ENGR 298: Engineering Analysis and Decision Making – Debugger

If you're not using it then you're making life harder…

Dr. Jason Forsyth
Department of Engineering
James Madison University

How code actually is written…

- The **debugger** is a tool that can **halt** and **step** through program execution when a **breakpoint** is hit.

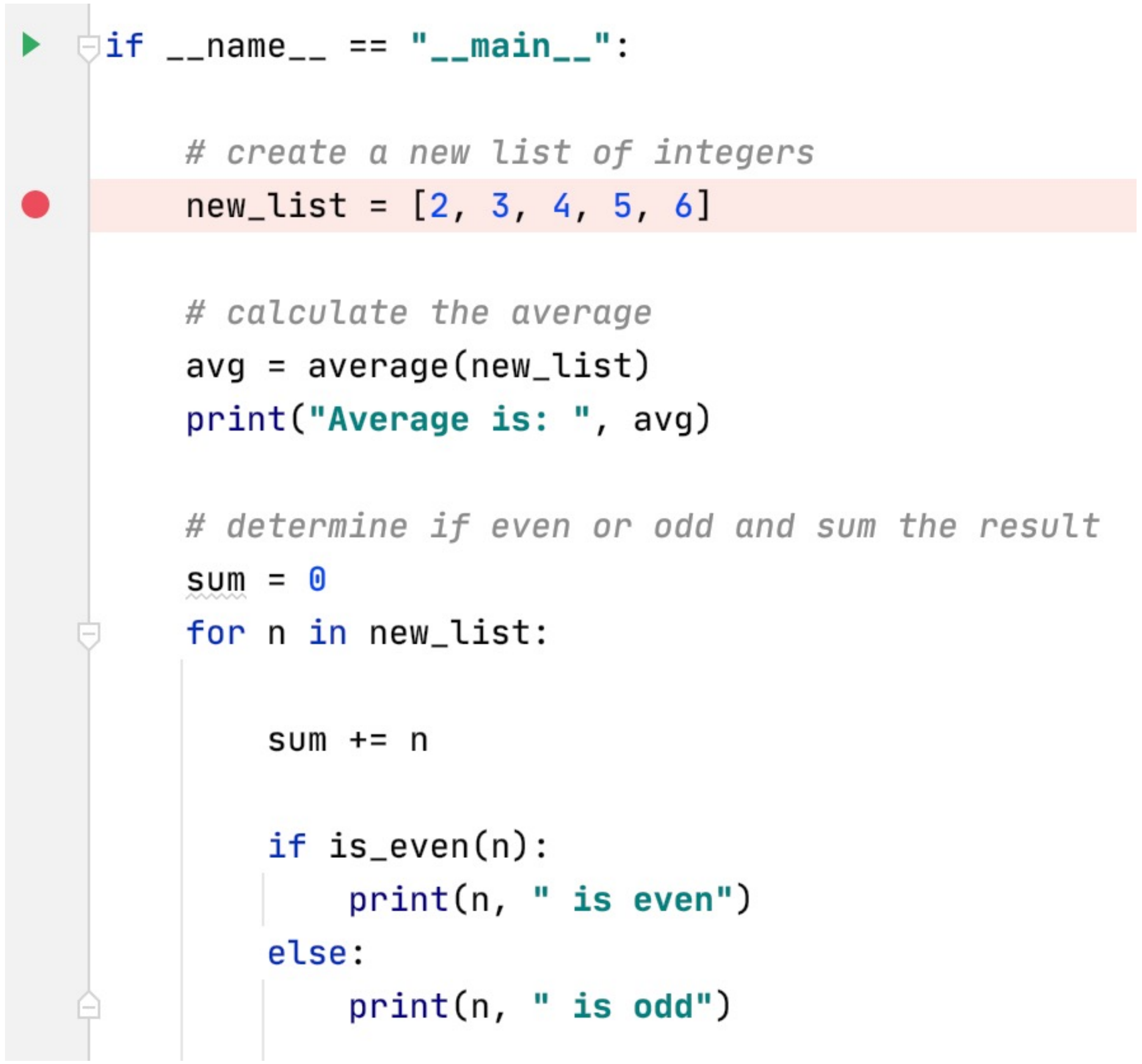
- While debugging, the program will run slower, but all internal values inside the executable can be seen and modified.

- Extremely useful for examining program logic to determine if the executable responds correctly to various inputs

- Program will remain in debug mode until it is **continued** or another breakpoint is hit.

