

# Debugging

# Debugging “Tools”

- IDLE Debuggers
  - If you are using it
  - Introduced in the lecture
- [pythontutor.com](http://pythontutor.com)
  - Good visualization
  - But limited to certain few steps
- `print()`

# Flip Coin Experiment

- I will flip a coin 1000 times and FOR EACH FLIP
  - I will record how many times I had flipped
  - If it is a head, I will record the number of heads



What you  
repeat for  
EACH time



# Can you see any bug?

```
import random

def flipCoins():
    print('I will flip a coin 1000 times. ')
    print('Guess how many times it will come up heads. ')
    flips = 0
    heads = 0
    while flips < 1000:
        if random.randint(0, 1) == 1:
            heads = heads + 1
            flips = flips + 1
        if flips == 500:
            print('Half way done, and heads has come up ' + str(heads) + ' times.')
    print()
    print('Out of 1000 coin tosses, heads came up ' + str(heads) + ' times!')

flipCoins()
```

# Running

```
I will flip a coin 1000 times.  
Guess how many times it will come up heads.  
Half way done, and heads has come up 500 times.  
Half way done, and heads has come up 500 times.  
  
Out of 1000 coin tosses, heads came up 1000 times!  
>>>
```

- Am I that lucky?!

# Can you see any bug?

```
import random

def flipCoins():
    print('I will flip a coin 1000 times. ')
    print('Guess how many times it will come up heads. ')
    flips = 0
    heads = 0
    while flips < 1000:
        if random.randint(0, 1) == 1:
            heads = heads + 1
            flips = flips + 1
        if flips == 500:
            print('Half way done, and heads has come up ' + str(heads) + ' times.')
    print()
    print('Out of 1000 coin tosses, heads came up ' + str(heads) + ' times!')

flipCoins()
```

Since I got 1000 heads out of 1000 flips, must be something wrong with the counting?

```
import random
```

```
def flipCoins():
```

```
    print('I will flip a coin 1000 times. ')
```

```
    print('Guess how many times it will come up heads. ')
```

```
    flips = 0
```

```
    heads = 0
```

```
    while flips < 1000:
```

```
        print("Flip:"+str(flips)+"      Head:"+str(heads))
```

```
        if random.randint(0, 1) == 1:
```

```
            heads = heads + 1
```

```
            flips = flips + 1
```

```
        if flips == 500:
```

```
            print('Half way done, and heads has come up ' + str(heads) + ' times.')
```

```
    print()
```

```
    print('Out of 1000 coin tosses, heads came up ' + str(heads) + ' times!')
```

```
flipCoins()
```

# How many things are wrong?

Flip:280	Head:280
Flip:281	Head:281
Flip:281	Head:281
Flip:281	Head:281
Flip:282	Head:282
Flip:282	Head:282
Flip:282	Head:282
Flip:282	Head:282
Flip:282	Head:282
Flip:282	Head:282
Flip:282	Head:282
Flip:282	Head:282
Flip:282	Head:282
Flip:283	Head:283
Flip:283	Head:283
Flip:283	Head:283
Flip:283	Head:283
Flip:283	Head:283

- #flips should be incremented every time, right?
- How come the numbers of flips and heads increment at the same time ONLY?



```
import random
```

```
def flipCoins():  
    print('I will flip a coin 1000 times. ')  
    print('Guess how many times it will come up heads. ')  
    flips = 0  
    heads = 0  
    while flips < 1000:  
        print("Flip:"+str(flips)+"          Head:"+str(heads))  
        if random.randint(0, 1) == 1:  
            heads = heads + 1  
            flips = flips + 1  
        if flips == 500:  
            print('Half way done, and heads has come up ' + str(heads) + ' times.')    print()  
    print('Out of 1000 coin tosses, heads came up ' + str(heads) + ' times!')
```

What should be the correct way?

```
flipCoins()
```

Try downloading the file and correct it yourself.

# You can use Pythontutor.com also

These Python Tutor users are asking for help right now. Please volunteer to help!

