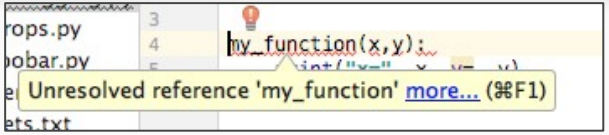
CSCI 141
Computer Science I

02-Debugging

Syntax Errors

- A syntax error is the result of using the language incorrectly
  - Forgetting def keyword when declaring a function
  - Forgetting a colon after an if condition
  - Invoking a misspelled function name
  - Forgetting commas between function parameters
  - Inconsistent indentation/whitespace



PyCharm will underline syntax errors in **red** and mousing over will display a popup with a descriptive error message.

```
>>> my_function(x,y):

SyntaxError: invalid syntax
>>> |
```

The Python interpreter will also provide immediate feedback about syntax errors.

Runtime Errors

- A runtime error is an application error that occurs during program execution
  - Using the wrong number of arguments to a function
  - Doing arithmetic with incompatible types
  - Divide by zero
  - Subscripting a string out of bounds

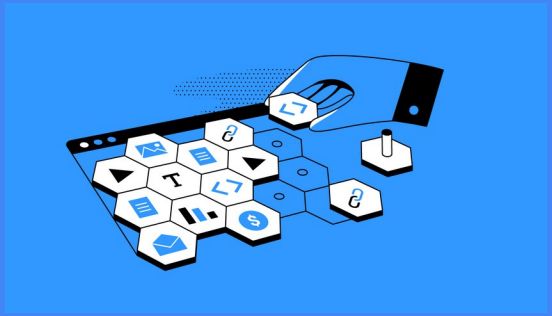
```
>>> divide(2, 0)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    divide(2, 0)
  File "<pyshell#4>", line 2, in divide
    return x / y
ZeroDivisionError: division by zero
```

The Python Interpreter (and IDLE) will print the traceback to the console.

```
/Library/Frameworks/Python.framework/Versions/3.5/bin/python3.5 /Users/robertstjacquesjr/Pyc
Traceback (most recent call last):
  File "/Users/robertstjacquesjr/PycharmProjects/strings/debugging.py", line 8, in <module>
    my_function(1, 2, 3)
TypeError: my_function() takes 2 positional arguments but 3 were given
```

PyCharm will hotlink the lines of code in the error message, allowing you to click and jump straight to the specific line.

Semantic Errors



- A semantic error occurs when a statement is syntactically valid, but does not do what the programmer intended
  - Using < instead of <= in a condition
  - “Off by one” errors looping too many/too few time
  - Path through a function that returns an incorrect result, or None
  - Order of operation mistake, e.g.
    - `1 + 5 * 6` vs `(1 + 5) * 6`

If some code does the wrong thing, it may not be the fault of that piece of code. It may instead be the fault of the code that called it.

Are the main cause for runtime errors.

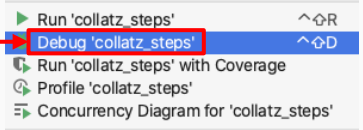
Debugging

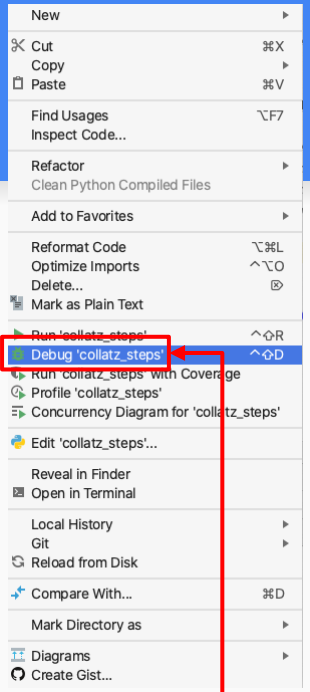
- Debugging is the process of finding and resolving defects, “bugs”, in a program that prevents correct operation
- The debugger allows the programmer to pause the program during execution so they can evaluate the state of the program, e.g. variable values, at critical points.
- Use of a debugger is only one way to debug. Could use print statements, or analyze/reason about the code.

The PyCharm Debugger

There are at least three ways to start the PyCharm debugger.

While your Python program is open, use the **Run** → **Debug…** menu option.





Click the debug icon on the toolbar (top right corner of the PyCharm window). For this to work, a run configuration must already exist

Right-click (or control-click) in the editor and select **Debug** from the pop-up menu.