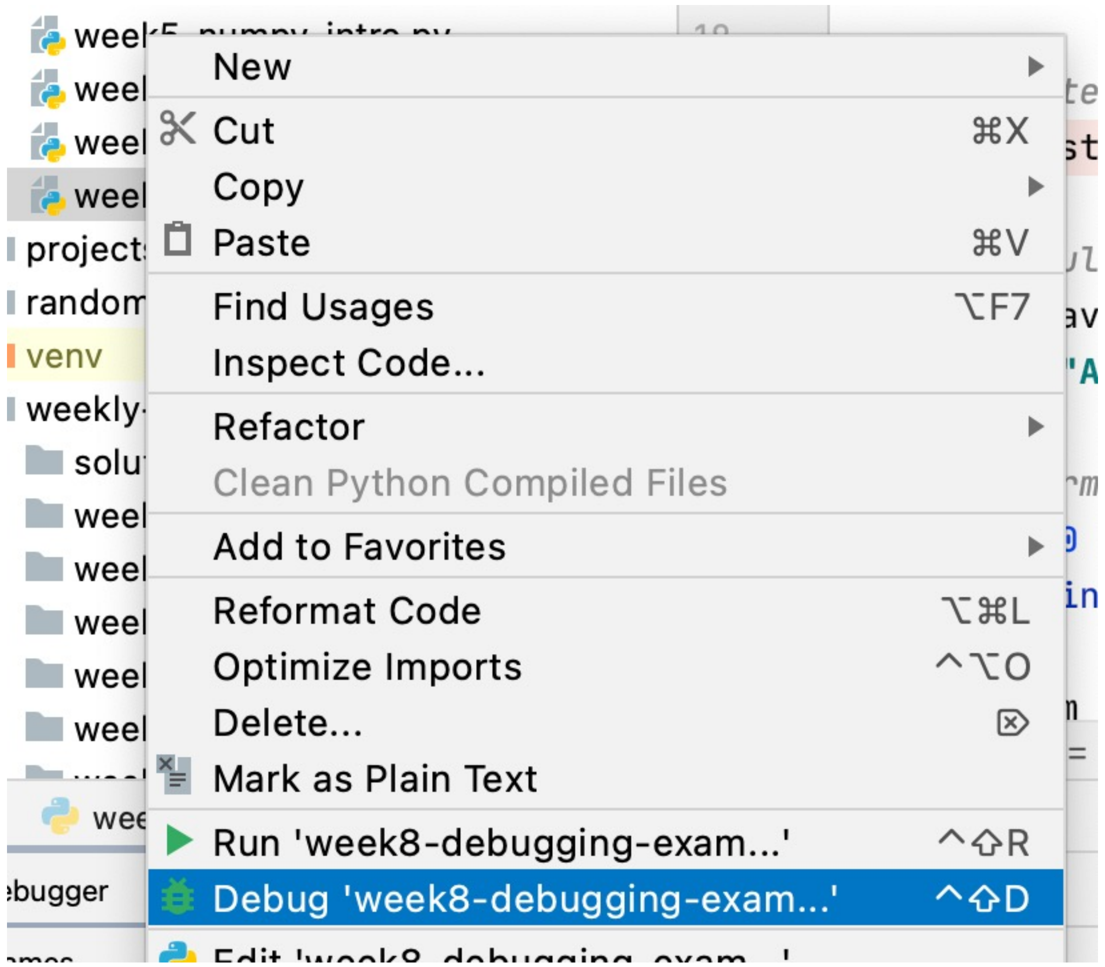
Setting a breakpoint: Click left of the code.

Program will run until break point is hit then will halt. If it is never reached, that tells you something about the program as well.

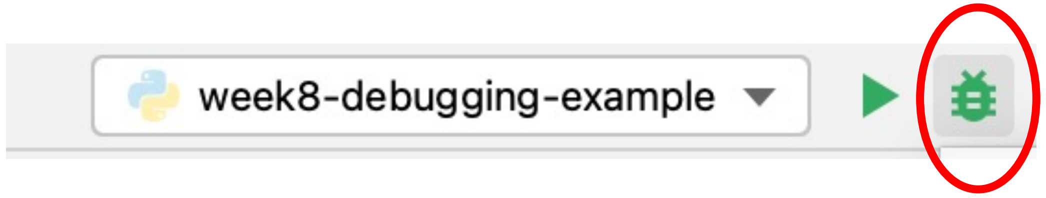


```
if __name__ == "__main__":

    # create a new list of integers
    new_list = [2, 3, 4, 5, 6]

    # calculate the average
    avg = average(new_list)
    print("Average is: ", avg)

    # determine if even or odd and sum the result
    sum = 0
    for n in new_list:

        sum += n

        if is_even(n):
            print(n, " is even")
        else:
            print(n, " is odd")
```

Launching the Debugger…



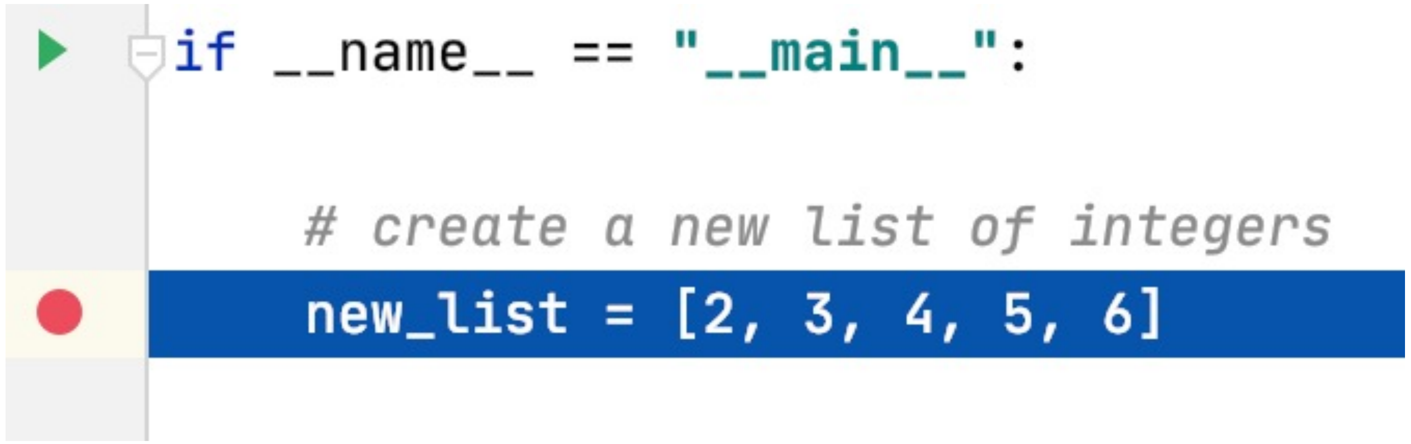
Right click on the file and select Debug



