UNIVERSITY OF CAMBRIDGE

CENTER-TBI

ICM+® Users' Meeting

Extending ICM+ with Python scripts with examples using CENTER-TBI data sets

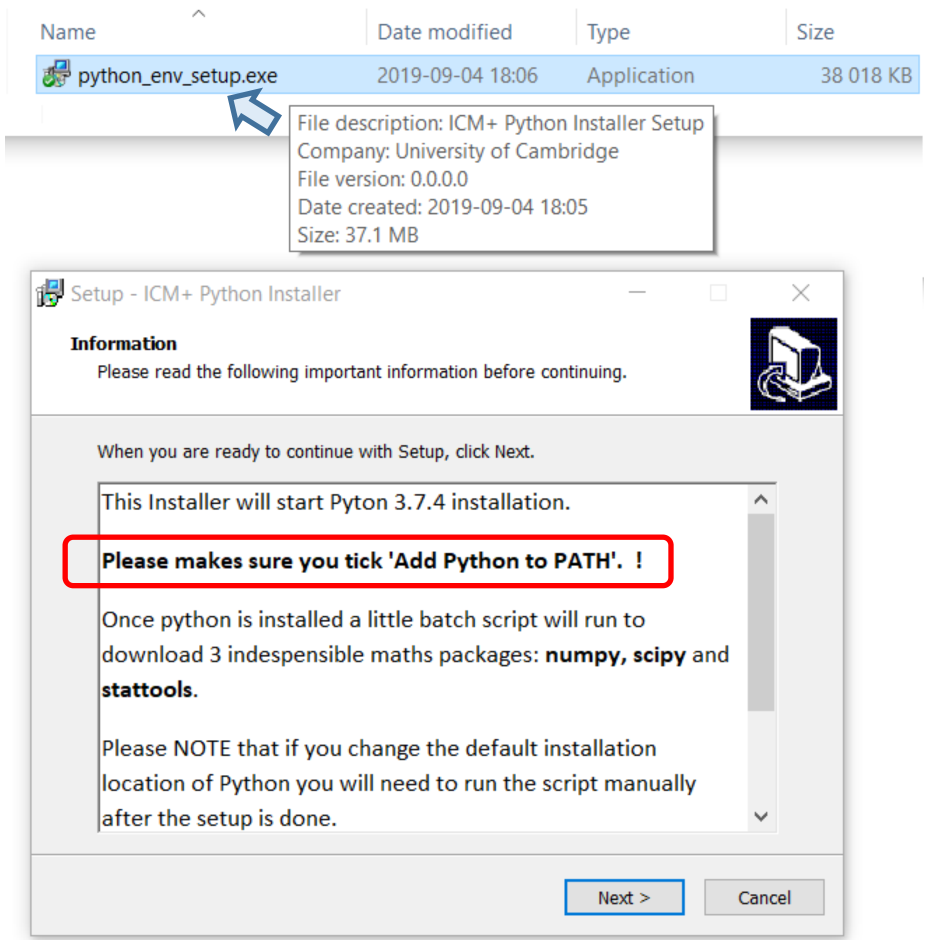
Michał M. Placek

mp963@cam.ac.uk

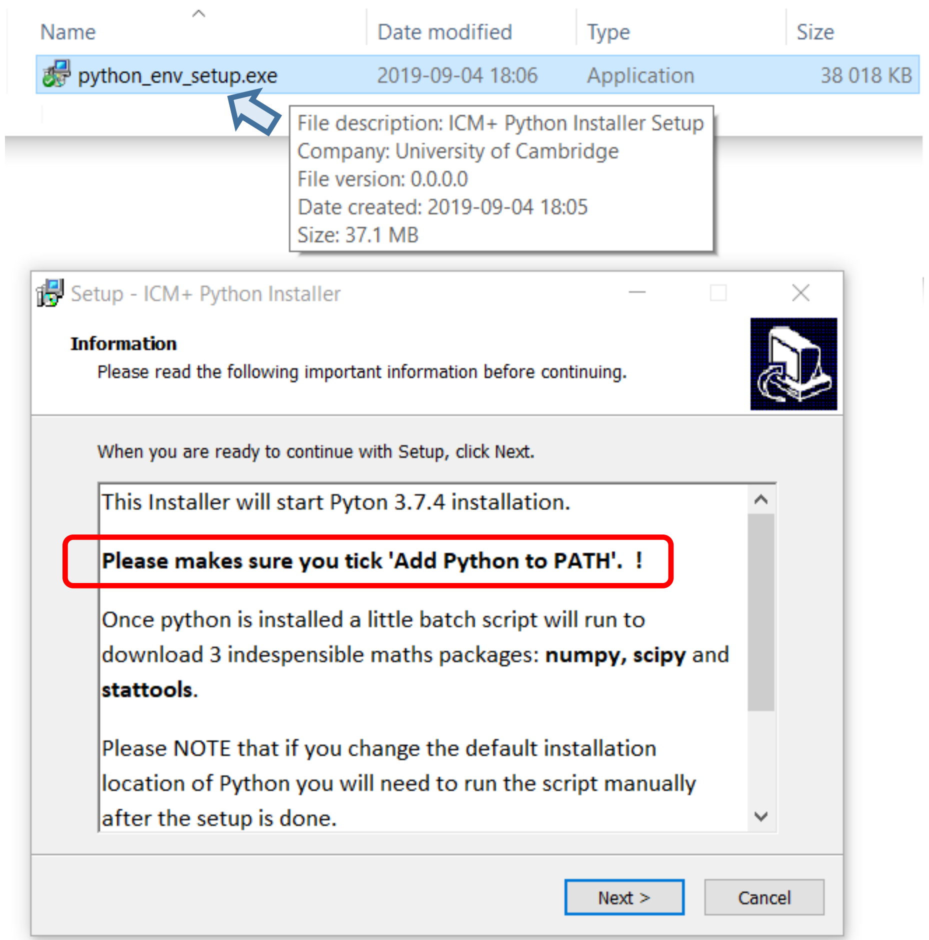
8th September 2019

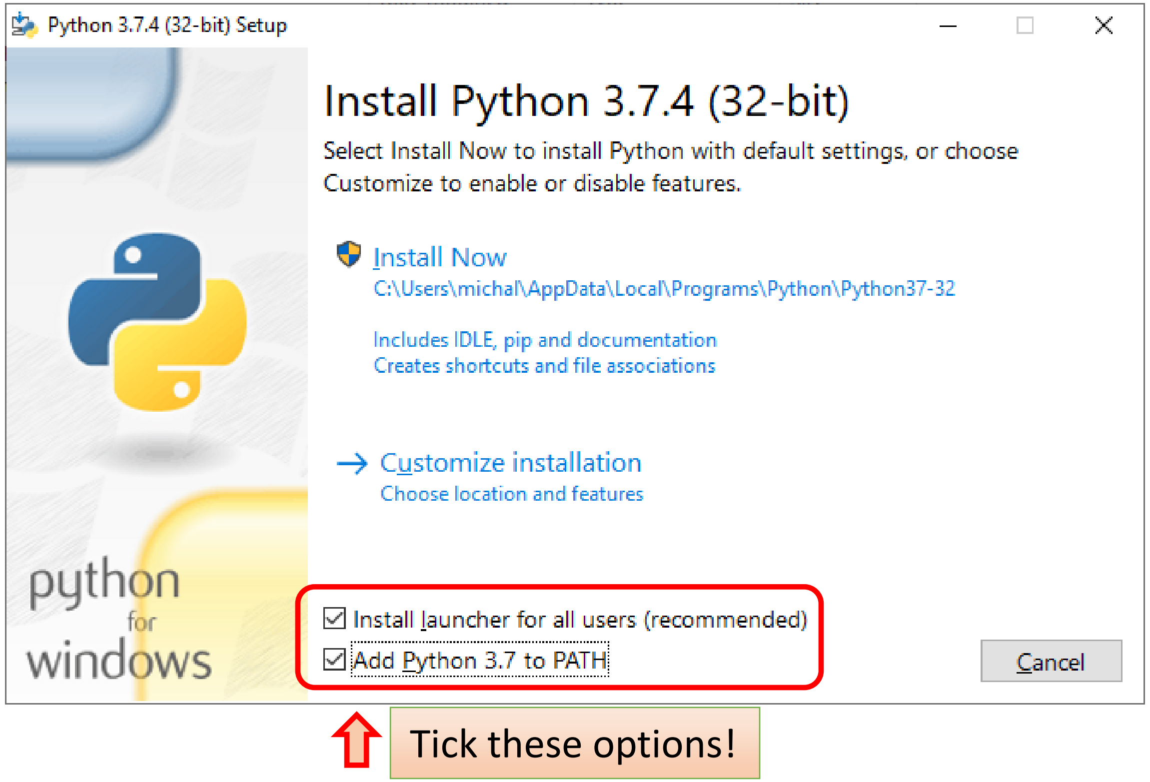
Division of Neurosurgery, Department of Clinical Neurosciences

