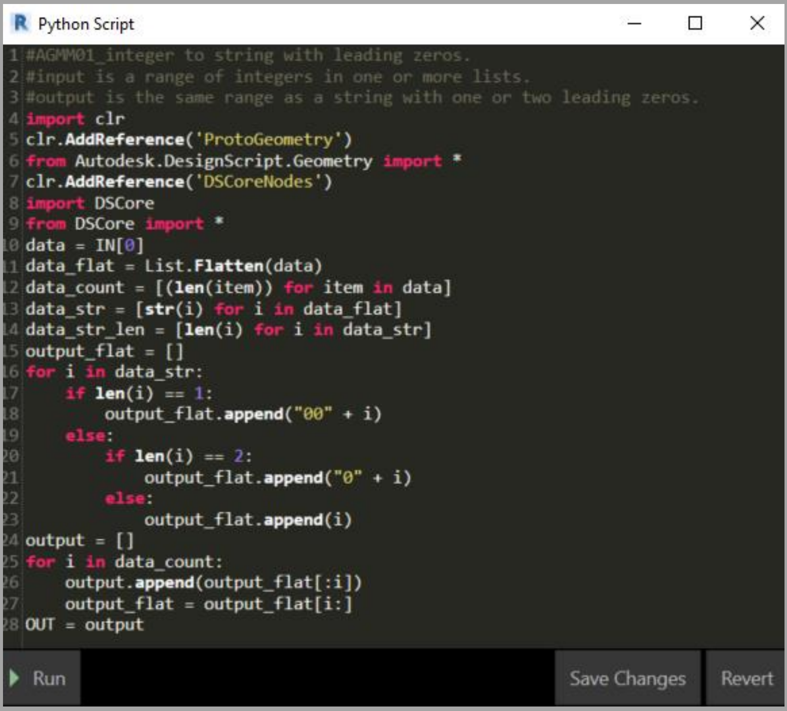
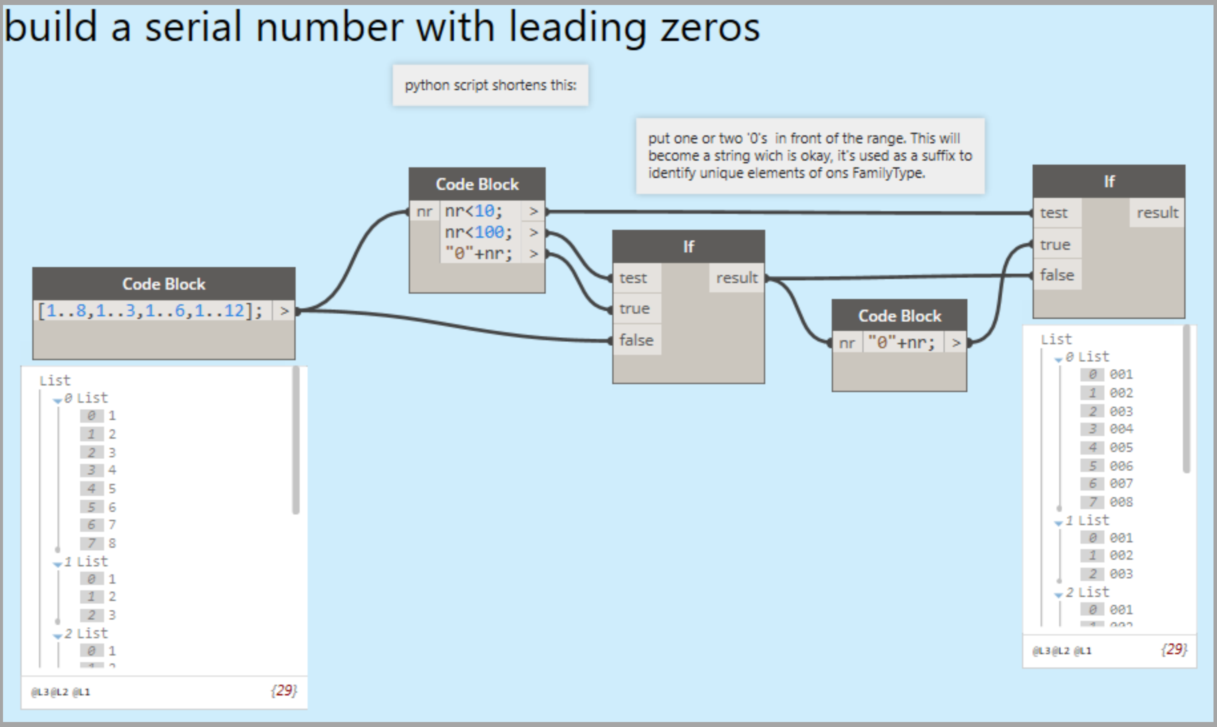
AGMM01_integer to string with leading zeros.

```python
#AGMM01_integer to string with leading zeros.
#input is a range of integers in one or more lists.
#output is the same range as a string with one or two leading zeros.
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
clr.AddReference('DSCoreNodes')
import DSCore
from DSCore import *
data = IN[0]
data_flat = List.Flatten(data)
data_count = [(len(item)) for item in data]
data_str = [str(i) for i in data_flat]
data_str_len = [len(i) for i in data_str]
output_flat = []
for i in data_str:
    if len(i) == 1:
        output_flat.append("00" + i)
    else:
        if len(i) == 2:
            output_flat.append("0" + i)
        else:
            output_flat.append(i)
output = []
for i in data_count:
    output.append(output_flat[:i])
    output_flat = output_flat[i:]
OUT = output
```



```python
#AGMM01_integer to string with leading zeros.
#input is a range of integers in one or more lists.
#output is the same range as a string with one or two leading zeros.
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
clr.AddReference('DSCoreNodes')
import DSCore
from DSCore import *
data = IN[0]
data_flat = List.Flatten(data)
data_count = [(len(item)) for item in data]
data_str = [str(i) for i in data_flat]
data_str_len = [len(i) for i in data_str]
output_flat = []
for i in data_str:
    if len(i) == 1:
        output_flat.append("00" + i)
    else:
        if len(i) == 2:
            output_flat.append("0" + i)
        else:
            output_flat.append(i)
output = []
for i in data_count:
    output.append(output_flat[:i])
    output_flat = output_flat[i:]
OUT = output
```

