CMSC201
Computer Science I for Majors

Lecture 11 – Functions (cont)

Last Class We Covered

- Functions
  - Why they're useful
  - When you should use them
- Defining functions
- Calling functions
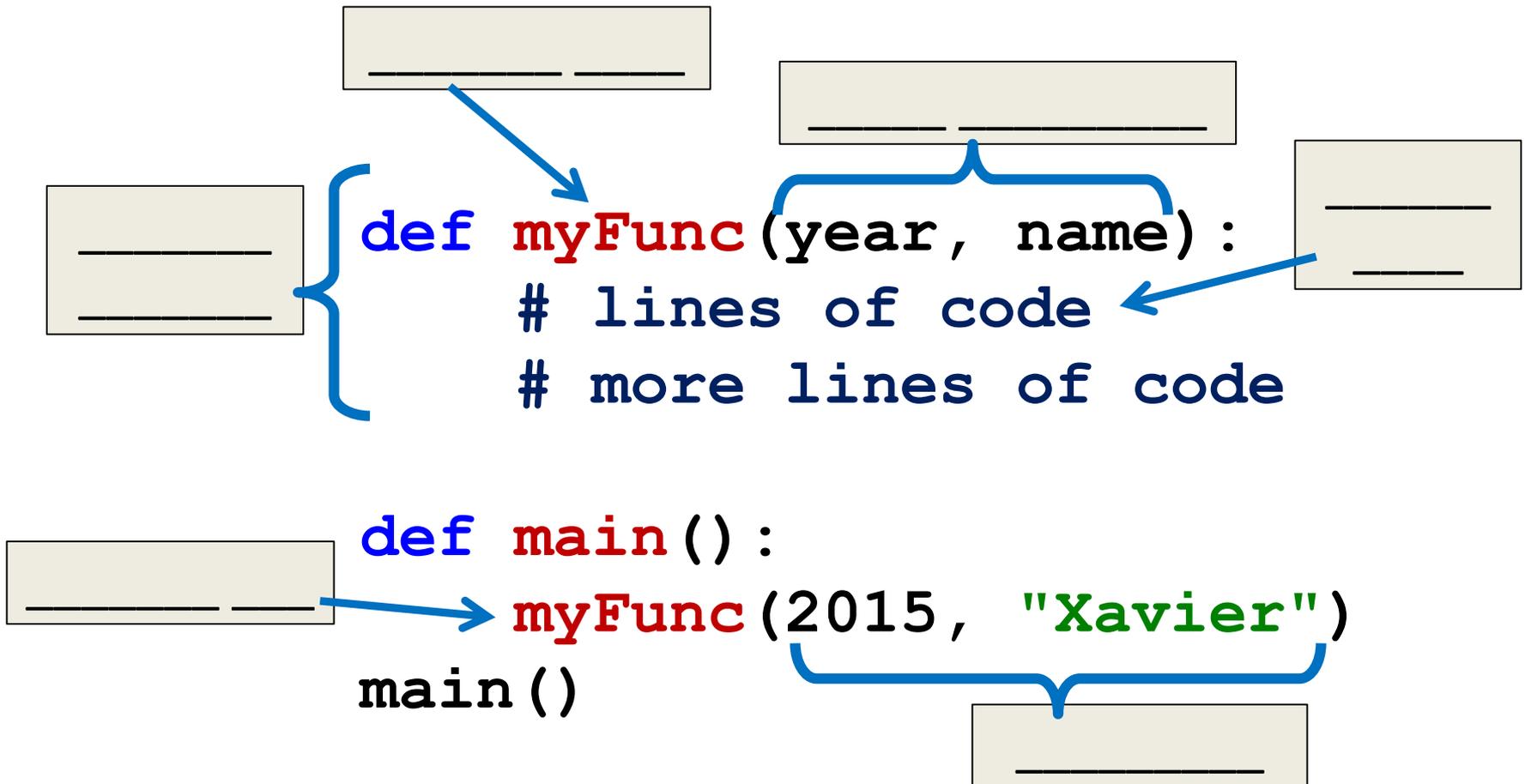- Variable scope
- Passing arguments

 www.umbc.edu

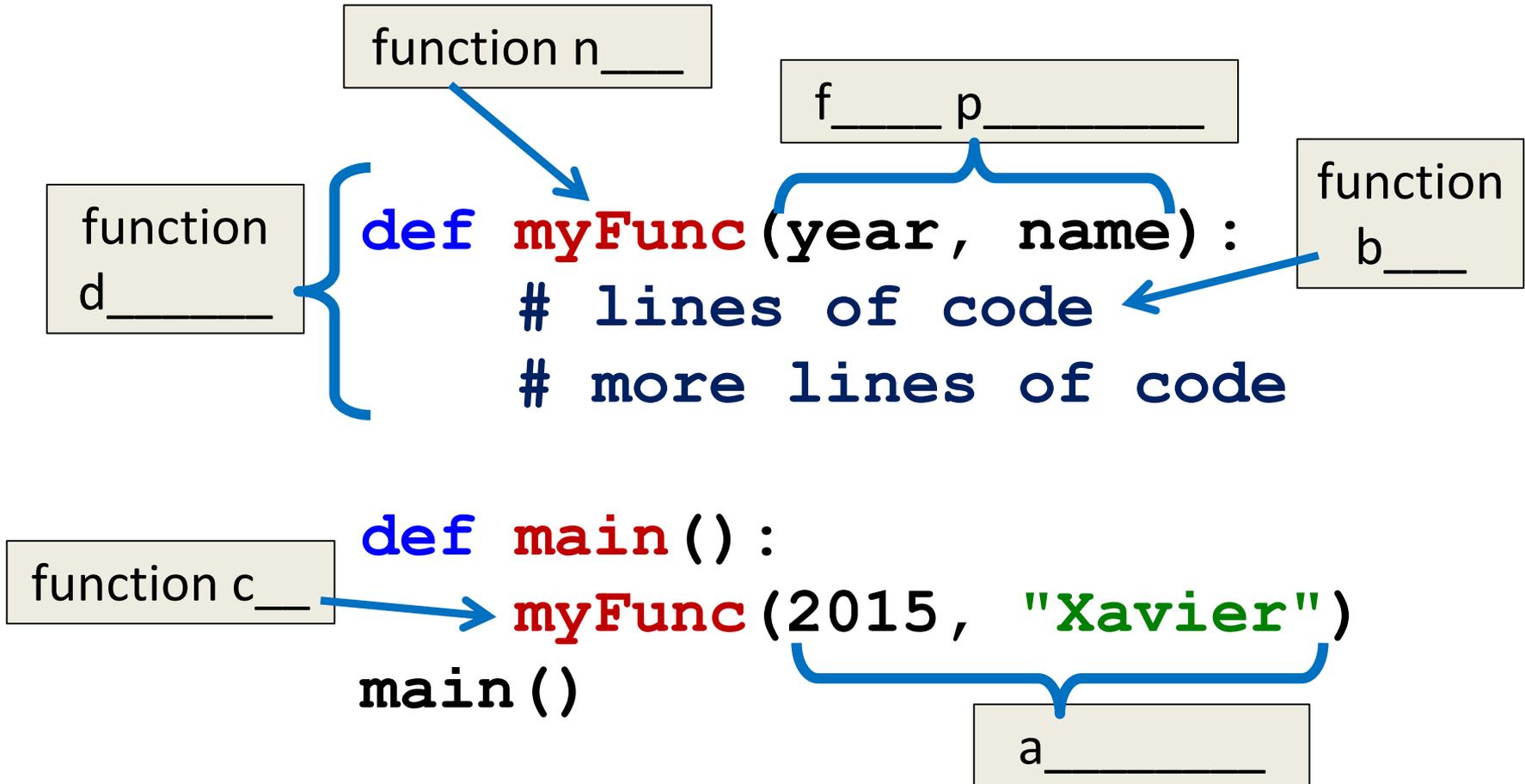# Any Questions from Last Time?