Python for ICM+ installation

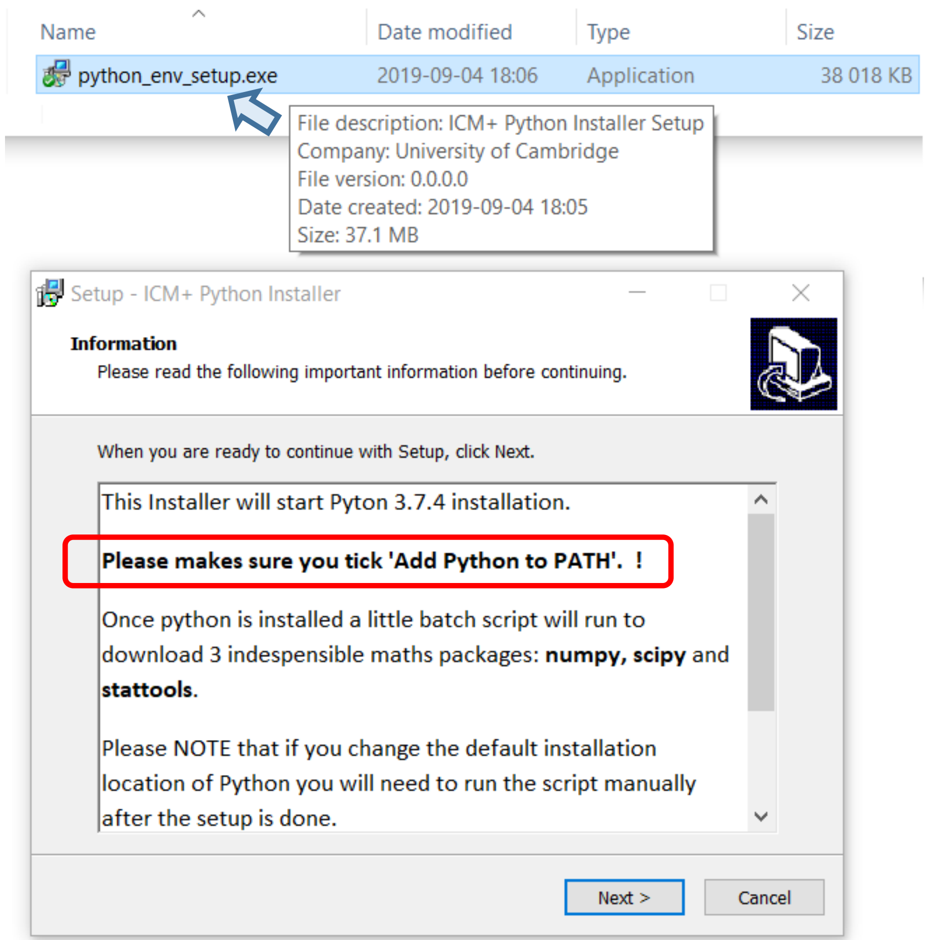


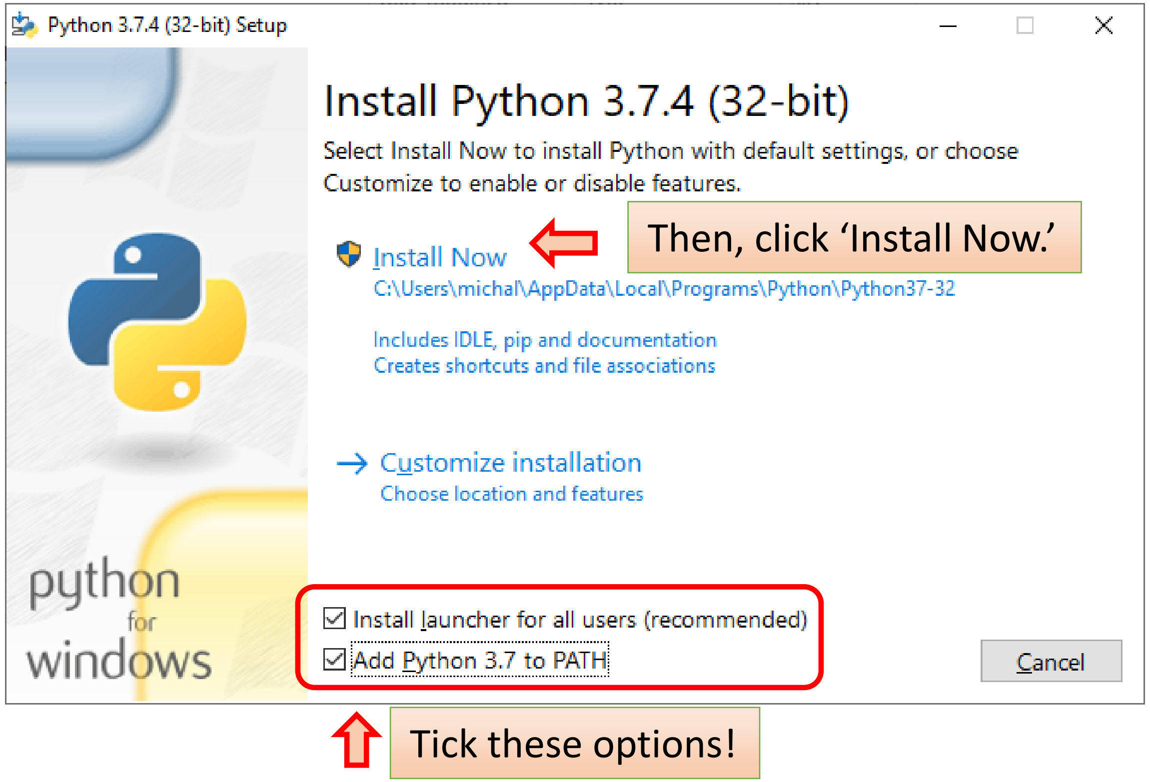
Python for ICM+ installation



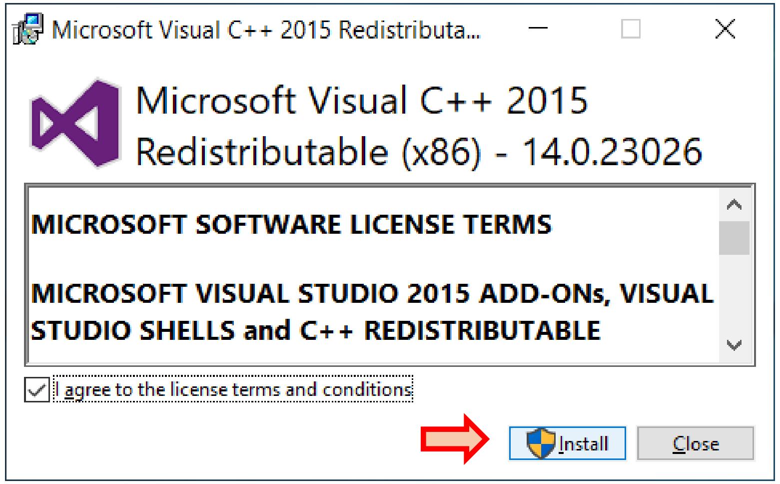


Python for ICM+ installation

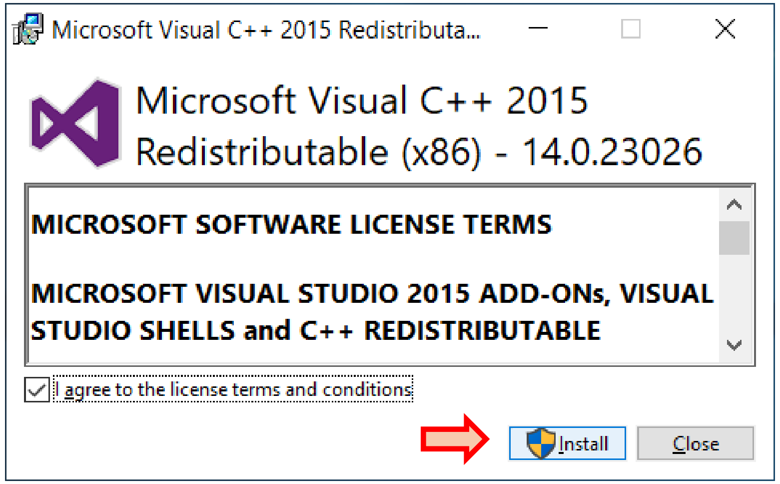


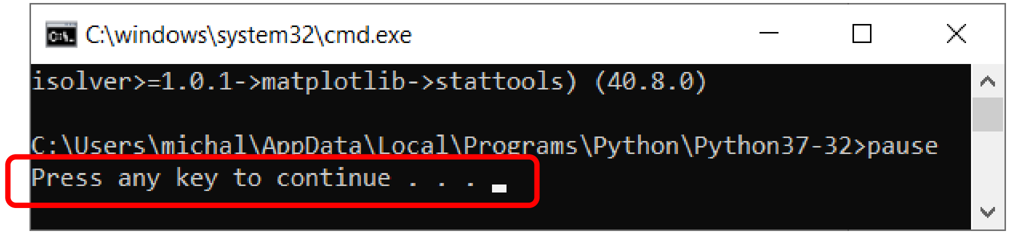


Python for ICM+ installation: cont'd

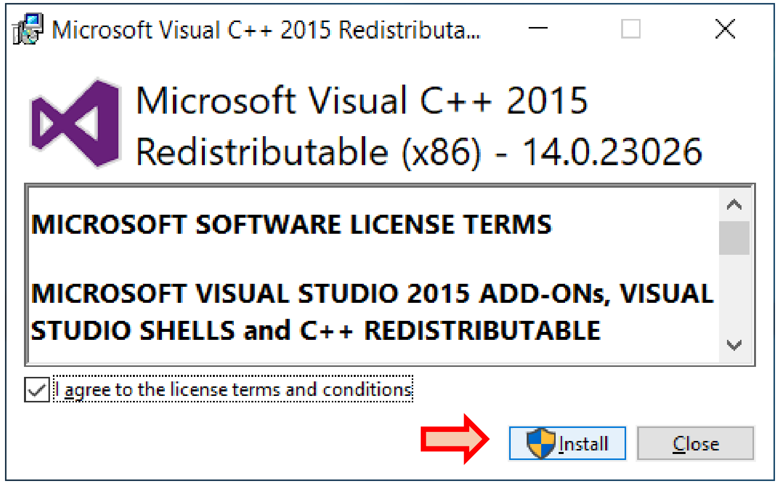


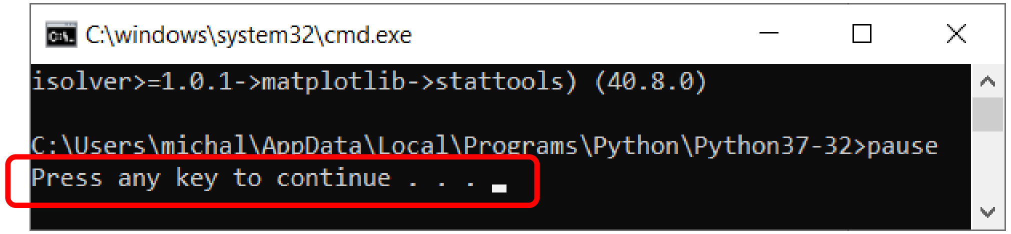
Python for ICM+ installation: cont'd

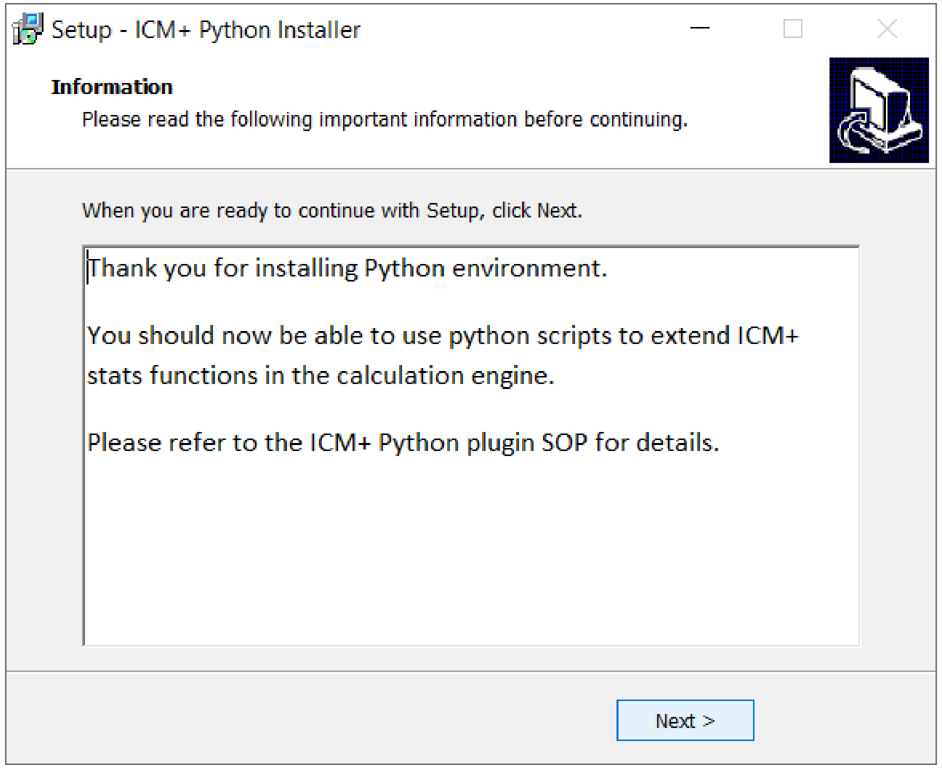




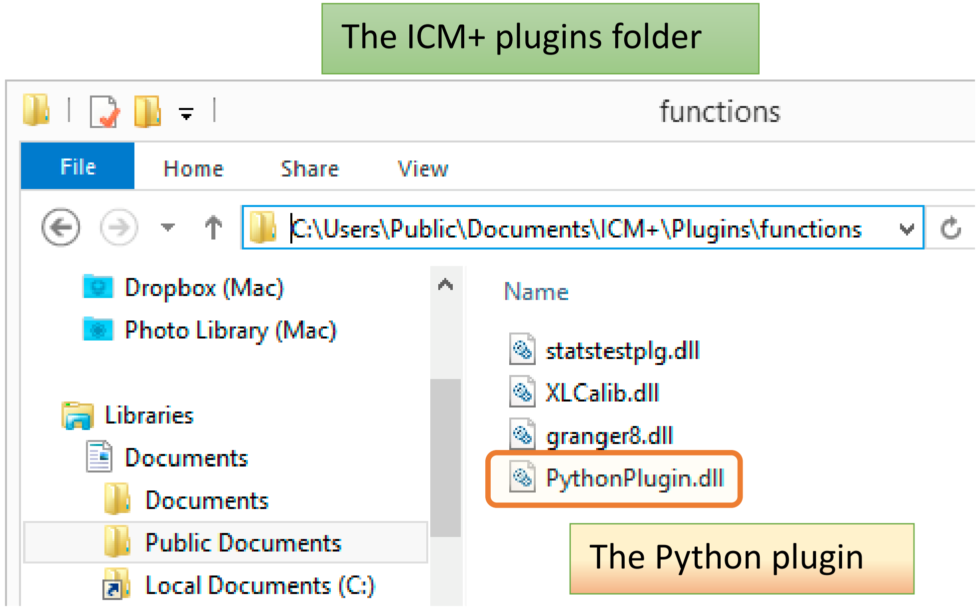
Python for ICM+ installation: cont'd





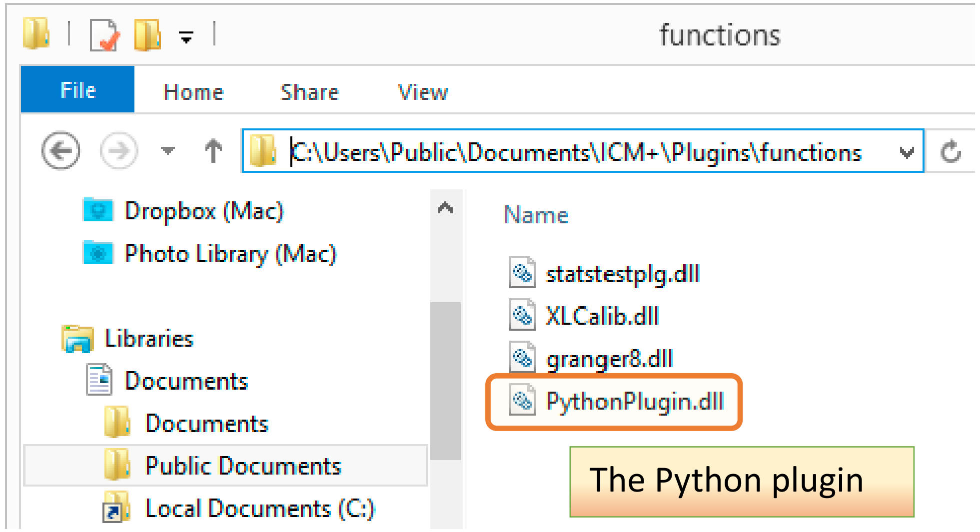


Python Plugin important directories



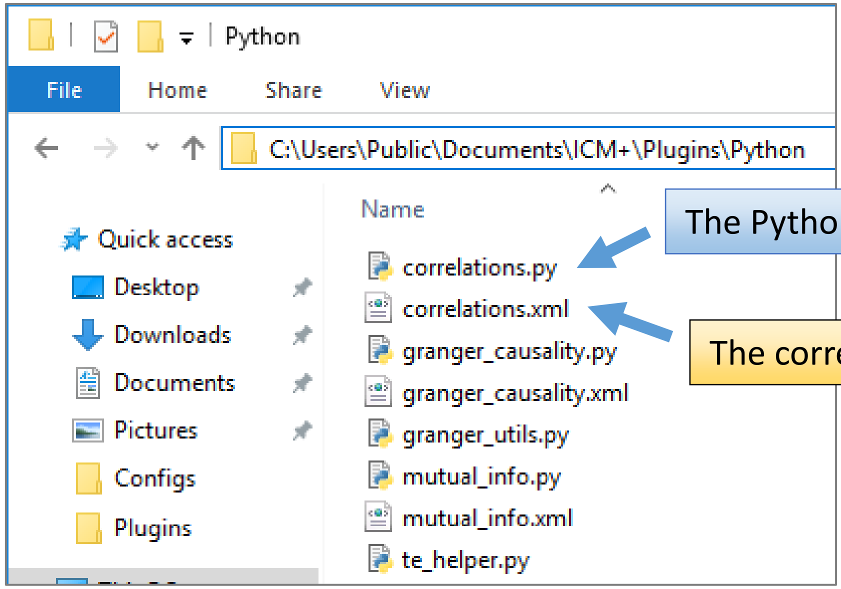
Python Plugin important directories

The ICM+ plugins folder



The Python plugin

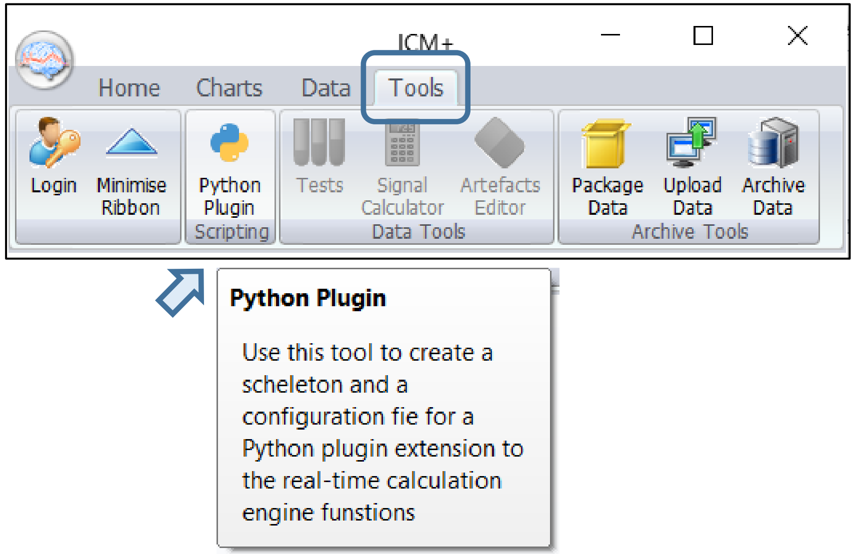
The ICM+ Python plugin folder



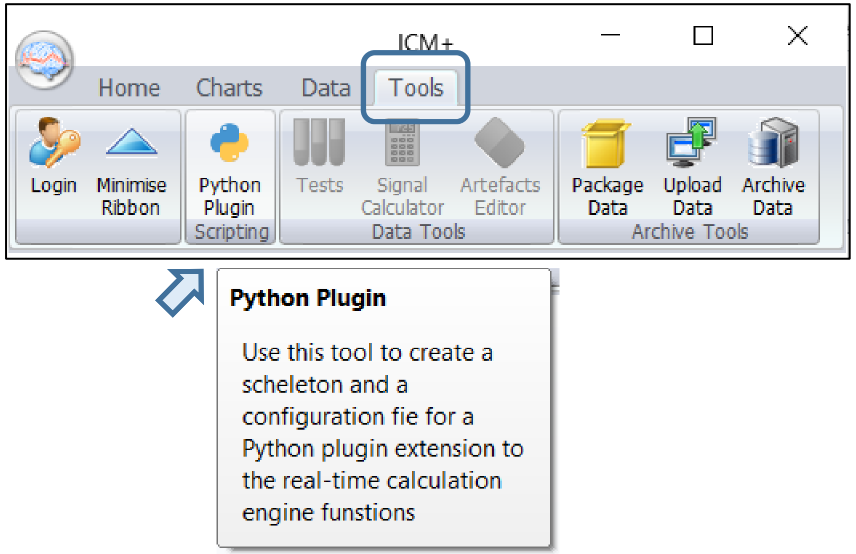
The Python script

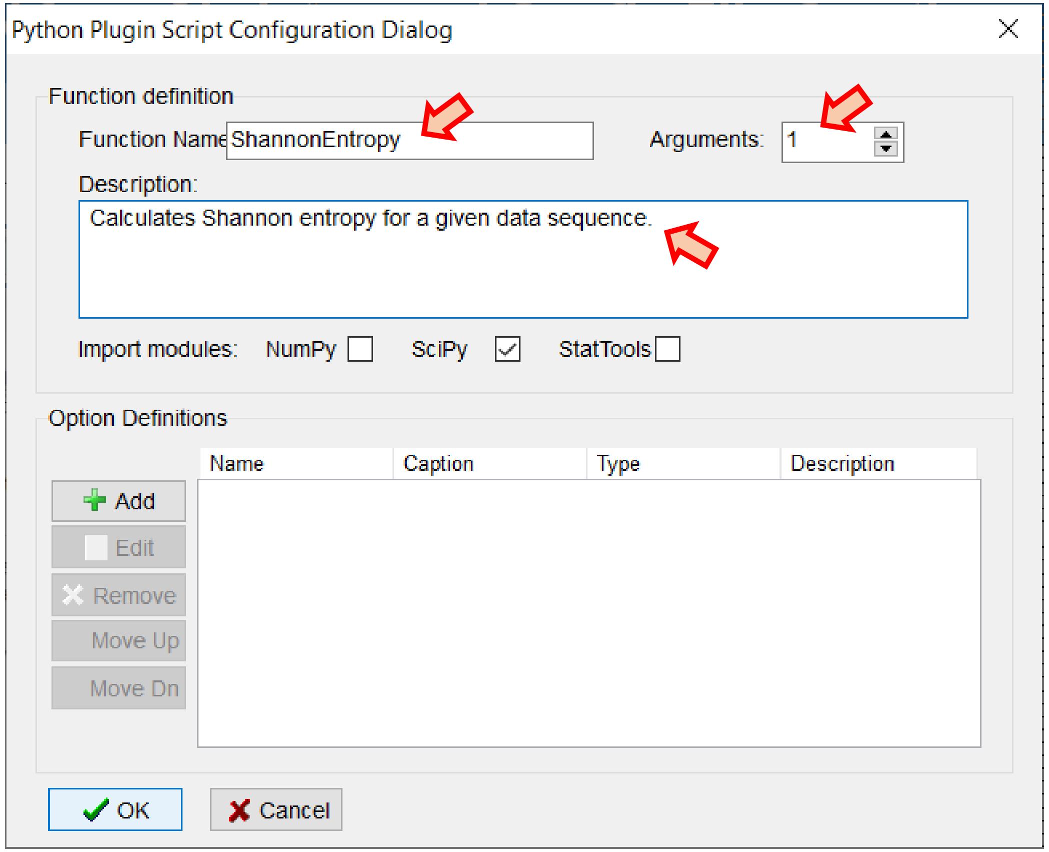
The corresponding config file

The ICM+ Tool for a Python template and its config file creation

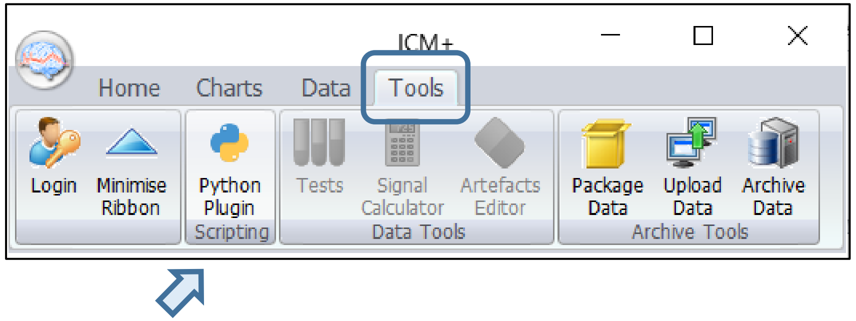


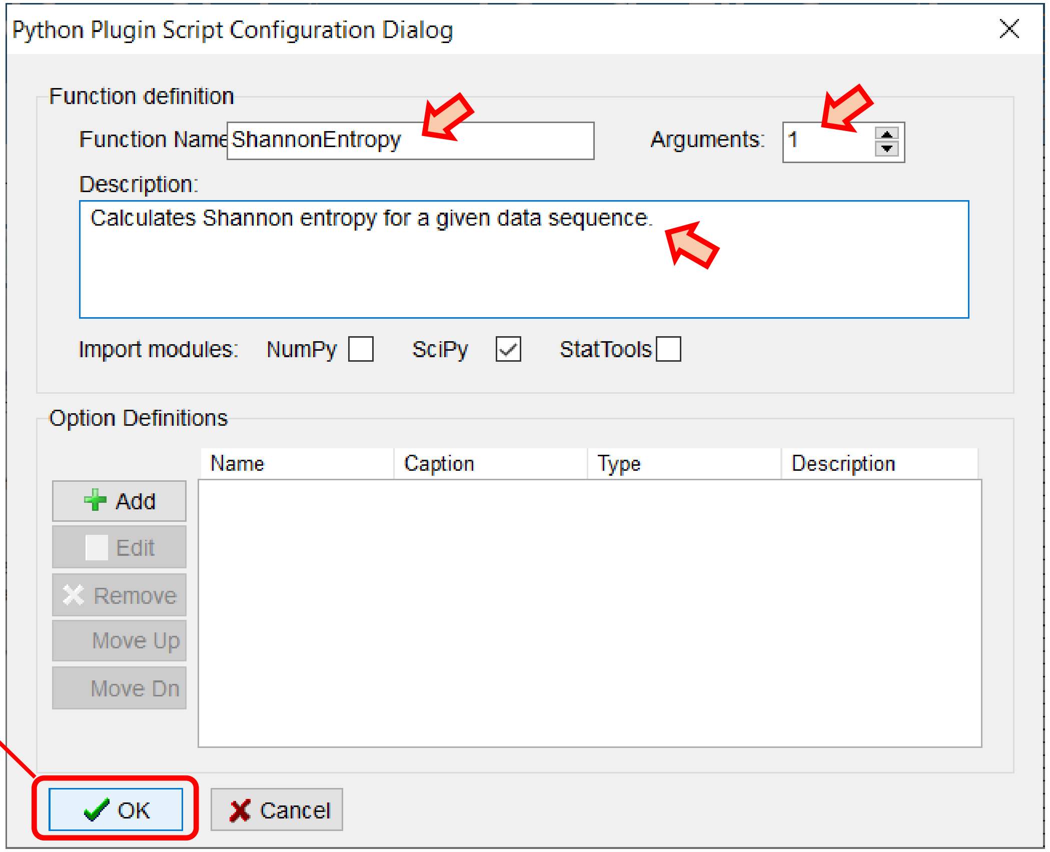
The ICM+ Tool for a Python template and its config file creation

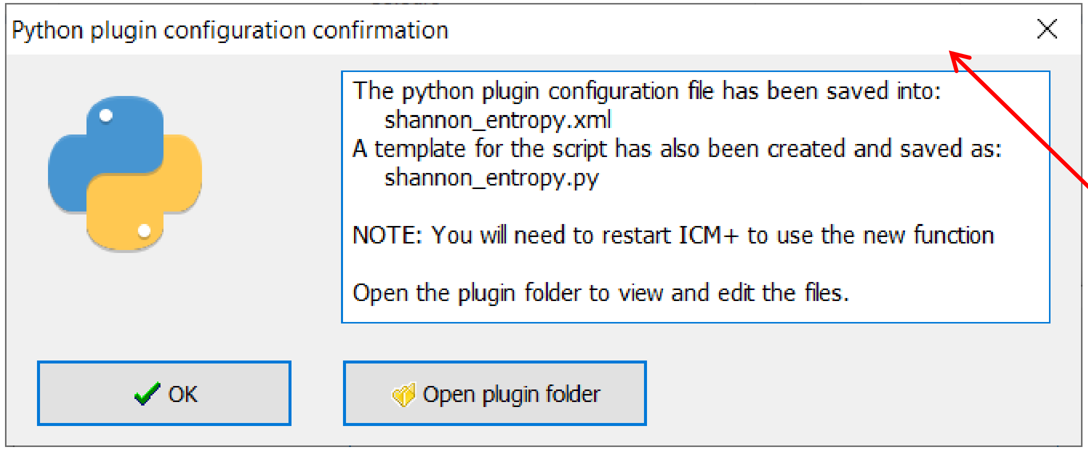




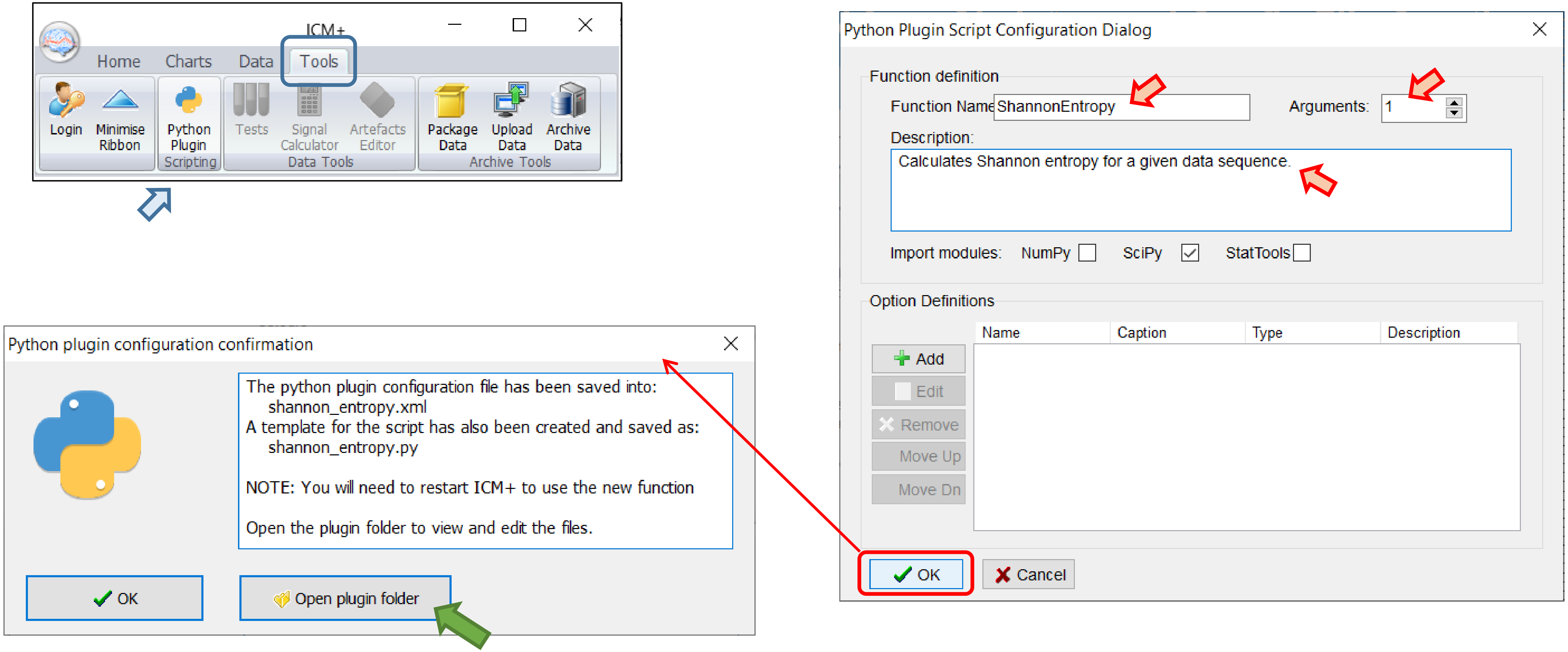
The ICM+ Tool for a Python template and its config file creation