What does this script do:


build a serial number with leading zeros

python script shortens this:

put one or two '0's in front of the range. This will become a string wich is okay, it's used as a suffix to identify unique elements of ons FamilyType.

```
[1..8,1..3,1..6,1..12];
```

```
nr<10;
nr<100;
"0"+nr;
```

```
"0"+nr;
```

```
List
 0 List
   0 1
   1 2
   2 3
   3 4
   4 5
   5 6
   6 7
   7 8
 1 List
   0 1
   1 2
   2 3
 2 List
   0 1
@L3 @L2 @L1                {29}
```

```
List
 0 List
   0 001
   1 002
   2 003
   3 004
   4 005
   5 006
   6 007
   7 008
 1 List
   0 001
   1 002
   2 003
 2 List
   0 001
@L3 @L2 @L1                {29}
```

It replaces this double if function to get one or two zeros in front of a range of numbers. It's used to add as a suffix to elements in the model so eacht instance of a window canbe counted and assembled by unique number. A serial number.

The script gets a list of ranges and turns that into strings. The length of the strings is counted to know if there needs to be a prefix of one or two leading zeros. The main problem I encountered here was the list. To go from integers to strings the list needed to be flattened. Then in the end the list needed to be chopped into it's original lists again. There is no List.Chop in the DSCoreNodes.

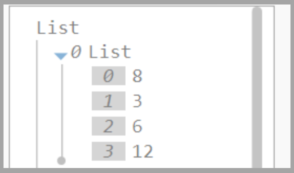
The lines explained:

Boilerplate:
It's all needed to adress the List.Flatten function.
The rest is basic Python.

data = IN[0]
one input that is called data. The input is a list of ranges.

data_flat = List.Flatten(data)
flatten the input in to one list.

data_count = [(len(item)) for item in data]
counts the amount of items in eacht list.Looking at the input in the previous image it will output this list:
(To test outputs just add them to OUT = data_count, output)


```
List
 0 List
   0 8
   1 3
   2 6
   3 12
```

data_str = [str(i) for i in data_flat]
we have the variable 'data_flat'.
We create a new variable 'data_str'.

Put every i (common shorthand for item) in ‘data_flat’ into the new ‘data_str’ but as a string.

