# Open Optical Networks

## Debug in Pycharm, Project Structure, pathlib

ANDREA D'AMICO – andrea.damico@polito.it

GIACOMO BORRACCINI – giacomo.borraccini@polito.it

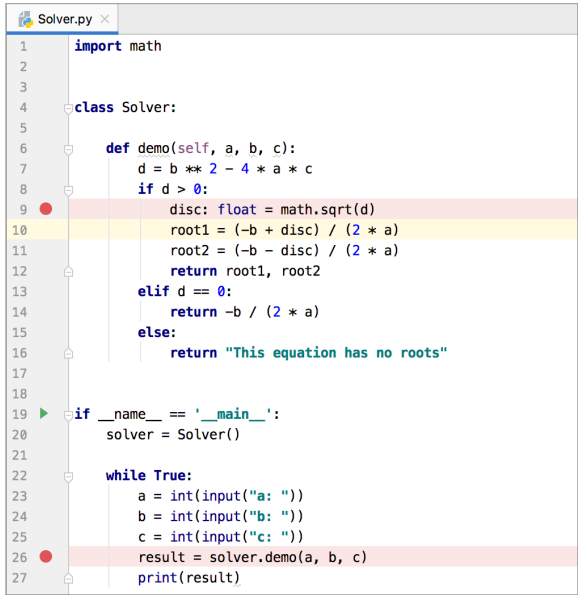
# DEBUG BASICS IN PYCHARM

# DEBUG BASICS IN PYCHARM

PLANET
Physical Layer Aware Networks

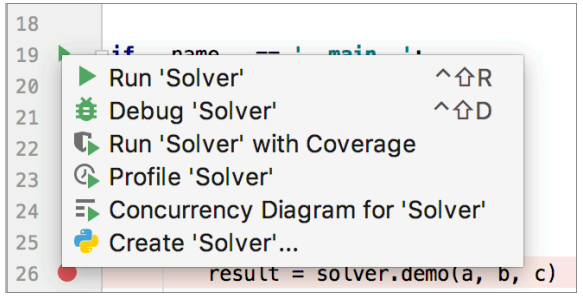
## Placing breakpoints

- Breakpoints are special markers that suspend program execution at a specific point. This lets you examine the program state and behavior.
- To place breakpoints, just click the gutter next to the line you want your application to suspend at.
- Once set, a breakpoint remains in your project until you remove it explicitly,




## Starting the debugger session

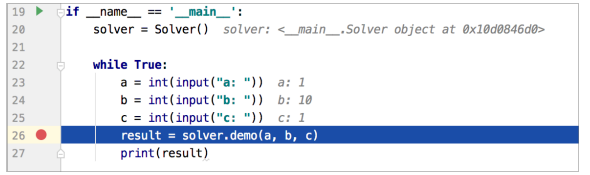
- PyCharm allows stating the debug session in several ways. For example, click the play button in the gutter and then select the command **Debug 'your-function'** in the popup menu.



# DEBUG BASICS IN PYCHARM

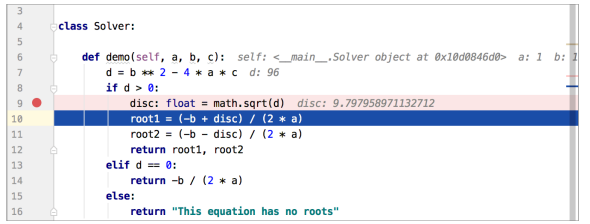
## Debug Session

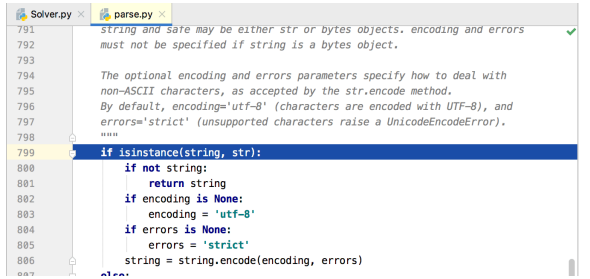
- Then the debugger suspends the program at the first breakpoint. It means that the line with the breakpoint is not yet executed. The line becomes blue.
- On the stepping toolbar of the debugger tab, click the button to move to the next breakpoint.




