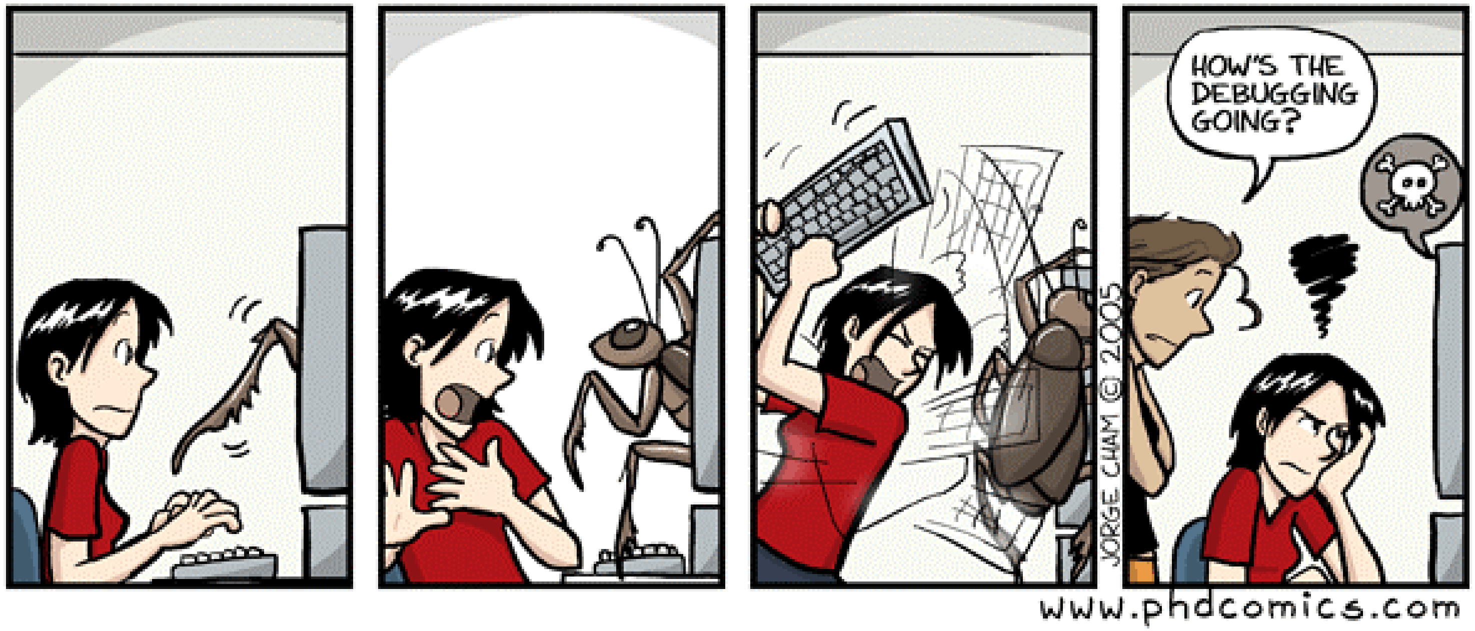
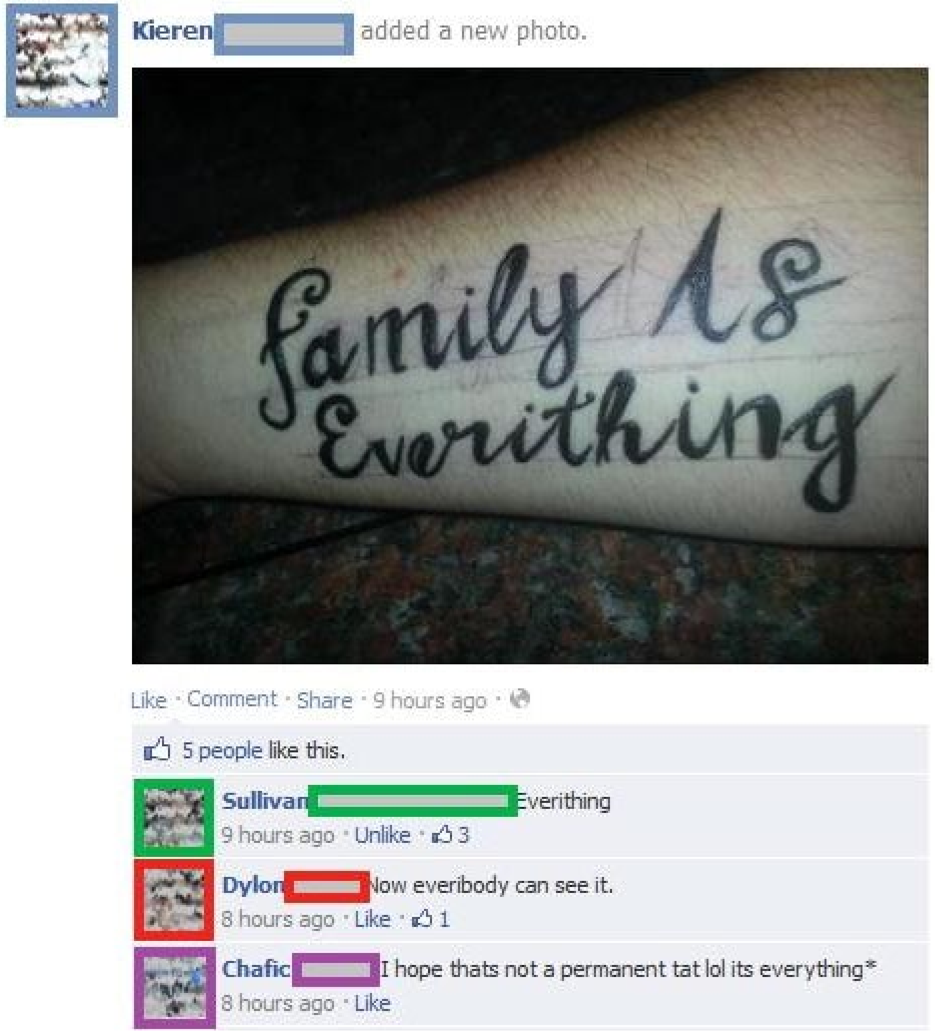
Bugs and debugging



The Very First Obstacle of Programming

- Syntax Error
  - A syntax error is an error in the source code of a program. Since computer programs **must follow strict syntax** to compile correctly, any aspects of the code that do not conform to the syntax of the programming language will produce a syntax error.

```
>>> x = 10
SyntaxError: invalid syntax
>>> |
```


Kieren [redacted] added a new photo.

Like · Comment · Share · 9 hours ago ·

5 people like this.

Sullivan [redacted] Everithing
9 hours ago · Unlike · 3

Dylon [redacted] Now everibody can see it.
8 hours ago · Like · 1

Chafic [redacted] I hope thats not a permanent tat lol its everything*
8 hours ago · Like

Sometime Errors are Fatal

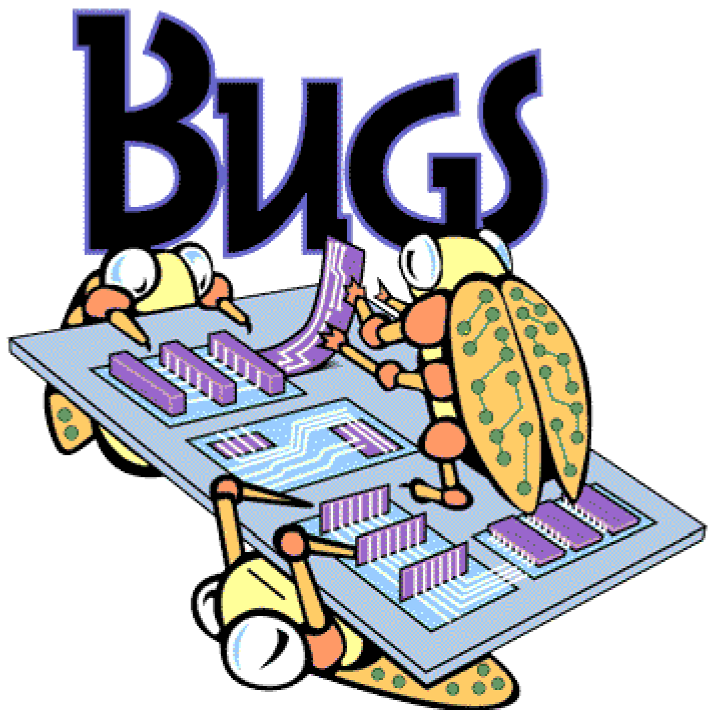
- https://www.youtube.com/watch?v=VjJgiDuHlRw

source is mentioned in our blog: www.unbelievablefactsblog.com

In 1962, a programmer omitted a single hyphen in the code for the Mariner I rocket causing it to explode shortly after take off. This typo cost NASA the today's equivalent of $630 million dollars.

Bugs?

