Introduction to Computation for the Humanities and Social Sciences

CS 3

Chris Tanner

Lecture 9

"I need reading material … dictionary"
— Eminem

Sept 14, 2018 (MGK diss track)

REFRESHER

- To check if an item is in a list, you can use “**in**”
- You have already seen this with Strings (checking if a substring exists)

```
cs3_students = [“Mary”, “John”, “Emily”]
cs17_students = [“Elizabeth”, “Mary”, “Frank”]

if “Emily” in names:
    print(“Emily is a student in cs3!”)

for student in cs3_students:
    if student in cs17_students:
        print(student + “ is in both courses!”)
```

Lecture 9

- Iteration (recap)
- Modules
    - sys.argv
    - os
- Dictionaries

Iteration

- Often, we want to repeat a set of computations many times on slightly different data
- To create an iterative process, we can make use of **loops**

Iteration

- **Definition:** A **loop (iteration)** is a construct of programming languages which provides the ability to repeat an operation a particular number of times

- It's one of the core functionalities of programming languages

- **For loops**: Perform the computation step *for* a specified number of times (e.g., **for** N times or **for** each item in a list)

- **While loops**: Perform the computation step *while* a given condition is True

Iterables

- Data Structures that you can iterate over are called **iterables**

- **List, Sets, Dictionaries, Tuples** are iterables, but so are a few other things: **Strings**, **input files**, etc

Iteration

for loop

```
cs17_students = [“Elizabeth”, “Mary”, “Frank”]

i = 0
for student in cs17_students:
    print(student)
    i += 1

print(i)
```

Iteration

for loop

```
cs17_students = [“Elizabeth”, “Mary”, “Frank”]

for (2) in (1):
    (3)
```

(1) **Input iterable:** a list, set, dictionary, range, etc

(2) **The *temporary* iteration variable:** give a descriptive name that matches the type of the individual item in the operable

(3) **The iteration computation**: indented code that occurs once for *each item* in the input iterable

Iteration

For Loops

range(x) function allows you to loop through numbers from 0 to x-1. look up the function for other options.

```
# loops through 50 times, where i will
# be 0 the 1st time, and 49 the last time
for i in range(50):
    print(“i: ” + str(i))
    if i % 3 == 0: # checks if remainder is 0
        print(str(i) + “ is divisible by 3!”)
```

Iteration

For Loops

Iterate through each item in a list

```
fruits = [‘apple’, ‘litchi’, ‘rambutan’, ‘banana’]

for fruit in fruits:
    print(“current fruit: ” + fruit)

print(“total # of fruits: “ + str(len(fruits)))
```

TERMINAL OUTPUT:

Iteration

For Loops

Iterate through each item in a list

```
fruits = [‘apple’, ‘litchi’, ‘rambutan’, ‘banana’]

for fruit in fruits:
    print(“current fruit: ” + fruit)

print(“total # of fruits: “ + str(len(fruits)))
```

TERMINAL OUTPUT:

Iteration

For Loops

Iterate through each item in a list

```
fruits = [‘apple’, ‘litchi’, ‘rambutan’, ‘banana’]

for fruit in fruits:
    print(“current fruit: ” + fruit)

print(“total # of fruits: “ + str(len(fruits)))
```

TERMINAL OUTPUT:

Iteration

For Loops

Iterate through each item in a list

```
fruits = [‘apple’, ‘litchi’, ‘rambutan’, ‘banana’]

for fruit in fruits:
    print(“current fruit: ” + fruit)

print(“total # of fruits: “ + str(len(fruits)))
```

TERMINAL OUTPUT:

```
current fruit: apple
```

Iteration

For Loops

Iterate through each item in a list

```
fruits = [‘apple’, ‘litchi’, ‘rambutan’, ‘banana’]

for fruit in fruits:
    print(“current fruit: ” + fruit)

print(“total # of fruits: “ + str(len(fruits)))
```

TERMINAL OUTPUT:

```
current fruit: apple
```

Iteration

For Loops

Iterate through each item in a list

```
fruits = [‘apple’, ‘litchi’, ‘rambutan’, ‘banana’]

for fruit in fruits:
    print(“current fruit: ” + fruit)

print(“total # of fruits: “ + str(len(fruits)))
```

TERMINAL OUTPUT:

```
current fruit: apple
current fruit: litchi
```

Iteration

For Loops

Iterate through each item in a list

```
fruits = [‘apple’, ‘litchi’, ‘rambutan’, ‘banana’]

for fruit in fruits:
    print(“current fruit: ” + fruit)

print(“total # of fruits: “ + str(len(fruits)))
```

TERMINAL OUTPUT:

```
current fruit: apple
current fruit: litchi
```

Iteration

For Loops

Iterate through each item in a list

```
fruits = [‘apple’, ‘litchi’, ‘rambutan’, ‘banana’]

for fruit in fruits:
    print(“current fruit: ” + fruit)

print(“total # of fruits: “ + str(len(fruits)))
```

TERMINAL OUTPUT:

```
current fruit: apple
current fruit: litchi
current fruit: rambutan
```

Iteration

For Loops

Iterate through each item in a list

```
fruits = [‘apple’, ‘litchi’, ‘rambutan’, ‘banana’]

for fruit in fruits:
    print(“current fruit: ” + fruit)

print(“total # of fruits: “ + str(len(fruits)))
```

TERMINAL OUTPUT:

```
current fruit: apple
current fruit: litchi
current fruit: rambutan
```

Iteration

For Loops

Iterate through each item in a list

```
fruits = [‘apple’, ‘litchi’, ‘rambutan’, ‘banana’]

for fruit in fruits:
    print(“current fruit: ” + fruit)

print(“total # of fruits: “ + str(len(fruits)))
```

TERMINAL OUTPUT:

```
current fruit: apple
current fruit: litchi
current fruit: rambutan
current fruit: banana
```

Iteration

For Loops

Iterate through each item in a list

```
fruits = [‘apple’, ‘litchi’, ‘rambutan’, ‘banana’]

for fruit in fruits:
    print(“current fruit: ” + fruit)

print(“total # of fruits: “ + str(len(fruits)))
```

TERMINAL OUTPUT:

```
current fruit: apple
current fruit: litchi
current fruit: rambutan
current fruit: banana
total # of fruits: 4
```

