Operators

Overview - Operators

In this section, we will learn about **operators**. By the end of this section you will be able to apply mathematical principles to any variable you come across in Pine Script. We'll be covering the following topics:

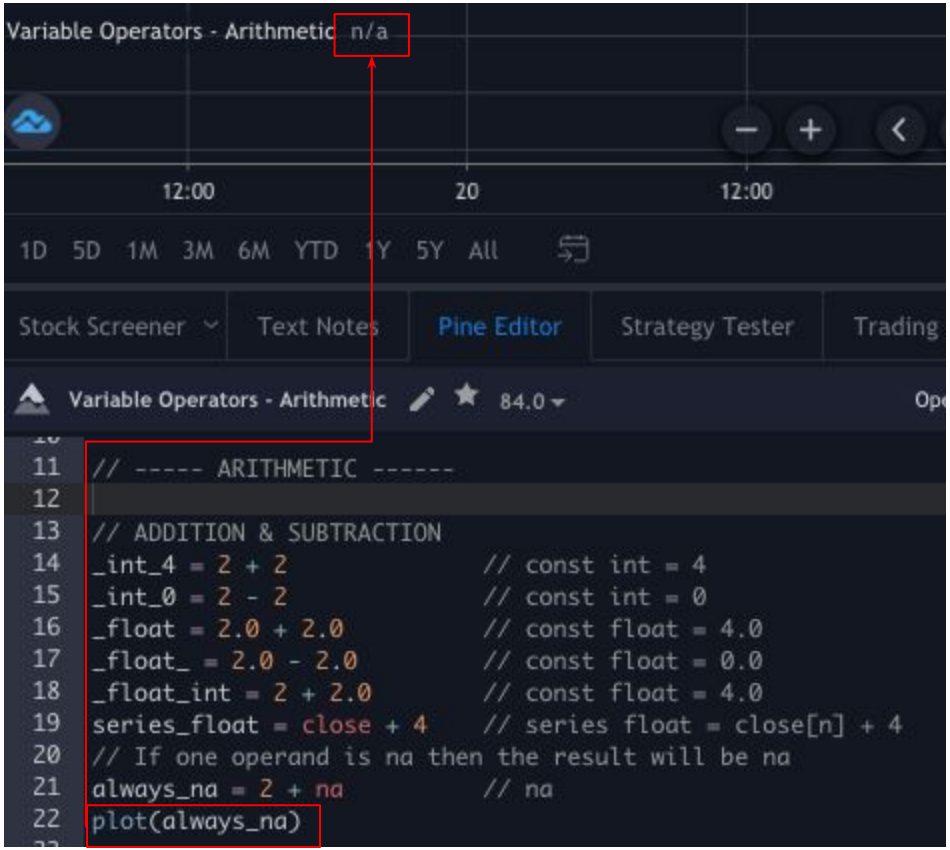
- Arithmetic
- Comparisons
- Logic

Arithmetic Operations

There are ***five*** arithmetic operators in Pine Script:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulo (%)

Operators		Unary	Binary	Works With Variables:
+	Addition	✓	✓	• All Forms (literal, const, input, simple, series) • Numbers (int, float) • na • string
-	Subtraction	✓	✓	• All Forms (literal, const, input, simple, series) • Numbers (int, float) • na
*	Multiplication	✗	✓	• All Forms (literal, const, input, simple, series) • Numbers (int, float) • na
/	Division	✗	✓	• All Forms (literal, const, input, simple, series) • Numbers (int, float) • na
%	Modulo	✗	✓	• All Forms (literal, const, input, simple, series) • Numbers (int, float) • na



	na	series	float	int
na	na	na	na	na
series	na	series	series	series
float	na	series	float	float
int	na	series	float	int

