The Question Concerning Techno  | {Dichtung/de} {Kunst/de} Das Dichterische bringt das Wahre in den Glan
26 |   | The Question Concerning Techno  | Weil das Wesen der Technik nichts Technisches ist, darum muß die wesen
27 |   | The Question Concerning Techno  | Je mehr wir uns der Gefahr nähern, um so heller beginnen die Wege ins
```

```
<<>>
DISPLAY:17   compress
<compress>
```

```
ATTENTION
Note 2 moved to 1
Note 3 moved to 2
Note 4 moved to 3
Note 5 moved to 4
Note 6 moved to 5
Note 7 moved to 6
Note 8 moved to 7
Note 9 moved to 8
Note 16 moved to 9
Note 17 moved to 10
Note 18 moved to 11
Note 19 moved to 12
Note 20 moved to 13
Note 21 moved to 14
Note 22 moved to 15
Note 23 moved to 16
Note 24 moved to 17
Note 25 moved to 18
Note 26 moved to 19
Note 27 moved to 20
```

PLATE 2.72: Using the compress command.

```
From 0 to 29
1            |   | /C                            | Welcome to NOTESCRIPTION!
2            |   | The Question Concerning Techno | Im folgenden fragen wir nach der {Technik/de}. Das Fragen baut an eine
3            |   | The Question Concerning Techno | Die {Technik/de} ist nicht das gleiche wie das {Wesen der Technik}.
4            |   | The Question Concerning Techno | Die gängige Vorstellung von der Technik, wonach sie ein Mittel ist und
5            |   | moderne Technik/de, Kraftwerk/ | Die instrumentale Bestimmung der Technik ist sogar so unheimlich richt
6            |   | The Question Concerning Techno | {Wahrheit/de} {Richtigkeit/de} {Wille/de}Gesetzt nun aber, die Technik
7            |   | The Question Concerning Techno | Man pflegt seit langem die Ursache als das Bewirkende vorzustellen. Wi
8            |   | The Question Concerning Techno | {ETYMOLOGY} {causality} Causa, casus, gehört zum Zeitwort cadere, fall
9            |   | The Question Concerning Techno | So ist denn das Ge-stell als ein Geschick der Entbergung zwar das We
10           |   | The Question Concerning Techno | {ETYMOLOGY} {Wesen/de} Schon wenn wir »Hauswesen«, »Staatswesen« sagen
11           |   | The Question Concerning Techno | Alles Wesende währt. Aber ist das Währende nur d+as Fortwährende?
12           |   | The Question Concerning Techno | Allein, wenn dieses Geschick, das Ge-stell, die äußerste Gefahr ist, n
13           |   | The Question Concerning Techno | Darum liegt alles daran, daß wir den Aufgang bedenken und andenkend hü
14           |   | The Question Concerning Techno | {ambiguity} Einmal fordert das {Ge-stell/de} in das Rasende des Bestel
15           |   | The Question Concerning Techno | Das Unaufhaltsame des Bestellens und das Verhaltene des Rettenden zieh
16           |   | The Question Concerning Techno | Blicken wir in das zweideutige Wesen der Technik, dann erblicken wir d
17           |   | Zwiesprache, προμος           | {Kunst/de} Am Beginn des abendländischen Geschickes stiegen in Grieche
18           |   | The Question Concerning Techno | {Dichtung/de} {Kunst/de} Das Dichterische bringt das Wahre in den Glan
19           |   | The Question Concerning Techno | Weil das Wesen der Technik nichts Technisches ist, darum muß die wesen
20           |   | The Question Concerning Techno | Je mehr wir uns der Gefahr nähern, um so heller beginnen die Wege ins
21           |   | VOID                          |
21.1.1.1     |   | VOID                          |
21.1.1.1.1   |   | VOID                          |
21.1.1.1.1.1 |   | \                             |
22           |   | VOID                          |
22.1.1       |   | VOID                          |
23           |   | VOID                          |
23.1         |   | VOID                          |
24           |   | VOID                          | ;;
```

```
<<>>
DISPLAY:23.1   rehome:1-24
<rehome:1-24>
```

```
ATTENTION
Note 21.1.1.1 moved to 21.1
Note 21.1.1.1.1 moved to 21.1.1
Note 21.1.1.1.1.1 moved to 21.1.1.1
Note 22.1.1 moved to 22.1
Successfully rehomed!
```

```
<<>>
DISPLAY:23.1
```

PLATE 2.73: Rehoming orphaned notes.

2.7.2.1.4. rehome

The rehome command is similar to compress, but moves notes in such a way as to eliminate orphan notes. The rehome command accepts as a value a collection of notes.

2.7.2.1.5. move

The move command allows you not only to move individual notes from one index location to another, but also to move collections of indexes and even transform the hierarchical structure of the notebook itself.

The syntax of the move command is as follows:

move:*INDEX*ESFROM;*INDEX*TO;[S/M/C];[yes/no] /$ /& /* /? /=

The first value indicates the collection of indexes that will be moved; the second the destination to which they will be moved; the third – either S, M or C indicated how the how the notes will be organized at the destination; and the fourth value whether the children of indexes will be included.

The first of the three modifiers are to indicate the third value, while the fourth and fifth are used for the fourth value. Thus:

(1) /$ ≈ (is equivalent to) S ≈ Subordinate the indexes under the destination index, preserving the hierarching structure of the source collection.

(2) /& ≈ M ≈ Make the indexes "compact" ("compressed") under the destination index, so that whatever hierarchical structure they have is eliminated. Whatever their structure in the source was, they will be turned into a series of siblings.

(3) /* ≈ C ≈ Organize the indexes as a series of "children" in the destination range.

(4) /? ≈ yes ≈ exclude "child" indexes from the source range.

(5) /= ≈ no ≈ don't exclude "child" indexes

It is important to note that the behavior of S/M/C depends on whether the destination range is already occupied. If there is an index in the destination, then the new notes will be placed in a subordinating relation – either (S), which creates a "microcosmic representation" of the source range subordinated to destination index, or (M), which flattens them into siblings, children of the same index; or (C), which turns them into a chain of children.

Observe, also, that *ARCADES* does not forbid you from moving the source range into itself, nor can this "crash" the system. This is because notes are moved individually, the interpretation of each "move" depends on the present state of the system after an individual move, and *ARCADES* will either refuse to execute a move (if the source index is empty) or will interpret it such that it can be done (if the destination is occupied). For this reason, though, it may be a bit tricky to foresee how a command will be executed when it moves indexes onto themselves.

```
<<>>
MOVEDEMO:6   all /$
<all /$>
```

From 0 to 5			
2		/C	Welcome to NOTESCRIPTION!
3		VOID	FIRST NOTE
4		VOID	SECOND NOTE
5		VOID	THIRD NOTE
6		VOID	FOURTH NOTE

```
<<>>
MOVEDEMO:6   move:3-6;7
<move:3-6;7>
[S]ubordinate,  [M]ake compact or [C]hildren?

No children?
```

ATTENTION
Note 3 moved to 7
MOVING to 8
Note 4 moved to 8
MOVING to 9
Note 5 moved to 9
MOVING to 10
Note 6 moved to 10

```
<<>>
MOVEDEMO:6
```

PLATE 2.74: A simple use of the move command.

```
<<>>
MOVEDEMO:6   all /$
<all /$>
```

From 0 to 5			
2		/C	Welcome to NOTESCRIPTION!
7		VOID	FIRST NOTE
8		VOID	SECOND NOTE
9		VOID	THIRD NOTE
10		VOID	FOURTH NOTE

```
<<>>
MOVEDEMO:6   move:8-10;7;M;no
<move:8-10;7;M;no>
M
```

ATTENTION
Note 8 moved to 7.1
Note 9 moved to 7.2
Note 10 moved to 7.3

```
<<>>
MOVEDEMO:6   all /$
<all /$>
```

From 0 to 5			
2		/C	Welcome to NOTESCRIPTION!
7		VOID	FIRST NOTE
7.1		VOID	SECOND NOTE
7.2		VOID	THIRD NOTE
7.3		VOID	FOURTH NOTE

```
<<>>
MOVEDEMO:6
```

PLATE 2.75: Moving with “make compact.”

```
<<>>
MOVEDEMO:8   all /$
<all /$>
```

From 0 to 6	
2 | | /C	Welcome to NOTESCRIPTION!
7 | | VOID	FIRST NOTE
7.1 | | VOID	SECOND NOTE
7.2 | | VOID	THIRD NOTE
7.3 | | VOID	FOURTH NOTE
8 | | VOID	FIFTH NOTE

```
<<>>
MOVEDEMO:8   move:7-7.3;8;S;no
<move:7-7.3;8;S;no>
S
```

ATTENTION
Note 7 moved to 8.7
Note 7.1 moved to 8.7.1
Note 7.2 moved to 8.7.2
Note 7.3 moved to 8.7.3

```
<<>>
MOVEDEMO:8   all /$
<all /$>
```

From 0 to 6	
2 | | /C	Welcome to NOTESCRIPTION!
8 | | VOID	FIFTH NOTE
8.7 | | VOID	FIRST NOTE
8.7.1 | | VOID	SECOND NOTE
8.7.2 | | VOID	THIRD NOTE
8.7.3 | | VOID	FOURTH NOTE

```
<<>>
MOVEDEMO:8
```

PLATE 2.76: Moving with “subordinate”.

```
<<>>
MOVEDEMO:9   all /$
<all /$>
```

From 0 to 7	
2 | | /C	Welcome to NOTESCRIPTION!
8 | | VOID	FIFTH NOTE
8.7 | | VOID	FIRST NOTE
8.7.1 | | VOID	SECOND NOTE
8.7.2 | | VOID	THIRD NOTE
8.7.3 | | VOID	FOURTH NOTE
9 | | VOID	SIXTH NOTE

```
<<>>
MOVEDEMO:9   move:8-8.8;9;M;no
<move:8-8.8;9;M;no>
M
```

ATTENTION
Note 8 moved to 9.1
Note 8.7 moved to 9.2
Note 8.7.1 moved to 9.3
Note 8.7.2 moved to 9.4
Note 8.7.3 moved to 9.5

```
<<>>
MOVEDEMO:9   all /$
<all /$>
```

From 0 to 7	
2 | | /C	Welcome to NOTESCRIPTION!
9 | | VOID	SIXTH NOTE
9.1 | | VOID	FIFTH NOTE
9.2 | | VOID	FIRST NOTE
9.3 | | VOID	SECOND NOTE
9.4 | | VOID	THIRD NOTE
9.5 | | VOID	FOURTH NOTE

```
<<>>
MOVEDEMO:9
```

PLATE 2.77: Notice how the order gets mixed up.

```
<<>>
MOVEDEMO:9   all /$
<all /$>
```

From 0 to 7			
2		/C	Welcome to NOTESCRIPTION!
9		VOID	SIXTH NOTE
9.1		VOID	FIFTH NOTE
9.2		VOID	FIRST NOTE
9.3		VOID	SECOND NOTE
9.4		VOID	THIRD NOTE
9.5		VOID	FOURTH NOTE

```
<<>>
MOVEDEMO:9   move:9-9.6;10-15
<move:9-9.6;10-15>
[S]ubordinate,  [M]ake compact or [C]hildren?

No children?
```

ATTENTION
Note 9 moved to 10
Note 9.1 moved to 8
MOVING to 9
Note 9.2 moved to 9
MOVING to 11
Note 9.3 moved to 11
MOVING to 12
Note 9.4 moved to 12
MOVING to 13
Note 9.5 moved to 13

```
<<>>
MOVEDEMO:9   all /$
<all /$>
```

From 0 to 7			
2		/C	Welcome to NOTESCRIPTION!
8		VOID	FIFTH NOTE
9		VOID	FIRST NOTE
10		VOID	SIXTH NOTE
11		VOID	SECOND NOTE
12		VOID	THIRD NOTE
13		VOID	FOURTH NOTE

```
<<>>
MOVEDEMO:9
```

PLATE 2.78: Moving from one range into another range.

2.7.2.1.6. copy

The copy command is identical to the move command, save that it does not delete the note from the source index.

2.7.2.1.7. Permanently deleting notes

The command permdel is used to permanently delete all the notes with negative indexes.

2.7.2.1.8. Showing notes with negative indexes

To see the notes with negative indexes – the temporarily deleted notes – use the command showdel.

2.7.2.2. "Content" commands

2.7.2.2.1. correctkeys

The command correctkeys is used to change the keys that are found over a collection of keys. It is important to note that it changes the keys globally, regardless of the scope of the collection. It accepts only one value, the collection of indexes, and one modifier /$, which specifies whether it should only correct keys, or keys together with tags.

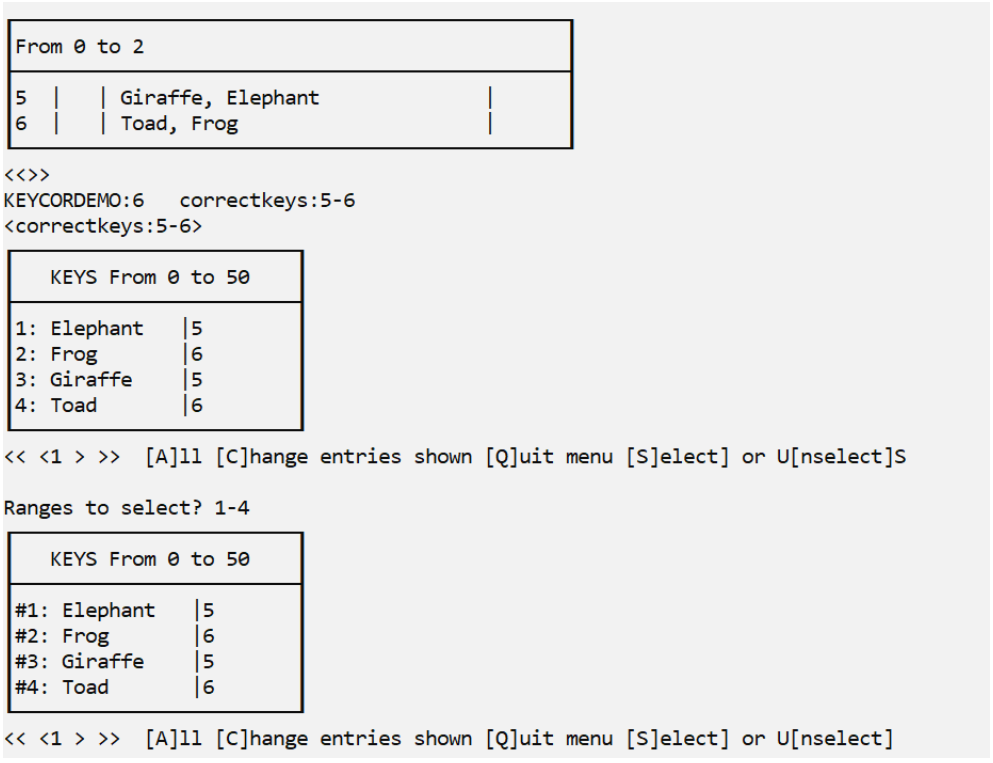


PLATE 2.79: Correcting keys I – Selecting keys to revise.

```
<< <1 > >>  [A]ll [C]hange entries shown [Q]uit menu [S]elect] or U[nselect]
Frog
6
Revise  Frog  Enter new term, RETURN to keep, or (d)elete.Frogo
Change Frog to Frogo
6
6  | |2019-05-11  | Toad, Frogo |
Toad
-3, 6
Revise  Toad  Enter new term, RETURN to keep, or (d)elete.Toado
Change Toad to Toado
6
6  | |2019-05-11  | Toado, Frogo |
Giraffe
5
Revise  Giraffe  Enter new term, RETURN to keep, or (d)elete.Giraffino
Change Giraffe to Giraffino
5
5  | |2019-05-11  | Giraffino, Elephant |
Elephant
-5, 5
Revise  Elephant  Enter new term, RETURN to keep, or (d)elete.Elephantino
Change Elephant to Elephantino
5
5  | |2019-05-11  | Giraffino, Elephantino
|
```

PLATE 2.80: Correcting keys II – Going through the selected keys.

```
<<>>
KEYCORDEMO:6   all /$
<all /$>
```

From 0 to 3			
2	|	/C	Welcome to NOTESCRIPTION!
5	|	Giraffino, Elephantino	
6	|	Toado, Frogo	

```
<<>>
```

PLATE 2.81: Correcting keys III – The final result.

2.7.2.2.2. reform

The command reform is used to correct the spelling and eliminate formatting errors across a collections of notes.

2.7.2.3. "Note" commands

2.7.2.3.1 revise

The command revise (alternate form: rev) is used to add to an existing note. It can either add the text of one existing note to another note, or call up the text inputter. The user can also define the text placed between the note, and whether the new text is placed before or after.

The revise command accepts three values and the modifiers /$ /& /* /?. The first value is used to select the index or collection of indexes to be revised, the second value is the index of the note to be added, and the third value is the text to be inserted between notes.

The /$ modifier is used to add the new text to the beginning rather than the end.

The /& modifier adds the new text both the beginning and the end.

The /* modifier selects /BREAK/ as the "breaker"

The /? modifier selects /NEW/ as the "breaker"

2.7.2.3.2 mergemany

The command mergemany is used to take a collection of indexes, and join them together into a single note. It does not delete the indexes that is conflates, and hence it can be understood as a mode of copying. It accepts three values and the modifiers /$ and /&.

The first value is the collection of indexes to be joined together. The second value is the index of the destination where the new note is to be placed. The third value is the manner in which the notes are to be joined together, and can be either (c)onflate or (e)mbed. The modifier /$ is the equivalent to the value (c)onflate while the modifier /& is the equivalent to the value (e)mbed. If you use both modifiers, it will repeat the action twice in each of the two manners.

The “embed” joins together displayed notes, together with brackets, whereas “conflate” merely combines the text of the notes and their keywords.

```
ALLWORK:7   mergemany:2-7;10;E
<mergemany:2-7;10;E>
```

10 | |2019-05-19 | SHINING, /C |

#1
2 | |2019-05-19 | /C |

Welcome to NOTESCRIPTION!

#2
3 | |2019-05-19 | SHINING |

All work and no play makes jack a dull boy.

#3
4 | |2019-05-19 | SHINING |

All work and no play makes jack a dull boy.

#4
5 | |2019-05-19 | SHINING |

All work and no play makes jack a dull boy.

#5
6 | |2019-05-19 | SHINING |

All work and no play makes jack a dull boy.

#6
7 | |2019-05-19 | SHINING |

All work and no play makes jack a dull boy.

PLATE 2.82: Merging several notes together with mergemany in the “embed” mode.

```
4  |  | SHINING |

All work and no play makes jack a dull boy.


5  |  | SHINING |

All work and no play makes jack a dull boy.


6  |  | SHINING |

All work and no play makes jack a dull boy.


7  |  | SHINING |

All work and no play makes jack a dull boy.


<<>>
ALLWORK:7   mergemany:3-7;12;C
<mergemany:3-7;12;C>

    ATTENTION
Note added at 12

<<>>
ALLWORK:7   12
<12>

12  | |2019-05-19  | SHINING |

All work and no play makes jack a dull boy.
All work and no play makes jack a dull boy.
All work and no play makes jack a dull boy.
All work and no play makes jack a dull boy.
All work and no play makes jack a dull boy.


<<>>
ALLWORK:7
```

PLATE 2.83: Merging several notes together with mergemany in the "conflate" mode.

2.7.2.3.3. conflate

The command conflate is similar to revise and mergmany. It joins a collection of notes together separated either by /BREAK/, /NEW/ or a user-defined term.

It accepts four values and the modifiers /$, /&, /*, and /?. The values consist in (1) the collection of the indexes of the notes to be joined together (2) the destination of the newly created note 3) (e)mpty, (b)reak, or (n)ew, and 4) a user defined "breaker".

The first three modifiers are equivalent, respectively, to (e)mpty, (b)reak, and (n)ew, while /? Is used to solicit a user-defined "breaker."

2.7.2.3.4. split

The split command can be used to automatically split the text of an existing note into regular columns. It accepts four values: the index of the note to split, the number of columns, the width of columns, and the "breaking" character used to split the text into words.

