Dictionaries

Chris Piech and Mehran Sahami

And Then There Were None

- The term **`None`** is used in Python to describe "no value"
  - For example, it is the value you would get from a function that doesn't return anything
  - WHAT?!
  - Example:

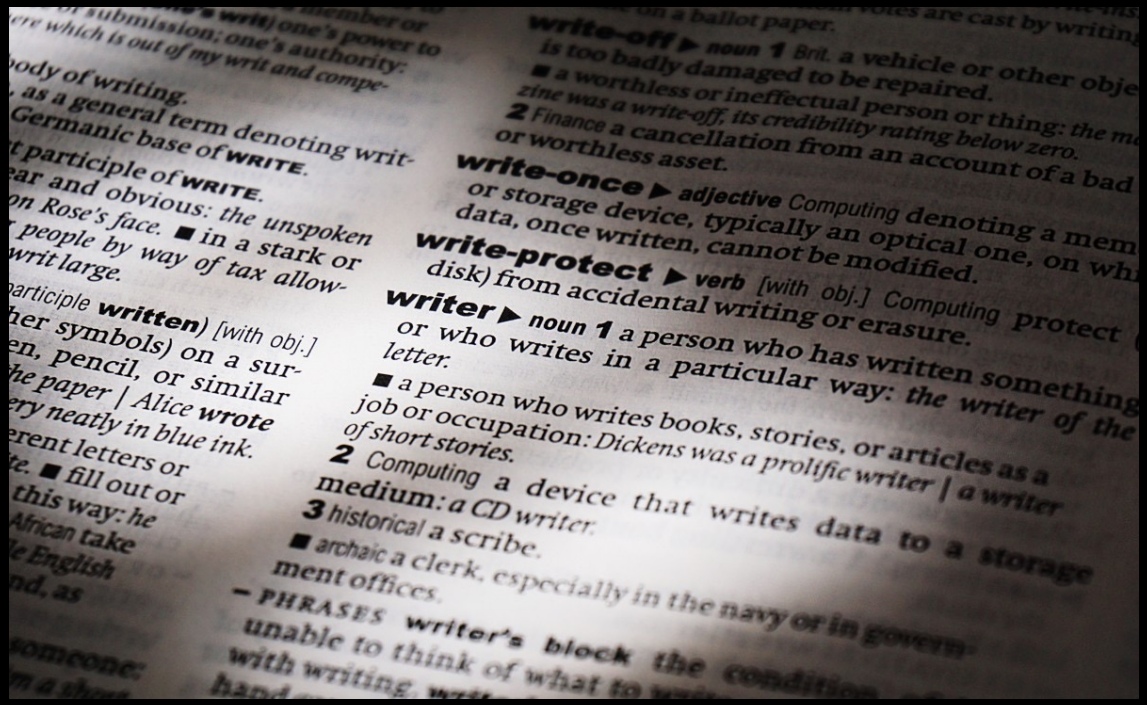
```
>>> x = print("hi")
>>> print(x)
None
```

  - Comparing anything to **`None`** (except **`None`**) is False
- Why does **`None`** exist?
  - Denotes when the suitcase for a variable has "nothing" in it

Learning Goals

1. Learning about dictionaries
2. Building programs using dictionaries

Dictionaries



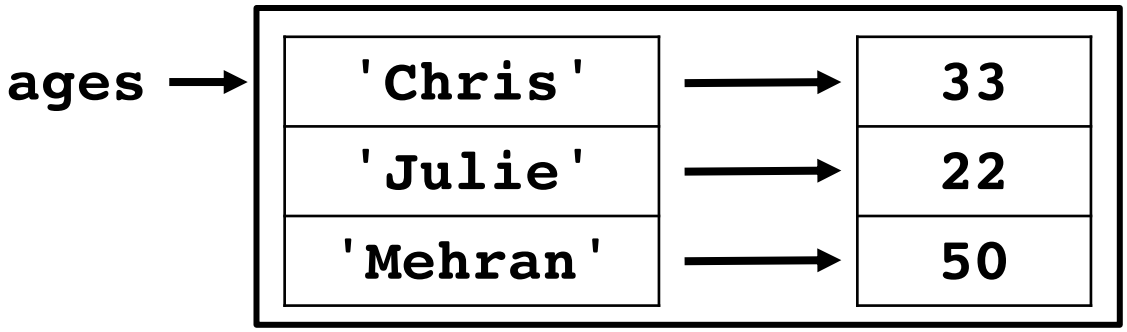
What are Dictionaries?