Today's Objectives

- To introduce value-returning functions
  - Common problems
  - Solutions to common problems
- To better grasp how values in the scope of a function actually work
- To practice function calls

Review: Parts of a Function
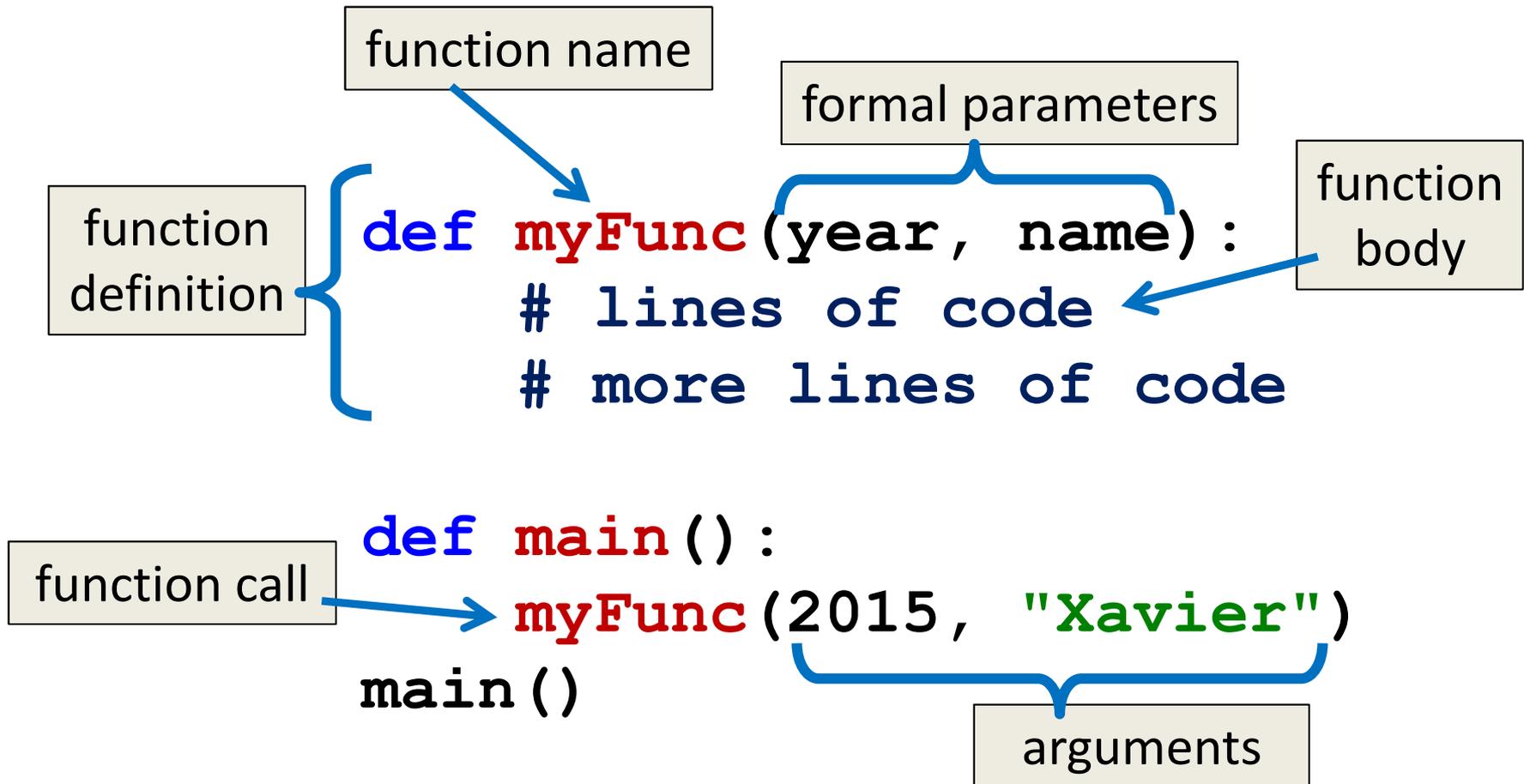
Function Vocabulary

_____ _____

_____ _____

_____

_____
_____

```
def myFunc(year, name):
    # lines of code
    # more lines of code
```

_____
_____

_____ _____

```
def main():
    myFunc(2015, "Xavier")
main()
```

_____

Function Vocabulary

function n____

f_____ p_____

function b____

function d_____

```
def myFunc(year, name):
    # lines of code
    # more lines of code
```

function c___

```
def main():
    myFunc(2015, "Xavier")
main()
```

a_____

 www.umbc.edu

Function Vocabulary

function name

formal parameters

function definition

```
def myFunc(year, name):
    # lines of code
    # more lines of code
```

function body

```
def main():
    myFunc(2015, "Xavier")
main()
```

function call

arguments

 www.umbc.edu

File Layout and Constants

www.umbc.edu

Layout of a Python File

header comment →

```python
# File:      gradeGetter.py
# Author:    Dr. Gibson
```

constants →

```python
MIN_GRADE = 0
MAX_GRADE = 100
```

definitions and function headers for all functions other than **main()** →

```python
# getInput() gets a value between...
# Input:      minn; an integer...
def getInput(minn, maxx):
    msg = "Please enter..."
    # more code here
    return val
```

main() definition →

```python
def main():
    grade = getInput(MIN_GRADE, MAX_GRADE)
    print("You got a", grade)
```

call to **main()** →

```python
main()
```

Global Constants

- ***Globals*** are variables declared outside of any function (including `main()`)