- user\_e21 from Atlanta, Georgia, US needs help with Python3 - [click to help](#) (active a minute ago, requested 7 hours ago)
- user\_ac6 from Roselle, New Jersey, US needs help with Python3 - [click to help](#) (active a minute ago, requested 4 hours ago)

- Paste your code and press “Visualize Execution”

Write code in Python 3.6

```
1 import random
2
3 def flipCoins():
4     print('I will flip a coin 1000 times. ')
5     print('Guess how many times it will come up heads. ')
6     flips = 0
7     heads = 0
8     while flips < 1000:
9         print("Flip:"+str(flips)+"      Head:"+str(heads))
10        if random.randint(0, 1) == 1:
11            heads = heads + 1
12            flips = flips + 1
13        if flips == 500:
14            print('Half way done, and heads has come up ' + str(heads) +
15            print()
16            print('Out of 1000 coin tosses, heads came up ' + str(heads) + ' times:
17
18
19 flipCoins()
```

Help improve this tool by completing a [short user survey](#).

Visualize Execution

Live Programming Mode

Advanced instructions: [setting breakpoints](#) | [hiding variables](#) | [live programming](#)

hide exited frames [default] ▼ inline primitives but don't nest objects [default] ▼  
draw pointers as arrows [default] ▼

Get live help!

Start private chat

(warning: chat service  
may crash at any time)

These Python Tutor users are asking for help right now. Please volunteer to help!

- user\_e21 from Atlanta, Georgia, US needs help with Python3 - [click to help](#) (active 2 minutes ago, requested 7 hours ago)
- user\_ac6 from Roselle, New Jersey, US needs help with Python3 - [click to help](#) (active 2 minutes ago, requested 4 hours ago)

Python 3.6

```
1 import random
2
3 def flipCoins():
4     print('I will flip a coin 1000 times. ')
5     print('Guess how many times it will come up heads. ')
6     flips = 0
7     heads = 0
8     while flips < 1000:
9         print("Flip:"+str(flips)+"      Head:"+str(heads))
10        if random.randint(0, 1) == 1:
11            heads = heads + 1
12        flips = flips + 1
13        if flips == 500:
14            print('Half way done, and heads has come up ' + s
15        print()
16        print('Out of 1000 coin tosses, heads came up ' + str(hea
17
18
19 flipCoins()
```

[Edit this code](#)

→ 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

Step 36 of 999

Forward >

Last >>

Print output (drag lower right corner to resize)

```
Flip:0      Head:0
Flip:1      Head:1
Flip:2      Head:2
Flip:2      Head:2
Flip:3      Head:3
```

Frames

Objects

Global frame

module instance

random

flipCoins

function

flipCoins()

flipCoins

flips

4

heads

4

Press Forward to  
advance one  
“step” of the  
program

# Function

Scope and Recursion

Scope

# Quick Scope Exercise

## Code

```
x = 0
```

```
def foo_printx():  
    print(x)
```

```
foo_printx()  
print(x)
```

## Output

```
0
```

```
0
```

# Quick Scope Exercise

## Code

```
x = 0
y = 999
def foo_printx(y):
    print(y)
```

```
foo_printx(x)
print(x)
```

## Output

```
0
0
```

# Quick Scope Exercise

## Code

```
x = 0
```

```
def foo_printx():
```

```
    x = 999
```

```
    print(x)
```

```
foo_printx()
```

```
print(x)
```

## Output

999

0



# Quick Scope Exercise

**Why?**

# Global Variables

## Code

```
x = 0
```

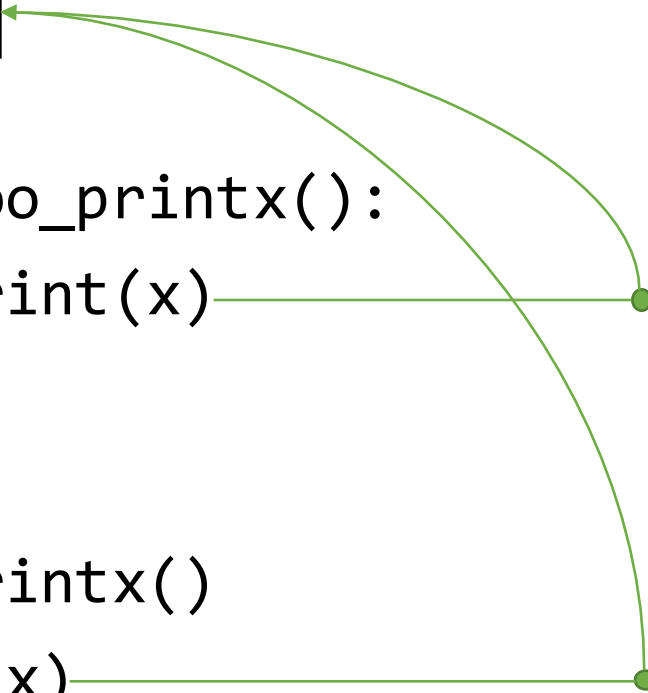
```
def foo_printx():  
    print(x)
```

```
foo_printx()  
print(x)
```

