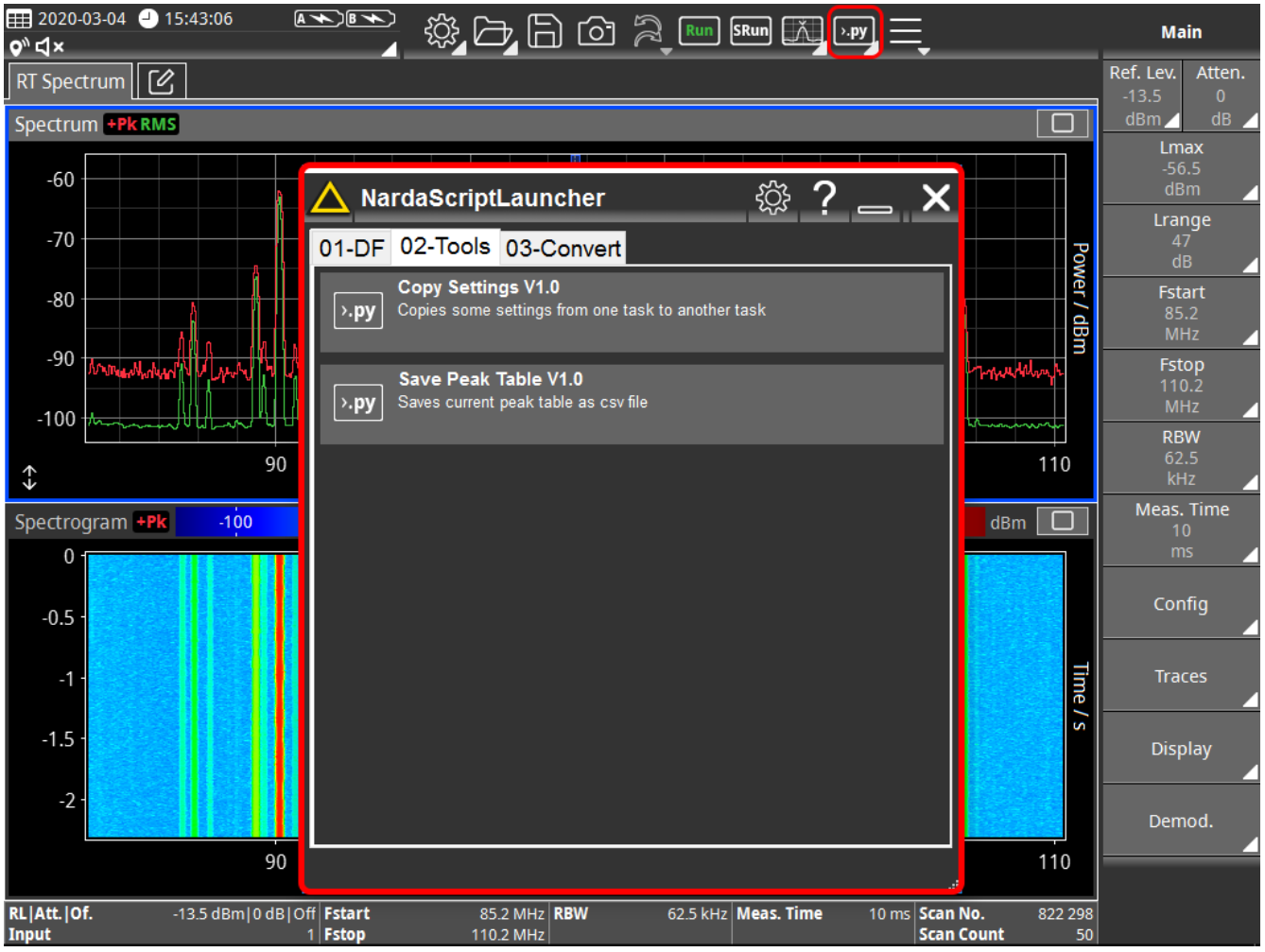
Developing Scripts for SignalShark with PyCharm

To meet specific applications, SignalShark supports Python based scripting. This means that users can write their individual Python code and implement customized functionalities. This document explains the first steps of how to develop such scripts with the Python development environment *PyCharm*.



Content

1 Introduction

1.1 “Narda Script Launcher” Manual

For more information about the “Narda Script Launcher”, Python programming in general, and writing your own scripts, see the detailed “Narda Script Launcher” Manual. This is available for download as an interactive HTML version and as a PDF document on the Narda website.

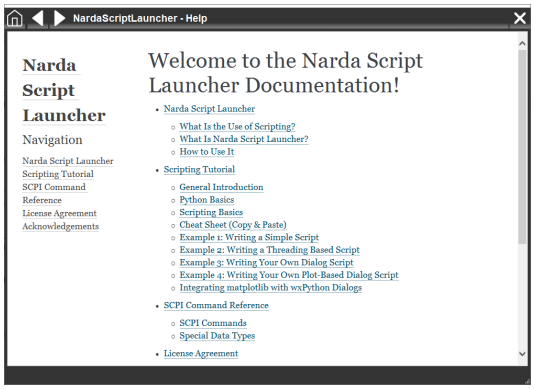


Fig. 1. Narda Script Launcher integrated help.

The manual covers the following topics among others:

- Narda Script Launcher
  - What Is Narda Script Launcher?
  - How to Use It
- Scripting Tutorial
  - Python Basics
  - Scripting Basics
  - Cheat Sheet (Copy & Paste)
  - Example 1: Writing a Simple Script
  - Example 2: Writing a Threading Based Script
  - Example 3: Writing Your Own Dialog Script
  - Example 4: Writing Your Own Plot-Based Dialog Script
- SCPI Command Reference
- License Agreement
- Acknowledgements

1.2 “Narda Script Launcher” and Scripts for SignalShark

SignalShark has many features and options making it a powerful real-time spectrum analyzer, monitoring receiver and direction finding system. In everyday work life, many different applications may slightly differ in their detail requirements. To meet also these specific needs, SignalShark supports Python based scripting. This means that users can write their individual Python code and implement customized functionalities.

If “Narda Script Launcher” is installed on SignalShark, the graphical user interface allows to select and start a script directly from within the SignalShark application (requires “Option, SCPI Remote Control”).

“Narda Script Launcher” can also be installed on a computer so that scripts can be debugged and operated remotely from computer via network connection to SignalShark. In such case, the scripts remain stored on the computer.

In practice, this makes it possible to:

- Automate routine tasks
- Provide guided measurements for novices using message boxes or wizards
- Add new measurement evaluation functions
- Provide complete measurement automation

Fig. 2. “Narda Script Launcher” - Main user interface

2 The Basic Setup to Code and Use SignalShark Scripts

This chapter shall clarify the basic setup of “Narda Script Launcher” and of PyCharm on SignalShark and on a computer.

Normally scripts are coded on a computer. Therefore the required development environment (here: PyCharm) must be installed on the computer.

In order to run scripts, “Narda Script Launcher” must be available either on the computer or on SignalShark or on both. It is recommended to install “Narda Script Launcher” on both devices! In such case it will be possible to…:

- … test, debug and run scripts remotely from computer via network connection to SignalShark.
- … test and run scripts locally on SignalShark after they have been transferred to SignalShark.

