Programing with Python in MD products

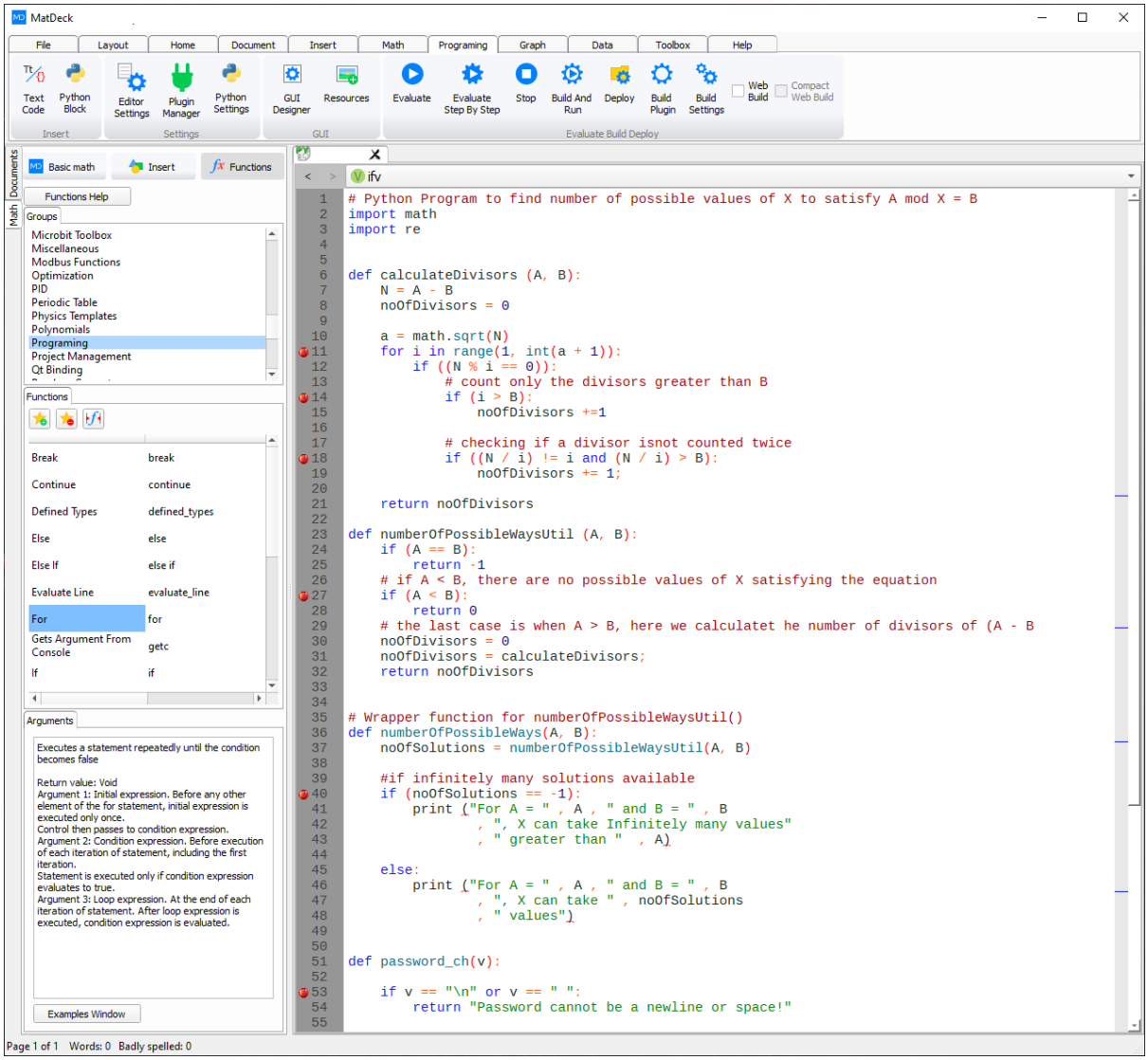
Contents

Users can initiate Python in 3 different ways in MatDeck products.

- Python IDE
- Python code blocks within documents
- Call functions

Python IDE

The image below shows classical MD Python IDE. Here, a dedicated Python Script file is used with breakpoints, step by step evaluation, Deploy exe and other features.



Python Block

Users can edit and code Python directly in MD documents via Python Blocks, allowing them you integrate their Python code with other MD features and functions. The image below is a screenshot of a MD document which utilizes Python and MD features. MD Graphs also change in real time.

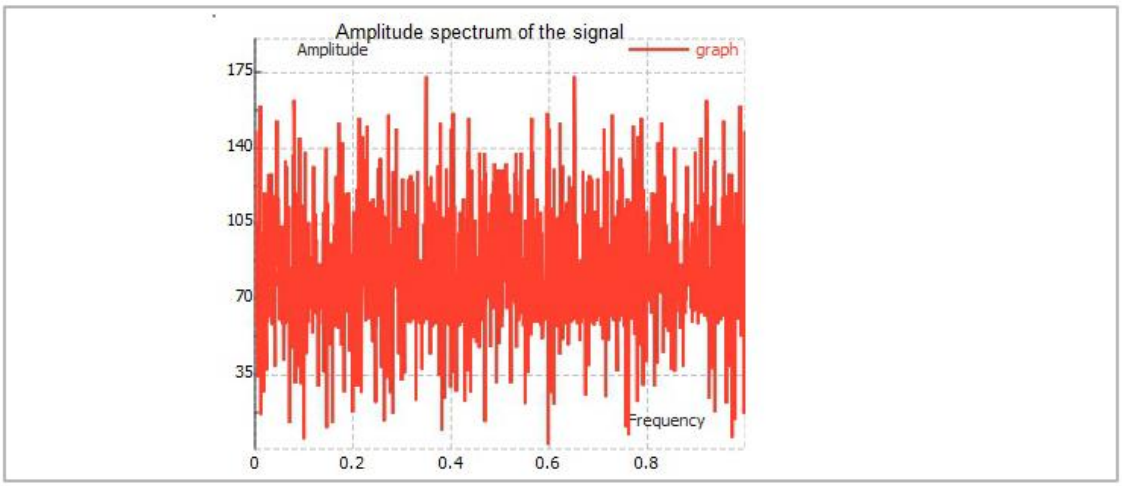
Transferring variables between MatDeck and Python

In this example, we will illustrate how variables defined in MatDeck script are transfered to Python, and vice versa. Furthermore, we will also demonstrate the use of graphs with exchanged variables. The demonstration is performed by the calculating the FFT of the random signal in both MatDeck, and Python.

