CCalc User Manual

Welcome to CCalc

by Fortunato Caragliano
Version 53.15 (2026)

Contents

CCalc Manual table of contents

The Converter Guide

Functions and Math Domains

Math Domain Guides

Graphing Guide

Appendix C - The Ticker

Appendix D - Result Tags: NUM, APPROX, and LABEL

Appendix E - Converter Categories and Base Units

Appendix F - Grapher Function Support and Limitations

Appendix G — Function Reference and Formula Catalogue

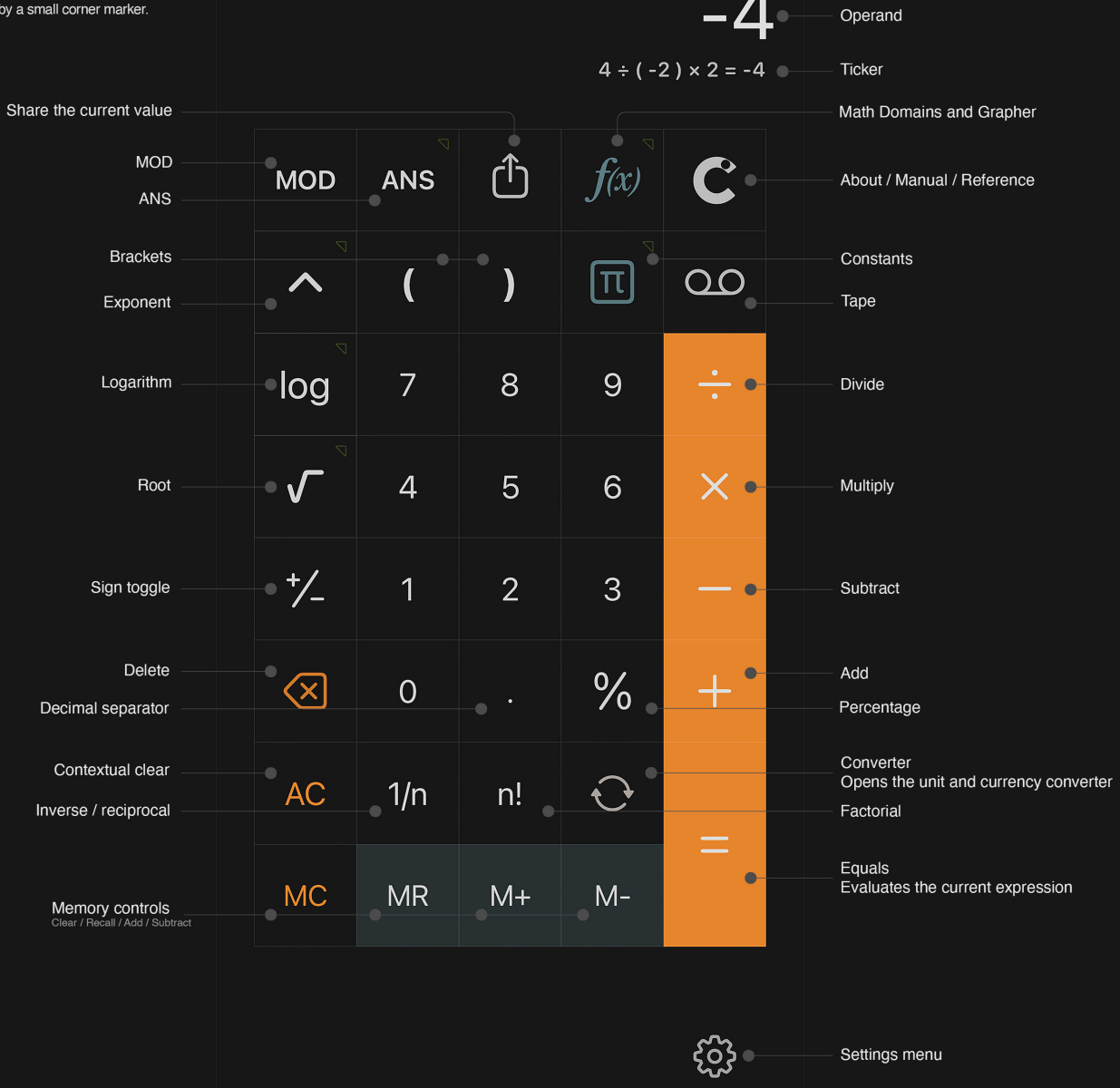
Appendix H - Technical Behavior Reference

Visual Keypad Reference

00:00 5G 100

This reference shows the main areas of the keypad.

Some keys support long-press actions or open a docklet, indicated by a small corner marker.



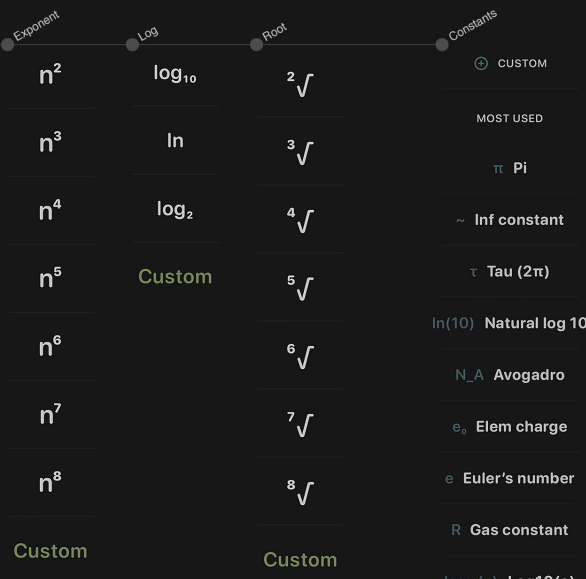
The exponent, logarithm, and root keys can open compact mini-docklets. These mini-docklets provide quick access to related powers, logarithms and roots.

Mini-docklet options

Exponent: n^2–n^8, or Custom for another exponent.
Logarithm: $\log_{10}$, ln, $\log_2$, or Custom for another base.
Root: $\sqrt[2]{}$–$\sqrt[8]{}$, or Custom for another root degree.

Constants docklet

The Constants docklet gives quick access to common mathematical, scientific and custom constants. The Most Used section keeps frequently selected constants near the top.



Docklets - Extended and Parked

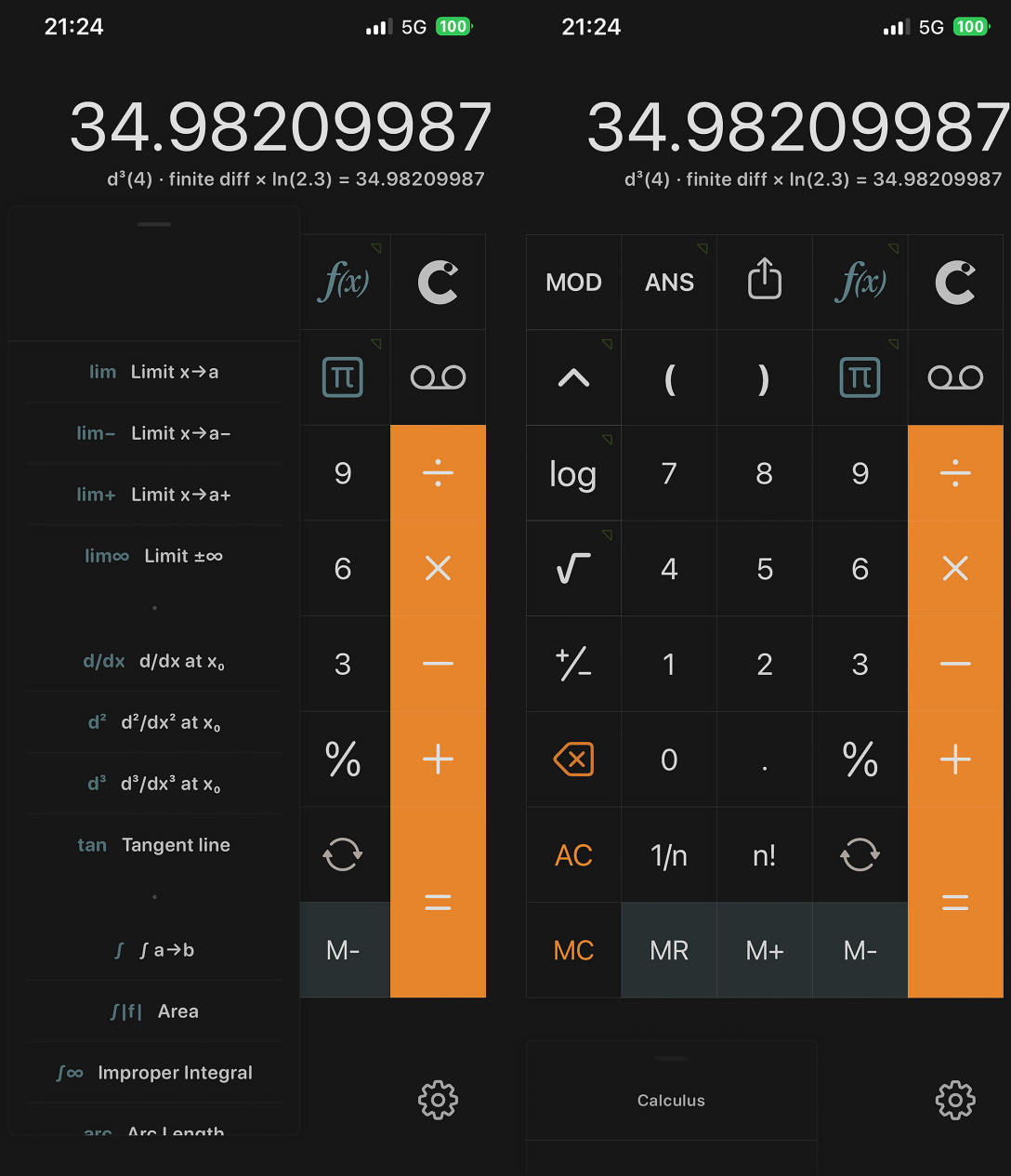
Docklets can be extended or parked.

When extended, a docklet shows the full function list for the selected math domain and partially covers the keypad while the user works inside that domain.

When parked, the docklet collapses near the bottom of the screen, keeping the selected math domain available while returning most of the keypad to normal use.

A parked docklet can be reopened when the user needs to work in the same domain.

Extended — Full function list.
Parked — Compact access state.
Docklet handle — Tap or drag to change state.
Selected domain — Shows the active docklet, here Calculus.



Graphing in CCalc

This reference shows several parts of the CCalc Grapher.

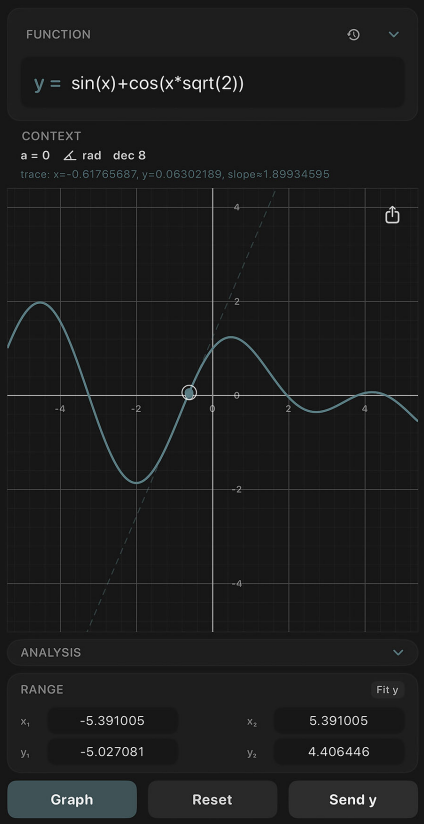
The Grapher is opened by choosing Graphing from the math domains area.

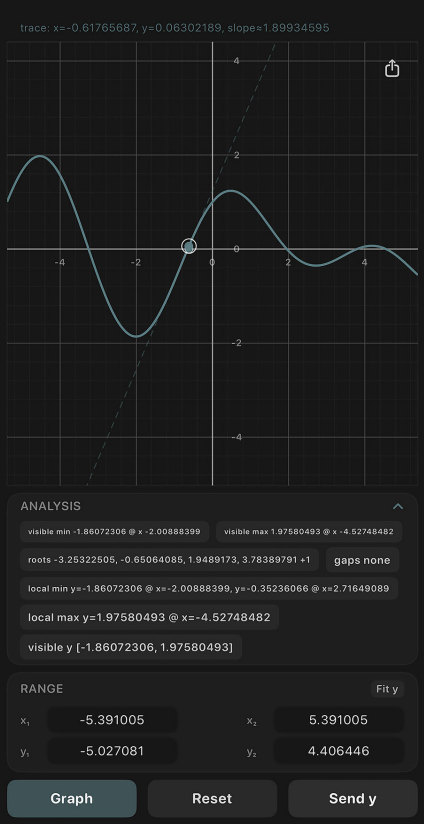
The example shown here graphs:

y = sin(x) + cos(x·sqrt(2))

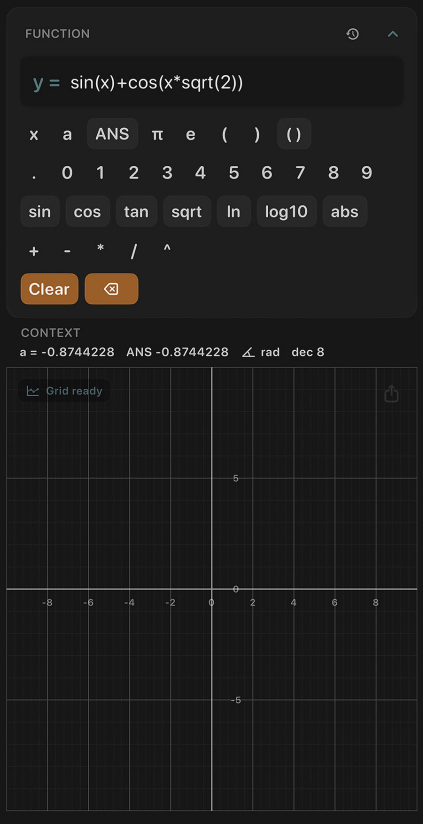
The graph view displays the selected point, the x and y coordinates, and the slope at that point.

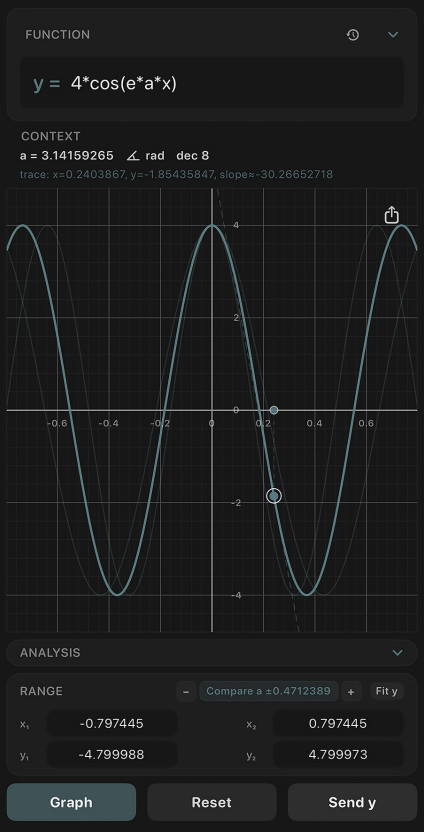
The same graph is also shown with the Analysis section open, where CCalc lists visible minima and maxima, roots, gaps, local extrema, and the visible y-range.

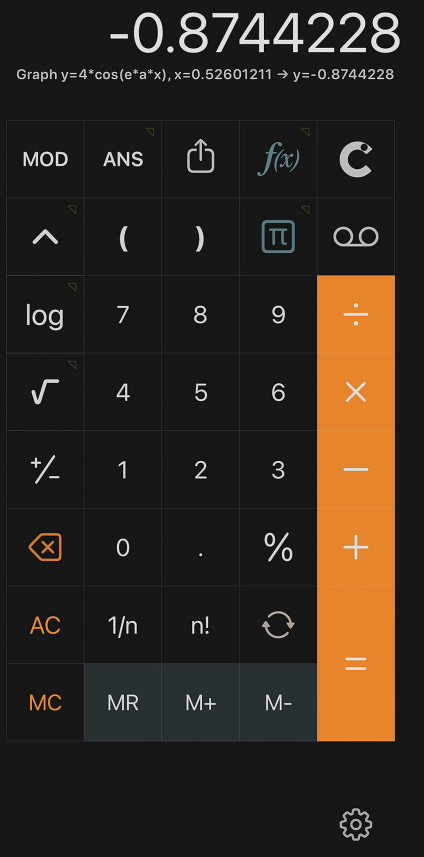




Additional Grapher tools are shown below: the function composition toolbox, a Compare a overlay example where a = π, and an example of Send y, which sends the selected y-value back to the main calculator.







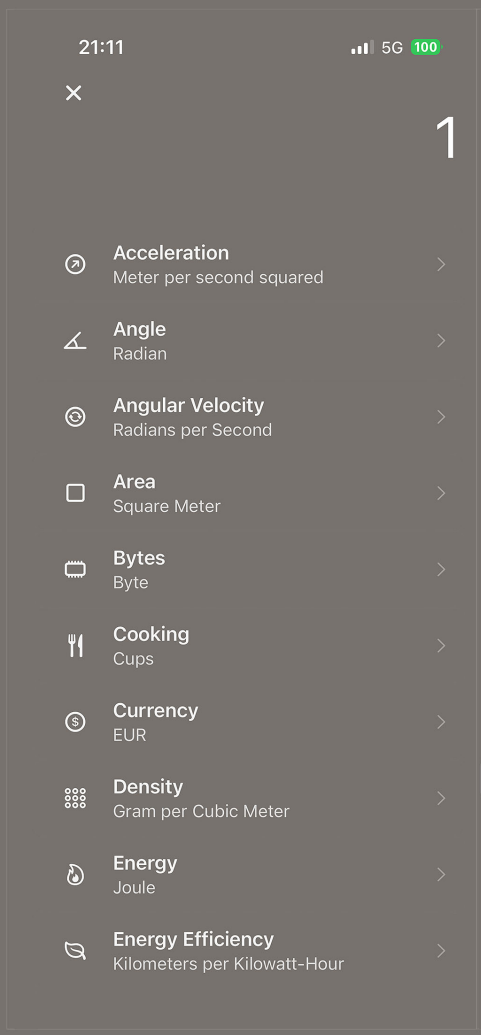
CCalc's Converter Flow

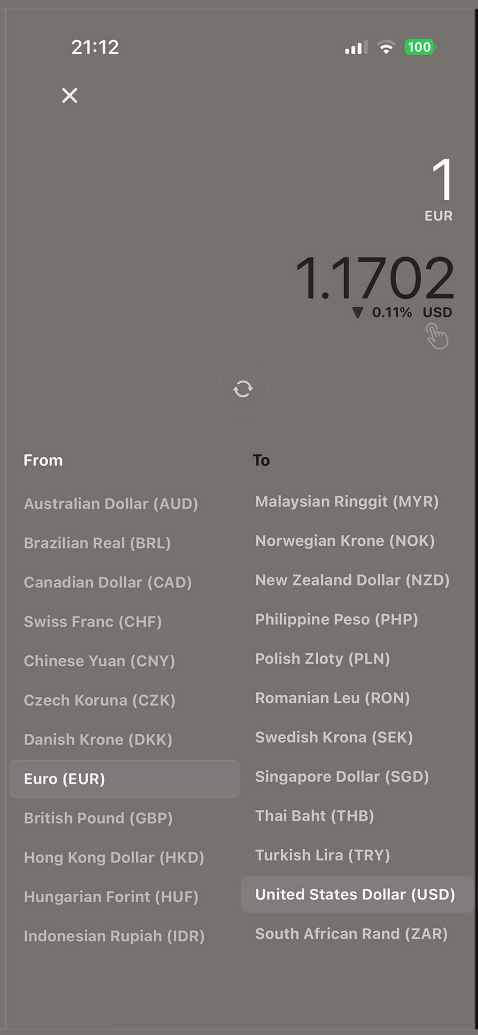
The Converter is accessed directly from the keypad.