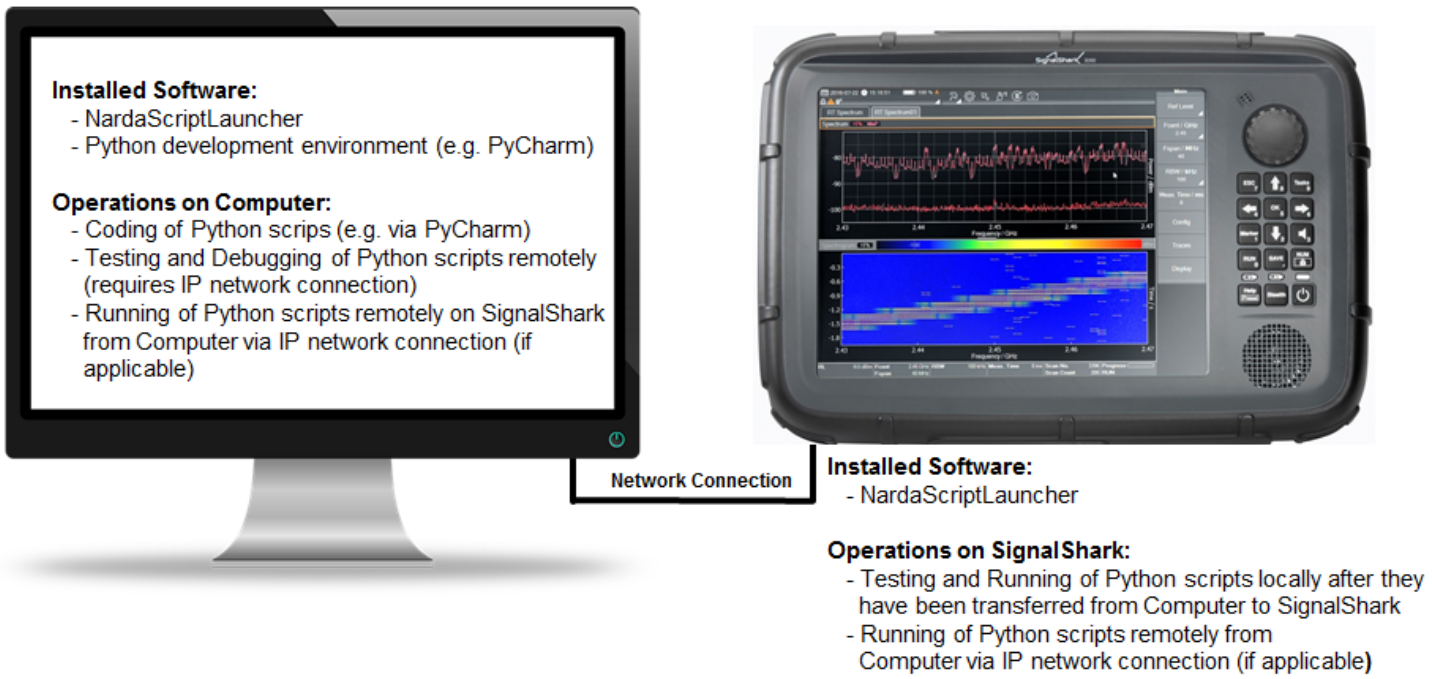


Fig. 3. “Narda Script Launcher” and PyCharm installation options for computer and SignalShark

3 How to Setup PyCharm

Note:

Narda Safety Test Solutions GmbH does not assume any warranty and liability for the use of 3rd party products like PyCharm. PyCharm is a recommendable code editor for Python, yet users are free to use any other Python code editor.

Before installing PyCharm, ensure that “Narda Script Launcher” has been installed to your computer and to SignalShark (see “Narda Script Launcher” Manual).

1.) If not available on your computer yet, download PyCharm, e.g. from: https://www.jetbrains.com/pycharm/download

2.) Install PyCharm with its default setup.

3.1 Creating a New Virtual Environment for All “Narda Script Launcher” Projects

Python offers the possibility to use so-called virtual environments for projects. These contain their own Python interpreter and allow using different Python packages with different dependencies for different projects.

1.) Open PyCharm

2.) Tap on “Create New Project” chapter

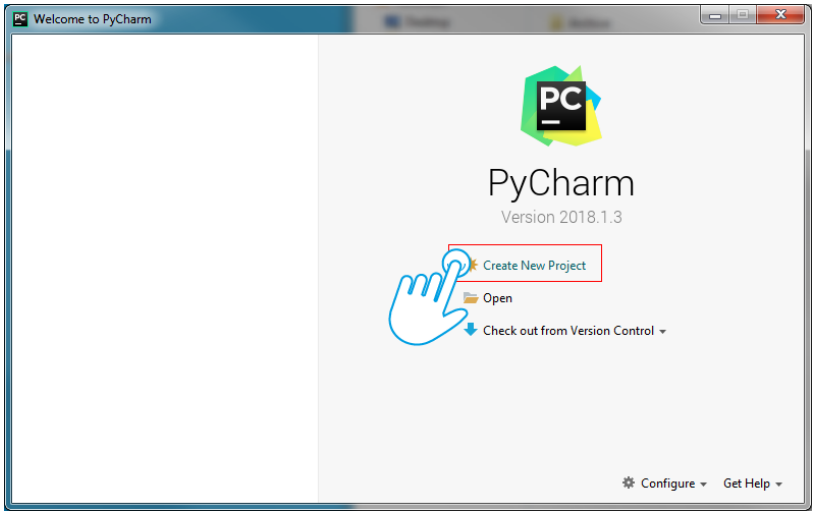


Fig. 4. Create new project in PyCharm

3.) Within the "Create Project" dialog:

a. Define a default project, e.g.:

"C:\Users\[WindowsUser]\PycharmProjects\untitled"

This project only serves the purpose of creating a proper project directory and can be deleted afterwards.

b. Expand the “Project Interpreter:” settings by clicking on "Project Interpreter: New Virtualenv environment"

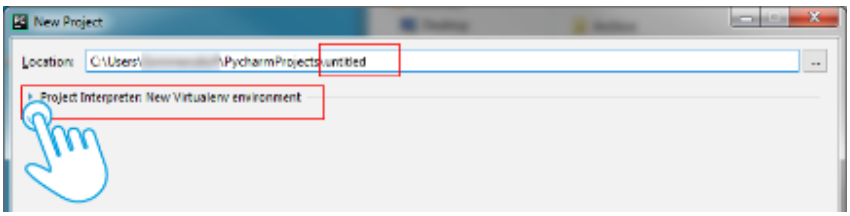


Fig. 5. PyCharm project configuration

c. Select "New environment using Virtualenv"

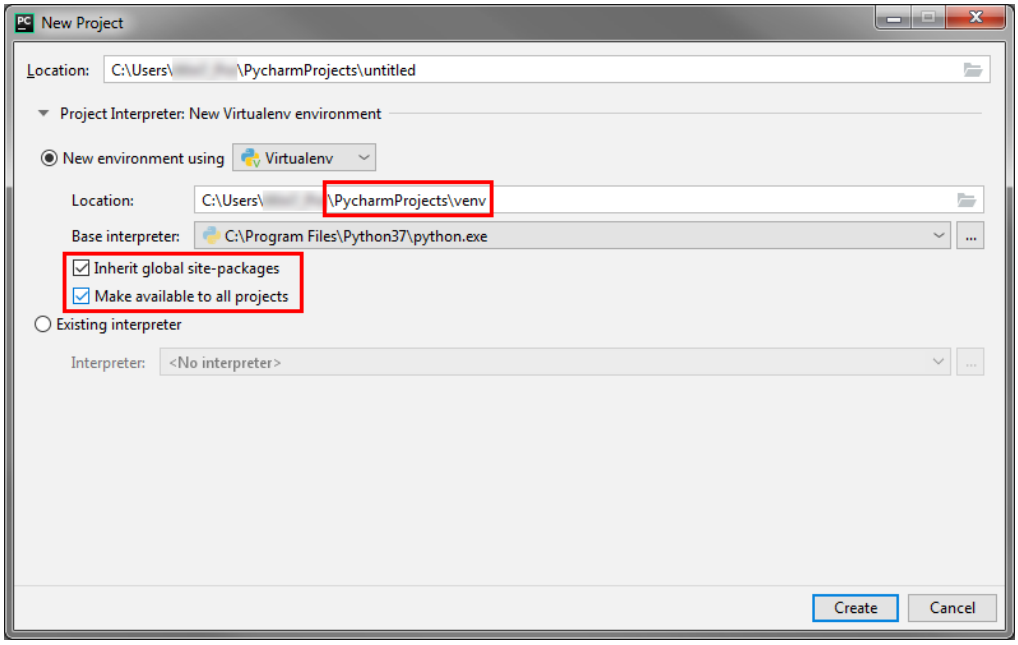


Fig. 6. PyCharm venv configuration

d. Set "Location" to "C:\Users\[WindowsUser]\PycharmProjects\venv"

