Functions

Prof Matthew Fricke

Version 1.0. Send corrections to mfricke@unm.edu.

What do you think of computer programming?

Yes, programming *is* confusing when you start.

Most of what programming languages do is an attempt to make programs less confusing.
This is especially true of functions.

Learning to program is like learning the piano.

Learning to program is like learning the piano.

You can understand the music sheet perfectly, but you still have to practice playing the music.

And like great literature…

Every good programmer should go to sleep reading a good program.*

*Paraphrasing Donald Knuth

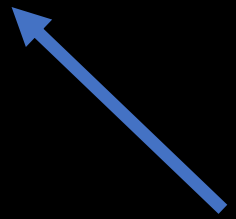
You have all been using functions already.

```
print( text_to_print )
```

```
some_var_name = input( text_to_print )
```

You have all been using functions already.

```
print( text_to_print )
```



This is the function name.

```
some_var_name = input( text_to_print )
```

You have all been using functions already.

```
print( text_to_print )
```

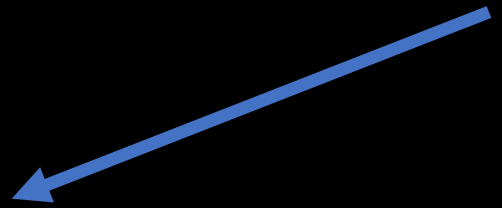
This is the function argument

```
some_var_name = input( text_to_print )
```

You have all been using functions already.

```
print( text_to_print )
```

This function returns a value



```
some_var_name = input( text_to_print )
```

You have all been using functions already.

```
print( text_to_print )
```

We can put as much code as we want
Inside a function and then give it a short name.

The short name helps us understand the code we are writing.

```
some_var_name = input( text_to_print )
```

You have all been using functions already.

```
print( text_to_print )
```

Variables gave friendly names to data.

Functions give friendly names to whole sections of code.

```
some_var_name = input( text_to_print )
```

This is a lot like functions in math:

x = cos(theta)

y = sqrt(x)

3 = sqrt(9)

1 = cos(0)

These functions take an input number and return an output Number

Python functions also take an input argument and return a value.

Here is what they might look like in python

```
def  square(x):
      return x*x

4 = square(2)
```

Here is what they might look like in python

```
def  cube(x):
        return square(x)*x

8 = cube(2)
```

Get to know your neighbors…

Work in groups of 2 or 3. Your mission is to

- Think of a simple problem you believe the class can code up.
- On a piece of paper write down the algorithm (the steps of the problem, don't worry about code)
- Text title for your problem

For example:

- Problem: return which of two numbers is biggest
- Algorithm:
  - Get two numbers
  - If the first number is biggest return that number
  - If the second number is biggest return the second number
  - Done

Implement an Algorithm

Top

Lots of good suggestions –
the top voted suggestion was odd even

We know how to write small programs

OK let’s code it up…

```
# This is my awesome odd or even program (sanitized)

# Ask the the user for a number
user_num_string = input("Enter a number")

# We really want a int not a string
user_num = int(user_num_string)

# Calculate odd or even
# I know how to tell if a number is even!
# By definition the remainder of the number / 2 is zero
if user_num % 2 == 0:
    # Choose the even message
    message_to_user = "Your number was even"
else:
    # Choose the odd message
    message_to_user = "ODD!!!!"

# We will tell the user whether their
# number was odd or even.

print(message_to_user)
```

```
Polychrome-3:~ matthew$ python3 even_or_odd.py
Enter a number9
ODD!!!!
(base) Polychrome-3:~ matthew$ python3 even_or_odd.py
Enter a number42
Your <redacted> number was even
(base) Polychrome-3:~ matthew$ python3 even_or_odd.py
Enter a numberI don't know one
Traceback (most recent call last):
  File "even_or_odd.py", line 7, in <module>
    user_num = int(user_num_string)
ValueError: invalid literal for int() with base 10: "I don't know one"
```

Now we have something useful…

We can write that code over and over every time we need it, every language, including Python provides a better way.

write reusable pieces/chunks of code, called functions

functions are not run in a program until they are “called” or “invoked” in a program

function characteristics:

- has a name
- has parameters(0 or more)
- has a docstring(optional but recommended)
- has a body
- returns something

Turn our odd-even code into a function…

```
def odd_even():
    # This is my awesome odd or even program (sanitized)

    # Ask the the user for a number
    user_num_string = input("Enter a number")

    # We really want a int not a string
    user_num = int(user_num_string)

    # Calculate odd or even
    # I know how to tell if a number is even!
    # By definition the remainder of the number / 2 is zero
    if user_num % 2 == 0:
        # Choose the even message
        message_to_user = "Your number was even"
    else:
        # Choose the odd message
        message_to_user = "ODD!!!!"

    # We will tell the user whether their
    # number was odd or even.

    print(message_to_user)
```

And to use it…

```
def odd_even():
        # This is my awesome odd or even program (sanitized)

        # Ask the the user for a number
        user_num_string = input("Enter a number")

        # We really want a int not a string
        user_num = int(user_num_string)

        # Calculate odd or even
        # I know how to tell if a number is even!
        # By definition the remainder of the number / 2 is zero
        if user_num % 2 == 0:
            # Choose the even message
            message_to_user = "Your number was even"
        else:
            # Choose the odd message
            message_to_user = "ODD!!!!"

        # We will tell the user whether their
        # number was odd or even.

        print(message_to_user)

odd_even():
```

```
Polychrome-3:~ matthew$ python3 even_or_odd.py
Enter a number9
ODD!!!!
(base) Polychrome-3:~ matthew$ python3 even_or_odd.py
Enter a number42
Your <redacted> number was even
(base) Polychrome-3:~ matthew$ python3 even_or_odd.py
Enter a numberI don't know one
Traceback (most recent call last):
  File "even_or_odd.py", line 7, in <module>
    user_num = int(user_num_string)
ValueError: invalid literal for int() with base 10: "I don't know one"
```

And to use it…

```
def odd_even():
        # This is my awesome odd or even program (sanitized)

        # Ask the the user for a number
        user_num_string = input("Enter a number")

        # We really want a int not a string
        user_num = int(user_num_string)

        # Calculate odd or even
        # I know how to tell if a number is even!
        # By definition the remainder of the number / 2 is zero
        if user_num % 2 == 0:
            # Choose the even message
            message_to_user = "Your number was even"
        else:
            # Choose the odd message
            message_to_user = "ODD!!!!"

        # We will tell the user whether their
        # number was odd or even.

        print(message_to_user)

odd_even():
```

```
Polychrome-3:~ matthew$ python3 even_or_odd.py
Enter a number9
ODD!!!!
(base) Polychrome-3:~ matthew$ python3 even_or_odd.py
Enter a number42
Your <redacted> number was even
(base) Polychrome-3:~ matthew$ python3 even_or_odd.py
Enter a numberI don't know one
Traceback (most recent call last):
  File "even_or_odd.py", line 7, in <module>
    user_num = int(user_num_string)
ValueError: invalid literal for int() with base 10: "I don't know one"
```

We get the same output so what was the point…?

Encapsulation, abstraction: naming code blocks makes Code much easier to understand… (less confusing)

Now we have something useful

Laziness is a virtue

Instead of writing a piece of code ourselves always ask if someone else has already done it

Think of functions as chunks of useful code.

Now we have something useful

Laziness is a virtue

- If someone already wrote code to solve the problem we can use their code:
  - If they already did it they often know/care more about that particular problem than we do.
  - If it already exists in public then its has probably been examined and used by other people.
  - Even if their code is flawed it gives us a place to work from.

Now we have something useful

Laziness is a virtue

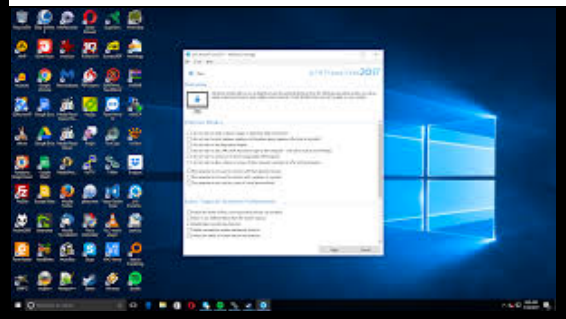
- If someone already wrote code to solve the problem we can use their code:
  - If they already did it they often know/care more about that particular problem than we do.
  - If it already exists in public then its has probably been examined and used by other people.
  - Even if their code is flawed it gives us a place to work from.
  - Get used to it this is how every programmer works.

Commercial Software

Apple

Bill Gates

Microsoft



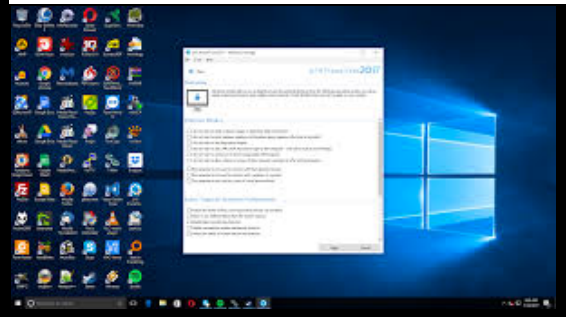
Steve Jobs

Commercial Software

Bill Gates

Apple

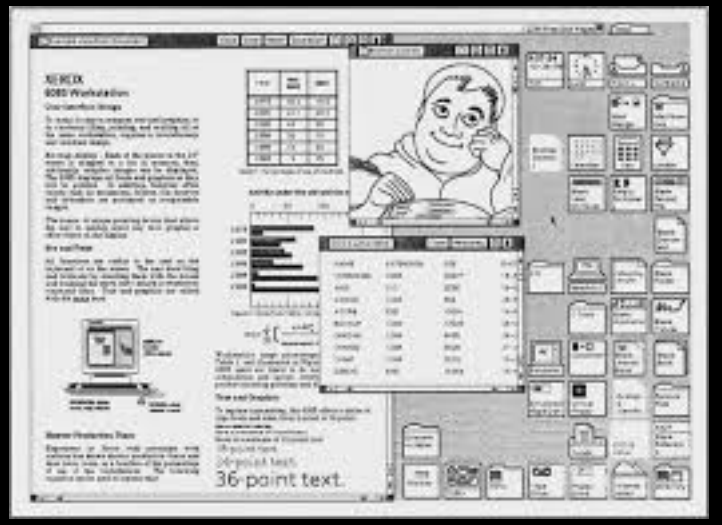
Microsoft



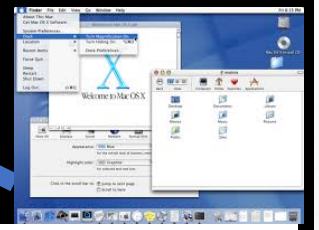
You stole my idea!