In order to calculate FFT in Python, it is necessary to import scipy package, and matplotlib package is required to plot graph. Therefore it is necessary to add these two packages to successfully run example below. In Windows, packages can be added using cmd window via commands such as:
>python -m pip install scipy
>python -m pip install matplotlib

Here is how we generate the random signal and calculate FFT in MatDeck. We use a MatDeck 2-D graph to show the amplitude spectrum of the signal.

```
1  nn := 4096 //length of the signal
2  x_in := normrandvec(0, 1, nn) //signal as random noise
3  X_in1 := fft1(x_in)
4  freq := ynodes(f, 0, 1, nn) //frequency axis
5  graph := join_mat_cols(freq, abs(X_in1)) //amplitude spectrum of the signal
```



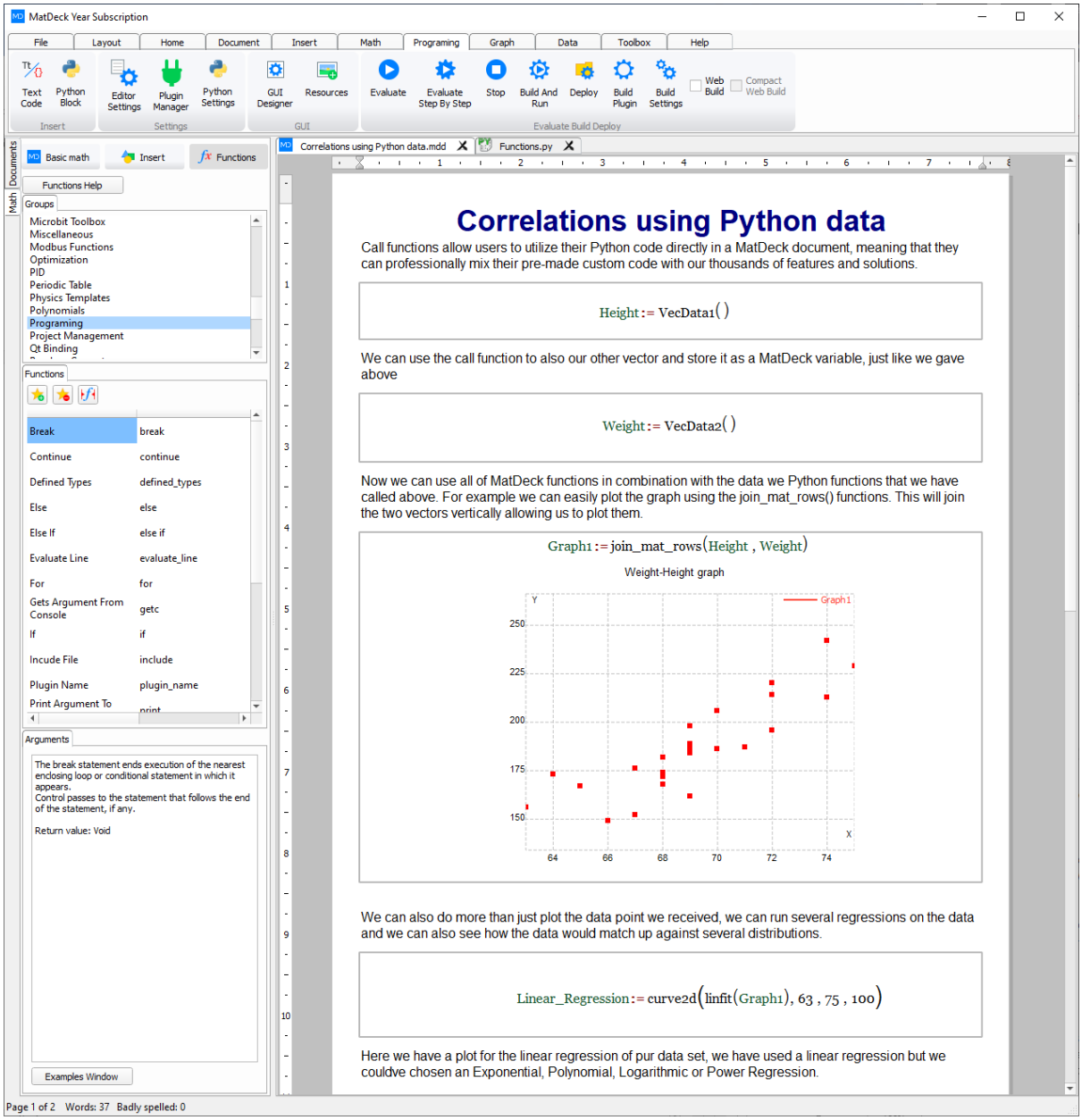
Here is how Python is used to calculate FFT. The Python code is written inside the MatDeck document, using the MatDeck variables ss and freq. The amplitude spectrum is plotted using the Python graph.

```
6  ss := 0 //Variable defined to store result in Python
7
8  #py
9  #This is python code
10 from scipy.fftpack import fft #we need fft function
11 ss1 = fft(x_in)
12 ss=ss1.tolist() #convert ss1 to list which is converted in C++ vector
13 import matplotlib.pyplot as plt  #we need pyplot to plot result
14 #It is possible to plot signal in Python, as well.
15 plt.plot(freq, abs(ss1))
```

Above is a perfect example of how the LabDeck Notes Engine allows user to combine Python Code with MatDeck Script as well as MD GUIs. This allows you to create interactive WYSIWYG and reproducible examples.

