

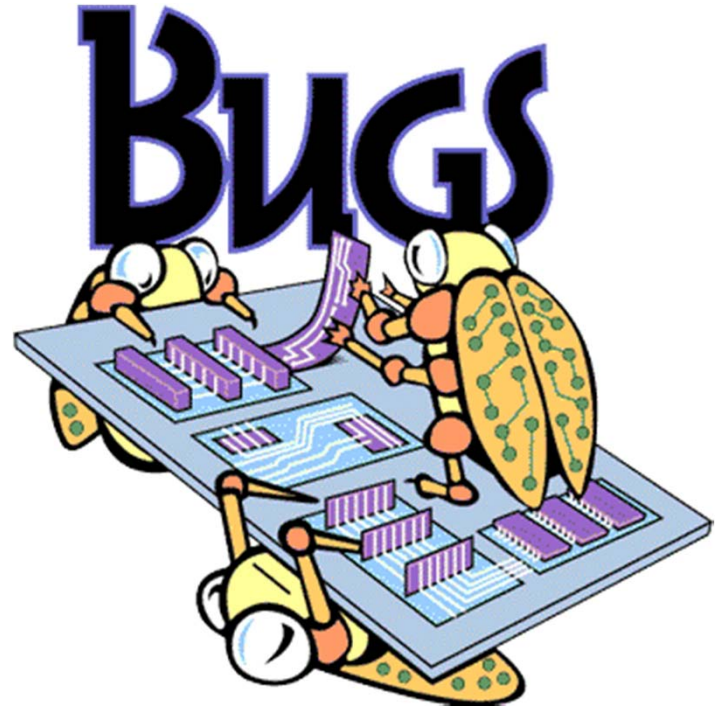


IT1007 INTRODUCTION TO PROGRAMMING

Python and C

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.



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
1300 (032) MP-MC 2.130476415 9.037 846 995 check
033 PRO 2 2.130476415 4.615925059(-2)
check 2.130676415

Relays 6-2 in 033 failed special speed test
in relay " 11.000 test.

Relays changed

1100 Started Cosine Tape (Sine check)
1525 Started Multi-Adder Test.

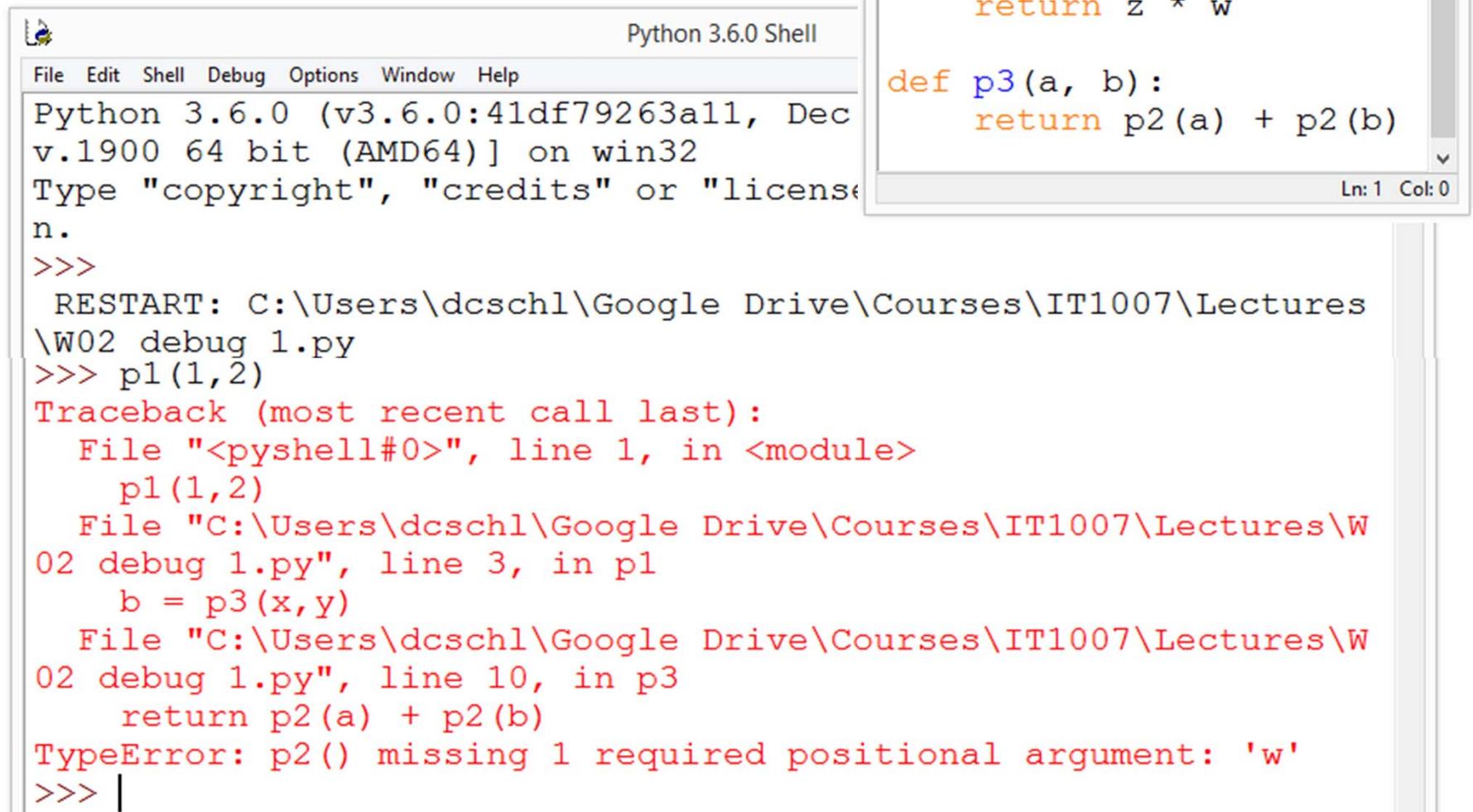
1545  Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.

