Style guide

Introduction

This style guide provides recommendations on how to name variables and organize your Pine scripts in a standard way that works well. Scripts that follow our best practices will be easier to read, understand and maintain.

You can see scripts using these guidelines published from the TradingView and PineCoders accounts on the platform.

Naming Conventions

We recommend the use of:

- `camelCase` for all identifiers, i.e., variable or function names: `ma`, `maFast`, `maLengthInput`, `maColor`, `roundedOHLC()`, `pivotHi()`.
- All caps `SNAKE_CASE` for constants: `BULL_COLOR`, `BEAR_COLOR`, `MAX_LOOKBACK`.
- The use of qualifying suffixes when it provides valuable clues about the type or provenance of a variable: `maShowInput`, `bearColor`, `bearColorInput`, `volumesArray`, `maPlotID`, `resultsTable`, `levelsColorArray`.

Script organization

The Pine Script™ compiler is quite forgiving of the positioning of specific statements or the version compiler annotation in the script. While other arrangements are syntactically correct, this is how we recommend organizing scripts:

```
<license>
<version>
<declaration_statement>
<import_statements>
<constant_declarations>
<inputs>
<function_declarations>
<calculations>
<strategy_calls>
<visuals>
<alerts>
```

<license>

If you publish your open-source scripts publicly on TradingView (scripts can also be published privately), your open-source code is by default protected by the Mozilla license. You may choose any other license you prefer.

The reuse of code from those scripts is governed by our House Rules on Script Publishing which preempt the author's license.

The standard license comments appearing at the beginning of scripts are:

```
// This source code is subject to the terms of the Mozilla Public License 2.0 at https:
// © username
```

<version>

This is the compiler annotation defining the version of Pine Script™ the script will use. If none is present, v1 is used. For v5, use:

```
//@version=5
```

<declaration_statement>

This is the mandatory declaration statement which defines the type of your script. It must be a call to either indicator(), strategy(), or library().

<import_statements>

If your script uses one or more Pine Script™ libraries, your import statements belong here.

<constant_declarations>

While there is a "constant" form in Pine Script™, there is no formal "constant" type. We nonetheless use "constant" to denote variables of any type meeting these criteria:

- They are initialized using a literal (e.g., `100` or `"AAPL"`) or a built-in of "const" form (e.g., `color.green`).
- Their value does not change during the script's execution, meaning their value is never redefined using `:=`.

We use `SNAKE_CASE` to name these variables and group their declaration near the top of the script. For example:

```
// ———— Constants
int     MS_IN_MIN   = 60 * 1000
int     MS_IN_HOUR  = MS_IN_MIN  * 60
int     MS_IN_DAY   = MS_IN_HOUR * 24

color   GRAY        = #808080ff
color   LIME        = #00FF00ff
color   MAROON      = #800000ff
color   ORANGE      = #FF8000ff
color   PINK        = #FF0080ff
color   TEAL        = #008080ff
color   BG_DIV      = color.new(ORANGE, 90)
color   BG_RESETS   = color.new(GRAY, 90)

string  RST1        = "No reset; cumulate since the beginning of the chart"
string  RST2        = "On a stepped higher timeframe (HTF)"
string  RST3        = "On a fixed HTF"
string  RST4        = "At a fixed time"
string  RST5        = "At the beginning of the regular session"
string  RST6        = "At the first visible chart bar"
string  RST7        = "Fixed rolling period"

string  LTF1        = "Least precise, covering many chart bars"
string  LTF2        = "Less precise, covering some chart bars"
string  LTF3        = "More precise, covering less chart bars"
string  LTF4        = "Most precise, 1min intrabars"

string  TT_TOTVOL     = "The 'Bodies' value is the transparency of the total volume can
string  TT_RST_HTF    = "This value is used when '" + RST3 +"' is selected."
string  TT_RST_TIME   = "These values are used when '" + RST4 +"' is selected.
  A reset will occur when the time is greater or equal to the bar's open time, and less
string  TT_RST_PERIOD = "This value is used when '" + RST7 +"' is selected."
```

In this example:

- The `RST*` and `LTF*` constants will be used as tuple elements in the `options` argument of `input.*()` calls.
- The `TT_*` constants will be used as `tooltip` arguments in `input.*()` calls. Note how we use a line continuation for long string literals.
- We do not use var to initialize constants. The Pine Script™ runtime is optimized to handle declarations on each bar, but using var to initialize a variable only the first time it is declared incurs a minor penalty on script performance because of the maintenance that var variables require on further bars.

Note that:

- Literals used in more than one place in a script should always be declared as a constant. Using the constant rather than the literal makes it more readable if it is given a meaningful name, and the practice makes code easier to maintain. Even though the quantity of milliseconds in a day is unlikely to change in the future, `MS_IN_DAY` is more meaningful than `1000 * 60 * 60 * 24`.
- Constants only used in the local block of a function or if, while, etc., statement for example, can be declared in that local block.