This line is short for this:

```
data_str = []                        create a varibale called ‘data_str’ which is an empty list.
for i in data_flat:                  for every item in ‘data_flat’,
  data_str.append(str(i))            append it as a string to the new empty list.
```

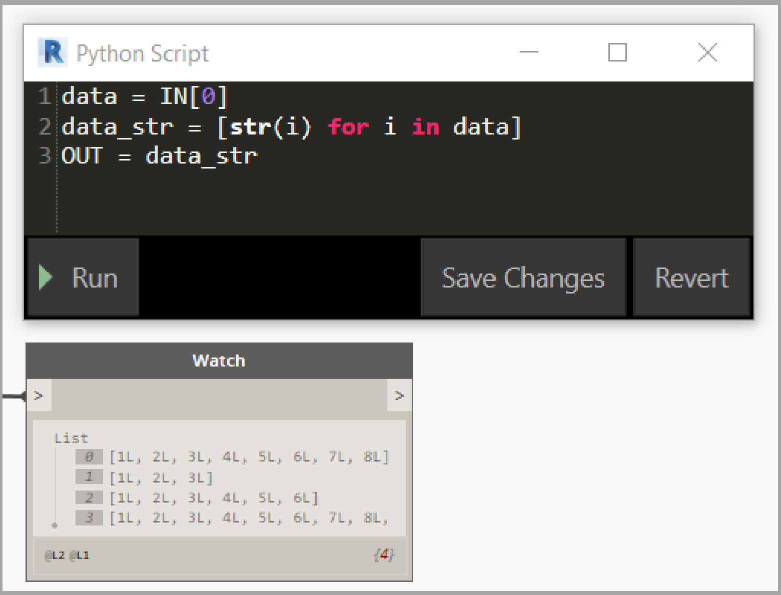
when coding, the long version is easier for beginners. Can be condensed later.

Here is why the list must be flattenned first:

The 1L represents a long integer in Python 2.x.

This makes no sense to me since 1, 2 etc. seems very short…

This will not happen in Python 3. From Dynamo 2.7 on, Python 3 will be supported. Still I want a list and not one line with numbers as a string, divided by a comma between brackets…



```
data = IN[0]
data_str = [str(i) for i in data]
OUT = data_str
```

```
List
  0 [1L, 2L, 3L, 4L, 5L, 6L, 7L, 8L]
  1 [1L, 2L, 3L]
  2 [1L, 2L, 3L, 4L, 5L, 6L]
  3 [1L, 2L, 3L, 4L, 5L, 6L, 7L, 8L,
@L2 @L1                              {4}
```

```
data_str_len = [len(i) for i in data_str]
```

we have the variable ‘data_str’.
We create a new variable ‘data_str_len’.
count every i in ‘data_str’ and put it into the new list ‘data_str_len’

```
output_flat = []`                          create new empty ist
for i in data_str:                         check every item in ‘data_str’
  if len(i) == 1:                          if this item is 1, so only one digit
    output_flat.append("00" + i)           add this item to the empty list with a prefix of 00
  else:                                    if not
    if len(i) == 2:                        if this item is 2, so only two digits
      output_flat.append("0" + i)          add this item to the empty list with a prefix of 0
    else:                                  if not
      output_flat.append(i)                add this item to the list without prefix
```

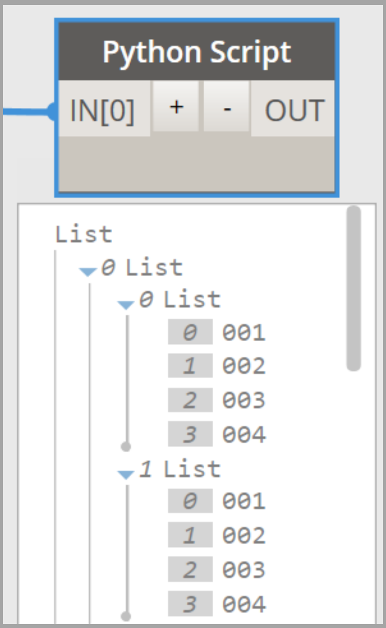
first create a new empty variable, a list called ‘output_flat’. The output we want but still flattenned. I want two leading zeros in front of the numbers 1 through 9, and one leading zero in front of 10 through 99. Since there is no need for larger numbers it stops here. But it’s easy to add another if-then-else when needed.

```
for i in data_count:
  output.append(output_flat[:i])
  output_flat = output_flat[i:]
```

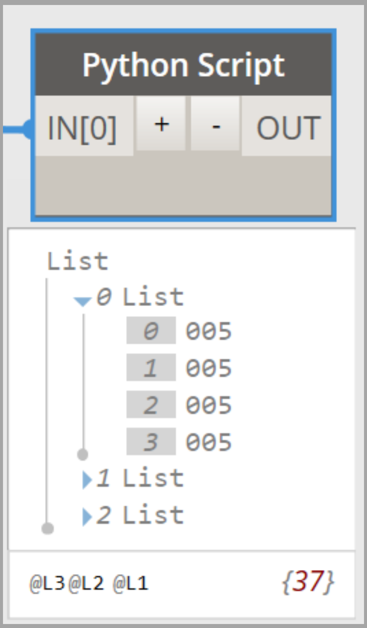
‘data_count’ gives the chop-length of the flattenned list.