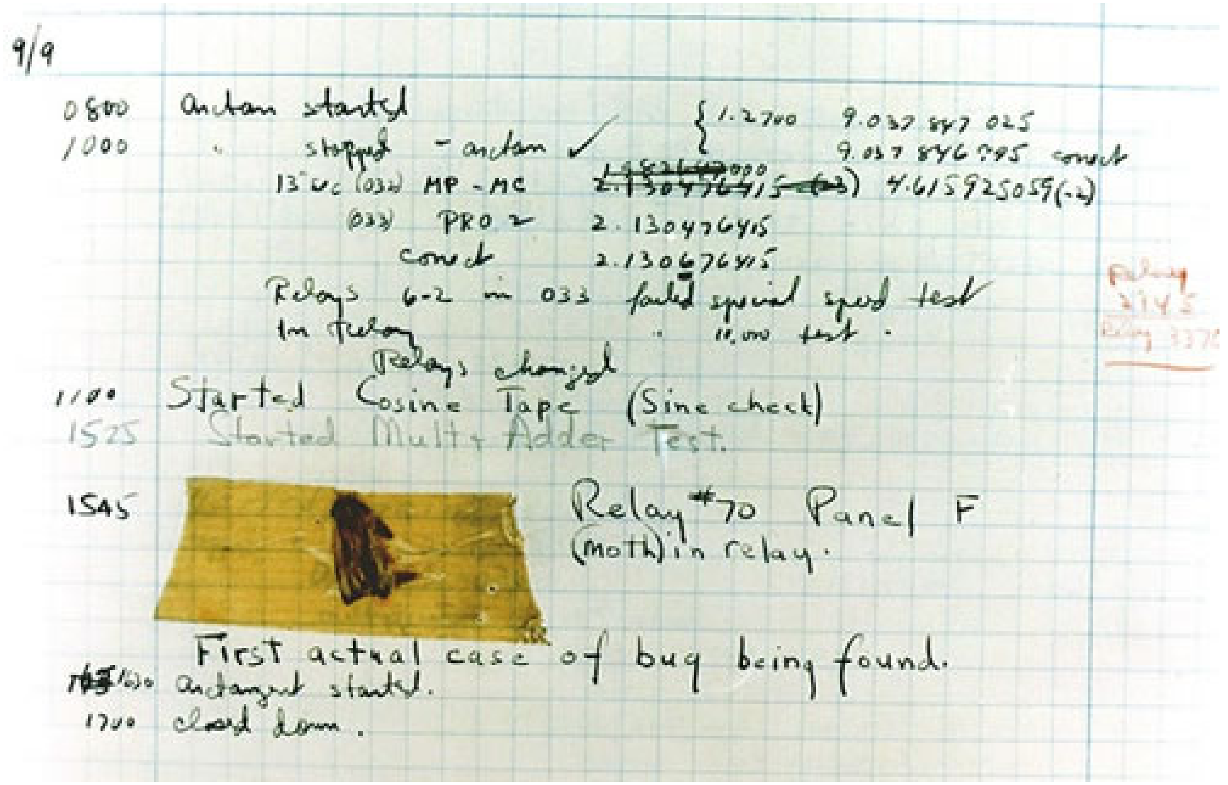
- In 1947, Grace Murray Hopper was working on the Harvard University Mark II Aiken Relay Calculator (a primitive computer).
- On the 9th of September, 1947, when the machine was experiencing problems, an investigation showed that there was a moth trapped between the points of Relay #70, in Panel F.



Mark I

Bugs?

- The operators removed the moth and affixed it to the log. (See the picture above.) The entry reads: "First actual case of bug being found."



9/9

0800 Antan started
1000 " stopped - antan ✓ {1.2700 9.037 847 025
9.037 846 995 correct
13" uc (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ ~~(-3)~~ 4.615925059(-2)
(033) PRO 2 2.130476415
correct 2.130676415
Relays 6-2 in 033 failed special speed test
In relay " 10,000 test .
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1630 Antangent started.
1700 closed down.

Relay 2145
Relay 3370

Humans make mistakes

You are only human

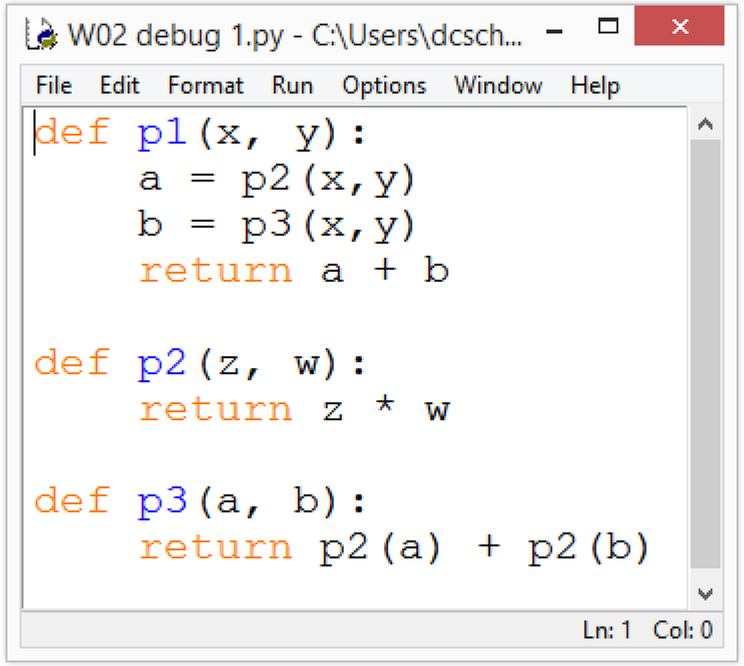
Therefore, you will make mistakes

Debugging

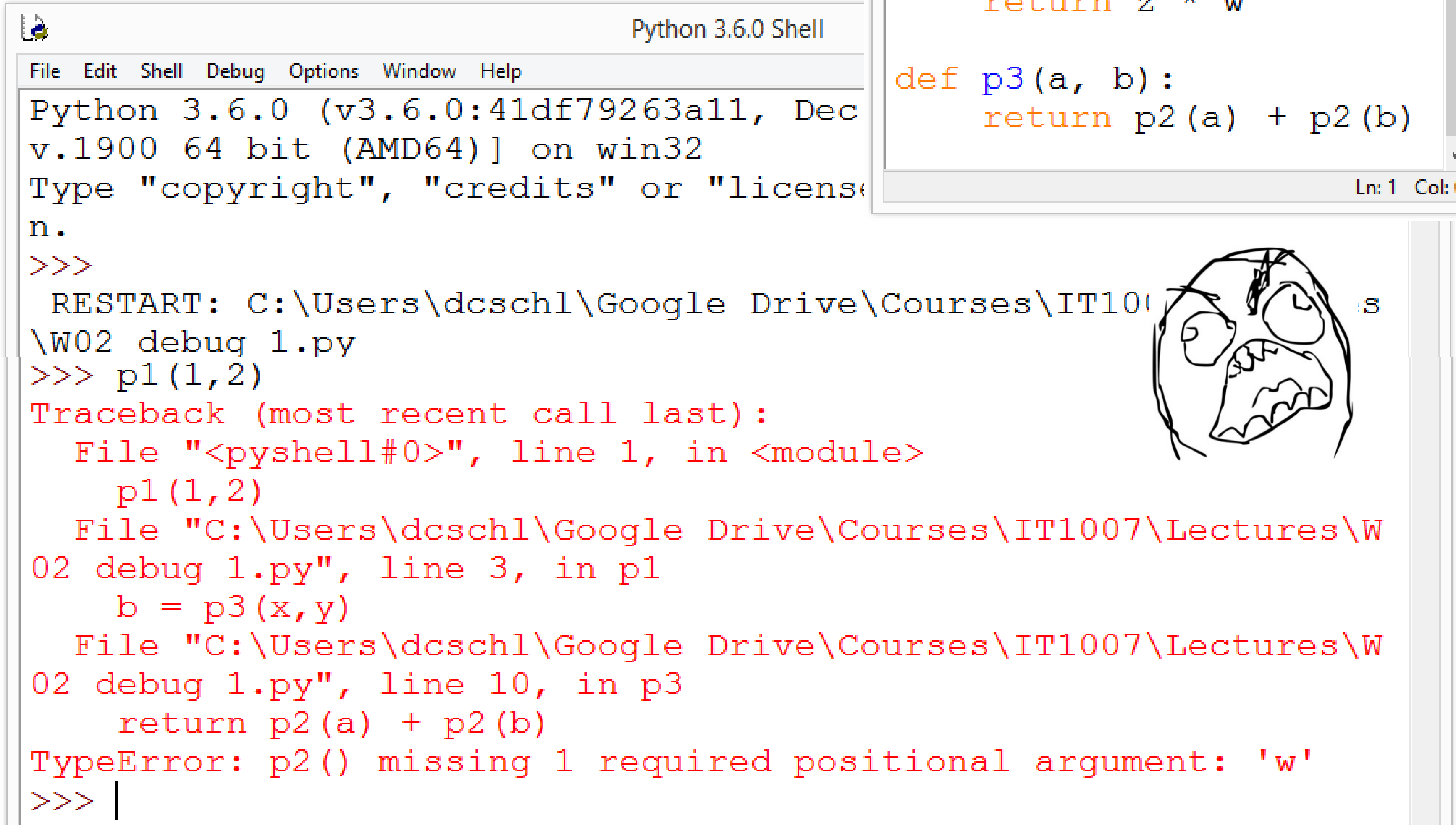
Debugging

- Means to remove errors (“bugs”) from a program.
- After debugging, the program is not necessarily error-free.
    - It just means that whatever errors remain are harder to find.
    - This is especially true for large applications.