- Accessible globally in your program
  - To all functions and code

- Your programs <u>may not</u> have global variables

- Your programs <u>may</u> use global **constants**
  - In fact, constants <u>should</u> be global

 www.umbc.edu

Return Statements

Giving Information to a Function

- Passing parameters provides a mechanism for <u>initializing</u> the variables in a function

- Parameters act as *inputs* to a function

- We can call a function many times and get <u>different results</u> by changing its parameters

Getting Information from a Function

- We've already seen numerous examples of functions that <u>return</u> values

    `int()` , **`str()`**, **`input()`**, etc.

- For example, **`int()`**
  - Takes in any string as its parameter
  - Processes the digits in the string
  - And <u>returns</u> an integer value

14   www.umbc.edu

Functions that Return Values

- To have a function return a value after it is called, we need to use the **return** keyword

```
def square(num):
    ans = num * num
    # return the square
    return ans
```

Handling Return Values

- When Python encounters **`return`**, it
  - Exits the function (immediately!)
    - Even if it's not the end of the function
  - Returns control back to where the function was called from

- The expression in the return statement is sent back to the caller as a ***return value***

Code Trace: Return from `square()`

Let's follow the flow of the code

```python
def main():
    x = 5
    y = square(x)
    print(y)
main()
```

```python
def square(num):
    ans = num * num
    return ans
```

Step 1: Call `main()`

Code Trace: Return from `square()`

Let's follow the flow of the code

```python
def main():
    x = 5
    y = square(x)
    print(y)
main()
```

```python
def square(num):
    ans = num * num
    return ans
```

→ (arrow pointing to `main()`)

Step 1: Call **main()**

Step 2: Pass control to **def main()**

Code Trace: Return from `square()`

Let's follow the flow of the code

x: [5]

```python
def main():
    x = 5
    y = square(x)
    print(y)
main()
```

```python
def square(num):
    ans = num * num
    return ans
```

Step 1: Call `main()`
Step 2: Pass control to `def main()`
Step 3: Set `x = 5`

Code Trace: Return from `square()`

Let's follow the flow of the code

`x:` `5`

```python
def main():
    x = 5
    y = square(x)
    print(y)
main()
```

```python
def square(num):
    ans = num * num
    return ans
```

Step 1: Call `main()`
Step 2: Pass control to `def main()`
Step 3: Set `x = 5`
Step 4: See the function call to `square()`

20 www.umbc.edu

Code Trace: Return from `square()`

Let's follow the flow of the code

`x:` `5`

```
def main():
    x = 5
    y = square(x)
    print(y)
main()
```

argument:
5

```
def square(num):
    ans = num * num
    return ans
```

Step 1: Call `main()`
Step 2: Pass control to `def main()`
Step 3: Set `x = 5`
Step 4: See the function call to `square()`
Step 5: Pass control from `main()` to `square()`, sending the argument `5`

 www.umbc.edu

Code Trace: Return from `square()`

Let's follow the flow of the code

x: 5

```
def main():
    x = 5
    y = square(x)
    print(y)
main()
```

→ `def square(num):`
    `ans = num * num`
    `return ans`

num: 5

Step 1: Call `main()`
Step 2: Pass control to `def main()`
Step 3: Set `x = 5`
Step 4: See the function call to `square()`
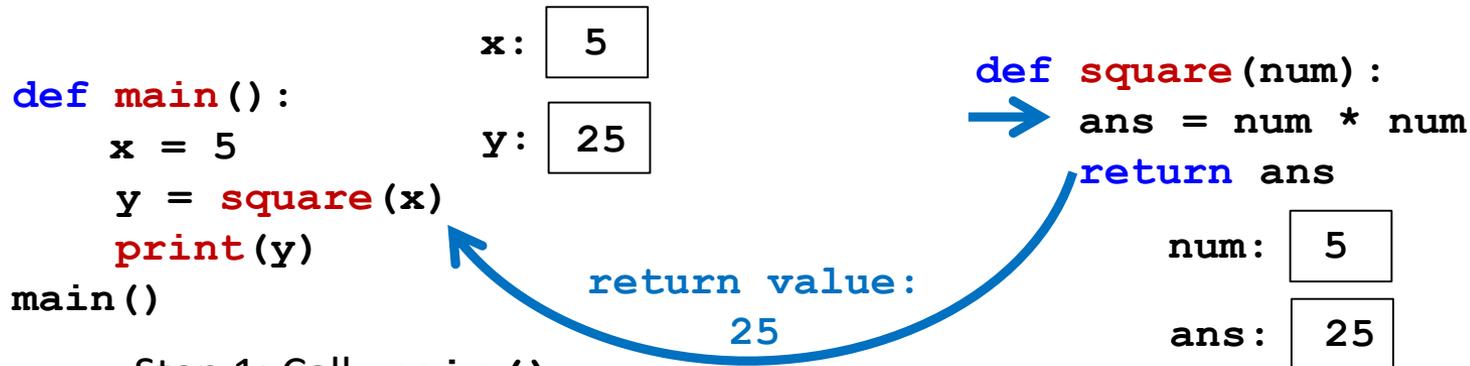Step 5: Pass control from `main()` to `square()`, sending the argument **5**
Step 6: Set the value of the formal parameter **num** in `square()` to **5**

www.umbc.edu

Code Trace: Return from `square()`

Let's follow the flow of the code

`x:` 5

```python
def main():
    x = 5
    y = square(x)
    print(y)
main()
```

