CMSC201
Computer Science I for Majors

Lecture 11 – Functions (cont)

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

Last Class We Covered

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

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

Any Questions from Last Time?

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

Today's Objectives

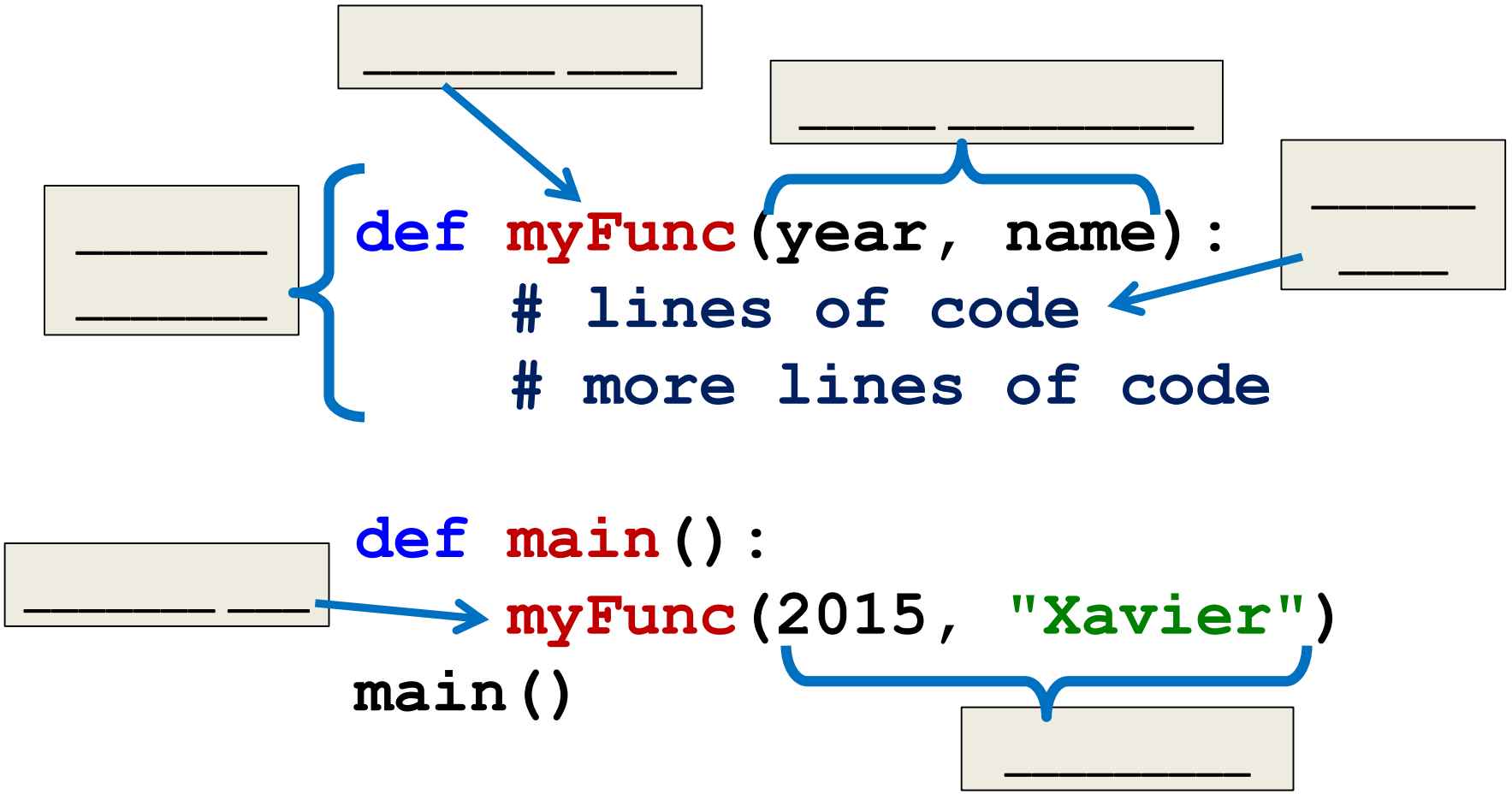
- To introduce value-returning functions
  - Common problems
  - Solutions to common problems
- To better grasp how values in the scope of a function actually work
- To practice function calls

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

Review: Parts of a Function

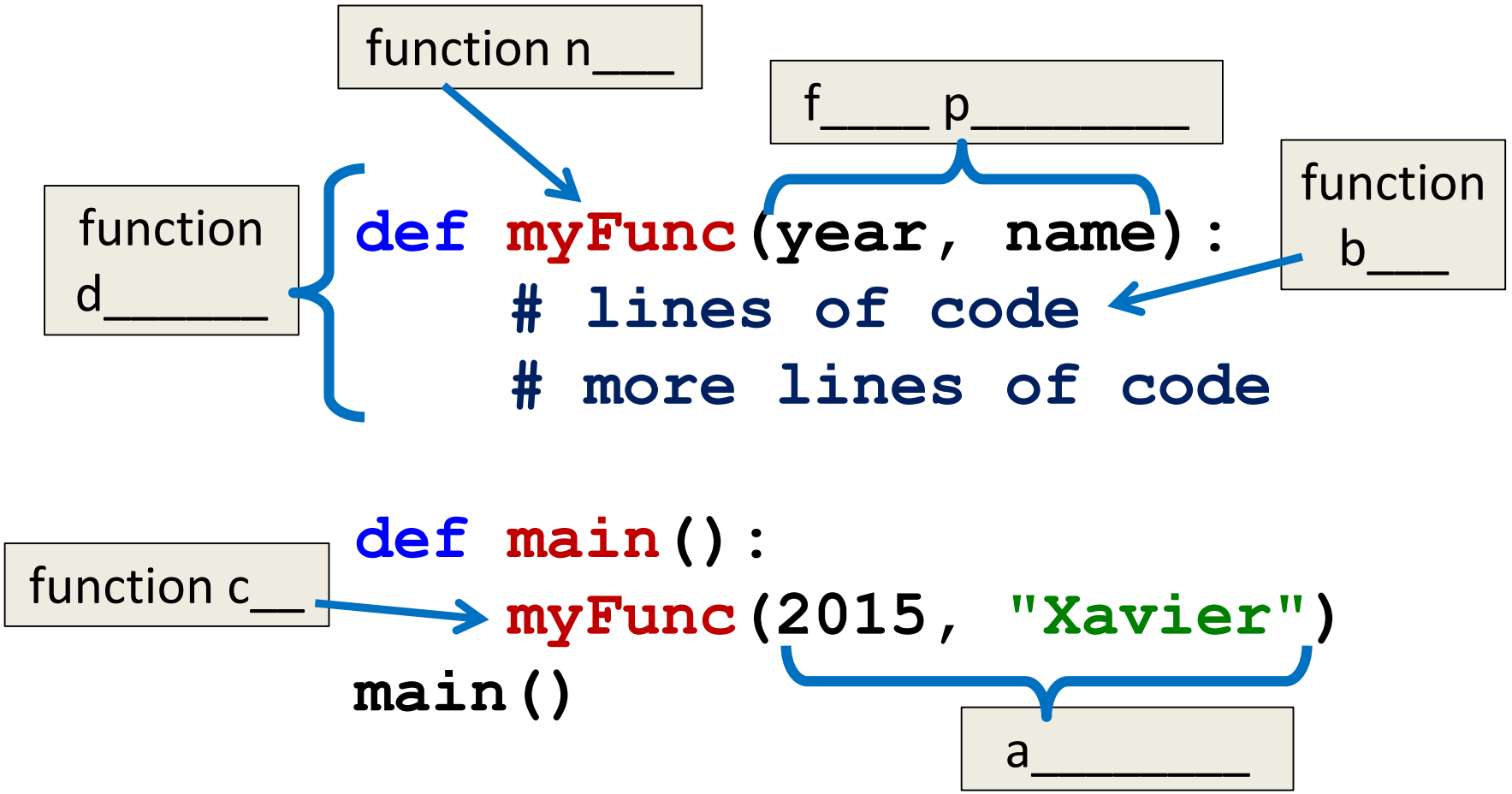
All materials copyright UMBC and Dr. Katherine Gibson unless otherwise noted