```
29  | |2019-05-10  | VOID |

Als Zarathustra dreissig Jahr alt war, verliess
er seine Heimat und den See seiner Heimat und
ging in das Gebirge. Hier genoss er seines
Geistes und seiner Einsamkeit und wurde dessen
zehn Jahr nicht müde. Endlich aber verwandelte
sich sein Herz, - und eines Morgens stand er
mit der Morgenröthe auf, trat vor die Sonne
hin und sprach zu ihr also"Du grosses Gestirn!
Was wäre dein Glück, wenn du nicht Die hättest,
welchen du leuchtest! Zehn Jahre kamst du hier
herauf zu meiner Höhle: du würdest deines Lichtes
und dieses Weges satt geworden sein, ohne mich,
meinen Adler und meine Schlange. Aber wir warteten
deiner an jedem Morgen, nahmen dir deinen Überfluss
ab und segneten dich dafür. Siehe! Ich bin
meiner Weisheit überdrüssig, wie die Biene,
die des Honigs zu viel gesammelt hat, ich bedarf
der Hände, die sich ausstrecken. Ich möchte
verschenken und austheilen, bis die Weisen
unter den Menschen wieder einmal ihrer Thorheit
und die Armen einmal ihres Reichthums froh
geworden sind. Dazu muss ich in die Tiefe steigen:
wie du des Abends thust, wenn du hinter das
Meer gehst und noch der Unterwelt Licht bringst,
du überreiches Gestirn! Ich muss, gleich dir,
untergehen, wie die Menschen es nennen, zu
denen ich hinab will. So segne mich denn, du
ruhiges Auge, das ohne Neid auch ein allzugrosses
Glück sehen kann! Segne den Becher, welche
überfliessen will, dass das Wasser golden aus
ihm fliesse und überallhin den Abglanz deiner
Wonne trage! Siehe! Dieser Becher will wieder
leer werden, und Zarathustra will wieder Mensch
werden." - Also begann Zarathustra's Untergang.
```

```
ATTENTION
Note added at 29
```

PLATE 2.84: Automatically splitting text with split (before).

```
SPLITDEMO:29   split:29;6
<split:29;6>
```

ATTENTION
Note added at 31

```
<<>>
SPLITDEMO:29   31
<31>
```

31	|2019-05-10	VOID |

Als Zarathustra dreissig Jahr alt war, verliess er seine Heimat und den See seiner Heimat und ging in das Gebirge. Hier genoss er seines Geistes und seiner Einsamkeit und wurde dessen zehn Jahr nicht müde. Endlich aber verwandelte sich sein Herz, - und	|eines Morgens stand er mit der Morgenröthe auf, trat vor die Sonne hin und sprach zu ihr also"Du grosses Gestirn! Was wäre dein Glück, wenn du nicht Die hättest, welchen du leuchtest! Zehn Jahre kamst du hier herauf zu meiner Höhle: du würdest	|deines Lichtes und dieses Weges satt geworden sein, ohne mich, meinen Adler und meine Schlange. Aber wir warteten deiner an jedem Morgen, nahmen dir deinen Überfluss ab und segneten dich dafür. Siehe! Ich bin meiner Weisheit überdrüssig, wie die Biene, die des Honigs	|zu viel gesammelt hat, ich bedarf der Hände, die sich ausstrecken. Ich möchte verschenken und austheilen, bis die Weisen unter den Menschen wieder einmal ihrer Thorheit und die Armen einmal ihres Reichthums froh geworden sind. Dazu muss ich in die Tiefe steigen: wie	|du des Abends thust, wenn du hinter das Meer gehst und noch der Unterwelt Licht bringst, du überreiches Gestirn! Ich muss, gleich dir, untergehen, wie die Menschen es nennen, zu denen ich hinab will. So segne mich denn, du ruhiges Auge, das ohne	|Neid auch ein allzugrosses Glück sehen kann! Segne den Becher, welche überfliessen will, dass das Wasser golden aus ihm fliesse und überallhin den Abglanz deiner Wonne trage! Siehe! Dieser Becher will wieder leer werden, und Zarathustra will wieder Mensch werden." - Also begann Zarathustra's Untergang.

```
<<>>
SPLITDEMO:29
```

PLATE 2.85: Automatically splitting text with split (before).

2.7.2.3.5. columns

The command column (alternative form: col) is used to convert a regular note into a note with columns. It can also be used to remove columns.

The columns command accepts two values, and the modifiers /$, /&, and /*.

The first value is the collection of indexes to convert to columns, while the second value is the character that is to be converted into the underline (_) mark distinguishing columns across the line.

The modifier /$ can be selected to undo columns while /& adds counters, and /* adds counters without converting to columns.

2.7.2.3.6. sidenote

The command sidenote can be used to take a collection of indexes and merges them together as a series of columns.

It accepts two values, and the single modifier /$.

The first value indicates a collection of indexes, and the second value the width of the columns. The modifier /$ is used to add counters.

```
<<>>
SIDENOTE:5   sidenote:3-5;35
<sidenote:3-5;35>
```

ATTENTION
Note added at 7

```
<<>>
SIDENOTE:5   7
<7>
```

7	2019-05-10	VOID

Im dunkeln Efeu saß ich, an der Pforte Des Waldes, eben, da der goldene Mittag, Den Quell besuchend, herunterkam Von Treppen des Alpengebirgs, Das mir die göttlichgebaute, Die Burg der Himmlischen heißt Nach alter Meinung, wo aber Geheim noch manches entschieden Zu Menschen gelanget; von da Vernahm ich ohne Vermuten Ein Schicksal, denn noch kaum War mir im warmen Schatten Sich manches beredend, die Seele Italia zu geschweift Und fernhin an	Jetzt aber, drin im Gebirg, Tief unter den silbernen Gipfeln Und unter fröhlichem Grün, Wo die Wälder schauernd zu ihm, Und der Felsen Häupter übereinander Hinabschaun, taglang, dort Im kältesten Abgrund hört Ich um Erlösung jammern Den Jüngling, es hörten ihn, wie er tobt', Und die Mutter Erd anklagt',[149] Und den Donnerer, der ihn gezeuget, Erbarmend die Eltern, doch Die Sterblichen flohn von dem Ort, Denn furchtbar war, da lichtlos er In den Fesseln sich wälzte, Das Rasen des Halbgotts unerfahrne Seele	Die Stimme wars des edelsten der Ströme, Des freigeborenen Rheins, Und anderes hoffte der, als droben von den Brüdern, Dem Tessin und dem Rhodanus, Er schied und wandern wollt, und ungeduldig ihn Nach Asia trieb die königliche Seele. ist Das Wünschen vor dem Schicksal. Blindesten aber Sind Göttersöhne. Denn es kennet der Mensch Sein Haus und dem Tier ward, wo Es bauen solle, doch jenen ist Der Fehl, daß sie nicht wissen wohin In die gegeben.

```
<<>>
SIDENOTE:5
```

PLATE 2.86: Merging notes into columns with sidenote.

2.8. REPRESENTING THE NOTEBOOK

In addition to simply displaying notes, either singly or en masse, *ARCADES* offers various more sophisticated methods for representing the content of a notebook.

2.8.1 Searching

The *ARCADES* search function has been designed to be both versatile and fast, allowing you to search for a variety of different attributes of the note, including keywords, words in the text, connected and disjoint phrases and tags. It allows for great logical expressiveness, with nestled parentheses, and the three basic logical operations (AND, OR, NOT), and the option of attaching "qualification" to individual terms. It can also be used to search over sequences, allowing you, for example, to find all the pages within a certain range in a certain text, or limit the search for a keyword accordingly. Finally, it works together with the ontological and relational knowledge base.

*2.8.1.1. Implementation of the search function

2.8.1.1.1. Elimination of reliance on eval.

The search function currently no longer depends on eval to parse search commands, but has a dedicated phrase parser.

2.8.1.1.2. Data structure

The speed of the search function is achieved by using dictionaries to keep track of all searchable terms rather than searching directly over the notes themselves. Separate dictionaries are reserved for words in text, keywords, tags. These dictionaries are not kept in the shelf, but in the pickled default file. This allows them to be reconstituted if they become corrupted.

When opening a notebook, you have the option of reconstituting the word dictionary. And it is also possible to do it at any subsequent point by using the refresh command.

2.8.1.2. Using the search function

2.8.1.2.1. Calling the search function

The search function can be initiated using either the search command, or a single question mark (?) as the abbreviated form.

If you are refeeding results from another command into a search, it will be necessary to avoid the abbreviated form, since the question mark is also used as the placeholder for previous result.

The search command accepts one value, the search phrase, as well as four modifiers.

These are:

(1) /$ | shows notes resulting from the search.

(2) /& | shows the results of the search organized by date

(3) /* | Overrides the deletion of the already constituted date-dictionary of the keys

(4) /? | Includes indexes in the date dictionary

The third and fourth modifier depend on the use of the second.

2.8.1.2.2. The search phrase

The search phrase consists in three elements: terms, specifiers, and logical operators.

2.8.1.2.2.1. The term

Search terms consist in a group of characters connected without spaces, --- a single word, for example ---, or a keyword phrase contained in arrow brackets <>. To search for a non-keyword phrase, the DOLLAR ($) is used in lieu of spaces.

The search term is ordinarily case sensitive, unless it is written in all caps.

2.8.1.2.2.1.1. Wildcards

The star (*) can be used as a wildcard.

For example:

(1) *pig	searches for words ending in "pig."
(2) frog*	searches for words beginning in "frog."
(3) *ab*	searches for any word containing "ab"
(4) a*b*c*d	searches for a word that starts with "a", ends with "d", and contains "b" and "c" in the middle in that order.

2.8.1.2.2.2. Specifiers

The following specifiers indicate whether the term is to be searched for in the text of the notes, among the keywords, among tags, or among the metatags (knowledge).

(1) TERM	as word in the note text
(2) <TERM>	as keyword
(3) <#TERM>	as tag
(4) <##TERM>	as metatag
(5) $This$is$a$phrase	to search for a literal phrase

2.8.1.2.2.3. Logical operators

The following logical operators can be used:

(1) vertical line (|)	OR
(2) ampersand (&)	AND
(3) tilde (~)	NOT (used only before terms with their specifiers)
(4) parentheses	

```
┌──────────────────────────────────────────┐
│                ATTENTION                 │
├──────────────────────────────────────────┤
│I learned that animal is a(n) creature    │
│Note added at 3                           │
└──────────────────────────────────────────┘
<<>>
<<>>
SEARCHDEMO:3   ?:<frog>
<?:<frog>>
┌──────────────────────────────────────────┐
│   |NONE   |include negative results with searches
│   |       |
└──────────────────────────────────────────┘
┌──────────────────────────────────────────┐
│RESULT for frog, frog/animal              │
├──────────────────────────────────────────┤
│1–2                                       │
└──────────────────────────────────────────┘
<<>>
SEARCHDEMO:3   ?:<#animal>
<?:<#animal>>
┌──────────────────────────────────────────┐
│   |NONE   |include negative results with searches
│   |       |
└──────────────────────────────────────────┘
┌──────────────────────────────────────────┐
│RESULT for frog, frog/animal, toad/animal │
├──────────────────────────────────────────┤
│1–3                                       │
└──────────────────────────────────────────┘
<<>>
SEARCHDEMO:3   ?:<##creature>
<?:<##creature>>
┌──────────────────────────────────────────┐
│   |NONE   |include negative results with searches
│   |       |
└──────────────────────────────────────────┘
┌──────────────────────────────────────────┐
│RESULT for frog, frog/animal, toad/animal │
├──────────────────────────────────────────┤
│1–3                                       │
└──────────────────────────────────────────┘
<<>>
SEARCHDEMO:3   ?:Lupu
<?:Lupu>
┌──────────────────────────────────────────┐
│   |NONE   |include negative results with searches
│   |       |
└──────────────────────────────────────────┘
┌──────────────────────────────────────────┐
│RESULT for Lupu                           │
├──────────────────────────────────────────┤
│1–2                                       │
└──────────────────────────────────────────┘
<<>>
SEARCHDEMO:3
```

PLATE 2.87: Searching for keywords, tags, metatags (knowledge), and text words. Notice the knowledge definition: animal=creature.

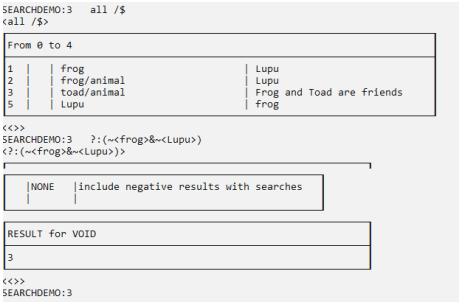


PLATE 2.88: Search with logical operators.

2.8.1.2.2.4. Qualifiers

Qualifiers are attached to individual terms in order to add further restrictions on the query.

2.8.1.2.2.4.1. The Qualifier Phrase

The Qualifier Phrase enclosed within double quotation marks, and affixed directly after the individual search term, with no blank space. The term consists in one or more qualifiers separated by exclamation marks.

Qualifiers may take either no value, a single integer value, a single string value, or a single range of value. The following are examples of valid search expressions with qualifiers:

1) ?:frog“count=1” | Search for notes in which the word “frog” appears once.

2) ?:<toad>“depth=1/3” | Search for notes whose index has a depth not less than
| and not greater than 3

3) ?:*ism”date=2020-1/2020-4!depth=4/5”

When the qualifier takes a range, then either the lower and upper value can be omitted.

2.8.1.2.2.4.2. List of the different qualifiers.

2.8.1.2.2.4.2.1. Metadata qualifiers.

The following qualifiers concern the metadata in the note. If the metadata is deficient, the qualifier will not be imposed.

date=BEFORE/AFTER | excludes notes composed earlier or later

user=USERNAME | only returns notes by USER

size=LESS/MORE | excludes note outside the size range

2.8.1.2.2.4.2.2. Index qualifiers

The following qualifiers concern the index of the note.

index=LEAST/ GREATES | exclude notes outside the given range

depth=LEAST/GREATEST	Exclude notes with indexes whose depth is
	greater or less than the given value
slice=LOWER/UPPER	
must	select the "must"-mode for slicing

2.8.1.2.2.4.2.3. Slicing

The "slice"-qualifier can be used to restrict the search results to notes whose indexes are contained "within" the upper and lower slice. A slice consists in of values, separated by PERIODS. Every value may either by be empty, or consist in either an integer, for the value at the head of the slice, or a natural number. Or in other words: a slice is an index, with some, but not all, of the elements removed.

Valid slices include: "1.2.3.4.5.6", "…5…","-10.1.1.1.1."

A given Index is "within" a lower and upper bound iff each element of the index is not less than equally-ranked element of the lower bound and not greater than the equally-ranked element of the upper bound. If a bound is not given, whether because either the upper or lower bound has not been defined or because ranked-element is missing, then the condition is automatically satisfied.

If the "must"-mode is selected, then the index *must* have elements of rank equal to all the rank of all the elements of the lower and upper bound. Or in other words, if an index is not as "deep" as the greatest depth of the upper or lower bound, then the index will not be "within" the slice.

2.8.1.2.2.4.2.3.1. Examples of slices.

1.1.1 is within "slice=1.1.1/2.2.2"

3.3.3 is not within "slice=/.2"

10.1^10 is within "slice=/.1.1.1.1.1.1.1.1.1.1"

10.1^10.2 is not within "slice=/.1.1.1.1.1.1.1.1.1.1.1.1"
10.1 is within "slice=10.1.1/12.1.1" in the "must"-mode.

10.1 is not within "slice=10.1.1/12.1.1" if not in the "must"-mode.

2.8.1.2.2.4.2.4. Count qualifiers.

The following qualifier can be used to exclude notes where a single search term appear in the text less than or more than a certain number of instances.

count=LESSTHAN/GREATERTHAN

The qualifier "strict" can be used to count *by* whole words. The result is more correct, but it is also a bit slower, since it must iterate over the Cartesian product of the punctuation marks.

2.8.1.2.2.4.2.4.1. Oddities of the count function

The "count"-function operates through two steps. First, the notes are identified in which a given word appears, according to the index, and, secondly, the occurrence of the search term is "counted" in the text of each of the resulting notes. For this reason, however, it can be a bit quirky. First, it cannot count words that are not automatically indexed. Second, even if the "strict"-mode is not selected, it will only consider notes in which the whole word appears at least once.

2.8.1.2.2.4.3. Combining qualifiers with other search features.

While qualifiers can only be attached to a single, atomic term – they cannot be, say, affixed to an expression in parentheses --, they can be used with every valid kind of search term, included negative terms, and impose to restrictions on the logical expressiveness of the search phrase.

2.8.1.2.2.5. Reserved Search Terms.

The search term _ALLNOTES_ is reserved, and can be used to retrieve all notes within the scope defined by the limit.

2.8.1.2.2.6. globalsearch

The command globalsearch can be used to search over all active notebook. An active notebook is a notebook that has been opened during the current session.

2.8.1.2.2.7. Searching over a range of notebooks

To search over a range of active notebooks, enclose a list of the notebooks, separated by commas, with curly braces and place it right before the search phrase.

```
search:{HEIDEGGER,HUSSERL}being

        | Search for the term "being" inn the notebooks HEIDEGGER and HUSSERL
```

2.8.2. Histograms

Histograms display a frequency chart for keywords or words in the text. To produce a histogram, simply use the histogram command, with a collection of indexes as the value. The modifier /$ is used to choose between text words and keywords.

2.8.3. Chronograms

Notesccription allows you to represent the keys in a collection of notes by date and time of composition or modification. The chronogram feature is very flexible, and sorts according to year, month, day, or even hour, in various combinations. It also allows you to choose between using the first date, the newest date, or all the dates associated with the notes in the index range, and also reveal or suppress the indexes associates with the keys.

2.8.3.1. Initiating a chronogram

To initiate a chronogram, use the constdates (alternative form: constitutedates) command, which accepts three values and the modifiers /$, /&, /* and /?. The first value consists in a collection of indexes, the second value the determinant, and the third value in either f(irst), n(ewest), or a(ll).

The modifiers /$ and /& can be used to define the determinant, respectively, as ym (year+month) or ymd (year+month+day) , in lieu of the second value. The modifier /* is used to show indexes, while /? Explicitly ask which dates are to be used, and thus can be substituted for a third value.

Consider the following commands:

(1) constdates:1-100;ymd*h;f /*

Create a date dictionary for indexes 1 to 100, with ymd*h as the determinant, using the date when the note was first entered, and displaying indexes.

(2) constdates:1-1000;ym /?

Create a date dictionary for indexes 1 to 1000, using ym as a determinant, and querying whether to use the first, newest, or all dates.