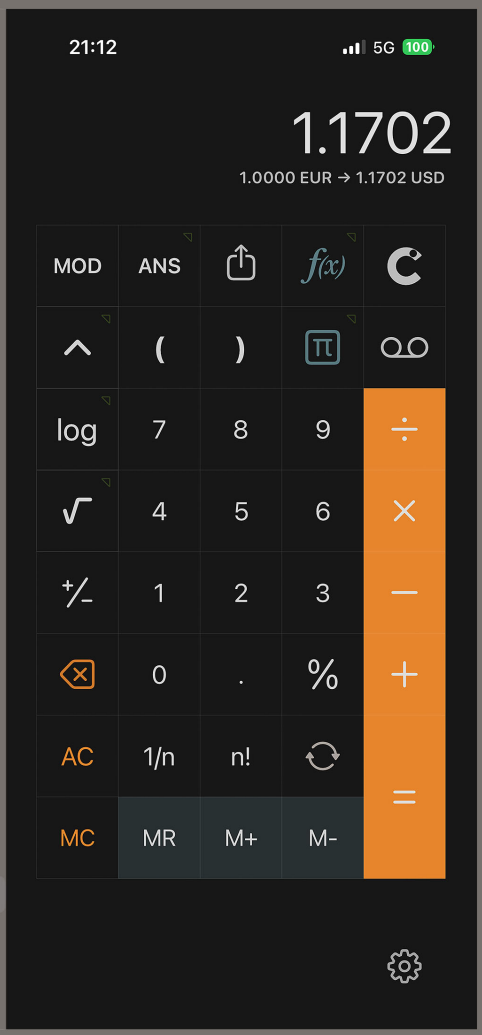
The current operand is first assigned to a conversion category. In this example, the selected category is Currency.

Inside the Unit Selector, the user chooses the original unit and the destination unit. CCalc performs the conversion and can send the converted value back to the main calculator display.

The returned value can then be used like any other calculator result.







The Tape System

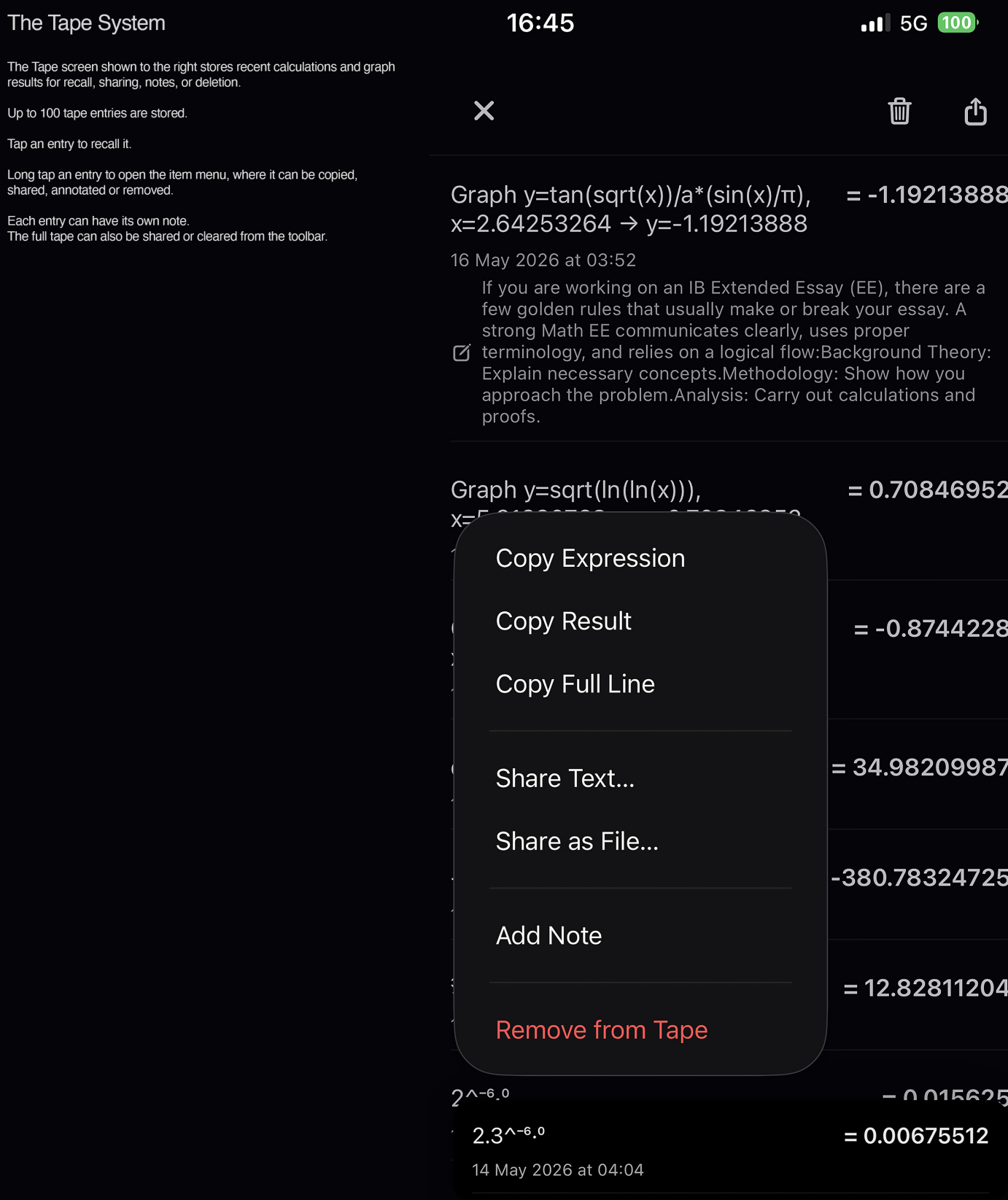
The Tape screen shown to the right stores recent calculations and graph results for recall, sharing, notes, or deletion.

Up to 100 tape entries are stored.

Tap an entry to recall it.

Long tap an entry to open the item menu, where it can be copied, shared, annotated or removed.

Each entry can have its own note.
The full tape can also be shared or cleared from the toolbar.

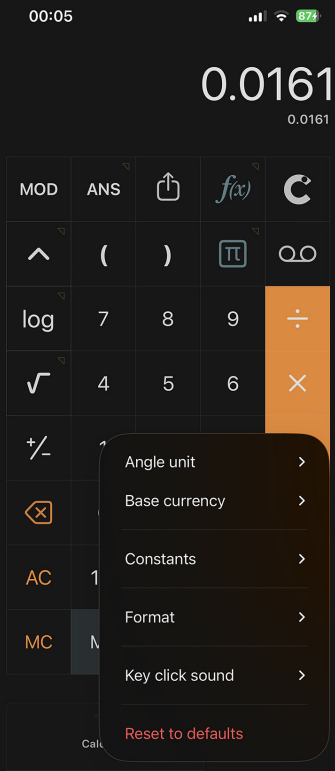


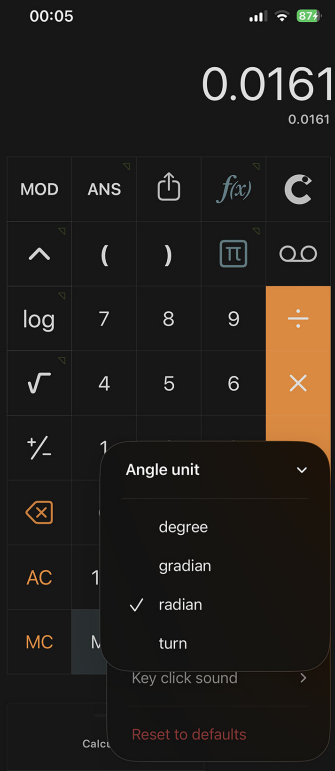
The Settings Menu

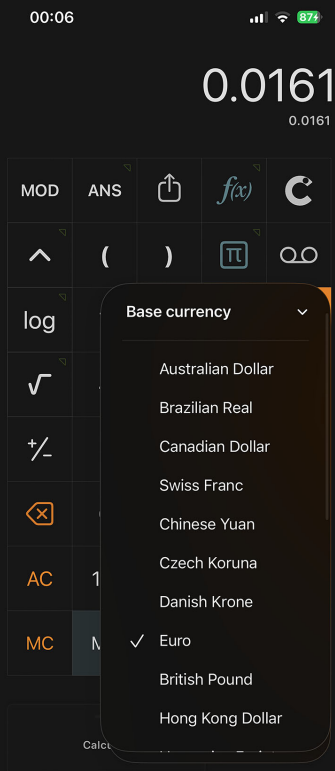
- Complete Settings Menu
- Angle Unit selection
- Base Currency selection
- Format options
- Constants Repository
- Key Click Sound selection

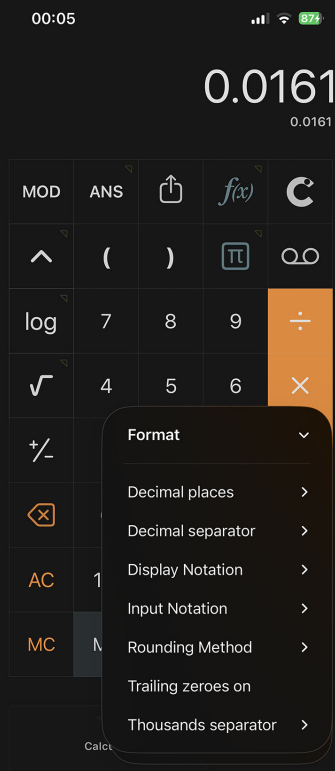
- The Settings Menu controls calculator behavior, formatting, constants, currency defaults, angle units, and sound feedback.

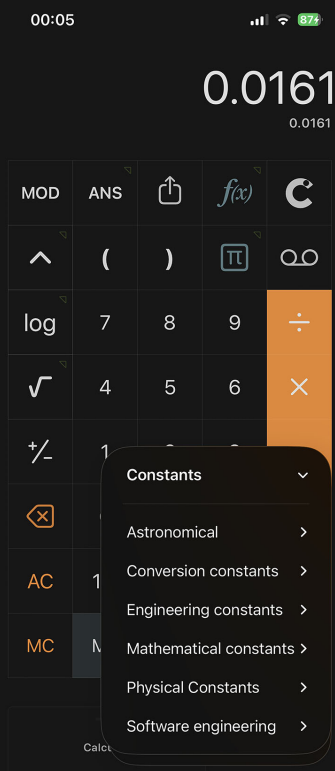
- Most settings apply immediately, so changes can be checked directly on the calculator display.

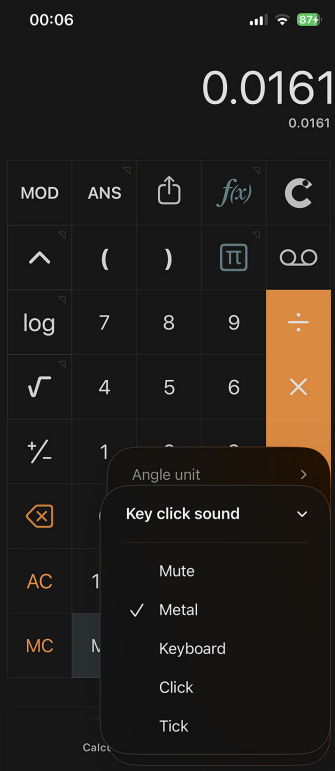












CCalc Overview

What CCalc Is

CCalc is a unified calculator environment built around a primary calculator interface and an integrated set of tools for unit conversion, function graphing, constants, and mathematical analysis.

While CCalc maintains the familiarity of a standard calculator, it extends the core experience with specialized docklets, a multi-category converter, a full-screen grapher, common and custom constants, and a comprehensive calculation tape.

CCalc is designed for inspectability: the ticker shows the active calculation flow, and results remain reusable across the app.

Who CCalc Is For

CCalc is designed for users who need more than basic arithmetic, including students, engineers, scientists, developers, finance users, and technically inclined users.

It is especially suited for people who work with formulas, unit systems, constants, graphs, repeated calculations, or technical reasoning, where a result is rarely an endpoint. A value may need to be reused, converted, stored, graphed, compared, labeled, inspected, or shared.

Core Design Principles

CCalc is built around five guiding principles:

Centralized Workspace

The main calculator remains the hub for input, output, and result flow.

Expandable Tools

Specialized docklets provide domain-specific power only when needed, without replacing the main calculator.

Current-Value Workflow

Tools such as the Converter and Grapher can work directly with the calculator’s active operand.

Operational Transparency

The ticker, result tags, display labels, warning messages, and tape help make calculation flow visible.

Reusable Results

Values are treated as reusable results that can move between modules and support continued calculation.

The CCalc System

The Calculator

The primary workspace for input and operations. It also serves as the return destination for results produced by other tools.

Docklets

Contextual panels focused on specific domains such as trigonometry, calculus, statistics, financials, equations, combinatorics, complex numbers, linear algebra, geometry, engineering, number theory, and constants. Docklets calculate results and can return them directly to the main calculator.

The Converter

A specialized tool for transforming the current calculator value between units of measurement or currencies.

The CCalc Grapher

A full-screen analytical environment for entering functions, tracing curves, inspecting slopes, reviewing visible graph features, and performing parameter sensitivity analysis.

Constants

A library of common and custom values available for insertion into calculations, docklets, and graphing workflows.

The Tape

A record of selected calculation history, allowing the user to review expressions, results, and significant calculation steps.

How to Use This Manual

This guide is structured in two layers:

Operational Chapters

A practical guide to using the app's features in a logical order, from basic arithmetic to advanced graphing.

Technical Appendices

A detailed reference documenting formulas, ticker logic, result tags, conversion categories, graphing behavior, precision rules, and mathematical edge cases.

Recommended reading path:

1. Starting with CCalc — essential calculator mechanics.
2. The Converter Guide — unit and currency transformations.
3. Functions and Math Domains — the docklet system.
4. Math Domain Guides — specific tools such as Calculus, Statistics, Engineering, and Linear Algebra.
5. The CCalc Grapher — graphing, tracing, analysis, and parameter comparison.
6. Settings — notation, angle units, currency behavior, formatting, and sound.
7. Appendices — technical rules, formulas, limitations, and precision behavior.

Starting with CCalc

Basic Operations

At its core, CCalc functions as a standard calculator. The user enters a value, selects an operator, enters a second value, and evaluates the expression.

The primary arithmetic and state operations include:

Arithmetic

Addition, subtraction, multiplication, division, percentages, powers, and roots.

State Controls

Sign toggling, deletion, clearing, and evaluation.

At this entry level, the main display shows the active value, while the ticker above it provides a compact record of the calculation, the active operation, or the most recent result.

In practical terms, the core workflow is:

1. Enter an initial number.
2. Select an operator.
3. Enter the next number.
4. Tap equals to evaluate.
5. Continue calculating directly from the displayed result if needed.

For example:

12 + 8 =

returns:

20

The result is immediately available as the operand for the next calculation.

Entering Numbers

Numbers are entered directly from the main keypad. CCalc supports integers, decimal values, and signed values.

For example:

125

12.5

-12.5

Decimal Separator

The decimal mark, period or comma, follows CCalc's selected format settings.

Sign Toggling

The sign-toggle key changes the sign of the active value directly. For example:

25

becomes:

-25

This modifies the current operand itself. It does not append a subtraction operator to the calculation.

The value currently being edited, displayed, or prepared for calculation is the current operand.

Editing Input

The display represents the value actively being typed. Digit entry adapts to the current state of the calculator.

Number Entry

Typing additional digits appends them to the right of the active integer or fractional portion.

For example:

123

can become:

1234

and:

12.3

can become:

12.34

Post-Evaluation Entry

If a calculation has just been completed, typing a new digit begins a fresh operand instead of extending the previous result.

External Returns

When CCalc receives a value from a docklet, the Converter, the Grapher, a constant, memory, or ANS, that value is treated as a completed operand. It can then participate in continued calculation without needing to be typed manually.

This distinction allows CCalc to preserve calculation flow while keeping the display familiar and predictable.

Deleting and Clearing Input

CCalc separates input correction from full reset.

Deleting Input

The delete key removes characters from the active, uncommitted input.

While typing a number, tapping delete removes the rightmost digit or decimal separator.

For example:

12345

after one delete becomes:

1234

and:

12.3

after one delete becomes:

12

If the display is showing a completed result or a value returned from another tool, delete may behave differently from active digit editing because that value is already committed to the calculator state.

Delete is used for correction.

Clearing Input

The clear key abandons the active calculation and resets the local calculation state.

Clearing may reset:

- the current displayed value;
- the active input entry;
- pending binary operations;
- temporary calculation state;
- temporary result flow;
- active display notation state where applicable.

Clear is used when the current calculation should be abandoned rather than corrected.

Key distinction: delete corrects the active input character by character; clear resets the calculation state so a new calculation can begin.

How the Display Works

The main display is the primary visual output of CCalc. It shows the active, usable value most likely to serve as the next operand.

The display may show:

- a number typed by the user;
- an evaluated calculation result;
- a value returned from a docklet;
- a unit or currency conversion result;
- a selected y value sent from the Grapher;
- a common or custom constant;
- a value recalled from memory;
- an ANS value;
- a formatted version of the current value.

The display is therefore not only a passive readout. It is the active value area of the calculator.

Some CCalc tools may also provide contextual labels alongside a value. These labels help identify the source, category, unit, or meaning of a result.

Understanding the Ticker

While the main display focuses on the active value, the ticker focuses on mathematical context.

The ticker is a compact information line situated below the operand display. It acts as an explanatory tool, rendering arithmetic expressions, pending operations, docklet result labels, converter descriptions, graphing coordinates, result tags, and system warnings.

For example, after evaluating:

12 + 8 =

the main display reads:

20

while the ticker may preserve the context:

12 + 8 = 20

When returning a value from a math domain docklet, the ticker may display the source operation:

sin(30 deg) [NUM]

When returning a coordinate from the Grapher, the ticker may record:

Graph y = sin(x), x = 1.5708 → y = 1

The ticker is a core component of CCalc's inspectability model. It helps explain how a result was produced, what operation is pending, or why a warning occurred.

Understanding Calculation Flow

Calculation flow defines how CCalc moves from input to operation to result.

The standard linear flow is:

First value → Operation → Second value → Result

For example:

7 × 6 =

returns:

42

In CCalc, calculation flow can also be non-linear. Operands do not need to be typed manually. They can originate from:

- direct keypad input;
- a previous calculation;
- ANS;
- memory;
- MOD;
- a docklet;
- the Converter;
- the Grapher;
- a common or custom constant.

This allows the user to calculate a value inside a specialized docklet, return it to the main display, chain an algebraic operation, pass the result to the Converter, and continue from the converted value.

The result is a calculator workflow where values can move between tools while remaining connected to the main calculation flow.

Operation Priority

When expression-style evaluation involves multiple operations, CCalc follows standard mathematical hierarchy.

The common priority order is:

1. Parentheses and grouping;
2. Exponents and roots;
3. Multiplication and division;
4. Addition and subtraction.

For example:

2 + 3 × 4

is interpreted as:

2 + (3 × 4)

and returns:

14

rather than:

20

To maintain transparency, the ticker should be read together with the display. In a multi-step or nested calculation, the display reflects the active value, while the ticker provides broader calculation context.

More detailed rules for priority, edge cases, and input modes are documented in the technical appendices.

Unary and Binary Operations

CCalc processes operations according to the number of values they require.

Unary Operations

A unary operation acts on one value. It can produce a result as soon as it has a valid input.

Examples include:

√x

sin(x)

1/x

n!

For example:

√16

returns:

4

because the operation already has the value it needs.

Binary Operations

A binary operation requires two values.

Examples include:

8 + 4

12 ÷ 3

5 × 6

2 ^ 8

Selecting a binary operator creates a pending operation until the second operand is entered and the expression is evaluated.

Docklets often use unary structures to transform the active display value directly, although more complex docklets may request additional parameters before calculating.

Immediate Results and Pending Operations

CCalc distinguishes between operations that have enough information to execute immediately and operations that are waiting for more input.

Immediate Results

An immediate result occurs when the selected operation already has the information it needs.

For example:

√16

returns:

4

because square root acts on a single value.

Immediate-result behavior may occur with:

- roots;
- reciprocals;
- sign toggling;
- factorials;

- many docklet functions;
- some constant insertions;
- converter returns;
- graph y value returns.

Pending Operations

A pending operation occurs when CCalc has registered an operator but still needs another value.

For example:

12 +

creates a pending addition. The calculation is not complete until the second operand is entered.

After entering:

8

and tapping equals, CCalc can evaluate:

12 + 8 = 20

The ticker serves as the visual indicator of this state. It can show the pending algebraic chain before the final result is committed.

Error Messages and Warning Feedback

If an operation violates mathematical rules, input requirements, or numerical limits, CCalc halts the calculation flow and alerts the user through visual messages on the display, detailed context in the ticker, and an optional audio cue.

