# The Definitive Guide to the Webull Script Editor: Building Custom Technical Indicators

## Section 1: Introduction to the Webull Scripting Environment

The Webull Script Editor is a powerful tool designed to empower traders and analysts to create, customize, and deploy their own technical indicators directly on the Webull platform. This capability transforms the charting experience from a passive consumption of pre-built tools to an active process of analytical creation. By enabling users to codify their unique trading logic and market perspectives, the editor facilitates a more nuanced and personalized approach to technical analysis. This guide serves as a comprehensive manual for mastering the Webull Script language, providing a detailed reference for its functions and a practical roadmap for building sophisticated indicators from the ground up.

### 1.1 Core Philosophy: A Pine Script Heritage

A foundational understanding of the Webull Script language begins with recognizing its origins. The language and its editor are explicitly inspired by Pine Script, the proprietary scripting language developed by TradingView for creating custom trading tools. This is not a superficial similarity; a direct comparison of the function library provided within the Webull editor and the established Pine Script v4/v5 documentation reveals a near one-to-one correspondence in function names, syntax, and purpose.2

This connection is the single most critical piece of information for any developer approaching the Webull Script Editor. The immediate implication is that the vast ecosystem of educational resources, open-source scripts, and community forums dedicated to Pine Script is directly applicable to Webull Script. The lack of extensive, official Webull-specific documentation is therefore not a significant impediment. Instead, developers can leverage the wealth of knowledge available on platforms like TradingView's own script library, developer forums, and educational resources like the Pine Script User Manual to solve problems and learn advanced techniques.4

For example, functions such as ema() for calculating an Exponential Moving Average, rsi() for the Relative Strength Index, and plot() for rendering data on a chart behave in a manner consistent with their Pine Script counterparts.3 This shared heritage means that a strategy or indicator concept found in a Pine Script tutorial can almost certainly be adapted for use in the Webull editor with minimal, if any, modification. This guide will proceed on the well-founded principle that Webull Script is a close dialect or direct implementation of Pine Script, allowing for a robust and detailed exploration of its capabilities.

### 1.2 The Script Execution Model: Thinking in Series

To write effective Webull scripts, one must understand its unique execution model, which differs fundamentally from general-purpose programming languages like Python or JavaScript.8 A Webull script does not run once to produce a final output. Instead, the script's code is executed sequentially on every historical bar of the chart, from the earliest available data point to the most recent, and it continues to execute on each incoming real-time price tick.9

The core data structure that facilitates this model is the **series**. A series is a continuous list of values where one value exists for each bar on the chart, stretching back in time from the current bar.5 Built-in variables such as

open, high, low, and close are not single floating-point numbers; they are series variables. When a script references close, it is accessing the closing price of the specific bar the script is currently executing on. To access historical values, a history-referencing operator, ``, is used. For instance, close refers to the closing price of the *previous* bar, and close refers to the closing price ten bars ago.

This bar-by-bar execution has a critical implication for variable declaration. By default, a variable is re-initialized on every bar. Consider the following code:

my\_var = 0

my\_var := my\_var + 1

plot(my\_var)

Without special handling, this script would plot the value 1 on every single bar, because my\_var is reset to 0 at the start of each bar's execution. To maintain a variable's state across bars—a necessity for cumulative calculations like moving averages or custom counters—specific language constructs must be used. This concept of state management is central to building any meaningful indicator.

Furthermore, the language provides built-in variables, often referred to as "bar states," that allow a script to be aware of the context of the bar it is currently executing on. A script can determine if it is running on the first bar of the dataset (barstate.isfirst), the most recent real-time bar (barstate.islast), or a historical bar (barstate.ishistory). This awareness is crucial for performing calculations that should only occur once or for drawing objects that should only appear on the last bar.

### 1.3 The Anatomy of a Webull Script

Every Webull script, whether a simple indicator or a complex strategy, follows a consistent structure. This structure can be broken down into four primary components: declaration, inputs, calculations, and outputs.3

1. **Declaration:** The very first line of a script is a declaration statement that defines its fundamental properties. For creating a custom indicator, this is typically the indicator() function. This function sets the name of the script as it will appear on the chart and configures key properties, such as whether it should be drawn directly on the main price chart (overlay=true) or in a separate pane below it (overlay=false).
2. **Inputs:** To make an indicator flexible and reusable, parameters should be defined as user-configurable inputs. This is achieved using input functions, which allow the user to change settings like the length of a moving average or the color of a plotted line without editing the source code.12 For example,  
   len = input(14, title="MA Length") creates a user input for a length parameter, with a default value of 14.
3. **Calculations:** This is the core of the script where the trading logic resides. In this section, the script calls various built-in functions, references price data (open, close, etc.), and performs mathematical operations to compute the indicator's values. The results of these calculations are stored in variables for later use.
4. **Outputs:** The final step is to visualize the results of the calculations on the chart. This is accomplished using a family of plotting and drawing functions. The most common is the plot() function, which draws a line on the chart representing a series of values. Other functions can be used to change the background color (bgcolor()), plot shapes (plotshape()), or draw text and labels (drawlabel()).13

A simple script to calculate and plot a 50-period Simple Moving Average (SMA) illustrates this structure:

Pine Script

// 1. Declaration  
indicator("Simple Moving Average", overlay=true)  
  
// 2. Inputs  
sma\_length = input(50, title="SMA Length")  
  
// 3. Calculations  
sma\_value = sma(close, sma\_length)  
  
// 4. Outputs  
plot(sma\_value, title="SMA", color=#FF5733)

This complete, functional script demonstrates the essential anatomy that forms the basis for all custom indicator development in the Webull Script Editor.

## Section 2: The Complete Function Reference

This section provides an exhaustive reference for the functions available within the Webull Script Editor. The functions are categorized by their primary purpose to facilitate easy navigation. Each function entry includes its syntax, a detailed description of its parameters and return values, an explanation of its use in technical analysis, and a practical code example.

### 2.1 Mathematical & Statistical Functions

These are the fundamental building blocks for performing numerical and statistical operations, forming the basis of most custom calculations.

**abs**

* **Syntax:** abs(number)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| number | series[integer/float] | Yes | The number or series from which to calculate the absolute value. |

* **Returns:** series[integer/float] - The absolute value of the input.
* **Description:** The abs function returns the non-negative value of a number. It is frequently used in technical analysis to measure the magnitude of price change or the difference between two indicators, regardless of direction. For example, it can be used to calculate the absolute deviation from a moving average.
* **Practical Code Example:**

Pine Script

// Calculate the absolute difference between the close and a 20-period SMA  
sma20 = sma(close, 20)  
price\_deviation = abs(close - sma20)  
plot(price\_deviation, title="Absolute Deviation from SMA")

**acos**

* **Syntax:** acos(x)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| x | series[float] | Yes | A value between -1 and 1. |

* **Returns:** series[float] - The arccosine of x, in radians.
* **Description:** The acos function calculates the inverse cosine.15 It is used in advanced cyclical analysis and geometric calculations, often for creating indicators based on trigonometric principles or market cycles modeled as waveforms.
* **Practical Code Example:**

Pine Script

// Normalize close price between -1 and 1 over the last 50 bars  
highest\_close = highest(close, 50)  
lowest\_close = lowest(close, 50)  
normalized\_close = 2 \* ((close - lowest\_close) / (highest\_close - lowest\_close)) - 1  
angle = acos(normalized\_close)  
plot(angle, title="Price Angle in Radians")

**asin**

* **Syntax:** asin(x)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| x | series[float] | Yes | A value between -1 and 1. |

* **Returns:** series[float] - The arcsine of x, in radians.
* **Description:** The asin function calculates the inverse sine.16 Similar to  
  acos, it is a specialized function for creating cyclical indicators or performing complex geometric calculations on price data.
* **Practical Code Example:**

Pine Script

// Normalize momentum over 14 periods  
mom14 = mom(close, 14)  
highest\_mom = highest(mom14, 50)  
lowest\_mom = lowest(mom14, 50)  
normalized\_mom = 2 \* ((mom14 - lowest\_mom) / (highest\_mom - lowest\_mom)) - 1  
cycle\_angle = asin(normalized\_mom)  
plot(cycle\_angle, title="Momentum Cycle Angle")

**atan**

* **Syntax:** atan(y, x)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| y | series[float] | Yes | The y-coordinate. |
| x | series[float] | Yes | The x-coordinate. |

* **Returns:** series[float] - The arctangent of y/x, in radians.
* **Description:** The atan function calculates the inverse tangent, often used to determine the angle of a trend line or the slope of an indicator.17 It can convert a rate of change (slope) into an angle, which can be useful for normalizing trend strength.
* **Practical Code Example:**

Pine Script

// Calculate the angle of a 20-period linear regression line  
linreg20 = linreg(close, 20, 0)  
slope = linreg20 - linreg20  
angle\_rad = atan(slope, 1)  
angle\_deg = angle\_rad \* 180 / 3.14159  
plot(angle\_deg, title="Trend Angle (Degrees)")

**avg**

* **Syntax:** avg(series1, series2,...)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| seriesN | series[float] | Yes | One or more series of data to average. |

* **Returns:** series[float] - The arithmetic average of the input series for each bar.
* **Description:** The avg function calculates the simple arithmetic mean of two or more data series on a bar-by-bar basis.18 It is useful for creating composite indicators, such as an average of multiple moving averages or a composite price from  
  open, high, low, and close.
* **Practical Code Example:**

Pine Script

// Create a composite moving average from a 10, 20, and 30-period EMA  
ema10 = ema(close, 10)  
ema20 = ema(close, 20)  
ema30 = ema(close, 30)  
composite\_ma = avg(ema10, ema20, ema30)  
plot(composite\_ma, title="Composite EMA")

**ceil**

* **Syntax:** ceil(number)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| number | series[float] | Yes | The number to round up. |

* **Returns:** series[integer/float] - The smallest integer greater than or equal to the input number.
* **Description:** The ceil (ceiling) function rounds a number up to the next highest integer.19 It can be used to discretize continuous data or to round calculation results to whole numbers, for example, when dealing with tick sizes or contract numbers.
* **Practical Code Example:**

Pine Script

// Round the ATR value up to the nearest whole number  
atr\_value = atr(14)  
rounded\_atr = ceil(atr\_value)  
plot(rounded\_atr, title="Ceiling of ATR")

**cos**

* **Syntax:** cos(angle)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| angle | series[float] | Yes | An angle in radians. |

* **Returns:** series[float] - The cosine of the angle.
* **Description:** The cos function is a trigonometric function used in advanced cyclical indicators, such as the MESA Sine Wave indicator, to model market cycles and generate predictive signals based on waveform analysis.
* **Practical Code Example:**

Pine Script

// Create a simple sine wave based on the bar index  
bar\_cycle = (bar\_index % 30) \* 2 \* 3.14159 / 30  
cosine\_wave = cos(bar\_cycle)  
plot(cosine\_wave, title="Cosine Wave Cycle")

**dev**

* **Syntax:** dev(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to analyze. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The mean absolute deviation of the series over the specified length.
* **Description:** The dev function calculates the mean absolute deviation, which is a measure of statistical dispersion. It represents the average distance between each data point in the series and the mean of the series. It can be used as an alternative to standard deviation for measuring volatility.
* **Practical Code Example:**

Pine Script

// Calculate and plot the 20-period mean absolute deviation of closing prices  
price\_deviation = dev(close, 20)  
plot(price\_deviation, title="Mean Absolute Deviation")

**exp**

* **Syntax:** exp(power)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| power | series[float] | Yes | The exponent to which Euler's number e is raised. |

