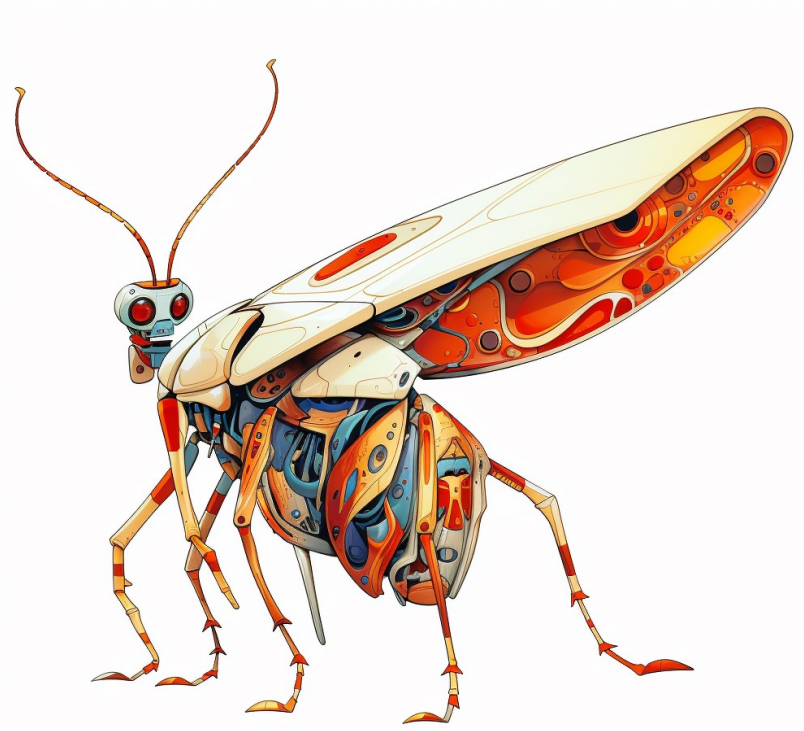
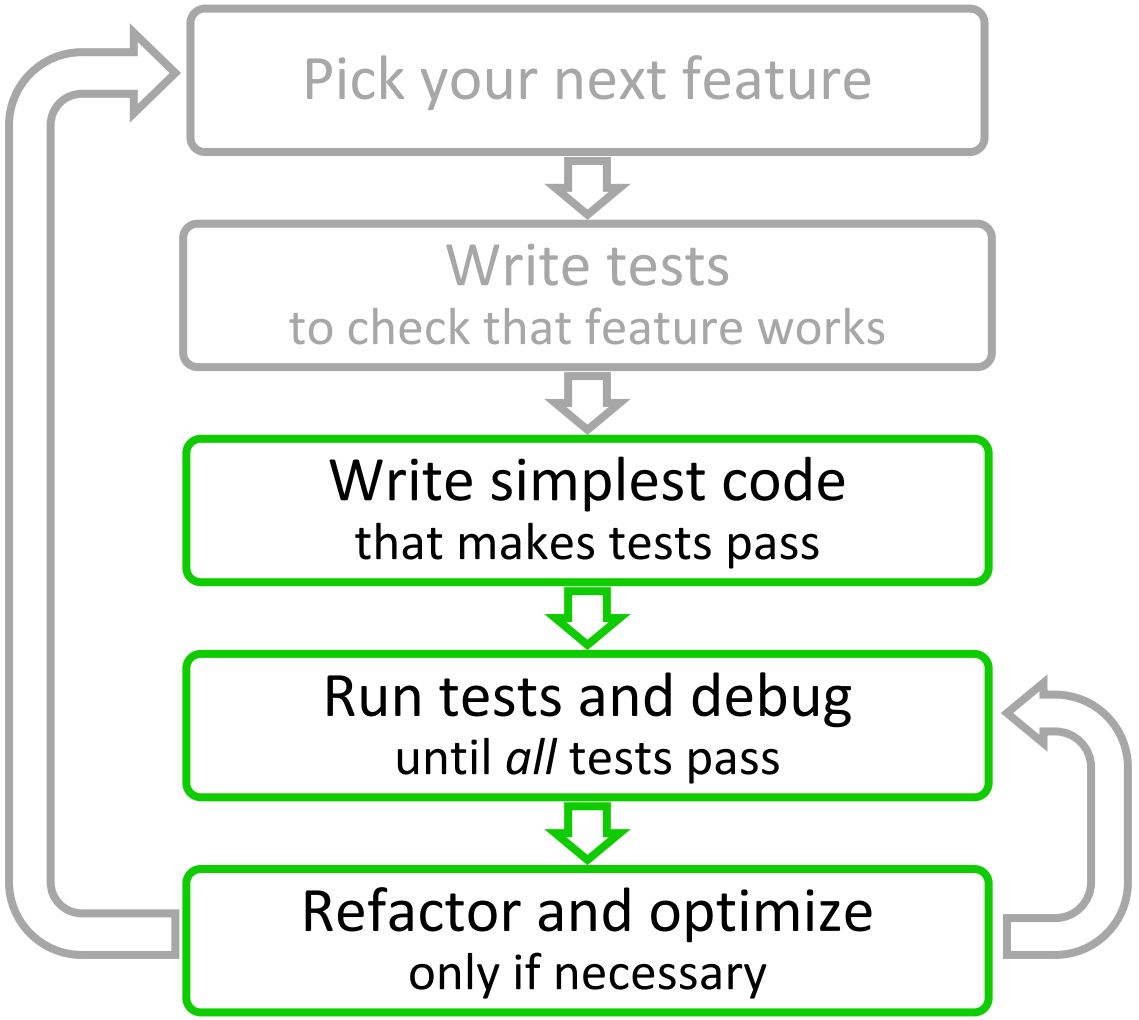
Debugging