## Explanation

• This 'x' refers to the outer 'x'

• This 'x' also refers to the outer 'x'



# Global vs Local Variables

## Code

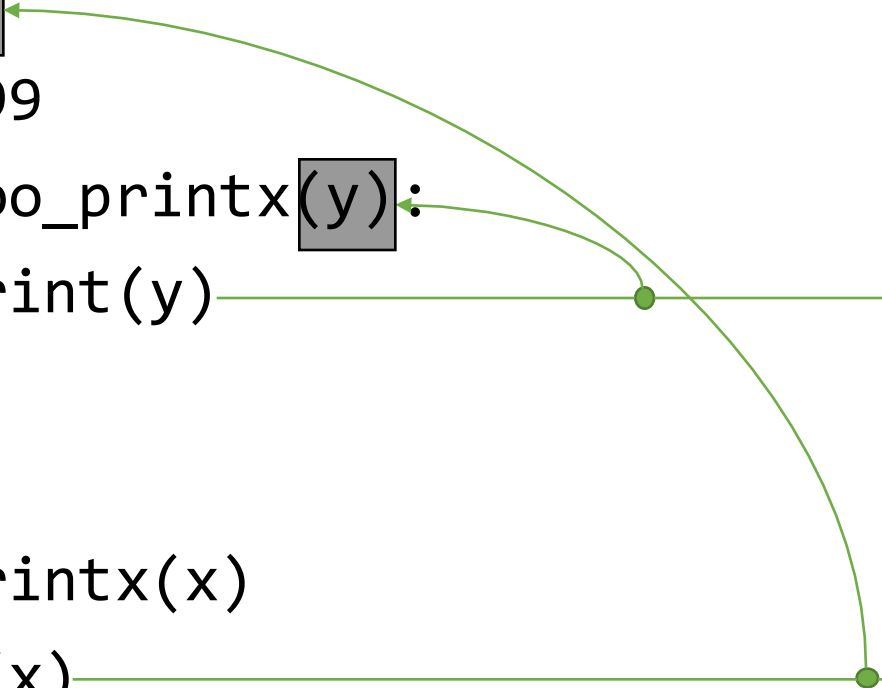
```
x = 0
y = 999
def foo_printx(y):
    print(y)

foo_printx(x)
print(x)
```

## Explanation

• This 'y' refers to the parameter  
pass-by-value

• This 'x' refers to the outer 'x'



# Global vs Local Variables

## Code

```
x = 0
```

```
def foo_printx():
```

```
    x = 999
```

```
    print()
```

```
foo_printx()
```

```
print(x)
```

## Explanation

- This 'x' is created new because of assignment

- This 'x' still refers to the outer 'x'

# Global vs Local Variables

## Code

```
x = 0
```

```
def foo_printx():
```

```
    x = 999  
    print()
```

```
foo_printx()  
print(x)
```

## Explanation

- Global scope
- Local scope
  - Local 'x' is born here
  - Will die when the function ends here
- The two 'x' will be different 'x'
  - '999' will only be available within function

# Rule of Thumb

## Code

```
x = 0
```

```
def foo_printx():
```

```
    x = 999          (2)
```

```
    print(x)        (1)
```

```
foo_printx()
```

```
print(x)
```

## Go up and go out cannot go in

- Simple case: x within function
  1. Start here
  2. Go up
    - Found!

# Rule of Thumb

## Code

```
x = 0 (6)
```

```
(5)
```

```
def foo_printx():  
    x = 999 (4)  
    print(x)
```

```
(3)
```

```
foo_printx() (2)
```

```
print(x) (1)
```

## Go up and go out cannot go in

- Harder case: x outside function

1. Start here
2. Go up
3. Go up
4. Cannot go in
5. Go up
6. Go up
  - Found!

# Global vs Local Variables

- A variable which is defined in the main body of a file is called a **global** variable. It will be **visible throughout the file**, and also inside any file which imports that file. EXCEPT...
- A variable which is defined inside a function is **local** to that function. It is accessible **from the point at which it is defined until the end of the function**, and exists for as long as the function is executing.
- The parameter names in the function definition behave like local variables, but they contain the values that we pass into the function when we call it.