* **Returns:** series[float] - The value of $e^{power}$.
* **Description:** The exp function calculates the exponential function, $e^x$, where e is Euler's number (approximately 2.71828).20 It is a core component in many statistical and financial formulas, including log-normal distributions and calculations for continuously compounded returns.
* **Practical Code Example:**

Pine Script

// Calculate continuously compounded return over one period  
log\_return = log(close / close)  
exp\_return = exp(log\_return) // This will be close to (close / close)  
plot(log\_return, title="Log Return")

**floor**

* **Syntax:** floor(number)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| number | series[float] | Yes | The number to round down. |

* **Returns:** series[integer/float] - The largest integer less than or equal to the input number.
* **Description:** The floor function rounds a number down to the nearest integer.11 It is the counterpart to  
  ceil and is used for similar purposes, such as discretizing data or ensuring calculations result in whole numbers.
* **Practical Code Example:**

Pine Script

// Round the RSI value down to the nearest integer  
rsi\_value = rsi(close, 14)  
floored\_rsi = floor(rsi\_value)  
plot(floored\_rsi, title="Floored RSI")

**log**

* **Syntax:** log(number)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| number | series[float] | Yes | The number for which to calculate the natural logarithm. Must be positive. |

* **Returns:** series[float] - The natural logarithm (base e) of the number.
* **Description:** The log function calculates the natural logarithm. In finance, it is essential for calculating log returns, which have desirable statistical properties (e.g., normality) and are used in volatility calculations and quantitative models.
* **Practical Code Example:**

Pine Script

// Calculate and plot the single-period log return of the close price  
log\_return = log(close / close)  
plot(log\_return, title="Logarithmic Return")

**log10**

* **Syntax:** log10(number)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| number | series[float] | Yes | The number for which to calculate the base-10 logarithm. Must be positive. |

* **Returns:** series[float] - The common logarithm (base 10) of the number.
* **Description:** The log10 function calculates the base-10 logarithm. It is often used to analyze data that spans several orders of magnitude, making large-scale changes more visually comprehensible on a chart.
* **Practical Code Example:**

Pine Script

// Plot the base-10 logarithm of the trading volume  
log\_volume = log10(volume)  
plot(log\_volume, title="Log10 of Volume")

**max**

* **Syntax:** max(series1, series2,...)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| seriesN | series[float] | Yes | Two or more series of data to compare. |

* **Returns:** series[float] - The largest value among the input series for each bar.
* **Description:** The max function returns the maximum value from a set of two or more series on a bar-by-bar basis. It is useful for creating indicator boundaries or finding the highest point among several different calculated lines.
* **Practical Code Example:**

Pine Script

// Find the highest value among three different moving averages  
ema10 = ema(close, 10)  
ema20 = ema(close, 20)  
ema30 = ema(close, 30)  
highest\_ma = max(ema10, ema20, ema30)  
plot(highest\_ma, title="Highest EMA")

**min**

* **Syntax:** min(series1, series2,...)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| seriesN | series[float] | Yes | Two or more series of data to compare. |

* **Returns:** series[float] - The smallest value among the input series for each bar.
* **Description:** The min function returns the minimum value from a set of two or more series on a bar-by-bar basis. It is the counterpart to max and is used for establishing lower boundaries or finding the lowest point among several indicators.
* **Practical Code Example:**

Pine Script

// Find the lowest value among three different moving averages  
ema10 = ema(close, 10)  
ema20 = ema(close, 20)  
ema30 = ema(close, 30)  
lowest\_ma = min(ema10, ema20, ema30)  
plot(lowest\_ma, title="Lowest EMA")

**pow**

* **Syntax:** pow(base, exponent)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| base | series[float] | Yes | The base number. |
| exponent | series[float] | Yes | The exponent. |

* **Returns:** series[float] - The base raised to the power of the exponent.
* **Description:** The pow function calculates exponentiation. It is used in a wide range of mathematical formulas, such as geometric means or polynomial regressions, allowing for non-linear transformations of price data.
* **Practical Code Example:**

Pine Script

// Square the value of the RSI  
rsi\_value = rsi(close, 14)  
rsi\_squared = pow(rsi\_value, 2)  
plot(rsi\_squared, title="RSI Squared")

**round**

* **Syntax:** round(number)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| number | series[float] | Yes | The number to round. |

* **Returns:** series[integer/float] - The number rounded to the nearest integer.
* **Description:** The round function rounds a number to the nearest integer, with halves being rounded up. It is commonly used to simplify indicator values for display or for comparison against discrete levels.
* **Practical Code Example:**

Pine Script

// Round the ATR to the nearest whole number  
atr\_value = atr(14)  
rounded\_atr = round(atr\_value)  
plot(rounded\_atr, title="Rounded ATR")

**sign**

* **Syntax:** sign(number)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| number | series[float] | Yes | The number to evaluate. |

* **Returns:** series[integer] - Returns 1 if the number is positive, -1 if it is negative, and 0 if it is zero.
* **Description:** The sign function extracts the sign of a number. It is extremely useful for determining the direction of a change or an indicator without regard to its magnitude. For example, it can be used to check if momentum is positive or negative.
* **Practical Code Example:**

Pine Script

// Determine the direction of the 14-period momentum  
mom14 = mom(close, 14)  
momentum\_direction = sign(mom14)  
plot(momentum\_direction, title="Momentum Direction (-1, 0, 1)")

**sin**

* **Syntax:** sin(angle)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| angle | series[float] | Yes | An angle in radians. |

* **Returns:** series[float] - The sine of the angle.
* **Description:** The sin function is a trigonometric function used to model cyclical patterns in market data. It is a key component of advanced cycle analysis indicators that attempt to identify periodic behavior in price movements.
* **Practical Code Example:**

Pine Script

// Create a simple sine wave based on the bar index  
bar\_cycle = (bar\_index % 30) \* 2 \* 3.14159 / 30  
sine\_wave = sin(bar\_cycle)  
plot(sine\_wave, title="Sine Wave Cycle")

**sqrt**

* **Syntax:** sqrt(number)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| number | series[float] | Yes | A non-negative number. |

* **Returns:** series[float] - The square root of the number.
* **Description:** The sqrt function calculates the square root of a number. Its most common application in finance is in the calculation of standard deviation, which is the square root of variance and a fundamental measure of volatility.21
* **Practical Code Example:**

Pine Script

// Manually calculate standard deviation as the square root of variance  
price\_variance = variance(close, 20)  
price\_stdev = sqrt(price\_variance)  
plot(price\_stdev, title="Manual Standard Deviation")

**stdev**

* **Syntax:** stdev(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to analyze. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The standard deviation of the series over the specified length.
* **Description:** The stdev function calculates the standard deviation, a key measure of price volatility or dispersion around an average.22 It is the core component of many indicators, most notably Bollinger Bands, where bands are placed a certain number of standard deviations away from a central moving average.
* **Practical Code Example:**

Pine Script

// Calculate and plot the 20-period standard deviation of closing prices  
stdev20 = stdev(close, 20)  
plot(stdev20, title="20-Period Standard Deviation")

**sum**

* **Syntax:** sum(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to sum. |
| length | integer | Yes | The number of bars to sum over. |

* **Returns:** series[float] - The moving sum of the series over the specified length.
* **Description:** The sum function calculates a moving sum of a data series over a specified lookback period. It is a fundamental building block for many other indicators. For example, a Simple Moving Average is calculated by taking the sum of prices over a period and dividing by that period.
* **Practical Code Example:**

Pine Script

// Calculate the total volume over the last 20 bars  
volume\_sum = sum(volume, 20)  
plot(volume\_sum, title="20-Bar Volume Sum")

**tan**

* **Syntax:** tan(angle)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| angle | series[float] | Yes | An angle in radians. |

* **Returns:** series[float] - The tangent of the angle.
* **Description:** The tan function is a trigonometric function used in specialized geometric or cyclical analysis. It can be used to relate an angle of a trend to its slope.
* **Practical Code Example:**

Pine Script

// Example of using tan in a cyclical context  
bar\_cycle = (bar\_index % 90) \* 2 \* 3.14159 / 90  
tan\_wave = tan(bar\_cycle)  
plot(tan\_wave, title="Tangent Wave")

**variance**

* **Syntax:** variance(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to analyze. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The variance of the series over the specified length.
* **Description:** The variance function calculates the statistical variance, which measures how far a set of numbers is spread out from their average value. It is the square of the standard deviation and is a fundamental measure of volatility.
* **Practical Code Example:**

Pine Script

// Calculate and plot the 20-period variance of closing prices  
variance20 = variance(close, 20)  
plot(variance20, title="20-Period Variance")

### 2.2 Data Series & Conditional Functions

These functions are used for accessing historical data, performing logical comparisons, and controlling script flow based on specific conditions.

**barslast**

* **Syntax:** barslast(condition)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| condition | series[boolean] | Yes | The condition to search for in previous bars. |

* **Returns:** series[integer] - The number of bars that have passed since the condition was last true. Returns na if the condition has never been true.
* **Description:** The barslast function is a powerful tool for temporal analysis. It counts the number of bars since a specific event occurred. This is useful for creating time-based rules, such as "exit a trade if 10 bars have passed since entry" or "only enter if a signal occurred within the last 5 bars."
* **Practical Code Example:**

Pine Script

// Find the number of bars since the price crossed above the 200-day SMA  
sma200 = sma(close, 200)  
cross\_up\_condition = cross\_up(close, sma200)  
bars\_since\_cross = barslast(cross\_up\_condition)  
plot(bars\_since\_cross, title="Bars Since SMA200 Cross Up")

**change**

* **Syntax:** change(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to measure. |
| length | integer | Optional | The lookback period. Defaults to 1. |

* **Returns:** series[float] - The difference between the current value of the source and its value length bars ago (source - source[length]).
* **Description:** The change function calculates the simple difference in a series over a specified period. It is the basis for momentum indicators, which measure the rate of price change.
* **Practical Code Example:**

Pine Script

// Calculate the price change over the last 10 bars  
price\_change\_10 = change(close, 10)  
plot(price\_change\_10, title="10-Bar Price Change")

**count**

* **Syntax:** count(condition, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| condition | series[boolean] | Yes | The condition to count. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[integer] - The number of times the condition was true over the last length bars.
* **Description:** The count function counts the occurrences of a true condition within a specified lookback window.23 It is useful for building indicators based on the frequency of events, such as "how many of the last 20 bars were up-days?"
* **Practical Code Example:**

Pine Script

// Count the number of bullish candles (close > open) in the last 20 bars  
is\_bullish = close > open  
bullish\_bar\_count = count(is\_bullish, 20)  
plot(bullish\_bar\_count, title="Number of Bullish Bars in Last 20")

**cross**

* **Syntax:** cross(series1, series2)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| series1 | series[float] | Yes | The first data series. |
| series2 | series[float] | Yes | The second data series. |

* **Returns:** series[boolean] - Returns true on the bar where the two series cross each other.
* **Description:** The cross function detects when two series intersect, regardless of direction.24 It is a fundamental function for generating trading signals, most famously in moving average crossover systems. It is a more general version of  
  cross\_up and cross\_down.
* **Practical Code Example:**

Pine Script

// Detect when the 9-period EMA crosses the 21-period EMA  
ema9 = ema(close, 9)  
ema21 = ema(close, 21)  
crossover\_event = cross(ema9, ema21)  
plotshape(crossover\_event, style=shape.xcross, color=#FFC300)

**cross\_down**

* **Syntax:** cross\_down(series1, series2)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| series1 | series[float] | Yes | The first data series (the one crossing down). |
| series2 | series[float] | Yes | The second data series (the one being crossed). |

