A VISUAL REFERENCE · AWK

# AWK,
# one change at a time

From one field to useful text transformations and reports. Complete source. A small change. Its actual result.

The Org file is the canonical source. Read the HTML guide beside your work, or download its PDF and source for offline reading.

> **Last changes: 2026-10-11.** Unified tour–chapter–example IDs, numbered chapter anchors and updated direct links, downloads and comparisons. Reader pages now omit publishing instructions and highlight PDF downloads.

---

**128**
worked examples

**22**
topic families

**1**
canonical Org source

READING THE TOUR

# Three ways to see a change

**Input records**

Inspect the complete local data, field separator and input-file reference.

**AWK program**

Read the whole Org source block, including its header and footer.

**Captured result**

Compare actual output with the input; each example starts a fresh AWK process.

The examples use portable AWK constructs and local input data. Org headers choose input files, command-line arguments, result formatting and tangling; they are separate from the AWK language. Explicit Noweb examples show their expanded program. Input records are not edited on disk; file-writing examples identify their generated output and show its read-back result.

Read from the beginning or jump to a stable example ID. Each example includes its complete input and captured result. Copy link preserves the address of the guide you are reading, including its current domain and path.

# Read this first

This reference starts with one field and builds toward small reports. Every numbered example has a stable ID, Awk code (with explicit expansion for the two reuse examples), an input reference, a focused change, and captured standard output. Each example starts a new Awk process; state does not carry between blocks. The two explicitly marked Noweb examples share a function definition.

The examples use portable Awk constructs and ASCII sample data. They were executed with mawk 1.3.4 in LC_ALL=C. They were not executed inside Emacs or on macOS in this build. GNU-specific features are discussed separately, not silently required. This is a reference, not a set of exercises.

# The execution model

An Awk program is a sequence of pattern/action rules. BEGIN runs once before input. For each record, matching rules run in source order. END runs once afterward. next skips to another record; exit finishes input processing but still runs END.

$0 is the whole record; $1 through $NF are fields. Ordinary variables have no dollar prefix. Input records are not edited on disk. print produces output; saving that output is a separate operation.

The main input has four rows and no header. Fields are name, group, score:

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

# Boundaries and practical traps

- FS="," is not a general CSV parser: quoted commas, embedded newlines and escaped quotes need a real CSV-aware parser. A recent GNU Awk offers --csv, but it is not used here.
- Portable array iteration order is unspecified. The examples explicitly preserve order. GNU asort/asorti and PROCINFO["sorted_in"] are extensions.
- Avoid GNU-only gensub, FPAT, arrays of arrays, and the third match argument in code intended for all Awk implementations. Check your actual interpreter, not just the command name.
- The examples use ASCII, LF line endings, and the C locale. Character counting, case conversion, decimal punctuation, and regex behavior can depend on implementation and locale. CRLF may leave a trailing carriage return.
- Numeric input must be validated for your data model. These examples do not provide a universal decimal/exponent parser. Large integers may lose precision in ordinary floating-point arithmetic.
- Awk regexes use extended regular expressions. Do not assume PCRE escapes such as `\d` or lookbehind. Prefer `[0-9]` and POSIX classes such as `[[:space:]]`.
- print uses OFS and ORS; printf does not. Width is a minimum, not a truncation limit. Escape a literal percent as %% in a printf format.
- Redirection with print value > "out.txt" opens a file and can truncate it when first opened. print value >> "out.txt" appends. close("out.txt") releases it. AWK-21-001 through AWK-21-003 demonstrate these effects on their named files under generated/. They reset their own outputs on each run; they do not edit the input fixtures.
- NR==FNR is a common two-file idiom but misidentifies records when the first file is empty. The lookup example uses the filename explicitly; rename its test if you rename the file.
- Awk has functions, not a portable preprocessor macro system. In Org, Noweb can assemble shared code with :noweb yes and . Most programs are complete. AWK-18-003 and AWK-18-004 explicitly demonstrate Noweb with the shared function below; their expanded standalone scripts are included.

# A quick command index

- Reading: $0, $1, $NF, NF, NR, FNR, FILENAME, FS, RS.
- Writing: print, printf, sprintf, OFS, ORS.
- Selecting: =, !, <, <=, >, >=, ~, !~, &&, ||, !, next.
- Calculating: ~~+, -, *, /, %, +=, +~~, int, BEGIN, END.
- Text: length, substr, index, match, RSTART, RLENGTH, sub, gsub, split, toupper, tolower.
- Structure: if, else, for, exit, function, return, in, associative arrays.
- Org: :in-file, :cmd-line, :results output verbatim, :exports both, :eval never-export, :tangle.

# Shared code for the Noweb examples

The two REUSE examples reference this one named block. In Emacs, C-c C-v v (org-babel-expand-src-block) previews the complete expanded program. :noweb yes expands on execution, tangling and native export. The custom reference preserves the reference and also shows its expanded form. The Python builder supports only the simple named source inclusion used here, not all Org Babel or Noweb features.

```
function twice(x) { return 2 * x }
```

# References and scope

The examples and explanations are original. Consult these primary references for the complete specifications and implementation-specific details:

- GNU Awk manual: https://www.gnu.org/software/gawk/manual/gawk.html
- Org Babel Awk documentation: https://orgmode.org/worg/org-contrib/babel/languages/ob-doc-awk.html
- GNU Awk, file redirection: https://www.gnu.org/software/gawk/manual/html_node/Redirection.html
- GNU Awk, closing streams: https://www.gnu.org/software/gawk/manual/html_node/Close-Files-And-Pipes.html
- Org manual, Noweb syntax: https://orgmode.org/manual/Noweb-Reference-Syntax.html
- Org manual, working with source code: https://orgmode.org/manual/Working-with-Source-Code.html

Version: 1.2. Generated results are verified by executing every numbered block; this is not a claim of cross-platform or Emacs integration testing. No timings or random results are used.

# Records and fields

AWK-01-001

# Print every record

Change: Start with the complete input unchanged.

Awk reads one record at a time. By default a record is one line; $0 is the whole current record. Braces enclose an action.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-01-001
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-01-001.awk :mkdirp yes
```

```
{ print $0 }
```

```
#+end_src
```

Downloads: AWK-01-001.awk

### Actual output

AWK · exit 0

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

AWK-01-002

# Print one field

Change: Replace $0 with $1.

$1 is the first field. The default field separator treats runs of spaces and tabs as one separator. Field numbering starts at 1.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-01-002
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-01-002.awk :mkdirp yes
```

```
{ print $1 }
```

```
#+end_src
```

Downloads: AWK-01-002.awk

### Actual output

AWK · exit 0

```
Ada
Ben
Cara
Dan
```

AWK-01-003

# Choose and reorder fields

Change: Select score first, then name.

A comma separates output expressions. print inserts OFS (one space by default) between them and ORS (a newline) afterward.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-01-003
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-01-003.awk :mkdirp yes
```

```
{ print $3, $1 }
```

```
#+end_src
```

Downloads: AWK-01-003.awk

### Actual output

AWK · exit 0

```
12 Ada
7 Ben
15 Cara
10 Dan
```

AWK-01-004

# Print the last field

Change: Use a computed field position.

NF is the number of fields in the current record. $NF accesses the field at that position; NF alone prints its count.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-01-004
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-01-004.awk :mkdirp yes
{ print $NF }
#+end_src
```

Downloads: AWK-01-004.awk

### Actual output

AWK · exit 0

```
12
7
15
10
```

AWK-01-005

# Show record and field counts

Change: Add context to each record.

NR counts records across all input files. NF is recalculated for each record. Both are built-in variables; neither needs a dollar sign.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-01-005
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-01-005.awk :mkdirp yes
```

```
{ print NR, NF, $1 }
```

```
#+end_src
```

Downloads: AWK-01-005.awk

### Actual output

AWK · exit 0

```
1 3 Ada
2 3 Ben
3 3 Cara
4 3 Dan
```

AWK-01-006

# Run once before input

Change: Add a header without changing the input.

BEGIN runs before any input is read. Rules then run in source order for each record. The quoted text is a string literal.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-01-006
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-01-006.awk :mkdirp yes

BEGIN { print "name score" }
{ print $1, $3 }

#+end_src
```

Downloads: AWK-01-006.awk

### Actual output

AWK · exit 0

```
name score
Ada 12
Ben 7
Cara 15
Dan 10
```

AWK-01-007

# Run once after input

Change: Add a final record count.

END runs after input finishes. It can report accumulated state. NR is zero when there were no input records.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-01-007
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-01-007.awk :mkdirp yes

{ print $1, $3 }
END { print "records", NR }

#+end_src
```

Downloads: AWK-01-007.awk

### Actual output

AWK · exit 0

```
Ada 12
Ben 7
Cara 15
Dan 10
records 4
```

# Patterns and selection

AWK-02-001

# Match a whole line

Change: Keep lines containing labA.

A /regular expression/ pattern tests the complete record. Only matching records trigger its action.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-02-001
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-02-001.awk :mkdirp yes
```

```
/labA/ { print $0 }
```

```
#+end_src
```

Downloads: AWK-02-001.awk

### Actual output

AWK · exit 0

```
Ada labA 12
Cara labA 15
```

AWK-02-002

# Use the default action

Change: Remove the explicit print; output is identical.

A pattern without an action means { print $0 }. An action without a pattern runs for every record.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-02-002
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-02-002.awk :mkdirp yes
/labA/
#+end_src
```

Downloads: AWK-02-002.awk

### Actual output

AWK · exit 0

```
Ada labA 12
Cara labA 15
```

AWK-02-003

# Compare a specific field

Change: Match a field exactly and print only the name.

== compares values. Here the second field must equal the complete string labA. = would assign a value instead.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-02-003
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-02-003.awk :mkdirp yes
```

```
$2 == "labA" { print $1 }
```

```
#+end_src
```

Downloads: AWK-02-003.awk

### Actual output

AWK · exit 0

```
Ada
Cara
```

AWK-02-004

# Use a numeric threshold

Change: Keep scores of at least 10.

>= means greater than or equal to. Input score fields contain numeric text, so comparing them with the numeric constant 10 gives a numeric comparison.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-02-004
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-02-004.awk :mkdirp yes
```

```
$3 >= 10 { print $1, $3 }
```

```
#+end_src
```

Downloads: AWK-02-004.awk

### Actual output

AWK · exit 0

```
Ada 12
Cara 15
Dan 10
```

AWK-02-005

# Combine conditions

Change: Require the group and the threshold together.

&& requires both conditions; || would accept either. Parentheses can group compound conditions.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-02-005
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-02-005.awk :mkdirp yes
```

```
$2 == "labA" && $3 > 12 { print $1 }
```

```
#+end_src
```

Downloads: AWK-02-005.awk

### Actual output

AWK · exit 0

```
Cara
```

AWK-02-006

# Match a field with a regex

Change: Select names starting with A or C.

