



A Journey with Python for MicroStation or MSPython

**Improving productivity through Python
programming**

Kees van Prooijen
Kees.vanProoijen@Bentley.com

File Home View Annotate Attach Analyze Curves Constraints Utilities Drawing Aids Content Collaborate Help PDK WMS/WMTS

OLE Named Expressions MDL Applications Close Tool Boxes Connect to Browser Display Convert Capture Image Play Record Stop detachAllHistory VBA Manager Python Manager Commit Initialize Design History Signatures Signature Cell Security Geographic Coordinate System Drawing Scale

Properties

Models (1)

- Default
 - DenHaag.dgn,binne
 - DenHaag.dgn,ridde

General

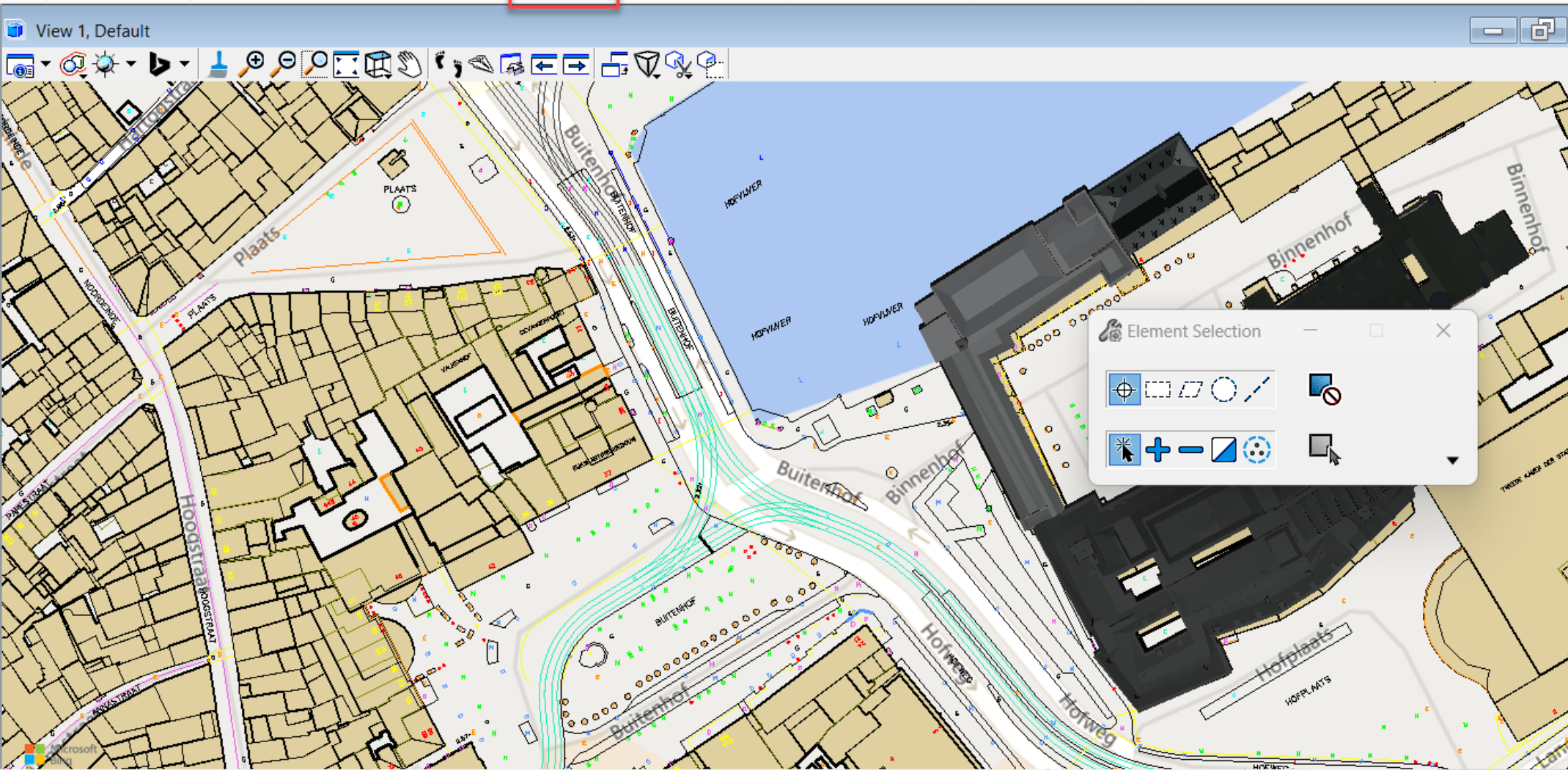
- Is Active: True
- Name: Default
- Design Type: Design
- Design Dir: 3D
- Is Markup: False
- Annotation: Full Size 1 = 1
- Design Sc: 1.0000
- Paper Sc: 1.0000
- Propagate: On
- Line Style: Global Line Style S
- Global Lin: 1.0000
- Update Fix: False

Angle Readout

- Direction: North
- Direction: Azimuth
- Format: ~DD.DDDD
- Accuracy: 0.1234
- Direction: Clockwise

Isometric

Explo... Prop... Level...



Element Selection

- Selection tools: Point, Window, Crossing, Lasso, Arc, Circle, Line, Polygon, Polyline, Text, Dimension, Image, etc.
- Options: Select, Add, Subtract, Intersect, Union, etc.

Basic information

- Python is introduced in MicroStation 2024
- Python v3.12 comes with MicroStation
- Python is an interpreter language, compiling is not needed
- Optionally a standalone Python Interpreter can be used
- The first release covers most aspects of MicroStation, we are working on exposing every feature in MicroStation.
- It allows Python like syntax to expose or access already existing C based objects classes and functions etc.
- [Documentation](#) | [API Presentations](#) | [FAQs](#) | [GitHub](#) | [Samples](#) | [Wikis](#) | [Blogs](#)

Why Python and why a new technology?

Ease of Learning and Readability:

Python's syntax is straightforward and clean, making it **accessible for beginners and efficient for experienced developers.**

Versatility:

Python is used in various domains, including **web development, data science, artificial intelligence,** and more.

Rich Ecosystem and Libraries:

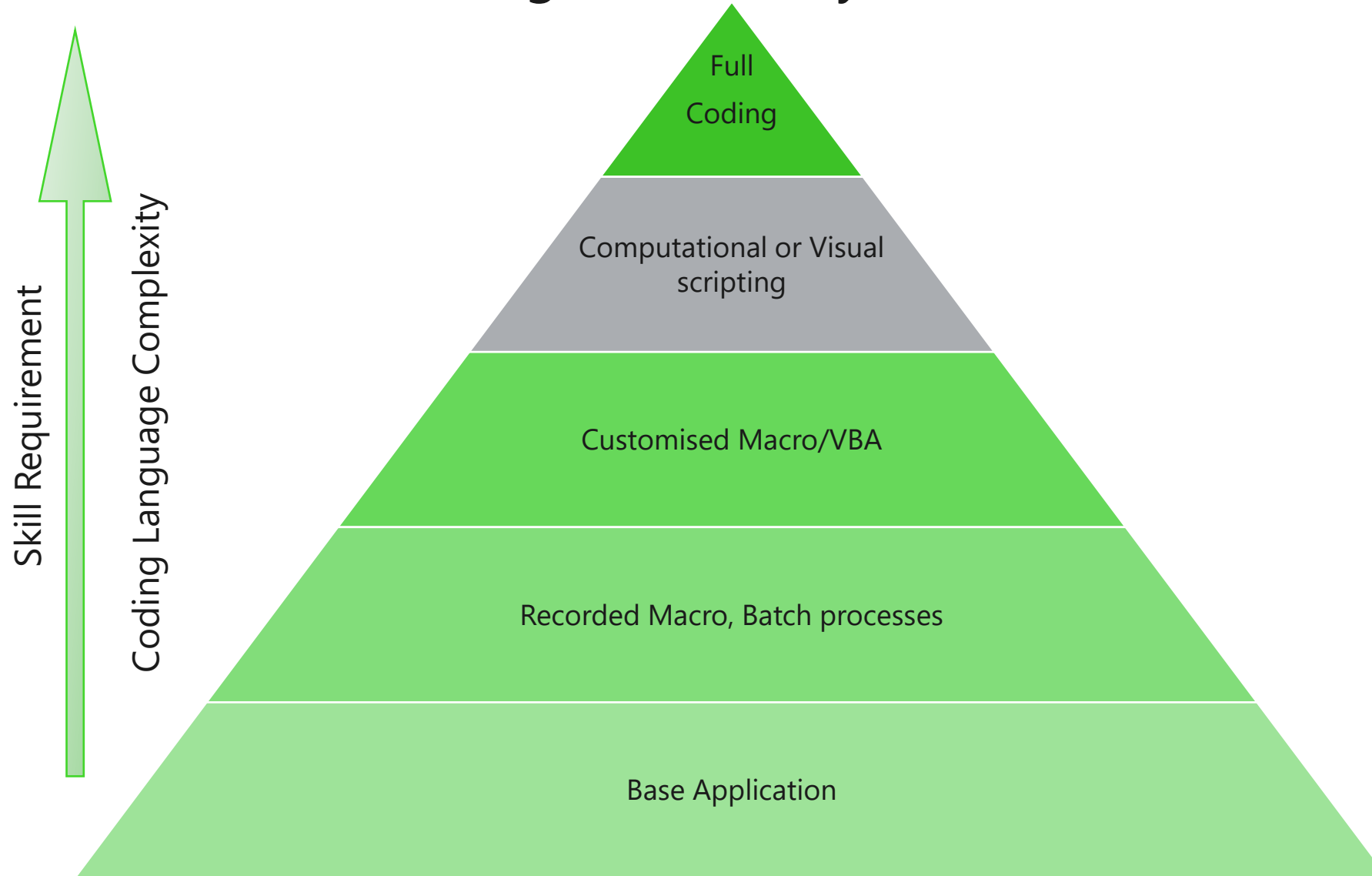
Python boasts a **vast collection of libraries and frameworks** that simplify complex tasks.