Call Functions

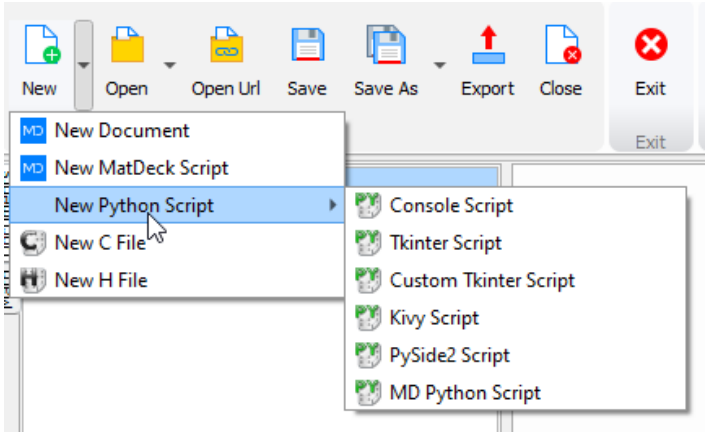
If a Python file is saved within the same directory as your MD document, users can call over functions from the Python code directly in the MD document on the canvas. In the example below, the Python functions VecData1 and VecData2 are called from the Python file in the same directory and used in the canvas.



Getting started with Python IDE

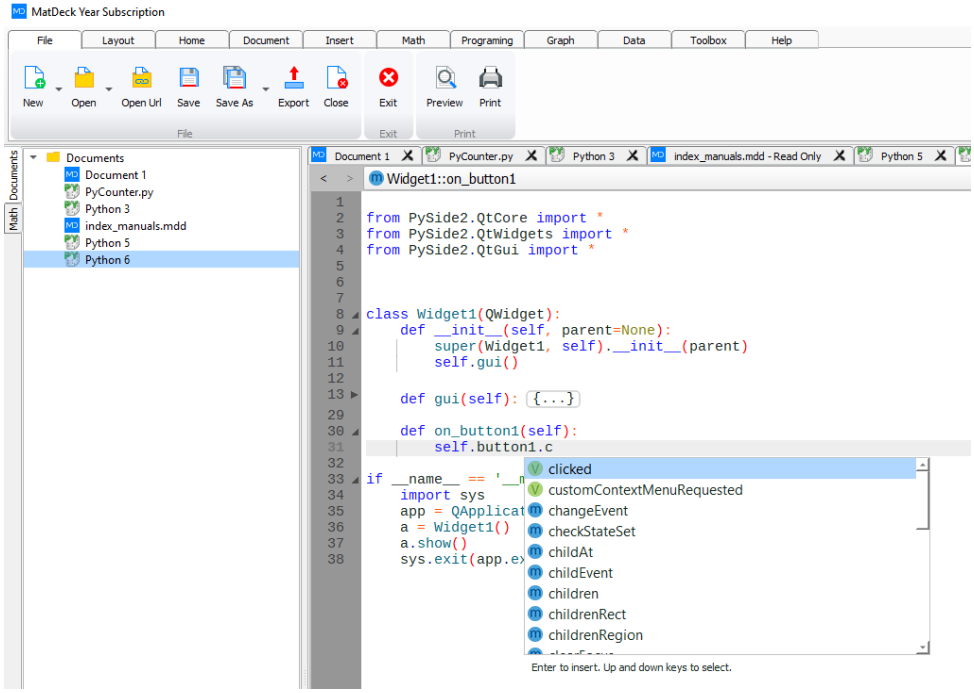
The first step in getting started with a Python IDE is to open the correct one. This is done by clicking on the small triangle on the left-hand side of the Open Icon.

Following this a menu will open up and you can choose which IDE you would use. The following IDE will be in the drop-down menu: Console, Tkinter, Kivy, Custom Tkinter, PySide2 and MD Python.



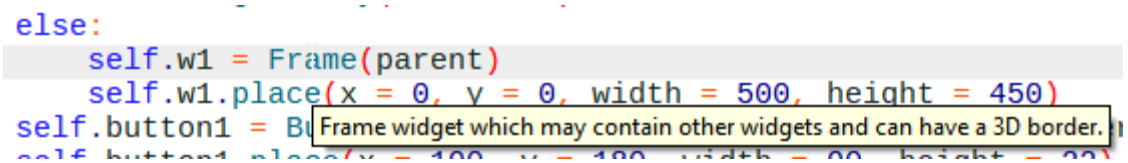
The console script will open an entirely blank Python Script which is designed to develop console programs and not visual applications or GUIS. The other script will open their corresponding GUI Designers as well as library specific IDEs.

Library-specific IDEs allow to produce accurate and relevant code suggestion while you type, this gives you less errors and more time to code. Our code suggestions and auto-complete will help you speed up you're coding by suggesting relevant variable, commands and Python functions.

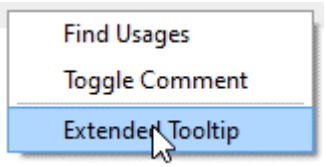


As you can see, our IDEs suggest the most suitable variables and also functions to speed up coding.

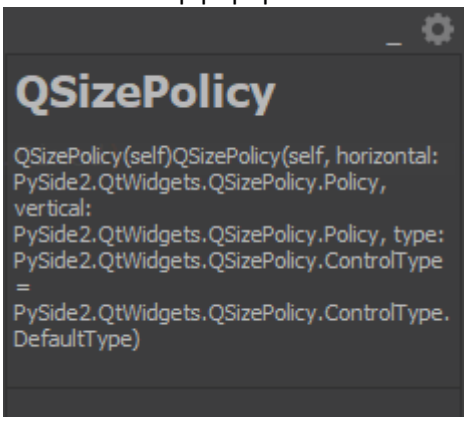
Extended Tooltip - Extended tooltip give the user information on any Python function that they hover there mouse over.



As you can see when we hover over the Frame() function, our tooltip appear giving us information on what the function does. We can also have the Extended Tooltip as a popup or even docked on the left-hand side of the IDE. To do this we need to left click on the IDE and select the Extended Tooltip Option.