Function Vocabulary



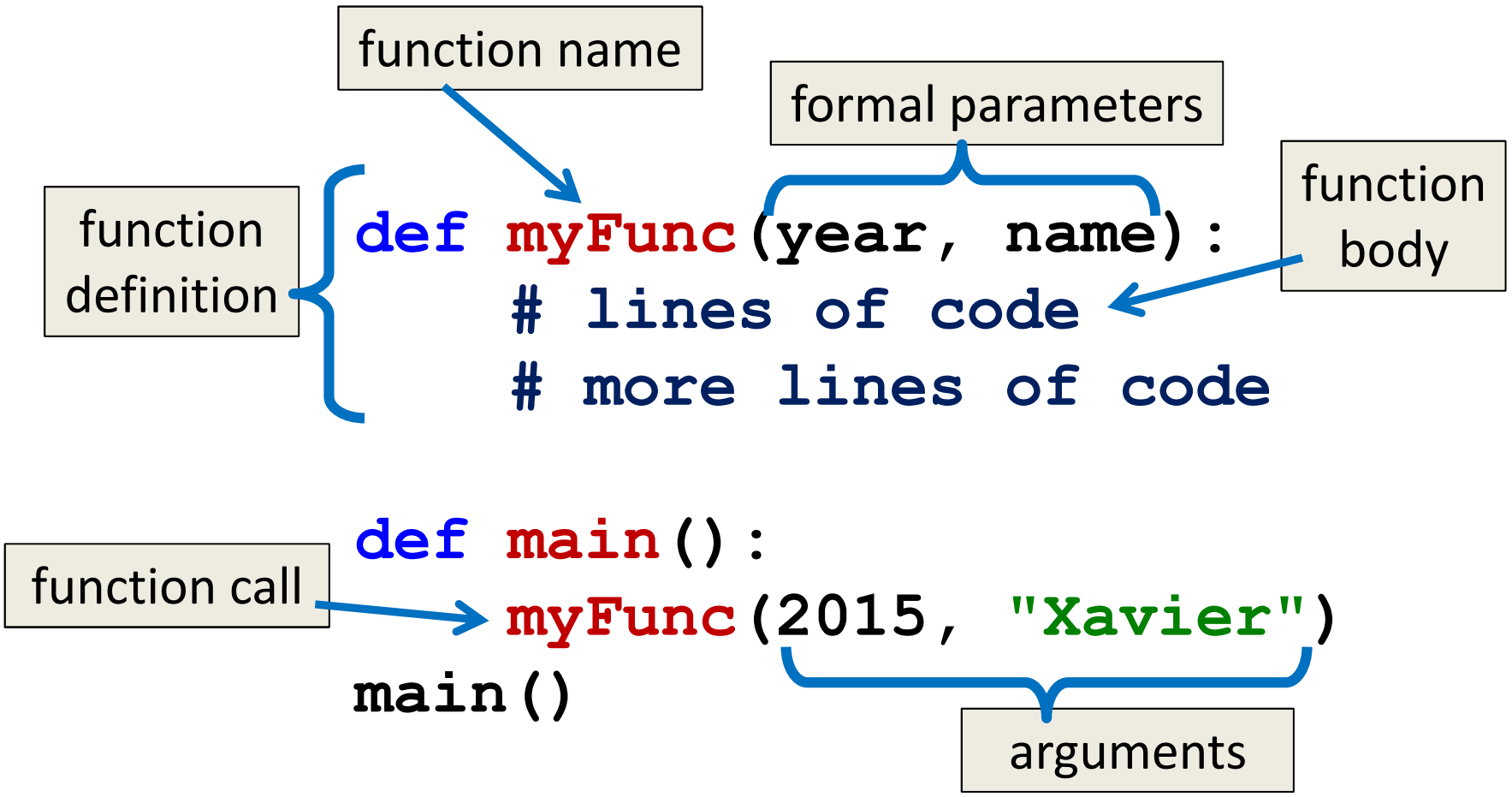
All materials copyright UMBC and Dr. Katherine Gibson unless otherwise noted

Function Vocabulary



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

Function Vocabulary



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

File Layout and Constants

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

Layout of a Python File

header comment

constants

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

main() definition

call to **main()**

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

MIN_GRADE = 0
MAX_GRADE = 100

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

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

main()
```


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

Note On Global Constants

- ***Globals*** are variables declared outside of any function (including **`main()`**)
- Accessible globally in your program
  - To all functions and code

- Your programs may not have global variables
- Your programs may use global **constants**
  - In fact, constants should be global

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

Return Statements

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

Giving Information to a Function

- Passing parameters provides a mechanism for initializing the variables in a function
- Parameters act as ***inputs*** to a function
- We can call a function many times and get different results by changing its parameters

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

Getting Information from a Function

- We've already seen numerous examples of functions that return values

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

- For example, **`len()`**
  - Takes in any list or string as its parameter
  - Counts the number of elements (or characters)
  - And returns an integer value

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

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
```


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

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 evaluated, then sent back to the caller as a ***return value***