- Dictionaries associate a ***key*** with a ***value***
  - Key is a *unique* identifier
  - Value is something we associate with that key
- Examples in the real world:
  - Phonebook
    - Keys: names
    - Values: phone numbers
  - Dictionary
    - Keys: words
    - Values: word definitions
  - US Government
    - Keys: Social Security number
    - Values: Information about an individual's employment in US

Dictionaries in Python

- Creating dictionaries
  - Dictionary start/end with braces
  - Key:Value pairs separated by colon
  - Each pair is separated by a comma

```
ages = {'Chris': 33, 'Julie': 22, 'Mehran': 50}
squares = {2: 4, 3: 9, 4: 16, 5: 25}
phone = {'Pat': '555-1212', 'Jenny': '867-5309'}
empty_dict = {}
```

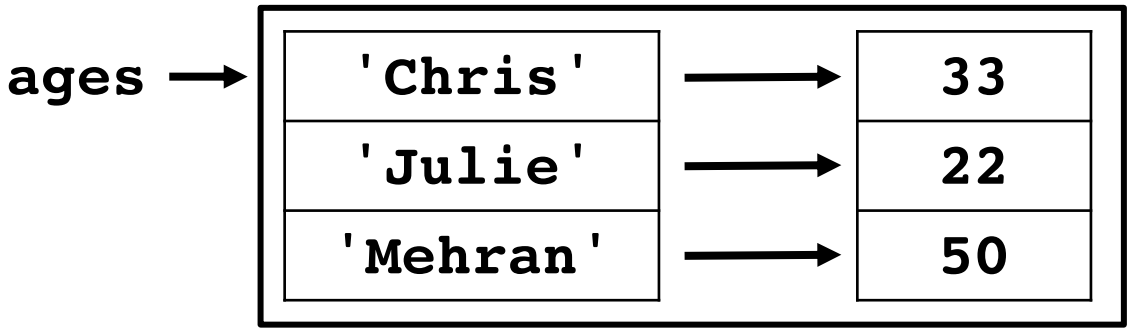


Accessing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Chris': 33, 'Julie': 22, 'Mehran': 50}
```

- Like a set of variables that are indexed by <u>keys</u>



- Use <u>key</u> to access associated <u>value</u>:

`ages['Chris']` is `33`

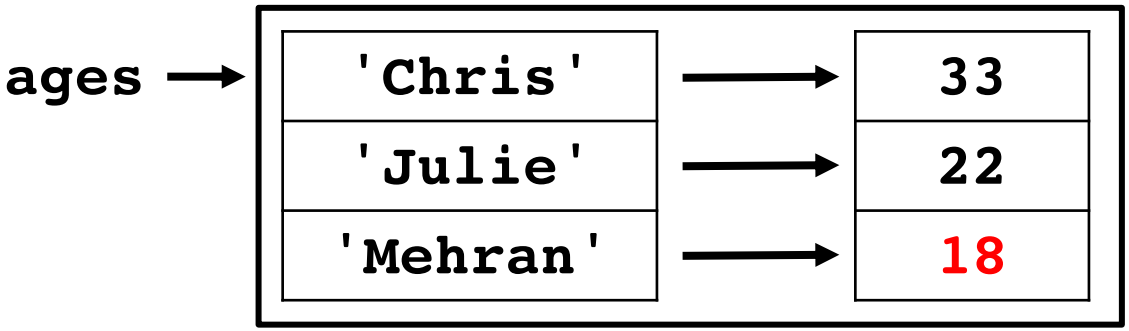
`ages['Mehran']` is `50`

Accessing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Chris': 33, 'Julie': 22, 'Mehran': 50}
```

