# Loops, Variables & Functions

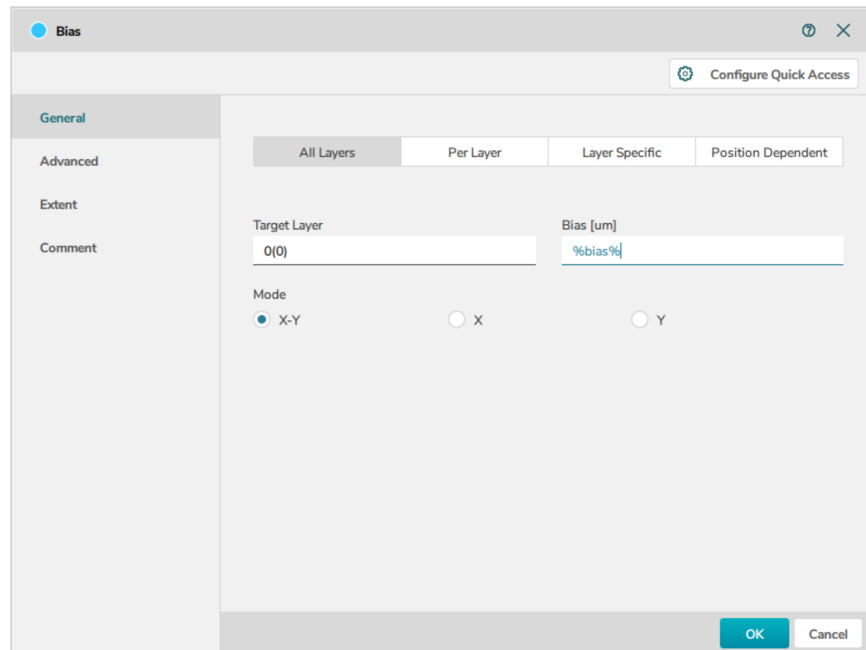
Variable handling is a key factor for effective and flexible working inside of flows and doing pattern processing. With this note we will explore the various options inside BEAMER for variables and how they can help in a day-to-day work environment.

## Content

Loops, Variables & Functions ..... 1
Places for variables ..... 2
Definitions of variables ..... 2
Global and Local variables ..... 2
Default values ..... 4
Math with a single variable ..... 4
Math with multiple variables ..... 6
Reading environmental variables ..... 7
Functions as Inputs ..... 7

## Places for variables

Variables can be used in the GUI at any location that allows text input. It could be simple fields such as the effective blur or the field size or even a path for the IMPORT module or EXPORT module.




## Definitions of variables

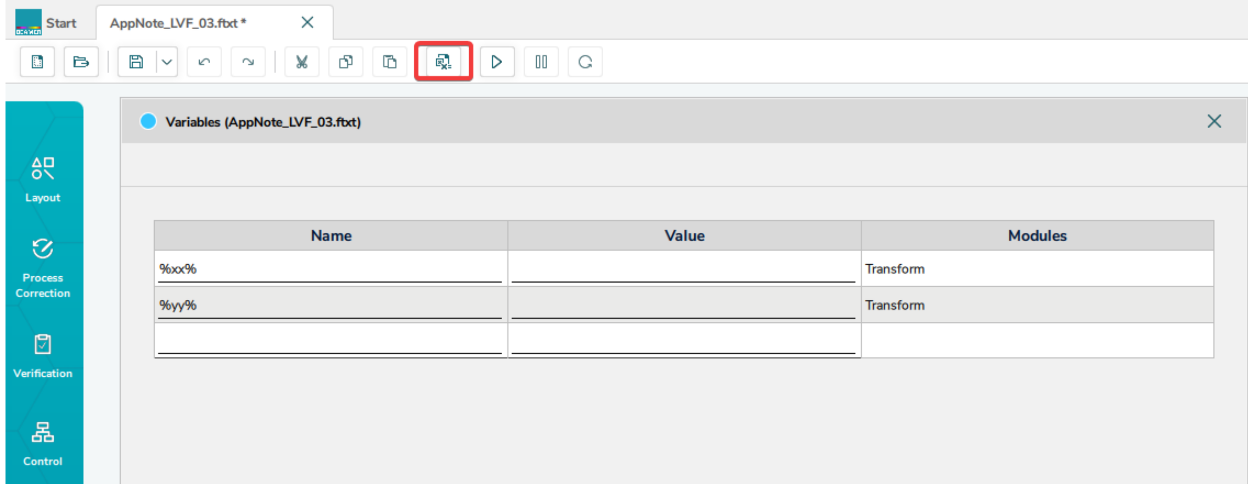
Each variable is denoted by a unique name enclosed within '%' symbols at both the beginning and end. For instance, %myvar% serves as an illustrative example of this naming convention.