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

Code Trace: Return from `square()`

Let's follow the flow of the code

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

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

Step 1: Call `main()`

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

Code Trace: Return from `square()`

Let's follow the flow of the code

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

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

Step 1: Call **`main()`**

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

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

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()
```

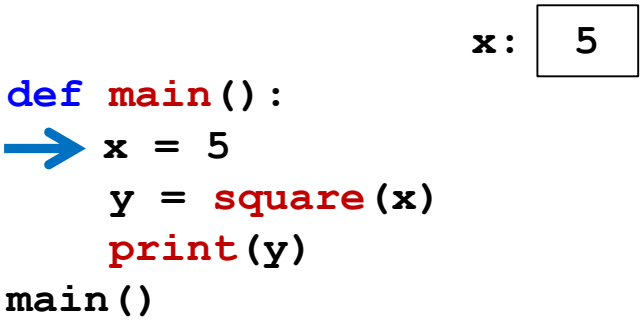
```
def square(num):
    ans = num * num
    return ans
```

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

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

Code Trace: Return from `square()`

Let's follow the flow of the code



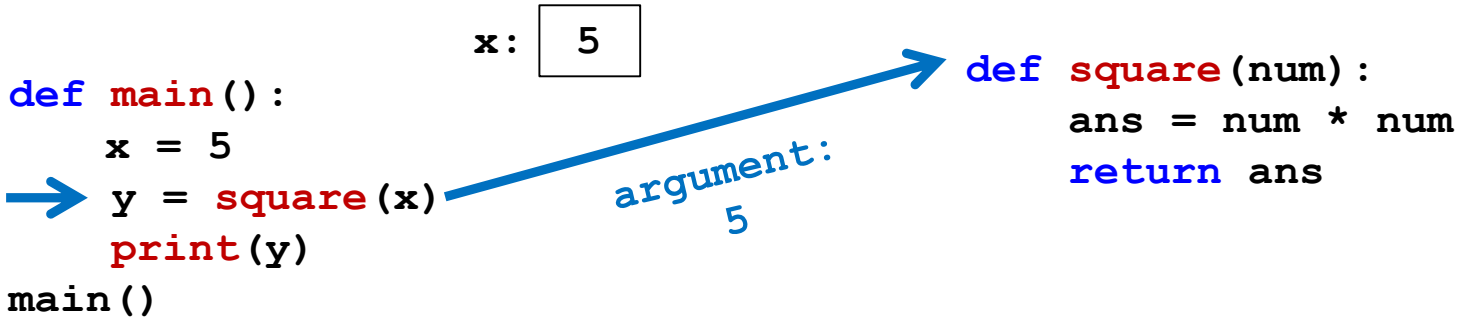

```
def square(num):
    ans = num * num
    return ans
```

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

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

Code Trace: Return from `square()`

Let's follow the flow of the code



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`**

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

Code Trace: Return from `square()`

Let's follow the flow of the code

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

x: 5

