Dictionaries

{106A: great}

Stanford | ENGINEERING
Computer Science

Housekeeping

- Assignment 4 due Tuesday Aug 1 at 11:59pm , grace period until Aug 2 11:59
- Midterm scores will be released early next week
- No OH tomorrow :(
- Tea time still on today!

Today

- **Introduce dictionaries**
  - **A whole new data structure!**
  - **Our last data structure!**

What are Dictionaries?

- Dictionaries associate a ***key*** with a ***value***
  - Key is a *unique* identifier
  - Value is something we associate with that key

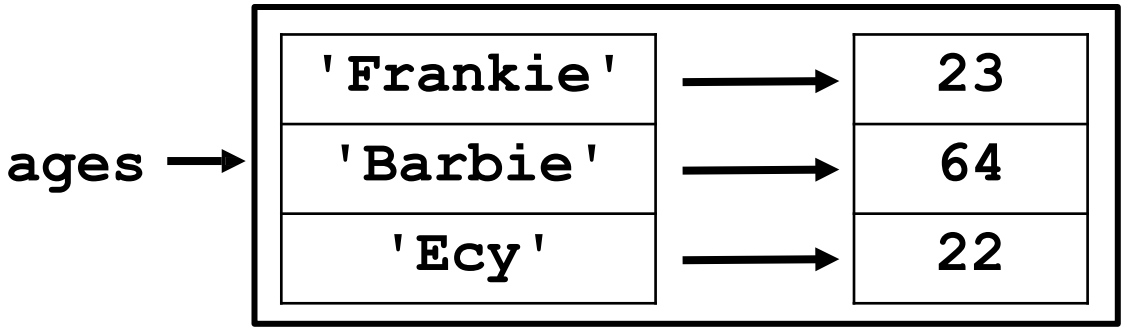
What are Dictionaries?

- 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 = {'Frankie': 23, 'Barbie': 64, 'Ecy': 22}
empty_dict = {}
```

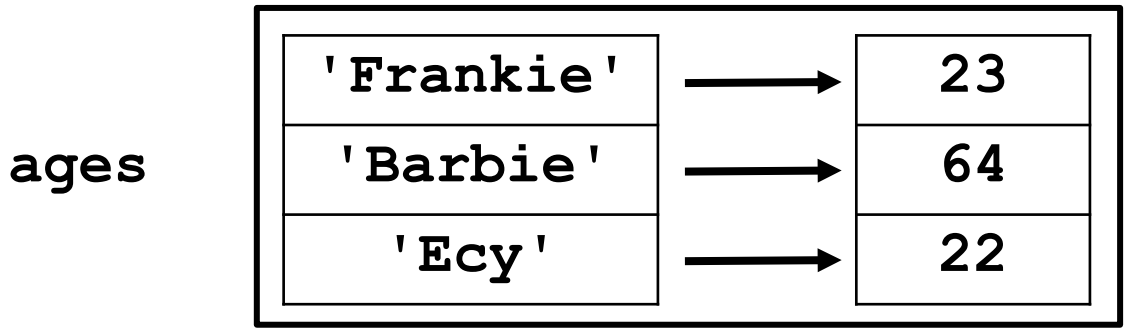


Accessing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Frankie': 23, 'Barbie': 64, 'Ecy': 22}
```

- Like a list of variables that are indexed by keys



- Use key to access associated value:

  `ages['Frankie']` is `23`

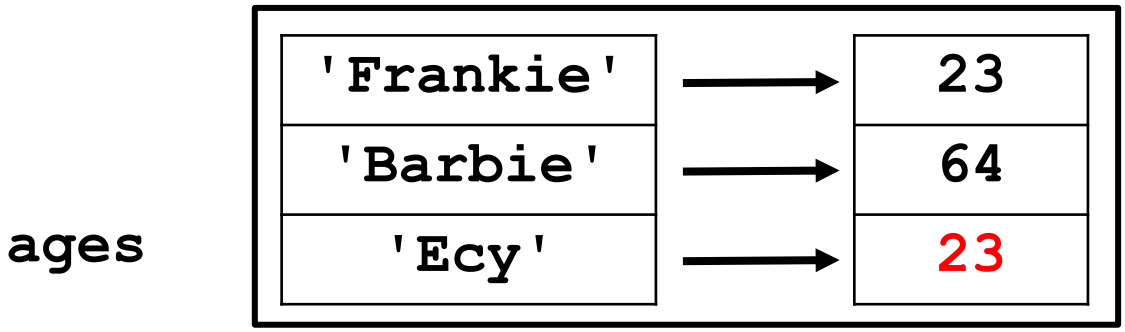
  `ages['Barbie']` is `64`

Changing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Frankie': 23, 'Barbie': 64, 'Ecy': 22}
```

- Like a list of variables that are indexed by keys



- Use key to access associated value:

```
ages['Frankie'] is  23
ages['Barbie'] is 64
```

- Can set values like regular variable:

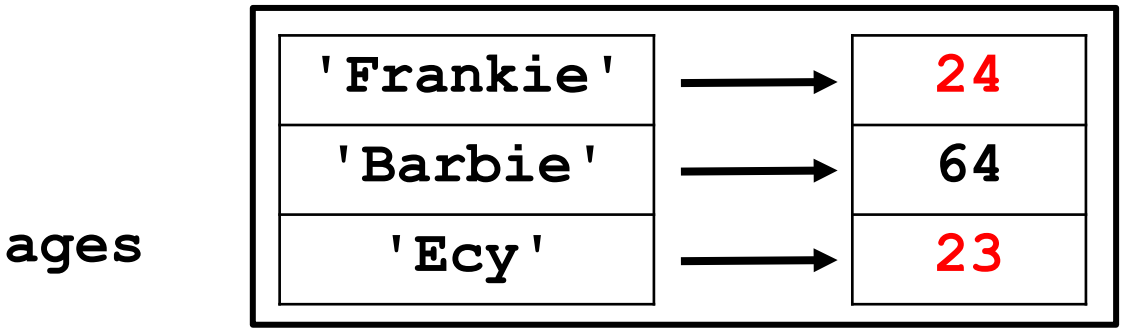
```
ages['Ecy'] = 23 # on April 25th!
```

Changing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Frankie': 23, 'Barbie': 64, 'Ecy': 22}
```

- Like a list of variables that are indexed by keys



- Use key to access associated value:

```
ages['Frankie'] is  23
ages['Barbie'] is 64
```

- Can set values like regular variable:

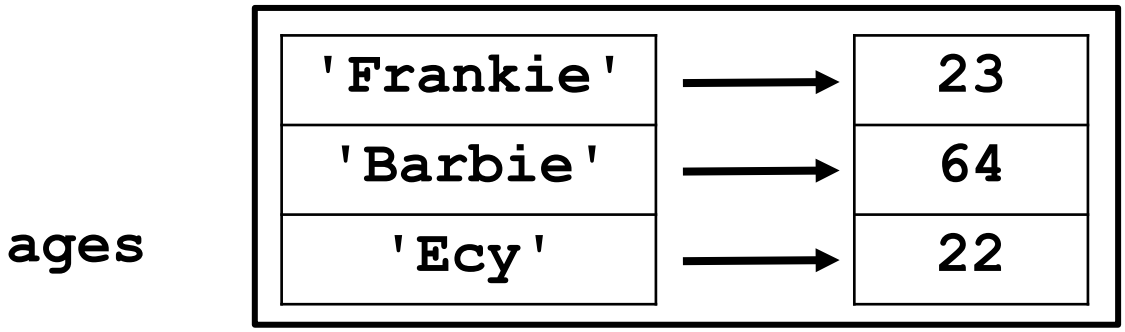
```
ages['Ecy'] = 23 # on April 25th!
ages['Frankie'] += 1 # Feb 24!
```