Warnings and errors may occur under conditions such as:

- division by zero;
- invalid factorial inputs, such as negative values or unsupported values;
- input values outside a function's mathematical domain;
- values exceeding CCalc's supported numerical range or precision behavior;
- invalid graphing syntax;
- unsupported unit conversions;
- missing required parameters.

For example:

5 ÷ 0

cannot produce a valid finite result.

A factorial may also fail if the input is not supported by the factorial operation.

When a warning occurs, CCalc may show a message instead of a numeric result. The ticker may also identify the operation or context that triggered the warning.

Warning feedback is not only a failure state. It is a diagnostic guide. When an error occurs:

1. Read the main display for the current warning or result state.
2. Read the ticker for the calculation context.
3. Correct the input, choose a valid operation, or clear the active calculation.
4. Continue from a valid state.

Core Calculator Tools

This chapter explains the core tools that support everyday calculation in CCalc: powers, roots, logarithms, ANS, MOD, the Memory System, and the Tape.

These tools are not separate calculator modes. They are part of the main calculator workflow and are designed to help the user transform values, reuse results, perform remainder-based calculations, store important values, and review calculation history.

Powers, Roots, and Logarithms

Overview

CCalc includes direct access to powers, roots, and logarithms from the main calculator keypad.

These tools are part of the core calculator workflow. They allow the user to transform the current operand, select a preset parameter, or create a pending operation depending on the selected function.

They are not full math-domain docklets. They behave as compact calculation tools connected directly to the keypad, the display, and the ticker.

Powers

The power key is used to raise a value to an exponent.

For example:

2 ^ 8 = 256

In this case, 2 is the base and 8 is the exponent.

The power operation is a binary-style operation because it requires two values: the base and the exponent.

The base is the current value, and the exponent is the value entered or selected for the operation.

The power mini-docklet provides preset exponent choices from 2 to 8. This allows the user to quickly raise the current value to a selected power without manually typing the exponent each time.

Square roots and custom roots

The root key calculates roots of the current value.

The most common root operation is the square root.

For example:

√16 = 4

The square root answers the question: which value multiplied by itself produces the input value?

The root mini-docklet provides preset root-index choices from 2 to 8. The standard square root uses index 2, while higher indexes allow cube roots, fourth roots, fifth roots, and so on up to index 8.

For example:

$\sqrt[3]{27} = 3$

This is because:

3 × 3 × 3 = 27

Preset root indexes allow root operations to remain accessible from the calculator while still supporting more specialized input.

Logarithms

The logarithm key provides access to logarithmic calculations.

A logarithm answers the question: what exponent is needed to produce a given value from a given base?

For example:

log10(1000) = 3

because:

10 ^ 3 = 1000

Depending on the selected option, CCalc may support common logarithms, natural logarithms, or custom-base logarithmic behavior.

Common logarithms use base 10.

Natural logarithms use base e.

Custom-base logarithms allow the user to specify the base when the standard bases are not the desired operation.

Mini-docklet behavior

Powers, roots, and logarithms may open compact mini-docklets rather than full math-domain docklets.

These mini-docklets are designed to stay close to the main keypad and preserve the normal calculator workflow.

They provide additional options without moving the user into a separate full-screen tool.

This keeps common operations fast while still allowing the user to choose a preset exponent, a preset root index, or a logarithm base when needed.

Custom parameter entry

Some power, root, and logarithm operations require an additional parameter.

Examples include:

- a preset exponent;
- a preset root index;
- a custom logarithm base.

For powers and roots, CCalc provides preset mini-docklet choices from 2 to 8.

For logarithms, the additional parameter may be the logarithm base.

When an additional parameter is required, CCalc uses the mini-docklet interface to collect that value before completing the calculation.

This allows the user to keep working from the active display value while providing only the additional value needed by the operation.

Results and ticker behavior

When a power, root, or logarithm operation is completed, the result becomes available as the current calculator value.

The ticker records the operation context so the user can inspect how the result was produced.

For example:

2 ^ 8 = 256

or:

√16 = 4

or:

log10(1000) = 3

This keeps these operations consistent with CCalc's broader calculation flow: the result is shown in the display, while the ticker explains the calculation context.

ANS

Overview

ANS represents the most recent completed calculator result.

It allows the user to reuse the last result without manually retyping it. This is useful when continuing a chain of calculations, building on a previous answer, or using the result of one operation as the starting point for another.

In CCalc, ANS behaves as a stored result value. After a calculation is completed, the final result can be recalled and used again in a new expression.

The ANS key also includes a secondary long-press behavior. A normal tap recalls the most recent completed result. A long press restores the last operand used in the previous completed calculation.

This gives the ANS key two related recall functions:

- tap ANS to reuse the last result;
- long-press ANS to restore the last operand.

What ANS is for

ANS is useful when the user wants to continue calculating from the previous answer.

For example:

25 × 4 = 100

The result is now available as ANS.

The user can then continue:

ANS + 15 = 115

Instead of typing 100 again, the user can reuse the previous result directly.

The long-press behavior is useful when the user wants to recover the last input value rather than the final answer.

For example:

10 × 3 = 30

Tap ANS:

30

Long-press ANS:

3

In this example, the completed result is 30, but the last operand used in the calculation is 3.

How ANS fits into the calculator flow

ANS is most useful after a completed calculation.

Typical uses include:

12 × 8 = 96

ANS ÷ 3 = 32

ANS + 10 = 42

Each completed result can become the next reusable answer.

This makes ANS especially useful for step-by-step calculations where each result becomes the basis for the next operation.

The long-press recall extends this workflow by allowing the user to recover the last operand from the previous calculation. This can be useful when repeating a calculation pattern, correcting a step, or reusing the last entered value without typing it again.

For example:

48 ÷ 6 = 8

Tap ANS recalls:

8

Long-press ANS restores:

6

This separates the final answer from the last operand, giving the user a quick way to reuse either value.

ANS and precision

ANS stores the calculator's result value, not only the visible text on the display.

This means that the stored value may retain more internal precision than the rounded display suggests, depending on the calculator's current formatting and decimal-place settings.

The visible result is formatted for readability, while ANS is intended to preserve the calculation flow.

The long-press last-operand recall restores the stored operand value used by the previous completed calculation. Like ANS, this is intended to support calculation flow and reduce unnecessary retyping.

When to use ANS

Use ANS when:

- the previous result is needed again;
- a calculation continues across several steps;
- retyping the previous result would be inconvenient;
- the user wants to avoid transcription mistakes.

Use long-press ANS when:

- the last operand is needed again;
- the user wants to repeat a calculation pattern;
- the last entered value is more useful than the final result;
- the user wants to recover the previous input without retyping it.

ANS is especially useful in technical, financial, scientific, and repeated calculations where intermediate results are frequently reused.

The tap and long-press behaviors make the ANS key a compact recall tool: tap for the last result, long press for the last operand.

MOD

Overview

MOD is the modulo operation.

It returns the remainder after division.

In ordinary division, the calculator shows how many times one number fits into another. MOD instead answers a different question: what is left over?

For example:

17 MOD 5 = 2

This is because 5 fits into 17 three times:

5 × 3 = 15

and the remainder is:

17 − 15 = 2

So the modulo result is 2.

What MOD is for

MOD is useful whenever the remainder matters more than the division result.

It is commonly used in:

- divisibility checks;
- cyclic patterns;
- time calculations;
- programming logic;
- number theory;
- repeated grouping;
- checking even and odd numbers.

Basic examples

Exact division

20 MOD 5 = 0

The result is 0 because 20 divides evenly by 5.

There is no remainder.

Division with a remainder

22 MOD 5 = 2

The result is 2 because:

5 × 4 = 20

and:

22 − 20 = 2

Even and odd numbers

MOD can be used to check whether a number is even or odd.

14 MOD 2 = 0

Since the remainder is 0, the number is even.

15 MOD 2 = 1

Since the remainder is 1, the number is odd.

MOD in practical calculations

MOD is useful when values repeat in cycles.

For example, clocks repeat every 12 hours.

17 MOD 12 = 5

This means that 17 hours after a 12-hour cycle returns to the position 5.

This same idea applies to many repeating systems: days of the week, rotating sequences, indexes, periodic behavior, and modular arithmetic.

MOD and division

MOD should not be confused with ordinary division.

17 ÷ 5 = 3.4

but:

17 MOD 5 = 2

Division gives the quotient.

MOD gives the remainder.

Both are related, but they answer different questions.

The Memory System

Overview

The Memory System allows the user to store a value separately from the active calculation.

This is useful when a number must be kept available while the user continues working on other calculations.

The memory value can be recalled, changed, added to, cleared, or reused depending on the memory operation selected.

What memory is for

The Memory System is useful when the user needs to preserve an important value while continuing to calculate.

Examples include:

- storing a subtotal;
- keeping a constant value available;
- saving an intermediate result;
- accumulating values;
- comparing a stored value with a new result;
- reusing a number across several calculations.

The memory value acts like a small independent storage area inside the calculator.

Memory and the active display

The active display shows the current number or result being used by the calculator.

The memory value is separate from this display.

This means the user can store a value in memory, continue calculating normally, and later recall the stored value when needed.

For example:

250

Store in memory

The user can then perform another calculation:

18 × 7 = 126

The stored memory value remains available.

When recalled, the calculator can bring back the stored value:

Memory recall → 250

Common memory actions

The exact memory controls may vary by interface layout, but the basic memory actions are:

Store

Stores the current displayed value in memory.

Example:

Display: 125

Store

Memory value: 125

Recall

Brings the stored memory value back into the calculator.

Example:

Memory value: 125

Recall

Display: 125

Clear

Clears the memory value.

After clearing memory, there is no stored value available until a new one is stored.

Add to memory

Adds the current displayed value to the stored memory value.

Example:

Memory value: 100 Display: 25

Add to memory

Memory value: 125

Subtract from memory

Subtracts the current displayed value from the stored memory value.

Example:

Memory value: 100 Display: 25

Subtract from memory

Memory value: 75

Memory as a working tool

Memory is especially useful when the user needs to keep one value aside while working through several operations.

For example, the user may calculate a subtotal, store it, calculate another subtotal, and then combine the two.

First subtotal: 48.75

Store in memory

Second subtotal: 31.20

Recall memory

48.75 + 31.20 = 79.95

This avoids retyping the first subtotal and reduces the risk of entering the wrong value.

Memory and ANS

Memory and ANS are related, but they are not the same.

ANS automatically represents the most recent completed result.

Memory stores a value intentionally.

ANS changes after new completed calculations.

Memory remains stored until the user changes it or clears it.

Use ANS for continuing from the last result.

Use Memory for preserving a value that should remain available across multiple calculations.

Understanding the Tape

Overview

The Tape records calculation activity so the user can review what has been done.

It acts like a visible calculation history, similar to the paper tape on a traditional printing calculator, but adapted to CCalc's digital interface.

The Tape helps the user understand, verify, and revisit previous calculations.

What the Tape is for

The Tape is useful when the user wants to:

- review recent calculations;
- check how a result was produced;
- compare multiple results;
- avoid losing previous work;
- inspect longer calculation sequences;
- keep track of calculation context.

Instead of only showing the current display and ticker, the Tape provides a broader record of completed calculator activity.

What appears on the Tape

The Tape records completed calculation entries.

A typical Tape entry may show the expression and its result:

12 × 8 = 96

Another entry may show a longer expression:

25 + 14 × 3 = 67

The purpose is to make the calculation trace readable and reviewable.

Tape entries and result labels

Some CCalc tools return values with labels or tags.

These tags help explain what kind of result was produced.

Examples may include:

[NUM]
[APPROX]
[LABEL]

These tags are used to clarify whether the result is a numeric value, an approximation, or a label-style result.

For example, a function may produce an approximate result:

tan(2.3 rad) [APPROX]

A label-style result may describe a condition, warning, or non-standard output:

Division by 0 [LABEL]

The exact Tape behavior depends on the kind of operation performed and whether the action produces a completed calculation entry.

Opening the Tape menu

The Tape includes a long-press menu for working with the calculation history.

To open the Tape menu, long-press a Tape entry.

The menu provides actions related to that entry, allowing the user to interact with previous calculations without manually retyping them.

What the Tape long-press menu is for

The long-press menu is useful when the user wants to reuse, annotate, or manage a previous Tape entry.

Depending on the entry and the available interface options, the menu may allow actions such as:

- restoring a previous calculation or result;
- adding or editing a note;
- copying entry information;
- reviewing entry details;
- clearing or managing Tape content.

This makes the Tape more than a passive history list. It allows previous work to become part of the active calculator workflow again.

Restoring from the Tape

When a Tape entry is restored, CCalc can bring the selected value or calculation context back into the calculator.

This is useful when the user wants to continue from an earlier result, repeat a related calculation, or correct work without starting from the beginning.

For example, if the Tape contains:

48.75 + 31.20 = 79.95

the user can long-press the entry and restore it instead of retyping the value or reconstructing the calculation manually.

Notes on Tape entries

The long-press menu can also be used to add or edit notes when notes are available.

Notes help the user label why a calculation was made or what it represents.

For example, a Tape entry may represent a subtotal, a converted value, a comparison, or a step in a longer workflow.

A note can make that context clearer when reviewing the Tape later.

Tape versus ticker

The ticker and the Tape are related, but they serve different purposes.

The ticker shows the current calculation context.

The Tape stores completed calculation history.

The ticker is immediate and temporary.

The Tape is historical and reviewable.

For example, while entering a calculation, the ticker may show the expression currently being built. Once the calculation is completed, the Tape can preserve the finished entry.

Tape and verification

The Tape is useful for checking work.

For example, if the final display shows:

145.8

the Tape can help the user see how that result was obtained.

Instead of relying only on the final number, the user can review the calculation that produced it.

This is especially useful for:

- financial calculations;
- unit conversions;
- technical calculations;
- repeated operations;
- multi-step work;
- checking input mistakes.

Tape and longer workflows

The Tape becomes more useful as calculations become longer.

A user may perform several related calculations in sequence:

120 × 0.22 = 26.4
120 + 26.4 = 146.4
146.4 ÷ 3 = 48.8

The Tape keeps these steps visible as a calculation trail.

This makes it easier to understand the path from the original value to the final result.

Tape as part of the CCalc workflow

The Tape is not only a record of past calculations. It is part of the working environment.

Together with the display, ticker, ANS, Memory, docklets, converter, and graphing tools, the Tape helps CCalc remain transparent.

The user can see not only the current result, but also the calculation path that led to it.

This supports one of CCalc's main goals: making advanced calculation more readable, traceable, and easier to trust.

Chapter summary

The core calculator tools provide value transformation, continuity, reuse, storage, and review.

Powers, roots, and logarithms transform values directly from the main calculator workflow.

ANS recalls the most recent completed result.
MOD returns the remainder after division.

The Memory System stores a user-selected value independently from the active calculation.

The Tape records calculation activity so the user can review and verify previous work.

The Tape long-press menu allows previous entries to be reused, annotated, copied, reviewed, or managed when the corresponding actions are available.

Together, these tools make CCalc more than a simple input-and-result calculator. They help the user continue calculations, preserve important values, understand intermediate steps, and keep a readable record of the work performed.

The Converter Guide

Opening the Converter

The Converter is opened from the main calculator keyboard.

It is part of the main CCalc workflow and is designed to work with the value already shown in the calculator display.

The Converter does not behave like a separate standalone utility where the user first opens a blank conversion screen and then types a number. Instead, the user enters a value in the calculator first, then opens the Converter to decide what that value represents.

In practical terms, the workflow is:

1. Enter a number in the calculator.
2. Open the Converter.
3. Choose a conversion category.
4. Choose the source unit.
5. Choose the destination unit.
6. Read the converted result.
7. Tap the result if it should be returned to the main calculator.

This makes the Converter useful both as a quick reference tool and as a continuation tool for active calculations.

Where the Converter icon is found

The Converter is opened by tapping the Converter icon on the main calculator keyboard.

This icon is the entry point to CCalc's unit and currency conversion system.

After the icon is tapped, CCalc opens the Converter category view, where the user chooses what kind of quantity is being converted.

Choosing a conversion category

The first Converter screen is the category view.

This view lists the available conversion categories, such as:

- Acceleration
- Angle
- Angular Velocity
- Area
- Bytes
- Cooking
- Currency
- Density
- Energy
- Energy Efficiency
- Force
- Fuel Consumption
- Length
- Lighting
- Magnetism
- Mass
- Moment of Inertia
- Power
- Pressure
- Radiation
- Radioactivity
- Speed
- Temperature
- Thermal Performance
- Time
- Torque
- Volume
- Weight

Each row shows:

- the category name
- a reference or base unit underneath
- an icon for quick recognition

For the Currency category, the row uses the currently selected default base currency rather than a permanently fixed label.

After a category is selected, the Converter opens the unit selector for that category.

Selecting the source unit

The source unit tells CCalc how to interpret the number that was already present in the calculator.

For example:

- if the calculator shows 10
- and the user opens the Torque category
- and chooses Newton Meter as the source unit

then CCalc interprets the calculator value as:

10 N·m

The source unit is selected from the From list on the left side of the unit selector.

This list represents the unit assigned to the original calculator operand.

Selecting the destination unit

The destination unit tells CCalc what unit the source value should be converted into.

The destination unit is selected from the To list on the right side of the unit selector.

For example:

- source value: 10
- source unit: Newton Meter
- destination unit: Foot Pound Force

The Converter reads this as:

10 N·m → ft·lbf

and produces the converted result.

Reading the converted result

The converted result appears in the destination display.

The unit shown under the result identifies the selected destination unit.

The unit selector screen contains:

- the original calculator operand at the top
- the selected source unit under the top operand
- the converted result underneath
- the selected destination unit under the result
- a From unit list on the left
- a To unit list on the right
- a swap button between them

As soon as both unit selections are valid, the conversion is attempted automatically. There is no separate Convert button.

Returning the result to the calculator

When a converted result is shown, tapping the destination result display sends that value back into the main calculator as the new current operand.

This allows the user to continue calculating immediately with the converted quantity.

For example:

1. Enter 25
2. Convert 25 ft·lbf to N·m
3. Tap the converted result
4. Use the converted value in the next calculator operation

This makes the Converter part of the working calculation chain rather than only a passive lookup screen.

When the result is committed back into the calculator:

- the converted value becomes the active operand
- the ticker is updated

- the Converter can dismiss and return control to the calculator flow

Unit conversion behavior

Most Converter categories use an internal base-unit system.

In these categories, conversion works by:

1. converting the source value into the category's base unit
2. converting from the base unit into the destination unit

This keeps the conversion logic consistent and expandable.

Examples of base-unit logic:

- Length uses Meter as base
- Mass uses Kilogram as base
- Power uses Watt as base
- Pressure uses Pascal as base
- Volume uses Liter as base
- Torque uses Newton Meter as base

Some categories require special handling.

Temperature uses dedicated formulas because Celsius, Fahrenheit, Kelvin, Rankine, and Reaumur involve offsets as well as scaling.

Density includes a special non-linear path for API Gravity, which cannot be treated as a normal direct factor conversion. The Converter first transforms API Gravity into the base density representation and then continues from there.

Torque should be expressed with proper torque notation, not force-per-distance wording.

Preferred naming:

- Newton Meter
- abbreviation: N·m

Related units:

- Foot Pound Force → ft·lbf
- Inch Pound Force → in·lbf

This matters because torque is not the same thing as "Newton per Meter."

Currency conversion behavior

Currency is handled dynamically rather than by static conversion factors.

Currencies are not stored as fixed values inside the conversion-rule database. Instead, CCalc retrieves exchange-rate data from Frankfurter, a European exchange-rate service that provides current and historical foreign-exchange reference data.

The Currency category should be understood as a daily reference converter, not a brokerage feed. It is intended to provide dependable day-level conversion rather than real-time execution pricing.

Currency values are fetched from rate data rather than hardcoded into the rules database.

CCalc also ensures that the selected base currency is present with a rate of 1.0, so the rate map always has a stable reference anchor.

If the source and destination currencies are the same, the Converter returns the original amount unchanged and suppresses percentage change, since no real conversion has taken place.

Base currency

The Converter uses the app's persisted default base currency as the active reference base.

This means the Currency category adapts to the user's chosen base rather than always assuming USD visually.

For Currency, the highlighted base depends on the currently selected default base currency.

The selected base currency is used as the reference anchor for currency-rate handling, and CCalc ensures that this base is available with a value of 1.0.

