# SAGE

*Structured Automated Goal Evaluation*

## User Manual

Version 0.40 Rev B

July 2026

# 1. Introduction

SAGE (Structured Automated Goal Evaluation) is a backward-chaining expert system shell with MYCIN-style certainty factors. It provides a browser-based environment for authoring knowledge bases, running consultations, and analyzing the inference process.

SAGE runs as a single offline HTML file with no server requirements. It uses CodeMirror 6 for syntax-highlighted editing and produces GUI consultation forms automatically from knowledge base declarations.

**New in v0.40: Embedded JavaScript commands are available in begin and end blocks for trusted knowledge bases. SAGE provides a limited API for facts, goals, session data, file access, messages, status, and logging. Rev B improves file selection and preserves browser permission for file pickers. All v0.39 features, including conditional asks and signed certainty factors, are retained.**

# 2. Getting Started

Open SAGE-v0.40-Rev-B.html in any modern browser. The public release is a self-contained HTML file, so no installation, server, supporting file, or Internet connection is required. The application presents two views: the Knowledge Base Editor and the Consultation view.

## 2.1 Toolbar

The toolbar provides quick access to common operations. All toolbar functions are also available through the menu bar.

## 2.2 Menu Bar

The menu bar contains six menus: File, Edit, KB (Knowledge Base tools), View, Consult, and Help. Keyboard shortcuts are shown alongside menu entries.

| Shortcut | Action |
|---|---|
| Ctrl+O | Open KB file |
| Ctrl+S | Save KB file |
| Ctrl+P | Parse KB |
| Ctrl+L | Lint KB |
| Ctrl+R | Start Consultation |
| Ctrl+D | Debug Consultation |
| Ctrl+F | Find / Replace |
| Ctrl+G | Go to Line |

# 3. Knowledge Base Language

SAGE knowledge bases use a period-terminated statement language. Every statement ends with a period (.). SAGE supports three comment styles: line comments beginning with # or //, which extend to the end of the line, and C-style block comments delimited by / *and* /, which may span multiple lines. Block comments are useful for temporarily disabling sections of a knowledge base during development.

## 3.1 Case Sensitivity

**Since v0.29, all SAGE keywords must be lowercase.** The lexer rejects uppercase keywords as syntax errors — they are not silently accepted or normalized. This applies to language keywords (if, then, and, or, etc.), comparison operators (eq, ne, gt, etc.), meta-proposition keywords (is_known, is_unknown, etc.), and all built-in function names.

Attribute names and string values are case-insensitive during consultation matching. The Pretty-Print command (KB > Pretty-Print) can be used to convert older knowledge bases with uppercase keywords to the current format.

## 3.2 Statement Types

A knowledge base consists of the following statement types: rules, ask declarations, goal declarations, computed values, user-defined functions, multivalued declarations, banner declarations, and begin/end blocks.

## 3.3 Rules

Rules are the primary building blocks. A rule concludes a value for an attribute, optionally with a certainty factor and conditions:

```
goal_attr = value [cf N]
    [if condition
    [and condition]
    [or condition]].
```

The certainty factor N may range from −100 to +100. A positive CF expresses confidence that the attribute has the stated value; a negative CF expresses confidence that the attribute does *not* have the stated value. When cf is omitted, the rule defaults to cf 100.

Conditions use relational operators: eq (equal), ne (not equal), gt, lt, ge, le. These same operators also work inside expressions and user-defined functions (see section 3.6). Meta-proposition operators test attribute state (see section 3.14).

## 3.4 Ask Declarations

An ask declaration defines how the system prompts the user for an attribute value:

```
attr = ask "prompt text"
```

```
    type typespec
    [default value]
    [info "help text"]
    [if conditions].
```

**Type specifications:**

| Type Spec | Description |
|---|---|
| yes_no | Yes/No radio buttons (capitalized labels) |
| legalvals(a, b, c) | Radio buttons for the specified values |
| integer(low, high) | Integer input with range validation |
| real(low, high) | Decimal number input with range validation |
| number(low, high) | Synonym for real |
| date | Date input (YYYY-MM-DD format) |
| text(length) | Single-line text input; max length in characters |
| textarea(length, height) | Multi-line text input; max length in characters, height in rows |

The text(length) type renders a single-line text box. The length parameter sets the maximum number of characters the user may enter. The input field width is capped at 80 characters. The textarea(length, height) type renders a multi-line text area. The length parameter sets the maximum character count, and height sets the number of visible rows. Both types require a non-empty response and enforce the maximum length during validation.