Changing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Frankie': 23, 'Barbie': 64, 'Ecy': 22}
```

- Like a list of variables that are indexed by keys



- Good and bad times with accessing pairs:

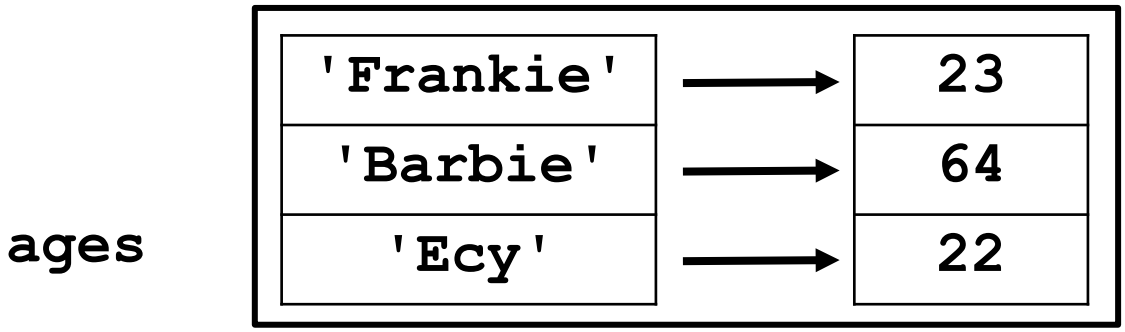
```
ecys_age = ages['Ecy']
print(ecys_age) # prints 22
kens_age = ages['ken'] KeyError: 'ken'
```

Changing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Frankie': 23, 'Barbie': 64, 'Ecy': 22}
```

- Like a list of variables that are indexed by keys



- Good and bad times with accessing pairs:

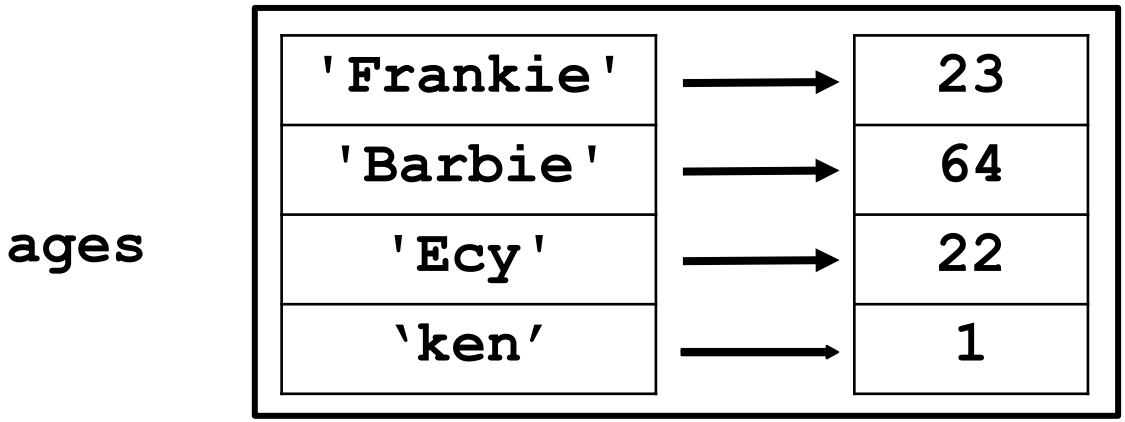
```
ecys_age = ages['Ecy']
print(ecys_age) # prints 22
ages['ken'] += 1 still KeyError: 'ken'
```

Changing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Frankie': 23, 'Barbie': 64, 'Ecy': 22}
```

- Like a list of variables that are indexed by keys



- Good and bad times with accessing pairs:

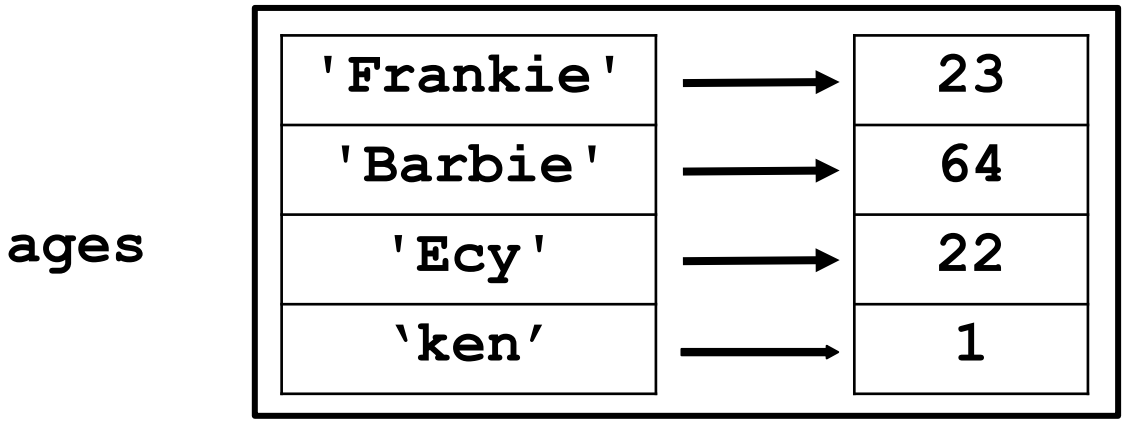
```
ecys_age = ages['Ecy']
print(ecys_age) # prints 22
ages['ken'] = 1 # adds 'ken' : 1 to ages
```

Changing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Frankie': 23, 'Barbie': 64, 'Ecy': 22}
```