Lecture 9

- Iteration (recap)
- Modules
  - sys.argv
  - os
- Dictionaries

Modules

- Libraries provide functionality written by others for use within our own program
- To make use of these libraries, we import their functionality which are built in **modules**
- Python has a LOT of core modules, you can see a list of them at https://docs.python.org/3/py-modindex.html
- Installing Anaconda initially, gave us even more modules we can use https://docs.continuum.io/anaconda/pkg-docs#python-3-6

Modules

importing a module

- We have already used random, and we briefly looked at math
- https://docs.python.org/3/library/random.html and https://docs.python.org/3/library/math.html

```
import random


def get_computers_choice():
    computers_choice = random.choice([“rock”, “paper”, “scissors”])
    return computers_choice
```

Modules

sys module

- The words that are typed after your Python program's name in the terminal are called **command-line arguments**

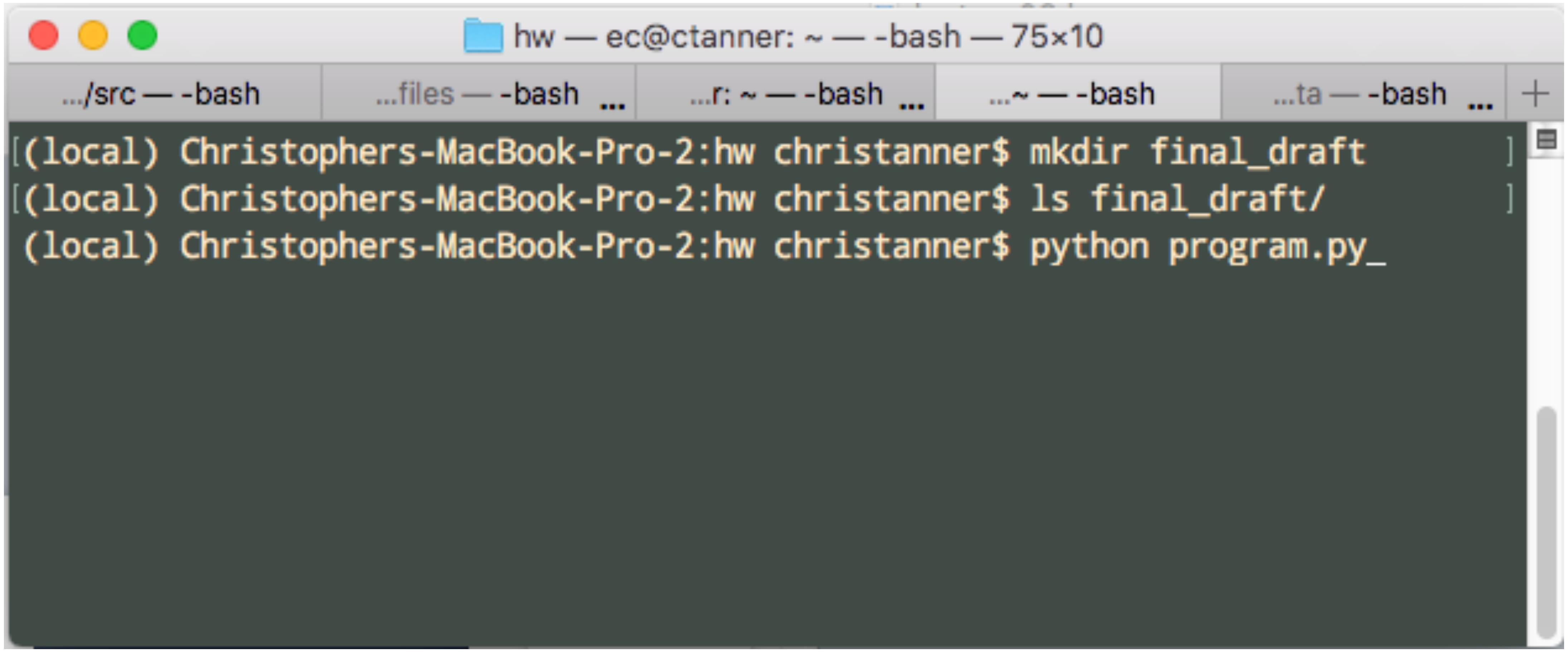
- They are available at **sys.argv**

- This provides a new modality of input for your programs, input specified at the time the program is run!

- **Highly useful:** instead of exclusively relying on manually-coded parameters like input filenames for our programs to use, we can let the user specify which filename (or other values) for our program to use!