Fix problems



Lisa Schwetlick and Pamela Hathway

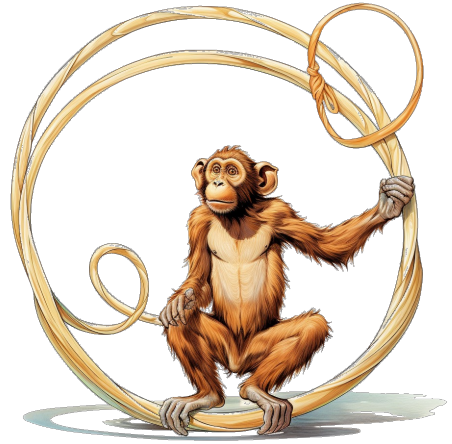
The agile development cycle



`pytest`

pdb

`timeit`
`cProfile`
`line_profiler`



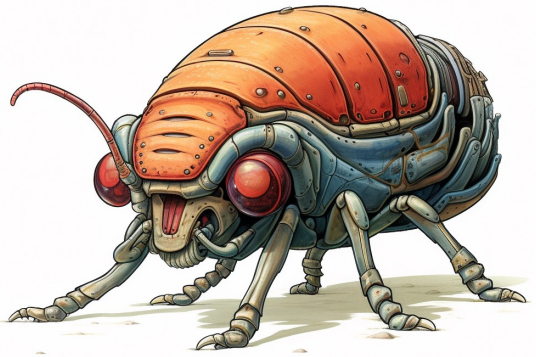
What is Debugging?

- The best way to debug is to avoid bugs
  - By writing tests, you *anticipate* the bugs
- Your test cases should already exclude a big portion of the possible causes

- Core idea in debugging: you can stop the execution of your application at any point, look at the state of the variables, and execute the code step by step

pdb, the Python debugger

- Command-line based debugger
- pdb opens an interactive shell, in which one can interact with the code
  - examine and change value of variables
  - execute code line by line
  - set up breakpoints
  - examine calls stack

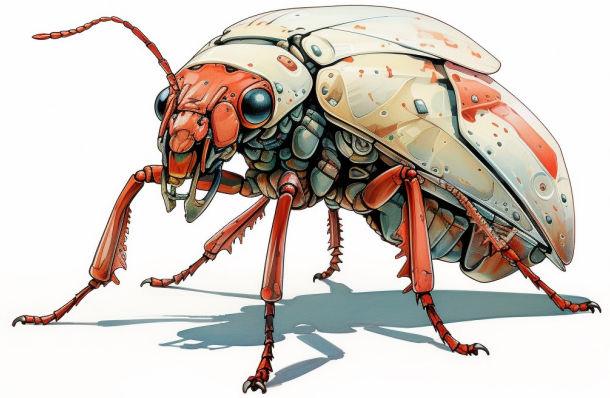


Entering the debugger

- Enter debugger at the start of a file:
  `python -m pdb myscript.py`

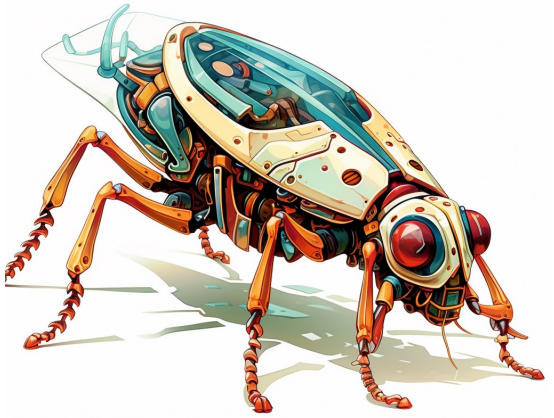
- Enter at a specific point in the code (easy alternative to `print`):

```
# some code here
# the debugger starts here
breakpoint()
# rest of the code
```



Demonstration!

