Dictionaries

Dictionary Basics

Dictionaries

{ key : value }

Exercise 1
Suppose dictionary `schedule` represents an employee's work schedule where the keys are a day of the week and the values are the number of hours scheduled that day. Write Processing code to:

(a) Create `schedule` such that the employee works 8 hours on Monday, 7.5 hours on Tuesday, 8 hours on Thursday, and 7.5 hours on Friday.

(b) Oops! Someone made a scheduling error. Update `schedule` so that the employee works Friday's hours on Wednesday instead. Remove Friday from the work schedule.

Solution

```
# CMPT 140 - Dictionaries
# Topic(s): Dictionary Basics


# Part (a)
schedule = {"Monday": 8, "Tuesday": 7.5, "Thursday": 8, "Friday": 7.5}

# Part (b)
schedule["Wednesday"] = schedule["Friday"]
del schedule["Friday"]
```

This code can be found in `cmpt140-ch16-py/cmpt140_ch16_work_schedule/cmpt140_ch16_work_schedule.pyde`

Exercise 2
Suppose we have dictionary `assets`

keys: the name of a website asset

values: the file name of that asset

Write Processing code which prepends the string `image/` to every file name that ends with `.png` or `.jpg`. (you can assume all file names have exactly one period)

Example `assets` input:

```
assets = {  "banner"    :  "banner.png",
            "homepage"  :  "index.html" }
```

Expected output:

```
assets = {  "banner"    :  "images/banner.png",
            "homepage"  :  "index.html" }
```

Solution

```
# CMPT 140 - Dictionaries
# Topic(s): Dictionary Basics


# for every asset, check if it's an image, if so, add directory name
for asset in assets:
    if ".png" in assets[asset] or ".jpg" in assets[asset]:
        assets[asset] = "images/" + assets[asset]
```

This code can be found in `cmpt140-ch16-py/cmpt140_ch16_update_assets/cmpt140_ch16_update_assets.pyde`

Exercise 3

We have a dictionary called `poll`, representing employee feedback data:

keys: employee usernames

values: one of three possible string values: "Computers", "Seating" or "Support"

Write function `poll_results` which takes dictionary parameter `poll` and **returns** a new dictionary whose keys are "Computers", "Seating", and "Support" and values are the number of employees who requested that improvement. Only create the keys in the new dictionary if at least one employee requested it!

Example `poll` input:

```
poll = {  "ab"   :  "Computers",
          "cc"   :  "Support",
          "bar"  :  "Computers" }
```

Expected output:

```
{  "Computers"  :  2,
   "Support"    :  1 }
```

Solution

```
# CMPT 140 - Dictionaries
# Topic(s): Dictionary Basics


def poll_results( poll ):
    """
    return dictionary of improvments mapped to the number of requests
    poll: dictionary of employee usernames to their improvment request
    return: dictionary mapping improvements to number of employees
    """
    results = {}

    # count number of employees who voted for each improvement
    for user in poll:
        # no one voted for improvement yet, so start count
        if poll[user] not in results:
            results[ poll[user] ] = 1
        # increment tally count by one
        else:
            results[ poll[user] ] = results[ poll[user] ] + 1

    return results
```

This code can be found in `cmpt140-ch16-py/cmpt140_ch16_poll_results/cmpt140_ch16_poll_results.pyde`

Using Dictionaries

Exercise 4
Is the following dictionary an example of a mapping or record? Explain why.

```
song = {  "artist"  :  "Elvis Presley",
          "title"   :  "Jailhouse Rock",
          "length"  :  146 }
```

Answer: Dictionary `song` is a record because it represents a collection of related information regarding one item as opposed to a collection of many independent items and their translations into other data.

Ciphers

In cryptography, messages are encoded and decoded using ciphers to transfer private information between two parties. Ciphers are legends that denote what character should be substituted by another in an encoded message in order to obtain the original (decoded) message.

Example encoded message and a **decoding** cipher:

```
msg = "SVJJT"
cipher = { "J":"L", "S":"H", "T":"O", "V":"E" }
```

Message decoded using cipher:

```
decoded_msg = "HELLO"
```

Exercise 5
Write Processing function `load_cipher` with parameter `fname` representing a filename that loads an **encoding** cipher from the file into a dictionary and then returns it. Each line in the file holds the original character followed by the encoding character.

