Intro to Coding with Python– Functions

Dr. Ab Mosca (they/them)

Slides based off slides courtesy of Jordan Crouser (https://jcrouser.github.io/)

Plan for Today

- Functions
  - basic components
  - definition vs. call
  - an analogy
  - parameters
  - returning values

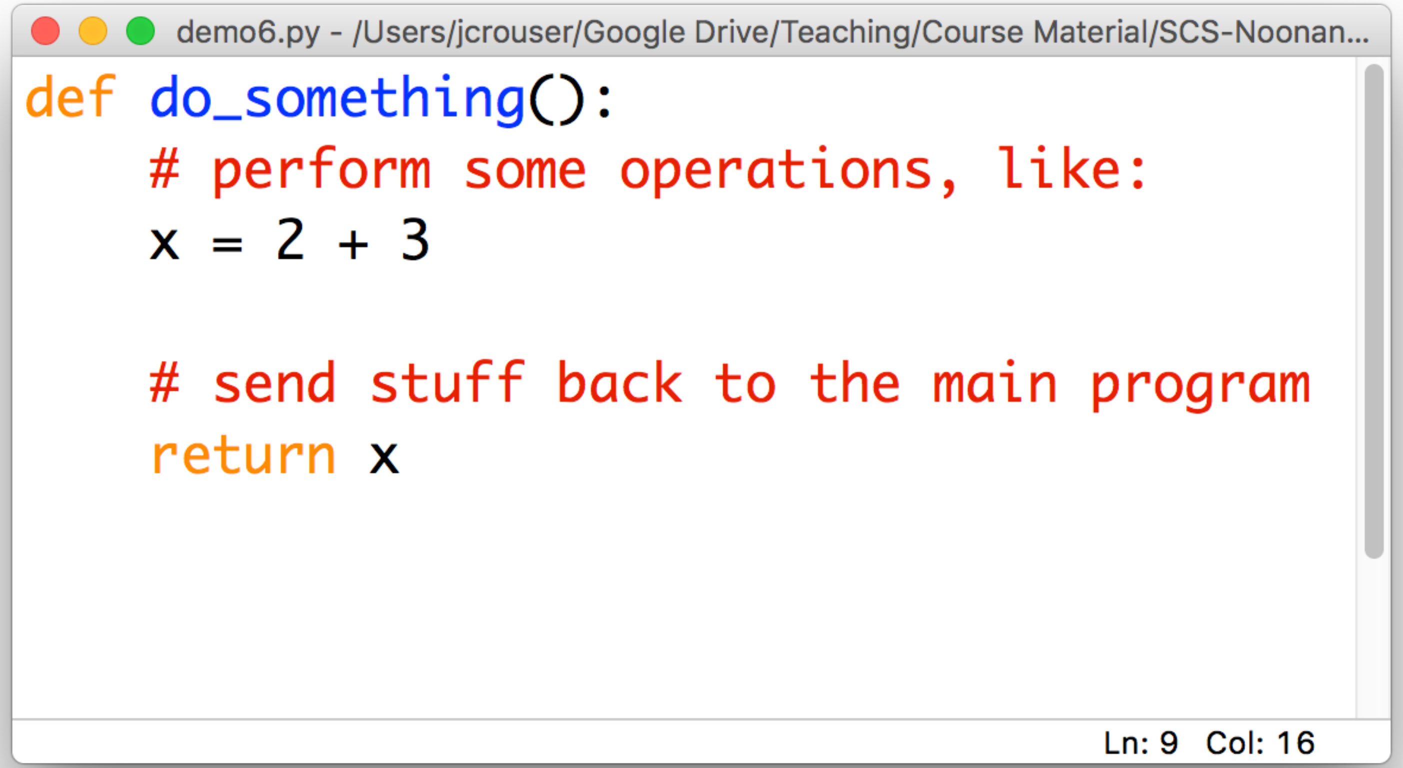
Functions

- **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

Basic components of a function

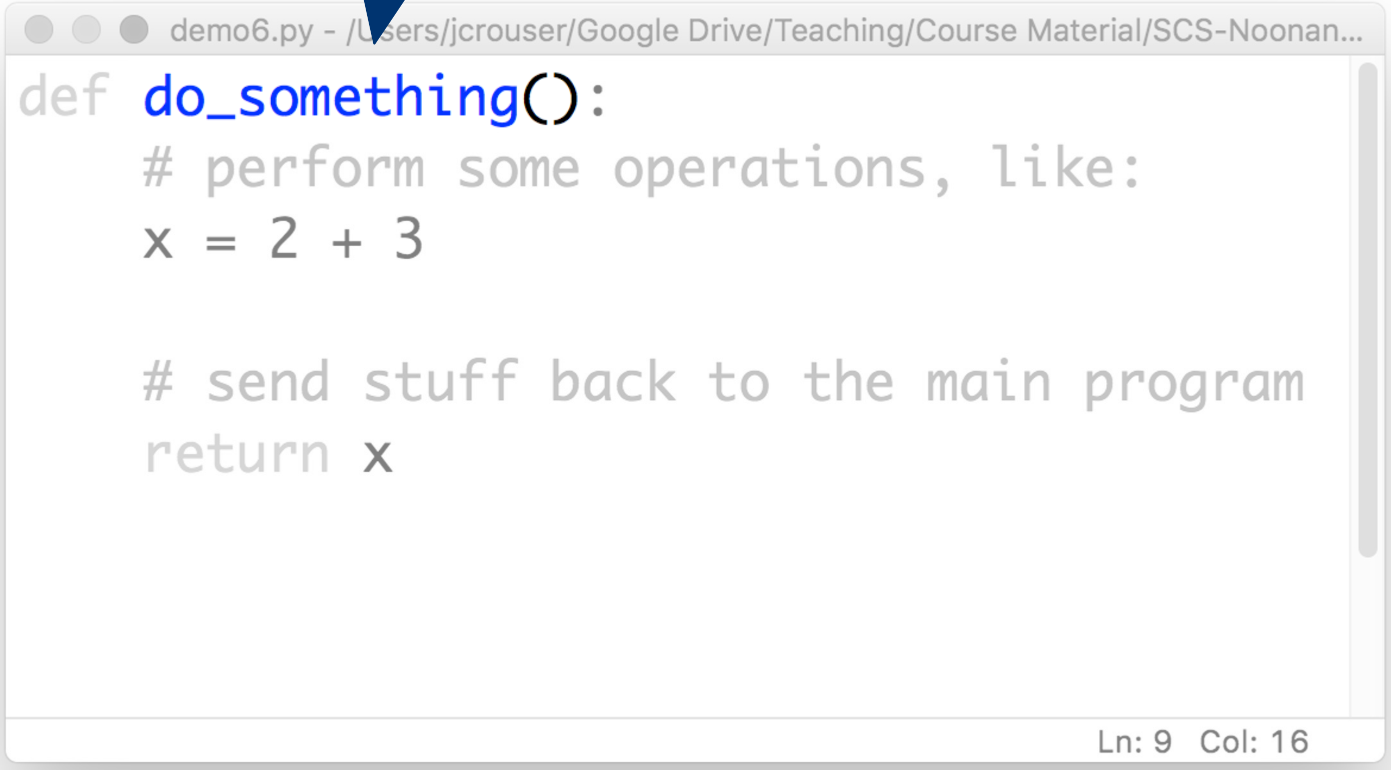

demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...

