

Data Processing and File Access

Contents

- Reading Text Files
- Creating Text Files
- Adding Lines to an Existing Text File
- Altering Items in a Text File
- Sets
- CSV Files
- Dictionary

Reading Text Files

```
def main():
    ## Display the names of the first three presidents.
    file = "FirstPresidents.txt"
    displayWithForLoop(file)
    print()
    displayWithListComprehension(file)
    print()
    displayWithReadline(file)

def displayWithForLoop(file):
    infile = open(file, 'r')
    for line in infile:
        print(line.rstrip())
    infile.close()

def displayWithListComprehension(file):
    infile = open(file, 'r')
    listPres = [line.rstrip() for line in infile]
    infile.close()
    print(listPres)
```

Reading Text Files

```
def displayWithReadline(file):  
    infile = open(file, 'r')  
    line = infile.readline()  
    while line != "":  
        print(line.rstrip())  
        line = infile.readline()  
    infile.close()
```

```
main()
```

```
[Run]
```

```
George Washington
```

```
John Adams
```

```
Thomas Jefferson
```

```
['George Washington', 'John Adams', 'Thomas Jefferson']
```

```
George Washington
```

```
John Adams
```

```
Thomas Jefferson
```

Reading Text Files

- When a text file is opened for input, a pointer is set to the beginning of the first line of the file

- Each time a statement of the form

```
strVar = infile.readline()
```

is executed, the current line is assigned to **strVar** and the pointer advances to the end of that line

- After all the lines of the file have been read, the **readline** method returns the empty string

FirstPresidents.txt

George Washington\nJohn Adams\nThomas Jefferson\n

When file
is opened

After 1st
readline

After 2nd
readline

After 3rd
readline

Creating Text Files

- A statement of the form

```
outfile = open(fileName, 'w')
```

creates and opens a new text file with the specified name for writing

- If *list1* is a list of strings, where each string ends with a newline character (`\n`), then the statement

```
outfile.writelines(list1)
```

writes each item of the list into the file as a line

- If the value of *strVar* is a string, the statement

```
outfile.write(strVar)
```

adds the value of *strVar* to the file

- After all writing has been finished, the file must be **closed** to guarantee that all data has been physically transferred to the disk

Creating Text Files

```
def main():  
    ## Create two files containing the first three presidents.  
    L = ["George Washington", "John Adams",  
         "Thomas Jefferson"]  
    outfile = open("FirstPresidents2.txt", 'w')  
    createWithWrite(L, outfile)  
    outfile = open("FirstPresidents3.txt", 'w')  
    createWithWritelines(L, outfile)  
  
def createWithWrite(L, outfile):  
    for i in range(len(L)):  
        outfile.write(L[i] + "\n")  
    outfile.close()
```

- If an existing file is opened for writing, its original content will be replaced by new content

Creating Text Files

```
def createWithWritelines(L, outfile):  
    # Append newline characters to the list's items.  
    for i in range(len(L)):  
        L[i] = L[i] + "\n"  
    # Write the list's items to the file.  
    outfile.writelines(L)  
    outfile.close()
```

```
main()
```

```
[Run]
```

- Each of the newly created files will look as follows when opened in a text editor

```
George Washington  
John Adams  
Thomas Jefferson
```


Creating Text Files

- The file `states.txt` contains the names of the U.S. states in the order they joined the union
 - The following program uses this file to create a text file named `StatesAlpha.txt` containing the states in alphabetical order

```
def main():  
    ## Create a text file containing the 50 states in  
    # alphabetical order.  
    statesList = createListFromFile("States.txt")  
    createSortedFile(statesList, "StatesAlpha.txt")  
  
def createListFromFile(fileName):  
    infile = open(fileName, 'r')  
    desiredList = [line.rstrip() for line in infile]  
    infile.close()  
    return desiredList
```