→ `def square(num):`
    `ans = num * num`
    `return ans`

`num:` 5

`ans:` 25

Step 1: Call `main()`
Step 2: Pass control to `def main()`
Step 3: Set `x = 5`
Step 4: See the function call to `square()`
Step 5: Pass control from `main()` to `square()`, sending the argument `5`
Step 6: Set the value of the formal parameter `num` in `square()` to `5`
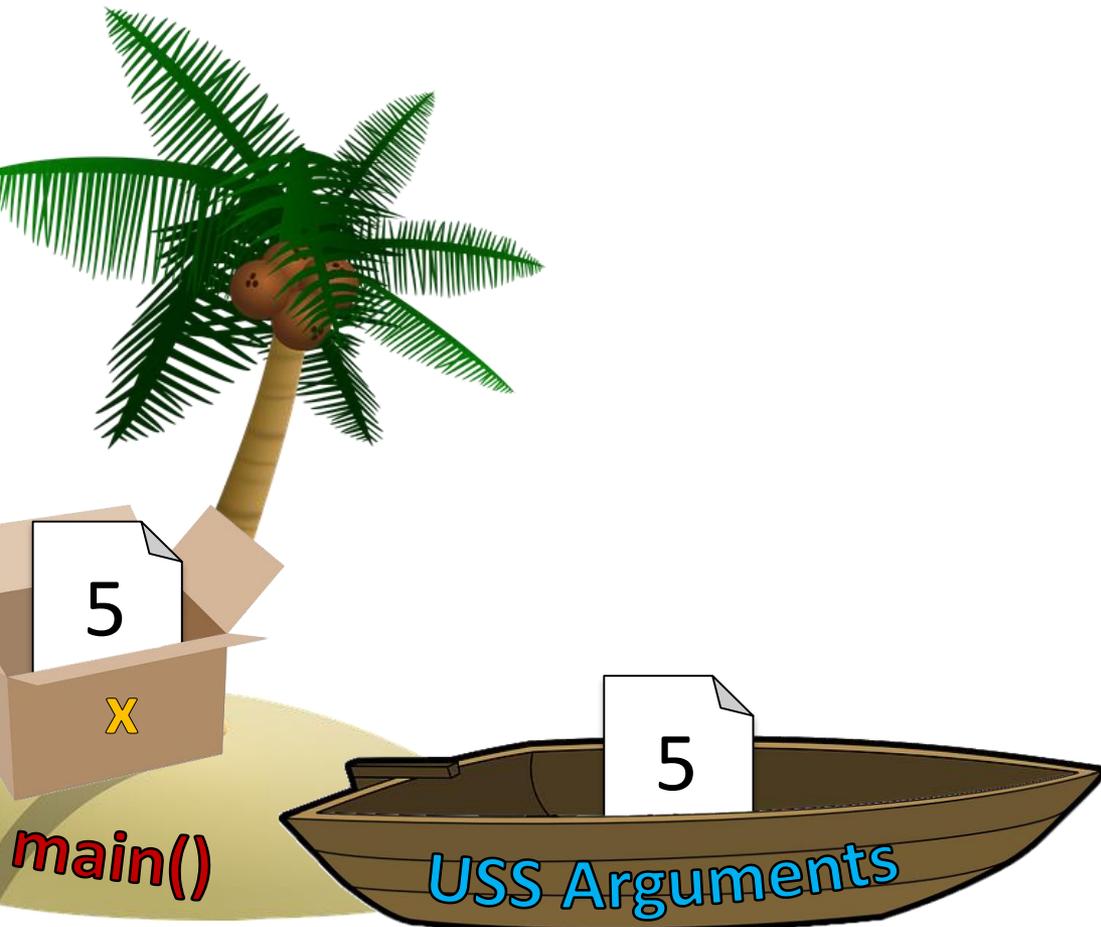Step 7: Calculate `ans = num * num`

 www.umbc.edu

Code Trace: Return from `square()`

Let's follow the flow of the code

x: 5

y: 25

```
def main():
    x = 5
    y = square(x)
    print(y)
main()
```

```
def square(num):
    ans = num * num
    return ans
```

num: 5

ans: 25

return value:
25

Step 1: Call **main()**
Step 2: Pass control to **def main()**
Step 3: Set **x = 5**
Step 4: See the function call to **square()**
Step 5: Pass control from **main()** to **square()**, sending the argument **5**
Step 6: Set the value of the formal parameter **num** in **square()** to **5**
Step 7: Calculate **ans = num * num**
Step 8: Return the value **25** to **main()** and set **y =** the returned value

Code Trace: Return from `square()`

Let's follow the flow of the code

x: | 5 |

y: | 25 |

```python
def main():
    x = 5
→   y = square(x)
    print(y)
main()
```

```python
def square(num):
    ans = num * num
    return ans
```

Step 1: Call **main()**
Step 2: Pass control to **def main()**
Step 3: Set **x = 5**
Step 4: See the function call to **square()**
Step 5: Pass control from **main()** to **square()**, sending the argument **5**
Step 6: Set the value of the formal parameter **num** in **square()** to **5**
Step 7: Calculate **ans = num * num**
Step 8: Return the value **25** to **main()** and set **y =** the returned value
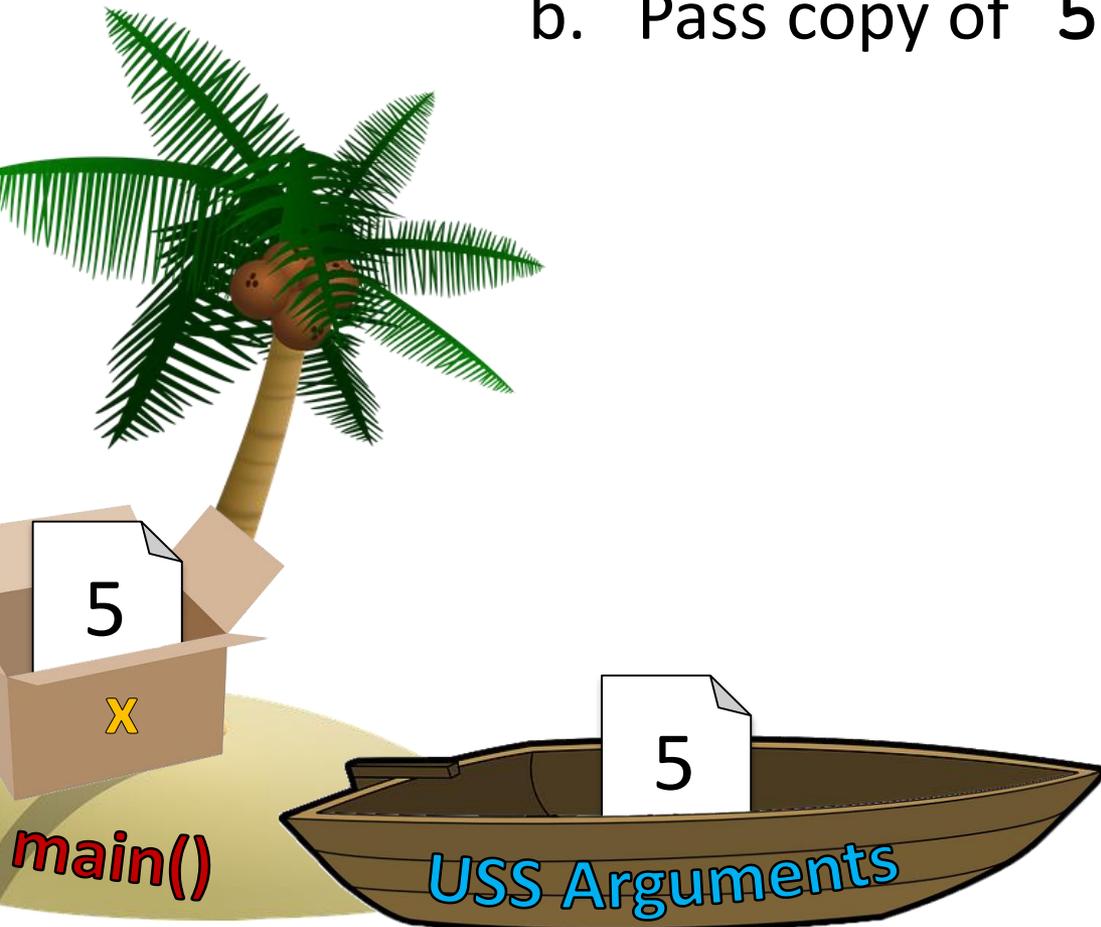Step 9: Print value of **y**