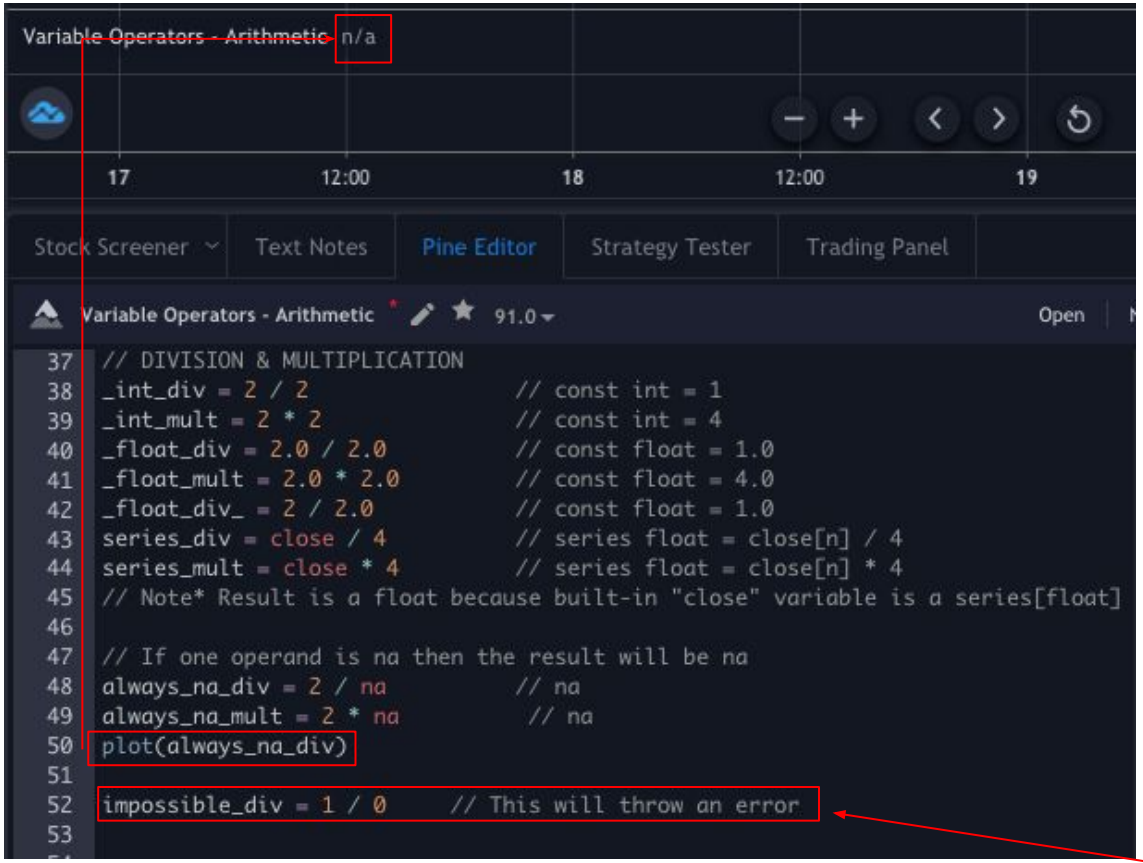
Arithmetic Examples - Addition & Subtraction

Notice the (-) operate negates the number, where the (+) has no effect.

Arithmetic Examples - Addition & Subtraction - Unary Binary

```
23
24  // String concatenation. Only works with the "+" operator.
25  string_1 = "current "
26  string_2 = "range"
27  concatenation = string_1 + string_2    // "current range" = "current " + "range"
28
```

+ Operator can also combine strings. (concatenation)



