Lecture 17:

DICTIONARIES

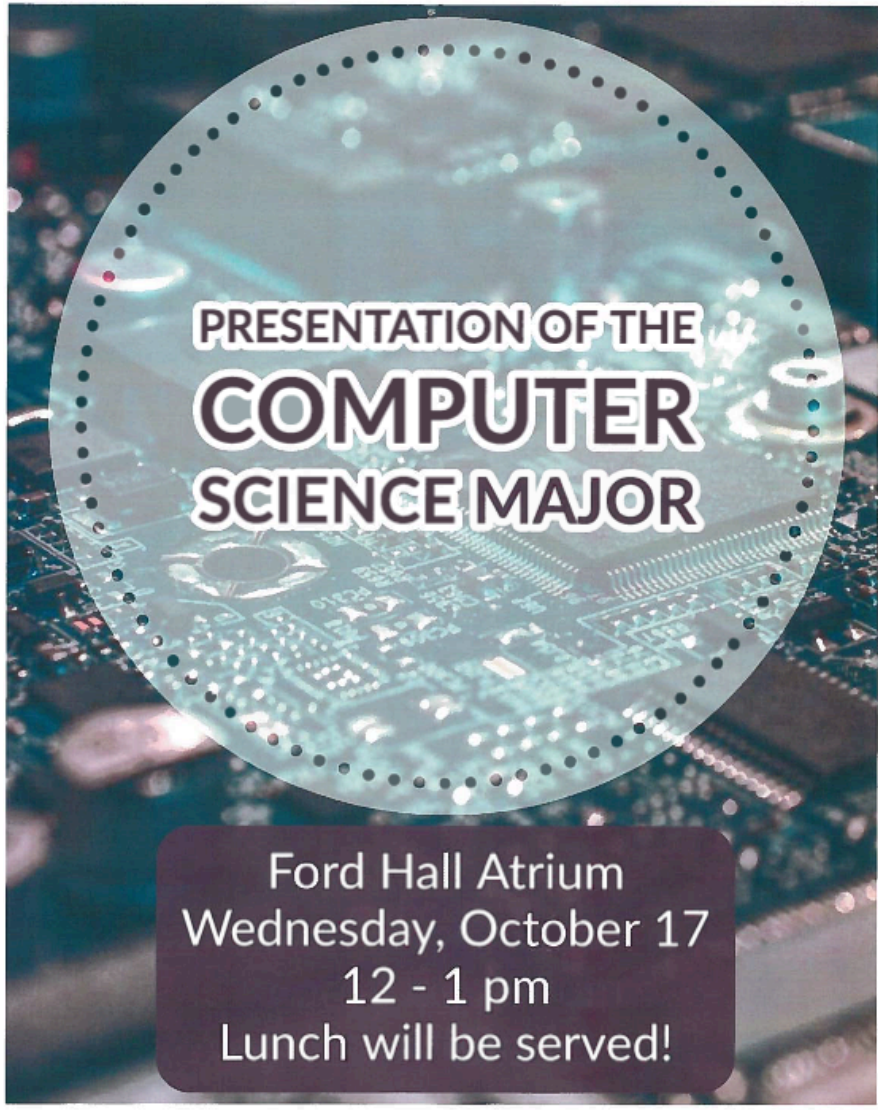
CSC111: Introduction to CS through Programming

R. Jordan Crouser

Assistant Professor of Computer Science

Smith College

Announcements



(we will get out at
11:50 today,
those who wish
to attend can
walk over together!)

Revisiting Assignment 6: music library

- In this assignment, you'll use lists (and other collections) to create a **music library** interface
- Using your interface, the user should be able to:
  - add a song (title, artist, album, link to youtube video, etc.)
  - delete an existing song
  - play a selected song's youtube video
  - print the contents of the library to the screen
- Hint: this week's lab will be **very helpful** in getting started!