# Crossing Boundaries

## Problem

- What if we want to modify variables from outside within the function?
  - Use “global” keyword
  - No local variables ‘x’ is created

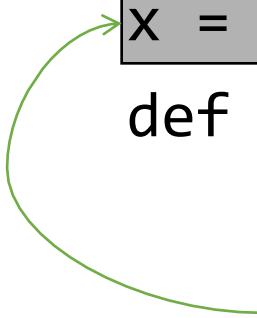
Output:

999

999

## Code

```
x = 0
def foo_printx():
    global x
    x = 999
    print(x)
```



```
foo_printx()
print(x)
```

# To Cross or Not to Cross

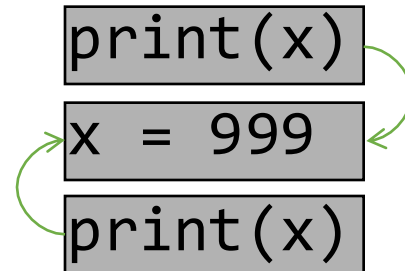
## Problem

- Consider the following code
- What is happening?
  - Second print(x) refers to x = 999
  - What about the first print(x)?
    - Also to x = 999
    - But it comes after!
    - This is an error
    - It has no value

## Code

```
x = 0
```

```
def foo_printx():
```



```
foo_printx()
```

```
print(x)
```

# Parameters are Local

## Code

```
def foo(x): (1)
```

```
    bar(x + 1) (2) (A)
```

```
    print(x) (3)
```

```
def bar(x): (A)
```

```
(C) x = x + x (B)
```

```
    print(x) (D)
```

```
print(foo(3)) (1)
```

## Explanation

1. Pass 3 to x in foo
2. Evaluate x + 1
  - A. Pass 4 to x in bar
  - B. Evaluate x + x
  - C. Assign to x
  - D. print(x) in bar
    - print 8
3. print(x) in foo
  - print 3

# Parameters are Local

## Code

```
def foo(x):  
    bar(x + 1)  
    print(x)  
  
def bar(x):  
    x = x + x  
    print(x)  
  
print(foo(3))
```

## Explanation

- The 'x' in bar is different from the 'x' in foo

# Convention

- Global variables are VERY **bad** practices
  - Especially if modifications are allowed
- 99% of the time, global variables are used as constants
  - Variables that every function can access
  - But not expected to be modified

```
POUNDS_IN_ONE_KG = 2.20462
def kg2pound(weight):
    return weight * POUNDS_IN_OKE_KG
def pound2kg(weight):
    return weight / POUNDS_IN_OKE_KG
```

Convention:  
use all CAPS

# Variable Scope Exercises

## Code

```
x = 1
y = 2
def foo(y):
    def bar(x):
        return x+y
    return bar(y)
print(foo(x))
```

## Output

2

# Variable Scope Exercises

## Code

```
x = 1
y = 2
def foo(x):
    def bar(x):
        return x+y
    return bar(y)
print(foo(x))
```

## Output

4

# Recursion



## Recap:burgerPrice(burger)

```
def burgerPrice(burger):  
    price = 0  
    for char in burger:  
        if char == 'B':  
            price += 0.5  
        : # code omitted  
    return price
```