* **Returns:** series[boolean] - Returns true on the bar where series1 crosses below series2.
* **Description:** The cross\_down function specifically detects a bearish crossover, where one series moves from being above another series to being below it.25 This is commonly used to generate sell or short signals, such as when a fast moving average crosses below a slow moving average (a "death cross").
* **Practical Code Example:**

Pine Script

// Generate a sell signal when the 50-period SMA crosses below the 200-period SMA  
sma50 = sma(close, 50)  
sma200 = sma(close, 200)  
death\_cross = cross\_down(sma50, sma200)  
plotshape(death\_cross, title="Sell Signal", style=shape.triangledown, location=location.abovebar, color=#FF0000)

**cross\_up**

* **Syntax:** cross\_up(series1, series2)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| series1 | series[float] | Yes | The first data series (the one crossing up). |
| series2 | series[float] | Yes | The second data series (the one being crossed). |

* **Returns:** series[boolean] - Returns true on the bar where series1 crosses above series2.
* **Description:** The cross\_up function detects a bullish crossover, where one series moves from being below another to being above it.26 This is a primary method for generating buy signals, such as when a fast moving average crosses above a slow moving average (a "golden cross").
* **Practical Code Example:**

Pine Script

// Generate a buy signal when the 50-period SMA crosses above the 200-period SMA  
sma50 = sma(close, 50)  
sma200 = sma(close, 200)  
golden\_cross = cross\_up(sma50, sma200)  
plotshape(golden\_cross, title="Buy Signal", style=shape.triangleup, location=location.belowbar, color=#00FF00)

**every**

* **Syntax:** every(condition, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| condition | series[boolean] | Yes | The condition to check. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[boolean] - Returns true if the condition has been true for every bar over the last length bars.
* **Description:** The every function checks for the persistence of a condition over a specified period.9 It is useful for confirming the strength of a trend or state, for example, "is the price above the 50-period moving average for 10 consecutive bars?"
* **Practical Code Example:**

Pine Script

// Check if the close has been above the 50-period SMA for 5 consecutive bars  
sma50 = sma(close, 50)  
is\_above\_sma = close > sma50  
strong\_uptrend = every(is\_above\_sma, 5)  
bgcolor(strong\_uptrend? #00FF0020 : na)

**falling**

* **Syntax:** falling(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to check. |
| length | integer | Yes | The number of consecutive bars the source must be falling. |

* **Returns:** series[boolean] - Returns true if the source has been decreasing for the last length bars.
* **Description:** The falling function checks if a series has been consistently decreasing over a specified period. It is a simple way to identify short-term downward momentum in price or in an indicator like the RSI.
* **Practical Code Example:**

Pine Script

// Detect when the close price has fallen for 3 consecutive bars  
is\_falling\_3\_bars = falling(close, 3)  
plotshape(is\_falling\_3\_bars, title="3 Falling Bars", style=shape.arrowdown, location=location.abovebar, color=#FF0000)

**hhv**

* **Syntax:** hhv(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series from which to find the highest value. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The highest value of the source over the last length bars.
* **Description:** The hhv (Highest High Value) function finds the maximum value a series has reached within a specified lookback window.21 It is a core component of many indicators, including the Stochastic Oscillator and Donchian Channels, and is used to identify recent price peaks or resistance levels.
* **Practical Code Example:**

Pine Script

// Plot a 20-period price channel using hhv and llv  
upper\_channel = hhv(high, 20)  
lower\_channel = llv(low, 20)  
plot(upper\_channel, title="Upper Channel")  
plot(lower\_channel, title="Lower Channel")

**highest**

* **Syntax:** highest(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series from which to find the highest value. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The highest value of the source over the last length bars.
* **Description:** The highest function is identical in functionality to hhv. It finds the maximum value of a series over a given period. It is used to define dynamic resistance levels, channel tops, and is a key part of many oscillator calculations.
* **Practical Code Example:**

Pine Script

// Find the highest RSI value over the last 50 bars  
rsi14 = rsi(close, 14)  
highest\_rsi = highest(rsi14, 50)  
plot(highest\_rsi, title="50-Period Highest RSI")

**iff**

* **Syntax:** iff(condition, value\_if\_true, value\_if\_false)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| condition | series[boolean] | Yes | The condition to evaluate. |
| value\_if\_true | any | Yes | The value to return if the condition is true. |
| value\_if\_false | any | Yes | The value to return if the condition is false. |

* **Returns:** any - The type of the return value depends on the types of value\_if\_true and value\_if\_false.
* **Description:** The iff function is a ternary conditional operator. It provides a concise, single-line way to choose between two values based on a condition. It is used extensively for defining dynamic colors, setting conditional plot values, or assigning state to variables.
* **Practical Code Example:**

Pine Script

// Plot a moving average and color it green when above the 200 SMA, red when below  
sma50 = sma(close, 50)  
sma200 = sma(close, 200)  
plot\_color = iff(sma50 > sma200, #00FF00, #FF0000)  
plot(sma50, color=plot\_color, linewidth=2)

**llv**

* **Syntax:** llv(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series from which to find the lowest value. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The lowest value of the source over the last length bars.
* **Description:** The llv (Lowest Low Value) function finds the minimum value a series has reached within a specified lookback window. It is the counterpart to hhv and is used to identify recent price troughs or support levels.
* **Practical Code Example:**

Pine Script

// Create a 50-period price channel  
upper\_band = hhv(high, 50)  
lower\_band = llv(low, 50)  
plot(upper\_band, "Upper")  
plot(lower\_band, "Lower")

**lowest**

* **Syntax:** lowest(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series from which to find the lowest value. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The lowest value of the source over the last length bars.
* **Description:** The lowest function is identical in functionality to llv. It finds the minimum value of a series over a given period and is essential for defining dynamic support levels and indicator floors.
* **Practical Code Example:**

Pine Script

// Find the lowest close price over the last 100 bars  
lowest\_close\_100 = lowest(close, 100)  
plot(lowest\_close\_100, title="100-Bar Lowest Close")

**na**

* **Syntax:** na
* **Parameters:** None.
* **Returns:** na - A special "not available" value.
* **Description:** na is a special value representing the absence of a number.27 It is used to prevent lines from being plotted on the chart under certain conditions. If a plot function is given  
  na as its value for a particular bar, nothing will be drawn for that bar, creating gaps in the plot.
* **Practical Code Example:**

Pine Script

// Only plot the 50-period SMA when the price is also above the 200-period SMA  
sma50 = sma(close, 50)  
sma200 = sma(close, 200)  
conditional\_sma = iff(close > sma200, sma50, na)  
plot(conditional\_sma, title="Conditional SMA 50")

**nz**

* **Syntax:** nz(source, replacement)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[any] | Yes | The data series to check for na values. |
| replacement | any | Optional | The value to use if source is na. Defaults to 0. |

* **Returns:** any - The original value of source if it is not na, otherwise the replacement value.
* **Description:** The nz (Not Zero) function replaces na values in a series with a specified value, typically 0. This is crucial for preventing calculation errors in cumulative sums or other operations where an na value would propagate through the entire calculation.
* **Practical Code Example:**

Pine Script

// Calculate a cumulative sum of price changes, ensuring no na values break the sum  
price\_change = change(close)  
// On the first bar, price\_change is na. nz() prevents an error.  
cumulative\_change = 0.0  
cumulative\_change := cumulative\_change + nz(price\_change)  
plot(cumulative\_change, title="Cumulative Price Change")

**rising**

* **Syntax:** rising(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to check. |
| length | integer | Yes | The number of consecutive bars the source must be rising. |

* **Returns:** series[boolean] - Returns true if the source has been increasing for the last length bars.
* **Description:** The rising function checks if a series has been consistently increasing over a specified period. It is a simple way to identify short-term upward momentum.
* **Practical Code Example:**

Pine Script

// Detect when the RSI has been rising for 3 consecutive bars  
rsi14 = rsi(close, 14)  
is\_rsi\_rising = rising(rsi14, 3)  
plotshape(is\_rsi\_rising, title="RSI Rising 3 Bars", style=shape.arrowup, location=location.belowbar, color=#00FF00)

**valuewhen**

* **Syntax:** valuewhen(condition, source, occurrence)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| condition | series[boolean] | Yes | The condition that triggers the value retrieval. |
| source | series[any] | Yes | The data series from which to get the value. |
| occurrence | integer | Optional | The n-th occurrence of the condition to look back to (0 is the most recent). Defaults to 0. |

* **Returns:** series[any] - The value of source on the bar where the n-th most recent occurrence of condition was true.
* **Description:** The valuewhen function is a powerful data retrieval tool. It allows a script to "look back in time" to a specific event and fetch the value of another series on that exact bar. It is essential for drawing historical support/resistance levels, calculating profit/loss from past signals, or referencing prices at key moments.
* **Practical Code Example:**

Pine Script

// Draw a horizontal line at the high of the bar where the last golden cross occurred  
sma50 = sma(close, 50)  
sma200 = sma(close, 200)  
golden\_cross = cross\_up(sma50, sma200)  
level\_to\_plot = valuewhen(golden\_cross, high, 0)  
plot(level\_to\_plot, style=plot.style\_circles, color=#FFD700)

### 2.3 Time & Date Functions

These functions provide access to the time and date components of each bar, enabling the creation of time-sensitive indicators and analysis.

**dayofmonth**

* **Syntax:** dayofmonth(time)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| time | series[integer] | Optional | A Unix timestamp. Defaults to the current bar's open time. |

* **Returns:** series[integer] - The day of the month (1-31).
* **Description:** The dayofmonth function returns the day of the month for each bar, based on the exchange's timezone.28 It is used to create logic that depends on specific days, such as highlighting the first trading day of the month or analyzing end-of-month effects.
* **Practical Code Example:**

Pine Script

// Highlight the background on the first trading day of each month  
is\_first\_day = dayofmonth == 1 and dayofmonth!= 1  
bgcolor(is\_first\_day? #0000FF20 : na)

**dayofweek**

* **Syntax:** dayofweek(time)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| time | series[integer] | Optional | A Unix timestamp. Defaults to the current bar's open time. |

* **Returns:** series[integer] - The day of the week (e.g., dayofweek.sunday, dayofweek.monday, etc.).
* **Description:** The dayofweek function returns the day of the week for each bar.29 This is useful for analyzing weekly patterns, such as the "Monday effect" or "Friday profit-taking," and for creating indicators that behave differently on certain days.
* **Practical Code Example:**

Pine Script

// Color bars based on the day of the week  
monday\_color = dayofweek == dayofweek.monday? #FF0000 : na  
friday\_color = dayofweek == dayofweek.friday? #0000FF : na  
barcolor(monday\_color)  
barcolor(friday\_color)

**hour**

* **Syntax:** hour(time)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| time | series[integer] | Optional | A Unix timestamp. Defaults to the current bar's open time. |

* **Returns:** series[integer] - The hour of the day (0-23) in the exchange's timezone.
* **Description:** The hour function returns the hour component of a bar's timestamp. It is essential for intraday analysis, allowing scripts to identify specific trading sessions (e.g., London open, New York session) or analyze time-of-day patterns.
* **Practical Code Example:**

Pine Script

// Highlight the first hour of the regular trading session (9:30 - 10:30 ET)  
is\_first\_hour = hour == 9 and minute >= 30 or hour == 10 and minute < 30  
bgcolor(is\_first\_hour? #FFFF0020 : na)

**minute**

* **Syntax:** minute(time)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| time | series[integer] | Optional | A Unix timestamp. Defaults to the current bar's open time. |

* **Returns:** series[integer] - The minute of the hour (0-59).
* **Description:** The minute function returns the minute component of a bar's timestamp. It is used for high-precision timing on intraday charts, such as executing logic in the first five minutes of an hour.
* **Practical Code Example:**

