Sets and Dictionaries

Dictionaries

Back to the data from our summer counting birds in a mosquito-infested swamp in northern Ontario

Back to the data from our summer counting birds in a mosquito-infested swamp in northern Ontario

How many birds of each kind did we see?

Back to the data from our summer counting birds in a mosquito-infested swamp in northern Ontario

How many birds of each kind did we see?

Input is a list of several thousand bird names

Back to the data from our summer counting birds in a mosquito-infested swamp in northern Ontario

How many birds of each kind did we see?

Input is a list of several thousand bird names

Output is a list of names and counts

Could use a list of [name, count] pairs

Could use a list of [name, count] pairs

```
def another_bird(counts, bird_name):
  for i in range(len(counts)):
    if counts[i][0] == bird_name:
      counts[i][1] += 1
      return
  counts.append([bird_name, 1])
```

Could use a list of [name, count] pairs

```
def another_bird(counts, bird_name):
  for i in range(len(counts)):
    if counts[i][0] == bird_name:
      counts[i][1] += 1
      return
  counts.append([bird_name, 1])
```

List of pairs

Could use a list of [name, count] pairs

```
def another_bird(counts, bird_name):
  for i in range(len(counts)):
    if counts[i][0] == bird_name:
      counts[i][1] += 1
      return
  counts.append([bird_name, 1])
```

Name to add

Could use a list of [name, count] pairs

```
def another_bird(counts, bird_name):
  for i in range(len(counts)):
    if counts[i][0] == bird_name:
      counts[i][1] += 1
      return
  counts.append([bird_name, 1)
```

Look at each pair already in the list

Could use a list of [name, count] pairs

```
def another_bird(counts, bird_name):
  for i in range(len(counts)):
    if counts[i][0] == bird_name:
      counts[i][1] += 1
      return
  counts.append([bird_name, 1])
```

If this is the bird we're looking for...

Could use a list of [name, count] pairs

```
def another_bird(counts, bird_name):
  for i in range(len(counts)):
    if counts[i][0] == bird_name:
      counts[i][1] += 1
      return
  counts.append([bird_name, 1])
```

...add 1 to its count and finish

Could use a list of [name, count] pairs

```
def another_bird(counts, bird_name):
  for i in range(len(counts)):
    if counts[i][0] == bird_name:
      counts[i][1] += 1
      return
  counts.append([bird_name, 1])
```

Otherwise, add a new pair to the list

Could use a list of [name, count] pairs

```
def another_bird(counts, bird_name):
  for i in range(len(counts)):
    if counts[i][0] == bird_name:
      counts[i][1] += 1
      return
  counts.append([bird_name, 1])
```

Pattern: handle an existing case and return in loop, or take default action if we exit the loop normally

Could use a list of [name, count] pairs

```
def another_bird(counts, bird_name):
  for i in range(len(counts)):
    if counts[i][0] == bird_name:
      counts[i][1] += 1
      return
  counts.append([bird_name, 1])
```

start []

Could use a list of [name, count] pairs

```
def another_bird(counts, bird_name):
  for i in range(len(counts)):
    if counts[i][0] == bird_name:
      counts[i][1] += 1
      return
  counts.append([bird_name, 1])
```

start	[]
loon	[['loon', 1]]

Could use a list of [name, count] pairs

```
def another_bird(counts, bird_name):
  for i in range(len(counts)):
    if counts[i][0] == bird_name:
      counts[i][1] += 1
      return
  counts.append([bird_name, 1])
```

start	[]
loon	[['loon', 1]]
goose	[['loon', 1], ['goose', 1]]

Could use a list of [name, count] pairs

```
def another_bird(counts, bird_name):
  for i in range(len(counts)):
    if counts[i][0] == bird_name:
      counts[i][1] += 1
      return
  counts.append([bird_name, 1])
```

start	[]
loon	[['loon', 1]]
goose	[['loon', 1], ['goose', 1]]
loon	[['loon', 2], ['goose', 1]]

There's a better way

There's a better way

Use a *dictionary*

There's a better way

Use a *dictionary*

An unordered collection of key/value pairs

There's a better way

Use a *dictionary*

An unordered collection of key/value pairs

Like set elements, keys are:

There's a better way

Use a *dictionary*

An unordered collection of key/value pairs

Like set elements, keys are:

- Immutable

There's a better way

Use a *dictionary*

An unordered collection of key/value pairs

Like set elements, keys are:

- Immutable

- Unique

There's a better way

Use a *dictionary*

An unordered collection of key/value pairs

Like set elements, keys are:

- Immutable
- Unique
- Not stored in any particular order

There's a better way

Use a *dictionary*

An unordered collection of key/value pairs

Like set elements, keys are:

- Immutable

- Unique

- Not stored in any particular order

No restrictions on values

There's a better way

Use a *dictionary*

An unordered collection of key/value pairs

Like set elements, keys are:

- Immutable
- Unique
- Not stored in any particular order

No restrictions on values

- Don't have to be immutable or unique

Create a dictionary by putting key:value pairs in {}

Create a dictionary by putting key:value pairs in {}

```
>>> birthdays = {'Newton' : 1642, 'Darwin' : 1809}
```

Create a dictionary by putting key:value pairs in {}

```
>>> birthdays = {'Newton' : 1642, 'Darwin' : 1809}
```

Retrieve values by putting key in []

Create a dictionary by putting key:value pairs in {}

```
>>> birthdays = {'Newton' : 1642, 'Darwin' : 1809}
```

Retrieve values by putting key in []

Just like indexing strings and lists

Create a dictionary by putting key:value pairs in {}

```
>>> birthdays = {'Newton' : 1642, 'Darwin' : 1809}
```

Retrieve values by putting key in []

Just like indexing strings and lists

```
>>> print birthdays['Newton']
```

1642

Create a dictionary by putting key:value pairs in {}

```
>>> birthdays = {'Newton' : 1642, 'Darwin' : 1809}
```

Retrieve values by putting key in []

Just like indexing strings and lists

```
>>> print birthdays['Newton']
```

1642

Just like using a phonebook or dictionary

Add another value by assigning to it

Add another value by assigning to it

```
>>> birthdays['Turing'] = 1612     # that's not right
```

Add another value by assigning to it

```
>>> birthdays['Turing'] = 1612     # that's not right
```

Overwrite value by assigning to it as well

Add another value by assigning to it

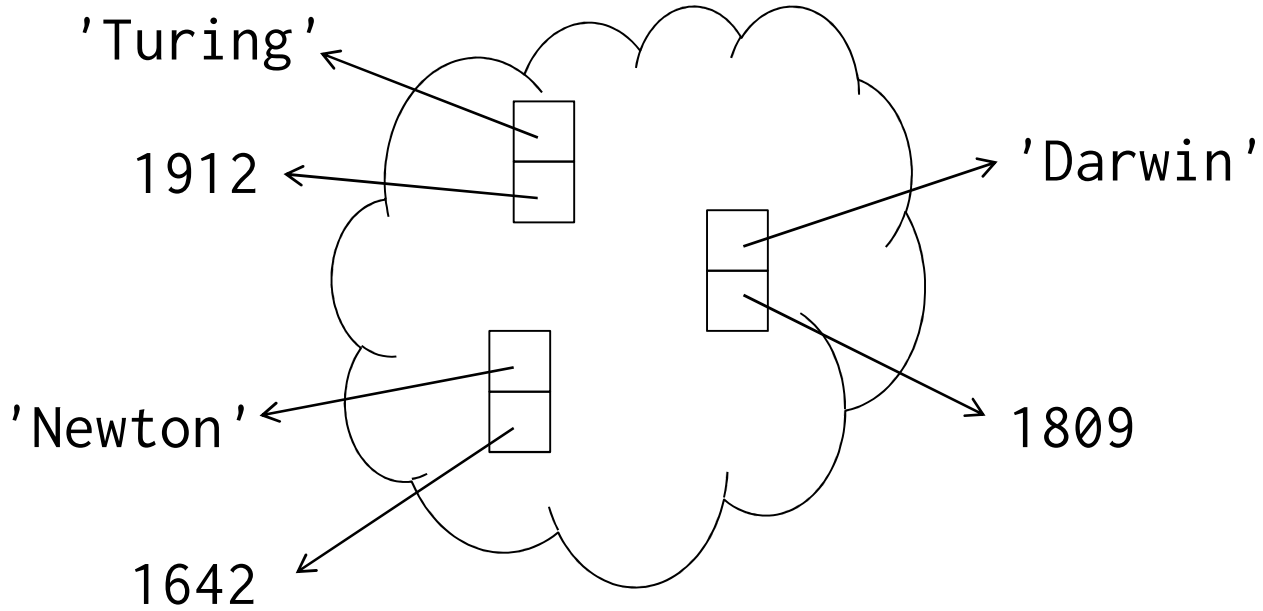
```
>>> birthdays['Turing'] = 1612     # that's not right
```

Overwrite value by assigning to it as well

```
>>> birthdays['Turing'] = 1912
>>> print birthdays
{'Turing' : 1912, 'Newton' : 1642, 'Darwin' : 1809}
```

Note: entries are *not* in any particular order

Note: entries are *not* in any particular order



Key must be in dictionary *before* use

Key must be in dictionary *before* use

```
>>> birthdays['Nightingale']
KeyError: 'Nightingale'
```

Key must be in dictionary *before* use

```
>>> birthdays['Nightingale']
KeyError: 'Nightingale'
```

