

Lecture 14:

FUNCTIONS

CSC111: Introduction to CS through Programming

R. Jordan Crouser

Assistant Professor of Computer Science

Smith College

Announcements

Fall Tech Trek in Boston

Friday, October 12, 8:00am-6:00pm

Now open to all class years.

Space is limited. **Sign up in Handshake by Tuesday 10/9 at 5:00pm**

The day is an opportunity for students to explore careers in technology and meet with professionals at their workplace. Hear from the representatives about their career paths and advice on getting started in the industry.

Bus transportation to and from the program will be provided. Lunch will be served.



HubSpot - A developer and marketer of software products for inbound marketing and sales



Audible - A seller and producer of spoken audio entertainment, information, and educational programming on the Internet.



DraftKings - A Boston daily fantasy sports contest provider. The company allows users to enter daily and weekly fantasy sports-related contests and win money based on individual player and team.

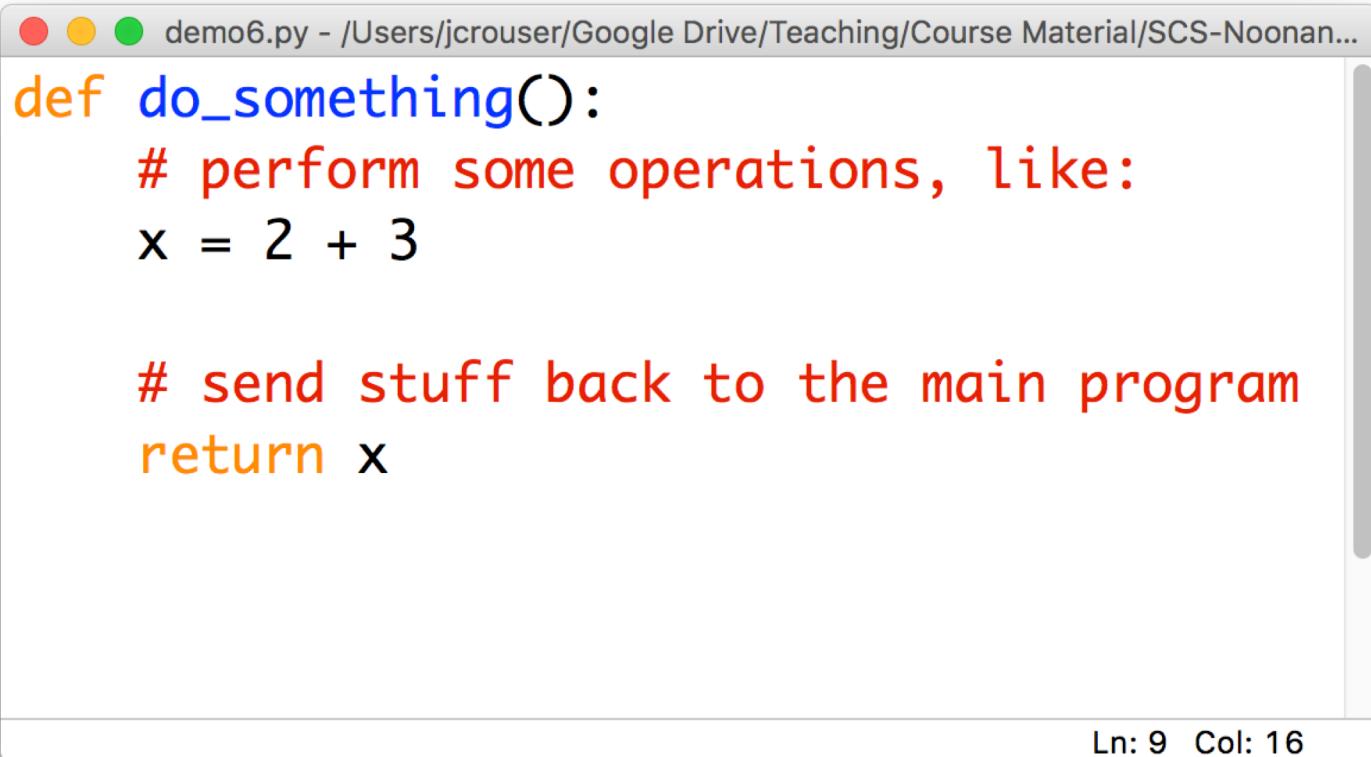
Outline

- ✓ Monday: FALL BREAK
- Wednesday: Functions
 - basic components
 - definition vs. call
 - an analogy
 - parameters
 - returning values
- LAB: MadLibs
- Friday: Catching Exceptions

Functions

- **Recall:** a **function** is a procedure / routine that takes in some input and does something with it (just like in math)
- We've seen lots of built-in functions:
 - `print(...)`
 - `input(...)`
 - `eval(...)`
 - `round(...)`