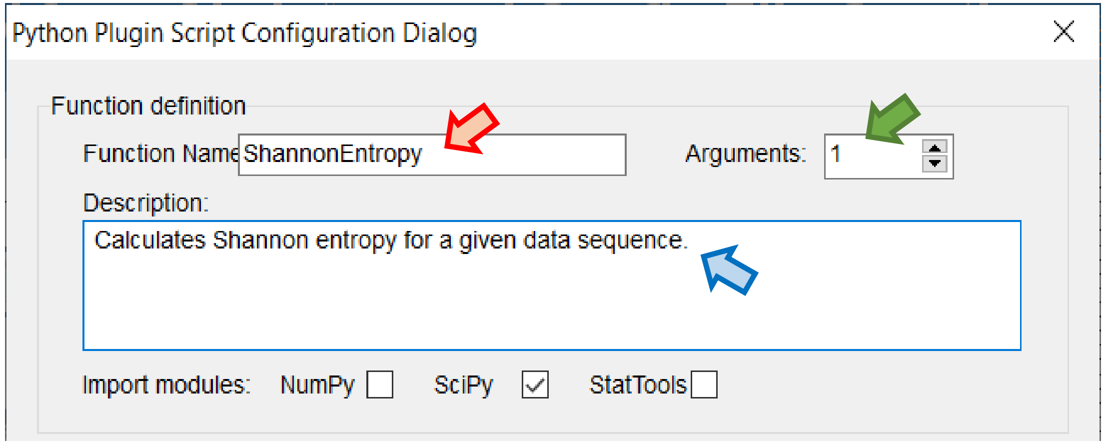
The ICM+ Tool for a Python template and its config file creation



Generated Configuration File

The XML config file generated by ICM+

```xml
<?xml version = "1.0"?>

<PyToICMPlusConfig>
    <Function Name="ShannonEntropy" Type="Stats" SignalsCount="1">
        <GUID>{7A097741-6044-4828-8371-B4BC3E6A1BFE}</GUID>
        <Description>Calculates Shannon entropy for a given data sequence.</Description>
    </Function>
</PyToICMPlusConfig>
```



The generated Python script

```python
#import ...
import scipy as sp

class ShannonEntropy:

    # DO NOT MODIFY THIS METHOD. It is a part of the ICM+--Python interface.
    def set_parameter(self, param_name, param_value):
        setattr(self, param_name, param_value)

    # You can append your own code to the constructor, if needed.
    # You should not set here values of parameters declared in your XML
    # configuration file because ICM+ will do it for you.
    # You will have to add your own code, only if you need to initialise some
    # extra data structures which were not declared in the XML config file.
    def __init__(self):
        self.sampling_freq = None

    # You can append your own code to the destructor but most likely
    # you will not need it.
    def __del__(self):
        pass

    # 'calculate' is the main work-horse function.
    # It is called with a data buffer (one or more) of size corresponding to the Calculation Window
    # It must return one floating-point number
    # It take the following parameters:
    # sig1 - input variable/signal 1
    # ts_time - part of the data time stamp - number of milliseconds since midnight
    # ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
    # It can also use the data sampling frequency:
    #    self.sampling_freq
    def calculate(self, sig1, ts_time, ts_date):
        # my_own_code_here
        result = 0.0
        return result
```

The generated Python script

```
#import ...
import scipy as sp

class ShannonEntropy:

    # DO NOT MODIFY THIS METHOD. It is a part of the ICM+--Python interface.
    def set_parameter(self, param_name, param_value):
        setattr(self, param_name, param_value)

    # You can append your own code to the constructor, if needed.
    # You should not set here values of parameters declared in your XML
    # configuration file because ICM+ will do it for you.
    # You will have to add your own code, only if you need to initialise some
    # extra data structures which were not declared in the XML config file.
    def __init__(self):
        self.sampling_freq = None

    # You can append your own code to the destructor but most likely
    # you will not need it.
    def __del__(self):
        pass

    # 'calculate' is the main work-horse function.
    # It is called with a data buffer (one or more) of size corresponding to the Calculation Window
    # It must return one floating-point number
    # It take th                 rs:
    # sig1 - inp
    # ts_time - part of the data time stamp - number of milliseconds since midnight
    # ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
    # It can also use the data sampling frequency:
    #    self.sampling_fr
    def calculate(self, sig1, ts_time, ts_date):
        # my_own_code_here
        result = 0.0
        return result
```

One input signal

The generated Python script

```python
#import ...
import scipy as sp

class ShannonEntropy:

    # DO NOT MODIFY THIS METHOD. It is a part of the ICM+--Python interface.
    def set_parameter(self, param_name, param_value):
        setattr(self, param_name, param_value)

    # You can append your own code to the constructor, if needed.
    # You should not set here values of parameters declared in your XML
    # configuration file because ICM+ will do it for you.
    # You will have to add your own code, only if you need to initialise some
    # extra data structures which were not declared in the XML config file.
    def __init__(self):
        self.sampling_freq = None

    # You can append your own code to the destructor but most likely
    # you will not need it.
    def __del__(self):
        pass

    # 'calculate' is the main work-horse function.
    # It is called with a data buffer (one or more) of size corresponding to the Calculation Window
    # It must return one floating-point number
    # It take th                    rs:
    # sig1 - inp
    # ts_time - part of the data time stamp - number of milliseconds since midnight
    # ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
    # It can also use the data sampling frequency:
    #    self.sampling_freq
    def calculate(self, sig1, ts_time, ts_date):
        # my_own_code_here
        result = 0.0
        return result
```

One input signal

Add your own code to the *calculate* method

The generated Python script

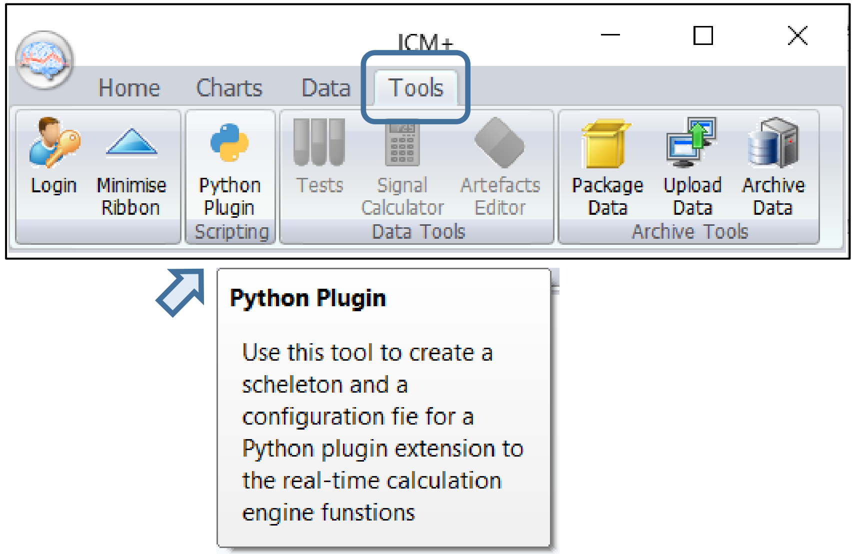
```python
#import ...
import scipy as sp

class ShannonEntropy:

    # DO NOT MODIFY THIS METHOD. It is a part of the ICM+--Python interface.
    def set_parameter(self, param_name, param_value):
        setattr(self, param_name, param_value)

    # You can append your own code to the constructor, if needed.
    # You should not set here values of parameters declared in your XML
    # configuration file because ICM+ will do it for you.
    # You will have to add your own code, only if you need to initialise some
    # extra data structures which were not declared in the XML config file.
    def __init__(self):
        self.sampling_freq = None

    # You can append your own code to the destructor but most likely
    # you will not need it.
    def __del__(self):
        pass

    # 'calculate' is the main work-horse function.
    # It is called with a data buffer (one or more) of size corresponding to the Calculation Window
    # It must return one floating-point number
    # It take th                 rs:
    # sig1 - inp
    # ts_time - part of the data time stamp - number of milliseconds since midnight
    # ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
    # It can also use the data sampling frequency:
    #    self.sampling_freq
    def calculate(self, sig1, ts_time, ts_date):
        # my_own_code_here
        result = 0.0
        return result
```

One input signal

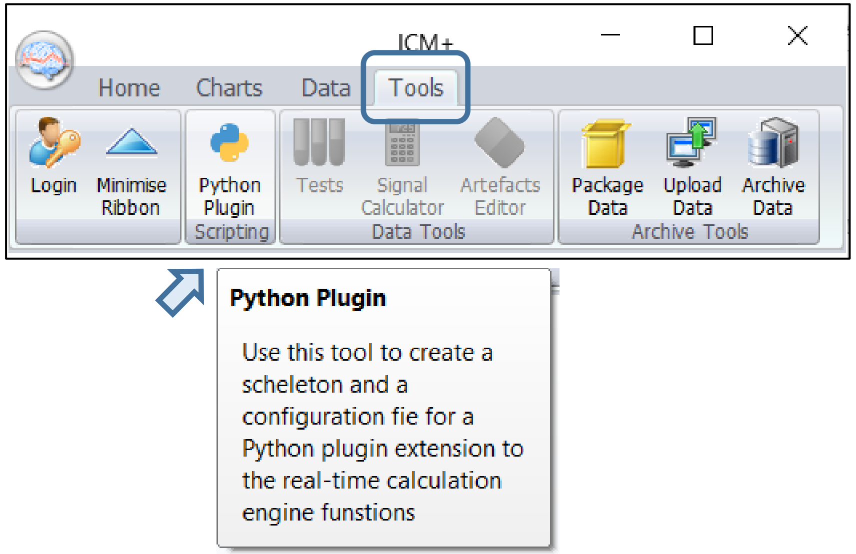
Add your own code to the *calculate* method

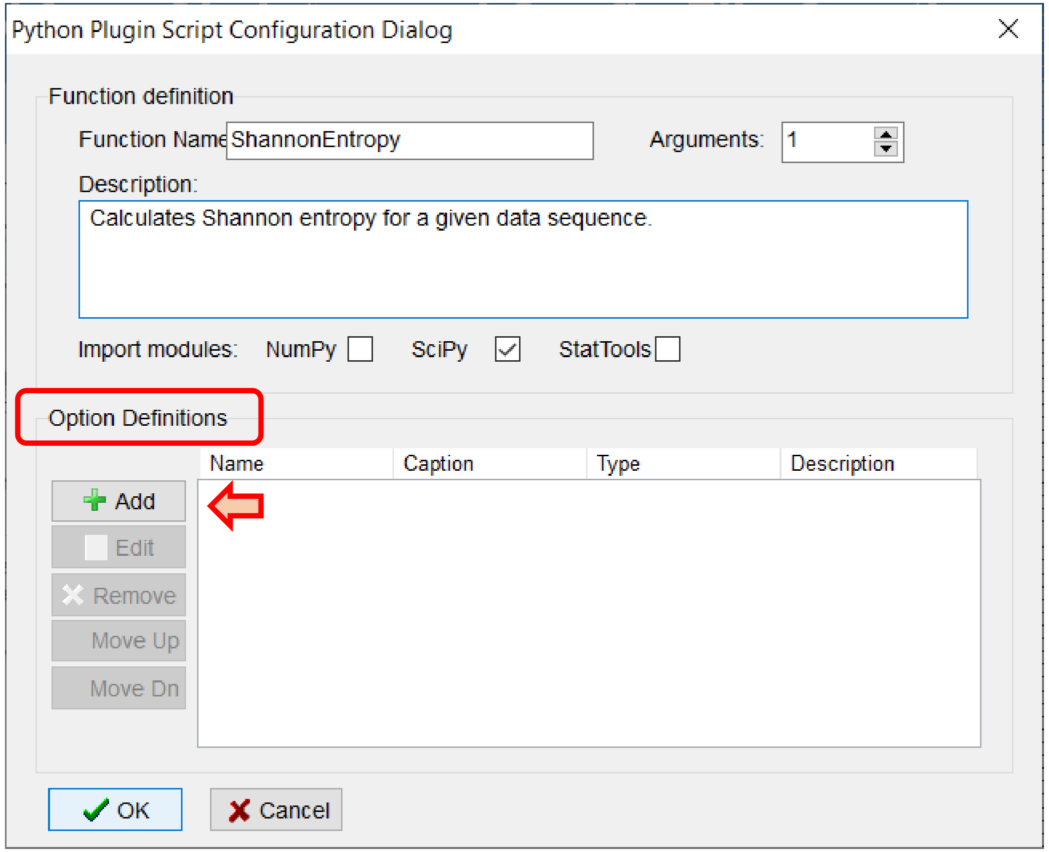
```python
from scipy import stats

def calculate(self, sig1, ts_time, ts_date):
    result = stats.entropy(sig1)
    return result
```

Adding options to the user-defined Python function

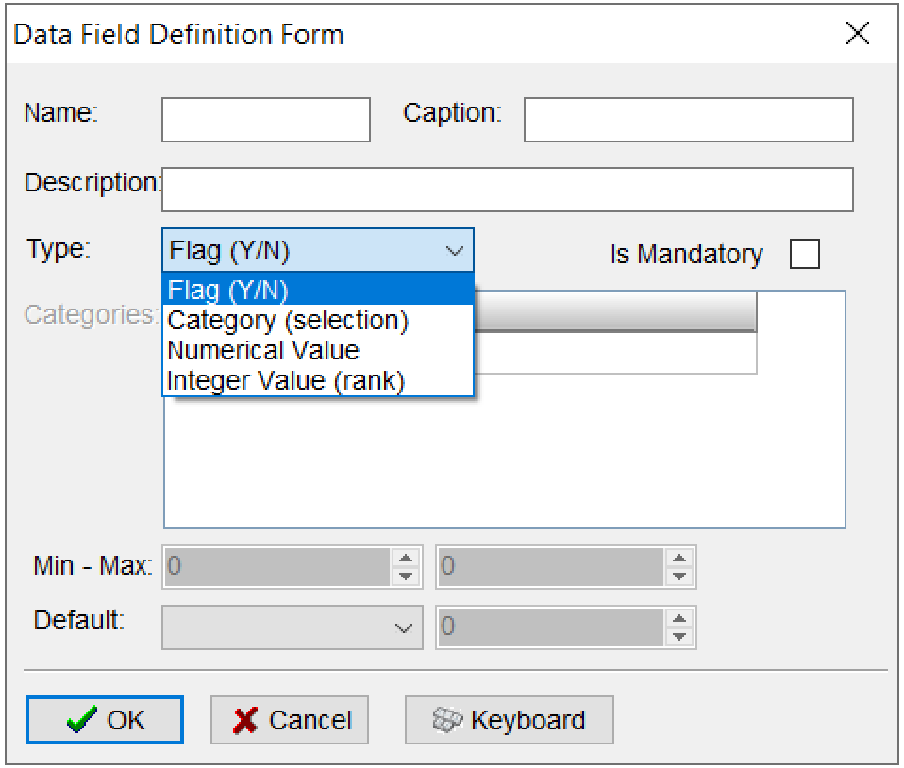


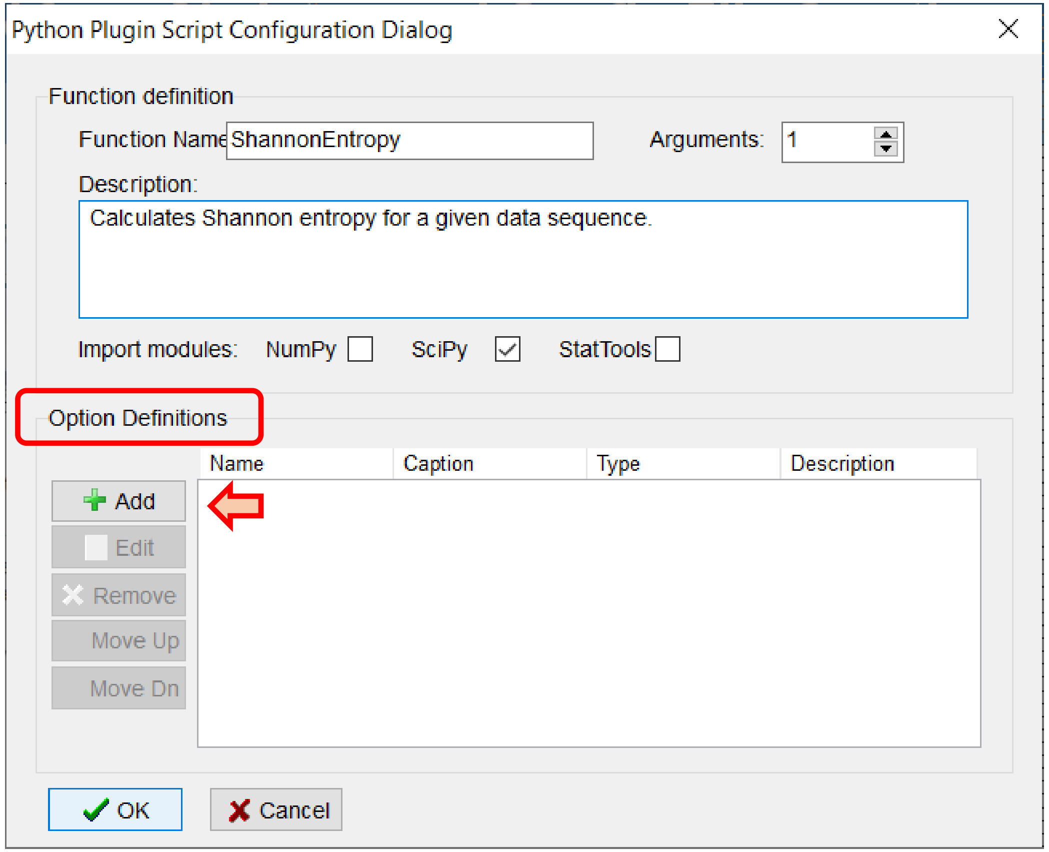
Adding options to the user-defined Python function