e. Do not change the default setting for "Base interpreter"

f. Check "Inherit global site-packages"

g. Check "Make available to all projects"

h. Click on "Create"

4.) Close PyCharm !

At the given project location a project folder and a venv folder for the virtual environment have been created by PyCharm as shown exemplarily below:

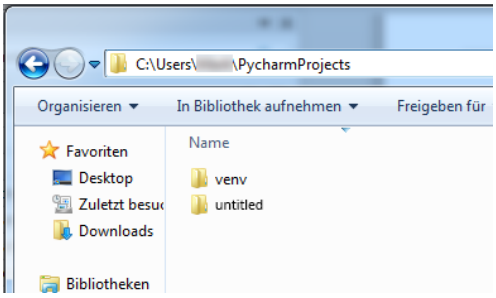


Fig. 7. Project folder in user directory after Project setup in PyCharm

5.) Delete the default project, here “untitled”, as it will not be required anymore.

3.2 The PyCharm Project “UserScriptDev”

The project UserScriptDev shall help you to start developing your own scripts quickly and easily.

1.) Copy the project folder provided by Narda (“UserScriptDev”) into the PyCharm project directory,
i.e.: "C:\Users\[WindowsUser]\PycharmProjects\UserScriptDev”

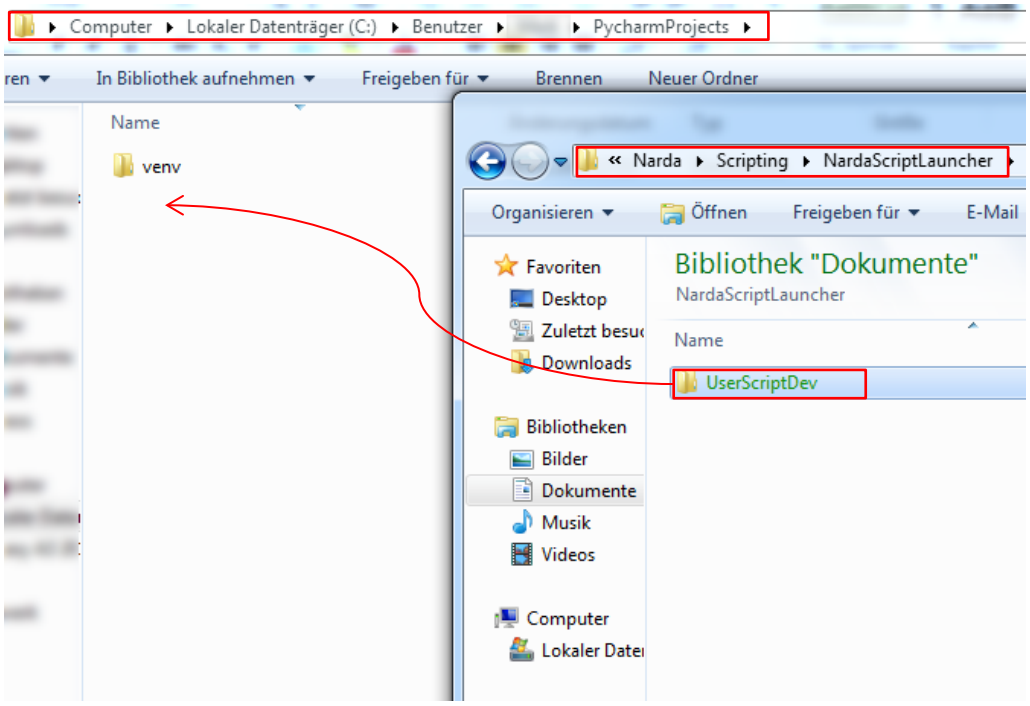


Fig. 8. Copy UserScriptDev (provided by Narda) into project directory

2.) Open the newly added PyCharm project “UserScriptDev” by the following steps:

a. Click on “Open” in the PyCharm application
b. Select the newly inserted project folder (here: “UserScriptDev”). Do not select any specific file within that project folder!
c. Click on “OK”

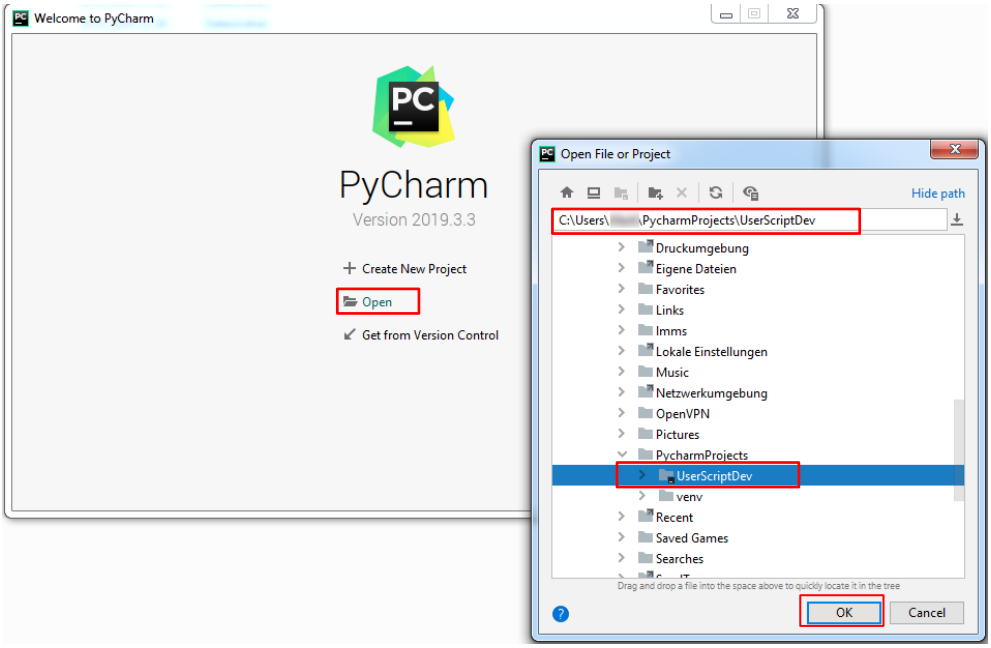


Fig. 9. Open project UserScriptDev

PyCharm will open the project as indicated below:

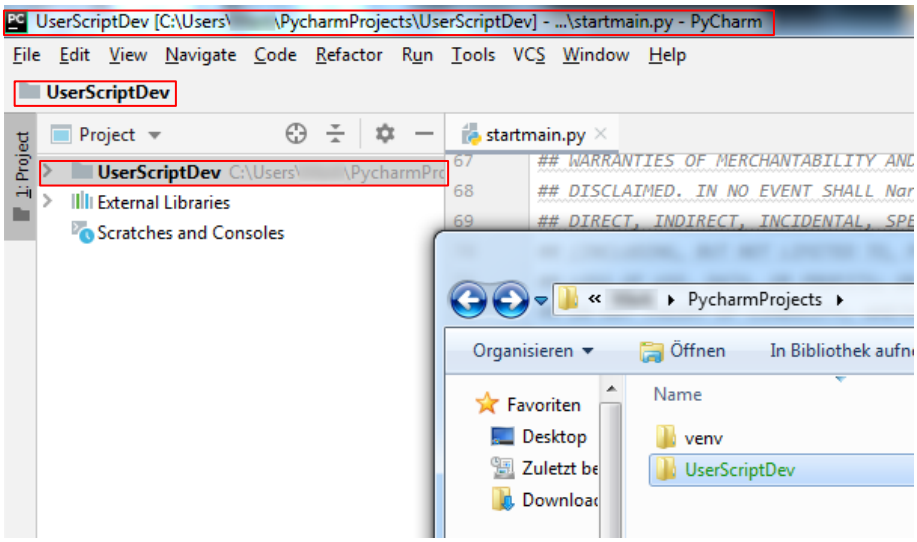


Fig. 10. Opened UserScriptDev project

The project directory and PyCharm are now correctly set up to code scripts for SignalShark using the PyCharm virtual environment in a capsuled directory.

> ***Note:***
>
> *If you exit PyCharm without explicitly closing the project, it will be displayed again the next time you open PyCharm!*

4 Packages and Templates in a PyCharm Project

This chapter gives a practical overview on script packages and templates in a PyCharm project.

1.) Start PyCharm

2.) Open an applicable project, here “UserScriptDev” as created according to section 3.2 The PyCharm Project “UserScriptDev”:

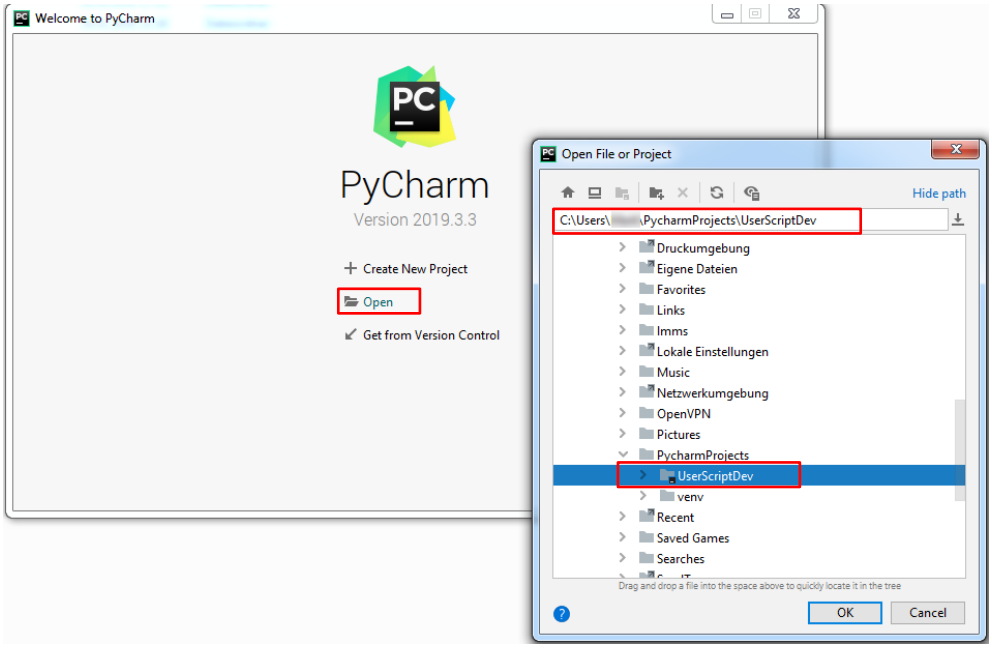


Fig. 11. Open UserScriptDev project

4.1 Modules and Packages

4.1.1 Modules (Python Files)

A SignalShark script is a Python file based on a “UserScript” template that can be displayed and executed by the “Narda Script Launcher”. It has the file extension “.py”.

A file containing Python code is also called a **module** in Python.

4.1.2 Packages (Python Folders)

A PyCharm project can contain many different modules for different purposes. To keep track of the available scripts within a project, it is useful to group them into appropriately named folders according to their application purpose. Such subfolders are called **packages** in Python.

The project UserScriptDev comes with four default packages:

- **baseexamples**
  - example01.py – Copy Settings
    Contains a simple script that copies some settings from one task to another task.
  - example02.py – Peak Trigger
    Contains a measurement thread based script that monitors the spectrum and shows a message if a signal exceeds a certain level.
  - pythonbasics.py – Python Basic Knowledge
    Contains basic explanation of the Python scripting language.
- **example03**
  - Example that shows how to write a script that has a graphical user interface.
- **example04**
  - Example that shows how to write a script that can display data graphically..
- **templates**
  - Contains several template files that can be used to write a new script

Note:

*There must be an empty file named “**__init__.py**” in each package (subfolder) so that “Narda Script Launcher” can recognize the scripts it contains.*

4.1.3 Module and Package Names

For the naming of modules and packages, special rules (naming conventions) must be observed.

- Modules should have short, all-lowercase names.
- Underscores can be used in the module name if it improves readability.
- Packages should also have short, all-lowercase names, although the use of underscores is discouraged.

4.1.4 Creating a New Package

To add a new package to a project, it is not sufficient to create a corresponding folder in the Windows Explorer directory manually! Instead, it is recommended to create a new package according to the following steps:

1.) In the project tree perform a right mouse click on project name, here: “UserScriptDev”

2.) Select “New”

3.) Select “Python Package”

4.) Define a suitable package name according to section 4.1.3.

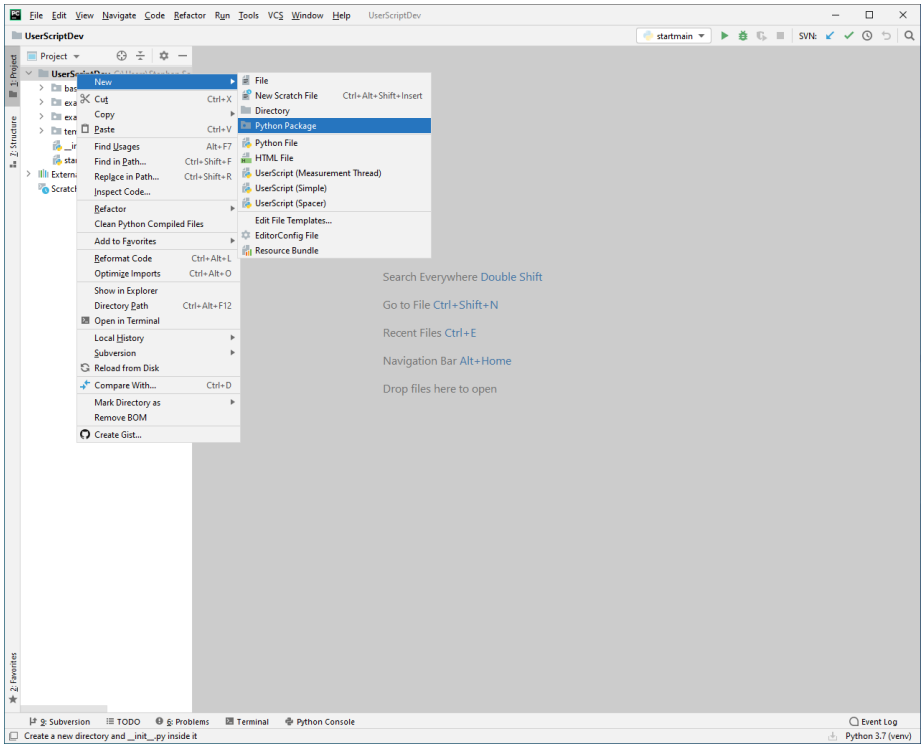


Fig. 12. PyCharm Creating a new package

4.1.5 Adding Packages by Copy & Paste

Additional ready-to-use scripts are offered on the Narda website from time to time. These are contained in zip-compressed folders (packages) and can be added to the project by unzipping them into the project directory.

