Error messages

The if statement is too long

This error occurs when the indented code inside an if statement is too large for the compiler. Because of how the compiler works, you won't receive a message telling you exactly how many lines of code you are over the limit. The only solution now is to break up your if statement into smaller parts (functions or smaller if statements). The example below shows a reasonably lengthy if statement; theoretically, this would throw `line 4: if statement is too long`.

```
//@version=5
indicator("My script")

var e = 0
if barstate.islast
    a = 1
    b = 2
    c = 3
    d = 4
    e := a + b + c + d

plot(e)
```

To fix this code, you could move these lines into their own function:

```
//@version=5
indicator("My script")
```

```
doSomeWork() =>
    a = 1
    b = 2
    c = 3
    d = 4

    result = a + b + c + d

if barstate.islast
    e := doSomeWork()

plot(e)
```

Script requesting too many securities

The maximum number of securities in script is limited to 40. If you declare a variable as a `request.security` function call and then use that variable as input for other variables and calculations, it will not result in multiple `request.security` calls. But if you will declare a function that calls `request.security` — every call to this function will count as a `request.security` call.

It is not easy to say how many securities will be called looking at the source code. Following example have exactly 3 calls to `request.security` after compilation:

```
//@version=5
indicator("Securities count")
a = request.security(syminfo.tickerid, '42', close)  // (1) first unique security call
b = request.security(syminfo.tickerid, '42', close)  // same call as above, will not pr

plot(a)
plot(a + 2)
plot(b)

sym(p) =>  // no security call on this line
    request.security(syminfo.tickerid, p, close)
plot(sym('D'))  // (2) one indirect call to security
plot(sym('W'))  // (3) another indirect call to security

request.security(syminfo.tickerid, timeframe.period, open)  // result of this line is n
```

Script could not be translated from: null

```
study($)
```

Usually this error occurs in version 1 pine scripts, and means that code is incorrect. Pine Script™ of version 2 (and higher) is better at explaining errors of this kind. So you can try to switch to version 2 by adding a special attribute in the first line. You'll get `line 2: no viable alternative at character '$'`

```
// @version=2
study($)
```

line 2: no viable alternative at character '$'

This error message gives a hint on what is wrong. `$` stands in place of string with script title. For example:

```
// @version=2
study("title")
```

Mismatched input <...> expecting <???>

Same as `no viable alternative`, but it is known what should be at that place. Example:

```
//@version=5
indicator("My Script")
    plot(1)
```

`line 3: mismatched input 'plot' expecting 'end of line without line continuation'`

To fix this you should start line with `plot` on a new line without an indent:

```
//@version=5
indicator("My Script")
plot(1)
```

Loop is too long (> 500 ms)

We limit the computation time of loop on every historical bar and realtime tick to protect our servers from infinite or very long loops. This limit also fail-fast indicators that will take too long to compute. For example, if you'll have 5000 bars, and indicator takes 500 milliseconds to compute on each of bars, it would have result in more than 16 minutes of loading.

```
//@version=5
indicator("Loop is too long", max_bars_back = 101)
s = 0
for i = 1 to 1e3  // to make it longer
    for j = 0 to 100
        if timestamp(2017, 02, 23, 00, 00) <= time[j] and time[j] < timestamp(2017, 02,
            s := s + 1
plot(s)
```

It might be possible to optimize algorithm to overcome this error. In this case, algorithm may be optimized like this:

```
//@version=5
indicator("Loop is too long", max_bars_back = 101)
bar_back_at(t) =>
    i = 0
    step = 51
    for j = 1 to 100
        if i < 0
            i := 0
            break
        if step == 0
            break
        if time[i] >= t
            i := i + step
            i
        else
            i := i - step
            i
        step := step / 2
        step
    i

s = 0
for i = 1 to 1e3  // to make it longer
    s := s - bar_back_at(timestamp(2017, 02, 23, 23, 59)) +
         bar_back_at(timestamp(2017, 02, 23, 00, 00))
    s
plot(s)
```

Script has too many local variables

This error appears if the script is too large to be compiled. A statement `var=expression` creates a local variable for `var`. Apart from this, it is important to note, that auxiliary variables can be implicitly created during the process of a script compilation. The limit applies to variables created both explicitly and implicitly. The limitation of 1000 variables is applied to each function individually. In fact, the code placed in a *global* scope of a script also implicitly wrapped up into the main function and the limit of 1000 variables becomes applicable to it. There are few refactorings you can try to avoid this issue:

```
var1 = expr1
var2 = expr2
var3 = var1 + var2
```

can be converted into:

```
var3 = expr1 + expr2
```

Pine Script™ cannot determine the referencing length of a series. Try using max_bars_back in the indicator or strategy function

The error appears in cases where Pine Script™ wrongly autodetects the required maximum length of series used in a script. This happens when a script's flow of execution does not allow Pine Script™ to inspect the use of series in branches of conditional statements (`if`, `iff` or `?`), and Pine Script™ cannot automatically detect how far back the series is referenced. Here is an example of a script causing this problem:

```
//@version=5
indicator("Requires max_bars_back")
test = 0.0
if bar_index > 1000
    test := ta.roc(close, 20)
plot(test)
```

In order to help Pine Script™ with detection, you should add the `max_bars_back` parameter to the script's `indicator` or `strategy` function:

```
//@version=5
indicator("Requires max_bars_back", max_bars_back = 20)
test = 0.0
if bar_index > 1000
    test := ta.roc(close, 20)
plot(test)
```

You may also resolve the issue by taking the problematic expression out of the conditional branch, in which case the `max_bars_back` parameter is not required:

```
//@version=5
indicator("My Script")
test = 0.0
roc20 = ta.roc(close, 20)
if bar_index > 1000
    test := roc20
plot(test)
```

In cases where the problem is caused by a **variable** rather than a built-in **function** (`vwma` in our example), you may use the `max_bars_back` function to explicitly define the referencing length for that variable only. This has the advantage of requiring less runtime resources, but entails that you identify the problematic variable, e.g., variable `s` in the following example:

```
//@version=5
indicator("My Script")
f(off) =>
    t = 0.0
    s = close
    if bar_index > 242
        t := s[off]
    t
plot(f(301))
```

This situation can be resolved using the `max_bars_back` **function** to define the referencing length of variable `s` only, rather than for all the script's variables:

```
//@version=5
indicator("My Script")
f(off) =>
    t = 0.0
    s = close
    max_bars_back(s, 301)
    if bar_index > 242
        t := s[off]
    t
plot(f(301))
```

When using drawings that refer to previous bars through `bar_index[n]` and `xloc = xloc.bar_index`, the time series received from this bar will be used to position the drawings on the time axis. Therefore, if it is impossible to determine the correct size of the buffer, this error may occur. To avoid this, you need to use `max_bars_back(time, n)`. This behavior is described in more detail in the section about drawings.

FAQ

Release notes