Steve Jobs

Commercial Software

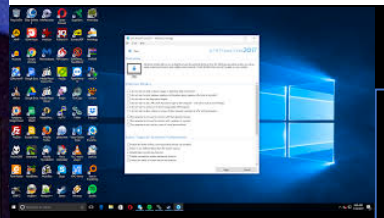


xerox



Apple

Microsoft



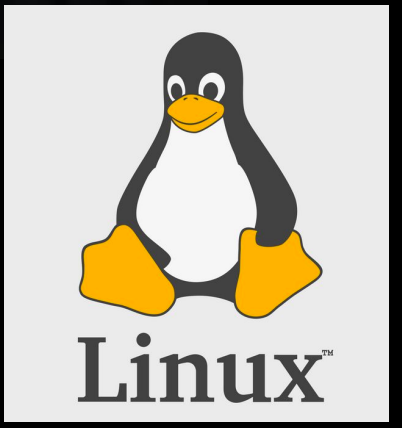
"Well, Steve … I think it's more like we both had this rich neighbour named Xerox and I broke into his house to steal the TV set and found out that you had already stolen it."

Open Source Software

Richard Stallman

Linus Torvolds

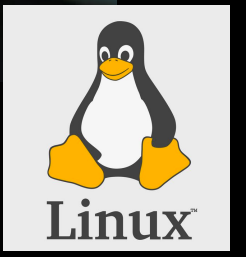
GNU:
GNU is Not Unix



UNIX

Where there is a shell, there is a way.

GNU:
GNU is Not Unix



On the one hand information wants to be expensive, because it's so valuable. The right information in the right place just changes your life. On the other hand, information wants to be free, because the cost of getting it out is getting lower and lower all the time. So you have these two fighting against each other.

On the one hand information wants to be expensive, because it's so valuable. The right information in the right place just changes your life. On the other hand, information wants to be free, because the cost of getting it out is getting lower and lower all the time. So you have these two fighting against each other.

What has this got to do with functions?

Functions encapsulate useful algorithms.
Valuable algorithms are easy to share.

Open source software shares all this value with everyone because then everyone benefits.

On the one hand information wants to be expensive, because it's so valuable. The right information in the right place just changes your life. On the other hand, information wants to be free, because the cost of getting it out is getting lower and lower all the time. So you have these two fighting against each other.

Open source software shares all this value with everyone because then everyone benefits.

Commercial companies do the same thing. They just spend a lot of time agreeing not to sue each other.

On the one hand information wants to be expensive, because it's so valuable. The right information in the right place just changes your life. On the other hand, information wants to be free, because the cost of getting it out is getting lower and lower all the time. So you have these two fighting against each other.

There are thousands of functions that programmers around the world have been writing for decades that are available for you to use.

You can access libraries of these functions with the import command.

Encapsulation and Laziness

It is very helpful to have this universe of functions other people have written.

Laziness here means using other peoples existing functions, even when that other person might be you later.

Encapsulation and Laziness

Functions let you break the problem up into manageable pieces. Function names help you organize all the parts of your code.

Solve one piece at a time to make it easier.

For example:

Let’s write a program to add up a sequence of integers between a start and end value.

For example given 0 and 5 we want:
0 + 1 + 2 + 3 + 4 + 5 = 16

For example:

Coding… coding…

Hmmm – should we use a while or for loop?

```
# A function that sums ints using a while loop
def while_add(start, end):
    total = 0
    current_num = start
    while current_num <= end:
        total=total + current_num
        current_num=current_num + 1

    return total

# A function that sums ints using a for loop
def for_add(start, end):
    total = 0
    for x in range(start, end+1):
        total = total + x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(while_add(x,y))
print(for_add(x,y))
```

```
bash-3.2$ python3 addints.py
Enter Start Number: 2
Enter End Number: 5
9
9
bash-3.2$python3 addints.py
Enter Start Number: 2
Enter End Number: 5
14
14
bash-3.2$ python3 addints.py
Enter Start Number: -1
Enter End Number: 4
9
9
```

For loops and while loops both work (in fact there is a theorem that says anything you can do with a while loop you can do with a for loop)

```
# A function that sums ints using a while loop
def while_add(start, end):
    total = 0
    current_num = start
    while current_num <= end:
        total=total + current_num
        current_num=current_nur

    return total

# A function that sums ints usir
def for_add(start, end):
    total = 0
    for x in range(start, end+1):
        total = total + x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(while_add(x,y))
print(for_add(x,y))
```

```
bash-3.2$ python3 addints.py
Enter Start Number: 2
Enter End Number: 5
9
9
```

The for loop is more elegant*. Which basically means you don´t have to think so hard about how it works - assuming you understand for loops.

both work (in
at says
anything you can do with a while loop
you can do with a for loop)

*You won´t be judged on elegance on exams and quizzes in this course.

For example:

Now that we have that code break up into groups and on a piece of paper write down the function that would do this… (5 mins)

For example:

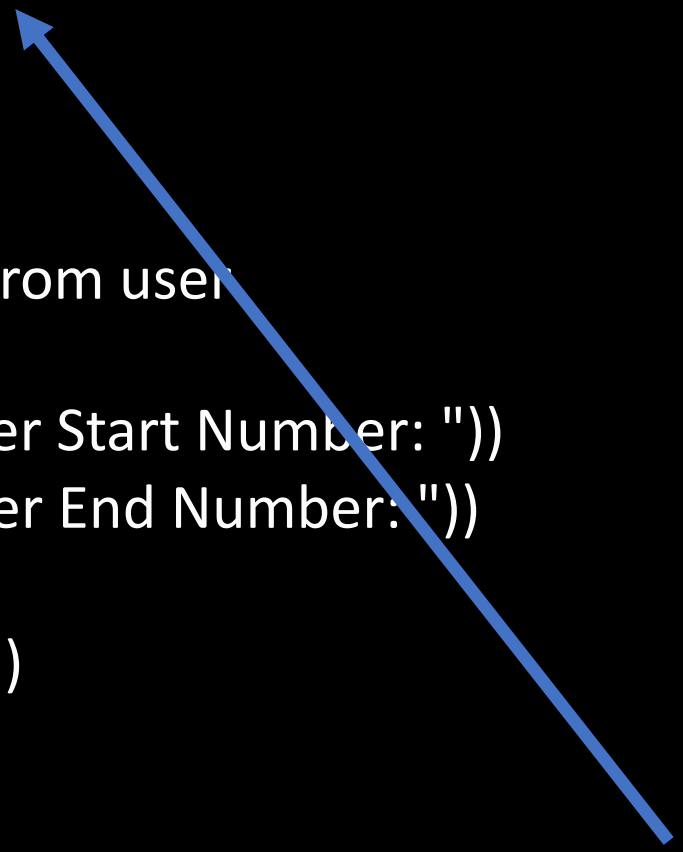
Let’s write a program to get the product of integers from a start to an end value.

For example given 1 and 3 we want:
1 x 2 x 3 = 6

For example:

Coding… coding…

```
# A function that sums ints using a for loop
def for_add(start, end):
    total = 0
    for x in range(start, end+1):
        total = total + x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_add(x,y))
```

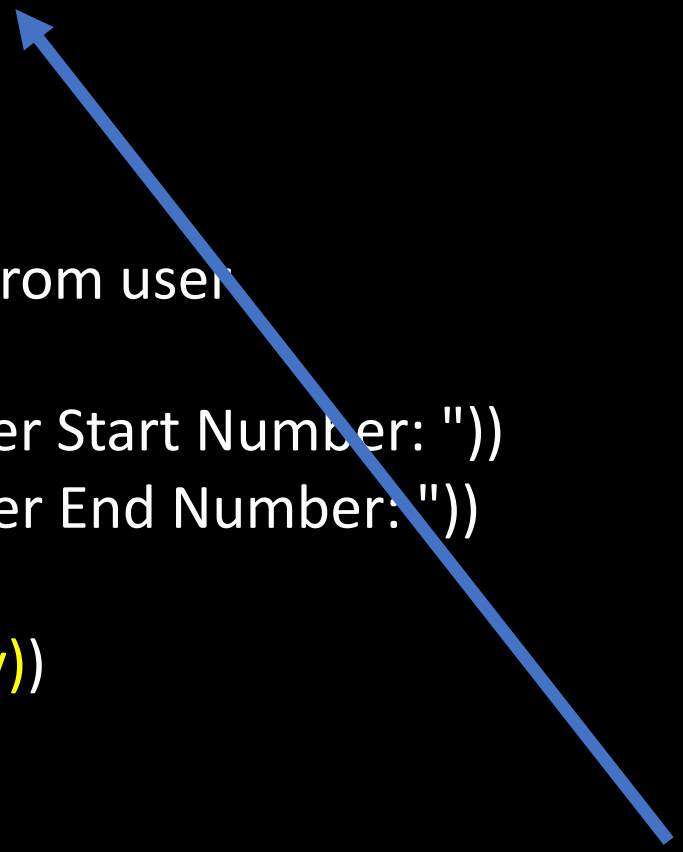


I guess we can just change the + to *

And the name of the function so it makes sense .

Also I will put it in a file called multints.py because having a program that multiplies ints called addints.py would just be cruel.