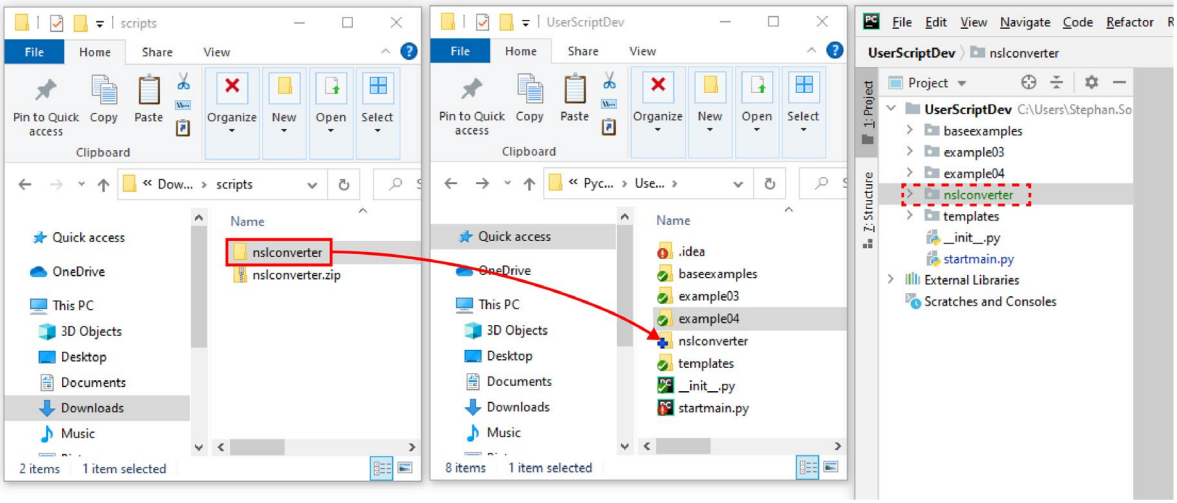


Fig. 13. Unzip package to PyCharm project directory.

4.2 Script Templates

A script is a Python file containing a class derived from “UsrScriptBase”. Thus, it can be recognized, displayed and executed by the “Narda Script Launcher”. It has the file extension “.py”.

It is recommended to create a new script based on one of the available template files as shown below:

1.) In the project tree, right click on the desired package folder (or create a new one)

2.) Click on “New”

3.) Choose the template that best suits your needs.

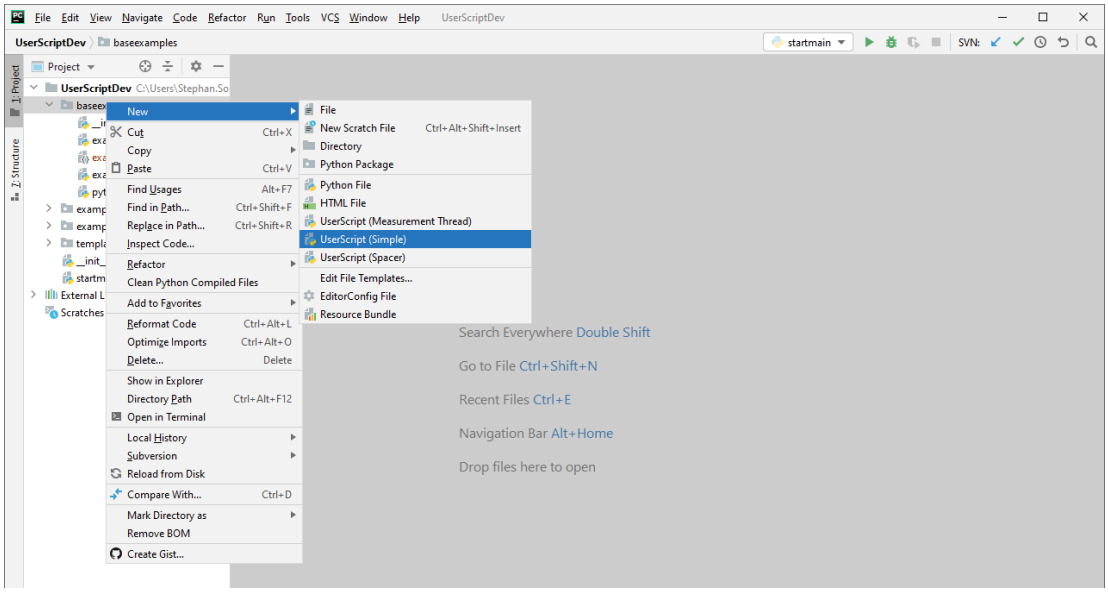


Fig. 14. New script via template

4.) Enter a valid file name, e.g. “myscript” according to section 4.1.3 and confirm with OK.

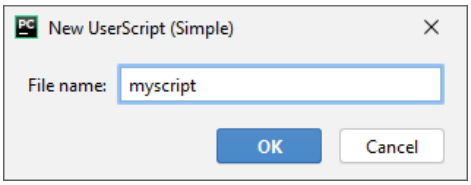


Fig. 15. Adequate renaming of the script file

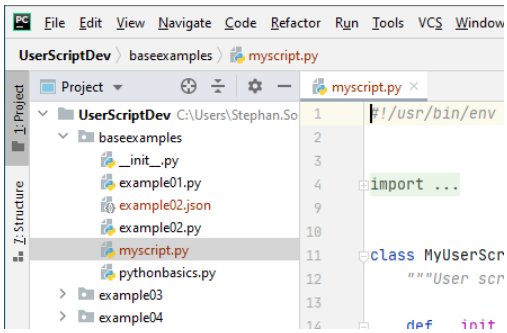


Fig. 16. New script based on a template

A detailed description of the individual templates can be found in the “Narda Script Launcher” Manual in the "Script templates" section of the chapter “Cheat Sheet (Copy & Paste)”.

5 Execute or Debug a Script Using PyCharm

This chapter describes how scripts can be debugged, tested and executed remotely from a computer.

It is assumed that PyCharm and “Narda Script Launcher” have been installed to a computer and that a SignalShark unit is connected to the computer via TCP network connection.