Test whether key is present using `in`

Key must be in dictionary *before* use

```
>>> birthdays['Nightingale']
KeyError: 'Nightingale'
```

Test whether key is present using `in`

```
>>> 'Nightingale' in birthdays
False
>>> 'Darwin' in birthdays
True
```

Use `for` to loop over keys

Use `for` to loop over keys

Unlike lists, where `for` loops over values

Use `for` to loop over keys

Unlike lists, where `for` loops over values

```
>>> for name in birthdays:
...   print name, birthdays[name]
```

```
Turing 1912
Newton 1642
Darwin 1809
```

Let's count those birds

Let's count those birds

```
import sys

if __name__ == '__main__':
  reader = open(sys.argv[1], 'r')
  lines = reader.readlines()
  reader.close()
  count = count_names(lines)
  for name in count:
    print name, count[name]
```

Let's count those birds

```
import sys

if __name__ == '__main__':
  reader = open(sys.argv[1], 'r')
  lines = reader.readlines()
  reader.close()
  count = count_names(lines)
  for name in count:
    print name, count[name]
```

Read all the data

Let's count those birds

```
import sys

if __name__ == '__main__':
  reader = open(sys.argv[1], 'r')
  lines = reader.readlines()
  reader.close()
  count = count_names(lines)
  for name in count:
    print name, count[name]
```

← Count distinct values

Let's count those birds

```
import sys

if __name__ == '__main__':
  reader = open(sys.argv[1], 'r')
  lines = reader.readlines()
  reader.close()
  count = count_names(lines)
  for name in count:
    print name, count[name]
```

Show results

```
def count_names(lines):
  '''Count unique lines of text, returning dictionary.'''

  result = {}
  for name in lines:
    name = name.strip()
    if name in result:
      result[name] = result[name] + 1
    else:
      result[name] = 1

  return result
```

```
def count_names(lines):
  '''Count unique lines of text, returning dictionary.'''

  result = {}
  for name in lines:
    name = name.strip()
    if name in result:
      result[name] = result[name] + 1
    else:
      result[name] = 1

  return result
```

Explain what we're doing to the next reader

```
def count_names(lines):
  '''Count unique lines of text, returning dictionary.'''

  result = {}
  for name in lines:
    name = name.strip()
    if name in result:
      result[name] = result[name] + 1
    else:
      result[name] = 1

  return result
```

Create an empty dictionary to fill

```
def count_names(lines):
  '''Count unique lines of text, returning dictionary.'''

  result = {}
  for name in lines:
    name = name.strip()
    if name in result:
      result[name] = result[name] + 1
    else:
      result[name] = 1

  return result
```

Handle input values one at a time

```
def count_names(lines):
  '''Count unique lines of text, returning dictionary.'''

  result = {}
  for name in lines:
    name = name.strip()
    if name in result:
      result[name] = result[name] + 1
    else:
      result[name] = 1

  return result
```

Clean up before processing

```
def count_names(lines):
  '''Count unique lines of text, returning dictionary.'''

  result = {}
  for name in lines:
    name = name.strip()
    if name in result:
      result[name] = result[name] + 1
    else:
      result[name] = 1

  return result
```

If we have seen this value before...

```
def count_names(lines):
  '''Count unique lines of text, returning dictionary.'''

  result = {}
  for name in lines:
    name = name.strip()
    if name in result:
      result[name] = result[name] + 1
    else:
      result[name] = 1

  return result
```

...add one to its count

```
def count_names(lines):
  '''Count unique lines of text, returning dictionary.'''

  result = {}
  for name in lines:
    name = name.strip()
    if name in result:
      result[name] = result[name] + 1
    else:
      result[name] = 1

  return result
```

But if it's the first time we have seen this name, store it with a count of 1

```
def count_names(lines):
  '''Count unique lines of text, returning dictionary.'''

  result = {}
  for name in lines:
    name = name.strip()
    if name in result:
      result[name] = result[name] + 1
    else:
      result[name] = 1

  return result
```

Return the result

Counter in action

Counter in action

start {}

Counter in action

start	`{}`
`loon`	`{'loon' : 1}`

Counter in action

start	`{}`
loon	`{'loon' : 1}`
goose	`{'loon' : 1, 'goose' : 1}`

Counter in action

start	`{}`
`loon`	`{'loon' : 1}`
`goose`	`{'loon' : 1, 'goose' : 1}`
`loon`	`{'loon' : 2, 'goose' : 1}`

Counter in action

start	`{}`
`loon`	`{'loon' : 1}`
`goose`	`{'loon' : 1, 'goose' : 1}`
`loon`	`{'loon' : 2, 'goose' : 1}`

But like sets, dictionaries are much more efficient than lookup lists

software carpentry

created by

Greg Wilson

July 2010