1630 Antan started.
1700 closed down.

Relay 3145
Relay 3370

W02 debug1.py



The image shows two windows from a Windows environment. The background window is the 'Python 3.6.0 Shell' with a menu bar (File, Edit, Shell, Debug, Options, Window, Help). It displays the Python version and architecture, followed by a traceback error. The foreground window is a code editor titled 'W02 debug 1.py - C:\Users\dcsc...' with a menu bar (File, Edit, Format, Run, Options, Window, Help). It contains the source code for three functions: p1, p2, and p3.

```
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\IT1007\Lectures
\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'
>>> |
```

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

```
W02 debug 1.py - C:\Users\dcshchl\Google Drive\Courses\IT1007\Lectures\W02 debug 1.py
File Edit Format Run Options Window Help
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)
```

Ln: 1 Col: 0

Traceback (most recent call last):

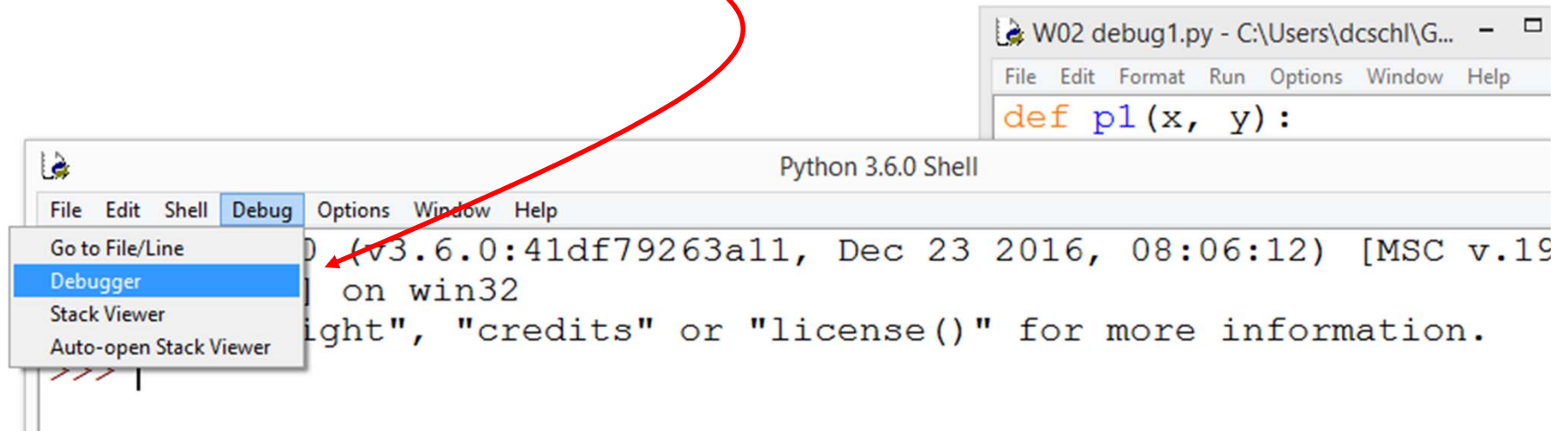
- 1 File "<pyshell#0>", line 1, in <module>
p1(1,2)
- 2 File "C:\Users\dcshchl\Google Drive\Courses\IT1007\Lectures\W02 debug 1.py",
line 3, in p1
b = p3(x,y)
- 3 File "C:\Users\dcshchl\Google Drive\Courses\IT1007\Lectures\W02 debug 1.py",
line 10, in p3
return p2(a) + p2(b)
- 4 TypeError: p2() missing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