- Debugging with pdb



Hands-on! – Issue #2

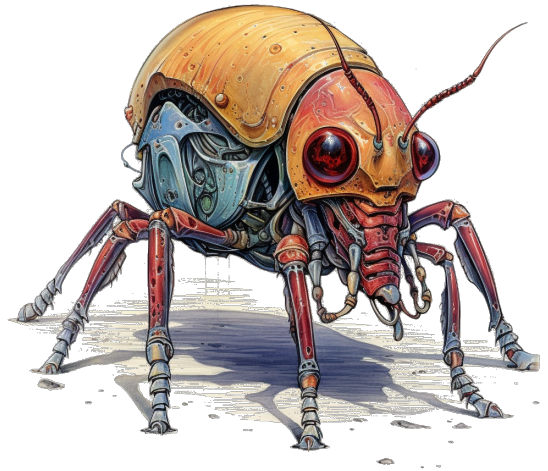
1) Go to `local_maxima_part3_debug/`**`local_maxima.py`** and execute it

2) Go to `local_maxima_part3_debug/`**`test_local_maxima.py`** and execute all the tests there using pytest

3) Fix the tests one by one and create a PR for each one you fix

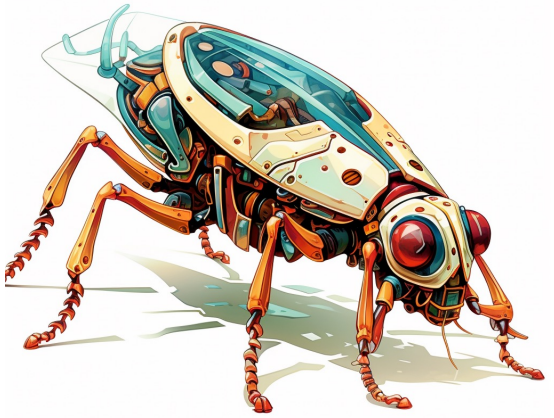
pdb cheatsheet:

h (help) [*command*]	print help about *command*
n (next)	execute current line of code, go to next line
c (continue)	continue executing the program until next breakpoint, exception, or end of the program
s (step into)	execute current line of code; if a function is called, follow execution inside the function
l (list)	print code around the current line
w (where)	show a trace of the function call that led to the current line
p (print)	print the value of a variable
q (quit)	leave the debugger
b (break) [*lineno* | *function*[, *condition*]]	set a breakpoint at a given line number or function, stop execution there if *condition* is fulfilled
cl (clear)	clear a breakpoint
! (execute)	execute a python command
<enter>	repeat last command

