Dictionaries

Chris Gregg
Based on slides by Chris Piech and Mehran Sahami
CS106A, Stanford University

Review of String

```python
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

```python
def count_words(filename):
    count = 0
    with open(filename, 'r') as file:   # Open file to read
        for line in file:
            line = line[:-1]                 # Remove newline
            word_list = line.split()         # Create list of words
            for word in word_list:           # Print words
                print(f"#{count}: {word}")
                count += 1
    print(f"{filename} contains {count} words")
```
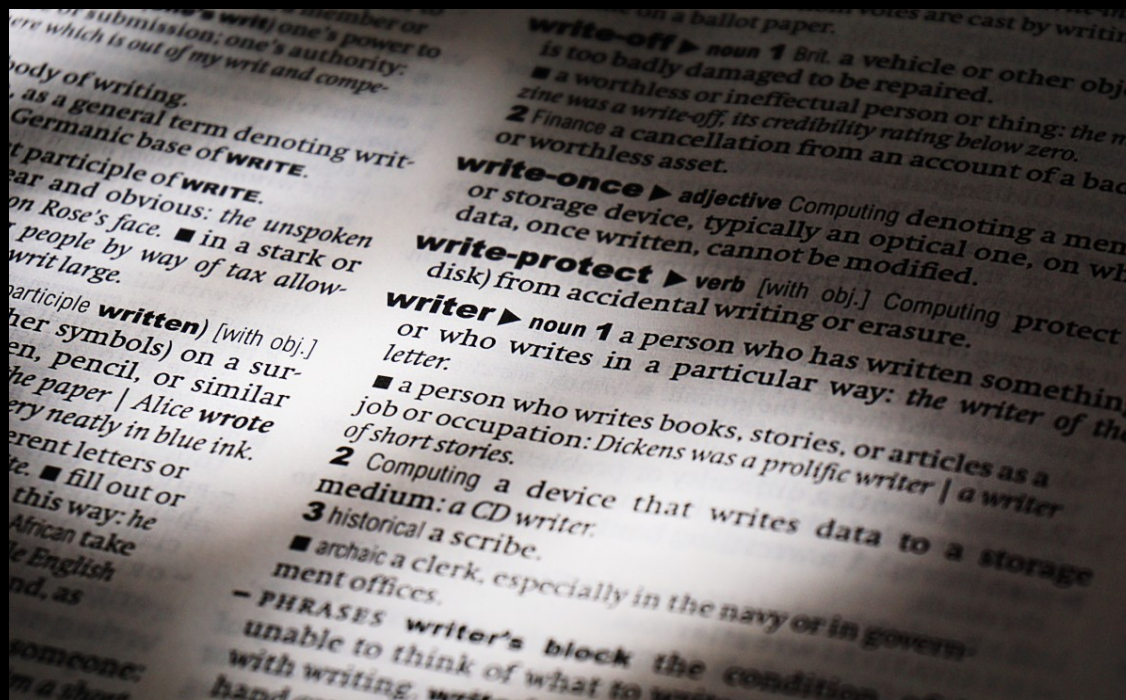
testfile.txt

```
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_**
  - <u>Key</u> is a *unique* identifier
  - <u>Value</u> 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
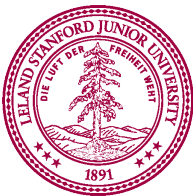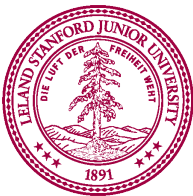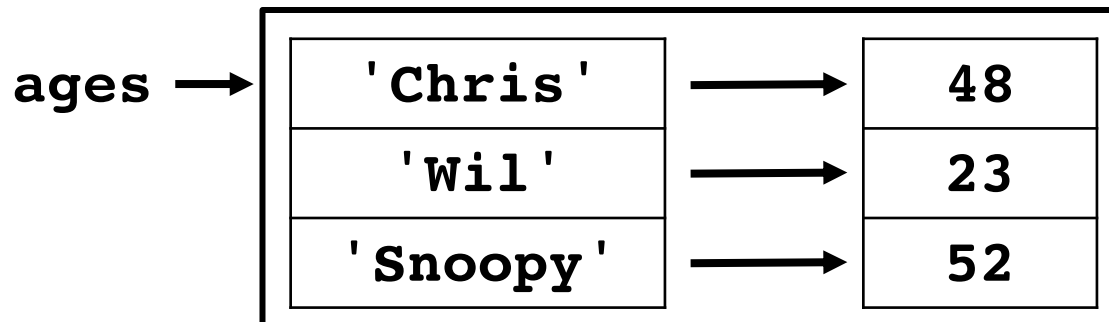
Dictionaries in Python