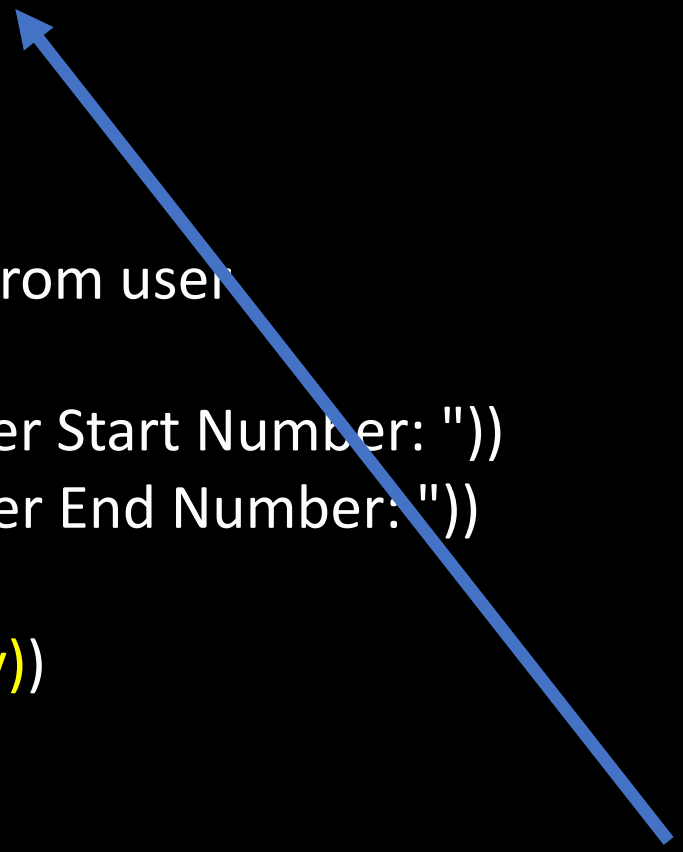
```
# A function that sums ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```



I guess we can just change the + to *
And the name of the function so it makes sense

```
# A function that sums ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

```
python3 multints.py
Enter Start Number: 3
Enter End Number: 10
0
```



I guess we can just change the + to *
And the name of the function so it makes sense

```
# A function that sums ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

```
python3 multints.py
Enter Start Number: 3
Enter End Number: 10
0
```

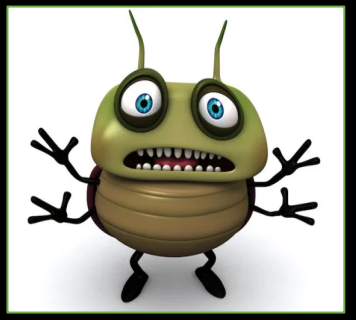
This isn’t right!

3*4*5*6*7*8*9*10 = 1814400

```
# A function that sums ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

```
python3 multints.py
Enter Start Number: 3
Enter End Number: 10
0
```

We have a bug!



3*4*5*6*7*8*9*10 = 1814400

Debugging

¨Granted that the actual mechanism is unerring in its processes, the *cards* may give it wrong orders.” Ada Lovelace (1843)

Debugging

“You were partly correct, I did find a ‘bug’ in my apparatus [telephone]” Thomas Edison, 1878

Debugging

You find and eliminate bugs by:

Tracing through your code

Testing your code

Debugging

You find and eliminate bugs by:

Tracing through your code

Testing your code

Prevent bug by writing clear understandable code

Debugging – Tracing Code

You can trace code by following each step and writing down the results at each step*

You can make that easier by printing results of statements when you run the code.

This is called debug output.

*You will need to be able to do this on the exams…

Expression	Current Value
Global Scope	
x	?
y	?
for_mult(x,y)	?
for_mult() Scope:	
total	?
start	?
end	?
x	?
Range(start,end+1)	?

```
# A function that sums ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

Expression	Current Value
Global Scope	
x	**3**
y	?
for_mult(x,y)	?
for_mult() Scope:	
total	?
start	?
end	?
x	?
Range(start,end+1)	?

User Enters: 3

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

Expression	Current Value
Global Scope	
x	3
y	**6**
for_mult(x,y)	?
for_mult() Scope:	
total	?
start	?
end	?
x	?
Range(start,end+1)	?

User Enters: 6

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

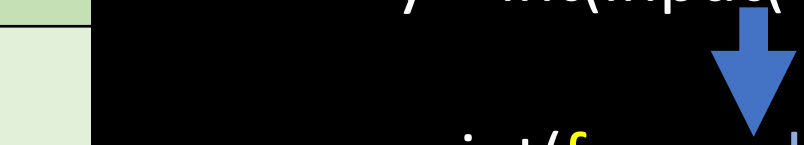
Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	?
start	?
end	?
x	?
Range(start,end+1)	?

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

Before print() the expression inside the () is executed

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	?
start	?
end	?
x	?
Range(start,end+1)	?

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```



Before print() the expression inside the () is executed

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	?
start	?
end	?
x	?
Range(start,end+1)	?

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

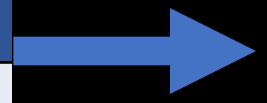
Evaluating the function makes us jump to the code inside the function

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	?
start	**3**
end	**6**
x	?
Range(start,end+1)	?

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

Evaluating the function makes us jump to the code inside the function

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	**0**
start	3
end	6
x	?
Range(start,end+1)	?



```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

Evaluating the function makes us jump to the code inside the function

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	0
start	3
end	6
x	**3**
Range(start,end+1)	**[3,4,5,6,7]**

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

Evaluating the function makes us jump to the code inside the function

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	0
start	3
end	6
x	3
Range(start,end+1)	[3,4,5,6,7]

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

Shadowing!

Evaluating the function makes us jump to the code inside the function

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	0 (0*3)
start	3
end	6
x	3
Range(start,end+1)	[3,4,5,6,7]

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

New value of total = the old value of total (zero) times 3

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	0 (0*3)
start	3
end	6
x	4
Range(start,end+1)	[3,4,5,6,7]

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

New value of total = the old value of total (zero) times 3

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	**0** (0*4)
start	3
end	6
x	4
Range(start,end+1)	[3,4,5,6,7]

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

New value of total = the old value of total (zero) times 4

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	0 (0*4)
start	3
end	6
x	5
Range(start,end+1)	[3,4,5,6,7]

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

New value of total = always zero times something… so always zero

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	**0** (0*5)
start	3
end	6
x	5
Range(start,end+1)	[3,4,5,6,7]

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

New value of total = always zero times something… so always zero

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	0 (0*5)
start	3
end	6
x	6
Range(start,end+1)	[3,4,5,6,7]

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

New value of total = always zero times something… so always zero

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	**0** (0*6)
start	3
end	6
x	6
Range(start,end+1)	[3,4,5,6,7]

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

New value of total = always zero times something… so always zero

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	0 (0*6)
start	3
end	6
x	7
Range(start,end+1)	[3,4,5,6,7]

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

New value of total = always zero times something… so always zero

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	?
for_mult() Scope:	
total	0
start	3
end	6
x	7
Range(start,end+1)	[3,4,5,6,7]

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

New value of total = always zero times something… so always zero

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	**0**
for_mult() Scope:	
total	**0**
start	3
end	6
x	7
Range(start,end+1)	[3,4,5,6,7]

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print( total )
```

Returns zero so for_mult(3,6) is zero.

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	0
for_mult() Scope:	
total	0
start	3
end	6
x	7
Range(start,end+1)	[3,4,5,6,7]

Prints zero to the user

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

Expression	Current Value
Global Scope	
x	3
y	6
for_mult(x,y)	0
for_mult() Scope:	
total	0
start	3
end	6
x	7
Range(start,end+1)	[3,4,5,6,7]

So our bug was that setting total to 0 at the start means it will always return 0

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
                                    nd+1):
                                    ser
                                    t Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

Prints zero to the user

Expression	Current Value
Global Scope	
x	3
y	
for_mu	
for_mu	
total	
start	
end	6
x	7
Range(start,end+1)	[3,4,5,6,7]

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 0
    for x in range(start, end+1):
```

The fix is to set total to 1 at the start.

(1 is the multiplication identity)

```
x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

Prints zero to the user

```
# A function that times ints using a for loop
def for_mult(start, end):
    total = 1
    for x in range(start, end+1):
        total = total * x

    return total

# Read numbers from user

x = int(input("Enter Start Number: "))
y = int(input("Enter End Number: "))

print(for_mult(x,y))
```

```
python3 multints.py
Enter Start Number: 3
Enter End Number: 6
360
```

3 x 4 x 5 x 6 is 360

Now we can be lazy…

Let´s write the factorial function.

Factorial n! is just 1 x 2 x 3 x … x n

Now we can be lazy…

Let´s write the factorial function.

Factorial n! is just 1 x 2 x 3 x … x n

We already solved a harder problem so lets be lazy and reuse our for_mult function.

```
# A function that multiplies ints using a for loop
def for_mult(start, end):
    total = 1
    for x in range(start, end+1):
        total = total * x

    return total

# Factorial function
def factorial(x):
    return for_mult(1, x)

x = int(input("Enter Number: "))
y = factorial(x)