Breakpoints

- A breakpoint is an intentional stopping or pausing place in a program, put in place for debugging purposes.

```
9   def collatz_steps(N):
10      """..."""
16      print(N, end=' ')
17      if N == 1:
18          return 1
19      elif N % 2 == 0:  # N is even
20          return 1 + collatz_steps(N // 2)
21      else:             # N is odd
22          return 1 + collatz_steps(3 * N + 1)
```

Breakpoints are added by clicking the mouse on the line number in the left margin of the editor.

A breakpoint icon will appear wherever a breakpoint has been set.

A breakpoint can be removed by clicking the icon again.

Breakpoints are only ”honored” in debug mode.

The PyCharm Debugger

Running the current program through the debugger will switch the PyCharm IDE to the debug layout.

The debug controls are here and here (we'll take a closer look in a moment).

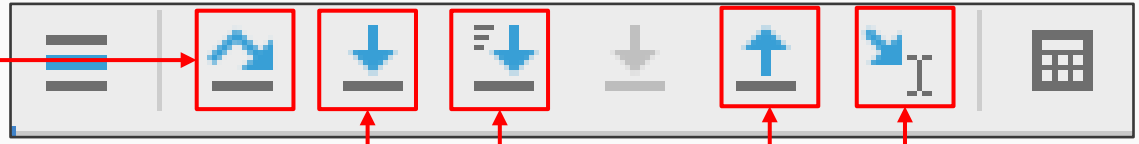
Can switch between the debugger details (default) and the console I/O

The next line of code to be executed is highlighted in the editor.

Variables that are in scope, as well as their current values, are displayed on the bottom right.

The trace of modules and functions called, including line numbers, is displayed on the bottom left.

Debugging Controls



Step Over. Execute the current line and move to the next one, even if it is a function call.

Step Into. Steps into a function to continue line-by-line debugging.

Step Into My Code. If you are calling code you did not write (e.g. `random.randint(a,b)`) this executes without stopping until execution reaches *your* next line of code.

Step Out. Runs to the end of the current function.

Run to Cursor. Click to place the cursor anywhere in the code. This will run to that point before stopping.

More Debugging Controls

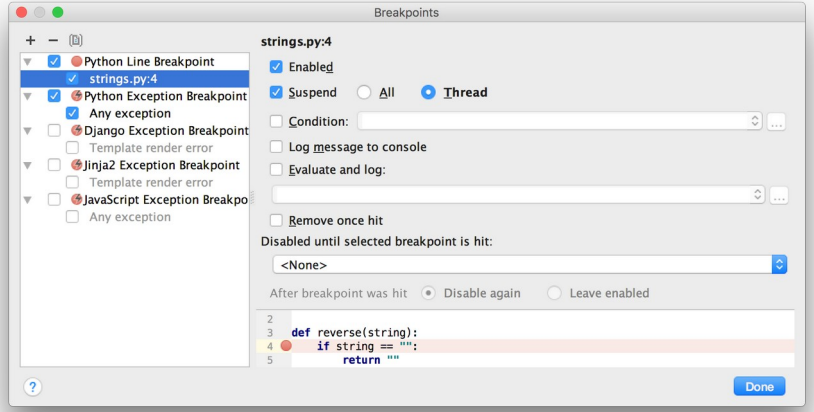
Rerun Debugger. Rerun the last program through the debugger again.

Resume Program. Run a currently paused program to the next breakpoint.

Stop. Cancel execution of the currently running program.

View Breakpoints. Opens a dialog that lists all breakpoints in the current project.

Mute Breakpoints. Execution will continue without pausing at breakpoints.



Debugging Exercise: `collatz_steps.py`

Add a breakpoint on the first statement in the `collatz_steps` function

Add a breakpoint on the first statement in the `main` function

```
9   def collatz_steps(n):
10      """..."""
16      print(n, end=' ')
17      if n == 1:
18          return 1
19      elif n % 2 == 0:  # N is even
20          return 1 + collatz_steps(n // 2)
21      else:             # N is odd
22          return 1 + collatz_steps(3 * n + 1)
23
24  def main():
25      """..."""
30      n = int(input('Enter N: '))
31      if n <= 0:
32          print('N > 0')
33      else:
34          print('\ncollatz(', n, ')=', collatz_steps(n))
35
36  if __name__ == '__main__':
37      main()
```