```
<?:<Aquinas> =>histiogram:? /$>
histiogram:? /$
```

	NONE	include negative results with searches

RESULT for Aquinas
47–49, 64, 1813

```
<<Plato, Statesman/text, TEACHING, page@?, steph@?>>
defaultnotebook:9
<>
```

CONCORDANCE	
1: Aquinas	47–49, 64, 1813
2: Being and Time/text	1813
3: Heidegger/20th centu ...	1813
4: Heidegger/German	1813
5: Heidegger/phenomenol ...	1813
6: Heidegger/philosophe ...	1813
7: Summa_Theologica	47–49, 64
8: emergence	47
9: language	48
10: sex	49

```
<<Plato, Statesman/text, TEACHING, page@?, steph@?>>
defaultnotebook:10
```

PLATE 2.89: A simple histogram.

2.8.3.2. The determinant

The determinant indicates what divisions of time (year=y, month=m, day=d, hour=h, minute=m, second=s, microsecond=x) will be used to organize the date dictionary.

The maximum determinant is ymd*hmsx. Any subset of the maximum determinant, with the star included, is a valid determinant. Hence valid determinants include:

> *h, *hm, *hms, *hmsx, *hx, *m, *ms, *mx, *s, *sx, *x, d, d*h, d*h, d*hm, d*hms, d*hmsx, d*hx, d*m, d*ms, d*mx, d*s, d*sx, d*x, m, m*h, m*h, m*hm, m*hms, m*hmsx, m*hx, m*m, m*ms, m*mx, m*s, m*sx, m*x, md, md*h, md*h, md*hm, md*hms, md*hmsx, md*hx, md*m, md*ms, md*mx, md*s, md*sx, md*x, y, y*h, y*h, y*hm, y*hms, y*hmsx, y*hx, y*m, y*ms, y*mx, y*s, y*sx, y*x, yd, yd*h, yd*h, yd*hm, yd*hms, yd*hmsx, yd*hx, yd*m, yd*ms, yd*mx, yd*s, yd*sx, yd*x, ym, ym*h, ym*h, ym*hm, ym*hms, ym*hmsx, ym*hx, ym*m, ym*ms, ym*mx, ym*s, ym*sx, ym*x, ymd, ymd*h, ymd*h, ymd*hm, ymd*hms, ymd*hmsx, ymd*hx, ymd*m, ymd*ms, ymd*mx, ymd*s, ymd*sx, ymd*x

Of course, not all of these are useful. Most, indeed, are absolutely pointless.

2.8.3.2.1. Showing and setting the determinant

The determinant can always be entered explicitly when initiating a chronogram. If no determinant is indicated either through a value or a modifier, then *ARCADES* will use the default determinant.

The default determinant is preset to "ym" – year and month – but it can be changed by using the changedet command.

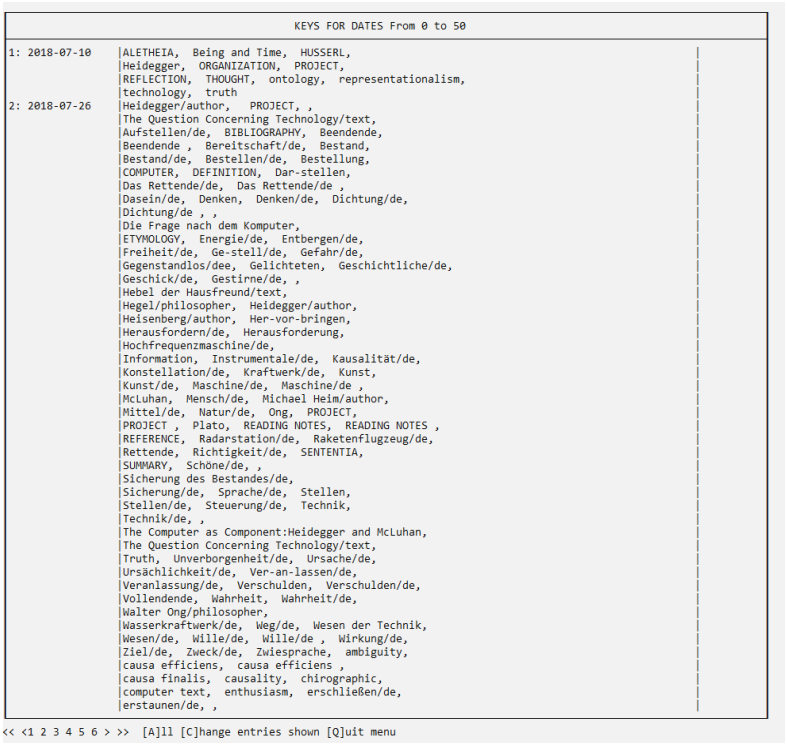
The current determinant can be displayed with showdet.

2.8.3.2.2. Displaying an existing chronogram

Chrongrams are stored in a dictionary using the determinant as key. You can use the command showdatedict to display an existing chronogram. This accepts the determinant to be displayed as a value, and the modifiers /$, /&, /*, /?, /=, which are used to add, respectively, year, month, day, hour, and minute. The minute can only be added after an hour.

2.8.3.2.3. Purging keys from the chronogram

Because the chronogram displays keys, it may be desirable not to have it show every single key, since otherwise it will become excessively large and, consequently, not very useful. *ARCADES* thus allows you to display a chonogram with selected keywords, or types of keywords, purged from it.

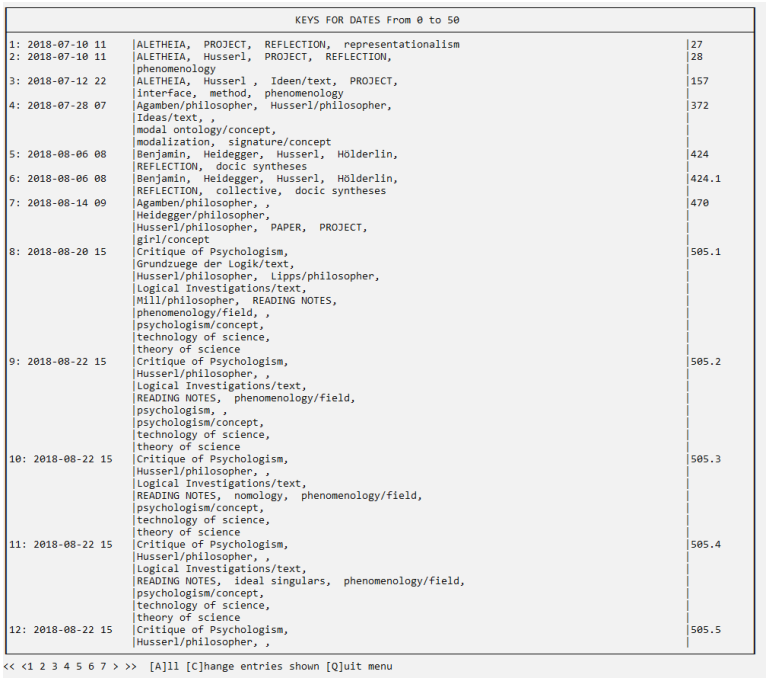


KEYS FOR DATES From 0 to 50

1: 2018-07-10	ALETHEIA, Being and Time, HUSSERL, Heidegger, ORGANIZATION, PROJECT, REFLECTION, THOUGHT, ontology, representationalism, technology, truth
2: 2018-07-26	Heidegger/author, PROJECT, , The Question Concerning Technology/text, Aufstellen/de, BIBLIOGRAPHY, Beendende, Beendende , Bereitschaft/de, Bestand, Bestand/de, Bestellen/de, Bestellung, COMPUTER, DEFINITION, Dar-stellen, Das Rettende/de, Das Rettende/de , Dasein/de, Denken, Denken/de, Dichtung/de, Dichtung/de , , Die Frage nach dem Komputer, ETYMOLOGY, Energie/de, Entbergen/de, Freiheit/de, Ge-stell/de, Gefahr/de, Gegenstandlos/dee, Gelichteten, Geschichtliche/de, Geschick/de, Gestirne/de, , Hebel der Hausfreund/text, Hegel/philosopher, Heidegger/author, Heisenberg/author, Her-vor-bringen, Herausfordern/de, Herausforderung, Hochfrequenzmaschine/de, Information, Instrumentale/de, Kausalität/de, Konstellation/de, Kraftwerk/de, Kunst, Kunst/de, Maschine/de, Maschine/de , McLuhan, Mensch/de, Michael Heim/author, Mittel/de, Natur/de, Ong, PROJECT, PROJECT , Plato, READING NOTES, READING NOTES , REFERENCE, Radarstation/de, Raketenflugzeug/de, Rettende, Richtigkeit/de, SENTENTIA, SUMMARY, Schöne/de, , Sicherung des Bestandes/de, Sicherung/de, Sprache/de, Stellen, Stellen/de, Steuerung/de, Technik, Technik/de, , The Computer as Component:Heidegger and McLuhan, The Question Concerning Technology/text, Truth, Unverborgenheit/de, Ursache/de, Ursächlichkeit/de, Ver-an-lassen/de, Veranlassung/de, Verschulden, Verschulden/de, Vollendende, Wahrheit, Wahrheit/de, Walter Ong/philosopher, Wasserkraftwerk/de, Weg/de, Wesen der Technik, Wesen/de, Wille/de, Wille/de , Wirkung/de, Ziel/de, Zweck/de, Zwiesprache, ambiguity, causa efficiens, causa efficiens , causa finalis, causality, chirographic, computer text, enthusiasm, erschließen/de, erstaunen/de, ,

<< <1 2 3 4 5 6 > >> [A]ll [C]hange entries shown [Q]uit menu

PLATE 2.90: defaultnotebook:1 ?:Heidegger =>constdates:?;ymd



KEYS FOR DATES From 0 to 50

1: 2018-07-10 11	ALETHEIA, PROJECT, REFLECTION, representationalism	27
2: 2018-07-10 11	ALETHEIA, Husserl, PROJECT, REFLECTION, phenomenology	28
3: 2018-07-12 22	ALETHEIA, Husserl , Ideen/text, PROJECT, interface, method, phenomenology	157
4: 2018-07-28 07	Agamben/philosopher, Husserl/philosopher, Ideas/text, , modal ontology/concept, modalization, signature/concept	372
5: 2018-08-06 08	Benjamin, Heidegger, Husserl, Hölderlin, REFLECTION, docic syntheses	424
6: 2018-08-06 08	Benjamin, Heidegger, Husserl, Hölderlin, REFLECTION, collective, docic syntheses	424.1
7: 2018-08-14 09	Agamben/philosopher, , Heidegger/philosopher, Husserl/philosopher, PAPER, PROJECT, girl/concept	470
8: 2018-08-20 15	Critique of Psychologism, Grundzuege der Logik/text, Husserl/philosopher, Lipps/philosopher, Logical Investigations/text, Mill/philosopher, READING NOTES, phenomenology/field, , psychologism/concept, technology of science, theory of science	505.1
9: 2018-08-22 15	Critique of Psychologism, Husserl/philosopher, , Logical Investigations/text, READING NOTES, phenomenology/field, psychologism, , psychologism/concept, technology of science, theory of science	505.2
10: 2018-08-22 15	Critique of Psychologism, Husserl/philosopher, , Logical Investigations/text, READING NOTES, nomology, phenomenology/field, psychologism/concept, technology of science, theory of science	505.3
11: 2018-08-22 15	Critique of Psychologism, Husserl/philosopher, , Logical Investigations/text, READING NOTES, ideal singulars, phenomenology/field, psychologism/concept, technology of science, theory of science	505.4
12: 2018-08-22 15	Critique of Psychologism, Husserl/philosopher, ,	505.5

<< <1 2 3 4 5 6 7 > >> [A]ll [C]hange entries shown [Q]uit menu

PLATE 2.91: defaultnotebook:4 ?:HUSSERL =>constdates:?;ymd*h /*

KEYS FOR DATES From 0 to 50

1: 2018-07-10	ALETHEIA, HUSSERL, Heidegger, PROJECT, REFLECTION	26
2: 2018-07-10	ALETHEIA, Husserl, PROJECT, REFLECTION, phenomenology	28
3: 2018-07-28	Agamben/philosopher, Husserl/philosopher, Ideas/text, , modal ontology/concept, modalization, signature/concept	372
4: 2018-08-06	Benjamin, Heidegger, Husserl, Hölderlin, REFLECTION, docic syntheses	424
5: 2018-08-06	Benjamin, Heidegger, Husserl, Hölderlin, REFLECTION, collective, docic syntheses	424.1
6: 2018-08-14	Agamben/philosopher, , Heidegger/philosopher, Husserl/philosopher, PAPER, PROJECT, girl/concept	470
7: 2018-08-20	Critique of Psychologism, Husserl/philosopher, , Logical Investigations/text, READING NOTES, phenomenology/field, psychologism/concept, technology of science, theory of science	505
8: 2018-08-20	Critique of Psychologism, Husserl/philosopher, , Logical Investigations/text, READING NOTES, phenomenology/field, psychologism/concept, technology of science, theory of science	505.1
9: 2018-08-20	Critique of Psychologism, Grundzuege der Logik/text, Husserl/philosopher, Lipps/philosopher, Logical Investigations/text, Mill/philosopher, READING NOTES, phenomenology/field, , psychologism/concept, technology of science, theory of science	505.1
10: 2018-08-20	Critique of Psychologism, Grundzuege der Logik/text, Husserl/philosopher, Lipps/philosopher, Logical Investigations/text, Mill/philosopher, READING NOTES, phenomenology/field, , psychologism/concept, technology of science, theory of science	505.1
11: 2018-08-20	Critique of Psychologism, Grundzuege der Logik/text,	505.1

<< <1 2 3 4 5 6 7 8 9 10 > >> [A]ll [C]hange entries shown [Q]uit menu

PLATE 2.92: ?:<HUSSERL> =>constdates:?;ymd /*

2.8.3.2.3.1. Showing a purged chronogram

This is done using the showdatedictpurge command, which takes two values and the modifiers /$ /& /* and /?.

The first value is the determinant, which functions as with showdatedict. The second value is the purge sequence. The modifier /? Is used to query for a purge sequence. The syntax of the purge sequence is explained below. Keep in mind, however, that if the purge sequence is entered through showdatedict, then the modifiers cannot be used.

2.8.3.2.3.2. The purge sequence

With the setpurgekeys, it is possible to instruct *ARCADES* to purge specific categories of keys as well as certain individual keys.

The command setpurgekeys accepts one value—a purge instruction-- and all five modifiers.

The purge instruction has the following syntax:

SPEC|KEY1,KEY2,KEY3…

SPEC consists in one or several of the following symbols, each of which is equivalent to a modifier:

a[ll caps]	/$
u[pper case]	/&
l[lower case]	/*
s[eqeunces]	/?
n[umbers]	/=

KEY is a valid search phrases, and yields all the applicable keywords.

So for example: *a* would yield all the keywords containing "a" in the middle.

2.8.3.2.3.3. Other purge-related commands

The following related commands are also available:

clearpurekeys	to clear all the purge keys.
showpurgekeys	to show the purge keys

2.8.3.2.4. Displaying active determinants

You can use the command activedet or actdet to display the determinants for which chronograms currently exist.

2.8.4. Clustering

Clustering allows you to organize a collection of notes into groups with "connected" keywords. Keywords are "connected" if they either belong to the same note, or are both "connected" to a third note, with this definition applied recursively to yield an exhaustive system of "connections" for every note. The relation of "connection" is reflective, symmetrical, transitive, and substitutive. The system of "connections" is called a cluster, and every note within a cluster is connected with every other note within it.

2.8.4.1. Monolithic and variegated notebooks

The usefulness of clustering depends on the structure of the notebook, and the keywords used. If keywords are assigned too liberally, then the clusters will become so large as too be useless. A notebook that clusters nicely is "variegated"; a notebook that clusters poorly is "monolithic."

```
PURGEDEMO:8   all /$
<all /$>
```

From 0 to 7			
2	|	/C	Welcome to NOTESCRIPTION!
3	|	frog, STUFFED ANIMAL, Lupu	
4	|	STUFFED ANIMAL, giraffe	
5	|	Yonsei, INSTITUTION	
6	|	Pyeol, cat, REAL ANIMAL	
7	|	Dal, cat, REAL ANIMAL	
8	|	Sully, cat, REAL ANIMAL	

```
<<>>
PURGEDEMO:8   constdates:
```

Determinants
, *h, *h, *hm, *hms, *hmsx, *hx, *m, *ms, *mx, *s, *sx, *x, d, d*h, d*h, d*hm, d*hms, d*hmsx, d*hx, d*m, d*ms, d*mx, d*s, d*sx, d*x, m, m*h, m*h, m*hm, m*hms, m*hmsx, m*hx, m*m, m*ms, m*mx, m*s, m*sx, m*x, md, md*h, md*h, md*hm, md*hms, md*hmsx, md*hx, md*m, md*ms, md*mx, md*s, md*sx, md*x, y, y*h, y*h, y*hm, y*hms, y*hmsx, y*hx, y*m, y*ms, y*mx, y*s, y*sx, y*x, yd, yd*h, yd*h, yd*hm, yd*hms, yd*hmsx, yd*hx, yd*m, yd*ms, yd*mx, yd*s, yd*sx, yd*x, ym, ym*h, ym*h, ym*hm, ym*hms, ym*hmsx, ym*hx, ym*m, ym*ms, ym*mx, ym*s, ym*sx, ym*x, ymd, ymd*h, ymd*h, ymd*hm, ymd*hms, ymd*hmsx, ymd*hx, ymd*m, ymd*ms, ymd*mx, ymd*s, ymd*sx, ymd*x

KEYS FOR DATES		
1: 2019-05-05	|/C, Dal, Girin, INSTITUTION, Lupu, Pyeol,	|
	|REAL ANIMAL, STUFFED ANIMAL, Yonsei,	|
	|cat, frog, giraffe, university	|

```
<<>>
PURGEDEMO:8
```

Purgekeys settings
ALL CAPS: False CAPS: False LOWER:True numbers: False sequences: False SETS:0

```
<<>>
PURGEDEMO:8   showdatedictpurge:ymd
<showdatedictpurge:ymd>
```

KEYS FOR DATES		
1: 2019-05-05	|/C, Dal, Girin, INSTITUTION, Lupu, Pyeol,	|
	|REAL ANIMAL, STUFFED ANIMAL, Yonsei	|

```
<<>>
PURGEDEMO:8
```

PLATE 2.93: The chronogram is first constituted using constdatedict, and then the chronogram is displayed purging lowercase keys.

```
PURGEDEMO:8   setpurgekeys:ul
<setpurgekeys:ul>
```

Purgekeys settings
ALL CAPS: False CAPS: True LOWER:True numbers: False sequences: False SETS:0

```
<<>>
PURGEDEMO:8   showdatedictpurge:ymd
<showdatedictpurge:ymd>
```

KEYS FOR DATES
1: 2019-05-05 |/C, INSTITUTION, REAL ANIMAL, STUFFED ANIMAL |

```
<<>>
PURGEDEMO:8
```

KEYS FOR DATES
1: 2019-05-05 |/C, INSTITUTION, REAL ANIMAL, STUFFED ANIMAL |

```
<<>>
PURGEDEMO:8   setpurgekeys:|INSTITUTION,REAL ANIMAL
<setpurgekeys:|INSTITUTION,REAL ANIMAL>
```

Purgekeys settings
ALL CAPS: False CAPS: False LOWER:False numbers: False sequences: False SETS:6

```
<<>>
PURGEDEMO:8   showdatedictpurge:ymd
<showdatedictpurge:ymd>
```

KEYS FOR DATES
1: 2019-05-05 |/C, Dal, Girin, Lupu, Pyeol, STUFFED ANIMAL, |Yonsei, cat, frog, giraffe, university |

```
<<>>
PURGEDEMO:8
```

PLATE 2.94: And then the same chronogram is displayed with capitalized and lowercase keywords purged, and then eliminating the keywords INSTITUTION and REAL ANIMAL.

In order to make monolithic notebooks seem less monolithic, and hence more amenable to clustering, *ARCADES* allows you to 1) use only a certain number of the least frequent keys and 2) purge keys from the keysets. These two operations can be combined. The reason for favoring the least frequent keys over the most frequent is obvious: the least frequent keys are more likely to be have a specific "differential" value in relation to other keys in the notebook, and hence to be semantically meaningful.

2.8.4.2. Creating clusters

To cluster a collection of indexes, use the cluster command. The cluster command accepts two values, and the modifiers /$ and /&. The first value is the collection of notes which you wish to cluster, while the second value is the number of least-frequent keys you wish to use. If nothing is entered for the second value, then *ARCADES* will use all keyword.

The modifiers have the following meaning:

(1) /$ | "Iterates over" the cluster

(2) /& | To use the purge sequence defined for the chronogram (see 2.8.3.2.3.2)

2.8.4.3. The difference between the two purge functions

The main advantage of using the "native" purge function is to avoid having to redefine the chronogram purge function. But there also are slight differences in functionality. The "cluster" purge function is able to handle the German β, and it lacks the capacity to purge sequences and numbers, as these are unlikely to interfere with the clustering.