Creating Text Files

```
def createSortedFile(listName, fileName):  
    listName.sort()  
    for i in range(len(listName)):  
        listName[i] = listName[i] + "\n"  
    outfile = open(fileName, 'w')  
    outfile.writelines(listName)  
    outfile.close()
```

```
main()
```

```
[Run]
```

- `StatesAlpha.txt` will look as follows when opened in a text editor

```
Alabama  
Alaska  
:  
Wisconsin  
Wyoming
```

Creating Text Files

- The file `USPres.txt` contains the U.S. presidents in the order they served and the file `VPres.txt` contains the names of the people who served as vice presidents of the U.S.
 - The following program creates a file named `Both.txt` containing the names of the presidents who also served as vice president

```
def main():
    ## Create a file of the presidents who also served as
    # vice-presidents.
    vicePresList = createListFromFile("VPres.txt")
    createNewFile(vicePresList, "USPres.txt", "Both.txt")

def createListFromFile(fileName):
    infile = open(fileName, 'r')
    desiredList = [line.rstrip() for line in infile]
    infile.close()
    return desiredList
```



Creating Text Files

```
def createNewFile(listName, oldFileName, newFileName):  
    infile = open(oldFileName, 'r')  
    outfile = open(newFileName, 'w')  
    for person in infile:  
        if person.rstrip() in listName:  
            outfile.write(person)  
    infile.close()  
    outfile.close()  
  
main()  
[Run]
```

- Both.txt will look as follows when opened in a text editor

```
John Adams  
Thomas Jefferson  
:  
Gerald Ford  
George H. W. Bush
```

Adding Lines to an Existing Text File

- A statement of the form

```
outfile = open(fileName, 'a')
```

allows the program to add lines to the end of the specified file
(the file is said to be opened for **append**)

```
def main():  
    ## Add next three presidents to the file containing  
    # first three presidents.  
    outfile = open("FirstPresidents.txt", 'a')  
    list1 = ["James Madison\n", "James Monroe\n"]  
    outfile.writelines(list1)  
    outfile.write("John Q. Adams\n")  
    outfile.close()
```

```
main()
```

```
[Run]
```

Adding Lines to an Existing Text File

- The file `FirstPresidents.txt` will now look as follows when opened in a text editor

```
George Washington  
John Adams  
Thomas Jefferson  
James Madison  
James Monroe  
John Q. Adams
```

Altering Items in a Text File

- Altering, inserting, or deleting a line of a text file cannot be made directly
 - A new file must be created by reading each item from the original file and recording it, with the changes, into the new file
 - The old file is then erased, and the new file is renamed with the name of the original file
- To gain access to the functions needed for these tasks, we must first import the standard library module `os` with the statement

```
import os
```

and then delete the specified file using the statement

```
os.remove(fileName)
```

Altering Items in a Text File

- The statement

```
os.rename(oldFileName, newFileName)
```

will change the name and possibly the path of a file

- The **remove** and **rename** functions **cannot be used with open files**
 - The second argument of the rename function cannot be the name of an existing file
 - An error message is generated if the file to be removed, renamed, or opened for reading does not exist
- To verify if a file exists before attempting to rename, delete, or read it, we can use

```
os.path.isfile(fileName)
```

that returns **True** if the specified file exists and **False** otherwise

Altering Items in a Text File

- Assume that the current folder does not contain a file named `ABC.txt` when the following program is run

```
import os.path

if os.path.isfile("ABC.txt"):
    print("File already exists.")
else:
    infile = open("ABC.txt", 'w')
    infile.write("a\nb\nc\n")
    infile.close()
```

- What happens the first time the program is run?
- What happens the second time the program is run?