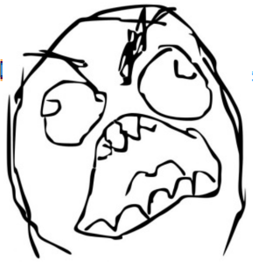
W02 debug1.py


W02 debug 1.py - C:\Users\dcsch...
File Edit Format Run Options Window Help
Ln: 1 Col: 0

```
def p1(x, y):
    a = p2(x,y)
    b = p3(x,y)
    return a + b

def p2(z, w):
    return z * w

def p3(a, b):
    return p2(a) + p2(b)
```

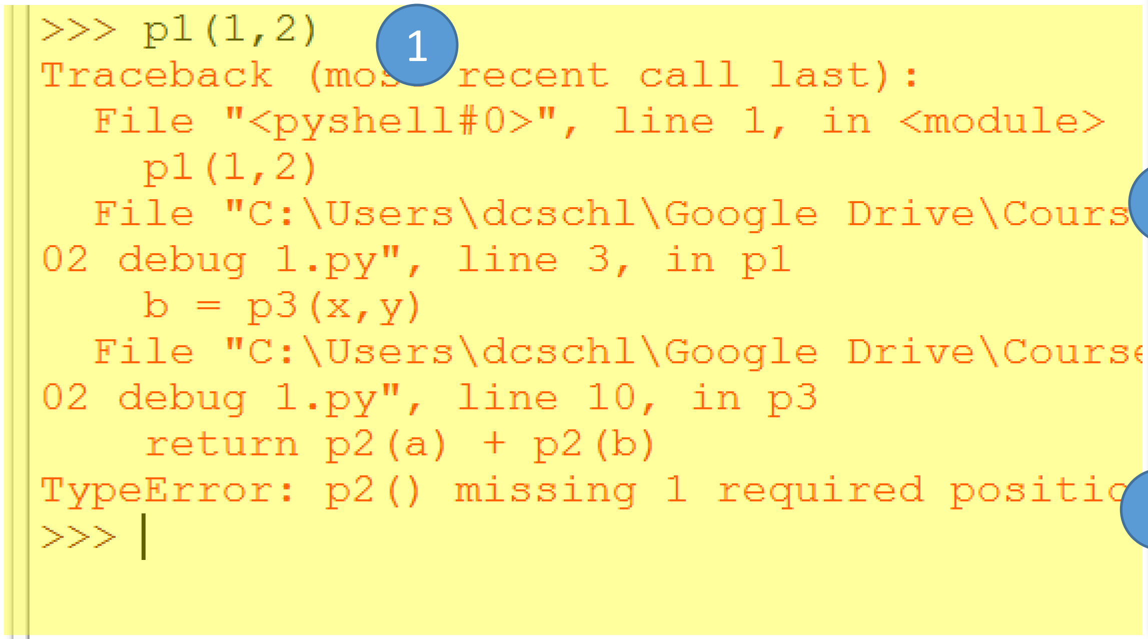


```
Python 3.6.0 (v3.6.0:41df79263a11, Dec
v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license
n.
>>>
 RESTART: C:\Users\dcschl\Google Drive\Courses\IT10      s
\W02 debug 1.py
>>> p1(1,2)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    p1(1,2)
  File "C:\Users\dcschl\Google Drive\Courses\IT1007\Lectures\W
02 debug 1.py", line 3, in p1
    b = p3(x,y)
  File "C:\Users\dcschl\Google Drive\Courses\IT1007\Lectures\W
02 debug 1.py", line 10, in p3
    return p2(a) + p2(b)
TypeError: p2() missing 1 required positional argument: 'w'
>>> 
```

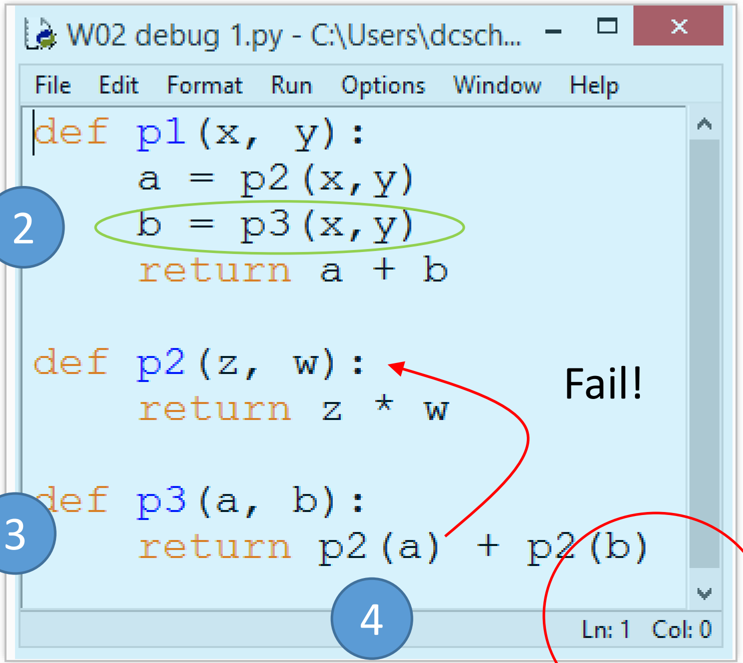


K

KEEP
CALM
AND
DON'T
PANIC
KeepCalmAndPosters.com



```
>>> p1(1,2)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    p1(1,2)
  File "C:\Users\dcschl\Google Drive\Cours
02 debug 1.py", line 3, in p1
    b = p3(x,y)
  File "C:\Users\dcschl\Google Drive\Course
02 debug 1.py", line 10, in p3
    return p2(a) + p2(b)
TypeError: p2() missing 1 required positio
>>> |
```



W02 debug 1.py - C:\Users\dcsch...
File Edit Format Run Options Window Help
Fail!
Ln: 1 Col: 0

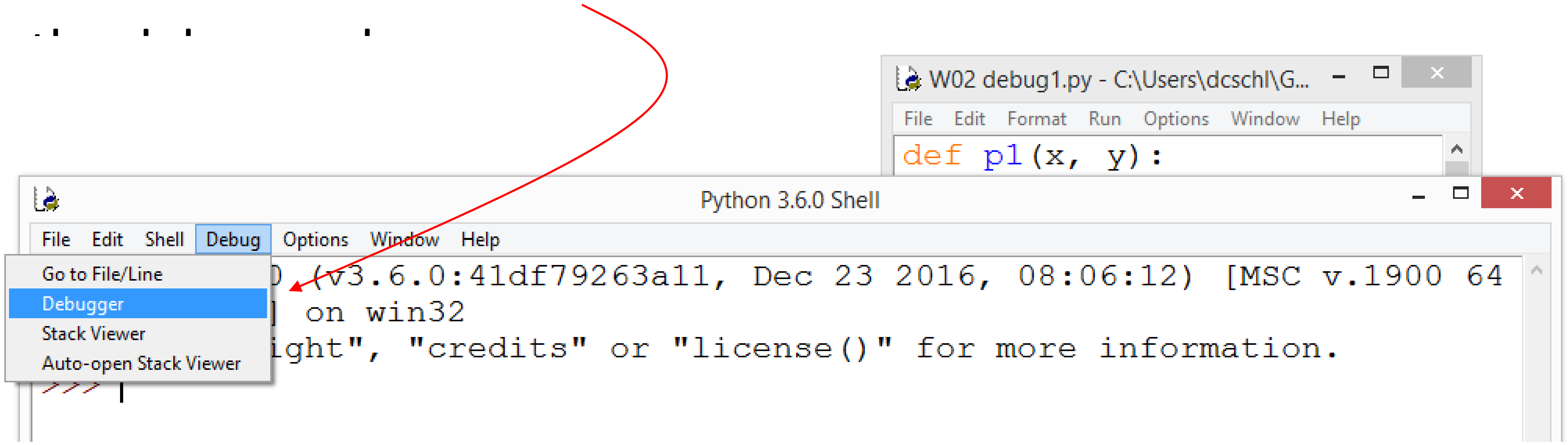
```
def p1(x, y):
    a = p2(x,y)
    b = p3(x,y)
    return a + b

def p2(z, w):
    return z * w

def p3(a, b):
    return p2(a) + p2(b)
```

```
Traceback (most recent call last):
1   File "<pyshell#0>", line 1, in <module>
      p1(1,2)
2   File "C:\Users\dcschl\Google Drive\Courses\IT1007\Lectures\W02 debug 1.py",
line 3, in p1
      b = p3(x,y)
3   File "C:\Users\dcschl\Google Drive\Courses\IT1007\Lectures\W02 debug 1.py",
line 10, in p3
      return p2(a) + p2(b)
4 TypeError: p2() missing 1 required positional argument: 'w'
```