Strong Community Support:

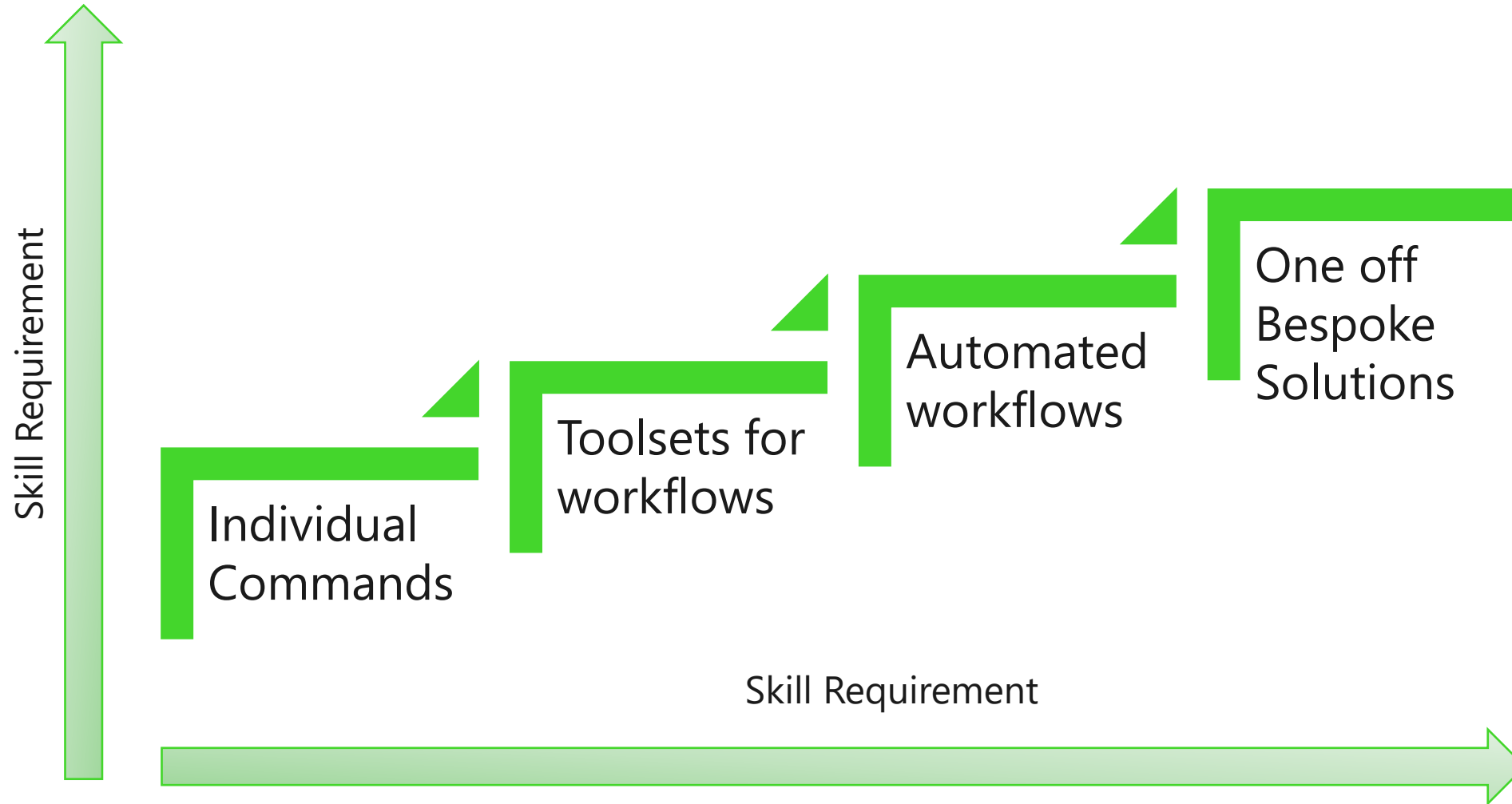
The Python community is active and supportive, providing **numerous resources for learning and problem-solving.**

*Geeksforgeeks.org

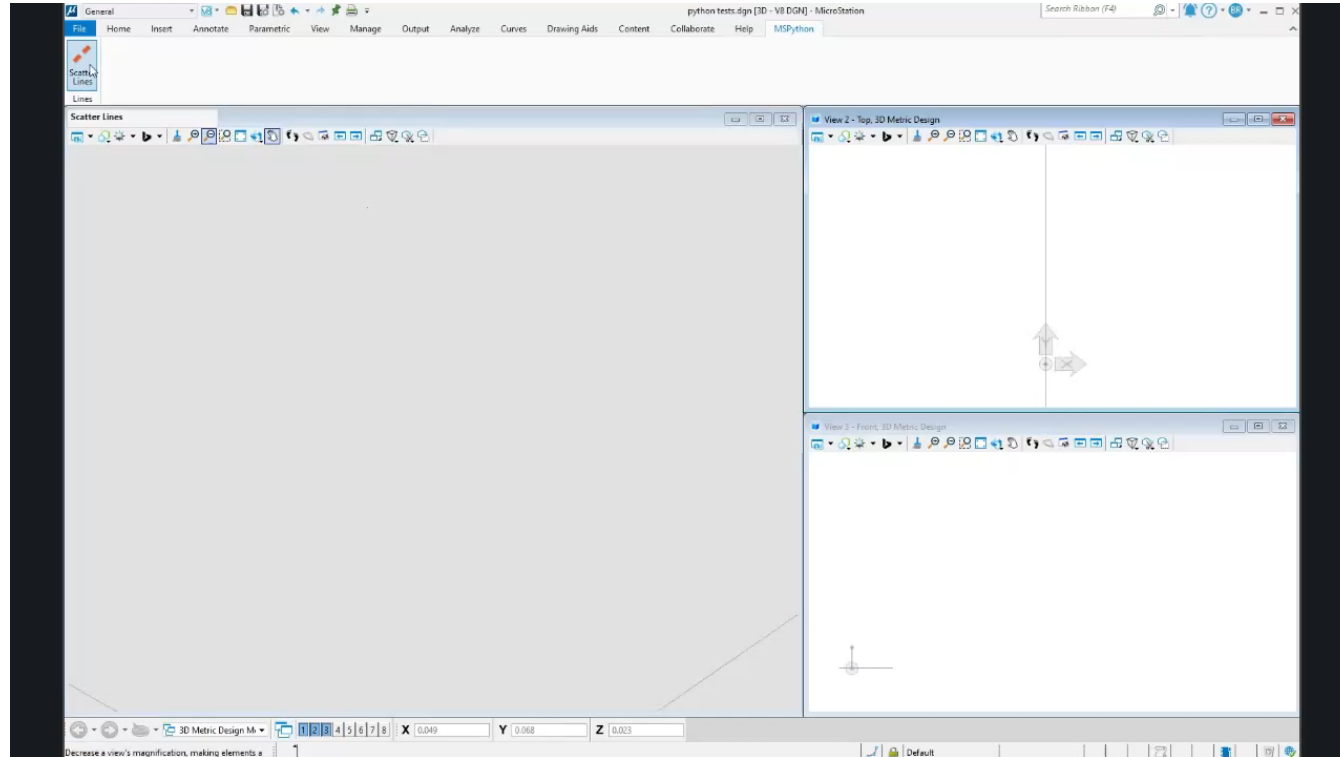
Digital Maturity Path



Digital Maturity Path



My Blog



[A journey with MSPython:](https://bentleysystems.service-now.com/community?id=community_blog&sys_id=7c6e52a51bcfc290b5f1da49cc4bcbc2)

https://bentleysystems.service-now.com/community?id=community_blog&sys_id=7c6e52a51bcfc290b5f1da49cc4bcbc2

Installation

- Python is installed with MS2024
- MicroStation detects the present of some IDEs* like Visual Studio Code (VSCode, free downloadable from <https://code.visualstudio.com>)
- Optionally you can manually add other IDEs

 MicroStation Python scripts **MUST be started** from inside a running instance of MicroStation.