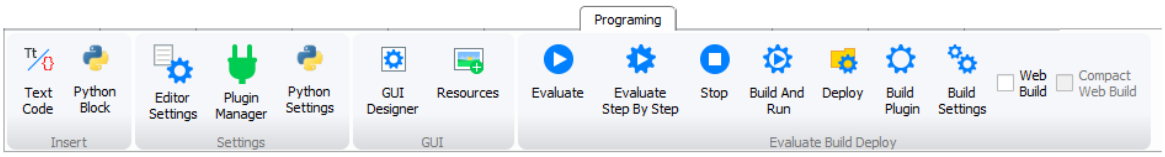
Once the Extended Tooltip is on, you can change how the tooltip is present using the settings icon on the top right hand side of any Extended Tooltip popup.



Above we can see the Docking in dark mode, as mentioned before to change or close it you will need to click the settings icon on the top right hand side.

Programming toolbar - Python

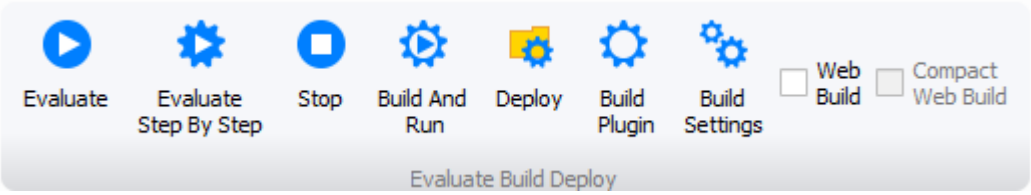
The programming toolbar is located at the top of the MatDeck document and houses various tools and functions used to program in MatDeck.



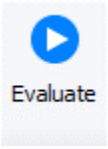
Here, users can manipulate and work with Python through different the options available.

Programming toolbar - Evaluate Build and Deploy

In all of our IDE, we have several options to run and evaluate your Python code.



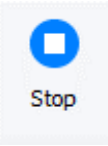
Evaluate - The first option and most conventional method is the Evaluate button, it will run the Python program without any interruptions or modifications.



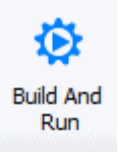
Please Note: The Evaluate button will offer debugging information such as any syntax error whereas Build and Run will only offer this information during the Build and Run and not during the runtime.

Evaluate Step By Step - The other method is to use the Evaluate Step By Step button. This will run a debugger on the Python code while stopping at intervals where code blocks/functions or key points in your code. This is done to better help you locate and resolve any bugs or logic issues.

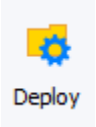
Stop – This is used to cease any kind of evaluation that is occurring on your Python code. It also works to stop the debugger and the Evaluate Step By Step.



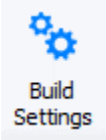
Build And Run – Evaluates the program as a sperate and independent application without creating a .exe file, it has the same effect as Evaluate but the program will not be run as a sub-program of MatDeck and the application will be allocated separate memory and run as the same speed as a .exe file, whereas Evaluate will run the program slower. However, Evaluate will start the program faster compared to Build and Evaluate, but the program itself will always be faster with Build and Run.



Deploy - The Deploy button will package your Python code as a .exe application file. This will allow you to share your code as an independent application on an unlimited amount of windows devices. Please note that you will need to add any additional files your application needs using the Build Settings Button. This will not run the program



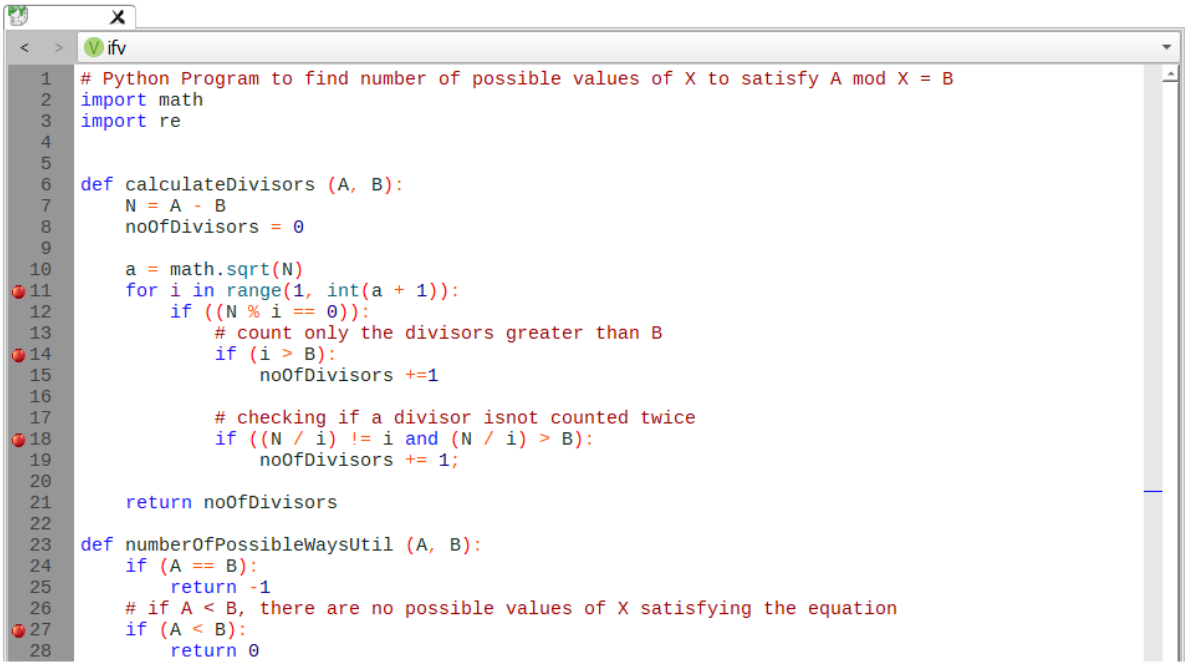
Build Settings - Build Settings is used to configure your code when it is packaged into a application. Here you will need to add any additional files that your code will call. This includes any images, Python files or any other file.



You can also choose whether or not you will deploy any MD Instruments as Images or as GUI Widgets.