Pine Script

// Plot a shape on the first bar of every hour  
is\_first\_minute = minute == 0 and minute!= 0  
plotshape(is\_first\_minute, style=shape.circle, location=location.belowbar)

**month**

* **Syntax:** month(time)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| time | series[integer] | Optional | A Unix timestamp. Defaults to the current bar's open time. |

* **Returns:** series[integer] - The month of the year (1-12).
* **Description:** The month function returns the month for each bar. It is used for analyzing seasonal or monthly patterns in market data, such as the "January effect" or end-of-quarter adjustments.
* **Practical Code Example:**

Pine Script

// Highlight the month of October  
is\_october = month == 10  
bgcolor(is\_october? #FFA50020 : na)

**second**

* **Syntax:** second(time)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| time | series[integer] | Optional | A Unix timestamp. Defaults to the current bar's open time. |

* **Returns:** series[integer] - The second of the minute (0-59).
* **Description:** The second function returns the second component of a bar's timestamp. It is used for the highest level of timing precision, typically on tick-based or second-based charts, for high-frequency analysis.
* **Practical Code Example:**

Pine Script

// This example is most effective on a seconds chart  
// Highlight the first 15 seconds of every minute  
is\_early\_minute = second >= 0 and second < 15  
bgcolor(is\_early\_minute? #FFC0CB20 : na)

**time**

* **Syntax:** time
* **Parameters:** None.
* **Returns:** series[integer] - The Unix timestamp of the opening time of the current bar.
* **Description:** The time variable holds the Unix timestamp (in milliseconds) for the opening of the current bar.28 This is the fundamental time value for each bar and is used as the default input for other time functions. It is essential for any time-based calculations or comparisons.
* **Practical Code Example:**

Pine Script

// Draw a label showing the Unix timestamp on the last bar  
if barstate.islast  
 drawlabel(bar\_index, high, "Time: " + str.tostring(time))

**year**

* **Syntax:** year(time)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| time | series[integer] | Optional | A Unix timestamp. Defaults to the current bar's open time. |

* **Returns:** series[integer] - The year (e.g., 2023).
* **Description:** The year function returns the year for each bar. It is used to segment analysis by year or to create logic that applies only to specific years in a backtest.
* **Practical Code Example:**

Pine Script

// Only calculate and plot an SMA for data from the year 2022 onwards  
sma100 = sma(close, 100)  
plot(year >= 2022? sma100 : na, title="SMA since 2022")

### 2.4 Technical Analysis: Trend & Moving Average Functions

This category includes a comprehensive suite of moving averages and other trend-following indicators. Moving averages are a cornerstone of technical analysis, used to smooth price data and identify the direction of the trend.30

**alma**

* **Syntax:** alma(source, length, offset, sigma)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |
| offset | float | Yes | A value from 0 to 1 that controls the MA's offset from the current bar. |
| sigma | float | Yes | A value that controls the smoothness of the MA. |

* **Returns:** series[float] - The Arnaud Legoux Moving Average.
* **Description:** The Arnaud Legoux Moving Average (ALMA) is an advanced moving average designed to reduce lag and improve smoothness simultaneously. The offset parameter shifts the filter's focus, while sigma adjusts its smoothness, allowing for fine-tuning to specific market conditions.
* **Practical Code Example:**

Pine Script

alma\_val = alma(close, 9, 0.85, 6)  
plot(alma\_val, title="ALMA", color=#9C27B0)

**dema**

* **Syntax:** dema(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Double Exponential Moving Average.
* **Description:** The Double Exponential Moving Average (DEMA) was developed to reduce the lag inherent in traditional EMAs. It is more responsive to recent price changes, making it suitable for shorter-term trend analysis and signal generation.
* **Practical Code Example:**

Pine Script

dema20 = dema(close, 20)  
plot(dema20, title="DEMA(20)", color=#4CAF50)

**ema**

* **Syntax:** ema(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Exponential Moving Average.
* **Description:** The Exponential Moving Average (EMA) is one of the most popular types of moving averages. It gives more weight to recent prices, making it react more quickly to price changes than a Simple Moving Average (SMA).7 It is widely used in trend-following strategies and as a component in other indicators like the MACD.
* **Practical Code Example:**

Pine Script

ema50 = ema(close, 50)  
plot(ema50, title="EMA(50)", color=#2196F3)

**hma**

* **Syntax:** hma(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Hull Moving Average.
* **Description:** The Hull Moving Average (HMA) is an extremely fast and smooth moving average. It aims to almost completely eliminate lag while simultaneously improving smoothness. It is often used by traders looking for very responsive trend signals.
* **Practical Code Example:**

Pine Script

hma16 = hma(close, 16)  
plot(hma16, title="HMA(16)", color=#E91E63)

**linreg**

* **Syntax:** linreg(source, length, offset)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period for the regression. |
| offset | integer | Yes | The offset from the current bar to calculate the value for. |

* **Returns:** series[float] - The value of a linear regression curve.
* **Description:** The linreg function calculates a linear regression trend line over a specified period. It represents the "best fit" straight line through the data points. It is used to identify the prevailing trend and can also be used for forecasting (by using a negative offset).
* **Practical Code Example:**

Pine Script

// Plot a 100-period linear regression line  
linreg100 = linreg(close, 100, 0)  
plot(linreg100, title="Linear Regression", color=#FF9800)

**sma**

* **Syntax:** sma(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Simple Moving Average.
* **Description:** The Simple Moving Average (SMA) is the most basic form of moving average. It calculates the arithmetic mean of prices over a specified number of past periods, giving equal weight to each price.18 It is a smooth indicator of the underlying trend, commonly used for long-term analysis.
* **Practical Code Example:**

Pine Script

sma200 = sma(close, 200)  
plot(sma200, title="SMA(200)", color=#9E9E9E)

**smma**

* **Syntax:** smma(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Smoothed Moving Average.
* **Description:** The Smoothed Moving Average (SMMA) is similar to an EMA but gives weight to prices far into the past. It is a very smooth trend indicator that moves more slowly than an EMA or SMA of the same period, making it suitable for identifying long-term trends.
* **Practical Code Example:**

Pine Script

smma100 = smma(close, 100)  
plot(smma100, title="SMMA(100)", color=#795548)

**swma**

* **Syntax:** swma(source)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |

* **Returns:** series[float] - The Symmetrically Weighted Moving Average.
* **Description:** The Symmetrically Weighted Moving Average is a specialized 4-period moving average with specific weights [1/6, 2/6, 2/6, 1/6]. It is a very short-term and smooth filter.
* **Practical Code Example:**

Pine Script

swma\_val = swma(close)  
plot(swma\_val, title="SWMA", color=#00BCD4)

**tema**

* **Syntax:** tema(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Triple Exponential Moving Average.
* **Description:** The Triple Exponential Moving Average (TEMA) is another moving average designed to reduce lag. It is even more responsive than a DEMA, making it highly sensitive to short-term price movements.
* **Practical Code Example:**

Pine Script

tema20 = tema(close, 20)  
plot(tema20, title="TEMA(20)", color=#673AB7)

**tr**

* **Syntax:** tr(handle\_na)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| handle\_na | boolean | Optional | If true, on the first bar it will use high - low instead of returning na. |

* **Returns:** series[float] - The True Range.
* **Description:** The True Range (TR) is a measure of volatility for a single period. It is defined as the greatest of the following: the current high minus the current low; the absolute value of the current high minus the previous close; the absolute value of the current low minus the previous close. TR is the fundamental component of the Average True Range (ATR) indicator.
* **Practical Code Example:**

Pine Script

true\_range = tr()  
plot(true\_range, title="True Range")

**vwap**

* **Syntax:** vwap(source)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |

* **Returns:** series[float] - The Volume-Weighted Average Price.
* **Description:** The Volume-Weighted Average Price (VWAP) is a trading benchmark that represents the average price a security has traded at throughout the day, based on both price and volume. It is an important indicator for institutional traders and is often used as a reference for execution quality. The VWAP calculation resets at the beginning of each new trading session.
* **Practical Code Example:**

Pine Script

// The typical price is often used as the source for VWAP  
typical\_price = (high + low + close) / 3  
vwap\_val = vwap(typical\_price)  
plot(vwap\_val, title="VWAP", color=#FFEB3B)

**wma**

* **Syntax:** wma(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Weighted Moving Average.
* **Description:** The Weighted Moving Average (WMA) is similar to an EMA in that it gives more weight to recent data. However, it does so in a linear fashion, whereas the EMA's weighting is exponential. The most recent price gets the highest weight, and the weight decreases linearly for older prices.
* **Practical Code Example:**

Pine Script

wma25 = wma(close, 25)  
plot(wma25, title="WMA(25)", color=#3F51B5)

**wpr**

* **Syntax:** wpr(length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Williams %R value.
* **Description:** Williams %R is a momentum indicator that is the inverse of the Stochastic Oscillator. It moves between 0 and -100 and is used to identify overbought (typically 0 to -20) and oversold (typically -80 to -100) levels.
* **Practical Code Example:**

Pine Script

wpr14 = wpr(14)  
plot(wpr14, title="Williams %R (14)")  
hline(-20, "Overbought")  
hline(-80, "Oversold")

### 2.5 Technical Analysis: Momentum & Oscillator Functions

Oscillators are indicators that fluctuate above and below a centerline or between set levels. They are primarily used to identify overbought or oversold conditions, divergences, and the momentum of price movements.31

**asi**

* **Syntax:** asi()
* **Parameters:** None.
* **Returns:** series[float] - The Accumulative Swing Index.
* **Description:** The Accumulative Swing Index (ASI) is a cumulative total of the Swing Index. It is a momentum indicator that attempts to show the "real" market trend by filtering out noise from open, high, low, and close prices.32 It is often used to confirm trendline breakouts.
* **Practical Code Example:**

Pine Script

asi\_val = asi()  
plot(asi\_val, title="Accumulative Swing Index")

**bias**

* **Syntax:** bias(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period for the moving average. |

* **Returns:** series[float] - The Bias Ratio.
* **Description:** The Bias Ratio measures how far the current price has deviated from a moving average, expressed as a percentage.33 It is an oscillator that can help identify overextended markets. High positive values suggest an overbought condition, while large negative values suggest an oversold condition.
* **Practical Code Example:**

Pine Script

// Calculate bias from a 25-period SMA  
bias25 = bias(close, 25)  
plot(bias25, title="Bias Ratio (25)")  
hline(0, "Centerline")

**cci**

* **Syntax:** cci(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Commodity Channel Index.
* **Description:** The Commodity Channel Index (CCI) is a versatile momentum oscillator that measures the current price level relative to an average price level over a given period.34 CCI is relatively high when prices are far above their average and relatively low when prices are far below. It is often used to identify cyclical trends and overbought/oversold conditions, typically using +100 and -100 as key levels.
* **Practical Code Example:**

Pine Script

cci20 = cci(close, 20)  
plot(cci20, title="CCI (20)")  
hline(100, "Overbought")  
hline(-100, "Oversold")

**cmo**

* **Syntax:** cmo(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Chande Momentum Oscillator.
* **Description:** The Chande Momentum Oscillator (CMO) is a momentum indicator that measures net momentum, similar to the RSI, but calculates it differently. It oscillates between +100 and -100. Readings above +50 are considered overbought, and readings below -50 are considered oversold.
* **Practical Code Example:**

Pine Script

cmo14 = cmo(close, 14)  
plot(cmo14, title="CMO (14)")  
hline(50, "Overbought")  
hline(-50, "Oversold")