Example cipher file data:

```
A,Z,
B,W,
  ⋮
```

Expected return data:

```
{ "A":"Z", "B":"W", ... }
```

Solution

```
def load_cipher( fname ):
    """
    reads cipher from file into dictionary and returns that
    fname: filename containing the cipher
    return: dictionary of cipher loaded from file
    """
    cipher = {}  # to contain cipher from file

    f = open(fname,"r")
    for line in f:
        c_pair = line.rstrip("\n")
        c_pair = c_pair.split(",")
        cipher[c_pair[0]] = c_pair[1]
    f.close()
    return cipher
```

This code can be found in `cmpt140-ch16-py/cmpt140_ch16_cipher/cmpt140_ch16_cipher.pyde`

Exercise 6
Write Processing function `invert_cipher` which takes an **encoding cipher** (i.e. that maps original characters to their encoded character) and returns a **decoding cipher** (i.e. that maps encoded characters to original characters).

Example cipher input:

```
{ "A":"Z", "B":"W", ... }
```

Expected return data:

```
{ "Z":"A", "W":"B", ... }
```

Solution

```
def invert_cipher( cipher ):
    """
    returns inverted cipher where keys becomes values & vice versa
    cipher: cipher to invert/swap key-values of
    return: cipher whose key-values have been inverted
    """
    inv_cipher = cipher.copy()
    for k in cipher:
        inv_cipher[cipher[k]] = k
    return inv_cipher
```

This code can be found in `cmpt140-ch16-py/cmpt140_ch16_cipher/cmpt140_ch16_cipher.pyde`

Exercise 7

Suppose we receive a secret message that we want to decode. Write Processing function `decode` which takes string `encoded_msg` and a **decoding** `cipher` and returns the decoded message. Characters unknown to the cipher remain the same in both messages.

Example encoded message and cipher input:

```
encoded_msg = "LOLOL!"
cipher = { "L":"H", "O":"A", ... }
```

Expected decoded message as output:

```
"HAHAH!"
```

Solution

```
def decode( encoded_msg,cipher ):
    """
    decodes the encoded message using the cipher and returns it
    encoded_msg: message to decode with cipher
    cipher: legend for decoding the message (keys are encoded characters)
    return: the decoded message
    """
    decoded_msg = ""  # to contain decoded message

    for em in encoded_msg:
        if em in cipher:
            decoded_msg = decoded_msg + cipher[em]
        else:
            decoded_msg = decoded_msg + em

    return decoded_msg
```

This code can be found in `cmpt140-ch16-py/cmpt140_ch16_cipher/cmpt140_ch16_cipher.pyde`

Exercise 8

Write Processing function `probability_same_animal()` which takes two records `rec_a` and `rec_b` and returns the percentage of traits that are identical to both animals. Assume that `rec_a` and `rec_b` have the same kind of traits recorded (i.e. the dictionaries have the same keys).

Example record inputs:

```
rec1 = {    "markings"  :  "scar on left hind leg",
             "species"  :  "red panda",
          "birth_year"  :  2014,
              "gender"  :  "female" }
rec2 = {    "markings"  :  "scar on right hind leg",
             "species"  :  "red panda",
          "birth_year"  :  2014,
              "gender"  :  "female" }
```

Expected output:

```
75.0
```

Solution

```
# CMPT 140 - Dictionaries
# Topic(s): Using Dictionaries (Records)


def probability_same_animal( rec_a,rec_b ):
    """
    returns probability between 0 and 100 that both records represent
    the same animal based on number of matching traits.

    rec_a: record containing information about an animal
    rec_b: other record containing information about an animal
    return: probability that both records represent the same animal
    """

    # sum the number of matching traits between both animal records
    n_matches = 0  # number of matching traits
    for trait in rec_a:
        if (rec_a[trait] == rec_b[trait]):
            n_matches = n_matches + 1

    # compute and return probability of records representing same animal
    prob = float(n_matches)/len(rec_a) * 100.0

    return prob
```

This code can be found in `cmpt140-ch16-py/cmpt140_ch16_records/cmpt140_ch16_records.pyde`