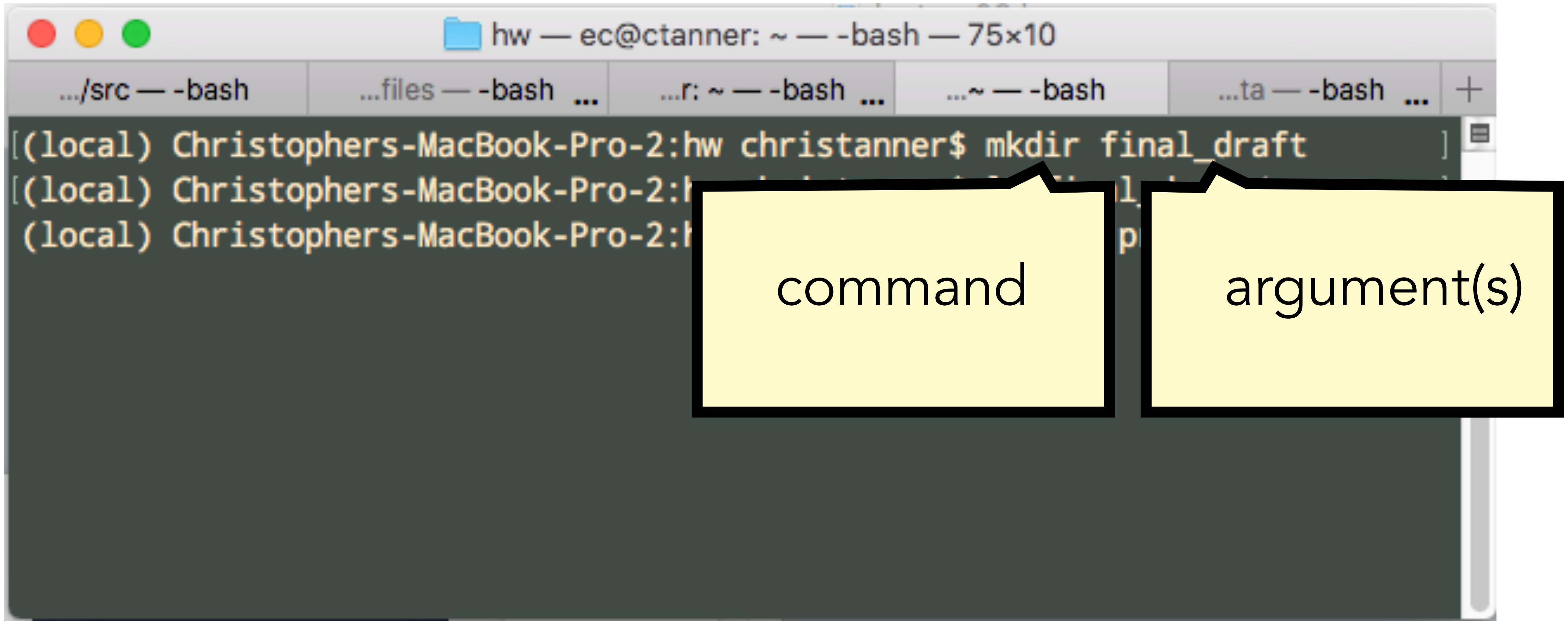
Modules

sys module



```
(local) Christophers-MacBook-Pro-2:hw christanner$ mkdir final_draft
(local) Christophers-MacBook-Pro-2:hw christanner$ ls final_draft/
(local) Christophers-MacBook-Pro-2:hw christanner$ python program.py_
```

sys module



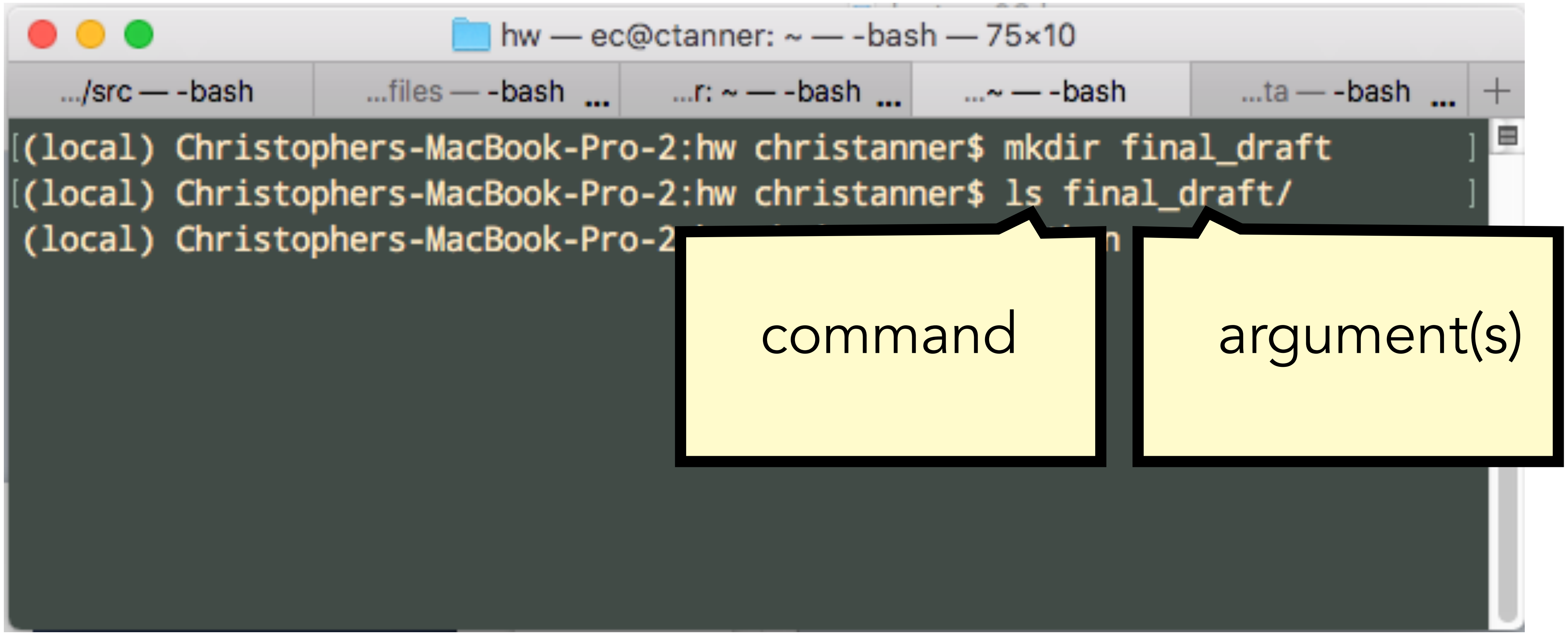
```
(local) Christophers-MacBook-Pro-2:hw christanner$ mkdir final_draft
(local) Christophers-MacBook-Pro-2:h
(local) Christophers-MacBook-Pro-2:h
```

command

argument(s)