print(y)
```

```
python3 factorial.py
Enter Number: 10
3628800
```

For example:

Let’s write code that keeps asking the user to enter a start and end value.

Then we can use our addnums() function to calculate the response.

The program should stop if the user enters the letter “q”.

For example:

Now that we have that code break up into groups and on a piece of paper write down the function that would do this… (5 mins)

For example:

Let’s test our program thoroughly (this is what Shadow Inc didn’t do and had big impact on our election process yesterday)

It is also what Boeing Inc didn’t do resulting in 386 dead in Max 80 crashes.

Helpful Laziness...

Dividing the problem you are trying to solve up into manageable parts is the most important thing.

Then work on each piece, one at a time.

If some part seems to hard – try breaking it up again.

Eventually you turn a hard problem into many easy problems. That is 90% of computer science.

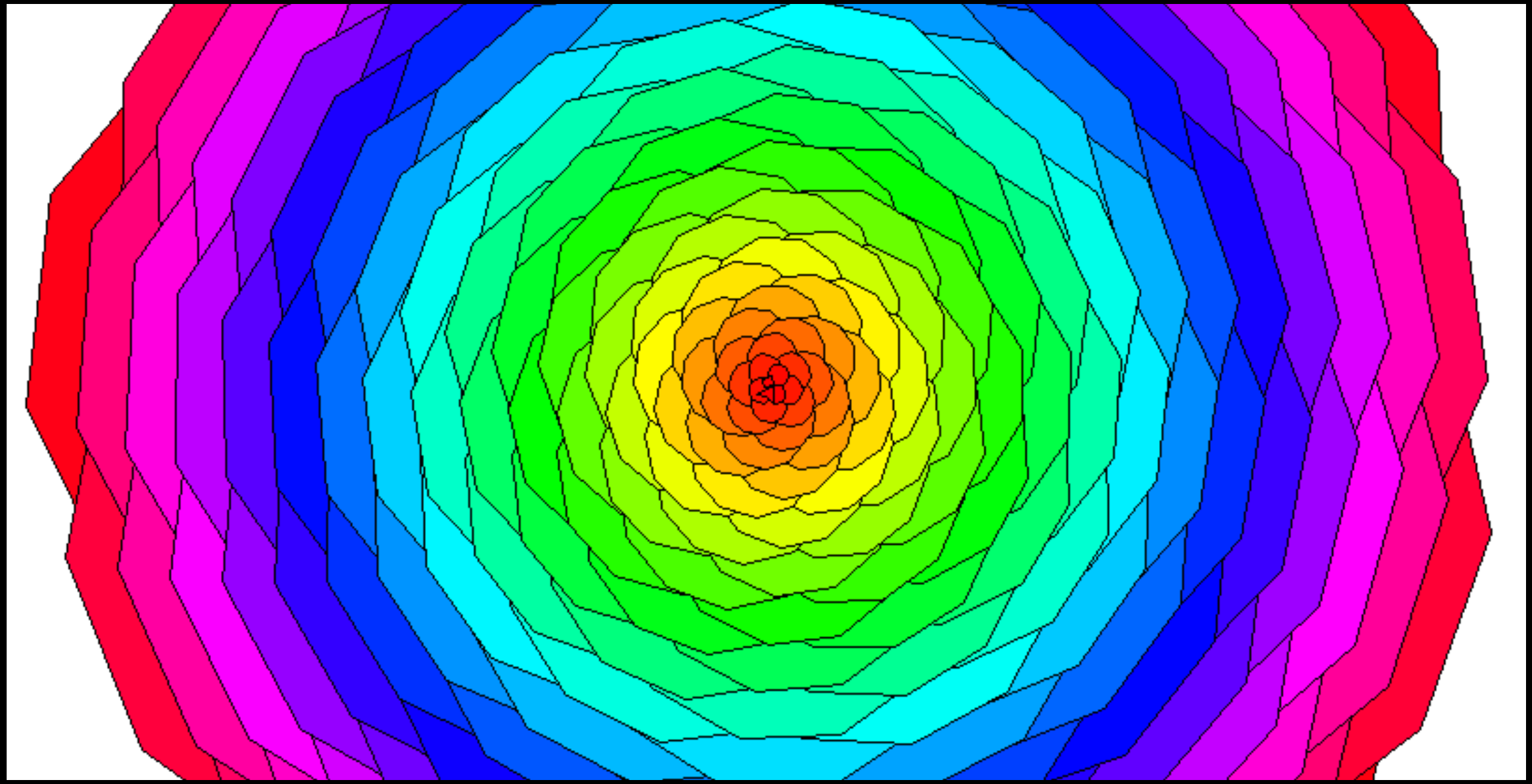
Now that we know functions and loops…

We have access to some powerful abilities….

The next example uses loops and function…

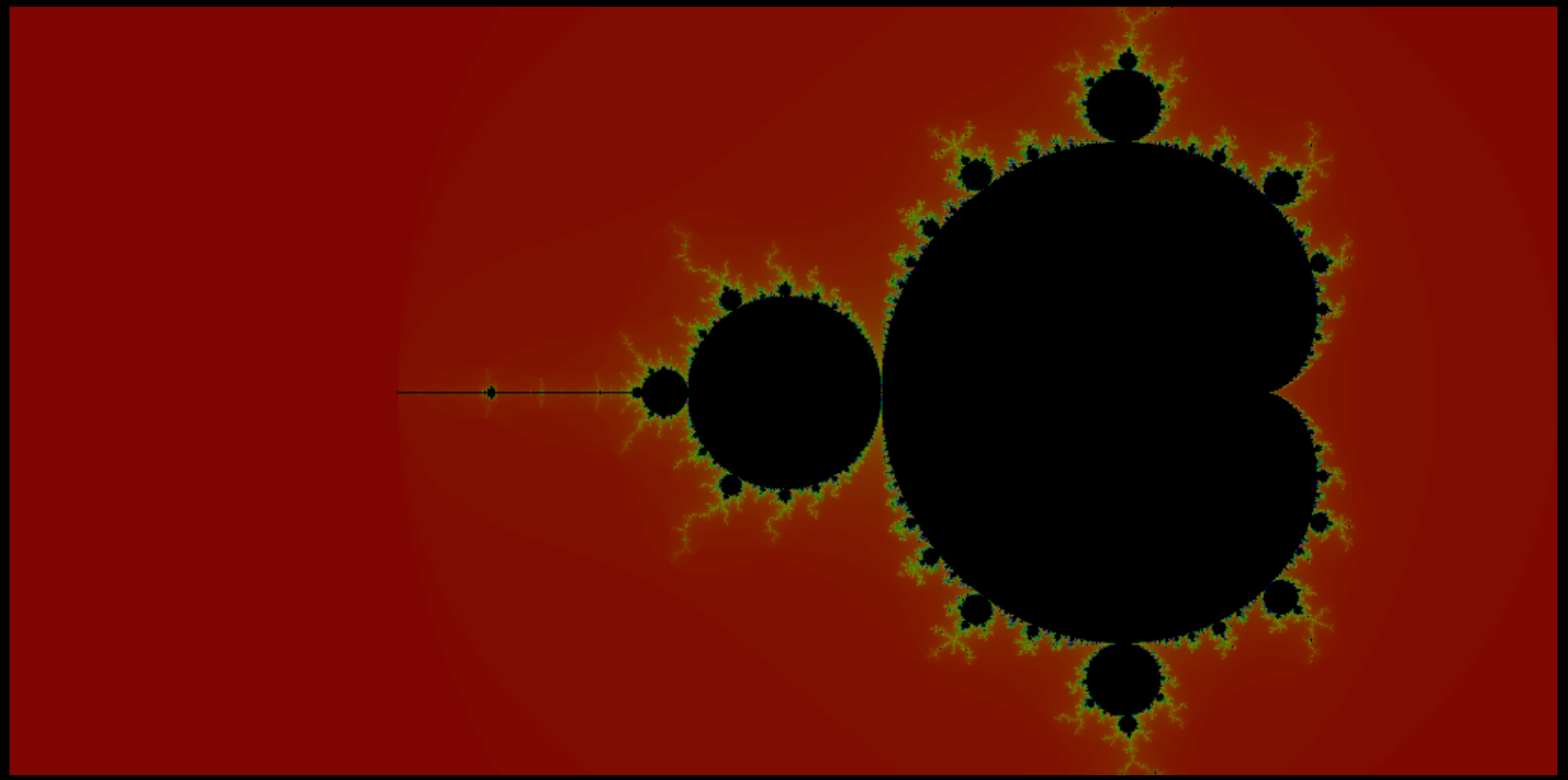
You will understand the rest by the end of the course (preview)

```
import turtle
import math
import colorsys

phi = 180 * (3 -
math.sqrt(5))

t = turtle.Pen()
t.speed(0)

def square(t, size):
    for tmp in range(0,4):
        t.forward(size)
        t.right(90)

num = 200

for x in reversed(range(0, num)):

t.fillcolor(colorsys.hsv_to_rgb(x/num,
1.0, 1.0))
    t.begin_fill()
    t.circle(5 + x, None, 11)
    square(t, 5 + x)
    t.end_fill()
    t.right(phi)
    t.right(.8)

turtle.mainloop()
```



```
# Python code for Mandelbrot Fractal

# Import necessary libraries
from PIL import Image
from numpy import complex, array
import colorsys

# setting the width of the output image as 1024
WIDTH = 1024

# a function to return a tuple of colors
# as integer value of rgb
def rgb_conv(i):
    color = 255 * array(colorsys.hsv_to_rgb(i / 255.0, 1.0, 0.5))
    return tuple(color.astype(int))

# function defining a mandelbrot
def mandelbrot(x, y):
    c0 = complex(x, y)
    c = 0
    for i in range(1, 1000):
        if abs(c) > 2:
            return rgb_conv(i)
        c = c * c + c0
    return (0, 0, 0)
```

```
# creating the new image in RGB mode
img = Image.new('RGB', (WIDTH, int(WIDTH / 2)))
pixels = img.load()

for x in range(img.size[0]):

    # displaying the progress as percentage
    print("%.2f %%" % (x / WIDTH * 100.0))
    for y in range(img.size[1]):
        pixels[x, y] = mandelbrot((x - (0.75 * WIDTH)) / (WIDTH / 4),
                                  (y - (WIDTH / 4)) / (WIDTH / 4))