Select from the quick actions toolbar

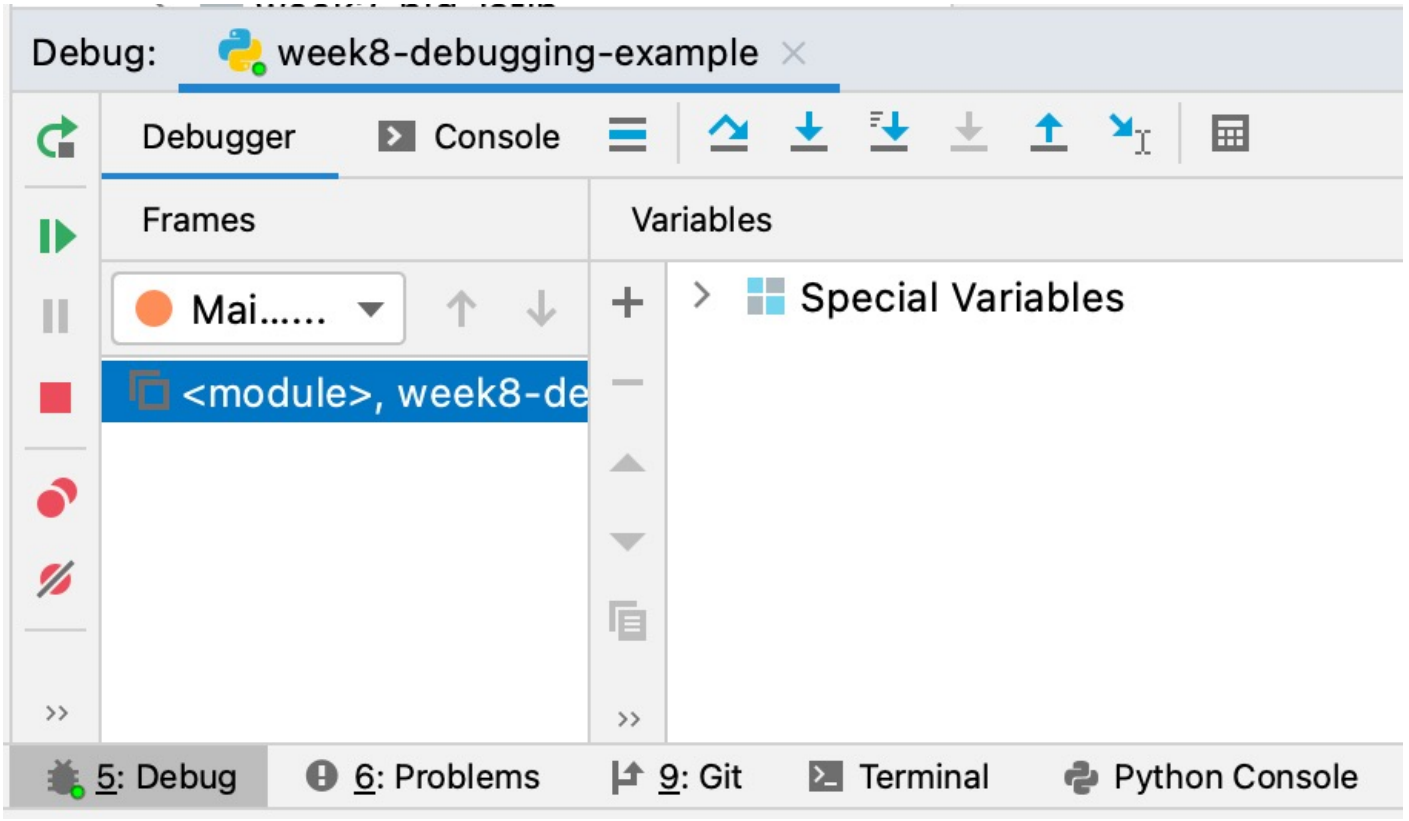
Line is high-lighted when breakpoint is hit.

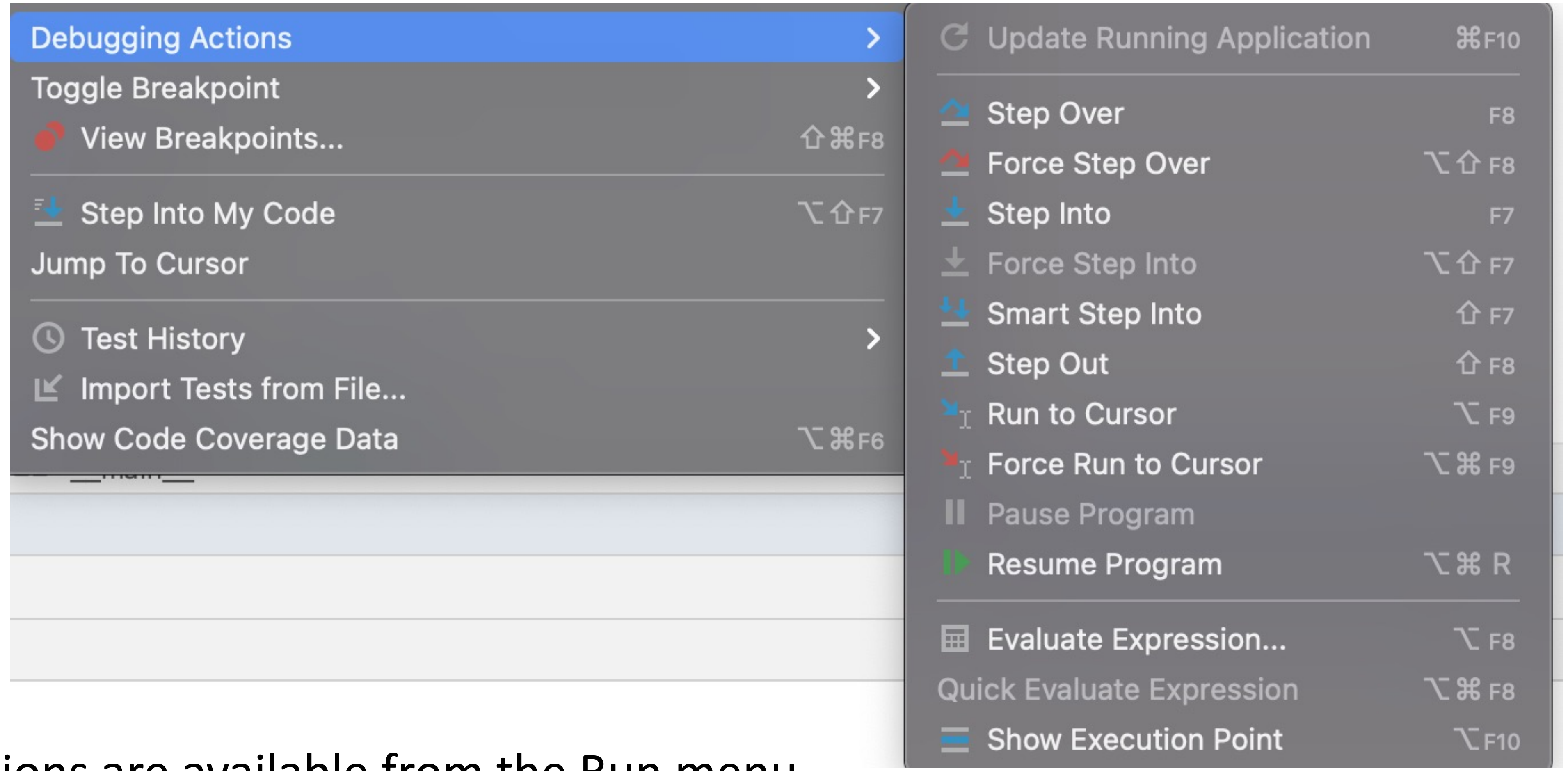
Important: line has yet to execute.



```
if __name__ == "__main__":

    # create a new list of integers
    new_list = [2, 3, 4, 5, 6]
```