Modules

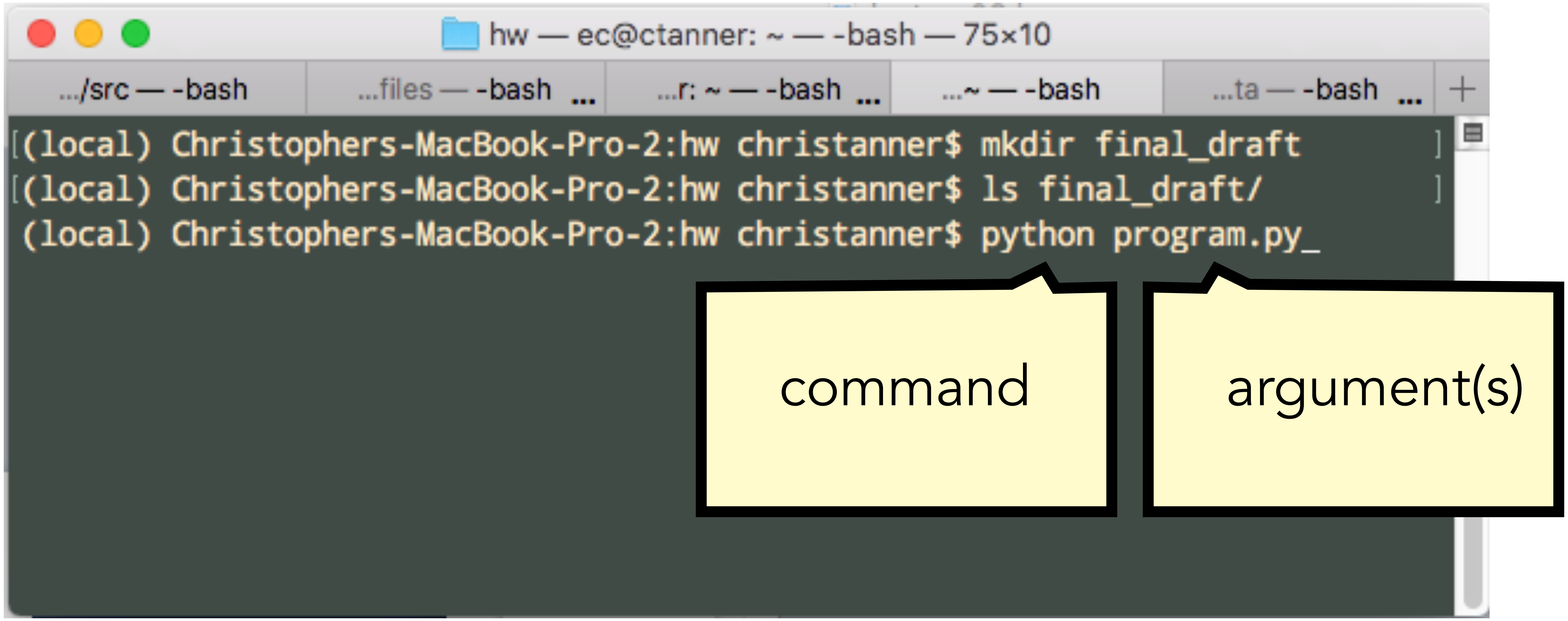
sys module



```
(local) Christophers-MacBook-Pro-2:hw christanner$ mkdir final_draft
(local) Christophers-MacBook-Pro-2:hw christanner$ ls final_draft/
(local) Christophers-MacBook-Pro-2
```

Modules

sys module



```
[(local) Christophers-MacBook-Pro-2:hw christanner$ mkdir final_draft
[(local) Christophers-MacBook-Pro-2:hw christanner$ ls final_draft/
(local) Christophers-MacBook-Pro-2:hw christanner$ python program.py_
```

Modules

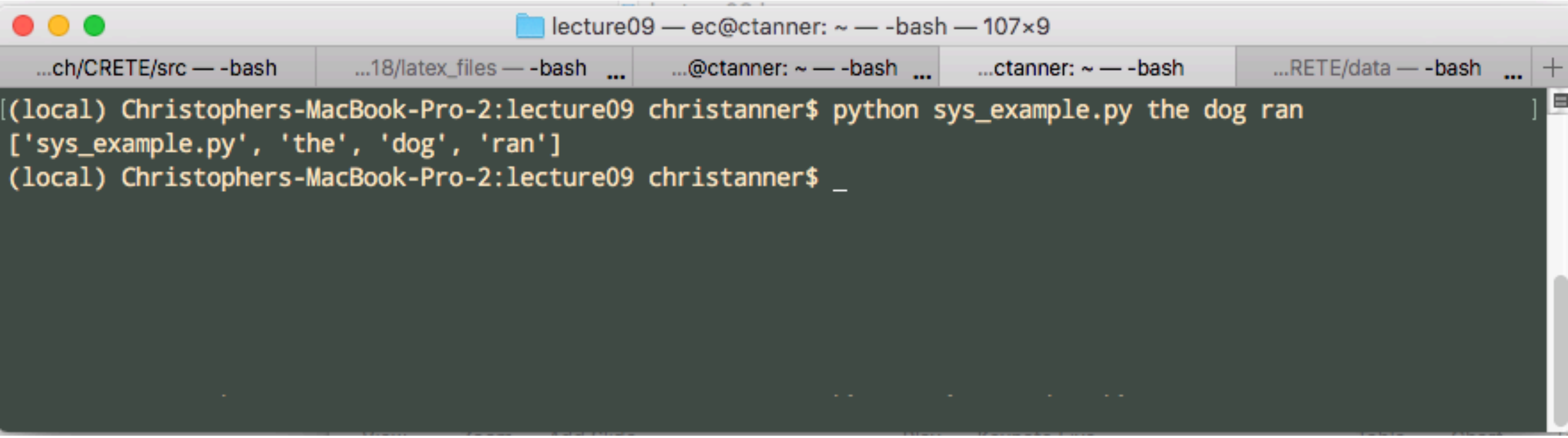
sys.argv

- **sys.argv** is a `list` of values that the user typed on the command line

sys_example.py

```python
import sys

print(sys.argv)
```



```
(local) Christophers-MacBook-Pro-2:lecture09 christanner$ python sys_example.py the dog ran
['sys_example.py', 'the', 'dog', 'ran']
(local) Christophers-MacBook-Pro-2:lecture09 christanner$ _
```