```
<<>>
CLUSTERDEMO:2
<>
```

3		Lupu, girin

```
<<>>
CLUSTERDEMO:3   all /$
<all /$>
```

From 0 to 9			
2		/C	Welcome to NOTESCRIPTION!
3		Lupu, girin	
4		Giraffe, girin	
5		tall, Giraffe	
6		University, Yonsei	
7		University, Korea University	
8		University, Princeton	
9		University, Ivy League	
10		Dartmouth, Ivy League	

```
<<>>
CLUSTERDEMO:3   cluster:2-10
<cluster:2-10>
```

CLUSTER #1
Giraffe, Lupu, girin, tall

CLUSTER #2
Dartmouth, Harvard, Ivy League, Korea University, Princeton, University, Yonsei

```
<<>>
CLUSTERDEMO:4
```

PLATE 2.95 Clustering

2.8.4.4. Defining the "cluster" purge function

The purge function for clustering cannot be defined during a function call, but only by using the cpara command, which accepts one value – a subset of the string "acl," according as you wish to purge all-cap words, capitalized words, or lower case words – and the modifiers /$, /&, and /*, which correspond, respectively, to these three different kinds of purges. Hence, to purge keywords that are in all caps, you could enter either:

(1) cpara:a

(2) cpara /$

2.8.4.5. Iterator over clusters

If you use the /$ modifier while initiating clustering, then *ARCADES* will create a series of iterators corresponding to the clusters, making it possible to advance from iterator to iterator – from cluster to cluster, in other words – resetting the iterator each time. To advance to the next iterator, simply enter a semicolon (;).

2.8.4.5.1. Killing clusters

The killclusters command can be used to eliminate the clustered iterators.

2.8.5. The fetch command.

The fetch command is used to select a group of notes directly by index.

2.8.5.1. The syntax of the fetch command.

The fetch command accepts a single value: the "fetch" phrase

The "fetch"-phrase, like them "search"-phrase, consists in atomic elements joined together by logical symbols and parentheses.

The logical symbols include:

| for logical union

& for logical intersection

2.8.5.2 The atomic elements.

The atomic elements of the fetch phrase consists of the following.

(1) one or more ranges of notes separated by parentheses.

(2) MARKED | For all marked noted

UNMARKED | For all unmarked notes

NEG | For negative-valued notes

POS | For positive-valued noted

ALL | For all notes

FLIPBOOK [or FB] | For the notes in the flipbook

(3) The name of a project

(4) The name of a variable.

2.8.5.3. Examples of fetch commands.

fetch:1-100 =Fetch notes with index values from 1 to 100.

fetch: NEG | (10-20 | 30-40) =Fetch negative valued notes, and noted from 10-20 or 30-40

fetch:ARISTOTLE & PLATO =Fetch notes belonging to both projects ARISTOTLE and PLATO

Observe that no distinction is made between variables and projects. Hence you should avoid using the names of projects (which are permanent parts of the notebook) as the names of variables.

2.8.5.4. Using fetch with search and sort.

The fetch command is most useful in conjunction with other commands, such as search or sort.

For example:

fetch:1-100 =>search:pleasure;? |Searches over the domain 1-100.

fetch:1-100 =>sort:?;becker |Sorts notes with indexes 1-100 by "becker" sequence key.

fetch:1-100 =>search:pleasure;? =>sort:?;becker =>multi:?

|Combining fetch, search, sort, multi

2.8.6 The sort command.

The sort command orders a group of notes according to a series of criteria. The sort command accepts two values: a range of notes, and the "sort"-phrase.

2.8.6.1. The "sort"-phrase.

The "sort"-phrase consists in one or more criteria, separated by _.

sort:CRITERION1_CRITERION2_CRITERION3_etc.

The criteria are applied to "sort" the notes in sequence. When sorting any establishing the ordering-relation between any two notes, the second criterion will only be applied if the first doesn't distinguish between them, and so on.

2.8.6.2. The "sort"-criterion.

The following are valid sort-criteria, which are used to determined a "sort-value."

(1) A sequence keyword

(2) DATE, USER, SIZE, KEYCOUNT, INDEX, TEXTLENGTH

(3) A parenthetical search phrase of the form: SPECIFIER#VAL1,VAL2,VAL3,VAL4…

A sequence keyword sorts according to the value attached to the keyword.

DATE, USER, SIZE sort according to metadata of the note.

KEYCOUNT sorts according to the total number of keywords.

INDEX sorts according to the value of the index.

TEXTLENGTH sorts according to the total length of the text in the note.

2.8.6.2.1. Parenthetical search phrases.

The SPECIFIER consists in one of the following:

K [=keyword]

KW[=keyword weighted]

T [=textword]

TC [=textword count]

TW [=textword weighted]

TCW [=textword count weighted]

Each VAL consists in a keyword or textword, and an option floating point number, which signifies the "weight" accorded to that element.

Consider the following examples:

K#ontology, substance, essence, ousia	sorts the notes according to the value determined by the number of these keywords found.
KW#ontology*5.0, metaphysics*3.0, essence *1.0, being*.5	kreyword search with different weights attached to the elements
TC#being,essence,substance	The sort value is the total number of these words found in the text.

2.8.6.3. Examples of sort commands.

sort:?;title_volume_book_chapter	Sorts by the values of the given sequence keys.

sort?;title_(KW#ontology*10.0,entity*5.0,being*2.0)_page

sort?:title_DATE

2.8.6.4. sort with search or fetch.

The results of a search or a fetch can be fed into a sort.

2.9. PROJECTS

ARCADES offer a simple yet powerful means of juggling multiple tasks while working in a single notebook. Say, for example, that you are simultaneously taking notes related to several different projects that you are working on. If you just add notes to your notebooks related to these various projects in whatever order you please, then, unless you invest a great deal of effort in keeping track of what is going on, you are likely to end up with a disorganized and largely useless notebook. Moreover, default keys will be pretty much useless, since they will have to be constantly changed, leading to errors – with the default keys carried over by accident to projects where they don't belong. Hence, you will be forced to add the keywords with each new note, leading, in turn, to inconsistencies in the keyword assignment. On the other hand, if you just use separate notebooks for different projects, then you will forego the advantages of having notes from different projects brought together in the same place. You will not be able to perform global searches or establish links between different notes from different projects. Nor will you be able to glean an overarching view of how your different projects are evolving.

2.9.1 The basic elements of a project

A "project" in *ARCADES* stores some basic information about the program-state and keeps track of the indexes of notes that have been added to the project. Finally, when a project has been activated and is in use, each new note will include, as sequence keyword, the name of the project together with a value indicating the order in which the note has been entered, with the initial value determined by the user.

Projects are stored in a special dictionary, which is a permanent attribute of the notebook. The name of the project serves as the identifying key. For each project, the following attributes are kept track of and stored:

1) The default keys, which are automatically included when the project is active.

2) The current position in the notebook.

3) The current entry mode, such as "connext" or "conchild"

4) A list of all the indexes that have been added to the notebook.

5) The date when the project began.

6) The date that it was last modified.

7) Whether it is currently open.

2.9.2. Project commands

Commands for using projects include: newproject, saveproject, showprojects, resumeproject, loadproject, endproject, flipproject, crrrentproject, archiveproject, unarchiveproject, showarchivedrpojects, deletearchivedrpoject, loadproject, dumpproject, renameproject.

2.9.2.1. newproject

The command newproject is used to initiate a new project The name of the new project is entered as a value. If, however, a project with this name already exists, and the project name is alphabetic, not already ending with a number, then a 1 will be suffixed to the project name. If the project name already has a numerical suffix, then the existing numerical suffix will be increased by 1.

So for example: HEIDEGGER > HEIDEGGER1 > HEIDEGGER2 > HEIDEGGER3

2.9.2.2. showprojects

To display all the projects that are currently attached to the notebook, type showprojects.

```
<<>>
defaultnotebook:1   showprojects
<showprojects>
```

From 0 to 19

#	PROJECTNAME	INDEX	KEYS
1	BENJAMIN	3322	[ ]/3247:3322
2	FICHTECLOSED	8	[Closed Commercial State, Fichte]/4015:197
3	FROMHYPERION	6	[ ]/3246:3246.8
4	HEGELHIST	3234	[1822-23/version, Hegel, INTRODUCTION, Ph ...]/
5	HEGELRIGHT	3	[Hegel, Philosophy of Right/text, READING ...]/4017:4017.2.10
6	IDEASFORPAPERS	8	[BIBLIOGRAPHY, Kant, Prolegomena/text, me ...]/145
7	KANTFOUNDATIONS	35	[Groundwork of the Metaphysics of Morals/ ...]/3248:3346.11.15.4
8	KANTPEACE	2556.13.1	[Kant, Perpetual Peace/text]/
9	KANTPROL	3161	[BIBLIOGRAPHY, Kant, Prolegomena/text, me ...]/
10	KOREAN	12.1	[section@1]/3338
11	PLATOREPUB	4017.1	[Plato, Republic, book@7.0]/141:197
12	POLITSEM	4.2	[1822-23/version, Hegel, INTRODUCTION, Ph ...]/
13	QUALTEL	4016.1.7	[PAPER, PROJECT, Postmodern Culture/journ ...]/4016:4016.1.8
14	QUORA	3293	[ ]/3244:3244.6
15	TRUTH	1208	[Kant, Perpetual Peace/text]/
16	YONSEI	3337.2	[Plato, Republic, book@7.0]/4013:4014.1.1.1
17	dreams	6	[ ]/3245

```
<<>>
defaultnotebook:1
```

PLATE 2.96: Displaying all the projects.

2.9.2.3. resumeproject

To resume a project that already exists, type resumeproject, with a single project or list of projects as a value. If a project is already active, this will be saved before a new project is activated. You will be asked if you wish to update the defaultkeys in the project. This command also accepts to modifiers: /$ and /&. Use the first of these to add a second project to an existing project, and the second if you do not wish to automatically update the position and the entrymode.

2.9.2.4. saveproject

To back up the current project, enter saveproject.

2.9.2.5. endproject, quitproject

To quit the current project, enter endproject (alternative form: quitproject).

2.9.2.6. flipproject

The command flipproject takes the indexes of the current project and sends them to the flipbook, allowing you to iterate over them.

2.9.2.7. currentproject

The command currentproject displays the indexes of the current project.

2.9.2.8. showproject

The command showproject can be used to display all the attributes of a project.

2.9.2.9. showprojectdates

The command showprojectdates, which accepts the modifiers /$ /& and /* and two values, can be used to display a chronogram of the projects. The first value can be used to restrict to a range of indexes, while the second value, or the modifiers, is used to set the determinant. The default determinant is "ymd", whereas the first modifier selects "ym"; the second modifier ads '*h', and the third modifier adds an 'm'.

2.9.2.10. archiveproject, unarchiveproject, showarchivedprojects

The commands archiveproject and unarchiveproject can be used to move projects into and out of the archive. To see the archived projects, use showarchivedprojects.

2.9.2.11. renameproject, deletearchivedproject

Use renameproject:OLDNAME;NEWNAME to rename a project, and deletearchivedproject to delete an archived project.

2.9.2.12. dumpproject, loadproject

The command dumproject saves the projects as a textfile, named by the date of the save. The command loadproject loads a textfile and restores the projects from it. It must be used with care, since it will erase the existing projects.

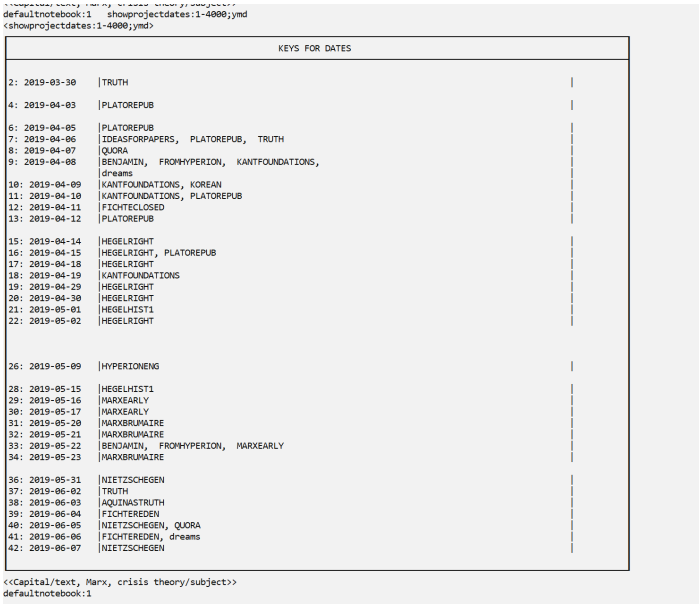


PLATE 2.97 A chronogram of projects

2.9.2.13. Restoring projects.

While it is recommended that you periodically backup projects using dumpprojects, especially if the pickle file is not automatically being backed up through a system that allows you to restore to a previous version, it is also possible to recostruct the projects from the notebook itself. This searches for sequences with a more-or-less continuous series of values, and extracts the rest of the information for the project from this. If you wish to discover and restore all the possible projects in the notebook, use restoreallprojects. To restore a single project, use restoreproject, with the identifier of the sequence as the value. Both commands accept a single modifier, /$, to suppress queries.

2.9.3 Temporarily including projects.

When entering the list of keywords for a note, you can include use +PROJECTNAME to temporarily add a project, and –PROJECTNAME to temporarily remove a project.

2.10 SWITCHING

ARCADES allows you to open several notebooks at once, switching back and forth between them. To switch, you use the switch command. The first time that you use it, it will exit the current notebook, and allow you to enter a new one. To do so, select (N) in the input menu. Once more than one notebook is active, future uses of switch will display the active notebooks, and allow you to switch to a new one, or remain in the old one, just by typing the number corresponding to it.

When you quit a notebook, the notebook will be closed and you will be returned to the "switch"-menu. Here it is possible to quit all the active notebooks, closing *ARCADES*.

2.11. BUFFER

Each active notebook is a unique notebook-object. *ARCADES*, however, also allows you to store notes in a common register shared by all the notebooks.

2.11.1. copyto

Enter copyto, with a collection of indexes as a value, to copy notes into the temporary buffer.

2.11.2. copyfrom

Enter copyfrom, with a single value indicating the number of notes, or using the modifier /$ to indicate all the notes in the temporary buffer.

The temporary buffer functions as a stack -- first in/first out – and notes are deleted as soon as they are "copied" out of it.

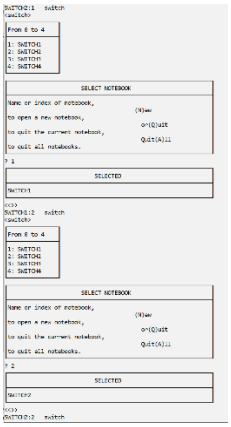


PLATE 2.98: Switching between notebooks.

```
<<>>
COPYTO:3.2   copyto:1-10
<copyto:1-10>
```

ATTENTION
1 copied to temporary buffer! 2 copied to temporary buffer! 3 copied to temporary buffer! 3.1 copied to temporary buffer! 3.2 copied to temporary buffer! 4 copied to temporary buffer! 5 copied to temporary buffer! 6 copied to temporary buffer! 7 copied to temporary buffer! 8 copied to temporary buffer! 9 copied to temporary buffer! 10 copied to temporary buffer!

```
<<>>
COPYTO:5   switch
<switch>
```

PLATE 2.99: Copying to the buffer.

```
COPYFROM:1   copyto:1-10
<copyto:1-10>
```

ATTENTION
2 copied to temporary buffer!

```
<<>>
COPYFROM:2   copyfrom /$
<copyfrom /$>
```

ATTENTION
Note added at 1 Note added at 3 Note added at 4 Note added at 3.1 Note added at 3.2 Note added at 5 Note added at 6 Note added at 7 Note added at 8 Note added at 9 Note added at 10 Note added at 11 Note added at 12 Note added at 13 Note added at 14 Note added at 3.3 Note added at 3.4 Note added at 15 Note added at 16 Note added at 17 Note added at 18 Note added at 19 Note added at 20 Note added at 21 Note added at 22

```
<<>>
COPYFROM:2
```

PLATE 2.100: Copying from the buffer.

2.12. ADVANCED REFEEDING COMMANDS

So as to increase the usefulness of *ARCADES*'s ability to "refeed" the results of commands, a number of advanced commands are available. These allow one to load text files into *ARCADES*, save them back to the hard drive, or even upload a function written into Python and apply it to the text.

2.12.1 load

The command load, which accepts a filename as a value, loads a text file and feeds it into the text command. If no value is entered, then it will call up the file menu.

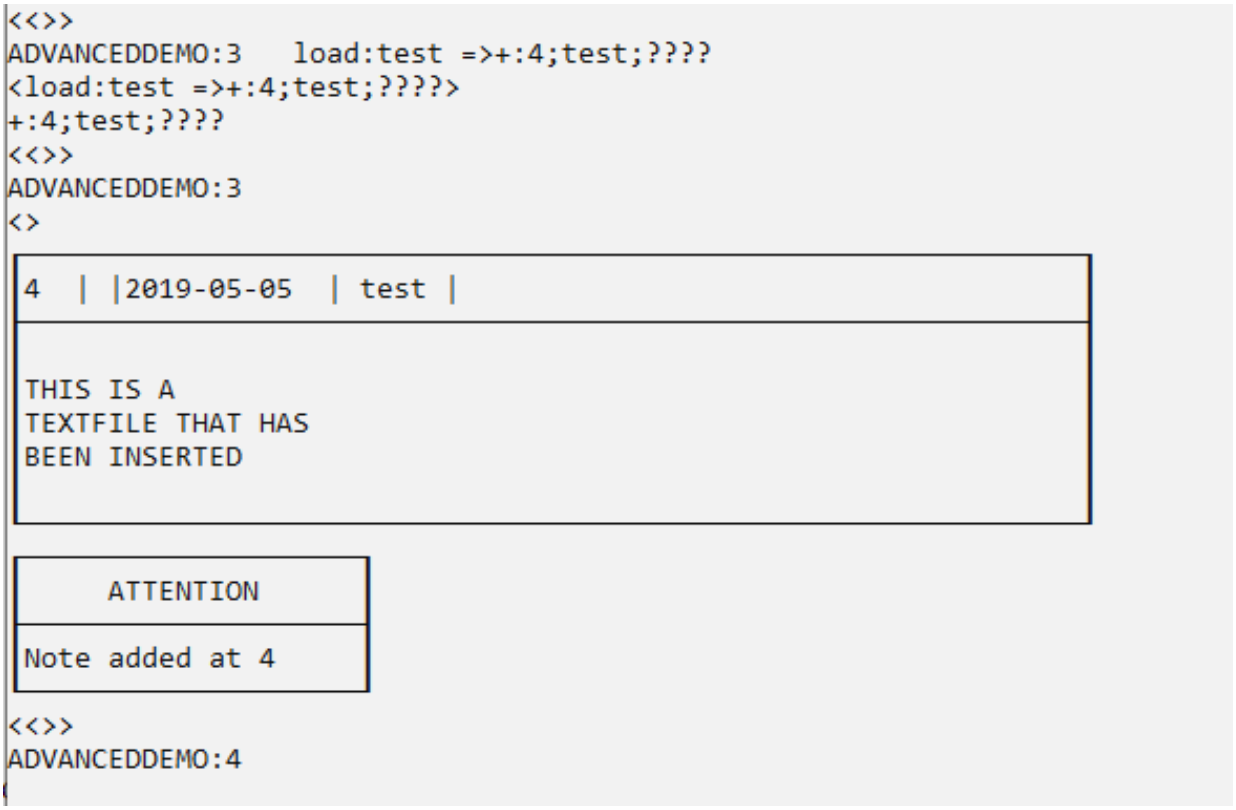


PLATE 2.101: Inserting a text file into a note by loading and refeeding.

2.12.2. save

The save command saves a text file to the disk drive. It accepts three values: the text to be saved, the filename, and the folder, which defaults to '/textfiles.'

The save command can be used both to save the *text* of a note, as well as formatted output, and hence can be combined with a variety of other commands, including show, multi, and also explode.

2.12.3. echo

The command echo simply prints the text in the value. It can be used with loadtext, and is also a useful way of inspecting the output sent to variables.

```
<<>>
ECHODEMO:2    echo:narcissus
<echo:narcissus>
narcissus
<<>>
ECHODEMO:2
```

PLATE 2:102: Echoing.

2.12.4. run

ARCADES allows you to apply user-defined Python scripts to process text. The script in question should be a function, named "generic," that accepts a single string as its value and returns a single string, and it should be saved as a ".py' file in the "/programs" folder. To apply the program to text, simply use the run command, which accepts two values: the name of the program, and the text itself to be processed. The text can be entered directly, but you can also use a variable or the quadruple question marks.

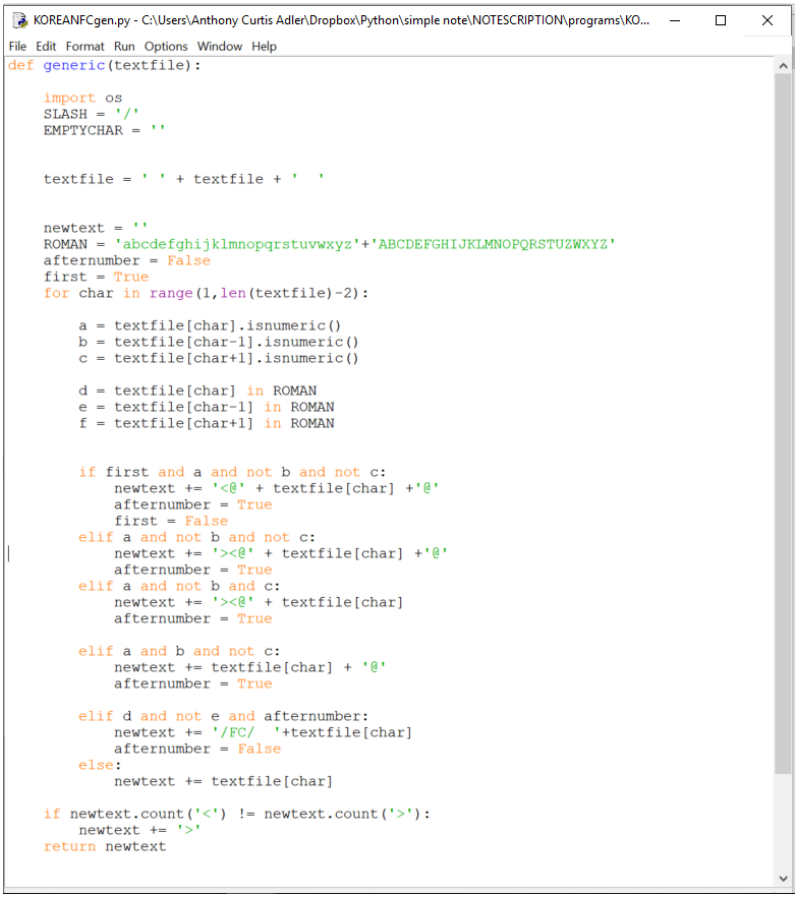


PLATE 2.103: Example of a valid program.

The above (PLATE 2.103) is an example of a program, used to turn a list of Korean vocabulary words, downloaded from the Web, into a flashcard script.