~ tests a string against a regex. ^ anchors its start; [AC] accepts A or C. !~ is the negated match operator.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-02-006
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-02-006.awk :mkdirp yes
```

```
$1 ~ /^[AC]/ { print $1 }
```

```
#+end_src
```

Downloads: AWK-02-006.awk

### Actual output

AWK · exit 0

```
Ada
Cara
```

AWK-02-007

# Invert a match

Change: Remove informational log records.

!~ means does not match. print with no argument prints $0. ^INFO requires INFO at the beginning.

### Input

#### data/logs.txt

```
INFO start
WARN slow
ERROR disk
INFO done
```

### Complete Org source block

```
#+name: AWK-02-007
#+begin_src awk :in-file data/logs.txt :tangle scripts/AWK-02-007.awk :mkdirp yes
```

```
$0 !~ /^INFO/ { print }
```

```
#+end_src
```

Downloads: AWK-02-007.awk

### Actual output

AWK · exit 0

```
WARN slow
ERROR disk
```

AWK-02-008

# Skip the first record

Change: Treat the first line as a header.

next stops processing the current record and reads another. Later rules do not run for the skipped record.

### Input

#### data/header.txt

```
name group score
Ada labA 12
Ben labB 7
Cara labA 15
```

### Complete Org source block

```
#+name: AWK-02-008
#+begin_src awk :in-file data/header.txt :tangle scripts/AWK-02-008.awk :mkdirp yes

NR == 1 { next }
{ print $1, $3 }

#+end_src
```

Downloads: AWK-02-008.awk

### Actual output

AWK · exit 0

```
Ada 12
Ben 7
Cara 15
```

AWK-02-009

# Take an inclusive range

Change: Keep records two through three.

A comma between patterns creates a range: it starts when the first pattern matches and includes the record matching the second.

### Input

### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-02-009
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-02-009.awk :mkdirp yes
```

```
NR == 2, NR == 3 { print }
```

```
#+end_src
```

Downloads: AWK-02-009.awk

### Actual output

AWK · exit 0

```
Ben labB 7
Cara labA 15
```

# Input and output separators

AWK-03-001

# Observe default whitespace splitting

Change: Try uneven whitespace without changing FS.

The default FS value is a single space with special behavior: repeated spaces/tabs separate fields; leading and trailing whitespace is ignored.

### Input

#### data/spaces.txt

```
  Ada   labA    12
  Ben     labB    7
```

### Complete Org source block

```
#+name: AWK-03-001
#+begin_src awk :in-file data/spaces.txt :tangle scripts/AWK-03-001.awk :mkdirp yes

{ print NF, $1, $3 }

#+end_src
```

Downloads: AWK-03-001.awk

### Actual output

AWK · exit 0

```
3 Ada 12
3 Ben 7
```

AWK-03-002

# Set the input separator

Change: Read semicolon-delimited records.

FS controls input field splitting. Set it in BEGIN so it applies to the first record. Here each semicolon separates fields.

### Input

#### data/delimited.txt

```
Ada;labA;12
Ben;labB;7
```

### Complete Org source block

```
#+name: AWK-03-002
#+begin_src awk :in-file data/delimited.txt :tangle scripts/AWK-03-002.awk :mkdirp yes

BEGIN { FS = ";" }
{ print $1, $3 }

#+end_src
```

Downloads: AWK-03-002.awk

### Actual output

AWK · exit 0

```
Ada 12
Ben 7
```

AWK-03-003

# Set the output separator

Change: Change only the output layout.

OFS controls separators inserted by print between comma-separated expressions. It does not change input parsing.

### Input

#### data/delimited.txt

```
Ada;labA;12
Ben;labB;7
```

### Complete Org source block

```
#+name: AWK-03-003
#+begin_src awk :in-file data/delimited.txt :tangle scripts/AWK-03-003.awk :mkdirp yes

BEGIN { FS = ";"; OFS = " | " }
{ print $1, $3 }

#+end_src
```

Downloads: AWK-03-003.awk

### Actual output

AWK · exit 0

```
Ada | 12
Ben | 7
```

AWK-03-004

# Change record endings

Change: Put names on one line with semicolons.

ORS replaces the newline automatically added by print. printf adds only the characters explicitly requested in its format.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-03-004
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-03-004.awk :mkdirp yes

BEGIN { ORS = "; " }
{ print $1 }
END { printf "\n" }

#+end_src
```

Downloads: AWK-03-004.awk

### Actual output

AWK · exit 0

```
Ada; Ben; Cara; Dan;
```

AWK-03-005

# Read blank-line-separated records

Change: Use one paragraph per record.

RS="" enables paragraph mode: blank lines separate records. FS="\n" then makes the lines within a paragraph into fields.

### Input

#### data/paragraphs.txt

```
Ada
labA

Ben
labB
```

### Complete Org source block

```
#+name: AWK-03-005
#+begin_src awk :in-file data/paragraphs.txt :tangle scripts/AWK-03-005.awk :mkdirp yes

BEGIN { RS = ""; FS = "\n" }
{ print NR, $1, $2 }

#+end_src
```

Downloads: AWK-03-005.awk

### Actual output

AWK · exit 0

```
1 Ada labA
2 Ben labB
```

# Output and formatting

AWK-04-001

# Concatenate strings

Change: Replace the default space with a literal colon.

Adjacent expressions concatenate without an operator. No comma means no automatic separator between these expressions.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-04-001
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-04-001.awk :mkdirp yes
```

```
{ print $1 ":" $3 }
```

```
#+end_src
```

Downloads: AWK-04-001.awk

### Actual output

AWK · exit 0

```
Ada:12
Ben:7
Cara:15
Dan:10
```

AWK-04-002

# Align columns

Change: Use fixed-width, aligned output.

%s formats a string; %d an integer. Widths 8 and 4 are minimum widths. - left-aligns. \n explicitly ends the line.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-04-002
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-04-002.awk :mkdirp yes
```

```
{ printf "%-8s %4d\n", $1, $3 }
```

```
#+end_src
```

Downloads: AWK-04-002.awk

### Actual output

AWK · exit 0

```
Ada      12
Ben       7
Cara     15
Dan      10
```

AWK-04-003

# Control decimal places

Change: Show a calculated value with two decimals.

%f formats a floating-point number. .2 requests two digits after the decimal point. / performs division.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-04-003
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-04-003.awk :mkdirp yes
```

```
{ printf "%s %.2f\n", $1, $3 / 4 }
```

```
#+end_src
```

Downloads: AWK-04-003.awk

### Actual output

AWK · exit 0

```
Ada 3.00
Ben 1.75
Cara 3.75
Dan 2.50
```

AWK-04-004

# Build a formatted string

Change: Store formatted output in a variable.

sprintf returns a string instead of printing. %02d pads an integer to at least two characters using leading zeros.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-04-004
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-04-004.awk :mkdirp yes
```

```
{ label = sprintf("%s:%02d", $1, $3); print label }
```

```
#+end_src
```

Downloads: AWK-04-004.awk

### Actual output

AWK · exit 0

```
Ada:12
Ben:07
Cara:15
Dan:10
```

# Arithmetic and summaries

AWK-05-001

# Accumulate a sum

Change: Replace per-record output with one total.

+= adds to the current value. A new numeric accumulator starts at zero. +0 ensures a numeric zero is printed for empty input.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-05-001
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-05-001.awk :mkdirp yes

{ total += $3 }
END { print total + 0 }

#+end_src
```

Downloads: AWK-05-001.awk

### Actual output

AWK · exit 0

```
44
```

AWK-05-002

# Calculate the mean

Change: Add a count and divide at the end.

n++ increments the count. if (n) guards against division by zero. The mean is the total divided by the number of processed records.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-05-002
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-05-002.awk :mkdirp yes

{ total += $3; n++ }
END { if (n) printf "%.2f\n", total / n; else print "no data" }

#+end_src
```

Downloads: AWK-05-002.awk

### Actual output

AWK · exit 0

```
11.00
```

AWK-05-003

# Count only selected records

Change: Count matches rather than all records.

Only matching records increment n. NR would count all input records, including rejected ones.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-05-003
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-05-003.awk :mkdirp yes

$3 >= 10 { n++ }
END { print n + 0 }

#+end_src
```

Downloads: AWK-05-003.awk

### Actual output

AWK · exit 0

```
3
```

AWK-05-004

# Find a maximum, including negatives

Change: Avoid a misleading zero baseline.

Initialize from the first record. Initializing a maximum to zero would fail when all values are negative. || short-circuits when the first test succeeds.

### Input

#### data/numbers.txt

```
-5
-2
-9
```

### Complete Org source block

```
#+name: AWK-05-004
#+begin_src awk :in-file data/numbers.txt :tangle scripts/AWK-05-004.awk :mkdirp yes

NR == 1 || $1 > maximum { maximum = $1 }
END { if (NR) print maximum; else print "no data" }

#+end_src
```

Downloads: AWK-05-004.awk

### Actual output

AWK · exit 0

```
-2
```

AWK-05-005

# Show a running total

Change: Move output from END into the record action.

State persists between input records within one Awk process. The running total is printed after adding each score.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-05-005
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-05-005.awk :mkdirp yes
```

```
{ total += $3; print $1, total }
```

```
#+end_src
```

Downloads: AWK-05-005.awk

### Actual output

AWK · exit 0

```
Ada 12
Ben 19
Cara 34
Dan 44
```

AWK-05-006

# Round versus truncate

Change: Compare numeric transformation with display formatting.

int truncates toward zero; it does not round to the nearest integer. A %.0f format displays zero fractional digits using rounding.

### Input

**data/empty.txt**

```
(empty file)
```

### Complete Org source block

```
#+name: AWK-05-006
#+begin_src awk :in-file data/empty.txt :tangle scripts/AWK-05-006.awk :mkdirp yes

BEGIN { print int(3.9), int(-3.9); printf "%.0f\n", 3.9 }

#+end_src
```

Downloads: AWK-05-006.awk

### Actual output

AWK · exit 0

```
3 -3
4
```

# Strings and regular expressions

AWK-06-001

# Change case

Change: Transform text without editing the file.

toupper and tolower return transformed strings. ASCII input and the C locale make these examples predictable.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-06-001
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-06-001.awk :mkdirp yes
```

```
{ print toupper($1), tolower($2) }
```

```
#+end_src
```

Downloads: AWK-06-001.awk

### Actual output

AWK · exit 0

```
ADA laba
BEN labb
CARA laba
DAN labb
```

AWK-06-002

# Take a substring

Change: Show the first two letters and the name length.

substr(string,start,length) uses a 1-based start. Omitting length takes the remaining suffix. length returns the string length.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-06-002
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-06-002.awk :mkdirp yes
```

```
{ print substr($1, 1, 2), length($1) }
```

```
#+end_src
```

Downloads: AWK-06-002.awk

### Actual output

AWK · exit 0

```
Ad 3
Be 3
Ca 4
Da 3
```

AWK-06-003

# Find a literal substring

Change: Locate lowercase a in each name.

index returns the 1-based position of the first literal match, or zero when absent. It is case-sensitive and does not interpret regex syntax.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-06-003
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-06-003.awk :mkdirp yes
```

```
{ print $1, index($1, "a") }
```

```
#+end_src
```

Downloads: AWK-06-003.awk

### Actual output

AWK · exit 0

```
Ada 3
Ben 0
Cara 2
Dan 2
```

AWK-06-004

# Replace the first match

Change: Replace only the first lowercase a.

sub replaces the first regex match in its target and returns the number of replacements (zero or one). Here $1 is modified.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-06-004
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-06-004.awk :mkdirp yes
```

```
{ sub(/a/, "@", $1); print $1 }
```

```
#+end_src
```

Downloads: AWK-06-004.awk

### Actual output

AWK · exit 0

```
Ad@
Ben
C@ra
D@n
```

AWK-06-005

# Replace every match

Change: Change only sub to gsub.

gsub replaces every nonoverlapping match in the target. Compare Cara with the preceding example to see the additional replacement.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-06-005
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-06-005.awk :mkdirp yes
```

```
{ gsub(/a/, "@", $1); print $1 }
```

```
#+end_src
```

Downloads: AWK-06-005.awk

### Actual output

AWK · exit 0

```
Ad@
Ben
C@r@
D@n
```

AWK-06-006

# Trim whitespace

Change: Remove leading and trailing spaces or tabs.