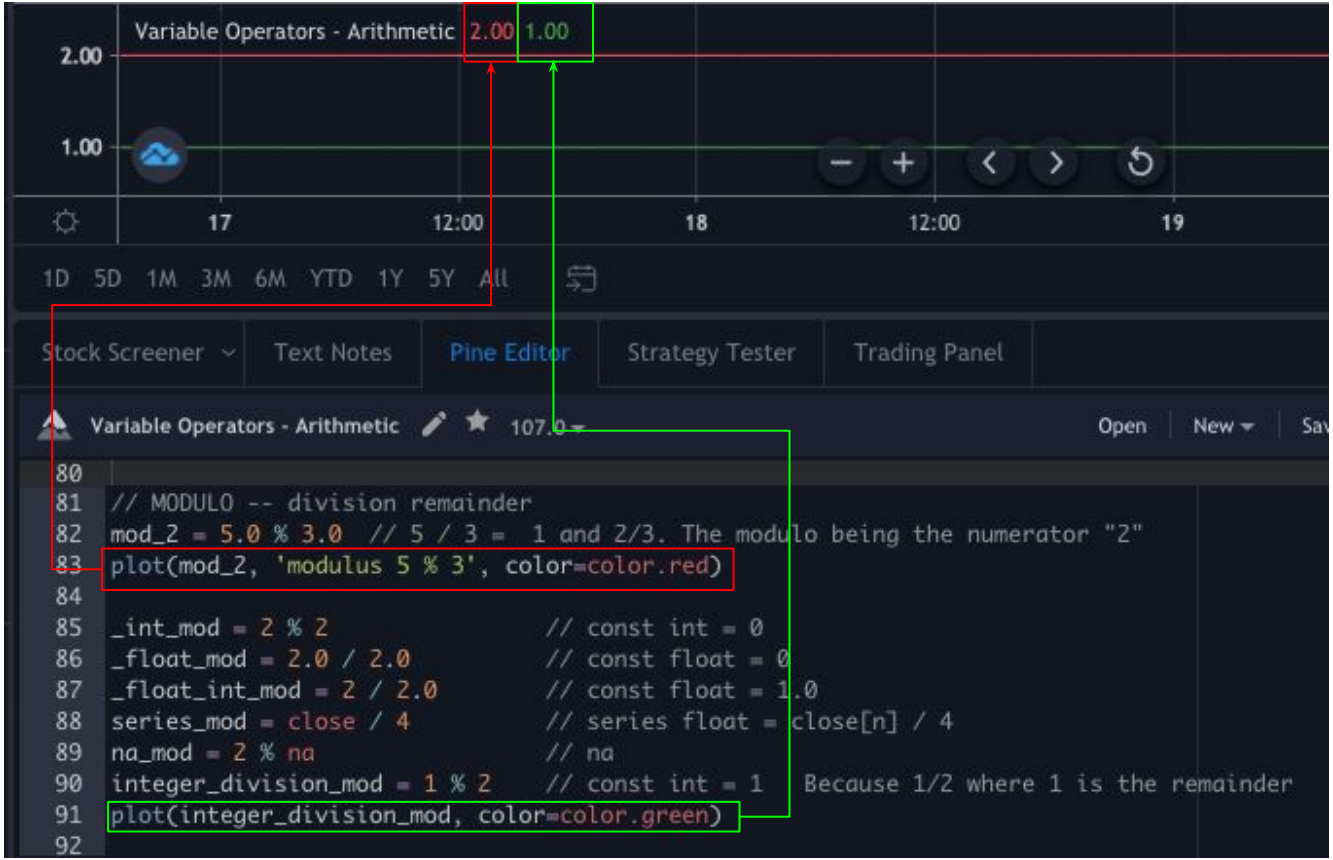
	na	series	float	int
na	na	na	na	na
series	na	series	series	series
float	na	series	float	float
int	na	series	float	int

Can’t divide by zero

Arithmetic Examples - Division & Multiplication

Be careful using only integers with division.
In this case where an int is returned, the remainder will be lost.

```
55  bad_fraction = 1 / 2           // This does not equal 0.5 because these are both ints so the result is an integer
56  good_fraction = 1.0 / 2        // Automatically casts to float
57  better_fraction = 1.0 / 2.0
```



Arithmetic Examples - Modulo (Division Remainder)

Comparison Operations

There are ***six*** comparison operators in Pine Script:

- Less Than (<)
- Less Than or Equal To (<=)
- Not Equal (!=)
- Equal (==)
- Greater Than (>)
- Greater Than or Equal To (>=)

Operators		Unary	Binary	Works With Variables:
<	Less Than	✗	✓	• All Forms (literal, const, input, simple, series) • Numbers (int, float) • na
<=	Less Than or Equal To	✗	✓	• All Forms (literal, const, input, simple, series) • Numbers (int, float) • na
!=	Not Equal	✗	✓	• All Forms (literal, const, input, simple, series) • Numbers (int, float) • na • string, line, label
==	Equal	✗	✓	• All Forms (literal, const, input, simple, series) • Numbers (int, float) • na • string, line, label
>	Greater Than	✗	✓	• All Forms (literal, const, input, simple, series) • Numbers (int, float) • na
>=	Greater Than or Equal To	✗	✓	• All Forms (literal, const, input, simple, series) • Numbers (int, float) • na

We can compare ints and floats with all six operators.

```
14  // numbers
15  lt_int       = 2 < 2          // const bool = false
16  lt_na        = 2 < na         // const bool = false
17  lte_int      = 2 <= 2         // const bool = true
18  e_int        = 2 == 2         // const bool = true
19  e_int_float  = 2 == 2.0       // const bool = true
20  ne_int       = 2 != 3         // const bool = true
21  gt_int       = 2 > -1          // const bool = true
22  gt_na        = 2 > na         // const bool = false
23  gte_int      = 2 >= 2         // const bool = true
24
```

**The comments to the right of each expression represents the returned variable.

Note that Pine Script doesn't differentiate between floats and integers when using comparison operators. In other languages, you might expect a different result.