Debug window at bottom of IDE appears and shows all current data. Nothing to show yet





Several debugger actions are available from the Run menu.

Step Over: execute the current line and move to the next one

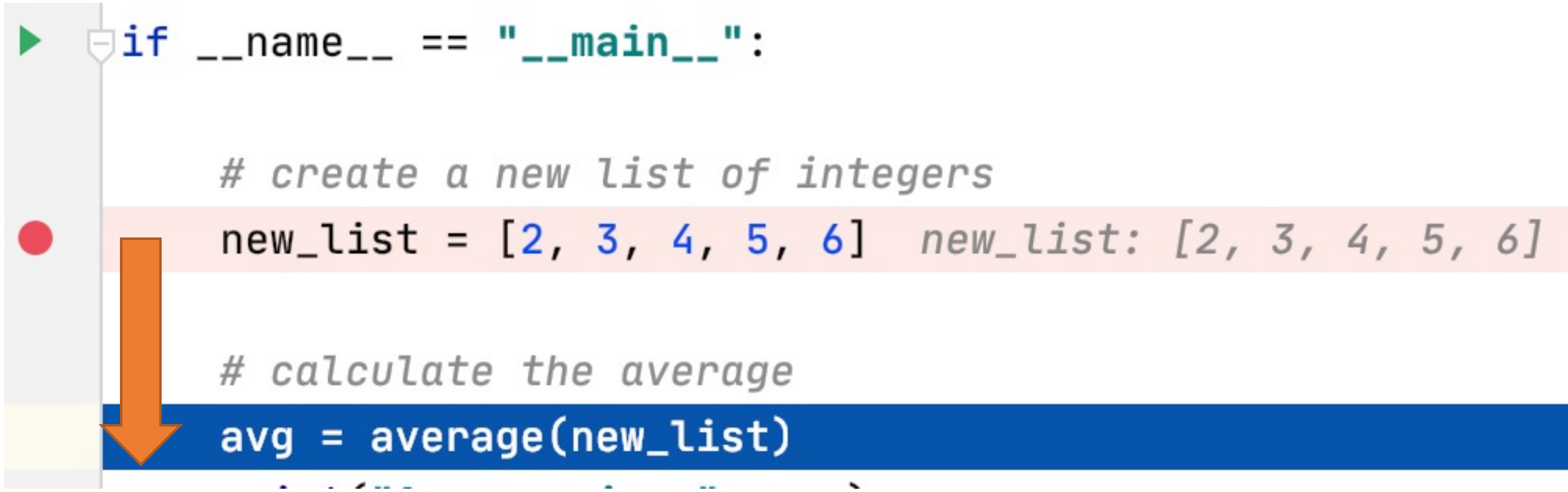
Step Into: if the current line is a function/method, drop into that function/method and show execution

Step Out: if you are in a method/function, jump out o fit.

Resume Program: stop debugging and run until completion or the next break point.

Step Over (F8) allows the line to execute. Control moves to the next line.