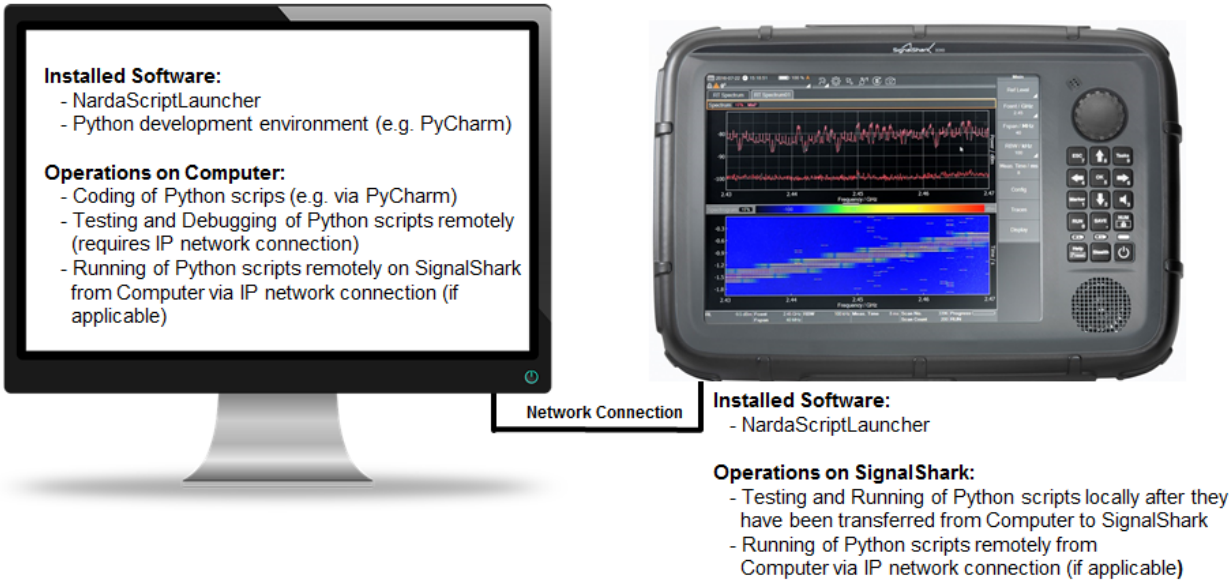


Fig. 17. “Narda Script Launcher” and PyCharm installation options for computer and SignalShark

5.1 Connecting Signalshark to a Computer

There are several ways to connect SignalShark to a computer.

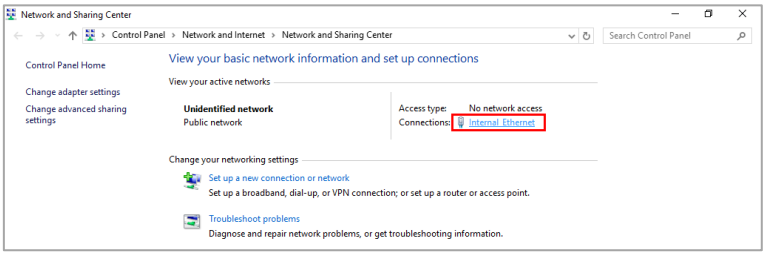
The easiest way is to connect the network port of the SignalShark directly to the network card of the computer using an Ethernet cable. An Ethernet crossover cable is no longer necessary with today's network cards. Since the IP address of the SignalShark is needed later, it is recommended to set a static IP address for SignalShark and the computer. The necessary steps are described in the following chapters.

Another possibility is to connect SignalShark and the computer via a router. The IP addresses are automatically assigned by the router via DHCP. If you install a USB Wi-Fi adapter on the SignalShark, the connection can even be wireless. Because the IP address of the SignalShark is needed later, you should configure the router to always assign the same IP address. Please refer to the manual of your router for details.

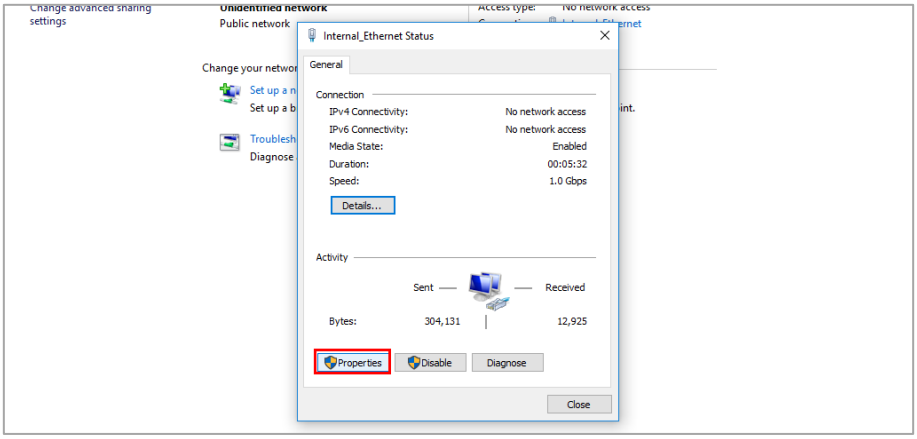
5.1.1 Network Settings on the PC for a Static IP Address.

1.) Right-click on the network icon in the lower right corner of your Windows screen and select “Open Network and Sharing Center”

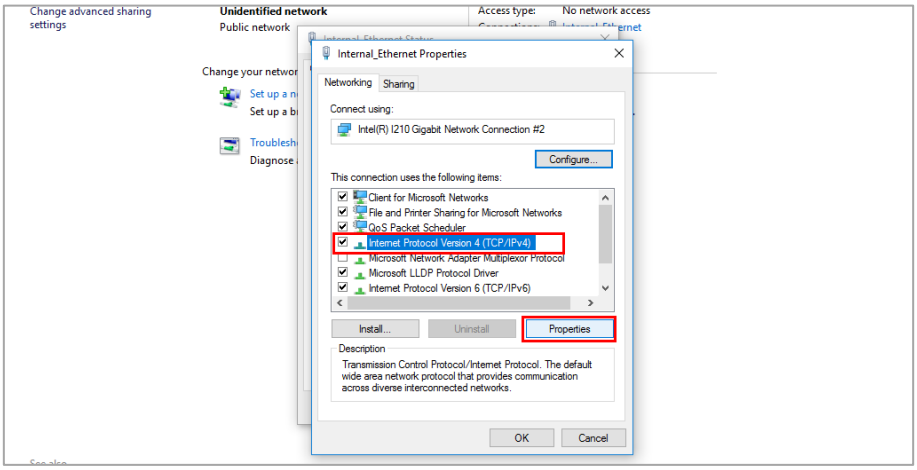
2.) Click on the network adapter to change its settings.



3.) Click on the “**Properties**” button



4.) Select the list item “**Internet Protocol Version 4 (TCP/IPv4)**” and click on the button “**Properties**”

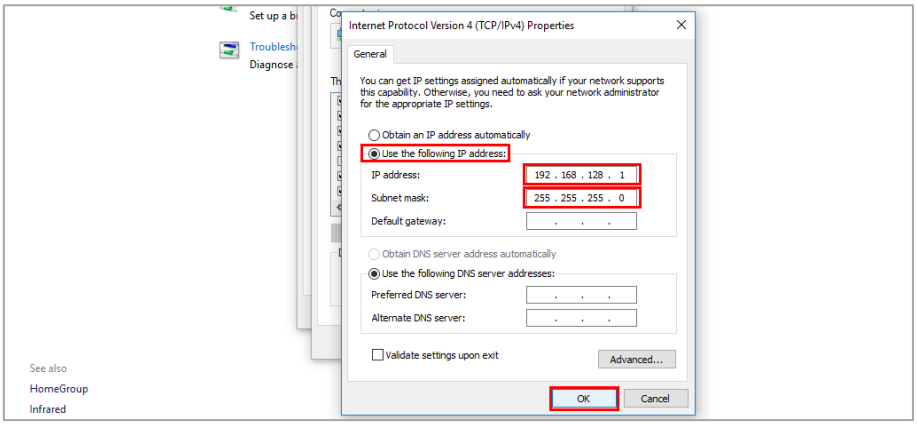


5.) Select “**Use the following IP address**”