```
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`**

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

Code Trace: Return from `square()`

Let's follow the flow of the code

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

x: 5

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

num: 5

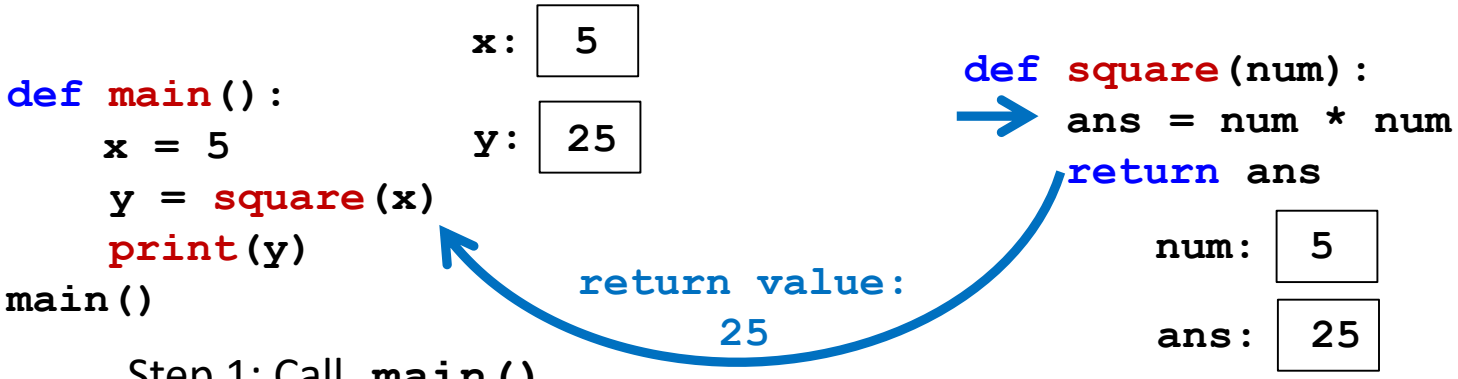
ans: 25

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`

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

Code Trace: Return from `square()`

Let's follow the flow of the code



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

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

Code Trace: Return from `square()`

Let's follow the flow of the code

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

x: 5

y: 25

```
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`

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

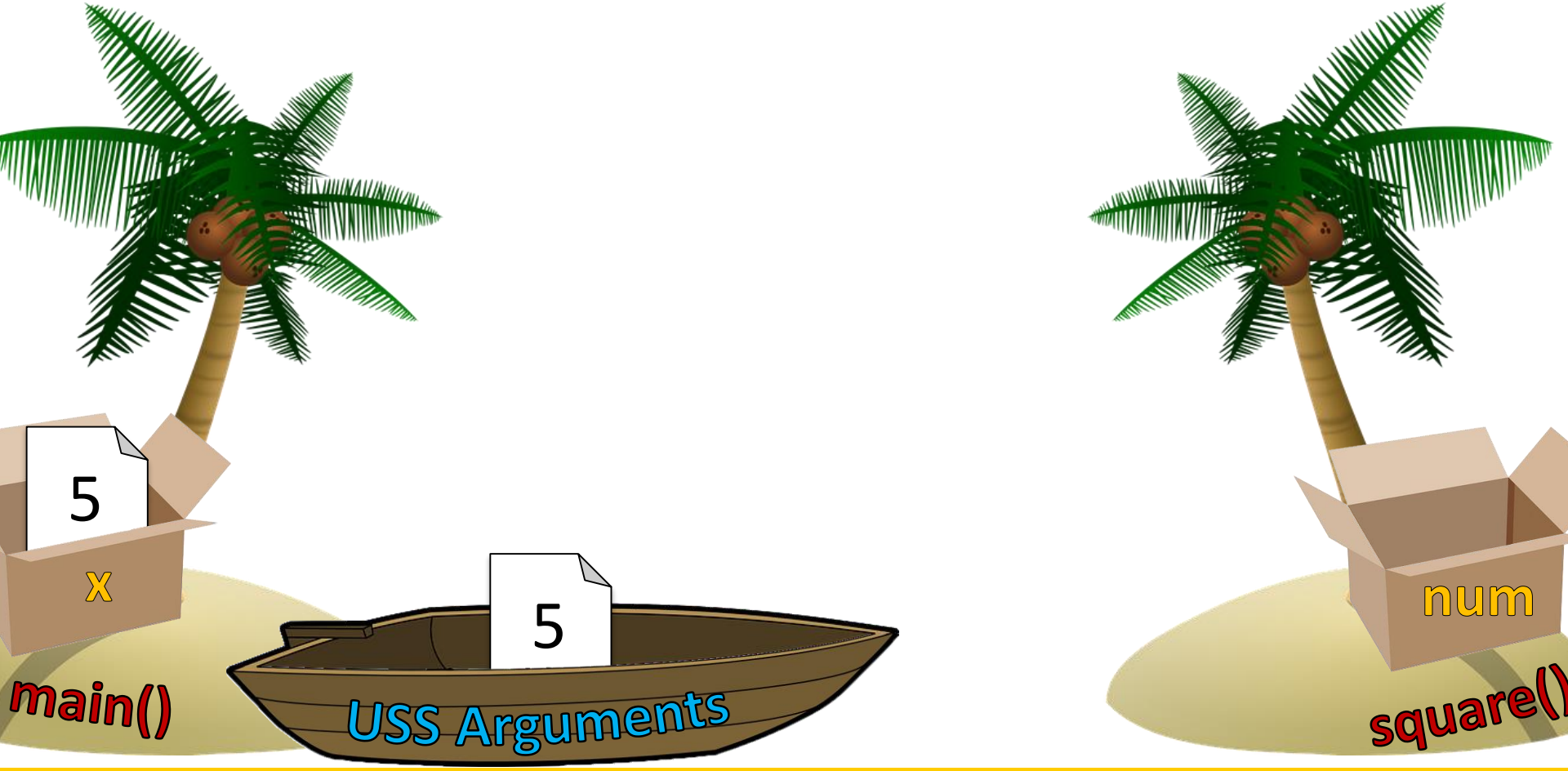