Modules

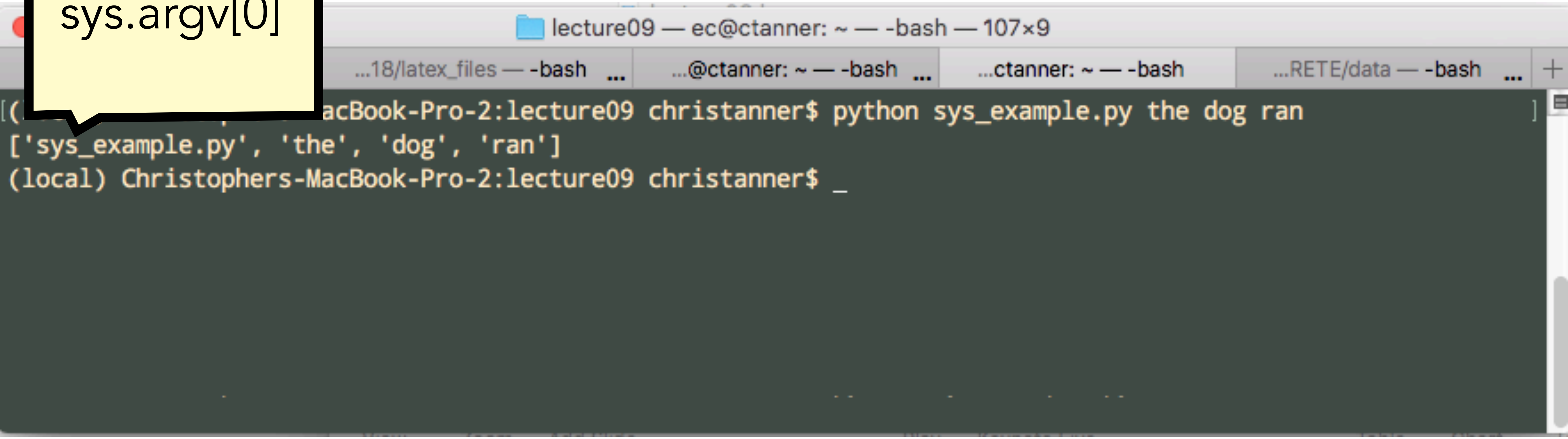
sys.argv

- **sys.argv** is a **`list`** of values that the user typed on the command line

sys_example.py

```
import sys

print(sys.argv)
```


lecture09 — ec@ctanner: ~ — -bash — 107×9
...18/latex_files — -bash
...@ctanner: ~ — -bash
...ctanner: ~ — -bash
...RETE/data — -bash

sys.argv[0]

```
[(                        acBook-Pro-2:lecture09 christanner$ python sys_example.py the dog ran
['sys_example.py', 'the', 'dog', 'ran']
(local) Christophers-MacBook-Pro-2:lecture09 christanner$ _
```

Modules

sys.argv

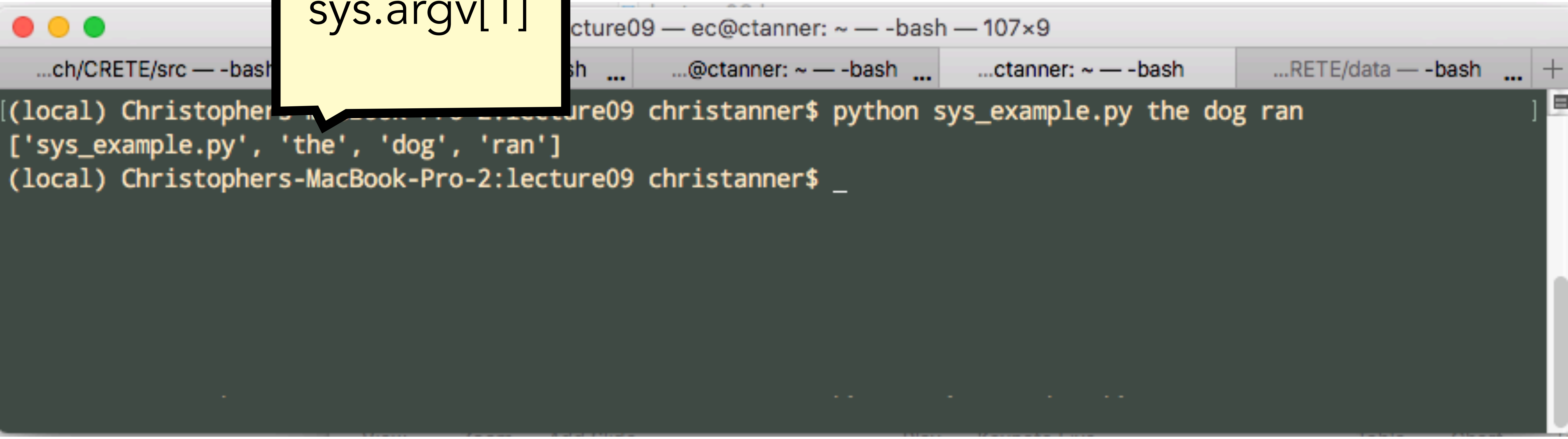
- **sys.argv** is a **`list`** of values that the user typed on the command line

sys_example.py

```python
import sys

print(sys.argv)
```

sys.argv[1]



```
(local) Christophers-MacBook-Pro-2:lecture09 christanner$ python sys_example.py the dog ran
['sys_example.py', 'the', 'dog', 'ran']
(local) Christophers-MacBook-Pro-2:lecture09 christanner$ _
```

