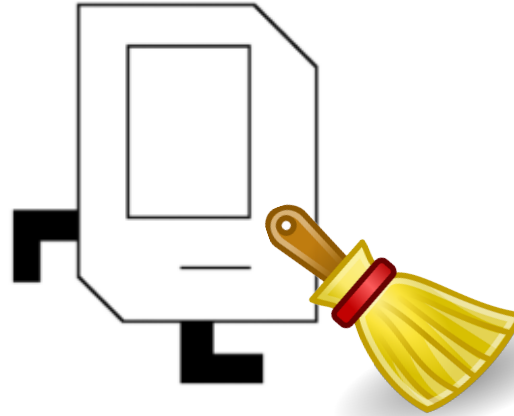


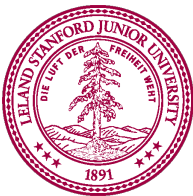
Dictionaries

Chris Piech and Mehran Sahami
CS106A, Stanford University

Housekeeping



- Mehran holding Ask Me Anything (AMA) on Zoom
 - May 14 at 4pm-5pm (PDT)
 - Should have gotten email yesterday with Zoom information



Review of String

```
PUNCTUATION = ' .!?,-:; '
```

```
def delete_punctuation(s):
```

```
    """
```

```
    Removes punctuation characters from a string and
    returns the resulting string.
```

```
    """
```

```
    result = ''
```

```
    for char in s:
```

```
        # Check char is not a punctuation mark
```

```
        if char not in PUNCTUATION:
```

```
            result += char    # append non-punctuation chars
```

```
    return result
```

Console:

```
>>> delete_punctuation('REMOVE --the-- punctuation!!!!')
'REMOVE the punctuation'
```

Reading Lines from a File

```
def count_words(filename):  
    count = 0  
    with open(filename, 'r') as file: # Open file to read  
        for line in file:  
            line = line.strip() # Remove newline  
            word_list = line.split() # Create list of words  
            for word in word_list: # Print words  
                print("#" + str(count) + ": " + word)  
                count += 1  
    print(filename + " contains " + str(count) + " words")
```

testfile.txt

```
Very few  
words here.
```

Console:

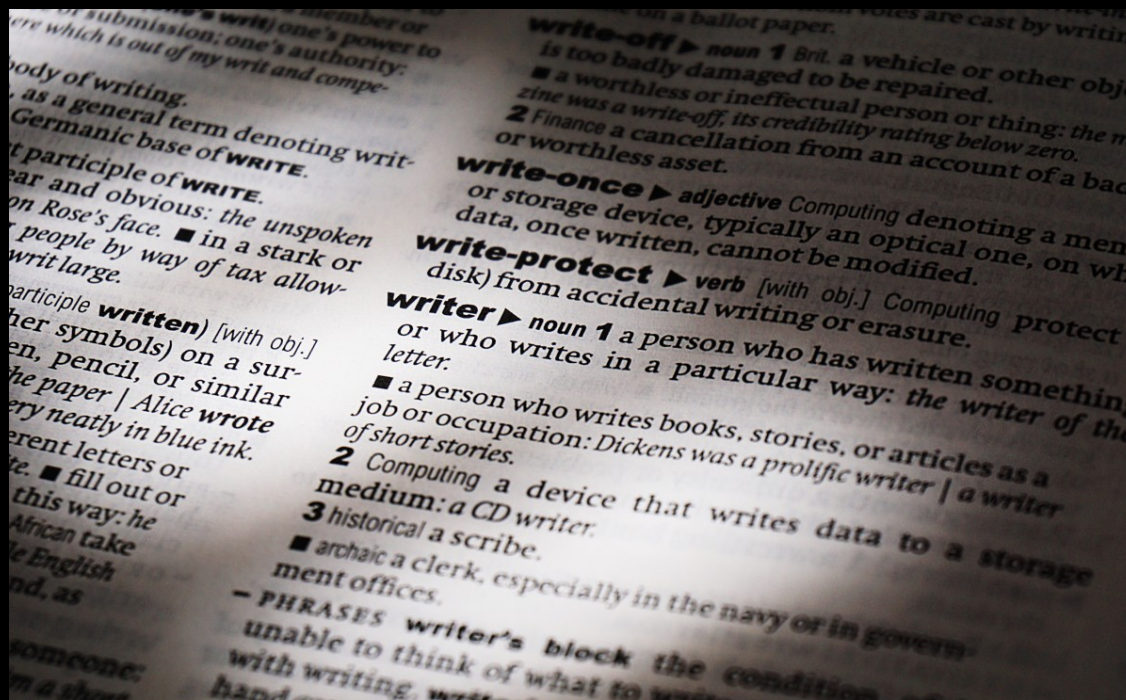
```
#0: Very  
#1: few  
#2: words  
#3: here.  
testfile.txt contains 4 words
```