Sets

- A set is an **unordered** collection of items (referred to as elements) with **no duplicates**
 - Sets can contain numbers, strings, tuples, and Boolean values
 - Some examples of sets are
 - `{ 'spam', 'ni' }, { 3, 4, 7 },`
 - `{ True, 'eleven', 7 }, { 'a', 'b', (3, 4) }`
 - Sets **cannot contain lists or other sets**
 - Since the elements have no order, they **cannot be indexed**
 - Slicing and list methods such as `sort` and `reverse` are meaningless
- Useful set operations include **membership** test, **subset** test, **set intersection**, and so on

Sets

```
>>> bri = {'brazil', 'russia', 'india'}
>>> 'india' in bri
True
>>> 'usa' in bri
False
>>> bric = bri.copy()
>>> bric.add('china')
>>> bri.remove('russia') # or bri.discard('russia')
>>> bri
{'brazil', 'india'}
>>> bric
{'brazil', 'russia', 'india', 'china'}
>>> bric.issuperset(bri) # or bri.issubset(bric)
True
>>> bri & bric # or bri.intersection(bric)
{'brazil', 'india'}
>>> bric.difference(bri) # or bric - bri
{'russia', 'china'}
>>> bric.union(bri) # or bric | bri
{'brazil', 'india', 'china', 'russia'}
```

Sets

- List, tuple, and set can be converted to one another by using the functions `list`, `tuple`, and `set`

```
>>> words = ['nudge', 'nudge', 'wink', 'wink']
>>> tuple(words)
('nudge', 'nudge', 'wink', 'wink')
>>> terms = set(words)
>>> print(terms)
{'wink', 'nudge'}
>>> list(terms)
['wink', 'nudge']
>>> tuple(terms)
('wink', 'nudge')
>>> alpha = ('a', 'b', 'c')
>>> set(alpha)
{'a', 'c', 'b'}
>>> terms.clear()    # clear is a set method
>>> terms
set()
```

Sets

- Although the elements of a set cannot be ordered, they can be placed into a list in a customized order using the `sorted` function

```
>>> bric = {'brazil', 'china', 'india', 'russia'}
>>> bric
{'china', 'india', 'brazil', 'russia'}
>>> sorted(bric)
['brazil', 'china', 'india', 'russia']
>>> sorted(bric, key=len, reverse=True)
['brazil', 'russia', 'china', 'india']
>>> bric
{'china', 'india', 'brazil', 'russia'}
```

- Like lists, sets can be created with `comprehension`

```
>>> {x * x for x in range(-3, 3)}
{0, 9, 4, 1}
```

Sets

- The following is a rewrite of the example program on [p.11](#) using set methods to create a file containing the names of presidents who also served as vice president

```
def main():
    ## Create a file of the presidents who also served
    # as vice-presidents.
    vicePresSet = createSetFromFile("VPres.txt")
    presSet = createSetFromFile("USPres.txt")
    bothPresAndVPresSet = createIntersection(vicePresSet,
                                              presSet)
    writeNamesToFile(bothPresAndVPresSet, "PresAndVPres.txt")

def createSetFromFile(fileName):
    # Assume that the last line of the file ends with
    # a newline character.
    infile = open(fileName, 'r')
    namesSet = {name for name in infile}
    infile.close()
    return namesSet
```

Sets

```
def createIntersection(set1, set2):  
    return set1.intersection(set2)  
  
def writeNamesToFile(setName, fileName):  
    outfile = open(fileName, 'w')  
    outfile.writelines(setName)  
    outfile.close()  
  
main()
```

- Set operations (`words = {'spam', 'ni'}`)

Methods and Functions	Example	Value of Set	Description
add	<code>words.add("eggs")</code>	<code>{"spam", "ni", "eggs"}</code>	adds item to set
discard	<code>words.discard{"ni"}</code>	<code>{"spam"}</code>	removes specified item
clear	<code>words.clear()</code>	<code>{}</code>	<code>{}</code> is the empty set
set	<code>set([3, 7, 3])</code>	<code>{3, 7}</code>	convert a list to a set
	<code>set((3, 7, 3))</code>	<code>{3, 7}</code>	convert a tuple to a set