Create a list called ‘output’ where wew put ‘output_flat’ in.
Slicing a list is done with this syntax: [:i]
Instead of i you can put 0, 1 etc in it to get the first,first two, first three items in the list.
Here is the difference between the first four and the fourth:

`output.append(output_flat[:4])` `output.append(output_flat[4])`



```
List
  0 List
    0 List
      0 001
      1 002
      2 003
      3 004
    1 List
      0 001
      1 002
      2 003
      3 004
```

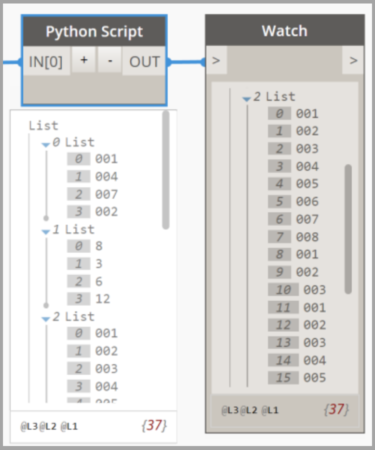


```
List
  0 List
    0 005
    1 005
    2 005
    3 005
  1 List
  2 List
@L3@L2 @L1          {37}
```

So we need [:1] but that still doesn’t work without an extra line.

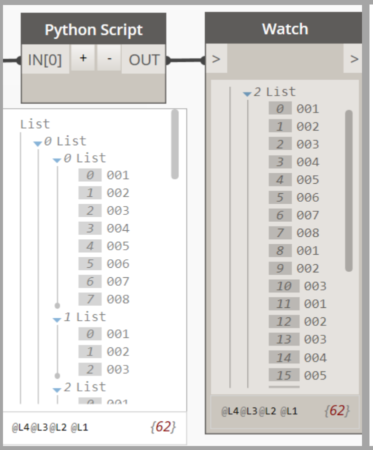
output.append(output_flat[i])
gives item 8,3,6 and 12.

output.append(output_flat[:1]) might look good but when looking at list 3 that should contain 001 through 012 you see that it stops at 008 and starts again. Only when we add the line output_flat = output_flat[i:] it works. This says to continue with i where it left off. Instead of starting at the beginning of output_flat when starting a new list, continue with the next item in output_flat.



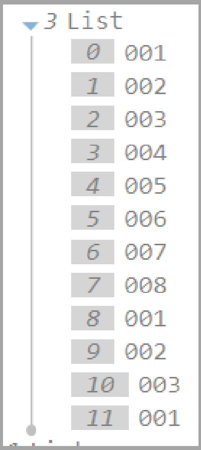
```
List
  0 List
    0 001
    1 004
    2 007
    3 002
  1 List
    0 8
    1 3
    2 6
    3 12
  2 List
    0 001
    1 002
    2 003
    3 004
@L3@L2 @L1          {37}
```

```
    2 List
      0 001
      1 002
      2 003
      3 004
      4 005
      5 006
      6 007
      7 008
      8 001
      9 002
      10 003
      11 001
      12 002
      13 003
      14 004
      15 005
@L3@L2 @L1          {37}
```



```
List
  0 List
    0 List
      0 001
      1 002
      2 003
      3 004
      4 005
      5 006
      6 007
      7 008
    1 List
      0 001
      1 002
      2 003
    2 List
@L4@L3@L2 @L1          {62}
```

```
    2 List
      0 001
      1 002
      2 003
      3 004
      4 005
      5 006
      6 007
      7 008
      8 001
      9 002
      10 003
      11 001
      12 002
      13 003
      14 004
      15 005
@L4@L3@L2 @L1          {62}
```



```
3 List
  0 001
  1 002
  2 003
  3 004
  4 005
  5 006
  6 007
  7 008
  8 001
  9 002
  10 003
  11 001
```