Historical currency movement

When historical rate data is available, CCalc can compare the current conversion rate against the closest available historical snapshot and show the relative percentage movement.

This is displayed beneath the converted currency result as a directional indicator.

The purpose of this indicator is to give the user a quick sense of whether the current currency relationship has moved up or down relative to the historical comparison point.

If the source and destination currencies are the same, percentage movement is suppressed because no real conversion has taken place.

Converter display behavior

The Converter also updates the ticker so the user can see exactly what happened.

A good converter ticker should answer four questions clearly:

1. What was the original amount?
2. What was the source unit?
3. What is the converted amount?
4. What is the destination unit?

Example ticker style for a normal unit conversion:

10 N·m → 7.376 ft·lbf

Example ticker style for a currency conversion:

100.0000 USD ■ 92.5400 EUR

If historical comparison is enabled for currency, the ticker or surrounding display may also reflect relative movement.

Using abbreviations in the ticker is important because:

- it keeps the ticker compact
- it preserves dimensional clarity
- it avoids long repeated unit names
- it makes technical quantities easier to recognize at a glance

Examples:

- N·m instead of Newton Meter
- ft·lbf instead of Foot Pound Force
- km/h instead of Kilometers per Hour
- kWh instead of Kilowatt-hour

The ticker should always make dimensionality explicit. A raw numeric transformation without units is not sufficient.

Bad:

10 → 7.376

Good:

10 N·m → 7.376 ft·lbf

Converter precision and formatting

Currency conversions are intentionally displayed and handled with four decimal places only.

This is a fixed design rule and does not follow the calculator's general decimal setting.

For regular unit conversions, displayed precision may follow the calculator's general formatting behavior, depending on the category and result type.

The goal is to keep results readable while preserving enough detail for practical calculation.

Technical naming should remain conceptually correct.

For example:

- torque should remain Newton Meter, not Newton per Meter
- angular quantities should preserve distinctions such as rad, deg, turn, rad/s, rpm, and rps
- energy and torque should not be confused even when some SI magnitude relationships appear similar

Some units may be numerically related in SI form but conceptually different in use.

For example:

- Joule and Newton Meter can share magnitude relationships
- but the ticker should preserve the intended physical meaning and context

Converter tips and limitations

In the unit lists, the base unit for the selected category may be shown with a subtle outline. This helps the user see which unit anchors the internal conversion logic.

Examples:

- Torque base unit: Newton Meter
- Length base unit: Meter
- Pressure base unit: Pascal

Tapping the From or To header reverses the visible order of that unit list. This affects browsing only. It does not change the meaning of the selected units.

The central swap button exchanges the active source and destination selections.

This is useful when the user wants to invert a conversion quickly, such as changing:

- N·m → ft·lbf

into:

- ft·lbf → N·m

without manually reselecting both sides.

The Converter is not just a table of factors. It is a working extension of the calculator that:

- starts from the current operand
- assigns that operand a source unit
- converts it into a destination unit within the same category
- displays the result clearly
- updates the ticker with dimensional context
- allows the converted result to be reused immediately in further calculations

That design makes it faster, more integrated, and more useful than a standalone converter. It supports both everyday conversions and more technical workflows involving torque, radiation, angular quantities, temperature, and currency.

Functions and Math Domains

Overview

CCalc includes a set of specialized function panels called docklets. A docklet is a compact mathematical tool panel that opens on top of the calculator, gives access to a specific group of functions, and sends results back into the main calculator flow.

Docklets are used for domains such as Calculus, Combinatorics, Complex, Cryptography, Engineering, Equations, Financials, Geometry, Linear Algebra, Number Theory, Statistics and Trigonometry. Graphing is also listed in Math Domains, but it opens as a full-screen tool rather than as a standard docklet.

Most docklets behave like small sliding panels: they can open, park, expand, close, and return values. The Graphing tool and Common Constants panel are related to the same function system, but they are special cases and do not behave exactly like standard docklets.

What is a docklet?

A docklet is a focused function panel for one mathematical domain.

Instead of placing every advanced function directly on the main keypad, CCalc groups related tools into separate docklets. This keeps the main calculator clean while still allowing access to specialized calculations.

Examples of docklet domains include:

Calculus

Combinatorics

Complex

Cryptography

Engineering

Equations

Financials

Geometry

Linear Algebra

Number Theory

Statistics

Trigonometry

Each docklet contains functions that belong to its domain. For example, the Trigonometry docklet contains trigonometric tools, the Statistics docklet contains statistical tools, and the Engineering docklet contains applied engineering formulas.

A docklet is not a separate calculator. It works with the value currently shown in the calculator and returns results back to the main calculator when appropriate.

Docklet functionality and UI

A standard docklet appears as a panel over the main calculator interface.

The panel is designed to be temporary and focused. It opens when the user chooses a domain, provides access to that domain’s functions, and can then be parked, expanded, or closed.

A docklet usually includes:

- A compact header area
- A list or grid of available functions
- Function buttons or controls
- Input prompts when a function needs additional values
- A result return path back to the calculator
- A close or dismiss behavior
- A parked state for keeping the tool available without occupying the full working area

The exact controls depend on the domain. Some docklets are simple lists of functions. Others contain more complex prompts, multi-value entry, grouped sections, or domain-specific controls.

The common design goal is that a docklet should feel like part of the calculator rather than a separate app screen.

Memory accent for used or cancelled operations

Some docklets use a small colored accent to help the user remember the most recent interaction inside the panel.

This accent may appear after a function has been selected, used, cancelled, or dismissed. Its purpose is not to change the calculation itself, but to provide a visual cue about what happened inside the docklet.

For example, a recently used operation may remain visually marked so the user can recognize the last selected function. A cancelled or dismissed operation may also leave a temporary visual indication, helping the user understand that the operation was interrupted rather than completed.

This behavior is especially useful in docklets that contain many closely spaced functions or grouped tools. It gives the docklet a short visual memory of the previous action without requiring the user to read the ticker or reopen a prompt.

The accent is only a visual aid. It does not mean that the operation is still active, and it does not by itself change the current operand, ticker, or calculation flow.

The parked state

Most docklets support a parked state.

When a docklet is parked, it collapses into a smaller visible form. This keeps the docklet available while giving more space back to the calculator. The docklet remains present, but it no longer occupies its expanded working area nor has focus.

The parked state is useful when the user wants to keep a domain nearby while continuing to calculate.

For example, a user may open the Trigonometry docklet, use a function, park it, continue a calculation on the main keypad, and then expand the docklet again when another trigonometric function is needed.

In the parked state, the docklet acts like a compact handle or shortcut back to the active domain.

Minimizing and maximizing a docklet

A docklet can be minimized by tapping or dragging its header area, depending on the docklet and the current interaction state. When minimized, the docklet moves into its parked state.

To maximize a parked docklet, the user taps the visible parked docklet handle. CCalc expands the docklet back into its full working state, restoring access to that domain’s functions and controls.

The same result can also be reached through Math Domains: if the user chooses a domain that is already active but parked, CCalc expands that docklet again instead of opening a duplicate copy.

The expanded state

The expanded state is the full working state of a docklet.

When expanded, the docklet shows its available functions and controls. This is the state used for selecting operations, entering additional parameters, and reviewing domain-specific choices.

When the user selects a domain from Math Domains, CCalc opens that docklet in its expanded state. If the same docklet is already active but parked, selecting it again from Math Domains brings it back to the expanded state.

The expanded state is therefore the main interaction mode for standard docklets.

Closing a docklet

A docklet can be closed when it is no longer needed.

Closing a docklet removes it from the calculator interface and returns the user to the normal calculator view. CCalc also clears temporary docklet-related display states when necessary, such as labels or special operand overlays used by some advanced domains.

Closing is different from parking:

- Parking keeps the docklet available in a compact form.
- Closing removes the docklet from the interface.

The Math Domains menu also includes a Close row. This closes the Math Domains menu and closes active docklets through the main docklet-closing behavior.

Auto-dismiss behavior

Docklets are designed to stay out of the way when they are not being used.

Some docklets may automatically return to the parked state after a period of inactivity. This prevents an expanded panel from occupying the calculator interface longer than necessary.

Auto-dismiss does not mean that the calculation is lost. It only changes the docklet’s visual state. The calculator value, ticker, and returned results remain part of the calculator flow according to the normal result behavior.

The purpose of auto-dismiss is to keep the workspace clean while still allowing quick return to the active domain.

Sending results back to the calculator

One of the most important docklet behaviors is sending results back to the main calculator.

When a docklet function produces a numeric result, CCalc can place that result into the calculator's current operand. The result then becomes available for continued calculation.

For example, if a docklet computes a value such as:
sin(30 deg) = 0.5

the calculator can receive 0.5 as the active value. The user can then continue with normal operations such as multiplication, addition, division, or further docklet functions.

This is what makes docklets integrated tools rather than isolated reference panels.

Docklet results in ongoing calculations

Docklet results can participate in ongoing calculations.

A result returned from a docklet may become the next value in the calculator's expression flow. This allows advanced functions to be used naturally inside larger calculations.

For example, the user may calculate:
12 × sin(30 deg)

or use a docklet result first and then continue:
cos(60 deg) + 8

The calculator treats the returned value as a usable operand. The ticker may show the function label, the result, or both depending on the function type and the current calculation state.

The goal is that docklet output should not feel separate from ordinary calculator input.

Result labels and ticker behavior

Docklets use the ticker to explain where a result came from.

When a function returns a value, the ticker may show a label such as:
sin(30 deg) [NUM]

or:
sqrtmean(x) [APPROX]

The exact label depends on the function and the domain.

The ticker is especially important because docklet results may come from complex operations. Instead of only showing a number, CCalc tries to preserve the meaning of the result. This helps the user understand whether the displayed value came from a trigonometric function, a statistical operation, a matrix-related function, an engineering formula, or another specialized tool.

Some docklets also use special display labels for results that are not best represented as a single plain number. For example, a complex-number operation may need to show a formatted complex value, while still keeping a numeric component available internally when appropriate.

Result tags: NUM, APPROX, and LABEL

CCalc uses result tags to clarify the type of output returned by a docklet.

The common tags are:
[NUM]

[APPROX]

[LABEL]

NUM

[NUM] means the result is a direct numeric value.

This is used when the function returns a normal calculator-ready number. The result can usually be placed into the current operand and used immediately in further calculations.

Example:
sin(30 deg) [NUM]

APPROX

[APPROX] means the result is numeric but should be understood as approximate.

This may happen when a function uses numerical methods, rounded values, iterative solving, floating-point approximation, or a formula where the result is best treated as an estimate.

Example:

root ~ 1.4142 [APPROX]

The result can still be useful in calculations, but the tag warns that the value should not be interpreted as exact.

LABEL

[LABEL] means the docklet has produced a result label, message, structured representation, or non-flow value rather than a standard calculator-ready operand.

This tag is used when the output cannot safely behave like an ordinary scalar number in the main calculation flow. The result may still be meaningful and useful, but it is being shown primarily as a representation or explanation rather than as a value to continue calculating with.

This can happen for several reasons:

- The result is not a single scalar number.
- The result is a structured object, such as a matrix, vector, complex representation, or formatted multi-part output.
- The result is a descriptive message.
- The result is a warning or error condition.
- The result is useful to read, but not appropriate to insert into an ongoing arithmetic expression.

Example:

No real solution [LABEL]

Another example would be a matrix-style or multi-component result where the displayed output is meaningful, but cannot be treated as a single ordinary operand.

A [LABEL] result should therefore be understood as a displayed result or explanation, not necessarily as a value that can continue through the calculator's normal numeric flow.

Math Domains: the centralized docklet hub

Math Domains is the centralized hub for choosing docklets.

It opens as a compact menu anchored near the function control area of the calculator. Instead of opening every domain from a separate main-keypad button, CCalc collects the advanced math domains in this single menu.

The Math Domains menu lists the available domains alphabetically by title.

The current domain list includes:

Calculus

Combinatorics

Complex

Cryptography

Engineering

Equations

Financials

Geometry

Graphing

Linear Algebra

Number Theory

Statistics

Trigonometry

The visible menu is scrollable. It is designed to show several domains at once while keeping the panel compact. If more domains exist than can fit comfortably, the list scrolls vertically.

The menu also includes a Close row at the bottom. This provides a quick way to close the menu and dismiss active docklet UI through the shared closing behavior.

Choosing a math domain

To choose a math domain, the user opens Math Domains and taps the desired domain name.

When a standard docklet domain is selected, CCalc closes the Math Domains menu and opens the selected docklet.

If another docklet is already active, CCalc switches to the newly selected docklet. If the selected docklet is already active but parked, CCalc expands it again.

This means the Math Domains menu works both as:

a launcher for new docklets

and as:

a return path to an already active docklet

The user does not need to manually close one standard docklet before opening another. Selecting a different domain routes the calculator to that domain.

Returning from a docklet to the calculator

The user can return from a docklet to the main calculator by parking or closing the docklet.

When the docklet is parked, the calculator becomes more accessible while the docklet remains available. When the docklet is closed, the calculator returns to its normal state without the docklet panel.

Results already sent back to the calculator remain available according to the normal calculation flow. Closing the docklet does not undo the returned value.

Some docklets may clear temporary labels or overlays when closed, especially if they use specialized result displays. This keeps the main calculator display clean after leaving a domain.

Graphing as a special case

Graphing appears in the Math Domains list, but it is not treated as a standard compact docklet.

When the user selects Graphing, CCalc opens the Grapher as a full-screen tool instead of routing it through the compact docklet panel system.

This is intentional.

Graphing needs more space than a normal docklet because it includes:

- A graph canvas
- Function entry
- Token tools
- Range controls
- Zooming and recentering
- Trace behavior
- Analysis output
- Graph history
- Sharing controls

Because of this, Graphing does not use the ordinary parked and expanded docklet states. It is launched from Math Domains, but it behaves as a separate full-screen graphing environment.

The Grapher can still interact with the calculator. It can use calculator values such as the current operand, ANS, and the parameter a, and it can send traced y values back to CCalc. However, its visual structure is different from the standard docklet model.

In this chapter, Graphing should therefore be understood as:

a Math Domains tool

but not:

a standard parked/expanded docklet. For full details on the Grapher, including function entry, tracing, analysis, graph history, and sharing, see the Graphing Guide.

Common Constants as a special case

Common Constants is also related to CCalc's function system, but it is not part of the Math Domains docklet list.

Common Constants is a compact constant picker. It provides quick access to frequently used mathematical and scientific constants. Its purpose is different from a full domain docklet.

A standard docklet usually opens a panel of functions for a mathematical domain. Common Constants instead lets the user insert or select known values such as constants.

Because of this, Common Constants should be described separately from Math Domains.

It is not selected from the Math Domains list, and it is not routed through the main docklet domain menu. It has its own compact behavior and is anchored to its own calculator interaction point.

Common Constants is best understood as:

a specialized picker for reusable values

rather than:

a full mathematical domain docklet

Relationship between docklets, Graphing, and Common Constants

CCalc uses a layered approach to advanced functions.

Standard docklets handle compact domain-specific calculations. Math Domains is the launcher and router for those docklets. Graphing is launched from Math Domains but opens as a full-screen graphing tool. Common Constants is a separate compact picker for constants.

This distinction is important because the user may see Graphing listed alongside docklets, but its behavior is intentionally different. Likewise, Common Constants may feel similar to a function panel, but it is not part of the Math Domains router.

Summary

Functions in CCalc are organized through a docklet system.

A standard docklet is a compact, domain-specific panel that gives access to advanced mathematical tools without overcrowding the main calculator. Docklets can open, expand, park, close, and return results to the main calculation flow.

Math Domains is the central hub used to choose these docklets. It lists the available mathematical domains and routes the user to the selected tool.

Graphing is included in Math Domains, but it opens as a full-screen Grapher rather than a standard docklet. Common Constants is another special case: it is a compact constant picker, not a Math Domains docklet.

Together, these systems let CCalc remain a clean calculator while still providing access to broad mathematical, scientific, financial, engineering, and graphing tools.

Math Domain Guides

Overview

Math Domains collect CCalc's advanced mathematical tools into focused docklets.

Each domain is designed around a specific type of work. Some domains handle familiar calculator functions, such as trigonometry, statistics, geometry, or percentages. Others support more specialized workflows, such as complex numbers, matrices, equations, cryptography, engineering formulas, calculus, or number theory.

This chapter gives a practical overview of the available math domains. It explains what each domain is for, when the user may want to open it, and what kind of result the domain usually returns to the main calculator.

This chapter is not a complete function-by-function reference. Detailed formulas, input requirements, examples, edge cases, and implementation limits belong in the domain appendixes.

How to read a domain guide

Each domain guide describes the purpose of a docklet rather than every formula inside it.

A domain guide answers four questions:

- What is this domain for?
- When would the user open it?
- How does it interact with the current calculator value?
- What kind of result does it usually return?

Some domain functions return ordinary numeric values that can continue through the calculator flow. Others return approximate results, structured outputs, or labeled results that are meant to be read rather than used as a normal operand.

The result tags described in the Functions and Math Domains chapter still apply:

[NUM]

The result is a direct numeric value and can usually continue through the calculator's normal expression flow.

[APPROX]

The result is numeric but should be understood as approximate. This is common for numerical methods, iterative solvers, floating-point formulas, statistical approximations, engineering estimates, and trigonometric calculations.

[LABEL]

The result is a displayed label, structured representation, message, vector, matrix, formula, list, or other non-flow output rather than a standard calculator-ready operand.

Calculus Guide

Overview

The Calculus domain provides numerical tools for limits, derivatives, integrals, roots, extrema, series approximations, multivariable calculus, and basic differential-equation models.

This docklet is designed for practical calculator-based calculus rather than symbolic algebra. The user selects a calculus operation, chooses a function template, enters the required parameters, and CCalc returns either a numeric result or a labeled expression.

The Calculus docklet uses predefined function templates rather than free-form expression entry. Depending on the selected operation, the user may choose from templates such as polynomial, power, exponential, logarithmic, sine, cosine, damped sine, rational, or constant functions. Multivariable tools use templates such as constant, plane, and quadratic 2D functions.

Function groups

The Calculus docklet includes tools for:

- limits;
- first, second, and third derivatives;
- tangent lines;
- definite integrals;
- unsigned area;
- improper integrals;
- arc length;
- volume of revolution;
- convolution;
- Fourier-style values;
- roots;

- minima and maxima;
- Taylor approximations;
- linearization;
- partial derivatives;
- gradients;
- directional derivatives;
- critical-point analysis;
- Hessians;
- double integrals;
- first-order and second-order differential-equation models.

Result behavior

Most Calculus outputs are numerical approximations. Limits, derivatives, integrals, roots, extrema, partial derivatives, double integrals, and differential-equation results generally return calculator-flow scalar values tagged as [APPROX].

Some tools produce displayed mathematical structures rather than ordinary scalar operands. Tangent lines, Taylor polynomials, linearization and critical-point descriptions are treated as labeled outputs. Gradient and Hessian tools may display vector or matrix-style information while also returning a scalar companion value for calculator flow.

Because Calculus relies heavily on numerical methods, results should be interpreted as approximations rather than symbolic proofs. Step size, interval choice, function behavior, discontinuities, and invalid domains can affect the result.

Combinatorics Guide

Overview

The Combinatorics domain provides counting, arrangement, sequence, partition, and probability tools.

Use this docklet when you need to count possibilities, arrange objects, select groups, evaluate common integer sequences, or compute discrete probability values.

Most Combinatorics tools require integer inputs such as n, r, or k. Probability tools may also use decimal inputs, such as probability p.

Function groups

The Combinatorics docklet includes tools for:

- factorials;
- permutations;
- combinations;
- derangements;
- k-tuples;
- combinations with repetition;
- multinomial counts;
- Fibonacci numbers;
- Catalan numbers;
- Bell numbers;
- triangular numbers;
- Stirling numbers;
- Lah numbers;
- integer partitions;
- compositions;
- Eulerian numbers;
- hypergeometric probabilities;
- binomial probability mass values.

Result behavior

Small and moderate Combinatorics results usually return calculator-flow numeric values tagged as [NUM].

Very large exact integer results may be displayed as string-style output instead. This allows CCalc to show large combinatorial counts without forcing them into ordinary decimal operand behavior.

Approximate large results may be displayed in scientific notation and tagged as [APPROX].

Probability tools return decimal probability values. Extremely small probability values may display as zero when they are below the practical numeric threshold used by the docklet.

Because combinatorial values can grow very quickly, the user should treat very large results carefully. Some outputs are exact, some are approximate, and some are displayed as labels because they are too large to behave naturally as standard calculator-flow numbers.