The IDLE Debugger

Using the IDLE Debugger

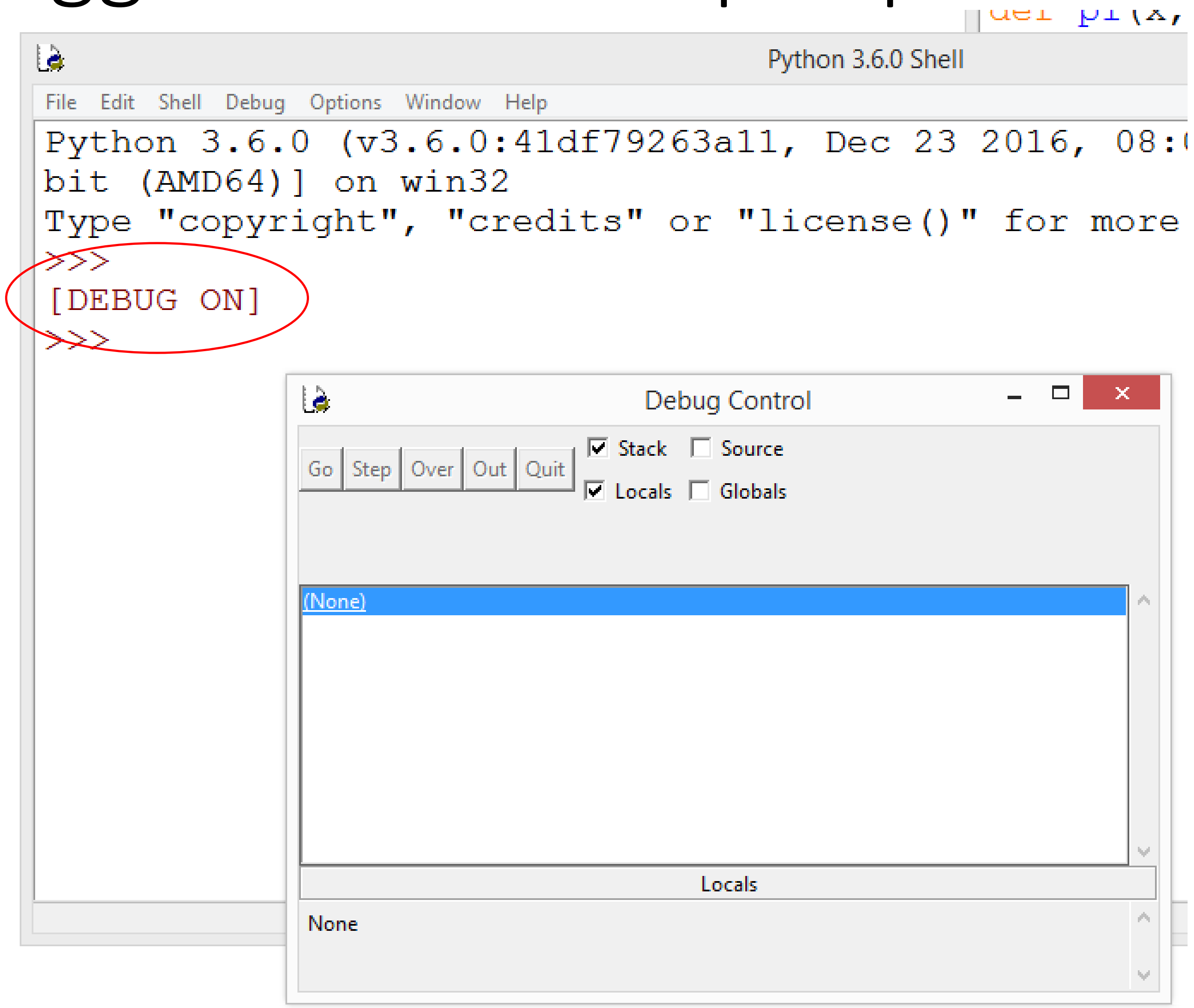
- Load in your source code
- Turn o



```
W02 debug1.py - C:\Users\dcschl\G...
def p1(x, y):
```

```
Python 3.6.0 Shell
0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64
] on win32
ight", "credits" or "license()" for more information.
>>> 
```

The Debugger Window Pops up



```
def pi(x,
```

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:
bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more
>>>
[DEBUG ON]
>>>
```

[✓ Stack] Source [✓ Locals] Globals

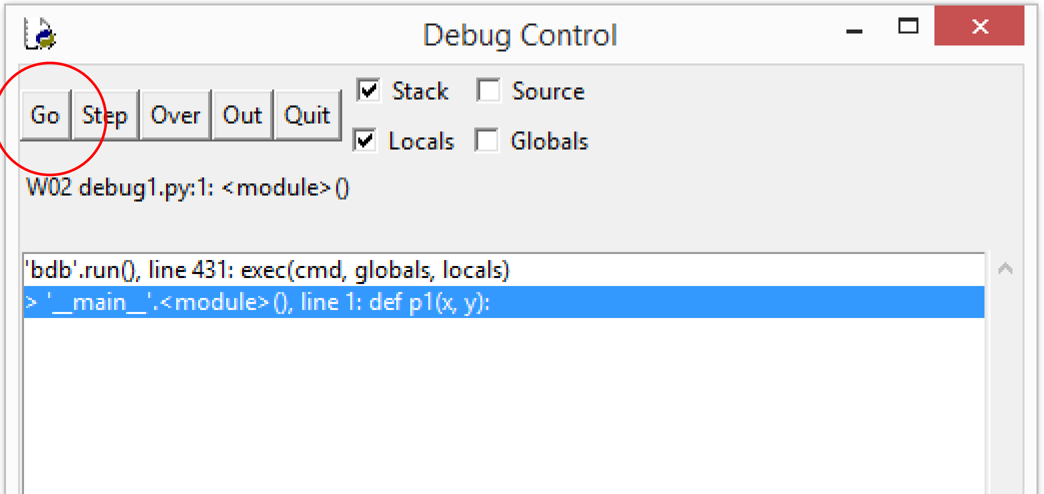
(None)

Locals

None

Using the IDLE Debugger

- Go to your source code window to “run”
- Then the debugger will pause the program at the first line of code and wait for you
- You can click the button “Go”
  - That will make the program run
  - At this point we don’t have any error
    - Because by “ru   ıs



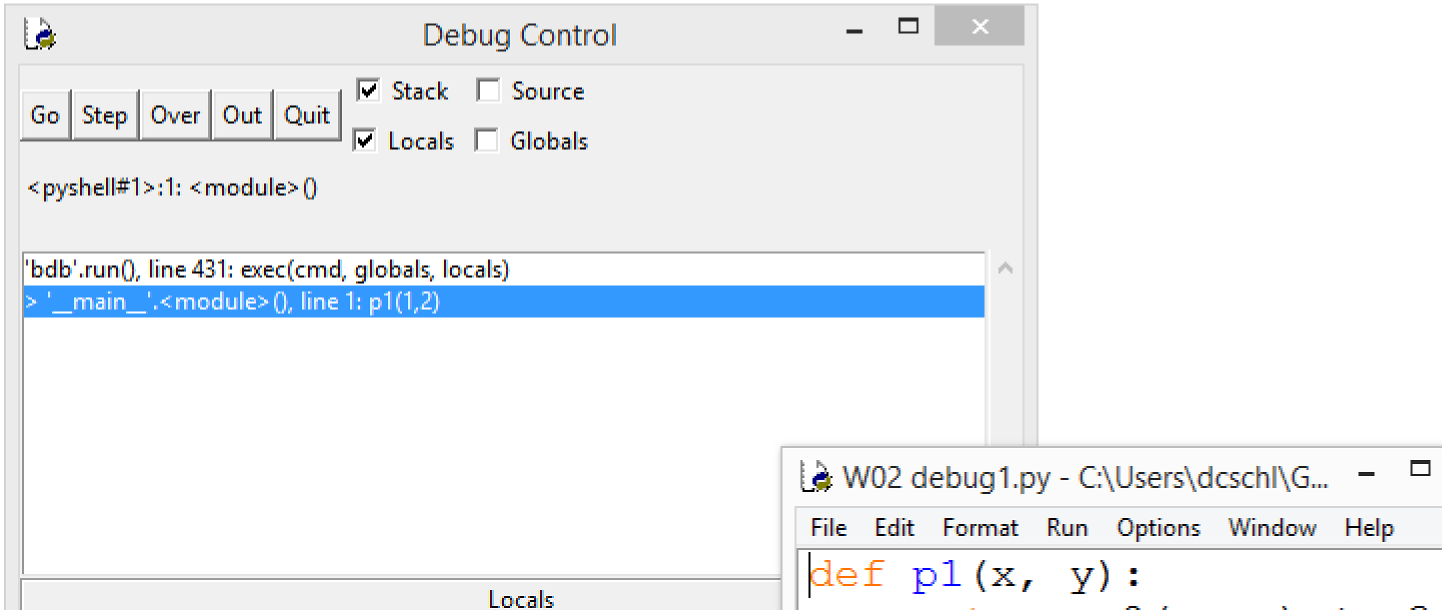
```
W02 debug1.py:1: <module>()
'bdb'.run(), line 431: exec(cmd, globals, locals)
> '__main__'.<module>(), line 1: def p1(x, y):
```

Using the IDLE Debugger

- Let’s execute the function in debug mode
- In the shell, type

```
p1(1,2)
```

- Then the debugger will pause at the first line of p1
  - If you type “go” now, you will



Stack [✓] Source [ ] Locals [✓] Globals [ ]