```python
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 name


demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...

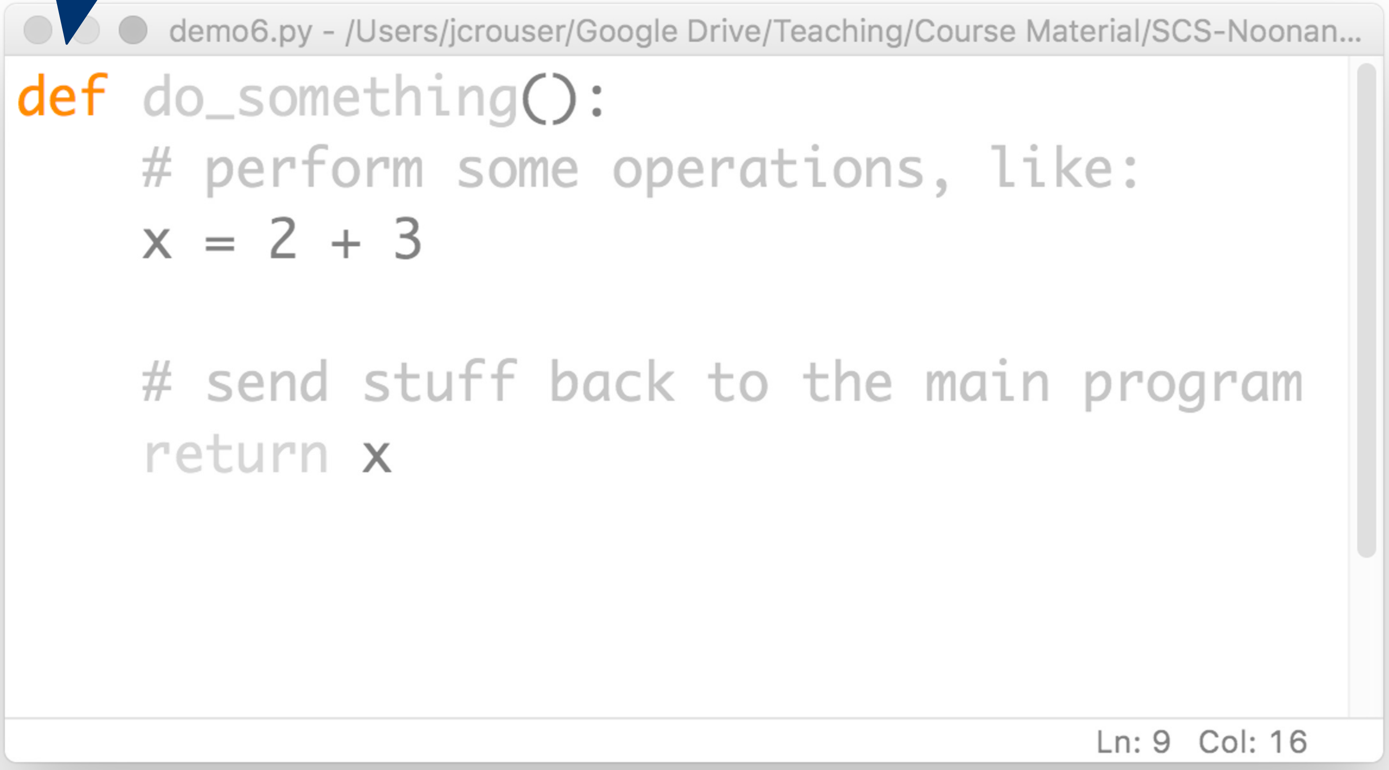
```
def do_something():
    # perform some operations, like:
    x = 2 + 3

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

Ln: 9 Col: 16

Convention: use **`_underscores_`** or **`camelCase`**

Basic components of a function

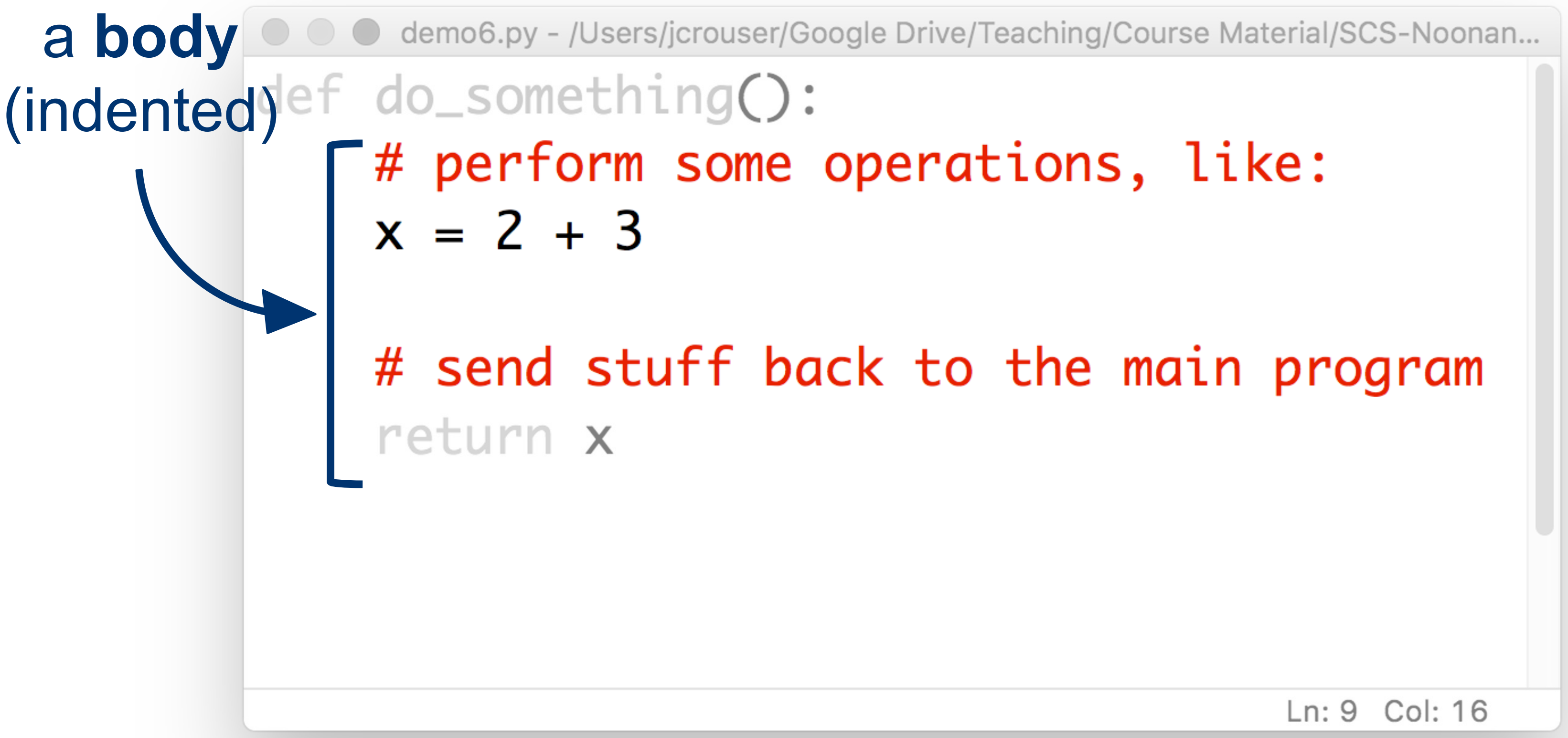
which is defined using the `def` keyword


