Dictionary Manipulations

Thomas Schwarz

Dictionary Creation

- Creating a dictionary using assignments

```
scrabble = {}
scrabble['a'] = 1
scrabble['b'] = 3
scrabble['c'] = 3
scrabble['d'] = 2
…
```

Dictionary Creation

- Creating a dictionary with pre-inserted values

```
scrabble = {'a':1, 'b'=3, 'c'=3, 'd'=2}
```

Dictionary Creation

- Creating a dictionary with dict and key-value pairs

```
scrabble = dict(a=1, b=3, c=3, d=2, e=1)
```

Dictionary Creation

- Creating a dictionary with dict and a list of key-value pairs

```
scrabble = dict( [('a',1), ('b',3), ('c',3), ('d',2), ('e',1)] )
```

Dictionary

- Checking for existence
  - Use the “in” keyword

```
>>> dicc = {1: "uno", 2: "dos", 3: "tres"}
>>> 1 in dicc
True
>>> 4 not in dicc
True
```

Dictionaries

- A simple program that “learns” Spanish words

```
def test():
    dicc = {}
    while True:
        astr = input("Enter an English word: ")
        if astr == "Stop it":
            return
        elif astr in dicc:
            print(dicc[astr])
        else:
            print("I have not yet learned this word")
            val = input("Please enter the Spanish word: ")
            dicc[astr] = val
```

Dictionaries

- Dictionaries have an internal structure
  - You will learn in Data Structures how to build dictionaries yourselves
  - For the moment, enjoy their power
- You can print dictionaries
  - You will notice that they change structure after inserts and not reflect the order in which you inserted elements
  - This is because they optimize access

Dictionaries

- Dictionaries have an internal structure
  - It uses *hashing* in order to assign locations internally
    - A hash is a long unsigned integer

```
hash('hello')
9713787544098718l8
hash(123456)
123456
hash(123.456)
10514644122014516l43
```

Dictionaries

- Deleting all entries in a dictionary
  - use the `clear()` method
- Deleting an entry without fear of creating a key error
  - Use an if statement
  - Use pop with a second argument None
    - `dicc.pop(1, None)`

Dictionaries

- Looping over keys
  - Simplest:
    - `for number in dicc:`
  - `iterkeys( )` or `iter` works the same way
    - `for number in dicc.iterkeys():`
    - `for number in iter(dicc):`

Multi-Dictionaries

- Problem:
  - Instead of associating one value with a key, we want to associate several values:
    - a “multi-dictionary”
- Solution:
  - The values of the dictionaries should be lists (or sets — coming week)

Multi-Dictionaries

- Example:
  - We want to pass through a file and create an index of important words with their occurrences

```
with open("alice.txt", encoding = "latin-1") as infile:
    dicc = {}
    word_number = 0
    for line in infile:
        for word in line.split():
            word = word.strip(":,.?![]'")
            word = word.lower()
            word_number +=1
            if len(word)>8:
                if word in dicc:
                    dicc[word].append(word_number)
                else:
                    dicc[word]=[word_number]
```

Calculating on Values

- Assume you have a dictionary with numerical values
  - For example: a dictionary with the prices of stocks on September 15, 2018

```
dstocks = {“tata”: 2063.30,
           “hdfc”: 2029.20,
           “hiul”: 1630.15,
            …
          }
```

- You want the average, the maximum, the minimum … price

Solution

- You can access the values of a dictionary through the values method.
  - values( ) returns an iterator of all the values in the dictionary

```
>>> dst = {"apple": 256.34, "fb": 145.23, "ibm": 98.34, "ms": 198.75}
>>> dst.values()
dict_values([256.34, 145.23, 98.34, 198.75])
>>> max(dst.values())
256.34
>>> sum(dst.values())/len(dst.values())
174.665
```

Calculating with keys

- Problem:
  - You want to calculate on the keys of a dictionary
- Solution:
  - The keys( ) method returns an iterator of the keys of a dictionary

Finding the most common item in a list

- We use a dictionary as a counter.
  - First way: We can do so by ourselves.
    - Create a dictionary
    - Pass through the list

```
def most_frequent(lista):
    counter = {}
    for x in lista:
        counter[x]=counter.get(x, 0)+1
```

get specifies a default value, it is otherwise equivalent to counter[x]

Finding the most common item in a list

- If we do not want to use get, we can just check whether the list-item is already in the dictionary

```
def most_frequent(lista):
counter = {}
for x in lista:
    if x in counter:
        counter[x]+=1
    else:
        counter[x]=1
```

Finding the most common item in a list

- After counting, we pass through the dictionary to find the maximum element.
  - Notice that we are interested in the key, not the value

```
def most_frequent(lista):
    counter = {}
    for x in lista:
        counter[x]=counter.get(x, 0)+1
    highest_seen = 0
    for x in counter:
        if counter[x]>highest_seen:
            best_key = x
            highest_seen = counter[x]
    return best_key
```

highest_seen contains the highest encountered **value**

Finding the most common item in a list

- After counting, we pass through the dictionary to find the maximum element.
  - Notice that we are interested in the key, not the value

```
def most_frequent(lista):
    counter = {}
    for x in lista:
        counter[x]=counter.get(x, 0)+1
    highest_seen = 0
    for x in counter:
        if counter[x]>highest_seen:
            best_key = x
            highest_seen = counter[x]
    return best_key
```

highest_seen is adjusted whenever we see a higher value in the counter

Finding the most common item in a list

- After counting, we pass through the dictionary to find the maximum element.
  - Notice that we are interested in the key, not the value

```
def most_frequent(lista):
    counter = {}
    for x in lista:
        counter[x]=counter.get(x, 0)+1
    highest_seen = 0
    for x in counter:
        if counter[x]>highest_seen:
            best_key = x
            highest_seen = counter[x]
    return best_key
```

but we also need to remember the key, which we record in best_key

Finding the most common item in a list

- After counting, we pass through the dictionary to find the maximum element.
  - Notice that we are interested in the key, not the value

```
def most_frequent(lista):
    counter = {}
    for x in lista:
        counter[x]=counter.get(x, 0)+1
    highest_seen = 0
    for x in counter:
        if counter[x]>highest_seen:
            best_key = x
            highest_seen = counter[x]
    return best_key
```

because the key with the highest counter value is the result that we return

Finding the most common item in a list

- But we can also use the work of others
  - The `Counter` in the collections module
    - You create a *new object* of type Counter

```
from collections import Counter

def most_frequent(lista):
    ctr = Counter()
```

Defines a new object called ctr ctr is an object of type Counter

Finding the most common item in a list

- Counters are (updated) like dictionaries
  - But they have a default value of 0

```
from collections import Counter


def most_frequent(lista):
    ctr = Counter()
    for item in lista:
        ctr[item] += 1
```

Here we add 1 to the value of `ctr[item]`

No need to initialize!

Finding the most common item in a list

- Counters have a method called `most_common`
  - Argument is the number of most common items
  - Returns a list of pairs

```
from collections import Counter

def most_frequent(lista):
    ctr = Counter()
    for item in lista:
        ctr[item] += 1
    return ctr.most_common(1)[0][0]
```

- Get a list of one elements.
- Get the first (and only) element of the list
- Get the first coordinate of that element