**cr**

* **Syntax:** cr(length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The CR indicator value.
* **Description:** The CR indicator (Energy indicator) is a momentum oscillator based on the relationship between the midpoint of the previous bar and the current high and low prices.35 It is used to gauge the strength of buyers and sellers.
* **Practical Code Example:**

Pine Script

cr26 = cr(26)  
plot(cr26, title="CR Indicator (26)")

**dmi**

* **Syntax:** dmi(len, adx\_len)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| len | integer | Yes | The lookback period for DI+ and DI-. |
| adx\_len | integer | Yes | The smoothing period for the ADX line. |

* **Returns:** [series[float], series[float], series[float]] - A tuple containing the ADX, DI+, and DI- lines.
* **Description:** The Directional Movement Index (DMI) is a comprehensive trend-following system composed of three lines: the ADX (Average Directional Index), DI+ (Positive Directional Indicator), and DI- (Negative Directional Indicator).36 The ADX line measures trend strength (a rising ADX suggests a strong trend, either up or down), while the crossover of DI+ and DI- provides directional signals.
* **Practical Code Example:**

Pine Script

[adx, di\_plus, di\_minus] = dmi(14, 14)  
plot(adx, title="ADX", color=#FF9800)  
plot(di\_plus, title="DI+", color=#00FF00)  
plot(di\_minus, title="DI-", color=#FF0000)

**dpo**

* **Syntax:** dpo(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Detrended Price Oscillator.
* **Description:** The Detrended Price Oscillator (DPO) is an indicator designed to remove the long-term trend from prices to make it easier to identify cycles.37 The DPO compares a past price to a moving average, effectively isolating the short-term cycle.
* **Practical Code Example:**

Pine Script

dpo21 = dpo(close, 21)  
plot(dpo21, title="DPO (21)")  
hline(0)

**macd**

* **Syntax:** macd(source, fast\_len, slow\_len, sig\_len)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| fast\_len | integer | Yes | The length for the fast EMA. |
| slow\_len | integer | Yes | The length for the slow EMA. |
| sig\_len | integer | Yes | The length for the signal line EMA. |

* **Returns:** [series[float], series[float], series[float]] - A tuple containing the MACD line, signal line, and histogram.
* **Description:** The Moving Average Convergence Divergence (MACD) is a classic trend-following momentum indicator that shows the relationship between two exponential moving averages of a security’s price.26 The MACD line is the difference between the two EMAs. The signal line is an EMA of the MACD line, and the histogram is the difference between the MACD and signal lines. Crossovers and divergences are key signals.
* **Practical Code Example:**

Pine Script

[macd\_line, signal\_line, hist] = macd(close, 12, 26, 9)  
plot(macd\_line, title="MACD", color=#2962FF)  
plot(signal\_line, title="Signal", color=#FF6D00)  
plot(hist, title="Histogram", style=plot.style\_histogram, color=#9E9E9E)

**mfi**

* **Syntax:** mfi(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on (typically hlc3). |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Money Flow Index.
* **Description:** The Money Flow Index (MFI) is a momentum indicator that incorporates both price and volume to measure buying and selling pressure. It is often referred to as a volume-weighted RSI. It oscillates between 0 and 100 and is used to identify overbought (typically > 80) and oversold (typically < 20) conditions.
* **Practical Code Example:**

Pine Script

mfi14 = mfi(hlc3, 14)  
plot(mfi14, title="MFI (14)")  
hline(80, "Overbought")  
hline(20, "Oversold")

**mom**

* **Syntax:** mom(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Momentum value.
* **Description:** The Momentum (MOM) indicator is a simple oscillator that measures the rate of price change over a given period. It is calculated as the difference between the current price and the price length periods ago. It can be used to identify the strength of a trend and potential reversal points.
* **Practical Code Example:**

Pine Script

mom10 = mom(close, 10)  
plot(mom10, title="Momentum (10)")  
hline(0)

**roc**

* **Syntax:** roc(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Rate of Change value.
* **Description:** The Rate of Change (ROC) indicator is a momentum oscillator that measures the percentage change in price between the current price and the price length periods ago. It is very similar to the Momentum indicator but is expressed as a percentage, which normalizes it for comparison across different securities.
* **Practical Code Example:**

Pine Script

roc12 = roc(close, 12)  
plot(roc12, title="Rate of Change (12)")  
hline(0)

**rsi**

* **Syntax:** rsi(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Relative Strength Index.
* **Description:** The Relative Strength Index (RSI) is one of the most popular momentum oscillators. It measures the speed and change of price movements by comparing the magnitude of recent gains to recent losses.31 It oscillates between 0 and 100. Values above 70 are typically considered overbought, and values below 30 are considered oversold.
* **Practical Code Example:**

Pine Script

rsi14 = rsi(close, 14)  
plot(rsi14, title="RSI (14)")  
hline(70, "Overbought")  
hline(30, "Oversold")

**stoch**

* **Syntax:** stoch(source, high, low, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series for the main line (typically close). |
| high | series[float] | Yes | The high price series. |
| low | series[float] | Yes | The low price series. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Stochastic Oscillator %K value.
* **Description:** The Stochastic Oscillator is a momentum indicator that compares a particular closing price of a security to a range of its prices over a certain period of time. It shows the location of the close relative to the high-low range over a set number of periods. It oscillates between 0 and 100 and is used to identify overbought (> 80) and oversold (< 20) conditions.
* **Practical Code Example:**

Pine Script

// This returns the %K line. A signal line (%D) is typically an SMA of %K.  
stoch\_k = stoch(close, high, low, 14)  
stoch\_d = sma(stoch\_k, 3)  
plot(stoch\_k, title="%K")  
plot(stoch\_d, title="%D")

**trix**

* **Syntax:** trix(source, length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The TRIX indicator value.
* **Description:** The TRIX indicator is a momentum oscillator that displays the percentage rate of change of a triple exponentially smoothed moving average. It is designed to filter out insignificant price movements, making it a good trend-following indicator.
* **Practical Code Example:**

Pine Script

trix18 = trix(close, 18)  
plot(trix18, title="TRIX (18)")  
hline(0)

### 2.6 Technical Analysis: Volatility Functions

Volatility functions measure the rate and magnitude of price fluctuations. They are crucial for risk management and for identifying periods of market consolidation and expansion.22

**atr**

* **Syntax:** atr(length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The Average True Range.
* **Description:** The Average True Range (ATR) is a key measure of market volatility. It is an average of the True Range (TR) over a specified period. A rising ATR indicates increasing volatility, while a falling ATR indicates decreasing volatility. It is commonly used for setting stop-loss levels and position sizing.
* **Practical Code Example:**

Pine Script

atr14 = atr(14)  
plot(atr14, title="Average True Range (14)")

**band**

* **Syntax:** band(source, length, mult)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period for the basis MA. |
| mult | float | Yes | The multiplier for the standard deviation. |

* **Returns:** [series[float], series[float], series[float]] - A tuple containing the basis, upper band, and lower band.
* **Description:** This function appears to be an alternative name or a simplified version of the Bollinger Bands function, boll.38 It calculates a middle band (moving average) and upper/lower bands based on a standard deviation multiplier.
* **Practical Code Example:**

Pine Script

[basis, upper, lower] = band(close, 20, 2)  
plot(basis, title="Basis")  
plot(upper, title="Upper Band")  
plot(lower, title="Lower Band")

**boll**

* **Syntax:** boll(source, length, mult)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period for the basis MA. |
| mult | float | Yes | The multiplier for the standard deviation. |

* **Returns:** [series[float], series[float], series[float]] - A tuple containing the basis, upper band, and lower band.
* **Description:** The boll function calculates Bollinger Bands, a classic volatility indicator.39 They consist of a middle band (typically a 20-period SMA) and upper and lower bands that are typically two standard deviations above and below the middle band. The bands widen when volatility increases and contract (squeeze) when volatility decreases.
* **Practical Code Example:**

Pine Script

[basis, upper, lower] = boll(close, 20, 2)  
p1 = plot(basis, color=#FF6D00)  
p2 = plot(upper, color=#2962FF)  
p3 = plot(lower, color=#2962FF)  
fill(p2, p3, color=#2962FF20)

**bbiboll**

* **Syntax:** bbiboll(source, length, mult)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |
| mult | float | Yes | The multiplier for the standard deviation. |

* **Returns:** series[float] - The Bollinger Bands %B value.
* **Description:** This function calculates the Bollinger Bands %B indicator. %B quantifies where the price is in relation to the bands. A value of 1 means the price is at the upper band, 0.5 means it's at the middle band, and 0 means it's at the lower band. It can be used to identify overbought/oversold conditions in a more normalized way.
* **Practical Code Example:**

Pine Script

percent\_b = bbiboll(close, 20, 2)  
plot(percent\_b, title="%B")  
hline(1, "Upper Band")  
hline(0, "Lower Band")

**keltner**

* **Syntax:** keltner(source, length, mult)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on. |
| length | integer | Yes | The lookback period. |
| mult | float | Yes | The multiplier for the Average True Range. |

* **Returns:** [series[float], series[float], series[float]] - A tuple containing the basis, upper band, and lower band.
* **Description:** This function calculates Keltner Channels. These are volatility-based bands placed above and below a moving average. Unlike Bollinger Bands which use standard deviation, Keltner Channels use the Average True Range (ATR) to set the channel distance. They are used to identify trends and potential reversal points.
* **Practical Code Example:**

Pine Script

[basis, upper, lower] = keltner(close, 20, 1.5)  
plot(basis, title="Basis")  
plot(upper, title="Upper KC")  
plot(lower, title="Lower KC")

### 2.7 Technical Analysis: Volume & Price-Volume Functions

These functions incorporate trading volume into their calculations, providing insights into the strength and conviction behind price movements.

**arbr**

* **Syntax:** arbr(length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| length | integer | Yes | The lookback period. |

* **Returns:** [series[float], series[float]] - A tuple containing the AR and BR lines.
* **Description:** This function calculates the AR (Popularity Ratio) and BR (Willingness to Buy/Sell Ratio) indicators.40 These are sentiment indicators from Asian technical analysis that measure market energy and buying/selling pressure.
* **Practical Code Example:**

Pine Script

[ar, br] = arbr(26)  
plot(ar, title="AR")  
plot(br, title="BR")

**emv**

* **Syntax:** emv(length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| length | integer | Yes | The lookback period for the moving average. |

* **Returns:** series[float] - The Ease of Movement value.
* **Description:** The Ease of Movement (EMV) indicator relates price change to volume and is designed to measure how easily prices are moving up or down.41 High positive values indicate price is moving up easily on low volume, while large negative values indicate price is moving down easily on low volume.
* **Practical Code Example:**

Pine Script

emv14 = emv(14)  
plot(emv14, title="Ease of Movement (14)")

**obv**

* **Syntax:** obv(source)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| source | series[float] | Yes | The data series to calculate on (typically close). |

* **Returns:** series[float] - The On-Balance Volume.
* **Description:** On-Balance Volume (OBV) is a cumulative momentum indicator that uses volume flow to predict changes in stock price. If the price closes higher, the day's volume is added to the OBV total. If it closes lower, the volume is subtracted. The direction of the OBV line is often more important than its absolute value and is used to confirm price trends.
* **Practical Code Example:**

Pine Script

obv\_val = obv(close)  
plot(obv\_val, title="On-Balance Volume")

**pv**

* **Syntax:** pv(length)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| length | integer | Yes | The lookback period. |

* **Returns:** series[float] - The PV indicator value.
* **Description:** This function likely calculates a Price-Volume indicator, which analyzes the relationship between price movement and trading volume to gauge the strength of a trend. The specific formula may vary, but it generally aims to confirm trends with corresponding volume patterns.
* **Practical Code Example:**

Pine Script

pv\_val = pv(20)  
plot(pv\_val, title="Price-Volume Indicator")

### 2.8 Plotting & Visual Rendering Functions

These functions are responsible for drawing all visual elements on the chart, turning raw data into actionable, graphical information.