DEMO
TIME

Outline

- ✓Recap: strings:
  - ✓indexing
  - ✓slicing
- ✓Lists
  - ✓the basics
  - ✓iterating
  - ✓methods
- ✓A6 Demo
- Dictionaries
  - motivation
  - defining a dictionary
  - converting multiple lists ←→ dictionaries

Recap: 15-minute exercise

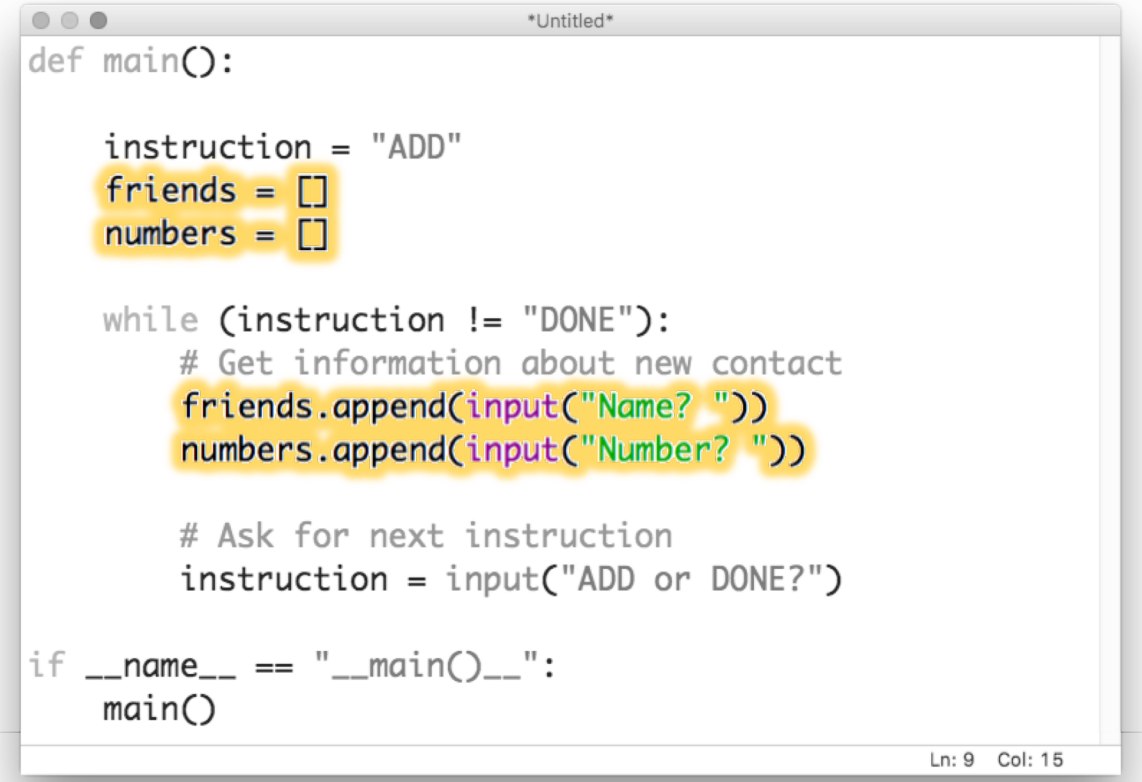
Write a program that:

- asks the user to **`input()`** names one at a time
- adds each new name to a list called **`friends`**
- and after each new name is added prints the list in alphabetical order

The program should loop until the user types "**`DONE`**"