Complex Numbers Guide

Overview

The Complex Numbers domain provides tools for entering, storing, transforming, and calculating with complex numbers.

This docklet works with a stored complex value:

z = Re + Im·i

The docklet can update this stored complex value, perform operations on it, display structured complex results, and return scalar values such as magnitude, argument, or unwrapped phase back into the normal calculator flow.

Function groups

The Complex Numbers docklet includes tools for:

- real and imaginary part entry;
- swapping real and imaginary parts;
- clearing the imaginary part;
- rectangular and polar conversion;
- conjugate;
- magnitude;
- argument;
- normalization;
- reciprocal;
- complex addition, subtraction, multiplication, and division;
- powers and roots;
- complex exponential and logarithmic functions;
- complex trigonometric functions;
- complex hyperbolic functions;
- stereographic-style projection;
- unit vector / cis input;
- approximate complex Gamma;
- roots of unity;
- phase unwrapping.

Angle behavior

Some Complex tools use the active CCalc angle unit.

The active angle unit affects tools such as:

- argument;
- rectangular/polar conversion;
- cis / unit vector;
- phase unwrap.

The complex logarithm keeps its internal imaginary component mathematically in radians, but displayed argument text follows the selected angle unit where appropriate.

Result behavior

Complex Numbers has a stronger distinction between scalar and structured output than many other domains.

Scalar results such as magnitude, argument, and phase unwrap can return calculator-flow numeric values.

Structured complex results such as complex numbers, root lists, roots of unity, and projection coordinates are usually displayed as labeled or approximate output rather than ordinary scalar operands.

In general:

- [NUM] is used when the docklet returns a plain scalar value.
- [APPROX] is used when a value is numerical or computed through approximation.
- [LABEL] is used when the output is primarily a complex representation, list, coordinate set, or other non-scalar display.

Cryptography Guide

Overview

The Cryptography domain provides integer-based tools for modular arithmetic, number theory, prime testing, factorization, public-key demonstrations, and bitwise operations.

This docklet is designed for cryptography-adjacent calculation rather than secure production cryptography. It helps the user inspect arithmetic structures commonly used in cryptographic systems: modular reduction, modular powers, inverses, congruences, primes, factors, RSA-style key relationships, Diffie–Hellman-style shared secrets, discrete logarithm searches, and bitwise transformations.

Function groups

The Cryptography docklet includes tools for:

- modular reduction;
- modular addition, subtraction, multiplication, and exponentiation;
- GCD;
- extended GCD;
- modular inverse;
- LCM;
- Euler's totient;
- prime testing;
- next prime;
- factorization;
- Chinese Remainder Theorem;
- RSA-style demonstration;
- Diffie–Hellman-style demonstration;
- discrete logarithm search;
- XOR;
- bit shifting.

Input behavior

The Cryptography docklet uses integer input. Decimal-style values are not appropriate because modular arithmetic, factorization, prime testing, bitwise operations, and public-key demonstrations are integer-based.

Several tools enforce additional constraints. For example, moduli must be nonzero, modular exponents must be nonnegative, and RSA or Diffie–Hellman demonstrations require prime inputs where appropriate.

Result behavior

Cryptography results fall into two main types.

Scalar numeric results can continue through the calculator flow. These include modular reduction, modular arithmetic, GCD, modular inverse, LCM, totient, next prime, and XOR.

Structured results are displayed as labels because they contain several values or a textual mathematical relationship. These include extended GCD, prime test, factorization, CRT, RSA, Diffie–Hellman, discrete logarithm, and bit shift.

The Cryptography docklet is intended for mathematical inspection, learning, and calculator-assisted number-theory work. It is not intended to replace dedicated cryptographic libraries, security tools, or production-grade cryptographic systems.

Engineering and Applied Math Guide

Overview

The Engineering and Applied Math domain provides applied formulas for decibels, circuit behavior, dynamics, kinematics, fluids, heat transfer, measurement, electrical networks, wire gauge, and simple mechanics of materials.

Use this docklet when the calculation is formula-driven and belongs to an applied physical, electrical, mechanical, or measurement context.

Most Engineering results return a numeric value directly into the calculator flow. The ticker identifies the formula used, the units of the result, and often the compact formula reference.

Function groups

The Engineering docklet includes tools for:

- decibel calculations;
- power and amplitude ratios;
- RC and RL time constants;
- cutoff frequency;
- low-pass filter magnitude and phase;
- RLC resonance and quality factor;
- second-order system metrics;
- overshoot, settling time, peak time, damping ratio, and natural frequency;
- rpm and angular velocity conversion;
- rotational power;
- constant-acceleration motion;
- dynamic pressure;
- Bernoulli pressure difference;
- Reynolds number;

- conductive and convective heat transfer;
- tolerance stack-ups;
- root-sum-square tolerance;
- linear interpolation;
- percent error;
- dilution;
- ideal gas pressure;
- equivalent series and parallel R, C, and L;
- capacitive and inductive reactance;
- voltage-divider output;
- electrical power formulas;
- AWG diameter and resistance estimates;
- stress;
- axial elongation;
- cantilever deflection.

Result behavior

Engineering functions usually return scalar numeric results and can therefore continue into the main calculator flow.

Some results are marked approximate when the computation represents an engineering estimate, a model-based formula, or a value derived from an approximation rather than an exact symbolic result.

The user should treat Engineering outputs as calculator-grade applied estimates unless the relevant appendix states that a formula is exact under its assumptions.

Equations Guide

Overview

The Equations domain provides algebra and equation-solving tools for linear, quadratic, cubic, polynomial, factoring, simplification, systems, intersections, and numerical root solving.

Unlike simple arithmetic docklets, many Equations tools return label-style results rather than a single calculator-flow number. This is because roots, factored forms, systems, and symbolic-style outputs often contain multiple values or formatted expressions.

Function groups

The Equations docklet includes tools for:

- linear equations;
- quadratic equations;
- cubic equations;
- discriminants;
- vertex and axis of symmetry;
- completing the square;
- polynomial evaluation;
- roots-to-coefficients;
- synthetic division;
- rational-root testing;
- Descartes’ rule of signs;
- simple factoring;
- GCF extraction;
- difference of squares;
- radical simplification;
- rationalization;
- binomial expansion;
- 2×2 and 3×3 systems;
- nonlinear intersections;
- Newton-style numerical solving.

Result behavior

The Equations docklet mixes flow-scalar and terminal-label results.

Flow-scalar results can continue directly into the calculator expression flow. Examples include linear solutions, discriminants, polynomial evaluation, rationalized decimal values, and expansion values.

Terminal-label results are shown as formatted output because they may contain roots, vectors, factored forms, symbolic text, or multiple values. Examples include quadratic and cubic roots, vertex forms, completed-square forms, systems, nonlinear intersections, simplified radicals, and factored expressions.

Equations uses the standard CCalc result-tag pattern:

- [NUM] for numeric calculator-flow results;
- [APPROX] for numerical approximations;
- [LABEL] for formatted symbolic or multi-value results.

Financials Guide

Overview

The Financials domain provides calculator-oriented tools for loans, investment returns, interest, growth, pricing, discounts, tax, and common consumer finance calculations.

It is designed as a practical financial calculator panel rather than a full accounting system. Most functions take one or more prompted values, compute a result, and return that result to the main CCalc display so it can continue to be used in later calculations.

The Financials docklet uses the app's current default currency label for money-related results. The currency label is informational. It tells the user which currency context the result belongs to, but it does not perform currency conversion.

Function groups

The Financials docklet includes tools for:

- payment;
- present value;
- future value;
- rate;
- number of periods;
- APR/APY conversion;
- CAGR;
- percent change;
- Rule of 72;
- simple interest;
- compound interest;
- inflation adjustment;
- net present value;
- internal rate of return;
- modified internal rate of return;
- profitability index;
- amortization;
- loan balance;
- margin;
- breakeven;
- ROI;
- discount;
- tax;
- tip and split.

Input behavior

Several Financials tools use cashflow lists. In standard dot-decimal mode, values are separated with commas. In comma-decimal mode, values are separated with semicolons to avoid confusing decimal commas with list separators.

Some time-value-of-money tools include a payment timing option. Payments may occur at the end of each period or at the beginning of each period.

Result behavior

Financials functions usually return numeric results to the main calculator display.

For example, a tax calculation may return the final total after tax. A loan payment calculation may return the payment amount. An ROI calculation may return a percentage result.

The ticker shows the financial function and its input context. It does not simply repeat the final result. This keeps the ticker useful as a compact explanation of what was calculated.

Financial results are generally tagged as [NUM].

Some calculations use numerical solving, especially RATE and IRR. These may fail when the input pattern does not produce a stable bracket or a real solution.

Financials is useful for estimation and calculator workflows. It is not a replacement for legal, tax, banking, or accounting software.

Geometry Guide

Overview

The Geometry domain provides shape, coordinate, angle, surface, volume, and topology tools for working with common 2D and 3D geometry directly from the calculator.

Use this docklet when you want to calculate geometric quantities from entered dimensions, coordinates, radii, angles, or counts.

Most Geometry tools return numeric results to the main CCalc operand, so the result can immediately be reused in further calculations.

Function groups

The Geometry docklet includes tools for:

- circle area;
- circumference;
- arc length;
- sector area;
- segment area;
- chord length;
- sagitta;
- rectangle area and perimeter;
- square area;
- parallelogram area;
- trapezoid area;
- triangle area;
- Heron’s formula;
- triangle perimeter;
- right-triangle side calculation;
- incircle radius;
- ellipse area;
- approximate ellipse perimeter;
- regular polygon area;
- polygon apothem;
- Pick’s theorem;
- 2D and 3D distance;
- midpoint;
- slope;
- angle from three points;
- point-to-line distance;
- sphere volume and surface area;
- spherical cap volume;
- cube volume and surface area;
- box volume and surface area;
- cylinder volume and surface area;
- cone volume and surface area;
- pyramid volume;
- square pyramid surface area;
- frustum volume;
- torus volume;
- degrees/radians conversion;
- law of cosines;
- law of sines;
- Euler characteristic;
- genus.

Angle behavior

The Geometry docklet uses the global CCalc angle unit where appropriate.

Angle-sensitive tools interpret angle inputs according to the current setting:

- degrees;
- radians;
- gradians;
- turns.

The explicit degrees/radians converter is separate from the global angle mode. It directly converts degrees to radians or radians to degrees based on its mode field.

Input behavior

Some Geometry tools allow a prompt field to be left blank. When the prompt says blank = operand, CCalc uses the current calculator operand as the input.

Coordinate-based tools group related fields visually. 2D points are shown as paired fields, and 3D points are shown as triplets.

Result behavior

Most Geometry functions return direct numeric results and are tagged as [NUM].

Approximate formulas, such as ellipse perimeter, are tagged as [APPROX].

Point-style or descriptive outputs, such as midpoint display, may be tagged as [LABEL] because the full result is a structured coordinate pair rather than a single ordinary scalar.

The Geometry docklet assumes consistent units. If a radius is entered in centimeters, area is returned in square centimeters and volume is returned in cubic centimeters.

Linear Algebra Guide

Overview

The Linear Algebra domain provides vector, matrix, system-solving, decomposition, eigenvalue, and matrix-analysis tools directly inside CCalc.

Use this docklet when you want to work with 2D or 3D matrices, 3D vectors, matrix products, determinants, inverses, ranks, eigenvalues, orthogonalization, and related linear algebra operations.

Most scalar results return directly to the main calculator operand. Matrix and vector results are shown as label-style results in the display and ticker.

Function groups

The Linear Algebra docklet includes tools for:

- dot product;
- cross product;
- vector norm;
- vector normalization;
- angle between vectors;
- vector distance;
- projection;
- scalar triple product;
- vector addition and subtraction;
- scalar-vector multiplication;
- matrix addition and subtraction;
- scalar-matrix multiplication;
- matrix multiplication;
- matrix-vector multiplication;
- matrix powers;
- transpose;
- commutator;
- trace;
- determinant;
- inverse;
- rank;
- condition number;
- RREF;
- LU decomposition;
- QR decomposition;
- Gram–Schmidt orthogonalization;
- solving systems;
- characteristic polynomial;
- eigenvalues;
- eigenvectors;
- SVD.

Input behavior

Vector fields are grouped horizontally where possible.

A 3D vector is entered as:

x y z

Two vectors are entered as:

ux uy uz

vx vy vz

A 2×2 matrix is entered row by row:

A11 A12

A21 A22

A 3×3 matrix is entered row by row:

A11 A12 A13

A21 A22 A23

A31 A32 A33

For matrix tools, the first prompt usually asks for matrix size. Enter 2 for a 2×2 matrix or 3 for a 3×3 matrix. After tapping Compute, the prompt expands to show the required matrix fields.

Some vector tools allow the first component to use the current calculator operand when left blank.

Result behavior

The Linear Algebra docklet has two result behaviors.

Scalar results are committed to the main calculator operand and can continue through the normal expression flow. Examples include dot product, norm, angle between vectors, distance, determinant, trace, rank, condition number, and first eigenvalue.

Vector and matrix results are shown as label-style results. Examples include cross product, normalized vector, matrix inverse, matrix product, RREF, LU decomposition, QR decomposition, SVD, and eigenvectors.

The docklet uses:

- [NUM] for direct numeric results;
- [APPROX] for numerical approximations;
- [LABEL] for vector, matrix, or descriptive results.

Current limits include 3D vectors, 2×2 and 3×3 matrices, non-negative integer matrix powers, and real-valued eigenvalue/eigenvector behavior. Complex spectral results are reported as unsupported.

Number Theory Guide

Overview

The Number Theory domain provides integer-focused tools for divisibility, digit properties, primes, factorization, modular arithmetic, integer sequences, congruences, and special number tests.

Use this docklet when you want to test or transform integers, factor numbers, work modulo n, compute number-theoretic functions such as $\phi(n)$, $\lambda(n)$, $\mu(n)$, or evaluate integer sequences such as Fibonacci, Catalan, triangular, pentagonal, and partition numbers.

Function groups

The Number Theory docklet includes tools for:

- even/odd testing;
- perfect-square testing;
- integer square root;
- digit sum;
- digital root;
- digit count;
- reverse digits;
- divisor count;
- divisor sum;
- aliquot sum;
- divisor list;
- Euler's $\phi(n)$;
- Carmichael $\lambda(n)$;
- Möbius $\mu(n)$;
- prime testing;
- Miller–Rabin-style testing;

- factorization;
- prime factors;
- previous/next prime;
- nth prime;
- prime count $\pi(x)$;
- GCD and LCM;
- extended GCD;
- modular reduction;
- modular power;
- modular inverse;
- discrete logarithm;
- multiplicative order;
- primitive root;
- linear congruence solving;
- Fibonacci;
- factorial;
- binomial coefficient;
- Catalan number;
- triangular number;
- pentagonal number;
- Chinese Remainder Theorem;
- Legendre symbol;
- Jacobi symbol;
- continued fraction;
- partition number;
- perfect, abundant, and deficient tests;
- palindrome test;
- Armstrong-number test.

Input behavior

Number Theory is integer-based. Prompt fields sanitize input as integers.

Some tools require nonnegative values, positive moduli, prime inputs, odd moduli, or values within practical computation caps. If a value violates the required condition, CCalc reports an explanatory error rather than committing a misleading result.

Result behavior

Many Number Theory tools return scalar integer values that can continue through the calculator flow. Examples include integer square root, digit sum, digital root, divisor count, divisor sum, $\phi(n)$, $\lambda(n)$, $\mu(n)$, modular power, modular inverse, multiplicative order, Fibonacci values, binomial coefficients, Catalan numbers, triangular numbers, pentagonal numbers, Legendre symbols, and Jacobi symbols.

Some outputs are descriptive or structured and are treated as label-style results. Examples include even/odd status, prime/composite status, factorization strings, divisor lists, extended GCD coefficients, CRT solutions, continued fractions, and special number classifications.

Large exact integer results may be displayed as labels when they are too large to behave naturally as ordinary calculator operands.

Statistics Guide

Overview

The Statistics domain provides descriptive statistics, distribution tools, probability tools, confidence intervals, hypothesis-test helpers, correlation, regression, and data-summary functions.

Use this docklet when you want to calculate from a list of measured values, evaluate a distribution, compare a statistic to a reference value, or inspect a relationship between two data series.

Function groups

The Statistics docklet includes tools for:

- mean;
- median;
- mode;
- minimum and maximum;
- range;
- sum;
- sample standard deviation;
- sample variance;

- coefficient of variation;
- quartiles;
- interquartile range;
- percentile;
- normal CDF;
- inverse normal;
- z-score;
- t-score;
- t-distribution CDF;
- binomial probability;
- Poisson probability;
- confidence interval for a mean;
- confidence interval for a proportion;
- one-sample t-test;
- two-sample t-test;
- z-test;
- chi-square CDF;
- p-value helper;
- correlation;
- linear regression;
- R^2;
- count;
- five-number summary.

Input behavior

Statistics often uses lists of values.

In dot-decimal mode, list values are separated with commas:

1.2,2.3,3.4

In comma-decimal mode, list values are separated with semicolons:

1,2;2,3;3,4

This avoids confusion between decimal commas and list separators.

Some tools require paired lists, such as X and Y values for correlation and regression. These lists must contain the same number of values.

Result behavior

Simple descriptive statistics usually return scalar numeric values and can continue through the calculator flow.

Structured summaries, such as min/max, quartiles, confidence intervals, two-sample test details, regression equations, and five-number summaries, may be displayed as label-style output because they contain multiple values.

Many distribution and test tools return approximate values because they rely on numerical approximations or distribution formulas. These results are tagged as [APPROX] where appropriate.

Statistics is intended for practical calculator analysis, not as a replacement for a full statistical package.

Trigonometry Guide

Overview

The Trigonometry domain provides trigonometric, inverse trigonometric, hyperbolic, and inverse hyperbolic functions.

Use this docklet when the current calculator operand should be passed through a trigonometric or hyperbolic function without opening a larger prompt-based domain.

Unlike many other docklets, Trigonometry works directly from the current calculator operand. Selecting a function immediately applies it to the current value.

Function groups

The Trigonometry docklet includes tools for:

- sine;
- cosine;
- tangent;
- cosecant;
- secant;
- cotangent;

- inverse sine;
- inverse cosine;
- inverse tangent;
- inverse cosecant;
- inverse secant;
- inverse cotangent;
- hyperbolic sine;
- hyperbolic cosine;
- hyperbolic tangent;
- hyperbolic cosecant;
- hyperbolic secant;
- hyperbolic cotangent;
- inverse hyperbolic sine;
- inverse hyperbolic cosine;
- inverse hyperbolic tangent;
- inverse hyperbolic cosecant;
- inverse hyperbolic secant;
- inverse hyperbolic cotangent.

Angle behavior

Ordinary trigonometric functions use the active CCalc angle unit.

If the angle unit is degrees, sin(30) means sin of 30 degrees. If the angle unit is radians, sin(30) means sin of 30 radians.

Inverse trigonometric functions return angles in the active angle unit.

Hyperbolic functions use plain numeric input and do not depend on the angle unit.

Result behavior

Trigonometry results are numerical and are usually tagged as [APPROX].

The ticker shows the function and argument, including the angle unit when relevant. In radians, CCalc may display recognizable π-multiple arguments when the value is close enough to a clean multiple of π.

If a function is undefined for the input, such as division by zero or an out-of-range inverse function, CCalc reports a labeled error instead of committing an invalid result.

Choosing the right domain

Many calculations can be approached from more than one domain.

For example:

- a triangle problem may belong in Geometry or Trigonometry;
- a slope or rate-of-change problem may belong in Calculus or Graphing;
- a system of equations may belong in Equations or Linear Algebra;
- a formula involving resistance and capacitance may belong in Engineering;
- a list of measured values may belong in Statistics;
- a whole-number divisibility question may belong in Number Theory.

The best domain is usually the one that matches the structure of the problem.

Use Trigonometry for angle functions.

Use Geometry for shape measurements.

Use Statistics for lists of values.

Use Linear Algebra for matrices and vectors.

Use Equations for solving equation templates.

Use Engineering for applied physical or technical formulas.

Use Calculus for rates, areas, roots, extrema, and numerical calculus tools.

Use Number Theory for integer properties, modular arithmetic, primes, and factorization.

Use Cryptography for modular arithmetic in cryptographic-style contexts.

Use Complex Numbers for calculations involving real and imaginary parts.

Use Combinatorics for counting, arrangements, and discrete probability.