- Perhaps unsurprisingly, Python lets us write custom functions as well (like yesterday's `main()` function)

Basic components of a function



The image shows a screenshot of a code editor window titled "demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...". The code editor displays the following Python code:

```
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x
```

The code consists of a single function definition. It starts with the keyword `def`, followed by the function name `do_something`. Inside the function, there is a comment `# perform some operations, like:` and a calculation `x = 2 + 3`. Below that, another comment `# send stuff back to the main program` is present, followed by the `return` statement with the variable `x`. In the bottom right corner of the code editor window, there is a status bar displaying "Ln: 9 Col: 16".

Basic components of a function

a name



```
demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x

Ln: 9 Col: 16
```

Convention: use _underscores_ or **camelCase**

Basic components of a function

which is defined
using the **def** keyword



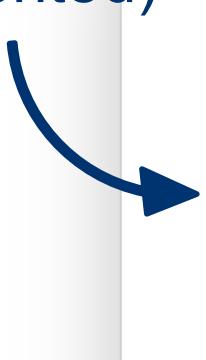
```
demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x

Ln: 9 Col: 16
```

Basic components of a function

a **body**
(indented)



```
demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x

Ln: 9 Col: 16
```

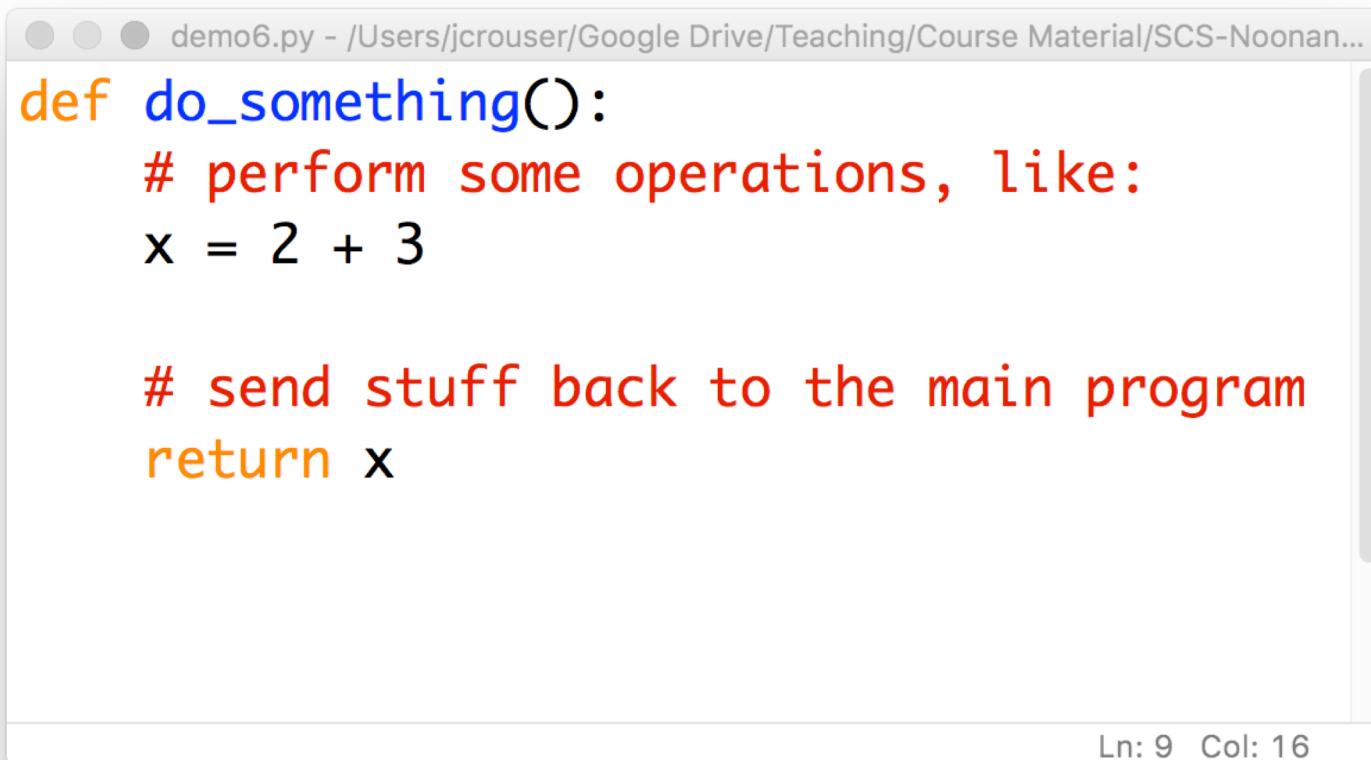
Basic components of a function

```
demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x
    a return (optional)

Ln: 9 Col: 16
```