# to display the created fractal after
# completing the given number of iterations
img.show()
```



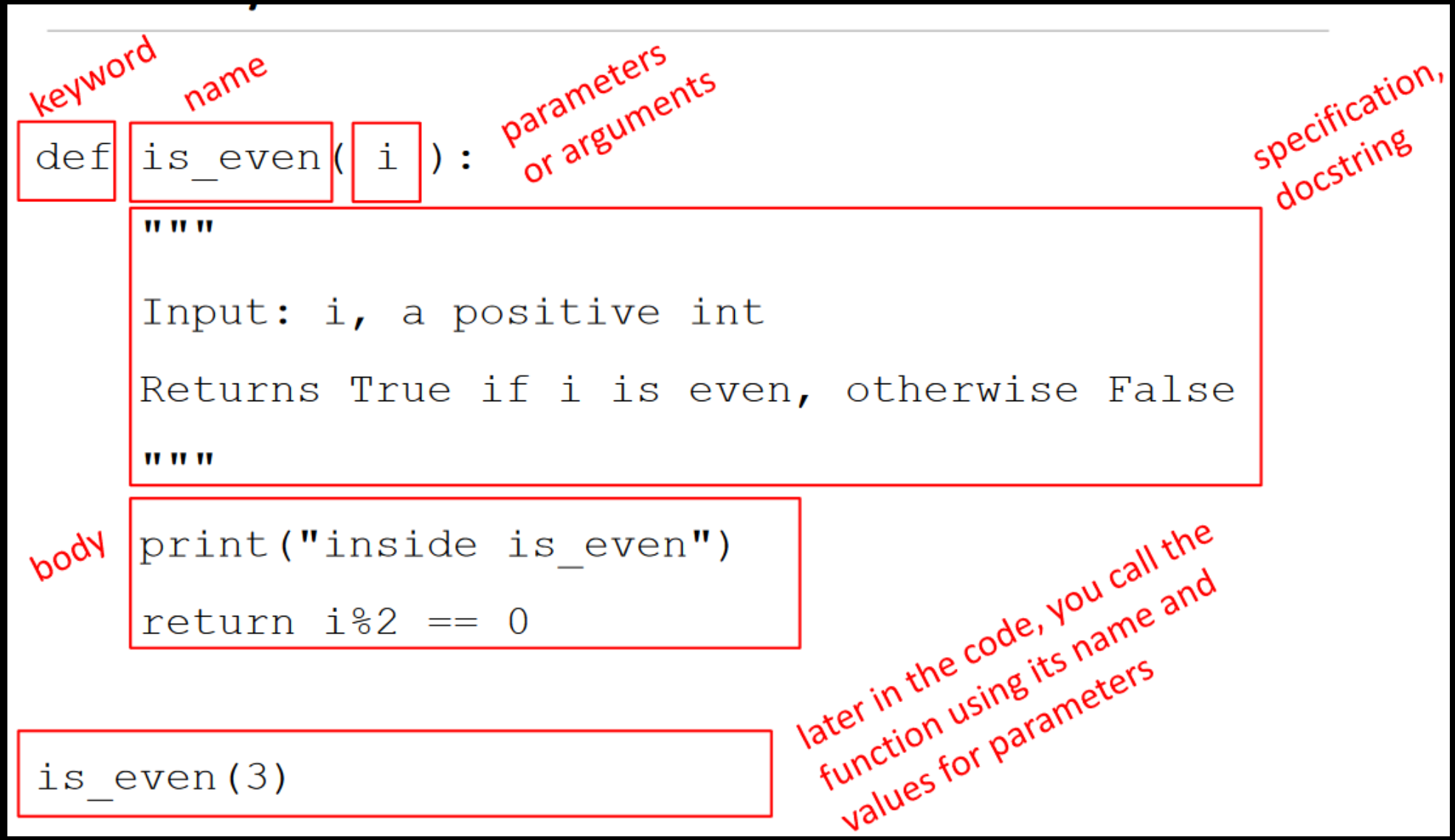
Another view point

The following slides are from MIT's Version of this course.

Dr. Ana Bell, Prof. Eric Grimson, Prof. John Guttag

https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/

Anatomy of a Python Function



```
def is_even( i ):
    """
    Input: i, a positive int
    Returns True if i is even, otherwise False
    """
    print("inside is_even")
    return i%2 == 0
```

run some commands

keyword

expression to evaluate and return

- **formal parameter** gets bound to the value of **actual parameter** when function is called
- new **scope/frame/environment** created when enter a function
- **scope** is mapping of names to objects

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```

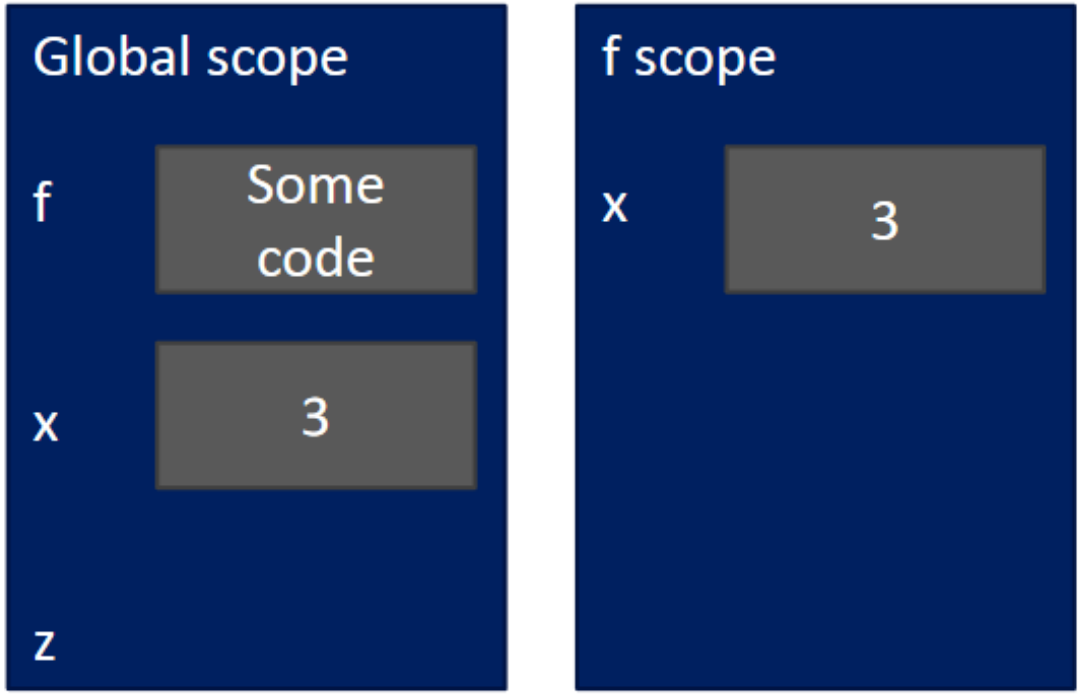
formal parameter

actual parameter

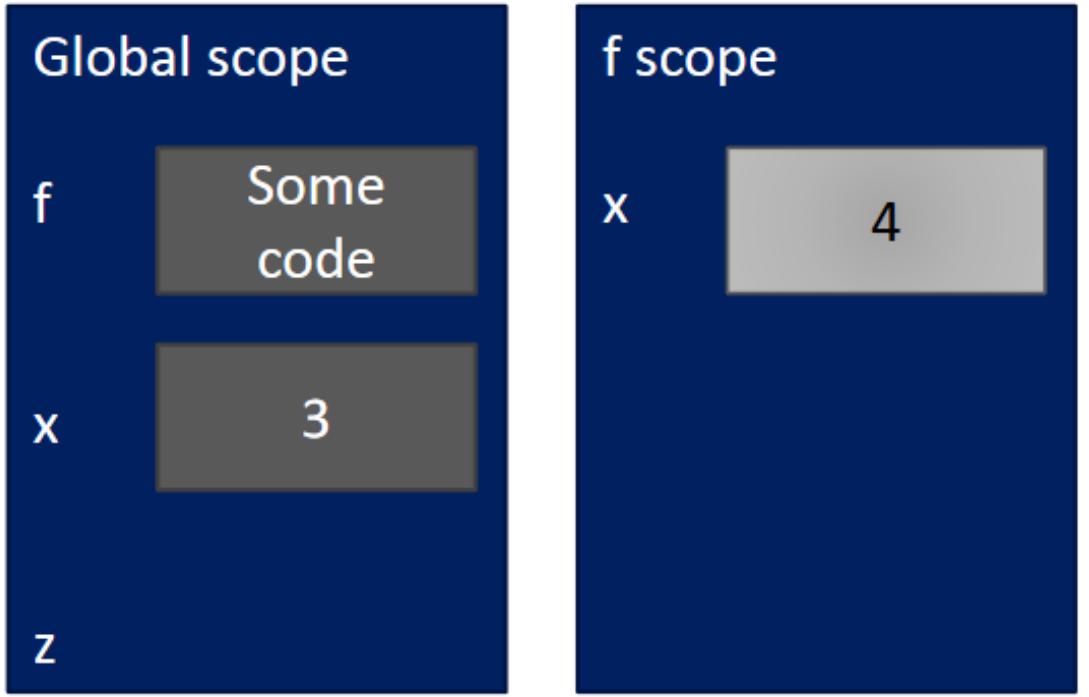
Function definition

Main program code
* initializes a variable x
* makes a function call f(x)
* assigns return of function to variable z

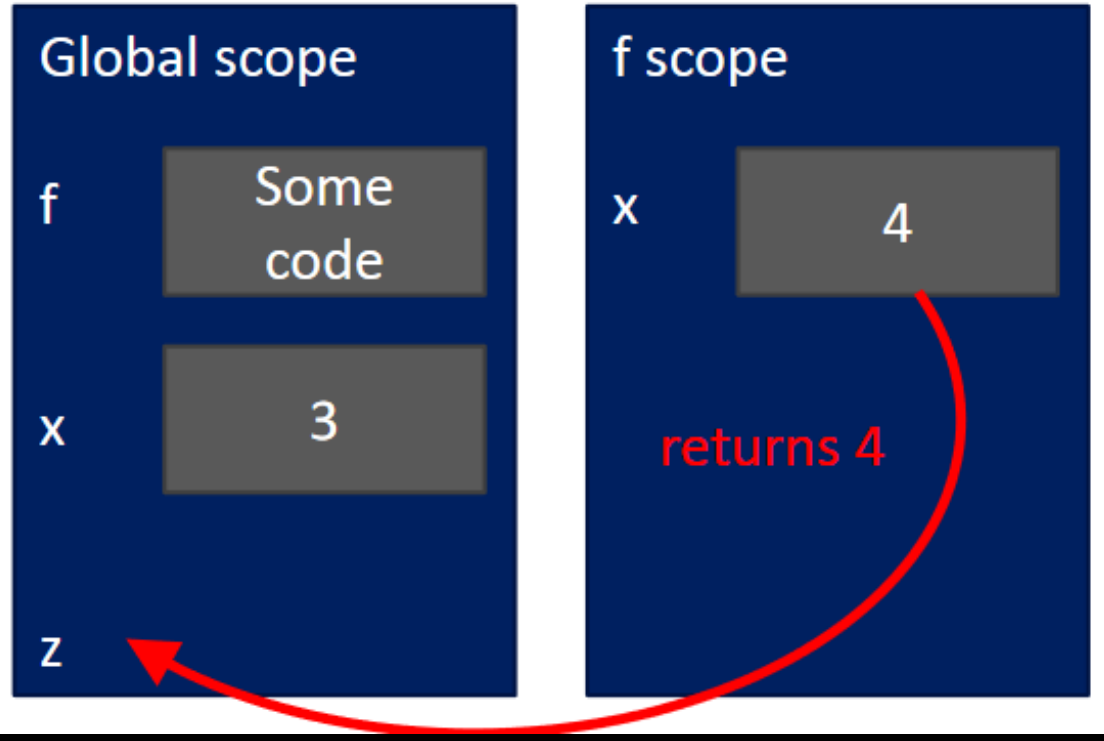
```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