Use Financials for loans, interest, investment return, pricing, tax, discounts, and tips.

Numeric, approximate, and labeled domain results

Different domains return different kinds of output.

A simple scalar result may return a direct numeric value:

sin(30°) [NUM]

A numerical method may return an approximate value:

root ≈ 1.4142 [APPROX]

A structured or non-flow result may return a label:

matrix result [LABEL]

This distinction matters because not every displayed result should be treated as a normal calculator operand.

In general:

- numeric scalar results can usually continue through the calculator flow;
- approximate scalar results can usually continue, but should be interpreted as estimates;
- labeled results are usually meant to be read, copied, or interpreted rather than inserted into the arithmetic flow.

The CCalc Grapher: A Comprehensive Guide

Opening the Grapher

The Grapher is opened from the Graphing entry in Math Domains.

Unlike standard docklets, the Grapher opens as a full-screen workspace. This gives enough room for the graph canvas, function entry, range controls, tracing, analysis, history, sharing tools, and gesture-based navigation.

The Grapher is designed for functions of x.

A typical workflow is:

1. Open the Grapher from Math Domains.
2. Enter a function.
3. Adjust the visible range if needed.
4. Tap Graph.
5. Inspect the plotted curve.
6. Trace the curve to read x, y, and slope values.
7. Review the Analysis section.
8. Use Compare a if the function contains the calculator parameter a.
9. Send a traced y value back to CCalc if needed.

The Grapher is not only a drawing tool. It is a calculation workspace connected to the main calculator.

Why the Grapher is separate from standard docklets

Most CCalc docklets are compact panels that appear over the main calculator interface.

The Grapher is different because graphing needs more space.

A graph requires:

• a large plotting area;
• visible axes and grid context;
• function entry controls;
• range controls;
• trace information;
• analysis results;
• history;
• sharing tools;
• room for gestures such as zooming and recentering.

For this reason, the Grapher uses a full-screen layout instead of the standard docklet format.

This keeps the graph readable and prevents the graphing controls from crowding the main calculator keypad.

Entering functions

Functions are entered in the expression field at the top of the Grapher.

The function is interpreted as:

y = f(x)

The user enters only the expression on the right side.

For example, entering:

sin(x)

means:

y = sin(x)

The user does not need to type y =.

Examples of valid function-style entries include:

x^2

sin(x)

cos(x)

sqrt(x)
ln(x)
abs(x)

sin(x) + sqrt(x)
1 / sin(tan(a - x))
a*x + 1
a*sin(x)
sin(a*x)

The expression should use the tokens and operators supported by the Grapher.

The Grapher is intended for readable function entry rather than free-form programming syntax.

Using the token tray

The token tray provides quick entry buttons for common graphing tokens.

It helps the user build functions without manually typing every symbol. This is especially useful on smaller screens, where direct text entry can be slower or more error-prone.

The token tray can include:

• numbers;
• x;
• a;
• ANS;
• π;
• e;
• operators;
• parentheses;
• functions such as sin, cos, tan, sqrt, ln, log10, and abs;
• edit controls such as delete-left and Clear.

Function tokens are inserted in graph-friendly form.

For example, tapping:

sin
inserts:

sin(x)

and tapping:

sqrt
inserts:

sqrt(x)

This helps preserve valid expression syntax.

The token tray can also build nested expressions. When a function token is inserted after another completed function expression, CCalc may wrap the existing expression inside the newly selected function.

For example:

sin
abs
can produce:

abs(sin(x))

Here, the second function becomes the outer function.

Operators create separate expression parts rather than wrapping the previous function. For example:

tan
-
cos
produces:

tan(x)-cos(x)

If a function token is inserted after another completed function on the right side of an operator, CCalc can wrap that most recent function expression.

For example:

tan

-

cos
sqrt
can produce:

tan(x)-sqrt(cos(x))

The Clear token clears the expression, plotted curve, trace point, slope, tangent line, Analysis data, and any active Compare a overlay.

Supported graph tokens

The Grapher supports a focused set of graphing tokens.

Common supported tokens include:

• x
• a
• ANS
• π
• e
• numeric values
• +, −, ×, ÷
• powers
• parentheses
• sin
• cos
• tan
• sqrt
• ln
• log10
• abs

These tokens allow the user to build a wide range of common mathematical expressions.

Examples:

x^2 + 3*x - 2

sin(x)

cos(x) + 1
sqrt(x)
ln(x)
abs(x - 2)

sin(x + sqrt(x))

The supported token set is intentionally practical. It covers common graphing needs while keeping the interface understandable.

Using x, a, ANS, π, and e

The Grapher supports several special values.

x

x is the independent graphing variable.

The Grapher evaluates the function by giving x many different values across the visible range.

For example:

x^2

means that y is calculated from x squared at each sampled point.

a

a is the calculator operand passed into the Grapher.

When the Grapher opens, the current value from the main calculator is available inside graph expressions as a.

For example, if the calculator currently shows:

5

then the graph expression:

sin(a - x)

is interpreted as:

sin(5 - x)

This allows the Grapher to use the calculator’s current operand as a graph parameter without requiring the user to retype it.

The value a is especially useful for parameter-based graphing, sensitivity checks, and Compare a exploration.

Changing a changes the function because the expression is being evaluated with a different calculator-supplied parameter value.

ANS

ANS represents the most recent completed calculator result.

Using ANS in a graph expression allows the user to include the previous calculator result in the function.

For example:

x + ANS

This lets the graph connect to the main calculator workflow.

a and ANS are related to calculator state, but they are not the same.

a represents the current calculator operand passed into the Grapher.

ANS represents the most recent completed calculator result.

π

π inserts pi.

It is especially useful for trigonometric and periodic functions in radians mode.

Examples:

sin(π*x)

cos(x / π)

When the Grapher is using a degree-based angle mode, trigonometric arguments are interpreted according to that mode. In that case, a π-based expression such as sin(π*x) is interpreted using π as a numeric value inside a degree-mode argument, not as an automatic switch to radians.

e

e inserts Euler’s number.

It is useful for exponential and logarithmic expressions.

Examples:

e^x

ln(e + x)

Together, x, a, ANS, π, and e allow the Grapher to work with variables, the current calculator operand, previous results, and common mathematical constants.

Graphing a function

After entering a function, tap Graph to draw it.

The Grapher evaluates the expression across the visible x range and plots the corresponding y values.

If a function is not defined for some x values, the Grapher may break the curve instead of drawing a misleading line across an invalid region.

For example:

sqrt(x)

is not defined for negative x values in real-number graphing.

The Grapher avoids connecting invalid points across the undefined part of the function.

Similarly:

1/x

does not draw a false vertical line through zero.

And:

tan(x)

avoids connecting large jumps across vertical asymptotes.

CCalc also refines certain domain boundaries numerically. For example, sqrt(x) with a range that includes negative x values can identify the valid boundary near:

x = 0

more accurately than a simple fixed sampling grid.

If the function has no valid points in the selected range, the graph displays a small status message.

Setting the visible range

The visible range controls which part of the function is shown.

The range includes four fields:

x_1

x_2

y_1

y_2

where:

x_1 = left edge of the graph
x_2 = right edge of the graph
y_1 = bottom edge of the graph
y_2 = top edge of the graph

For example:

x_1 = -10
x_2 = 10
y_1 = -10
y_2 = 10

shows the standard centered graph window.

Changing the range changes the graph window.

A narrower range zooms into a smaller part of the function.

A wider range shows more of the function at once.

Range control is important because different functions need different viewing windows.

For example:

sin(x)

is easy to see on a standard range.

But a function with restricted domains or large vertical movement may require a custom range. For example, for:

tan(x)-sqrt(x)*18

a useful range in radians may be:

x_1 = 0
x_2 = 10
y_1 = -80
y_2 = 40

because sqrt(x) is undefined for negative x, and the -18√x term pulls the graph downward.

If the range fields are invalid, unordered, or non-finite, CCalc shows an invalid range message and clears the plotted graph and active Compare a overlay.

Zooming

The Grapher supports zooming to inspect the curve more closely or view more of the function.

Zooming changes the visible graph window.

Zooming in makes the visible range smaller.

Zooming out makes the visible range larger.

This is useful when:

• a curve is too compressed;
• a root or intersection needs closer inspection;
• local behavior needs to be examined;
• the function changes rapidly in a small area;
• the user wants to see more of the surrounding curve.

Zooming is part of the visual exploration workflow.

It does not change the function itself. It only changes the viewing window.

If Compare a is active, the overlay curves are rebuilt for the new zoomed range.

Trace, slope, and tangent data clear when the graph is recalculated through zooming. This prevents stale trace information from remaining after the visible range changes.

Recentering

Recentering moves the visible graph window so the user can focus on a different part of the curve.

Double-tapping on the graph recenters the visible x range around the double-tap location.

This is an x-only translation. It changes:

x_1

x_2

while preserving:

y_1

y_2

This is useful for moving left or right along a curve without changing the vertical view.

For example, if a curve's interesting behavior is near x = 20, but the current range is centered around x = 0, recentering helps bring that region into view.

Recentering changes the view, not the function.

If Compare a is active, the overlay curves are rebuilt for the re-centered range.

Fitting the y-range

The Range section includes a Fit y control.

After a graph has been calculated, Fit y adjusts only the vertical range:

y_1

y_2

The horizontal range remains unchanged:

x_1

x_2

CCalc examines the valid plotted y values in the current x window and adjusts the y-range to show the visible vertical extent of the function with padding.

Fit y is useful when the curve is too high, too low, too flat, or mostly outside the visible window.

For smooth functions, this usually gives a clean fitted view.

For functions with strong asymptotes or spikes, the fitted range may become very large because the graph includes extreme y values near discontinuities.

If Compare a is active, Fit y recalculates the main curve and refreshes the Compare a overlay curves for the new fitted range.

Tracing the curve

After a curve is graphed, the user can tap or drag on the graph to select a nearby point on the function.

The trace display shows:

trace: x=..., y=..., slope≈...

The selected point is marked on the curve. A projection marker shows the selected x position on the x-axis.

The traced value represents the sampled function output at the selected graph point:

y = f(x)

Trace mode turns the graph from a static image into an interactive calculation surface.

Trace selection follows the main curve only. If Compare a is active, the faint overlay curves are not trace targets.

Reading x and y values

The trace display shows the selected x value and corresponding y value.

The x value identifies the input.

The y value identifies the function output at that input.

For example, for the function:

x^2

a trace point may show:

x = 4

y = 16

The displayed values are useful for quick inspection, approximation, and calculation follow-up.

Because graphing is visual and sampled, traced values should be understood as graph-assisted numeric readings, not always exact symbolic solutions.

Reading the slope

The Grapher can show the slope at the traced point when slope information is available.

The slope describes how fast the function is changing at that point.

A positive slope means the curve is rising.

A negative slope means the curve is falling.

A slope near zero means the curve is relatively flat at that point.

For example, on a curve with a local maximum or local minimum, the slope may be close to zero.

Reading the slope is useful for:

• understanding increasing and decreasing behavior;
• identifying flat points;
• inspecting tangent behavior;
• studying local function behavior;
• connecting graphing with derivative intuition.

The slope is a numerical estimate, not a symbolic derivative.

Tangent line

When a trace point is selected and a slope can be estimated, CCalc draws a subtle dashed tangent line through the selected point.

The tangent line represents the local direction of the curve at the trace point.

For a selected point:

(x_0, y_0)
with slope:

m

the tangent line corresponds to:

$y = y_0 + m(x - x_0)$
The tangent line is a visual aid.

It clears when the graph is recalculated, zoomed, reset, re-centered, or replaced by a new graph.

The tangent line belongs to the main curve only. Compare a overlays do not receive tangent lines.

Sending y back to CCalc

When a trace point is selected, the Send y button appears.

Tapping Send y sends the selected y value back to the main calculator as the current operand.

For example, if the trace shows:

x=6.4627, y=2.3636
then tapping Send y sends:

2.3636

back into the calculator.

The user can then continue calculating with that value.

CCalc also records the graph result in the Tape using a descriptive line such as:

Graph y=sqrt(x)-tan(x)*cos(x), x=6.4627 → y=2.3636
This keeps the graph result connected to the function and the selected x-coordinate.

Compare a overlays are not sent back to the calculator. Send y always sends the selected value from the main curve.

Understanding the Analysis panel

The Analysis panel summarizes important visible behavior of the plotted function.

It helps the user understand the curve without relying only on visual inspection.

The Analysis panel may include:

• visible minimum and maximum;
• roots;
• local minima and maxima;
• domain gaps;
• vertical asymptotes;

• visible y-range;
• Compare a information when parameter exploration is active.
The Analysis panel is based on the currently visible graph range.

This means changing the range, zooming, recentering, or fitting the graph may change the reported analysis.

Analysis is always based on the main curve only. Compare a overlays do not affect Analysis values.

Visible minimum and maximum

Visible minimum and visible maximum describe the lowest and highest sampled y values found in the current visible graph window.

These values help the user understand the vertical spread of the curve.

For example, the Analysis panel may show:

visible min y=-1.4142 @ x=-0.7854
visible max y=1.4142 @ x=2.3562
These are visible sampled extremes, not necessarily local turning points.

For functions with asymptotes, a visible minimum or maximum may come from the graph approaching the edge of the visible y-range rather than from a true local extremum.

The word visible is important.

The Grapher is reporting what it finds inside the current viewing range, not necessarily the absolute minimum or maximum of the entire mathematical function.

Roots

Roots are x values where the function crosses or touches the x-axis.

At a root:

y = 0
For example, the function:

x - 3

has a root at:

x = 3
because:

3 - 3 = 0
The Grapher can identify roots that appear within the visible range.

Roots outside the current range are not shown until the view is changed to include them.

Root detection is numerical and depends on the current graph window and sampling.

Local minima and maxima

Local minima and local maxima are turning points or nearby extreme points in the visible curve.

A local minimum is a point where the curve is lower than nearby points.

A local maximum is a point where the curve is higher than nearby points.

For example, a wave curve may have several local maxima and minima across the visible range.

The Grapher reports these as visible local features.

As with other analysis results, these are range-dependent and graph-based. A different viewing range may show different local extrema.

Local extrema are numerical estimates. They are useful for graph inspection, but they are not symbolic proof.

Domain gaps

Domain gaps are places where the function is not defined or where the Grapher detects a break in the curve.

For example:

sqrt(x)
has no real value for:

x < 0
and:

ln(x)
has no real value for:

x ≤ 0

Similarly, functions with division by zero may have breaks where the denominator becomes zero.

A gap count and a gap interval may both appear.

These are not duplicates.

For example:

Gaps: 1

means the Grapher detected one gap.

A line such as:

Gap: [-1.9656, 0.0018]
describes the approximate x-interval where that gap occurs.

The first item is the count. The second item gives the location.

Vertical asymptotes

A vertical asymptote is a place where the function grows very large in magnitude near a certain x-value.

For example:

1 / x
has a vertical asymptote at:

x = 0
because the function becomes unbounded near zero.

Another example is:

tan(x)
In radians mode, tan(x) has vertical asymptotes near:

π/2 + nπ
In degrees mode, the corresponding locations are:

90° + n·180°
The Grapher may detect visible vertical-asymptote behavior and report it in the Analysis panel.

Asymptote detection is conservative and numerical. It is based on visible graph behavior such as large jumps and broken curve segments. It may miss some asymptotes and may not identify every mathematical discontinuity.

Visible y-range

The visible y-range describes the vertical span of the plotted function in the current view.

It helps the user understand how much the function varies over the selected x-range.

For example, a graph may show:

visible y [-1.4142, 1.4142]
This is different from the graph window's y-axis range.

The graph window's y-axis range is set by:

y_1
y_2
The visible y-range describes the function values found within the visible graph, not merely the axis settings.

Compare a parameter exploration

Compare a is available for expressions that use a.

In CCalc, a represents the current calculator operand passed into the Grapher.

Compare a shows how the graph would change if that operand were adjusted slightly.

For example, if the calculator operand is:

5

and the expression is:

sin(a - x)

the main curve uses:

sin(5 - x)

Compare a can then show nearby versions of the curve using values slightly below and above 5.

This helps the user understand how the current calculator operand affects the shape or position of the graph.

Compare a is useful for:

• sensitivity checks;
• parameter exploration;
• coefficient testing;
• understanding shifts;
• seeing how the calculator operand affects curve shape or position.

Compare a is meant to be unobtrusive. It supports exploration without turning the graph into a crowded multi-curve workspace.

Compare a availability

Compare a appears only when it can be used meaningfully.

Compare a is hidden when:

• the expression does not contain standalone a;
• no valid graph has been plotted;
• the plotted curve has fewer than two valid points.

When Compare a is off, the graph shows only the main curve:

y = f(x, a)

where a is the current calculator operand.

When Compare a is enabled, CCalc draws two faint overlay curves behind the main curve:

y = f(x, a - Δ)
y = f(x, a + Δ)

The main curve remains unchanged and visually dominant. It still represents the actual calculator operand exactly.

The overlay curves are intentionally subtle. They are meant for sensitivity analysis, not decorative curve manipulation.

The question Compare a answers is:

How does this graph change if the calculator parameter a changes slightly?

Compare a delta

The Compare a chip displays the active delta value:

Compare a ±0.125

The delta is based on the current calculator operand and the selected sensitivity scale:

Δ = max(abs(a) * selected scale, 0.1)

The default selected scale is:

5%

For example, if:

a = 2.5

then:

abs(2.5) * 0.05 = 0.125

so Compare a displays:

Compare a ±0.125

and the overlay curves use:

a - 0.125

a + 0.125

The minimum delta is 0.1, so very small a values still produce a visible comparison.

Adjusting Compare a sensitivity

When Compare a is enabled, two small buttons appear beside the chip:

− Compare a ±Δ +

The − button decreases the sensitivity step.

The + button increases the sensitivity step.

The available sensitivity scales are:

1%

1.5%

2%

3%

5%

7.5%

10%

15%

20%

30%

The default is:

5%

Changing the sensitivity immediately recalculates the overlay curves.

The main curve does not change.

This means:

main curve: f(x, a)
lower overlay: f(x, a - Δ)
upper overlay: f(x, a + Δ)
The − and + buttons only change Δ.
They do not change the calculator operand itself.

Compare a examples

For:

a*sin(x)
a controls vertical scale. For positive a, this is the amplitude; for negative a, the curve is also reflected across the x-axis.

If:

a = 2.3
Δ = 0.115
the curves are:

main: 2.3*sin(x)
lower: 2.185*sin(x)
upper: 2.415*sin(x)
The overlays appear slightly above and below the main curve, especially near peaks and troughs.

For:

sin(a*x)
a controls the horizontal frequency or rate of oscillation. Larger |a| values make the oscillations occur more rapidly. Negative a reverses the sign of the sine output because sin(-u) = -sin(u).

The curves are:

main: sin(a*x)
lower: sin((a - Δ)*x)
upper: sin((a + Δ)*x)
Near x = 0, all three curves may visually merge because:
sin((a - Δ)*0) = sin(0)
sin(a*0) = sin(0)
sin((a + Δ)*0) = sin(0)
Farther from zero, the phase difference accumulates and the overlay curves separate from the main curve.

For:

a + 0.5*tan(x)
or equivalently:

a + sin(x)/(2*cos(x))
a shifts the graph vertically.

Compare a shows nearby vertically shifted branches. Since tangent-like functions have repeated asymptotes, the comparison can look busier near discontinuities.

For:

1 / sin(tan(a - x))
changing a shifts the internal relationship between a and x.

This can move steep regions, undefined regions, and repeating features across the x-axis.

Compare a helps the user see that behavior visually.

Compare a and graph operations

Compare a refreshes automatically when the graph is recalculated through:

• Graph;
• Fit y;
• pinch zoom;
• double-tap x-recenter.

Compare a turns off and clears its overlays when:

• Reset is tapped;
• the expression is cleared;
• an invalid range is submitted;
• an empty expression is graphed;
• a history item is loaded;
• the new graphed expression does not use a.

Trace, tangent, slope, and Analysis remain based on the main curve only.

Compare a overlays are visual reference curves and do not affect trace selection or Analysis values.

Graph history

The Grapher keeps a recent function history.

When a function is graphed successfully, CCalc stores the expression and its graph range.

The history can be opened from the Function section using the history icon.

A recalled history item restores:

• function expression;
• x_1;
• x_2;
• y_1;
• y_2;
• angle mode record.

The recalled function is not treated as a static picture. It is ready to be graphed again.

This is especially useful for expressions that use a.