A “function definition”



The image shows a screenshot of a code editor window titled "demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...". The code editor displays the following Python function definition:

```
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x
```

The code is color-coded: "def" and "do_something" are in orange, "# perform some operations, like:" and "# send stuff back to the main program" are in red, and "x" and "2 + 3" are in black. A blue bracket on the left side of the slide indicates the scope of the function definition. In the bottom right corner of the code editor window, it says "Ln: 9 Col: 16".

Discussion

What happens if we **run** this program?

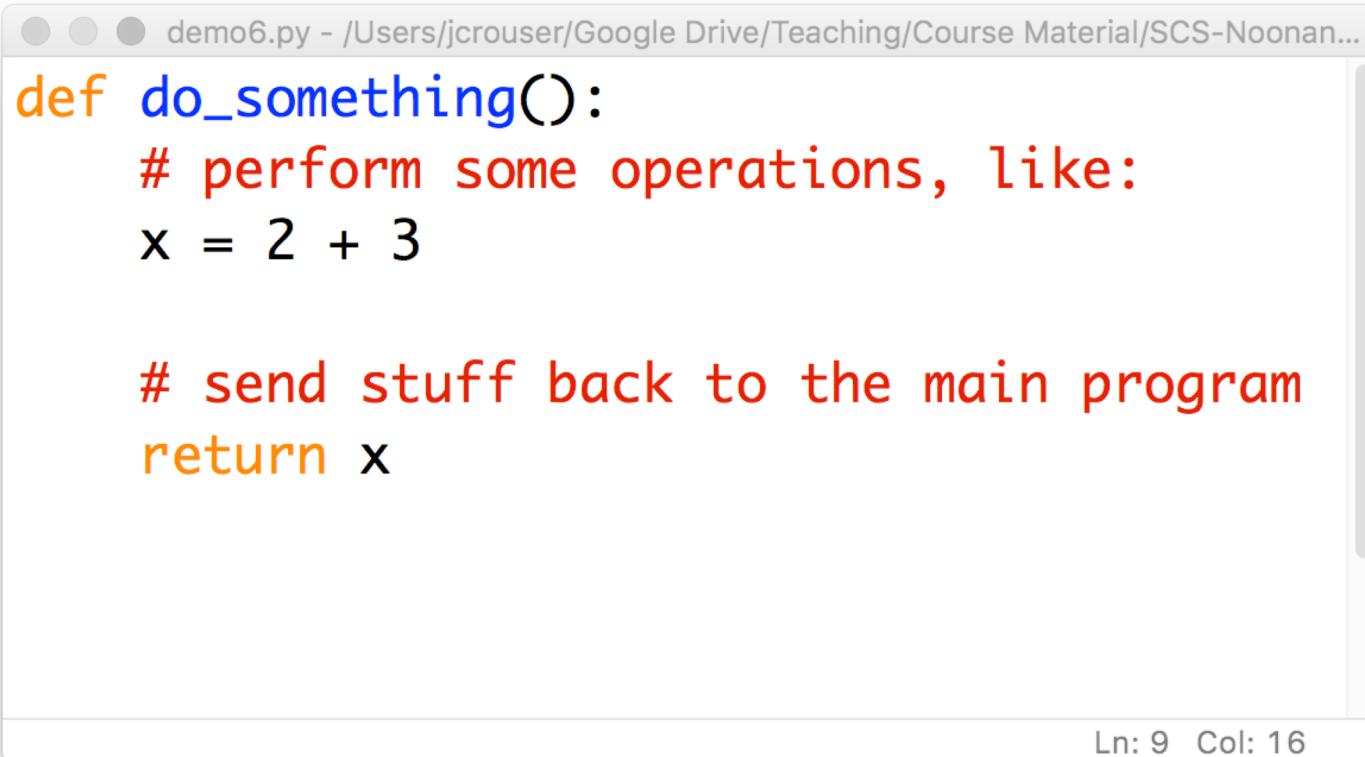
```
demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x
```

Col: 16



A “function definition” is a **description**



