

Lecture 17:

DICTIONARIES

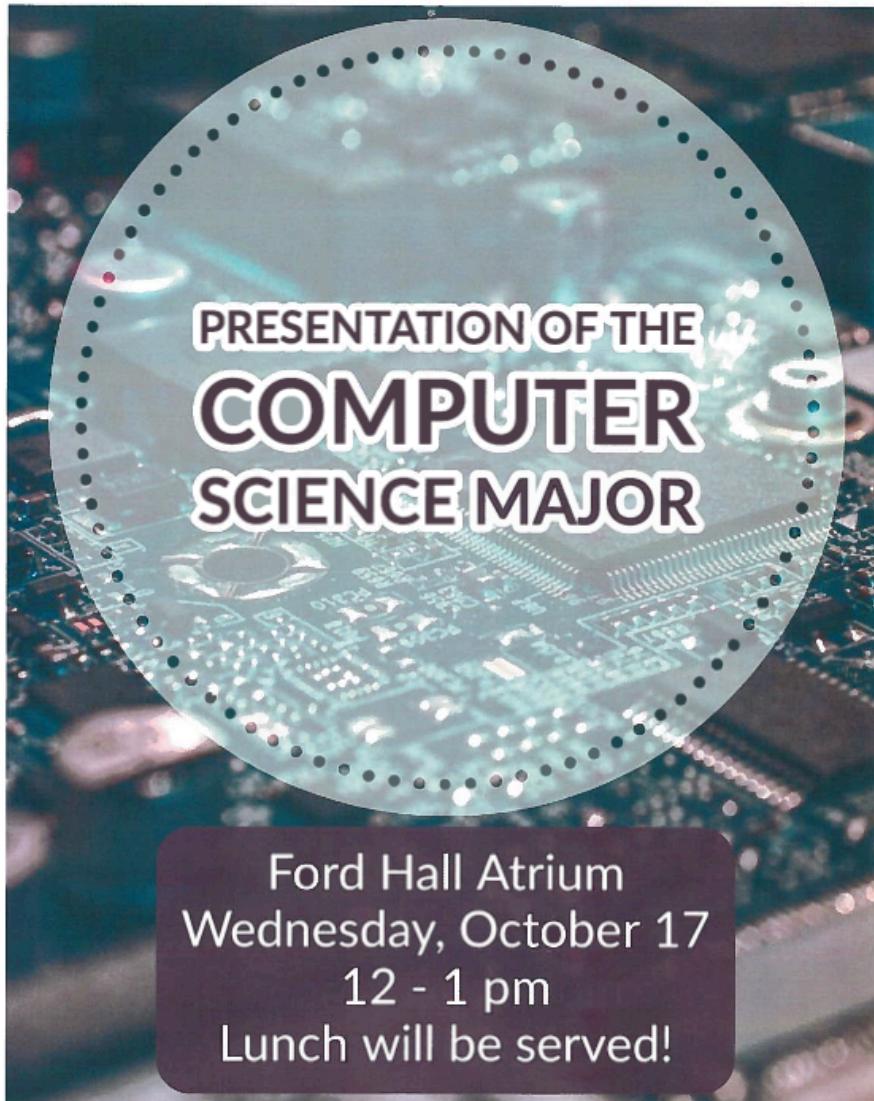
CSC111: Introduction to CS through Programming

R. Jordan Crouser

Assistant Professor of Computer Science

Smith College

Announcements



Ford Hall Atrium
Wednesday, October 17
12 - 1 pm
Lunch will be served!

(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!

D E M O

T Y M E

Outline

✓ Recap: strings:

- ✓ indexing
- ✓ slicing

✓ Lists

- ✓ the basics
- ✓ iterating
- ✓ methods

✓ A6 Demo

- Dictionaries

- motivation
- defining a dictionary
- converting multiple lists \leftrightarrow dictionaries

Recap: 15-minute exercise

Write a program that:

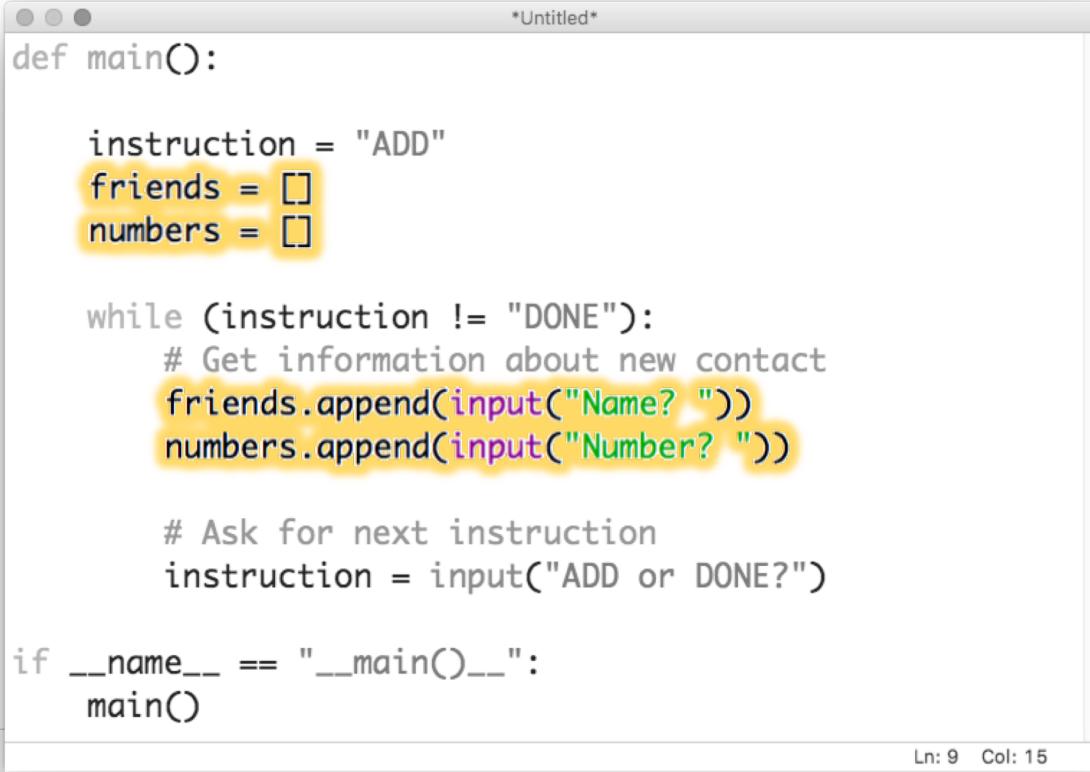
- asks the user to `input()` names one at a time
- adds each new name to a list called `friends`
- and after each new name is added prints the list in alphabetical order

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**:



```
*Untitled*
def main():

    instruction = "ADD"
    friends = []
    numbers = []

    while (instruction != "DONE"):
        # Get information about new contact
        friends.append(input("Name? "))
        numbers.append(input("Number? "))

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

if __name__ == "__main__":
    main()

Ln: 9 Col: 15
```

Motivation

- If we want to access the data later:



```
*Untitled*
```

```
print(friends[0]) # Joe
print(numbers[0]) # 200-5712

print(friends)
print(numbers)

print(friends[1])
print(numbers[1]) # 972-272-2782

print(friends[2])
print(numbers[2]) # 1-288-2897
```

Ln: 8 Col: 32

- Or worse, modify it...

```
*Untitled*
```

```
print(friends.pop(1)) # Bye,
print(numbers.pop(1)) # Ali!
```