**bgcolor**

* **Syntax:** bgcolor(color, offset, editable, show\_last, title, force\_overlay)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| color | series[color] | Yes | The color to apply to the background. Use na for no color. |
| offset | integer | Optional | Shifts the background coloring forward or backward. |
| editable | boolean | Optional | If true, the color can be edited in the indicator settings. |
| show\_last | integer | Optional | If set, only colors the background for the last n bars. |
| title | string | Optional | The title for the setting in the indicator's properties. |
| force\_overlay | boolean | Optional | Forces the background to apply to the main chart pane. |

* **Returns:** void
* **Description:** The bgcolor function changes the color of the chart's background on a bar-by-bar basis.13 It is a powerful tool for visual signaling, used to highlight specific market conditions, trading sessions, or trend states across the entire chart.
* **Practical Code Example:**

Pine Script

// Color the background green when RSI is oversold and red when overbought  
rsi14 = rsi(close, 14)  
bg\_color = iff(rsi14 > 70, #FF000020, iff(rsi14 < 30, #00FF0020, na))  
bgcolor(bg\_color)

**fill**

* **Syntax:** fill(plot1, plot2, color)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| plot1 | plot | Yes | The ID of the first plot, returned from a plot() call. |
| plot2 | plot | Yes | The ID of the second plot, returned from a plot() call. |
| color | series[color] | Optional | The color of the fill. |

* **Returns:** void
* **Description:** The fill function colors the area between two plotted lines.42 This is used to create visual zones, such as the cloud in an Ichimoku indicator or the space between Bollinger Bands, making chart analysis more intuitive.
* **Practical Code Example:**

Pine Script

// Fill the area between a 20-period and 50-period EMA  
ema20 = plot(ema(close, 20))  
ema50 = plot(ema(close, 50))  
fill(ema20, ema50, color=ema(close, 20) > ema(close, 50)? #00FF0020 : #FF000020)

**plot**

* **Syntax:** plot(series, title, color, linewidth, style, trackprice, histbase, offset, join, editable, show\_last, display)
* **Parameters:** Many parameters are optional for customizing appearance.

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| series | series[float] | Yes | The data series to plot. |
| title | string | Optional | The name of the plot in the settings and data window. |
| color | series[color] | Optional | The color of the plot. |
| linewidth | integer | Optional | The thickness of the line. |
| style | constant | Optional | The style of the plot (e.g., plot.style\_line, plot.style\_histogram). |

* **Returns:** plot - An ID that can be used by the fill() function.
* **Description:** plot is the primary function for drawing data on the chart.12 It can render lines, histograms, circles, crosses, and area plots. Nearly every indicator will use  
  plot to display its calculated values.
* **Practical Code Example:**

Pine Script

// Plot the 14-period RSI as a purple line  
rsi14 = rsi(close, 14)  
plot(rsi14, title="RSI", color=#6A1B9A, linewidth=2)

**plotarrow**

* **Syntax:** plotarrow(series, title, colorup, colordown, offset, minheight, maxheight, editable, show\_last, display)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| series | series[float] | Yes | A series of values. A positive value plots an up arrow, a negative value plots a down arrow, and zero plots nothing. |
| colorup | color | Optional | Color for the up arrows. |
| colordown | color | Optional | Color for the down arrows. |

* **Returns:** void
* **Description:** The plotarrow function draws up or down arrows on the chart. It is a specialized plotting function designed for explicitly showing directional signals, such as the direction of momentum or trend.
* **Practical Code Example:**

Pine Script

// Plot an up arrow when momentum is positive, down arrow when negative  
mom10 = mom(close, 10)  
plotarrow(mom10, colorup=#00FF00, colordown=#FF0000)

**plotbar**

* **Syntax:** plotbar(open, high, low, close, title, color, editable, show\_last, display)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| open | series[float] | Yes | The open price series for the bars. |
| high | series[float] | Yes | The high price series for the bars. |
| low | series[float] | Yes | The low price series for the bars. |
| close | series[float] | Yes | The close price series for the bars. |
| color | series[color] | Optional | The color of the bars. |

* **Returns:** void
* **Description:** The plotbar function allows a script to draw its own custom price bars on the chart. This is useful for creating synthetic charts or visualizing price action based on custom calculations rather than the underlying security's actual prices.
* **Practical Code Example:**

Pine Script

// Plot bars based on a 3-period moving average of each OHLC component  
avg\_open = sma(open, 3)  
avg\_high = sma(high, 3)  
avg\_low = sma(low, 3)  
avg\_close = sma(close, 3)  
plotbar(avg\_open, avg\_high, avg\_low, avg\_close, title="Smoothed Bars")

**plotcandle**

* **Syntax:** plotcandle(open, high, low, close, title, color, wickcolor, bordercolor, editable, show\_last, display)
* **Parameters:** Similar to plotbar, but with options for wick and border colors.
* **Returns:** void
* **Description:** The plotcandle function draws custom candlesticks on the chart. It is used for the same purposes as plotbar but provides the traditional candlestick visualization, which many traders prefer for analyzing price action.
* **Practical Code Example:**

Pine Script

// Plot Heikin-Ashi style candles (simplified calculation)  
ha\_close = (open + high + low + close) / 4  
ha\_open = (nz(ha\_open) + nz(ha\_close)) / 2  
ha\_high = max(high, ha\_open, ha\_close)  
ha\_low = min(low, ha\_open, ha\_close)  
candle\_color = ha\_close > ha\_open? #00FF00 : #FF0000  
plotcandle(ha\_open, ha\_high, ha\_low, ha\_close, color=candle\_color)

**plotchar**

* **Syntax:** plotchar(series, title, char, location, color, offset, text, editable, size, show\_last, display)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| series | series[boolean] | Yes | A boolean series. The character is plotted when the series is true. |
| char | string | Optional | The character to plot. |
| location | constant | Optional | Where to place the character relative to the bar (e.g., location.abovebar). |

* **Returns:** void
* **Description:** The plotchar function plots a single text character on the chart when a condition is met. It is a lightweight way to add simple, non-intrusive signals or markers to the chart.
* **Practical Code Example:**

Pine Script

// Plot a star character above bars that close in the top 10% of their range  
is\_strong\_close = (close - low) / (high - low) > 0.9  
plotchar(is\_strong\_close, char='\*', location=location.abovebar, color=#FFD700)

**plotshape**

* **Syntax:** plotshape(series, title, style, location, color, offset, text, editable, size, show\_last, display)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| series | series[boolean] | Yes | A boolean series. The shape is plotted when the series is true. |
| style | constant | Optional | The shape to plot (e.g., shape.triangleup, shape.cross). |
| location | constant | Optional | Where to place the shape relative to the bar. |

* **Returns:** void
* **Description:** The plotshape function is one of the most common ways to display trading signals. It draws a predefined shape on the chart when a specified condition is true, making it ideal for marking entry/exit points, divergences, or other key events.
* **Practical Code Example:**

Pine Script

// Plot a green upward triangle for a bullish engulfing candle  
bullish\_engulfing = close > open and close < open and close > open and open < close  
plotshape(bullish\_engulfing, style=shape.triangleup, location=location.belowbar, color=#00FF00, size=size.small)

**drawlabel**

* **Syntax:** drawlabel(x, y, text, xloc, yloc, color, style, textcolor, size, textalign, tooltip)
* **Parameters:**

| Parameter | Type | Required | Description |
| --- | --- | --- | --- |
| x | integer | Yes | The x-coordinate (bar index) for the label. |
| y | float | Yes | The y-coordinate (price level) for the label. |
| text | string | Yes | The text to display in the label. |

* **Returns:** label - An ID for the created label object.
* **Description:** The drawlabel function creates a text label at a specific coordinate on the chart.14 Unlike  
  plotchar or plotshape, labels are drawing objects that persist and can be modified or deleted programmatically. They are used for displaying detailed, dynamic information directly on the chart.
* **Practical Code Example:**

Pine Script

// On the last bar, draw a label showing the current RSI value  
if barstate.islast  
 rsi\_val = rsi(close, 14)  
 drawlabel(bar\_index, high, "RSI: " + str.tostring(rsi\_val, "#.##"))

**drawtext**

* **Syntax:** drawtext(x, y, text, xloc, yloc, color, size, textalign, tooltip)
* **Parameters:** Similar to drawlabel.
* **Returns:** text - An ID for the created text object.
* **Description:** The drawtext function is similar to drawlabel but draws plain text without a background box or border. It is used for creating more minimalist visual displays of information on the chart.
* **Practical Code Example:**

Pine Script

// Mark the highest high of the last 50 bars with its price value  
highest\_high\_50 = highest(high, 50)  
bar\_of\_highest = barslast(high == highest\_high\_50)  
if barstate.islast  
 drawtext(bar\_index - bar\_of\_highest, highest\_high\_50, str.tostring(highest\_high\_50), location=location.abovebar)

## Section 3: Practical Implementation: Crafting Indicators from First Principles

Understanding individual functions is the first step; combining them to create functional, insightful indicators is the goal. This section provides step-by-step tutorials for building three distinct types of custom indicators, progressively introducing more advanced concepts and visualization techniques.

### 3.1 Tutorial 1: Building a Dynamic Multi-EMA Crossover Indicator

This tutorial will create a classic trend-following indicator. The objective is to plot two user-configurable Exponential Moving Averages (EMAs) and visually mark the points where they cross, which are often interpreted as buy or sell signals.24

**Concepts Covered:**

* Defining user inputs for flexibility.
* Calculating multiple indicator values.
* Plotting lines with dynamic colors.
* Using conditional logic (cross\_up, cross\_down) to detect events.
* Plotting shapes (plotshape) to mark signals on the chart.

**Step-by-Step Implementation:**

1. Declaration and Inputs:  
   Begin by declaring the script as an indicator that will overlay the main price chart. Then, create inputs for the lengths of the fast and slow EMAs, allowing the user to customize the indicator without editing the code.  
   Pine Script  
   // 1. Declaration  
   indicator("EMA Crossover Signal", overlay=true)  
     
   // 2. Inputs  
   fast\_length = input(9, title="Fast EMA Length")  
   slow\_length = input(21, title="Slow EMA Length")
2. Calculations:  
   Next, calculate the two EMAs using the ema() function and the input lengths. Also, define the boolean conditions for the bullish (golden cross) and bearish (death cross) signals using the cross\_up() and cross\_down() functions.  
   Pine Script  
   // 3. Calculations  
   fast\_ema = ema(close, fast\_length)  
   slow\_ema = ema(close, slow\_length)  
     
   buy\_signal = cross\_up(fast\_ema, slow\_ema)  
   sell\_signal = cross\_down(fast\_ema, slow\_ema)
3. Plotting and Visualization:  
   Finally, plot the two EMA lines on the chart. To enhance readability, the color of the fast EMA can be changed based on whether it is above or below the slow EMA. Then, use plotshape() to place distinct markers on the chart whenever a buy\_signal or sell\_signal occurs.  
   Pine Script  
   // 4. Outputs  
   fast\_color = fast\_ema > slow\_ema? #00FF00 : #FF0000  
   plot(fast\_ema, title="Fast EMA", color=fast\_color, linewidth=2)  
   plot(slow\_ema, title="Slow EMA", color=#808080, linewidth=2)  
     
   plotshape(buy\_signal, title="Buy Signal", style=shape.triangleup, location=location.belowbar, color=#00FF00, size=size.small)  
   plotshape(sell\_signal, title="Sell Signal", style=shape.triangledown, location=location.abovebar, color=#FF0000, size=size.small)

This complete script provides a versatile and visually clear trend-following tool. Users can adjust the EMA lengths to suit different timeframes and trading styles, while the explicit crossover signals remove ambiguity from the analysis.