Learning Goals

1. Learning about dictionaries
2. Building programs using dictionaries

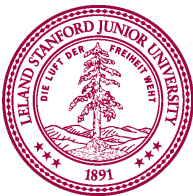


Dictionaries



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



Dictionaries in Python

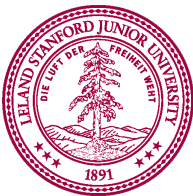
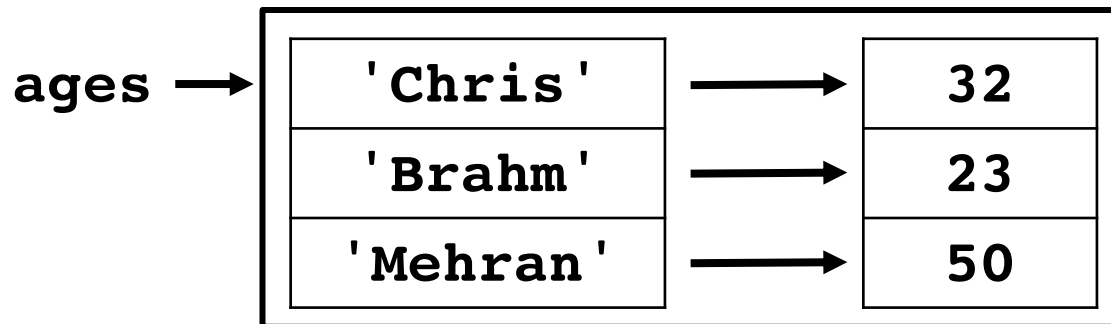
- Creating dictionaries
 - Dictionary start/end with braces
 - Key:Value pairs separated by colon
 - Each pair is separated by a comma

```
ages = {'Chris': 32, 'Brahm': 23, 'Mehrnan': 50}
```

```
squares = {2: 4, 3: 9, 4: 16, 5: 25}
```

```
phone = {'Pat': '555-1212', 'Jenny': '867-5309'}
```

```
empty_dict = {}
```

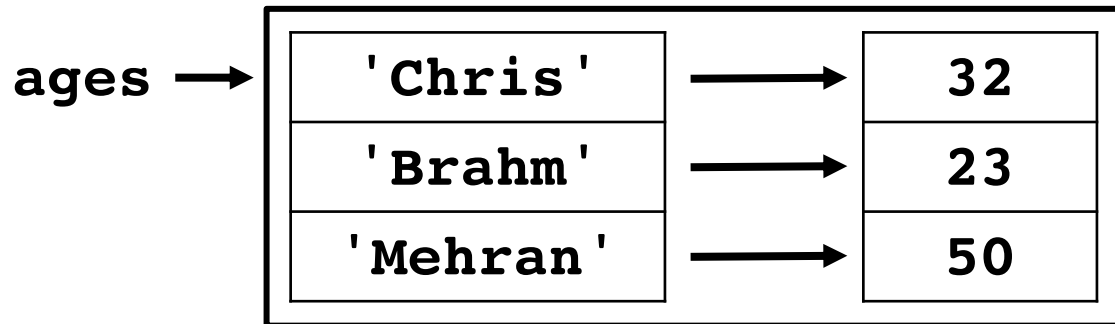


Accessing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Chris': 32, 'Brahm': 23, 'Mehrnan': 50}
```

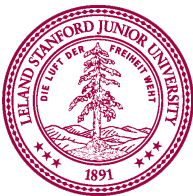
- Like a set of variables that are indexed by keys



- Use key to access associated value:

```
ages['Chris'] is 32
```

```
ages['Mehrnan'] is 50
```

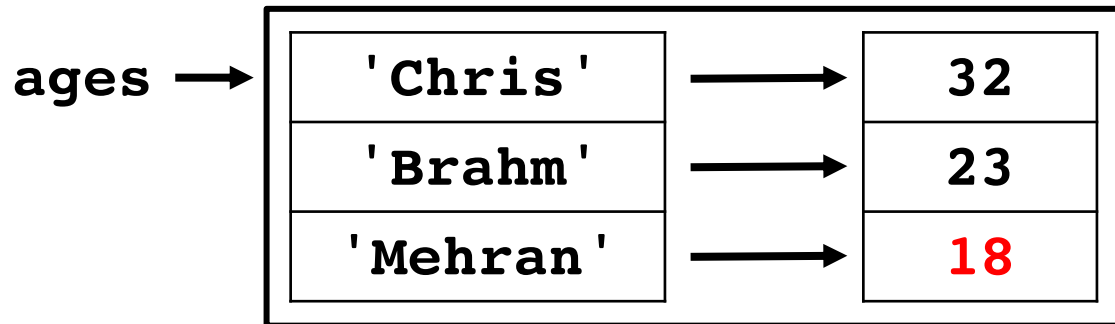


Accessing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Chris': 32, 'Brahm': 23, 'Mehran': 50}
```

- Like a set of variables that are indexed by keys



- Use key to access associated value:

```
ages['Chris'] is 32
```

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

- Can set values like regular variable:

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

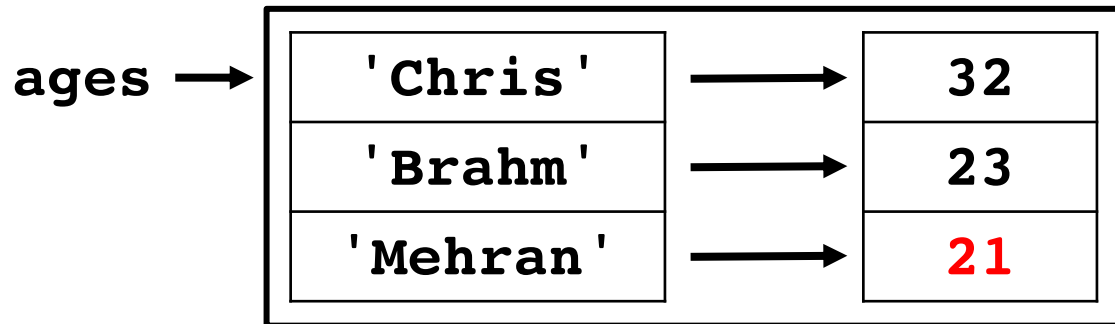


Accessing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Chris': 32, 'Brahm': 23, 'Mehran': 50}
```

- Like a set of variables that are indexed by keys



- Use key to access associated value:

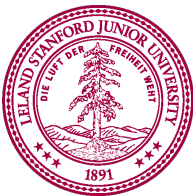
```
ages['Chris'] is 32
```

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

- Can set values like regular variable:

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

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

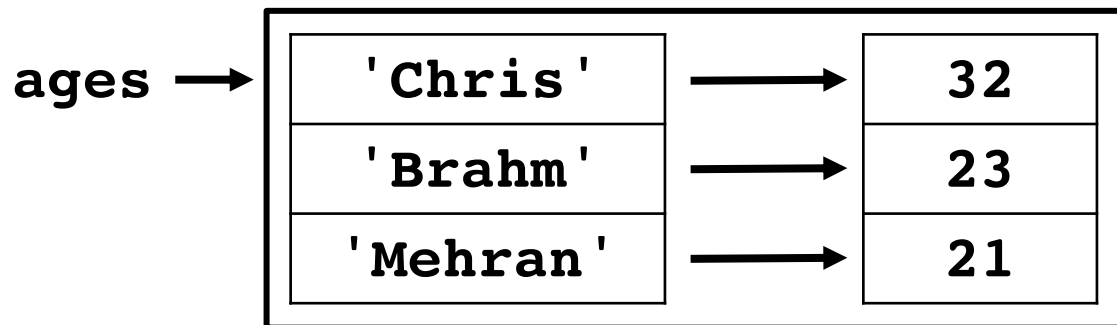


Accessing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Chris': 32, 'Brahm': 23, 'Mehran': 50}
```

- Like a set of variables that are indexed by keys



- Good and bad times with accessing pairs:

```
>>> brahms_age = ages['Brahm']
```

```
>>> brahms_age
```

```
23
```

```
>>> santas_age = ages['Santa Claus']
```

```
KeyError: 'Santa Claus'
```

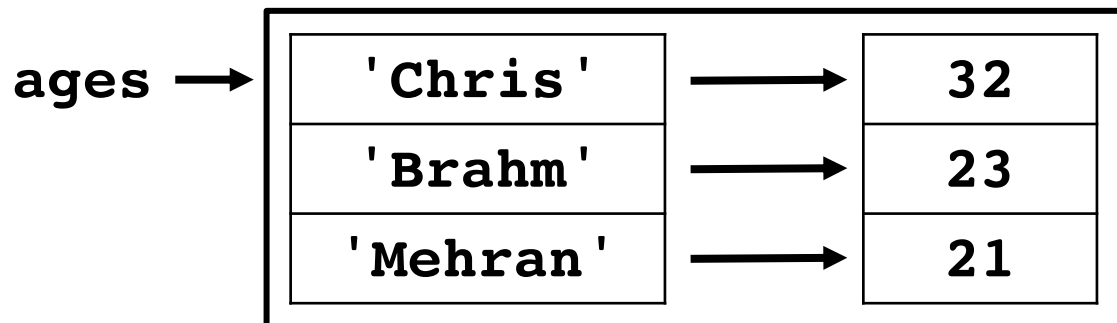


Accessing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Chris': 32, 'Brahm': 23, 'Mehran': 50}
```

- Like a set of variables that are indexed by keys



- Checking membership

```
>>> 'Brahm' in ages
```

```
True
```

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

```
True
```

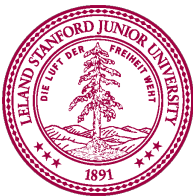


Adding Elements to Dictionary

- Can add pairs to a dictionary:

phone = {}

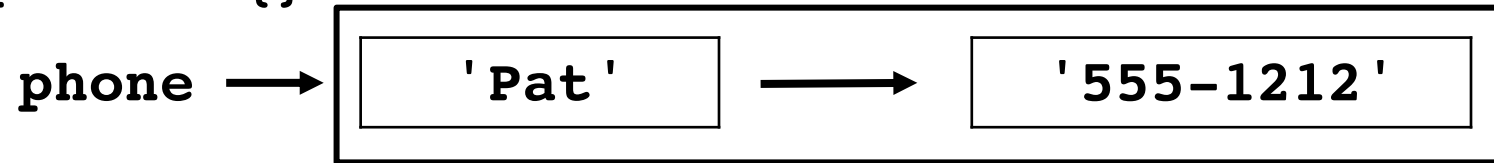
phone → *Empty dictionary*



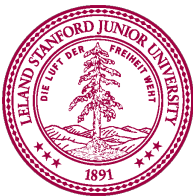
Adding Elements to Dictionary

- Can add pairs to a dictionary:

```
phone = {}
```



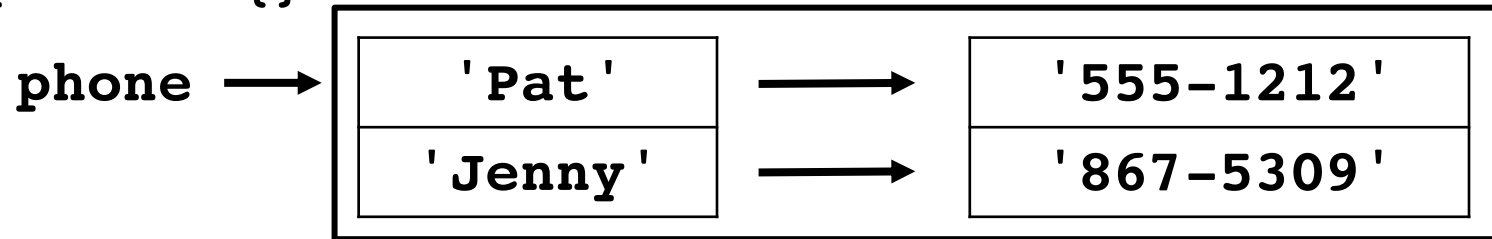
```
phone[ 'Pat' ] = '555-1212'
```



Adding Elements to Dictionary

- Can add pairs to a dictionary:

```
phone = {}
```



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

```
phone[ 'Jenny' ] = '867-5309'
```

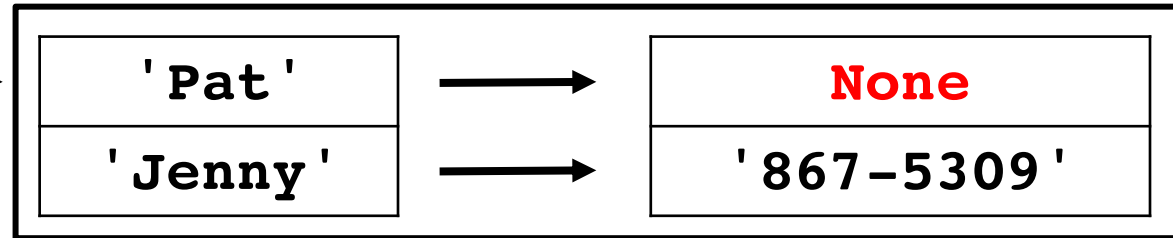


Adding Elements to Dictionary

- Can add pairs to a dictionary:

```
phone = {}
```

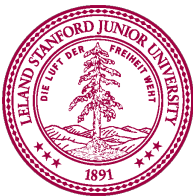
```
phone →
```



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

```
phone['Jenny'] = '867-5309'
```

```
phone['Pat'] = None
```

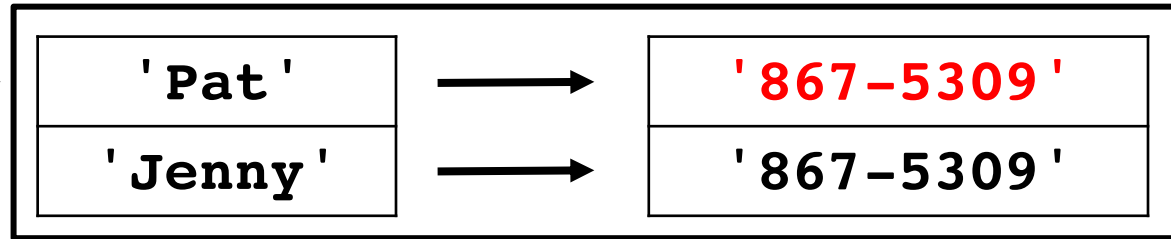


Adding Elements to Dictionary

- Can add pairs to a dictionary:

```
phone = {}
```

```
phone →
```

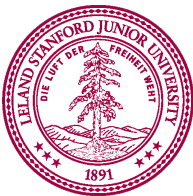


```
phone[ 'Pat' ] = '555-1212'
```

```
phone[ 'Jenny' ] = '867-5309'
```

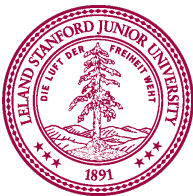
```
phone[ 'Pat' ] = None
```

```
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 List in a Function

```
def have_birthday(dict, name):  
    print("You're one year older, " + name + "!")  
    dict[name] += 1  
  
def main():  
    ages = {'Chris': 32, 'Brahm': 23, 'Mehran': 50}  
    print(ages)  
    have_birthday(ages, 'Chris')  
    print(ages)  
    have_birthday(ages, 'Mehran')  
    print(ages)
```

Terminal:

```
{'Chris': 32, 'Brahm': 23, 'Mehran': 50}  
You're one year older, Chris!  
{'Chris': 33, 'Brahm': 23, 'Mehran': 50}  
You're one year older, Mehran!  
{'Chris': 33, 'Brahm': 23, 'Mehran': 51}
```

Dictionary-palooza! (Part 1)

```
ages = {'Chris': 32, 'Brahm': 23, 'Mehran': 50}
```

- Function: dict.get(key)

- Returns value associated with key in dictionary. Returns **None** if key doesn't exist.

```
>>> print(ages.get('Chris'))
```

```
32
```

```
>>> 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))
```

```
32
```

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

```
100
```



Dictiona-palooza! (Part 2)

```
ages = {'Chris': 32, 'Brahm': 23, '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:

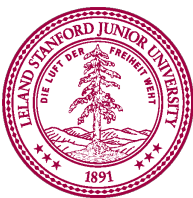
```
for key in ages.keys():  
    print(str(key) + ", " + str(ages[key]))
```

Terminal:

```
Chris, 32  
Brahm, 23  
Mehran, 50
```

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

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



Dictiona-palooza! (Part 3)

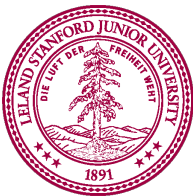
```
ages = {'Chris': 32, 'Brahm': 23, 'Mehran': 50}
```

- 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, 32  
Brahm, 23  
Mehran, 50
```



Dictionary-palooza! (Part 4)

```
ages = {'Chris': 32, 'Brahm': 23, '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:

```
32  
23  
50
```

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

```
>>> list(ages.values())  
[32, 23, 50]
```



Dictiona-palooza! (Part 5)

```
ages = {'Chris': 32, 'Brahm': 23, 'Mehran': 50}
```

- Function: *dict*.pop(key)

- Removes key/value pair with the given key. Returns value from that key/value pair.

```
>>> ages
```

```
>>> {'Chris': 32, 'Brahm': 23, 'Mehran': 50}
```

```
>>> ages.pop('Mehran')
```

```
50
```

```
>>> ages
```

```
{'Chris': 32, 'Brahm': 23}
```

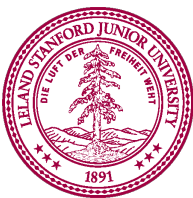
- Function: *dict*.clear()

- Removes all key/value pairs in the dictionary.

```
>>> ages.clear()
```

```
>>> ages
```

```
{}
```



Functions You Can Apply

```
ages = {'Chris': 32, 'Brahm': 23, 'Mehran': 50}
```

- Function: `len(dict)`

- Returns number of key/value pairs in the dictionary

```
>>> ages
```

```
{'Chris': 32, 'Brahm': 23, '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': 32, 'Brahm': 23, 'Mehran': 50}
```

```
>>> del ages['Mehran']
```

```
>>> ages
```

```
{'Chris': 32, 'Brahm': 23}
```



Putting it all together:
`count_each_word.py`




(And we'll also throw in files
as a bonus concept!)

Bonus fun:
phonebook.py

Learning Goals

1. Learning about dictionaries
2. Building programs using dictionaries



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