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
```

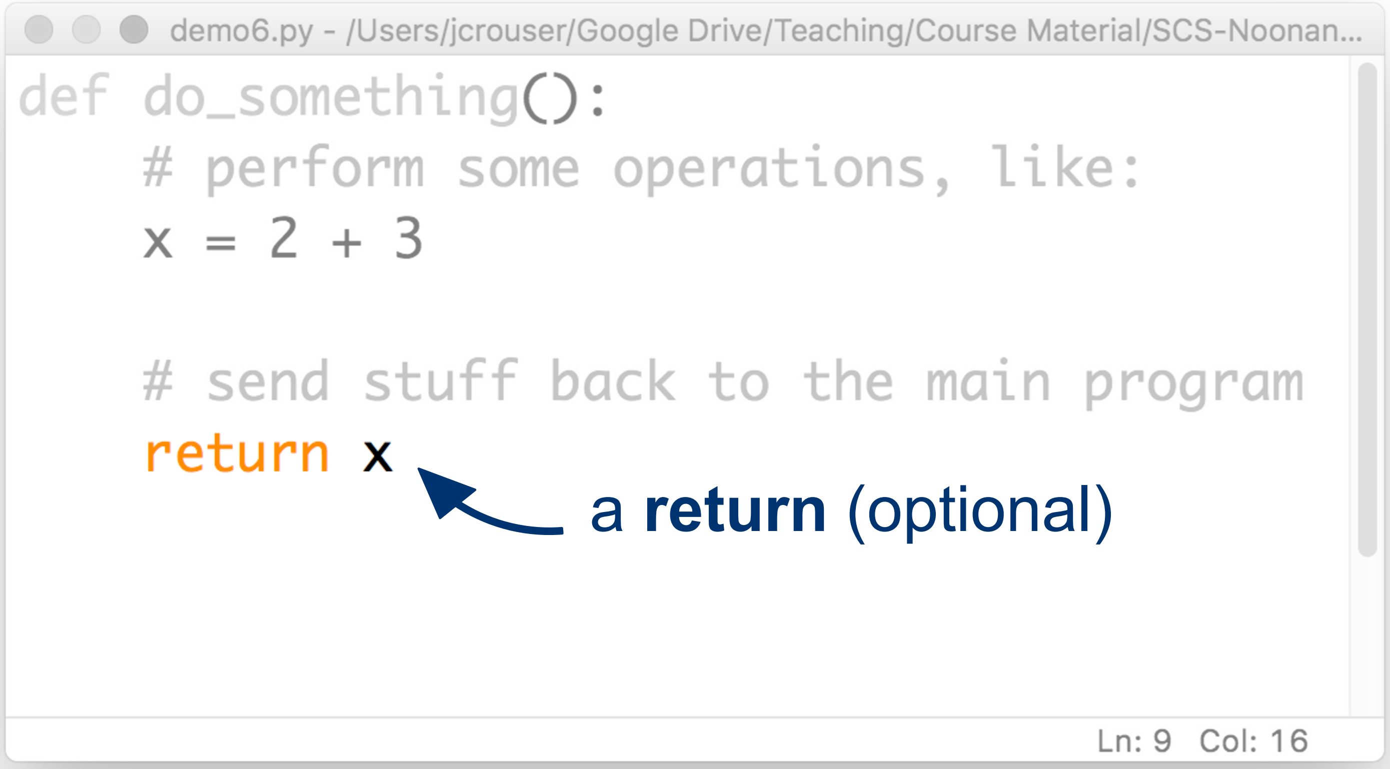
Ln: 9   Col: 16

Basic components of a function


a **body**
(indented)
demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...
Ln: 9 Col: 16

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

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

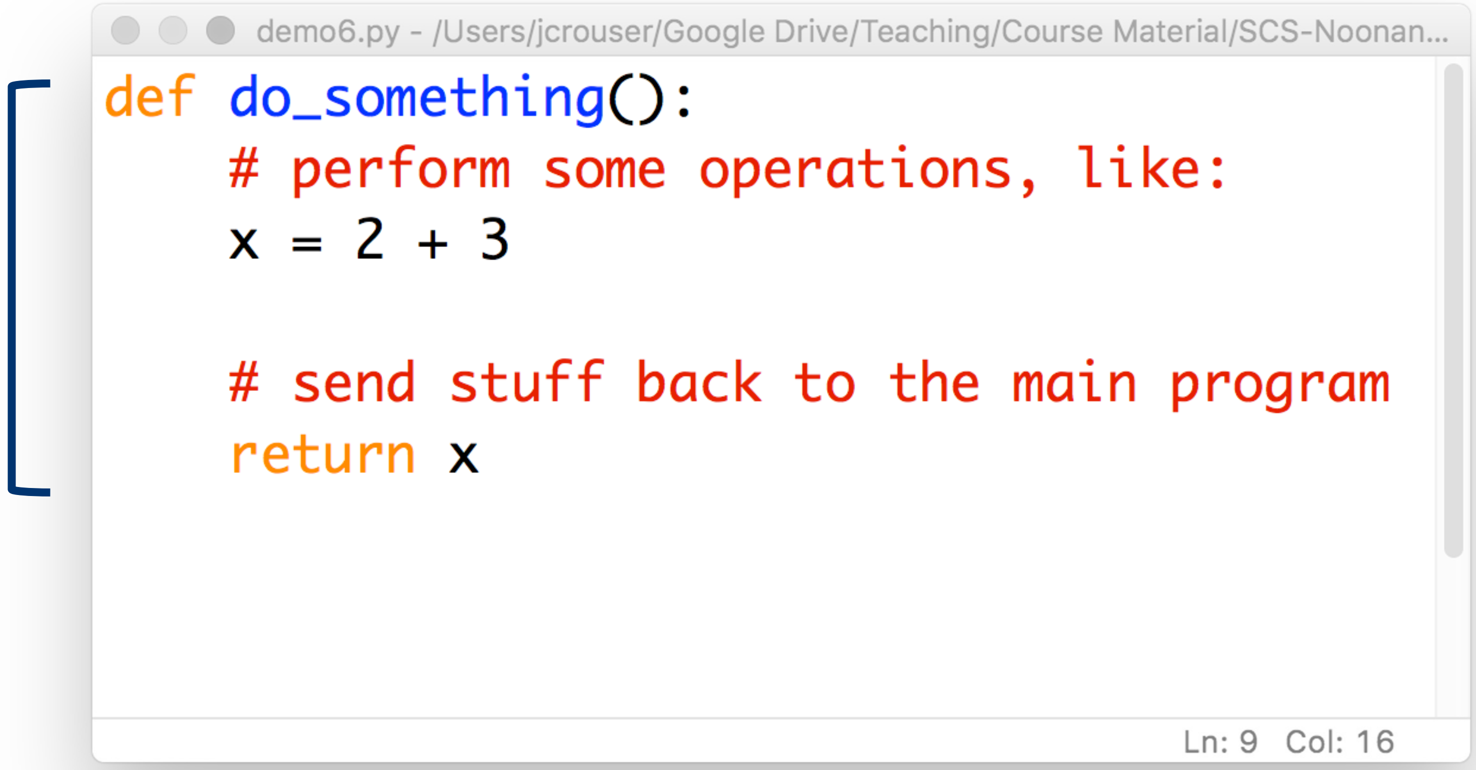
Basic components of a function



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
```