^ and $ anchor the start and end. + requires one or more characters. | gives alternatives. Brackets in the output expose the trimmed boundaries.

### Input

#### data/messy.txt

```
  Ada
Ben
```

### Complete Org source block

```
#+name: AWK-06-006
#+begin_src awk :in-file data/messy.txt :tangle scripts/AWK-06-006.awk :mkdirp yes
```

```
{ gsub(/^[     ]+|[    ]+$/, "", $0); print "[" $0 "]" }
```

```
#+end_src
```

Downloads: AWK-06-006.awk

### Actual output

AWK · exit 0

```
[Ada]
[Ben]
```

AWK-06-007

# Extract a regex match

Change: Extract the first sequence of digits.

match sets RSTART and RLENGTH. On failure it returns zero; the condition prevents trying to extract a nonexistent match.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-06-007
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-06-007.awk :mkdirp yes

{ if (match($0, /[0-9]+/))
    print substr($0, RSTART, RLENGTH) }

#+end_src
```

Downloads: AWK-06-007.awk

### Actual output

AWK · exit 0

```
12
7
15
10
```

# Arrays and grouping

AWK-07-001

# Count by group

Change: Keep a separate count for each group.

Associative array keys are strings. Each group has its own counter. Explicit key order keeps this small example deterministic.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-07-001
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-07-001.awk :mkdirp yes

{ count[$2]++ }
END { print "labA", count["labA"] + 0; print "labB", count["labB"] + 0 }

#+end_src
```

Downloads: AWK-07-001.awk

### Actual output

AWK · exit 0

```
labA 2
labB 2
```

AWK-07-002

# Sum by group

Change: Replace ++ with += $3.

An array can accumulate totals as well as counts. Only the update expression changes from the preceding example.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-07-002
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-07-002.awk :mkdirp yes

{ sum[$2] += $3 }
END { print "labA", sum["labA"] + 0; print "labB", sum["labB"] + 0 }

#+end_src
```

Downloads: AWK-07-002.awk

### Actual output

AWK · exit 0

```
labA 27
labB 17
```

AWK-07-003

# Keep the first occurrence

Change: Remove duplicate lines while preserving first-seen order.

A missing array value acts as zero. Post-increment returns the old value; ! makes the first occurrence true. Later occurrences have a nonzero old value.

### Input

#### data/words.txt

```
red
blue
red
green
blue
```

### Complete Org source block

```
#+name: AWK-07-003
#+begin_src awk :in-file data/words.txt :tangle scripts/AWK-07-003.awk :mkdirp yes
```

```
!seen[$0]++ { print }
```

```
#+end_src
```

Downloads: AWK-07-003.awk

### Actual output

AWK · exit 0

```
red
blue
green
```

AWK-07-004

# Remember group order

Change: Report arbitrary groups in their first-seen order.

in checks membership. A second array records first-seen keys. for (key in array) has unspecified order in portable Awk.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-07-004
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-07-004.awk :mkdirp yes

!($2 in seen) { seen[$2] = 1; order[++n] = $2 }
{ count[$2]++ }
END { for (i = 1; i <= n; i++)
    print order[i], count[order[i]] }

#+end_src
```

Downloads: AWK-07-004.awk

### Actual output

AWK · exit 0

```
labA 2
labB 2
```

AWK-07-005

# Split a string into an array

Change: Parse a string independently of input records.

split returns the number of fields and fills array entries 1 through n. Its third argument is the separator pattern.

### Input

### data/empty.txt

```
(empty file)
```

### Complete Org source block

```
#+name: AWK-07-005
#+begin_src awk :in-file data/empty.txt :tangle scripts/AWK-07-005.awk :mkdirp yes

BEGIN { n = split("red:green:blue", a, ":");
    for (i = 1; i <= n; i++) print i, a[i] }

#+end_src
```

Downloads: AWK-07-005.awk

### Actual output

AWK · exit 0

```
1 red
2 green
3 blue
```

# Control flow and functions

AWK-08-001

# Use if and else

Change: Label each row using a threshold.

if selects one action; else provides the alternative. Both paths still process just the current input record.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-08-001
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-08-001.awk :mkdirp yes

{ if ($3 >= 10) print $1, "pass";
  else print $1, "review" }

#+end_src
```

Downloads: AWK-08-001.awk

### Actual output

AWK · exit 0

```
Ada pass
Ben review
Cara pass
Dan pass
```

AWK-08-002

# Use a conditional expression

Change: Condense the preceding if/else into one expression.

condition ? yes : no returns a value. Parentheses make the conditional expression unambiguous inside print.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-08-002
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-08-002.awk :mkdirp yes
```

```
{ print $1, ($3 >= 10 ? "pass" : "review") }
```

```
#+end_src
```

Downloads: AWK-08-002.awk

### Actual output

AWK · exit 0

```
Ada pass
Ben review
Cara pass
Dan pass
```

AWK-08-003

# Loop over fields

Change: Inspect every field of the first record.

for initializes i, tests the bound, and increments after each iteration. $i accesses a field dynamically.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-08-003
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-08-003.awk :mkdirp yes
```

```
NR == 1 { for (i = 1; i <= NF; i++) print i, $i }
```

```
#+end_src
```

Downloads: AWK-08-003.awk

### Actual output

AWK · exit 0

```
1 Ada
2 labA
3 12
```

AWK-08-004

# Stop early

Change: Read just the first two records.

exit stops reading input. END still runs. Use next instead when only the current record should be skipped.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-08-004
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-08-004.awk :mkdirp yes

{ print $1 }
NR == 2 { exit }
END { print "read", NR }

#+end_src
```

Downloads: AWK-08-004.awk

### Actual output

AWK · exit 0

```
Ada
Ben
read 2
```

AWK-08-005

# Define a reusable function

Change: Move a calculation into a function.

function defines a named operation. Parameters are local to the function; scalar arguments are passed by value. return supplies the result.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-08-005
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-08-005.awk :mkdirp yes

function scaled(x, factor) { return x * factor }
{ print $1, scaled($3, 2) }

#+end_src
```

Downloads: AWK-08-005.awk

### Actual output

AWK · exit 0

```
Ada 24
Ben 14
Cara 30
Dan 20
```

# Validation and edge cases

AWK-09-001

# Validate before arithmetic

Change: Reject malformed scores explicitly.

This deliberately narrow validation accepts only nonnegative integer strings. Adapt it for signed values or decimals. Do not silently interpret NA as zero.

### Input

#### data/bad.txt

```
Ada labA 12
Ben labB NA
Cara labA 15
```

### Complete Org source block

```
#+name: AWK-09-001
#+begin_src awk :in-file data/bad.txt :tangle scripts/AWK-09-001.awk :mkdirp yes

$3 !~ /^[0-9]+$/ { print "invalid:", $1; next }
{ total += $3 }
END { print "total", total + 0 }

#+end_src
```

Downloads: AWK-09-001.awk

### Actual output

AWK · exit 0

```
invalid: Ben
total 27
```

AWK-09-002

# Handle an empty file

Change: Run a guarded mean on empty input.

An empty input file triggers BEGIN and END but no record actions. This branch avoids division by zero and states the absence of data.

### Input

#### data/empty.txt

```
(empty file)
```

### Complete Org source block

```
#+name: AWK-09-002
#+begin_src awk :in-file data/empty.txt :tangle scripts/AWK-09-002.awk :mkdirp yes

{ total += $3; n++ }
END { if (n) print total / n; else print "no data" }

#+end_src
```

Downloads: AWK-09-002.awk

### Actual output

AWK · exit 0

```
no data
```

AWK-09-003

# Rebuild a record after editing

Change: Double the score and emit semicolon-separated records.

Assigning to a field rebuilds $0 using OFS. Merely setting OFS does not rewrite an untouched $0. The original input file remains unchanged.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-09-003
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-09-003.awk :mkdirp yes

BEGIN { OFS = ";" }
{ $3 = $3 * 2; print $0 }

#+end_src
```

Downloads: AWK-09-003.awk

### Actual output

AWK · exit 0

```
Ada;labA;24
Ben;labB;14
Cara;labA;30
Dan;labB;20
```

# Org headers and multiple files

AWK-10-001

# Pass a command-line parameter

Change: Move the threshold out of the program.

The block header uses :cmd-line -v limit=12. Awk -v assigns a variable before processing begins, including before BEGIN.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-10-001
#+begin_src awk :in-file data/scores.txt :cmd-line -v limit=12 :tangle scripts/
AWK-10-001.awk :mkdirp yes
```

```
$3 >= limit { print $1, $3 }
```

```
#+end_src
```

Downloads: AWK-10-001.awk

### Actual output

AWK · exit 0

```
Ada 12
Cara 15
```

AWK-10-002

# Compare NR and FNR

Change: Process two inputs in sequence.

The :cmd-line header supplies two files. FNR resets for each file; NR keeps increasing. This block omits :in-file to avoid an extra input file.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

#### data/delimited.txt

```
Ada;labA;12
Ben;labB;7
```

### Complete Org source block

```
#+name: AWK-10-002
#+begin_src awk :cmd-line data/scores.txt data/delimited.txt :tangle scripts/
AWK-10-002.awk :mkdirp yes
```

```
{ print FILENAME, NR, FNR, $1 }
```

```
#+end_src
```

Downloads: AWK-10-002.awk

### Actual output

AWK · exit 0

```
data/scores.txt 1 1 Ada
data/scores.txt 2 2 Ben
data/scores.txt 3 3 Cara
data/scores.txt 4 4 Dan
data/delimited.txt 5 1 Ada;labA;12
data/delimited.txt 6 2 Ben;labB;7
```

AWK-10-003

# Join a small lookup table

Change: Enrich each name with its group location.

Read the lookup file first. FILENAME selects it explicitly, avoiding the empty-first-file trap of NR==FNR. This exact filename must match the command-line argument.

### Input

#### data/lookup.txt

```
labA Aachen
labB Juelich
```

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-10-003
#+begin_src awk :cmd-line data/lookup.txt data/scores.txt :tangle scripts/AWK-10-003.awk :mkdirp
yes

FILENAME == "data/lookup.txt" { place[$1] = $2; next }
{ print $1, ($2 in place ? place[$2] : "UNKNOWN") }

#+end_src
```

Downloads: AWK-10-003.awk

### Actual output

AWK · exit 0

```
Ada Aachen
Ben Juelich
Cara Aachen
Dan Juelich
```

# Putting it together

AWK-11-001

# Create a compact report

Change: Assemble the earlier building blocks into one report.

Combine stable group order, per-group totals and counts, and formatted output. Each group has at least one record, so its denominator cannot be zero.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-11-001
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-11-001.awk :mkdirp yes

BEGIN { printf "%-8s %6s\n", "GROUP", "MEAN" }
!($2 in seen) { seen[$2] = 1; order[++n] = $2 }
{ sum[$2] += $3; count[$2]++ }
END { for (i = 1; i <= n; i++) {
    g = order[i]
    printf "%-8s %6.2f\n", g, sum[g] / count[g]
} }

#+end_src
```

Downloads: AWK-11-001.awk

### Actual output

AWK · exit 0

```
GROUP      MEAN
labA      13.50
labB       8.50
```

AWK-11-002

# Draw text bars

Change: Make the score differences immediately visible.

A loop prints one # per score point. printf avoids newlines between symbols. This simple chart assumes nonnegative integer scores of manageable size.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-11-002
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-11-002.awk :mkdirp yes

{ printf "%-5s ", $1;
  for (i = 0; i < $3; i++) printf "#";
  printf " %d\n", $3 }

#+end_src
```

Downloads: AWK-11-002.awk

### Actual output

AWK · exit 0

```
Ada   ############ 12
Ben   ####### 7
Cara  ############### 15
Dan   ########## 10
```

# More control over fields and values

AWK-12-001

# Change one field

Change: Replace the group field without selecting columns manually.

