# Calculation Variables - Let() function

- Let functions with only one variable declaration can be defined on one line.

```
Let ( ~someVar = FunctionalArea » Tablename::fieldName ; If ( ~someVar = "Active"; True; False ) )
```

- Example #1 Let function with indenting - Brackets on dedicated lines.

```
Let (
        [
            ~privateVariable = List ( "one" ; "two" ; "three" );
            $localVariable = Substitute ( ~privateVariable ; [ ¶ ; "," ] );
            $$globalVariableTopValue = GetValue ( ~privateVariable ; 1 )
        ];
            "Your Let function result is " & $localVariable
)
```

- Example #2 Let function with indenting - Initial bracket on same line as Let function with inline notes.

```
Let ( [
            // Notes about variables below
            ~privateVariable = List ( "one" ; "two" ; "three" );
            $localVariable = Substitute ( ~privateVariable ; [ ¶ ; "," ] );

            /*
              Use extra lines and embedded block comments
              if more information is needed to describe
              what is going on with the calculation logic!
            */
            $$globalVariableTopValue = GetValue ( ~privateVariable ; 1 )
        ];
            "Your Let function result is " & $localVariable
)
```

- Let functions with multiple variables use both opening and closing brackets on their own lines
  Note: both opening and closing brackets should be indented to stand out.

| | |
|---|---|
| `Let (`<br>`  [`<br>`  variable = expression`<br>`  ];`<br>`  "result is indented 2 tabs"`<br>`)` | good |
| `Let ( [` | acceptable |
| `Let (`<br>`[` | bad |

Because the standard started out with the opening bracket on the same line as the function name, yet does not impede readability, it can be considered the shorthand version and perfectly acceptable.

- Closing Let variable declarations end on their own line. This indicates the start of the result.

| | |
|---|---|
| `];` | good |

| | |
|---|---|
| `endOfFunction ) ];` | bad |

- Calculation scoped variables use camelCase and are identified by a preceding variable indicator of ~ (tilde). The ~ character in these standards represents the private scope.
  This makes it easy to distinguish calculation variables from custom function arguments, $variables, and Table::fieldNames

| | |
|---|---|
| `~someVariable` | good |
| `someVariable` | bad |

- Use present tense verbs or adjectives to indicate Boolean variable status on both calculation and locally scoped variables.

```
$hasReturns
~isTrailing
$containsSpaces
not ~containsEmailAddress
```