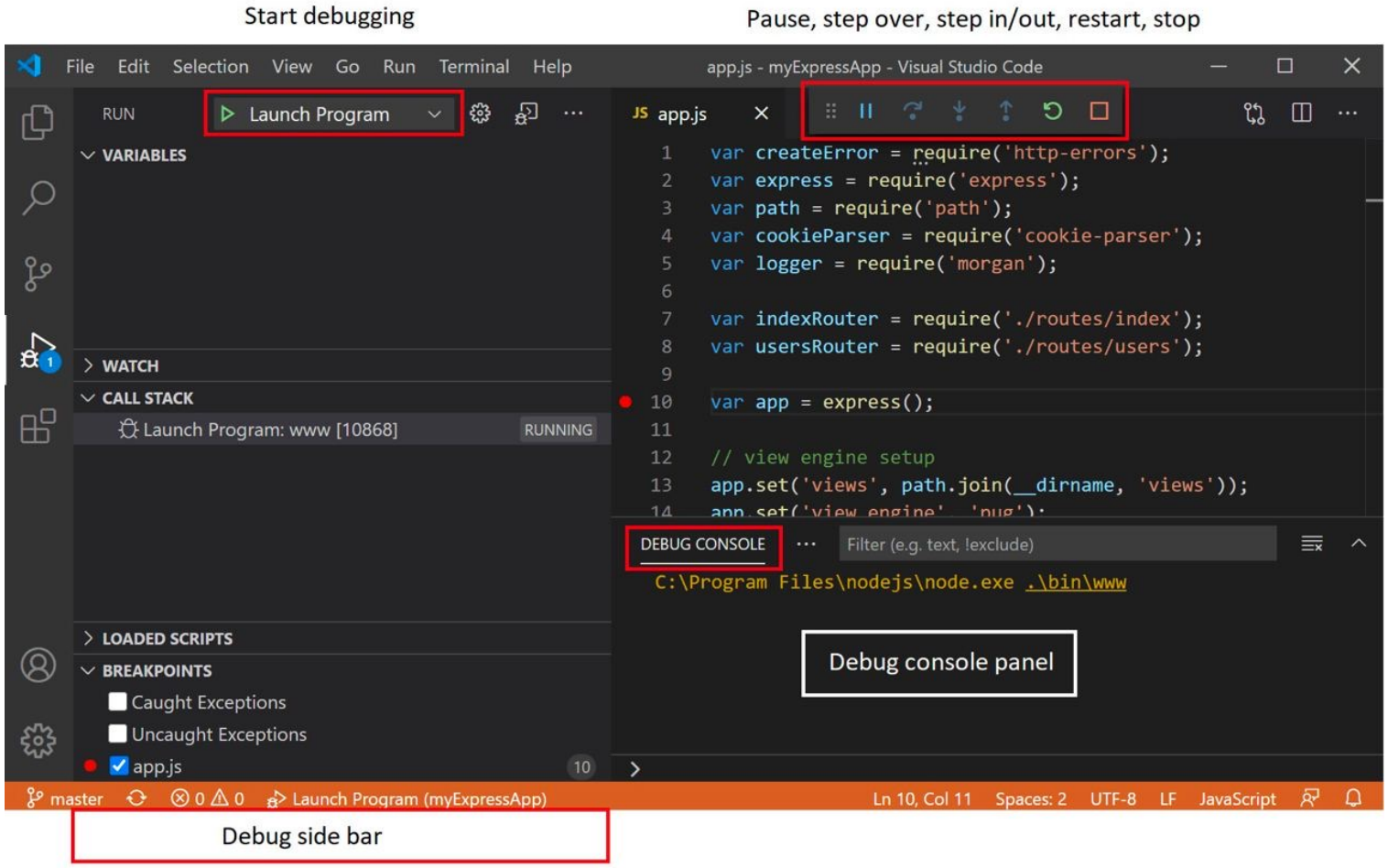


Demonstration!

- Debugging with pdb
- Debugging from an IDE



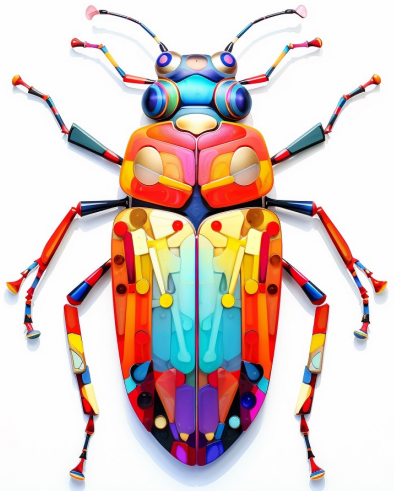
Entering the debugger from VSCode



How to react to a bug

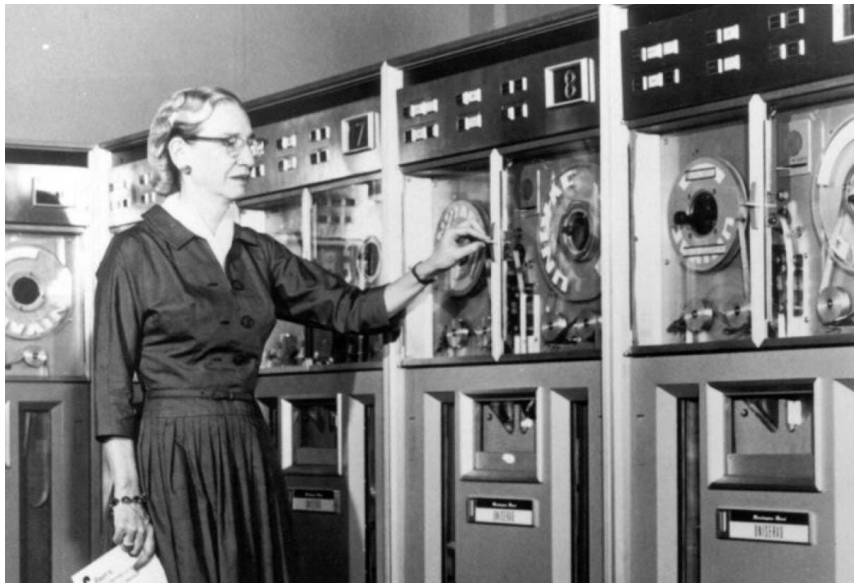
1. Add a test that matches the behavior you expect. It will fail and reproduce the bug
2. Debug and fix the the bug
3. Run the tests until they all pass (go back to 2 if necessary)