Ln: 2 Col: 28

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

The image shows a screenshot of a Python code editor window titled '*Untitled*'. The code is a script for managing contacts. It starts with a main function that initializes an instruction variable to 'ADD' and an empty contacts dictionary. A while loop continues until the instruction is 'DONE'. Inside the loop, it prompts for a new friend's name and number, adds them to the contacts dictionary, and then asks for the next instruction. Finally, if the script is run directly, it calls the main function.

```
def main():

    instruction = "ADD"
    contacts = {}

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

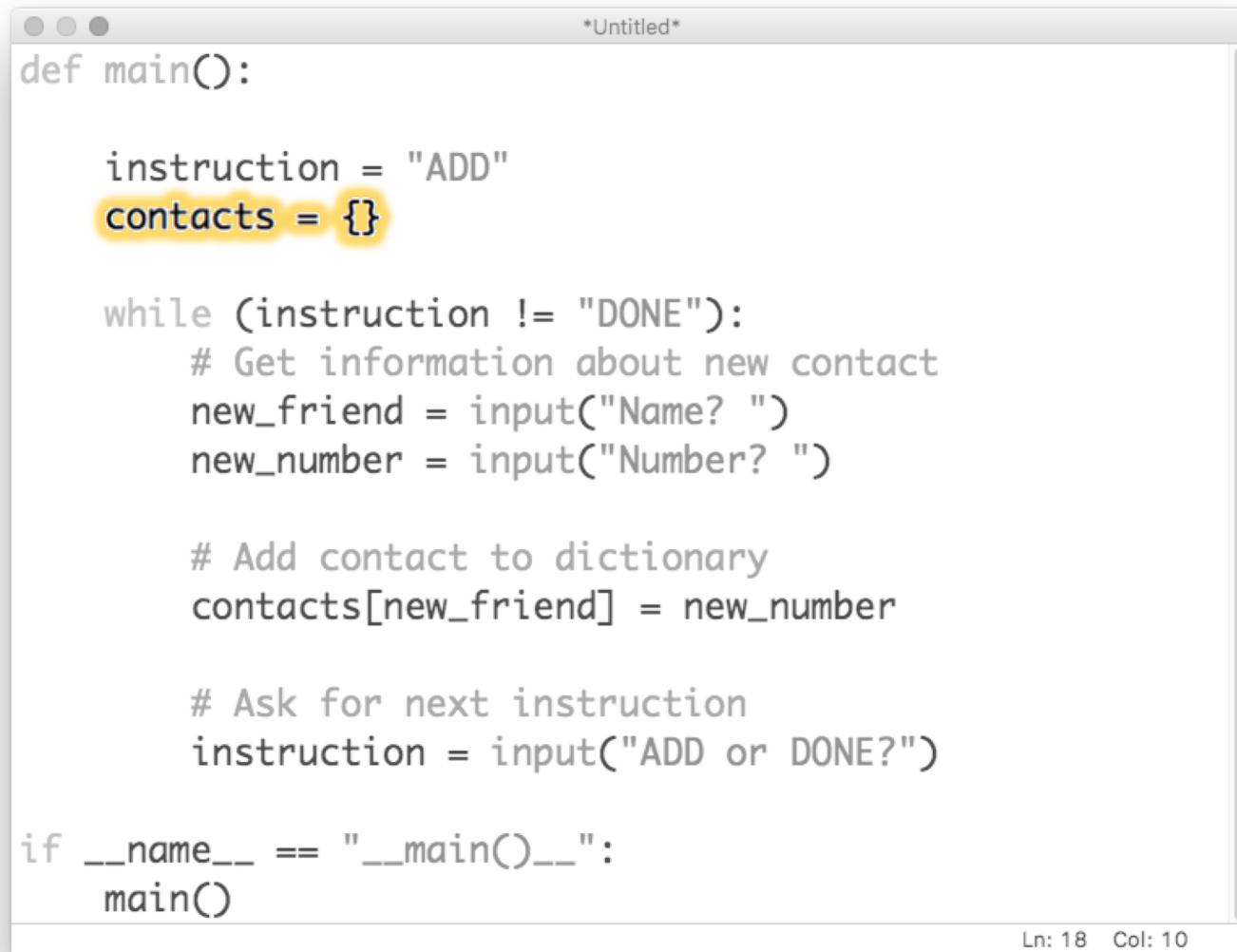
        # Add contact to dictionary
        contacts[new_friend] = new_number

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

if __name__ == "__main__":
    main()
```