```
1
것
 A thing or  an object
2
 하다
 To do
3
 있다
 To be
4
 수
 way, method, Number
5
 하다
 To do
6
 나
 I
7
 없다
 Do not exist, absent
8
 않다
To not do
9
 사람
 Person
10
 우리
 we, our
```

It yields text formatted like this, which can, in turn, be interpreted by *ARCADES* as two-sided flashcards.[1]

```
<@1@
것
 /FC/  A thing or  an object
><@2@
 하다
 /FC/  To do
><@3@
 있다
 /FC/  To be
><@4@
 수
 /FC/  way, method, Number
><@5@
 하다
 /FC/  To do
><@6@
 나
 /FC/  I
><@7@
 없다
 /FC/  Do not exist, absent
><@8@
 않다
/FC/  To not do
><@9@
 사람
 /FC/  Person
>
```

[1] Unfortunately, though, the Korean script does not function properly in the terminal, and for the flashcards to work, *ARCADES* must be run from *Idle*, which however undermines the visual appearance of the notes.

```
<<>>
RUNDEMO:3   run:KOREANFCgen;{{@koreanvocab}} =>echo:????
<run:KOREANFCgen;{{@koreanvocab}} =>echo:????>
echo:????
def generic(textfile):

    import os
    SLASH = '/'
    EMPTYCHAR = ''


    textfile = ' ' + textfile + '  '


    newtext = ''
    ROMAN = 'abcdefghijklmnopqrstuvwxyz'+'ABCDEFGHIJKLMNOPQRSTUZWXYZ'
    afternumber = False
    first = True
    for char in range(1,len(textfile)-2):

        a = textfile[char].isnumeric()
        b = textfile[char-1].isnumeric()
        c = textfile[char+1].isnumeric()

        d = textfile[char] in ROMAN
        e = textfile[char-1] in ROMAN
        f = textfile[char+1] in ROMAN


        if first and a and not b and not c:
            newtext += '<@' + textfile[char] +'@'
            afternumber = True
            first = False
        elif a and not b and not c:
            newtext += '><@' + textfile[char] +'@'
            afternumber = True
        elif a and not b and c:
            newtext += '><@' + textfile[char]
            afternumber = True

        elif a and b and not c:
            newtext += textfile[char] + '@'
            afternumber = True

        elif d and not e and afternumber:
            newtext += '/FC/  '+textfile[char]
            afternumber = False
        else:
            newtext += textfile[char]

    if newtext.count('<') != newtext.count('>'):
        newtext += '>'
    return newtext
```

PLATE 2.104: When run is activated, the program is loaded and displayed. The result of the conversion has been sent to "echo."

2.12.5. interpret

The command interpret can be used to interpret the text of a notescript given as the value.

2.12.6. runinterpret

The command runinterpret combines loadtext and run: it interprets a text file after having applied a Python script to it. The command runinterpret accepts two values: the name of the file, in the directory "/programs", of the program to be run, and the name of the text file to which it is to be applied.

2.12.7. explode

The command explode yields an index, keyword list, and text from a single note, which in turn can be accessed with single, double, and quadruple question marks.

2.12.8. invert

The command invert takes a list of indexes and yields a list of all the indexes that are contained in the notebook yet not in the list.

2.12.9. keys

The command keys (alternative forms: key, k) can be used to refeed a collection of keys into another command. It accepts one value – the collection of indexes from which the keys are to be taken – and the modifiers /$, /&, /*, /?. The modifier /$ generates a histogram from the keys, whereas the modifiers /&, /*, and /?, are used, respectively, to include "all-cap," capitalized, and lower-case keywords. If neither of these three modifiers are invoked, then it will show all keywords.

This generates a search phrase, which can be refed into another command by using double question marks (??).

2.12.10. tags

The command tags (alternate forms: tag, t) displays the tags from a collection of indexes, given as its first value.

This generates a search phrase, which can be refed into another command by using double question marks (??).

2.12.11. text

The command text, which accepts three values and the modifiers /$, /&, and /*, is used to show the most relevant words in a text, determined according to either their frequency in the note, their scarcity in the notebook, or both.

The first value is the collection of indexes, the second value is the number of words according to frequency in text, and third value is the number of words according to their scarcity in the notebook.

The modifier /$ presents the intersection of both sets, while the modifier /& presents them by decreasing frequency in the notebook and /* by increasing frequency in the text of the selected notes.

2.13. MACROS, DEFINITIONS, AND KNOWLEDGE

ARCADES offers various different kind of simple macros, which can be used for entering commands, keywords, and text. These include:

(1) codes	used to define special codes for text entry.
(2) macros	used as shorthand in entering text
(3) keymacros	used in entering keywords

(4) command macros | used for entering commands

In addition to these macros functions, other features in *ARCADES* share similar functionality.

These are:

(5) key definitions | used to automatically assign keywords to text

(6) knowledge | the "knowledge base" for defining metatags.

(7) spelling | words added to the spelling dictionaries.

2.13.1. Basic functionality

The following is an account of the basic functionality of codes, macros, key macros, and command marcros. The treatment of the key definitions, knowledge base, and spelling dictionary will be treated separately.

2.13.1.1. Codes

Codes are used for entering special characters, and especially those which have a reserved function in *ARCADES*, such as the left arrow (<) and right arrow (>).

4 codes are predefined in *ARCADES*:

(1) < = /060/ (2) > = /062/

(3) { =/123/ (4) } =/125/

The use of the slash is not inherent to a code. But it is recommended that you only define codes that are unlikely to appear in ordinary text.

2.13.1.2. Macros

Macros are used to facilitate entry of commonly used terms. Codes are different from macros in several respects:

1) There are no predefined macros.

2) To be invoked, a macro must be preceded by an underline (_) .

3) When defining codes, the "from" and "to" seem to be reversed.

This last point is a bit tricky and confusing. The reason has to do with the fact that, in the case of codes, the code itself is the expansion of the symbol to which it correlates.

2.13.1.3. Key macros

Key macros are used specifically for facilitating the entry of a sequence of keywords. They offer an alternative to default keywords and projects. If you are simultaneously working on several different notetaking projects at once, it might make sense, instead of constantly switching between projects, to define a key macro. Once a key macro has been defined, it can be invoked, when entering keywords, by preceding it with a DOLLAR ($).

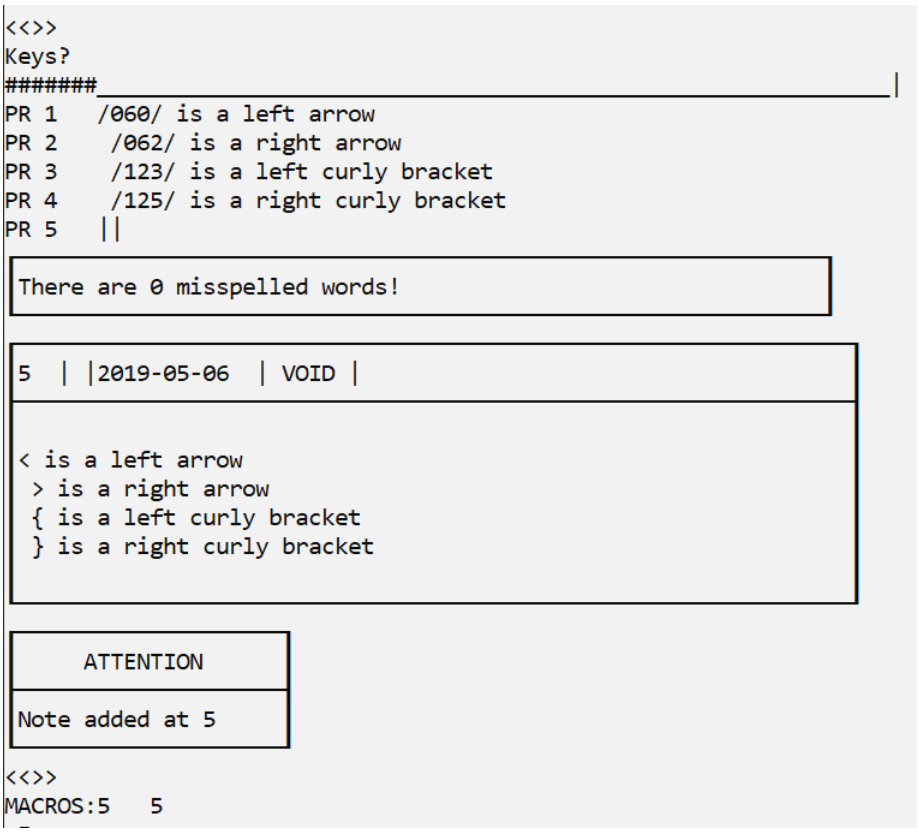


PLATE 2.105: Using predefined codes.

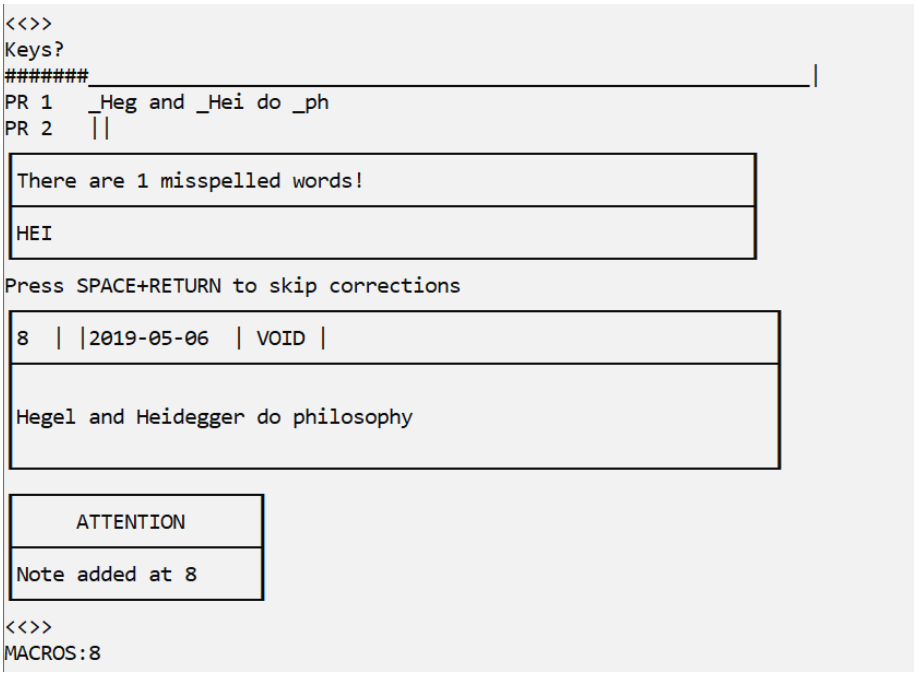


PLATE 2.106: With the macros ph=philosophy, Hei=Heidegger, Heg=Hegel.

.

```
<<>>
Keys? $FROG
########_____________________________________________________________|
PR 1   ||

There are 0 misspelled words!

9  | |2019-05-06  | Seojin/friend, Gom/enemy,
green/color, Juveee/child, Lupu/frog, Girin/enemy
|

     ATTENTION
Note added at 9

<<>>
MACROS:9
```

PLATE 2.107: The key macro "FROG" expands to the terms: Girin/enemy, Gom/enemy, Juveee/child, Lupu/frog, Seojin/friend, green/color.

2.13.1.4. Command macros

Command macros can be used either to redefine commands, or facilitate the entry of a string of commands.

In the following example, the rather useless macro "firstten" is defined as "show:1-10," showing the first ten notes in the notebook.

```
From this? firstten
to this? show:1-10
Add| firstten : show:1-10?yes

A)dd
D)elete
S)how
C)lear
Q)uit

<<>>
MACROS:2   firstten
<show:1-10>

From 0 to 6

2  |   | /C                          | Welcome to NOTESCRIPTION!
3  |   | VOID                        | < is a left bracket > is a right
5  |   | VOID                        | < is a left arrow > is a right arrow { is a left curly bracket }
6  |   | VOID                        | Heidegger
8  |   | VOID                        | Hegel and Heidegger do philosophy
9  |   | Seojin/friend, Gom/enemy    |

<<>>
MACROS:2
```

PLATE 2.108: Defining and using a command macro.

It is important to use some care in defining command macros, and, above all, make sure that they don't interact with other elements of the command phrase or overwrite existing commands.

```
<<>>
MACROS:2   firstten
<show:1-10>

From 0 to 6

2  |   | /C                              | Welcome to NOTESCRIPTION!
3  |   | VOID                            | < is a left bracket > is a right
5  |   | VOID                            | < is a left arrow > is a right arrow { is a left curly bracket }
6  |   | VOID                            | Heidegger
8  |   | VOID                            | Hegel and Heidegger do philosophy
9  |   | Seojin/friend, Gom/enemy        |

<<>>
MACROS:2   +:10;test;firstten
<+:10;test;show:1-10>

10  | |2019-05-06  | test |

show

    ATTENTION

Note added at 10
```

PLATE 2.109: Once the command macro "firstten" has been defined, it cannot be directly entered into the text of the note through the command prompt.

It is recommended, indeed, either only to use unusual terms for command macros or to precede them with some symbol, such as a DOLLAR.

2.13.1.5. Advanced command macro definitions.

ARCADES offers some special codes for entering more sophisticated command macros. To activate these, the entire command macro must be preceded by an AT.

These include:

(1) FIRST | the first index

(2) LAST | the last index

(3) FILE | the name of the current file

(4) BACKUP | the current filename combined the current date

(5) NOW | POUND + the current date

It is also possible to include queries, simply by including the query phrase in square brackets.

2.13.2. Macro commands

ARCADES offers four classes of commands for managing macros: change, default, record, clear. Each of these combines with the four types of macros to yield sixteen distinct commands:

(1) changecodes

(2) changemacros

(3) changekeymacros

(4) changecommandmacros

(5) defaultcodes

(6) defaultmacros

(7) defautkeymacros

(8) defaultcommandmacros

(9) recordcodes

(10) recordmacros

(11) recordkeymacros

(12) recordcommandmacros

(13) clearcodes

(14) clearmacros

(15) clearkeymacros

(16) clearcommandmacros

2.13.2.1. changecodes, changemacros, changekeymacros, changecommandmacros

These commands all call up the "console," which allows for the entry and deletion of the four different kinds of macros.

2.13.2.2. defaultcodes, defaultmacros, defaultkeymacros, defaultcommandmacros

In addition to entering macros through the console, it is also possible to load macros from notes contained within a notebook. These notes can, likewise, be easily transferred between notebooks (using formout and loadtext).

The "default" commands all work in the same way. They search for all the notes in the notebook with a certain identifying keyword, and then interpret the text contained in these notes.

2.13.2.2.1. The identifying keyword

The identifying keyword consists in two parts: an identifying key, and an optional suffix.

```
MACROS:2   changemacros
<changemacros>
```

```
A)dd
D)elete
S)how
C)lear
Q)uit
```

```
A
From this? HY
to this? Hwa Young
Add| HY : Hwa Young?yes
```

```
A)dd
D)elete
S)how
C)lear
Q)uit
```

```
D
```

```
1 HY=Hwa Young
2 Heg=Hegel
3 Hei=Heidegger
4 L=Lupu
5 ph=philosophy
```

```
Delete?3
Delete|Hei : Heideggeryes
```

```
DELETING
```

```
1 HY=Hwa Young
2 Heg=Hegel
3 L=Lupu
4 ph=philosophy
```

```
Delete?
```

```
A)dd
D)elete
S)how
C)lear
Q)uit
```

```
C
Are you sure you want to clear?yes
```

```
A)dd
D)elete
S)how
C)lear
Q)uit
```

```
q
<<>>
MACROS:2
```

PLATE 2.110: The macro-console

2.13.2.2.1.1. The identifying key

The identifying key is simply the type of macro written out in all-caps.

(1) codes	| CODES
(2) macros	| MACROS
(3) keymacros	| KEYMACROS
(4) commandmacros	| COMMANDMACROS

2.13.2.2.1.2. The suffix

The suffix is Unicode-8 string that can be entered through the terminal, excluding special characters that would interfere with keyword entry such as SLASH (/), PERIOD (/), COMMA (,), or AT (@).

Valid suffixes include:

(1)	| 1
(2)	| frog
(3)	| $
(4)	| This is a suffix

2.13.2.2.2. The text of the default note

The text of the default note consists simply in a list of "equations."

It is important to avoid spaces within the equation, since these will be interpreted as demarcating separate codes.

2.12.2.2.2.1. Some examples

The following are some further examples of how codes, macros, command macros, and key macros can be used.

```
DEFAULTDEMO:0   CODESgerman
<CODESgerman>
########_____________________________________________________________|
PR 1   ß=/ess/ ä=/aum/ Ä=/Aum/ ö=/oum/ Ö=/Oum/ ü=/uum/ Ü=/Uum/
PR 2   ||
```

1 | 2019-05-07 | CODESgerman |
ß=/ess/ ä=/aum/ Ä=/Aum/ ö=/oum/ Ö=/Oum/ ü=/uum/ Ü=/Uum/

ATTENTION
Note added at 1

```
<<>>
DEFAULTDEMO:1   defaultcodes:german
<defaultcodes:german>
```

```
1 <=/060/
2 >=/062/
3 {=/123/
4 }=/125/
5 Ä=/Aum/
6 Ö=/Oum/
7 Ü=/Uum/
8 ß=/ess/
9 ä=/aum/
10 ö=/oum/
11 ü=/uum/
```

```
<<>>
DEFAULTDEMO:1   +
<+>
<<>>
Keys?
########_____________________________________________________________|
PR 1   H/oum/lderlin is a great poet!
PR 2   ||
```

2 | 2019-05-07 | VOID |
Hölderlin is a great poet!

PLATE 2.111: EXAMPLE #1: German special character codes

```
<<>>
Keys? MACROSphil
#######_____________________________________________________________|
PR 1   %%
PO 2   K=Kant
PO 3   H=Heidegger
PO 4   KrV=Kritik~der~reinen~Vernunft
PO 5   SuZ=Sein~und~Zeit
PO 6   ||
```

3 | 2019-05-07 | MACROSphil |
K=Kant H=Heidegger KrV=Kritik~der~reinen~Vernunft SuZ=Sein~und~Zeit

ATTENTION
Note added at 3

```
<<>>
MACRODEMO:3   defaultmacros:phil
<defaultmacros:phil>
```

1 H=Heidegger 2 K=Kant 3 KrV=Kritik der reinen Vernunft 4 SuZ=Sein und Zeit

```
<<>>
MACRODEMO:3   +
<+>
<<>>
Keys?
#######_____________________________________________________________|
PR 1   _K wrote the _KrV
PR 2
PR 3   _H wrote the _SuZ
PR 4   ||
```

4 | 2019-05-07 | VOID |
Kant wrote the Kritik der reinen Vernunft Heidegger wrote the Sein und Zeit

ATTENTION
Note added at 4

```
<<>>
MACRODEMO:4
```

PLATE 2.112: EXAMPLE #2: Macros for the names of philosophers

Notice that the tilde (~) is converted into a space when the macro is loaded; this prevents the title from being cut off after the first word.

.

