## COMP 141

CS1: Programming Fundamentals
November 24, 2014

### Announcements

**Reminder**
- Program 8 has been assigned – Due Dec. 9th by 11:55pm
- You may work with a partner on Program 8

---

### Dictionaries

- **Dictionary**: object that stores a collection of data
  - Create a dictionary
    ```python
    phonebook = {'Chris': '555-1111', 'Katie': '555-2222'}
    ```
  - Retrieving a value using the key
    ```python
    if 'Katie' in phonebook:
        print(phonebook['Katie'])
    ```
  - Adding elements to an existing dictionary
    ```python
    if 'JoAnne' not in phonebook:
        phonebook['JoAnne'] = '555-3333'
    ```
  - Deleting elements from a dictionary
    ```python
    if 'Chris' in phonebook:
        del phonebook['Chris']
    ```

---

### Getting the Number of Elements and Mixing Data Types

- **len function**: used to obtain number of elements in a dictionary
  ```python
  len(phonebook)
  ```
- Keys must be immutable objects, but associated values can be any type of object
  - One dictionary can include keys of several different immutable types
  - Values stored in a single dictionary can be of different types
Creating an Empty Dictionary and Using for Loop to Iterate Over a Dictionary

- **To create an empty dictionary:**
  - Use `{}`
  - Use built-in function `dict()`
  - Elements can be added to the dictionary as program executes
- **Use a for loop to iterate over a dictionary**
  - General format: `for key in dictionary:`

Some Dictionary Methods

- **get method:** gets a value associated with specified key from the dictionary
  - Format: `dictionary.get(key, default)`
  - `default` is returned if key is not found
  - Alternative to `[]` operator
  - Cannot raise KeyError exception
- **keys method:** returns all the dictionaries keys as a sequence
  - Format: `dictionary.keys()`

Some Dictionary Methods (cont’d.)

- **values method:** returns all the dictionaries values as a sequence
  - Format: `dictionary.values()`
  - Use a for loop to iterate over the values

```python
>>> wordDictionary = {'the': 16, 'a': 7, 'whose': 1, 'its': 3}
>>> wordDictionary.keys()
['the', 'whose', 'a', 'its']
>>> wordDictionary.get('the', 0)
16
>>> wordDictionary.get('later', 0)
0
>>> wordDictionary.values()
[1, 7, 1, 3]
>>> 
>>> for key in wordDictionary:
    print(key, wordDictionary[key])
its 3
whose 1
a 7
the 16
```
**Dictionary Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td>Clears the contents of a dictionary.</td>
</tr>
<tr>
<td>get</td>
<td>Gets the value associated with a specified key. If the key is not found, the method does not raise an exception. Instead, it returns a default value.</td>
</tr>
<tr>
<td>items</td>
<td>Returns all the keys in a dictionary and their associated values as a sequence of tuples.</td>
</tr>
<tr>
<td>keys</td>
<td>Returns all the keys in a dictionary as a sequence of tuples.</td>
</tr>
<tr>
<td>pop</td>
<td>Returns the value associated with a specified key and removes that key-value pair from the dictionary. If the key is not found, the method returns a default value.</td>
</tr>
<tr>
<td>popitem</td>
<td>Returns a randomly selected key-value pair as a tuple from the dictionary and removes that key-value pair from the dictionary.</td>
</tr>
<tr>
<td>values</td>
<td>Returns all the values in the dictionary as a sequence of tuples.</td>
</tr>
</tbody>
</table>

---

**Practice with Dictionaries As a Class**

- Write a program that opens the file text.txt (from Program 4) and counts the number of occurrences of each word.
- Print out each word and the number of times it occurs on a separate line.
- Loop through the dictionary to find the word that occurred the most – print that out.
- Hints: You will need to go through the file line by line and split the line – the split function returns a list – loop through that list of words and put them into your dictionary and keep track of how many times you saw each word.

```python
def main():
    inputString = input("Please enter a string of words: ")
    wordDictionary = {}

    inputString = inputString.lower()
    inputString = inputString.replace('!', ' ')
    inputString = inputString.replace('?', ' ')
    inputString = inputString.replace(',', ' ')
    inputString = inputString.replace(':', ' ')
    inputString = inputString.replace('-', ' ')
    words = inputString.split(' ')       

    for word in words:
        if word not in wordDictionary:
            wordDictionary[word] = 1
        else:
            wordDictionary[word] += 1

    for key in wordDictionary:
        print(key, wordDictionary[key])

main()
```
Caesar Cipher

- Type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet.
- Named after Julius Caesar, who used it in his private correspondence.

Example of a Caesar Cipher

Plain: ABCDEFGHIJKLMNOPQRSTUVWXYZ Cipher: XYZABCDEFGHIJKLMNOPQRSTUVWXYZS

- Transformation can be represented by aligning two alphabets.
- This one uses a left rotation of three places, equivalent to a right shift of 23

Encrypting a Message

Ciphertext: QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD
Plain: the quick brown fox jumps over the lazy dog

- Look up each letter of the message in the "plain" line and write down the corresponding letter in the "cipher" line.
- The encryption can also be represented using modular arithmetic
  - Transform the letters into numbers, according to the scheme, A = 0, B = 1, ..., Z = 25.
  - Encryption of a letter by a shift \( n \) can be described mathematically as,
    \[ E_n(x) = (x + n) \mod 26 \]

Encrypting

```python
def main():
    plainText = "the quick brown fox jumps over the lazy dog"
    cipherText = encode(plainText, 3)
    print("CipherText after encode call: ", cipherText)
    def encode(text, s):
        text = text.lower()
        alphabet = "abcdefghijklmnopqrstuvwxyz"
        cipher = ""
        for ch in text:
            if ch in alphabet:
                cipher += alphabet[(alphabet.index(ch) + s) % 26]
            else:
                cipher += ch
        return cipher
main()
```

Program Output

CipherText after encode call: wkh txlfn eurzq ira mpsv ryhu wkh oduh grj
Decryption of a Message

- Opposite process of encryption
  - Look up each letter of the message in the "cipher" line and write down the corresponding letter in the "plain" line.
- Can also use modular arithmetic to solve $D_n(x) = (x - n) \mod 26$

Decryption

```python
def main():
    cipherText = "wkh hark wrxq ire mavq syh wkh odwh qrj"
    plaintext = decode(cipherText, 3)
    print("PlainText after decode call: ", plaintext)

def decode(text, n):
    text = text.lower()
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    cipher = ""
    for ch in text:
        if ch in alphabet:
            cipher += alphabet[(alphabet.index(ch) - n) % 26]
        else:
            cipher += ch
    return cipher

main()
```

Program Output

PlainText after decode call: the quick brown fox jumps over the lazy dog

Breaking the Cipher

- If you were not the intended recipient of the message, you wouldn't be told the shift (n).
- To determine what n is:
  - Analyze the frequency of each character in the encrypted message
  - Since 'E' is the most common letter used in English, you should find the shift from E to the most common letter in the message

Practice

- Write a Caesar cipher program to decode the message in cipherText.txt (on website and in Public folder).
- You will need to read in the file (you can read it in all as 1 line if you'd like).
- Get the frequency count for each letter (don't include spaces).
- Find the shift between the max letter and 'e'
- Use the shift to decode the message