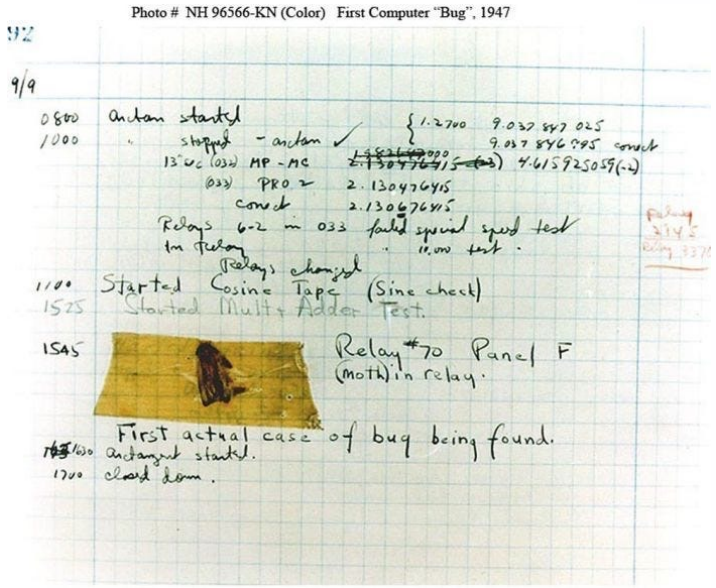
- Now your bug is fixed *and* it will never occur again!



Where does ”debug” come from

Team around Grace Hopper at Harvard found a moth in their computer in 1947 in maybe the first description of a computer bug.





Set up debugging in IDEs

- Generally IDEs can do very similar things.

- But…
- VSCode/VSCodium is *really* cumbersome to set up (my opinion).
- Pycharm is a lot less hassle to set up (no setup needed) and better to use and with better functionality (my opinion). caveat: I use the paid version, so I am only 95% confident all features are available in free version

- … However at this school we only have access to VSCodium -.-

Set up debugging in VSCode

- Debugging configurations for Python apps in Visual Studio Code (links to code.visualstudio.com/docs/python/debugging)

You need to do these steps

1. **install Python Extension (automatically installs Python Debugging extension**

Python Debugger Extension

The Python Debugger extension is automatically installed along with the Python extension for VS Code. It offers debugging features with debugpy for several types of Python applications, including scripts, web apps, remote processes and more.

To verify it's installed, open the **Extensions** view (⇧⌘X) and search for `@installed python debugger`. You should see the Python Debugger extension listed in the results.

Set up debugging in VSCode

- [Debugging configurations for Python apps in Visual Studio Code](code.visualstudio.com/docs/python/debugging) (links to code.visualstudio.com/docs/python/debugging)

You need to do these steps

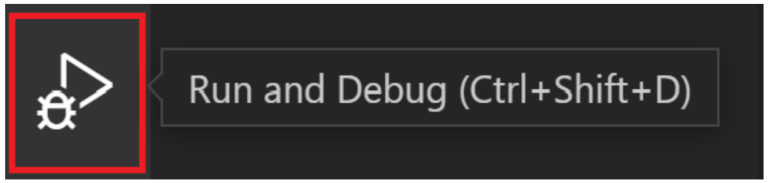
1. install Python Extension (automatically installs Python Debugging extension
2. **In order to call the debugger on the current file, you need to create a launch.json file**

Initialize configurations

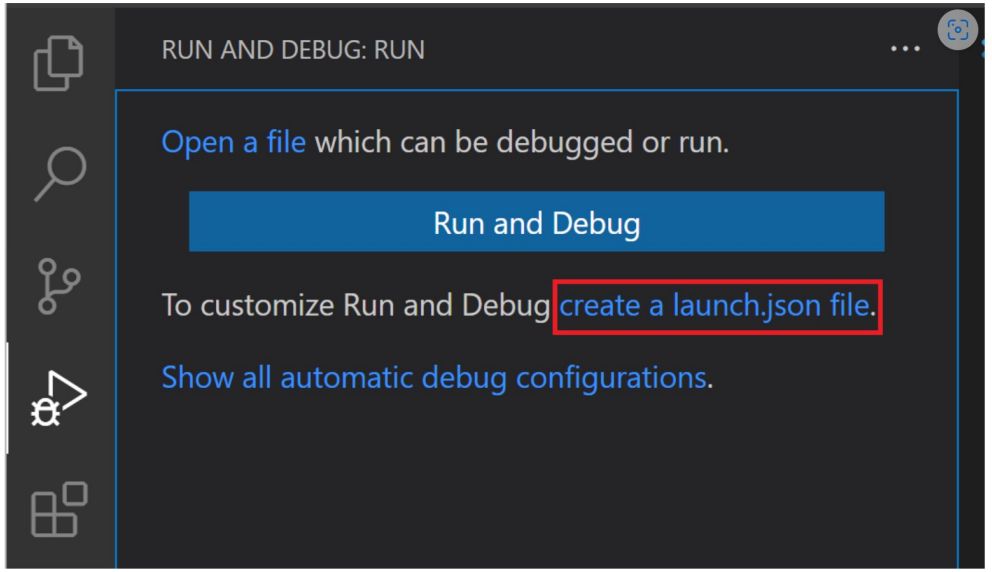
A configuration drives VS Code's behavior during a debugging session. Configurations are defined in a `launch.json` file that's stored in a `.vscode` folder in your workspace.