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

```
ages = {'Chris': 48, 'Wil': 23, 'Snoopy': 52}
squares = {2: 4, 3: 9, 4: 16, 5: 25}
phone = {'Pat': '555-1212', 'Jenny': '867-5309'}
empty_dict = {}
```

ages →

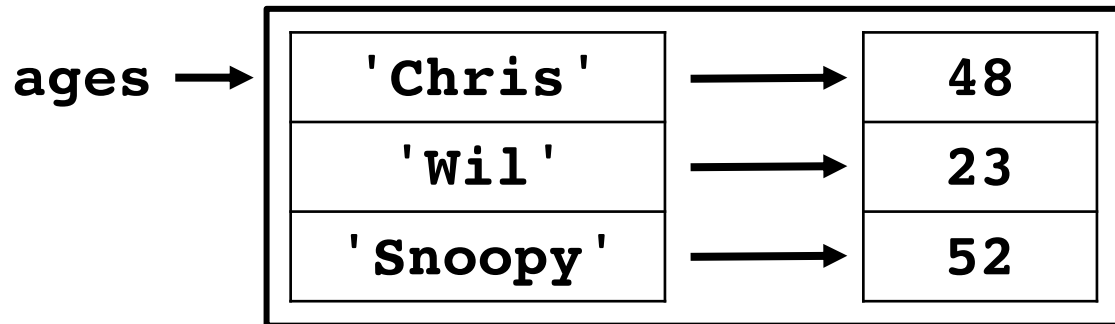|	
'Chris' →	48
'Wil' →	23
'Snoopy' →	52

Accessing Elements of Dictionary

- Consider the following dictionary:

  `ages = {'Chris': 48, 'Wil': 23, 'Snoopy': 52}`

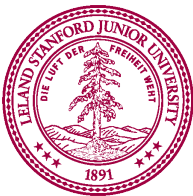- Like a set of variables that are indexed by <u>keys</u>

ages →	'Chris'	→	48
	'Wil'	→	23
	'Snoopy'	→	52

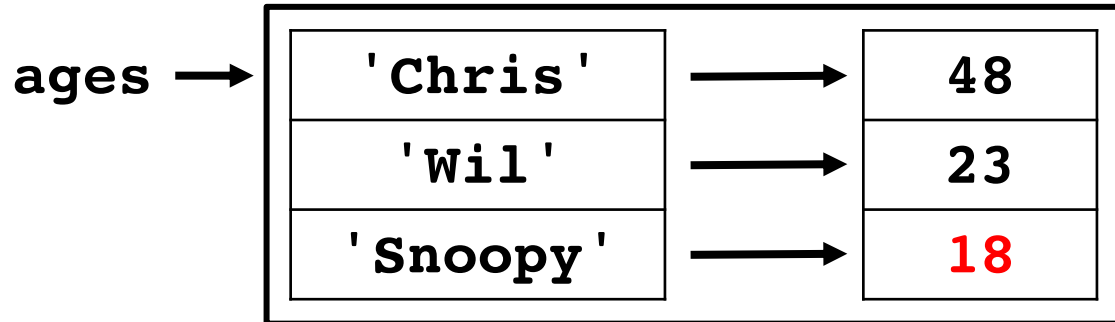- Use <u>key</u> to access associated <u>value</u>:

  `ages['Chris']` is **48**
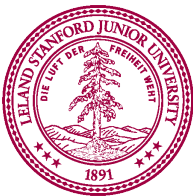
  `ages['Snoopy']` is **52**

Accessing Elements of Dictionary

- Consider the following dictionary:

  `ages = {'Chris': 48, 'Wil': 23, 'Snoopy': 52}`

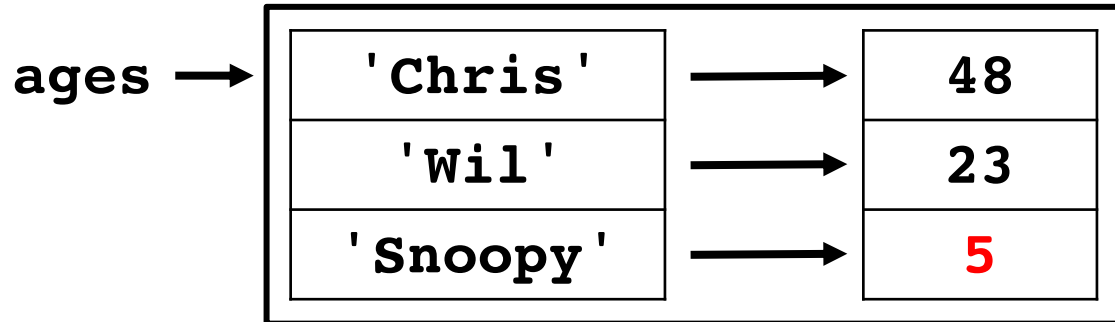- Like a set of variables that are indexed by <u>keys</u>

ages →	**'Chris'** →	**48**
	'Wil' →	**23**
	'Snoopy' →	**18**

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

  `ages['Chris']` is **48**

  `ages['Snoopy']` is **52**

- Can set <u>values</u> like regular variable:
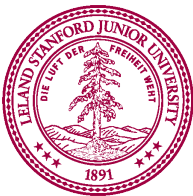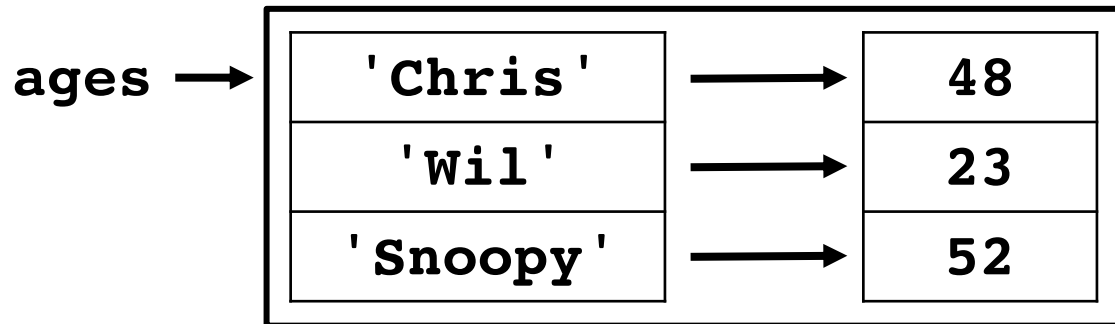
  `ages['Snoopy'] = 18`

Accessing Elements of Dictionary

- Consider the following dictionary:

  `ages = {'Chris': 48, 'Wil': 23, 'Snoopy': 52}`

- 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 `48`

  `ages['Snoopy']` is `52`

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

  `ages['Snoopy'] = 18`

  `ages['Snoopy'] -= 13`

Accessing Elements of Dictionary

- Consider the following dictionary:

  `ages = {'Chris': 48, 'Wil': 23, 'Snoopy': 52}`

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

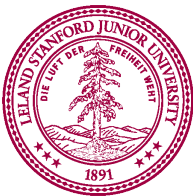| ages →	'Chris'	→	48
	'Wil'	→	23
	'Snoopy'	→	52