Debugging Exercise

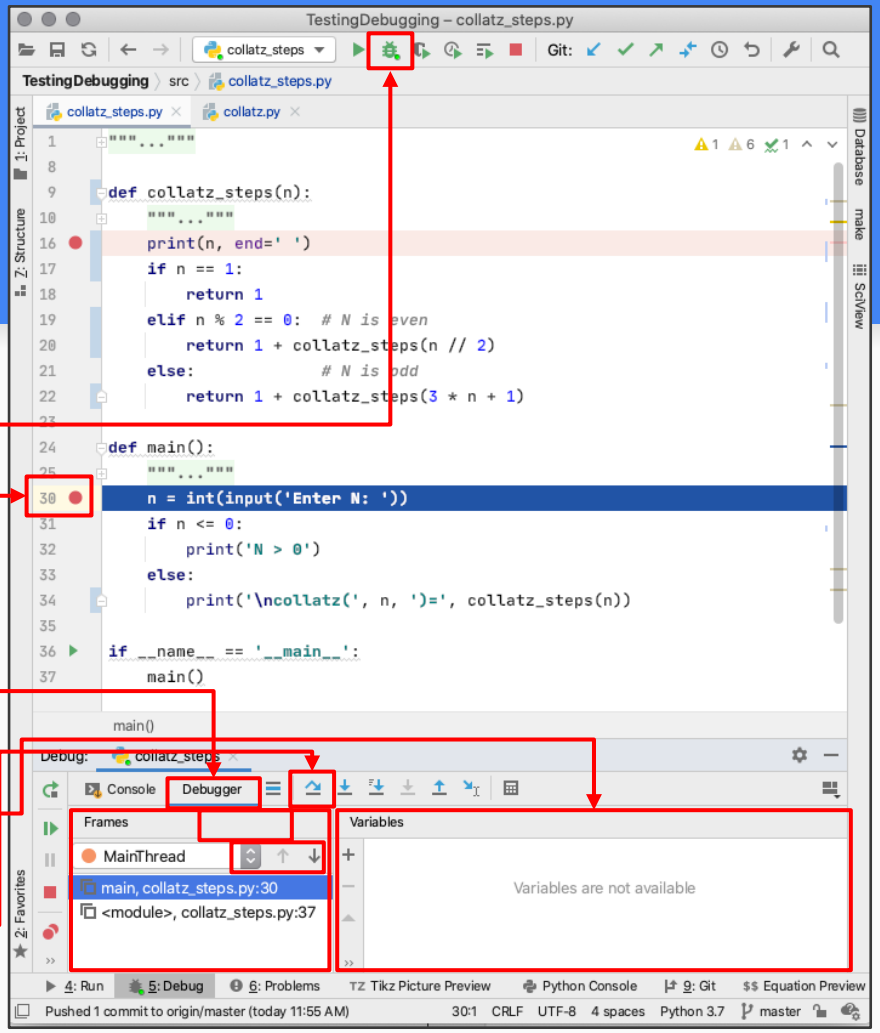
Run `collatz_steps` in debug mode

Execution pauses on the first breakpoint in `main()`

The **Debugger** tab is selected and the call stack is shown

No variables are in scope yet

Step over to execute line 30



Debugging Exercise

Switch to **Console** for I/O

Enter a value of 5 for N in the console and hit ENTER

TestingDebugging – collatz_steps.py

```
"""..."""

def collatz_steps(n):
    """..."""
    print(n, end=' ')
    if n == 1:
        return 1
    elif n % 2 == 0:  # N is even
        return 1 + collatz_steps(n // 2)
    else:             # N is odd
        return 1 + collatz_steps(3 * n + 1)

def main():
    """..."""
    n = int(input('Enter N: '))
    if n <= 0:
        print('N > 0')
    else:
        print('\ncollatz(', n, ')=', collatz_steps(n))

if __name__ == '__main__':
    main()
```

Debug: collatz_steps

Console | Debugger

```
Connected to pydev debugger (build 202.6397.98)
Enter N:
>? 5
```

Debugging Exercise

Switch back to **Debugger**

The program has advanced to the next line

The editor shows known values for variables

The values for variables in scope is shown

TestingDebugging – collatz_steps.py

```
"""..."""

def collatz_steps(n):
    """..."""
    print(n, end=' ')
    if n == 1:
        return 1
    elif n % 2 == 0:  # N is even
        return 1 + collatz_steps(n // 2)
    else:             # N is odd
        return 1 + collatz_steps(3 * n + 1)

def main():
    """..."""
    n = int(input('Enter N: '))  n: 5
    if n <= 0:
        print('N > 0')
    else:
        print('\ncollatz(', n, ')=', collatz_steps(n))

if __name__ == '__main__':
    main()
```