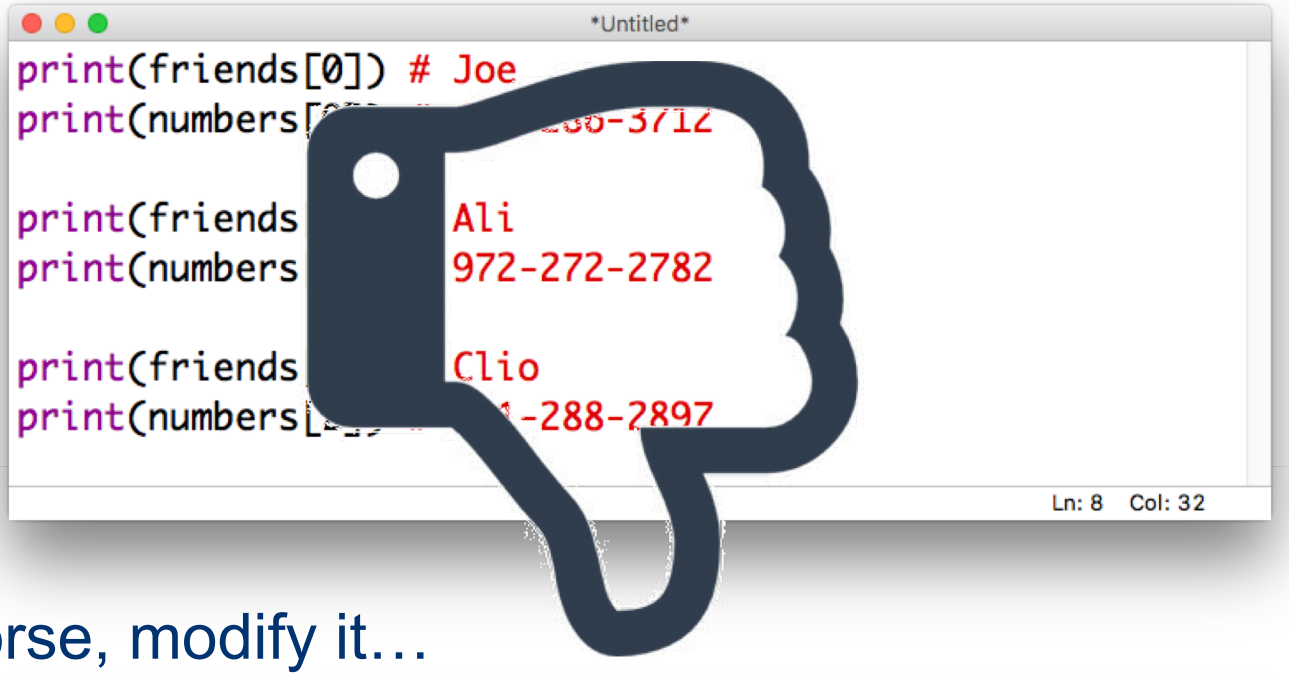
Motivation

- Imagine we want to use the previous exercise to create a contact list. Could do it with **multiple lists**:

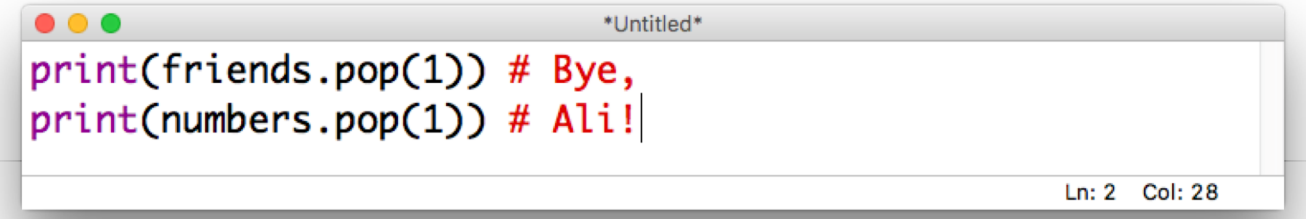


Motivation

- If we want to access the data later:



- Or worse, modify it…



What we really want

- Each name should “map” to the corresponding number:

```
“Joe” → “413-286-3712”
“Ali” → “972-272-2782”
“Clio” → “291-288-2897”
```

- That way, we could access the **`number`** using the **`name`**:

```
contacts[“Joe”] # “413-286-3712”
```

Introducing: **`dictionaries`**

- **`lists`** were **ordered** sets of objects, and we accessed their contents via position (index)
- **`dictionaries`** are **unordered** sets, and we can access their contents via **`keys`**
- We declare them using **`{…}`** ← "curly braces" like this:

```
contacts = {}
```

Contacts: take 2

Untitled

```
def main():

    instruction = "ADD"
    contacts = {}

    while (instruction != "DONE"):
        # Get information about new contact
        new_friend = input("Name? ")
        new_number = input("Number? ")

        # Add contact to dictionary
        contacts[new_friend] = new_number

        # Ask for next instruction
        instruction = input("ADD or DONE?")

if __name__ == "__main()__":
    main()
```

Ln: 18 Col: 10

Contacts: take 2

Untitled

```
def main():

    instruction = "ADD"
    contacts = {}

    while (instruction != "DONE"):
        # Get information about new contact
        new_friend = input("Name? ")
        new_number = input("Number? ")

        # Add contact to dictionary
        contacts[new_friend] = new_number

        # Ask for next instruction
        instruction = input("ADD or DONE?")

if __name__ == "__main()__":
    main()
```

Ln: 18 Col: 10

Contacts: take 2

Untitled

```
def main():

    instruction = "ADD"
    contacts = {}

    while (instruction != "DONE"):
        # Get information about new contact
        new_friend = input("Name? ")
        new_number = input("Number? ")

        # Add contact to dictionary
        contacts[new_friend] = new_number

        # Ask for next instruction
        instruction = input("ADD or DONE?")

if __name__ == "__main()__":
    main()
```

Ln: 18 Col: 10

Contacts: take 2

Untitled

```
def main():

    instruction = "ADD"
    contacts = {}

    while (instruction != "DONE"):
        # Get information about new contact
        new_friend = input("Name? ")
        new_number = input("Number? ")

        # Add contact to dictionary
        contacts[new_friend] = new_number

        # Ask for next instruction
        instruction = input("ADD or DONE?")

if __name__ == "__main()__":
    main()
```

Ln: 18 Col: 10

Interesting dilemma

What happens when we **iterate** over a **dictionary**?