- Good and bad times with accessing pairs:

  ```
  >>> wils_age = ages['Wil']
  >>> wils_age
  23
  >>> santas_age = ages['Santa Claus']
  KeyError: 'Santa Claus'
  ```
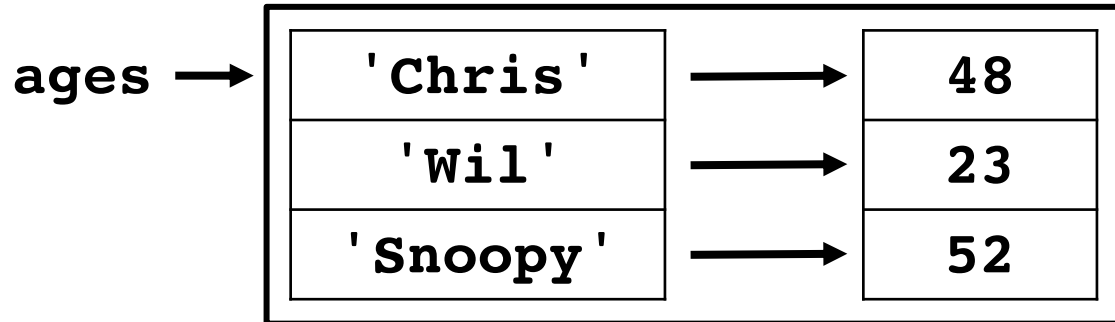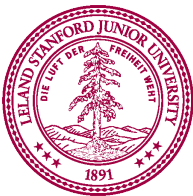
Accessing Elements of Dictionary

- Consider the following dictionary:

  ```
  ages = {'Chris': 48, 'Wil': 23, 'Snoopy': 52}
  ```

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

ages →				
	'Chris'	→	48	
	'Wil'	→	23	
	'Snoopy'	→	52	

- Checking membership

  ```
  >>> 'Wil' in ages
  True
  >>> 'Santa Claus' not in ages
  True
  ```
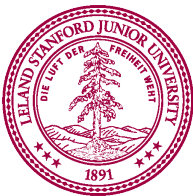
Adding Elements to Dictionary

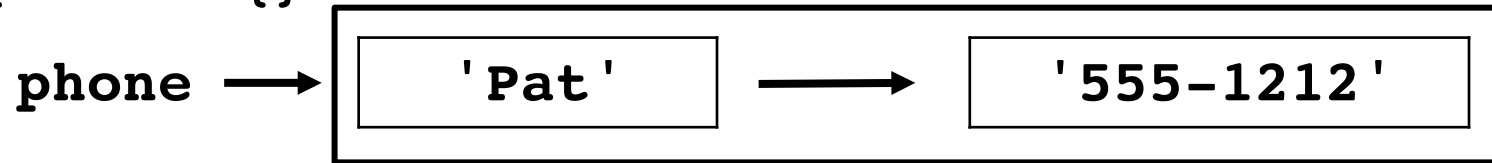- Can add pairs to a dictionary:

    `phone = {}`
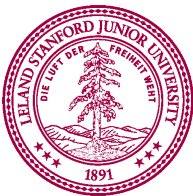
    phone ⟶ *Empty dictionary*

Adding Elements to Dictionary

- Can add pairs to a dictionary:

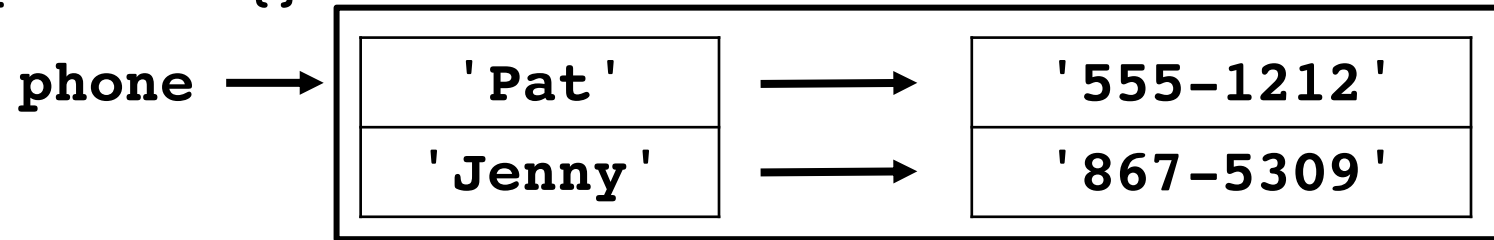```
phone = {}
```

`phone` → `'Pat'` → `'555-1212'`