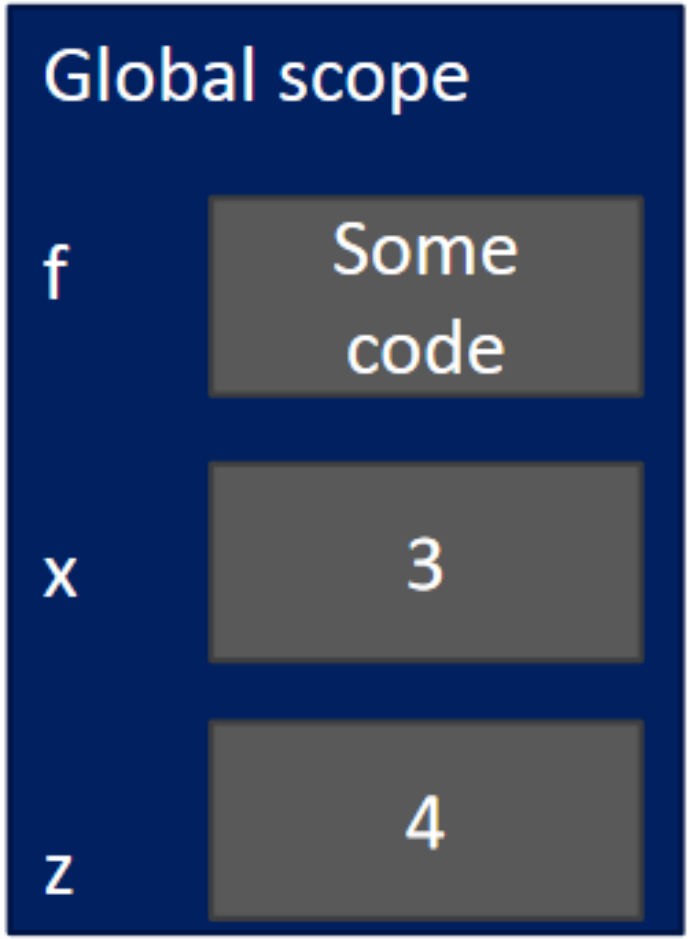
```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x


x = 3
z = f( x )
```



```
def is_even(i):
    """
    Input: i, a positive int
    Does not return anything
    """
    i%2 == 0
```

without a return statement

- Python returns the value **None, if no `return` given**
- represents the absence of a value

- arguments can take on any type, even functions

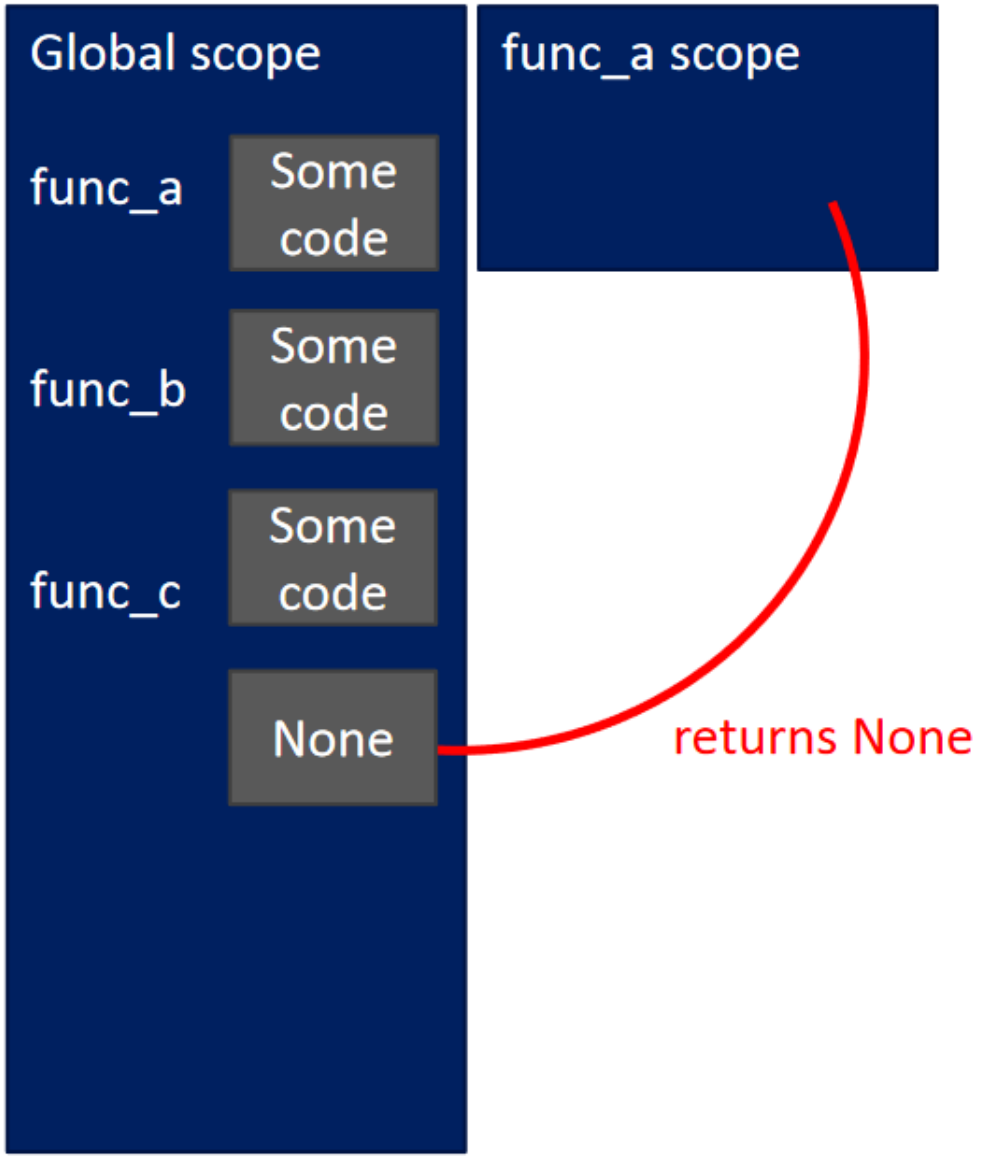
```
def func_a():
    print 'inside func_a'
def func_b(y):
    print 'inside func_b'
    return y
def func_c(z):
    print 'inside func_c'
    return z()
print func_a()
print 5 + func_b(2)
print func_c(func_a)
```