The image shows a screenshot of a code editor window titled "demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...". The code editor displays the following Python code:

```
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x
```

In the bottom right corner of the code editor, there is a status bar with the text "Ln: 9 Col: 16".

(but not a **directive**)

Function calls: “hey, Python! do this”



```
demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x

y = do_something()
```

A blue arrow points from the text "a function call" to the opening parenthesis of the function call "do_something()".

a function **call**

Ln: 9 Col: 16

Function calls: “hey, Python! do this”

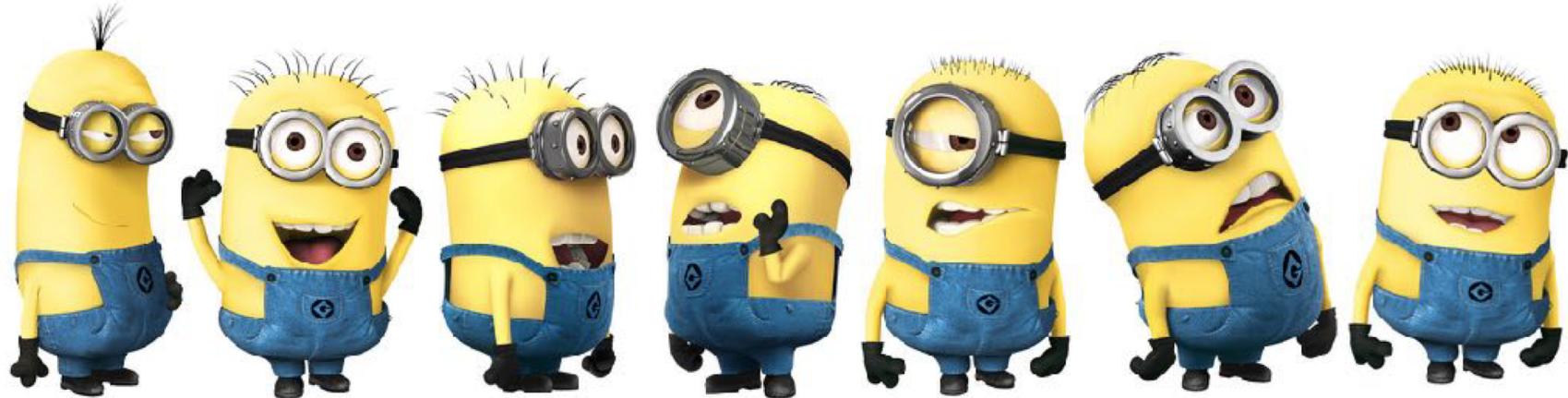


```
demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x 5

y =
Ln: 9 Col: 16
```