Assigning to a field rebuilds $0 using OFS. Only the in-memory record changes; the input file is not overwritten.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-12-001
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-12-001.awk :mkdirp yes
```

```
{ $2 = "all"; print }
```

```
#+end_src
```

Downloads: AWK-12-001.awk

### Actual output

AWK · exit 0

```
Ada all 12
Ben all 7
Cara all 15
Dan all 10
```

AWK-12-002

# Append a computed field

Change: Add a fourth field containing twice the score.

$(NF + 1) addresses the first field beyond the record. Assigning there increases NF and rebuilds the record. Parentheses make the computed field index clear.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-12-002
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-12-002.awk :mkdirp yes
```

```
{ $(NF + 1) = $3 * 2; print }
```

```
#+end_src
```

Downloads: AWK-12-002.awk

### Actual output

AWK · exit 0

```
Ada labA 12 24
Ben labB 7 14
Cara labA 15 30
Dan labB 10 20
```

AWK-12-003

# Keep only the first two fields

Change: Drop the score from the current record.

Reducing NF drops trailing fields and rebuilds $0 on the Awk implementations targeted here. Printing $1, $2 is a simpler alternative when no further record editing is needed.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-12-003
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-12-003.awk :mkdirp yes
```

```
{ NF = 2; print }
```

```
#+end_src
```

Downloads: AWK-12-003.awk

### Actual output

AWK · exit 0

```
Ada labA
Ben labB
Cara labA
Dan labB
```

AWK-12-004

# Empty is not the same as remove

Change: Clear the group value while keeping its field position.

The second field still exists after assigning an empty string. NF remains 3. The two adjacent pipes make the empty field visible.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-12-004
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-12-004.awk :mkdirp yes

BEGIN { OFS = "|" }
{ $2 = ""; print NF, $0 }

#+end_src
```

Downloads: AWK-12-004.awk

### Actual output

AWK · exit 0

```
3|Ada||12
3|Ben||7
3|Cara||15
3|Dan||10
```

AWK-12-005

# Remove a middle field

Change: Compare a removed field with an empty field.

Shift every following field one position left, then reduce NF. Unlike clearing $2, this removes the field from the record structure.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-12-005
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-12-005.awk :mkdirp yes

{ for (i = 2; i < NF; i++) $i = $(i + 1)
  NF--; print }

#+end_src
```

Downloads: AWK-12-005.awk

### Actual output

AWK · exit 0

```
Ada 12
Ben 7
Cara 15
Dan 10
```

AWK-12-006

# Modulo: choose alternating records

Change: Print records 1 and 3.

% is the remainder operator. Odd record numbers have remainder 1 when divided by 2. A header would count as a record too.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-12-006
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-12-006.awk :mkdirp yes
```

```
NR % 2 == 1 { print NR, $1 }
```

```
#+end_src
```

Downloads: AWK-12-006.awk

### Actual output

AWK · exit 0

```
1 Ada
3 Cara
```

AWK-12-007

# Choose a column with -v

Change: Make the selected column a parameter.

The header passes -v column=2. $column reads the field whose position is stored in column. Validate dynamic positions in general-purpose scripts.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-12-007
#+begin_src awk :in-file data/scores.txt :cmd-line -v column=2 :tangle scripts/
AWK-12-007.awk :mkdirp yes
```

```
{ print $column }
```

```
#+end_src
```

Downloads: AWK-12-007.awk

### Actual output

AWK · exit 0

```
labA
labB
labA
labB
```

AWK-12-008

# Distinguish text from numeric comparison

Change: Expose conversion rather than relying on implicit types.

Literal strings compare lexically here: "10" precedes "2". Adding zero forces numeric conversion. Awk input fields can have numeric-string attributes, so do not generalize from quoted literals to every field comparison.

### Input

#### data/empty.txt

```
(empty file)
```

### Complete Org source block

```
#+name: AWK-12-008
#+begin_src awk :in-file data/empty.txt :tangle scripts/AWK-12-008.awk :mkdirp yes

BEGIN {
  a = "10"; b = "2"
  print "text:", (a < b)
  print "number:", (a + 0 < b + 0)
}

#+end_src
```

Downloads: AWK-12-008.awk

### Actual output

AWK · exit 0

```
text: 1
number: 0
```

# Regular expressions, one feature at a time

AWK-13-001

# Anchor the end of a field

Change: Compare an anywhere match with an end-anchored match.

$ anchors the end of the tested string. Here Ada and Cara end in lowercase a; Dan contains a but does not end with it.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-13-001
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-13-001.awk :mkdirp yes
```

```
$1 ~ /a$/ { print $1 }
```

```
#+end_src
```

Downloads: AWK-13-001.awk

### Actual output

AWK · exit 0

```
Ada
Cara
```

AWK-13-002

# Match alternatives

Change: Accept either of two complete group names.

Parentheses group alternatives. ^ and $ anchor the entire group, so only the complete alternatives match. ^labA|labB$ would anchor the alternatives differently.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-13-002
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-13-002.awk :mkdirp yes
```

```
$2 ~ /^(labA|labB)$/ { print $1, $2 }
```

```
#+end_src
```

Downloads: AWK-13-002.awk

### Actual output

AWK · exit 0

```
Ada labA
Ben labB
Cara labA
Dan labB
```

AWK-13-003

# Use a character class

Change: Select records starting with an uppercase letter.

[[:upper:]] is a POSIX character class for uppercase letters. Its interpretation can depend on the locale. The examples run with LC_ALL=C.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-13-003
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-13-003.awk :mkdirp yes
```

```
$0 ~ /^[[:upper:]]/ { print $1 }
```

```
#+end_src
```

Downloads: AWK-13-003.awk

### Actual output

AWK · exit 0

```
Ada
Ben
Cara
Dan
```

AWK-13-004

# Pass a dynamic pattern

Change: Keep the program fixed and change the pattern.

The header passes -v pattern=^A. A variable on the right of ~ is interpreted as a regular expression. Prefer /.../ for fixed patterns; dynamic strings introduce another escaping layer.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-13-004
#+begin_src awk :in-file data/scores.txt :cmd-line -v pattern=^A :tangle scripts/
AWK-13-004.awk :mkdirp yes
```

```
$1 ~ pattern { print $1 }
```

```
#+end_src
```

Downloads: AWK-13-004.awk

### Actual output

AWK · exit 0

```
Ada
```

AWK-13-005

# Match a literal dot

Change: Select filenames containing a dot.

An unescaped dot in a regex matches any character. [.] is a convenient way to match a literal dot without a backslash.

### Input

#### data/filenames.txt

```
report.txt
README
plot.svg
archive
```

### Complete Org source block

```
#+name: AWK-13-005
#+begin_src awk :in-file data/filenames.txt :tangle scripts/AWK-13-005.awk :mkdirp yes
```

```
$1 ~ /[.]/ { print $1 }
```

```
#+end_src
```

Downloads: AWK-13-005.awk

### Actual output

AWK · exit 0

```
report.txt
plot.svg
```

AWK-13-006

# Reuse the complete match in replacement

Change: Wrap every digit sequence in brackets.

In sub/gsub replacement text, & inserts the complete matched text. Omitting the third argument modifies $0. Literal ampersands need additional escaping.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-13-006
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-13-006.awk :mkdirp yes
```

```
{ gsub(/[0-9]+/, "[&]"); print }
```

```
#+end_src
```

Downloads: AWK-13-006.awk

### Actual output

AWK · exit 0

```
Ada labA [12]
Ben labB [7]
Cara labA [15]
Dan labB [10]
```

# Array membership, cross-tabs and ordering

AWK-14-001

# Test membership without testing the value

Change: Separate key existence from truthiness.

in tests whether a key exists, even if its value is zero or empty. Testing the stored value instead would confuse a present zero with a missing value.

### Input

**data/empty.txt**

```
(empty file)
```

### Complete Org source block

```
#+name: AWK-14-001
#+begin_src awk :in-file data/empty.txt :tangle scripts/AWK-14-001.awk :mkdirp yes

BEGIN {
  value["Ada"] = 0
  print ("Ada" in value), ("Ben" in value)
  print (value["Ada"] != 0)
}

#+end_src
```

Downloads: AWK-14-001.awk

### Actual output

AWK · exit 0

```
1 0
0
```

AWK-14-002

# Delete a single key

Change: Remove membership, not just the stored value.

delete array[key] removes one entry. Assigning "" would leave the key present. Avoid whole-array delete syntax when targeting older Awk implementations.

### Input

#### data/empty.txt

```
(empty file)
```

### Complete Org source block

```
#+name: AWK-14-002
#+begin_src awk :in-file data/empty.txt :tangle scripts/AWK-14-002.awk :mkdirp yes

BEGIN {
  value["Ada"] = 12
  delete value["Ada"]
  print ("Ada" in value)
}

#+end_src
```

Downloads: AWK-14-002.awk

### Actual output

AWK · exit 0

```
0
```

AWK-14-003

# Count combinations of two keys

Change: Build a small cross-tabulation.

Portable multidimensional syntax combines subscripts using SUBSEP; it is not an array of arrays. Fixed output order is deliberate. Columns are group, pass count, review count.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-14-003
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-14-003.awk :mkdirp yes

{ count[$2, ($3 >= 10 ? "pass" : "review")]++ }
END {
  print "labA", count["labA", "pass"] + 0, count["labA", "review"] + 0
  print "labB", count["labB", "pass"] + 0, count["labB", "review"] + 0
}

#+end_src
```

Downloads: AWK-14-003.awk

### Actual output

AWK · exit 0

```
labA 2 0
labB 1 1
```

AWK-14-004

# Find a maximum per group

Change: Track both a winning value and its associated name.

Membership initializes each group from its own first observation. A strict > comparison keeps the earlier record when maximum values tie.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-14-004
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-14-004.awk :mkdirp yes

!($2 in best) || $3 > best[$2] {
  best[$2] = $3; name[$2] = $1
}
END { print name["labA"], best["labA"]
      print name["labB"], best["labB"] }

#+end_src
```

Downloads: AWK-14-004.awk

### Actual output

AWK · exit 0

```
Cara 15
Dan 10
```

AWK-14-005

# Sort a small set numerically

Change: Produce a deterministic numeric ordering without GNU extensions.

Insertion sort is portable but quadratic in the worst case. This is suitable for tiny demonstrations; prefer an external sort or GNU asort for large collections. All values here are numeric.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-14-005
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-14-005.awk :mkdirp yes

{ a[++n] = $3 }
END {
  for (i = 2; i <= n; i++) {
    x = a[i]; j = i - 1
    while (j >= 1 && a[j] > x) { a[j + 1] = a[j]; j-- }
    a[j + 1] = x
  }
  for (i = 1; i <= n; i++) print a[i]
}

#+end_src
```

Downloads: AWK-14-005.awk

### Actual output

AWK · exit 0

```
7
10
12
15
```

AWK-14-006

# Emit a histogram with fixed bins

Change: Aggregate scores into ranges instead of exact groups.

Two counters represent two bins. The label 0-9 assumes nonnegative integer data; change the labels or validation for other input domains.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-14-006
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-14-006.awk :mkdirp yes

{ if ($3 < 10) low++; else high++ }
END { print "0-9", low + 0; print "10+", high + 0 }

#+end_src
```

Downloads: AWK-14-006.awk

### Actual output

AWK · exit 0

```
0-9 1
10+ 3
```

# State and streaming transformations

AWK-15-001

# Print a difference from the previous row

Change: Compare each score with the preceding score.