### 3.2 Tutorial 2: Developing a Visualized RSI Oscillator with Dynamic Zones

This tutorial focuses on creating a Relative Strength Index (RSI) indicator that uses background coloring to visually emphasize overbought and oversold conditions. This approach makes it easier to identify critical market states at a glance.

**Concepts Covered:**

* Plotting an indicator in a separate pane (overlay=false).
* Drawing horizontal lines (hline) for key levels.
* Using conditional logic (iff) for dynamic coloring.
* Applying color to the indicator's background (bgcolor).

**Step-by-Step Implementation:**

1. Declaration and Inputs:  
   Declare the script as an indicator that will appear in its own pane below the price chart. Create inputs for the RSI length and the overbought/oversold levels.  
   Pine Script  
   // 1. Declaration  
   indicator("Visualized RSI", overlay=false)  
     
   // 2. Inputs  
   rsi\_length = input(14, title="RSI Length")  
   overbought\_level = input(70, title="Overbought Level")  
   oversold\_level = input(30, title="Oversold Level")
2. Calculations:  
   Calculate the RSI value using the rsi() function. Then, create a conditional color variable for the background. The background will be colored red when the RSI is in the overbought zone and green when it is in the oversold zone.  
   Pine Script  
   // 3. Calculations  
   rsi\_value = rsi(close, rsi\_length)  
     
   bg\_color = iff(rsi\_value > overbought\_level, #FF000020,  
    iff(rsi\_value < oversold\_level, #00FF0020, na))  
     
   The na value ensures that no color is applied when the RSI is in the neutral zone between the two levels. The #FF000020 format specifies a color (red) with transparency.
3. Plotting and Visualization:  
   Plot the RSI line itself. Then, draw horizontal lines at the overbought and oversold levels for clear reference. Finally, apply the calculated bg\_color to the indicator's background using the bgcolor() function.  
   Pine Script  
   // 4. Outputs  
   plot(rsi\_value, title="RSI", color=#800080)  
     
   hline(overbought\_level, "Overbought", color=#FF0000)  
   hline(oversold\_level, "Oversold", color=#00FF00)  
     
   bgcolor(bg\_color)

This indicator provides a much richer visual experience than a standard RSI plot. The background coloring instantly draws the analyst's attention to periods of potential market exhaustion, making it faster to identify trading opportunities or risks.

### 3.3 Tutorial 3: Advanced Visualization: Bollinger Bands with Conditional Filling

This advanced tutorial demonstrates how to create a highly informative Bollinger Bands indicator. It will not only plot the bands but also use the fill() function to color the space between them, with the color changing to signal a potential "Bollinger Squeeze" breakout.

**Concepts Covered:**

* Handling multiple return values from a single function (boll).
* Using the fill() function to color the area between plots.
* Creating a "squeeze" condition based on the bandwidth.
* Applying conditional coloring to the fill area to represent market state.

**Step-by-Step Implementation:**

1. Declaration and Inputs:  
   Declare the script as an overlay indicator. Provide inputs for the Bollinger Bands length and standard deviation multiplier.  
   Pine Script  
   // 1. Declaration  
   indicator("Advanced Bollinger Bands", overlay=true)  
     
   // 2. Inputs  
   bb\_length = input(20, title="BB Length")  
   bb\_mult = input(2.0, title="BB Multiplier")
2. Calculations:  
   Calculate the Bollinger Bands using the boll() function, which conveniently returns the basis (middle), upper, and lower bands in a single call. Then, calculate the Bollinger Bandwidth, which is a measure of how wide the bands are. A very low bandwidth indicates a "squeeze." Finally, define the fill color based on market conditions: a neutral color during normal periods, and a distinct color (e.g., yellow) when the price closes outside the bands, signaling a potential breakout.  
   Pine Script  
   // 3. Calculations  
   [basis, upper, lower] = boll(close, bb\_length, bb\_mult)  
     
   // Squeeze condition: Bandwidth is at its lowest point in the last 50 bars  
   bandwidth = (upper - lower) / basis  
   is\_squeeze = bandwidth == lowest(bandwidth, 50)  
     
   // Fill color logic  
   fill\_color = is\_squeeze? #FFFF0020 : #0000FF20 // Yellow for squeeze, blue otherwise
3. Plotting and Visualization:  
   Plot the three bands. Crucially, assign the return values of the upper and lower band plots to variables (p\_upper, p\_lower). These plot IDs are required by the fill() function. Then, call fill() to color the space between these two plots using the dynamic fill\_color.  
   Pine Script  
   // 4. Outputs  
   plot(basis, title="Basis", color=#FFA500)  
   p\_upper = plot(upper, title="Upper Band", color=#0000FF)  
   p\_lower = plot(lower, title="Lower Band", color=#0000FF)  
     
   fill(p\_upper, p\_lower, color=fill\_color)

This indicator transforms the standard Bollinger Bands into a more dynamic analytical tool. The colored fill provides an immediate visual cue for the market's volatility state, highlighting periods of consolidation (squeeze) that often precede significant price moves.

## Section 4: Advanced Scripting Techniques and Best Practices

As scripts grow in complexity, adhering to best practices becomes essential for creating indicators that are robust, readable, and efficient. This section covers advanced techniques that elevate scripts from simple calculations to professional-grade analytical tools.

### 4.1 Writing Clean and Readable Code

The clarity of the source code is paramount for long-term use, debugging, and sharing. A script that is difficult to understand is difficult to trust and maintain.

* **Meaningful Variable Names:** Use descriptive names for variables. Instead of x and y, use fast\_ema and slow\_ema. This makes the logic of the script self-evident.
* **Comments:** Use comments (//) to explain the *why* behind the code, not just the *what*. Explain the purpose of a complex calculation or the trading rationale behind a condition.12 Organize code into logical sections with comment headers (e.g.,  
  // --- Inputs ---, // --- Calculations ---).
* **Logical Structure:** Group related lines of code. Define all inputs at the beginning, followed by all calculations, and finally all plotting functions. This standardized structure, as demonstrated in the tutorials, makes scripts predictable and easy to navigate.

### 4.2 Creating Dynamic and Interactive Visuals

The true power of custom scripting lies in the ability to create information-dense visuals that go far beyond simple lines. The goal is to have the chart communicate complex market states intuitively.

* **Dynamic Labels:** The drawlabel() and drawtext() functions are exceptionally powerful for this purpose.14 They can be used to display dynamic data directly on the chart. For example, a label can be programmed to appear next to a key pivot point showing its exact price, or a label on the last bar could display the current values of multiple indicators (RSI, ATR, etc.) in a single block of text. This consolidates information and reduces the need for multiple indicator panes.
* **Conditional Coloring:** As shown in the tutorials, color should be used as a data layer, not just for aesthetics. Every visual element—lines, backgrounds, fills, shapes—can have its color determined by a conditional expression. A moving average can change color based on its slope, or candlestick bodies can be colored based on volume intensity. This allows the analyst to absorb complex, multi-factor conditions at a glance.

### 4.3 Understanding and Avoiding Common Pitfalls

Several common issues can arise when scripting, particularly for those new to the series-based execution model.

* **Repainting:** An indicator "repaints" if its historical values change as new bars are added. This is a critical flaw, as it can make backtesting results look deceptively profitable. Repainting typically occurs when a script's calculation on a historical bar uses information that would not have been available at that time (i.e., data from the "future"). A common cause is using functions that look ahead in time or improperly referencing data from higher timeframes. Scripts should be carefully designed to ensure all calculations on a given bar use only data from that bar or previous bars.
* **Execution Model Errors:** The most common error is forgetting that variables are re-initialized on every bar. For any cumulative or stateful calculation, variables must be managed properly to carry their value from one bar to the next.
* **Resource Limitations:** Webull's platform, like any other, has computational limits. Excessively complex scripts with numerous calculations, especially those involving loops or intensive statistical functions over very long periods, can become slow to load or may hit execution limits.20 It is good practice to write efficient code, avoiding redundant calculations. For example, if an  
  ema(close, 20) is needed multiple times, calculate it once and store it in a variable rather than calling the function repeatedly.

By mastering these advanced techniques and being mindful of potential pitfalls, developers can create custom indicators that are not only powerful but also reliable and user-friendly.

## Appendix: Quick Function Reference Table

This table provides a high-density, scannable reference for all documented functions, designed for users who need a quick syntax or functional reminder.