## Stepping

- If you use the stepping toolbar debugger, you'll move to the next line. For example, click the button (*Step over*)
- If you click the button (*Step into*) you proceed the debug into the current function.



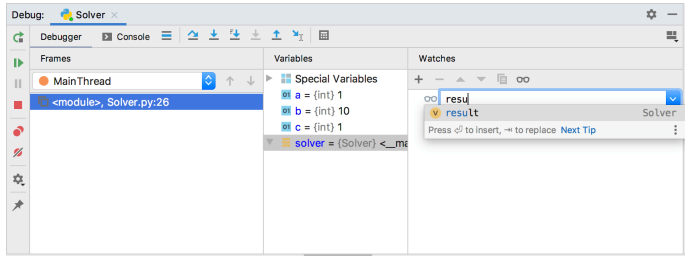



# DEBUG BASICS IN PYCHARM

PLANET
Physical Layer Aware Networks

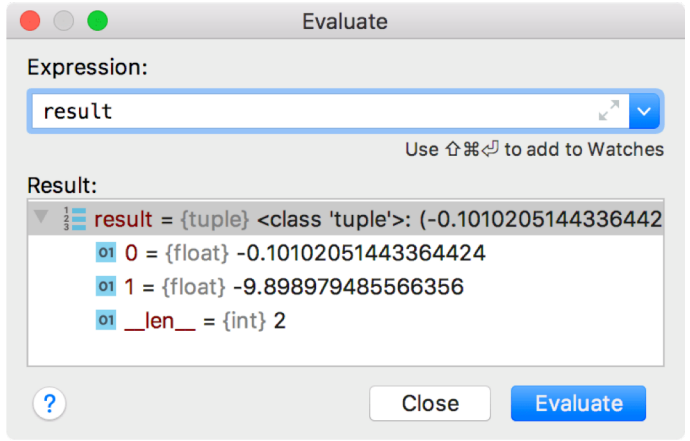
## Watching

- When the program execution continues to the scope that defines the variable.




## Evaluating expressions

- Finally, you can evaluate any expression at any time. For example, if you want to see the value of the variable, click the button  and then in the dialog that opens, click **Evaluate.** PyCharm gives you the possibility to evaluate any expression.




## Reference

- https://www.jetbrains.com/help/pycharm/part-1-debugging-python-code.html#start-debugger-session