6.) Enter a suitable static IP address for the PC (e.g. “**192.168.128.1**”)

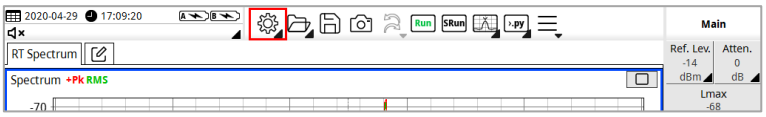
7.) Enter a valid subnet mask (e.g. “**255.255.255.0**”)

8.) Confirm with **OK**

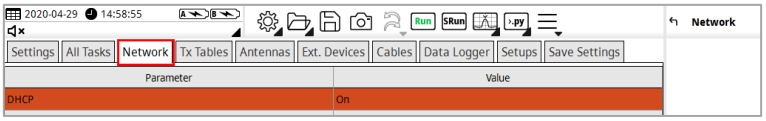


5.1.2 Network Settings on the SignalShark for a Static IP Address

1.) Click on the settings icon in the SignalShark toolbar



2.) Select the "**Network**" tab



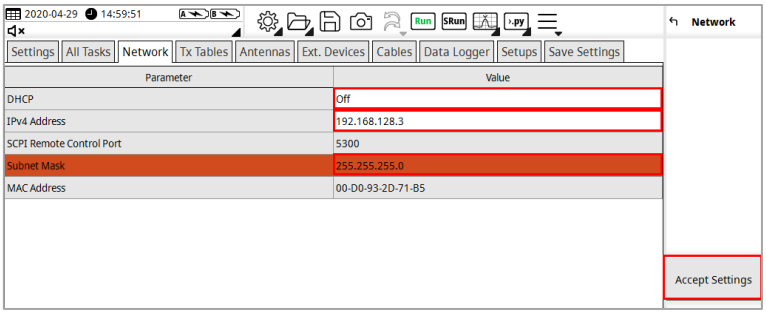
3.) Set "**DHCP**" **off**

4.) Enter a suitable static IP address for the SignalShark (e.g. "**192.168.128.3**")

5.) Enter a valid subnet mask (e.g. "**255.255.255.0**")

6.) Confirm with "**Accept Settings**"

7.) Enter the administrator password to confirm (default is "**changeme**")



5.2 Adapting the Scripting IP Address

1.) Open PyCharm

2.) Create or open a PyCharm project according to the previous chapters.

3.) Open the file "**startmain.py**" by a double click on the filename in the project tree

4.) Change the IP address of the file to the IP address of your connected SignalShark

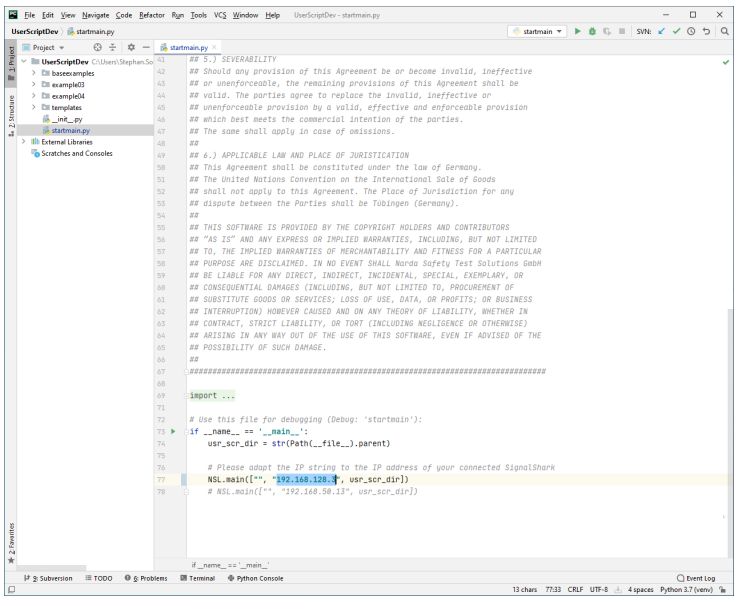


Fig. 18. Setup IP address in PyCharm for accessing SignalShark remotely

5.3 Run / Debug

To run “Narda Script Launcher” or to start it in debug mode:

1.) Right mouse click on "**startmain.py**"

2.) Select “**Run ‘startmain’**” or "**Debug 'startmain'**”

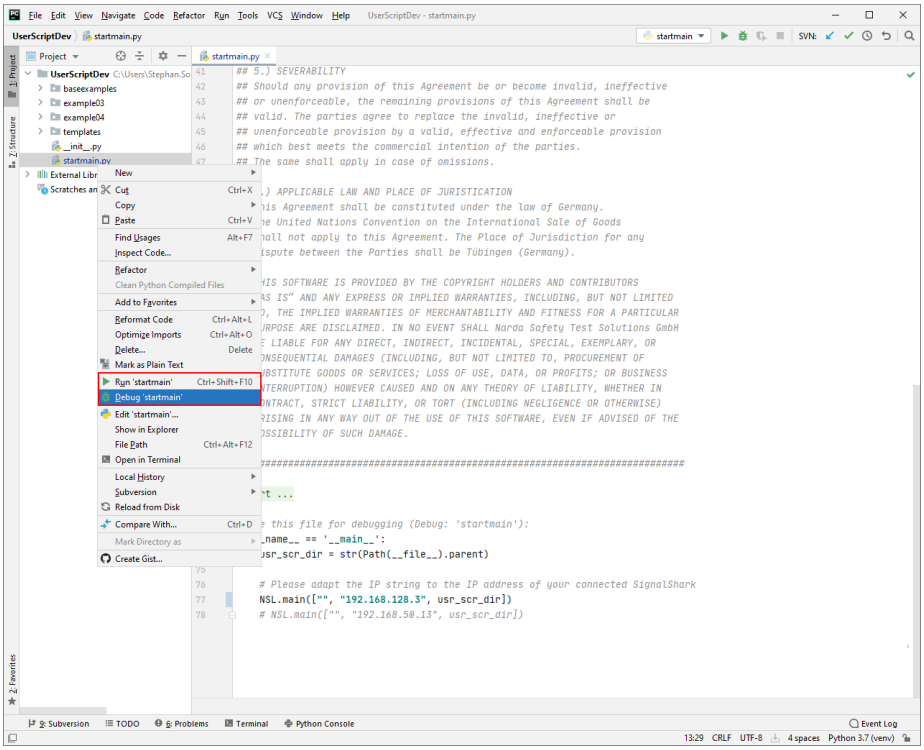


Fig. 19. Run or Debug ´startmain´

Shortly after, the "Narda Script Launcher" will open and any script present in the currently opened project on your computer can be started by clicking the corresponding button.

