Time series

Much of the power of Pine Script™ stems from the fact that it is designed to process *time series* efficiently. Time series are not a form or a type; they are the fundamental structure Pine Script™ uses to store the successive values of a variable over time, where each value is tethered to a point in time. Since charts are composed of bars, each representing a particular point in time, time series are the ideal data structure to work with values that may change with time.

The notion of time series is intimately linked to Pine Script™'s execution model and type system concepts. Understanding all three is key to making the most of the power of Pine Script™.

Take the built-in open variable, which contains the "open" price of each bar in the dataset, the *dataset* being all the bars on any given chart. If your script is running on a 5min chart, then each value in the open time series is the "open" price of the consecutive 5min chart bars. When your script refers to open, it is referring to the "open" price of the bar the script is executing on. To refer to past values in a time series, we use the [] history-referencing operator. When a script is executing on a given bar, `open[1]` refers to the value of the open time series on the previous bar.

While time series may remind programmers of arrays, they are totally different. Pine Script™ does use an array data structure, but it is a completely different concept than a time series.

Time series in Pine Script™, combined with its special type of runtime engine and built-in functions, are what makes it easy to compute the cumulative total of close values without using a for loop, with only `ta.cum(close)`. This is possible because although `ta.cum(close)` appears rather static in a script, it is in fact executed on each bar, so its value becomes increasingly larger as the close value of each new bar is added to it. When the script reaches the rightmost bar of the chart, `ta.cum(close)` returns the sum of the close value from all bars on the chart.

Similarly, the mean of the difference between the last 14 high and low values can be expressed as `ta.sma(high - low, 14)`, or the distance in bars since the last time the chart made five consecutive higher highs as `barssince(rising(high, 5))`.

Even the result of function calls on successive bars leaves a trace of values in a time series that can be referenced using the [] history-referencing operator. This can be useful, for example, when testing the close of the current bar for a breach of the highest high in the last 10 bars, but excluding the current bar, which we could write as `breach = close > highest(close, 10)[1]`. The same statement could also be written as `breach = close > highest(close[1], 10)`.

The same looping logic on all bars is applied to function calls such as `plot(open)` which will repeat on each bar, successively plotting on the chart the value of open for each bar.

Do not confuse "time series" with the "series" form. The *time series* concept explains how consecutive values of variables are stored in Pine Script™; the "series" form denotes variables whose values can change bar to bar. Consider, for example, the timeframe.period built-in variable which is of form "simple" and type "string", so "simple string". The "simple" form entails that the variable's value is known on bar zero (the first bar where the script executes) and will not change during the script's execution on all the chart's bars. The variable's value is the chart's timeframe in string format, so `"D"` for a 1D chart, for example. Even though its value cannot change during the script, it would be syntactically correct in Pine Script™ (though not very useful) to refer to its value 10 bars ago using `timeframe.period[10]`. This is possible because the successive values of timeframe.period for each bar are stored in a time series, even though all the values in that particular time series are similar. Note, however, that when the [] operator is used to access past values of a variable, it yields a result of "series" form, even though the variable without an offset is of another form, such as "simple" in the case of timeframe.period.

When you grasp how time series can be efficiently handled using Pine Script™'s syntax and its execution model, you can define complex calculations using little code.

Execution model

Script structure

Options v: v5