Two predefined variables hold specific significance and are restricted for distinct purposes. These variables are %IMPORT% and %EXPORT%. They retain the most recent path associated with the Import module and Export module, respectively. This mechanism enables automatic tracking of paths for these specific operations.

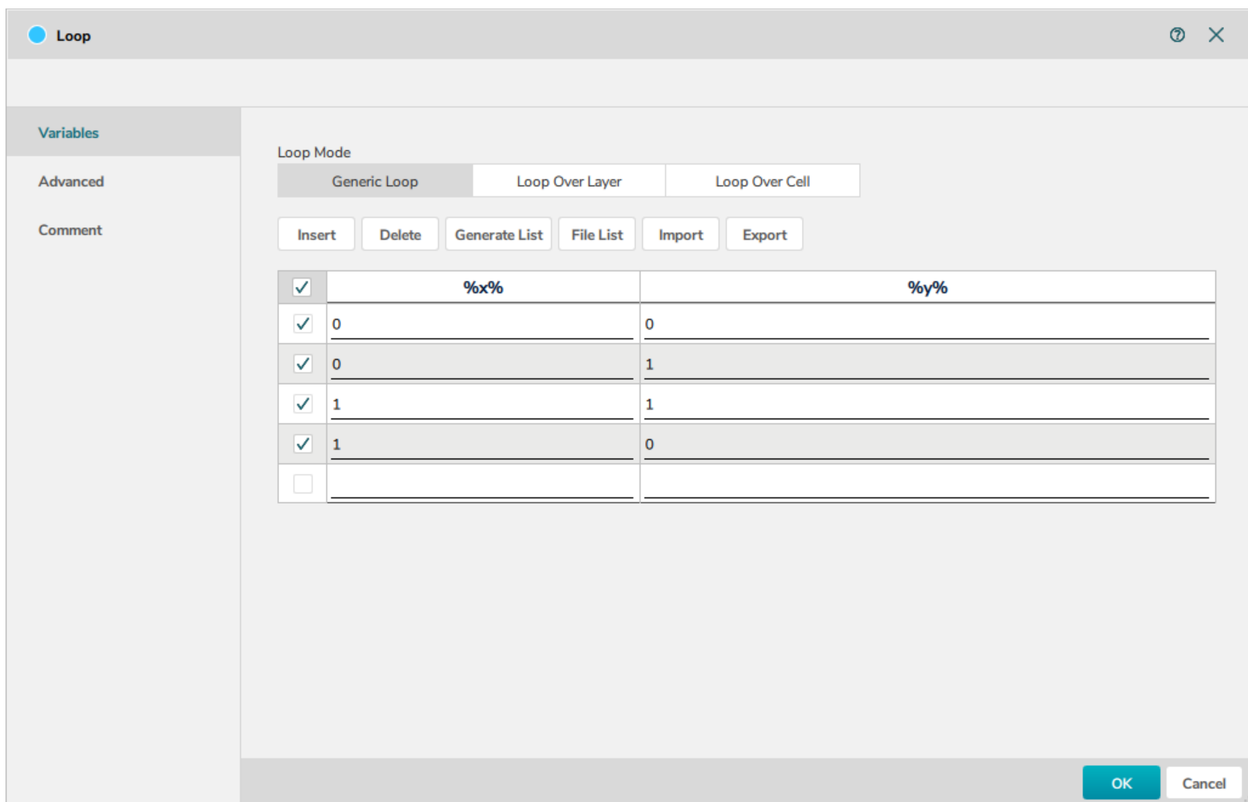
Beyond these predefined variables, users are granted the freedom to choose any other name for their variables, barring a specific combination. Specifically, a combination involving a prefix followed by %EnvVar_% is reserved for future exploration and utilization.