**Examples:**

```
nomenclature = ask "What is the nomenclature?"
    type text(80)
    info "Spell out any acronyms.".

description = ask "Describe the project scope."
    type textarea(2000, 8)
    info "Provide a detailed description.".
```

**Conditional asks (v0.39):** An ask declaration may include an optional if clause that gates whether the question is presented. When the engine seeks an attribute with a conditional ask, it evaluates the condition first. If the condition is false, the ask is suppressed and the attribute remains unknown—no value is assigned, not even a default. If the condition is true (or if there is no condition), the question is presented normally. Conditions use the same syntax as rule conditions.

Example:

```
cough_productive = ask "Is the cough productive?"
    type yes_no
```

```
    if has_cough eq yes.
```

In this example, the user is only asked about productive cough when has_cough is yes. If has_cough is no or unknown, cough_productive remains unknown and any rules depending on it are not satisfied.

## 3.5 Goal Declarations

Goals tell SAGE which attributes to pursue during consultation:

```
goal attribute.
```

## 3.6 Computed Values

Rules can derive values from expressions using the compute keyword:

```
result = compute expression.

result = compute expression
    if condition.
```

Expressions support arithmetic (+, −, *, /, ^), comparison operators (eq, ne, gt, lt, ge, le), string functions, type conversion, math functions, and date functions (see section 4). Comparison operators return 1 (true) or 0 (false) as numeric values.

## 3.7 User-Defined Functions

Named, parameterized pure functions can be defined and called from any expression:

```
function name(param1, param2) = expression.
```

Example:

```
function bmi(w, h) = round(w / h^2 * 703, 1).
patient_bmi = compute bmi(weight, height).
```

**Forward references:** User-defined functions may call other user-defined functions that are declared later in the knowledge base. Function names are resolved at runtime, not at parse time, so declaration order does not matter.

## 3.8 Multivalued Attributes

By default, attributes hold a single value. The multivalued declaration allows an attribute to accumulate multiple values, ranked by certainty factor:

```
multivalued(attribute).
```

## 3.9 Meta-Propositions

Meta-proposition operators test properties of an attribute rather than its value. **Since v0.38, meta-propositions use single-token syntax** (underscore-joined) rather than the two-word syntax used in earlier versions:

| Syntax | Meaning |
|---|---|
| attr is_known | Attribute has a value with CF ≥ 20 |
| attr is_unknown | Inverse of is_known |
| attr is_sought | Forces evaluation of attr (always true) |
| attr is_unique | Exactly one value with CF ≥ 20 |
| attr is_definite | Value known with CF = 100 |

The is_sought meta-proposition is useful for ensuring an attribute is evaluated before testing other conditions, as a side effect.

**Migration from earlier versions:** Replace "is known" with "is_known", "is unknown" with "is_unknown", and so on. The Pretty-Print command can assist with this conversion.

## 3.10 Date Type and Arithmetic

SAGE supports date values natively. Date literals use the #YYYY-MM-DD# syntax. The special keyword today evaluates to the current date.

| Function | Description |
|---|---|
| date_diff(d1, d2) | Days between two dates (d1 − d2) |
| date_add(d, n) | Add n days to date d |
| date_year(d) | Extract year from date |
| date_month(d) | Extract month (1–12) from date |
| date_day(d) | Extract day of month from date |
| date_format(d, fmt) | Format date (YYYY, MM, DD, Mon, MMMM, DDD, D) |

Example:

```
deadline_date = ask "Delivery deadline?" type date.
days_left = compute date_diff(deadline_date, today).
```

## 3.11 Certainty Factors

**Since v0.38, SAGE uses signed certainty factors** ranging from −100 to +100. A CF of +100 represents absolute certainty that the attribute has the concluded value. A CF of −100 represents absolute certainty that the attribute does *not* have the concluded value. A CF of 0 represents complete uncertainty. A CF of +20 or above is considered “known.”