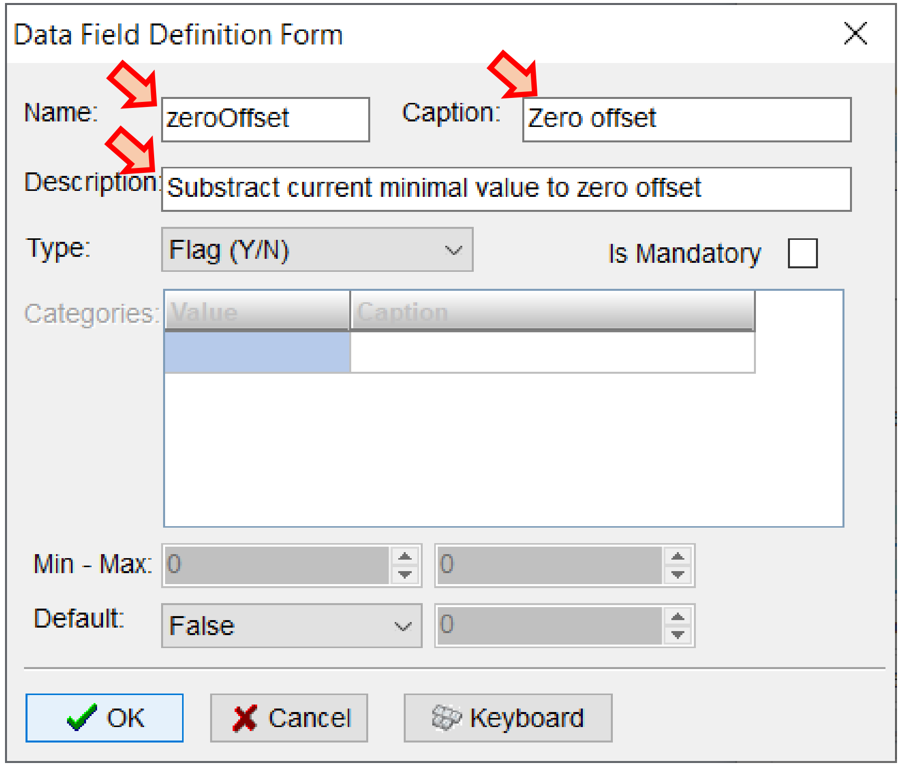


Adding options to the user-defined Python function

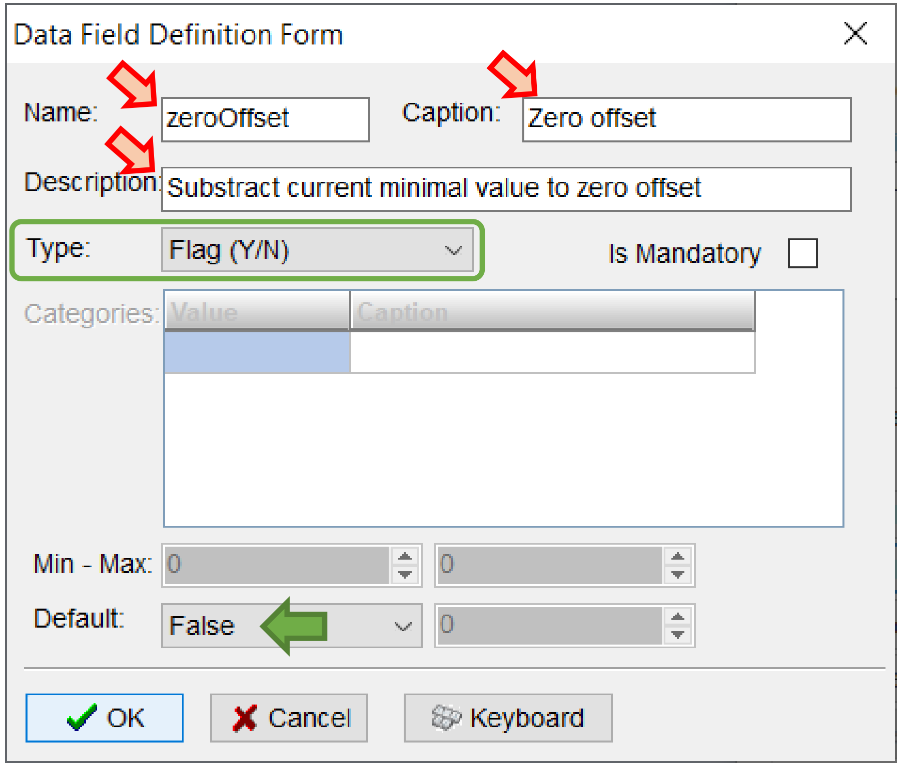




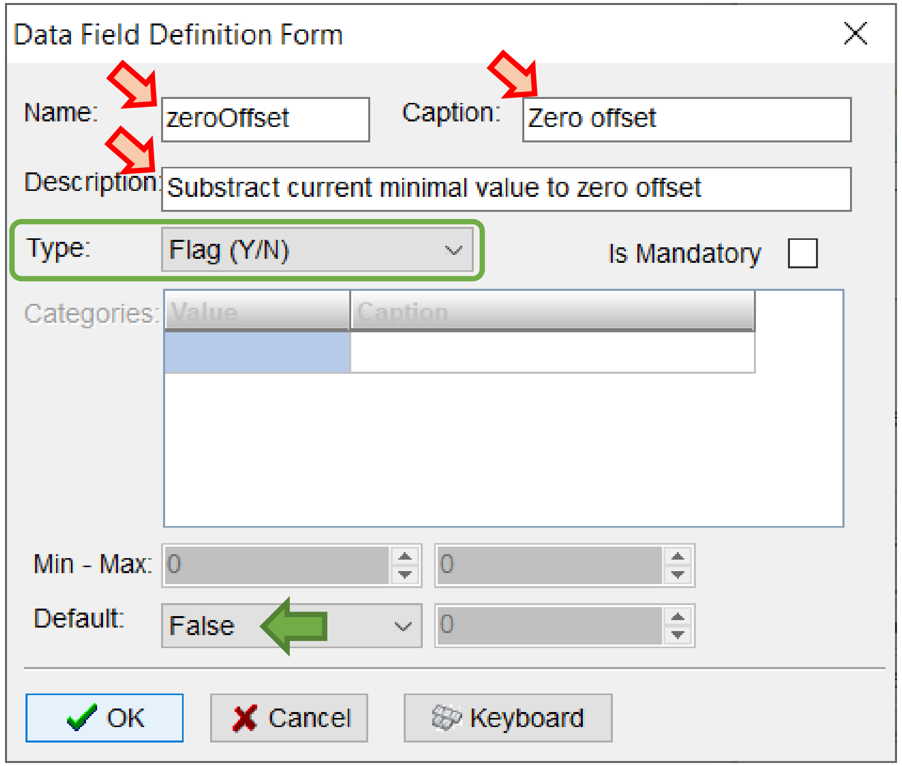
Adding an option of the type ’flag’

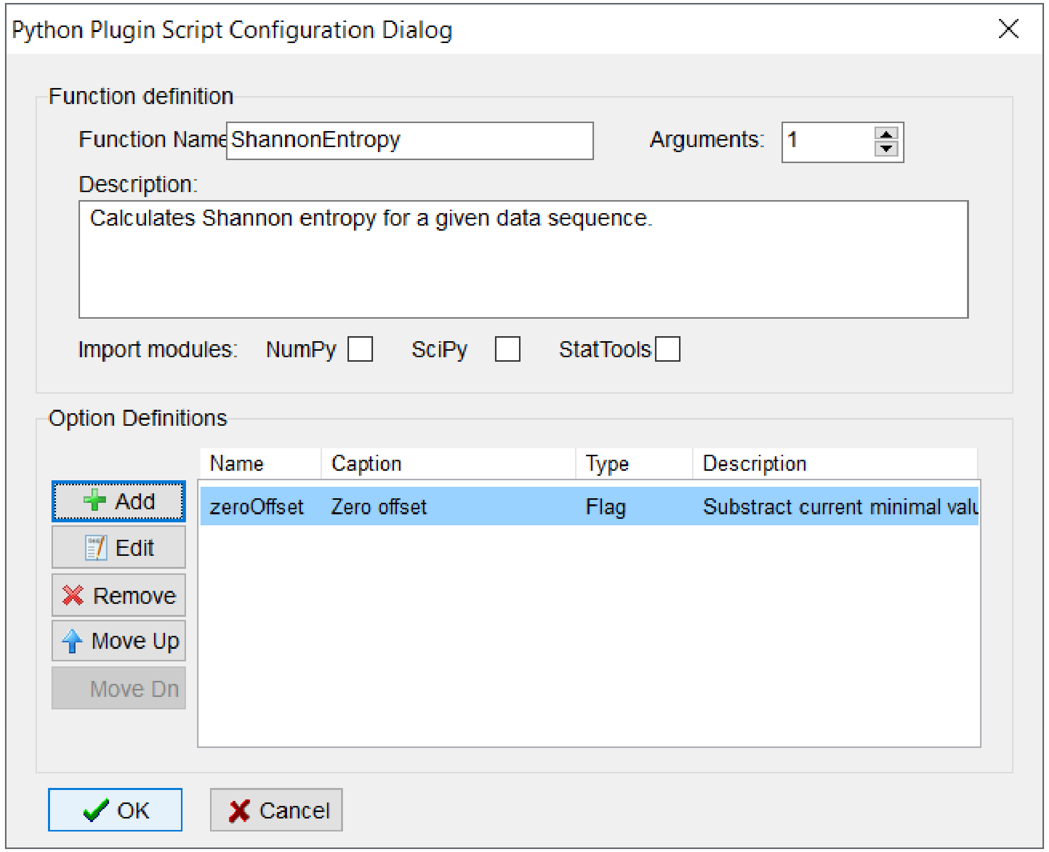


Adding an option of the type ’flag’

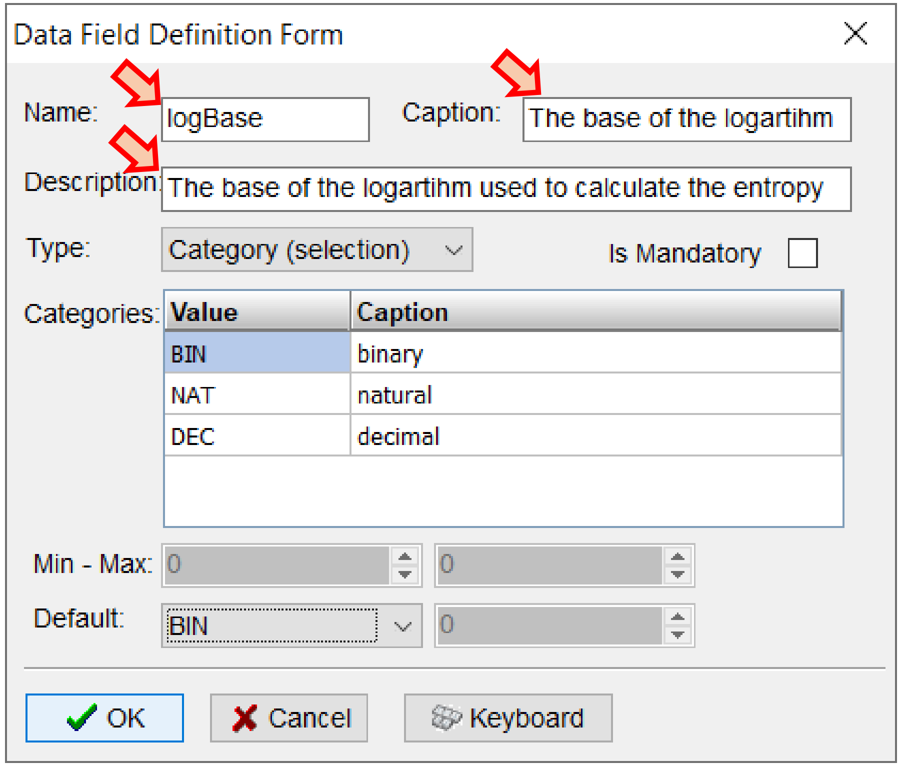


Adding an option of the type 'flag'

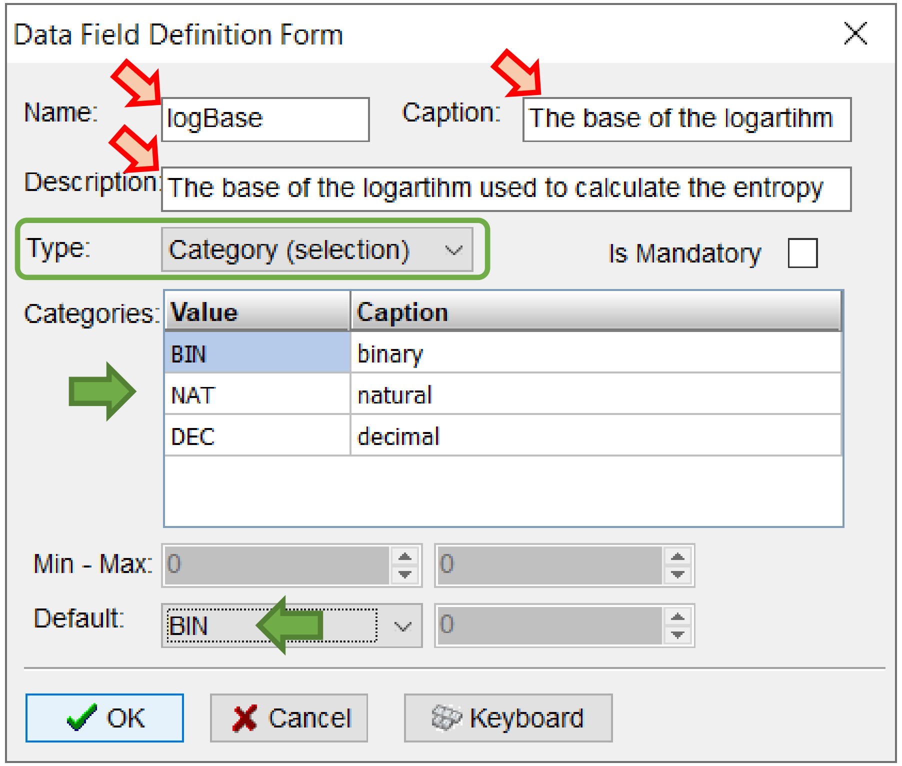




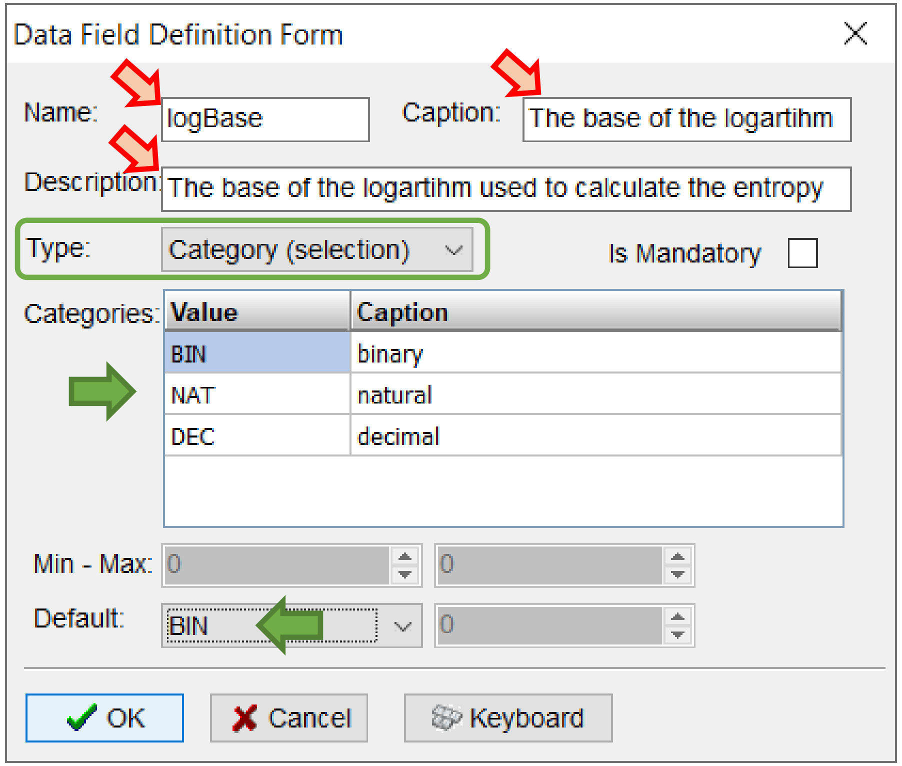
Adding an option of the type ’category’

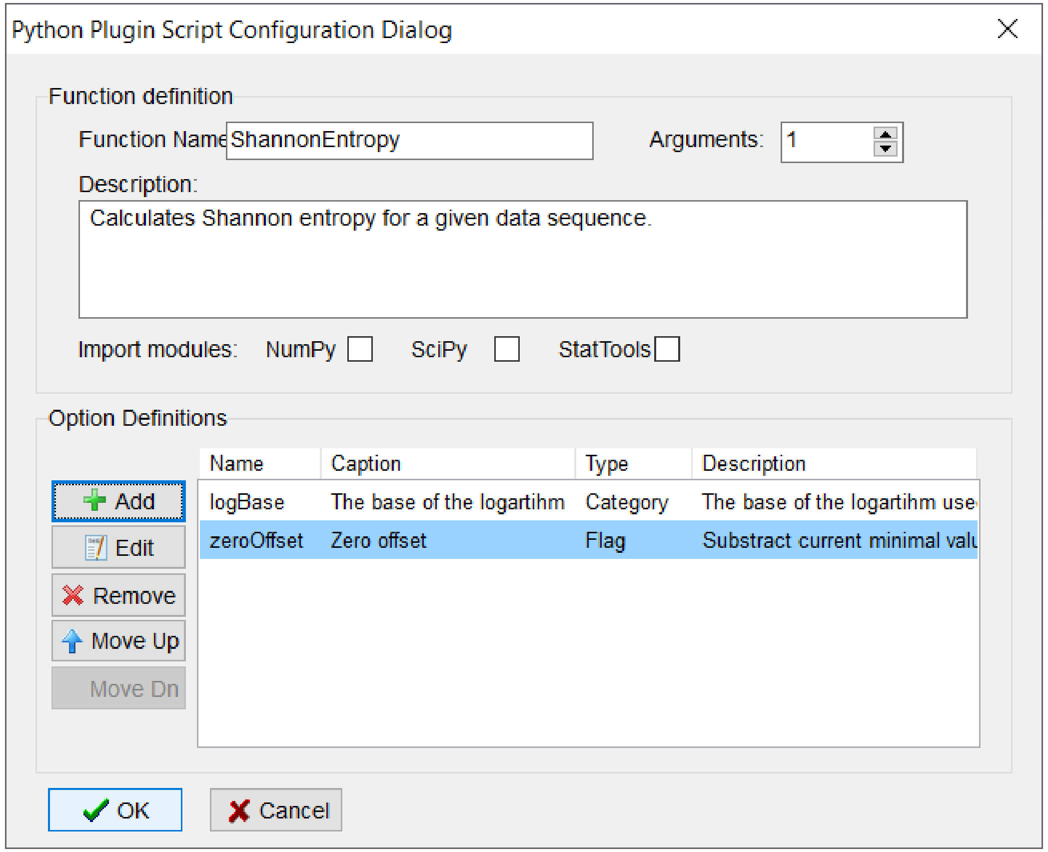


Adding an option of the type ’category’



Adding an option of the type ’category’



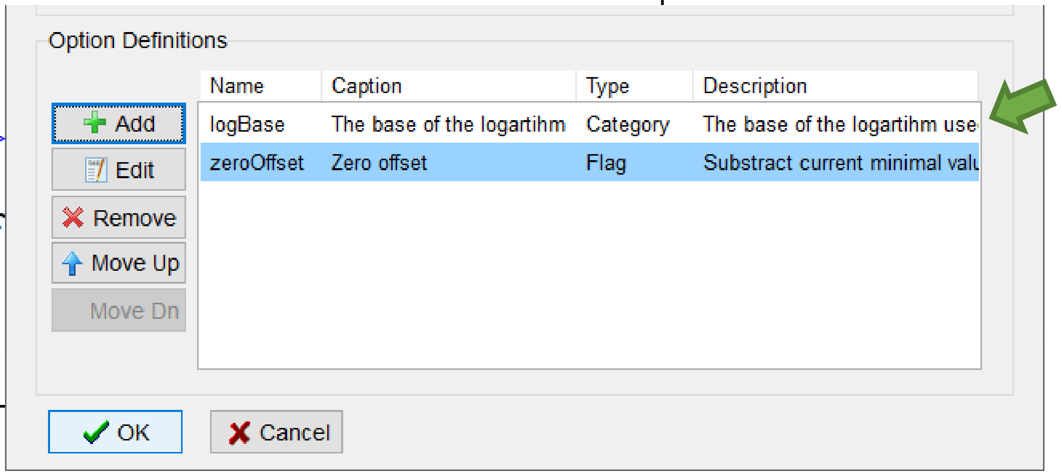


Generated Configuration File


```xml
<?xml version = "1.0"?>

<PyToICMPlusConfig>
    <Function Name="ShannonEntropy" Type="Stats" SignalsCount="1">
        <GUID>{3DE497F8-5AF6-40D5-907E-02B2CCDF19C5}</GUID>
        <Description>Calculates Shannon entropy for a given data sequence.</Description>
        <Parameter ShortName="logBase" IsMandatory="False">
            <Caption>The base of the logartihm</Caption>
            <Description>The base of the logartihm used to calculate the entropy</Description>
            <Type Name="StringList">
                <Item Value="BIN" Caption="binary" IsDefault="True"/>
                <Item Value="NAT" Caption="natural"/>
                <Item Value="DEC" Caption="decimal"/>
            </Type>
        </Parameter>
        <Parameter ShortName="zeroOffset" IsMandatory="False">
            <Caption>Zero offset</Caption>
            <Description>Substract current minimal value to zero offset</Description>
            <Type Name="Bool" DefaultValue="False"/>
        </Parameter>
    </Function>
</PyToICMPlusConfig>
```

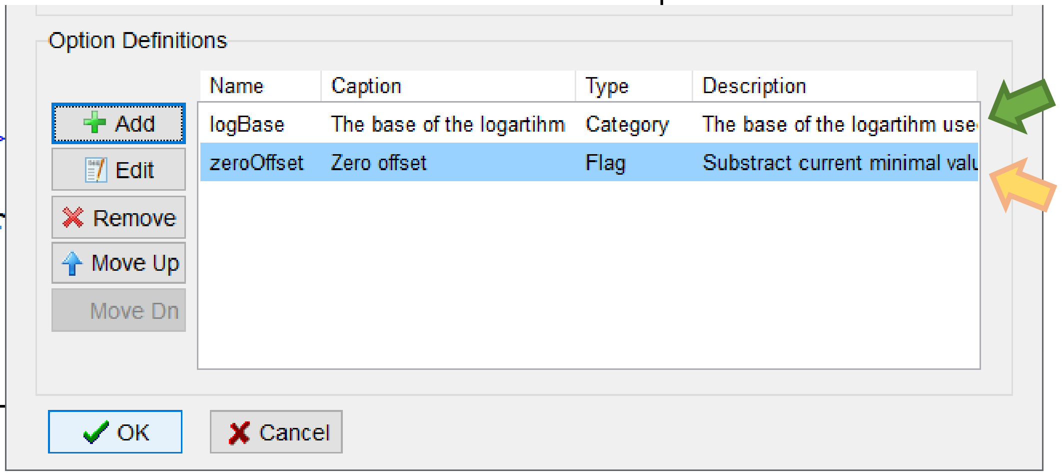
Generated Configuration File

```
<?xml version = "1.0"?>

<PyToICMPlusConfig>
    <Function Name="ShannonEntropy" Type="Stats" SignalsCount="1">
        <GUID>{3DE497F8-5AF6-40D5-907E-02B2CCDF19C5}</GUID>
        <Description>Calculates Shannon entropy for a given data sequence.</Description>
        <Parameter ShortName="logBase" IsMandatory="False">
            <Caption>The base of the logartihm</Caption>
            <Description>The base of the logartihm used to calculate the entropy</Description>
            <Type Name="StringList">
                <Item Value="BIN" Caption="binary" IsDefault="True"/>
                <Item Value="NAT" Caption="natural"/>
                <Item Value="DEC" Caption="decimal"/>
            </Type>
        </Parameter>
        <Parameter ShortName="zeroOffset" IsMandatory="False">
            <Caption>Zero offset</Caption>
            <Description>Substract current minimal value to zer
            <Type Name="Bool" DefaultValue="False"/>
        </Parameter>
    </Function>
</PyToICMPlusConfig>
```