Sets

```
>>> {1, 2, 3} | {3, 4}      # set union
{1, 2, 3, 4}
>>> {1, 2, 3} & {3, 4}      # set intersection
{3}
>>> {1, 2, 3} - {3, 4}      # set difference
{1, 2}
>>> {1, 2, 3} ^ {3, 4}      # symmetric difference
{1, 2, 4}
>>> 3 in {1, 2, 3}          # is an element of
True
>>> x = {1, 2, 3} - {3, 4}    # x = {1, 2}
>>> x.add(5)                  # x = {1, 2, 5}
>>> y = x.copy()              # y is a copy of x
>>> y.discard(1)
>>> y
{2, 5}
>>> x
{1, 2, 5}
```


CSV Files

- Text files considered so far had a single piece of data per line
- Consider CSV (comma separated values) formatted file
 - Several items of data on each line
 - Items separated by commas
- The file `un.txt` contains the members of UN
 - Countries listed in alphabetical order
 - Each record contains data about a country: name, continent, population (in million), land area (in square miles)

```
Canada,North America,34.8,3855000
France,Europe,66.3,211209
New Zealand,Australia/Oceania,4.4,103738
Nigeria,Africa,177.2,356669
Pakistan,Asia,196.2,310430
Peru,South America,30.1,496226
```

Accessing the Data in a CSV File

- The `split` method is used to access the fields

```
def main():
    ## Display the countries in a specified continent.
    continent = input("Enter the name of a continent: ")
    continent = continent.title()    # Allow for all lower
    if continent != "Antarctica":   # case letters.
        infile = open("UN.txt", 'r')
        for line in infile:
            data = line.split(',')
            if data[1] == continent:
                print(data[0])
        infile.close()
    else:
        print("There are no countries in Antarctica.")

main()
```

Accessing the Data in a CSV File

[Run]

Enter the name of a continent: **South America**

Argentina

Bolivia

Brazil

Chile

Colombia

Ecuador

Guyana

Paraguay

Peru

Suriname

Uruguay

Venezuela

Analyzing the Data in a CSV File with a List

- Data can be analyzed by placing data into a list
 - Items of the list are other lists holding the contents of a single line of the file

```
def main():  
    ## Create a file containing all countries and areas,  
    ## ordered by area.  
    ## Display first five lines of the file.  
    countries = placeRecordsIntoList("UN.txt")  
    countries.sort(key=lambda country: country[3],  
                  reverse=True) # sort by area  
    displayFiveLargestCountries(countries)  
    createNewFile(countries) # Create file of countries and  
                             # their areas.
```

Analyzing the Data in a CSV File with a List

```
def placeRecordsIntoList(fileName):
    infile = open(fileName, 'r')
    listOfRecords = [line.rstrip() for line in infile]
    infile.close()
    for i in range(len(listOfRecords)):
        listOfRecords[i] = listOfRecords[i].split(',')
        listOfRecords[i][2] = eval(listOfRecords[i][2])
                                # population
        listOfRecords[i][3] = eval(listOfRecords[i][3])
                                # area
    return listOfRecords

def displayFiveLargestCountries(countries):
    print("{0:20}{1:9}".format("Country", "Area (sq. mi.)"))
    for i in range(5):
        print("{0:20}{1:9,d}".format(countries[i][0],
                                     countries[i][3]))
```

Analyzing the Data in a CSV File with a List

```
def createNewFile(countries):  
    outfile = open("UNbyArea.txt", 'w')  
    for country in countries:  
        outfile.write(country[0] + ',' + str(country[3])  
                        + "\n")  
    outfile.close()
```

main()

[Run]

Country	Area (sq. mi.)
Russian Federation	6,592,800
Canada	3,855,000
United States	3,794,066
China	3,696,100
Brazil	3,287,597

Analyzing the Data in a CSV File with a List

- The first three lines of the CSV file `UNbyArea.txt` are

```
Russian Federation,6592800  
Canada,3855000  
United States,3794066
```

- CSV files can be converted to Excel spreadsheets
 - Open `UN.txt` file in Excel, select comma when asked for delimiter
- Spreadsheets can be converted to CSV files
 - Click on “Save As” from the FILE menu, choose “CSV (Comma delimited)” in the “Save as type” dropdown box

