

Python Debug in VSCode

Son Nguyen

Why IDE?

(Integrated Development Environment)

- **Consolidate different aspects of writing a computer program**

- Color code editor, tendency to help you write better code

- Docstring
- Code completion
- Parameter suggestion
- Auto format code

- Terminal/command line

- Quickly test function in a python environment

- Debugger

- Symbolic debugger, inspect at runtime

- Version control

- Embedded git

- Remote SSH

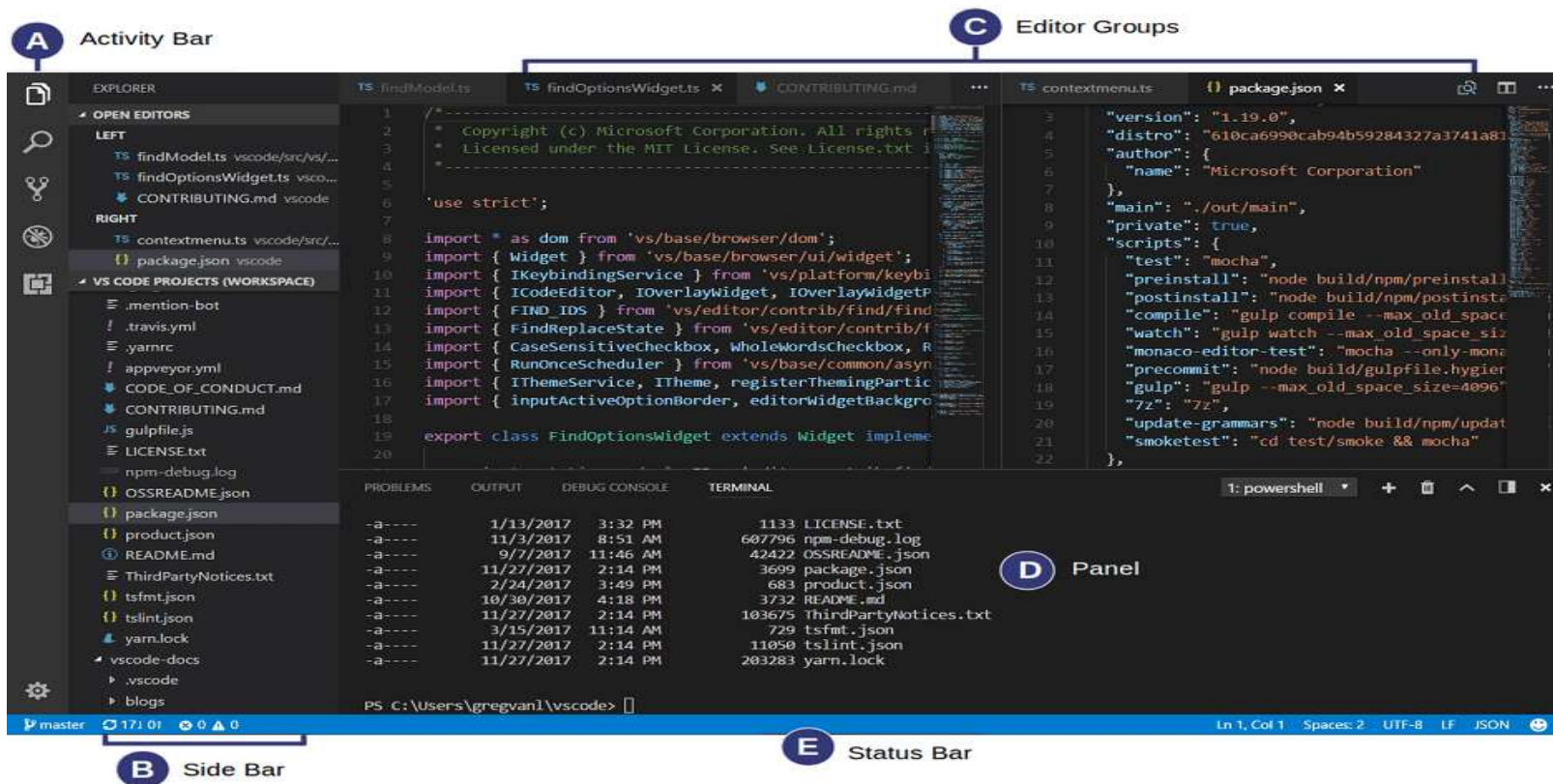
- Write code in local machine, run code in server

Why VSCode?