THE IDLE DEBUGGER

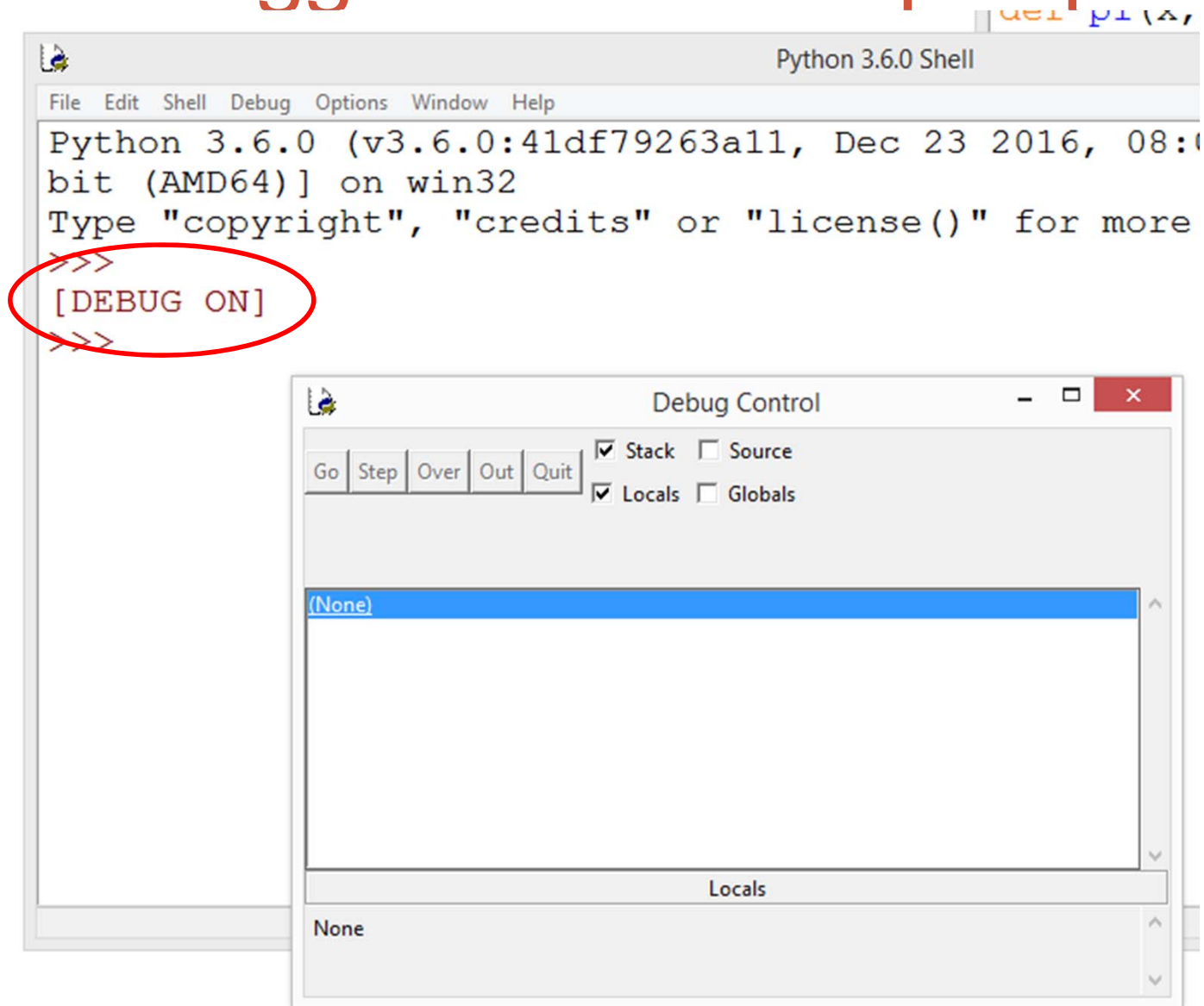


Using the IDLE Debugger

- Load in your source code
- Turn on the debugger by

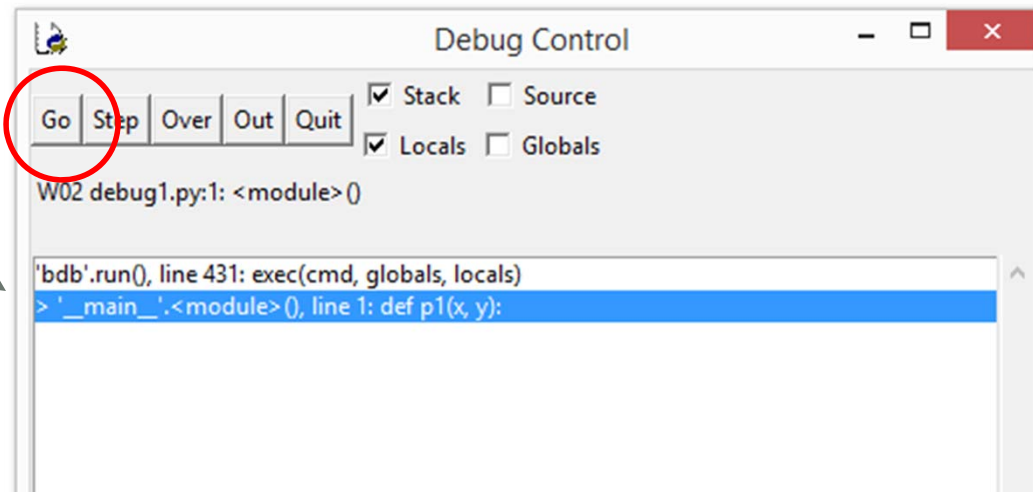


The Debugger Window Pops up