When multiple rules conclude the same value for an attribute, their certainty factors are combined using the MYCIN signed pooling formula:

When both CFs are non-negative:

$$CF_{12} = CF_1 + CF_2 \times (1 - CF_1/100)$$

When both CFs are non-positive:

$$CF_{12} = CF_1 + CF_2 \times (1 + CF_1/100)$$

When CFs have opposite signs:

$$CF_{12} = (CF_1 + CF_2) \;/\; (1 - \min(|CF_1|, |CF_2|)/100)$$

This formula allows additional evidence to either increase or decrease certainty. For example, one rule might conclude a value with cf 80 while another concludes the same value with cf −30, reflecting contradictory evidence. The pooled result correctly reduces the overall certainty.

**Rule certainty and condition certainty:** The certainty of a rule's conclusion is limited by the certainty of the conditions that allowed the rule to fire. Specifically, the result CF is the rule's declared CF scaled by the minimum condition CF: result = round(ruleCF × conditionCF / 100). A rule with cf 80 whose conditions are satisfied at CF 90 produces a result CF of 72.

**CF bar display:** When displaying results and ranked multivalued conclusions, SAGE shows a visual confidence bar alongside each CF value. The bar uses filled blocks (▓) to represent magnitude and a sign indicator (+/−/0) to show direction:

```
+▓▓▓▓▓▓▓▒  CF 76     (strongly positive)
-▓▓▓▓▒▒▒▒  CF -45    (moderately negative)
0▒▒▒▒▒▒▒▒  CF 0      (completely uncertain)
```

## 3.12 Associative Arrays

An associative array (sometimes called a dictionary or map) is a collection of named entries. Unlike a simple attribute, which holds a single value, an associative array can hold many values, each identified by a unique key.

**Creating entries:** Write rules whose goal attribute uses bracket notation with a string key:

```
risk["technical"] = high cf 90
    if has_legacy_code eq yes.
```

**Reading from an associative array:** Use bracket notation with a string key in an expression:

```
risk_score = compute score(risk["technical"], 3).
```

Array keys are string expressions that are evaluated at runtime.

## 3.13 Banner Declarations

A banner declaration associates informational text with an attribute. When SAGE needs the attribute's value during backward chaining, it displays the banner text to the user and records the text as the attribute's value with CF 100. For multi-line text, use triple-quoted strings ("""). Line breaks within triple-quoted strings are preserved. The banner() function can also be called directly in expressions, including inside begin blocks (see section 3.14).

## 3.14 Begin and End Blocks

Inspired by AWK's BEGIN and END blocks, SAGE provides begin and end blocks for setup and cleanup that run before and after the main consultation:

```
begin
    banner("Welcome to the consultation.").

end
    banner("Consultation complete.").
```

Commands within a begin or end block are period-terminated expressions, optionally enclosed in curly braces { }. The begin block executes after the knowledge base is loaded but before goals are pursued. The end block executes after all goals have been resolved. During undo/replay, banners in begin blocks are skipped to prevent redisplay.
Embedded JavaScript (v0.40): A begin or end block may contain a javascript command with exactly one triple-quoted argument. JavaScript is not accepted elsewhere in a knowledge base. Run only knowledge bases from sources you trust.

```
begin
   javascript("""
      sage.session.started = true;
      sage.log.info("Consultation started.");
   """).
```

The sage API provides facts (set, get, retract, and related queries), goals (add, list, evaluate), a mutable session object shared from begin through end, file open/save operations, messages and status, logging, and consultation controls. Embedded code may be asynchronous. The Help > Quick Reference dialog lists the available API members.

**Execution order:** begin → goals → end.

# 4. Built-In Functions Reference

SAGE provides 47 built-in functions organized into four categories. All function names are lowercase.

## 4.1 String Functions (20)

| Function | Description |
|---|---|
| concat(a, b, …) | Concatenate 2–10 strings |
| upper(s) | Convert to uppercase |
| lower(s) | Convert to lowercase |
| trim(s) | Remove leading/trailing whitespace |
| length(s) | Number of characters |
| substr(s, start [, len]) | Substring (0-based start) |
| contains(s, sub) | True if s contains sub |
| starts_with(s, prefix) | True if s starts with prefix |
| ends_with(s, suffix) | True if s ends with suffix |
| replace(s, old, new) | Replace all occurrences |
| capitalize(s) | Capitalize each word |
| space_to_underscore(s) | Replace spaces with underscores |
| underscore_to_space(s) | Replace underscores with spaces |
| index_of(s, sub) | Position of sub in s, or −1 if not found |
| char_at(s, index) | Character at position (0-based) |
| pad_left(s, length, char) | Pad s on left to reach length with char |
| pad_right(s, length, char) | Pad s on right to reach length with char |
| repeat(s, count) | Repeat s count times |
| to_number(s) | Convert string to number (0 if not numeric) |
| to_string(n) | Convert number to string |