* IDE: Independent Developers Environment

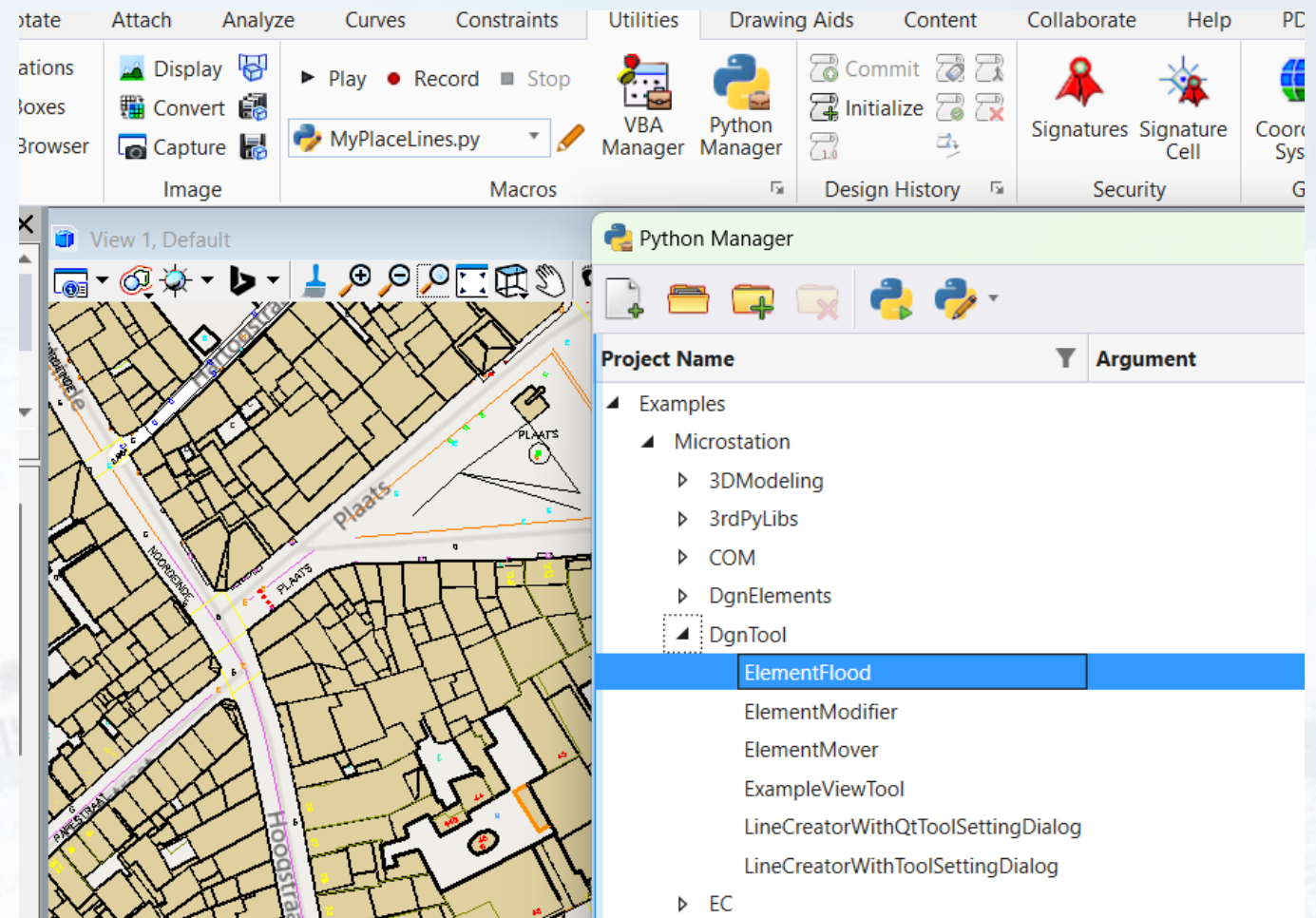
Getting Started

- Start MS
- Open de Python Manager
- Run a Python script from Examples > Microstation > DgnTool

⚠ Python scripts can be started with a keyin:

Python Load <Script_Name>.py

See [Part 4 Looking at a sample file](#)



New Configuration Variables

MS_LIBRARY_PATH	Library Path	System
MS_MDL	MDL Applications	System
MS_PYTHON	The Path to The Python Interpreter	System
MS_PYTHONNEWPROJECTDIRECTORY	Directory to Put New Python File	System
MS_PYTHONSAMPLES	Directory Containing the Python Samples	System
MS_PYTHONSCRIPTS	Directory List in Dialog Python Manager	System
MS_PYTHONSEARCHDIRECTORIES	Directories to Search For Python File	System
MS_XCOMMAND_APPS	XCommand Table Auto-load	System



Important Configuration Variables

MS_PYTHON

Tells MicroStation where the delivered interpreter is stored.

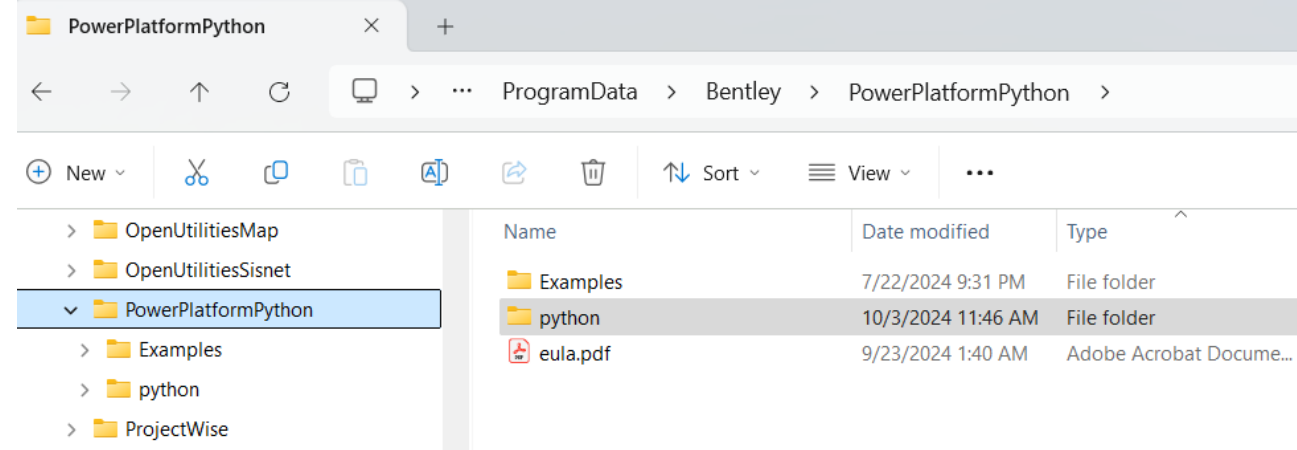
Default:

`$(ALLUSERSPROFILE)\Bentley\PowerPlatformPython\python\python/`

e.g.

`C:\ProgramData\Bentley\PowerPlatformPython\python\`

Check Python version with keyin: *Python Query Version*



Important Configuration Variables

MS_PYTHONNEWPROJECTDIRECTORY

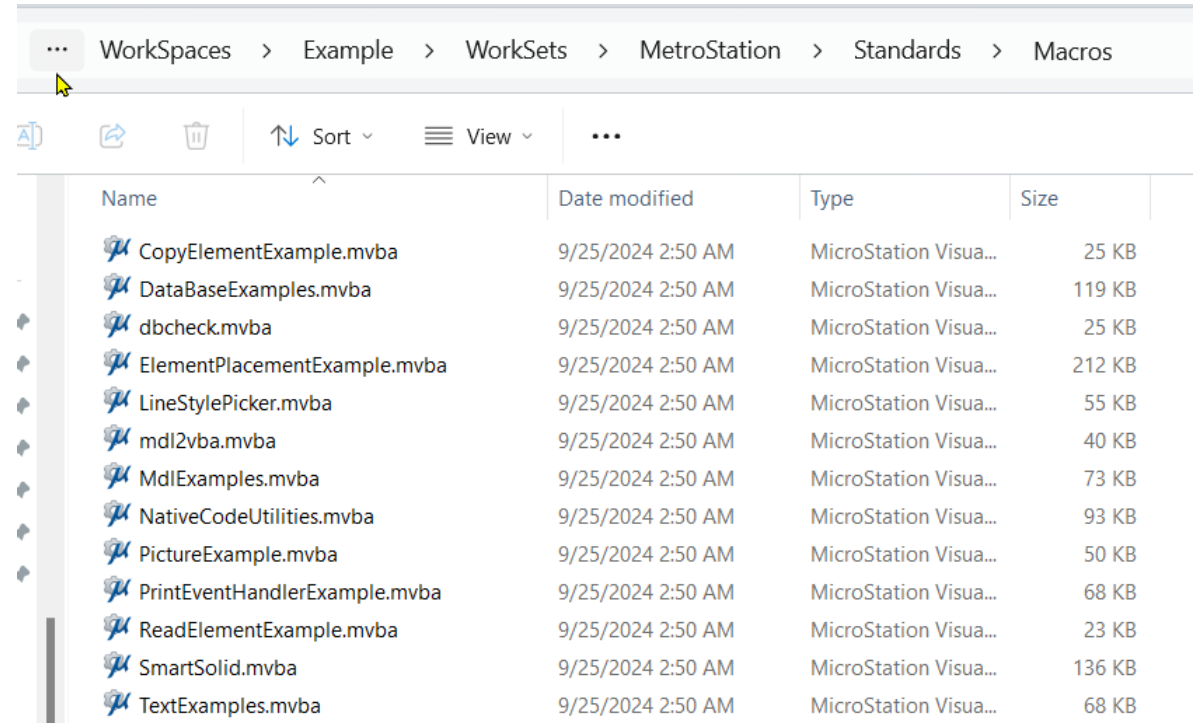
Tells MicroStation where New “Python” files should be created.

Default:

`$(_USTN_WORKSETSTANDARDS)MACROS/`

e.g.

C:\ProgramData\Bentley\MicroStation
2024\Configuration\WorkSpaces\TMC
Winterschool
2024\WorkSets\Python\Standards\Macros\



The screenshot shows a Windows File Explorer window with the address bar displaying the path: WorkSpaces > Example > WorkSets > MetroStation > Standards > Macros. The main pane shows a list of files with columns for Name, Date modified, Type, and Size. All files are .mvba files and were last modified on 9/25/2024 at 2:50 AM. The file types are listed as 'MicroStation Visua...'. The sizes range from 23 KB to 136 KB.

Name	Date modified	Type	Size
CopyElementExample.mvba	9/25/2024 2:50 AM	MicroStation Visua...	25 KB
DataBaseExamples.mvba	9/25/2024 2:50 AM	MicroStation Visua...	119 KB
dbcheck.mvba	9/25/2024 2:50 AM	MicroStation Visua...	25 KB
ElementPlacementExample.mvba	9/25/2024 2:50 AM	MicroStation Visua...	212 KB
LineStylePicker.mvba	9/25/2024 2:50 AM	MicroStation Visua...	55 KB
mdl2vba.mvba	9/25/2024 2:50 AM	MicroStation Visua...	40 KB
MdlExamples.mvba	9/25/2024 2:50 AM	MicroStation Visua...	73 KB
NativeCodeUtilities.mvba	9/25/2024 2:50 AM	MicroStation Visua...	93 KB
PictureExample.mvba	9/25/2024 2:50 AM	MicroStation Visua...	50 KB
PrintEventHandlerExample.mvba	9/25/2024 2:50 AM	MicroStation Visua...	68 KB
ReadElementExample.mvba	9/25/2024 2:50 AM	MicroStation Visua...	23 KB
SmartSolid.mvba	9/25/2024 2:50 AM	MicroStation Visua...	136 KB
TextExamples.mvba	9/25/2024 2:50 AM	MicroStation Visua...	68 KB