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 “running” the code, we just define the three functions

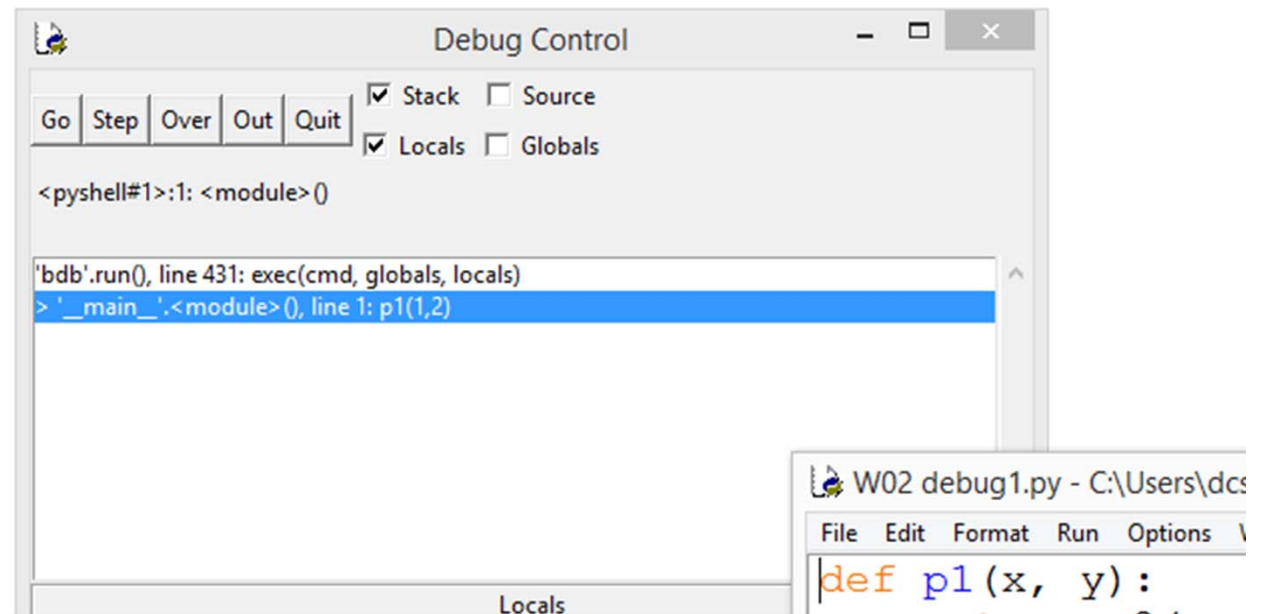


Using the IDLE Debugger

- 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 get an error like the last time