Modules

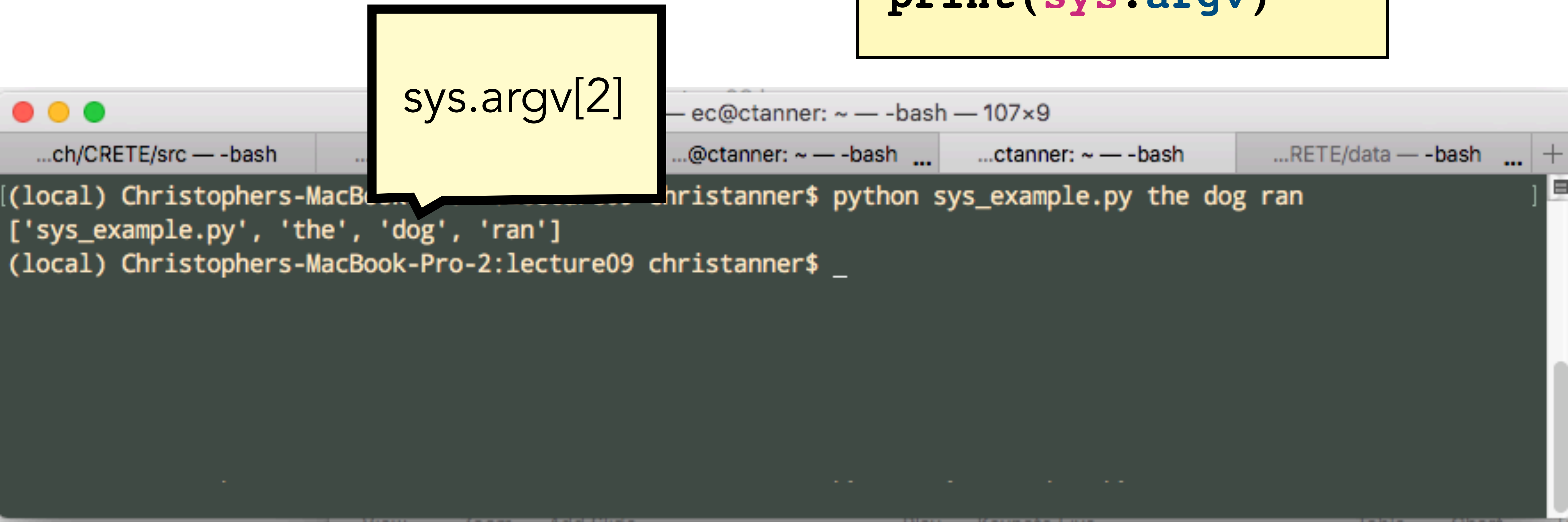
sys.argv

- **sys.argv** is a **`list`** of values that the user typed on the command line

sys_example.py

```
import sys

print(sys.argv)
```



sys.argv[2]

```
(local) Christophers-MacB                    hristanner$ python sys_example.py the dog ran
['sys_example.py', 'the', 'dog', 'ran']
(local) Christophers-MacBook-Pro-2:lecture09 christanner$ _
```

Modules

sys.argv

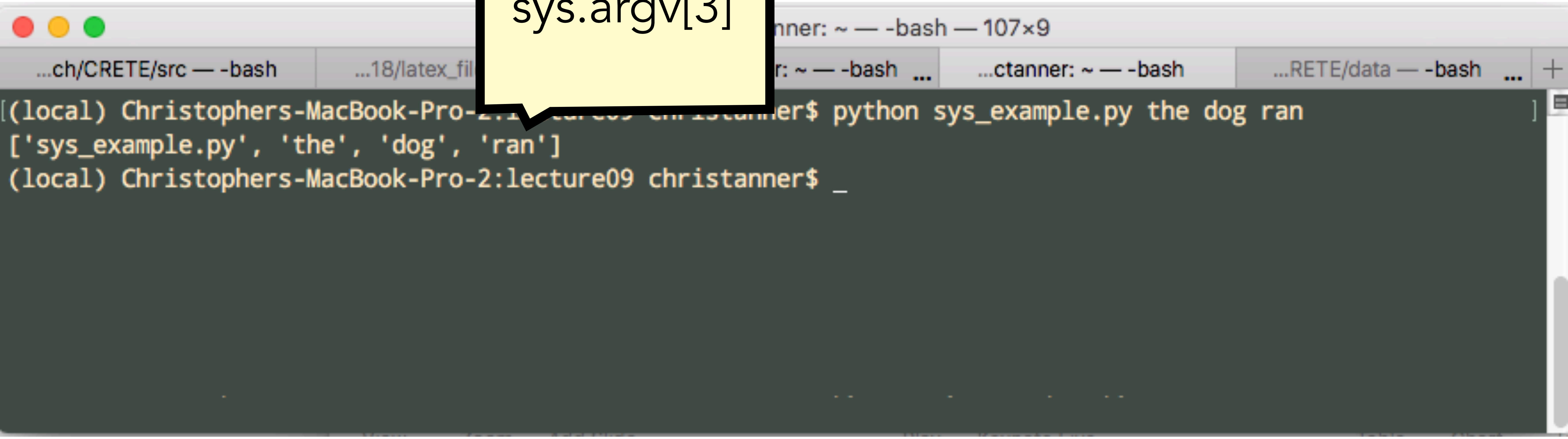
- **sys.argv** is a **`list`** of values that the user typed on the command line

sys_example.py

```python
import sys

print(sys.argv)
```

sys.argv[3]



```
(local) Christophers-MacBook-Pro-2:lecture09 christanner$ python sys_example.py the dog ran
['sys_example.py', 'the', 'dog', 'ran']
(local) Christophers-MacBook-Pro-2:lecture09 christanner$ _
```

Modules

sys.argv

“the_office.csv”

```
import sys
def main():
    input_file = sys.argv[1]

if __name__ == “__main__”:
    main()
```

```
$  python demo.py the_office.csv
```

Lecture 9

- Iteration (recap)
- Modules
  - sys.argv
  - os
- Dictionaries

Modules

OS (operating system)

- Provides system-independent operating system functionality
- Remember a **path** directs the system where to find a file
- It is a combination of the directories that contain the file and the filename itself
- **os.path** provides useful functionality for paths

Modules

os.path.exists(path_name)

- This function will return a `boolean` that is **True** if it exists on your computer, otherwise it will return **False**

```
import sys
import os
def main():
    input_file = sys.argv[1]
    if not os.path.exists(input_file):
        print(“File “ + filename + “ does not exist”)
        sys.exit()

if __name__ == “__main__”:
    main()
```

Lecture 9

- Iteration (recap)
- Modules
  - sys.argv
  - os
- Dictionaries

Computation — Data Structures

- As our computations get more involved, we can’t rely on text files to have every piece of data exactly the way we want it (we shouldn’t read any particular input file more than once anyway).

- Hence, the importance of and need for storing data into **variables**

- Depending on the computations we care to do, different data structures are more appropriate and helpful than others.

- Often, several different data structures may work, but there’s a limit on what each structure (e.g., list or a single-valued one) can do.

Dictionaries

- Lists are great, but what if you need to store separate pieces of data, each with its own data (e.g., several different counts or lists)

- Example: the number of avocados sold in each US State (according to an input file)