Dictionary

- A dictionary is a collection of comma-separated pairs of the form

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

- The keys must be unique immutable objects (such as strings, numbers, or tuples)
- The value associated with `key1` is given by the expression `d[key1]`
- The `dict` function converts a list of two-item lists or two-item tuples into a dictionary

```
>>> list1 = ["one", 1], ["two", 2], ["three", 3]
>>> dict(list1)
{'one': 1, 'two': 2, 'three': 3}
>>> list2 = ("one", 1), ("two", 2), ("three", 3)
>>> dict(list2)
{'one': 1, 'two': 2, 'three': 3}
```


Dictionary

```
addr = { 'Swaroop'      : 'swaroop@swaroopch.com',
         'Larry'       : 'larry@wall.org',
         'Matsumoto'    : 'matz@ruby-lang.org',
         'Spammer'     : 'spammer@hotmail.com'
        }
print("Swaroop's address is", addr['Swaroop'])

# Deleting a key-value pair
del addr['Spammer']
print('\nThere are {} contacts in the address-book\n' \
      .format(len(addr)))

for name, address in list(addr.items()):
    print('Contact {} at {}'.format(name, address))

# Adding a key-value pair
addr['Guido'] = 'guido@python.org'
if 'Guido' in addr:
    print("\nGuido's address is", addr['Guido'])
```

Dictionary

[Run]

```
Swaroop's address is swaroop@swaroopch.com
```

```
There are 3 contacts in the address-book
```

```
Contact Swaroop at swaroop@swaroopch.com
```

```
Contact Matsumoto at matz@ruby-lang.org
```

```
Contact Larry at larry@wall.org
```

```
Guido's address is guido@python.org
```

- The operation `list(d.items())` returns a list of tuples where each tuple contains a key value pair
- New key-value pairs can be added by simply using the indexing operator to access a key and assign its value

Dictionary

- Dictionary operations:

Operation	Description
<code>len(d)</code>	number of items (that is <i>key:value</i> pairs) in the dictionary
<code>x in d</code>	has value <code>True</code> if <i>x</i> is a key of the dictionary
<code>d[key1] = value1</code>	if <i>key1</i> is already a key in the dictionary, changes the value associated with <i>key1</i> to <i>value1</i> ; otherwise, adds the item <i>key1:value1</i> to the dictionary.
<code>d[key1]</code>	returns the value associated with <i>key1</i> . Raises an error if <i>key1</i> is not a key of <i>d</i> .
<code>d.get(key1, default)</code>	if <i>key1</i> is not a key of the dictionary, returns the default value. Otherwise, returns the value associated with <i>key1</i> .
<code>list(d.keys())</code>	returns a list of the keys in the dictionary.
<code>list(d.values())</code>	returns a list of the values in the dictionary.
<code>list(d.items())</code>	returns a list of two-tuples of the form (<i>key</i> , <i>value</i>) where <i>d(key) = value</i> .
<code>list(d)</code>	returns a list of the keys in the dictionary.
<code>tuple(d)</code>	returns a tuple of the keys in the dictionary.
<code>set(d)</code>	returns a set of the keys in the dictionary.

Dictionary

- Dictionary operations:

<code>set(d)</code>	returns a set of the keys in the dictionary.
<code>c = {}</code>	creates an empty dictionary.
<code>c = dict(d)</code>	creates a copy of the dictionary d.
<code>del d[key1]</code>	removes the item having <i>key1</i> as key Raises an exception if <i>key1</i> is not found
<code>d.clear()</code>	removes all items (that is <i>key:value</i> pairs) from the dictionary.
<code>for k in d:</code>	iterates over all the keys in the dictionary.
<code>d.update(c)</code>	merges all of dictionary <i>c</i> 's entries into dictionary <i>d</i> . If two items have the same key, the value from <i>c</i> replaces the value from <i>d</i> .
<code>max(d)</code>	largest value of <code>d.keys()</code> , provided all keys have the same data type.
<code>min(d)</code>	smallest value of <code>d.keys()</code> , provided all keys have the same data type.