```
<pyshell#1>:1: <module>()
'bdb'.run(), line 431: exec(cmd, globals, locals)
> '__main__'.<module>(), line 1: p1(1,2)
```

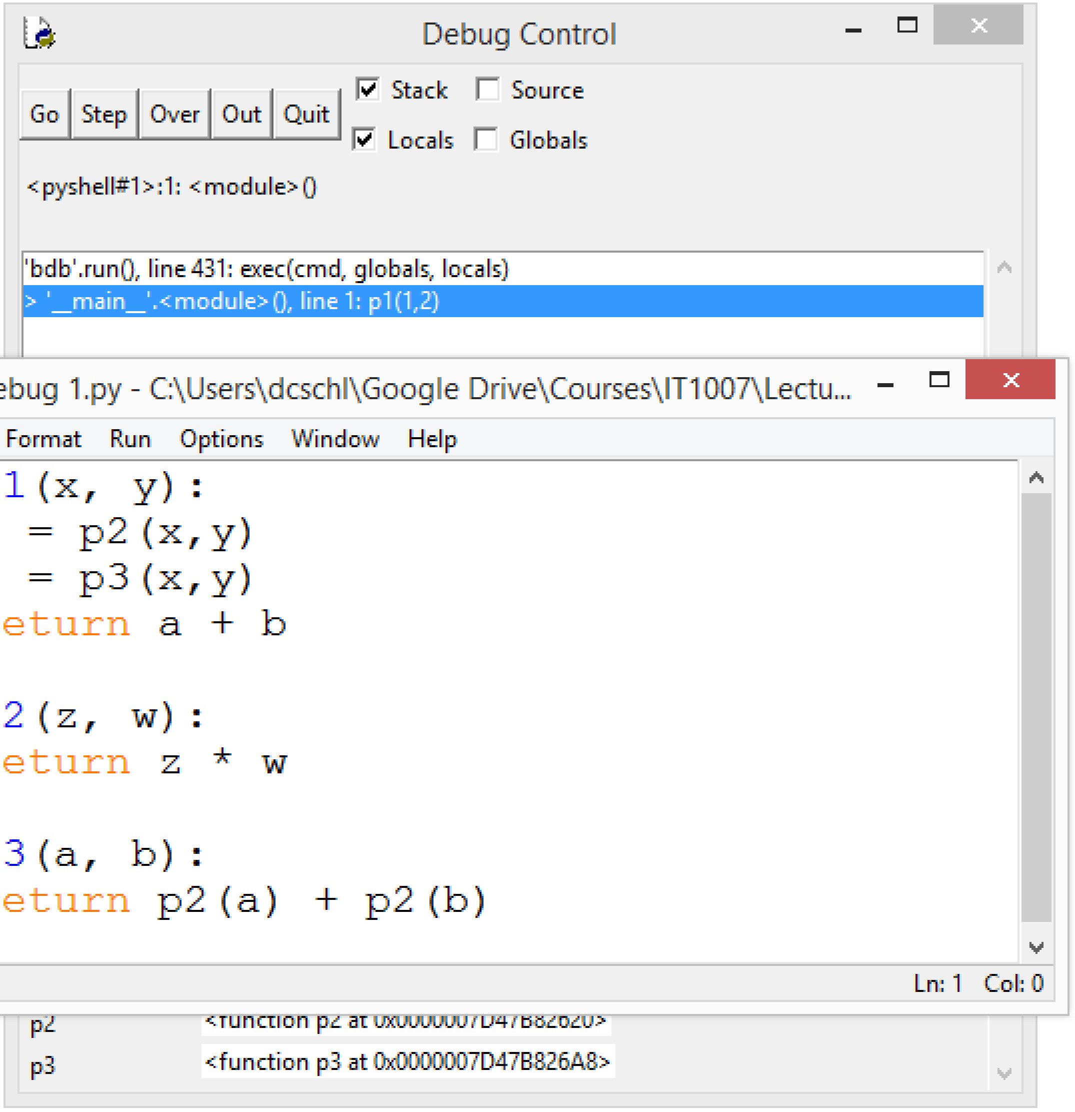
```
def p1(x, y):
```

Using the IDLE Debugger

- Go
  - Clicking this will run the program until the next break point is reached. You can insert break points in your code by right clicking and selecting Set Breakpoint. Lines that have break points set on them will be highlighted in yellow.
- Step
  - This executes the next statement. If the statement is a function call, it will enter the function and stop at the first line.
- Over
  - This executes the next statement just as Step does. But it does not enter into functions. Instead, it finishes executing any function in the statement and stops at the next statement in the same scope.
- Out
  - This exits the current function and stops in the caller of the current function.
  - After using Step to step into a function, you can use Out to quickly execute all the statements in the function and get back out to the outer function.
- Quit: This terminates execution.

Using the IDLE Debugger

- Currently in line 1
- Click "Step" goes to line 2



Stack [✓], Source [ ], Locals [✓], Globals [ ]

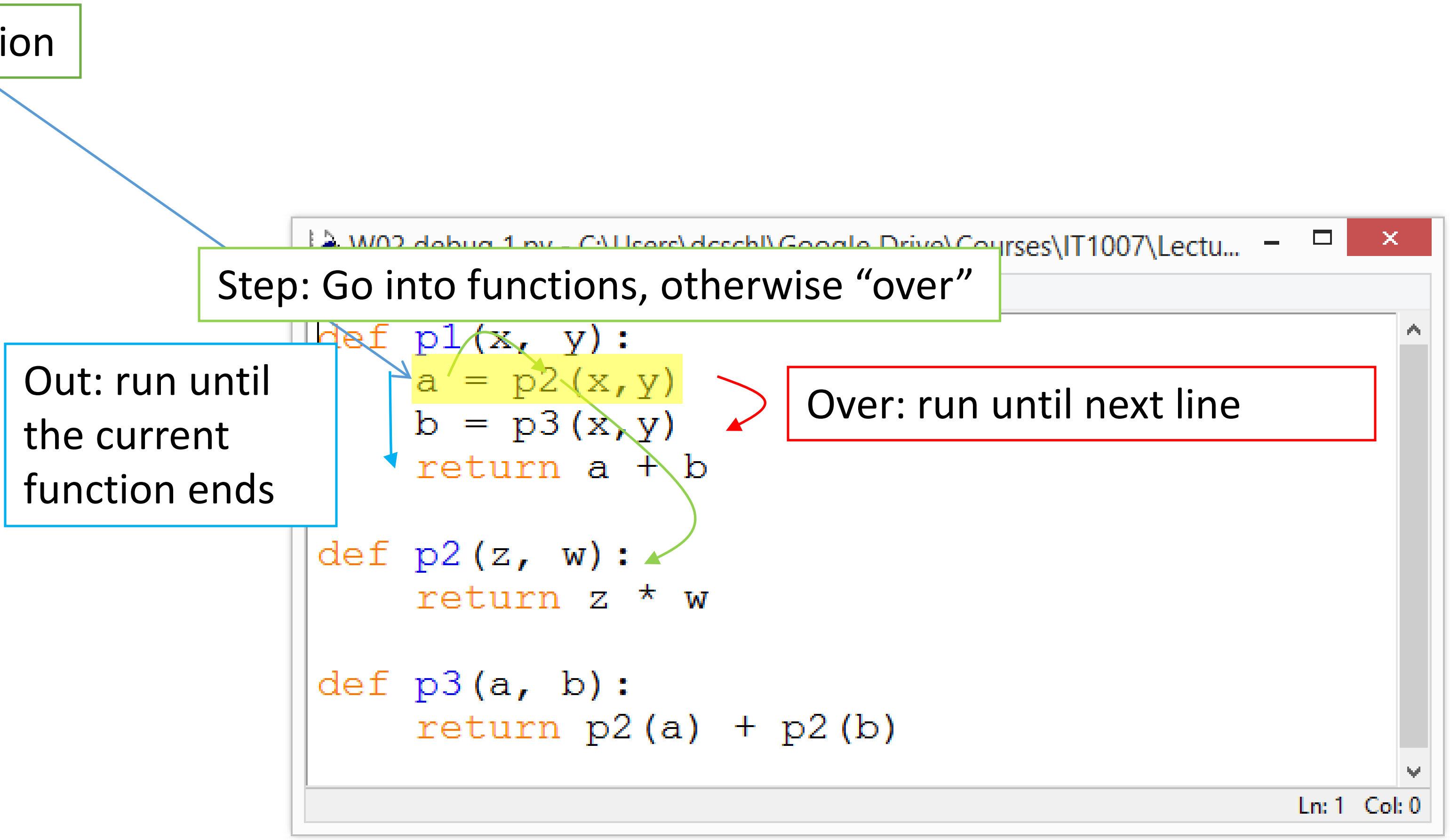
```
<pyshell#1>:1: <module>()
```

```
'bdb'.run(), line 431: exec(cmd, globals, locals)
> '__main__'.<module>(), line 1: p1(1,2)
```

```python
def p1(x, y):
    a = p2(x,y)
    b = p3(x,y)
    return a + b

def p2(z, w):
    return z * w

def p3(a, b):
    return p2(a) + p2(b)
```

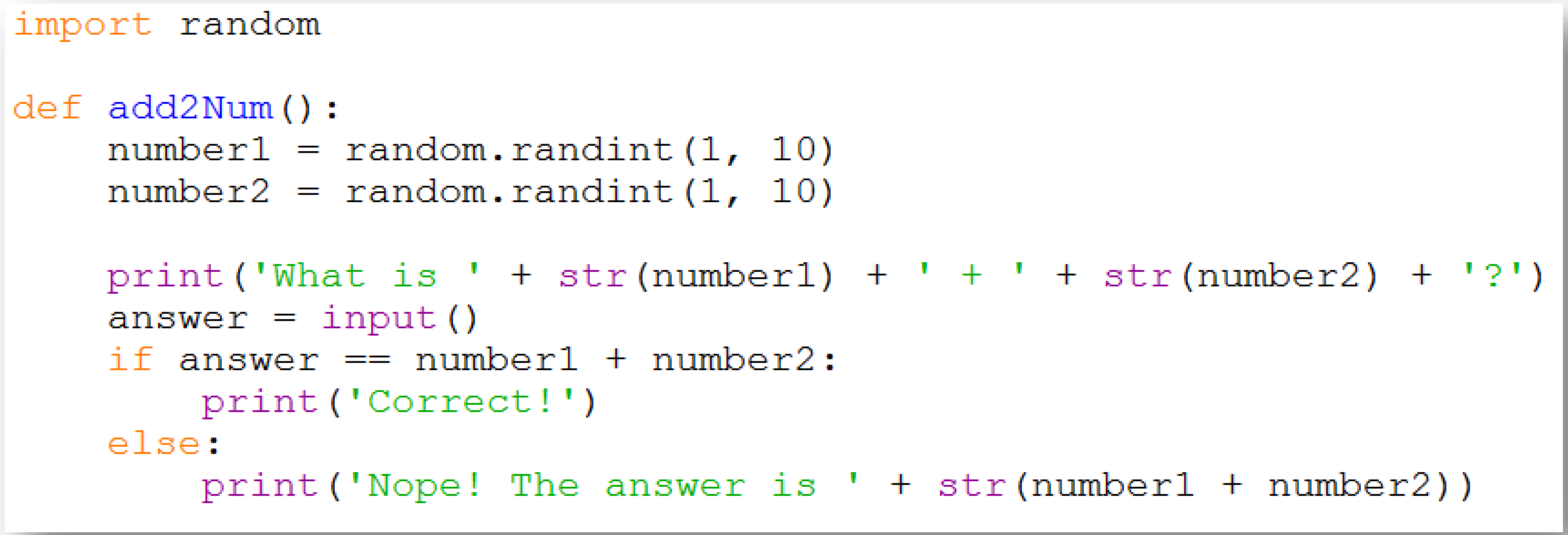
p2	<function p2 at 0x0000007D47B82620>
p3	<function p3 at 0x0000007D47B826A8>

Using the IDLE Debugger

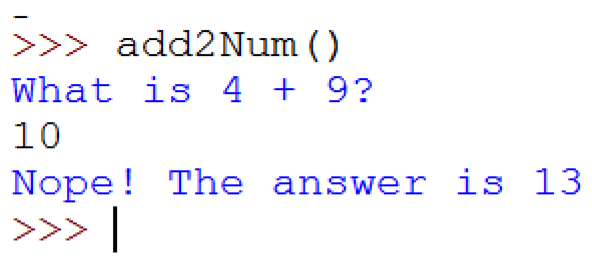

Current position
Step: Go into functions, otherwise "over"
Out: run until the current function ends
Over: run until next line
W02 debug 1.py - C:\Users\dcsschl\Google Drive\Courses\IT1007\Lectu...
Ln: 1 Col: 0