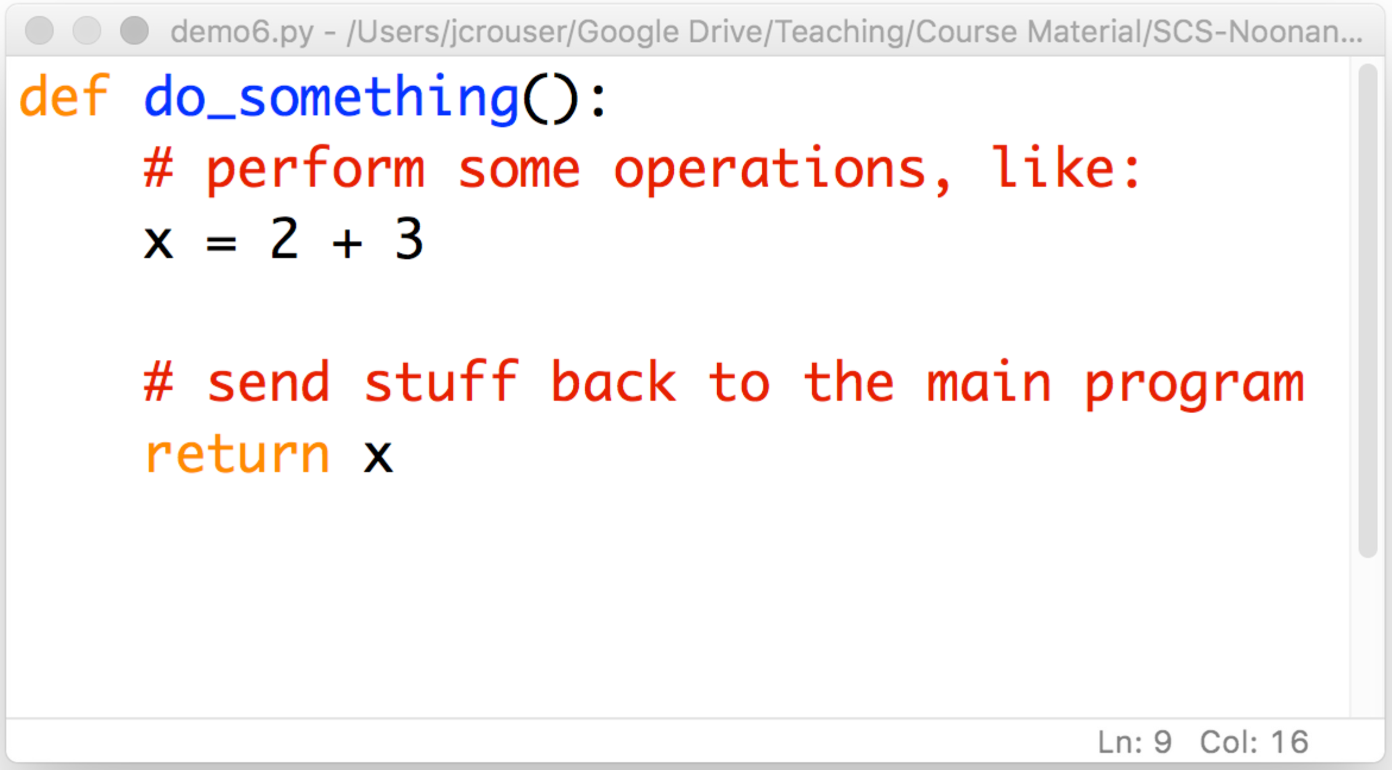
A “function definition”