Island Example

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

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

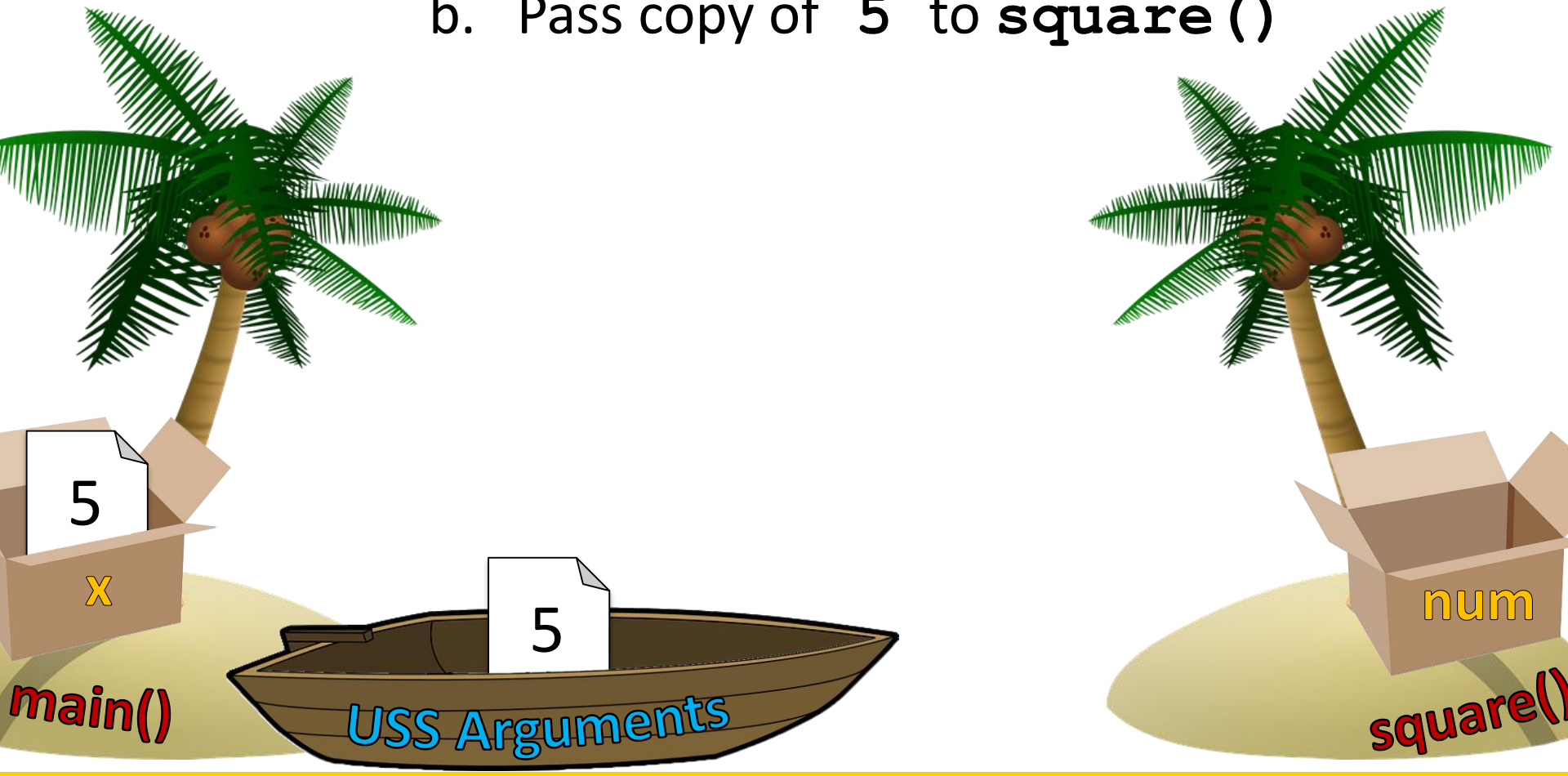
1. Function `square()` is called

a. Make copy of `x`'s value



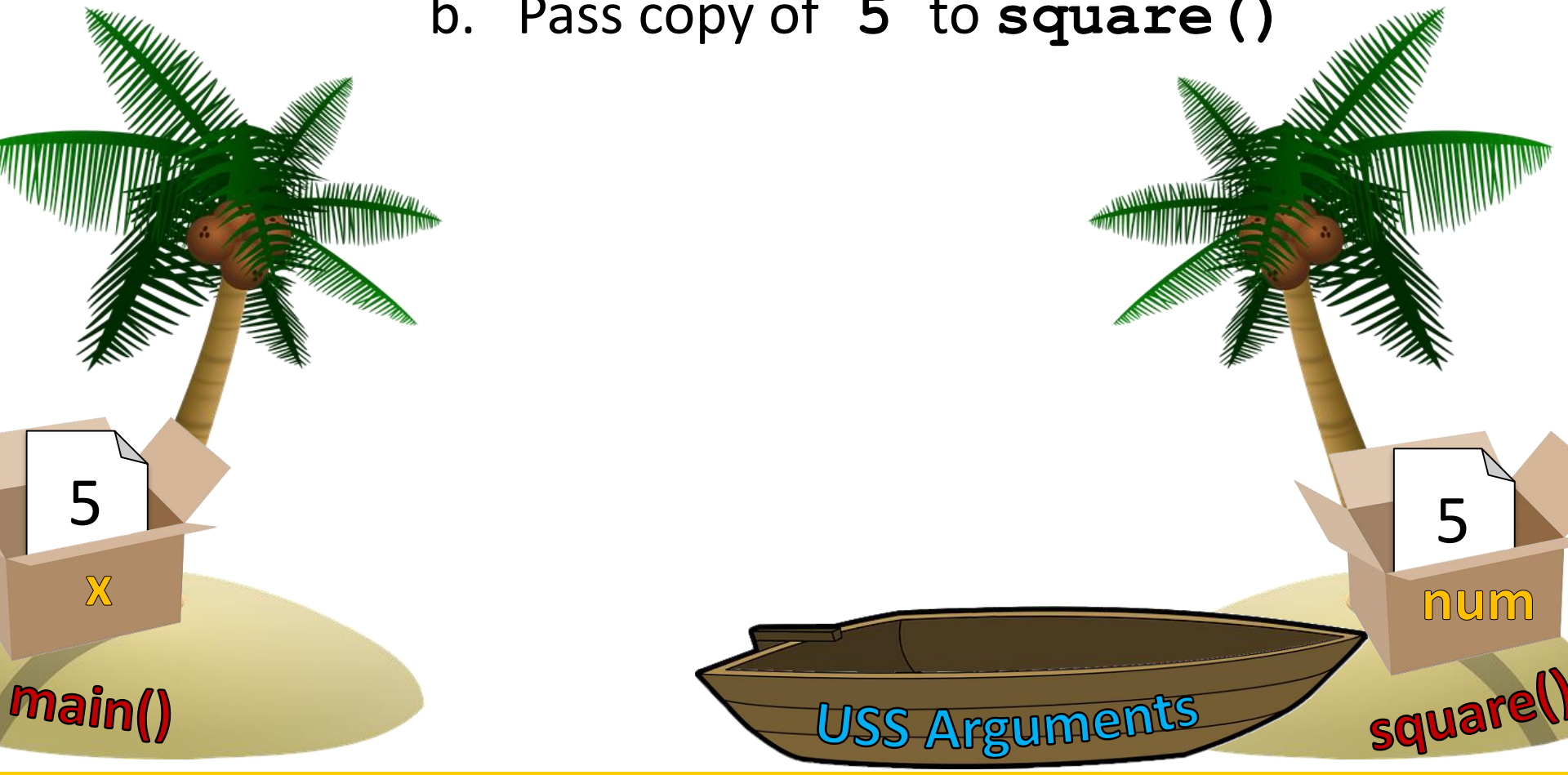
All materials copyright UMBC and Dr. Katherine Gibson unless otherwise noted

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



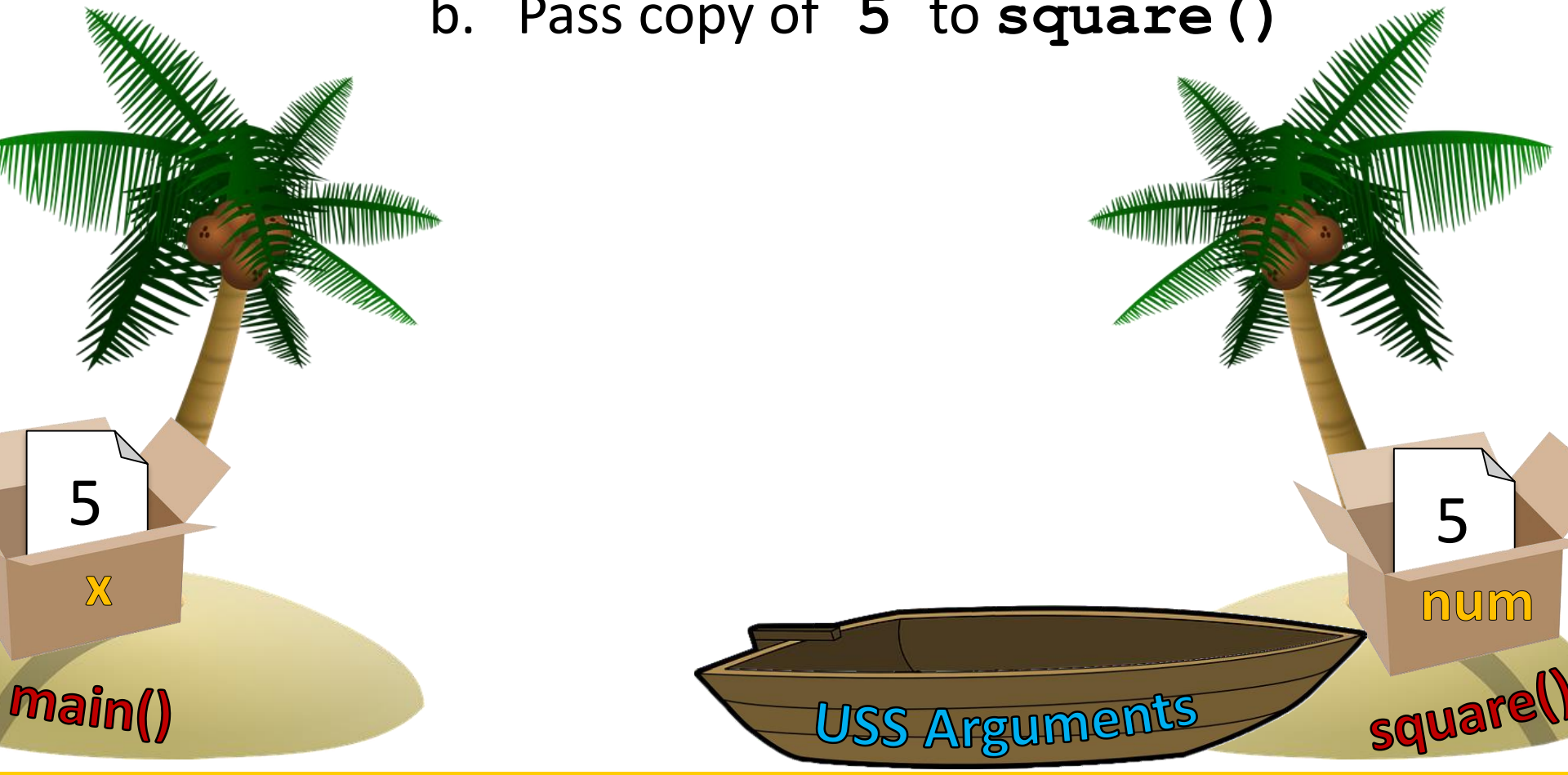
All materials copyright UMBC and Dr. Katherine Gibson unless otherwise noted

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



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()`**

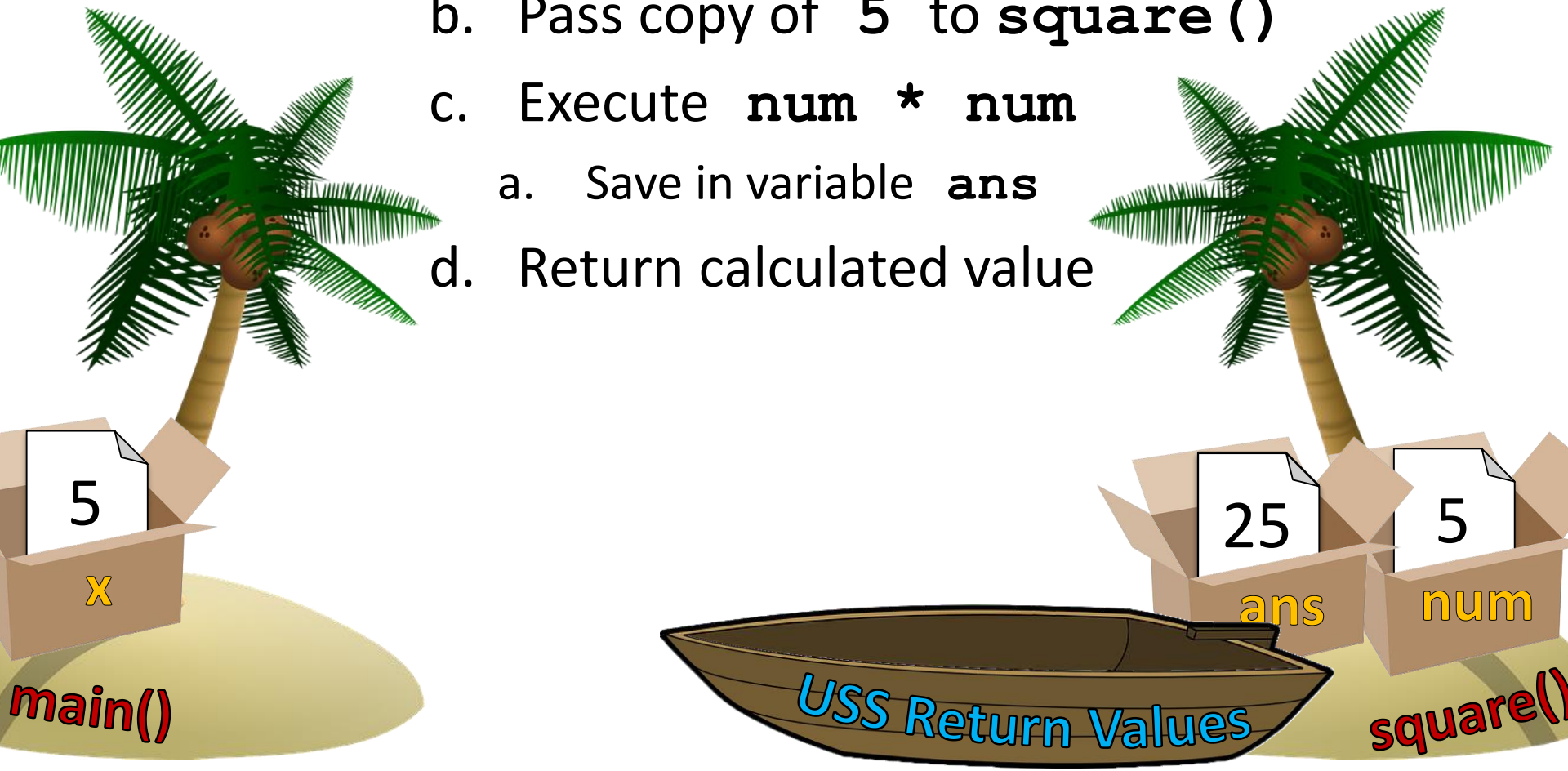


All materials copyright UMBC and Dr. Katherine Gibson unless otherwise noted

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

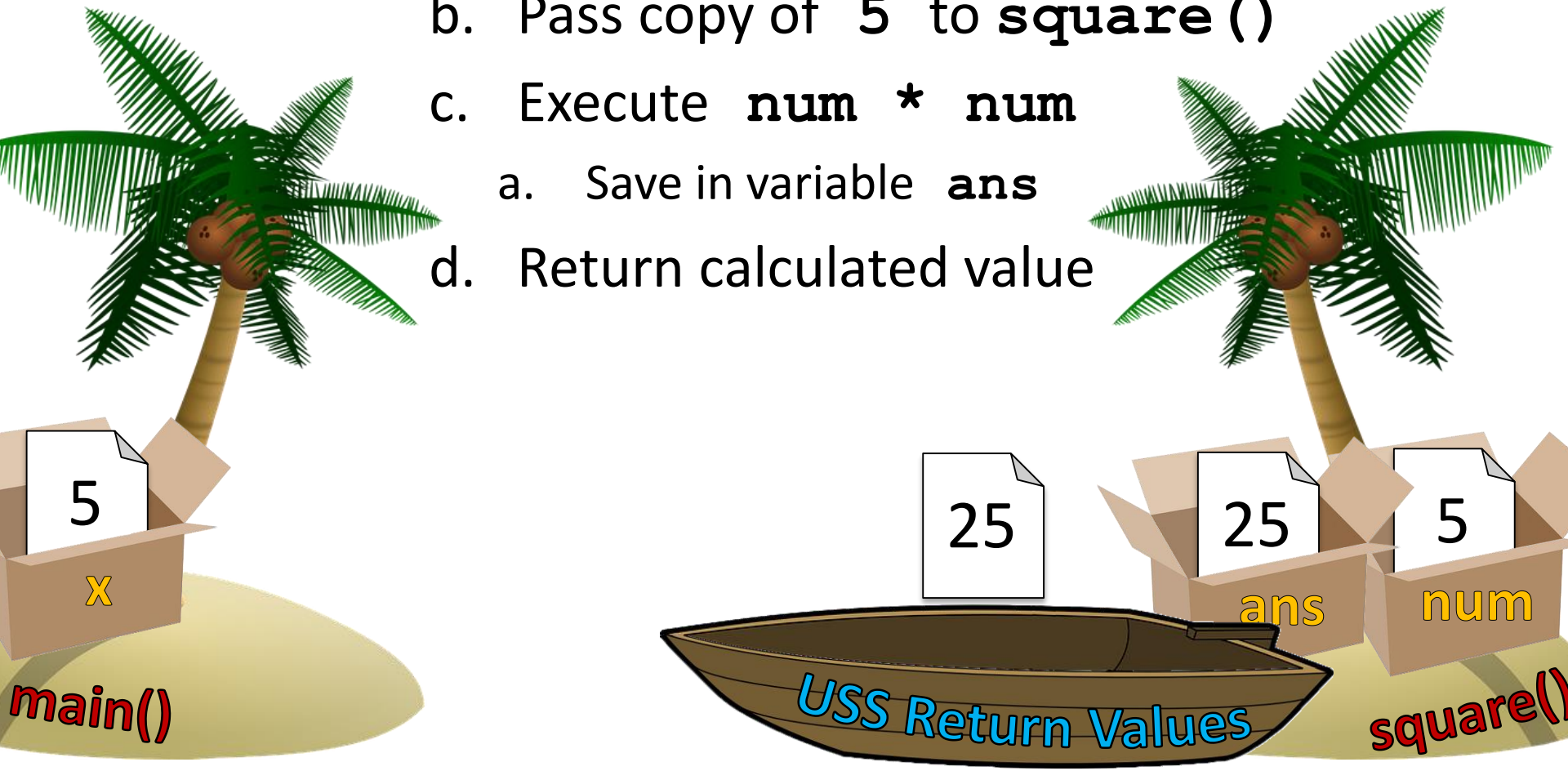
All materials copyright UMBC and Dr. Katherine Gibson unless otherwise noted