```
Keys? COMMANDMACROS
#######_____________________________________________________________|
PR 1   firsthundred=show:1-100
PR 2   secondhundred=show:101-200
PR 3   thirdhundred=show:201-300
PR 4   ||
```

9	|2019-05-07	COMMANDMACROS |
firsthundred=show:1-100 secondhundred=show:101-200 thirdhundred=show:201-300		

ATTENTION
Note added at 9

```
<<>>
DEFAULTDEMO:9   defaultcommandmacros
<defaultcommandmacros>
Suffix
```

1 firsthundred=show:1-100 2 secondhundred=show:101-200 3 thirdhundred=show:201-300

```
<<>>
DEFAULTDEMO:9   firsthundred
<show:1-100>
```

From 0 to 9			
1	|	CODESgerman	ß=/ess/ ä=/aum/ Ä=/Aum/ ö=/oum/ Ö=/Oum/ ü=/uum/ Ü=/Uum/
2	|	VOID	|
3	|	MACROSphil	F=FichteWL=WissenschaftslehreHeg=HegelPdG=Phänomenologie~des~Geist
4	|	VOID	Fichte wrote the Wissenschaftslehre Hegel wrote the Phänomenologie des
5	|	VOID	|
6	|	||	|
7	|	VOID	Fichte wrote the WissenschaftslehreHegel wrote the Phänomenologie de
8	|	VOID	|
9	|	COMMANDMACROS	firsthundred=show:1-100 secondhundred=show:101-200 thirdhundred=show:2

```
<<>>
DEFAULTDEMO:9
```

```
<<>>
DEFAULTDEMO:209   secondhundred
<show:101-200>
```

From 0 to 14			
101	|	VOID	NOTE
102	|	VOID	NOTE
103	|	VOID	NOTE
104	|	VOID	NOTE
105	|	VOID	NOTE
106	|	VOID	NOTE
107	|	VOID	NOTE
108	|	VOID	NOTE
109	|	VOID	NOTE
110	|	VOID	NOTE
111	|	VOID	NOTE
112	|	VOID	NOTE
113	|	VOID	NOTE
114	|	VOID	NOTE

```
<<>>
DEFAULTDEMO:209
```

PLATE 2.113: EXAMPLE #3: Command macros

```
<<>>
MACRODEMO:2   +
<+>
<<>>
Keys? KEYMACROSphil
#######_____________________________________________________________|
PR 1   Heid:Heidegger,German,phenomenologist,20th-Century
PR 2   Kant:Kant,German,Idealist,18th-Century,Enlightenment,Prussian
PR 3   ||
```

5 | 2019-05-07 | KEYMACROSphil |
Heid:Heidegger,German,phenomenologist,20th-Century Kant:Kant,German,Idealist,18th-Century,Enlightenment,Prussian

ATTENTION
Note added at 5

```
<<>>
MACRODEMO:5   defaultkeymacros:phil
<defaultkeymacros:phil>
```

KEYWORDS : DEFINITIONS
1 Heid: 20th-Century, German, Heidegger, german, phenomenologist 2 Kant: 18th-Century, Enlightenment, German, Idealist, Kant, Prussian

```
<<>>
MACRODEMO:5   +
<+>
<<>>
Keys? $Heid
#######_____________________________________________________________|
PR 1   Heidegger wrote Being and Time
PR 2   ||
```

6 | 2019-05-07 | German, 20th-Century, Heidegger, phenomenologist, german |
Heidegger wrote Being and Time

ATTENTION
Note added at 6

```
<<>>
```

PLATE 2.114: EXAMPLE #4: Key macros.

Notice the KEYMACROS use a COLON (:) instead of EQUAL (=).

2.13.2.3. recordcodes, recordmacros, recordkeymacros, recordcommandmacros

The “record”-commands can be used to record macro definitions to a note.

```
1 Heg=Hegel
2 K=Kant
3 KrV=Kritik der reinen Vernunft
4 WL=Wissenschaft der Logik

A)dd
D)elete
S)how
C)lear
Q)uit
q
<<>>
RECORDDEMO:3   recordmacros:phil2
<recordmacros:phil2>

     ATTENTION
Note added at 4

<<>>
RECORDDEMO:3   4
<4>

4  | |2019-05-07  | MACROSphil2 |

Heg=Hegel
K=Kant
KrV=Kritik~der~reinen~Vernunft
WL=Wissenschaft~der~Logik

<<>>
RECORDDEMO:3   clearmacros
<clearmacros>
<<>>
RECORDDEMO:3   defaultmacros:phil2
<defaultmacros:phil2>

1 Heg=Hegel
2 K=Kant
3 KrV=Kritik der reinen Vernunft
4 WL=Wissenschaft der Logik

<<>>
RECORDDEMO:3
```

PLATE 2.115: Recording macro definitions to a note.

*2.13.2.4. clearcodes, clearmacros, clearkeymacros, clearcommandmacros

While it is possible to clear all entries in a macro through the console, this does not actually destroy and reconstitute an object, as may be necessary if you make changes to the Python modules defining the objects (“abbreviations.py”, “keydefinitions.py”, “keymacrodefinitions.py”). To do this, you must use the “clear”-commands.

2.14. KNOWLEDGEBASE

ARCADES offers two different ways of defining relations between search terms. The first is a simple classificatory knowledgebase, while the second allows for the definition of relational facts.

2.14.1.Classificatory knowledge.

ARCADES includes a simple "knowledge base," which can be used to define ontological facts and apply these in executing searches. An ontological fact consists in the knowledge that one kind of thing can be subsumed under another kind of thing. For example:

(1) a "ontologist" is a "philosopher"

(2) a "philosopher" is a "theorist"

(3) a "theorist" is a "scientist"

(4) a "scientist" is a "thinker"

(5) a "thinker" is a "human being"

(6) a "human being" is an "animal"

(7) an "animal" is a "living being"

(8) a "living being" is a "being"

Given these definitions, it would then be possible to search for all "ontologists" by searching for "scientist" or "human being" or even "being." And we might add these two further definitions:

(9) an "oncologist" is a "medical doctor"

(10) a "medical doctor" is a "scientist"

Then "scientist" would search for both "ontologist" and "oncologist."

The lowest level classification ("ontologist") is a "tag," where the higher classifications are "meta-tags." The reason for this distinction, which might seem arbitrary, is because the knowledge base and the tags are stored separately. The metatags can be thought of as a classificatory schemes supervening on the keys and tags, and is, indeed, much easier to change and manipulate. Knowledge can be easily forgotten without changing the notebook itself!

2.14.1.1. Defining knowledge when entering keys

It is possible to "teach" *ARCADES* an ontological fact when entering keywords with tags. Simply use RIGHTBRACKET after the tag, followed by the metatag.

For example:

(1) Heidegger/philosopher]thinker

(2) Rilke/poet]eauthor

2.14.1.2. Knowledge commands: learn, forget, allknowledge

Ontologically facts can be defined directly through the commands learn and forget. Both take two values: the lower-level classification and the higher-level classification.

The command allknowledge presents all the ontological facts that *ARCADES* has been taught.

```
<<>>
KNOWLEDGEDEMO:4   +
<+>
<<>>
Keys? Heidegger/philosopher=thinker
########_____________________________________________________|
PR 1   Heidegger is a philosopher
PR 2   ||
```

```
4  | |2019-05-07  | Heidegger/philosopher |
```

```
Heidegger is a philosopher
```

```
                  ATTENTION
```

```
I learned that philosopher is a(n) thinker
Note added at 4
```

```
<<>>
KNOWLEDGEDEMO:4   learn:thinker;theorist//learn:theorist;intellectual
<learn:thinker;theorist//learn:theorist;intellectual>
<<>>
KNOWLEDGEDEMO:4   allknowledge
<allknowledge>
```

```
I know that philosopher is a(n) thinker
I know that philospher is a(n) thinker
I know that philosopher is what it is
I know that philospher is what it is
I know that theorist=intellectual is a(n)
I know that thinker=theorist is a(n)
I know that thinker=theorist is what it is
I know that theorist=intellectual is what it is
I know that thinker is a(n) theorist
I know that theorist is a(n) intellectual
```

```
<<>>
KNOWLEDGEDEMO:4   ?:<##intellectual>
<?:<##intellectual>>
```

```
   |NONE   |include negative results with searches
   |       |
```

```
RESULT for Heidegger/philosopher
```

```
4
```

```
<<>>
KNOWLEDGEDEMO:4
```

PLATE 2.116. The knowledge base (example 1).

```
KNOWLEDGEDEMO:4   allknowledge
<allknowledge>

I know that philosopher is a(n) thinker
I know that philospher is a(n) thinker
I know that philosopher is what it is
I know that philospher is what it is
I know that theorist=intellectual is a(n)
I know that thinker=theorist is a(n)
I know that thinker=theorist is what it is
I know that theorist=intellectual is what it is
I know that thinker is a(n) theorist
I know that theorist is a(n) intellectual
I know that scientist is what it is

<<>>
KNOWLEDGEDEMO:4   forget:theorist;intellectual
<forget:theorist;intellectual>
unlearning
<<>>
KNOWLEDGEDEMO:4   learn:theorist;scientist
<learn:theorist;scientist>
<<>>
KNOWLEDGEDEMO:4   ?:<##intellectual>
<?:<##intellectual>>

   |NONE   |include negative results with searches
   |       |

RESULT for VOID

<<>>
KNOWLEDGEDEMO:4   ?:<##scientist>
<?:<##scientist>>

   |NONE   |include negative results with searches
   |       |

RESULT for Heidegger/philosopher

4

<<>>
KNOWLEDGEDEMO:4
```

PLATE 2.117: The knowledge base (Example 2)

2.14.1.3. The “knowledge” console

The “knowledge base” also has the following console commands, whose operation is completely analogous to the four macros: changeknowledge, defaultknowledge, recordknowledge, clearknowledge.

Knowledge is recorded using EQUAL (=).

```
<<>>
KNOWLEDGEDEMO:4   changeknowledge
<changeknowledge>

(L)earn
(U)nlearn
S)how
C)lear
Q)uit

q
<<>>
KNOWLEDGEDEMO:4   +
<+>
<<>>
Keys? KNOWLEDGE
########_____________________________________________________________|
PR 1   Lupu=frog
PR 2   frog=amphibian
PR 3   ||

5  | |2019-05-07  | KNOWLEDGE |

Lupu=frog frog=amphibian

     ATTENTION

Note added at 5

<<>>
KNOWLEDGEDEMO:5   defaultknowledge
<defaultknowledge>
Suffix?

I know that philosopher is a(n) thinker
I know that philospher is a(n) thinker
I know that philosopher is what it is
I know that philospher is what it is
I know that theorist=intellectual is a(n)
I know that thinker=theorist is a(n)
I know that thinker=theorist is what it is
I know that theorist=intellectual is what it is
I know that thinker is a(n) theorist
I know that intellectual is what it is
I know that theorist is a(n) scientist
I know that Lupu is a(n) frog
I know that Lupu is what it is
I know that frog is a(n) amphibian

<<>>
KNOWLEDGEDEMO:5   recordknowledge:all
```

PLATE 2.118: Loading knowledge from a note.

```
<<>>
KNOWLEDGEDEMO:5   recordknowledge:all
<recordknowledge:all>
```

ATTENTION
Note added at 6

```
<<>>
KNOWLEDGEDEMO:5   6
<6>
```

6 | |2019-05-07 | KNOWLEDGEall |
philosopher=thinker philospher=thinker theorist=intellectual= thinker=theorist= thinker=theorist theorist=scientist Lupu=frog frog=amphibian

```
<<>>
KNOWLEDGEDEMO:5
```

PLATE 2.119: Saving knowledge to a note.

2.14.2. Relational knowledge.

The general knowledge base stores and analyzes knowledge expressed as nodes, with directed and nondirected relations between them, and with attributes attached to them. This knowledge can be implemented in searches, making it possible, say, to search for all the descendants of Queen Elizabeth, or all the students of Martin Heidegger.

2.14.2.1. Initiating the general knowledgebase.

It is possible to use either a temporary knowledgebase, a knowledgebase attached to your notebook, or a knowledgebase common to all notebooks.

Upon starting up *ARCADES*, you have the option of which to select.

ATTENTION
DEFAULT DICTIONARY LOADED KEY DICTIONARY LOADED TAG DICTIONARY LOADED WORD DICTIONARY LOADED

DIVIDED
True

```
{}
(3) NO SHELF (4) IN SPECIFIC DATABASE (5) IN GENERAL DATABASE
```

PLATE 2.120 Options for selecting general knowledgebase.

2.14.2.2. The basic structure of knowledge.

Knowledge consists in relations between nodes. Possible relations include directed, non-directed, reciprocal. It is also possible attach a "definition" to a node, and to define complex relations from simpler relations.

Examples of directed relations are: *parent*, *child*.

Examples of non-directed relations are: *sibling*, *spouse*.

Examples of reciprocal relations are: *parent – child*

Examples of complex relations are: *grandparent*, *aunt*

The logical expressiveness of the knowledgebase is currently limited; it is not possible, for example, to give multiple definitions of a complex relation.

2.14.2.3. The general knowledge prompt.

The simplest way to access the general knowledge base is with the command generalknowledge or gk, which calls up a prompt, which can then be used to directly enter knowledge.

Command syntax is as follows:

NODE1,NODE2,NODE2… | Creates one or more nodes

RELATIONNAME:RELATIONTYPE;CONTENT | Define a new relation

RELATIONTYPE = DIRECTED,NONDIRECTED,RECIPROCAL,COMPLEX, ATTRIBUTE

FROMNODE1,FROMNODE2…;RELATION1,RELATION2…;TONODE1,TONODE2… | Learn a relation between nodes.

$ Shows all nodes.

$$NODE;RELATION1,RELATION2 | Finds nodes related through relations in sequence.

$$NODE;RELATION* | Find all nodes related by relation to the top node

$$$ Shows all relations

$$$$ Display the entire knowledgebase.

2.14.2.4. Entering knowledge directly in the text of the note.

It is also possible to include calls to the knowledge base in the text of a note by surrounding them with double curly braces.

2.14.2.5. Entering knowledge calls as a keyword.

A call can be entered as a keyword, either within the text by surrounding with single curly brackets or in the list of keywords. IN THIS CASE THE NODE is ALSO ENTERED AS A KEYWORD.

Here the call syntax is different, and more limited. No lists are possible. QUESTION MARK is used before the call and between the relation and the content. Nodes can be defined, and relations between nodes established, but relations cannot be created.

2.14.2.6. Searching over relations

It is possible to search over the relations of a node. The syntax is ?NODE?RELATION. To search for keywords, enclose with <>.

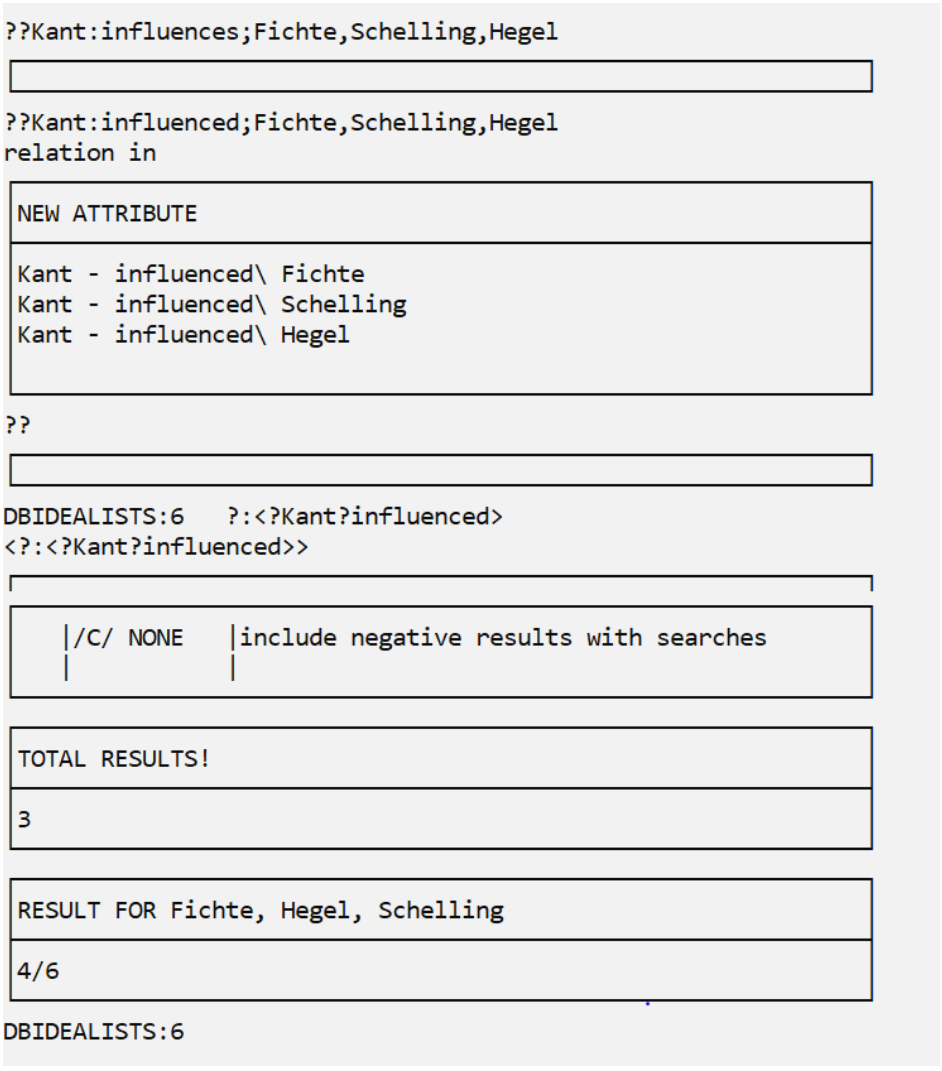


PLATE 2.121 Searching over relations

2.14.2.7. The general knowledge console.

Use the command changegeneralknowledge to access the general knowledge console, which offers a variety of options for directly interacting with the knowledge database.