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
```

Discussion

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

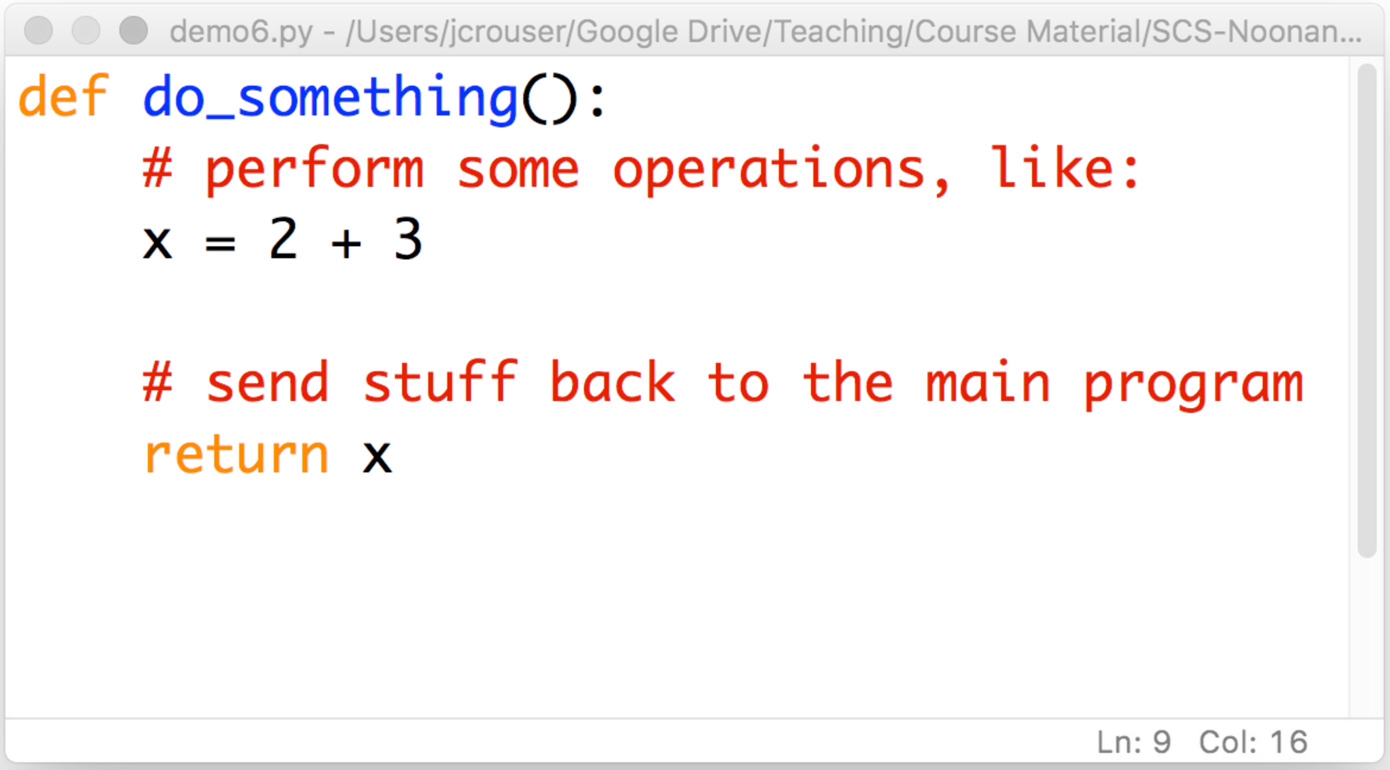


demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...

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

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

Ln: 9   Col: 16

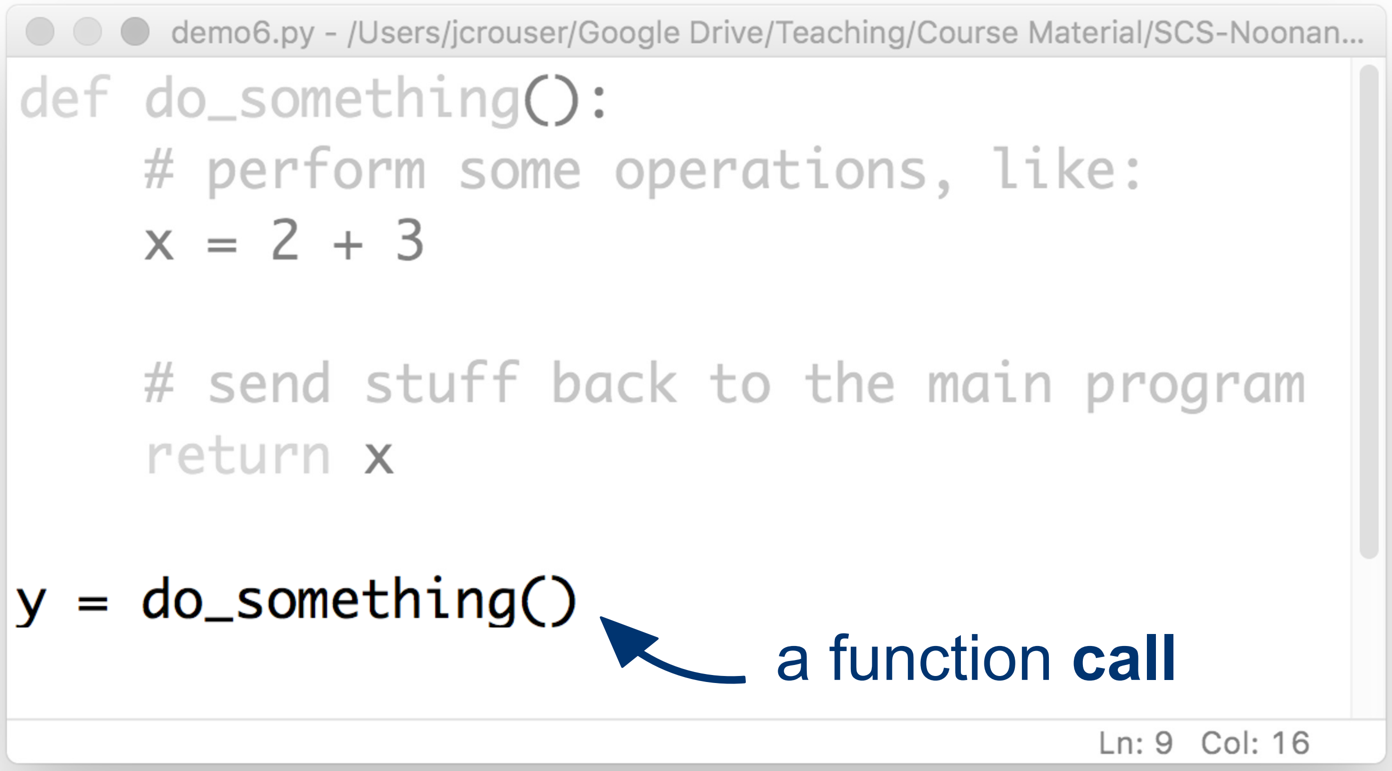
A "function definition" is a **description**


demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...
Ln: 9 Col: 16

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

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

(but not a **directive**)

Function calls: "hey, Python! do this"

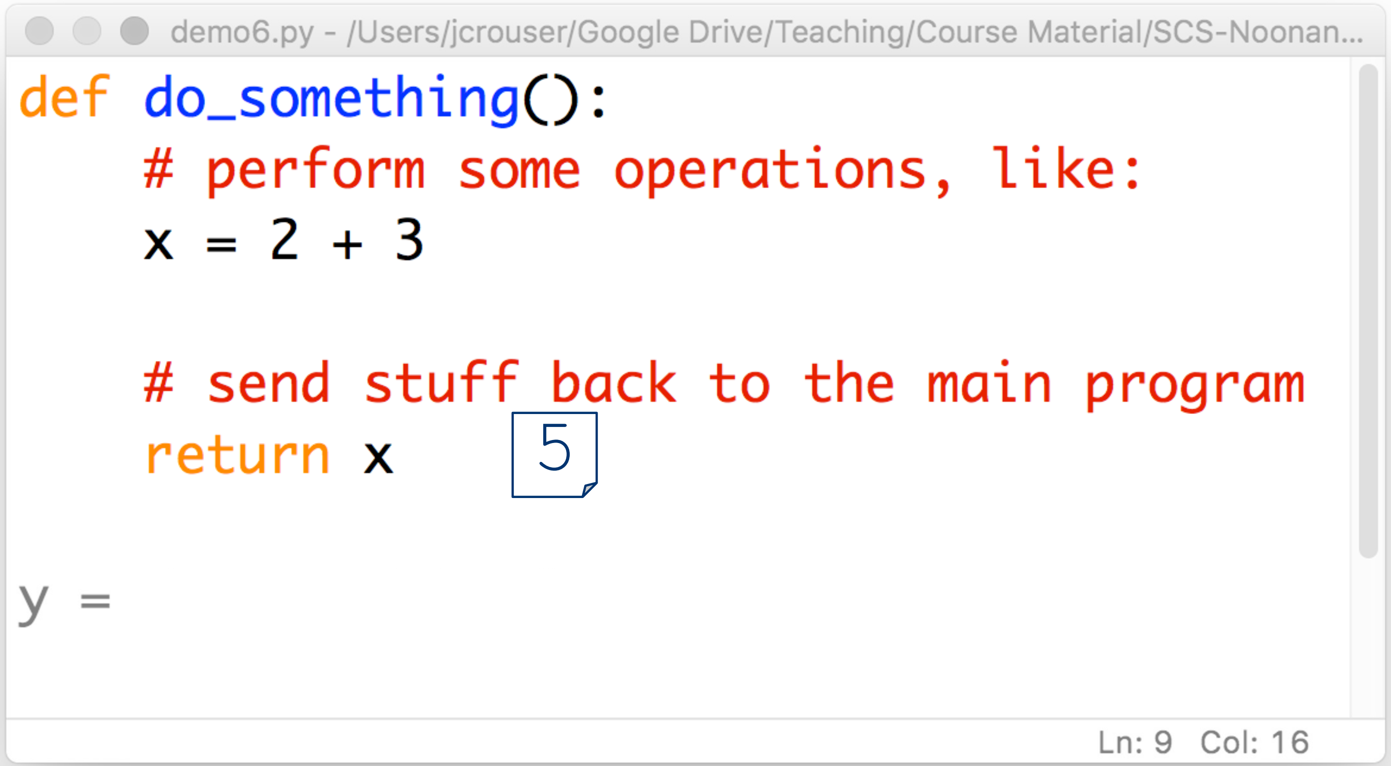


demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...

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

    # send stuff back to the main program
    return x

y = do_something()
```

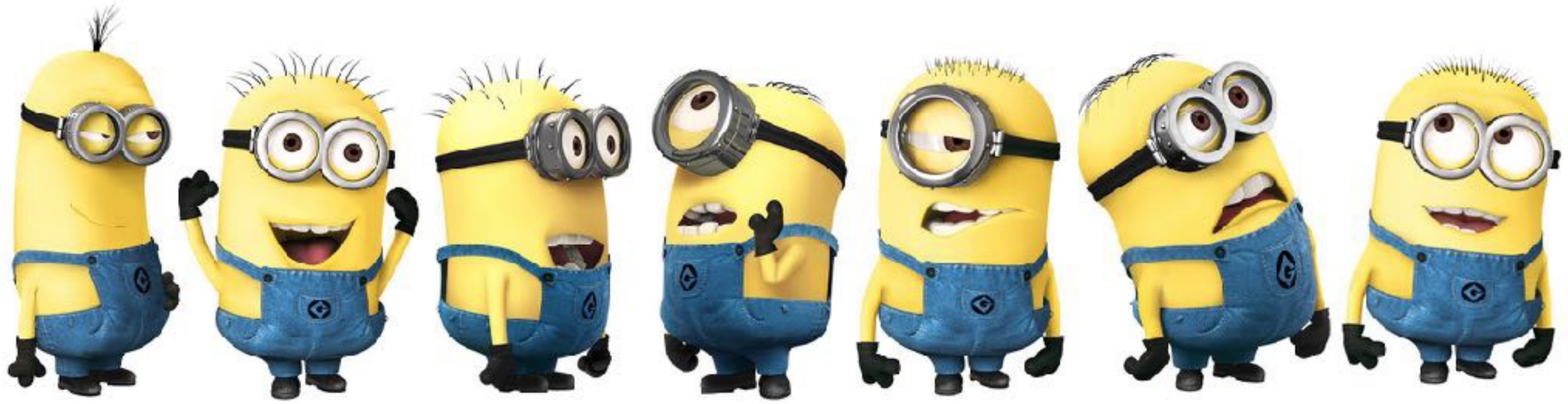
a function **call**

Function calls: "hey, Python! do this"


demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...
Ln: 9 Col: 16

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

    # send stuff back to the main program
    return x      5


y =
```

An analogy



`functions` are your **MINIONS**

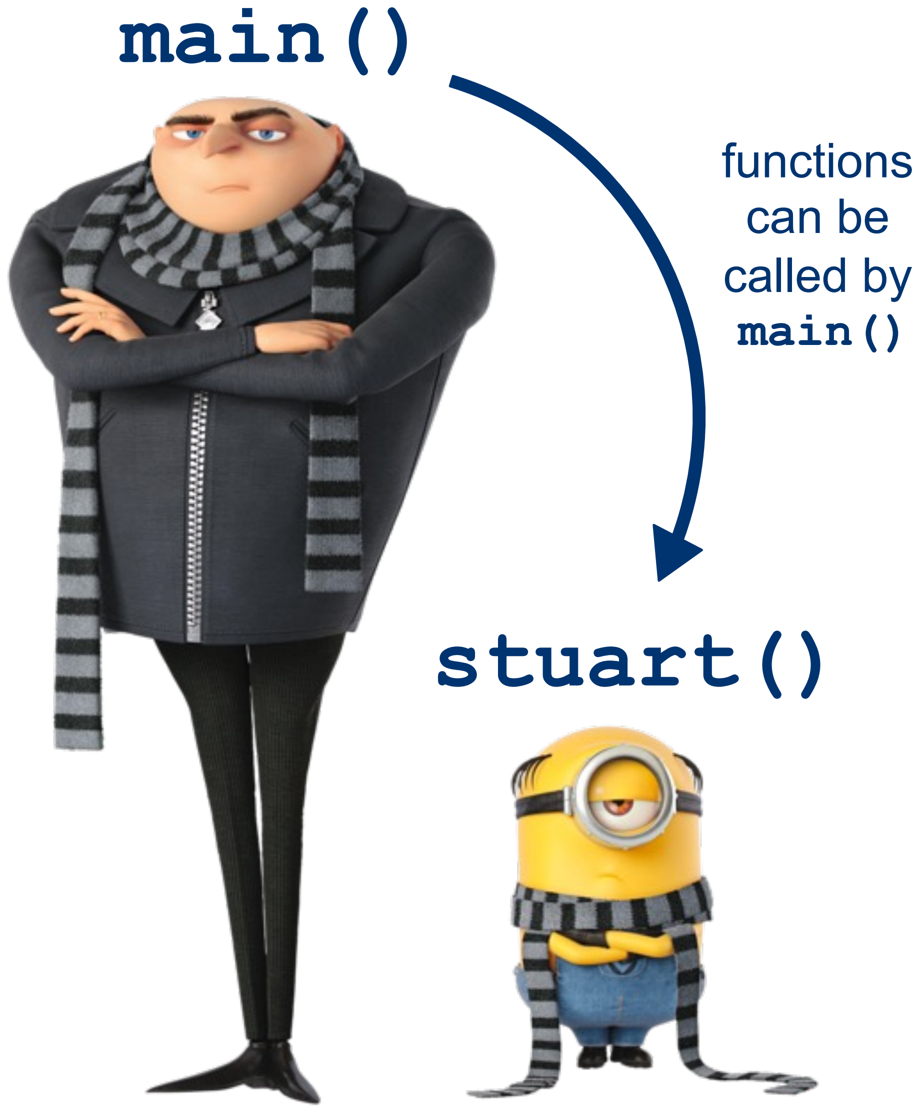
An analogy

`functions` have **NAMES**

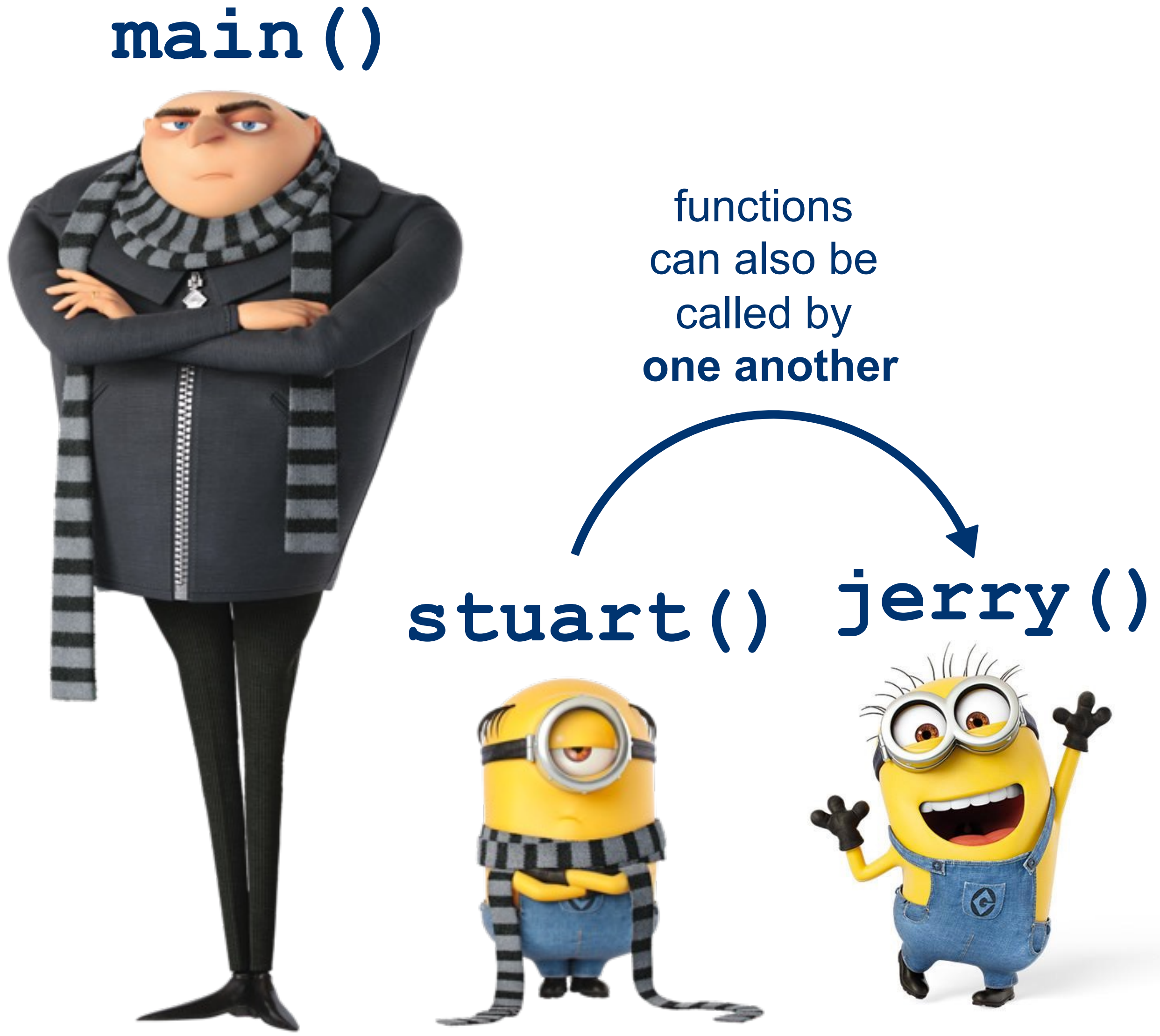
An analogy

they only work when you **CALL** them

An analogy



An analogy

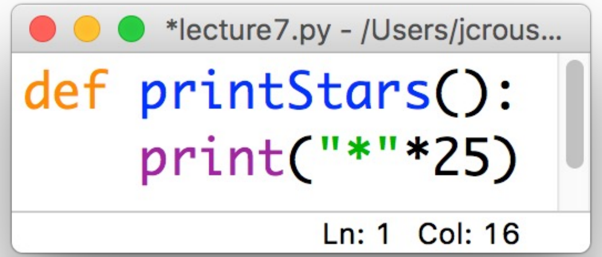

main()
functions
can also be
called by
one another
stuart()
jerry()

Two kinds of functions

Some functions always do the same thing

Two kinds of functions

Some functions always do the same thing



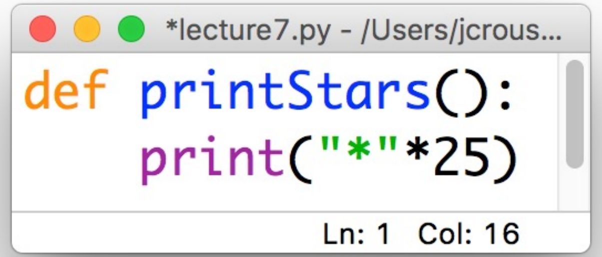
```
def printStars():
    print("*"*25)
```