```
def p1(x, y):
    a = p2(x,y)
    b = p3(x,y)
    return a + b

def p2(z, w):
    return z * w

def p3(a, b):
    return p2(a) + p2(b)
```

More Debugging (BuggyAddNum)

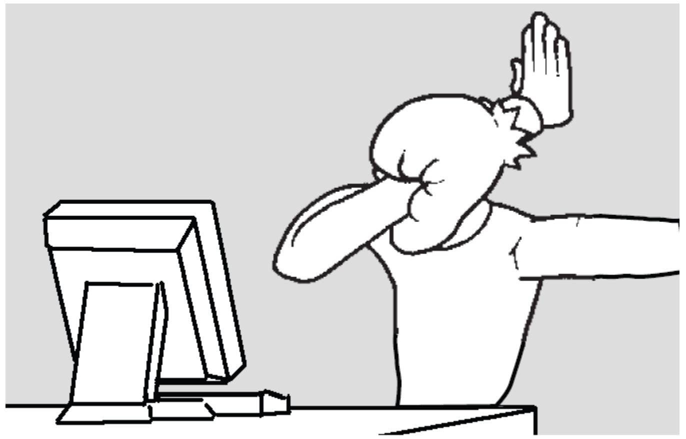


```
import random

def add2Num():
    number1 = random.randint(1, 10)
    number2 = random.randint(1, 10)

    print('What is ' + str(number1) + ' + ' + str(number2) + '?')
    answer = input()
    if answer == number1 + number2:
        print('Correct!')
    else:
        print('Nope! The answer is ' + str(number1 + number2))
```



```
>>> add2Num()
What is 4 + 9?
10
Nope! The answer is 13
>>> 
```

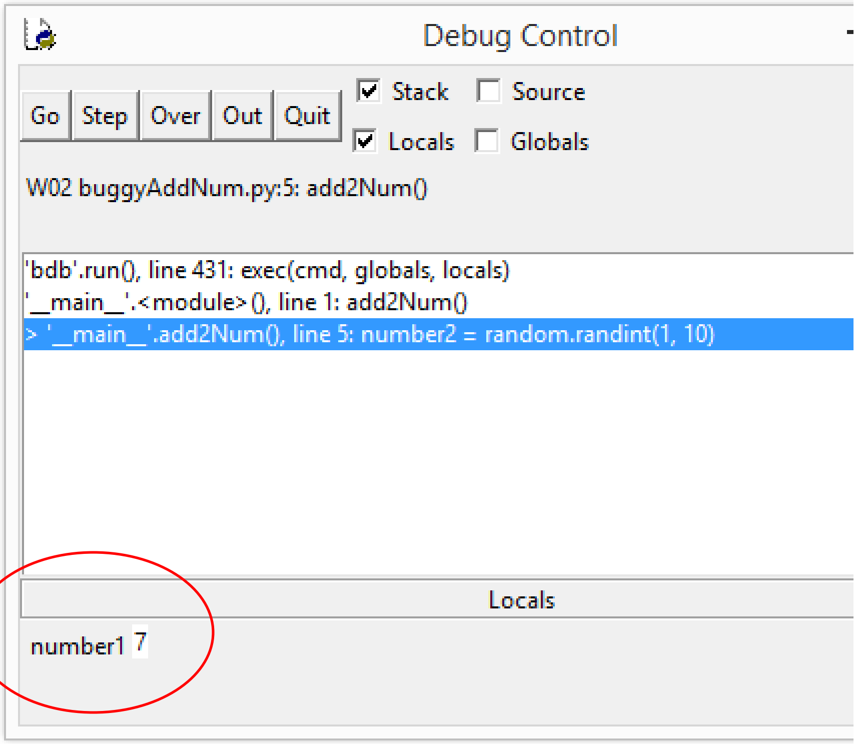
```
>>> add2Num()
What is 6 + 5?
11
Nope! The answer is 11
>>> add2Num()
What is 5 + 9?
14
Nope! The answer is 14
>>> |
```



```
import random

def add2Num():
    number1 = random.randint(1, 10)
    number2 = random.randint(1, 10)

    print('What is ' + str(number1) + ' + ' + str(number2) + '?')
    answer = input()
    if answer == number1 + number2:
        print('Correct!')
    else:
        print('Nope! The answer is ' + str(number1 + number2))
```

Turn on Debugger

- After a few steps

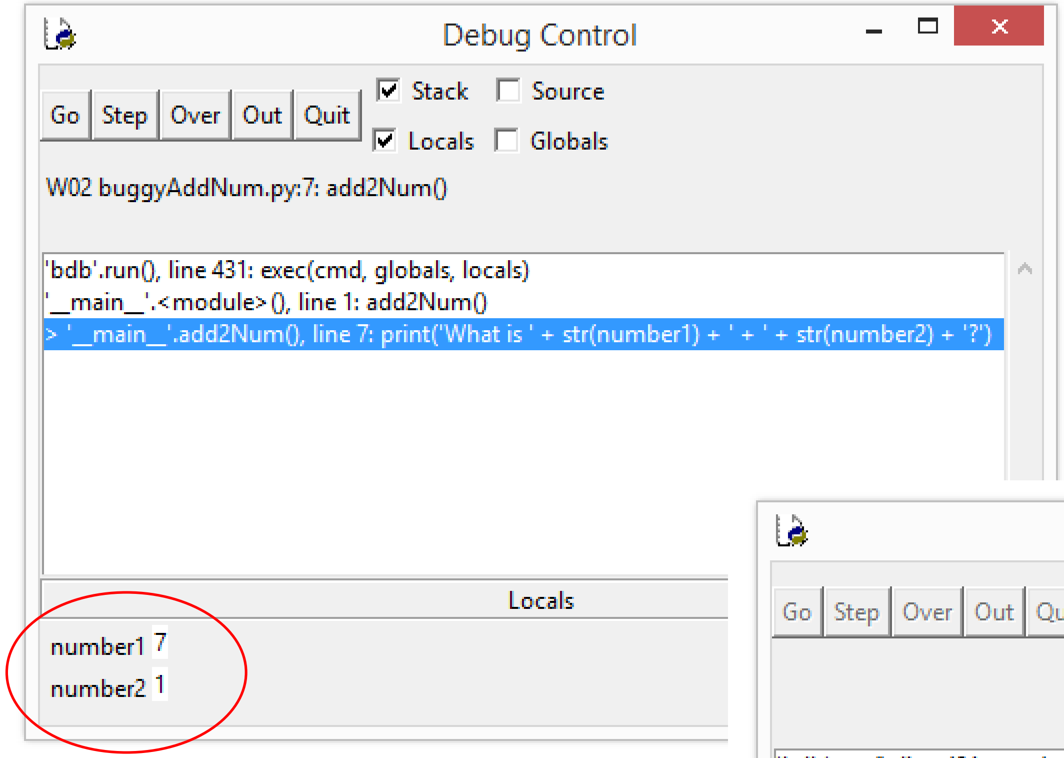


Stack [✓] Source [ ] Locals [✓] Globals [ ]

```
W02 buggyAddNum.py:5: add2Num()

'bdb'.run(), line 431: exec(cmd, globals, locals)
'__main__'.<module>(), line 1: add2Num()
> '__main__'.add2Num(), line 5: number2 = random.randint(1, 10)
```

Locals