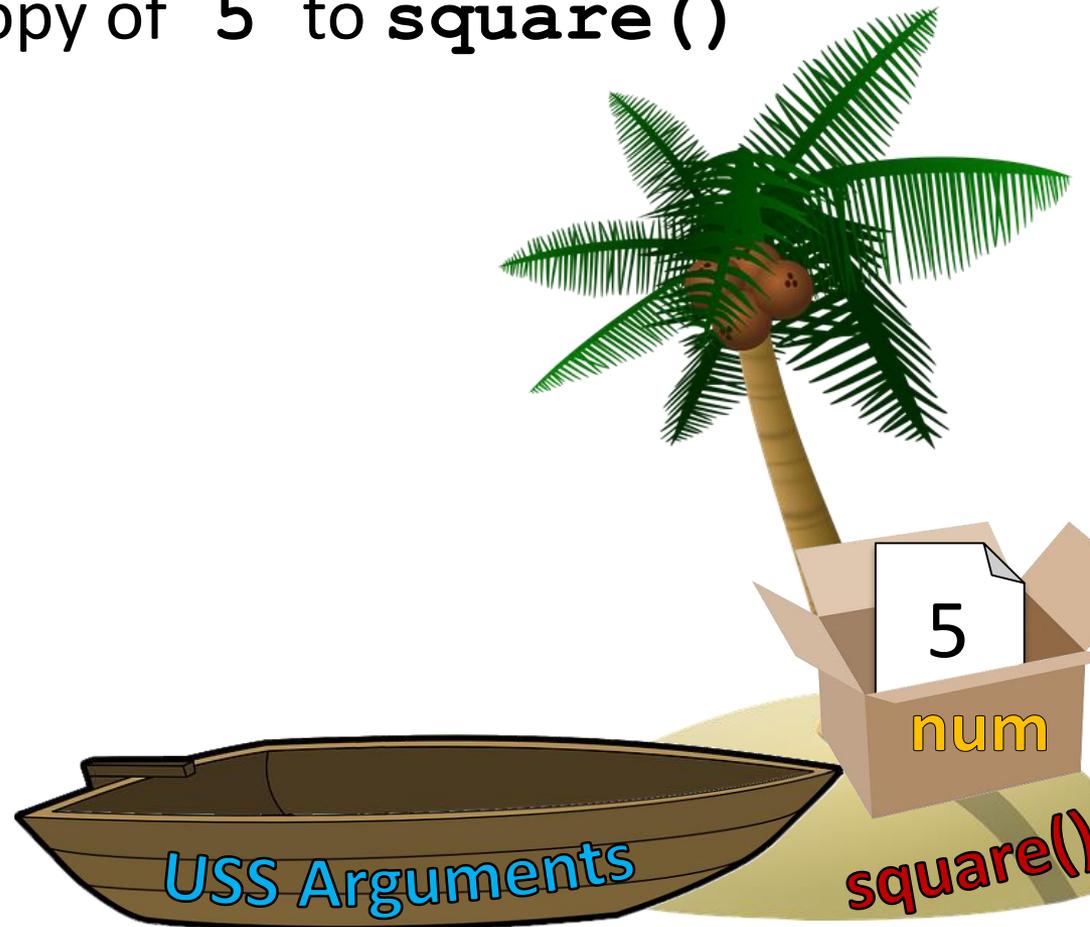www.umbc.edu

Island Example

5

x

main()

 www.umbc.edu

1. Function **square()** is called
   a. Make copy of **x**'s value

All materials copyright UMBC and Dr. Katherine Gibson unless otherwise noted   www.umbc.edu

1. Function **square()** is called
   a. Make copy of **x**'s value
   b. Pass copy of **5** to **square()**

5

x

5

main()

USS Arguments

num

square()

 www.umbc.edu

1. Function **square()** is called
   a. Make copy of **x**'s value
   b. Pass copy of **5** to **square()**

5

x

main()

5

num

USS Arguments

square()

All materials copyright UMBC and Dr. Katherine Gibson unless otherwise noted

1. Function **square()** is called
   a. Make copy of **x**'s value
   b. Pass copy of **5** to **square()**

5

x

main()

5

num

square()