Ln: 18 Col: 10

Contacts: take 2



The image shows a screenshot of a code editor window titled "Untitled". The code is written in Python and defines a function named main(). This function initializes a variable instruction to "ADD" and creates an empty dictionary contacts. It then enters a loop where it repeatedly asks for a name and number, adds them to the contacts dictionary, and then asks for the next instruction. Finally, if the script is run directly (as opposed to imported), it calls the main() function.

```
*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**?



A screenshot of a terminal window titled '*demo10.py - /Users/jcrouser/G...'. The code inside the window is:

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

Below the code, the status bar shows 'Ln: 1 Col: 6'.



dictionary methods: .keys()

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



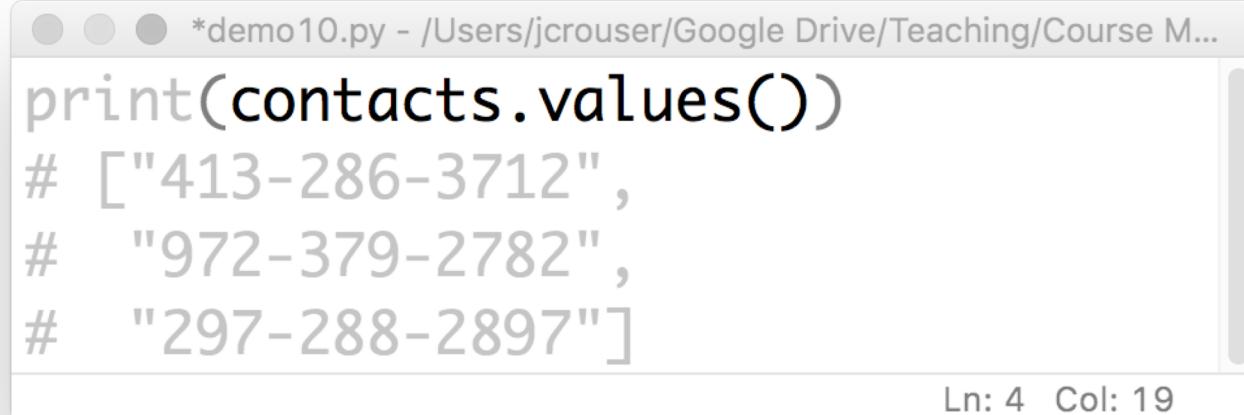
A screenshot of a terminal window titled "*demo10.py - /Users/jcrouser/Google Drive/Teaching/Course M...". The window contains the following Python code:

```
print(contacts.keys())
# ["Joe", "Ali", "Clio"]
```

The code prints the keys of a dictionary named contacts, resulting in a list: ["Joe", "Ali", "Clio"]. The status bar at the bottom right of the terminal window shows "Ln: 1 Col: 1".

dictionary methods: .values()

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



A screenshot of a terminal window titled "*demo10.py - /Users/jcrouser/Google Drive/Teaching/Course M...". The window contains the following Python code:

```
print(contacts.values())
# ["413-286-3712",
# "972-379-2782",
# "297-288-2897"]
```

The code prints a list of phone numbers from a dictionary named contacts. The terminal status bar at the bottom right shows "Ln: 4 Col: 19".

dictionary methods: .items()

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

The image shows a terminal window with three gray circular icons in the top-left corner. The title bar reads "*demo10.py - /Users/jcrouser/Google Drive/Teaching/Course Mater...". The main area contains the following Python code:

```
for key, value in contacts.items():
    print(key, value)
```

In the bottom-right corner of the terminal window, there is a status bar with the text "Ln: 2 Col: 21".

dictionary methods: .copy()

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

```
*demo10.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...
contacts = {"Joe": "413-286-3712",
             "Ali": "972-379-2782",
             "Clio": "297-288-2897"

contacts2 = contacts.copy()
```



Ln: 5 Col: 25

just like with
a **list**

The **zip**(...) function

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

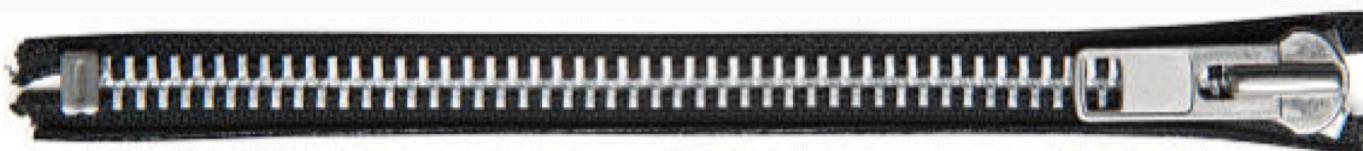
```
*Untitled*
```

```
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