Rules run in source order. Calculate the difference before replacing previous. The first row initializes state and produces no difference.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-15-001
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-15-001.awk :mkdirp yes

NR > 1 { print $1, $3 - previous }
{ previous = $3 }

#+end_src
```

Downloads: AWK-15-001.awk

### Actual output

AWK · exit 0

```
Ben -5
Cara 8
Dan -5
```

AWK-15-002

# Calculate a three-record moving mean

Change: Bound memory while maintaining a rolling statistic.

A three-slot ring buffer discards the oldest value as each new one arrives. The first two windows are partial; thereafter the mean always uses three records.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-15-002
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-15-002.awk :mkdirp yes

{ slot = (NR - 1) % 3
  sum -= window[slot]; window[slot] = $3; sum += $3
  n = (NR < 3 ? NR : 3)
  printf "%s %.2f\n", $1, sum / n
}

#+end_src
```

Downloads: AWK-15-002.awk

### Actual output

AWK · exit 0

```
Ada 12.00
Ben 9.50
Cara 11.33
Dan 10.67
```

AWK-15-003

# Start and stop a section

Change: Extract section content without emitting delimiters.

A state variable selects records between marker lines. next keeps the marker lines out of the output. This simple format does not support nested sections.

### Input

#### data/sections.txt

```
outside
START
alpha
beta
STOP
after
```

### Complete Org source block

```
#+name: AWK-15-003
#+begin_src awk :in-file data/sections.txt :tangle scripts/AWK-15-003.awk :mkdirp yes

/^START$/ { active = 1; next }
/^STOP$/ { active = 0; next }
active { print }

#+end_src
```

Downloads: AWK-15-003.awk

### Actual output

AWK · exit 0

```
alpha
beta
```

AWK-15-004

# Compare with a range pattern

Change: Replace explicit state with an inclusive range.

Range patterns include the starting and ending records. Compare this output with the state-based example, which excludes both marker lines.

### Input

#### data/sections.txt

```
outside
START
alpha
beta
STOP
after
```

### Complete Org source block

```
#+name: AWK-15-004
#+begin_src awk :in-file data/sections.txt :tangle scripts/AWK-15-004.awk :mkdirp yes

/^START$/, /^STOP$/ { print }

#+end_src
```

Downloads: AWK-15-004.awk

### Actual output

AWK · exit 0

```
START
alpha
beta
STOP
```

AWK-15-005

# Remember the latest section header

Change: Attach context to subsequent records.

Header records update section; later nonempty records inherit it. NF skips empty lines. The fixture begins with a header; add validation if that cannot be assumed.

### Input

#### data/config.txt

```
[labA]
Ada
Cara

[labB]
Ben
Dan
```

### Complete Org source block

```
#+name: AWK-15-005
#+begin_src awk :in-file data/config.txt :tangle scripts/AWK-15-005.awk :mkdirp yes

/^\[/ { section = $0; next }
NF { print section, $0 }

#+end_src
```

Downloads: AWK-15-005.awk

### Actual output

AWK · exit 0

```
[labA] Ada
[labA] Cara
[labB] Ben
[labB] Dan
```

AWK-15-006

# Merge consecutive duplicate lines

Change: Use constant state instead of a growing seen array.

Only adjacent duplicates are removed. A later reappearance is kept. Compare with AWK-07-003, which removes duplicates across the entire input.

### Input

#### data/runs.txt

```
red
red
blue
red
```

### Complete Org source block

```
#+name: AWK-15-006
#+begin_src awk :in-file data/runs.txt :tangle scripts/AWK-15-006.awk :mkdirp yes

NR == 1 || $0 != previous { print }
{ previous = $0 }

#+end_src
```

Downloads: AWK-15-006.awk

### Actual output

AWK · exit 0

```
red
blue
red
```

# Several files and explicit joins

AWK-16-001

# Skip one header per file

Change: Repeat header handling for every input file.

FNR resets to 1 at each file, so this skips each file header. NR==1 would skip only the first header across the complete input stream.

### Input

#### data/header.txt

```
name group score
Ada labA 12
Ben labB 7
Cara labA 15
```

#### data/header2.txt

```
name group score
Eve labC 8
Finn labC 14
```

### Complete Org source block

```
#+name: AWK-16-001
#+begin_src awk :cmd-line data/header.txt data/header2.txt :tangle scripts/AWK-16-001.awk :mkdirp
yes
```

```
FNR == 1 { next }
{ print $1, $3 }
```

```
#+end_src
```

Downloads: AWK-16-001.awk

### Actual output

AWK · exit 0

```
Ada 12
Ben 7
Cara 15
Eve 8
Finn 14
```

AWK-16-002

# Keep only rows with a lookup match

Change: Drop unmatched groups explicitly.

This inner join emits only matching score records. Multiple lookup rows with the same key would overwrite earlier values; this fixture has unique lookup keys.

## Input

### data/lookup.txt

```
labA Aachen
labB Juelich
```

### data/unmatched.txt

```
Ada labA 12
Eve labC 8
```

## Complete Org source block

```
#+name: AWK-16-002
#+begin_src awk :cmd-line data/lookup.txt data/unmatched.txt :tangle scripts/
AWK-16-002.awk :mkdirp yes

FILENAME == "data/lookup.txt" { city[$1] = $2; next }
$2 in city { print $1, city[$2] }

#+end_src
```

Downloads: AWK-16-002.awk

## Actual output

AWK · exit 0

```
Ada Aachen
```

AWK-16-003

# Preserve rows without a lookup match

Change: Change only the treatment of unmatched groups.

This left join keeps every record from the second file. The placeholder distinguishes a missing lookup from an omitted output row.

### Input

#### data/lookup.txt

```
labA Aachen
labB Juelich
```

#### data/unmatched.txt

```
Ada labA 12
Eve labC 8
```

### Complete Org source block

```
#+name: AWK-16-003
#+begin_src awk :cmd-line data/lookup.txt data/unmatched.txt :tangle scripts/
AWK-16-003.awk :mkdirp yes

FILENAME == "data/lookup.txt" { city[$1] = $2; next }
{ print $1, ($2 in city ? city[$2] : "UNKNOWN") }

#+end_src
```

Downloads: AWK-16-003.awk

### Actual output

AWK · exit 0

```
Ada Aachen
Eve UNKNOWN
```

AWK-16-004

# Read a side file with getline

Change: Load a lookup without making it part of the main input sequence.

getline line < path reads a separate stream into line. It returns 1 for a record, 0 at EOF, -1 on error; it does not replace $0 or advance NR/FNR. close releases the file. The normal input remains scores.txt.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

#### data/lookup.txt

```
labA Aachen
labB Juelich
```

### Complete Org source block

```
#+name: AWK-16-004
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-16-004.awk :mkdirp yes
```

```
BEGIN {
  path = "data/lookup.txt"
  while ((status = (getline line < path)) > 0) {
    split(line, fields); city[fields[1]] = fields[2]
  }
  close(path)
  if (status < 0) { print "lookup read error"; exit 1 }
}
{ print $1, ($2 in city ? city[$2] : "UNKNOWN") }
```

```
#+end_src
```

Downloads: AWK-16-004.awk

### Actual output

AWK · exit 0

```
Ada Aachen
Ben Juelich
Cara Aachen
Dan Juelich
```

# Small statistical summaries

AWK-17-001

# Find a minimum and a maximum

Change: Report the observed range.

Initialize both extrema from real input. This works for negative scores too, unlike an arbitrary starting value of zero.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-17-001
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-17-001.awk :mkdirp yes

NR == 1 { low = high = $3 }
{ if ($3 < low) low = $3; if ($3 > high) high = $3 }
END { if (NR) print low, high; else print "no data" }

#+end_src
```

Downloads: AWK-17-001.awk

### Actual output

AWK · exit 0

```
7 15
```

AWK-17-002

# Compute an online mean

Change: Update the mean record by record.

The incremental update maintains a mean using constant memory. It avoids a separate potentially large total but still uses finite-precision arithmetic.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-17-002
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-17-002.awk :mkdirp yes
```

```
{ n++; mean += ($3 - mean) / n }
END { if (n) printf "%.2f\n", mean; else print "no data" }
```

```
#+end_src
```

Downloads: AWK-17-002.awk

### Actual output

AWK · exit 0

```
11.00
```

AWK-17-003

# Compute sample standard deviation

Change: Extend the online mean with a measure of spread.

Welford-style updates accumulate squared deviations without subtracting two large sums. n-1 gives the sample variance; sqrt converts it to standard deviation.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-17-003
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-17-003.awk :mkdirp yes

{ n++; delta = $3 - mean
  mean += delta / n
  m2 += delta * ($3 - mean)
}
END { if (n > 1) printf "%.3f\n", sqrt(m2 / (n - 1))
      else print "need two values" }

#+end_src
```

Downloads: AWK-17-003.awk

### Actual output

AWK · exit 0

```
3.367
```

AWK-17-004

# Normalize against a reference

Change: Express each score as a percentage of a known maximum.

The header supplies -v maximum=20. %% prints a literal percent sign in printf. A configurable denominator must be checked for zero in a general-purpose tool.

### Input

**data/scores.txt**

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-17-004
#+begin_src awk :in-file data/scores.txt :cmd-line -v maximum=20 :tangle scripts/
AWK-17-004.awk :mkdirp yes
```

```
{ printf "%s %.1f%%\n", $1, 100 * $3 / maximum }
```

```
#+end_src
```

Downloads: AWK-17-004.awk

### Actual output

AWK · exit 0

```
Ada 60.0%
Ben 35.0%
Cara 75.0%
Dan 50.0%
```

# Functions and literate reuse

AWK-18-001

# Give a function a local scratch variable

Change: Prevent a helper function from modifying global scratch state.

All function parameters are local. The unpassed extra parameter text acts as a local scratch variable; spacing before it is just a convention. The global text retains its value.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-18-001
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-18-001.awk :mkdirp yes

function label(name,    text) {
  text = "[" name "]"; return text
}
BEGIN { text = "outside" }
{ print label($1) }
END { print text }

#+end_src
```

Downloads: AWK-18-001.awk

### Actual output

AWK · exit 0

```
[Ada]
[Ben]
[Cara]
[Dan]
outside
```

AWK-18-002

# Pass an array to a function

Change: Reuse an operation that updates a shared accumulator.

Arrays are passed by reference: changing a[key] changes the caller's array. Scalar arguments are passed by value. Do not pass a scalar where the function expects an array.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-18-002
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-18-002.awk :mkdirp yes

function increment(a, key) { a[key]++ }
{ increment(count, $2) }
END { print count["labA"], count["labB"] }

#+end_src
```

Downloads: AWK-18-002.awk

### Actual output

AWK · exit 0

```
2 2
```

AWK-18-003

# Reuse a named Noweb block

Change: Reuse source text while keeping the actual output visible.