> **Note**: To change debugging configuration, your code must be stored in a folder.

To initialize debug configurations, first select the **Run** view in the sidebar:



If you don't yet have any configurations defined, you'll see a button to **Run and Debug** and a link to create a configuration (launch.json) file:



Set up debugging in VSCode

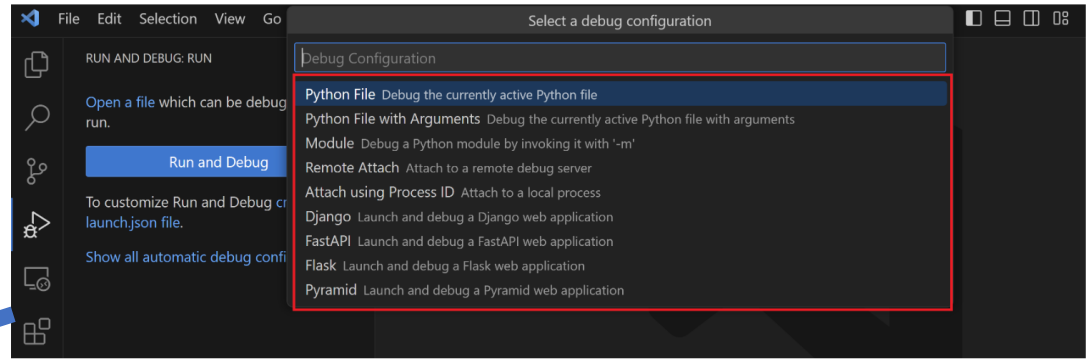
- [Debugging configurations for Python apps in Visual Studio Code](code.visualstudio.com/docs/python/debugging) (links to code.visualstudio.com/docs/python/debugging)

You need to do these steps

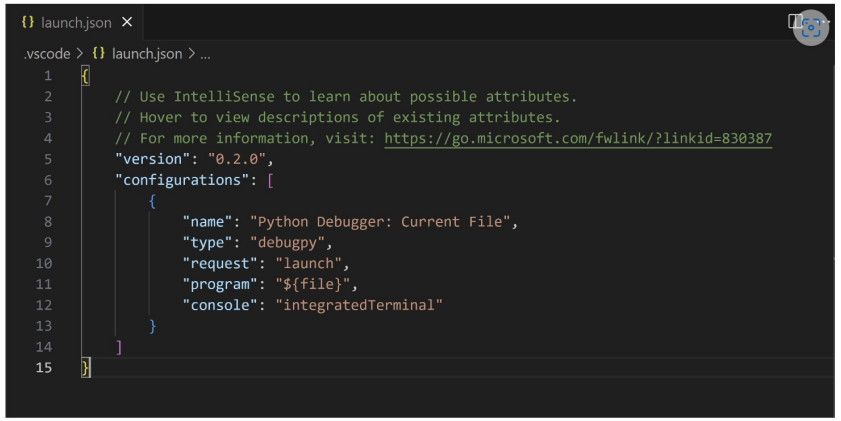
1. install Python Extension (automatically installs Python Debugging extension
2. **In order to call the debugger on the current file, you need to create a launch.json file**

To generate a `launch.json` file with Python configurations, do the following steps:

1. Select the **create a launch.json file** link (outlined in the image above) or use the **Run** > **Open configurations** menu command.
2. Select **Python Debugger** from the debugger options list.
3. A configuration menu will open from the Command Palette allowing you to choose the type of debug configuration you want to use for our Python project file. If you want to debug a single Python script, select **Python File** in the **Select a debug configuration** menu that appears.