Important Configuration Variables

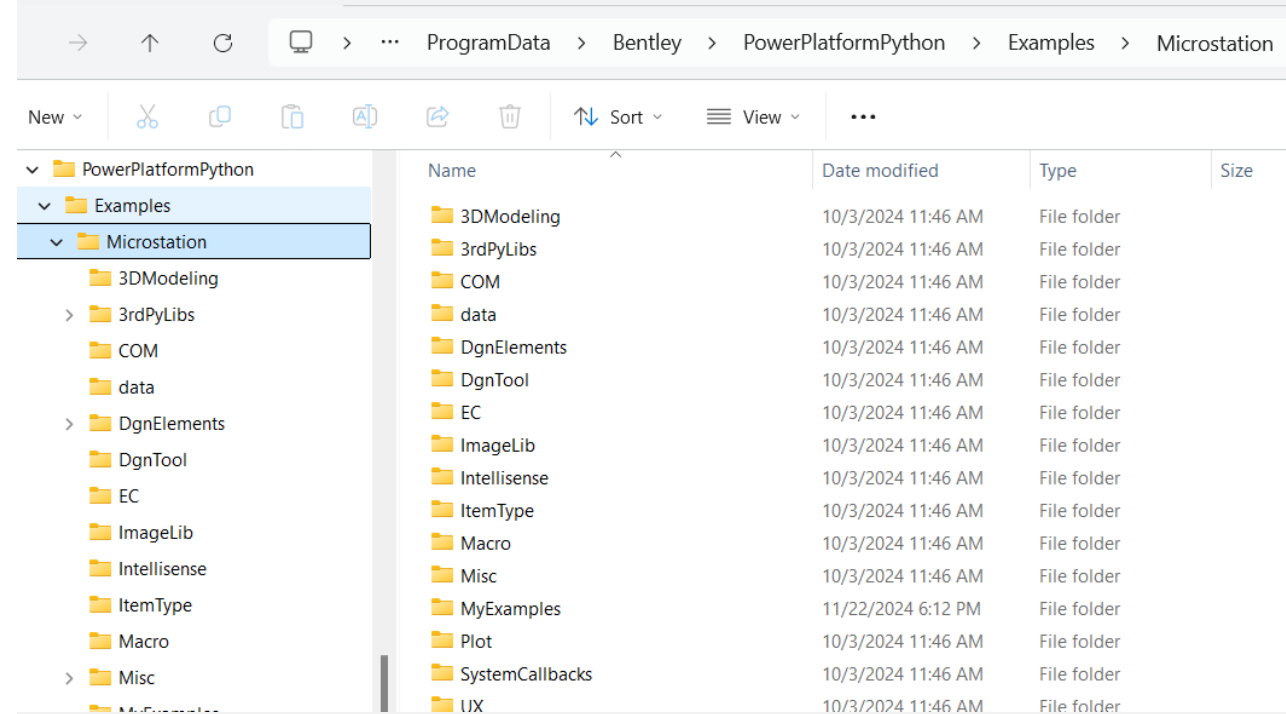
MS_PYTHONSAMPLES

Tells MicroStation where the Bentley delivered examples are stored

Defaults to new folder in **ProgramData**:
C:\ProgramData\Bentley\PowerPlatformPython\
Examples\

See also GitHub:

[MicroStationPython/MSPythonSamples at main · BentleySystems/MicroStationPython \(github.com\)](https://github.com/BentleySystems/MicroStationPython)



Examples

C:\ProgramData\Bentley\PowerPlatformPython\Examples\MicroStation

Name	Date modified	Type
3DModeling	10/3/2024 11:46 AM	File folder
3rdPyLibs	10/3/2024 11:46 AM	File folder
COM	10/3/2024 11:46 AM	File folder
data	10/3/2024 11:46 AM	File folder
DgnElements	10/3/2024 11:46 AM	File folder
DgnTool	10/3/2024 11:46 AM	File folder
EC	10/3/2024 11:46 AM	File folder
ImageLib	10/3/2024 11:46 AM	File folder
Intellisense	10/3/2024 11:46 AM	File folder
ItemType	10/3/2024 11:46 AM	File folder
Macro	10/3/2024 11:46 AM	File folder
Misc	10/3/2024 11:46 AM	File folder
MyExamples	11/22/2024 6:12 PM	File folder
Plot	10/3/2024 11:46 AM	File folder
SystemCallbacks	10/3/2024 11:46 AM	File folder
UX	10/3/2024 11:46 AM	File folder

A comprehensive list broken down by subject matter

In future new examples will be updated and made available between releases on [GitHub](#)

Important Configuration Variables

MS_PYTHONSCRIPTS

Tells MicroStation what folders to search for python files to list in the Python Manager dialog

Defaults to the folder with delivered samples and the folder where new projects are stored:

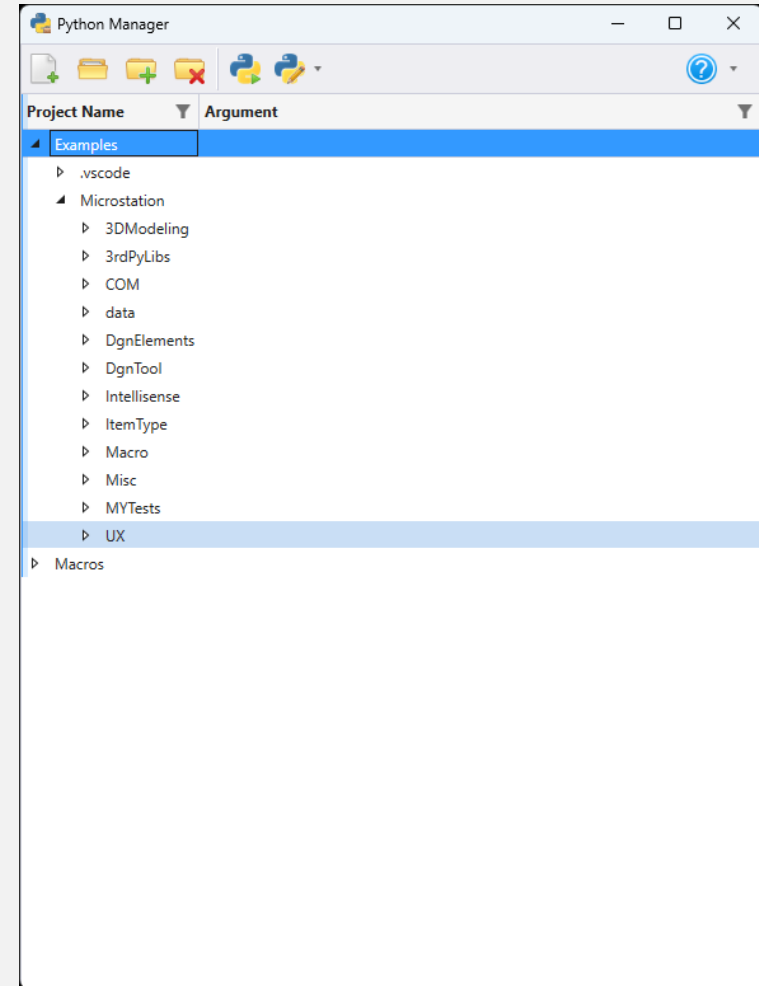
`$(MS_PYTHONSAMPLES);$(MS_PYTHONNEWPROJECTDIRECTORY)`

E.g.:

`C:\ProgramData\Bentley\PowerPlatformPython\Examples\`

`C:\ProgramData\Bentley\MicroStation
2024\Configuration\WorkSpaces\TMC
Winterschool`

`2024\WorkSets\Python\Standards\Macros\`



Important Configuration Variables

MS_PYTHONSEARCHDIRECTORIES

A list of additional folders from which to harvest Python files when running keyin *Python Load <ms python script>.py*

Default:

\$(_USTN_WORKSETSTANDARDS)Macros\
e.g.

C:\ProgramData\Bentley\MicroStation

2024\Configuration\WorkSpaces\TMC

Winterschool

2024\WorkSets\Python\Standards\Macros\

Install Configure Visual Studio Code (VS Code)

Download from

<https://code.visualstudio.com/Download>

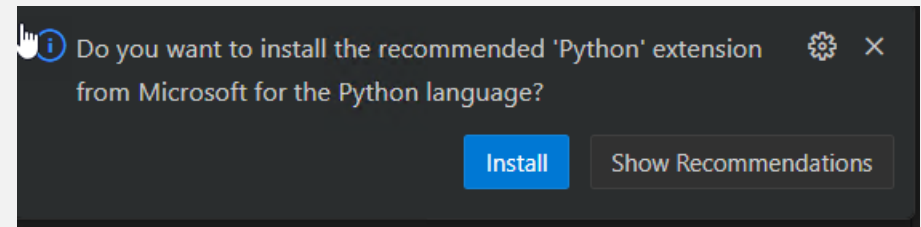
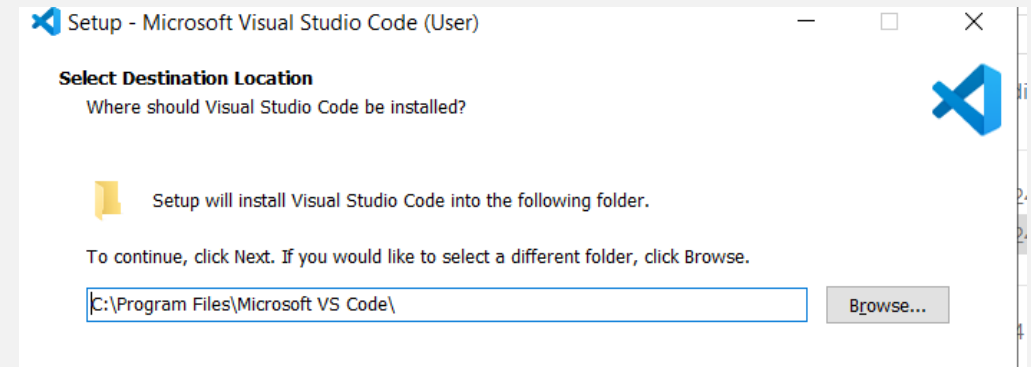
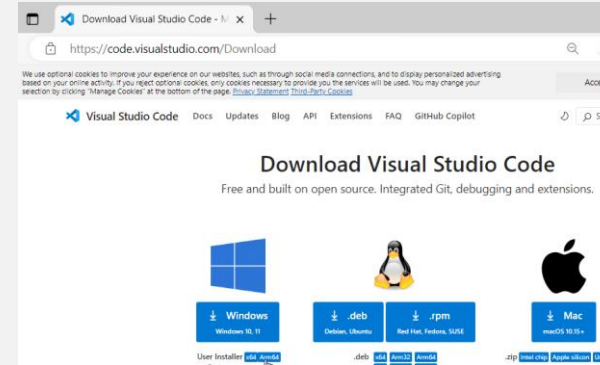
Install for editing/writing Python:

1. Visual Studio Code with VSCodeUserSetup-x64-1.95.3.exe into the folder
C:\Program Files\Microsoft VS Code

2. Add the Python extension:
<https://marketplace.visualstudio.com/items?itemName=ms-python.python>

More info:

<https://code.visualstudio.com/docs/languages/python>



Configure VS Code IntelliSense

Intellisense is the autocompletion of keyword and object properties, methods and functions. It may also include prompts.