Debug: collatz_steps

Console | Debugger

Frames: MainThread; main, collatz_steps.py:31; <module>, collatz_steps.py:37

Variables: n = {int} 5

Debugging Exercise

Select **Resume** to continue execution

The program advances to the next breakpoint in `collatz_steps()`

A stack frame for the first `collatz_steps()` call is pushed onto the call stack

In this new stack frame, n=5

TestingDebugging – collatz_steps.py

```
"""..."""

def collatz_steps(n):  n: 5
    """..."""
    print(n, end=' ')
    if n == 1:
        return 1
    elif n % 2 == 0:  # N is even
        return 1 + collatz_steps(n // 2)
    else:             # N is odd
        return 1 + collatz_steps(3 * n + 1)

def main():
    """..."""
    n = int(input('Enter N: '))
    if n <= 0:
        print('N > 0')
    else:
        print('\ncollatz(', n, ')=', collatz_steps(n))

if __name__ == '__main__':
    main()
```

Debug: collatz_steps

Console | Debugger

Frames

MainThread

collatz_steps, collatz_steps.py:16

main, collatz_steps.py:34

<module>, collatz_steps.py:37

Variables

n = {int} 5

Debugging Exercise

Press **Step Over**

The program advances to line 17

Switching to **Console** shows the output from line 16

TestingDebugging – collatz_steps.py

TestingDebugging 〉 src 〉 collatz_steps.py

```
1   """..."""
8
9   def collatz_steps(n):   n: 5
10      """..."""
16      print(n, end=' ')
17      if n == 1:
18          return 1
19      elif n % 2 == 0:   # N is even
20          return 1 + collatz_steps(n // 2)
21      else:              # N is odd
22          return 1 + collatz_steps(3 * n + 1)
23
24  def main():
25      """..."""
30      n = int(input('Enter N: '))
31      if n <= 0:
32          print('N > 0')
33      else:
34          print('\ncollatz(', n, ')=', collatz_steps(n))
35
36  if __name__ == '__main__':
37      main()
```

collatz_steps()

Debug: collatz_steps

Console | Debugger

```
Connected to pydev debugger (build 202.6397.98)
Enter N: >? 5
5
>>>
```

4: Run | 5: Debug | 6: Problems | Tikz Picture Preview | Python Console | 9: Git | Equation Preview

Pushed 1 commit to origin/master (today 11:55 AM) 1:1 CRLF UTF-8 4 spaces Python 3.7 master

Debugging Exercise

Switch back to **Debugger**

Press **Step Over**

The program advances to line 19

Press **Step Over**

The program advances to line 22

TestingDebugging – collatz_steps.py

```
1   """..."""
8
9   def collatz_steps(n):   n: 5
10      """..."""
16      print(n, end=' ')
17      if n == 1:
18          return 1
19      elif n % 2 == 0:   # N is even
20          return 1 + collatz_steps(n // 2)
21      else:              # N is odd
22          return 1 + collatz_steps(3 * n + 1)
23
24  def main():
25      """..."""
30      n = int(input('Enter N: '))
31      if n <= 0:
32          print('N > 0')
33      else:
34          print('\ncollatz(', n, ')=', collatz_steps(n))
35
36  if __name__ == '__main__':
37      main()
```

Debug: collatz_steps

Console | Debugger

Frames: MainThread
- collatz_steps, collatz_steps.py:22
- main, collatz_steps.py:34
- <module>, collatz_steps.py:37