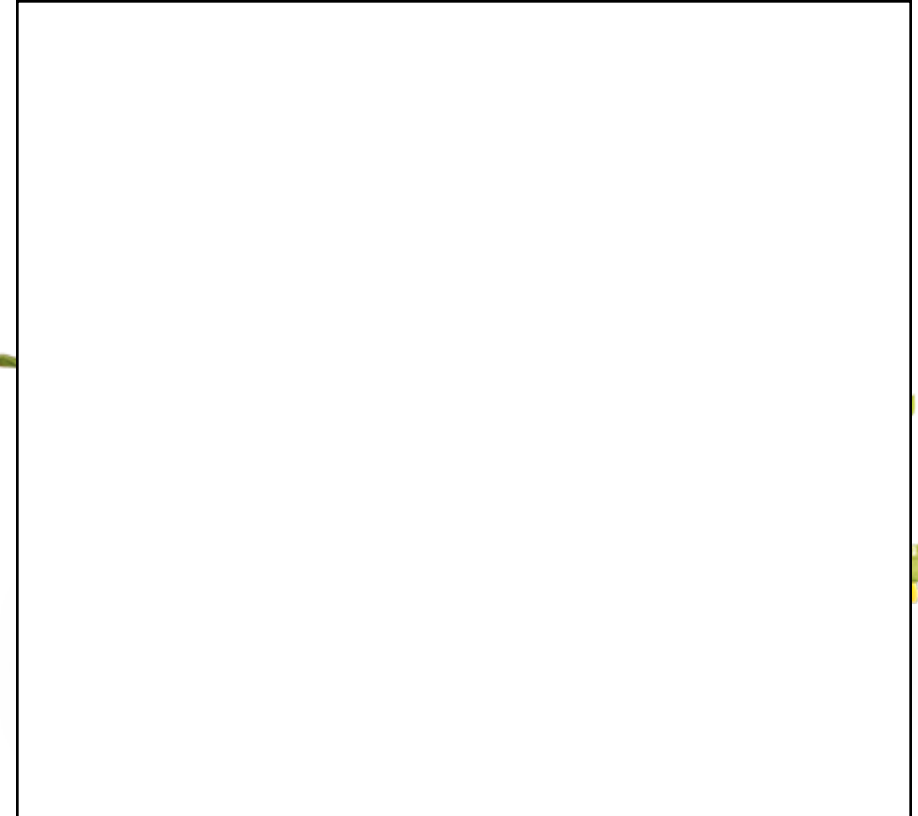
# burgerPrice(burger) in recursion?

- Idea
  - bigMac = 'BPVOBPVOCB'
  - What's the price
    - burgerPrice(bigMac) = 6.7
  - How?
    - B is bun, costs 0.5
    - The rest is 'PVOBPVOCB'
      - How to get?
        - String slicing: theRest = bigMac[1:]
      - How much?
        - Recursion: burgerPrice(theRest)
    - Total: 0.5 + burgerPrice(theRest)



# burgerPrice(burger) in recursion?

- Idea
  - bigMac = 'BPVOBPVOCB'
  - What's the price
    - burgerPrice(bigMac) = 7.5
  - How?
    - When to stop?
      - No more burger
      - price = 0



# burgerPrice(burger) in recursion?

- Code

```
def burgerPrice(burger):  
    if burger == '':  
        price = 0  
    else:  
        if burger[0] == 'B':  
            return 0.5 + burgerPrice(burger[1:])  
        : # code omitted
```



# Recursion vs Iteration

- Sum
  - Given a positive number  $n$ , the sum of all digits is obtained by adding the digit one-by-one
  - For example, the sum of 52634 =  $5 + 2 + 6 + 3 + 4 = 20$
  - Write a function `sum(n)` to compute the sum of all the digits in  $n$
- Fact
  - Factorial is defined (recursively) as  $n! = n * (n - 1)!$  such that  $0! = 1$
  - Write a function `fact(n)` to compute the value of  $n!$
- Can you do it in both recursion and iteration?

# Sum

## Iteration

```
def sum(n):  
    res = 0  
    while n > 0:  
        res = res + n%10  
        n = n//10  
    return res
```

base/initial value

computation

continuation/next value

## Recursion

```
def sum(n):  
    if n == 0:  
        return 0  
    else:  
        return n%10 + sum(n//10)
```

stop/base case (*they are related, how?*)

temporary result variables  
not needed in recursion (*why?*)

# Factorial

## Iteration

```
def fact(n):  
    res = 1  
    while n > 0:  
        res = res * n  
        n = n-1  
    return res
```

base/initial value

computation

continuation/next value

## Recursion

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

stop/base case (*they are related, how?*)

temporary result variables  
not needed in recursion (*why?*)

# Additional Practice

- Final Sum
  - Given a positive number  $n$ , the final sum is obtained by repeatedly computing the sum of  $n$  until the sum is a single digit
  - For example, the sum of  $52634 = 5 + 2 + 6 + 3 + 4 = 20$
  - The sum of  $20 = 2 + 0 = 2$
  - The final sum of  $52634 = 2$
  - Write a function `final_sum(n)` to compute the final sum of  $n$



# Additional Practice

- Euler Constant
  - Euler constant is the value  $e$  that has a special property where the derivative of  $e^x$  is  $e^x$
  - The value of  $e^x$  can be approximated using the following formula
    - $e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$
  - Write a function `find_e(x,n)` to find the approximation of  $e^x$  up to  $n+1$  steps
    - In other words, the last value in the approximation will be  $\frac{x^n}{n!}$