```
<changegeneralknowledge>
(E)nter
(?)+PHRASE
(A)ll
(N)odes
(R)elations
(K)nowledge
(C)onverses
Comple(X)
C(L)ear)
(Q)uit
ex(P)ort
(I)nport
```

PLATE 2.122 The console for change general knowledge.

2.14.2.8. Additional general knowledge commands.

Use switchgeneralknowledge to switch between the temporary knowledgebase, the common knowledgebase, and the knowledge base attached to the notebook.

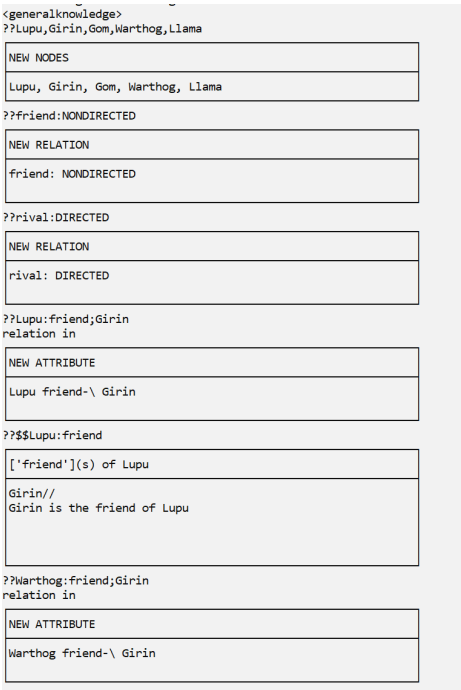


PLATE 2.123 Defining nodes and relations

```
??$$Warthog:friend*
```

['friend*'](s) of Warthog
Girin,Lupu,Warthog// Girin is the friend of Warthog Lupu is the friend of Girin Girin is the friend of Lupu Warthog is the friend of Girin

```
??
```

PLATE 2.124 Using * to find all related nodes

```
??pig,pug,pog
```

NEW NODES
pig, pug, pog

```
??squib,squab:DIRECTED
```

NEW RELATION
squib: DIRECTED squab: DIRECTED

```
??pig,pug,pog:squib,squab;pig,pug,pog
relation in
relation in
relation in
relation in
relation in
relation in
```

NEW ATTRIBUTE
pig - squib\ pug pig - squib\ pog pug - squib\ pig pug - squib\ pog pog - squib\ pig pog - squib\ pug pig - squab\ pug pig - squab\ pog pug - squab\ pig pug - squab\ pog pog - squab\ pig pog - squab\ pug

```
??
```

PLATE 2.125 Teaching multiple relations between multiple modes

2.14.3. Equivalences

In additional to classificatory and relational knowledge, *ARCADES* can also define equivalences. Equivalences are automatically invoked when searching for a term, regardless of the level of the term—i.e. whether it is a keyword, text word, or a tag.

2.14.3.1. Defining equivalences.

Equivalences can be defined when entering keywords and tags into a note.

Simply link together a set of equivalent terms with EQUAL.

For example: Venus=morning star=evening star/planet=moving star[celestial body

> This defines two new equivalence classes ({Venus, morning star, evening star},{planet, moving body}) and also "Venus" with the tag "planet", and classifies a "planet" as a "celestial body." Notice that only the first term of the list of equivalences is defined as a tag or a keyword.

2.14.3.2. The equivalence console.

The command changeequivalences can be used to invoke the console to modify the equivalence classes. Notice that you can add new equivalence classes, delete equivalence classes, delete single terms, merge equivalence classes, and also show all the equivalence definitions.

2.14.3.3. Suspending automatic equivalence fetching

The command yieldequivalences can be used to turn off or on the automatic fetching of equivalences with searches.

2.14.3.4. Fetching equivalences for a single term.

When automatic fetching is suspended, a PLUS before the term can be used to fetch equivalences for that term.

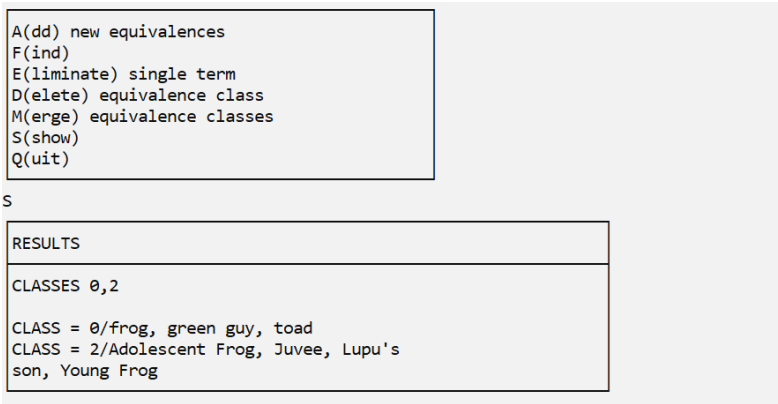


PLATE 2.125b The Equivalence Console

2.14.3.5. Complex equivalences

In addition to simple equivalences, *ARCADES* also allows you to define complex equivalences, involving a combination of any number of simple terms and complex terms, where a complex terms consists of logical expression composed of two or more simple terms joined with parentheses, &, |, and ~..

Examples of valid complex terms include:

(Lupu & Girin)

((Lupu | Girin) & random frogs).

These should best be entered from the equivalence console, since a bracketed phrase should not be used as a keyword.

When a equivalence is made between a logical phrase and other phrases, then *ARCADES*, when running a search, tests to see if the equivalence is logically entailed by the search phrase. It does not use text substitution.

ARCADES will search for further entailments from complex phrases. If there are many levels of logic, it may begin to take some time. This is because the run time of the algorithm for the logical analysis of the search phrase increases exponentially according to the number of logical terms in the search phrase.

```
RESULTS for equivalencetest

CLASSES 0,1,2,3,4,5,6,7,8,9,10

CLASS = 0/(green & frog), Lupu
CLASS = 1/(giraffe & slow), Girin
CLASS = 2/(Lupu & son), Juvee
CLASS = 3/(cat & girl), Dal
CLASS = 4/(cat & male), Pyeol
CLASS = 5/(Pyeol & Dal), friends
CLASS = 6/(friends & Lupu), monstrosity
CLASS = 7/(Lupu & Girin), enemies
CLASS = 8/(~Lupu & ~Girin), others
CLASS = 9/bigfoot, smallfoot
CLASS = 10/(science & art), philosophy
```

Plate 2.125c. Equivalences classes as seen from the equivalence console.

```
DBequivalencetest:1   ?:<giraffe> & <slow> & <frog> & <green>
<?:<giraffe> & <slow> & <frog> & <green>>
```

SEARCH CONSTITUTION
1/SE [3] & [2] & [0] & [1] 2/E [3] & [2] & [0] & [1] 3/CE([0] | ([2] & [1])) 4/SE ([0]) | (([2]) & ([1])) 5/E (([3] | ([5] & [1]))) | ((([5] | ([2] & [0]))) & (([1] | ([6] & [4])))) 6/CE(([3] | ([5] & [1]))) | ((([5] | ([2] & [0]))) & (([1] | ([6] & [4])))) 7/SE (([3] | ([5] & [1]))) | ((([5] | ([2] & [0]))) & (([1] | ([6] & [4])))) 8/E ((([3] | ([5] & [1])) | (([5] | ([2] & [0])) & ([1] | ([6] & [4])))))) | (((([5] | ([2] & [0])) | ([2] & [0]))) & ((([1] | ([6] & [4])) | ([6] & [4])))) 9/CE((([3] | ([5] & [1])) | (([5] | ([2] & [0])) & ([1] | ([6] & [4])))))) | (((([5] | ([2] & [0])) | ([2] & [0]))) & ((([1] | ([6] & [4])) | ([6] & [4]))))

SEARCH TERM
(((<enemies> | (<Lupu> & <Girin>)) | ((<Lupu> | (<green> & <frog>)) & (<Girin> | (<giraffe> & <slow>))))) | ((((<Lupu> | (<green> & <frog>)) | (<green> & <frog>))) & (((<Girin> | (<giraffe> & <slow>)) | (<giraffe> & <slow>))))

TOTAL RESULTS!
1

FOUND KEYWORDS
enemies, Lupu, Girin

RESULT FOR Girin, Lupu, enemies
8

Plate 2.125d. A search who two enchained entailments.

2.15. KEY DEFINITIONS

ARCADES allows you to load text by paragraph, converting each paragraph into a note, while automatically assigning keywords to each note based on the words that appear in the text.

2.15.1 Loading by paragraph

Loading raw text into *ARCADES* can be initiated through two different commands: loadbyparagraph and splitload. They differ in only one respect: loadbyparagraph splits up the text using the paragraph mark ("/n"), whereas splitload splits on a user-defined string.

The command splitbyparagraph accepts one value – the name of the file to be loaded – and the modifiers /$, /&, /*. The first modifier is used to suppress the "key" function, which calls up existing keys. The "key" function is automatically suppressed if there are more than 50 keys in the notebook. The second modifier is used to invoke automatic key definitions. The third modifier is used to repress querying the user to add notes and continue.

The command splitload is similar, except that it accepts, as a second value, the character string used to split the text into paragraphs.

2.15.2. The console-commands

The commands changekeydefinitions, defaultkeydefinitions, recordkeydefinitions, and clearkeydefinitions can be used to define and change key definitions.

The formatting of default key definitions is as follows:

KEY:DEFINITION1,DEFINITION2,DEFINITION3…

The "definitions" are the words in the text corresponding to a given key.

```
22  | |2019-05-07  | KEYDEFINITIONS |

epistemology:idea,knowledge
ontology:essence,existence,existent,substance
theology:God,infinite,infinity

<<>>
SPINOZAETHICS:11
```

PLATE 2.126: A simple example of automatic key definitions.

```
KEYDEF:4    changekeydefinitions
<changekeydefinitions>

A)dd
D)elete
S)how
C)lear
Q)uit

A
Keys?Girin
Definitions?giraffe

A)dd
D)elete
S)how
C)lear
Q)uit

A
Keys?Girin
Definitions?tall

A)dd
D)elete
S)how
C)lear
Q)uit

S

          KEYWORDS : DEFINITIONS
1 Girin: Buddhist, Giraffe, giraffe, tall
2 Lupu: frog, swamp

 KEYWORDS : DEFINITIONS
1 Buddhist: Girin
2 frog: Lupu
3 Giraffe: Girin
4 giraffe: Girin
5 swamp: Lupu
6 tall: Girin
```

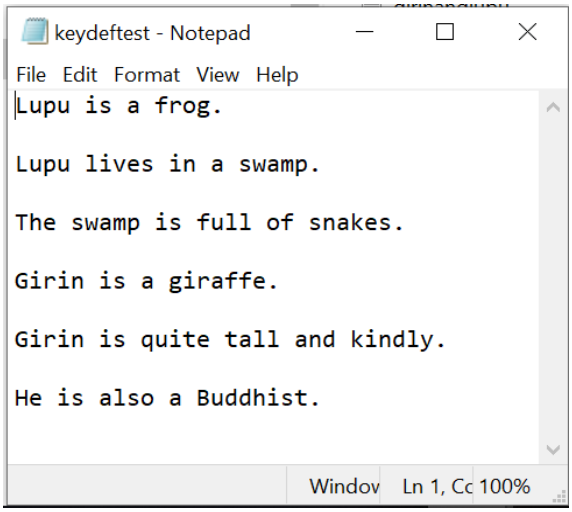


PLATE 2. 127: (1/3) The key definitions and the text to which they are applied.

```
Are you sure you want to open: keydeftest? yes
```

Lupu
<<Lupu>> is a frog.

Lupu
<<Lupu>> lives in a swamp.

Lupu
The swamp is full of snakes.

Girin
<<Girin>> is a giraffe.

Girin
<<Girin>> is quite tall and kindly.

Girin
He is also a Buddhist.

ATTENTION
Note added at 3 Note added at 4 Note added at 5 Note added at 6 Note added at 7 Note added at 8

```
<<>>
KEYDEF:2
```

PLATE 2.128: (2/3) The application of the key definitions. (II)

The text has been divided into paragraphs.

```
KEYDEF:2   all /$
<all /$>
```

From 0 to 7			
2	|	/C	Welcome to NOTESCRIPTION!
3	|	Lupu	Lupu is a frog.
4	|	Lupu	Lupu lives in a swamp.
5	|	Lupu	The swamp is full of snakes.
6	|	Girin	Girin is a giraffe.
7	|	Girin	Girin is quite tall and kindly.
8	|	Girin	He is also a Buddhist.

```
<<>>
KEYDEF:2
```

PLATE 2.129: (3/3). The final result of the application.

2.16. SPELLING

ARCADES includes a basic spellchecker, with support for English, French, Spanish, and German. The spellingchecker includes the capacity to learn new words, and keeps separate dictionaries for all dictionaries. New words can be added during text entry, through a "Console", or by loading in new words from notes.

2.16.1. Activating and deactivating the spellchecker

Use the spelling command to toggle the spellchecker on or off.

2.16.2. Selecting a language

The spellchecker allows you to choose between English, French, Spanish, and German. To do so, use the language command with "en", "es", "fr", "de" as a value.

2.16.2. Using the spellchecker during text entry

If the spellchecker is activated, then *ARCADES*, following text entry, ultimately checks for misspelled words. It checks using the dictionary of the language that has been selected, although it automatically switches to German if an excessive number of errors in the other language are present.

2.16.3. Using the console

To call up the console, use the command spelldictionary. The console allows you to add new words, delete words, change the language, and display the words that have been added.

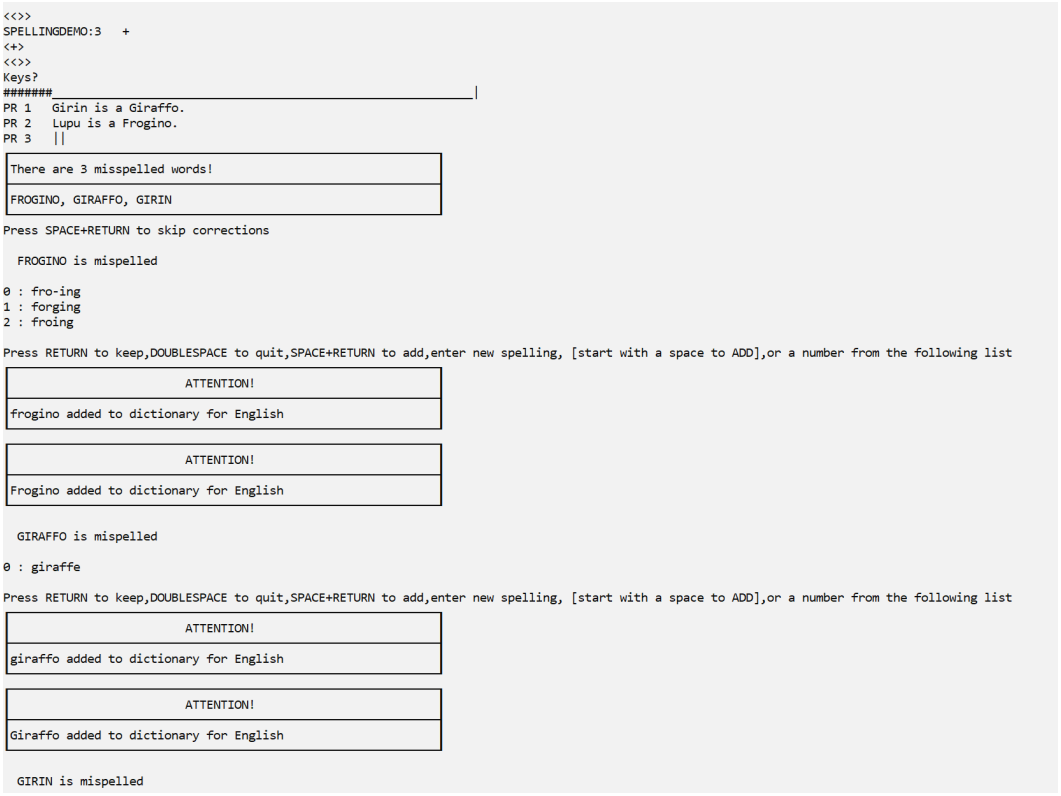


PLATE 2. 130 Adding new words during spell check (before).

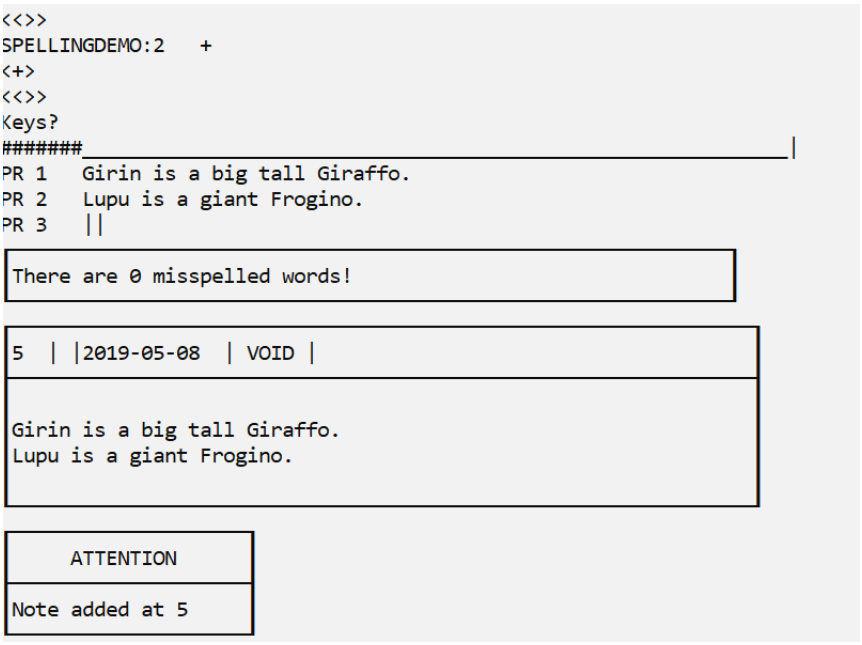


PLATE 2. 131: Adding new words during spell check. (after)

```
SPELLINGDEMO:5   spelldictionary
<spelldictionary>
```

SPELLING DICTIONARY
A(dd) new word D(elete) L(oad)words from text S(how) words C(hange) language (E)rase (Q)uit

```
?A
New word to add?Magnifico
```

ATTENTION!
magnifico added to dictionary for English

SPELLING DICTIONARY
A(dd) new word D(elete) L(oad)words from text S(how) words C(hange) language (E)rase (Q)uit

```
?S
```

ENGLISH
frogino, Frogino, Giraffo, giraffo, girin, Girin, magnifico

SPELLING DICTIONARY
A(dd) new word D(elete) L(oad)words from text S(how) words C(hange) language (E)rase (Q)uit

```
?
```

PLATE 2.132: The spelling console.

2.16.4 Loading from notes

The defaultspelling command operates analogously to the "default"-commands for macros, key definitions, and the knowledge base. There are only two differences.

(1) The suffix must contain the language-abbreviation ("en", "es", "de", "fr") in addition to the optional specifier.

(2) The text consists simply in a list of words to be added to the spelling dictionary for the language indicated.

```
<<>>
SPELLINGDEMO:5   +
<+>
<<>>
Keys? SPELLINGenFROG
########_____________________________________________________|
PR 1   frogolo frogino frogetto frogalicious frogtastic
PR 2   ||
```

```
There are 4 misspelled words!
frogolo, frogalicious, frogtastic, frogetto
```

```
Press SPACE+RETURN to skip corrections
```

```
6  | |2019-05-08  | SPELLINGenFROG |
frogolo frogino frogetto frogalicious frogtastic
```

```
     ATTENTION
Note added at 6
```

```
<<>>
SPELLINGDEMO:6   defaultspelling:enFROG
<defaultspelling:enFROG>
```

```
                        ATTENTION!
frogolo, frogalicious, frogtastic, frogetto
added to dictionary for English
```

```
                        ATTENTION!
 added to dictionary for English
```

```
<<>>
SPELLINGDEMO:6   spellingdictionary
<spellingdictionary>
```

PLATE 2.133 Loading new words into the spelling dictionary from a note.

2.17. SETTINGS, CONFIGURATIONS, MENUS, AND HELP

2.17.1. Settings

The command showsettings can be used to show all the string, integer, and Boolean values stored as persistent properties. Most, but not all, of these properties can be manually adjusted.

2.17.1.1. Miscellaneous binary commands

(1) usesequence | disables use of sequences

(2) boxconfigs | displays configuration in a "boxy" way.

(3) autobackup | disables automatic backup

(4) curtail | eliminates lines at beginning of note

(5) showdate | shows dates when displaying note

(6) showshow | displays all notes in short form

(7) carryoverkeys | carries over keys from parents to children

(8) carryall | carries over keys from all parents, or first two parents

(9) negresults | includes negative indexes in search results

2.17.1.2. Miscellaneous integer settings

(10) trim | sets the length of all keywords when displaying notes as list

(11) setlongmax | sets the maximum number of notes shown in "long" mode

(12) smallsize | sets the small size of notes for multidisplay

2.17.2. Configurations

The commands showconfigurations can be used to show the non-persistent attributes of the notebook.

2.17.2.1 The command saveconfigurations and loadconfigurations

The commands saveconfigurations and loadconfigurations can be used to save and load configurations. Both accept a single value: the identifier of the configurations. The configurations are save as a .pkl file, with "_config.pkl" automatically added to the identifier to yield the filename.

2.17.3. Help and menus

2.17.3.1. Menus

The command bigmenu displays a large menu showing all the commands, while menu calls up a top-level menu from which the lower-level menus can be selected. Both menus can be used to enter commands. The bigmenu is so large that it is nearly impossible to get it to display properly without a special monitor

```
CONFIGURATIONS:1   showsettings
<showsettings>
```

SETTINGS
enterhelp : True formattinghelp : True updated data : True user : ANTHONY displayonstart : True variablesize : True size : 60 trim : 30 orderkeys : False numberof : 5 curtail : True header : 1 setitflag : False footer : 1 leftmargin : 0 showdate : False sortbydate : False determinant : ym keysbefore : True keysafter : False carryoverkeys : True carryall : True returnquit : 3 returnquiton : False indentmultiplier : 4 smallsize : 50

```
<<>>
CONFIGURATIONS:1
```

PLATE 2.134: Settings

```
CONFIGURATIONS:5   showconfigurations
<showconfigurations>
```

CONFIGURATIONS
pause : ON longshow count : 60 shortshow count : 5 always next : Off always child : Off quickenter : Off longmax : 100 autobackup : ON flipout : Off shortshow : Off all cap purge : ON first cap purge : Off lower case purge : Off children too : ON box configs : Off auto multi : ON show full top : ON

```
<<>>
CONFIGURATIONS:6
```

PLATE 2.135: Configurations

```
<<>>
CONFIGURATIONS:6   menu
<menu>
```

COMMANDS
1: COMMANDS 2: DISPLAY 3: SEARCHING 4: ORGANIZING NOTES 5: DEFAULTS 6: INPUT/OUTPUT 7: DEFAULT 8: ADVANCED DISPLAY 9: KNOWLEDGE BASE & SYSTEM 10: ADVANCED

```
?3
```

SEARCHING
1: search, ? 2: terms,??? 3: textsearch, ?? 4: constdates, 5: constitutedates 6: activedet, 7: actdet 8: showdatedict 9: ahowdatedictpurge 10: cleardatedict 11: changedet 12: showdet 13: setpurgekeys 14: clearpurgekeys 15: showpurgekeys 16: searchlog 17: clearlog, 18: clearsearchlog

```
?
```

PLATE 2.136: The small help menu.