Tells IDEs like **Visual Studio Code** where to find the additional Bentley Modules that need to be imported in each script

Set System Environment variable **PYTHONPATH**:

```
C:/ProgramData/Bentley/PowerPlatform  
Python/Examples/MicroStation/Intelli  
sense/;C:/Program  
Files/Bentley/MicroStation  
2024/MicroStation/
```

See KB article [MicroStation Python: VS Code IntelliSense](#)

Variable	Value
PROJ_LIB	C:\Program Files\PostgreSQL\13\share\contrib\postgis-3.0\proj
PSModulePath	%ProgramFiles%\WindowsPowerShell\Modules;C:\WINDOWS\system32\WindowsPowerSh
PYTHONPATH	C:/ProgramData/Bentley/PowerPlatformPython/Examples/MicroStation/Intellisense/;C:/Pro

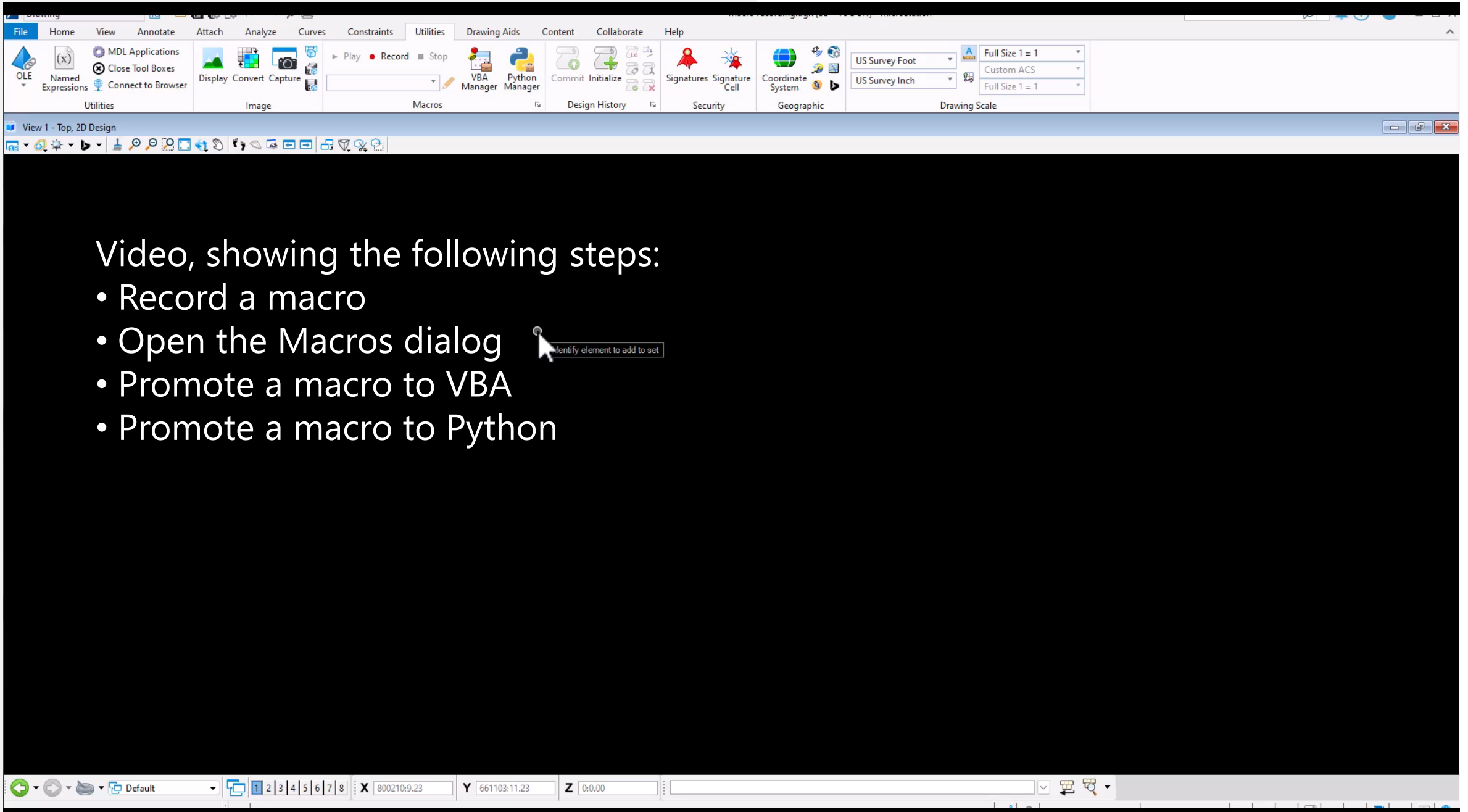
```
Function to select elements by its RGBColor  
...  
userColorIndex : int      color index value  
...  
def selectElementsbyColor(userColorIndex):  
    #Get active  
    ACTIVEMODEL (constant) ACTIVEMODEL: Any Ref  
    dgnModel = ACTIVEMODEL.GetDgnModel()  
    dgnfile = dgnModel.GetDgnFile()  
    #Get all graphical elements from the model  
    graphicalElements = dgnModel.GetGraphicElements()  
    selSetManager = SelectionSetManager.GetManager()  
  
    for elRefCnt in range(len(list(graphicalElements))):  
        perElementRef = list(graphicalElements)[elRefCnt]  
        elementId = perElementRef.GetElementId()  
        eeh = EditElementHandle(perElementRef, dgnModel)  
        eh = ElementHandle(perElementRef)  
  
        msElement = MSElement()  
        msElement = eeh.GetElement ()  
        isGraphics = msElement.ehdr.isGraphics  
        isInvisible = msElement.hdr.dhdr.props.b.invisible
```

Configure **VisualStudio Code**

- Set System Environment variable PYTHONPATH
- Start MS
- Open de Python Manager
- Select and edit a Python script

See KB article [MicroStation Python: VS Code IntelliSense](#)

```
1  # -*- coding: utf-8 -*-
2
3  ...
4  /*-----*
5  | $Copyright: (c) 2023 Bentley Systems, Incorporated. All rights reserved. $
6  +-----*/
7  ...
8  import math
9
10 from MSPyBentley import *
11 from MSPyBentleyGeom import *
12 from MSPyEObjects import *
13 from MSPyDgnPlatform import *
14 from MSPyMstn
15
16 # This sample
17 # 1. Invoke."
18 # 2. Input 2
19
20 #PyCadInputQue
21
22 # Alternative
23 PyCadInputQue
24
25 startPoint = DPoint3d(16.59365402793434185469, -1.52215075309855607522, 0.000000000000000000)
26
27 point = startPoint
```



Looking for patterns and similarities

```
Sub Bmrplaceline_macro()  
  Dim startPoint As Point3d  
  Dim point As Point3d, point2 As Point3d  
  Dim lngTemp As Long  
  Dim oMessage As CadInputMessage  
  
  ' Send a keyin that can be a command string  
  CadInputQueue.SendKeyin "ribbon grouppopup  
*\Home\Placement"  
  
  CadInputQueue.SendKeyin "PLACE SMARTLINE "  
  
  ' Coordinates are in master units  
  startPoint.X = 99445.2222846922  
  startPoint.Y = 80172.6725482137  
  startPoint.Z = 0#  
  
  ' Send a data point to the current command  
  point.X = startPoint.X  
  point.Y = startPoint.Y  
  point.Z = startPoint.Z  
  CadInputQueue.SendDataPoint point, 1  
  
  point.X = startPoint.X + 29.0018883704615  
  point.Y = startPoint.Y + 13.1697723690449  
  point.Z = startPoint.Z  
  CadInputQueue.SendDataPoint point, 1  
  
  ' Send a reset to the current command  
  CadInputQueue.SendReset  
  
  CommandState.StartDefaultCommand  
End Sub
```

VBA

```
1 from MSPyBentley import *  
2 from MSPyBentleyGeom import *  
3 from MSPyEObjects import *  
4 from MSPyDgnPlatform import *  
5 from MSPyMstnPlatform import *  
6  
7 startPoint = DPoint3d (0.0, 0.0, 0.0)  
8 point = DPoint3d (0.0, 0.0, 0.0)  
9  
10 PyCadInputQueue.SendKeyin ("ribbon grouppopup *\\Home\\Placement" )  
11 PyCadInputQueue.SendKeyin ("PLACE SMARTLINE " )  
12  
13 startPoint.x = 99445.22228469219407998025  
14 startPoint.y = 80172.67254821369715500623  
15 startPoint.z = 0.00000000000000000000  
16  
17 point.x = startPoint.x  
18 point.y = startPoint.y  
19 point.z = startPoint.z  
20 PyCadInputQueue.SendDataPoint (point, 1)  
21  
22 point.x = startPoint.x + 29.00188837046152912080  
23 point.y = startPoint.y + 13.16977236904494930059  
24 point.z = startPoint.z  
25 PyCadInputQueue.SendDataPoint (point, 1)  
26  
27 PyCadInputQueue.SendReset()  
28  
29 PyCommandState.StartDefaultCommand()
```