call `func_a`, takes no parameters

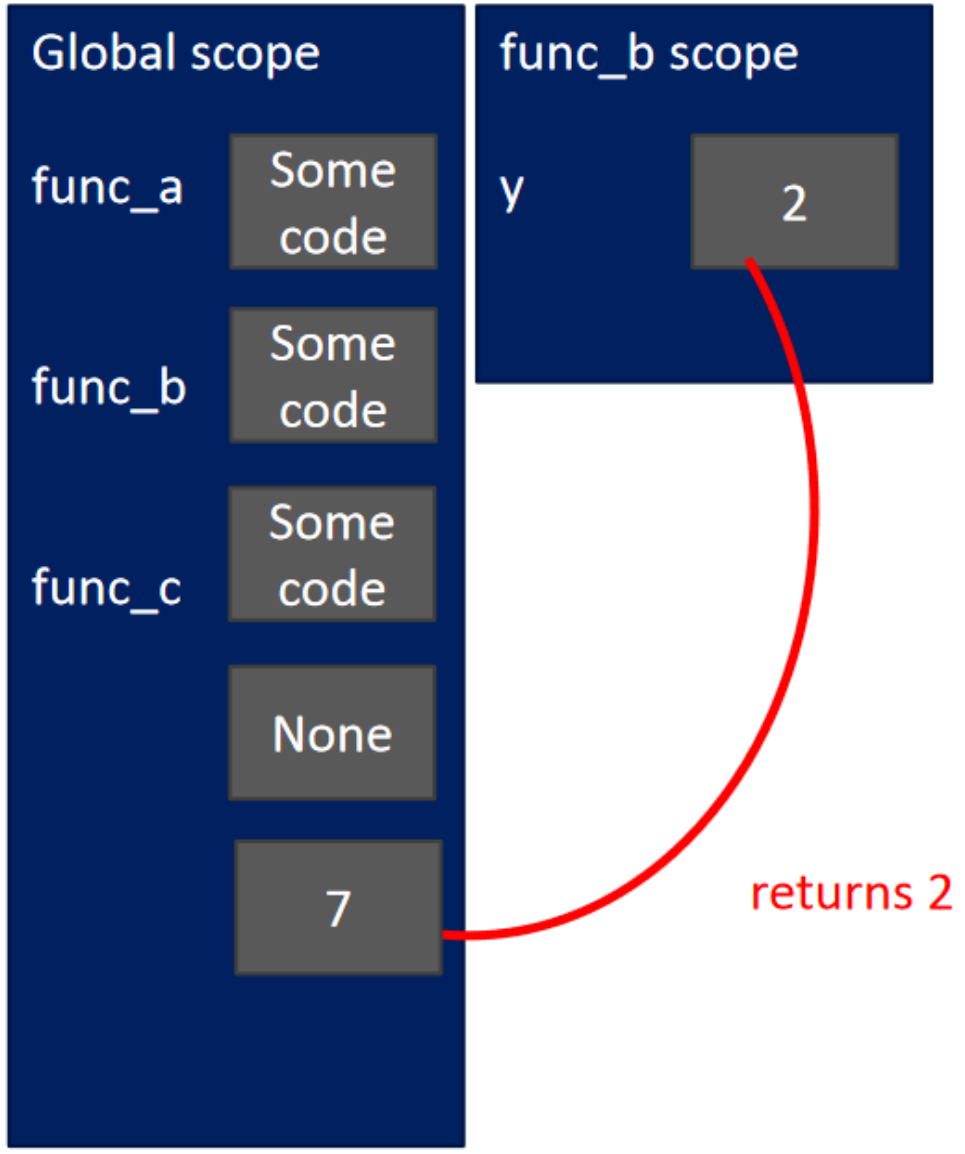
call `func_b`, takes one parameter

call `func_c`, takes one parameter, another function

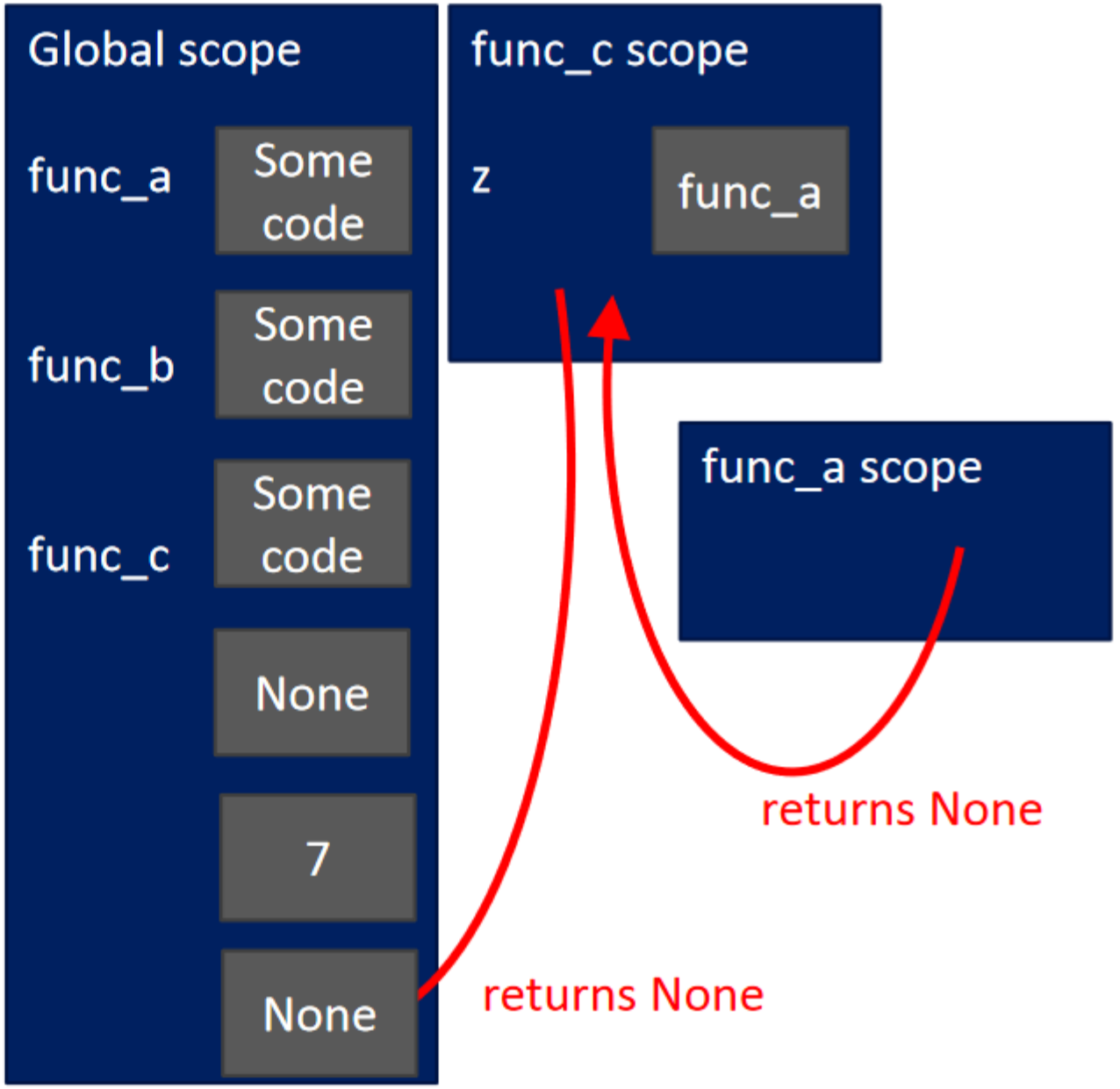
```
def func_a():
    print 'inside func_a'
def func_b(y):
    print 'inside func_b'
    return y
def func_c(z):
    print 'inside func_c'
    return z()
print func_a()
print 5 + func_b(2)
print func_c(func_a)
```



```
def func_a():
    print 'inside func_a'
def func_b(y):
    print 'inside func_b'
    return y
def func_c(z):
    print 'inside func_c'
    return z()
print func_a()
print 5 + func_b(2)
print func_c(func_a)
```



```
def func_a():
    print 'inside func_a'
def func_b(y):
    print 'inside func_b'
    return y
def func_c(z):
    print 'inside func_c'
    return z()
print func_a()
print 5 + func_b(2)
print func_c(func_a)
```



- inside a function, **can access** a variable defined outside
- inside a function, **cannot modify** a variable defined outside -- can using **global variables**, but frowned upon

```
def f(y):
    x = 1
    x += 1
    print(x)

x = 5
f(x)
print(x)
```

x is re-defined in scope of f

different x objects

```
def g(y):
    print(x)
    print(x + 1)

x = 5
g(x)
print(x)
```

x from outside g

x inside g is picked up from scope that called function g

```
def h(y):
    x += 1

x = 5
h(x)
print(x)
```

UnboundLocalError: local variable 'x' referenced before assignment

- inside a function, **can access** a variable defined outside
- inside a function, **cannot modify** a variable defined outside -- can using **global variables**, but frowned upon

```
def f(y):
    x = 1
    x += 1
    print(x)

x = 5
f(x)
print(x)
```

```
def g(y):
    print(x)

x = 5
g(x)
print(x)
```

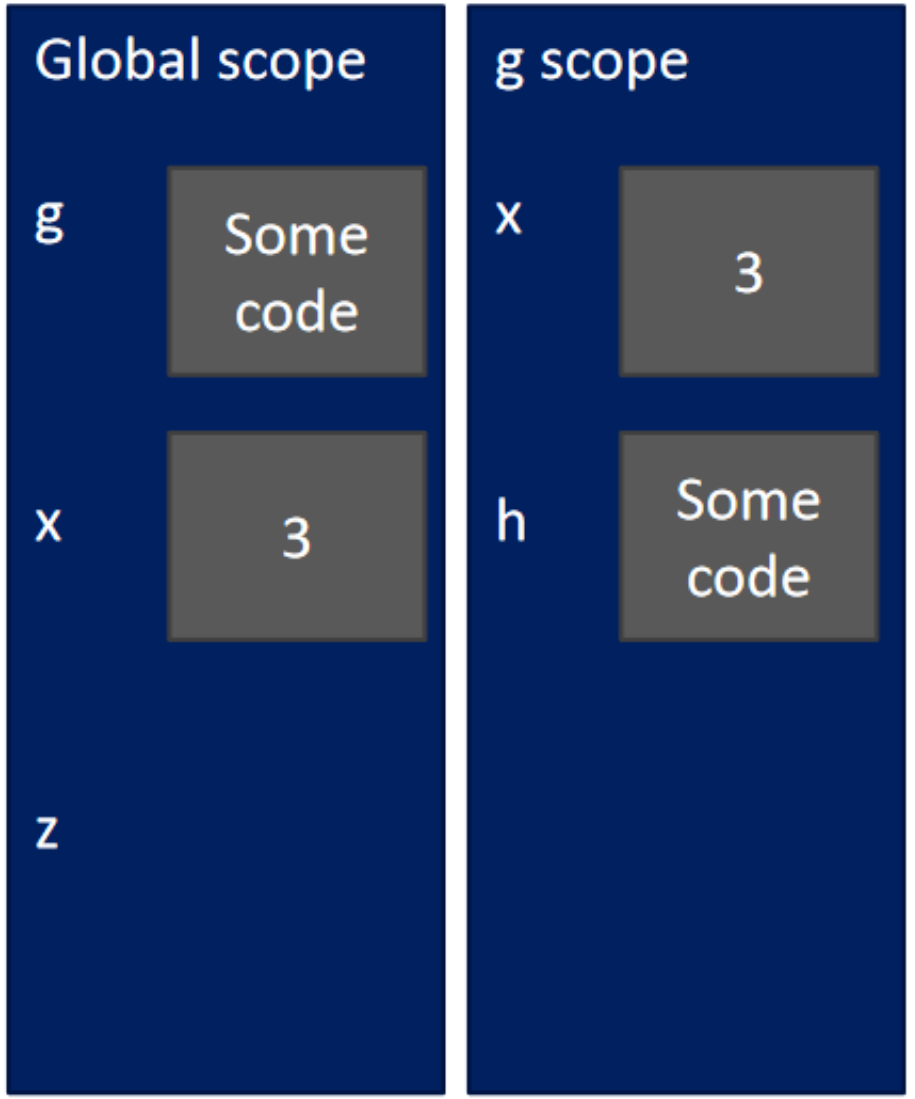
```
def h(y):
    x += 1

x = 5
h(x)
print(x)
```