```
[, >,<, ] (Q)uit
```

DISPLAY		
123[return]	displays the next note	displays note with INDEX 123
all, $	levels to show /$ suppress quick mode /&no children /* no brackets /? suppress short show /= showdates %indexrange	show all notes
show, s	indexrange;levels to show /&no children /* no brackets /? suppress short show /= show dates	show some notes
inc		incremental show
indexes, ind, i		show indexes
keys, key, k	/$ histiogram /& all caps /* upperkeys /? lowerkeys	show keys =>COMMAND:?? to feed back results
tags, tag, t		show tags
text	index range;# of words1;# of words2;gets important words in the text /$ for intersection of words /& by decreasing frequency in notebase /$ by increasing frequency in note itself	
keysfortags		show keys for tags
defaultkeys, dfk		show default keys
showdel		show soft-deleted notes
keystags		show tags and their keys
flipforward,ff		flipcard forward
flibback, fb		flipcard back
flipreset,fr		return to the first side
flipto,ft		go to side

```
[, >,<, ] (Q)uit
```

PLATE 2.137: The help screen.

The command help shows the commands, together with information about them, as a series of tables, allow the user to move back and forth between them.

```
<<>>
CONFIGURATIONS:7   help:searchlog
<help:searchlog>
```

COMMAND = searchlog
show the search log

```
<<>>
CONFIGURATIONS:7   help:move
<help:move>
```

COMMAND = move
indexrange;indexrange;S or M or C;Yes/no.. S=Subordinate /$.. M=Make Compact /&.. Children /*
move to notes from.. sourcerange to destinationrange Preserves hierarchical.. structure when subordinating Collapses hierarchical.. structure Each note is a child of the last

```
<<>>
CONFIGURATIONS:2   help:showdet
<help:showdet>
```

COMMAND = showdet
show determinants

```
<<>>
CONFIGURATIONS:3
```

PLATE 2.138: The command help may also accept as a value the name of a command.

2.18 Additional features.

2.18.1. Scientific calculator

ARCADES includes a simple scientific calculator, which can be activated with the command calculate.

```
DBIDEALISTS:2   calculate
<calculate>
           A SIMPLE SCIENTIFIC CALCULATOR
                        by
                 Anthony Curtis Adler

         OPERATORS = +,/,-,*,%(mod),^(power), ()
         FUNCTIONS abs,floor,fmod,frexp,gcd,remainder,
         trunc,exp,expml,logn,logx,log1p,log2,log10,
         power,sum,acos,asin,atan,atan2,cos,hypot,
         sin,tan,degrees,radians,acost,asinh,atanh,cosh,
         sing,tanh,erf,erfc,gamma,lgamma,neg

         CONSTANTS pi, e, tau, inf, nan

         Any alphanumeric phrase can serve as a variable.
         To return an entry from the log, type @line@.

         ALL to show the log

?X=10
 X                       | 10 = 10.0
?Y=20
 Y                       | 20 = 20.0
?((X+Y)^4)+(3/4)
                         | ((X+Y)^4)+(3/4) = 810000.75
?
1 : 810000.75            | ((X+Y)^4)+(3/4)
?QUIT
```

PLATE 139 Scientific calculator

2.18.2. Truth table generator

The command truthtable can be used to generate truth table from a logical phrase. The logical phrase can be entered as a value, or via the prompt.

```
DBIDEALISTS:2   truthtable
<truthtable>
```

```
TRUTH TABLE
```

```
          TRUTH TABLE GENERATOR

         by Anthony Curtis Adler


     ENTER A PHRASE TO INTERPRET
     USING THE FOLLOWING SYMBOLS:

AND,and,&,^                 = LOGICAL AND

OR,or,V,|                   = LOGICAL OR
iff,is equivalent to,<>     = LOGICAL EQUIVALENCY
>,=>,->,implies             = LOGICAL IMPLICATION
-,~                         = NEGATION
[] ()                       = BRACKETS
```

```
?X->(Y OR -Z)
```

```
TRUTH TABLE FOR X->(Y OR -Z)
```

```
TRUTH TABLE FOR X->(Y OR -Z)

(0):  X  Y  Z  | True
(1): ~X  Y  Z  | True
(2):  X  Y ~Z  | True
(3): ~X  Y ~Z  | True
(4):  X ~Y  Z  | False
(5): ~X ~Y  Z  | True
(6):  X ~Y ~Z  | True
(7): ~X ~Y ~Z  | True
```

```
DBIDEALISTS:2
```

PLATE 140 Truth table

2.19. HOUSEKEEPING

2.19.1 Houskeeping commands

ARCADES offers a few commands that can be used for "housekeeping." These include:

(1) refresh | Reconstitutes the word dictionary

(2) updatetags | Goes through all the keys in the notebook and updates tags

2.19.2 Diagnostics

In order to faciliate the diagnostics of problems with the consistency of notebook (the correspondence of the notebook keys with the dictionaries kept in the pickle file), *ARCADES* automatically records all actions taken in the notebook, as well as the result of the inconsistency test, to a textfile, identified by the name of the notebook + DIAG.

```
_____________
INITIALIZE
_____________
1          (2019-06-26 16:53:52.556667) :: enterhelp
2          (2019-06-26 16:53:56.855170) :: formathelp
3          (2019-06-26 16:54:01.555603) :: fromtext
4          (2019-06-26 16:54:05.699512) :: newconvertmode
5          (2019-06-26 16:54:19.899529) :: switchconvertmode
6          (2019-06-26 16:54:28.501520) :: convertdefinitions
7          (2019-06-26 16:54:41.417975) :: +
8          (2019-06-26 16:55:03.366262) :: 3
9          (2019-06-26 21:38:30.995592) :: seqformone
10         (2019-06-26 21:38:57.938319) :: seqformone:FROG
11         (2019-06-26 21:39:16.206861) :: seqformone:n
12         (2019-06-26 21:39:25.051132) :: seqformtwo:n
13         (2019-06-26 21:39:43.654562) :: seqformtwo:OTHER TEXT
14         (2019-06-26 21:39:50.749524) :: seqformtwo:OTHER TEXT
15         (2019-06-26 21:39:54.864274) :: seqformtwo
16         (2019-06-26 21:45:20.595445) :: +
17         (2019-06-26 21:45:31.464197) :: quit
_____________
TERMINATE
_____________
_____________
INITIALIZE
_____________
1          (2019-06-26 22:27:39.726813) :: quit
_____________
TERMINATE
_____________
_____________
INITIALIZE
_____________
1          (2019-06-26 22:28:04.120849) :: all /$
2          (2019-06-26 22:28:06.224218) :: quit
_____________
TERMINATE
_____________
```

PLATE 2.141 An example of a diagnostic text file.

2.19.4 The register.

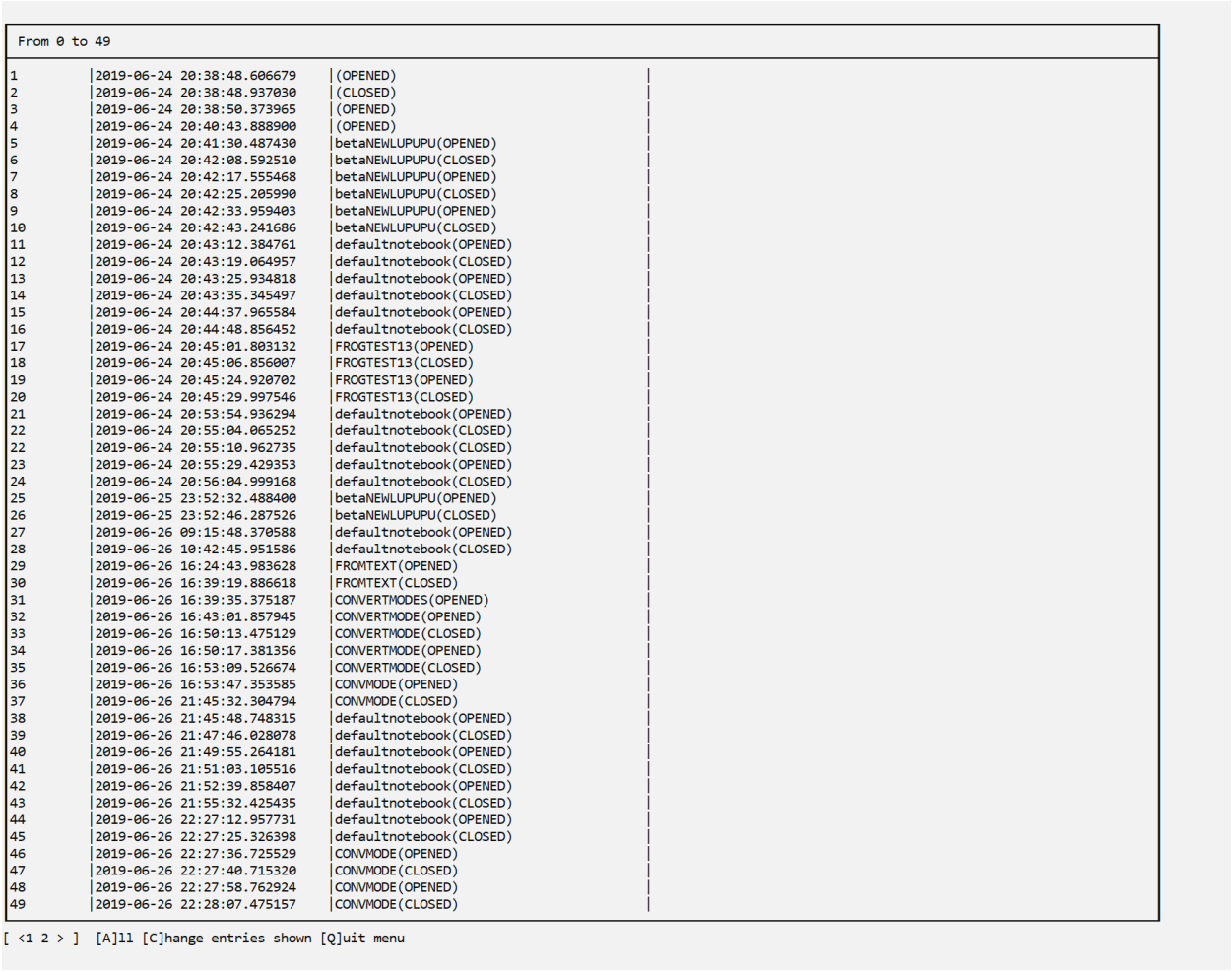
ARCADES automatically keeps track which notebooks have been opened and closed, as well as the projects that were activated while they were open, and the index position at the time of closing.

Upon starting up, *ARCADES* gives you the option of viewing the register. It is possible either to see the entire registry, or to search for a notebook, a projectname, a status, or even for notebooks that were accessed within a certain time period.

```
(1) Show the entire registry
(2) Show registry for a notebook
(3) Search
(4) Quit

?1
```

PLATE 2.142 The registry menu



```
From 0 to 49

1         |2019-06-24 20:38:48.606679    |(OPENED)                          |
2         |2019-06-24 20:38:48.937030    |(CLOSED)                          |
3         |2019-06-24 20:38:50.373965    |(OPENED)                          |
4         |2019-06-24 20:40:43.888900    |(OPENED)                          |
5         |2019-06-24 20:41:30.487430    |betaNEWLUPUPU(OPENED)             |
6         |2019-06-24 20:42:08.592510    |betaNEWLUPUPU(CLOSED)             |
7         |2019-06-24 20:42:17.555468    |betaNEWLUPUPU(OPENED)             |
8         |2019-06-24 20:42:25.205990    |betaNEWLUPUPU(CLOSED)             |
9         |2019-06-24 20:42:33.959403    |betaNEWLUPUPU(OPENED)             |
10        |2019-06-24 20:42:43.241686    |betaNEWLUPUPU(CLOSED)             |
11        |2019-06-24 20:43:12.384761    |defaultnotebook(OPENED)           |
12        |2019-06-24 20:43:19.064957    |defaultnotebook(CLOSED)           |
13        |2019-06-24 20:43:25.934818    |defaultnotebook(OPENED)           |
14        |2019-06-24 20:43:35.345497    |defaultnotebook(CLOSED)           |
15        |2019-06-24 20:44:37.965584    |defaultnotebook(OPENED)           |
16        |2019-06-24 20:44:48.856452    |defaultnotebook(CLOSED)           |
17        |2019-06-24 20:45:01.803132    |FROGTEST13(OPENED)                |
18        |2019-06-24 20:45:06.856007    |FROGTEST13(CLOSED)                |
19        |2019-06-24 20:45:24.920702    |FROGTEST13(OPENED)                |
20        |2019-06-24 20:45:29.997546    |FROGTEST13(CLOSED)                |
21        |2019-06-24 20:53:54.936294    |defaultnotebook(OPENED)           |
22        |2019-06-24 20:55:04.065252    |defaultnotebook(CLOSED)           |
22        |2019-06-24 20:55:10.962735    |defaultnotebook(CLOSED)           |
23        |2019-06-24 20:55:29.429353    |defaultnotebook(OPENED)           |
24        |2019-06-24 20:56:04.999168    |defaultnotebook(CLOSED)           |
25        |2019-06-25 23:52:32.488400    |betaNEWLUPUPU(OPENED)             |
26        |2019-06-25 23:52:46.287526    |betaNEWLUPUPU(CLOSED)             |
27        |2019-06-26 09:15:48.370588    |defaultnotebook(OPENED)           |
28        |2019-06-26 10:42:45.951586    |defaultnotebook(CLOSED)           |
29        |2019-06-26 16:24:43.983628    |FROMTEXT(OPENED)                  |
30        |2019-06-26 16:39:19.886618    |FROMTEXT(CLOSED)                  |
31        |2019-06-26 16:39:35.375187    |CONVERTMODES(OPENED)              |
32        |2019-06-26 16:43:01.857945    |CONVERTMODE(OPENED)               |
33        |2019-06-26 16:50:13.475129    |CONVERTMODE(CLOSED)               |
34        |2019-06-26 16:50:17.381356    |CONVERTMODE(OPENED)               |
35        |2019-06-26 16:53:09.526674    |CONVERTMODE(CLOSED)               |
36        |2019-06-26 16:53:47.353585    |CONVMODE(OPENED)                  |
37        |2019-06-26 21:45:32.304794    |CONVMODE(CLOSED)                  |
38        |2019-06-26 21:45:48.748315    |defaultnotebook(OPENED)           |
39        |2019-06-26 21:47:46.028078    |defaultnotebook(CLOSED)           |
40        |2019-06-26 21:49:55.264181    |defaultnotebook(OPENED)           |
41        |2019-06-26 21:51:03.105516    |defaultnotebook(CLOSED)           |
42        |2019-06-26 21:52:39.858407    |defaultnotebook(OPENED)           |
43        |2019-06-26 21:55:32.425435    |defaultnotebook(CLOSED)           |
44        |2019-06-26 22:27:12.957731    |defaultnotebook(OPENED)           |
45        |2019-06-26 22:27:25.326398    |defaultnotebook(CLOSED)           |
46        |2019-06-26 22:27:36.725529    |CONVMODE(OPENED)                  |
47        |2019-06-26 22:27:40.715320    |CONVMODE(CLOSED)                  |
48        |2019-06-26 22:27:58.762924    |CONVMODE(OPENED)                  |
49        |2019-06-26 22:28:07.475157    |CONVMODE(CLOSED)                  |

[ <1 2 > ]  [A]ll [C]hange entries shown [Q]uit menu
```

PLATE 2.143 The registry

2.19.4.1 Protection again file corruption

ARCADES uses the registry to prevent you from opening a notebook that has not been properly closed. This is no longer necessary now that *ARCADES* has been built around *SQLITE*, since *SQLITE* automatically locks the database to protect against file corruption.

If you attempt to open a file that has not been closed, *ARCADES* will display an alert, and the give you several options. You can either open it as a read only notebook, with all notebook modifying functions suspending, or go back to the main notebook selection menu, or – if you have ascertained that the file is not presently open but was simply not closed properly – you can correct the registry and continue.

```
Are you sure you want to open: betaNEWLUPUPU? yes
betaNEWLUPUPU
```

ATTENTION
betaNEWLUPUPU is still in use or has not been closed properly!

SELECT
(o)pen as read only (c)orrect registry and continue (s)elect another?

```
?
```

PLATE 2.144 Correcting the registry

APPENDIX A

EXAMPLES OF VALID SEARCH QUERIES

SINGLE TERMS

(1) ?:Heidegger	# Search for all notes containing "Heidegger"
	# in the text
(2) search:Heidegger	# variant form of (1)
(3) ?:HEIDEGGER	# Like (1), but capitalized or non-capitalized variants.
(4) ?:<Heidegger>	# Like (1), but searches for a keyword.
(5) ?:<HEIDEGGER>	# Like (1), but for keywords.
(6) ?:*egge*	# Search for all notes with words containingt "egge"
(7) ?:<*egge*>	# (6) for keywords
(8) ?:Hei*	# Like (6), but starting with "Hei"
(9) ?:<Hei*>	# Like (7), but for keywords.
(10) ?:<#philosopher> `	# Finds all notes with keywords tagged "philosopher"
(11) ?:<##theoretician>	# Find all notes with tags subsumed under the genus "intellectual"
(12) ?:<?Husserl?infleunced?>	#Search for all note with keywords "influenced" by #"Husserl"
(13) ?:<?Husserl?influenced*>	#Like (13) but searches over the entire tree of |influences
(14) ?:<page@1/page@10>	#Searches for sequence keyword "page" with values
	# between 1 and 9
(15) ?:BeingandTime$	# Search for the exact phrase "Being and Time"
(16) ?:$Being$Time$	# Search for a phrase beginning with "Being"
	# and ending with "Time"
(17) ?:$$ing$im$$	# Wildcard phrase search for a phrase beginning *ing* and ending *im*

QUALIFIED TERMS

(18) ?:Heidegger"date=2020-1/2020-6"	# (1), restricted to notes between 2020-1 and 2020-6
(19) ?:<Heidegger>"index=/1000"	# (4), restricted to notes with indexes <= 1000

(20) ?:<Heidegger>"index=/1000!depth=1/1" # (19), but also restricted to notes with indexes of depth 1

(21) ?:Heidegger"count=3/" # Notes in which "Heidegger" appears at least 3 times

(22) ?:Heidegger"slice=.1.1.1.1.1.1/.1.1.1.1.1.1" # (1), restricted to notes with Index elements rank 2-7 = 1

(23) ?:Heidegger"user=USER" # (1), restricted to notes by USER

(24) ?:Heidegger"size=30/70" # (1), restricted to notes of size <=70 and >=30.

(25) ?:_ALLNOTES_"depth=/1" # Retrieve all notes in search scope with depth <=1

COMPLEX TERMS

(26) ?:Heidegger & Husserl #Notes containing both "Heidegger" and
"Husserl" in the text

(27) ?:<HEIDEGGER> | <HUSSERL> # Notes with keyword "Heidegger" or "Husserl"

(28) ?:(Heidegger & Husserl) & (<HEIDEGGER> | <HUSSERL>)
Must satisfy conditions of (26) and (27)

(29) ?:((Heidegger | Husserl) & Kant) | (Deleuze & Spinoza)

(30) ?:((Heidegger"count=3/4" | Husserl) & Kant"count=5/6")
| (Deleuze & Spinoza) & <#scientist>

REFEEDING

(31) ?:Heidegger => show:? # Execute (1) and feed the result into the show.

LIMITING

(32) ?:Heidegger %%1-100 # Limit (1) to the indexes 1-100

There is no inherent limit to the logical complexity of the search phrase.