For example:

a*sin(x)

can be recalled later and re-evaluated using the calculator’s current operand as the new value of a.

When a history item is loaded, the current plotted graph is cleared until Graph is tapped again.

Compare a is also turned off and any overlay curves are cleared, so the recalled setup starts cleanly.

The Tape records graph results. Function History stores reusable graph setups. These serve different purposes.

Gesture navigation

The Graphing workspace supports direct gesture navigation on the graph canvas.

Tap or drag

A one-finger tap or drag traces the curve and selects the nearest graph point.

This is used to inspect:

• x;
• y;
• slope.

It is also used to prepare a value for Send y.

Trace selection follows the main curve only.

Pinch zoom

Pinching on the graph zooms the visible range in or out.

Pinch outward:

zooms in

Pinch inward:

zooms out

The range fields update after the zoom, and the graph is recalculated for the new visible window.

Trace, slope, and tangent data clear when the graph is recalculated through zooming.

If Compare a is active, the overlay curves are rebuilt for the new range.

Double-tap recenter

Double-tapping on the graph recenters the visible x range around the double-tap location.

This changes:

x_1

x_2

while preserving:

y_1

y_2

If Compare a is active, the overlay curves are rebuilt for the re-centered range.

Copying Analysis values

Trace and Analysis values are selectable.

The user can select and copy individual values such as:

trace: x=..., y=..., slope≈...
local max y=... @ x=...
asymptotes x≈...
visible y [...]

These fields are not editable.

They are selectable text for convenience.

Sharing graph images

The Grapher can share a graph image.

The share icon is dimmed until a graph has been calculated.

Once a graph exists, the graph can be shared as an image with accompanying context.

The shared image can include:

• graph canvas;
• main curve;
• Compare a overlay curves, if enabled;
• trace marker, if selected;
• x-axis projection, if selected;
• tangent line, if available;
• compact graph caption.

A shared image is useful when the user wants to save, send, or document the visual graph.

This is useful for:

• notes;
• reports;
• messages;
• technical review;
• saving a visual result;
• showing a function’s behavior to someone else.

Sharing a graph image preserves the visual result.

Sharing graph reports

The Grapher can also share a text report.

A graph report is useful when the user wants the numeric and analytical context rather than only the image.

A report may include information such as:

• function;
• x-range;

• y-range;
• angle mode;
• decimal setting;
• trace point, if selected;
• slope, if available;
• Analysis values;
• visible minima and maxima;
• roots;
• gaps;
• asymptote information;
• Compare a information when parameter exploration is active.

For example:

CCalc Graph

y=tan(x)
x[-10, 10]
y[-10, 10]
angle: rad

Decimals: 8

trace: x=7.97552836, y=-8.18673279, slope≈68.02277702
Analysis

visible min: y=-9.87491184 at x=-7.75305895
visible max: y=9.87491184 at x=7.75305895
roots: -9.42157953, -6.28476085, -3.14794216, 0 +3
gaps: none

asymptotes x≈-7.85317019, -4.7163515, -1.57953281, 1.57953281, 4.7163515 +1
visible y: [-9.87491184, 9.87491184]

If no trace point is selected, the shared report omits trace and slope data.

Compare a overlays are included in the shared image when Compare a is active.

The text report remains focused on the main curve and Analysis values.

A graph image shows what the curve looks like.

A graph report explains what the Grapher found.

Reset

The Reset button restores the Graphing workspace to its default state.

Reset clears:

• function text;
• range fields;
• active graph range;
• plotted curve;
• Compare a state;
• Compare a overlay curves;
• trace marker;
• slope;
• tangent line;
• graph message;
• Analysis data;
• token cursor state.

The default range is restored to:

$x_1 = -10$
$x_2 = 10$
$y_1 = -10$
$y_2 = 10$

Grapher precision and limitations

The Grapher is a numerical and visual tool.

It samples the function across the visible range and uses numerical methods to draw and analyze the curve.

This makes it practical and interactive, but it also means results should be understood as graph-based approximations.

Important limitations include:

• roots are detected within the visible range;
• local minima and maxima depend on sampling and range;
• gaps are approximate detected intervals;
• asymptotes are inferred from visible behavior;
• slope readings are approximate;
• tracing gives numeric readings from the plotted curve;
• changing the range may change the analysis results;
• some functions may require zooming or range adjustment to understand clearly.

The Grapher is designed to be accurate enough for practical visual analysis, exploration, and calculator workflow support.

It is not a symbolic algebra system.

It does not prove exact roots, exact extrema, or exact asymptotes symbolically.

Practical notes

Some functions require an appropriate viewing window.

If a graph looks empty, clipped, or unexpectedly vertical, the range may need adjustment.

For trigonometric functions, confirm the active angle mode. A graph such as:

tan(x)
looks very different in radians than it does in degrees.

For functions involving square roots or logarithms, remember that parts of the x-range may be invalid.

For example:

sqrt(x)
is only valid for:

x ≥ 0
and:

ln(x)
is only valid for:

x > 0
The Grapher distinguishes practical graph interruptions from mathematical interpretation.

For example:

gaps
indicate regions where the graph could not draw valid real points, while:

asymptotes
indicate likely vertical blow-up behavior detected numerically.

For functions such as:

tan(x)
the graph may show visible minimum and maximum values even though the function has no true local extrema.

In that case, the visible min/max describe the sampled values inside the current view, while local min/max remains absent.

Compare a is most readable with moderate sensitivity settings. Large sensitivity values can be useful, but they may make functions with asymptotes, spikes, or dense oscillation look visually busy. Reducing the sensitivity with the − button keeps the comparison subtle.

For best results:

• choose a range appropriate to the function;
• use zoom and recentering to inspect important regions;
• use Fit y when the curve is difficult to see vertically;
• use trace for local readings;
• treat Analysis results as visible-range results;
• use Compare a when the effect of the calculator operand should be explored;
• send y back to CCalc when the traced output should become part of the next calculation.

The Grapher extends CCalc by making functions visible, traceable, analyzable, shareable, and connected to the calculator’s main operand flow.

Constants in CCalc

CCalc includes a constants system for inserting useful values directly into calculation flow.

Constants are available in two main places:

• the compact Constants Docklet, opened from the π key

• the larger Global Constants Repository, accessed from the Settings menu

The Constants Docklet is designed for speed. It gives quick access to common constants, custom constants, and frequently used constants while the user is actively calculating.

The Global Constants Repository is designed for depth. It can contain a much larger constants database without overloading the compact docklet interface.

Together, these two systems give CCalc both fast access and a broader technical reference layer.

Common Constants

Common Constants are CCalc's built-in fast-access constants.

They are available from the Constants Docklet, which is opened from the main calculator keyboard by tapping the π key.

The π key is the entry point to CCalc's built-in and custom constants system. The docklet gives quick access to frequently used mathematical, physical, engineering, chemical, and user-defined constants.

These constants cover several practical groups, including:

• mathematical constants

• logarithmic constants

• fundamental physics constants

• electricity and electronics constants

• chemistry and mole constants

• practical conversion references

• air and fluid constants

• material and particle constants

Examples include:

π Pi

τ Tau (2π)

e Euler's number

c Speed of light

h Planck constant

G Gravitational constant

k_B Boltzmann constant

N_A Avogadro constant

R Gas constant

atm Standard atmosphere

ρ_w Water density

E_steel Steel modulus

m_e Electron mass

The built-in constants are static and curated. They are intended to provide quick access to values commonly used in mathematics, science, engineering, chemistry, and applied calculations.

The Constants Docklet is not meant to show every constant known to the app. It is the working subset: the constants most useful during active calculation.

Custom Constants

Custom Constants are user-defined constants.

They are created from the Constants Docklet using the + custom action.

The user can enter:

• a symbol

• a name

• a numeric value

For example:

Symbol: k_1

Name: Test coefficient

Value: 1.23e-12

or:

Symbol: μx

Name: Micro factor

Value: 0.0000000345

Scientific notation is allowed, so very small or very large constants can be entered compactly.

Examples:

6.626e-34

1.602e-19

2.99792458e8

1.23e-12

The input is parsed as a numeric value and stored as a Decimal.

This makes custom constants practical for scientific and engineering workflows where values may span many orders of magnitude.

Custom constants are stored separately from the built-in constants. This keeps the built-in library stable while allowing the user to personalize the docklet.

Custom constants are intentionally limited to keep the docklet clean and usable.

Current limits:

• maximum custom constants: 25

• maximum name length: 20 characters

• maximum symbol length: 8 characters

If the user enters a name or symbol that is too long, the constant is rejected rather than silently truncated.

This is intentional.

For example, a long name such as:

Very small calibration coefficient should not be silently saved as:

Very small calibrati

Silent truncation could make the constant ambiguous later. Rejection keeps the user in control and protects the layout.

The Global Constants Repository

In addition to the compact Constants Docklet, CCalc maintains a larger constants database available from the Settings menu.

This larger database acts as the app's central constants repository. It is separate from the docklet's fast-access picker and is intended to provide a broader reference layer for constants that may not need to appear in the compact docklet interface.

In practical terms:

Constants Docklet → fast insertion during calculation

Settings Repository → larger constants reference library

The Global Constants Repository exists so CCalc can maintain a much larger set of constants without overloading the docklet.

The Settings repository solves this by giving the app a place for:

• extended physical constants

• engineering reference values

• chemistry and atomic constants

• material properties

• electromagnetic constants

• thermal and fluid constants

• specialized values that are useful but not frequently needed

This lets CCalc support a larger technical knowledge base while keeping the docklet focused.

Using constants in calculations

When the user taps a constant, CCalc inserts the constant's numeric value into the calculator.

At the same time, the ticker preserves the constant's symbolic label.

For example, selecting π inserts its numeric value internally:

3.141592653589793…

but the ticker can show:

π

This is important because symbolic notation is easier to read than long numeric values.

Constants can be used as normal operands.

For example:

π × 2

c ÷ 1000

R × T ÷ V

$G \times m_1 \times m_2 \div r^2$

A constant can start a calculation, replace the current entry, or be inserted after an operator depending on the active expression state.

This allows constants to participate naturally in the same calculation flow as typed numbers.

Using constants in docklets

Constants can also support docklet-based workflows.

A constant inserted into the calculator can become the current operand and then be used by a docklet function.

For example, a user may insert a physical constant, then open an engineering, trigonometry, statistics, geometry, financial, or scientific docklet and apply a function to that value.

Because constants enter the same calculation flow as normal numbers, docklets can treat them as numeric operands while the ticker can preserve the symbolic context.

This is useful when a docklet calculation depends on a known reference value, coefficient, physical constant, or user-defined parameter.

Examples include:

• using π in geometry or trigonometry

• using c in physics or engineering calculations

• using R in gas-law or thermodynamic workflows

• using a custom coefficient in an applied calculation

• using a material constant in an engineering docklet

The constants system is therefore not only a lookup feature. It is a way to feed meaningful values into CCalc's broader tool system.

Using constants in graphing

Constants can also be useful in graphing workflows.

A constant may be inserted into the calculator flow and then used as part of an expression, parameter, or value that supports graphing.

For example:

π

e

c

k_1

A_ref

These values can help the user build expressions that are easier to understand and easier to reuse.

Symbolic constants are especially useful when graphing because they keep expressions readable.

For example:

sin(πx)

is easier to recognize than an expression built only from the decimal expansion of π.

Custom constants can also be useful when the user wants to reuse a coefficient or parameter across multiple related expressions.

This is especially helpful in applied graphing, sensitivity checks, engineering curves, or scientific expressions where constants represent real quantities rather than arbitrary numbers.

Constants and expression flow

Constants are designed to participate in normal CCalc expression flow.

They can be used:

• as the first value in a calculation

• after an operator

• as part of a longer expression chain

• before using a docklet function

• as a reusable value in technical workflows

For example:

π × 2

12 ÷ c

G × 5

R × 300

The inserted constant is numeric internally, but its symbolic form can remain visible in the ticker.

This helps the calculation stay both accurate and readable.

A constant can start a calculation, continue a calculation, or act as the input for another tool.

The goal is that constants should feel like normal calculator values, not like separate reference notes that must be copied manually.

Constants and precision

Constants are stored as numeric values and inserted into calculations as numbers.

The visible ticker may show a symbol, but the calculation uses the stored numeric value.

This allows the user to work with readable expressions without manually typing long decimal expansions.

For example, π may appear symbolically in the ticker, but the calculation uses its numeric value internally.

This matters for constants such as:

• π
• e
• c
• h
• G
• k_B
• N_A
• custom scientific coefficients

Some constants may contain many digits, very small values, or very large values.

Scientific notation helps keep these values practical:

6.626e-34

1.602e-19

2.99792458e8

1.23e-12

The displayed result may be rounded according to the calculator's formatting and decimal-place settings, but the constant's stored value is intended to support accurate calculation flow.

The visible expression is formatted for readability. The constant value is used for calculation.

Managing custom constants

Custom constants are managed from the Common Constants Docklet.

The + custom action opens the custom constant dialog.

The user can create a custom constant by entering:

• a symbol
• a name
• a numeric value

Custom constants are intentionally add-only.

This keeps the docklet simple and avoids turning a fast picker into a database editor.

The row display should still use single-line behavior as a backup:

one row

one symbol

one name

no wrapping

This preserves the docklet's speed and shape.

The docklet also cleans old invalid saved constants on load, so earlier test constants that exceeded the limit can be removed automatically.

Best practices for constants

Use the Constants Docklet for constants needed during active calculation.

Use the Global Constants Repository in Settings when a broader reference library is needed.

Use short, recognizable symbols for custom constants.

Good symbols:

k_1

μx

C_0

A_ref

Avoid long phrases as symbols.

Less useful symbols:

coefficient for test batch

my long value

Use clear names that explain the constant without becoming too long.

The Settings Menu

Overview

The Settings menu is opened from the gear icon in the main CCalc view. It controls app-wide behavior such as angle units, currency base, number formatting, display notation, input notation, constants, and key click sound.

Most settings are stored and reused after the app is closed. When a setting changes, CCalc usually refreshes the display immediately, clears the relevant formatting cache, and briefly shows a confirmation message in the ticker. The Settings menu also includes Reset to defaults, which opens a confirmation alert before resetting stored settings and clearing the tape history.

Opening the Settings Menu

Tap the gear icon in the main calculator view.

The Settings menu contains:

- Reset to defaults
- Key click sound
- Format
- Constants
- Base currency
- Angle unit

Some items open submenus. Other items act immediately when tapped.

Reset to Defaults

Purpose

Reset to defaults restores stored settings to their default state.

Behavior

Selecting Reset to defaults does not reset immediately. It opens a confirmation alert:

Reset to defaults?

The alert explains:

This will reset all settings and clear the tape history. This can’t be undone.

The user can choose:

- Reset
- Cancel

Reset

When Reset is selected, CCalc:

- plays the menu sound
- gives light haptic feedback
- resets stored settings
- clears the format cache
- clears the tape history
- sets the ticker to:

All stored settings reset to defaults.

Cancel

Cancel closes the alert without changing settings.

Angle Unit

Purpose

The Angle unit setting controls how CCalc interprets angle-based calculations in trigonometric, geometric, and graphing-related features that depend on an angle unit.

The available angle units are:

- Degrees
- Radians
- Gradians
- Turns

The setting is stored globally as angleUnit.

Degrees

Degrees divide a full rotation into 360 units.

In degrees:

180° = π radians
360° = 1 full turn
Use degrees for common geometric and everyday angle calculations.

Radians

Radians are the standard mathematical angle unit.

In radians:

π radians = 180°
2π radians = 1 full turn
Radians are usually preferred for mathematical, graphing, and analytical work.

Gradians

Gradians divide a right angle into 100 units.

In gradians:

100 grad = 90°
200 grad = π radians
400 grad = 1 full turn
This is useful for centesimal angle systems.

Turns

Turns measure angle as full rotations.

In turns:

0.25 turn = 90°
0.5 turn = 180°
1 turn = 360°
Turns are useful when thinking in fractions of a full rotation.

Angle unit behavior in docklets and graphing

Angle-aware docklets read the global angle unit and convert internally as needed.

For example, trigonometric functions convert the current operand into radians before evaluating sine, cosine, tangent, and related functions. Inverse trigonometric functions return the angle in the currently selected angle unit.

The Geometry docklet also uses the selected angle unit for functions such as arc length, sector area, chord length, segment area, angle between three points, Law of Cosines, and Law of Sines.

The ticker may show the angle unit so the user can see how the input was interpreted, such as:

sin(90 deg)

sin(1.57079633 rad)

Base Currency

Purpose

The Base currency setting controls the default currency used by the currency conversion system.

The setting is stored globally as:

DefaultBase
The default value is:

USD

Selecting the base currency

The Base currency menu lists available currency codes using live exchange-rate data when available.

If live rates are not currently loaded, CCalc falls back to cached exchange-rate data. If that is unavailable, it tries cached historical currency data. The currently selected base currency is always included in the menu even if the live/cached list is otherwise empty.

When the user selects a base currency, CCalc:

• normalizes the currency code to uppercase
• updates the shared conversion model
• triggers the currency base update
• shows a temporary ticker confirmation, such as:

Base currency set to Euro

The menu displays localized currency names when available. If the localized name is unavailable, it displays the currency code.

Currency refresh behavior

When the base currency changes, CCalc calls the shared conversion model to update the base currency.

That means the selected base becomes the reference currency for future currency conversions.

The conversion model handles the actual exchange-rate fetching and caching behavior outside the Settings menu.

Currency cache behavior

The Base currency menu tries to build its currency list from three sources:

1. Live in-memory exchange rates.
2. Cached exchange rates for the current base.
3. Cached historical exchange rates for the current base.

The relevant cache keys are based on the selected base currency, for example:

cachedExchangeRates_USD
cachedHistoricalRates_USD

This allows the menu to remain useful even when rates were previously loaded but are not currently active in memory.

Format

The Format submenu controls number presentation and input-display behavior.

It contains:

• Thousands separator
• Trailing zeroes
• Rounding Method
• Input Notation
• Display Notation
• Decimal separator
• Decimal places

Decimal Places

Purpose

Decimal places controls how many digits CCalc displays after the decimal separator.

The menu provides values:

0 through 8

Behavior

When a decimal-place value is selected, CCalc:

• plays the menu sound
• gives light haptic feedback
• updates the stored decimal-place value
• clears the number-format cache
• refreshes the current display
• updates the ticker
• briefly shows a confirmation message

Example:

Decimal places set to 4

After a short delay, the temporary ticker confirmation is cleared and the normal ticker is restored.

Important note

Decimal places affect displayed formatting and rounded output in many docklets. They do not necessarily mean that every internal intermediate calculation is limited to that number of digits.

Decimal Separator

Purpose

Decimal separator controls whether CCalc displays decimal numbers with a dot or a comma.

Available choices:

- Comma
- Dot

Dot

Dot mode uses:

.

Example:

1234.56

Comma

Comma mode uses:

,

Example:

1234,56

Behavior

When the decimal separator changes, CCalc:

- updates the stored locale identifier
- updates the active locale
- clears the number-format cache
- refreshes the current display
- temporarily shows a ticker message

Examples:

Decimal separator set to comma

Decimal separator set to dot

The ticker text is also adjusted so decimal marks match the selected separator where possible.

List-entry consequence

Some docklets that accept lists must avoid confusing decimal commas with list commas.

In comma-decimal mode, list-based tools generally use semicolons as list separators.

Example:

1,2;2,3;3,4

In dot-decimal mode, list-based tools generally use commas as list separators.

Example:

1.2,2.3,3.4

Thousands Separator

Purpose

Thousands separator controls how large numbers are grouped visually.

Available choices:

- Space
- Dot
- Comma

Space

Example:

1 000 000

This is the neutral/default grouping style when neither comma nor dot grouping is selected.

Dot

Example:

1.000.000

Comma

Example:

1,000,000

Behavior

When the thousands separator changes, CCalc:

• plays the menu sound
• gives light haptic feedback
• updates the separator flags
• refreshes the ticker
• temporarily shows a confirmation message

Examples:

Thousands separator set to space

Thousands separator set to dot

Thousands separator set to comma

Interaction with decimal separator

The thousands separator and decimal separator should be chosen so the display remains readable.

For example:

1,234.56

and:

1.234,56

use opposite decimal/grouping conventions.

Trailing Zeroes

Purpose

Trailing zeroes controls whether CCalc pads formatted numbers with zeroes up to the selected number of decimal places.

Behavior

The menu item toggles the setting.