Using a Dictionary to Simplify a Long if-elif

```
def main():
    ## Determine an admission fee based on age group.
    print("Enter the person's age group ", end="")
    ageGroup = input("(child, minor, adult, or senior): ")
    print("The admission fee is",
          determineAdmissionFee(ageGroup), "dollars." )

def determineAdmissionFee(ageGroup):
    if ageGroup == "child":      # age < 6
        return 0                # free
    elif ageGroup == "minor":    # age 6 to 17
        return 5                # $5
    elif ageGroup == "adult":    # age 18 to 64
        return 10
    elif ageGroup == "senior":   # age >= 65
        return 8

main()
```

Using a Dictionary to Simplify a Long if-elif

- The rewrite of `determineAdmissionFee` function below, replaces the `if-elif` statement with a dictionary

```
def main():
    ## Determine an admission fee based on age group.
    print("Enter the person's age group ", end="")
    ageGroup = input("(child, minor, adult, or senior): ")
    print("The admission fee is",
          determineAdmissionFee(ageGroup), "dollars." )

def determineAdmissionFee(ageGroup):
    dict = {"child":0, "minor":5, "adult":10, "senior":8}
    return dict[ageGroup]
```

```
main()
```

```
[Run]
```

```
Enter the person's age group (child, minor, adult, or senior):
adult
The admission fee is 10 dollars.
```

Using a Dictionary as a Frequency Table

```
def main():
    ## Analyze word frequencies in the Gettysburg Address,
    ## which is written in a single line.
    listOfWords = formListOfWords("Gettysburg.txt")
    freq = createFrequencyDictionary(listOfWords)
    displayWordCount(listOfWords, freq)
    displayMostCommonWords(freq)

def formListOfWords(fileName):
    infile = open(fileName)
    originalLine = infile.readline().lower()
    infile.close()
    # Remove punctuation marks from the line.
    line = ""
    for ch in originalLine:
        if ('a' <= ch <= 'z') or (ch == " "):
            line += ch
    # Place the individual words into a list.
    listOfWords = line.split()
    return listOfWords
```

Using a Dictionary as a Frequency Table

```
def createFrequencyDictionary(listOfWords):  
    ## Create dictionary with each item having the form  
    ## word:word frequency.  
    freq = {}    # an empty dictionary  
    for word in listOfWords:  
        freq[word] = 0  
    for word in listOfWords:  
        freq[word] = freq[word] + 1  
    return freq  
  
def displayWordCount(listOfWords, freq):  
    print("The Gettysburg Address contains", len(listOfWords),  
          "words.")  
    print("The Gettysburg Address contains", len(freq),  
          "different words.")  
    print()
```


Using a Dictionary as a Frequency Table

```
def displayMostCommonWords(freq):  
    ## Common words are those with frequency > 5.  
    print("The most common words and their frequencies are:")  
    listOfMostCommonWords = []  
    for word in freq.keys():  
        if freq[word] >= 6:  
            listOfMostCommonWords.append((word, freq[word]))  
    listOfMostCommonWords.sort(key=lambda x: x[1],  
                                reverse=True)  
    for item in listOfMostCommonWords:  
        print("    ", item[0] + ': ', item[1])  
  
main()
```

Using a Dictionary as a Frequency Table

[Run]

The Gettysburg Address contains 268 words.

The Gettysburg Address contains 139 different words.