## 4.2 Math Functions (19)

| Function | Description |
|---|---|
| abs(n) | Absolute value |
| round(n [, decimals]) | Round (optional decimal places) |
| floor(n) | Round down |
| ceil(n) | Round up |

| | |
|---|---|
| min(a, b, …) | Minimum of 2–10 values |
| max(a, b, …) | Maximum of 2–10 values |
| mod(a, b) | Modulus (remainder) |
| pow(base, exp) | Exponentiation |
| sqrt(n) | Square root |
| sin(n), cos(n), tan(n) | Trigonometric (radians) |
| asin(n), acos(n), atan(n) | Inverse trigonometric |
| atan2(y, x) | Two-argument arctangent |
| rad(degrees) | Degrees to radians |
| deg(radians) | Radians to degrees |
| log(n) | Natural logarithm |
| log10(n), log2(n) | Base-10 and base-2 logarithm |
| exp(n) | e raised to the power n |
| pi() | The constant π |
| e() | The constant e |

## 4.3 Date Functions (7)

| Function | Description |
|---|---|
| date(s) | Convert ISO string to date value |
| date_diff(d1, d2) | Days between two dates (d1 − d2) |
| date_add(d, n) | Add n days to date |
| date_year(d) | Extract year |
| date_month(d) | Extract month (1–12) |
| date_day(d) | Extract day of month |
| date_format(d, fmt) | Format date (YYYY, MM, DD, Mon, MMMM, DDD, D) |

## 4.4 Display (1)

| Function | Description |
|---|---|
| banner(text) | Display informational text and return it |

# 5. KB Tools

## 5.1 Parse

KB > Parse (Ctrl+P) analyzes the knowledge base text using the character-by-character lexer and builds the internal rule structures. Parse reports the number of rules, ask declarations, goals, and any errors. Parsing is required before starting a consultation.

**The lexer:** The lexer walks the source text one character at a time, producing a stream of typed tokens. Syntax errors, including uppercase keywords, are reported with exact line and column numbers.

## 5.2 Pretty-Print

KB > Pretty-Print normalizes the knowledge base text. It lowercases all SAGE keywords, standardizes indentation, and preserves comments.

## 5.3 Lint

KB > Lint (Ctrl+L) performs static analysis on the parsed knowledge base and reports potential issues. The lint panel displays results in a table with severity, line number, and message. Clicking a row jumps to the corresponding line in the editor.

| # | Check | Severity |
|---|---|---|
| 1 | Uppercase SAGE keywords | Error |
| 2 | Undefined attributes | Warning |
| 3 | Dead asks | Hint |
| 4 | Unprovable goals | Warning |
| 5 | Unreachable rules | Hint |
| 6 | CF values out of range (−100–100) | Error |
| 7 | Duplicate rules | Warning |
| 8 | Circular dependencies | Error |
| 9 | Parse errors surfaced in lint | Error |
| 10 | Empty begin/end blocks | Warning |
| 11 | No GOAL declaration found | Error |
| 12 | Banner attribute conflicts | Warning |
| 13 | Embedded JavaScript trust warning | Warning |

# 6. Consultation

## 6.1 Starting a Consultation

Consult > Start Consultation (Ctrl+R) parses the knowledge base and begins backward chaining from the declared goals. SAGE automatically generates GUI forms for each question based on the ask type specifications.

**Execution order:** begin → goals → end.

### 6.2 Debug Consultation

Consult > Debug Consultation (Ctrl+D) runs the consultation in step-through mode. The debugger panel appears at the bottom of the screen and pauses at each decision point in the inference engine.

### 6.3 Explanation

During consultation, Consult > Why? explains why the current question is being asked. Consult > How? explains how a concluded value was derived.

### 6.4 Undo / Backtrack

The Back button undoes the last answer and replays all previous answers to return to the prior question.

### 6.5 Trace Log

View > Trace Log shows a complete log of the consultation. The trace can be saved via File > Save Trace as Text.