- **Many options: VSCode, Spyder, PyCharm**
- **Why VSCode**
 - Lightning fast, open source code editor
 - Rich extensions support: code completion, docstring, Docker, jupyter, GitLens ...
 - Support multiple programming languages – over 100 programming languages, editing/debugging the same across languages.
 - Nice looking 😊

VSCode interface

Typical VSCode interface (largely customizable)



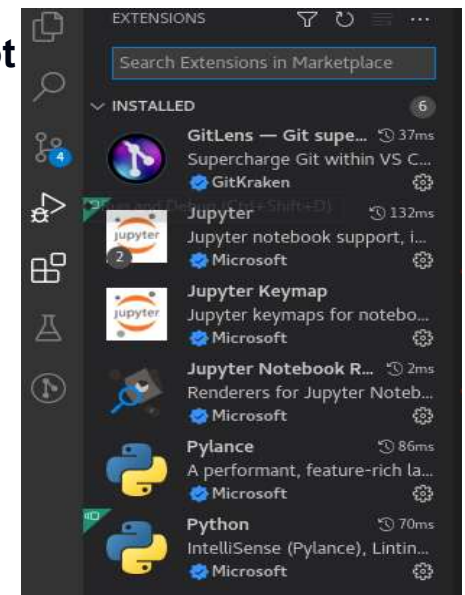
<https://code.visualstudio.com/docs/getstarted/userinterface>

How to load VSCode in BioHPC

VSCode was installed as a module in BioHPC

Source activateVAEGANEnv.sh
module load vscode
code .

- First thing to do is install Python extension.
- Useful extensions: Jupyter, GitLens, PyLance, Remote – SSH, Github Copilot
- Choose the course anaconda environment as interpreter from status bar
Bottom right of VSCode should look like below



Stack trace

- Series of error messages printed out when the code fail to run, often starts with “Traceback”
Traceback (most recent call last):
File "test.py", line 4, in <module>
y = np.matmul(x, W)
ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, with gufunc signature (n?,k),(k,m?)->(n?,m?) (size 3 is different from 4)
- Error message often has information about which file/line cause the problem, what is the problem
- Only helpful when the bug causes error, does not work with “silent bugs”.


Symbolic debugger

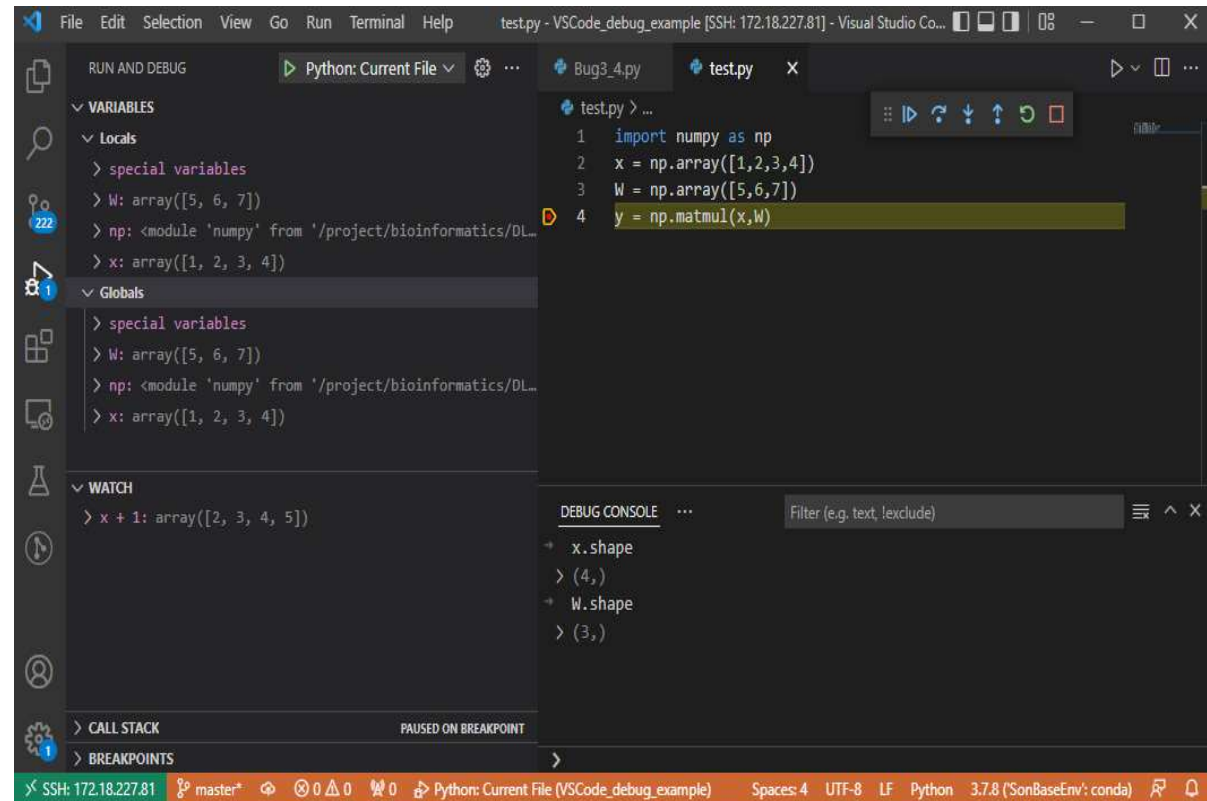
- Let users to inspect directly a running program.
- Stop at breakpoints why keeping all objects/variables of current scope.