- Like a list of variables that are indexed by keys



- Check membership with `in`

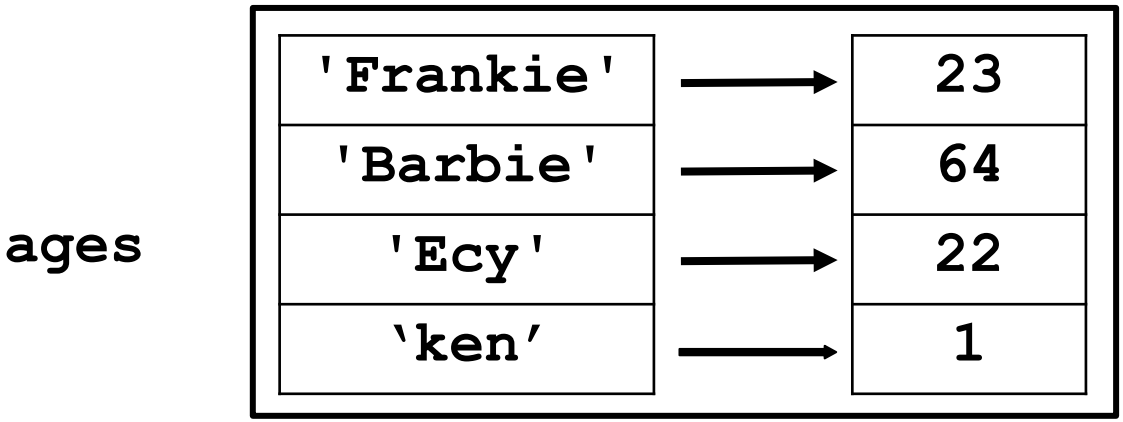
```
print(('allan' in ages)) # prints False
print(('Frankie' in ages)) # prints True
```

Changing Elements of Dictionary

- Consider the following dictionary:

```
ages = {'Frankie': 23, 'Barbie': 64, 'Ecy': 22}
```

- Like a list of variables that are indexed by keys



- Check membership with **in** - only for keys!

```
print(('allan' in ages)) # prints False
print(('Frankie' in ages)) # prints True
print((22 in ages)) # prints False
```

Adding Elements to Dictionary

- Can add pairs to a dictionary:

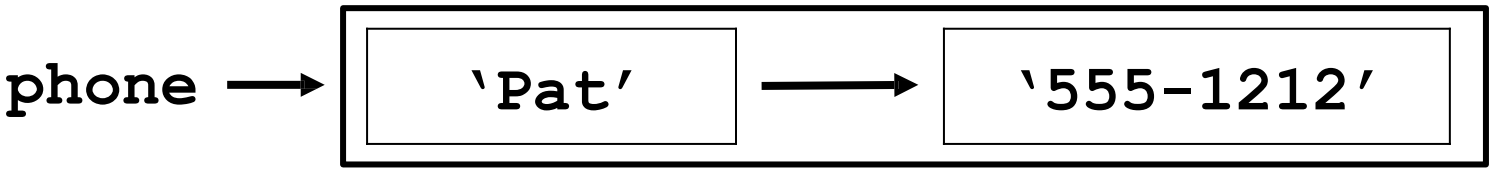
```
phone = {}
```

`phone` ⟶ *Empty dictionary*

Adding Elements to Dictionary

- Can add pairs to a dictionary:

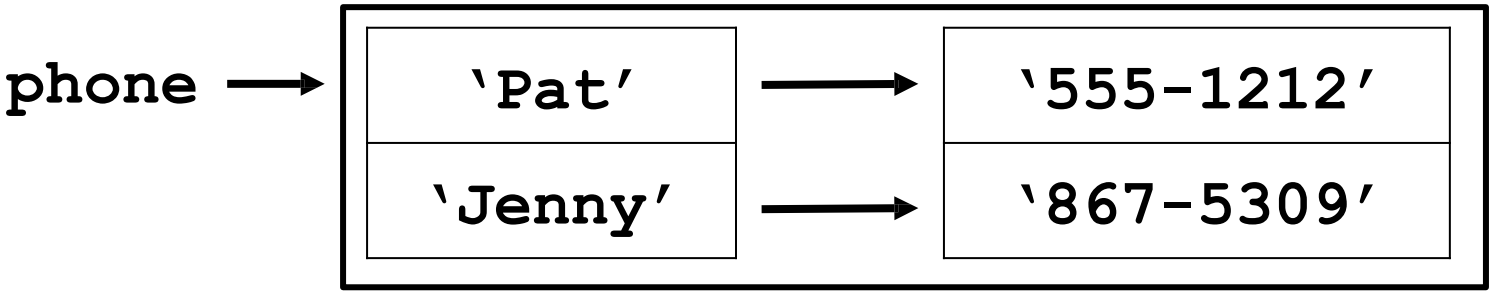
```
phone = {}
phone['Pat'] = '555-1212'
```



Adding Elements to Dictionary

- Can add pairs to a dictionary:

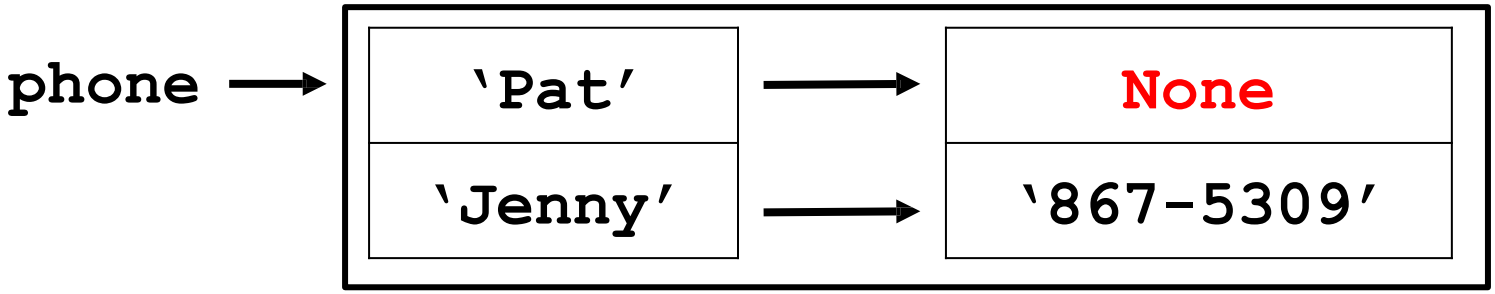
```
phone = {}
phone['Pat'] = '555-1212'
phone['Jenny'] = '867-5309'
```



Adding Elements to Dictionary

- Can add pairs to a dictionary:

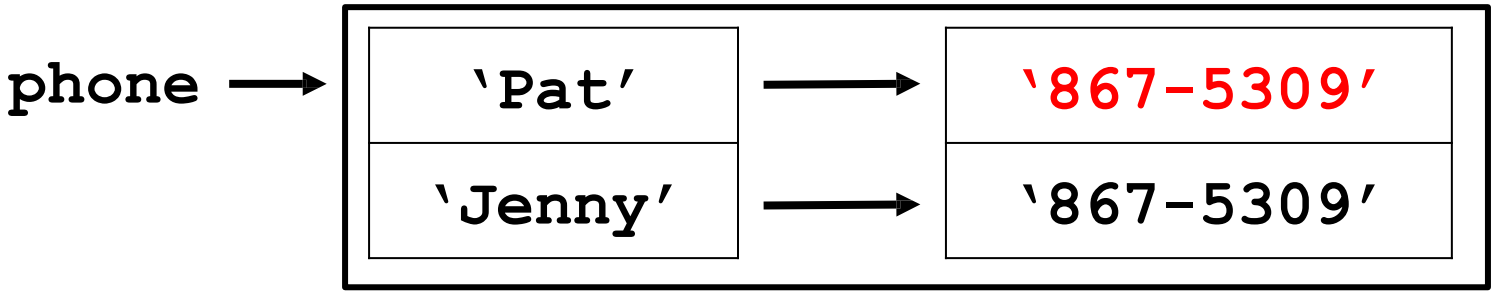
```
phone = {}
phone['Pat'] = '555-1212'
phone['Jenny'] = '867-5309'
phone['Pat'] = None
```



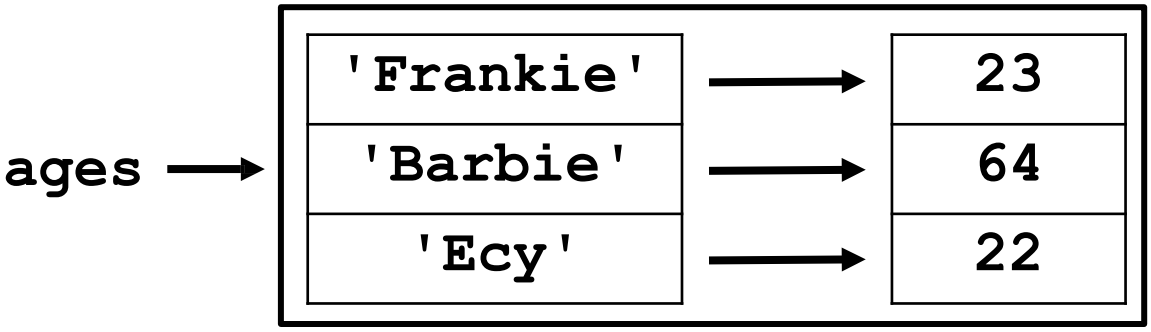
Adding Elements to Dictionary

- Can add pairs to a dictionary:

```
phone = {}
phone['Pat'] = '555-1212'
phone['Jenny'] = '867-5309'
phone['Pat'] = None
phone['Pat'] = '867-5309'
# duplicate values allowed
```



Looping over a dictionary




```
for key in ages:
  print(f"{key} -> ages[key]")
  # the same as
  # print(key + " -> " + ages[key])
```

Terminal:

Frankie -> 23

Barbie -> 64

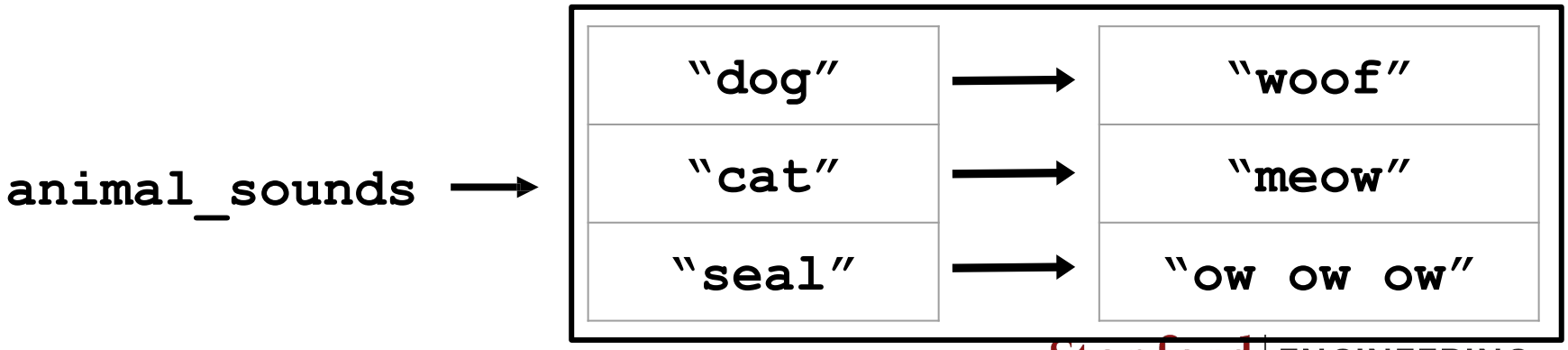
Ecy -> 24

Dict Review

```
# 1. Make a new Dict
animal_sounds = {}

# 2. Put things into the Dict
animal_sounds["dog"] = "woof"
animal_sounds["cat"] = "meow"
animal_sounds["seal"] = "ow ow ow"

# 3. Get things out of the Dict
dog_sound = animal_sounds["dog"] # "woof"
```



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.

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

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.

Dictiona-palooza! (Part 1)

```
ages = {'Frankie': 23, 'Ecy': 22, 'Barbie': 64}
```

- 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(key)
```

Terminal:

```
Frankie
Ecy
Barbie
```

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

```
>>> list(ages.keys())
['Frankie', 'Ecy', 'Barbie']
```

Dictiona-palooza! (Part 2)

```
ages = {'Frankie': 23, 'Ecy': 22, 'Barbie': 64}
```

- 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:

```
23
22
64
```

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

```
>>> list(ages.values())
[24, 22, 64]
```

Dictiona-palooza! (Part 2)

```
ages = {'Frankie': 23, 'Ecy': 22, 'Barbie': 64}
```

- Function: ***dict*.items()**
  - Returns a range of key, value pairs
  - Can use that to loop over all key value pairs in a dictionary:

```
for key, value in ages.items():
    print(f"{key}, {value}")
```

Terminal:

```
Frankie, 23
Ecy, 22
Barbie, 64
```

Dictiona-palooza! (Part 3)

```
ages = {'Frankie': 23, 'Ecy': 22, 'Barbie': 64}
```

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

```
print(ages) # {'Frankie': 23, 'Ecy': 22, 'Barbie': 64}
print(ages.pop('Ecy')) # 22
print(ages) # {'Frankie': 23, 'Barbie': 64}
```

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

```
ages.clear()
print(ages) # {}
```

Functions You Can Apply

```
ages = {'Frankie': 23, 'Ecy': 22, 'Barbie': 64}
```

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

```
print(len(ages)) # 3
```

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

```
del ages['Frankie']
print(ages) # {'Ecy': 22, 'Barbie': 64}
```

phonebook.py

- Write a program that reads in a csv with this format:

```
name1, phone_number1
name2, phone_number2
```

...

- And stores the data in a dictionary structured like so:

```
{
    'name1': 'phone_number1',
    'name2': 'phone_number2'
}
```

- Also takes in as command line arguments any number of names and prints the associated phone number, if it exists!

To Pycharm! phonebook.py

count_words.py

- Write a program that takes in a filename as command line input

- Decompose a function that reads every line in the file and counts the number of times each word, **case insensitive,** appears - save this in a dictionary and return. Don't worry about punctuation
  - Write doctests to test before moving on

- The command line should also take in any number of words, and print the number of times that word appears in the file

To Pycharm! count_words

(on your own)Modify data_processing

- Write a program that allows the user to specify the filename of a CSV, a column number in that CSV, a min_frequency and a max_frequency, and any number of string values
- Display a bar chart representing the frequency with which each string value appears in the specified column in the dataset
- (Demo in the started code)
- Use the pre-made **`make_bar_chart`** function
- Decompose logic to process the file
- Use it on our anonymized Assn0 answers!

(on your own)data_processing_dict.py Milestones

1. Understand provided code

2. Write function that returns a dictionary of **label: frequency** for each string in the given list of values

3. Test above function on small dataset

4. Modify make_bar_chart to expect a dictionary, not two lists

5. Call **make_bar_chart**

106A Milestone: Core Datastructures

A✅atasets can be represented by:

- **Dictionaries** ✅
- **Lists** ✅
  - Strings ✅
  - Floats ✅
  - Integers ✅
  - Booleans ✅

Recap

- Dictionaries exist
- They associate keys to values, and we can look up values using keys
- They look like this:

```
{
    key1: value1,
    key2: value2,
    ...
}
```

- Can access/change values with `dict[key]`