## Global and Local variables

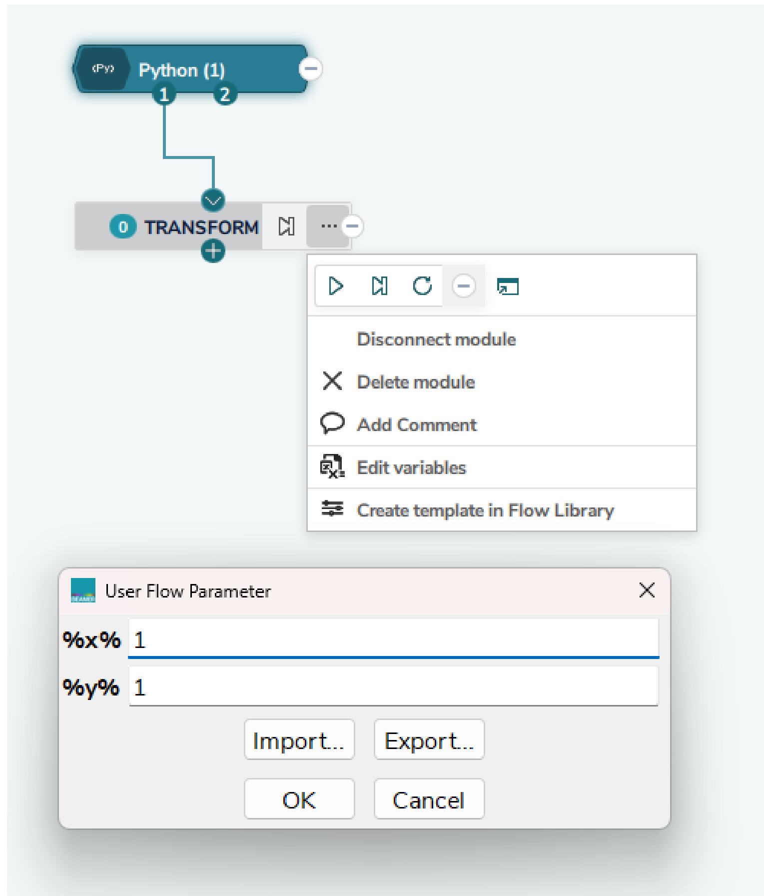
The concept of local and global variables is very typical in programming. BEAMER adapted this concept to have local variables with loops and global variables for the remainder of the flow. Defining a variable anywhere in a module sets a global variable.




Using Loops defines a frame that sets locally within the loop an environment for variables. It can access variables that are defined globally but does not allow the same name to be re-used outside of the loop. A local variable takes dominance over the globally defined one, meaning we have a method of overwriting global variables with local / loop variables.




Custom Modules/Flow also fall under this restriction. They allow the usage of variables that are queried when taking a custom module to the flow, and variables inside these custom flows will overwrite the identically named global variable.




Adding to any module a variable that is not within a user custom flow or a loop will create a variable entry in the list and can be referred to as a global variable. This one can be used anywhere else, and thus it can be a convenient method to change a value at multiple locations while changing it only once in the lists of variables.

## Default values

Working with user flows or global variables, setting a default value can be quite advantageous. A default value implies that the variable is initially loaded with a specific value, which can later be changed either through the variable list or by configuring user module parameters. There is no need to set a default value for loop variables, as they are automatically reset with each iteration.

Defining a default value is a straightforward process. Simply insert a set of parentheses containing the desired value after the variable's name but before the final '%'. This action will preload the variable with the designated default value. It is important to note that within the BEAMER framework, all variables are treated as strings. The processing of variable is happening at the execution of the module, so the value of the variable is then converted into the integers, floats or strings that the input field is actually expecting. As user you do not need to worry about this part.

As an example:

```
%name(20)%
```

In this instance, `%name%` is established with a default value of 20, achieved by appending `(20)`.