An analogy



functions are your **MINIONS**

An analogy



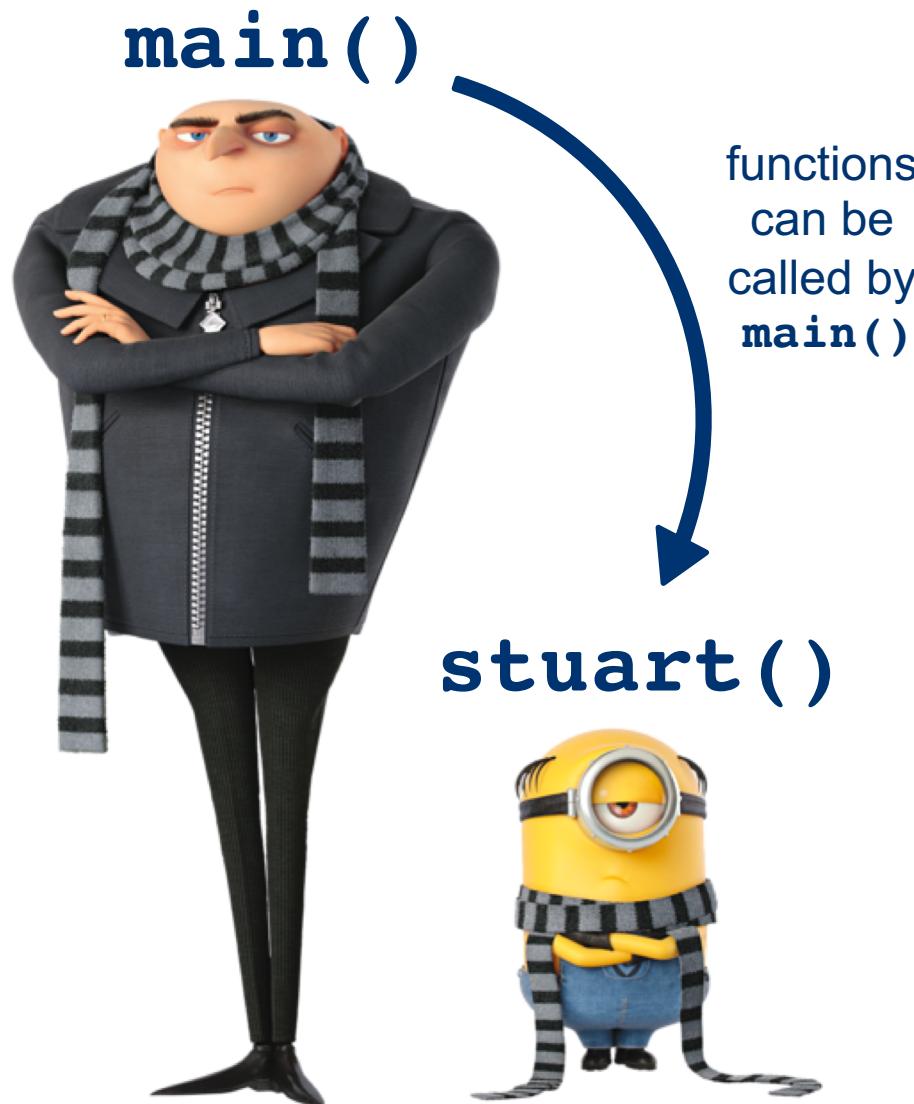
functions have NAMES

An analogy



they only work when you **CALL** them

An analogy



An analogy

`main()`



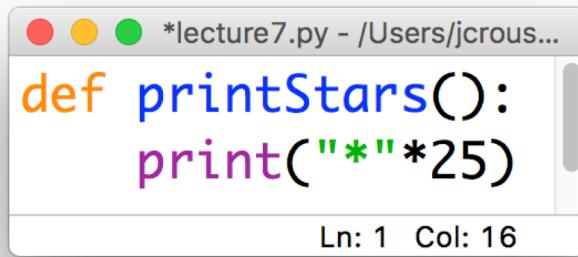
functions
can also be
called by
one another

`stuart()` `jerry()`



Two kinds of functions

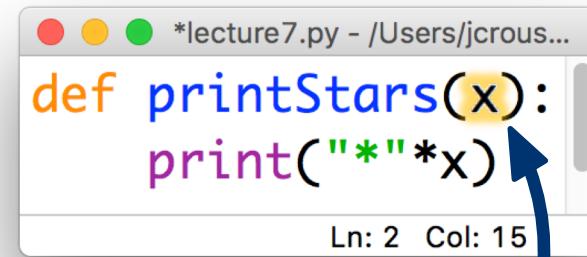
Some functions always
do the same thing



```
*lecture7.py - /Users/jcrou...  
def printStars():  
    print("*"*25)  
Ln: 1 Col: 16
```

printStars()
printStars()
printStars()

Others adjust their behavior
based on **what we give them**



```
*lecture7.py - /Users/jcrou...  
def printStars(x):  
    print("*"*x)  
Ln: 2 Col: 15
```

“parameter”

printStars(5)