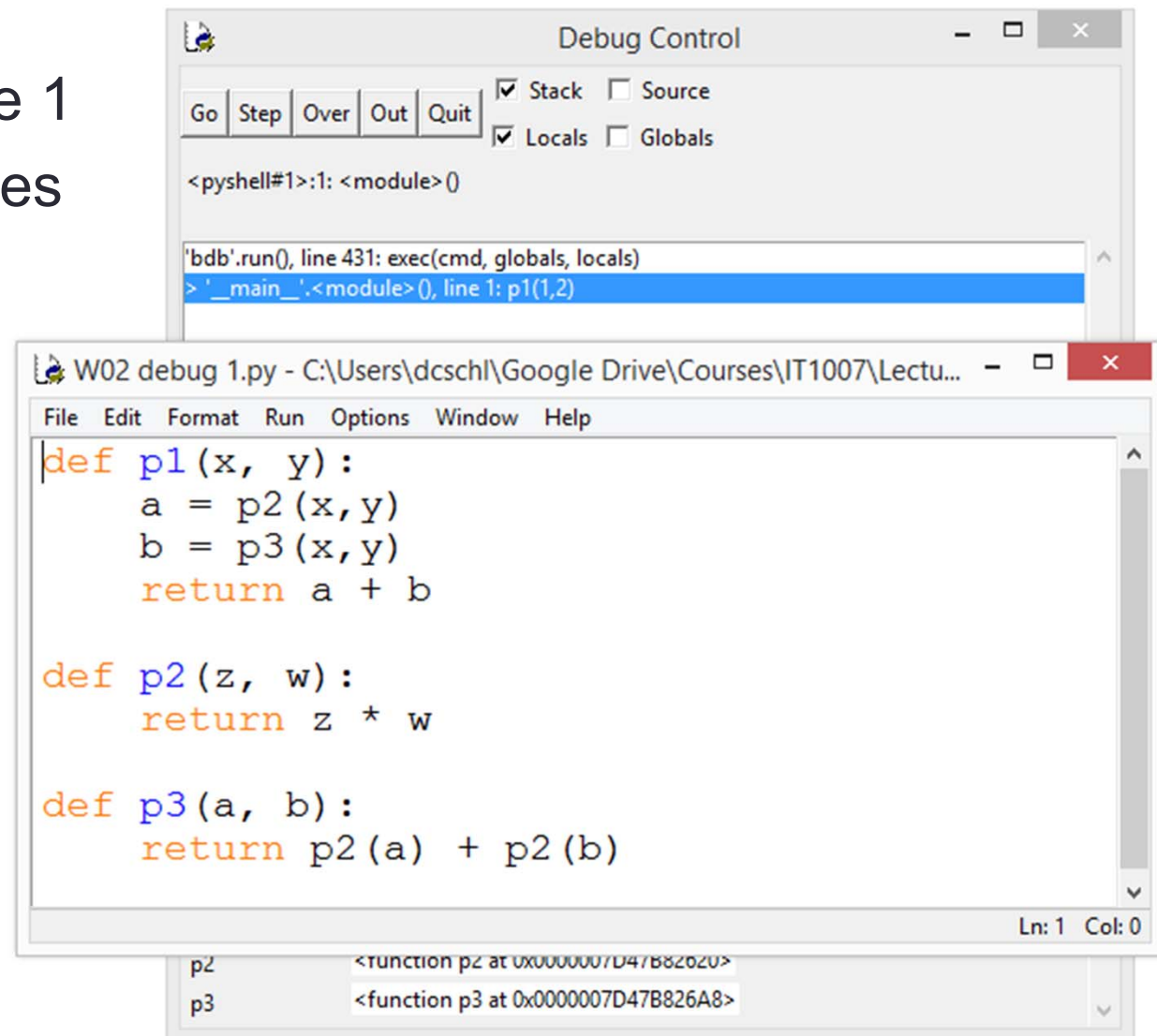


Using the IDLE Debugger

- 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



Using the IDLE Debugger

Current
position

Step: Go into functions, otherwise "over"

Out: run until
the current
function ends

Over: run until next line

```
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)
```

Ln: 1 Col: 0

More Debugging (BuggyAddNum)

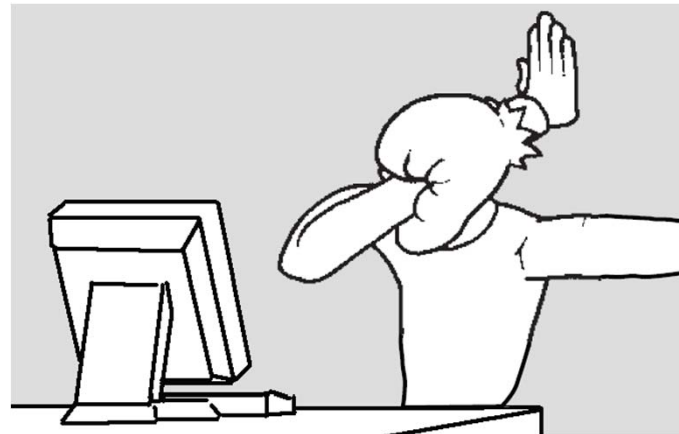
```
import random

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  
>>> |
```