The header sets :noweb yes. Org inserts the named awk-twice source block before execution or tangling. Its definition is printed in the shared-code appendix. The generated .awk script contains the expanded function.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-18-003
#+begin_src awk :in-file data/scores.txt :noweb yes :tangle scripts/AWK-18-003.awk :mkdirp yes

<<awk-twice>>
{ print $1, twice($3) }

#+end_src
```

### Actual output

AWK · exit 0

```
Ada 24
Ben 14
Cara 30
Dan 20
```

### Expanded program (Org Babel Noweb)

```
function twice(x) { return 2 * x }
{ print $1, twice($3) }
```

Downloads: AWK-18-003.awk

AWK-18-004

# Reuse the function in a summary

Change: Use one shared implementation in a second context.

The same named function now participates in an aggregate. A source edit in awk-twice changes both examples at the next rebuild. This is text inclusion by Org, not an Awk macro feature.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-18-004
#+begin_src awk :in-file data/scores.txt :noweb yes :tangle scripts/AWK-18-004.awk :mkdirp yes

<<awk-twice>>
{ total += twice($3) }
END { print total + 0 }

#+end_src
```

### Actual output

AWK · exit 0

```
88
```

### Expanded program (Org Babel Noweb)

```
function twice(x) { return 2 * x }
{ total += twice($3) }
END { print total + 0 }
```

Downloads: AWK-18-004.awk

AWK-18-005

# Inspect the input without running it as code

Change: Inspect record shape as a small validation report.

An Awk data file is read as records, not executed as program text. Program text comes from the source block (or the script supplied with -f). Keep code and input paths distinct in automated pipelines.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-18-005
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-18-005.awk :mkdirp yes
{ print "line", NR, "fields", NF }
#+end_src
```

Downloads: AWK-18-005.awk

### Actual output

AWK · exit 0

```
line 1 fields 3
line 2 fields 3
line 3 fields 3
line 4 fields 3
```

AWK-18-006

# Emit a reusable tab-separated result

Change: Create a small, documented handover format.

A tab character separates each column. This fixture contains no embedded tabs or newlines in values. The captured output can feed another tool; arbitrary text needs an explicit escaping or serialization policy.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-18-006
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-18-006.awk :mkdirp yes

BEGIN { OFS = "\t"; print "name", "group", "double_score" }
{ print $1, $2, $3 * 2 }

#+end_src
```

Downloads: AWK-18-006.awk

### Actual output

AWK · exit 0

```
name    group   double_score
Ada     labA    24
Ben     labB    14
Cara    labA    30
Dan     labB    20
```

# Cleaning and validating real-world records

AWK-19-001

# Ignore comments and blank lines

Change: Remove scaffolding while preserving data.

The first rule skips lines whose first non-whitespace character is #. NF rejects blank records. A # inside a data value is not treated as a comment.

### Input

#### data/comments.txt

```
# measurements

Ada labA 12
  # provisional
Ben labB 7
```

### Complete Org source block

```
#+name: AWK-19-001
#+begin_src awk :in-file data/comments.txt :tangle scripts/AWK-19-001.awk :mkdirp yes

/^[[:space:]]*#/ { next }
NF { print }

#+end_src
```

Downloads: AWK-19-001.awk

### Actual output

AWK · exit 0

```
Ada labA 12
Ben labB 7
```

AWK-19-002

# Check the expected field count

Change: Validate record shape before doing arithmetic.

NF is the field count after splitting. Requiring exactly three fields catches missing and surplus columns, but does not validate their meanings.

### Input

#### data/ragged.txt

```
Ada labA 12
Ben labB
Cara labA 15 extra
```

### Complete Org source block

```
#+name: AWK-19-002
#+begin_src awk :in-file data/ragged.txt :tangle scripts/AWK-19-002.awk :mkdirp yes

NF != 3 { print "reject line", NR, "fields", NF; next }
{ print "accept", $1 }

#+end_src
```

Downloads: AWK-19-002.awk

### Actual output

AWK · exit 0

```
accept Ada
reject line 2 fields 2
reject line 3 fields 4
```

AWK-19-003

# Validate signed decimals and exponents

Change: Accept a documented numeric grammar.

The optional sign is followed by digits with an optional fractional part, or a leading dot plus digits. An optional exponent has its own optional sign. This grammar excludes NaN, infinity, hexadecimal and thousands separators.

### Input

#### data/numeric.txt

```
12
-2.5
.75
1e3
NA
12x
```

### Complete Org source block

```
#+name: AWK-19-003
#+begin_src awk :in-file data/numeric.txt :tangle scripts/AWK-19-003.awk :mkdirp yes

$1 ~ /^[+-]?([0-9]+([.][0-9]*)?|[.][0-9]+)([eE][+-]?[0-9]+)?$/ {
  printf "number %.3f\n", $1 + 0; next
}
{ print "reject", $1 }

#+end_src
```

Downloads: AWK-19-003.awk

### Actual output

AWK · exit 0

```
number 12.000
number -2.500
number 0.750
number 1000.000
reject NA
reject 12x
```

AWK-19-004

# Treat missing values separately from zero

Change: Do not turn absent measurements into zero.

With a literal semicolon separator, an empty third field remains distinguishable from a numeric zero. The fixture uses only numbers, empty values and NA; arbitrary other text would need numeric validation.

### Input

#### data/missing.txt

```
Ada;labA;12
Ben;labB;NA
Cara;labA;
Dan;labB;0
```

### Complete Org source block

```
#+name: AWK-19-004
#+begin_src awk :in-file data/missing.txt :tangle scripts/AWK-19-004.awk :mkdirp yes

BEGIN { FS = ";" }
$3 == "" || $3 == "NA" { print $1, "missing"; next }
{ print $1, "present", $3 + 0 }

#+end_src
```

Downloads: AWK-19-004.awk

### Actual output

AWK · exit 0

```
Ada present 12
Ben missing
Cara missing
Dan present 0
```

AWK-19-005

# Compute a mean over observed values

Change: Aggregate the preceding classification.

The denominator counts observed values, including zero. Missing values are counted separately. Reporting both counts makes the mean easier to interpret.

### Input

#### data/missing.txt

```
Ada;labA;12
Ben;labB;NA
Cara;labA;
Dan;labB;0
```

### Complete Org source block

```
#+name: AWK-19-005
#+begin_src awk :in-file data/missing.txt :tangle scripts/AWK-19-005.awk :mkdirp yes

BEGIN { FS = ";" }
$3 == "" || $3 == "NA" { missing++; next }
{ sum += $3; n++ }
END {
  if (n) printf "mean %.2f\n", sum / n
  else print "no observations"
  print "observed", n + 0, "missing", missing + 0
}

#+end_src
```

Downloads: AWK-19-005.awk

### Actual output

AWK · exit 0

```
mean 6.00
observed 2 missing 2
```

AWK-19-006

# Choose a column by its header

Change: Stop hard-coding a column number.

The first row maps header names to positions. Column order may change without changing the lookup expression. Duplicate header names would overwrite earlier positions; the next example detects them.

### Input

#### data/reordered.txt

```
score name group
12 Ada labA
7 Ben labB
```

### Complete Org source block

```
#+name: AWK-19-006
#+begin_src awk :in-file data/reordered.txt :tangle scripts/AWK-19-006.awk :mkdirp yes

NR == 1 {
  for (i = 1; i <= NF; i++) column[$i] = i
  if (!("score" in column)) { print "missing score column"; exit 1 }
  next
}
{ print $(column["score"]) }

#+end_src
```

Downloads: AWK-19-006.awk

### Actual output

AWK · exit 0

```
12
7
```

AWK-19-007

# Detect duplicate column names

Change: Check the assumption behind name-based access.

A header-level seen array reports each repeated name. exit stops after the header because no data rows are needed for this check.

### Input

#### data/duplicateheader.txt

```
name score score
Ada 12 13
```

### Complete Org source block

```
#+name: AWK-19-007
#+begin_src awk :in-file data/duplicateheader.txt :tangle scripts/AWK-19-007.awk :mkdirp yes

NR == 1 {
  for (i = 1; i <= NF; i++) {
    if ($i in seen) print "duplicate", $i, "at", i
    seen[$i] = 1
  }
  exit
}

#+end_src
```

Downloads: AWK-19-007.awk

### Actual output

AWK · exit 0

```
duplicate score at 3
```

AWK-19-008

# Normalize a simple decimal-comma field

Change: Separate the delimiter from the decimal convention.

Under LC_ALL=C, numeric conversion expects a decimal point. This conversion assumes exactly one decimal comma, no thousands separator, and otherwise valid numeric text; it is not a general locale parser.

### Input

#### data/decimalcomma.txt

```
Ada;12,5
Ben;7,25
```

### Complete Org source block

```
#+name: AWK-19-008
#+begin_src awk :in-file data/decimalcomma.txt :tangle scripts/AWK-19-008.awk :mkdirp yes

BEGIN { FS = ";" }
{ value = $2; sub(/,/, ".", value)
  printf "%s %.2f\n", $1, value + 0
}

#+end_src
```

Downloads: AWK-19-008.awk

### Actual output

AWK · exit 0

```
Ada 12.50
Ben 7.25
```

# Reshaping small tables

AWK-20-001

# Turn two measurement columns into rows

Change: Change a wide table into a long table.

Each wide record produces two long records. The sample ID is repeated; each measurement name comes from the header. This version deliberately fixes the number of measurement columns.

### Input

#### data/wide.txt

```
sample before after
A 2 5
B 4 6
```

### Complete Org source block

```
#+name: AWK-20-001
#+begin_src awk :in-file data/wide.txt :tangle scripts/AWK-20-001.awk :mkdirp yes

NR == 1 { first = $2; second = $3; next }
{ print $1, first, $2
  print $1, second, $3 }

#+end_src
```

Downloads: AWK-20-001.awk

### Actual output

AWK · exit 0

```
A before 2
A after 5
B before 4
B after 6
```

AWK-20-002

# Unpivot any number of measurement columns

Change: Generalize only the measurement count.

The loop replaces two explicit print statements. This assumes every data row follows the same header and has the expected number of fields; use a field-count check for irregular input.

### Input

#### data/wide.txt

```
sample before after
A 2 5
B 4 6
```

### Complete Org source block

```
#+name: AWK-20-002
#+begin_src awk :in-file data/wide.txt :tangle scripts/AWK-20-002.awk :mkdirp yes

NR == 1 {
  for (i = 2; i <= NF; i++) measurement[i] = $i
  next
}
{ for (i = 2; i <= NF; i++) print $1, measurement[i], $i }

#+end_src
```

Downloads: AWK-20-002.awk

### Actual output

AWK · exit 0

```
A before 2
A after 5
B before 4
B after 6
```

AWK-20-003

# Pivot known measurement names into columns

Change: Reverse the long representation.

A composite key stores each sample/measurement pair. First-seen sample order is retained. The fixture has every pair exactly once; duplicate pairs would keep the last value.

### Input

#### data/long.txt

```
A before 2
A after 5
B before 4
B after 6
```

### Complete Org source block

```
#+name: AWK-20-003
#+begin_src awk :in-file data/long.txt :tangle scripts/AWK-20-003.awk :mkdirp yes

!($1 in seen) { seen[$1] = 1; order[++n] = $1 }
{ value[$1, $2] = $3 }
END {
  print "sample", "before", "after"
  for (i = 1; i <= n; i++) {
    id = order[i]; print id, value[id, "before"], value[id, "after"]
  }
}

#+end_src
```

Downloads: AWK-20-003.awk

### Actual output

AWK · exit 0

```
sample before after
A 2 5
B 4 6
```

AWK-20-004

# Mark a missing pivot cell explicitly

Change: Avoid silent blanks when a pair is absent.

Parenthesized subscripts can test membership of a composite key. This distinguishes an absent cell from a present zero. Missing-value spelling is a deliberate output convention.