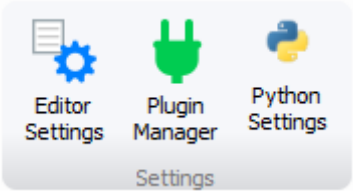
Breakpoints

Breakpoints in MatDeck are selected lines in Python code where the IDE will stop compiling. To add a Breakpoint right click on the left-hand side of number in the programming line, to remove them need to click on the red dot. Breakpoints are illustrated by red dots next to the code line number.

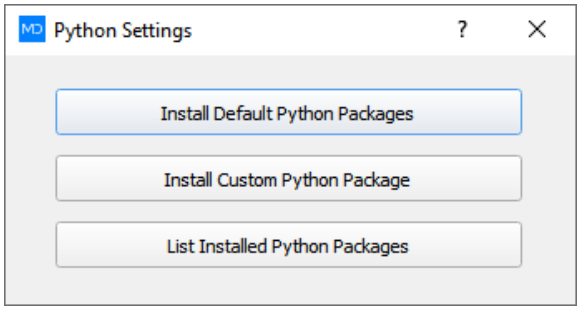


The IDE will compile each line of code until a Breakpoint is reached. Once, a Breakpoint has been reached, the user will have to evaluate the program. Similarly, the IDE will compile all the code after the Breakpoint until it reaches another Breakpoint. To reach a Breakpoint and execute the code, you will need to Evaluate the document.

Programming Toolbar - Settings

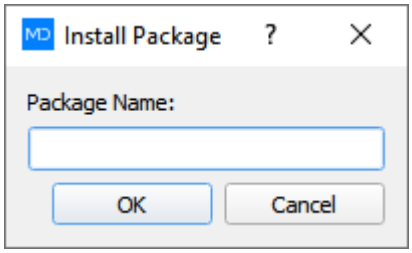


Python Settings - Python Settings allow the user to download any Python libraries without the need of any command prompt or difficulties. There are three options: Install Default Python Packages, Install Custom Python Packages and List Installed Python Packages.



The Install Default Python Packages will download all the necessary Python Modules/Libraries that MatDeck needs.

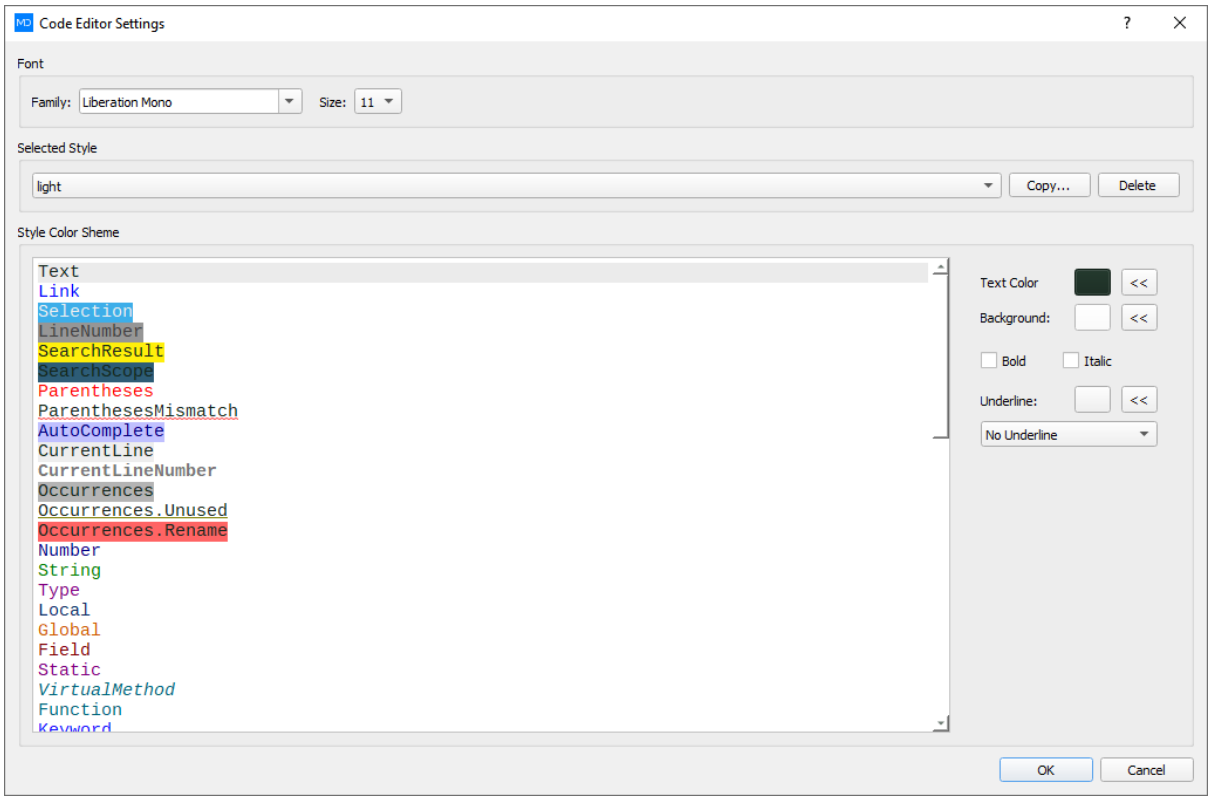
The Install Custom Python Packages will prompt a small message box with a text box.



Here, you will only need to add the name of the library, nothing else. From there, MatDeck will download the library for you.

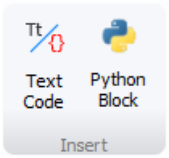
The List Installed Python Packages will list all installed Python packages as well as what version is installed.

Editor Settings - Similar to other IDEs, MatDeck provides the user with editor settings. Here, users can edit the aesthetic features of the IDE itself.