<inputs>

It is **much** easier to read scripts when all their inputs are in the same code section. Placing that section at the

beginning of the script also reflects how they are processed at runtime, i.e., before the rest of the script is executed.

Suffixing input variable names with `input` makes them more readily identifiable when they are used later in the script: `maLengthInput`, `bearColorInput`, `showAvgInput`, etc.

```
// ———— Inputs
string  resetInput              = input.string(RST2,        "CVD Resets",
string  fixedTfInput            = input.timeframe("D",      "  Fixed HTF:  ",
int     hourInput               = input.int(9,              "  Fixed time hour:  ",
int     minuteInput             = input.int(30,             "minute",
int     fixedPeriodInput        = input.int(20,             "  Fixed period:  ",
string  ltfModeInput            = input.string(LTF3,        "Intrabar precision",
```

<function_declarations>

All user-defined functions must be defined in the script's global scope; nested function definitions are not allowed in Pine Script™.

Optimal function design should minimize the use of global variables in the function's scope, as they undermine function portability. When it can't be avoided, those functions must follow the global variable declarations in the code, which entails they can't always be placed in the <function_declarations> section. Such dependencies on global variables should ideally be documented in the function's comments.

It will also help readers if you document the function's objective, parameters and result. The same syntax used in libraries can be used to document your functions. This can make it easier to port your functions to a library should you ever decide to do so.

```
//@version=5
indicator("<function_declarations>", "", true)

string SIZE_LARGE  = "Large"
string SIZE_NORMAL = "Normal"
string SIZE_SMALL  = "Small"

string sizeInput = input.string(SIZE_NORMAL, "Size", options = [SIZE_LARGE, SIZE_NORMAI

// @function        Used to produce an argument for the `size` parameter in built-in fu
// @param userSize  (simple string) User-selected size.
// @returns         One of the `size.*` built-in constants.
// Dependencies:    SIZE_LARGE, SIZE_NORMAL, SIZE_SMALL
getSize(simple string userSize) =>
    result =
      switch userSize
        SIZE_LARGE  => size.large
        SIZE_NORMAL => size.normal
        SIZE_SMALL  => size.small
        => size.auto

if ta.rising(close, 3)
    label.new(bar_index, na, yloc = yloc.abovebar, style = label.style_arrowup, size =
```

<calculations>

This is where the script's core calculations and logic should be placed. Code can be easier to read when variable declarations are placed near the code segment using the variables. Some programmers prefer to place all their non-constant variable declarations at the beginning of this section, which is not always possible for all variables, as some

may require some calculations to have been executed before their declaration.

<strategy_calls>

Strategies are easier to read when strategy calls are grouped in the same section of the script.

<visuals>

This section should ideally include all the statements producing the script's visuals, whether they be plots, drawings, background colors, candle-plotting, etc. See the Pine Script™ User Manual's section on here for more information on how the relative depth of visuals is determined.

<alerts>

Alert code will usually require the script's calculations to have executed before it, so it makes sense to put it at the end of the script.

Spacing

A space should be used on both sides of all operators, except unary operators (`-1`). A space is also recommended after all commas and when using named function arguments, as in `plot(series = close)`

```
int a = close > open ? 1 : -1
var int newLen = 2
newLen := min(20, newlen + 1)
float a = -b
float c = d > e ? d - e : d
int index = bar_index % 2 == 0 ? 1 : 2
plot(close, color = color.red)
```

Line wrapping

Line wrapping can make long lines easier to read. Line wraps are defined by using an indentation level that is not a multiple of four, as four spaces or a tab are used to define local blocks. Here we use two spaces:

```
plot(
  series = close,
  title = "Close",
  color = color.blue,
  show_last = 10
  )
```

Vertical alignment

Vertical alignment using tabs or spaces can be useful in code sections containing many similar lines such as constant declarations or inputs. They can make mass edits much easier using the Pine Script™ Editor's multi-cursor feature (ctrl + alt + ↑/↓):

```
// Colors used as defaults in inputs.
color COLOR_AQUA  = #0080FFff
color COLOR_BLACK = #000000ff
color COLOR_BLUE  = #013BCAff
color COLOR_CORAL = #FF8080ff
color COLOR_GOLD  = #CCCC00ff
```

Explicit typing

Including the type of variables when declaring them is not required and is usually overkill for small scripts; we do not systematically use it. It can be useful to make the type of a function's result clearer, and to distinguish a variable's declaration (using `=`) from its reassignments (using `:=`). Using explicit typing can also make it easier for readers to find their way in larger scripts.

Writing scripts

Debugging