MSPython

Looking for patterns and similarities

Bentley delivered Modules

- Mandatory
- Specific order

```
1 from MSPyBentley import *
2 from MSPyBentleyGeom import *
3 from MSPyECOobjects import *
4 from MSPyDgnPlatform import *
5 from MSPyDgnView import *
6 from MSPyMstnPlatform import *
7 startPoint = DPoint3d (0.0, 0.0, 0.0)
8 point      = DPoint3d (0.0, 0.0, 0.0)
9
10 PyCadInputQueue.SendKeyin ("ribbon group popup *\\Home\\Placement" )
11 PyCadInputQueue.SendKeyin ("PLACE SMARTLINE " )
12
13 startPoint.x = 99445.22228469219407998025
14 startPoint.y = 80172.67254821369715500623
15 startPoint.z = 0.00000000000000000000
16
17 point.x = startPoint.x
18 point.y = startPoint.y
19 point.z = startPoint.z
20 PyCadInputQueue.SendDataPoint (point, 1)
21
22 point.x = startPoint.x + 29.00188837046152912080
23 point.y = startPoint.y + 13.16977236904494930059
24 point.z = startPoint.z
25 PyCadInputQueue.SendDataPoint (point, 1)
26
27 PyCadInputQueue.SendReset()
28
29 PyCommandState.StartDefaultCommand()
```

MSPython

Looking for patterns and similarities

Element Types

- Names
- Required inputs
- No declarations

In Python hoeven variabelen niet expliciet gedeclareerd te worden voordat je ze gebruikt. Je kunt een variabele eenvoudigweg toewijzen door een waarde eraan toe te kennen

```
1 from MSPyBentley import *
2 from MSPyBentleyGeom import *
3 from MSPyECObjects import *
4 from MSPyDgnPlatform import *
5 from MSPyMstnPlatform import *
6
7 startPoint = DPoint3d (0.0, 0.0, 0.0)
8 point      = DPoint3d (0.0, 0.0, 0.0)
9
10 PyCadInputQueue.SendKeyin ("ribbon grouppopup *\\Home\\Placement" )
11 PyCadInputQueue.SendKeyin ("PLACE SMARTLINE " )
12
13 startPoint.x = 99445.22228469219407998025
14 startPoint.y = 80172.67254821369715500623
15 startPoint.z = 0.00000000000000000000
16
17 point.x = startPoint.x
18 point.y = startPoint.y
19 point.z = startPoint.z
20 PyCadInputQueue.SendDataPoint (point, 1)
21
22 point.x = startPoint.x + 29.00188837046152912080
23 point.y = startPoint.y + 13.16977236904494930059
24 point.z = startPoint.z
25 PyCadInputQueue.SendDataPoint (point, 1)
26
27 PyCadInputQueue.SendReset()
28
29 PyCommandState.StartDefaultCommand()
```

MSPython

Looking for patterns and similarities

Interactions

- Mouse actions
- Required inputs

```
1 from MSPyBentley import *
2 from MSPyBentleyGeom import *
3 from MSPyEObjects import *
4 from MSPyDgnPlatform import *
5 from MSPyMstnPlatform import *
6
7 startPoint = DPoint3d (0.0, 0.0, 0.0)
8 point      = DPoint3d (0.0, 0.0, 0.0)
9
10 PyCadInputQueue.SendKeyin ("ribbon grouppopup *\\Home\\Placement" )
11 PyCadInputQueue.SendKeyin ("PLACE SMARTLINE " )
12
13 startPoint.x = 99445.22228469219407998025
14 startPoint.y = 80172.67254821369715500623
15 startPoint.z = 0.00000000000000000000
16
17 point.x = startPoint.x
18 point.y = startPoint.y
19 point.z = startPoint.z
20 PyCadInputQueue.SendDataPoint (point, 1)
21
22 point.x = startPoint.x + 29.00188837046152912080
23 point.y = startPoint.y + 13.16977236904494930059
24 point.z = startPoint.z
25 PyCadInputQueue.SendDataPoint (point, 1)
26
27 PyCadInputQueue.SendReset()
28
29 PyCommandState.StartDefaultCommand()
```

MSPython

Looking for patterns and similarities

```
Sub Bmrplaceline_macro()  
  Dim startPoint As Point3d  
  Dim point As Point3d, point2 As Point3d  
  Dim lngTemp As Long  
  Dim oMessage As CadInputMessage  
  
  ' Send a keyin that can be a command string  
  CadInputQueue.SendKeyin "ribbon grouppopup  
*\Home\Placement"  
  
  CadInputQueue.SendKeyin "PLACE SMARTLINE "  
  
  ' Coordinates are in master units  
  startPoint.X = 99445.2222846922  
  startPoint.Y = 80172.6725482137  
  startPoint.Z = 0#  
  
  ' Send a data point to the current command  
  point.X = startPoint.X  
  point.Y = startPoint.Y  
  point.Z = startPoint.Z  
  CadInputQueue.SendDataPoint point, 1  
  
  point.X = startPoint.X + 29.0018883704615  
  point.Y = startPoint.Y + 13.1697723690449  
  point.Z = startPoint.Z  
  CadInputQueue.SendDataPoint point, 1  
  
  ' Send a reset to the current command  
  CadInputQueue.SendReset  
  
  CommandState.StartDefaultCommand  
End Sub
```

VBA

```
1 from MSPyBentley import *  
2 from MSPyBentleyGeom import *  
3 from MSPyEObjects import *  
4 from MSPyDgnPlatform import *  
5 from MSPyMstnPlatform import *  
6  
7 startPoint = DPoint3d (0.0, 0.0, 0.0)  
8 point = DPoint3d (0.0, 0.0, 0.0)  
9  
10 PyCadInputQueue.SendKeyin ("ribbon grouppopup *\\Home\\Placement" )  
11 PyCadInputQueue.SendKeyin ("PLACE SMARTLINE " )  
12  
13 startPoint.x = 99445.22228469219407998025  
14 startPoint.y = 80172.67254821369715500623  
15 startPoint.z = 0.00000000000000000000  
16  
17 point.x = startPoint.x  
18 point.y = startPoint.y  
19 point.z = startPoint.z  
20 PyCadInputQueue.SendDataPoint (point, 1)  
21  
22 point.x = startPoint.x + 29.00188837046152912080  
23 point.y = startPoint.y + 13.16977236904494930059  
24 point.z = startPoint.z  
25 PyCadInputQueue.SendDataPoint (point, 1)  
26  
27 PyCadInputQueue.SendReset()  
28  
29 PyCommandState.StartDefaultCommand()
```

MSPython



Create a script using recording

- Start recording
- Convert macro to VBA and to Python
- Adjust the script
- See [Part 6 Recording a Macro](#)



Macro Recording Conversion

Set variables for **number of points** and the **max x** and **max y ranges**

See Part 7 Customising the Macro

```
#Limits
```

```
no_of_points = int(100)
```

```
max_x = int(10000)
```

```
max_y = int(10000)
```

Macro Recording Conversion

Added a **random** number generator to create **lists of start and end points**

See Part 7 Customising the Macro

```
import random # Import 3rd party Libraries
```

```
# Define the list that will be filled tuples of coordinates  
endPoint_List = []  
startPoint_List = []
```

```
#Create a list with tuples  
for _ in range (no_of_points):
```

```
    start_coord = [(random.randint(0, max_x), random.randint(0,max_y))]
```

```
    end_coord = [(random.randint(0, max_x), random.randint(0,max_y))]  
    startPoint_List.append (start_coord)
```

```
    endPoint_List.append (end_coord)
```