USS Arguments

All materials copyright UMBC and Dr. Katherine Gibson unless otherwise noted   www.umbc.edu
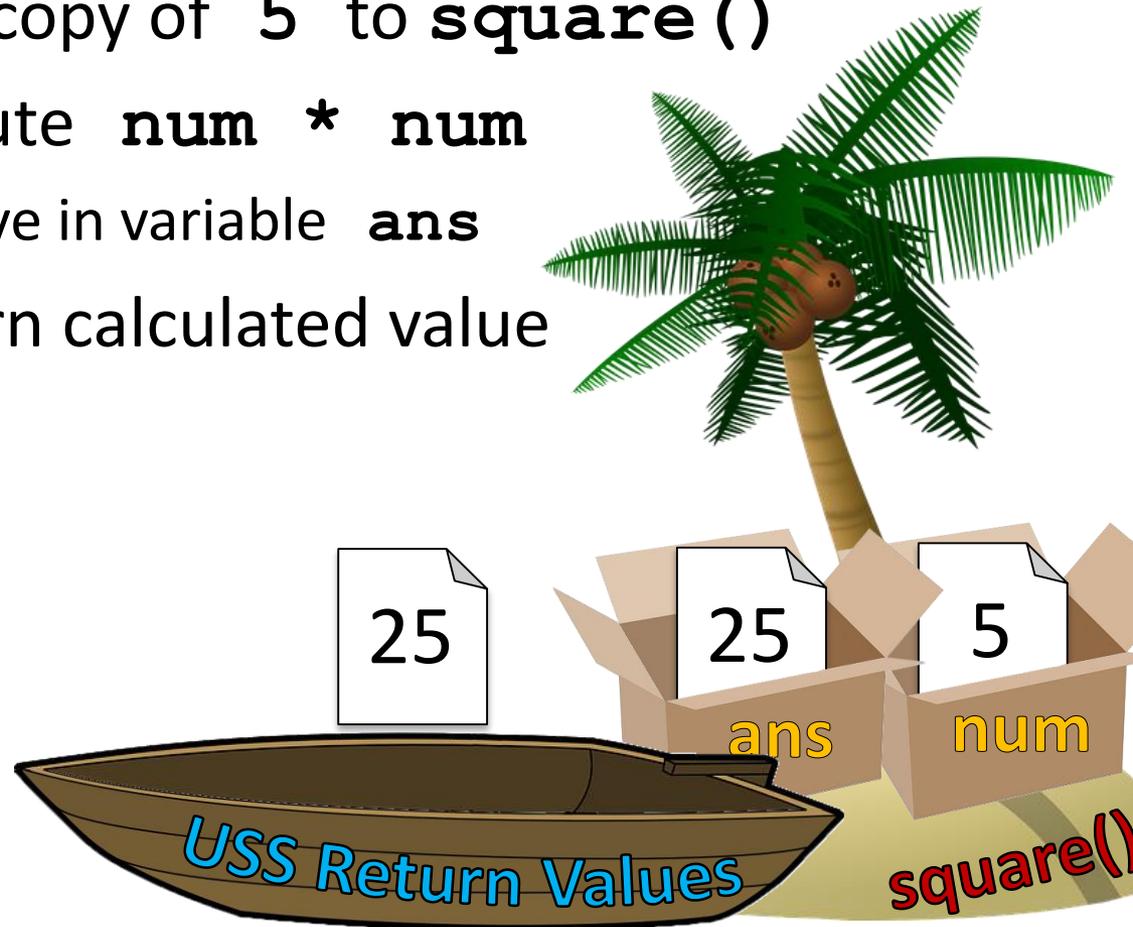
1. Function **square()** is called

    a. Make copy of **x**'s value

    b. Pass copy of **5** to **square()**

    c. Execute **num * num**

        a. Save in variable **ans**

5

x

25

5

ans   num

main()

square()

 www.umbc.edu

1. Function **square()** is called
   a. Make copy of **x**'s value
   b. Pass copy of **5** to **square()**
   c. Execute **num * num**
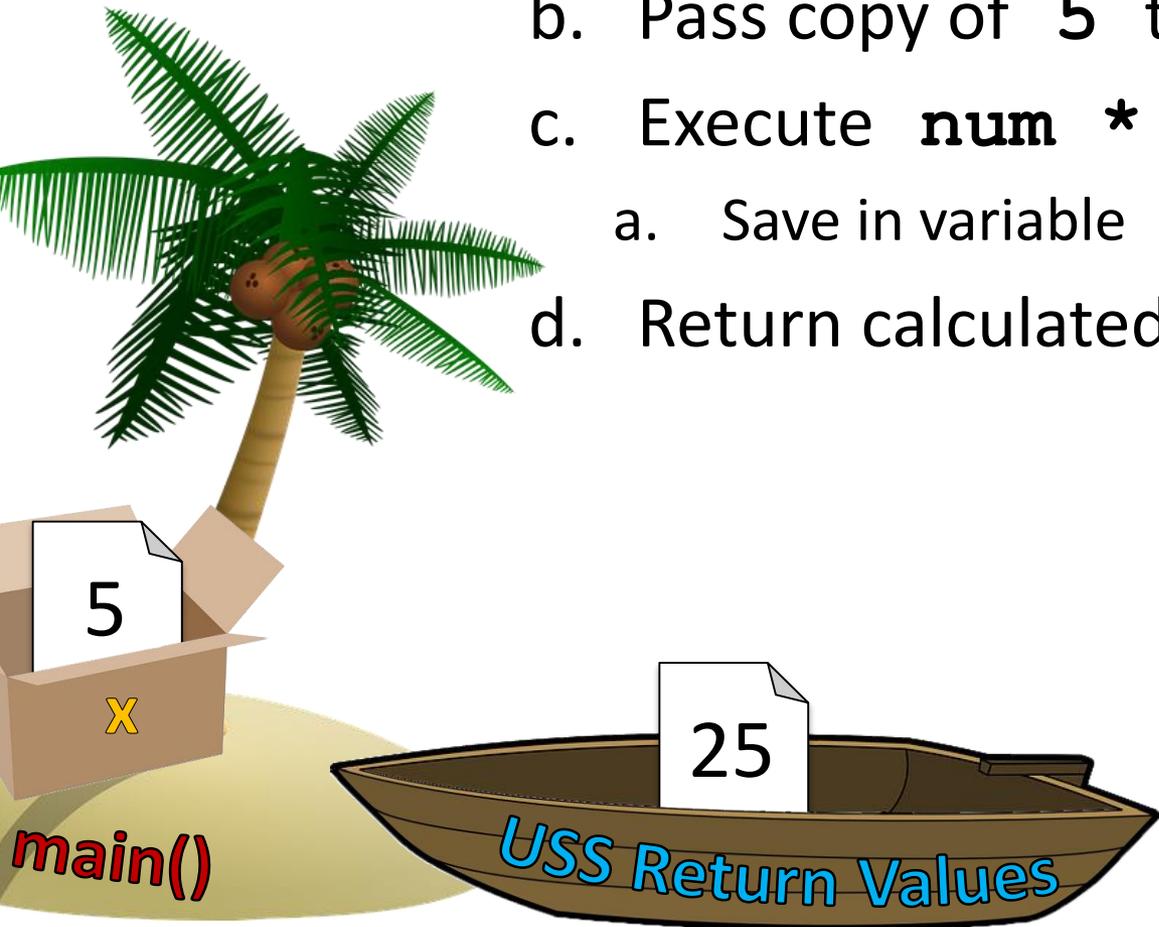   a. Save in variable **ans**
   d. Return calculated value