### Input

#### data/longmissing.txt

```
A before 2
A after 5
B before 0
```

### Complete Org source block

```
#+name: AWK-20-004
#+begin_src awk :in-file data/longmissing.txt :tangle scripts/AWK-20-004.awk :mkdirp yes

!($1 in seen) { seen[$1] = 1; order[++n] = $1 }
{ value[$1, $2] = $3 }
END {
  for (i = 1; i <= n; i++) {
    id = order[i]
    a = ((id, "before") in value ? value[id, "before"] : "NA")
    b = ((id, "after") in value ? value[id, "after"] : "NA")
    print id, a, b
  }
}

#+end_src
```

Downloads: AWK-20-004.awk

### Actual output

AWK · exit 0

```
A 2 5
B 0 NA
```

AWK-20-005

# Transpose a small rectangular table

Change: Swap the two dimensions of a tiny table.

Rows become columns, including the original header. All cells are retained in memory. This example assumes a rectangular whitespace-separated table without spaces inside individual values.

### Input

#### data/wide.txt

```
sample before after
A 2 5
B 4 6
```

### Complete Org source block

```
#+name: AWK-20-005
#+begin_src awk :in-file data/wide.txt :tangle scripts/AWK-20-005.awk :mkdirp yes

{ rows++; columns = NF
  for (i = 1; i <= NF; i++) cell[rows, i] = $i
}
END {
  for (i = 1; i <= columns; i++) {
    for (j = 1; j <= rows; j++)
      printf "%s%s", (j == 1 ? "" : " "), cell[j, i]
    printf "\n"
  }
}

#+end_src
```

Downloads: AWK-20-005.awk

### Actual output

AWK · exit 0

```
sample A B
before 2 4
after 5 6
```

AWK-20-006

# Summarize consecutive groups

Change: Use constant memory when the input is grouped.

Only the current group total is stored. Identical keys must be contiguous; otherwise the same key produces more than one summary row. END flushes the last group.

### Input

#### data/grouped.txt

```
Ada labA 12
Cara labA 15
Ben labB 7
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-20-006
#+begin_src awk :in-file data/grouped.txt :tangle scripts/AWK-20-006.awk :mkdirp yes

NR > 1 && $2 != group { print group, total; total = 0 }
{ group = $2; total += $3 }
END { if (NR) print group, total }

#+end_src
```

Downloads: AWK-20-006.awk

### Actual output

AWK · exit 0

```
labA 27
labB 17
```

AWK-20-007

# Split a list-valued field into rows

Change: Expand one list value into several records.

A semicolon separates the ID from a comma-separated list. split handles the inner list. There is no quoting or escaping in this deliberately simple format.

### Input

#### data/lists.txt

```
A;red,blue
B;green
```

### Complete Org source block

```
#+name: AWK-20-007
#+begin_src awk :in-file data/lists.txt :tangle scripts/AWK-20-007.awk :mkdirp yes

BEGIN { FS = ";" }
{ n = split($2, item, /,/)
  for (i = 1; i <= n; i++) print $1, item[i]
}

#+end_src
```

Downloads: AWK-20-007.awk

### Actual output

AWK · exit 0

```
A red
A blue
B green
```

AWK-20-008

# Number rows within each group

Change: Add a per-group sequence number.

Each group has its own counter, so records do not need to be contiguous. Prefix increment returns the new value, making the first record number 1.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-20-008
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-20-008.awk :mkdirp yes
```

```
{ print $2, ++number[$2], $1 }
```

```
#+end_src
```

Downloads: AWK-20-008.awk

### Actual output

AWK · exit 0

```
labA 1 Ada
labB 1 Ben
labA 2 Cara
labB 2 Dan
```

# Output files and interchange formats

AWK-21-001

# Write a new result file

Change: Write selected columns and inspect the resulting file.

The first > opens and truncates this named output; subsequent writes keep using the open stream. END closes it and reads it back so the displayed output is the actual file content. The generated directory must already exist.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-21-001
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-21-001.awk :mkdirp yes

BEGIN { out = "generated/AWK-21-001.txt" }
{ print $1, $3 > out }
END {
  close(out)
  while ((getline line < out) > 0) print line
  close(out)
}

#+end_src
```

Downloads: AWK-21-001.awk · AWK-21-001.txt

### Actual output

AWK · exit 0

```
Ada 12
Ben 7
Cara 15
Dan 10
```

AWK-21-002

# Append after closing the stream

Change: Make reopening behavior visible.

BEGIN deliberately resets this example file for repeatable runs. Each >> then reopens it in append mode. Using > after each close would truncate earlier output. Closing for every row is educational, not a performance recommendation.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-21-002
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-21-002.awk :mkdirp yes

BEGIN {
  out = "generated/AWK-21-002.txt"
  printf "%s", "" > out; close(out)
}
{ print $1 >> out; close(out) }
END {
  while ((getline line < out) > 0) print line
  close(out)
}

#+end_src
```

Downloads: AWK-21-002.awk · AWK-21-002.txt

### Actual output

AWK · exit 0

```
Ada
Ben
Cara
Dan
```

AWK-21-003

# Route two groups into separate files

Change: Partition output using a condition.

Two fixed paths avoid turning input values directly into filenames. Non-labA rows go to the second file. Both files are initialized, closed after writing, and read back in a fixed order.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

## Complete Org source block

```
#+name: AWK-21-003
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-21-003.awk :mkdirp yes

BEGIN {
  a = "generated/AWK-21-003-a.txt"
  b = "generated/AWK-21-003-b.txt"
  printf "%s", "" > a; close(a)
  printf "%s", "" > b; close(b)
}
{ out = ($2 == "labA" ? a : b); print $1 >> out }
END {
  close(a); close(b)
  print "labA:"
  while ((getline line < a) > 0) print line
  close(a); print "other:"
  while ((getline line < b) > 0) print line
  close(b)
}

#+end_src
```

Downloads: AWK-21-003.awk · AWK-21-003-a.txt · AWK-21-003-b.txt

## Actual output

AWK · exit 0

```
labA:
Ada
Cara
other:
Ben
Dan
```

AWK-21-004

# Quote text fields for CSV output

Change: Handle commas and quotes when emitting CSV.

Each field is enclosed in double quotes; internal double quotes are doubled. This writes valid CSV for the two fields in the fixture. The input is simple semicolon-delimited text, not a CSV parser.

### Input

#### data/csvtext.txt

```
Ada, Jr.;labA
Ben "B";labB
```

### Complete Org source block

```
#+name: AWK-21-004
#+begin_src awk :in-file data/csvtext.txt :tangle scripts/AWK-21-004.awk :mkdirp yes

function csv(text) {
  gsub(/"/, "\"\"", text)
  return "\"" text "\""
}
BEGIN { FS = ";"; OFS = "," }
{ print csv($1), csv($2) }

#+end_src
```

Downloads: AWK-21-004.awk

### Actual output

AWK · exit 0

```
"Ada, Jr.","labA"
"Ben ""B""","labB"
```

AWK-21-005

# Build fixed-width records

Change: Contrast string truncation with numeric minimum width.

For %s, the precision .6 limits the displayed string length; the width 6 pads shorter strings. %03d zero-pads integers to at least three places but does not truncate longer numbers.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-21-005
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-21-005.awk :mkdirp yes
```

```
{ printf "%-6.6s|%-5.5s|%03d\n", $1, $2, $3 }
```

```
#+end_src
```

Downloads: AWK-21-005.awk

### Actual output

AWK · exit 0

```
Ada   |labA |012
Ben   |labB |007
Cara  |labA |015
Dan   |labB |010
```

AWK-21-006

# Produce an Org table

Change: Create output that can be reused in a report.

The first two lines form an Org table header and separator. The fixture contains no vertical bars or newlines in values. In this tour, output remains verbatim; :results output raw would insert the emitted table as Org markup.

### Input

#### data/scores.txt

```
Ada labA 12
Ben labB 7
Cara labA 15
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-21-006
#+begin_src awk :in-file data/scores.txt :tangle scripts/AWK-21-006.awk :mkdirp yes

BEGIN {
  print "| Name | Score |"
  print "|------+-------|"
}
{ printf "| %s | %d |\n", $1, $3 }

#+end_src
```

Downloads: AWK-21-006.awk

### Actual output

AWK · exit 0

```
| Name | Score |
|------+-------|
| Ada | 12 |
| Ben | 7 |
| Cara | 15 |
| Dan | 10 |
```

AWK-21-007

# Count replacements while cleaning text

Change: Report how much the transformation changed.

gsub returns the number of matched runs, not the number of individual whitespace characters. Because the target is a copy, $0 is preserved. This collapses whitespace but does not trim its ends.

### Input

#### data/spaces.txt

```
  Ada   labA    12
  Ben     labB    7
```

### Complete Org source block

```
#+name: AWK-21-007
#+begin_src awk :in-file data/spaces.txt :tangle scripts/AWK-21-007.awk :mkdirp yes

{ line = $0; n = gsub(/[[:space:]]+/, " ", line)
  print "replacements", n, "text=[" line "]"
}

#+end_src
```

Downloads: AWK-21-007.awk

### Actual output

AWK · exit 0

```
replacements 2 text=[Ada labA 12]
replacements 2 text=[Ben labB 7]
```

AWK-21-008

# Read the same side file twice

Change: Reset a redirected input stream.

After the first pass, close releases the input stream. A later getline can then reopen the file at the beginning. Without close, the second pass would remain at EOF. The fixture exists; production code should also check for -1 read errors.

### Input

#### data/empty.txt

```
(empty file)
```

#### data/lookup.txt

```
labA Aachen
labB Juelich
```

### Complete Org source block

```
#+name: AWK-21-008
#+begin_src awk :in-file data/empty.txt :tangle scripts/AWK-21-008.awk :mkdirp yes

BEGIN {
  path = "data/lookup.txt"
  for (pass = 1; pass <= 2; pass++) {
    n = 0
    while ((getline line < path) > 0) n++
    close(path)
    print "pass", pass, "records", n
  }
}

#+end_src
```

Downloads: AWK-21-008.awk

### Actual output

AWK · exit 0

```
pass 1 records 2
pass 2 records 2
```

# Small end-to-end recipes

AWK-22-001

# Tokenize and count words in first-seen order

Change: Combine text normalization, splitting and arrays.

Punctuation becomes whitespace and case is folded before counting. First-seen order makes the report deterministic. This C-locale tokenization is a simple ASCII policy; it is not linguistic word segmentation.

### Input

#### data/sentence.txt

```
Data, data! Open data.
Open tools.
```

### Complete Org source block

```
#+name: AWK-22-001
#+begin_src awk :in-file data/sentence.txt :tangle scripts/AWK-22-001.awk :mkdirp yes

{ line = tolower($0); gsub(/[^[:alnum:]]+/, " ", line)
  n = split(line, word, " ")
  for (i = 1; i <= n; i++) {
    w = word[i]
    if (!(w in count)) order[++k] = w
    count[w]++
  }
}
END { for (i = 1; i <= k; i++) print order[i], count[order[i]] }

#+end_src
```

Downloads: AWK-22-001.awk

### Actual output

AWK · exit 0

```
data 3
open 2
tools 1
```

AWK-22-002

# Summarize a small request log

Change: Turn several record-level facts into one summary.

The fixture schema is path, numeric status, bytes. A status of 400 or above is counted as an error. Real logs may contain quoting, missing values or other fields and require a suitable parser.

### Input

#### data/requests.txt

```
/home 200 120
/data 200 450
/home 404 30
/api 500 10
```

### Complete Org source block

```
#+name: AWK-22-002
#+begin_src awk :in-file data/requests.txt :tangle scripts/AWK-22-002.awk :mkdirp yes

{ requests++; bytes += $3
  if ($2 >= 400) errors++
}
END { print "requests", requests + 0
      print "errors", errors + 0
      print "bytes", bytes + 0 }

#+end_src
```

Downloads: AWK-22-002.awk

### Actual output

AWK · exit 0

```
requests 4
errors 2
bytes 610
```

AWK-22-003

# Aggregate transferred bytes by path

Change: Add a grouping dimension to the log summary.