Generated Configuration File

```xml
<?xml version = "1.0"?>

<PyToICMPlusConfig>
    <Function Name="ShannonEntropy" Type="Stats" SignalsCount="1">
        <GUID>{3DE497F8-5AF6-40D5-907E-02B2CCDF19C5}</GUID>
        <Description>Calculates Shannon Entropy for a given data sequence.</Description>
        <Parameter ShortName="logBase" IsMandatory="False">
            <Caption>The base of the logartihm</Caption>
            <Description>The base of the logartihm used to calculate the entropy</Description>
            <Type Name="StringList">
                <Item Value="BIN" Caption="binary" IsDefault="True"/>
                <Item Value="NAT" Caption="natural"/>
                <Item Value="DEC" Caption="decimal"/>
            </Type>
        </Parameter>
        <Parameter ShortName="zeroOffset" IsMandatory="False">
            <Caption>Zero offset</Caption>
            <Description>Substract current minimal value to zer
            <Type Name="Bool" DefaultValue="False"/>
        </Parameter>
    </Function>
</PyToICMPlusConfig>
```

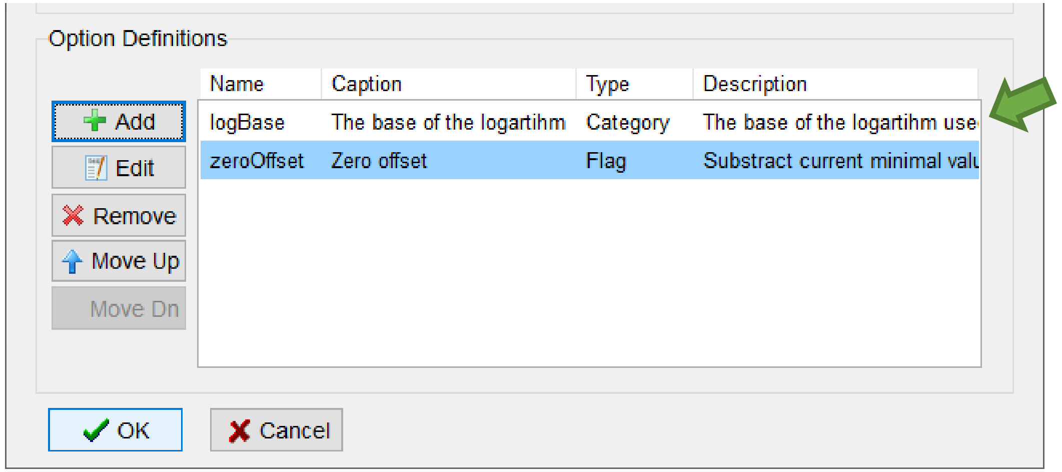


How to use the configured options in the Python script

```python
# 'calculate' is the main work-horse function.
# It is called with a data buffer (one or more) of size corresponding to the Calculation Window
# It must return one floating-point number
# It take the following parameters:
# sig1 - input variable/signal 1
# ts_time - part of the data time stamp - number of milliseconds since midnight
# ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
# It can also use the data sampling frequency:
#    self.sampling_freq
# and the following variables already set at the initialisation time (via function options):
#    self.logBase - The base of the logartihm
#    self.zeroOffset - Substract current minimal value to zero offset

def calculate(self, sig1, ts_time, ts_date):

    if self.zeroOffset == True:
        sig1 = np.array(sig1) - min(sig1)

    if self.logBase == 'BIN':
        base = 2
    elif self.logBase == 'NAT':
        base = math.e
    elif self.logBase == 'DEC':
        base = 10

    result = stats.entropy(sig1, None, base)
    return result
```

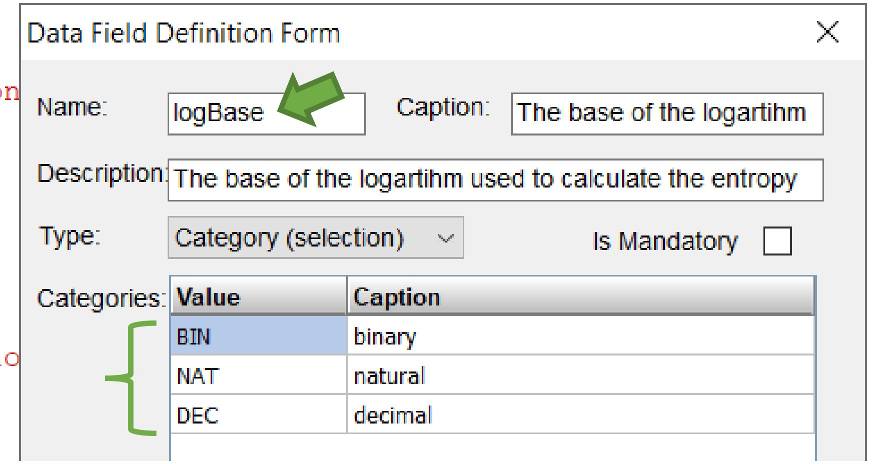
How to use the configured options in the Python script

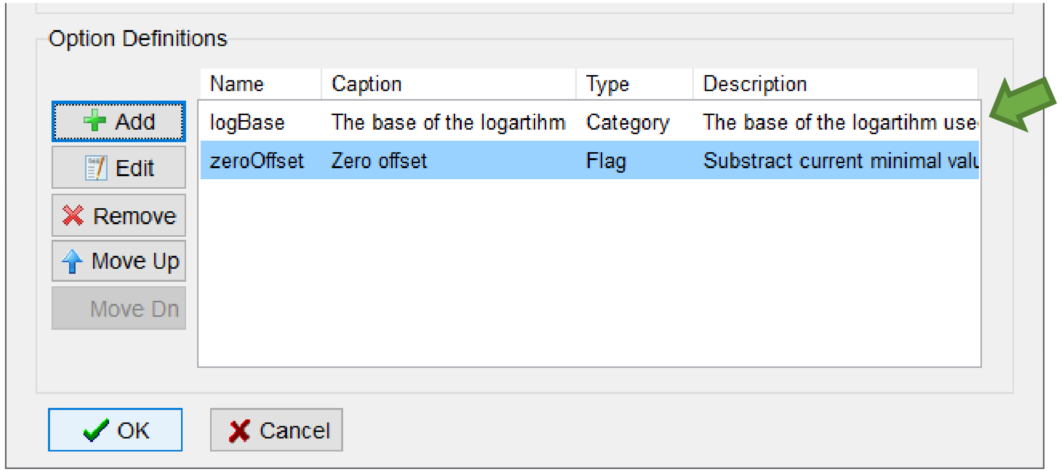
```
# 'calculate' is the main work-horse function.
# It is called with a data buffer (one or more) of size corresponding to the Calculation Window
# It must return one floating-point number
# It take the following parameters:
# sig1 - input variable/signal 1
# ts_time - part of the data time stamp - number of milliseconds since midnight
# ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
# It can also use the data sampling frequency:
#     self.sampling_freq
# and the following variables already set at the initialisation time (via function options):
#     self.logBase - The base of the logartihm
#     self.zeroOffset - Substract current minimal value to zero offset

def calculate(self, sig1, ts_time, ts_date):

    if self.zeroOffset == True:
        sig1 = np.array(sig1) - min(sig1)

    if self.logBase == 'BIN':
        base = 2
    elif self.logBase == 'NAT':
        base = math.e
    elif self.logBase == 'DEC':
        base = 10

    result = stats.entropy(sig1, None, base)
    return result
```



How to use the configured options in the Python script

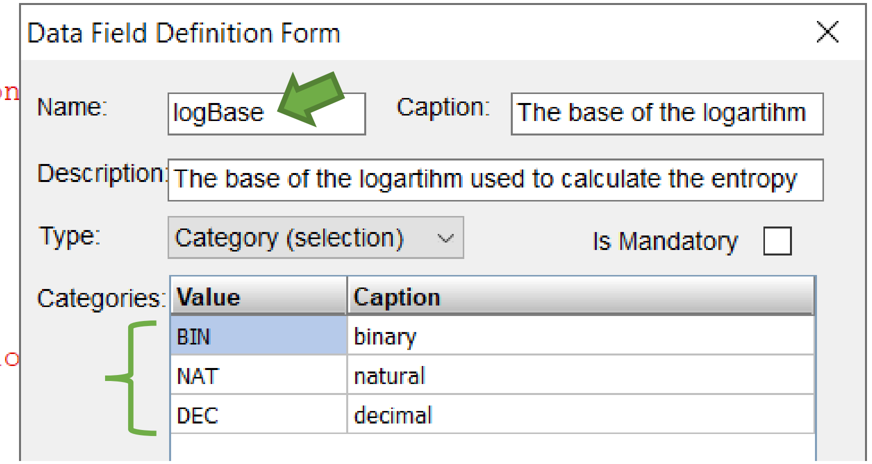
```python
# 'calculate' is the main work-horse function.
# It is called with a data buffer (one or more) of size corresponding to the Calculation
# It must return one floating-point number
# It take the following parameters:
# sig1 - input variable/signal 1
# ts_time - part of the data time stamp - number of milliseconds since midnight
# ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
# It can also use the data sampling frequency:
#     self.sampling_freq
# and the following variables already set at the initialisation time (via function optio
#     self.logBase - The base of the logartihm
#     self.zeroOffset - Substract current minimal value to zero offset

def calculate(self, sig1, ts_time, ts_date):

    if self.zeroOffset == True:
        sig1 = np.array(sig1) - min(sig1)

    if self.logBase == 'BIN':
        base = 2
    elif self.logBase == 'NAT':
        base = math.e
    elif self.logBase == 'DEC':
        base = 10

    result = stats.entropy(sig1, None, base)
    return result
```

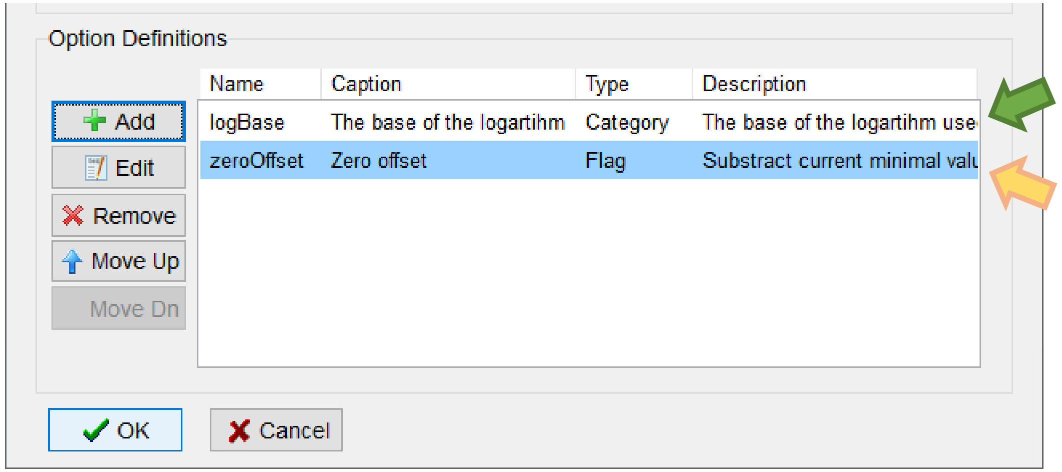




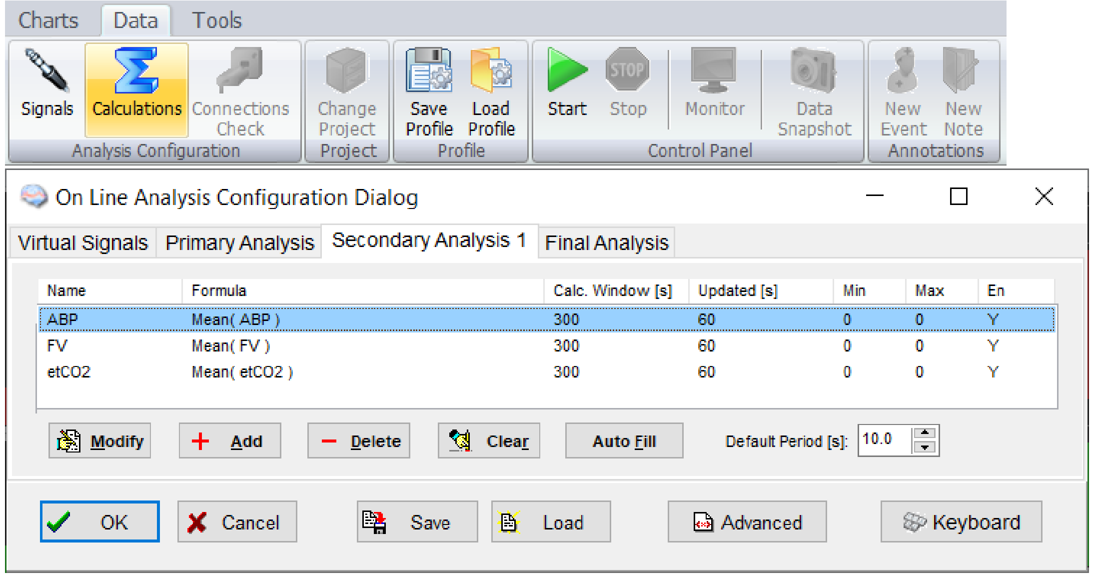
How to use the configured options in the Python script

```python
# 'calculate' is the main work-horse function.
# It is called with a data buffer (one or more) of size corresponding to the Calculation
# It must return one floating-point number
# It take the following parameters:
# sig1 - input variable/signal 1
# ts_time - part of the data time stamp - number of milliseconds since midnight
# ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
# It can also use the data sampling frequency:
#    self.sampling_freq
# and the following variables already set at the initialisation time (via function optio
#    self.logBase - The base of the logartihm
#    self.zeroOffset - Substract current minimal value to zero offset

def calculate(self, sig1, ts_time, ts_date):

    if self.zeroOffset == True:
        sig1 = np.array(sig1) - min(sig1)

    if self.logBase == 'BIN':
        base = 2
    elif self.logBase == 'NAT':
        base = math.e
    elif self.logBase == 'DEC':
        base = 10

    result = stats.entropy(sig1, None, base)
    return result
```