A short video about debugging Python code in VSCode

<https://www.youtube.com/watch?v=w8QHoVam1-I>

VSCode debugger

- Start debugger: Run → Start Debugging
- Debugger will stop at Exceptions or Breakpoints
 - If there is no Exception/Breakpoint, the code will run normally
- Useful panels: 
 - Step panels
 - Continue, step over, step into, step out, restart, stop
 - VARIABLES: Locals and Globals variables
 - WATCH: add object/function you want to monitor
 - DEBUG CONSOLE: where you can interact with all variables in VARIABLES



VSCode debugger

■ 1. Auto catch Exception

- VSCode can auto catch (stop at) most of the exceptions before it happens



The screenshot shows a VS Code editor window with a Python file named `test.py`. The code contains the following lines:

```
1 import numpy as np
2 x = np.array([1,2,3,4])
3 W = np.array([5,6,7])
4 y = np.matmul(x,W)
```

Line 4 is highlighted, and a yellow bug icon is visible in the left margin. Below the code editor, a red error message is displayed:

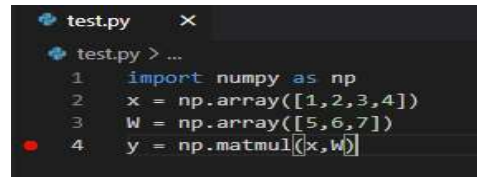
Exception has occurred: ValueError ×
matmul: Input operand 1 has a mismatch in its core dimension 0, with gufunc signature (n?,k),(k,m?)->(n?,m?) (size 3 is different from 4)
File "/archive/bioinformatics/DLLab/src/SonNguyen/VSCode_debug_example/test.py", line 4, in <module>
y = np.matmul(x,W)

- When there is no exception (or there are exceptions that VSCode misses), we have to place breakpoints to tell VSCode to stop.

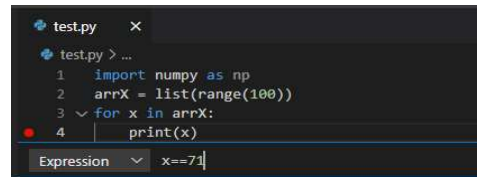
VSCode debugger

■ 2. Breakpoints

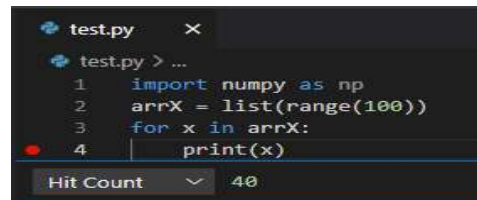
- Breakpoints are used to tell the debugger where to stop. There are *modified breakpoints* that are also useful.
- Conditional break points - Expression: only stop when a condition happens. Useful when the bug happens only on a few cases in a big loop.
- Hit Count: stop when the given line of code was about to be executed n^{th} times ($n=40$ in the example).
- Log message: does not stop, but log a message to debug console instead, similar to `print()` function debugging style, but you do not have to change the code to see the message.



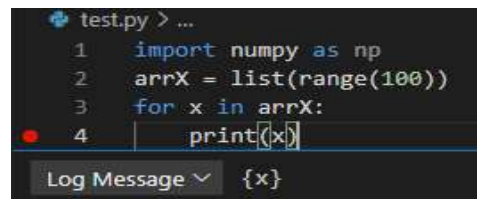
```
test.py  X
test.py > ...
1  import numpy as np
2  x = np.array([1,2,3,4])
3  W = np.array([5,6,7])
4  y = np.matmul(x,W)
```



```
test.py  X
test.py > ...
1  import numpy as np
2  arrX = list(range(100))
3  for x in arrX:
4  | print(x)
Expression  x==71
```



```
test.py  X
test.py > ...
1  import numpy as np
2  arrX = list(range(100))
3  for x in arrX:
4  | print(x)
Hit Count  40
```



```
test.py > ...
1  import numpy as np
2  arrX = list(range(100))
3  for x in arrX:
4  | print(x)
Log Message {x}
```

Excercise

- **Now is time for bug hunting and fixing.**
- **There are 4 files with 5 bugs**
 - The first 3 (bug0.py, Bug1.py, Bug2.py), each file has a single bug. They are all independent
 - Bug3_4.py has 2 bugs in it, please fix the first then the second bug.
 - All bugs have *hints*, hope they helps
 - Solutions will be posted Friday night.



Questions?