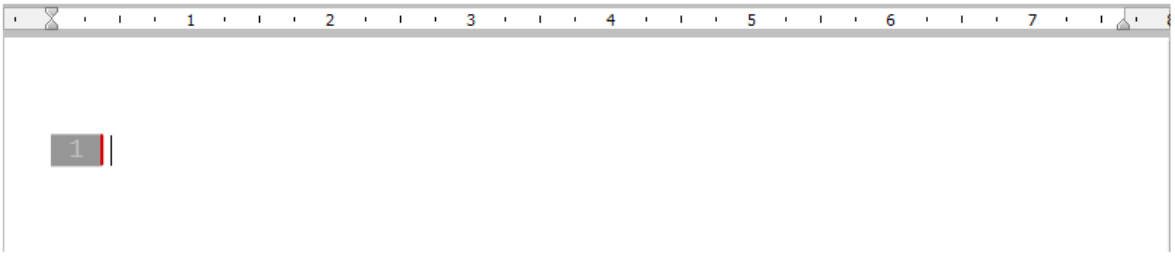


Users can edit the font family that is used when Python code is written and the Style Colour Scheme window below shows examples of what different occurrences of code would look like in the selected font family.

Programming Toolbar – Python Blocks



The insert box of the Programming toolbar allows users to place Text Code and Python Blocks directly into MD documents. Text Code blocks are used to initialise MatDeck script anywhere in MD documents where Python blocks can be dropped in. Simply press the Text Code button for a programming line to be established as such below.



Here, users can code in MD script directly into the MD document and can use MD script with other MD functions and features. To insert a Python block into the Text Code block, simply make sure the code line is selected and press Python Block.

A Python code block such as the one above will be created. Any code written within the #py and ### tags will be in executable Python. Python code written within the block can be directly used with the rest of the MD document and integrated with other MatDeck features such as below.

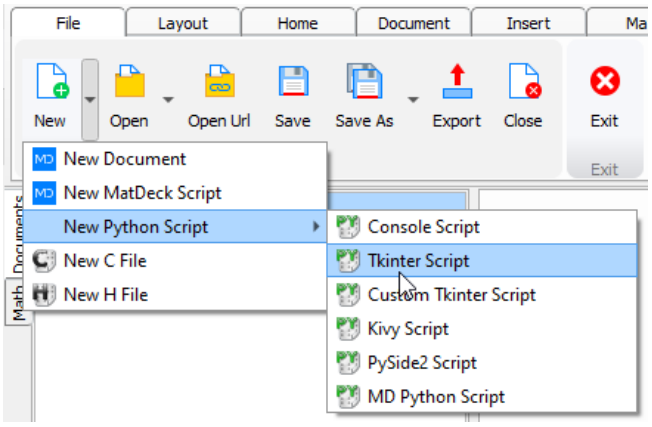
Using Python Blocks, we can combine our code with narrative text-editing, MatDeck Script, GUIs as well as any custom GUIs you have made, all of this can be combine in one document.

```
#py
import numpy as np
import matplotlib.pyplot as plt


fig, (ax1, ax2) = plt.subplots(2, 1)
# make a little extra space between the subplots
fig.subplots_adjust(hspace=0.5)

dt = 0.01
t = np.arange(0, 30, dt)

# Fixing random state for reproducibility
np.random.seed(19680801)


nse1 = np.random.randn(len(t))                 # white noise 1
nse2 = np.random.randn(len(t))                 # white noise 2
r = np.exp(-t / 0.05)

cnse1 = np.convolve(nse1, r, mode='same') * dt   # colored noise 1
cnse2 = np.convolve(nse2, r, mode='same') * dt   # colored noise 2

# two signals with a coherent part and a random part
s1 = 0.01 * np.sin(2 * np.pi * 10 * t) + cnse1
s2 = 0.01 * np.sin(2 * np.pi * 10 * t) + cnse2

ax1.plot(t, s1, t, s2)
ax1.set_xlim(0, 5)
ax1.set_xlabel('Time')
ax1.set_ylabel('s1 and s2')
ax1.grid(True)

cxy, f = ax2.csd(s1, s2, 256, 1. / dt)
ax2.set_ylabel('CSD (dB)')
plt.show()
###
```

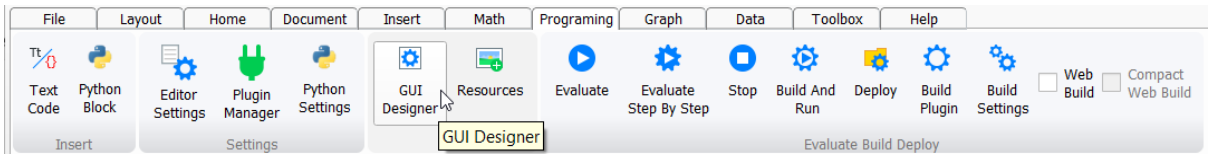
Opening GUI Designer

Please Note – There is a dedicated GUI Designer manual which explains GUI Designer use in greater detail. For more information, please refer to the GUI Designer manual.