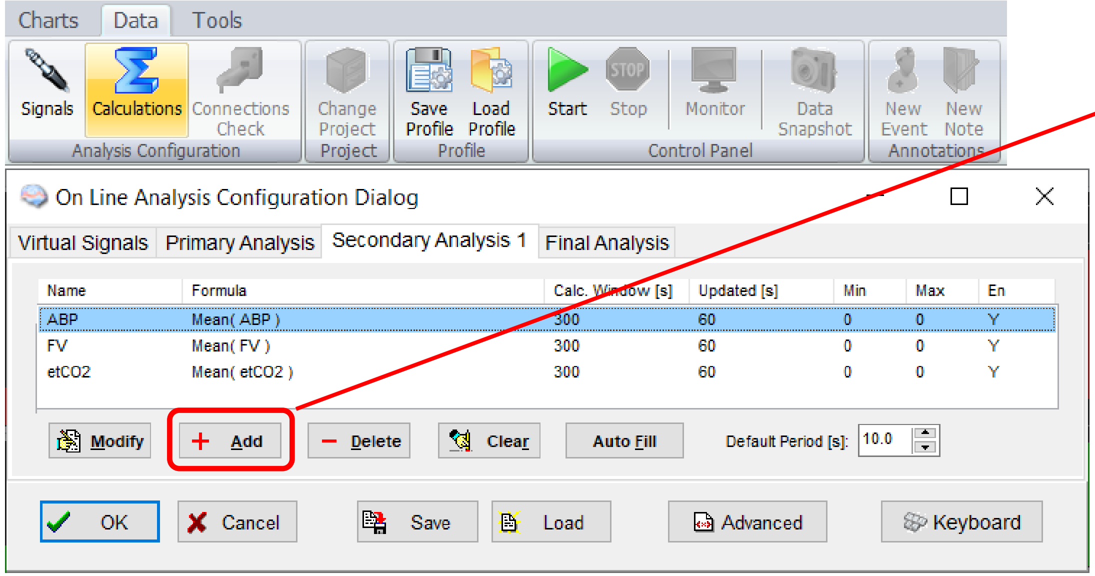


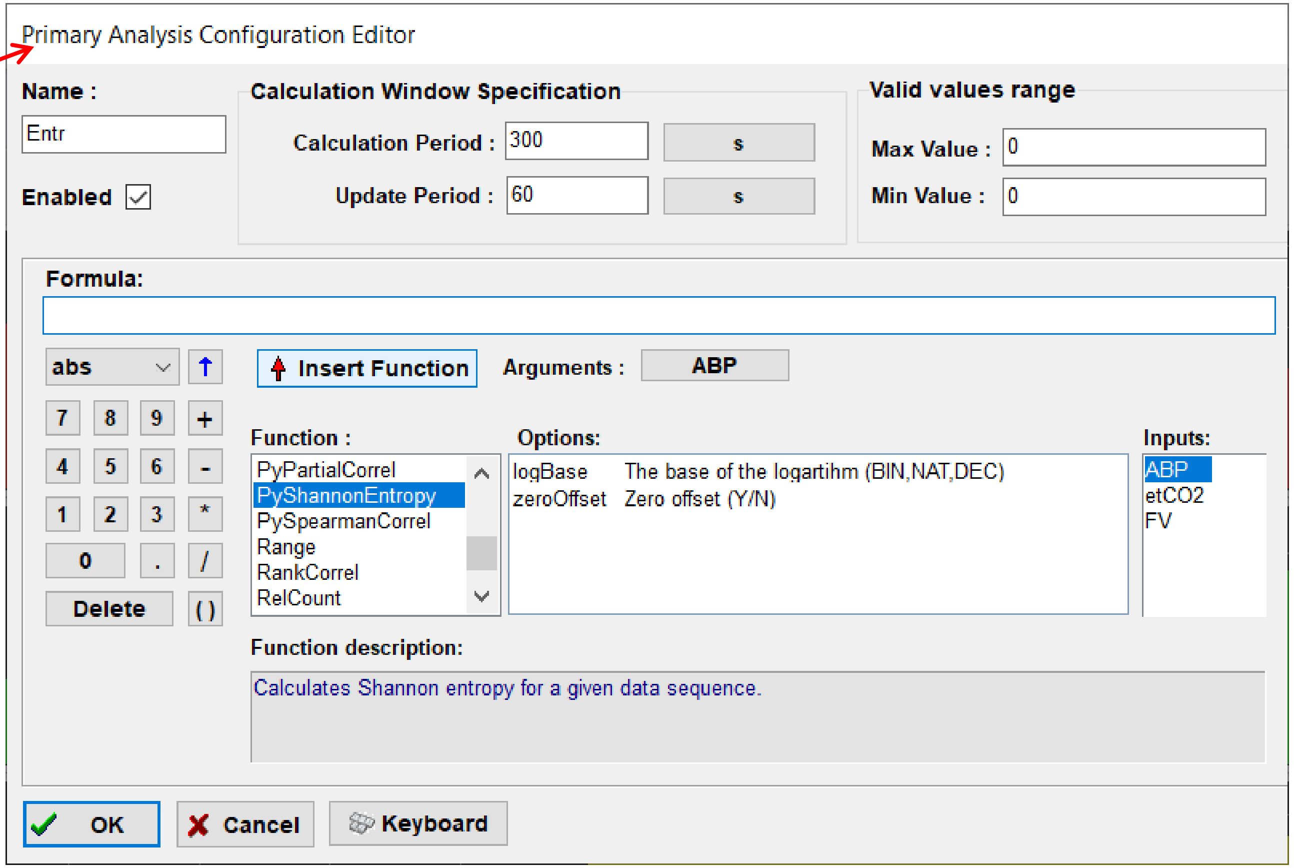


Using user-defined Python function in ICM+

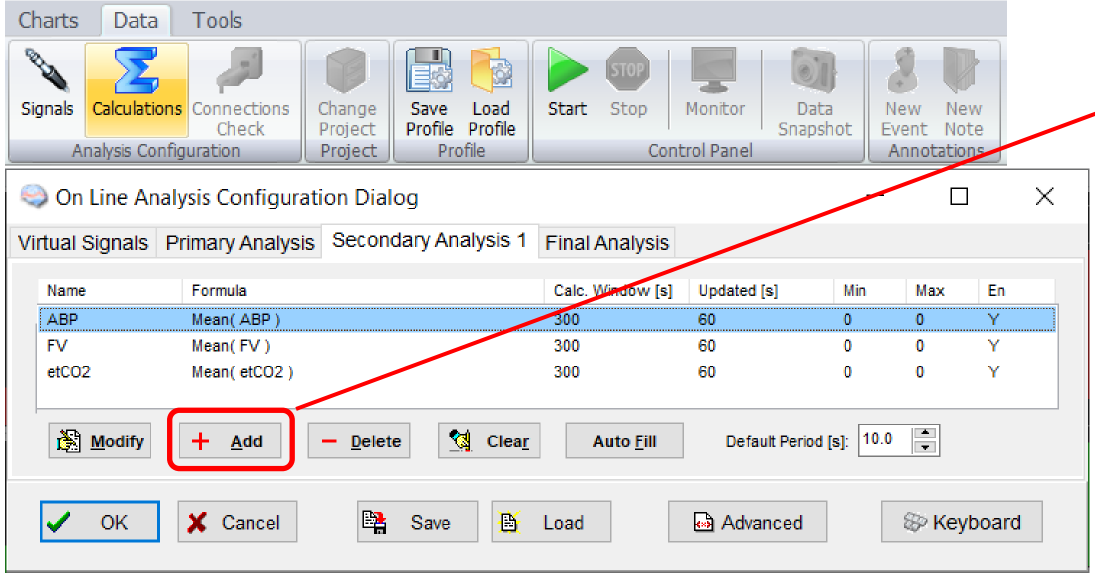


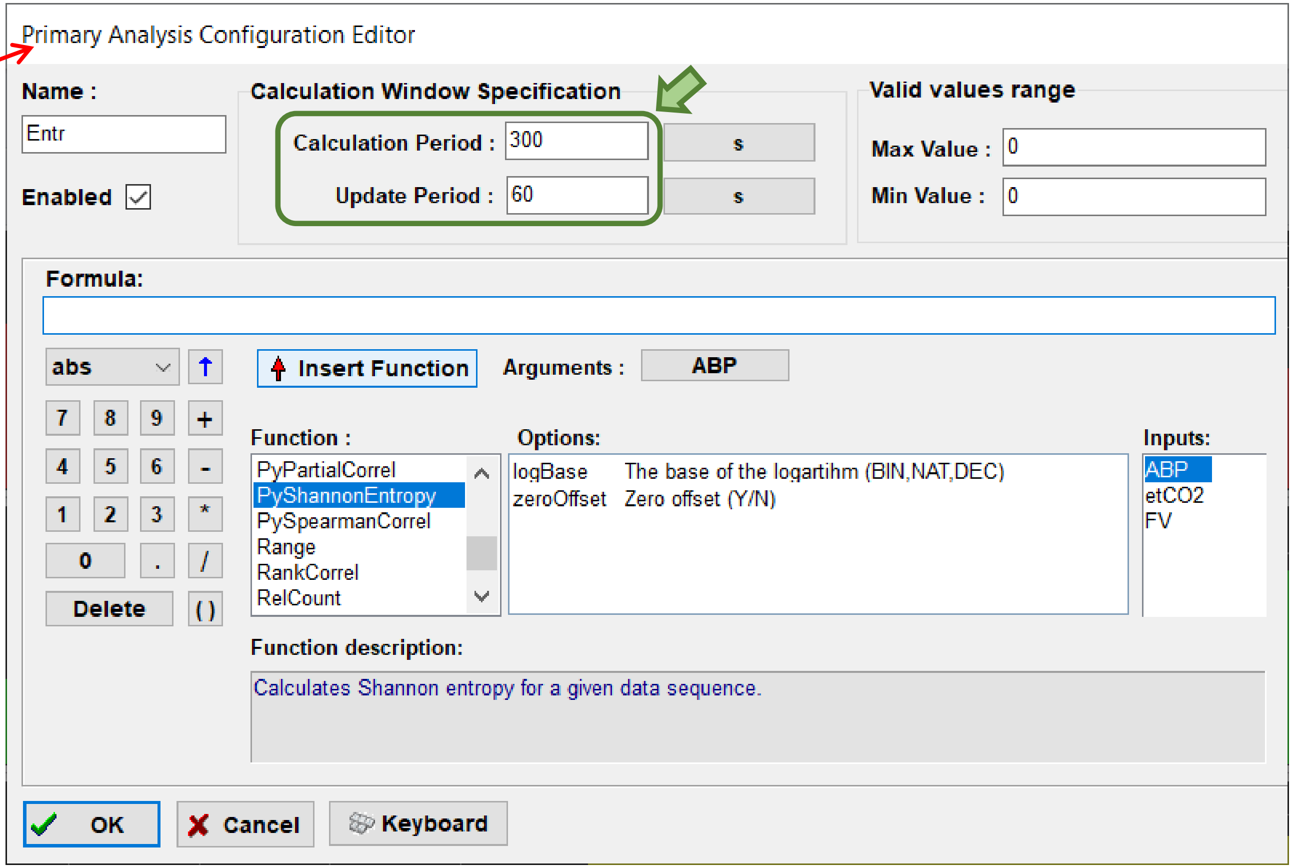
Using user-defined Python function in ICM+



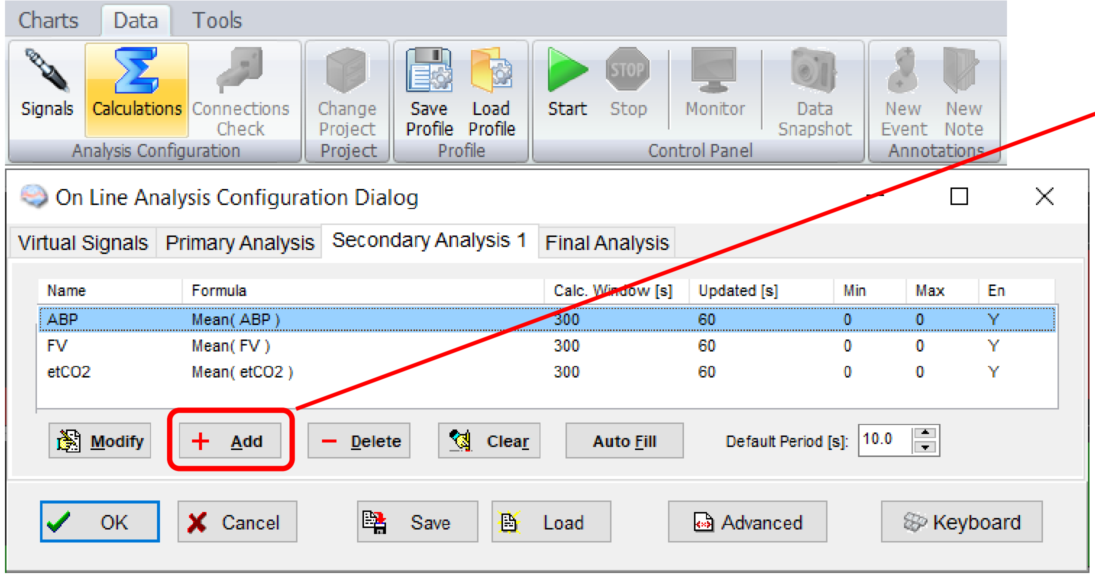


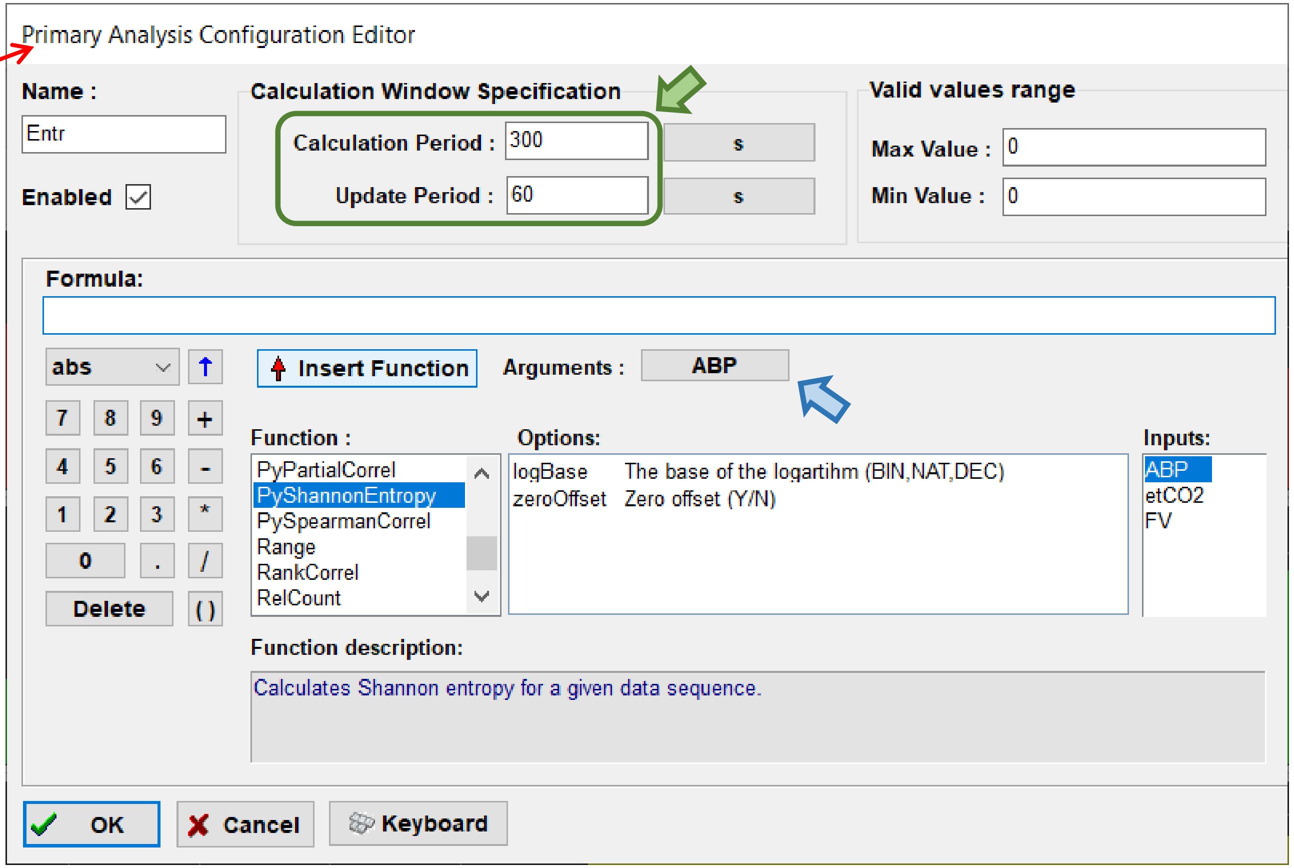
Using user-defined Python function in ICM+



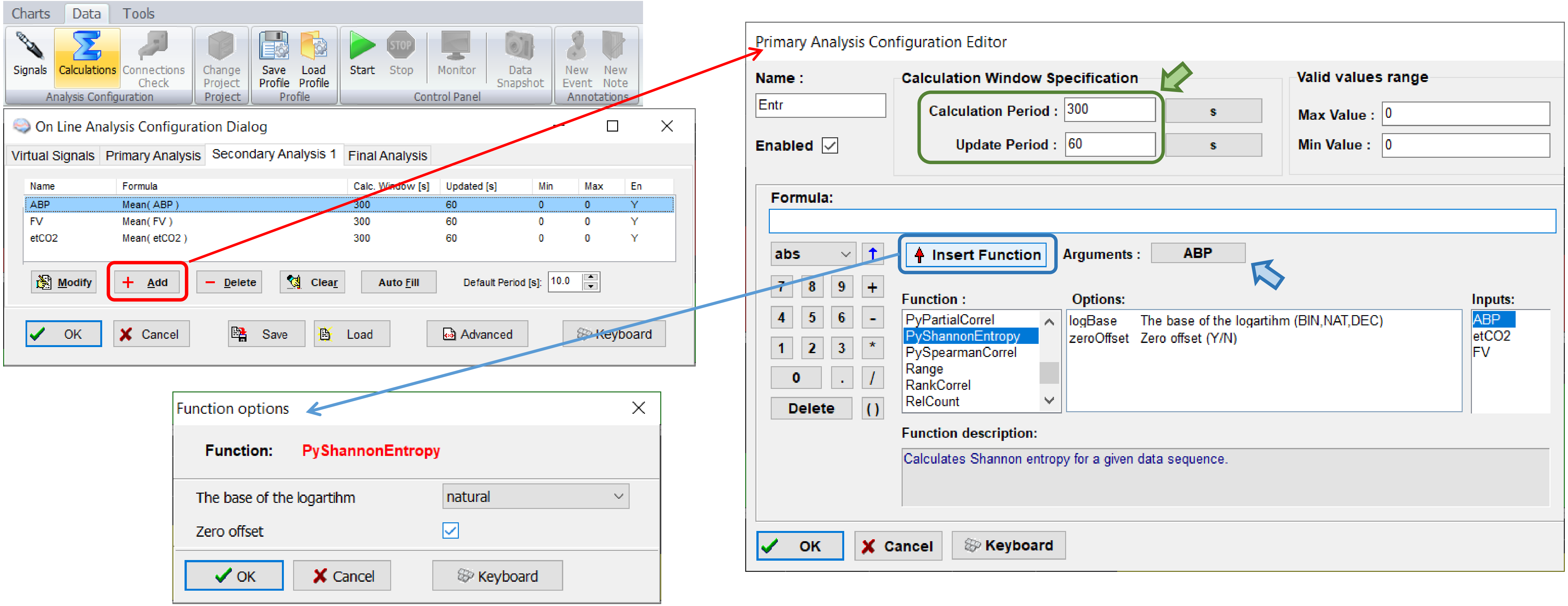


Using user-defined Python function in ICM+

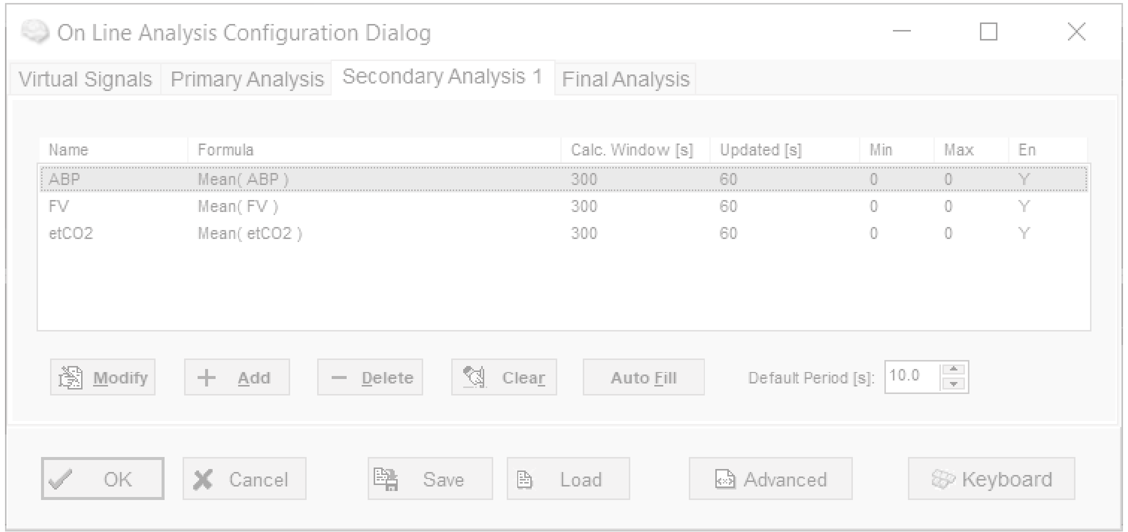


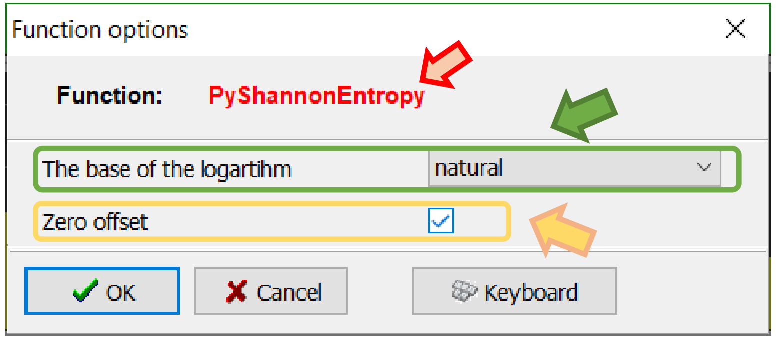


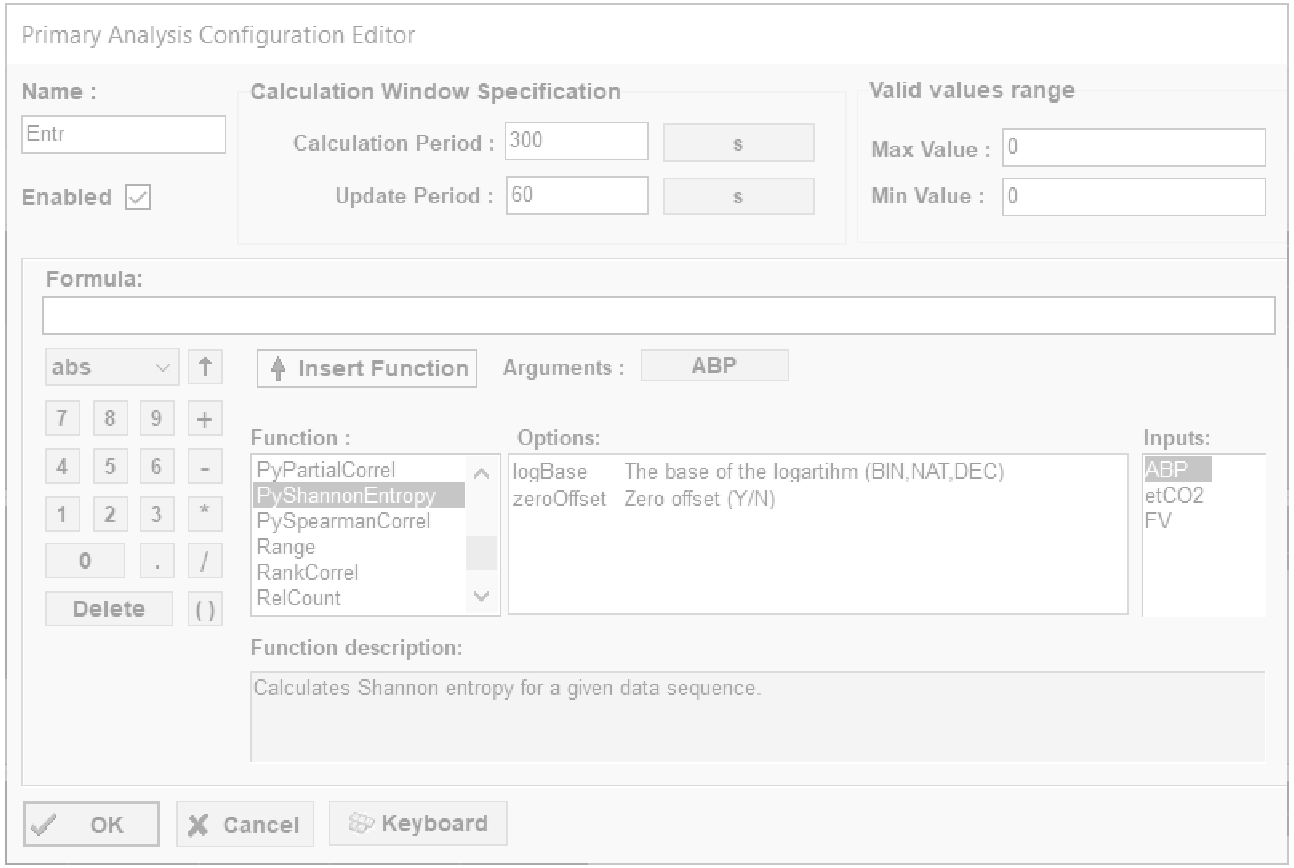
Using user-defined Python function in ICM+



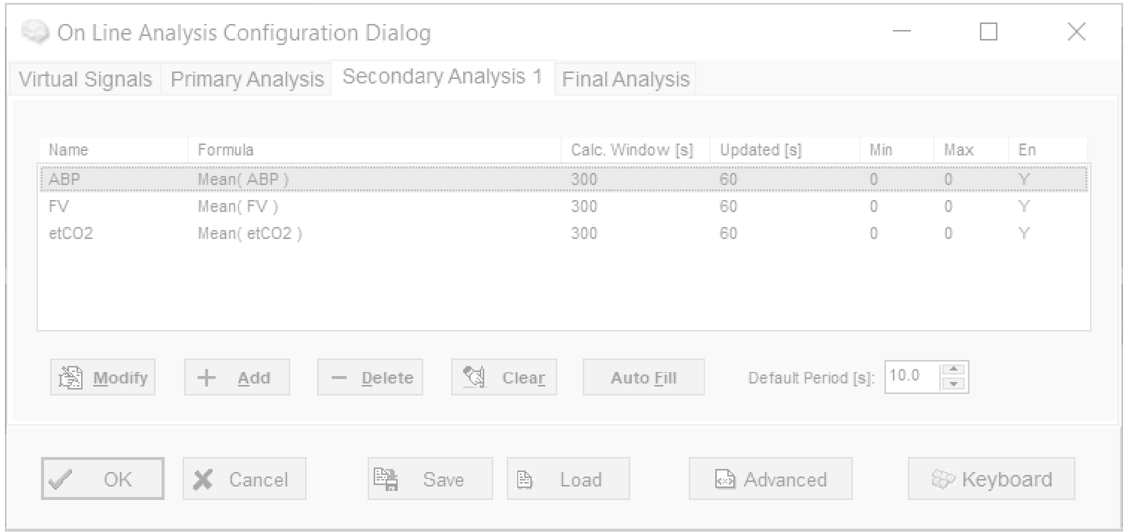
Using user-defined Python function in ICM+