### 6.6 Results

View > Results displays all goal attributes and their concluded values after consultation. For single-valued goals, the value and its CF are shown. For multivalued goals, results are displayed as a ranked list with visual CF bars showing both direction and magnitude.

### 6.7 Auto-Ask Report

**New in v0.38.** View > Auto-Ask Report lists all attributes that required auto-generated questions during consultation because they lacked explicit ASK declarations. The report also suggests ASK declaration templates for each attribute, making it easy to add proper prompts to the knowledge base. After a consultation completes, if any attributes were auto-asked, SAGE logs a count in the system trace.

### 6.8 Session Persistence

Consultations are auto-saved to browser localStorage after each answer. Sessions can also be saved to .consult files and reloaded via File > Resume Session. User inputs can be saved as JSON via the Save button on the consultation form.

## 7. Architecture

**Engine layer (sage IIFE):** Contains the lexer, parser, AST builder, compiler, inference engine, expression evaluator, pretty-printer, linter, and debugger.

**UI layer:** Contains the CodeMirror 6 editor, view management, menu system, consultation forms, trace log, lint panel, and debug panel.

**Expression evaluator:** The expression evaluator supports both word-based comparison operators (eq, ne, gt, lt, ge, le) and symbolic operators (==, !=, >, <, >=, <=). Comparisons return 1 (true) or 0 (false) as numeric values. Numeric comparisons use numeric ordering; string comparisons use case-insensitive lexicographic ordering, consistent with rule conditions. Embedded JavaScript environment: Trusted begin/end commands run asynchronously through a restricted sage API. A per-consultation session object provides mutable state from begin through end.

**Signed CF pooling:** The combineEvidenceCf function implements the full MYCIN signed pooling formula for the −100 to +100 range, with three cases: both positive, both negative, and mixed sign. The combineCf function scales a rule's CF by its condition certainty. Both functions clamp results to −100..+100.

# 8. Version History

| Version | Key Features |
|---|---|
| **v0.40 Rev B** | Embedded JavaScript commands restored in begin/end blocks for trusted knowledge bases; restricted sage API for facts, goals, session data, files, UI, logging, and consultation control; Rev B improves file selection and browser file-picker reliability. |
| **v0.39** | Conditional ask declarations with if clause to gate questions on prior answers; lint CF range check corrected for signed certainty factors (−100..+100); inline JavaScript section removed from documentation |
| **v0.38** | Signed certainty factors (−100..+100) with full MYCIN signed pooling formula; meta-propositions changed to single-token syntax (is_known, is_unknown, is_sought, is_unique, is_definite); Auto-Ask Report; visual CF bar display; user inputs JSON export; removed data structures (list, stack, queue, AVL tree), document builder, paragraph numbering, xlsx export, higher-order functions, wildcards, record types, and short DS aliases to focus on inference |
| **v0.33** | Legalvals quote-stripping; optional numeric defaults; short aliases for DS functions (append, prepend, push, pop, enqueue, dequeue); raw XML injection for xlsx styling |
| **v0.32** | Higher-order list functions: list_map, list_reduce, list_filter, list_foreach; collection-to-list conversions; word-based comparison operators in expressions; total built-in count 102 |
| **v0.31** | 11 new built-in functions including xlsx_export; embedded SheetJS for offline .xlsx export |
| **v0.30b** | Multiline info text; banner Back/Next navigation; C-style block comments; begin/end with curly braces; text/textarea type specs; File System Access API file picker |
| v0.29 | Character-by-character lexer, strict lowercase keyword enforcement, legacy mode removal, forward references resolved at runtime |
| v0.28 | Banner declarations, begin/end blocks, 82 built-in commands |
| v0.27 | 40 new built-in commands: data structures, paragraph numbering, document builder |
| v0.26 | Step-through debugger, linter enhancements |
| v0.25 | Date type and date arithmetic |
| v0.24 | Linter with 6 static analysis checks |

| | |
|---|---|
| v0.23 | Pretty-printer |
| v0.22 | CodeMirror 6 bundle extracted to external file |
| v0.21 | AST-driven compile as sole runtime path |
| v0.20 | AST-based parser with parity testing |
| v0.19 | Go to Line, complete editor menu system |
| v0.18 | Inline javascript() declarations with triple-quoted strings |
| v0.15 | Case-sensitive keywords, meta-propositions, undo/backtrack, consultation persistence |
| v0.13 | Exponentiation, triple-quoted strings, user-defined functions |