Comparison Examples - numbers

We can only use the Equal To (==) and Not Equal To (!=) operators when comparing strings.

```
25  // strings
26  lt_string       = "" < ""           // ERROR
27  lte_string      = "" <= ""          // ERROR
28  e_string        = "a" == "a"        // const bool = true
29  e_string_na     = "" == na          // const bool = true
30  e_string_na2    = " " == na         // const bool = false
31  ne_string       = "" != " "         // const bool = true
32  gt_string       = "a" > "b"         // ERROR
33  gte_string      = "a" >= "b"        // ERROR
34
```

**The comments to the right of each expression represents the returned variable.

Note that an empty string (“”) and na are considered to be equal here. A string with a space is not considered empty.

Comparison Examples - strings

We can compare series (floats and ints) with all six operators.
Other series[type] are possible as well, but limited to == and !=

```
36  // series
37  lt_series       = close < 2          // series bool = [true, false]
38  lt_series_na    = close < na         // series bool = [true, false]
39  lte_series      = low <= high        // series bool = [true, true]
40  e_series        = close == 2         // series bool = [true, false]
41  ne_series       = close != close     // series bool = [false, false]
42  gt_series       = high > 1.0         // series bool = [true, false]
43  gte_series      = high >= low        // series bool = [true, false]
```

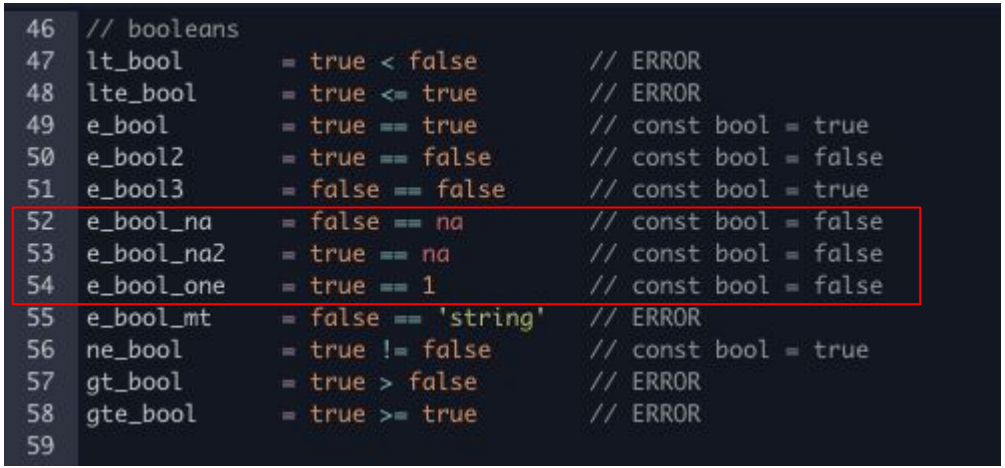
**The comments to the right of each expression represents the returned variable. In our example, the return value is non deterministic because we don't know the values of ohlc

Note: Comparing against na values still returns a series filled with booleans true or false.

Series comparison operations will always return a series[bool]

Comparison Examples - series

We can only use the Equal To (==) and Not Equal To (!=) operators when comparing booleans.



```
// booleans
lt_bool       = true < false        // ERROR
lte_bool      = true <= true        // ERROR
e_bool        = true == true        // const bool = true
e_bool2       = true == false       // const bool = false
e_bool3       = false == false      // const bool = true
e_bool_na     = false == na         // const bool = false
e_bool_na2    = true == na          // const bool = false
e_bool_one    = true == 1           // const bool = false
e_bool_mt     = false == 'string'   // ERROR
ne_bool       = true != false       // const bool = true
gt_bool       = true > false        // ERROR
gte_bool      = true >= true        // ERROR
```

**The comments to the right of each expression represents the returned variable.

Note: In some programming languages, boolean values can be interchanged with 0 and 1 (false, true). This is not the case with Pine.