5

x

main()

25

ans

5

num

USS Return Values

square()

1. Function **square()** is called

   a. Make copy of **x**'s value

   b. Pass copy of **5** to **square()**

   c. Execute **num * num**

      a. Save in variable **ans**
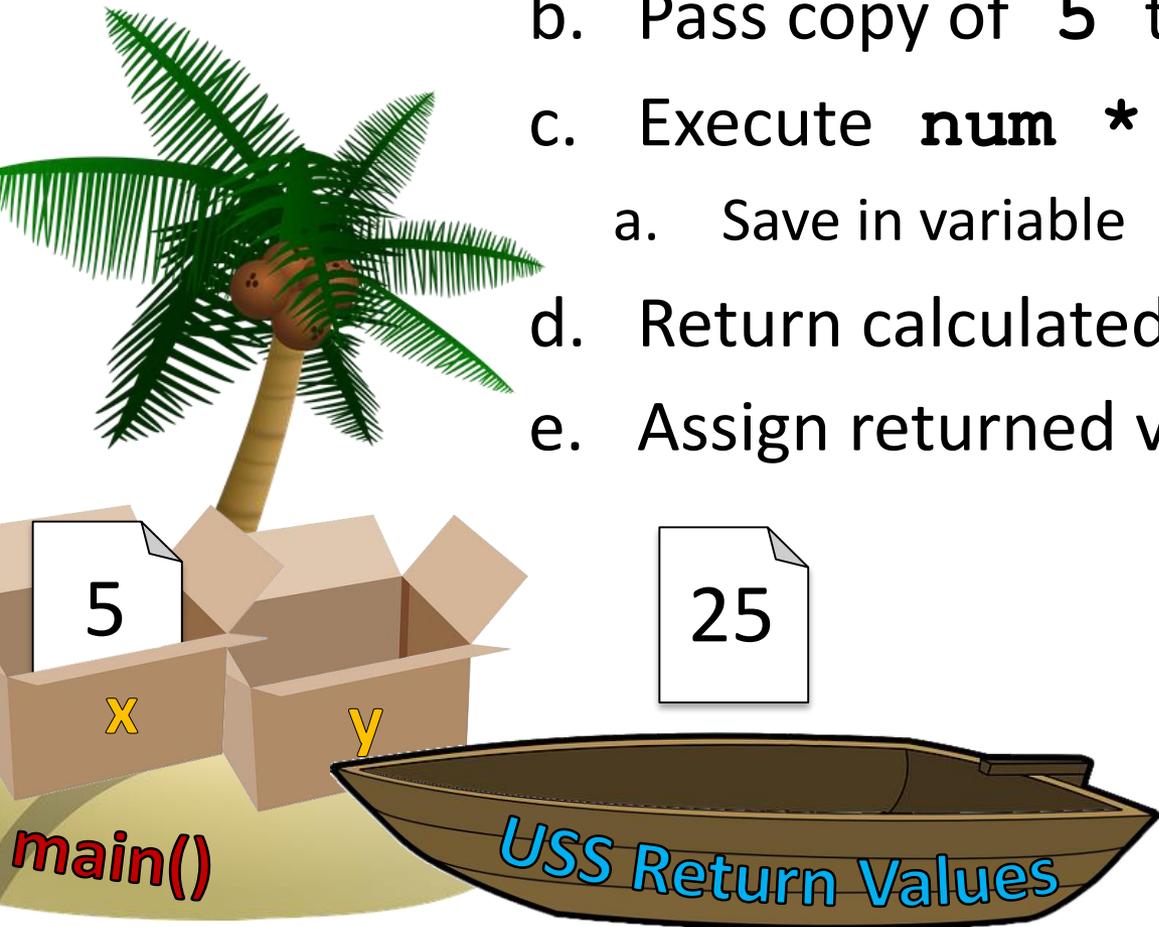
   d. Return calculated value

5

x

main()

25

25

ans

5

num

USS Return Values

square()

1. Function **`square()`** is called
   a. Make copy of **x**'s value
   b. Pass copy of **5** to **`square()`**
   c. Execute **`num * num`**
      a. Save in variable **`ans`**
   d. Return calculated value

5

x

main()

25

USS Return Values

25   5

ans   num

square()

1. Function **square()** is called
   a. Make copy of **x**'s value
   b. Pass copy of **5** to **square()**
   c. Execute **num * num**
      a. Save in variable **ans**
   d. Return calculated value
   e. Assign returned value to **y**

5

x

25

y

main()

USS Return Values

All materials copyright UMBC and Dr. Katherine Gibson unless otherwise noted   www.umbc.edu

`None` and Common Problems

www.umbc.edu

Every Function Returns *Something*

- All Python functions return a value
  - Even if they don't have a **return** statement

- Functions without an explicit **return** pass back a special object, called **None**
  - **None** is the <u>absence</u> of a value

Common Errors and Problems

- Writing a function that returns a value but…

- Forgetting to include the **return** statement

```
>>> def multiply(num1, num2):
...         print("doing", num1, "*", num2)
...         answer = num1 * num2
>>> product = multiply(3, 5)
doing 3 * 5
>>> print(product)
None
```

Variable assigned to the return value will be **None**

Common Errors and Problems

- Writing a function that returns a value but…

- Forgetting to assign that value to anything

```
>>> product = 0
>>> def multiply(num1, num2):
...      print("doing", num1, "*", num2)
...      return num1 * num2
>>> multiply(7, 8)
doing 7 * 8
>>> print(product)
0
```

The variable **product** was not updated; the code should have read
`product = multiply(7, 8)`

Common Errors and Problems

- If your value-returning functions produce strange messages, check to make sure you used the **`return`** correctly!

```
TypeError: unsupported operand type(s)
for *: 'NoneType' and 'int'
```

```
TypeError: 'NoneType' object is not
iterable
```

"Modifying" Parameters

www.umbc.edu

Bank Interest Example

- Suppose you are writing a program that manages bank accounts

- One function we would need to create is one to accumulate interest on the account

```
def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    balance = newBalance
```

 www.umbc.edu

Bank Interest Example

- We want to set the balance of the account to a new value that includes the interest amount

```python
def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    balance = newBalance

def main():
    amount = 1000
    rate = 0.05
    addInterest(amount, rate)
    print(amount)
main()
```

What is the output of this code?