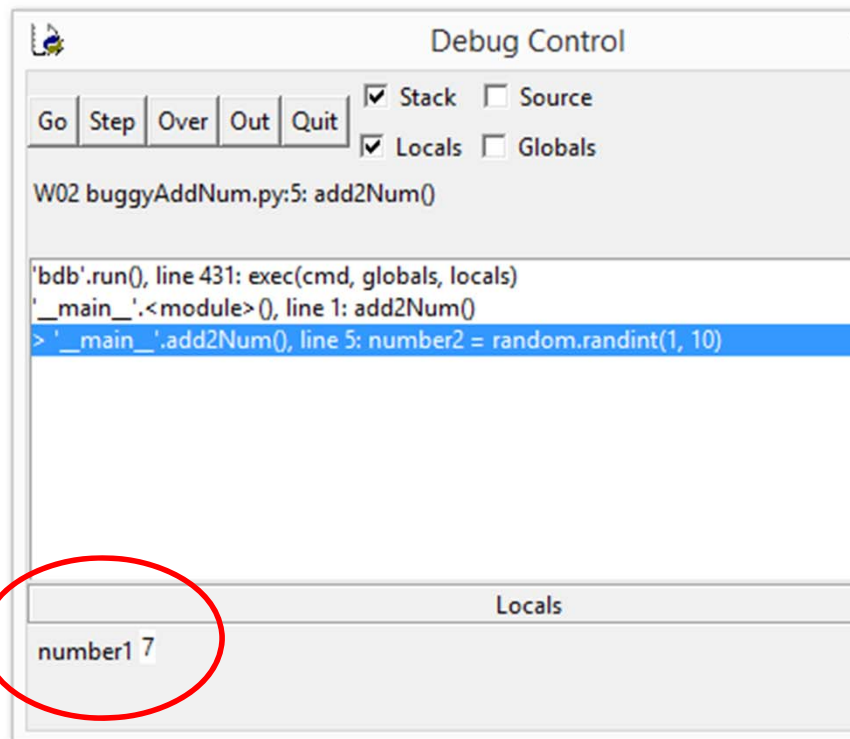


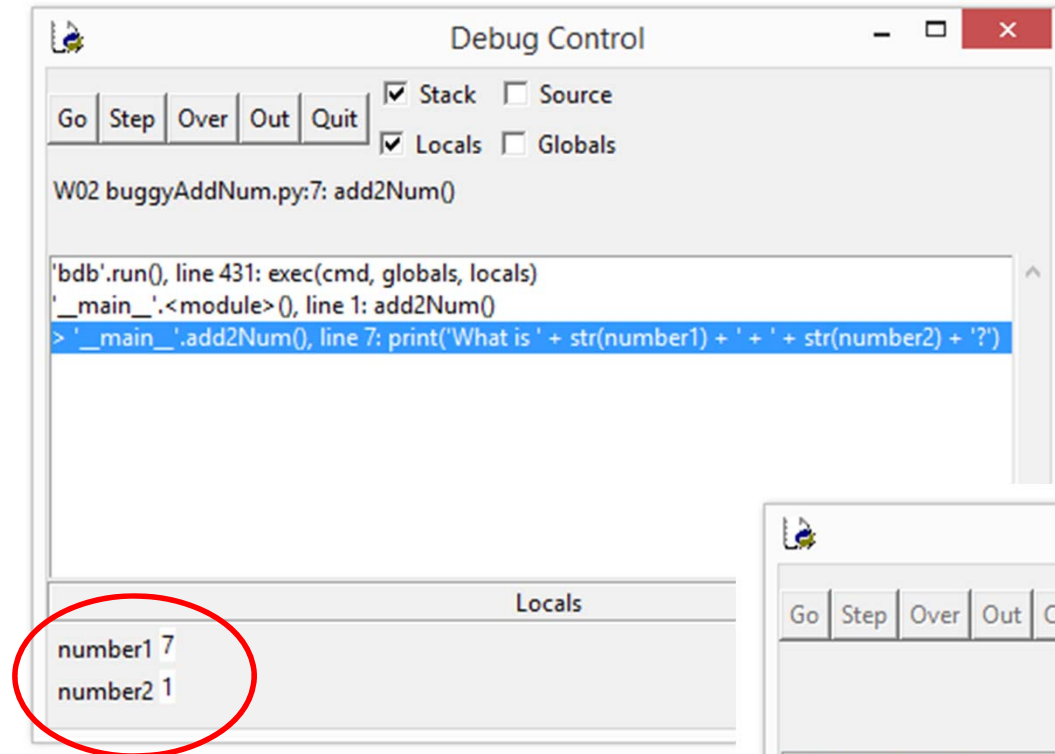
```
import random
```

```
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))
```