x from global/main program scope

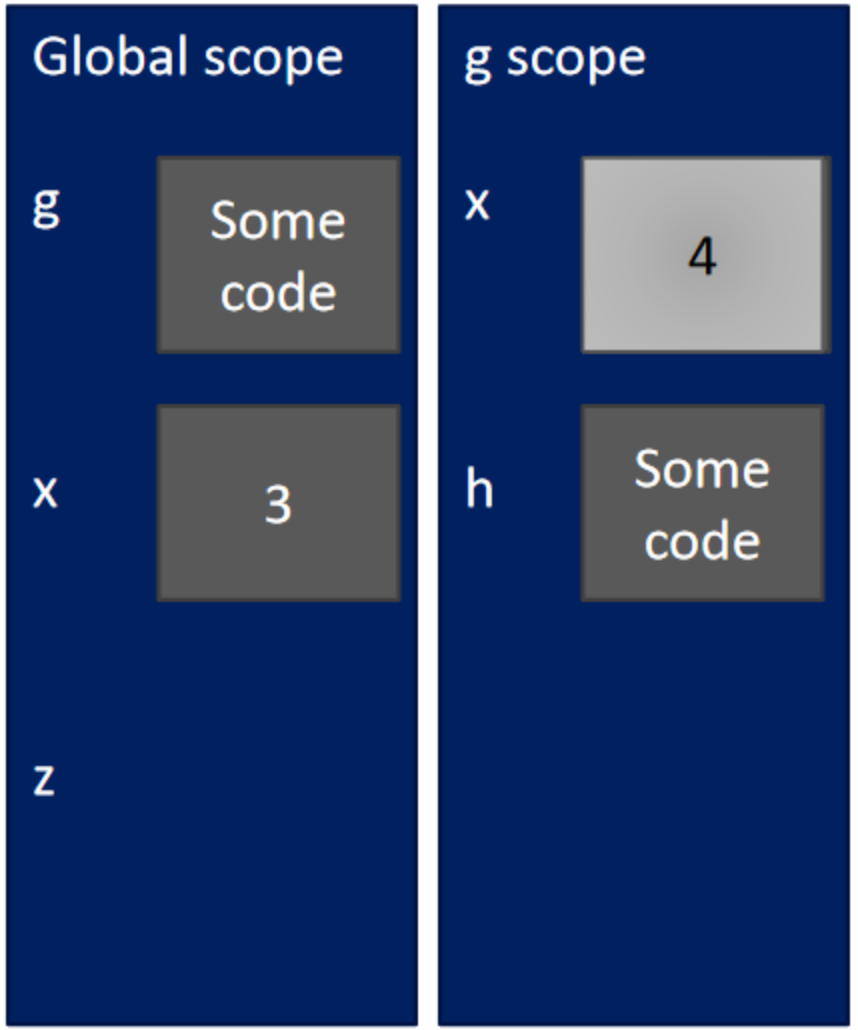
```
def g(x):
    def h():
        x = 'abc'
    x = x + 1
    print('g: x =', x)
    h()
    return x
```

Some code

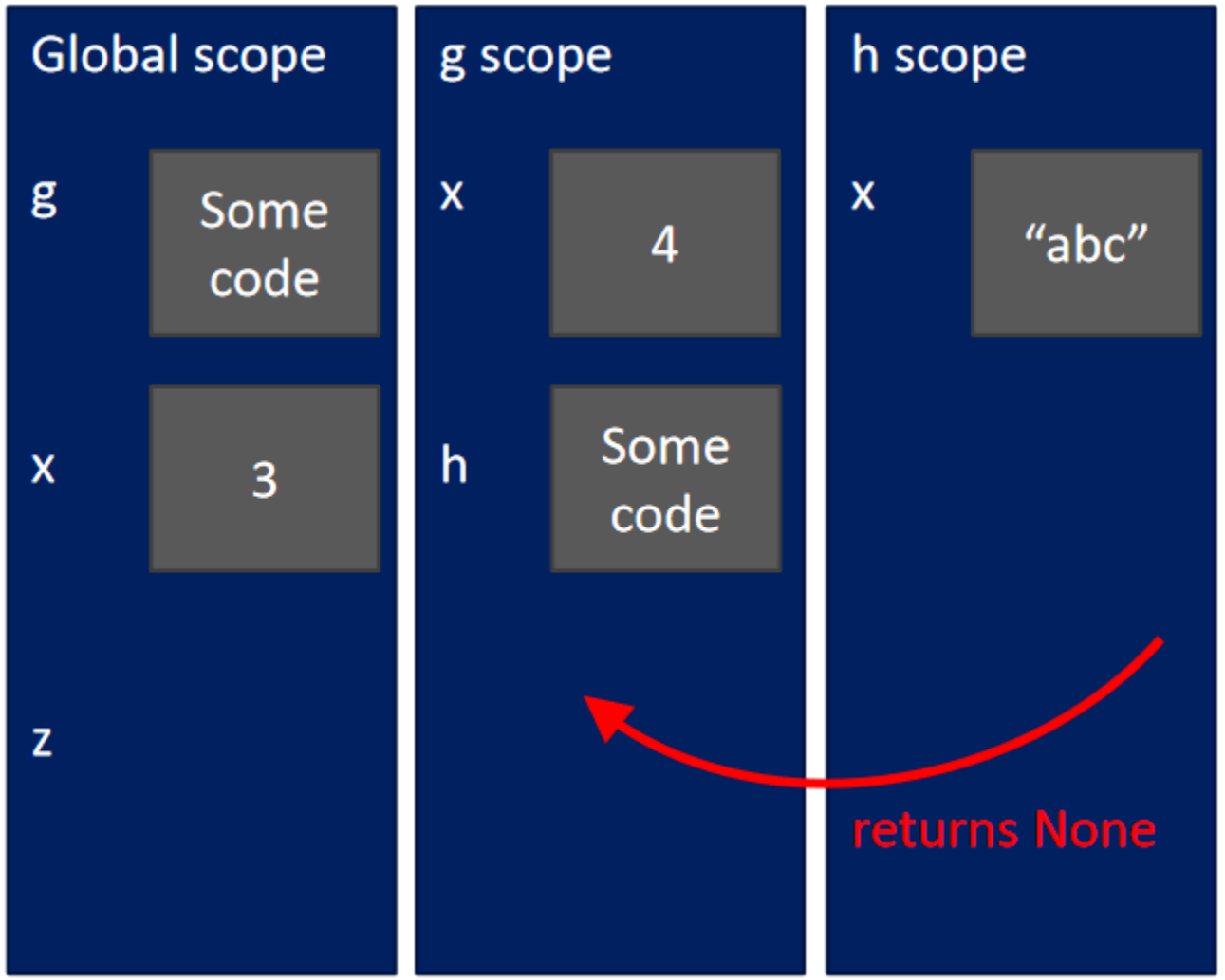
```
def g(x):
    def h():
        x = 'abc'
    x = x + 1
    print('g: x =', x)
    h()
    return x


x = 3
z = g(x)
```



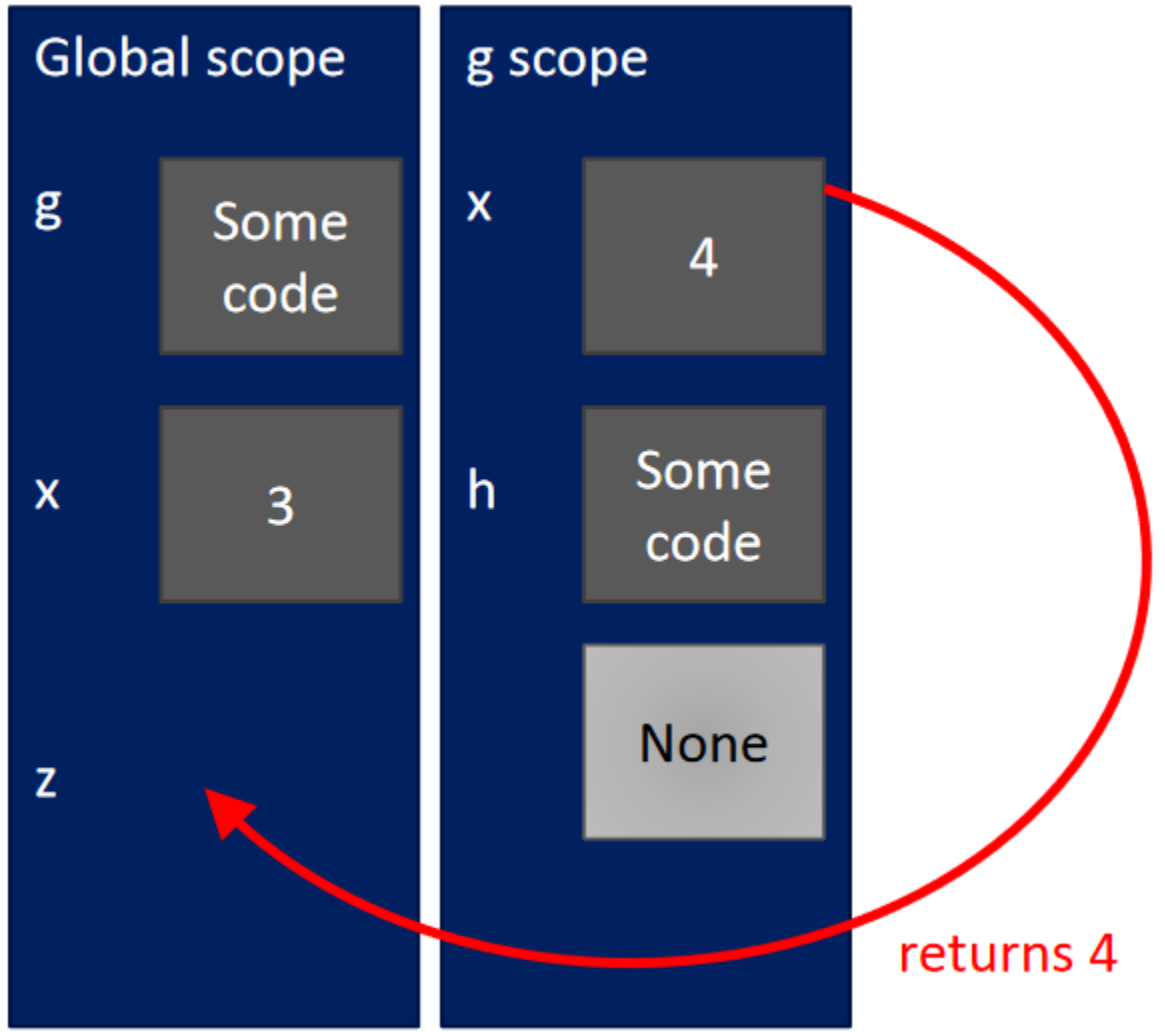
```
def g(x):
    def h():
        x = 'abc'
    x = x + 1
    print('g: x =', x)
    h()
    return x


x = 3
z = g(x)
```



```
def g(x):
    def h():
        x = 'abc'
    x = x + 1
    print('g: x =', x)
    h()
    return x

x = 3
z = g(x)
```



```
def g(x):
    def h():
        x = 'abc'
    x = x + 1
    print('g: x =', x)
    h()
    return x

x = 3
z = g(x)
```



```
def g(x):
    def h():
        x = 'abc'
    x = x + 1
    print('g: x =', x)
    h()
    return x

x = 3
z = g(x)
```