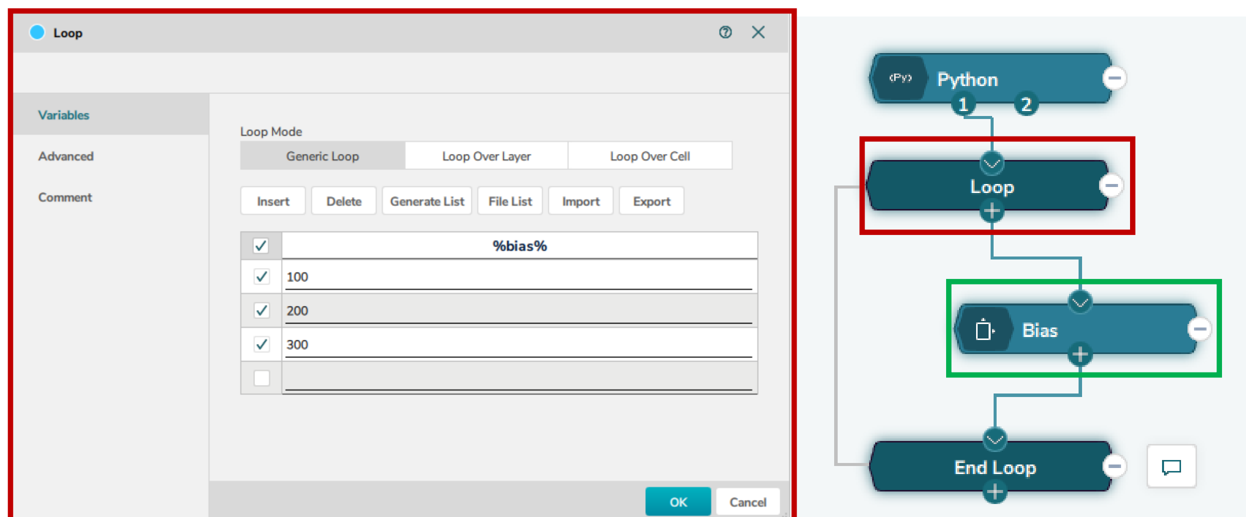
## Math with a single variable

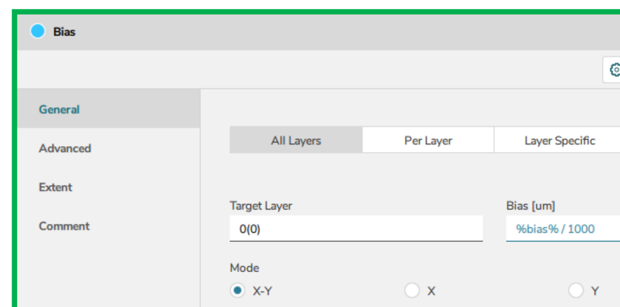
Math operations on single variables are possible by including the desired formula in the field. For instance, if I would like to set my variable in units of nm but the field expects an input of µm, one can apply the conversion operation (division or multiplication).

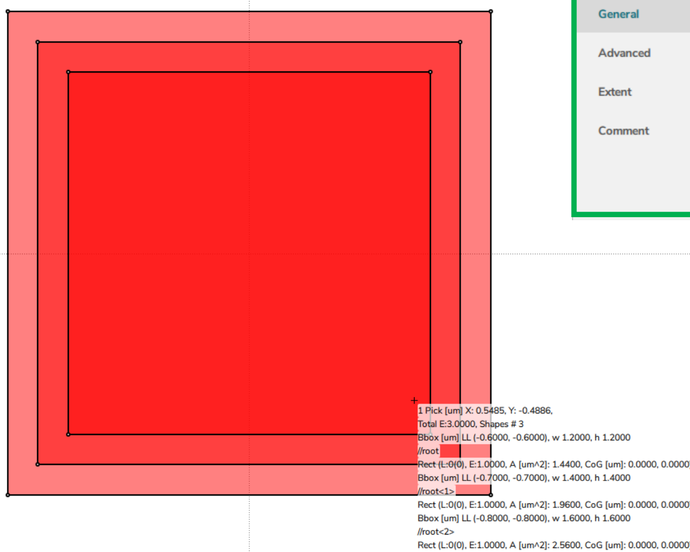
Example:

```
%bias% / 1000
```

This would convert the nanometers units to µm in the input field, here the Bias XY



The parser that processes the inputs is based on C++. Therefore, a helpful link to check all available possibilities is: C++ Math Expression Parser

## Math with multiple variables

With variables, you can do strings and math operations. String creation can be a simple chain of text and variables.