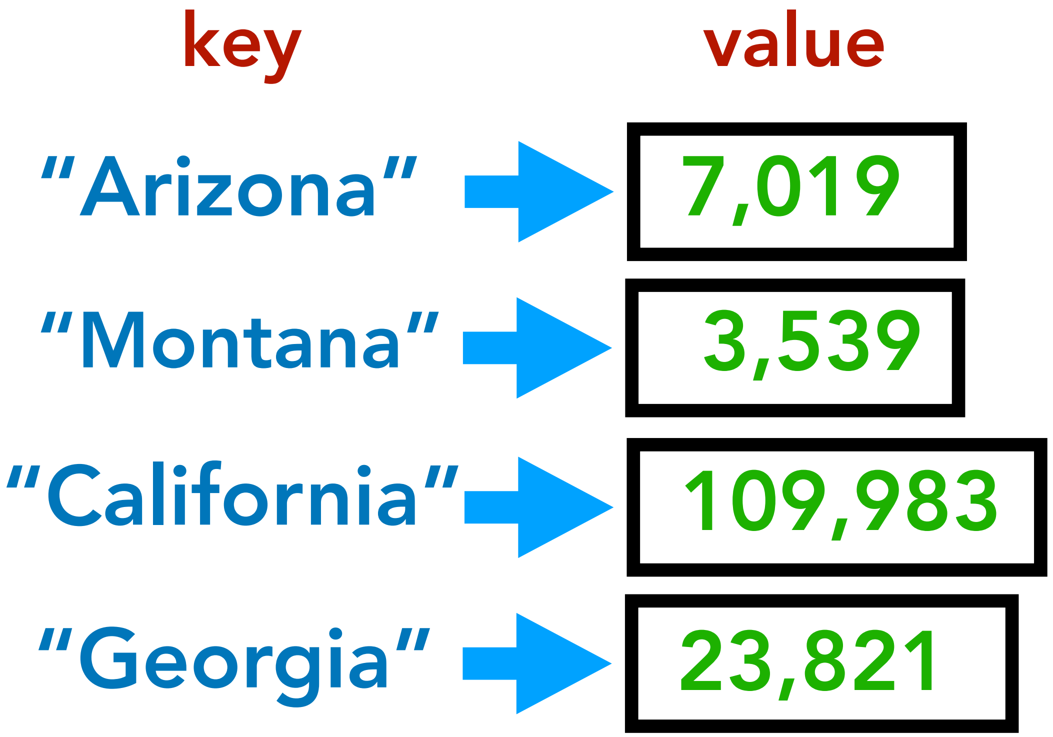
- You usually don't even know how many things you'll need to store (e.g., which US States we will encounter in the file)

- So, how could we possibly create a list for, say, each of the 39 States that happen to be in the file?

Dictionaries

- A **Dictionary** (aka **Hashmap**) is an incredibly powerful data structure

- They have "**keys**", where each **key** maintains its own unique value(s) (i.e., each key may have its own Integer value)

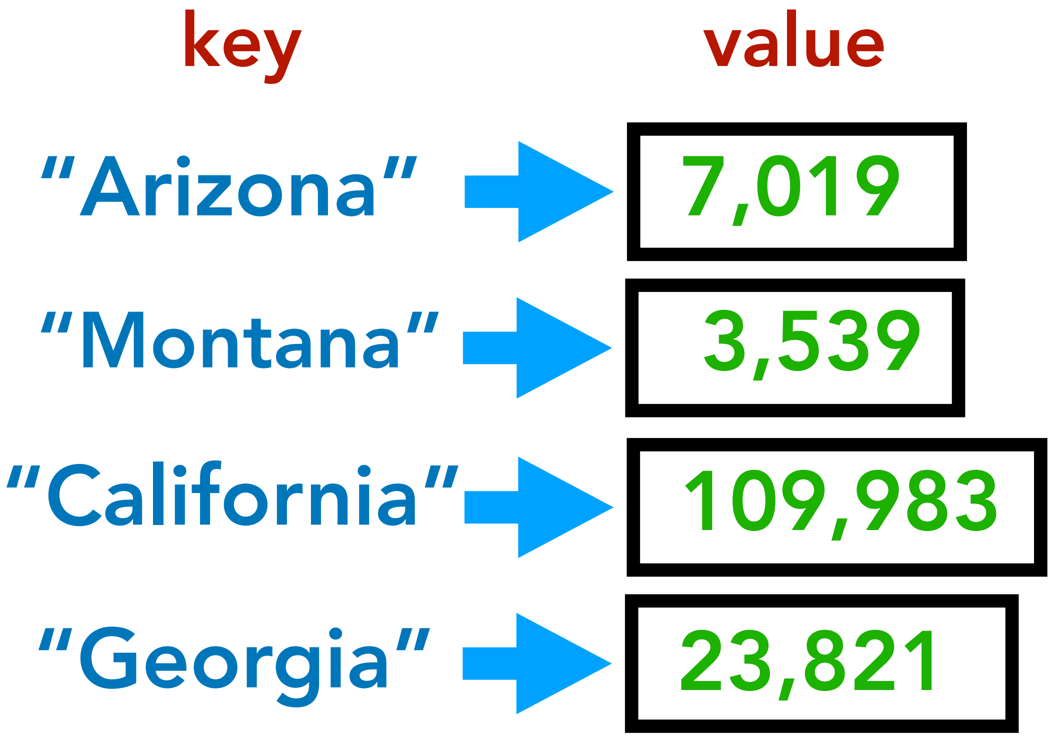


key	value
"Arizona"	7,019
"Montana"	3,539
"California"	109,983
"Georgia"	23,821

Dictionaries

- A **Diction** cture
- They have lue(s) (i.e., each

Metaphor: can think of the **key** as being a mailbox address . Each **key** is unique and can store its own stuff, independent of the other keys.