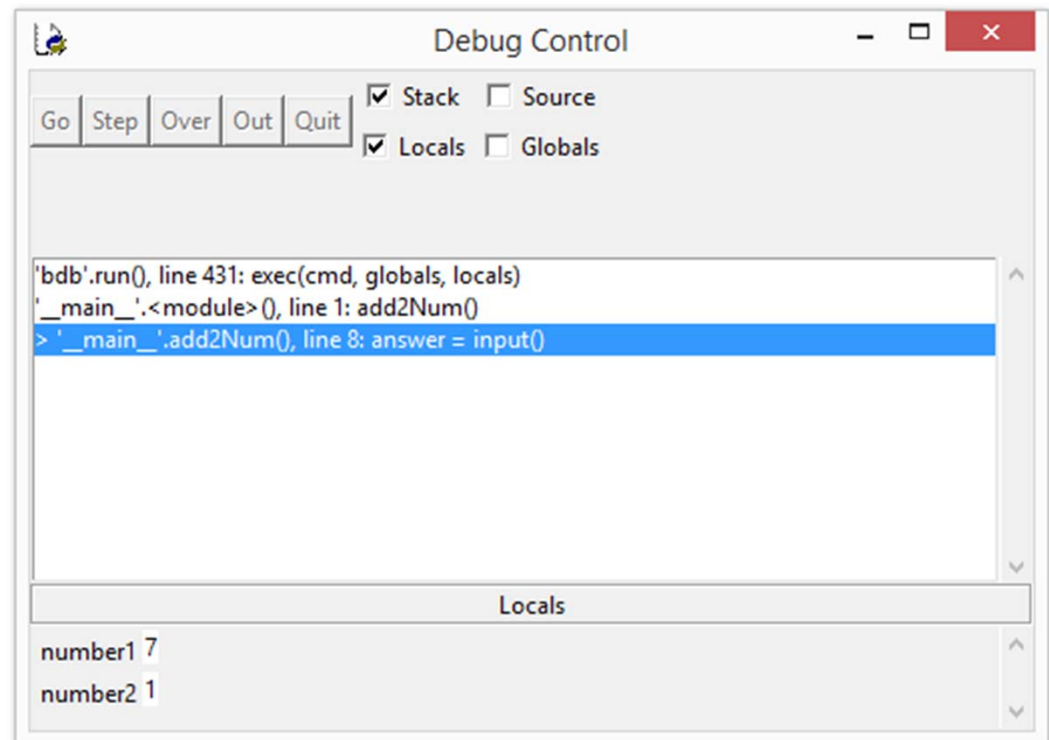
Turn on Debugger

- After a few steps

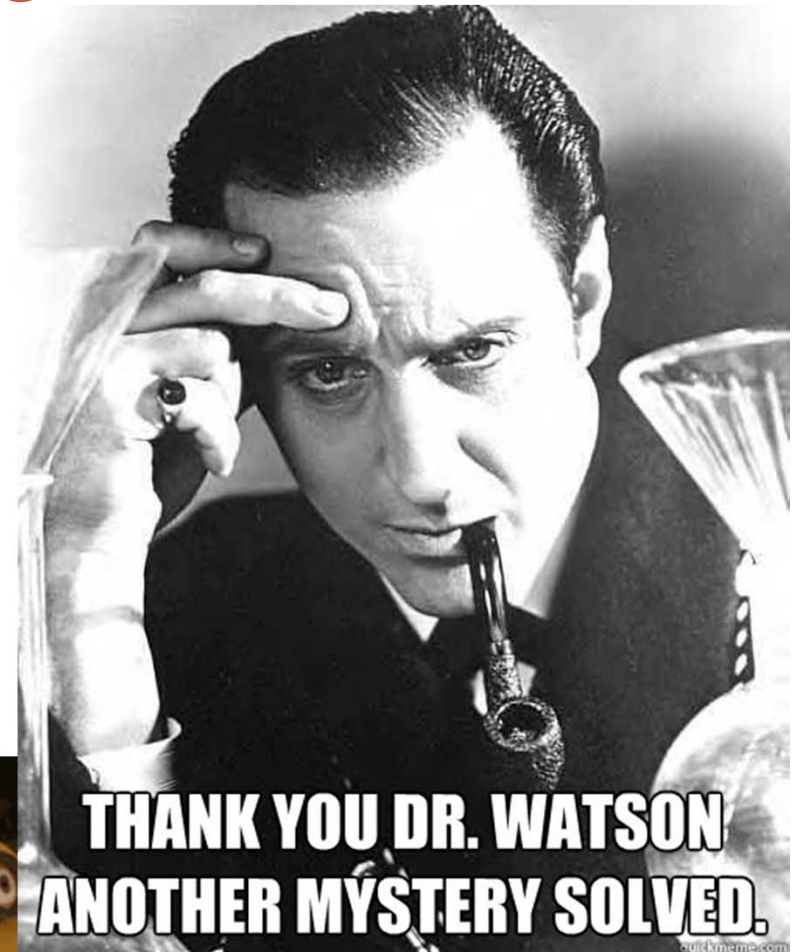
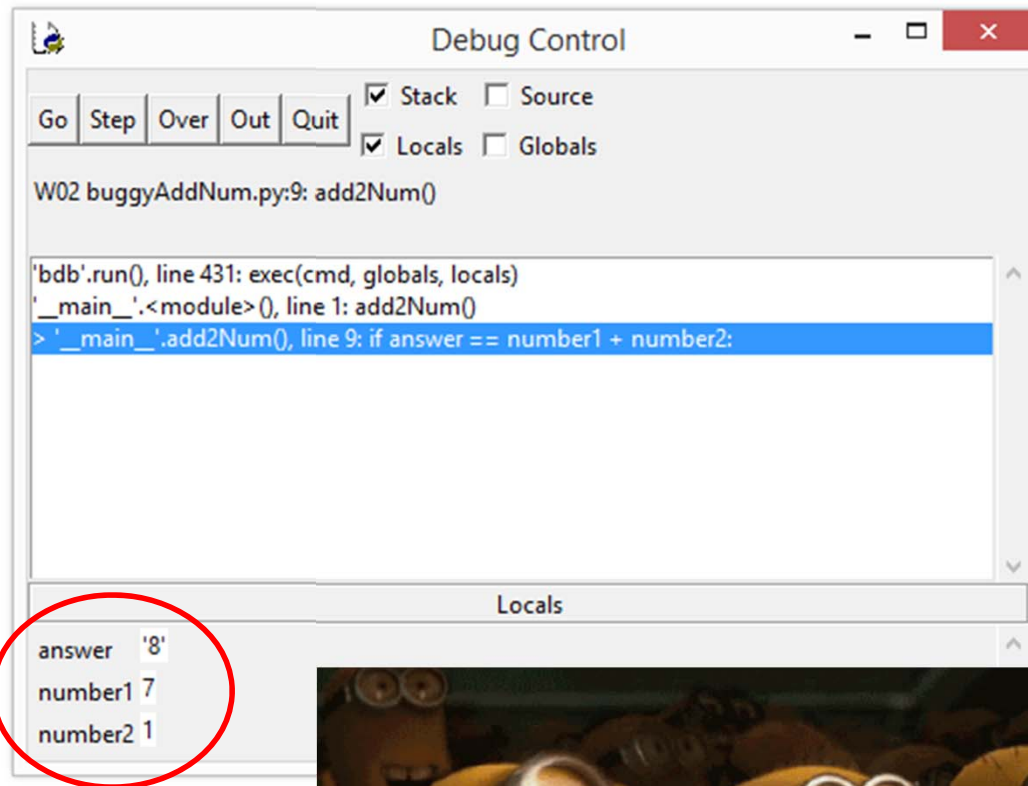




```
>>> add2Num()  
What is 7 + 1?  
8|
```