```
printStars()
printStars()
printStars()
```

Two kinds of functions

Some functions always do the same thing

Others adjust their behavior based on what we give them


*lecture7.py - /Users/jcrous...

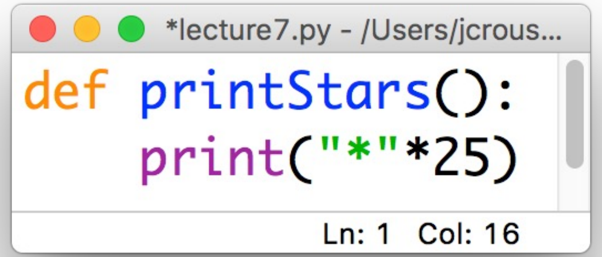
```
def printStars():
    print("*"*25)
```

Ln: 1   Col: 16

```
printStars()
printStars()
printStars()
```

Two kinds of functions

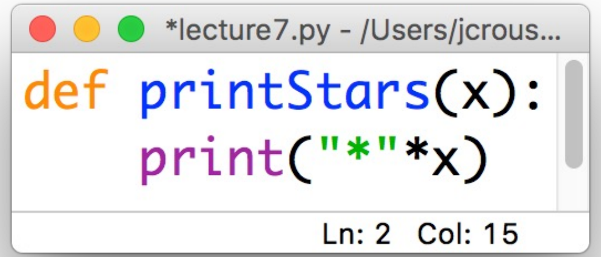
Some functions always do the same thing



```
def printStars():
    print("*"*25)
```

```
printStars()
printStars()
printStars()
```

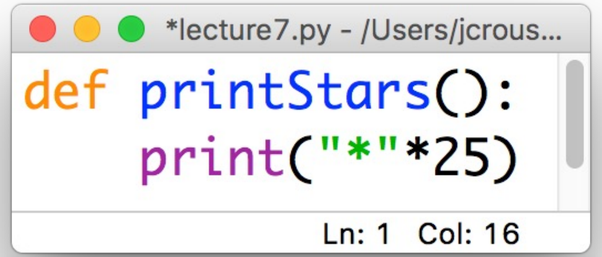
Others adjust their behavior based on what we give them



```
def printStars(x):
    print("*"*x)
```

Two kinds of functions

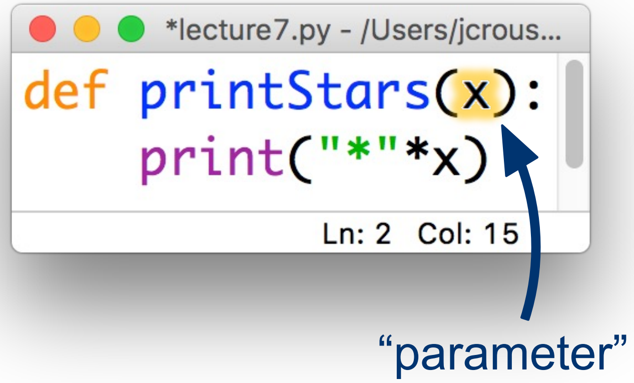
Some functions always do the same thing



```
def printStars():
    print("*"*25)
```

```
printStars()
printStars()
printStars()
```

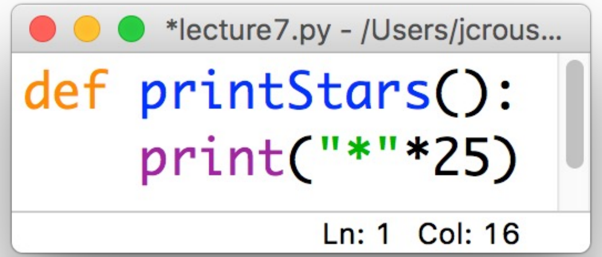
Others adjust their behavior based on what we give them



```
def printStars(x):
    print("*"*x)
```

Two kinds of functions

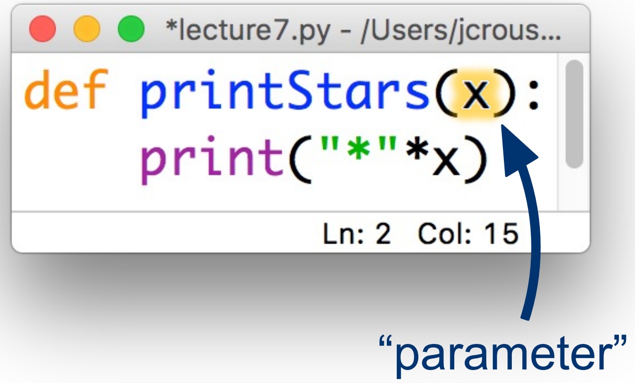
Some functions always do the same thing



```python
def printStars():
    print("*"*25)
```

```python
printStars()
printStars()
printStars()
```

Others adjust their behavior based on what we give them