```
phone['Pat'] = '555-1212'
```
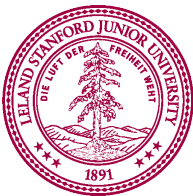
# Adding Elements to Dictionary

- 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'` ⟶ | **None** |
| `'Jenny'` ⟶ | `'867-5309'` |

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

Adding Elements to Dictionary

- Can add pairs to a dictionary:

`phone = {}`

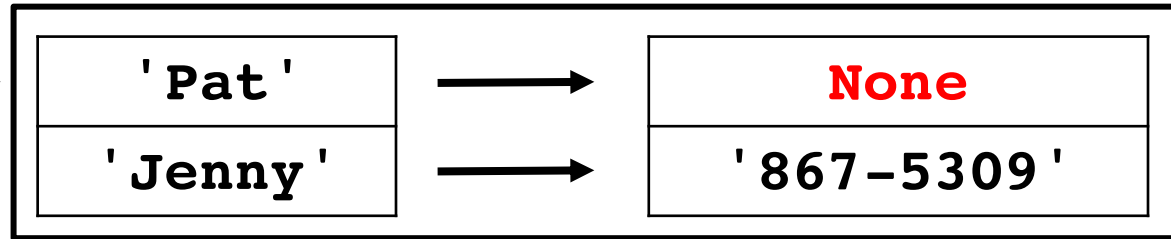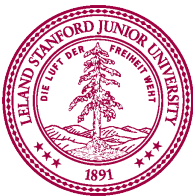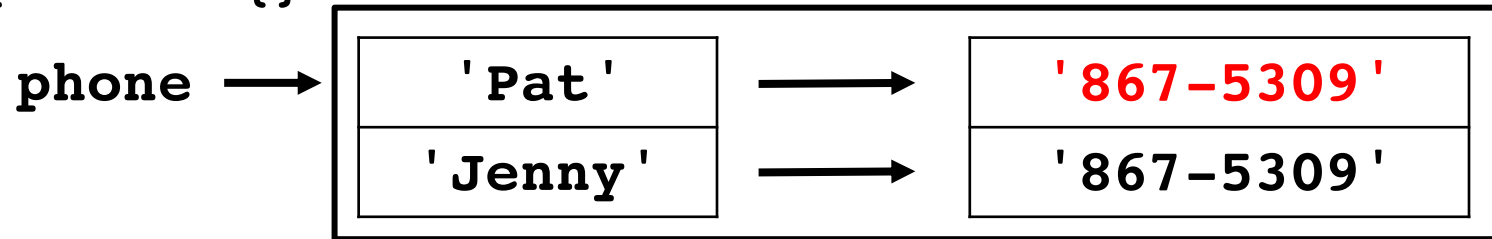```
phone['Pat'] = '555-1212'
phone['Jenny'] = '867-5309'
phone['Pat'] = None
phone['Pat'] = '867-5309'
```
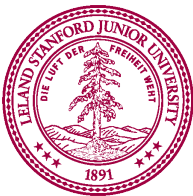
# A Word About Keys/Values

- Keys must be <u>immutable</u> types
  - E.g., int, float, string
  - Keys <u>cannot</u> 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 <u>mutable</u> or <u>immutable</u> types
  - E.g., int, float, string, <u>lists</u>, <u>dictionaries</u>
  - Values <u>can</u> be changed in place

- Dictionaries are <u>mutable</u>
  - Changes made to a dictionary in a function persist after the function is done.
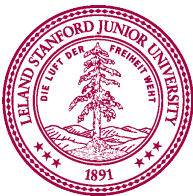
# Changing List in a Function