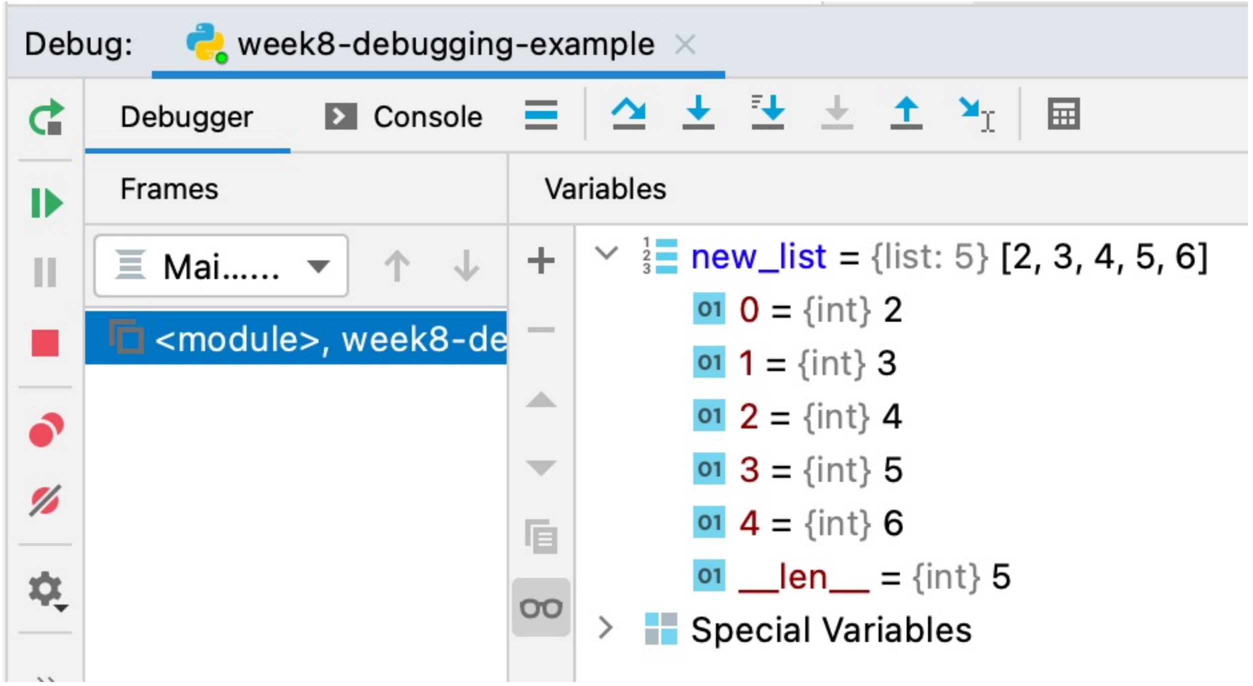
Result of operation show in code window.



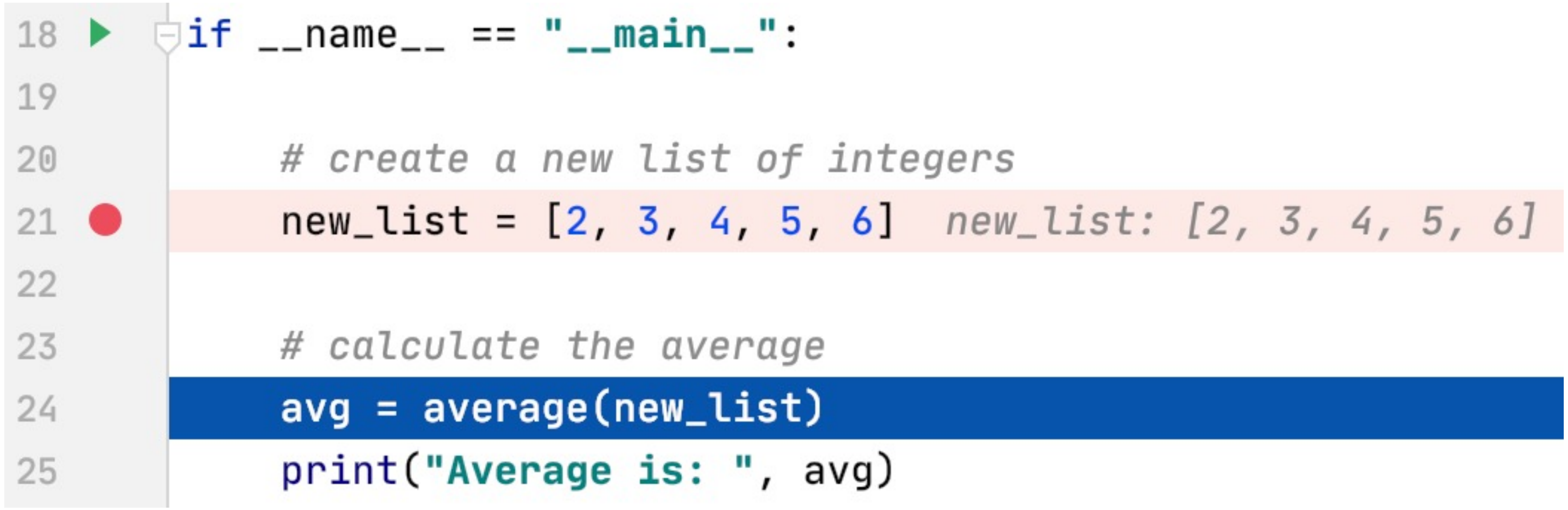
```
if __name__ == "__main__":

    # create a new list of integers
    new_list = [2, 3, 4, 5, 6]  new_list: [2, 3, 4, 5, 6]

    # calculate the average
    avg = average(new_list)
```

Debug window now shows "new_list" as it was created in the previous line.

Can click on list to view contents.