Using the IDLE Debugger



Another Debugger: Pythontutor.com

Start shared session

[What are shared sessions?](#)

```
def getSchedule(day):
    classInfo = []
    if day == 1 or day == 2:
        #Mon and Tue
        classInfo.append("lec class")
    if day == 2 or day == 4 or day == 6:
        #Wed, Fri, Sat
        classInfo.append("art history: 10 am")
        classInfo.append("biology: 11 am")
        classInfo.append("composition: 12 pm")
```

20 minute test

Test your Python debugging skills!

Help us with our research project

UNIVERSITY of WASHINGTON

getSchedule

day: 2

classInfo

lec class

art history: 10 am

biology: 11 am

Write code in Python 3.6

```
1 def p1(x, y):
2     a = p2(x,y)
3     b = p3(x,y)
4     return a + b
5
6 def p2(z, w):
7     return z * w
8
9 def p3(a, b):
10    return p2(a) + p2(b)
11
12 p1(1,2)
```

Support our research and practice Python by trying our new [debugging skill test!](#)

Start shared session

[What are shared sessions?](#)

```
1 def get_historical_data():
2     classInfo = {}
3     if day == 1 or day == 2:
4         # Run and get
5         classInfo.push("no classes")
6     if day == 3 or day == 4 or day == 5:
7         # Run, wait, try
8         classInfo.append("art history: 10 am")
9         classInfo.append("biology: 11 am")
10    return classInfo
```

20 minute test

Test your Python debugging skills!

Control Back One Step Forward Save

get_historical_data
day: 3
classInfo

for
"art history: 10 am" "biology: 11 am"

Help us with our research project

UNIVERSITY of WASHINGTON

Python 3.6

```
1 def p1(x, y):
2     a = p2(x,y)
3     b = p3(x,y)
4     return a + b
5
6 def p2(z, w):
7     return z * w
8
9 def p3(a, b):
10    return p2(a) + p2(b)
11
12 p1(1,2)
```

[Edit code](#) | [Live programming](#)

→ line that has just executed

→ next line to execute

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

<< First

< Back

Program terminated

Forward >

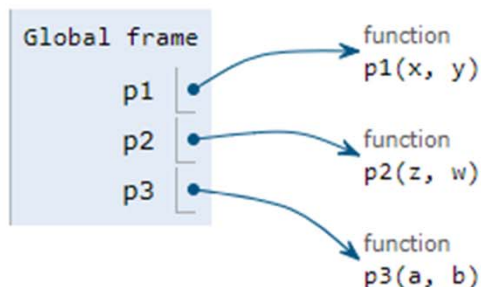
Last >>

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

Visualized using [Python Tutor](#) by [Philip Guo \(@pqbovine\)](#)

Frames

Objects



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”