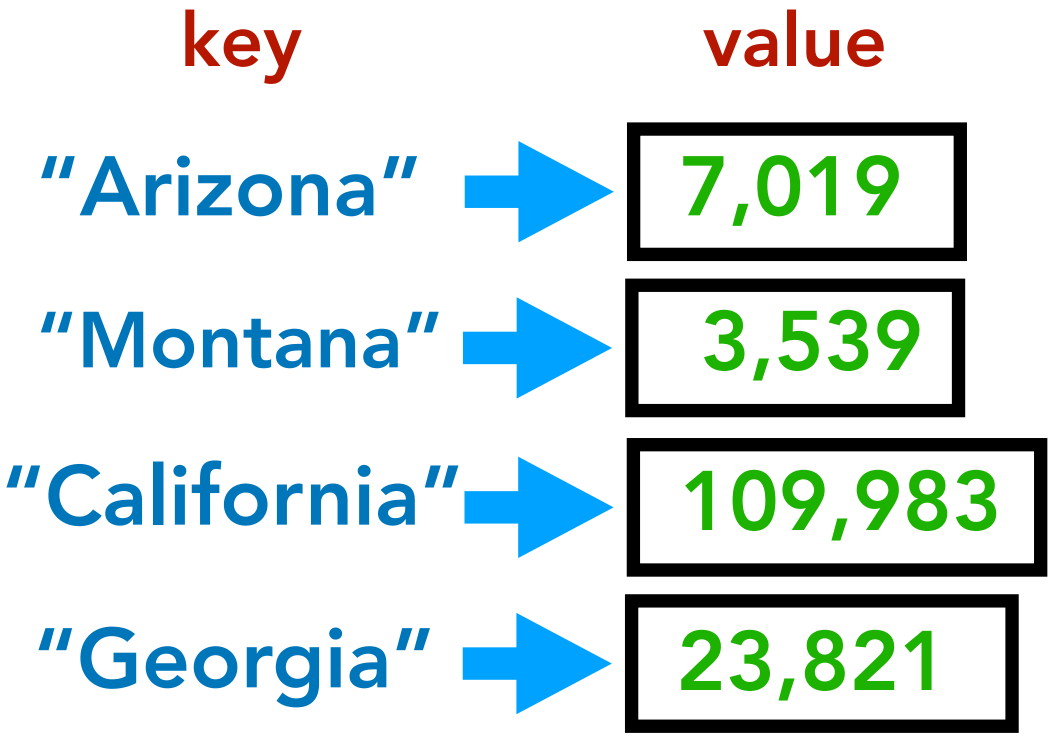


key	value
“Arizona”	7,019
“Montana”	3,539
“California”	109,983
“Georgia”	23,821

Dictionaries

- The **key** should be **single-valued** (i.e., not a list) and can be any primitive data type

- The **value** can be any *data structure* (single-valued, list, set, tuple, *another* dictionary, etc), which can hold any data type you want.

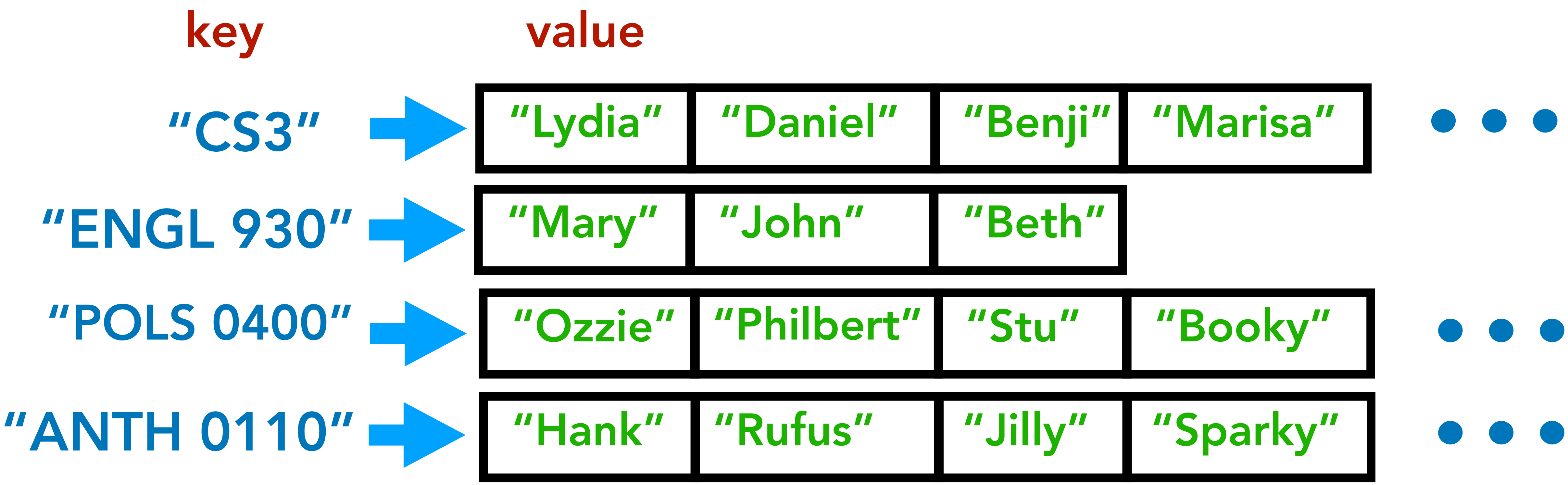


key	value
"Arizona"	7,019
"Montana"	3,539
"California"	109,983
"Georgia"	23,821

Dictionaries

- The **key** should be **single-valued** (i.e., not a list) and can be any primitive data type

- The **value** can be any *data structure* (single-valued, list, set, tuple, *another* dictionary, etc), which can hold any data type you want.



key	value				
"CS3"	"Lydia"	"Daniel"	"Benji"	"Marisa"	• • •
"ENGL 930"	"Mary"	"John"	"Beth"		
"POLS 0400"	"Ozzie"	"Philbert"	"Stu"	"Booky"	• • •
"ANTH 0110"	"Hank"	"Rufus"	"Jilly"	"Sparky"	• • •

Dictionaries

Important: each **key** in a dictionary is unique from the others and is only present at most one time — it's impossible to store the same key multiple times.

Dictionaries

- First, you must always initialize the dictionary (curly brackets!):

```
avocado_sales = {}
```

- Second, we can directly assign values by specifying both the **key** and **value**:

```
avocado_sales[“Georgia”] = 8193
```

or

```
# assume roster_list is a list that we already
created and contains all cs3 students
students[“cs3”] = roster_list
```

Dictionaries

- If we want to either **access** or **update** the value for a certain **key** (e.g., to print it or add to a count), we must first ensure the key even exists!

```
# prints the value (i.e., count)
# for the key ‘Georgia’
if “Georgia” in state_sales:
    print(state_sales[“Georgia”])
```

- Metaphor: the **key** is like a mailbox address. Each mailbox can contain values (mail), but before we check the mail (i.e., access/print values) or deliver new mail (i.e., update it), we need to ensure the mailbox address even exists!

Dictionaries

Example of Updating a key’s value

```
num_sold = int(columns[17])

# update the num sold, if we already have a running total
if “Georgia” in state_sales:
   state_sales[“Georgia”] = state_sales[“Georgia”] + num_sold
else: # or initialize it to
   state_sales[“Georgia”] = num_sold
```

Dictionaries

Iterate through the Dictionary’s Keys

```
# iterate through the keys
for state in state_sales.keys():
   print(state_sales[state]) # prints the value for the key
```

REAL-TIME CODING

Lab Time