Comparison Examples - booleans

We can only use the Equal To (==) and Not Equal To (!=) operators when comparing colors.

```
63  // color
64  e_color         = color.green == color.green  // const bool = true
65  e_hex           = #4CAF50 != #4CAF50          // const bool = false
66  e_hex_color     = color.green == #4CAF50      // const bool = true
67  e_comp          = e_color == e_hex            // const bool = true
68  e_hex_2         = #4caf50 == #4CAF50          // const bool = false
69
```

**The comments to the right of each expression represents the returned variable.

Note: The hex values for colors are NOT case-sensitive; so both operands are identical, valid colors; but strangely they don't equally compare.

Comparison Examples - color

We can only use the Equal To (==) and Not Equal To (!=) operators when comparing lines & labels.

```
// label & line
var e_line_1 = line.new(0, 0, 0, 0)     // These two lines are not the same
var e_line_2 = line.new(0, 0, 0, 0)     // at least when being compared with ==
e_line_bool  = e_line_1 == e_line_2     // series[bool] or [false] = series[line] == series[line]
// Same for label.
```

Note: Lines and Labels are limited to only 50 per chart, so each is treated as unique; hence why a comparison between two identical lines turns out to be false.

Comparison Examples - lines & labels

Logical Operations

There are ***three*** logical operators in Pine Script:

- not
- and
- or

Operators		Unary	Binary	Works With Variables:
not	Negation	✓	✗	• All Forms (literal, const, input, simple, series) • Numbers (int, float) • na • booleans
and	Logical Conjunction	✗	✓	• All Forms (literal, const, input, simple, series) • Numbers (int, float) • na • booleans
or	Logical Disjunction	✗	✓	• All Forms (literal, const, input, simple, series) • Numbers (int, float) • na • booleans

We can use all logical operators with booleans.

```
// booleans
not_bool        = not true                // const bool = false
not_bool2       = not false               // const bool = true
and_bool        = true and true           // const bool = true
and_bool2       = true and false          // const bool = false
and_bool3       = false and false         // const bool = false
andnot_bool     = false and not false     // const bool = false
andnot_bool2    = not false and not false // const bool = true
or_bool         = true or true            // const bool = true
or_bool2        = true or false           // const bool = true
or_bool3        = false or false          // const bool = false
ornot_bool      = false or not false      // const bool = true
ornot_bool2     = not false or not false  // const bool = true

```

**The comments to the right of each expression represents the returned variable.

Logical Examples - booleans

We can use all logical operators with numbers.

```
26  // numbers
27  // Numbers are logically true even if they are negative
28  not_int         = not 2                     // const bool = false
29  and_if          = 2 and 2.0                 // const bool = true
30  or_int          = 2 or 2                    // const bool = true
31  or_int2         = -2 and 0                  // const bool = true
32  lt_na           = not na                    // const bool = true
```

**The comments to the right of each expression represents the returned variable.

Note: Numbers are logically true, even if they are 0 or negative.

Logical Examples - numbers

We can use all logical operators with series.

```
35  // series
36  not_series      = not close                 // series bool = [true, false]
37  and_series      = close and open            // series bool = [true, false]
38  or_series       = low or high               // series bool = [true, false]
39  and_series2     = close and true            // series bool = [true, false]
40  and_series3     = close and na              // series bool = [false, false]
41
```

**The comments to the right of each expression represents the returned variable.

Note: As with arithmetic and comparison operators; any operator used with a series will return a series.

Logical Examples - series

Recap

- Arithmetic Operations
  - Addition (+), Subtraction (-), Multiplication (*), Division (/), Modulo (%)
  - “+” can also concatenate strings
  - Can’t divide by 0
  - Be careful dividing integers
  - Arithmetic with “na” returns “na”
- Comparison Operations
  - Works with numbers, strings, booleans, line, label, na; in all forms.
  - Less Than (<), Less Than or Equal To (<=), Not Equal (!=), Equal (==), Greater Than (>), Greater Than or Equal To (>=)
  - Comparisons always return series[bool] or bool result.
  - Comparisons with series always return series[bool]
  - Floats and ints compare true (2 == 2.0)
  - Empty strings and na compare true (“” == na)
- Logical Operations
  - not, and, or
  - Works with numbers, booleans, na, in all forms
  - “not” is unary, and can be used with “and/or”
  - numbers are logically true, even “0” or negatives.