1. The Python Debugger extension then creates and opens a `launch.json` file that contains a pre-defined configuration based on what you previously selected, in this case, **Python File**. You can modify configurations (to add arguments, for example), and also add custom configurations.



this file with this content should be created automatically!!

Set up debugging in VSCode

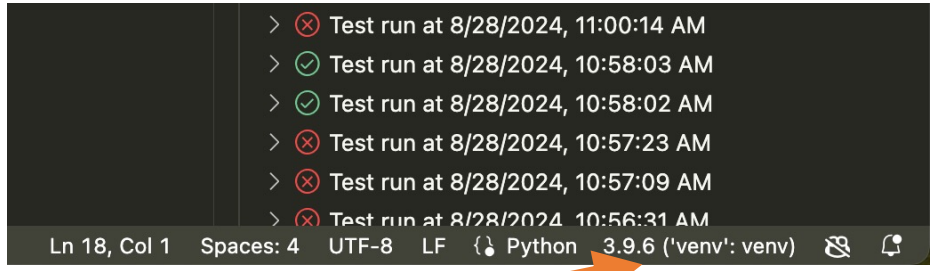
- [Debugging configurations for Python apps in Visual Studio Code](code.visualstudio.com/docs/python/debugging) (links to code.visualstudio.com/docs/python/debugging)

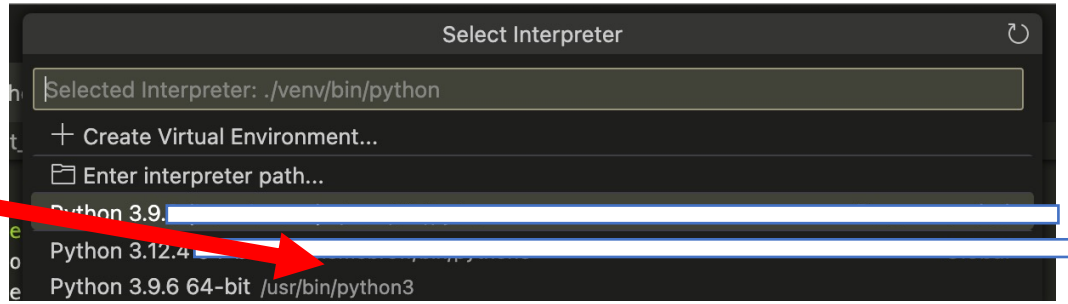
You need to do these steps

1. install Python Extension (automatically installs Python Debugging extension
2. In order to call the debugger on the current file, you need to create a launch.json file
3. **Make sure that the main system Python is selected as the Python Interpreter**

Click on displayed Python version in **bottom right corner**, which will bring up the menu seen here on the right. The paths will be different, make sure you select the Python in **usr/bin/python3**

→ generally, you can specify / create virtual environments here





Set up debugging in VSCode

- accessing EVERYTHING in VSCode
  - Mac: Command + Shift + p
  - Linux: Control + Shift + p (?)

  - then type what you want to do / access e.g.
    - open preferences (type “preferences“)
    - setting interpreter (type „Interpreter“)
    - open Keyboard shortcuts settings (type „Keyboard“)

Set up debugging in VSCode

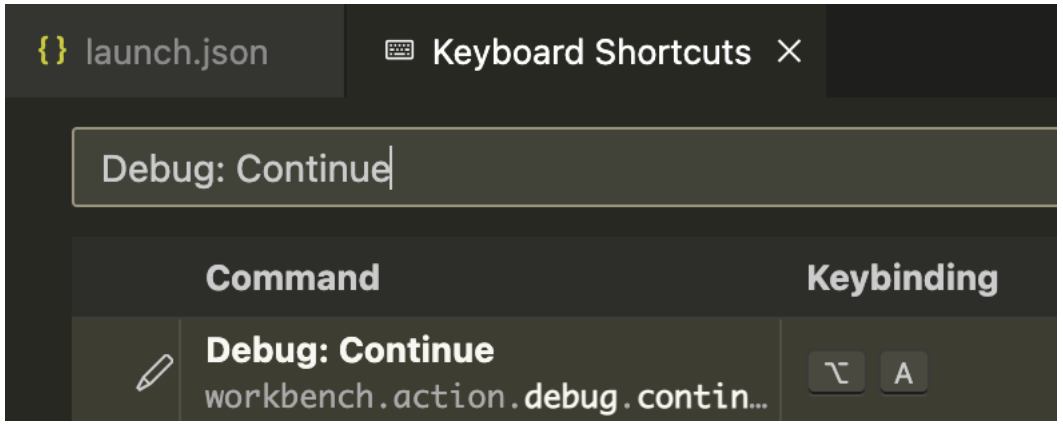
- you are done with setting up!