printStars(32)
printStars(1527)

15-minute exercise: Happy Birthday

- Write a function called `happyBirthday(name)` that takes in a string `name` and prints out the lyrics to the song "Happy Birthday" with the name inserted:

Happy birthday to you!
Happy birthday to you!
Happy birthday, dear NAME
Happy birthday to you!

- Call this function from inside the `main()` function, and use `input(...)` to get the value of `name` from the user

Parameters

- Functions can be defined to take in **multiple** parameters:

```
lecture7.py - /Users/jcrouser/Google Drive/Teaching/Course...
def emphasize(word, char):
    print(char.join(list(word)))

emphasize("Tuesday", "-")
```

Ln: 1 Col: 24

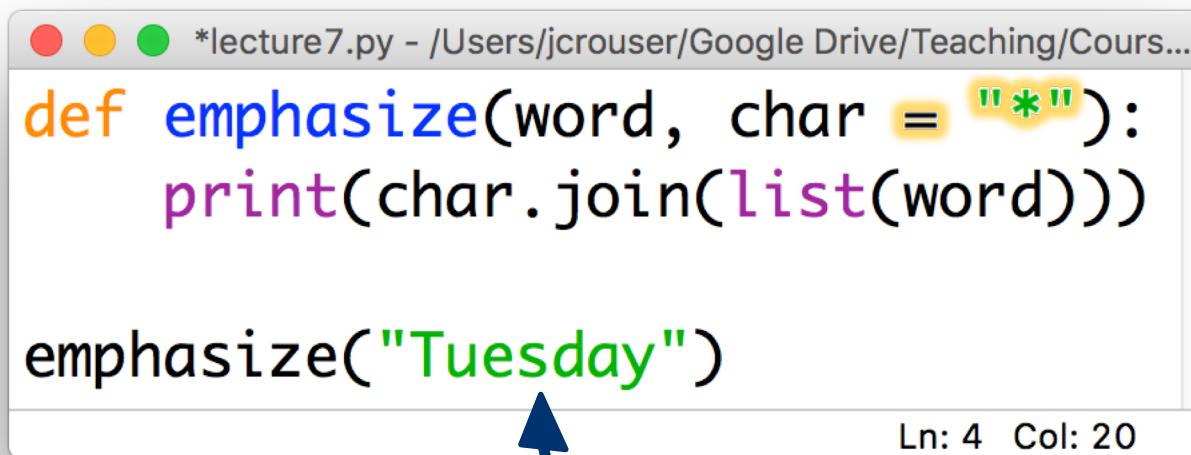
char = "-"
word = "Tuesday"