```python
def printStars(x):
    print("*"*x)
```

```python
printStars(5)
printStars(32)
printStars(1527)
```

15-minute exercise: Happy Birthday

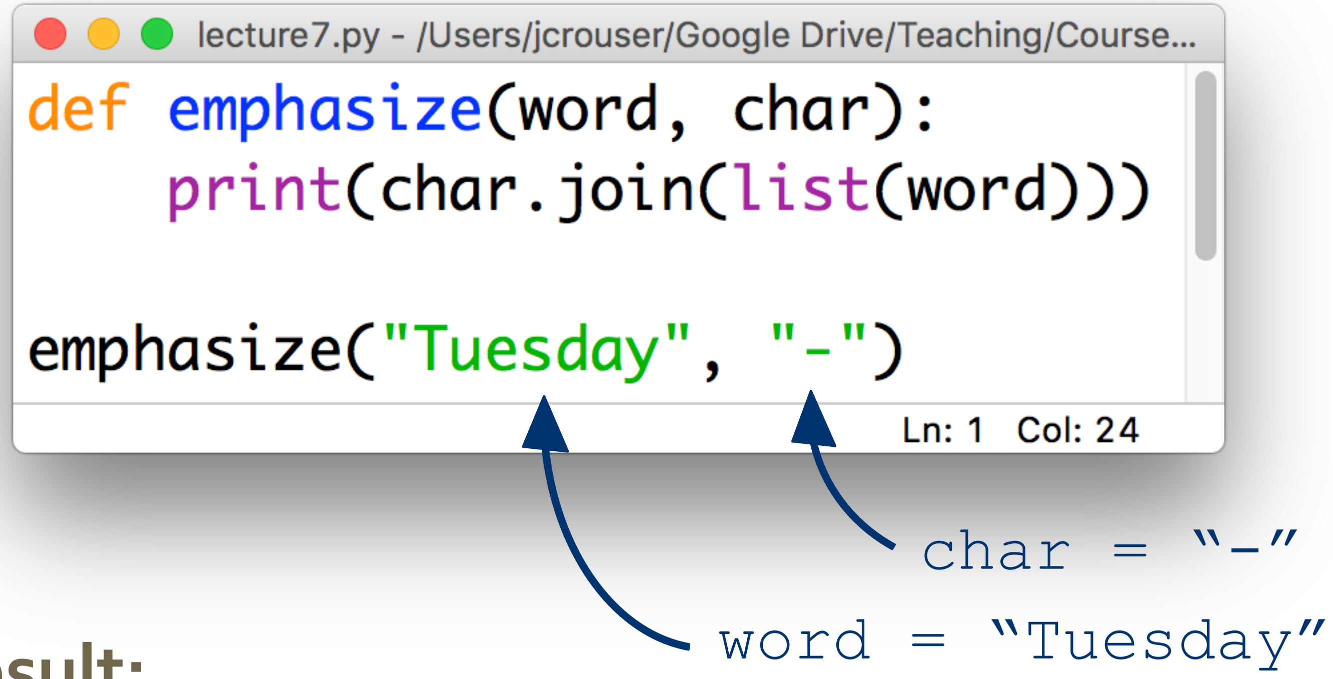
- Write a function named **`happyBirthday`** that takes in a string, **`name`**, and prints out the lyrics to the song "Happy Birthday" with the name inserted:

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

- Use **`input(…)`** to get the user's name, and then call your function with the user's name to print their happy birthday song

Parameters

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



lecture7.py - /Users/jcrouser/Google Drive/Teaching/Course...

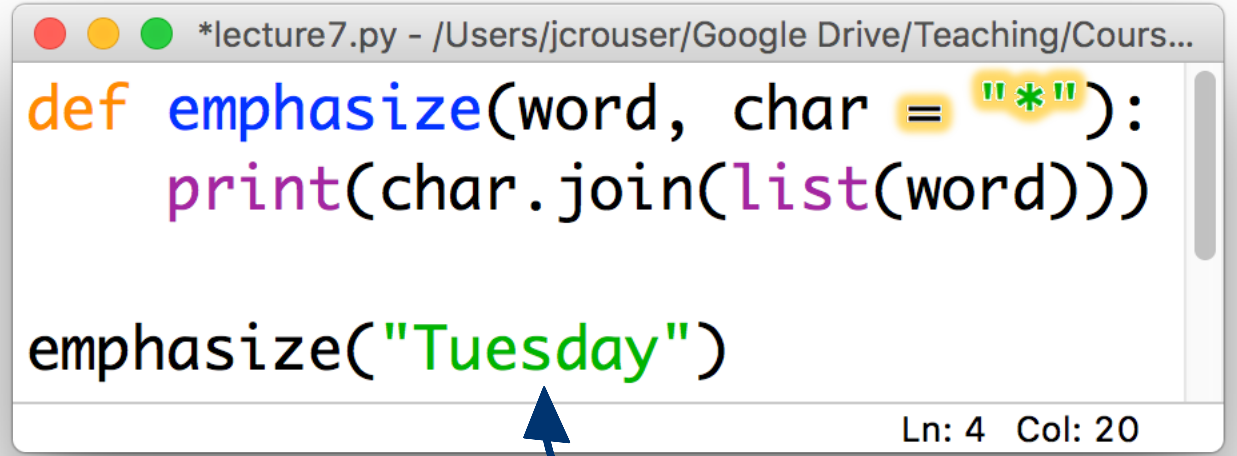
```python
def emphasize(word, char):
    print(char.join(list(word)))

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

- **Result:**

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

Default parameters

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



*lecture7.py - /Users/jcrouser/Google Drive/Teaching/Cours...
Ln: 4   Col: 20

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

emphasize("Tuesday")
```

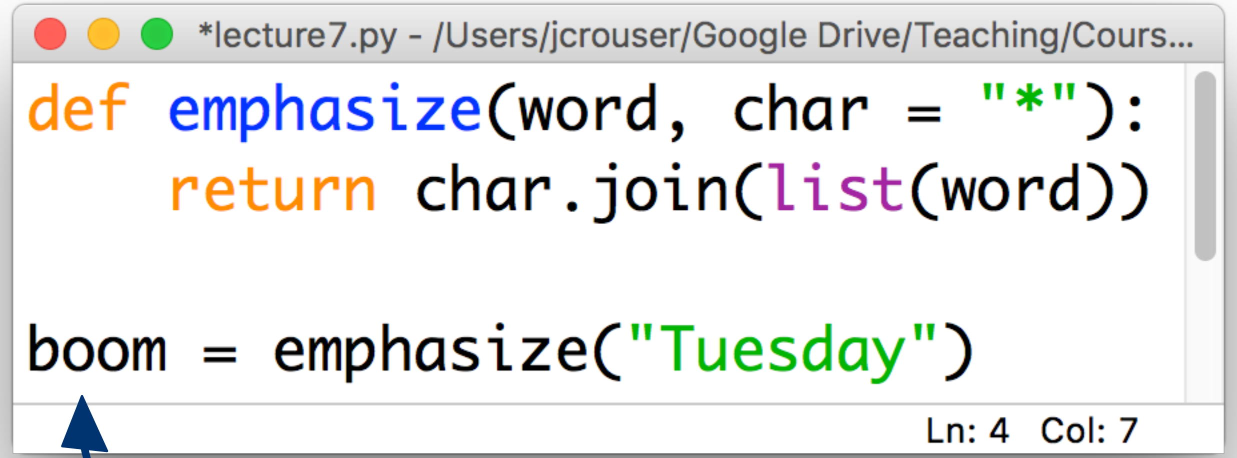
only one parameter

- **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...

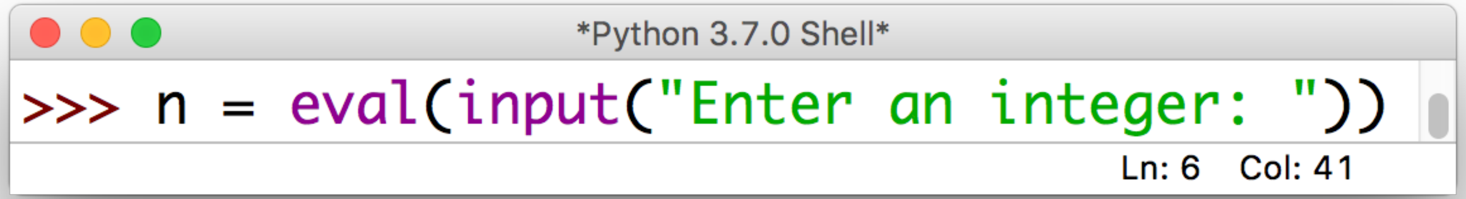
```
def emphasize(word, char = "*"):
    return char.join(list(word))

boom = emphasize("Tuesday")
```

Ln: 4   Col: 7

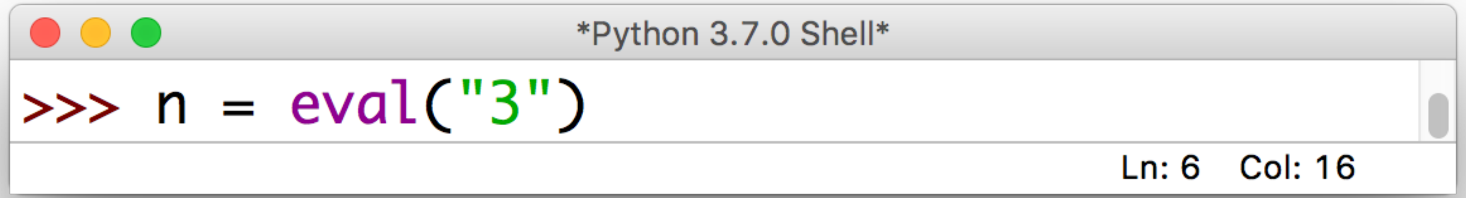
the results of the **`return`** in **`emphasize()`** are stored in **`boom`**

Advanced: chaining functions

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



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



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

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: verb, what the function does
- **Important**: one function = one task