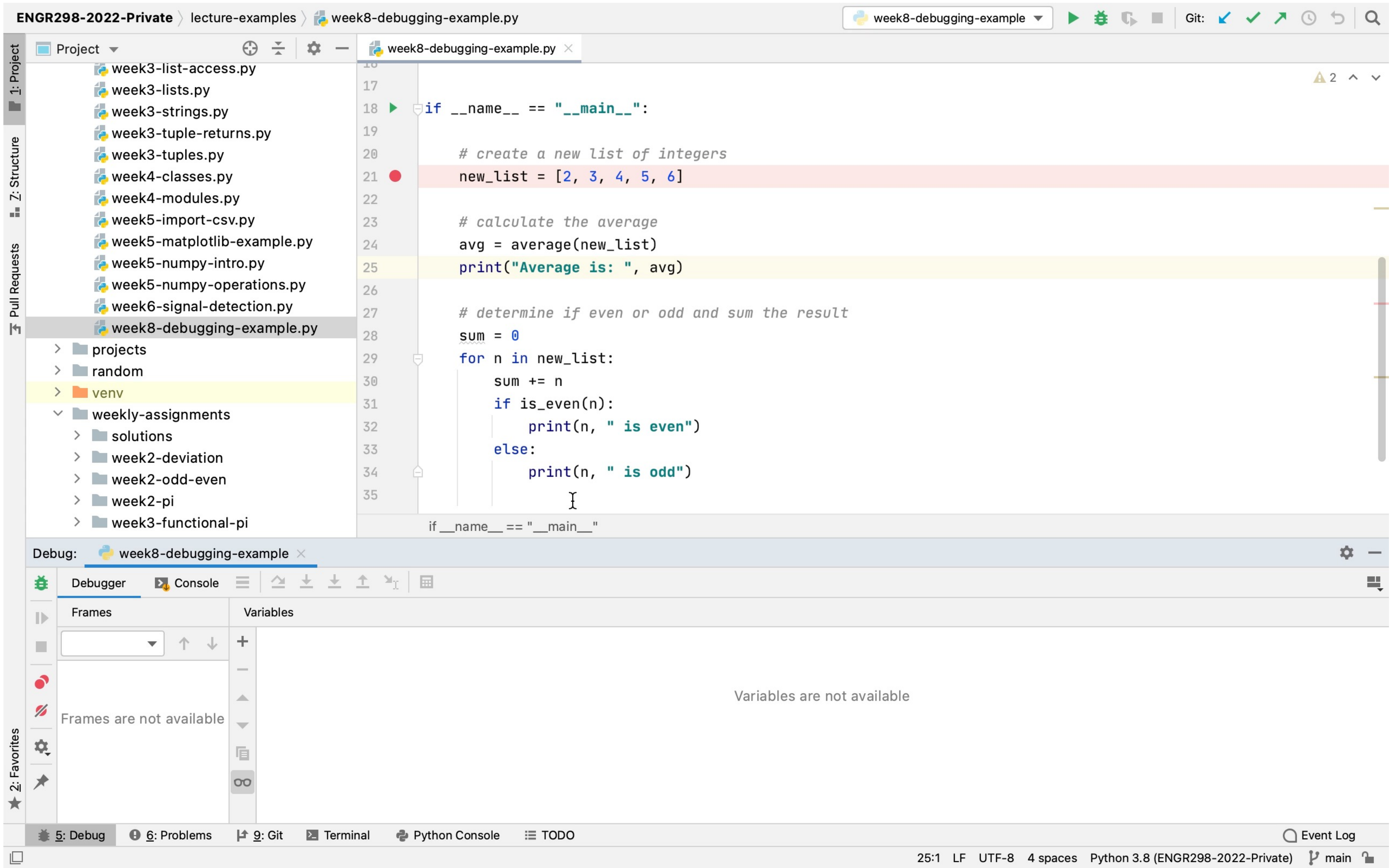
```
new_list = {list: 5} [2, 3, 4, 5, 6]
    0 = {int} 2
    1 = {int} 3
    2 = {int} 4
    3 = {int} 5
    4 = {int} 6
    __len__ = {int} 5
Special Variables
```



```
18  if __name__ == "__main__":
19
20      # create a new list of integers
21      new_list = [2, 3, 4, 5, 6]  new_list: [2, 3, 4, 5, 6]
22
23      # calculate the average
24      avg = average(new_list)
25      print("Average is: ", avg)
```

Stepping will go over any line (regardless of how complex the operation is). In the line above, if we step over line 24, average() will execute and then control will pass to line 25.

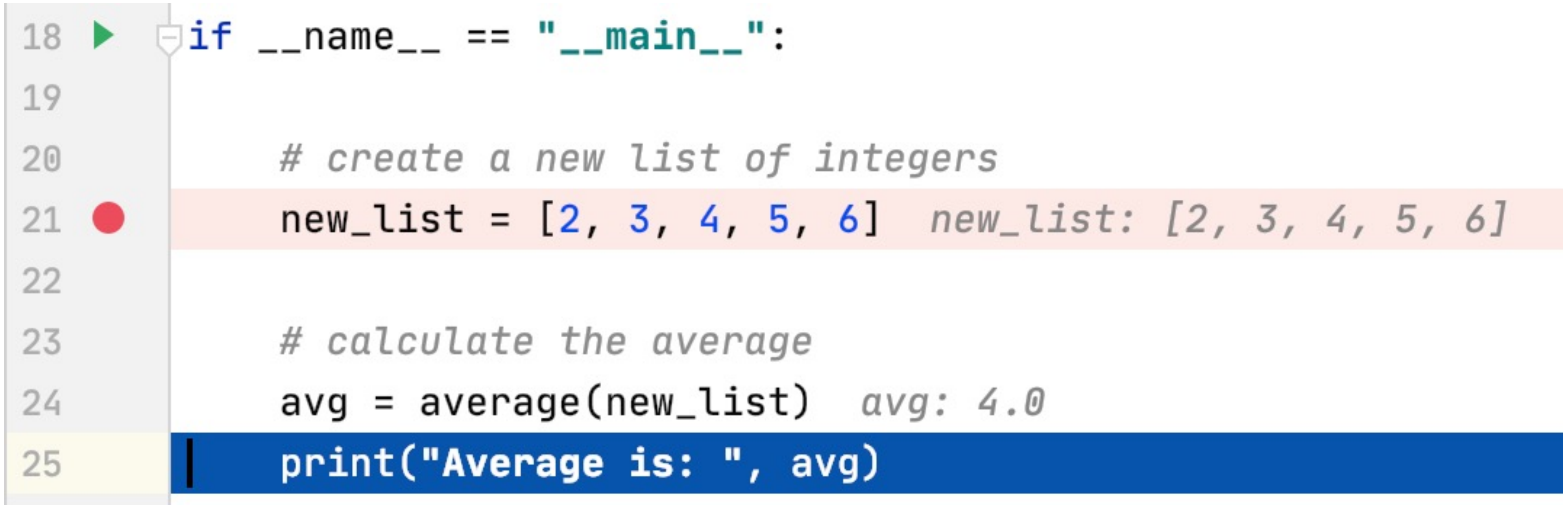
If you “step” over a long operation (loading a file) then the debugger may take some time to return control