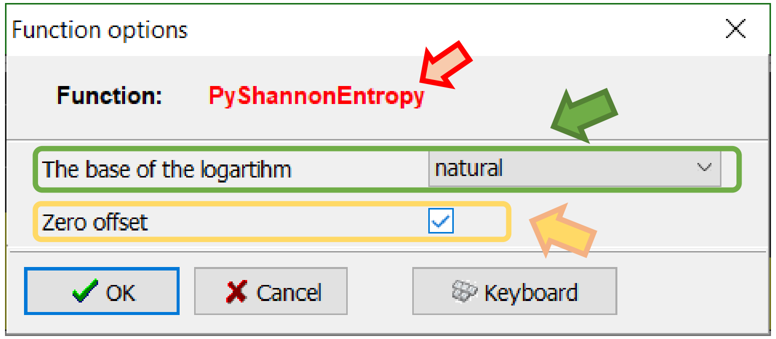


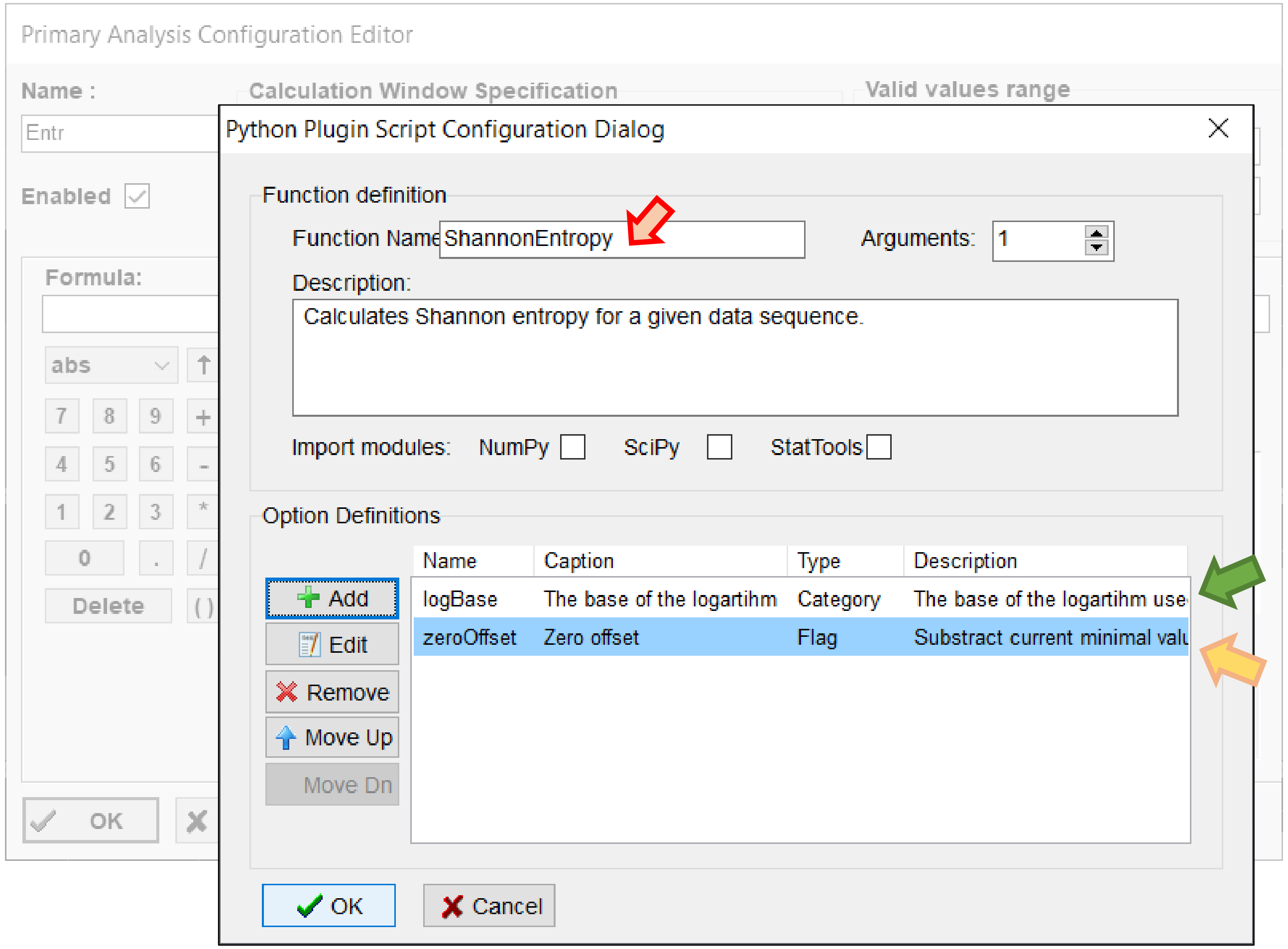




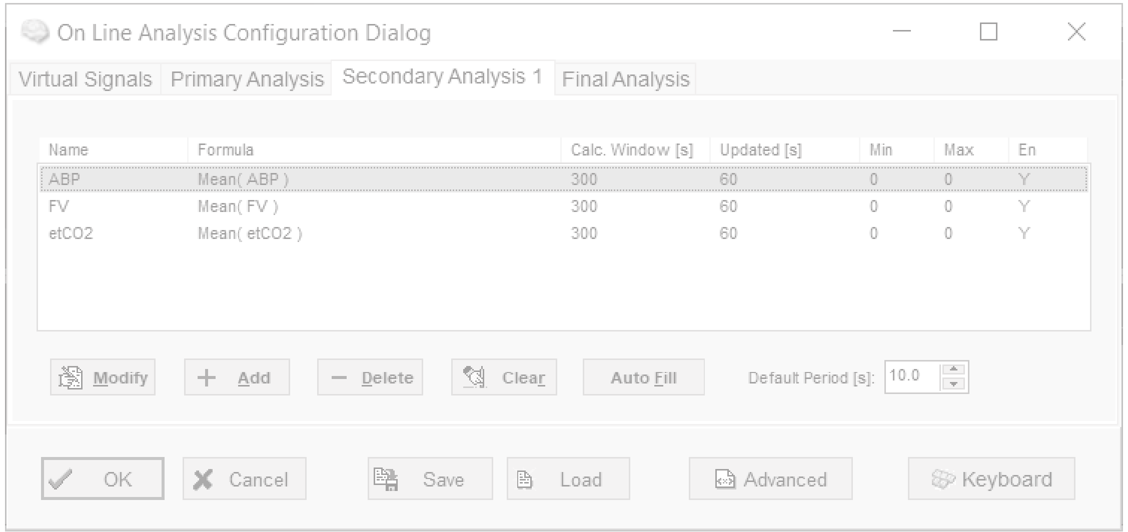
Using user-defined Python function in ICM+

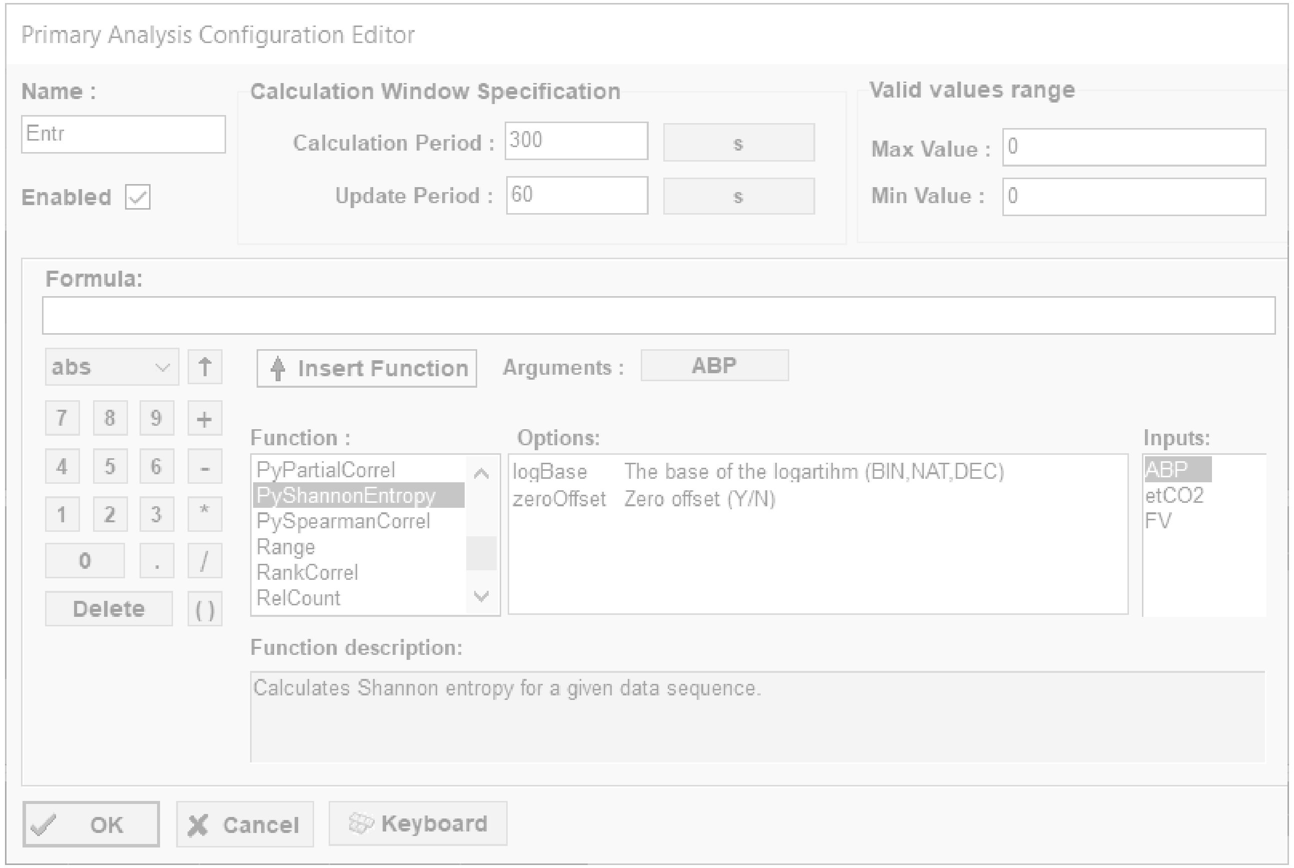


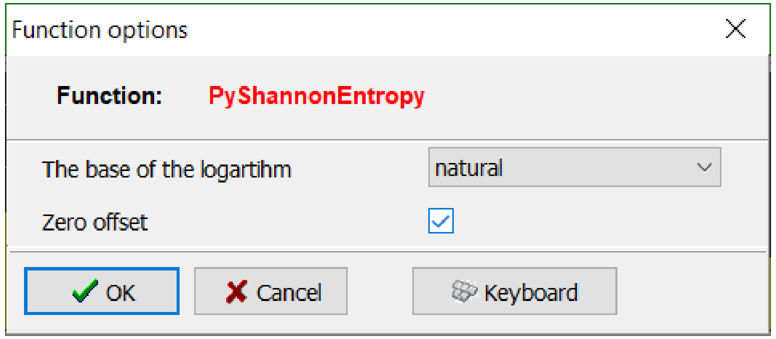




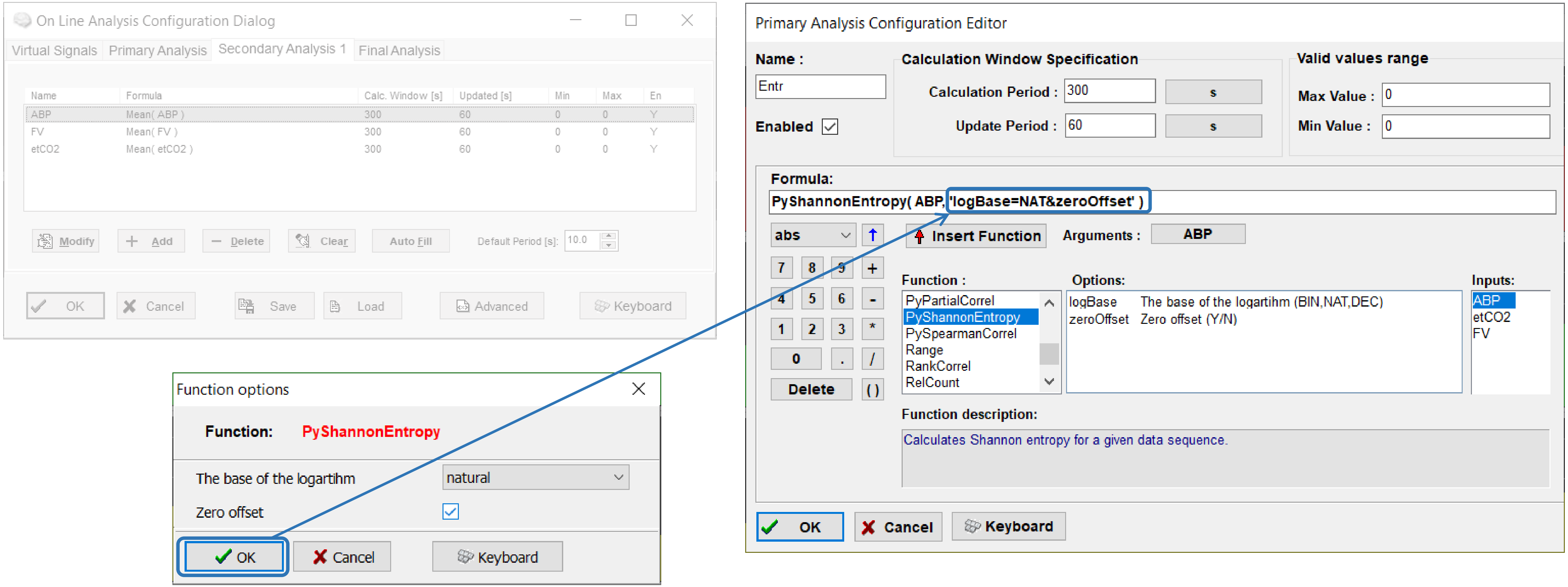
Using user-defined Python function in ICM+



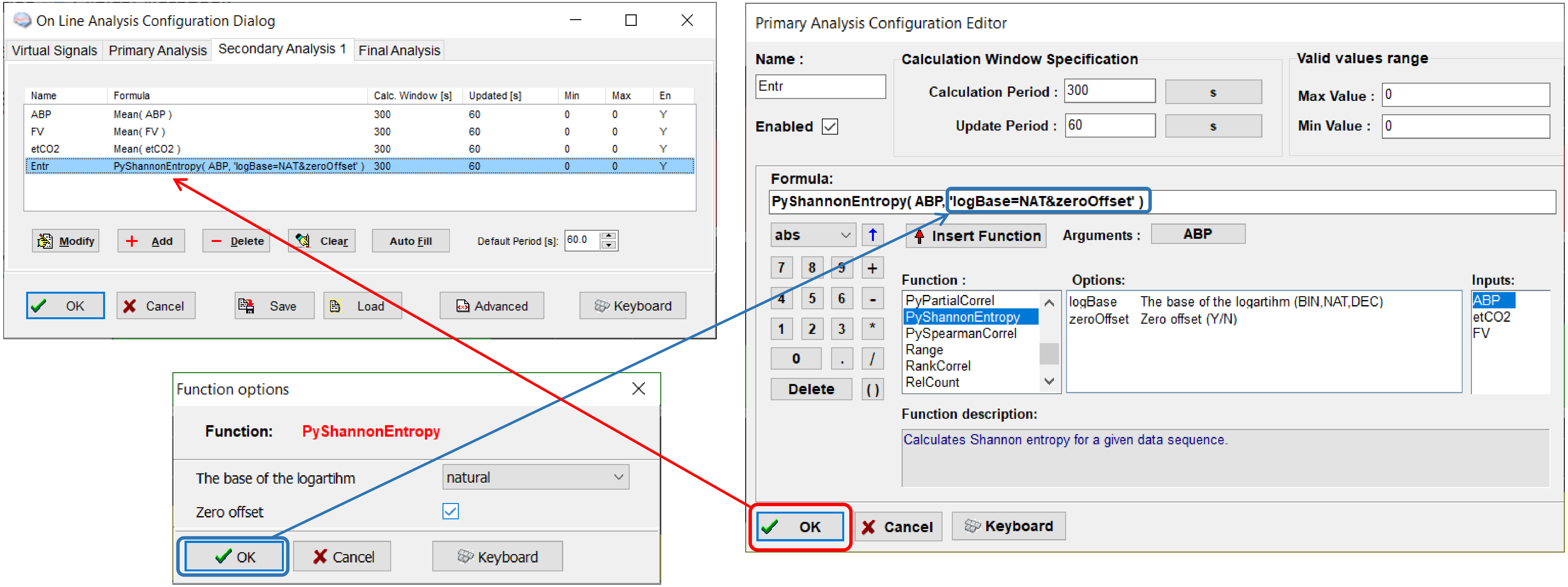




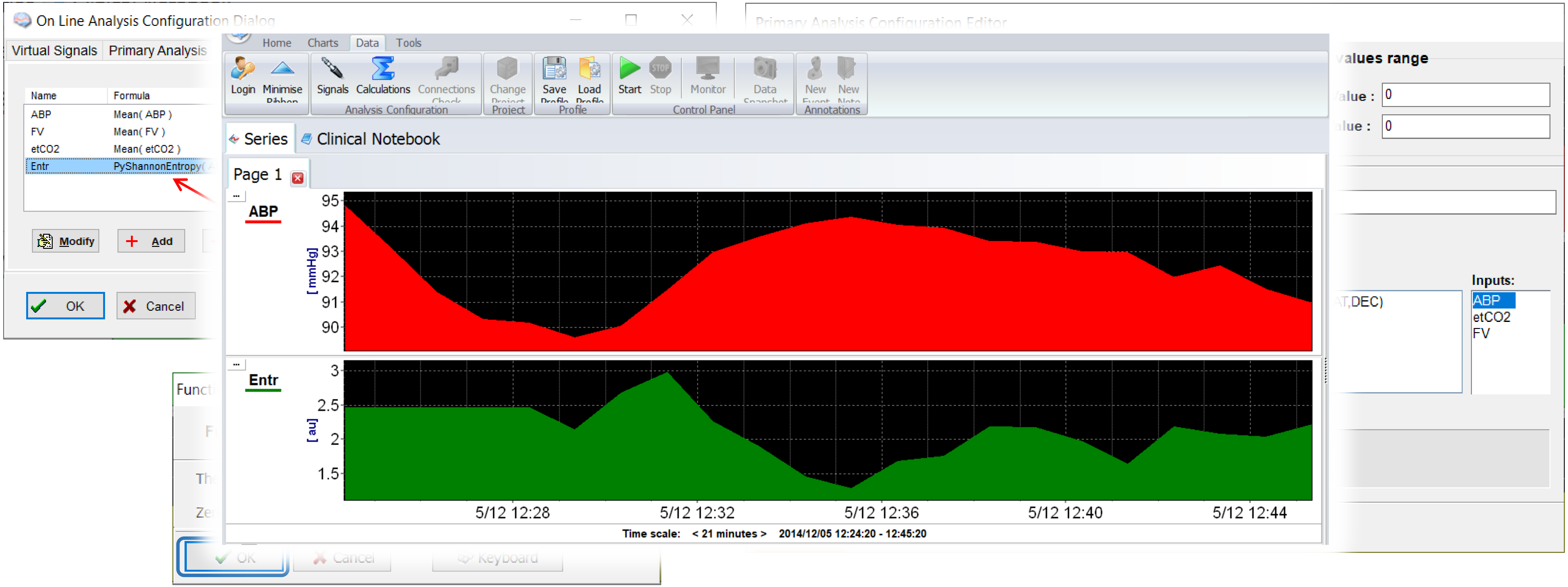
Using user-defined Python function in ICM+



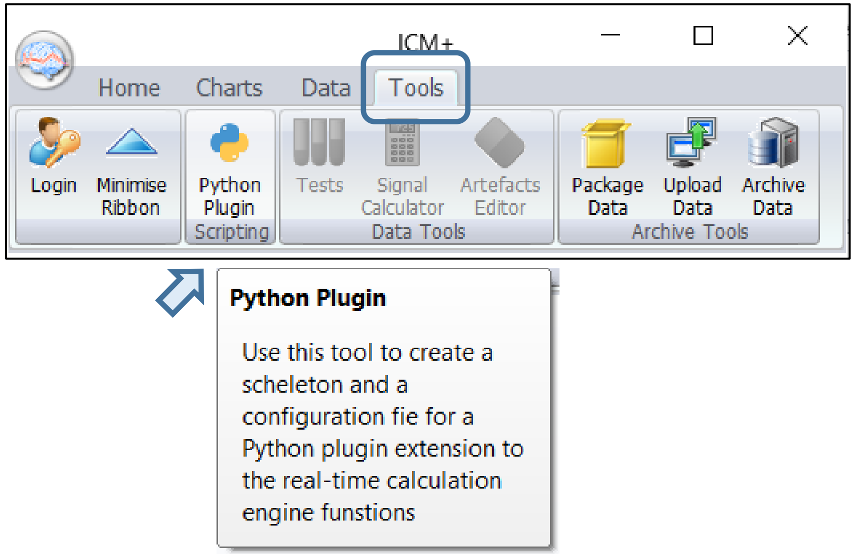
Using user-defined Python function in ICM+