```
number1 7
```

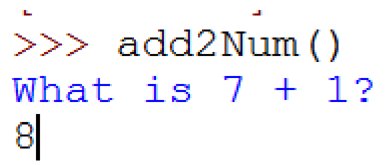


W02 buggyAddNum.py:7: add2Num()

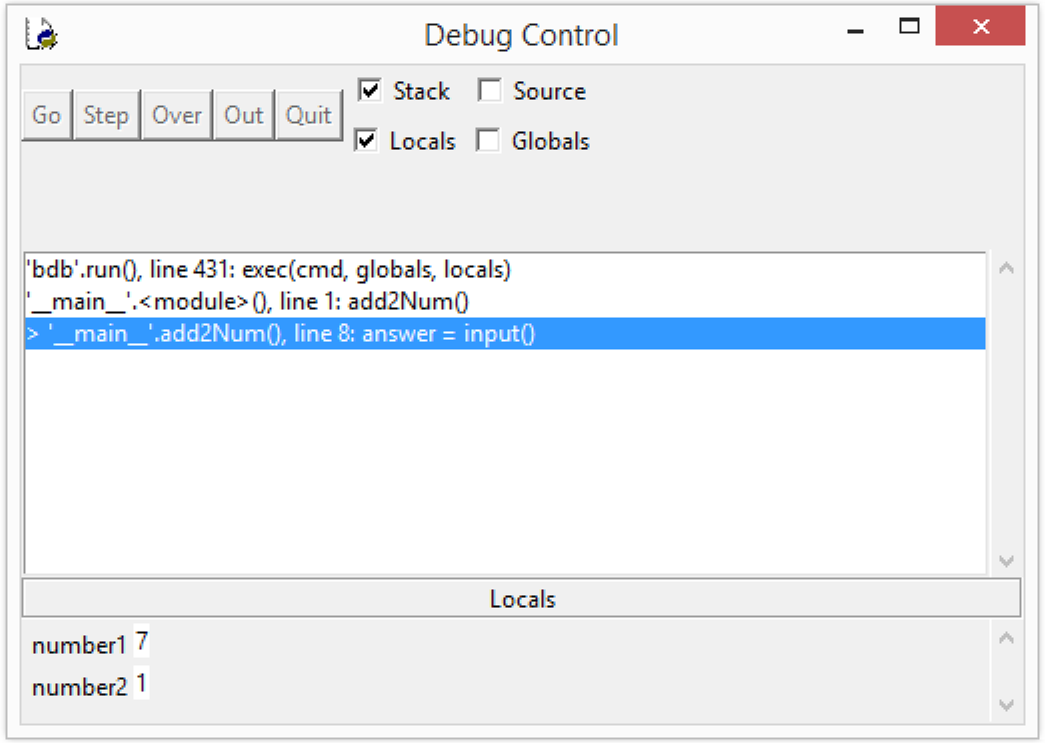
```
'bdb'.run(), line 431: exec(cmd, globals, locals)
'__main__'.<module>(), line 1: add2Num()
> '__main__'.add2Num(), line 7: print('What is ' + str(number1) + ' + ' + str(number2) + '?')
```

Locals

```
number1 7
number2 1
```



```
>>> add2Num()
What is 7 + 1?
8
```

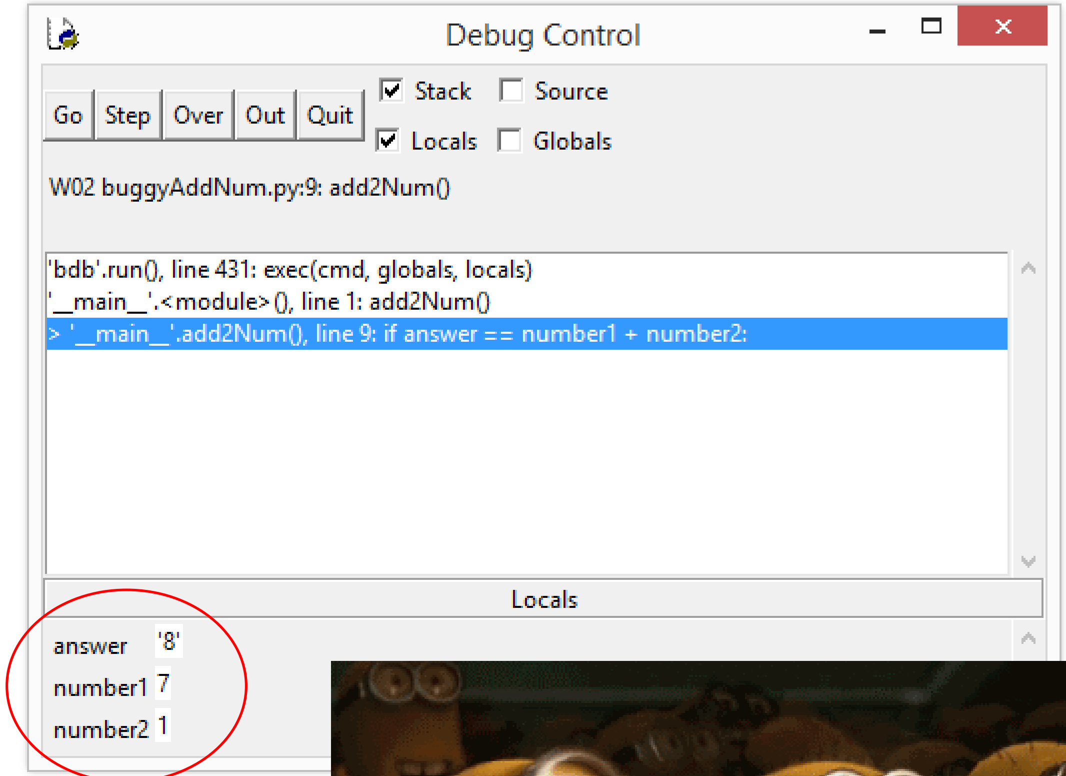


```
'bdb'.run(), line 431: exec(cmd, globals, locals)
'__main__'.<module>(), line 1: add2Num()
> '__main__'.add2Num(), line 8: answer = input()
```

Locals

```
number1 7
number2 1
```

Using the IDLE Debugger


Debug Control

Go Step Over Out Quit

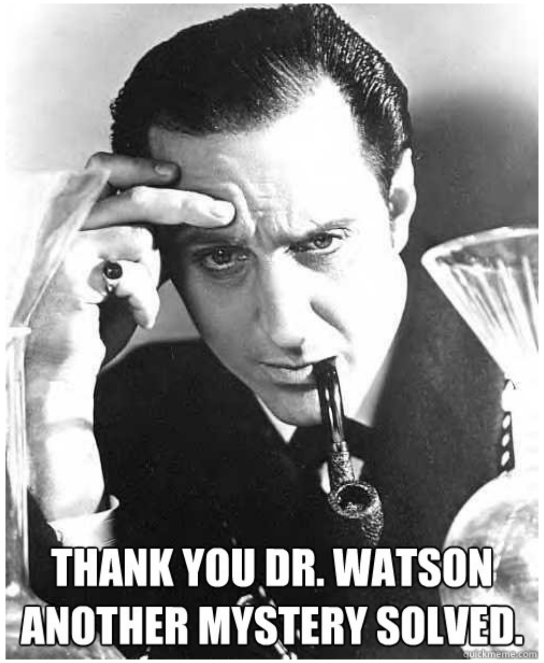
[✓ Stack] Source

[✓ Locals] Globals

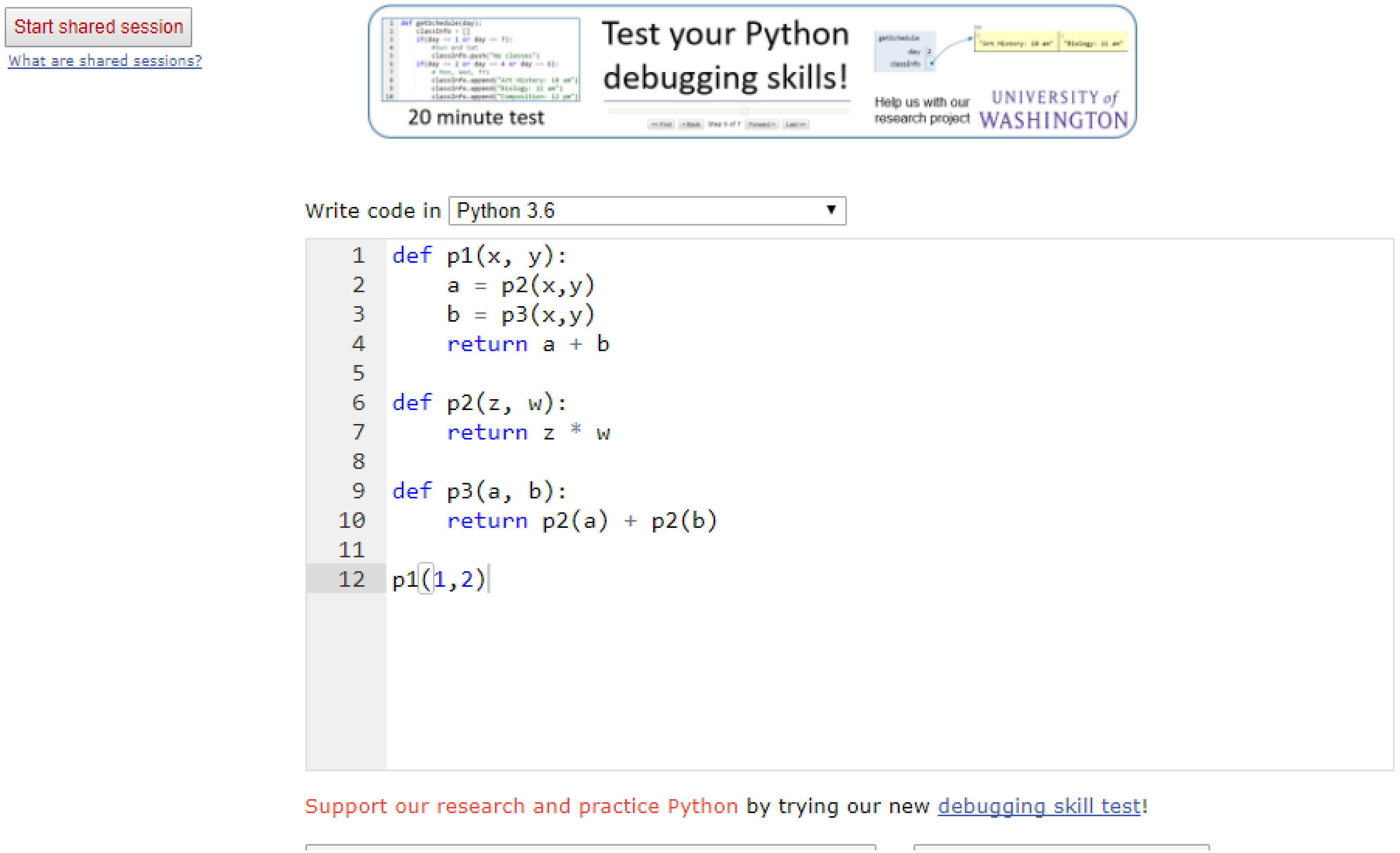
```
W02 buggyAddNum.py:9: add2Num()