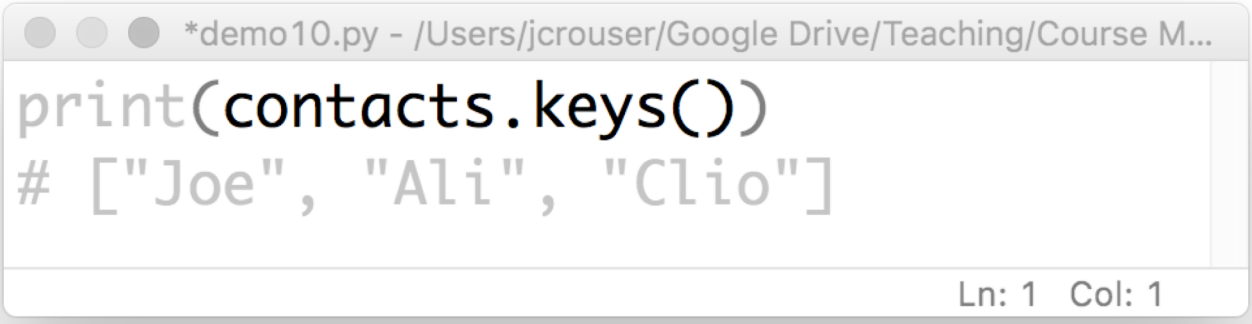
*demo10.py - /Users/jcrouser/G...

```
for c in contacts:
    print(c)
```