ENGR298-2022-Private 〉 lecture-examples 〉 week8-debugging-example.py

week8-debugging-example.py

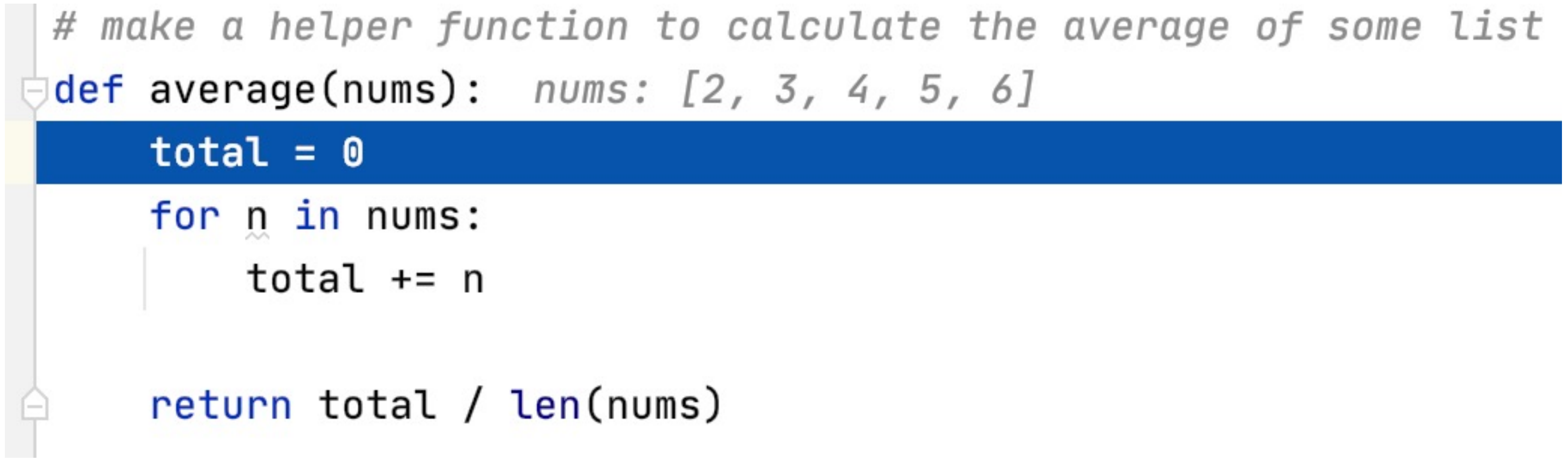
```
if __name__ == "__main__":

    # create a new list of integers
    new_list = [2, 3, 4, 5, 6]

    # calculate the average
    avg = average(new_list)
    print("Average is: ", avg)

    # determine if even or odd and sum the result
    sum = 0
    for n in new_list:
        sum += n
        if is_even(n):
            print(n, " is even")
        else:
            print(n, " is odd")

```

Average() has returned with the result 4.0 which was stored in avg



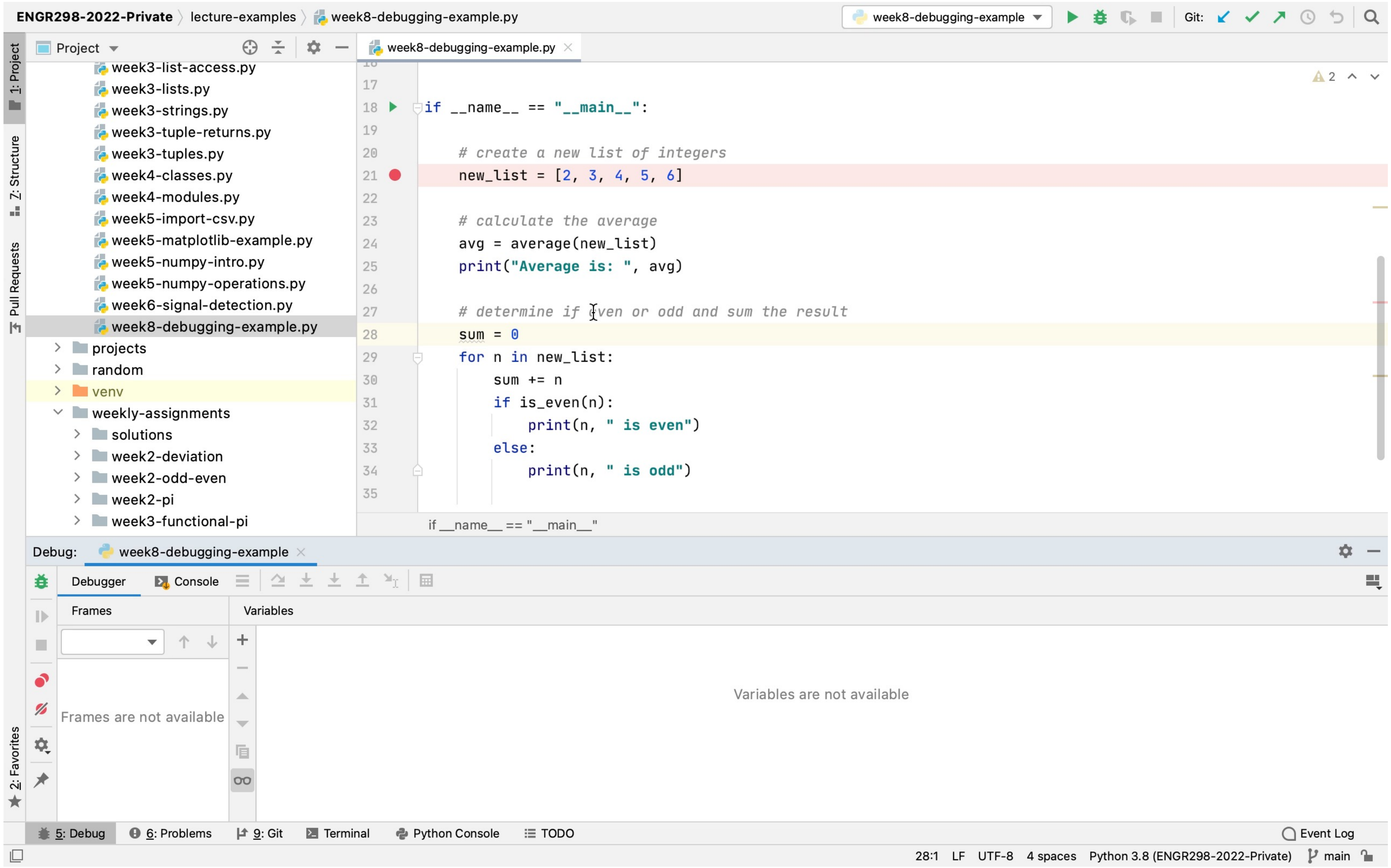
```
if __name__ == "__main__":

    # create a new list of integers
    new_list = [2, 3, 4, 5, 6]  new_list: [2, 3, 4, 5, 6]

    # calculate the average
    avg = average(new_list)  avg: 4.0
    print("Average is: ", avg)
```