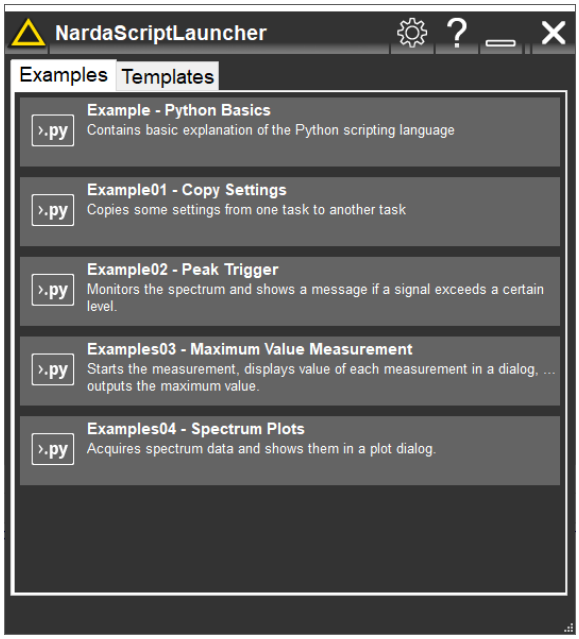


Fig. 20. “Narda Script Launcher” after start via PyCharm

5.4 Setting a “Breakpoint”

You can set a breakpoint for a specific line of code by clicking with the mouse pointer to the right of the corresponding line number. In debug mode, the execution of the program is stopped at this point.
If you hover the mouse pointer over a variable in this situation, you can display the contents of the variable.
With the menu item “Run→Step Over” or by pressing the F8-key, you can execute the code line by line to test it.

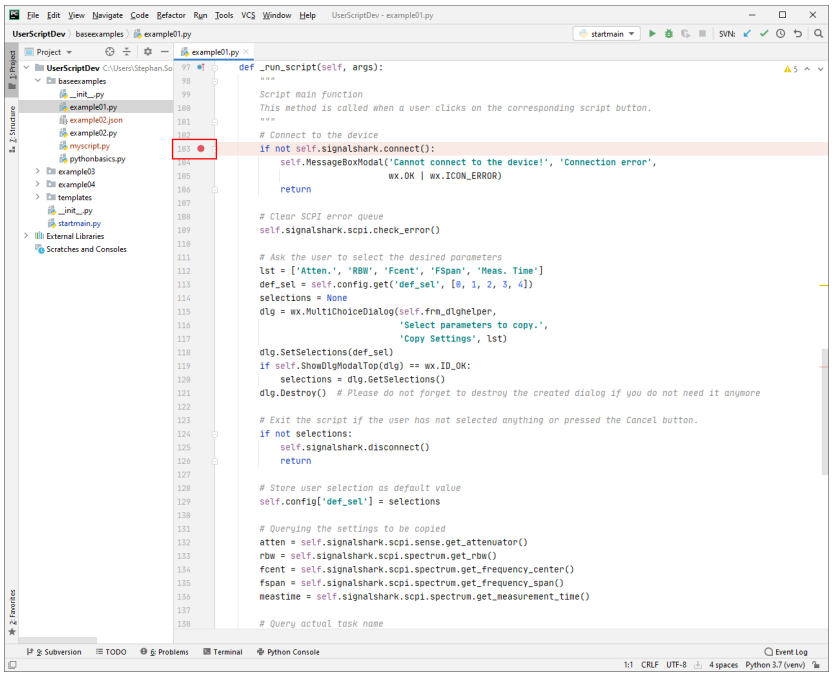


Fig. 21. PyCharm, adding a breakpoint.

5.5 Code Completion

PyCharm offers the function “Code Completion” that automatically makes suggestions for the completion of a text input. This is especially helpful when using SCPI functions in a script.

1.) Type in “`self.signalshark.scpi.`” in your script code.

Once you have entered the last dot, a list of possible SCPI command groups is displayed. The same applies to the available SCPI commands.

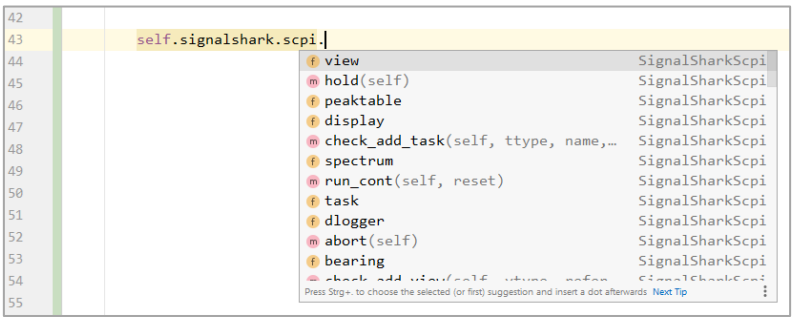


Fig. 22. Example of Code completion

6 Appendix

6.1 Disclaimer

THE OPEN SOURCE SOFTWARE IS PROVIDED “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL Narda Safety Test Solutions GmbH BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THE RESPECTIVE LICENSE TERMS MAY PROVIDE FURTHER INFORMATION.

NARDA PROVIDES LINKS TO OFFERS FROM THIRD PARTIES IN THIS DOCUMENTATION. THE PROVIDERS OF THESE PAGES ARE EXCLUSIVELY RESPONSIBLE FOR THE CONTENTS AND IN PARTICULAR ANY DAMAGES THAT MAY ARISE FROM THE USE OR NON-USE OF THE INFORMATION PROVIDED IN THIS WAY. NARDA ONLY CHECKS THE PAGES AT THE TIME THE LINK IS CREATED. ALL SUBSEQUENT CHANGES ARE THE RESPONSIBILITY OF THE PROVIDER.

Narda Safety Test Solutions GmbH does not assume any warranty and liability for the use of 3rd party products (PyCharm, mathplotlib, PySerial …).

6.2 Python Naming Conventions

Source: https://visualgit.readthedocs.io/en/latest/pages/naming_convention.html

6.2.1 General

- Avoid using names that are too general or too wordy. Strike a good balance between the two.
  - Bad: data_structure, my_list, info_map, dictionary_for_the_purpose_of_storing_data_representing_word_definitions
  - Good: user_profile, menu_options, word_definitions
- Don’t use ambiguous letters/numbers such as “O”, “I”, or “l”
- When using CamelCase names, capitalize all letters of an abbreviation (e.g. HTTPServer)

6.2.2 Packages

- Package names should be all lower case
- When multiple words are needed, an underscore should separate them
- It is usually preferable to stick to 1 word names

6.2.3 Modules

- Module names should be all lower case
- When multiple words are needed, an underscore should separate them
- It is usually preferable to stick to 1 word names

6.2.4 Classes

- Class names should follow the UpperCaseCamelCase convention
- Python’s built-in classes, however are typically lowercase words
- Exception classes should end in “Error”

6.2.5 Global (Module-level) Variables

- Global variables should be all lowercase
- Words in a global variable name should be separated by an underscore

6.2.6 Instance Variables

- Instance variable names should be all lower case
- Words in an instance variable name should be separated by an underscore
- Non-public instance variables should begin with a single underscore
- If an instance name needs to be mangled, two underscores may begin its name

6.2.7 Methods

- Method names should be all lower case
- Words in an method name should be separated by an underscore
- Non-public method should begin with a single underscore
- If a method name needs to be mangled, two underscores may begin its name

6.2.8 Method Arguments

- Instance methods should have their first argument named ‘self’.
- Class methods should have their first argument named ‘cls’

6.2.9 Functions

- Function names should be all lower case
- Words in a function name should be separated by an underscore

6.2.10 Constants

- Constant names must be fully capitalized
- Words in a constant name should be separated by an underscore.

Narda Safety Test Solutions GmbH
Sandwiesenstrasse 7
72793 Pfullingen, Germany
Phone +49 7121 97 32 0
info.narda-de@L3Harris.com

L3Harris Narda STS
North America Representative Office
435 Moreland Road
Hauppauge, NY11788, USA
Phone +1 631 231 1700
NardaSTS@L3Harris.com

Narda Safety Test Solutions GmbH
Beijing Representative Office
Xiyuan Hotel, No. 1 Sanlihe Road, Haidian
100044 Beijing, China
Phone +86 10 6830 5870
support@narda-sts.cn

www.narda-sts.com