Example:

Variables would be %x% and %y% indexing a matrix position. To export the results of these as a file, the export path would look like this:

```
Samplefile_posX%x%_posY%y%.gds
```

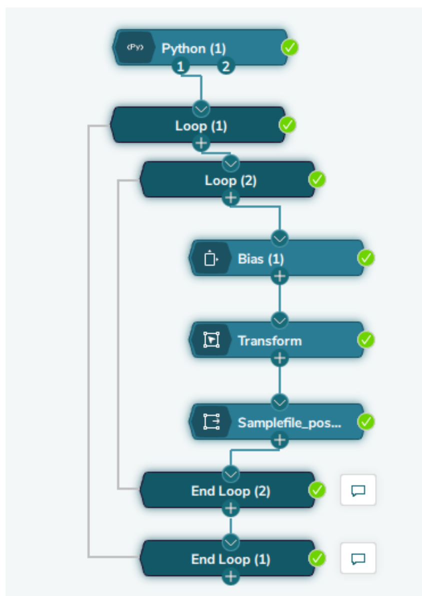
This results in a file list like:

Samplefile_posX0_posY0.gds, Samplefile_posX0_posY1.gds, Samplefile_posX0_posY2.gds, Samplefile_posX1_posY0.gds,…

In case some math would be needed, the syntax is %( )% wrapping the formula. Taking the same example, let us add an offset to the posX and posY indicating a different series in the files. The offset in the design is represented by the TRANSFORM shifting the pattern. In addition, each series is getting a unique bias applied to the pattern.

```
Samplefile_posX%(%bias%+%x%)%_posY%(%bias%+%y%)%.gds
```

The series is now 100, 200, …




```
Name
----
Samplefile_posX100_posY100.gds
Samplefile_posX100_posY101.gds
Samplefile_posX101_posY100.gds
Samplefile_posX101_posY101.gds
Samplefile_posX200_posY200.gds
Samplefile_posX200_posY201.gds
Samplefile_posX201_posY200.gds
Samplefile_posX201_posY201.gds
Samplefile_posX300_posY300.gds
Samplefile_posX300_posY301.gds
Samplefile_posX301_posY300.gds
Samplefile_posX301_posY301.gds
```

## Reading environmental variables

In some instances, getting information from the environment variables of your operating system can be helpful. A typical use case would be to store a specific user or project name, for example, in a variable and then append this to file names or include it to make decisions based on their values within the flow.

`%EnvVar_<name>%` is a fixed prefix to the variable name to access the value of that specific variable and call it to the BEAMER environment.

For example, we could put in every file we export the project name that has been set to an environmental variable plus the username:

```
%EnvVar_ProjectName%_%EnvVar_UserName%_myproject_01.gds
```

would return for ProjectName 'VariableDemo' and UserName 'Ritter'

VariableDemo_Ritter_myproject_01.gds

## Functions as Inputs

The IF and SELECT module support functions to make flow decisions based on parameters from the incoming layout. These functions use a Python-orientated syntax to read information from layout. Besides the syntax in the GUI, a specific Python function is also available.

The main difference is that the GUI syntax is skipping any parameters, as the present layout is the parameter. In Python, we can use various layouts and therefore they are addressed.