- Like a set of variables that are indexed by <u>keys</u>



- Use <u>key</u> to access associated <u>value</u>:

`ages['Chris']` is `33`

`ages['Mehran']` is `50`

- Can set <u>values</u> like regular variable:

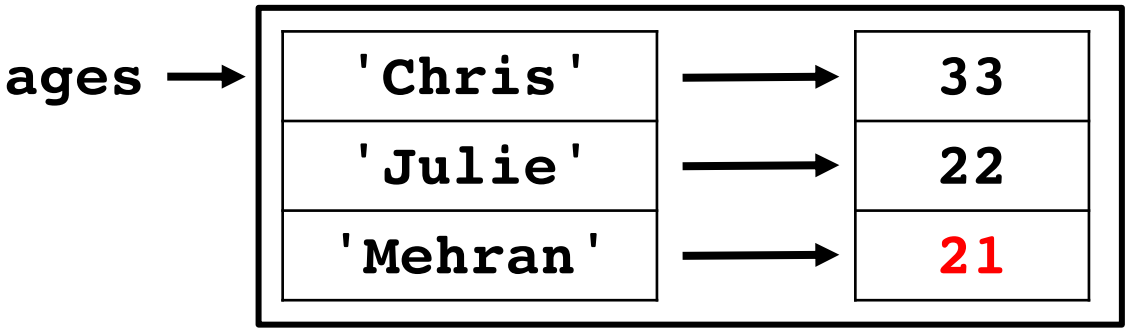
```
ages['Mehran'] = 18
```

Accessing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Chris': 33, 'Julie': 22, 'Mehran': 50}
```

- Like a set of variables that are indexed by <u>keys</u>



- Use <u>key</u> to access associated <u>value</u>:

`ages['Chris']` is `33`

`ages['Mehran']` is `50`

- Can set <u>values</u> like regular variable:

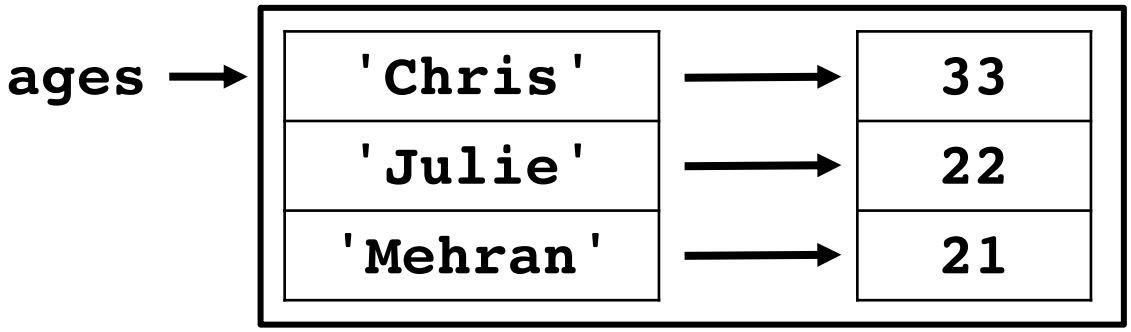
```
ages['Mehran'] = 18
ages['Mehran'] += 3
```

Accessing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Chris': 33, 'Julie': 22, 'Mehran': 50}
```

- Like a set of variables that are indexed by <u>keys</u>



- Good and bad times with accessing pairs:

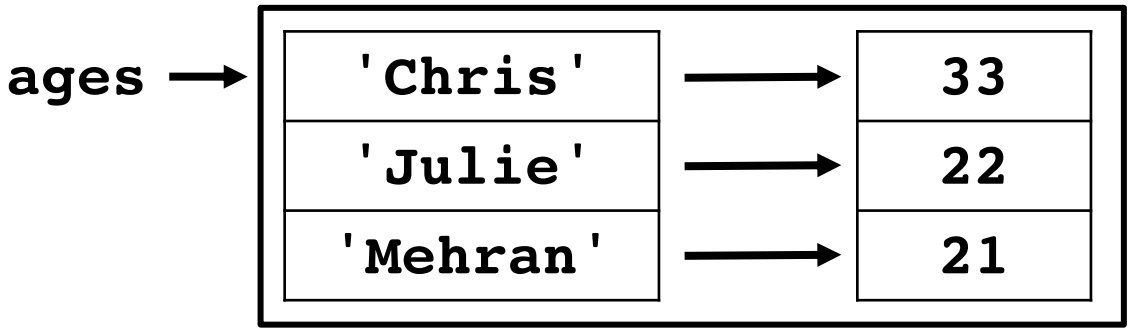
```
>>> julies_age = ages['Julie']
>>> julies_age
22
>>> santas_age = ages['Santa Claus']
KeyError: 'Santa Claus'
```

Accessing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Chris': 33, 'Julie': 22, 'Mehran': 50}
```

- Like a set of variables that are indexed by <u>keys</u>



- Checking membership

```
>>> 'Julie' in ages
True
>>> 'Santa Claus' not in ages
True
```

Adding Elements to Dictionary

- Can add pairs to a dictionary:

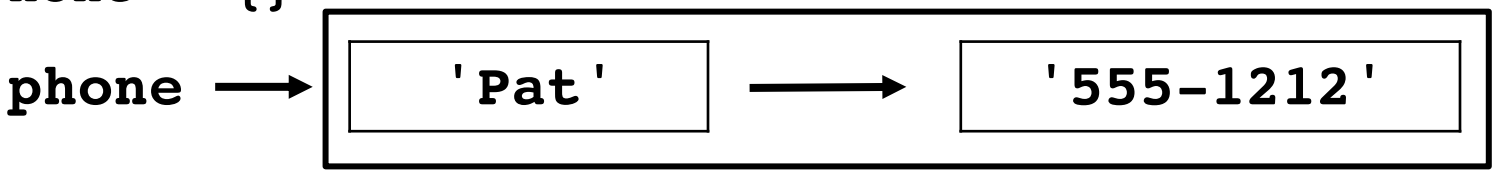
```
phone = {}
```

phone ⟶ *Empty dictionary*

Adding Elements to Dictionary

- Can add pairs to a dictionary:

```
phone = {}
```

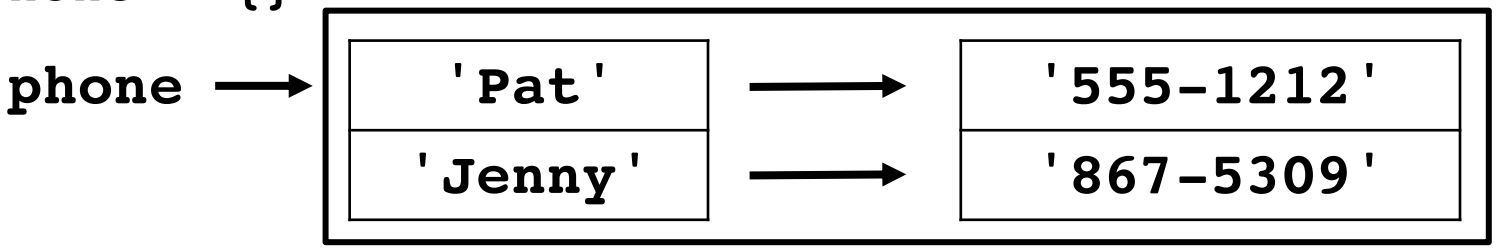



```
phone['Pat'] = '555-1212'
```

Adding Elements to Dictionary

- Can add pairs to a dictionary:

```
phone = {}
```

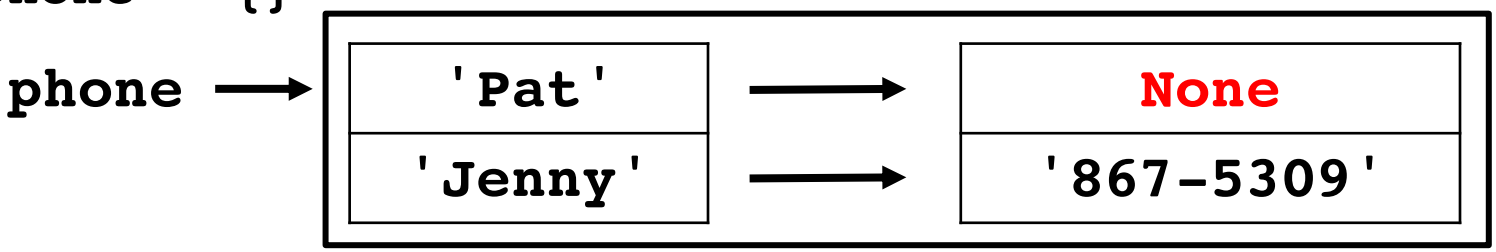



```
phone['Pat'] = '555-1212'
phone['Jenny'] = '867-5309'
```

Adding Elements to Dictionary

- Can add pairs to a dictionary:

```
phone = {}
```

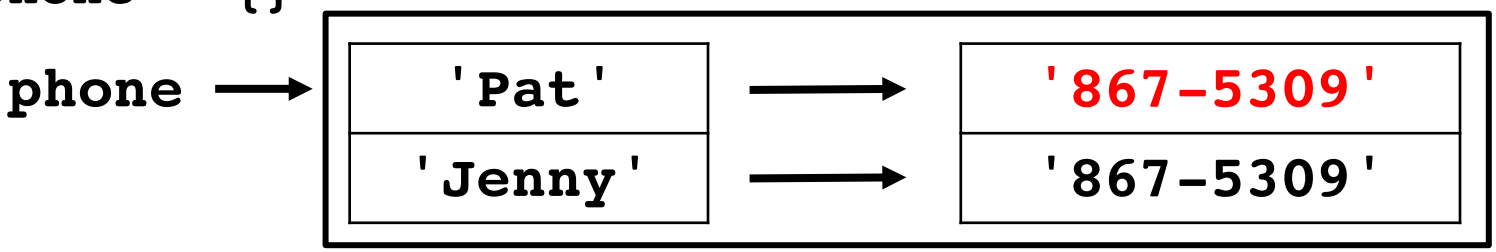



```
phone['Pat'] = '555-1212'
phone['Jenny'] = '867-5309'
phone['Pat'] = None
```

Adding Elements to Dictionary

- Can add pairs to a dictionary:

```
phone = {}
```




```
phone['Pat'] = '555-1212'
phone['Jenny'] = '867-5309'
phone['Pat'] = None
phone['Pat'] = '867-5309'
```

A Word About Keys/Values

- Keys must be immutable types
  - E.g., int, float, string
  - Keys cannot be changed in place
  - If you want to change a key, need to remove key/value pair from dictionary and then add key/value pair with new key.
- Values can be mutable or immutable types
  - E.g., int, float, string, lists, dictionaries
  - Values can be changed in place
- Dictionaries are mutable
  - Changes made to a dictionary in a function persist after the function is done.

Changing Dictionary in a Function

```
def have_birthday(dict, name):
    print("You're one year older, " + name + "!")
    dict[name] += 1

def main():
    ages = {'Chris': 33, 'Julie': 22, 'Mehran': 50}
    print(ages)
    have_birthday(ages, 'Chris')
    print(ages)
    have_birthday(ages, 'Mehran')
    print(ages)
```

Terminal:

```
{'Chris': 33, 'Julie': 22, 'Mehran': 50}
You're one year older, Chris!
{'Chris’: 34, 'Julie': 22, 'Mehran': 50}
You're one year older, Mehran!
{'Chris’: 34, 'Julie': 22, 'Mehran': 51}
```

Dictiona-palooza! (Part 1)

```
ages = {'Chris': 33, 'Julie': 22, 'Mehran': 50}
```

- Function: ***dict*.get(key)**
  - Returns value associated with key in dictionary. Returns **None** if key doesn't exist.

```
>>> print(ages.get('Chris'))
33
>>> print(ages.get('Santa Claus'))
None
```

- Function: ***dict*.get(key, default)**
  - Returns value associated with key in dictionary. Returns default if key doesn't exist.

```
>>> print(ages.get('Chris', 100))
33
>>> print(ages.get('Santa Claus', 100))
100
```

Dictiona-palooza! (Part 2)

```
ages = {'Chris': 33, 'Julie': 22, 'Mehran': 50}
```

- Function: ***dict*.keys()**
  - Returns something similar to a range of the keys in dictionary
  - Can use that to loop over all keys in a dictionary:

```
for key in ages.keys():
    print(str(key) + " -> " + str(ages[key]))
```

Terminal:

```
Chris -> 33
Julie -> 22
Mehran -> 50
```

  - Can turn **keys()** into a list, using the **list** function

```
>>> list(ages.keys())
['Chris', 'Julie', 'Mehran']
```

Dictiona-palooza! (Part 3)

```
ages = {'Chris': 33, 'Julie': 22, 'Mehran': 50}
```

- Can also loop over a dictionary using for-each loop just using name of dictionary:

```
for key in ages:
    print(str(key) + " -> " + str(ages[key]))
```

Terminal:

```
Chris -> 33
Julie -> 22
Mehran -> 50
```

Dictiona-palooza! (Part 4)

```
ages = {'Chris': 33, 'Julie': 22, 'Mehran': 50}
```

- Function: ***dict*.values()**
  - Returns something similar to a range of the values in dictionary
  - Can use that to loop over all keys in a dictionary:

```
for value in ages.values():
    print(value)
```

Terminal:

```
33
22
50
```

  - Can turn **values()** into a list, using the **list** function

```
>>> list(ages.values())
[33, 22, 50]
```

Dictiona-palooza! (Part 5)

```
ages = {'Chris': 33, 'Julie': 22, 'Mehran': 50}
```

- Function: ***dict*.pop(key)**
  - Removes key/value pair with the given key. Returns value from that key/value pair.

```
>>> ages
>>> {'Chris': 33, 'Julie': 22, 'Mehran': 50}
>>> ages.pop('Mehran')
50
>>> ages
{'Chris': 33, 'Julie': 22}
```

- Function: ***dict*.clear()**
  - Removes all key/value pairs in the dictionary.

```
>>> ages.clear()
>>> ages
{}
```

Functions You Can Apply

```
ages = {'Chris': 33, 'Julie': 22, 'Mehran': 50}
```

- Function: **len(*dict*)**
  - Returns number of key/value pairs in the dictionary

```
>>> ages
{'Chris': 33, 'Julie': 22, 'Mehran': 50}
>>> len(ages)
3
```

- Function: **del *dict*[key]**
  - Removes key/value pairs in the dictionary.
  - Similar to **pop**, but doesn't return anything.

```
>>> ages
{'Chris': 33, 'Julie': 22, 'Mehran': 50}
>>> del ages['Mehran']
>>> ages
{'Chris': 33, 'Julie': 22}
```

Putting it all together: count_characters.py

Bonus fun:
phonebook.py

Learning Goals

1. Learning about dictionaries
2. Building programs using dictionaries

{'breakfast':  ,
'lunch':  ,
'dinner':  }