The most common words and their frequencies are:

that: 13

the: 11

we: 10

to: 8

here: 8

a: 7

and: 6

Storing Dictionaries in Binary Files

- Methods that store dictionaries to, and retrieve dictionaries from binary files must be imported from a module named pickle
 - Binary format can only be accessed by special readers

```
import pickle
```

```
outfile = open(filename, 'wb')
```

```
pickle.dump(dictionaryName, outfile)
```

```
outfile.close()
```

```
infile = open(filename, 'rb')
```

```
dictionaryName = pickle.load(infile)
```

```
infile.close()
```

- The extension for binary files is “dat” (e.g., “UNdict.dat”)

Dictionary-Valued Dictionaries

- Dictionary's values can be any type of object including a dictionary
- Consider the dictionary

```
nations = {"Canada":{"cont":"North America",  
                    "popl":34.8, "area":3855000},  
           "France":{"cont":"Europe", "popl":66.3,  
                    "area":211209},  
           ...}
```

- The value of `nations["Canada"]` would be the dictionary
`{"cont":"Europe", "popl":66.3, "area":211209}`
- The value of `nations["France"]["cont"]` would be `Europe`

Dictionary-Valued Dictionaries

```
import pickle

def main():
    ## Display countries (and their population) from
    ## a specified continent.
    nations = getDictionary("UNdict.dat")
    print("Enter the name of a continent", end='')
    continent = input("other than Antarctica: ")
    continentDict = constructContinentNations(nations,
                                              continent)

    displaySortedResults(continentDict)

def getDictionary(fileName):
    infile = open(fileName, 'rb')
    countries = pickle.load(infile)
    infile.close()
    return countries
```

Dictionary-Valued Dictionaries

```
def constructContinentNations(nations, continent):
    ## Reduce the full 193 item dictionary to a dictionary
    ## consisting solely of the countries in the specified
    ## continent.
    continentDict = {} # an empty dictionary
    for nation in nations: # or nations.keys()
        if nations[nation]["cont"] == continent:
            continentDict[nation] = nations[nation]
    return continentDict

def displaySortedResults(dictionaryName):
    ## Display countries in descending order by population.
    continentList = sorted(dictionaryName.items(),
                           key=lambda k: k[1]["popl"], reverse=True)
    for k in continentList:
        print(" {0:s}: {1:,.2f}".format(k[0], k[1]["popl"]))

main()
```

Dictionary-Valued Dictionaries

[Run]

```
Enter the name of a continent other than Antarctica: Europe
  Russian Federation: 142.50
  Germany: 81.00
  United Kingdom: 66.70
  France: 66.30
  Italy: 61.70
  Spain: 47.70
  . . .
```

Using a Dictionary with Tuples as Keys

- USpresStatesDict.dat holds a dictionary whose items look like

`('Kennedy', 'John'): 'Massachusetts'`

`('Reagan', 'Ronald'): 'California'`

```
import pickle

def main():
    ## Displays the presidents from the given state ordered
    ## alphabetically by their last names.
    presDict = \
        createDictFromBinaryFile("USpresStatesDict.dat")
    state = getState(presDict)
    displayOutput(state, presDict)

def createDictFromBinaryFile(fileName):
    infile = open(fileName, 'rb')
    dictionary = pickle.load(infile)
    infile.close()
    return dictionary
```


Using a Dictionary with Tuples as Keys

```
def getState(dictName):
    state = input("Enter the name of a state: ")
    if state in dictName.values():
        return state
    else:
        return "There are no presidents from " + state + '.'

def displayOutput(state, dictName):
    if state.startswith("There"):
        print(state)
    else:
        print("Presidents from", state + ':')
        for pres in sorted(dictName): # in sorted list of names
            if dictName[pres] == state:
                print(" " + pres[1] + " " + pres[0])

main()
```

Using a Dictionary with Tuples as Keys

[Run]

Enter the name of a state: **Virginia**

Presidents from Virginia:

Thomas Jefferson

James Madison

James Monroe

John Tyler

George Washington

Dictionary Comprehension

- Dictionaries can be created with dictionary comprehension, e.g.,
`{x: x * x for x in range(4)}`
- Dictionary comprehension can be used to extract a subset of a dictionary, e.g.,

```
NE = ["Maine", "Connecticut", "New Hampshire",  
      "Massachusetts", "Vermont", "Rhode Island"]  
subSet = {key: presDict[key] for key in presDict  
          if presDict[key] in NE}
```