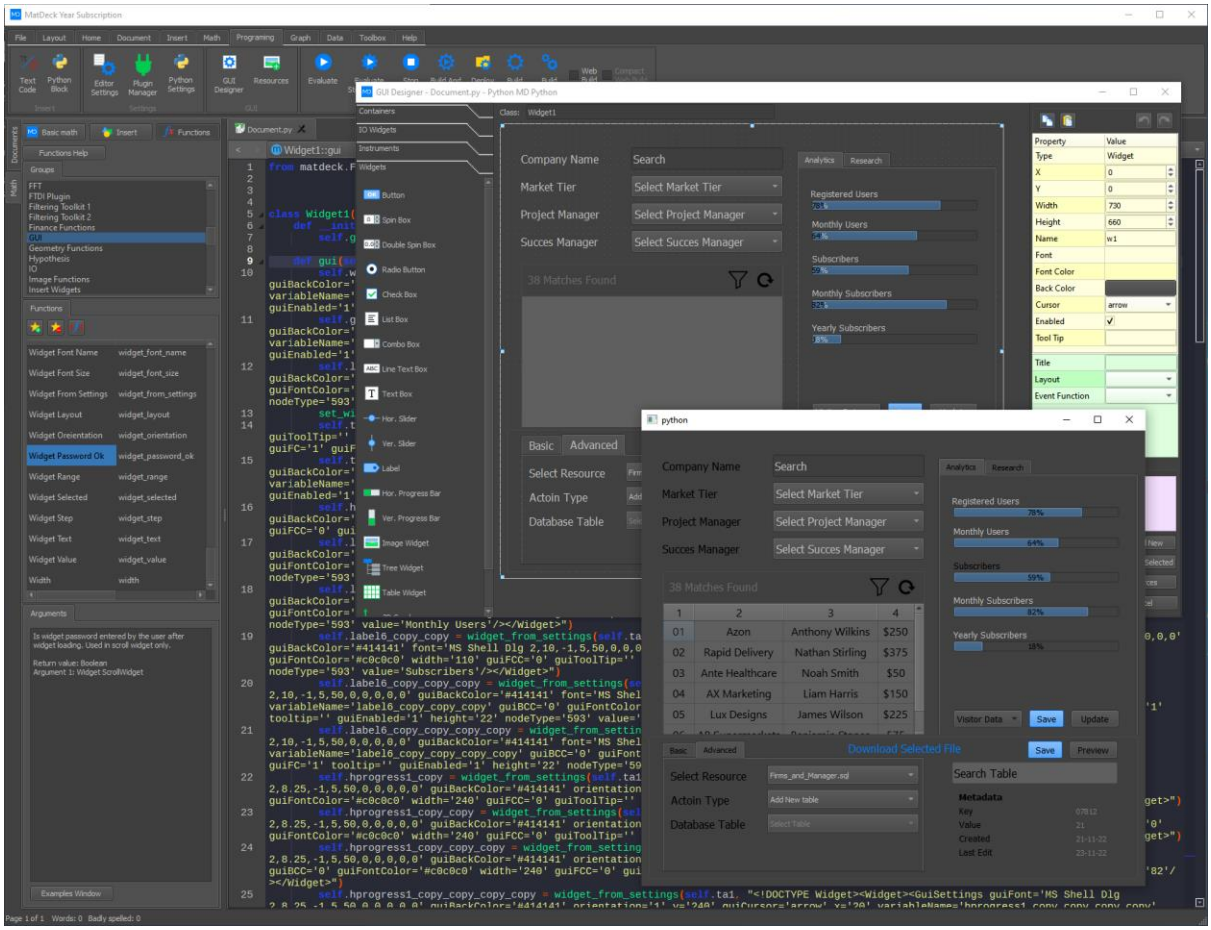
The first thing you need to do to open the MD Python GUI Designer is to select the MD Python Script. This is done by hovering over the small triangle next to the New icon.



To open the MD Python GUI Designer, go to Programming tab and press the ‘GUI Designer’ button



A new window will open, which will contain a list of all the widgets that have been created in the current MD Python Script (Picture 2). From this window you can create a new widget, edit or delete the existing one. To open the MD Python GUI Designer, you will need to click the New button, the edit button will only open the MD Python GUI Designer if there is a widget beforehand.

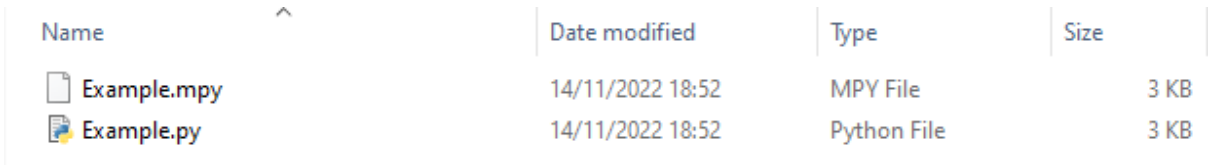


The image above shows a screenshot of how the GUI Designer can be used in Python.

MPY GUI Designer Files in MD Products

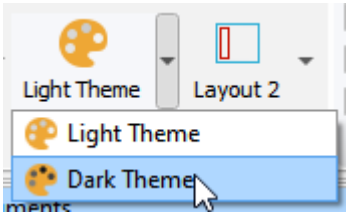
MPY Files

Our GUI Designers generate 2 files, one file is a normal Python file and the other file is used by our GUI Designer to save and remember what your GUI looked like in the GUI Designer, it is a .mpy file and it cannot be opened. The file is just for MatDeck.



Dark Mode

MatDeck and all its applications can be changed to dark mode via the Layout Tab.



Once you have selected the dark mode, you will have to restart MatDeck to apply the change. The Dark Theme will also be applied to all GUI Designers, SCADA, Virtument and other Toolboxes. Please Note that this will affect our GUI Designers the most as any GUI generated will also be in a dark theme.

MD Python

MD Python is the official Python Binding for MatDeck Script, it brings all 1600+ MD function to be used directly in Python. This allows users to use Pythons easy Syntax with our efficient function to bring simple solutions to complex.

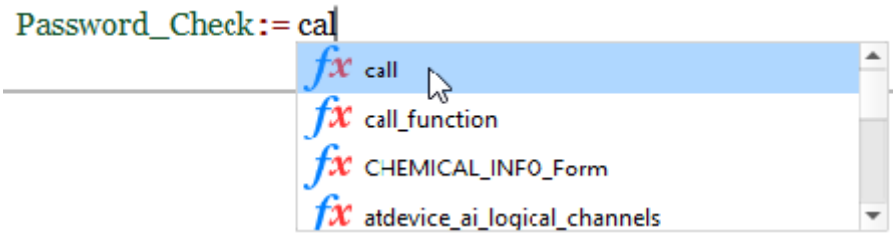
With MD Python, you can also use MD Instruments, these virtual instruments allow you to read and write data in real time.

Call Functions

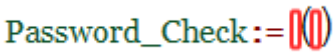
Call functions allow the user to utilize Python functions directly in a MatDeck Canvas and Document, to do this **you will first need to save the Python file in the same directory as the MD file**. For instance, here is an example Python function which we will use in our MatDeck document.

```
1  import re
2
3  def password_ch(v):
4      if v == "\n" or v == " ":
5          return "Password cannot be a newline or space!"
6      if 9 <= len(v) <= 20:
7          # checks for occurrence of a character
8          if re.search(r'(.)\1\1', v):
9              return "Weak Password: Same character repeats three or more times in a row"
10         # checks for occurrence of same string
11         if re.search(r'(..)(.*?)\1', v):
12             return "Weak password: Same string pattern repetition"
13         else:
14             return "Strong Password!"
15     else:
16         return "Password length must be 9-20 characters!"
```

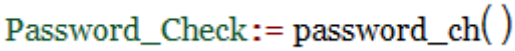
From there, you will need to type call into the canvas, this will produce a blank function with no name or arguments.