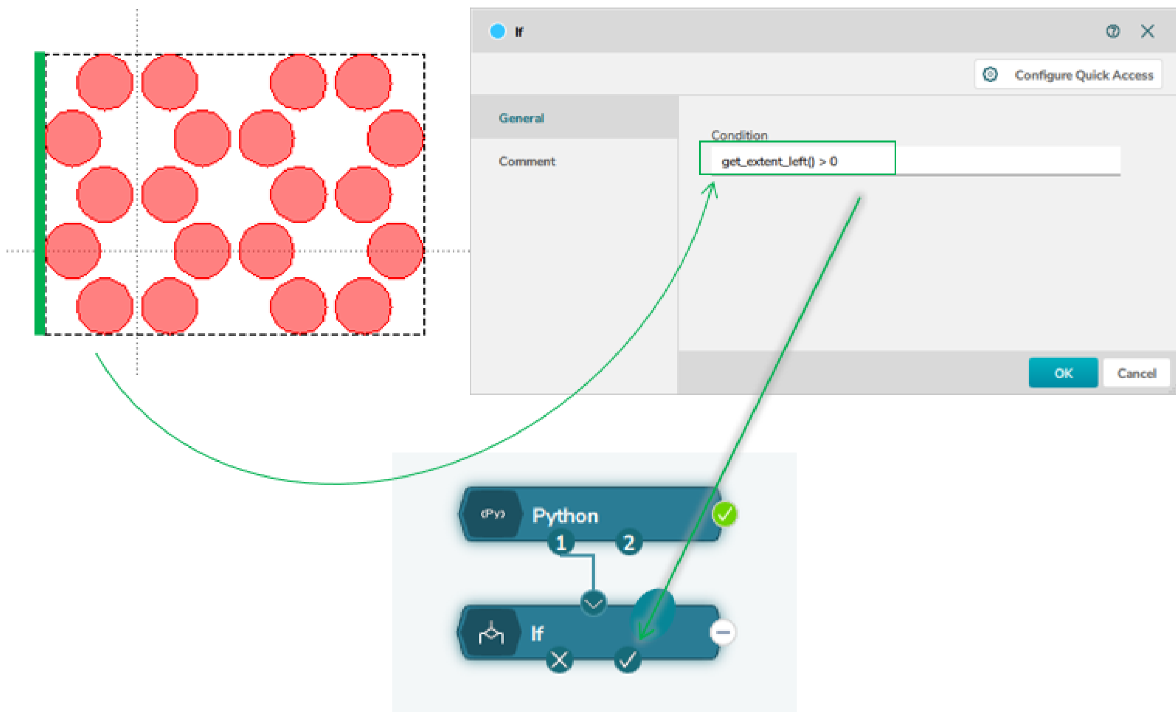
GUI: `get_area()`

Python: `BEAMER.get_area(in1)`

In the IF or SELECT modules these can be compared against criteria, for example, to check if a design is in the positive x coordinates one would use an IF module with these criteria:

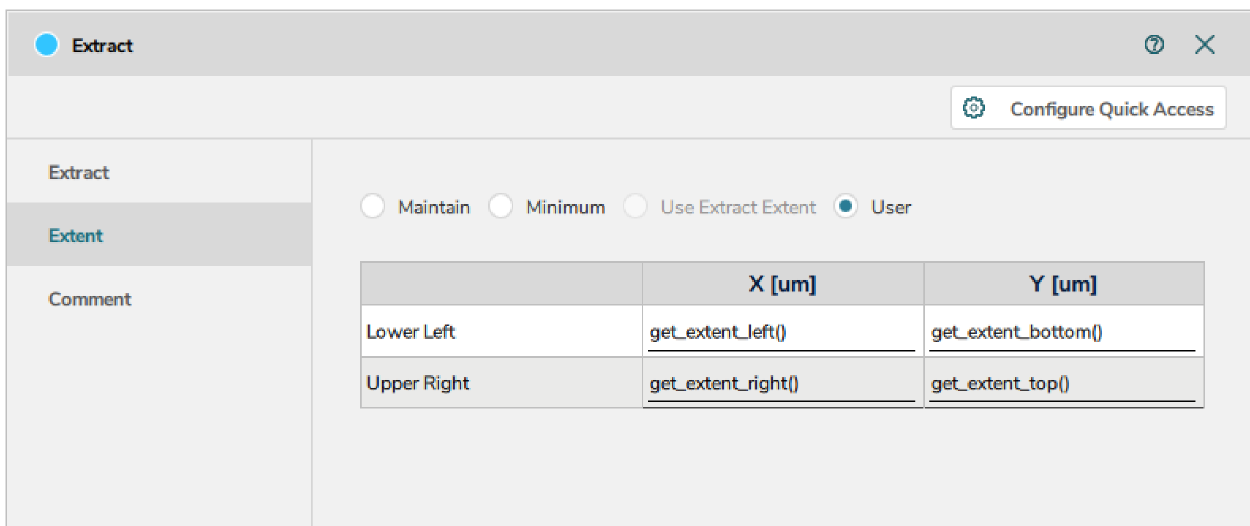
```
get_extent_left() > 0
```

The flow would then branch into the right branch of the IF module in case the condition would be true, otherwise, it would branch left.




Another example here compares Python with GUI. The goal is to grow the extent of the current pattern by 5µm in each direction. This should be independent of the pattern coming in and therefore should work for all layouts without the user needing to inspect the pattern extent and adjust.

GUI:




Python:

Python Dialog

Preview | Interactive View

Python Script

Comment

```
out1 = BEAMER.extract_layer( in1,
    { 'ExtentMode' : 'User',
      'LowerLeftX' : BEAMER.get_extent_left(in1) - 5,
      'LowerLeftY' : BEAMER.get_extent_bottom(in1) -5,
      'UpperRightX' : BEAMER.get_extent_right(in1) +5,
      'UpperRightY' : BEAMER.get_extent_top(in1) + 5
    } )
```

- Modules
- Evaluations
- Interface
- LayoutPy
- Math
- Examples
- Snippets