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



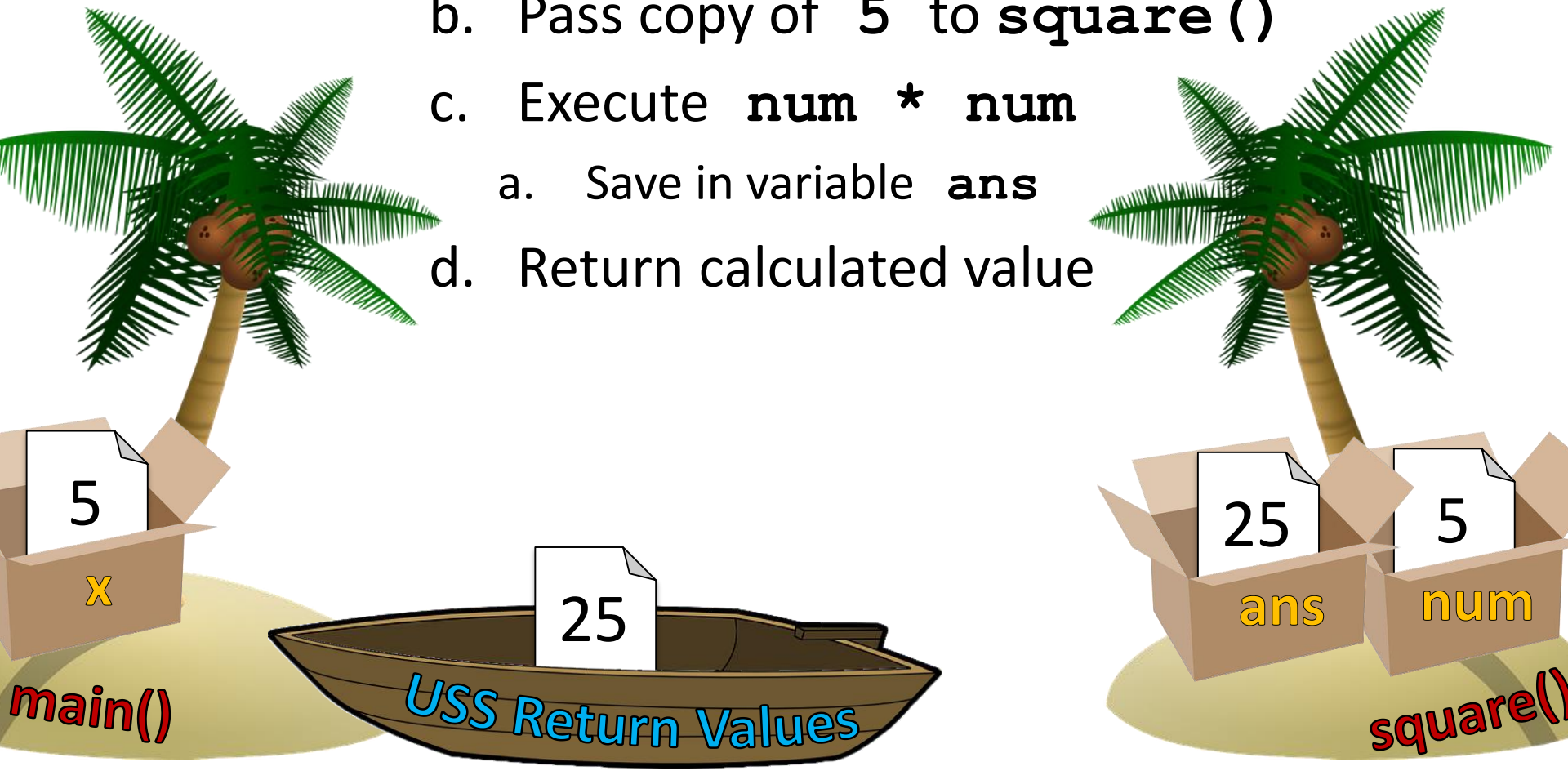
All materials copyright UMBC and Dr. Katherine Gibson unless otherwise noted

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



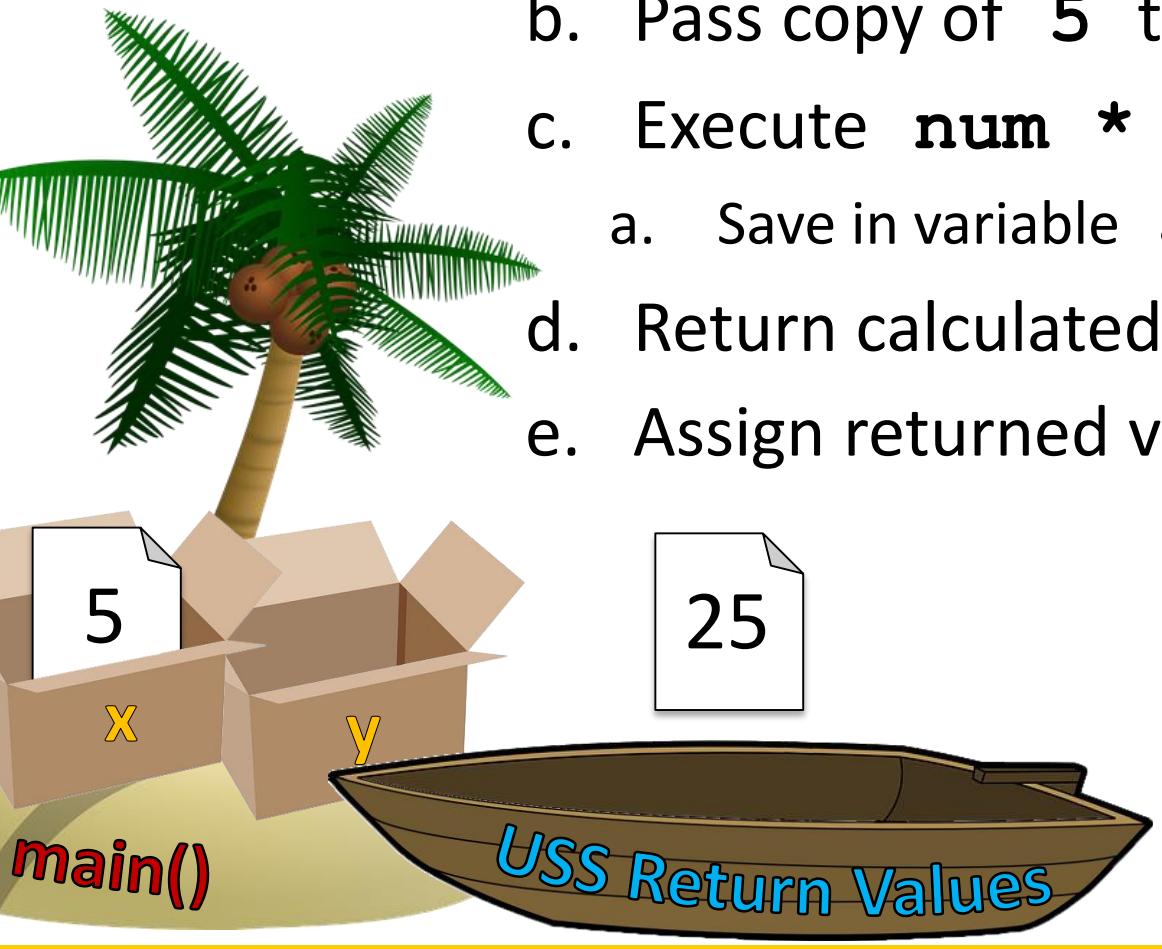
All materials copyright UMBC and Dr. Katherine Gibson unless otherwise noted

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



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()`**
   c. Execute **`num * num`**
      a. Save in variable **`ans`**
   d. Return calculated value
   e. Assign returned value to **`y`**



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

`None` and Common Problems

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

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 absence of a value

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

Example (That We'll Break Soon)

- Here is a simple toy example:

```
def multiply(num1, num2):
    print("doing", num1, "*", num2)
    answer = num1 * num2
    return answer
```

What is the output of this code?

- Assume that this code is in **`main()`**:

```
product = multiply(6, 3)
print("result is", product)
```

```
doing 6 * 3
result is 18
```


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

Problem #1

- Forgetting to write a **`return`** statement

```
def multiply(num1, num2):
    print("doing", num1, "*", num2)
    answer = num1 * num2

product = multiply(3, 5)
print("result is", product)
```

- What is the code's output now?

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

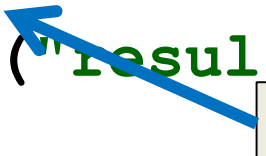
Problem #1

- Forgetting to write a **`return`** statement

```
def multiply(num1, num2):
    print("doing", num1, "*", num2)
    answer = num1 * num2

product = multiply(3, 5)
print("result is", product)
```

```
doing 3 * 5
result is None
```



Variable given the return value has a value of **`None`**

- What is the co

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

Problem #2

- Forgetting to assign the returned value

```
def multiply(num1, num2):
    print("doing", num1, "*", num2)
    return num1 * num2

multiply(7, 8)
print("result is", product)
```

- What is the code's output now?

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

Problem #2

- Forgetting to assign the returned value

```
def multiply(num1, num2):
    print("doing", num1, "*", num2)
    return num1 * num2


multiply(7, 8)
print("result is", product)
```

```
doing 7 * 8
[syntax error]
```

- What is the code's output r

Should have assigned **product** to the return value of **multiply**

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

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
```


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

“Modifying” Parameters

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

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
```


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

Bank Interest Example

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

```
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 wanted to happen?