- next are optional keyboard shortcuts

Set up debugging in VSCode III

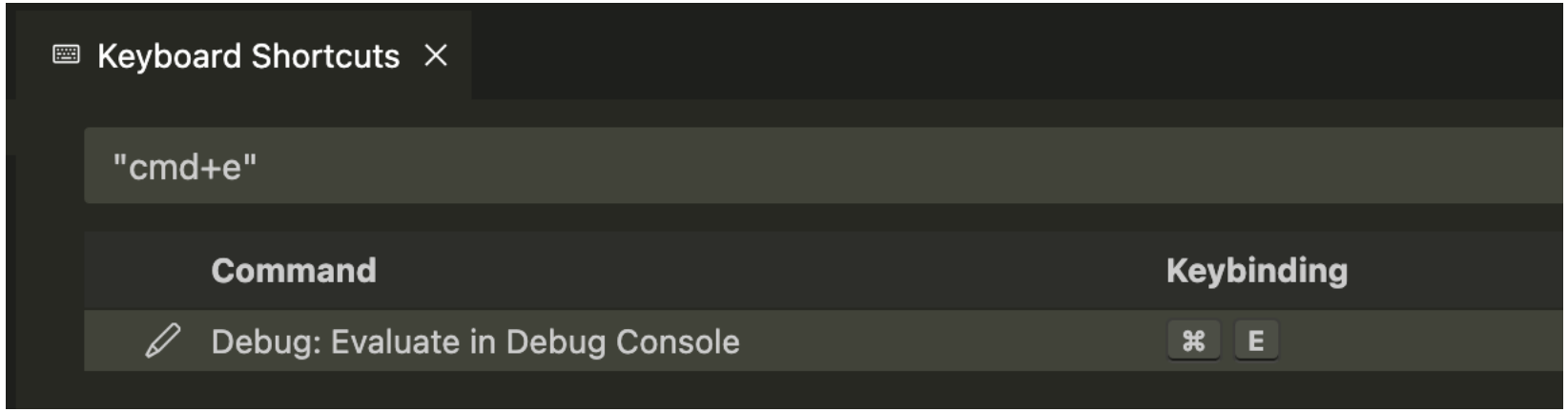
- other settings – controlling debugger with keyboard shortcuts

- you can bind debugging actions to the keyboard shortcuts you want (continue, step/next, step into, restart …)



Set up debugging in VSCode IV

- other settings – executing highlighted code in debug console

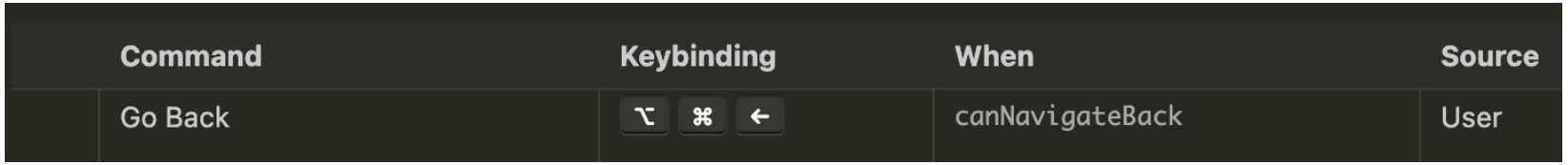


Debugging example

- inspecting state of code
  - variables
  - call stack incl. variables

Other useful IDE things

- go to function definitions / usages by clicking
- hover over functions to see docstrings
- go to last place you were before



Command	Keybinding	When	Source
Go Back	⌥ ⌘ ←	canNavigateBack	User

- Run all tests, debug tests
- Search across all code
- adding import statements support

Set up debugging in VSCode +

- another example launch configuration – with command line arguments (not needed in this school I think)
- add this into configurations list in launch.json

```
{
"name": “<file_name> with data",
"type": "debugpy",
"request": "launch",
"program": "${workspaceFolder}/src/pkg_name/main.py", // replace with the
path to main.py
"console": "integratedTerminal",
"args": ["--input_dir", "data", "--output_file", "output_dir/file.csv"],
},
```

Set up debugging in VSCode +

- another example launch configuration – you can also debug streamlit applications
- add this into configurations list in launch.json

```
{
"name": "main Streamlit",
"type": "debugpy",
"request": "launch",
"program":
"/Users/pamelahathway/Library/Caches/pypoetry/virtualenvs/pkg_name-
E5WlELVw-py3.12/bin/streamlit",
"args": ["run", "${workspaceFolder}/src/pkg_name/main.py"],
"env": {"PYTHONPATH": "${workspaceFolder}"},
"console": "integratedTerminal"
}
```