- Result:

T-u-e-s-d-a-y

Default parameters

- We can include a “default” value for some (or all) of them:



A screenshot of a Python code editor window titled "*lecture7.py - /Users/jcrouser/Google Drive/Teaching/Cours...". The code defines a function `emphasize` that takes two parameters: `word` and `char`. The `char` parameter has a default value of `"*"`. Below the function definition, there is a call to `emphasize("Tuesday")`.

```
def emphasize(word, char = "*"):
    print(char.join(list(word)))

emphasize("Tuesday")
```

only one parameter

- Result:

T*u*e*s*d*a*y

Returning values

- We may want to **return** the results rather than print them:

```
*lecture7.py - /Users/jcrouser/Google Drive/Teaching/Cours...
def emphasize(word, char = "*"):
    return char.join(list(word))

boom = emphasize("Tuesday")
Ln: 4 Col: 7
```

the results of the **return** in
emphasize() are stored in **boom**

Advanced: chaining functions

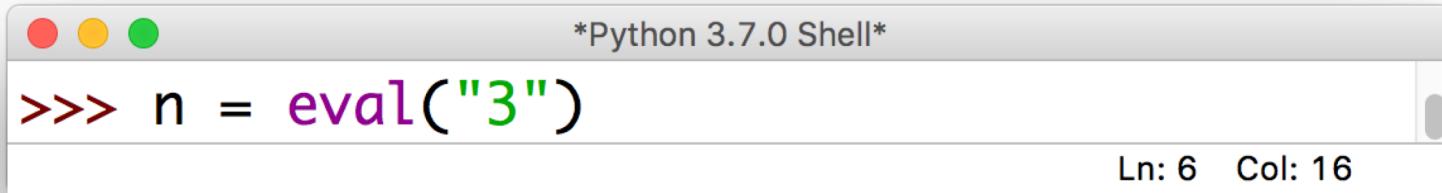
- Return values allow us to call functions **inside** other function calls:



Python 3.7.0 Shell

```
>>> n = eval(input("Enter an integer: "))
```

Ln: 6 Col: 41



Python 3.7.0 Shell

```
>>> n = eval("3")
```

Ln: 6 Col: 16

Recap: functions

- If you have to do something **multiple times**, then you probably want a function: this helps to “modularize” code (i.e. organize it for easy reuse)
- **Define** once, **call** as many times as necessary
- Naming convention: use **camelCase**
- **Important:** one function = one task



(mini) Assignment 5: encrypt/decrypt

For this assignment, you'll write 3 functions:

- **encrypt(...)** will take in a string containing a message, and encrypt it using a variation of the PIG LATIN cipher*
- **decrypt(...)** will undo the effects of **encrypt(...)**, enabling us to recover the original message
- **main(...)** will ask the user to enter a message, then call **encrypt(...)** and print the results, and finally call **decrypt(...)** on the encrypted message and print the decrypted original message

* for specific details, see assignment on Moodle

Outline

- ✓ Monday: FALL BREAK
- ✓ Wednesday: Functions
 - ✓ basic components
 - ✓ definition vs. call
 - ✓ an analogy
 - ✓ parameters
 - ✓ returning values
- LAB: MadLibs
- Friday: Catching Exceptions



It was a(n) _____ October day. I awoke to the

adjective

smell of _____ roasting in the _____.

type of food

room in house

I _____ down the stairs and much to my

verb (past tense, -ed)

_____, I ran directly into a(n) _____

emotion

adjective

_____. I found this very strange, because...

noun