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

What's Going On?

- It was intended that the 5% would be added to the amount, returning $1050
- Was $1000 the desired output?

- No – so what went wrong?
- This is a very common mistake to make!
  - Let's trace through the code and figure it out

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

Tracing the Bank Interest Code

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

local variables of **`main()`**

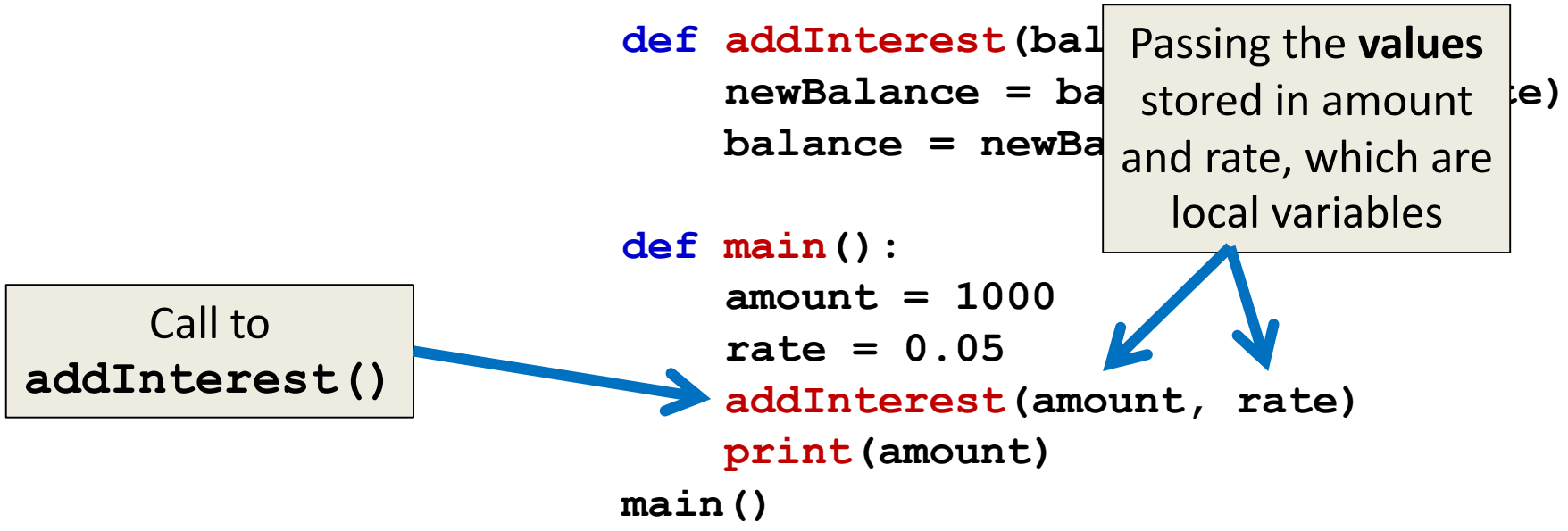
```
def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    balance = newBalance

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


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

Tracing the Bank Interest Code

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



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

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()
```

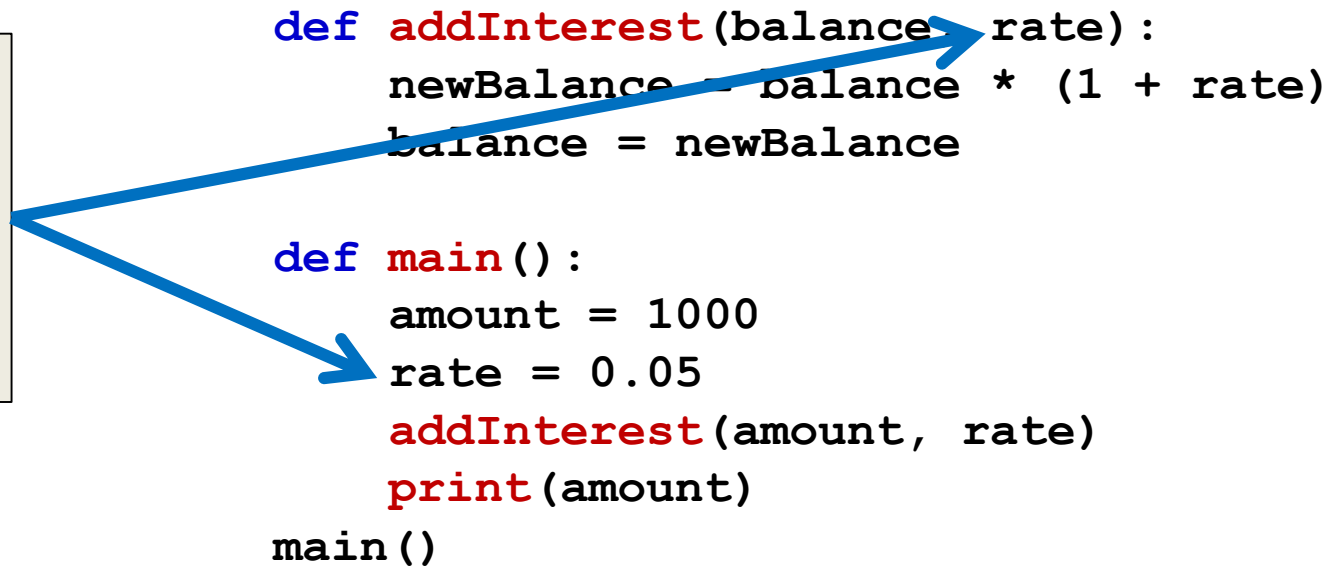
balance = 1000
rate = 0.05

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

Tracing the Bank Interest Code

- Even though the parameter **`rate`** appears in both **`main()`** and **`addInterest()`**, they are two separate 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()
```


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

Scope

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

- The function does not have access to the original variable in **`main()`**

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

New Bank Interest Code

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

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


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

New Bank Interest Code

```
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

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

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

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

Announcements

- HW 5 is out on Blackboard now
  - Due by Friday (March 15th) at 8:59:59 PM
- We'll hand back the midterms next class
  - Survey #1 is due Tuesday at midnight
  - During class:
    - Cover correct answers to exam
    - Discuss ways to improve on future exams
    - Extra credit opportunity

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

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/

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