1000

Is this what we expected?

What's Going On?

- It was intended that the 5% would be added to the amount, returning $1050

- Was $1000 the expected output?

- No – so what went wrong?

- This is a very common mistake to make!
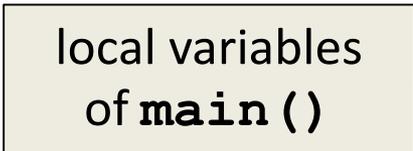 - Let's trace through the code and figure it out

 www.umbc.edu

Tracing the Bank Interest Code

- First, we create two variables that are local to **main()**

```python
def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    balance = newBalance

def main():
    amount = 1000
    rate = 0.05
    addInterest(amount, rate)
    print(amount)
main()
```

local variables of **main()**

 www.umbc.edu

Tracing the Bank Interest Code

- Second, we call **addInterest()** and pass the values of the local variables of **main()** as arguments

```python
def addInterest(bal
    newBalance = ba          e)
    balance = newBa

def main():
    amount = 1000
    rate = 0.05
    addInterest(amount, rate)
    print(amount)
main()
```

Call to
addInterest()

Passing the values stored in amount and rate, which are local variables
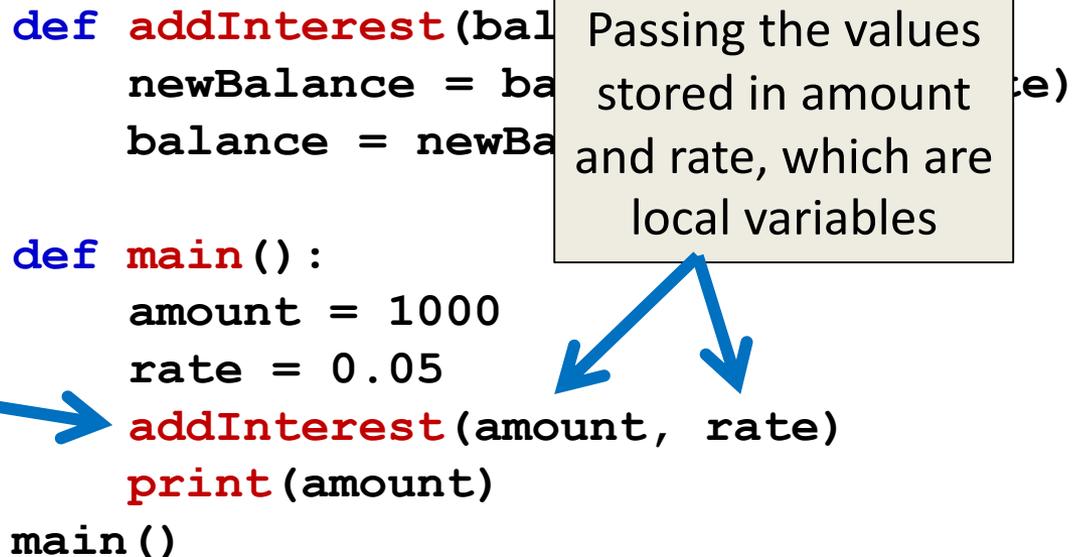
Tracing the Bank Interest Code

- Third, when control is passed to **addInterest()**, the formal parameters (balance and rate) are set to the value of the arguments (amount and rate)
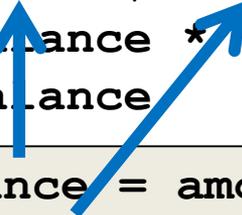
Control passes to
addInterest()

```
def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    balance = newBalance


def main():
    amount = 1
    rate = 0.05
    addInterest(amount, rate)
    print(amount)
main()
```

balance = amount = 1000
    rate = rate = 0.05

Tracing the Bank Interest Code

- Even though the parameter **rate** appears in both **main()** and **addInterest()**, they are two <u>separate</u> variables because of scope

Even though rate exists in both **main()** and **addInterest()**, they are in two separate scopes

```
def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    balance = newBalance

def main():
    amount = 1000
    rate = 0.05
    addInterest(amount, rate)
    print(amount)
main()
```

Scope

- In other words, the *formal parameters* of a function only receive the <u>values</u> of the *arguments*

- The function does <u>not</u> have access to the original variable in `main()`

New Bank Interest Code

```python
def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    return newBalance

def main():
    amount = 1000
    rate = 0.05
    amount = addInterest(amount, rate)
    print(amount)
main()
```

  www.umbc.edu

New Bank Interest Code

```python
def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    return newBalance

def main():
    amount = 1000
    rate = 0.05
    amount = addInterest(amount, rate)
    print(amount)
main()
```

These are the only parts we changed

 www.umbc.edu

Daily emacs Shortcut

- `M + %`
  - (Meta + Shift + 5)
  - Search and replace
    - Keeps correct case! (cat -> dog, Cat -> Dog, CAT -> DOG)

- First, type the thing to search for; hit Enter

- Second, type the thing replace it with; Enter
  - Hit "y" or "n" for each highlighted instance to indicate if you want to replace that one

          www.umbc.edu

Announcements

- HW 5 is out on Blackboard now
  - Due by Friday (Oct 13th) at 8:59:59 PM

- Midterm is <u>in class</u>, October 18th and 19th
  - Survey #1 will be released that week as well
  - Review packet out on the website now
  - You ***MUST*** use a pencil on the exam!
    - Exams taken in pen will not be graded

Image Sources

- Cardboard box:
  - https://pixabay.com/p-220256/
- Wooden ship (adapted from):
  - https://pixabay.com/p-307603/
- Coconut island (adapted from):
  - https://pixabay.com/p-1892861/
- Dollar sign:
  - https://pixabay.com/p-634901/

 www.umbc.edu