'bdb'.run(), line 431: exec(cmd, globals, locals)
'__main__'.<module>(), line 1: add2Num()
> '__main__'.add2Num(), line 9: if answer == number1 + number2:
```

Locals

```
answer  '8'
number1 7
number2 1
```



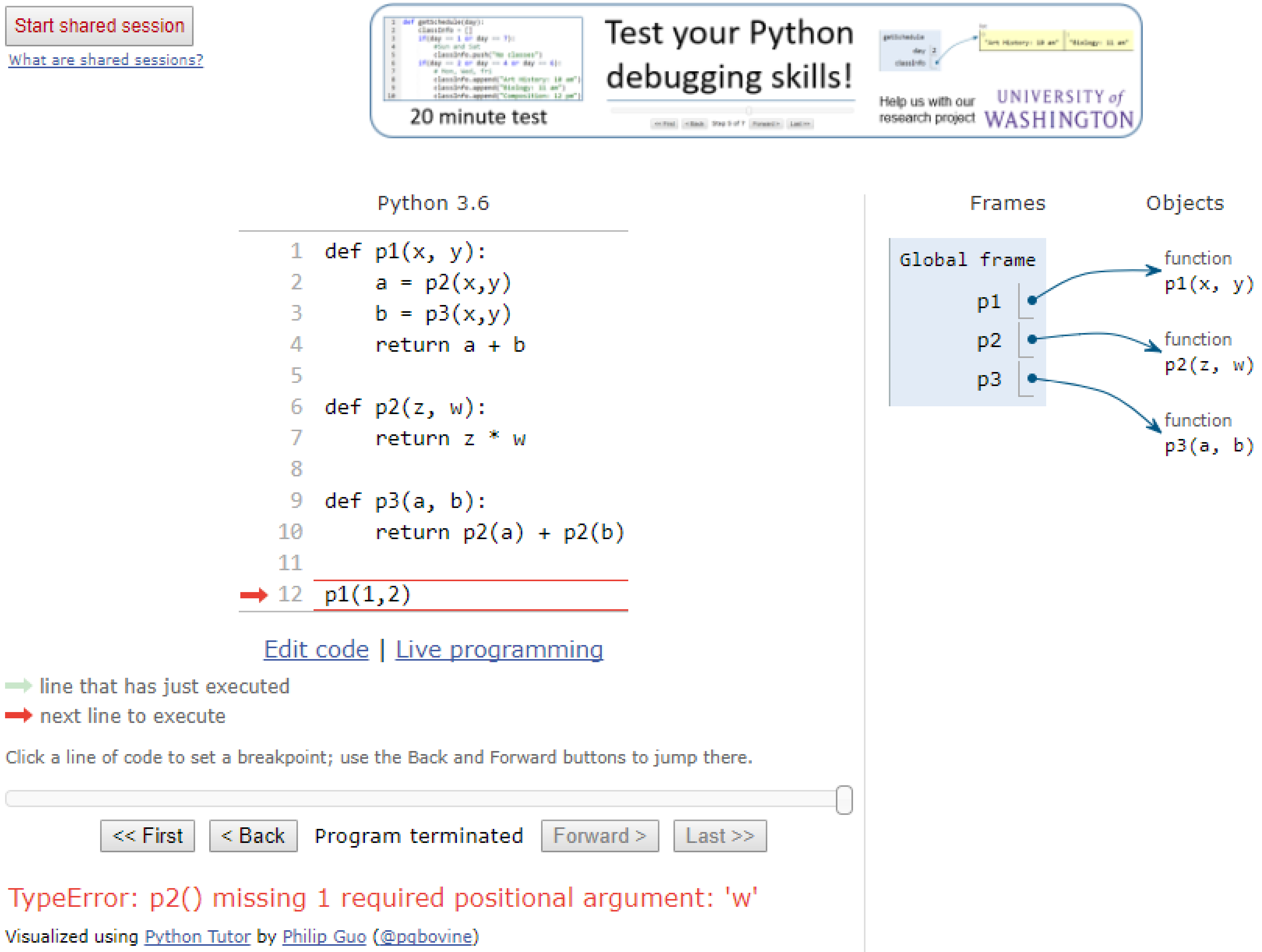
Another Debugger: pythontutor.com


Start shared session
What are shared sessions?
20 minute test
Test your Python debugging skills!
Help us with our research project
UNIVERSITY of WASHINGTON

Write code in Python 3.6

```
def p1(x, y):
    a = p2(x,y)
    b = p3(x,y)
    return a + b

def p2(z, w):
    return z * w

def p3(a, b):
    return p2(a) + p2(b)

p1(1,2)
```

Support our research and practice Python by trying our new debugging skill test!


Start shared session
What are shared sessions?
20 minute test
Test your Python debugging skills!
Help us with our research project
UNIVERSITY of WASHINGTON

Python 3.6

```
def p1(x, y):
    a = p2(x,y)
    b = p3(x,y)
    return a + b

def p2(z, w):
    return z * w

def p3(a, b):
    return p2(a) + p2(b)

p1(1,2)
```

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.

```
TypeError: p2() missing 1 required positional argument: 'w'
```

Visualized using Python Tutor by Philip Guo (@pgbovine)

Common Types of Errors

Common Types of Errors

- Omitting return statement

```
def square(x):
    x * x       # no error msg!
```

- Incompatible types

```
x = 5
def square(x):
    return x * x
x + square
```

- Incorrect # args

```
square(3,5)
```

Common Types of Errors

- Syntax
```
def proc(100)
    do_stuff()
        more()
```
- Arithmetic error
```
x = 3
y = 0
x/y
```
- Undeclared variables
```
x = 2
x + k
```

Common Types of Errors

- Infinite loop (from bad inputs)

```
def factorial(n):
  if n == 0:
    return 1
  else:
    return n * factorial(n-1)


fact(2.1)
fact(-1)
```

Common Types of Errors

- Infinite loop (forgot to decrement)

```
def fact_iter(n):
    counter, result = n, 1
    while counter!= 0:
        result *= counter
    return result
```

Common Types of Errors

- Numerical imprecision

```
def foo(n):
    counter, result = 0,0
    while counter != n:
        result += counter
        counter += 0.1
    return result

foo(5)
```

`counter` never exactly equals *n*

Common Types of Errors

- Logic

```
def fib(n):
  if n < 2:
    return n
  else:
    return fib(n-1) + fib(n-1)
```

How to debug?

- Think like a detective
  - Look at the clues: error messages, variable values.
  - Eliminate the impossible.
  - Run the program again with different inputs.
    - Does the same error occur again?

How to debug?

- Work backwards
  - From current sub-problem backwards in time
- Use a debugger
  - IDLE has a simple debugger
  - Overkill for our class
- Trace a function
- Display variable values

Displaying variables

```python
debug_printing = True
def debug_print(msg):
  if debug_printing:
    print(msg)

def foo(n):
  counter, result = 0,0
  while(counter != n):
    debug_print(f'{counter}, {n}, {result}')
    counter, result = counter + 0.1, result + counter
  return result
```

Example

```
def fib(n):
  debug_print(f'n:{n}')
  if n < 2:
    return n
  else:
    return fib(n-1) + fib(n-1)
```

Other tips

- State assumptions clearly.

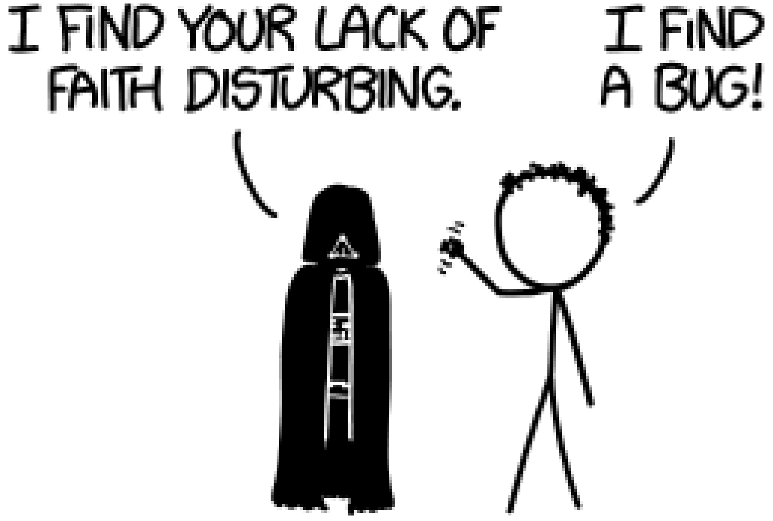
```
def factorial(n): # n integer >= 0
  if n == 0:
    return 1
  else:
    return n * factorial(n-1)
```

- Test each function before you proceed to the next.
  - Remember to test boundary cases

Summary

- Compound data helps us to reason at a higher conceptual level.
- Abstraction barriers separate usage of a compound data from its implementation.
- Only functions at the interface should be used.
- We can choose between different implementations as long as contract is fulfilled.

Debugging is an Art



Maths vs CS vs Engineering

- Three good friends, an engineer, a mathematician and a computer scientist, are driving on a highway that is in the middle of no where. Suddenly one of the tires went flat and they have no spare tire.

Maths vs CS vs Engineering

- Engineer
    - “Let’s use bubble gum to patch the tire and use the strew to inflate it again”
- Mathematician
    - “I can prove that there is a good tire exists in somewhere this continent”
- Computer Scientist
    - “Let’s remove the tire, put it back, and see if it can fix itself again”