Macro Recording Conversion

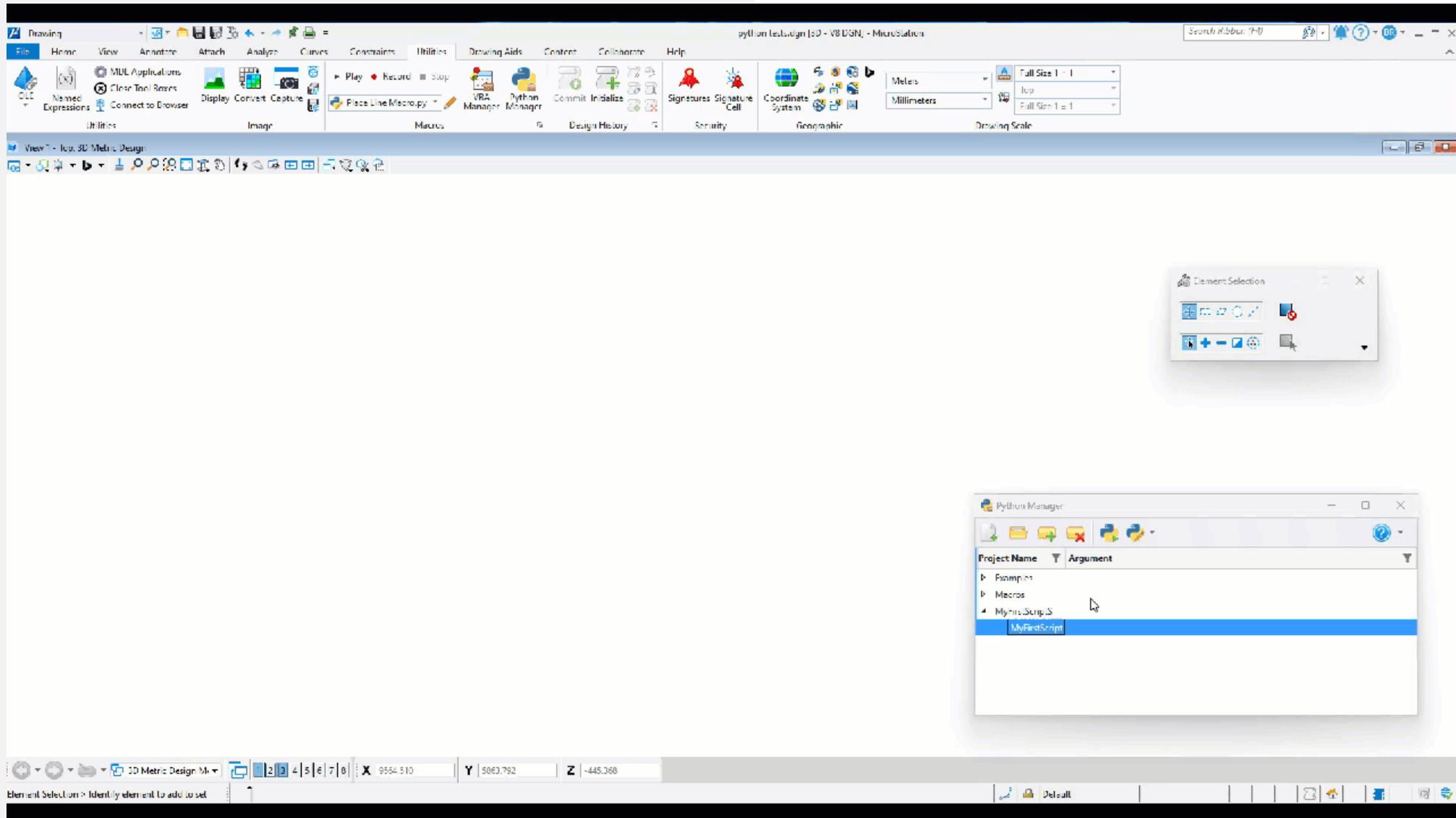
Added a **loop** to draw the lists

See Part 7 Customising the Macro

```
PyCadInputQueue.SendKeyin ("Place Smartline " )

for x in range( no_of_points -1):
    startPoint = DPoint3d (startPoint_List[x][0][0],startPoint_List[x][0][1], 0.0)
    point = startPoint
    PyCadInputQueue.SendDataPoint (point, 1)
    startPoint = DPoint3d (endPoint_List[x][0][0],endPoint_List[x][0][1], 0.0)
    point = startPoint
    PyCadInputQueue.SendDataPoint (point, 1)
    PyCadInputQueue.SendReset()
```

Macro Recording Conversion



Macro Recording Conversion

Could it be Faster and more performant, could I create an input dialog?

Improvements and performance

[See Part 8 Improving Performance](#)

```
# Create line element

status=LineHandler.CreateLineElement(
    eeh, None, seg, ACTIVEMODEL.Is3d(), ACTIVEMODEL)

if BentleyStatus.eSUCCESS != status:
    return False

if BentleyStatus.eSUCCESS != eeh.AddToModel():
    return False
```

Looked at Example code for specific means of doing things.

Found the create line method.

Requires:

- EditElementHandle
- A Dsegment3D
- An Active Model type
- An Active Model

Improvements and performance

In MicroStation, an **EditElementHandle** is a crucial concept when working with Python for scripting and automation. It serves as a writable handle to an element within a design file, allowing you to interact with and modify the element programmatically.

```
eeh=EditElementHandle() # initialises an element
```

Improvements and performance

In MicroStation, a DSegment3d is a data structure used to represent a **3D line segment**. It is defined by two points: **a start point and an end point**.

```
# initialises a line element requireing start and end points  
seg=DSegment3d(point1, point2)
```

Improvements and performance

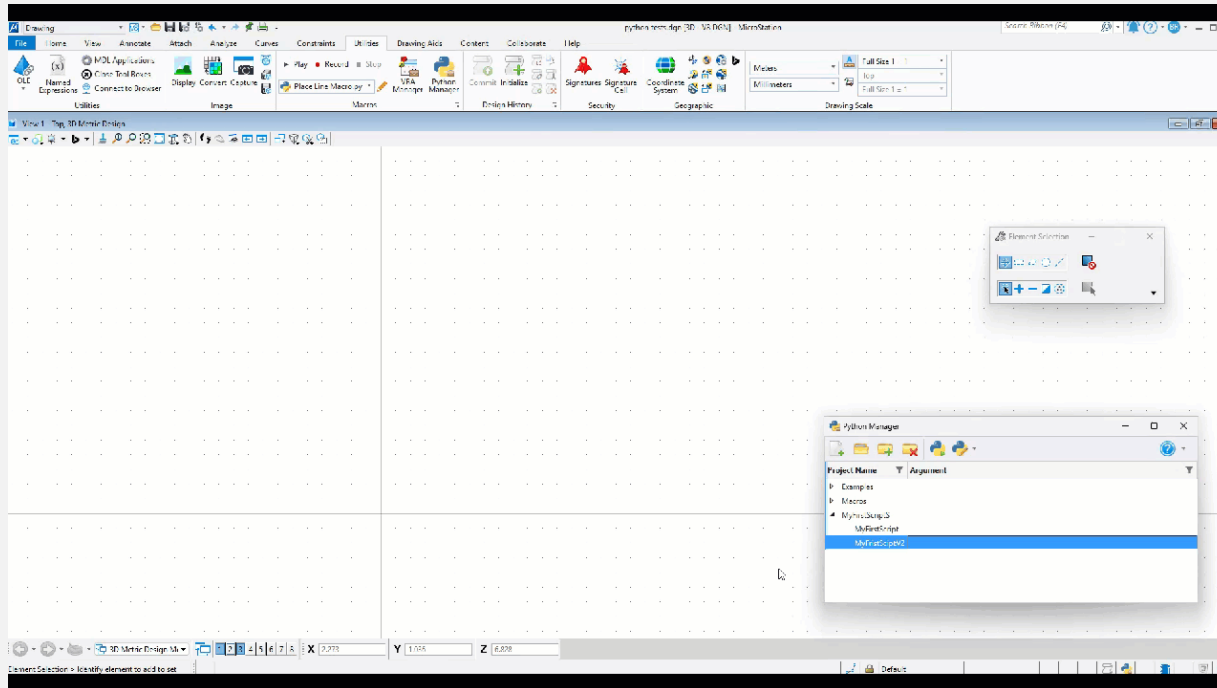
Data must be written to a file or a model within a file. Therefore, the open or “active” model needs to be captured and included in many commands that create elements or make changes to models or dgns.

```
ACTIVEMODEL=ISessionMgr.ActiveDgnModelRef  
  
if ACTIVEMODEL is None:  
    .....  
    return False
```

Improvements and performance

Using MSPython integration and techniques

1. No longer Simulating Mouse clicks and Accudraw Compass updates
2. Content is drawn much faster



Adding a UI

MSPython has two major libraries or modules for UI **Tkinter** and **Qt**.

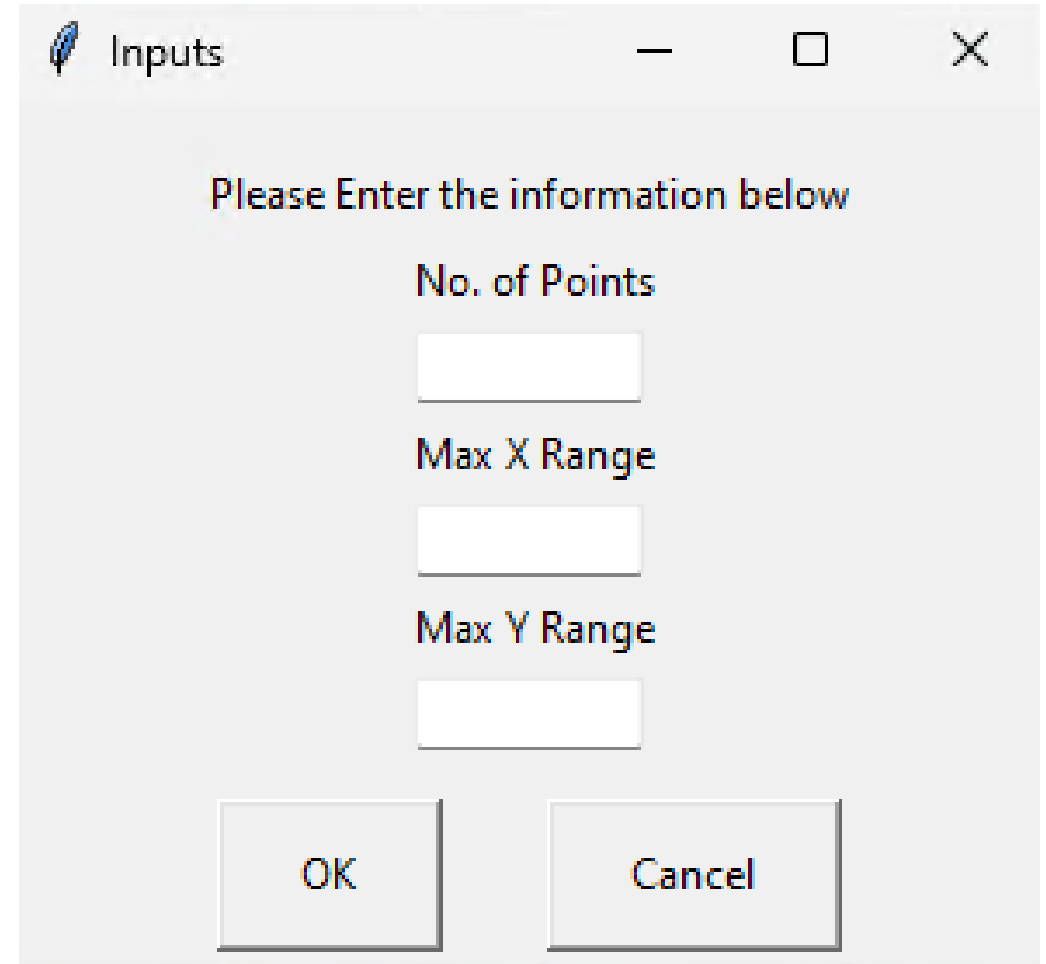
Other modules are available from Python and can be imported in your projects

I needed an input dialog for my project.