What if we wanted to see what happens in average()? We would step in!



```
# make a helper function to calculate the average of some list
def average(nums):  nums: [2, 3, 4, 5, 6]
    total = 0
    for n in nums:
        total += n

    return total / len(nums)
```

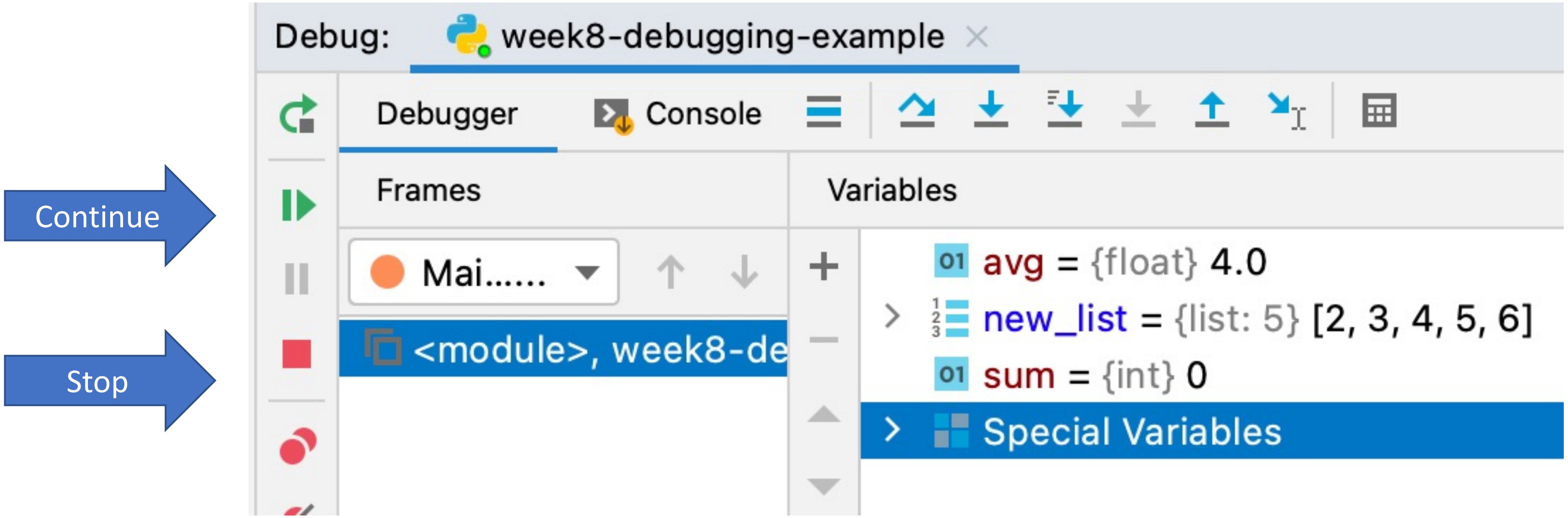
While in average() step over can be used to walk through the execution. This can run if you like, but you may wish to "step out" to bounce out of the function to where the program had previously "stepped in".



ENGR298-2022-Private 〉 lecture-examples 〉 week8-debugging-example.py

week8-debugging-example.py

```
if __name__ == "__main__":

    # create a new list of integers
    new_list = [2, 3, 4, 5, 6]

    # calculate the average
    avg = average(new_list)
    print("Average is: ", avg)

    # determine if even or odd and sum the result
    sum = 0
    for n in new_list:
        sum += n
        if is_even(n):
            print(n, " is even")
        else:
            print(n, " is odd")
```



avg = {float} 4.0
new_list = {list: 5} [2, 3, 4, 5, 6]
sum = {int} 0

When debugging is completed, select Continue to resume the program. It will run until completion or the next break point.

Stop will terminate the debugging session and immediately end the program.

Best Practices Using the Debugger

- When writing a new function, loop, or logic, create simple test cases to determine if the solution is correct. Build the program from several correct "blocks" rather than writing a whole program and then debugging.

- Each weekly-assignment will have some code that tests your solution. Use the debugger to walk through each one and see what works/doesn't.

- No need to start debugging from the beginning. Place break points in trouble spots, run the program, and wait until you hit it.

- Break points can also be set in places where your code ”shouldn't” go. Very nice to have a break point on a line that catches an error/problem. Can see the result before it occurs.

Some final comments before spring break…

- I appreciate the added engagement the last two weeks. More emails and office visits. Please continue to reach out. I know the work is challenging, but I hope, also interesting.
- Almost all the “weekly-assignments” have been modified to run in Gradescope. Examine the Canvas assignment for links to new templates. Submit to Gradescope for final grade.
- After spring break will resume very “application-based” approach for ~4 weeks. Final 2-3 weeks will be independent project.