| Function | Syntax | Brief Description |
| --- | --- | --- |
| **Mathematical & Statistical** |  |  |
| abs | abs(number) | Calculates the absolute value of a number. |
| acos | acos(x) | Calculates the arccosine in radians. |
| asin | asin(x) | Calculates the arcsine in radians. |
| atan | atan(y, x) | Calculates the arctangent in radians. |
| avg | avg(series1, series2,...) | Calculates the arithmetic average of multiple series. |
| ceil | ceil(number) | Rounds a number up to the next integer. |
| cos | cos(angle) | Calculates the cosine of an angle in radians. |
| dev | dev(source, length) | Calculates the mean absolute deviation. |
| exp | exp(power) | Calculates Euler's number e raised to a power. |
| floor | floor(number) | Rounds a number down to the previous integer. |
| log | log(number) | Calculates the natural logarithm (base e). |
| log10 | log10(number) | Calculates the common logarithm (base 10). |
| max | max(series1, series2,...) | Returns the maximum value among multiple series. |
| min | min(series1, series2,...) | Returns the minimum value among multiple series. |
| pow | pow(base, exponent) | Raises a base to the power of an exponent. |
| round | round(number) | Rounds a number to the nearest integer. |
| sign | sign(number) | Returns the sign of a number (-1, 0, or 1). |
| sin | sin(angle) | Calculates the sine of an angle in radians. |
| sqrt | sqrt(number) | Calculates the square root of a number. |
| stdev | stdev(source, length) | Calculates the standard deviation. |
| sum | sum(source, length) | Calculates the moving sum of a series. |
| tan | tan(angle) | Calculates the tangent of an angle in radians. |
| variance | variance(source, length) | Calculates the statistical variance. |
| **Data Series & Conditional** |  |  |
| barslast | barslast(condition) | Counts the number of bars since a condition was true. |
| change | change(source, length) | Calculates the difference in a series over a period. |
| count | count(condition, length) | Counts the number of true occurrences in a period. |
| cross | cross(series1, series2) | Detects when two series cross in either direction. |
| cross\_down | cross\_down(series1, series2) | Detects when series1 crosses below series2. |
| cross\_up | cross\_up(series1, series2) | Detects when series1 crosses above series2. |
| every | every(condition, length) | Checks if a condition was true for every bar in a period. |
| falling | falling(source, length) | Checks if a series has been decreasing for length bars. |
| hhv | hhv(source, length) | Finds the highest high value in a period. |
| highest | highest(source, length) | Finds the highest value of a series in a period. |
| iff | iff(condition, true\_val, false\_val) | Ternary operator; returns one of two values based on a condition. |
| llv | llv(source, length) | Finds the lowest low value in a period. |
| lowest | lowest(source, length) | Finds the lowest value of a series in a period. |
| na | na | A special value representing "not available". |
| nz | nz(source, replacement) | Replaces na values with a specified value (default 0). |
| rising | rising(source, length) | Checks if a series has been increasing for length bars. |
| valuewhen | valuewhen(cond, src, occ) | Gets the value of a series when a condition was last true. |
| **Time & Date** |  |  |
| dayofmonth | dayofmonth(time) | Returns the day of the month (1-31). |
| dayofweek | dayofweek(time) | Returns the day of the week. |
| hour | hour(time) | Returns the hour of the day (0-23). |
| minute | minute(time) | Returns the minute of the hour (0-59). |
| month | month(time) | Returns the month of the year (1-12). |
| second | second(time) | Returns the second of the minute (0-59). |
| time | time | Returns the Unix timestamp of the bar's open. |
| year | year(time) | Returns the year. |
| **Trend & Moving Averages** |  |  |
| alma | alma(src, len, offset, sigma) | Calculates the Arnaud Legoux Moving Average. |
| dema | dema(source, length) | Calculates the Double Exponential Moving Average. |
| ema | ema(source, length) | Calculates the Exponential Moving Average. |
| hma | hma(source, length) | Calculates the Hull Moving Average. |
| linreg | linreg(src, len, offset) | Calculates a linear regression curve. |
| sma | sma(source, length) | Calculates the Simple Moving Average. |
| smma | smma(source, length) | Calculates the Smoothed Moving Average. |
| swma | swma(source) | Calculates the Symmetrically Weighted Moving Average. |
| tema | tema(source, length) | Calculates the Triple Exponential Moving Average. |
| tr | tr(handle\_na) | Calculates the True Range. |
| vwap | vwap(source) | Calculates the Volume-Weighted Average Price. |
| wma | wma(source, length) | Calculates the Weighted Moving Average. |
| wpr | wpr(length) | Calculates the Williams %R oscillator. |
| **Momentum & Oscillators** |  |  |
| asi | asi() | Calculates the Accumulative Swing Index. |
| bias | bias(source, length) | Calculates the Bias Ratio from a moving average. |
| cci | cci(source, length) | Calculates the Commodity Channel Index. |
| cmo | cmo(source, length) | Calculates the Chande Momentum Oscillator. |
| cr | cr(length) | Calculates the CR energy indicator. |
| dmi | dmi(len, adx\_len) | Calculates the Directional Movement Index (ADX, DI+, DI-). |
| dpo | dpo(source, length) | Calculates the Detrended Price Oscillator. |
| macd | macd(src, fast, slow, sig) | Calculates Moving Average Convergence Divergence. |
| mfi | mfi(source, length) | Calculates the Money Flow Index. |
| mom | mom(source, length) | Calculates the Momentum oscillator. |
| roc | roc(source, length) | Calculates the Rate of Change oscillator. |
| rsi | rsi(source, length) | Calculates the Relative Strength Index. |
| stoch | stoch(src, high, low, len) | Calculates the Stochastic Oscillator (%K). |
| trix | trix(source, length) | Calculates the TRIX momentum oscillator. |
| **Volatility** |  |  |
| atr | atr(length) | Calculates the Average True Range. |
| band | band(src, len, mult) | Calculates Bollinger-style bands. |
| boll | boll(src, len, mult) | Calculates Bollinger Bands. |
| bbiboll | bbiboll(src, len, mult) | Calculates the Bollinger Bands %B indicator. |
| keltner | keltner(src, len, mult) | Calculates Keltner Channels. |
| **Volume** |  |  |
| arbr | arbr(length) | Calculates the AR and BR sentiment indicators. |
| emv | emv(length) | Calculates the Ease of Movement indicator. |
| obv | obv(source) | Calculates On-Balance Volume. |
| pv | pv(length) | Calculates a Price-Volume indicator. |
| **Plotting & Visuals** |  |  |
| bgcolor | bgcolor(color,...) | Sets the background color of the chart. |
| fill | fill(plot1, plot2, color) | Fills the area between two plots. |
| plot | plot(series,...) | Primary function to draw lines, histograms, etc. |
| plotarrow | plotarrow(series,...) | Plots up or down arrows based on series values. |
| plotbar | plotbar(o, h, l, c,...) | Draws custom price bars. |
| plotcandle | plotcandle(o, h, l, c,...) | Draws custom candlesticks. |
| plotchar | plotchar(series,...) | Plots a text character when a condition is true. |
| plotshape | plotshape(series,...) | Plots a shape when a condition is true. |
| drawlabel | drawlabel(x, y, text,...) | Creates a persistent text label at a coordinate. |
| drawtext | drawtext(x, y, text,...) | Creates persistent plain text at a coordinate. |

#### Works cited

1. Webull Script Editor Syntax (Functions)
2. Quickstart guide - Pine Script - TradingView, accessed September 17, 2025, <https://www.tradingview.com/pine-script-docs/v4/quickstart-guide/>
3. Custom — Indicatori e strategie - TradingView, accessed September 17, 2025, <https://it.tradingview.com/scripts/custom/>
4. Learning Pine Script™ Roadmap - PineCoders, accessed September 17, 2025, <https://www.pinecoders.com/learning_pine_roadmap/>
5. Pine Script® User Manual - TradingView, accessed September 17, 2025, <https://www.tradingview.com/pine-script-docs/>
6. Exponential Moving Average (EMA) — Indicators and Strategies - TradingView, accessed September 17, 2025, <https://www.tradingview.com/scripts/ema/>
7. Pine Script 101: The Simplest Way to Code Your First Trading Strategy - Medium, accessed September 17, 2025, <https://medium.com/@betashorts1998/pine-script-101-the-simplest-way-to-code-your-first-trading-strategy-ef8cc2905e19>
8. Language / Execution model - TradingView, accessed September 17, 2025, <https://www.tradingview.com/pine-script-docs/language/execution-model/>
9. Floor Pivot 2025 | PDF | Computing - Scribd, accessed September 17, 2025, <https://www.scribd.com/document/836343871/Floor-Pivot-2025>
10. A Comprehensive Guide to Pine Script for TradingView - PineConnector, accessed September 17, 2025, <https://www.pineconnector.com/blogs/pico-blog/a-comprehensive-guide-to-pine-script-for-tradingview>
11. Visuals / Backgrounds - Pine Script - TradingView, accessed September 17, 2025, <https://www.tradingview.com/pine-script-docs/visuals/backgrounds/>
12. Drawing trade data labels in Pine Script, accessed September 17, 2025, <https://courses.theartoftrading.com/pages/drawing-trade-data-labels-in-pine-script>
13. acos - Script command - Ansys Optics, accessed September 17, 2025, <https://optics.ansys.com/hc/en-us/articles/360034925453-acos-Script-command>
14. asin - Script command – Ansys Optics, accessed September 17, 2025, <https://optics.ansys.com/hc/en-us/articles/360034405574-asin-Script-command>
15. ArcTan (ATan) - CQG IC Help, accessed September 17, 2025, <https://help.cqg.com/cqgic/25/Documents/arctanatan.htm>
16. Pine Script SMA (Simple Moving Average) - Complete TradingView Guide - Offline Pixel, accessed September 17, 2025, <https://offline-pixel.github.io/pinescript-strategies/pine-script-SMA.html>
17. ceil - Script command - Ansys Optics, accessed September 17, 2025, <https://optics.ansys.com/hc/en-us/articles/360034406174-ceil-Script-command>
18. Master TradingView Pine Script: Create Custom Trading Indicators - WunderTrading, accessed September 17, 2025, <https://wundertrading.com/journal/en/learn/article/mastering-tradingview-scripts>
19. Pine Script Historical Volatility - Complete TradingView Guide - Offline Pixel, accessed September 17, 2025, <https://offline-pixel.github.io/pinescript-strategies/pine-script-HistoricalVolatility.html>
20. TECHNICAL ANALYSIS - Math MUNI, accessed September 17, 2025, <https://www.math.muni.cz/~mrezac/Bloomberg/Technical-Analysis-Bloomberg.pdf>
21. EVERYTHING you need to know about FOR LOOPS in Pine Script - YouTube, accessed September 17, 2025, <https://www.youtube.com/watch?v=Phm2NTLiE4M>
22. Top 5 Pine Script Strategies ( September 2025 ) | by offline-pixel - Medium, accessed September 17, 2025, <https://dranolia.medium.com/top-5-pine-script-strategies-that-everyone-should-know-eb27cc72afc3>
23. Cross Up Cross Down - RTMath, accessed September 17, 2025, <https://rtmath.net/assets/docs/finanalysis/html/8b70f2ac-10e2-49e1-9397-fe5cb0075fb9.htm>
24. PineScript and MACD Crossover: A Comprehensive Coding Guide - Unofficed, accessed September 17, 2025, <https://unofficed.com/courses/mastering-market-indicators/lessons/pinescript-and-macd-crossover/>
25. Language / Type system - TradingView, accessed September 17, 2025, <https://www.tradingview.com/pine-script-docs/v4/language/type-system/>
26. PINE Concepts Time | PDF | Computer Programming - Scribd, accessed September 17, 2025, <https://www.scribd.com/document/812147419/PINE-concepts-Time>
27. Dayofweek.friday unexpected results : r/pinescript - Reddit, accessed September 17, 2025, <https://www.reddit.com/r/pinescript/comments/10aabza/dayofweekfriday_unexpected_results/>
28. Technical Analysis - A Beginner's Guide - Corporate Finance Institute, accessed September 17, 2025, <https://corporatefinanceinstitute.com/resources/career-map/sell-side/capital-markets/technical-analysis/>
29. Understanding Technical Indicators: Types, Uses, and Key Examples - Investopedia, accessed September 17, 2025, <https://www.investopedia.com/terms/t/technicalindicator.asp>
30. Mastering TradingView Pine Script & AI-Powered Trading | Sergey Buzz - Skillshare, accessed September 17, 2025, <https://www.skillshare.com/en/classes/mastering-tradingview-pine-script-and-ai-powered-trading/441734558>
31. Bias — Indicators and Strategies — TradingView — India, accessed September 17, 2025, <https://in.tradingview.com/scripts/bias/>
32. Demystifying the Commodity Channel Index (CCI) on TradingView Platform - Eightcap Labs, accessed September 17, 2025, <https://www.eightcap.com/labs/demystifying-the-commodity-channel-index-cci-on-tradingview-platform/>
33. 1-Year Median ₹ Turnover (Cr) — Indicator by vikrantk\_13 - TradingView, accessed September 17, 2025, <https://www.tradingview.com/script/k4zWC8aX-1-Year-Median-Turnover-Cr/>
34. DMI - thinkorswim Learning Center, accessed September 17, 2025, <https://toslc.thinkorswim.com/center/reference/Tech-Indicators/studies-library/C-D/DMI>
35. alessioricco/PineScript-Custom-Indicators-TradingView - GitHub, accessed September 17, 2025, <https://github.com/alessioricco/PineScript-Custom-Indicators-TradingView>
36. This is a pine script for a bollinger band strategy for use in Trading View. - GitHub, accessed September 17, 2025, <https://github.com/TravisLeeRichardson/Bollinger-Bands>
37. Bollinger Bands Strategy - AlgoTest's Product Documentation, accessed September 17, 2025, <https://docs.algotest.in/signals/pinescripts/bb_bands_strategy/>
38. How to get the SPREAD in Pine Script. - YouTube, accessed September 17, 2025, <https://www.youtube.com/watch?v=aUZVjHgH444>
39. EMV Scripting Service (CSNBESC and CSNEESC) - IBM, accessed September 17, 2025, <https://www.ibm.com/docs/SSLTBW_2.4.0/com.ibm.zos.v2r4.csfb400/EMVScripting.htm>
40. Visuals / Fills - TradingView, accessed September 17, 2025, <https://www.tradingview.com/pine-script-docs/visuals/fills/>