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

def main():
    ages = {'Chris': 48, 'Wil': 23, 'Snoopy': 52}
    print(ages)
    have_birthday(ages, 'Chris')
    print(ages)
    have_birthday(ages, 'Snoopy')
    print(ages)
```

Terminal:

```
{'Chris': 48, 'Wil': 23, 'Snoopy': 52}
You're one year older, Chris!
{'Chris': 49, 'Wil': 23, 'Snoopy': 52}
You're one year older, Snoopy!
{'Chris': 49, 'Wil': 23, 'Snoopy': 53}
```

Dictiona-palooza! (Part 1)

```
ages = {'Chris': 48, 'Wil': 23, 'Snoopy': 52}
```

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

  ```
  >>> print(ages.get('Chris'))
  48
  >>> print(ages.get('Santa Claus'))
  None
  ```

- Function: _dict_.`get(key, default)`
  - Returns value associated with key in dictionary. Returns <u>default</u> if key doesn't exist.

  ```
  >>> print(ages.get('Chris', 100))
  48
  >>> print(ages.get('Santa Claus', 100))
  100
  ```

Dictiona-palooza! (Part 2)

```
ages = {'Chris': 48, 'Wil': 23, 'Snoopy': 52}
```

- Function: _dict_.`keys()`
    - Returns something similar to a range of the <u>keys</u> in dictionary
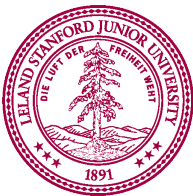   - Can use that to loop over all keys in a dictionary:

```
for key in ages.keys():
    print(f"{key}, {ages[key]}")
```

Terminal:

```
Chris, 48
Wil, 23
Snoopy, 52
```

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

```
>>> list(ages.keys())
['Chris', 'Wil', 'Snoopy']
```

Dictiona-palooza! (Part 3)

```
ages = {'Chris': 48, 'Wil': 23, 'Snoopy': 52}
```

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

```
for key in ages:
    print(f"{key}, {ages[key]}")
```

Terminal:

```
Chris, 48
Wil, 23
Snoopy, 52
```

Dictiona-palooza! (Part 4)

```
ages = {'Chris': 48, 'Wil': 23, 'Snoopy': 52}
```

- Function: _dict_.`values()`
  - Returns something similar to a range of the <u>values</u> in dictionary
  - Can use that to loop over all keys in a dictionary:
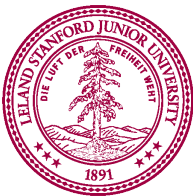
```
for value in ages.values():
    print(value)
```

Terminal:

```
48
23
52
```

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

```
>>> list(ages.values())
[48, 23, 52]
```

Dictiona-palooza! (Part 5)

```
ages = {'Chris': 48, 'Wil': 23, 'Snoopy': 52}
```

- Function: _dict_`.pop(key)`
  - <u>Removes</u> key/value pair with the given key. Returns <u>value</u> from that key/value pair.

```
>>> ages
>>> {'Chris': 48, 'Wil': 23, 'Snoopy': 52}
>>> ages.pop('Snoopy')
52
>>> ages
{'Chris': 48, 'Wil': 23}
```
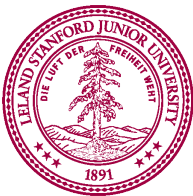
- Function: _dict_`.clear()`
  - <u>Removes</u> all key/value pairs in the dictionary.

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

Functions You Can Apply

```
ages = {'Chris': 48, 'Wil': 23, 'Snoopy': 52}
```

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

```
>>> ages
{'Chris': 48, 'Wil': 23, 'Snoopy': 52}
>>> len(ages)
3
```

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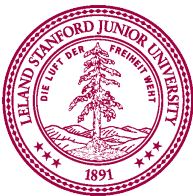 - Similar to **pop**, but doesn't return anything.

```
>>> ages
{'Chris': 48, 'Wil': 23, 'Snoopy': 52}
>>> del ages['Snoopy']
>>> ages
{'Chris': 48, 'Wil': 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': 🌯}