# PROJECT STRUCTURE

# PYTHON PACKAGES

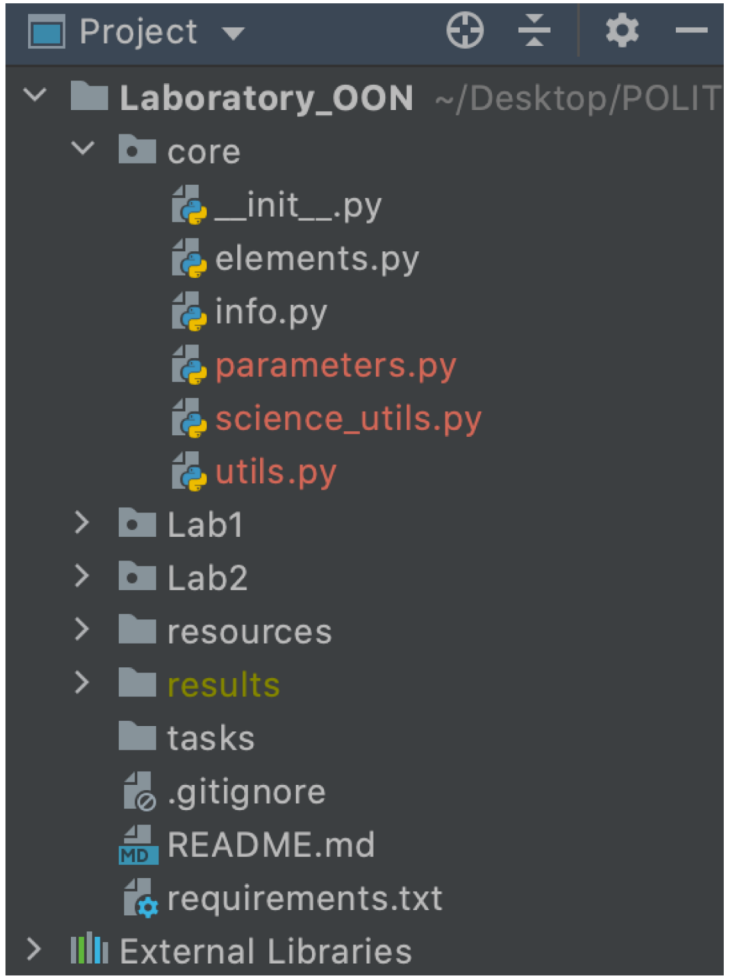
PLANET
Physical Layer Aware Networks

- **Modules** in Python are simply Python files with a .py extension. The name of the module will be the name of the file. A Python module can have a set of functions, classes or variables defined and implemented.
- Python defines two types of packages, *regular packages* and *namespace packages*.
- A **regular package** is typically implemented as a directory containing an *__init__.py* file. When a regular package is imported, this *__init__.py* file is implicitly executed, and the objects it defines are bound to names in the package's namespace. The *__init__.py* file can contain the same Python code that any other module can contain, and Python will add some additional attributes to the module when it is imported.
- A **namespace package** is a composite of various portions, where each portion contributes a sub-package to the parent package. Portions may reside in different locations on the file system. Portions may also be found in zip files, on the network, or anywhere else that Python searches during import. Namespace packages may or may not correspond directly to objects on the file system; they may be virtual modules that have no concrete representation.

# PROJECT STRUCTURE

PLANET
Physical Layer Aware Networks

- The *Project Structure* is a fundamental aspect which deeply influences the efficiency and the usability of your code.
- There are so many solutions for implementing smartly a git project.
- In this slide, we want to suggest you a possible project structure which could particularly effective for our purpose.



# PROJECT STRUCTURE

PLANET
Physical Layer Aware Networks

- Folders:
  - ***Core***: package that contains all the engine modules. All the computational effort is described within this package.
  - ***Resources***: directory that contains all the input files.
  - ***Results***: directory that contains all the output files.
  - ***Tasks***: directory that contains all the main python scripts. These internal scripts call functions and objects contained in core and they are not called by other scripts.
- Core modules:
  - ***elements***: it describes all the objects that are elements of the scenario to emulate.
  - ***parameters***: it contains objects that contain only attributes (no methods, no functions)
  - ***science_utils***: it is a sort of function library or it contains objects with only methods. This module contains all the expressions with physical meaning.
  - ***utils***: it contains functions for general general use (plotting, saving, loading…)

# PATHLIB

# PATHLIB

PLANET
Physical Layer Aware Networks

- **pathlib** module offers classes representing filesystem paths with semantics appropriate for different operating systems. It instantiates a concrete path for the platform the code is running on.

  Documentation: https://docs.python.org/3/library/pathlib.html

- `from pathlib import Path`
- When a module is loaded from a file in Python, *__file__* is set to its path. You can then use that with other functions to find the directory that the file is located in.
- `root  = Path(__file__).parent`
- *.parent*: the directory containing the file, or the parent directory if path is a directory. It can be called more than one time (*path.parent.parent*)
- `folder = root / 'folder_name'`
- `file = folder / 'file_name'`