Ln: 1 Col: 6

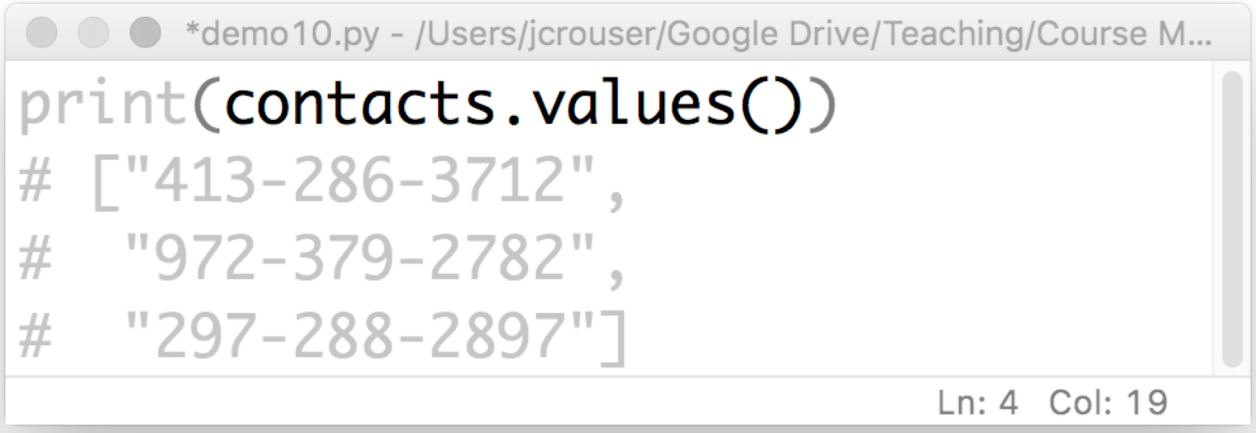
`dictionary` methods: `.keys()`

- If you want to get a list of the **`keys`** in a **`dictionary`**



`dictionary` methods: **`.values()`**

- If you want a **`list`** of the **`values`** in a **`dictionary`**

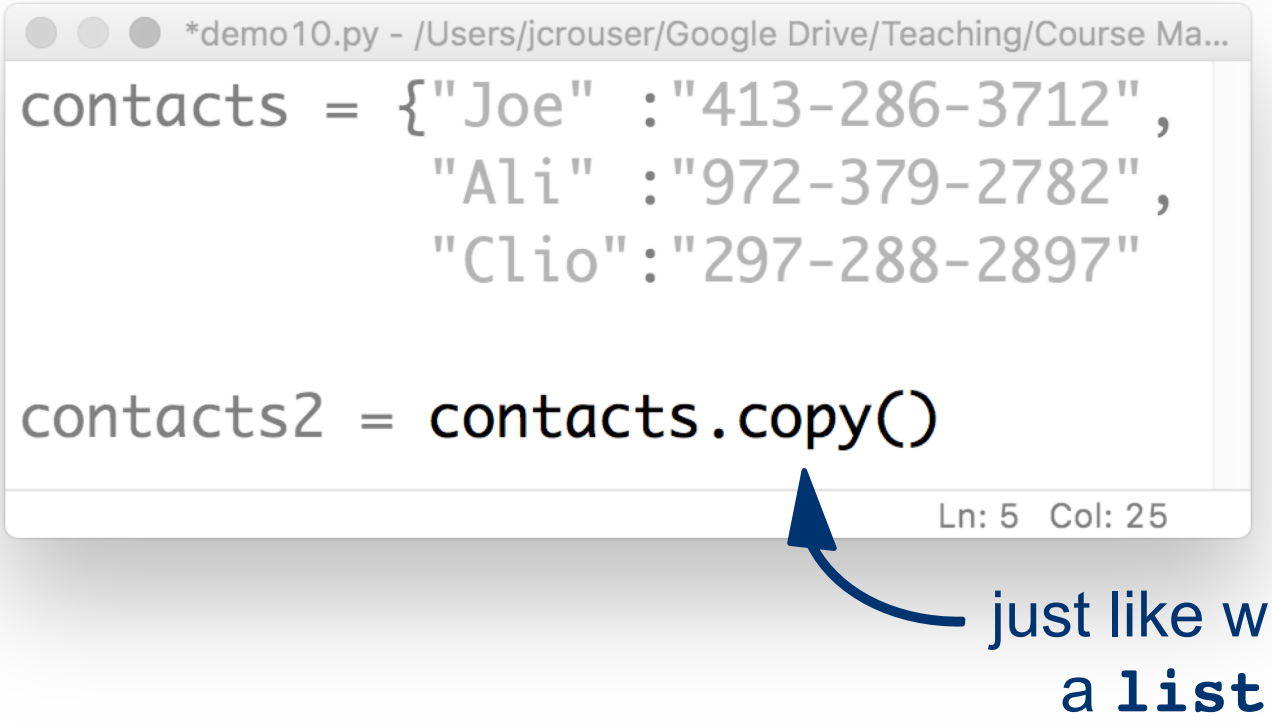


`dictionary` methods: **`.items()`**

- If you want a **`list`** of the **`key,value`** pairs in a **`dictionary`**

`dictionary` methods: **`.copy()`**

- If you want to **copy** the **`dictionary`** :



The `zip(…)` function

- If you want to combine two **`lists`** into one **`dictionary`**, use a comprehension and the **`zip(…)`** function:

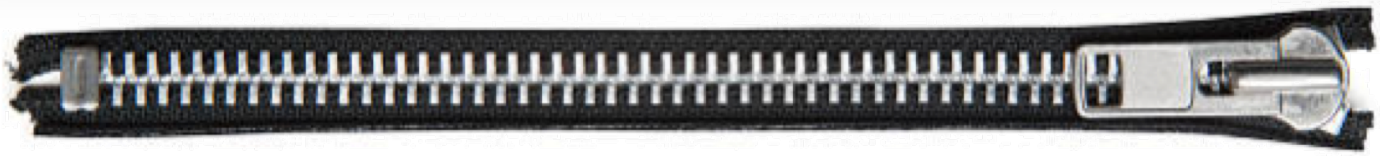
Untitled

```
names = ["Joe", "Ali", "Clio"]

numbers = ["413-286-3712",
           "972-272-2782",
           "291-288-2897"]

contacts = {name:number for name, number in zip(names, numbers)}
```

Ln: 7 Col: 35



Recap

- **`strings`**: immutable ordered collections of characters
- **`lists`**: mutable ordered collections of objects
- **`dictionaries`**: mutable unordered collections of objects

Passing “by reference”

What does this mean when we pass

a list / dictionary as **input to a function**?

Coming up

- ✓Recap: strings:
- ✓Lists
- ✓Dictionaries
- Life Skill #4: Responsible Code Reuse