If trailing zeroes are disabled, a value may display compactly:

12.5

If trailing zeroes are enabled and decimal places are set to 4, the same value may display as:

12.5000

When toggled, CCalc:

• plays the menu sound
• gives light haptic feedback
• clears the number-format cache
• refreshes the current display
• updates the ticker
• briefly shows one of these messages:

Trailing zeroes enabled

Trailing zeroes disabled

Menu wording

The button text indicates the action available:

Trailing zeroes on

Trailing zeroes off

So if the menu says Trailing zeroes on, tapping it enables trailing zeroes.

Rounding Method

Purpose

Rounding Method controls the rounding rule used when CCalc formats or rounds values.

Available choices:

• Round Nearest
• Round Half to Even
• Round Half Down
• Round Up
• Round Down
• Round Towards Zero
• Truncate

Round Nearest

Rounds to the nearest representable value according to the selected decimal places.

Round Half to Even

Also known as banker's rounding. Halfway values round to the nearest even final digit.

Example idea:

2.5 → 2
3.5 → 4
when rounding to zero decimal places.

Round Half Down

Halfway cases round downward rather than upward.

Round Up

Rounds upward.

Round Down

Rounds downward.

Round Towards Zero

Rounds toward zero.

Examples:

1.9 → 1
−1.9 → −1
when rounding to zero decimal places.

Truncate

Cuts off digits beyond the selected decimal place without normal rounding.

Behavior

When a rounding method is selected, CCalc:

• plays the menu sound
• gives haptic feedback
• stores the selected rounding method
• briefly shows a ticker message

Examples:

Rounding method set to Round Nearest

Rounding method set to Round Half to Even

Rounding method set to Truncate

Display Notation

Purpose

Display Notation controls whether CCalc displays very large or very small numbers using a compact notation style.

The actual menu is named:

Display Notation

It has two parts:

1. Mode
2. Style

Display notation modes

The available modes are:

• Off
• Forced On
• Forced Off
• Auto

Off

Display notation is disabled.

The ticker shows:

Display notation: Off

Forced On

Display notation is manually enabled.

The ticker shows:

Display notation: Forced On

Forced Off

Display notation is manually disabled.

The ticker shows:

Display notation: Forced Off

Auto

Auto mode enables compact notation only when the current value is very large or very small.

The thresholds in the supplied settings code are:

large: |x| ≥ 100000000000000
small: 0 < |x| < 0.000000001
So Auto may activate notation for values at or above 1e14, or below 1e-9 but greater than zero.

The ticker shows:

Display notation: Auto

Display notation styles

The available styles are:

• Scientific
• SI Prefix
• Engineering

Standard notation

Standard notation corresponds to ordinary display formatting with display notation off or forced off.

Example:

1234567

In the current code, there is no separate “Standard” style button. Standard display is achieved by turning Display Notation Off or Forced Off.

Scientific notation

Scientific notation displays numbers as powers of ten.

Example:

1.23×10^{6}

The style is selected with:

Scientific

Engineering notation

Engineering notation is similar to scientific notation, but the exponent is normally a multiple of 3.

Example:

1.23×10^{6}
123×10^{3}
The style is selected with:

Engineering (ENG)

SI prefix notation

SI prefix notation expresses values using metric prefixes where appropriate.

Examples:

1.2 k

3.4 M

5.6 µ

The style is selected with:

SI Prefix (SI)

Behavior

When the notation mode or style changes, CCalc:

• plays the menu sound
• gives haptic feedback
• updates stored notation settings
• refreshes the current display
• briefly shows a ticker message

Examples:

Display style: scientific

Display style: engineering

Display style: si

Input Notation

Purpose

Input Notation controls how CCalc interprets entered operations.

Available choices:

• Infix notation
• Polish notation
• Reverse Polish notation

The menu is named:

Input Notation

Only one special notation mode can be active at a time.

Infix notation

Infix notation is the default calculator style.

Example:

2 + 3

The operator sits between operands.

Selecting this mode disables both Polish and Reverse Polish notation.

Ticker message:

Input notation: Infix (Default)

Polish notation

Polish notation places the operator before the operands.

Example:

+ 2 3

Selecting Polish notation enables PN and disables RPN.

Ticker message:

Input notation: Polish (PN)

Reverse Polish notation

Reverse Polish notation places the operator after the operands.

Example:

2 3 +

Selecting Reverse Polish notation enables RPN and disables PN.

Ticker message:

Input notation: Reverse Polish (RPN)

Key Click Sound

Purpose

Key click sound controls the sound used for key taps and menu feedback.

Available sounds:

- Tick
- Click
- Keyboard
- Metal
- Mute

If sound is muted, the menu shows:

- Unmute

Sound feedback in docklets and menus

CCalc uses sound feedback in several places:

- key taps
- menu selections
- docklet actions
- warnings
- parking and unparking docklets

Changing the key click sound affects the click-style feedback. Other sounds, such as menu, warning, park, or unpark sounds, may use separate sound paths inside the audio manager.

Constants

Purpose

The Settings menu includes a Constants submenu.

Constants are grouped by category. Selecting a constant inserts its value into the calculator flow.

Behavior

When a constant is selected, CCalc:

- plays the menu sound
- gives light haptic feedback
- inserts the constant as a display-number token
- updates the current operand
- updates the display
- updates the ticker
- clears stale ticker label overrides

Starting with a constant

The supplied menu handling explicitly supports starting a calculation with a constant.

If the expression is empty, the selected constant is inserted as the first token instead of being treated as zero.

Replacing a number with a constant

If the last expression token is a number or display-number token, selecting a constant replaces that last token.

If the last token is an operator or left parenthesis, the constant is appended.

If the last token is a right parenthesis, CCalc plays a warning sound instead of inserting the constant.

Settings Persistence

Several settings are saved automatically when changed, including:

- decimal places
- trailing zeroes
- display notation enabled state
- selected locale
- dot/comma separator choices
- audio click sound
- click volume

• menu volume
• mute state

The Settings menu loads stored settings when it appears.

Currency Conversion Result Handling

The Settings area also listens for completed conversion notifications.

When a conversion completes, CCalc:

• receives the converted result
• stores it as the current operand
• quantizes currency results to 4 decimal places
• quantizes non-currency conversion results using the selected decimal-place setting
• updates the display
• stores the result as ANS
• prepares the result for further operations
• resets the expression token stream to the converted value

Currency results are displayed with forced 4-decimal formatting.

Practical Notes

Use Angle unit before working with trigonometry, geometry, or graphing functions that depend on angular interpretation.

Use Base currency before working with the converter’s currency category.

Use Decimal separator and Thousands separator together so numbers remain readable.

Use Trailing zeroes when fixed-width numeric output is preferred.

Use Display Notation Auto when you want ordinary numbers to remain readable but very large or very small values to switch automatically.

Use Input Notation only if you specifically want PN or RPN-style entry. For ordinary calculator use, keep Infix (Default).

Use Reset to defaults carefully as it also clears the tape history.

Advanced Calculator Behavior

Overview

CCalc is designed to behave like a standard calculator for ordinary arithmetic, while also supporting more advanced expression flow from constants, docklets, conversions, graphing, and labeled specialist results.

This section explains how CCalc decides what can continue as part of a calculation, what becomes a final descriptive result, how the ticker labels results, and how formatting, rounding, warnings, and invalid states are handled.

Expression Flow

What expression flow means

Expression flow is the path a value takes through CCalc after it is entered, calculated, selected from a constant, returned from a docklet, or produced by another tool.

A value can be used in different ways:

as the current displayed operand

as part of a longer expression

as the result of a docklet calculation

as ANS

as a labeled result in the ticker

as a display-only result

In ordinary use, CCalc lets the user continue from the current displayed value. For example:

12 + 8 = 20

× 3 = 60

The result 20 becomes the current operand and can continue into the next operation.

Operation Priority in Longer Expressions

Standard expression priority

CCalc evaluates longer expressions using normal arithmetic precedence where supported by the expression system.

This means powers and roots are evaluated before multiplication and division, and multiplication and division are evaluated before addition and subtraction.

Example:

2 + 3 × 4

is interpreted as:

2 + (3 × 4) = 14

not:

(2 + 3) × 4 = 20

Example:

2 + 3² × 4

is interpreted as:

2 + (3² × 4) = 38

not:

(2 + 3)² × 4 = 100

Parentheses

When parentheses are used, they override normal operation priority.

Example:

(2 + 3) × 4 = 20

Parentheses are especially important when expressions combine ordinary arithmetic with constants, docklet returns, roots, powers, or graphing values.

Unary Operations

What unary operations are

A unary operation acts on one value.

Examples include:

√x

sin(x)

ln(x)

n!

1/x

percentage-style transformations

docklet functions that use the current operand

Unary operations usually take the current operand, transform it, and return a new value.

Example:

sin(2 rad)

The operand 2 is used as the input. The result becomes the new current operand.

Unary operations and the ticker

A unary operation usually writes a descriptive label into the ticker.

Example:

sin(2 rad) [APPROX]

The display shows the numeric result, while the ticker explains where it came from.

Unary operations from docklets

Many docklet functions are unary in behavior even when they are mathematically specialized.

Example:

Circle Area

If a prompt field says “blank = operand,” leaving that field empty makes CCalc use the current calculator operand as the input.

If the current operand is:

10

and the user computes circle area with blank radius input, CCalc treats the radius as 10.

Binary Operations

What binary operations are

A binary operation combines two values.

Examples include:

a + b

a − b

a × b

a ÷ b

a ^ b

a mod b

The first value is usually stored as the previous operand. The second value becomes the current operand.

Example:

12 + 8

Here:

previous operand = 12

operation = +

current operand = 8

When = is pressed, CCalc evaluates the operation.

Binary operations and expression continuation

After a binary operation is evaluated, the result becomes available for further calculations.

Example:

12 + 8 = 20

20 ÷ 5 = 4

The user does not need to re-enter 20.

Chained Calculations

Continuing after a result

CCalc supports chaining by allowing the latest numeric result to become the next operand.

Example:

5 × 6 = 30

+ 12 = 42

The result 30 becomes the left side of the next operation.

Chaining with docklet results

Numeric docklet results can also continue into ordinary calculations.

Example:

Circle Area result → 314.15926536

÷ 2

The numeric result can be reused as an operand.

Chaining limits

Not every result is meant to flow into another calculation. Some results are descriptive, structured, or label-only. These are not always safe as numeric operands.

Example:

MID → (2; 5)

A midpoint contains more than one value. CCalc may display it as a label result rather than treating the full point as a single numeric operand.

Starting a Calculation with Constants

Constants as first operands

CCalc supports starting an expression directly with a constant.

Example:

π × 2

The constant is inserted as the first expression token, not treated as if it came after zero.

This is important because:

π × 2

must mean:

3.14159265 × 2

not:

0 × 2

Constants after operators

If the current expression ends with an operator, selecting a constant inserts it as the next operand.

Example:

12 × π

Constants replacing numbers

If the current expression ends with a number and the user selects a constant, CCalc may replace the last number with the constant.

Example:

12

then selecting π changes the active operand to π.

Constants after closed parentheses

If the expression ends with a right parenthesis, CCalc may reject direct constant insertion and play a warning sound, because the expression would become ambiguous without an operator.

Reusing Previous Results

ANS behavior

CCalc can reuse the previous result as ANS in supported areas.

This allows the user to build from the last completed calculation without manually copying the value.

Example:

ANS + 5

Result continuation

In ordinary calculator flow, the last numeric result is already the current operand.

Example:

9 × 9 = 81

√x

The square-root operation applies to 81.

Conversion results

When a unit or currency conversion completes, the converted result becomes the current operand. It can then participate in further operations.

For currency conversions, CCalc may preserve a fixed number of decimal places suitable for currency display.

Docklet Results as Operands

Docklet result types

Docklets can return different kinds of results:

numeric scalar results

approximate numeric results

label-only results

structured results

error or warning states

The result type determines whether the returned value can safely continue in normal calculator flow.

Scalar docklet results

A scalar result is a single numeric value.

Examples:

sin(2 rad)

Circle Area

det(A)

mean

gcd

These results can usually become operands.

Example:

det(A) → 5
× 10 = 50

Structured docklet results

Some docklet functions return structures rather than a single number.

Examples:

matrix inverse

cross product

linear regression equation

midpoint

factorization list

confidence interval

These results are useful to read, but they are not always meaningful as a single operand.

In these cases, CCalc may display the result in the ticker or operand label area while preventing it from flowing as an ordinary scalar.

Approximate Results

What approximate means

An approximate result is a numeric value that may involve floating-point approximation, iterative methods, irrational numbers, statistical approximations, or functions that cannot be represented exactly in decimal form.

Examples:

sin(2 rad)

ellipse perimeter

normal CDF

eigenvalues

condition number

Approximate results are marked with:

[APPROX]

Why approximate results are marked

The tag tells the user that the displayed number is a computed approximation, not an exact symbolic value.

Example:

sin(2 rad) [APPROX]

The value is still usable as an operand, but the user should understand that it has been rounded for display.

Label-Only Results

What label-only means

A label-only result is a result that is primarily descriptive rather than a clean scalar number.

Examples:

factorization: $2^3 \times 3 \times 5$

linear regression: $y = 2x + 1$

midpoint: (4; 7)

matrix inverse

list of divisors

confidence interval

Label-only results are marked with:

[LABEL]

Why label-only results exist

Some mathematical results are not single numbers.

A matrix, vector, point, list, equation, or interval cannot always be represented by one operand without losing meaning.

CCalc therefore uses label-style output to keep the result readable.

Label-only and continued calculations

A label-only result may display a number internally for technical reasons, but the main result should be read from the label.

Do not assume every label-only result is suitable for immediate arithmetic continuation.

Numeric Results

What numeric means

A numeric result is a scalar value intended to be used as a number.

Numeric results are marked with:

[NUM]

Examples:

Circle Area

Rectangle Area

gcd

Euler φ(n)

rank(A)

Numeric results and operands

A numeric result normally becomes:

currentOperand

display

ANS

available input for the next operation

Example:

φ(12) → 4 [NUM]

× 3 = 12

Ticker Labels

Purpose of the ticker

The ticker explains what CCalc just did.

The display shows the current value. The ticker shows context.

Example display:

0.90929743

Example ticker:

sin(2 rad) [APPROX]

The ticker tells the user that the displayed value came from sin(2 rad).

Ticker labels after ordinary arithmetic

For ordinary binary calculations, the ticker may show the full expression and result.

Example:

12 + 8 = 20

Ticker labels after docklet operations

Docklet ticker labels usually show the function, inputs, unit context, and result tag.

Example:

A□[L²](r:10) → 314.15926536 [NUM]

Temporary ticker confirmations

Settings changes and menu actions may temporarily replace the ticker with a confirmation.

Example:

Decimal places set to 4

After a short delay, the normal ticker may return.

Display Labels

Purpose of the display

The display is the main visible value area.

For ordinary numeric results, it shows the current operand.

Example:

42

For some operations, it may show a formatted result with a unit suffix.

Example:

90 deg

For invalid states, it may show an error message.

Example:

Division by 0

Display versus ticker

The display is for the active value.

The ticker is for explanation.

Example:

Display: 314.15926536

Ticker: A□[L²](r:10) → 314.15926536 [NUM]

Operand Labels

Purpose of operand labels

Operand labels allow CCalc to show a meaningful result that is not just a plain number.

They are useful for structured outputs such as:

vectors

matrices

factorizations

intervals

coordinate points

regression equations

Numeric operand with label context

Some functions may store a numeric value internally while showing a richer label externally.

Example:

MID → (4; 7)

The x-coordinate may be available internally, but the user-facing result is the point label.

Clearing operand labels

When a new ordinary numeric result is produced, stale operand labels should be cleared so that the display does not keep showing an old structured result.

Warning States

What warning states mean

A warning state means CCalc prevented an operation that would be invalid, ambiguous, undefined, or unsafe.

Typical warning triggers include:

division by zero

invalid input

out-of-range input

missing required prompt fields

singular matrix

impossible triangle

unsupported complex result

overflow

Warning states often produce:

warning sound

ticker message

display message

disabled Compute button

Warning sound

CCalc uses a warning sound when the user attempts something invalid, such as tapping outside a firm prompt dialog or trying to compute with incomplete input.

Invalid Input States

Missing input

Prompt-based docklets usually require all mandatory fields to be filled before Compute is enabled.

Some fields intentionally allow blank input when the placeholder says:

blank = operand

In that case, leaving the field blank tells CCalc to use the current calculator operand.

Invalid numeric input

Invalid numeric input may include:

empty mandatory fields

non-numeric text

malformed decimal values

invalid integer fields

wrong list separators

negative values where only positive values are allowed

Locale-sensitive input

In dot-decimal mode, list input may use commas as separators:

1.2,2.3,3.4

In comma-decimal mode, list input may use semicolons as separators:

1,2;2,3;3,4

Using the wrong separator can cause an invalid input state.

Division by Zero

Ordinary division by zero

Division by zero is rejected.

Example:

12 ÷ 0

produces a warning state instead of a valid numeric result.

Docklet division by zero

Docklets also check for division by zero.

Examples:

csc(x) when sin(x) = 0
sec(x) when cos(x) = 0
slope when x2 − x1 = 0
modular inverse when no inverse exists

The ticker or prompt may show:

Division by 0

or a more specific message.

Out-of-Range Results

Domain restrictions

Some functions are only defined over certain input ranges.

Examples:

arcsin(x): −1 ≤ x ≤ 1
arccos(x): −1 ≤ x ≤ 1
arcosh(x): x ≥ 1
artanh(x): −1 < x < 1
square root: x ≥ 0
triangle area from sides: triangle inequality must hold

Out-of-range messages

When an input is outside the allowed domain, CCalc rejects the calculation and shows a warning.

Examples:

Out of range (−1…1)

No real solution

Invalid triangle

Singular matrix

Complex eigenvalues not supported

No real solution

Some functions reject results that would require unsupported complex numbers.

Example:

√−1

or an eigenvalue calculation that produces complex eigenvalues in a docklet that only supports real eigenvalues.

Formatting and Rounding Behavior

Decimal places

The selected decimal-place setting controls how many digits CCalc shows after the decimal separator.

Many docklets round their committed numeric results according to this setting.

Example:

decimal places = 4
π → 3.1416

Rounding once at commit

Docklets generally try to round once when committing a result back to the calculator. This avoids repeated rounding at several stages of the same calculation.

The usual flow is:

compute full result

round according to decimalPlaces

normalize −0 to 0

commit result

update display

update ticker

Negative zero normalization

Very small negative values can round to:

-0

CCalc normalizes this to:

0

This prevents confusing display output.

Decimal separator

The decimal separator setting affects displayed values and some ticker text.

Dot mode:

3.1416

Comma mode:

3,1416

Thousands separator

Large values may be grouped according to the selected thousands separator.

Examples:

1 000 000

1.000.000

1,000,000

Trailing zeroes

If trailing zeroes are enabled, CCalc may pad the display to the selected decimal-place count.

Example with 4 decimal places:

12.5 → 12.5000

If trailing zeroes are disabled:

12.5

Display notation

For very large or very small values, CCalc may use display notation depending on the selected notation mode.

Possible styles include:

Scientific

Engineering

SI Prefix

Display notation changes how a value is shown, not the mathematical value itself.

Practical Examples

Ordinary expression

2 + 3 × 4 = 14

The multiplication is evaluated before the addition.

Parenthesized expression

(2 + 3) × 4 = 20

Parentheses override normal priority.

Docklet scalar result reused

Circle Area with r = 10 → 314.15926536

÷ 2 → 157.07963268

The circle area result is numeric and can continue.

Approximate trig result

sin(2 rad) → 0.90929743 [APPROX]

The result is numeric but approximate.

Label-only factorization

factor(360) → $2^3 \times 3^2 \times 5$ [LABEL]

The result is descriptive. It should be read as a factorization, not as a plain scalar expression.

Invalid geometry input

Triangle sides: 1, 2, 10

This fails because the triangle inequality is not satisfied.

Possible message:

Invalid triangle

Division by zero in slope

Slope from (1,2) to (1,5)

This fails because:

x2 − x1 = 0

Possible message:

Division by 0

Summary

Advanced calculator behavior in CCalc is based on one central idea:

A result may be numeric, approximate, or descriptive.

Numeric scalar results can usually continue as operands. Approximate results can also continue, but are marked clearly. Label-only results preserve important mathematical meaning when the result is not a single clean number.

The display shows the active value. The ticker explains the operation. Operand labels preserve structured results. Warning states protect the user from invalid, ambiguous, or unsupported calculations.