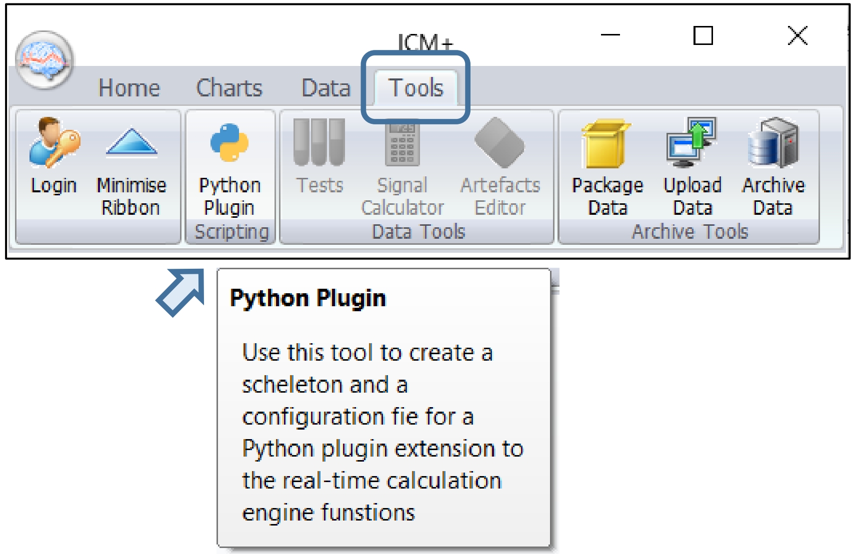
Using user-defined Python function in ICM+

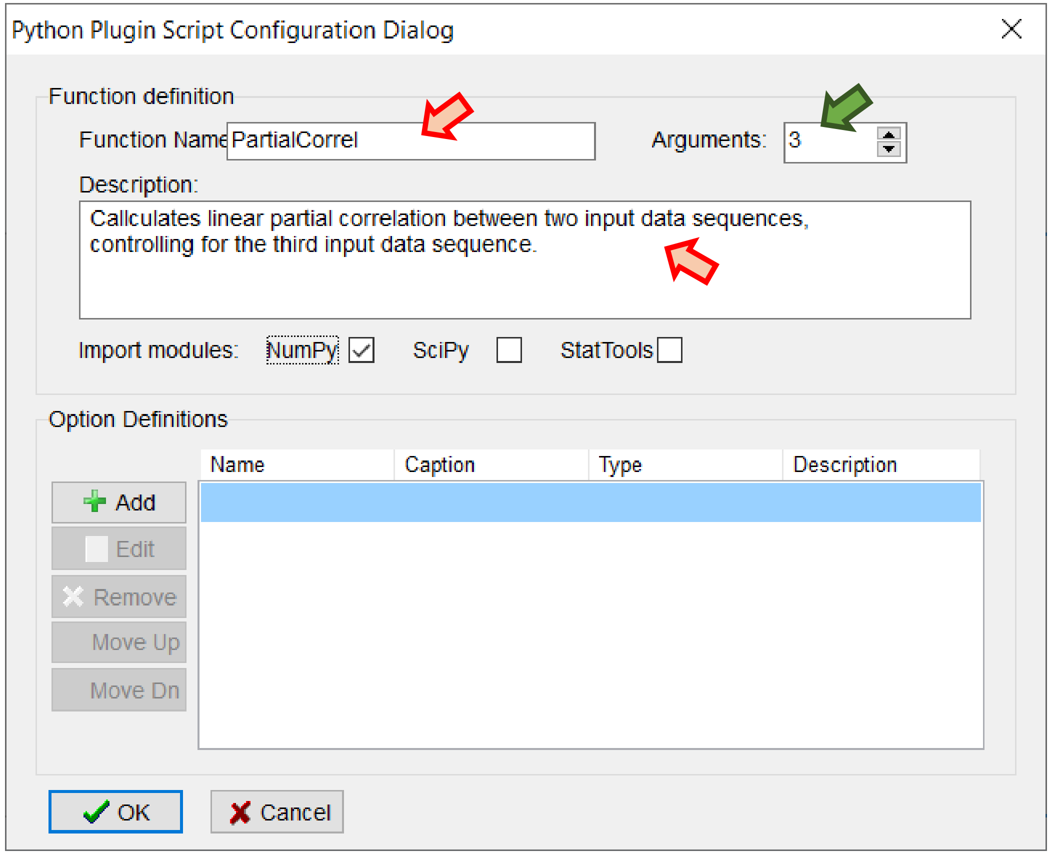


Partial correlation – example of a function with three inputs



Partial correlation – example of a function with three inputs





The generated Python script

```
import numpy as np


class PartialCorrel:

    # DO NOT MODIFY THIS METHOD. It is a part of the ICM+--Python interface.
    def set_parameter(self, param_name, param_value):
        setattr(self, param_name, param_value)

    # ...
    # ...

    # 'calculate' is the main work-horse function.
    # It is called with a data buffer (one or more) of size corresponding to the Calculation Window
    # It must return one floating-point number
    # It take the following parameters:
    # sig1 - input variable/signal 1
    # sig2 - input variable/signal 2
    # sig3 - input variable/signal 3
    # ts_time - part of the data time stamp - number of milliseconds since midnight
    # ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
    # It can also use the data sampling frequency:
    #    self.sampling_freq
    def calculate(self, sig1,sig2,sig3, ts_time, ts_date):
        # my_own_code_here
        result = 0.0
        return result
```

The generated Python script

```python
import numpy as np


class PartialCorrel:

    # DO NOT MODIFY THIS METHOD. It is a part of the ICM+--Python interface.
    def set_parameter(self, param_name, param_value):
        setattr(self, param_name, param_value)

    # ...
    # ...

    # 'calculate' is the main work-horse function.
    # It is called with a data buffer (one or more) of size corresponding to the Calculation Window
    # It must return one floating-point number
    # It take the following parameters:
    # sig1 - input variable/signal 1
    # sig2 - input variable/signal 2
    # sig3 - input vari
    # ts_time - part of                 mp - number of milliseconds since midnight
    # ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
    # It can also use the data  ampling frequency:
    #     self.sampling_freq
    def calculate(self, sig1,sig2,sig3, ts_time, ts_date):
        # my_own_code_here
        result = 0.0
        return result
```

Three input signals

The generated Python script

```
import numpy as np
from par_corr_module import partial_corr
```

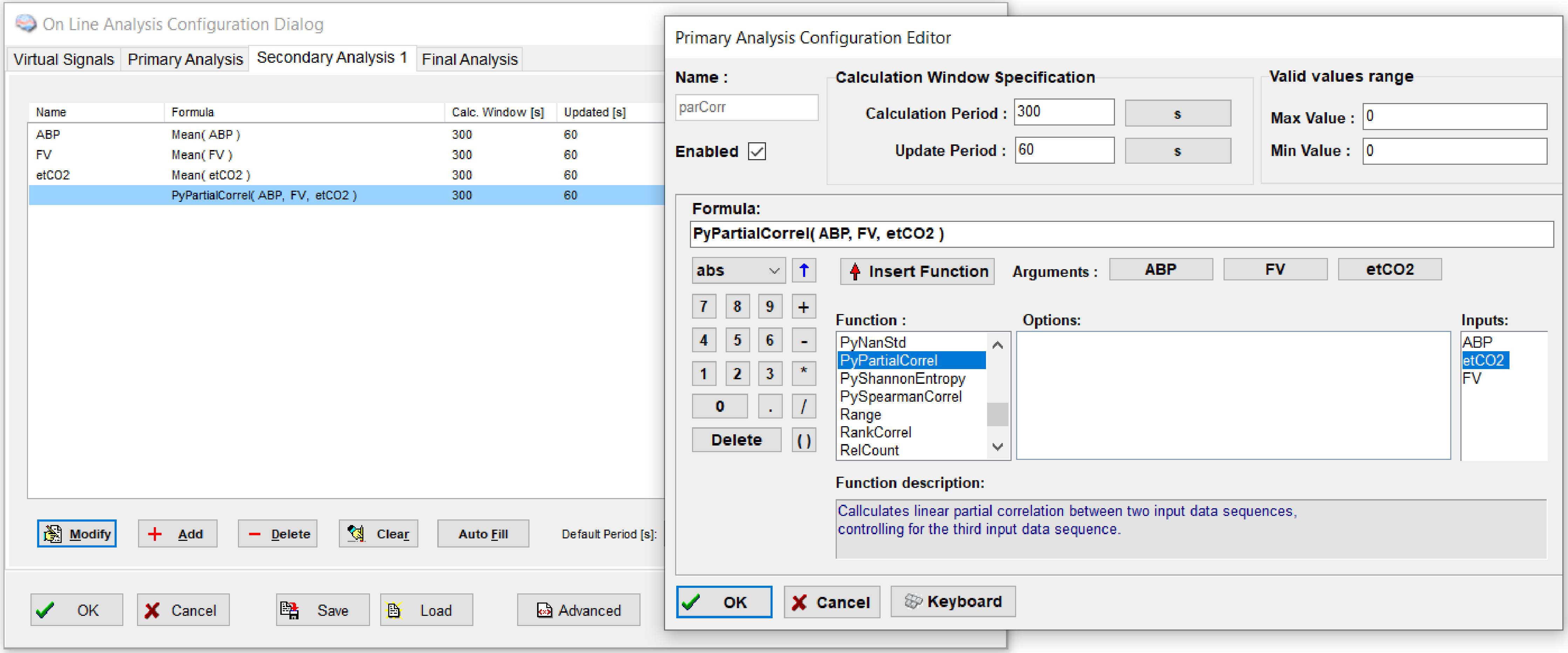
Importing function from another module

```
class PartialCorrel:

    # DO NOT MODIFY THIS METHOD. It is a part of the ICM+--Python interface.
    def set_parameter(self, param_name, param_value):
        setattr(self, param_name, param_value)

    # ...
    # ...

    # 'calculate' is the main work-horse function.
    # It is called with a data buffer (one or more) of size corresponding to the Calculation Window
    # It must return one floating-point number
    # It take the following parameters:
    # sig1 - input variable/signal 1
    # sig2 - input variable/signal 2
    # sig3 - input vari
    # ts_time - part of                  mp - number of milliseconds since midnight
    # ts_date - Part of                    stamp - One plus number of days since 1/1/0001
    # It can also use the data   ampling frequency:
    #    self.sampling_freq
    def calculate(self, sig1,sig2,sig3, ts_time, ts_date):

        A = np.array([sig1, sig2, sig3]).transpose()
        R_coefficients = partial_corr(A)
        return R_coefficients[0,1]
```

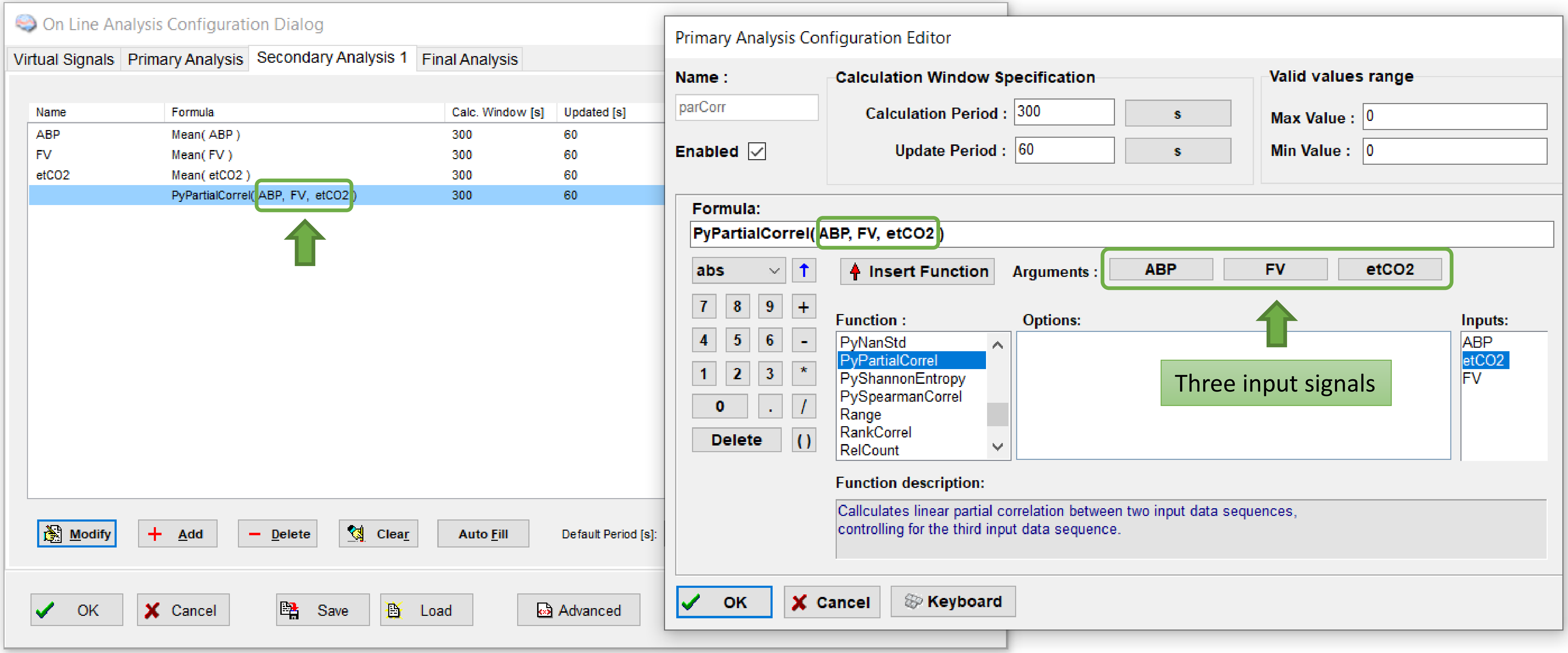
Three input signals

My own code added to the *calculate* method

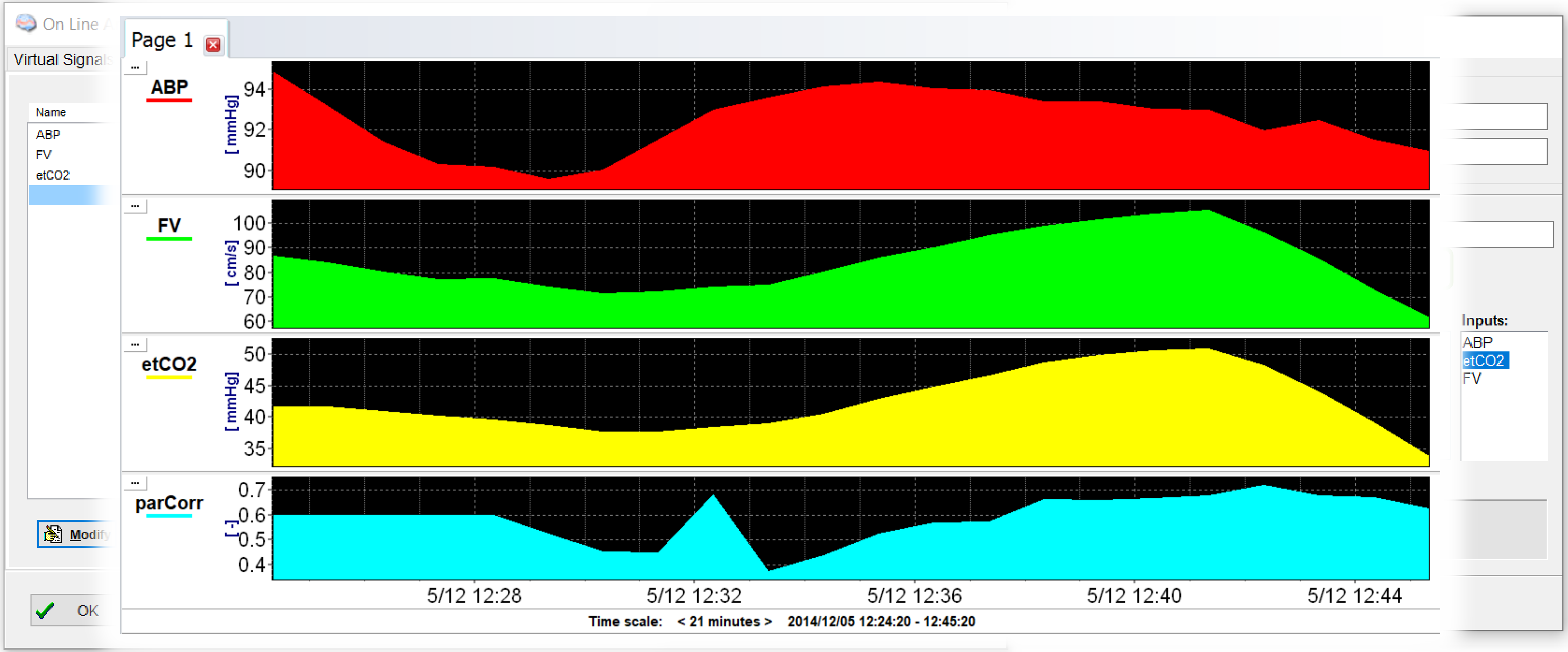
Using user-defined Python function in ICM+



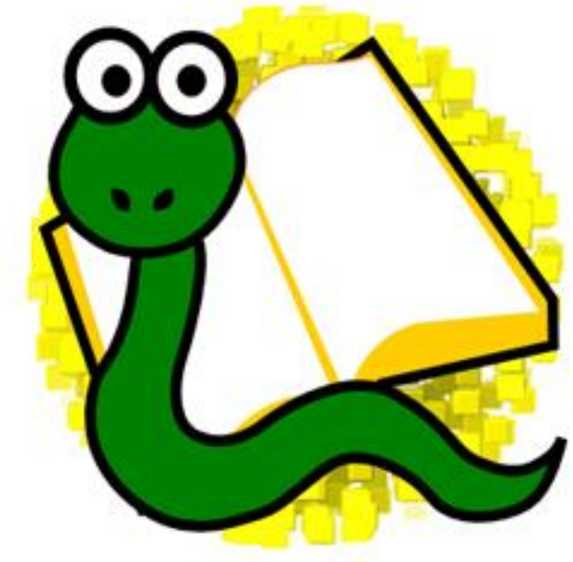
Using user-defined Python function in ICM+



Using user-defined Python function in ICM+

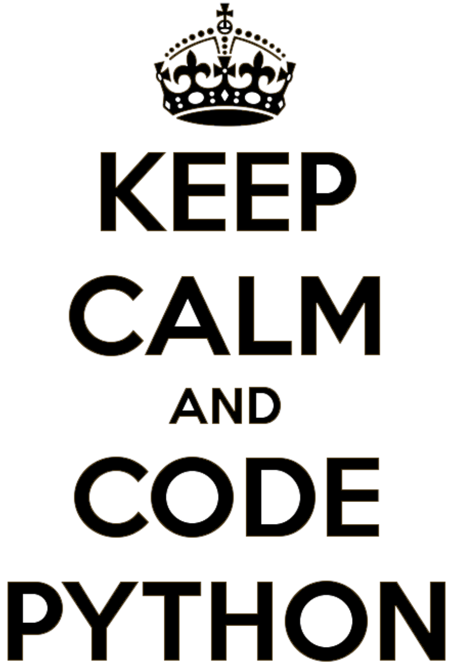


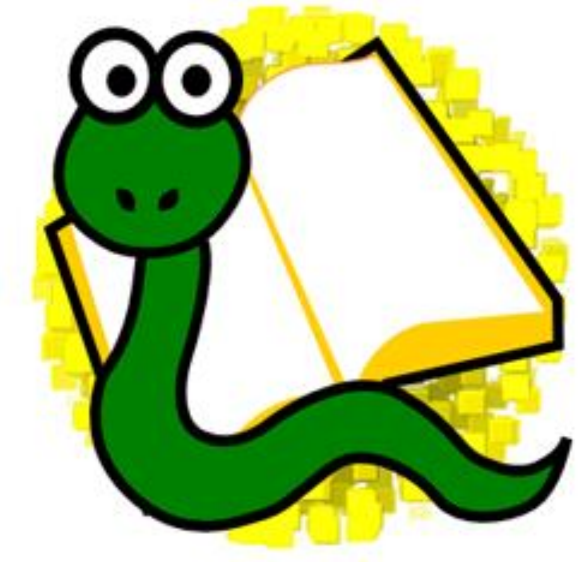
Happy pythoning!



Based on image by Jeffrey Elkner

Happy pythoning!





Based on image by Jeffrey Elkner

I invite you to see my poster (**#321**, Monday, 12:00–13:00)