Please note that we are not using the function call_function() but the function call().



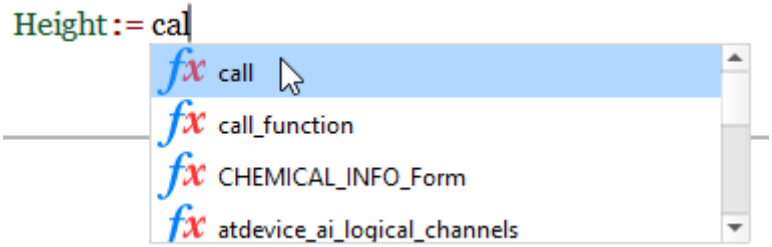
Now, all you need to do is type in the function name and all of its arguments.



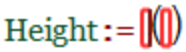
Example of Call Functions

As you can see in the example below, call functions allow us to combine Python code we already have with MD features and functions. Here we use the call functions to utilize the Python functions VecData1() and VecData2() while being able to use MD functions such as our Linear Regression as well as our 2D Graphs.

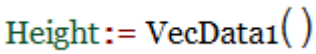
As mentioned above, we use these Python functions by first using the call function.



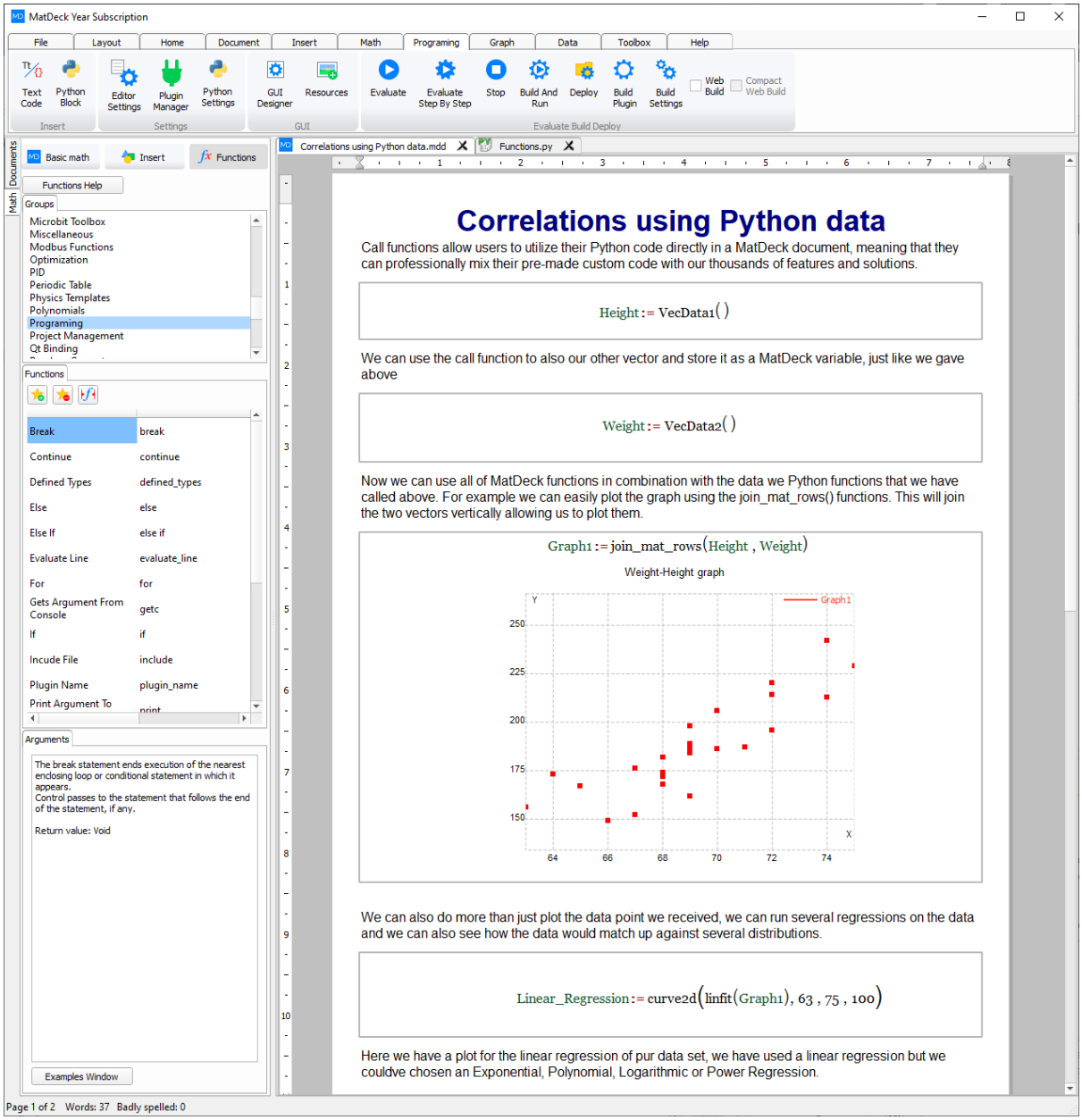
Please note that we are not using the function call_function() but the function call(). Once we select call(), a blank function will appear.



From there we can enter the name of our Python function.

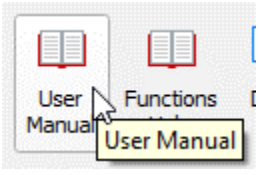


Now, the Python function is active and ready for use.
We will also apply the same steps to get the Python function VecData2().



Python Manuals

All Manuals are available in the Help Tab under User Manuals.



When clicked, a Read-only page will open which contains all MD Manuals, near the bottom of the page, all Python Manuals will be available including the Tkinter, Kivy, Custom Tkinter, PySide2 and MD Python GUI Designer Manuals.