[Part 9 Adding inputs and UI](#)

or

[MicroStation Python: Create a User Interface](#)



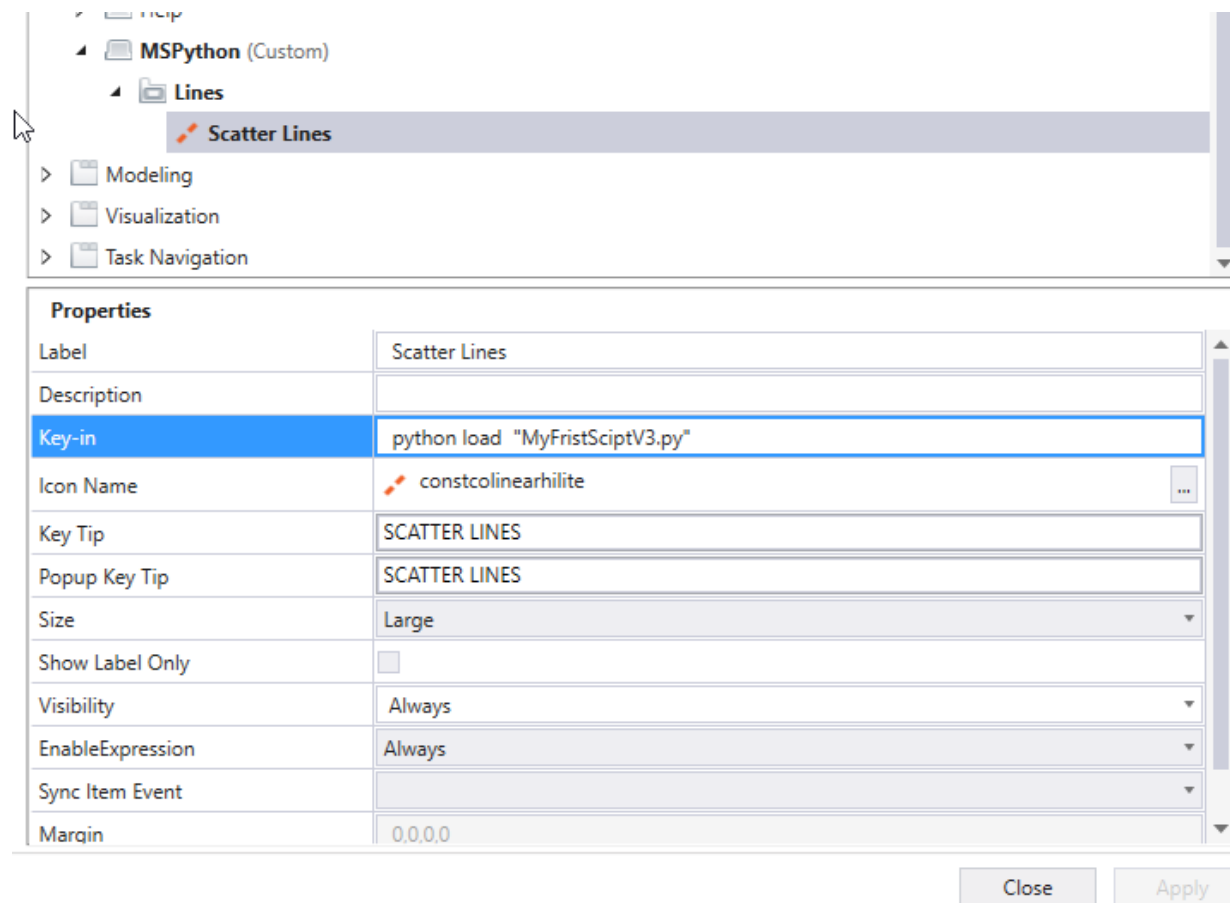
Customising the RIBBON

Python scripts loaded via simple keyin
Python Load <Script_Name>.py

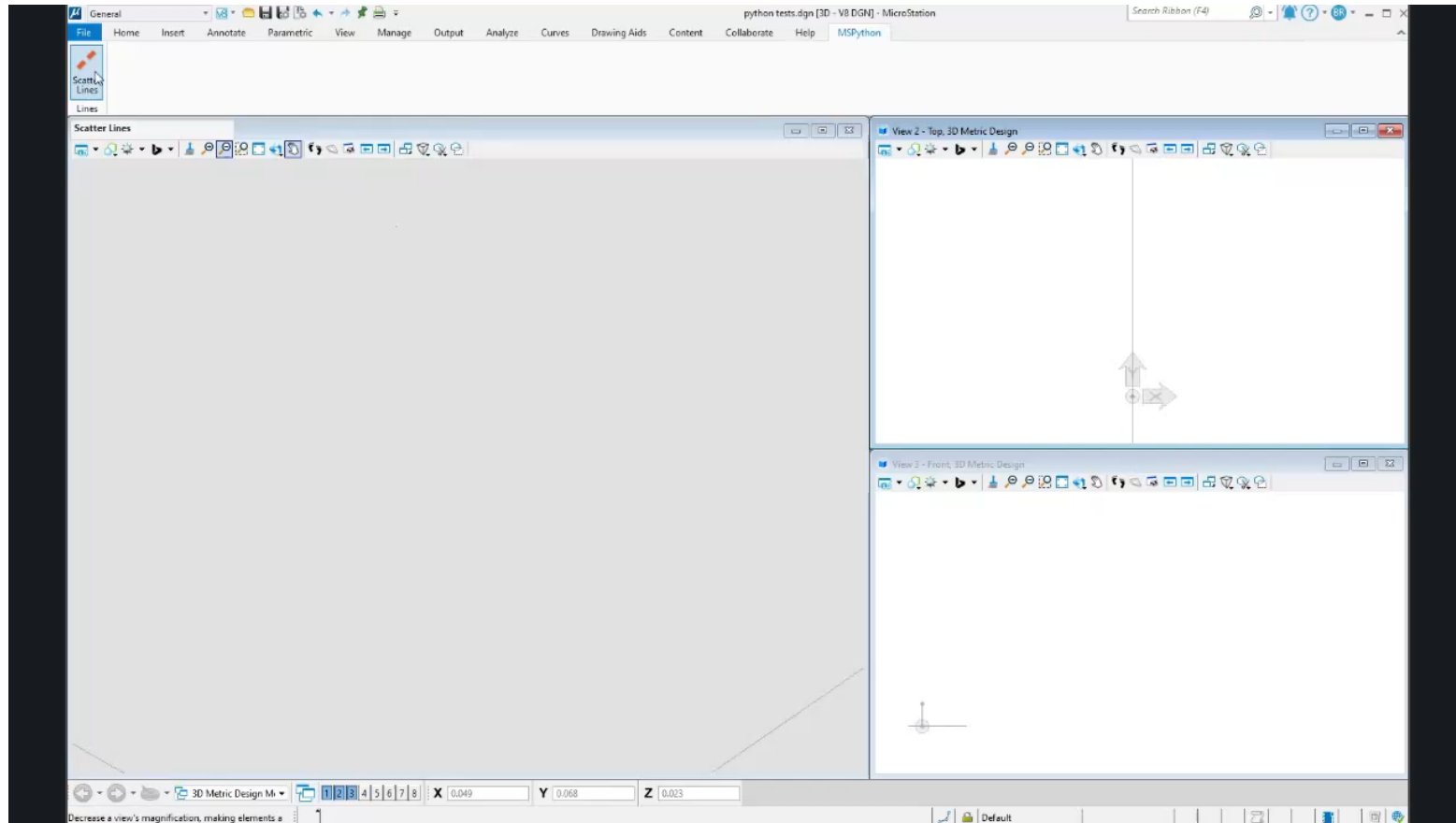
E.g. an example of using a variable to load a python script:

```
$ python load  
$(MS_PYTHONSAMPLES)MicroStation\  
Macro\PlaceLine.py
```

Make sure the custom RIBBON is in path
MS_GUIDGNLIBLIST

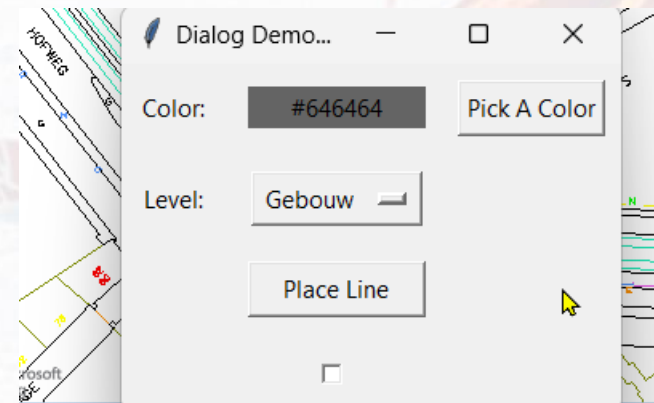


Final Version



Create a new script

- Create a new script
- Copy the code from [MicroStation Python: Create a User Interface](#)



Here is the complete script

```
from MSPyBentley import *
from MSPyBentleyGeom import *
from MSPyEObjects import *
from MSPyDgnPlatform import *
from MSPyDgnView import *
from MSPyMstnPlatform import *

import ctypes
from tkinter import *
from tkinter import colorchooser

root = Tk()

# Message box function
def MsgBox (title, text, style):
    return ctypes.windll.user32.MessageBoxW (0, text, title, style)

# RGB to String function
def rgb_to_string(rgb_tuple):
    return ', '.join(map(str, rgb_tuple))

# Hex to RGB function
def hex_to_rgb(hex_color):
```


Resources

[MicroStation - MicroStation Python Wikis - Communities \(service-now.com\)](#)

[MicroStation Programming Blog - My Journey with MSPython - Communities \(service-now.com\)](#)

[BentleySystems/MicroStationPython: MicroStation Python Implementation, Examples, Tests, Build Scripts \(github.com\)](#)

[MSPython API Docs: Overview | iTwin Platform \(bentley.com\)](#)

For learning Python : YouTube, Degreed and LinkedIn

Python: [Documentation](#) | [API Presentations](#) | [FAQs](#) | [GitHub](#) | [Samples](#) | [Wikis](#) | [Blogs](#)



A Journey with Python for MicroStation or MSPython

Thank you!

Kees van Prooijen
Kees.vanProoijen@Bentley.com