The aggregation key is the complete path string. Query strings would remain part of that key; the next example shows an explicit alternative.

### Input

#### data/requests.txt

```
/home 200 120
/data 200 450
/home 404 30
/api 500 10
```

### Complete Org source block

```
#+name: AWK-22-003
#+begin_src awk :in-file data/requests.txt :tangle scripts/AWK-22-003.awk :mkdirp yes

!($1 in seen) { seen[$1] = 1; order[++n] = $1 }
{ bytes[$1] += $3 }
END { for (i = 1; i <= n; i++) print order[i], bytes[order[i]] }

#+end_src
```

Downloads: AWK-22-003.awk

### Actual output

AWK · exit 0

```
/home 150
/data 450
/api 10
```

AWK-22-004

# Remove a query string before grouping

Change: Choose the grouping key deliberately.

[?] matches a literal question mark; .* consumes the suffix. This groups by the textual path before the query. It does not decode percent escapes or otherwise normalize URLs.

### Input

#### data/queries.txt

```
/data?x=1
/data?x=2
/home
```

### Complete Org source block

```
#+name: AWK-22-004
#+begin_src awk :in-file data/queries.txt :tangle scripts/AWK-22-004.awk :mkdirp yes

{ path = $1; sub(/[?].*$/, "", path)
  if (!(path in seen)) { seen[path] = 1; order[++n] = path }
  count[path]++
}
END { for (i = 1; i <= n; i++) print order[i], count[order[i]] }

#+end_src
```

Downloads: AWK-22-004.awk

### Actual output

AWK · exit 0

```
/data 2
/home 1
```

AWK-22-005

# Keep the last value for each key

Change: Separate value selection from output ordering.

Each new value replaces the previous one for the same key, while the output order follows each key's first appearance. Last means last in file order, not latest by timestamp.

### Input

#### data/updates.txt

```
A 2
B 4
A 5
```

### Complete Org source block

```
#+name: AWK-22-005
#+begin_src awk :in-file data/updates.txt :tangle scripts/AWK-22-005.awk :mkdirp yes

!($1 in seen) { seen[$1] = 1; order[++n] = $1 }
{ latest[$1] = $2 }
END { for (i = 1; i <= n; i++) print order[i], latest[order[i]] }

#+end_src
```

Downloads: AWK-22-005.awk

### Actual output

AWK · exit 0

```
A 5
B 4
```

AWK-22-006

# Compare two key-value snapshots

Change: Start a comparison with two explicit event types.

The first file is a lookup; the second drives output order. This reports additions and changes but deliberately omits deletions. Keys are unique in each snapshot.

### Input

#### data/before.txt

```
A 2
B 4
C 6
```

#### data/after.txt

```
A 3
C 6
D 8
```

### Complete Org source block

```
#+name: AWK-22-006
#+begin_src awk :cmd-line data/before.txt data/after.txt :tangle scripts/AWK-22-006.awk :mkdirp
yes
```

```
FILENAME == "data/before.txt" { old[$1] = $2; next }
!($1 in old) { print "added", $1, $2; next }
old[$1] != $2 { print "changed", $1, old[$1], "->", $2 }
```

```
#+end_src
```

### Actual output

AWK · exit 0

```
changed A 2 -> 3
added D 8
```

Downloads: AWK-22-006.awk

AWK-22-007

# Detect keys removed from a snapshot

Change: Complete the comparison with deletions.

Keep the old key order and collect membership from the new snapshot. END can then identify old keys that never appeared in the second file.

### Input

#### data/before.txt

```
A 2
B 4
C 6
```

#### data/after.txt

```
A 3
C 6
D 8
```

## Complete Org source block

```
#+name: AWK-22-007
#+begin_src awk :cmd-line data/before.txt data/after.txt :tangle scripts/AWK-22-007.awk :mkdirp
yes

FILENAME == "data/before.txt" {
  if (!($1 in old)) order[++n] = $1
  old[$1] = $2; next
}
{ present[$1] = 1 }
END { for (i = 1; i <= n; i++)
  if (!(order[i] in present)) print "removed", order[i]
}

#+end_src
```

Downloads: AWK-22-007.awk

## Actual output

AWK · exit 0

```
removed B
```

AWK-22-008

# Emit clean data and a quality summary

Change: Assemble validation, transformation and evidence of the result.

Two early rejection rules keep invalid rows out of the calculation. The output separates accepted records with a data prefix from a small audit summary. The score grammar here is deliberately limited to nonnegative integers.

### Input

#### data/quality.txt

```
Ada labA 12
Ben labB NA
Cara labA
Dan labB 10
```

### Complete Org source block

```
#+name: AWK-22-008
#+begin_src awk :in-file data/quality.txt :tangle scripts/AWK-22-008.awk :mkdirp yes

NF != 3 { shape++; next }
$3 !~ /^[0-9]+$/ { numeric++; next }
{ accepted++; total += $3; print "data", $1, $2, $3 }
END { print "accepted", accepted + 0, "total", total + 0
      print "bad_shape", shape + 0, "bad_number", numeric + 0 }

#+end_src
```

Downloads: AWK-22-008.awk

### Actual output

AWK · exit 0

```
data Ada labA 12
data Dan labB 10
accepted 2 total 22
bad_shape 1 bad_number 1
```

# Example and command index


- AWK-01-001 · Print every record
- AWK-01-002 · Print one field
- AWK-01-003 · Choose and reorder fields
- AWK-01-004 · Print the last field
- AWK-01-005 · Show record and field counts
- AWK-01-006 · Run once before input
- AWK-01-007 · Run once after input
- AWK-02-001 · Match a whole line
- AWK-02-002 · Use the default action
- AWK-02-003 · Compare a specific field
- AWK-02-004 · Use a numeric threshold
- AWK-02-005 · Combine conditions
- AWK-02-006 · Match a field with a regex
- AWK-02-007 · Invert a match
- AWK-02-008 · Skip the first record
- AWK-02-009 · Take an inclusive range
- AWK-03-001 · Observe default whitespace splitting
- AWK-03-002 · Set the input separator
- AWK-03-003 · Set the output separator
- AWK-03-004 · Change record endings
- AWK-03-005 · Read blank-line-separated records
- AWK-04-001 · Concatenate strings
- AWK-04-002 · Align columns
- AWK-04-003 · Control decimal places
- AWK-04-004 · Build a formatted string
- AWK-05-001 · Accumulate a sum
- AWK-05-002 · Calculate the mean
- AWK-05-003 · Count only selected records
- AWK-05-004 · Find a maximum, including negatives
- AWK-05-005 · Show a running total
- AWK-05-006 · Round versus truncate
- AWK-06-001 · Change case
- AWK-06-002 · Take a substring
- AWK-06-003 · Find a literal substring
- AWK-06-004 · Replace the first match
- AWK-06-005 · Replace every match
- AWK-06-006 · Trim whitespace
- AWK-06-007 · Extract a regex match
- AWK-07-001 · Count by group
- AWK-07-002 · Sum by group
- AWK-07-003 · Keep the first occurrence
- AWK-07-004 · Remember group order
- AWK-07-005 · Split a string into an array
- AWK-08-001 · Use if and else
- AWK-08-002 · Use a conditional expression
- AWK-08-003 · Loop over fields
- AWK-08-004 · Stop early
- AWK-08-005 · Define a reusable function
- AWK-09-001 · Validate before arithmetic
- AWK-09-002 · Handle an empty file
- AWK-09-003 · Rebuild a record after editing
- AWK-10-001 · Pass a command-line parameter
- AWK-10-002 · Compare NR and FNR
- AWK-10-003 · Join a small lookup table

- AWK-11-001 · Create a compact report
- AWK-11-002 · Draw text bars
- AWK-12-001 · Change one field
- AWK-12-002 · Append a computed field
- AWK-12-003 · Keep only the first two fields
- AWK-12-004 · Empty is not the same as remove
- AWK-12-005 · Remove a middle field
- AWK-12-006 · Modulo: choose alternating records
- AWK-12-007 · Choose a column with -v
- AWK-12-008 · Distinguish text from numeric comparison
- AWK-13-001 · Anchor the end of a field
- AWK-13-002 · Match alternatives
- AWK-13-003 · Use a character class
- AWK-13-004 · Pass a dynamic pattern
- AWK-13-005 · Match a literal dot
- AWK-13-006 · Reuse the complete match in replacement
- AWK-14-001 · Test membership without testing the value
- AWK-14-002 · Delete a single key
- AWK-14-003 · Count combinations of two keys
- AWK-14-004 · Find a maximum per group
- AWK-14-005 · Sort a small set numerically
- AWK-14-006 · Emit a histogram with fixed bins
- AWK-15-001 · Print a difference from the previous row
- AWK-15-002 · Calculate a three-record moving mean
- AWK-15-003 · Start and stop a section
- AWK-15-004 · Compare with a range pattern
- AWK-15-005 · Remember the latest section header
- AWK-15-006 · Merge consecutive duplicate lines
- AWK-16-001 · Skip one header per file
- AWK-16-002 · Keep only rows with a lookup match
- AWK-16-003 · Preserve rows without a lookup match
- AWK-16-004 · Read a side file with getline
- AWK-17-001 · Find a minimum and a maximum
- AWK-17-002 · Compute an online mean
- AWK-17-003 · Compute sample standard deviation
- AWK-17-004 · Normalize against a reference
- AWK-18-001 · Give a function a local scratch variable
- AWK-18-002 · Pass an array to a function
- AWK-18-003 · Reuse a named Noweb block
- AWK-18-004 · Reuse the function in a summary
- AWK-18-005 · Inspect the input without running it as code
- AWK-18-006 · Emit a reusable tab-separated result
- AWK-19-001 · Ignore comments and blank lines
- AWK-19-002 · Check the expected field count
- AWK-19-003 · Validate signed decimals and exponents
- AWK-19-004 · Treat missing values separately from zero
- AWK-19-005 · Compute a mean over observed values
- AWK-19-006 · Choose a column by its header
- AWK-19-007 · Detect duplicate column names
- AWK-19-008 · Normalize a simple decimal-comma field
- AWK-20-001 · Turn two measurement columns into rows
- AWK-20-002 · Unpivot any number of measurement columns
- AWK-20-003 · Pivot known measurement names into columns
- AWK-20-004 · Mark a missing pivot cell explicitly
- AWK-20-005 · Transpose a small rectangular table
- AWK-20-006 · Summarize consecutive groups
- AWK-20-007 · Split a list-valued field into rows
- AWK-20-008 · Number rows within each group
- AWK-21-001 · Write a new result file
- AWK-21-002 · Append after closing the stream
- AWK-21-003 · Route two groups into separate files
- AWK-21-004 · Quote text fields for CSV output
- AWK-21-005 · Build fixed-width records
- AWK-21-006 · Produce an Org table
- AWK-21-007 · Count replacements while cleaning text
- AWK-21-008 · Read the same side file twice
- AWK-22-001 · Tokenize and count words in first-seen order
- AWK-22-002 · Summarize a small request log

- AWK-22-003 · Aggregate transferred bytes by path
- AWK-22-004 · Remove a query string before grouping
- AWK-22-005 · Keep the last value for each key
- AWK-22-006 · Compare two key-value snapshots
- AWK-22-007 · Detect keys removed from a snapshot
- AWK-22-008 · Emit clean data and a quality summary

---

Content: CC0 1.0. Structure and publishing framework: CC BY 4.0 by Lukas C. Bossert. Repository source · Publishing framework source · License scope and third-party notices.