Variables: n = {int} 5

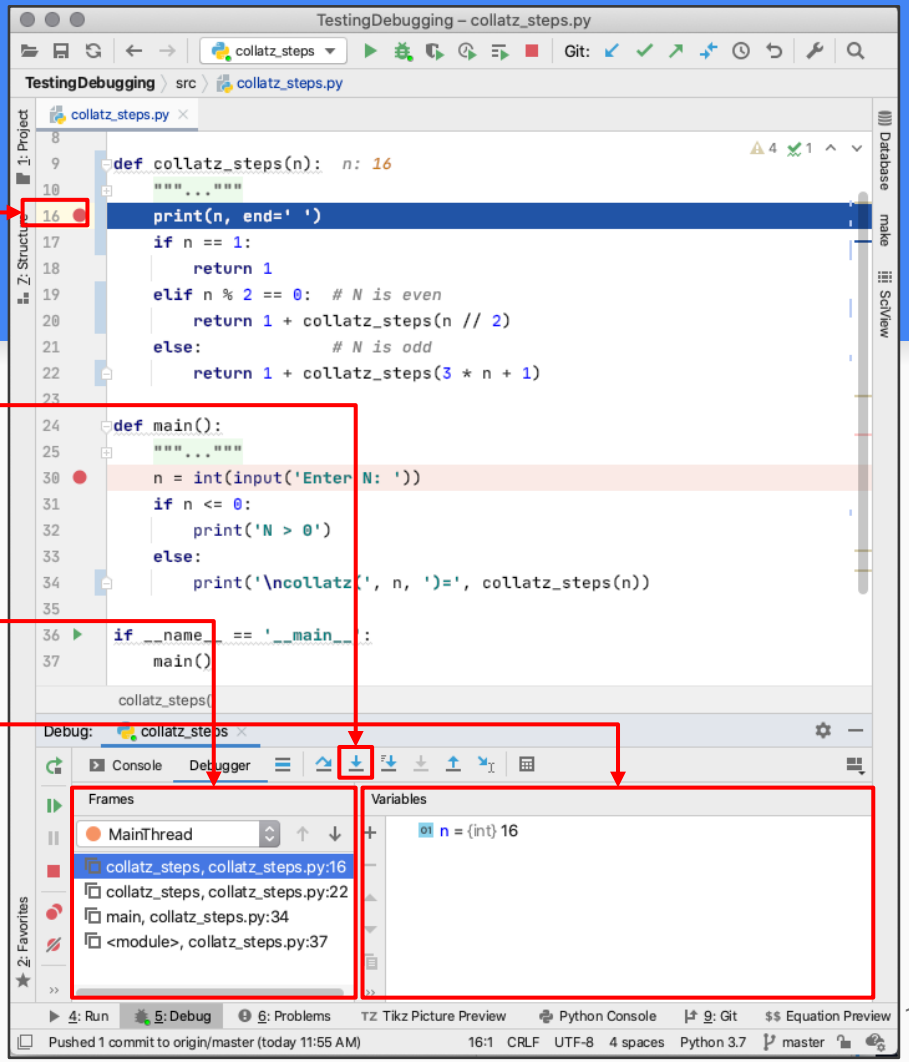
Debugging Exercise

Press **Step Into**

The program advances to line 16

A new stack frame is pushed

In this new stack frame, n=16



Debugging Exercise

Can switch back to lower stack frames at any point

The editor shows the line number, 22, for this stack frame

In this stack frame, n=5

TestingDebugging – collatz_steps.py

```
10      """..."""
16      print(n, end=' ')
17      if n == 1:
18          return 1
19      elif n % 2 == 0:  # N is even
20          return 1 + collatz_steps(n // 2)
21      else:             # N is odd
22          return 1 + collatz_steps(3 * n + 1)
23
24  def main():
25      """..."""
30      n = int(input('Enter N: '))
31      if n <= 0:
32          print('N > 0')
33      else:
34          print('\ncollatz(', n, ')=', collatz_steps(n))
35
36  if __name__ == '__main__':
37      main()
```

Debug: collatz_steps

Console | Debugger

Frames

MainThread

collatz_steps, collatz_steps.py:16

collatz_steps, collatz_steps.py:22

main, collatz_steps.py:34

<module>, collatz_steps.py:37

Variables

n = {int} 5

Debugging Exercise

Remove the two breakpoints by clicking on them again

Click **Resume**

The program runs to completion and the final output is displayed

TestingDebugging – collatz_steps.py

```
    """..."""
    print(n, end=' ')
    if n == 1:
        return 1
    elif n % 2 == 0:  # N is even
        return 1 + collatz_steps(n // 2)
    else:             # N is odd
        return 1 + collatz_steps(3 * n + 1)

def main():
    """..."""
    n = int(input('Enter N: '))
    if n <= 0:
        print('N > 0')
    else:
        print('\ncollatz(', n, ')=', collatz_steps(n))

if __name__ == '__main__':
    main()
```

```
pydev debugger: process 27634 is connecting

Connected to pydev debugger (build 202.6397.98)
Enter N: 5 16 8 4 2 1
collatz( 5 )= 6

Process finished with exit code 0
```