[220] Dictionaries

Meena Syamkumar
Mike Doescher

Cheaters caught: 0
(Through P4)
Learning Objectives Today

Data structures
• definition
• motivation

Dictionaries in Python
• creation, lookup
• updates, deletes

When to use dictionaries over lists
• holes in the labels
• non-integer labels

Chapter 11 of Think Python
Today's Outline

Data Structures

Mappings

Dictionaries

Mutations: Updates, Deletes, and Inserts

Coding examples
Vocabulary: a list is an example of a **data structure**
Data Structures

Definition (from Wikipedia):

a **data structure** is a **collection of data values**, the **relationships** among them, and the functions or **operations** that can be applied to the data.
Data Structures

Definition (from Wikipedia):

A **data structure** is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.

- Every value has an index, representing an order within the list.
- A list can contain a bunch of values of varying types.
- Functions and operations include `L.sort()`, `len(L)`, `L.pop(0)`, `L.append(x)`, update, iterate (for loop), etc.
## Data Structures

Definition (from Wikipedia):

A **data structure** is a *collection of data values*, the **relationships** among them, and the functions or **operations** that can be applied to the data.

<table>
<thead>
<tr>
<th></th>
<th>values</th>
<th>relationships</th>
<th>operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>list</strong></td>
<td>anything</td>
<td>ordered (0,1,...)</td>
<td>indexing, pop, len, index, slicing, in, iteration (for), ...</td>
</tr>
<tr>
<td><strong>set</strong></td>
<td>?? ??</td>
<td>no ordering</td>
<td>in, ==</td>
</tr>
<tr>
<td><strong>dict</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>...</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*suggested note-taking*
Motivation: lots of data

For loops:
• copy/paste is a pain
• don’t know how many times to copy/paste before program runs

For data structures:
• creating many variables is a pain
  (imagine your program analyzes ten thousand values)
• don’t know how many values you will have before program runs
Today's Outline

Data Structures

Mappings

Dictionaries

Mutations: Updates, Deletes, and Inserts

Coding examples
Mappings

Common data structure approach:
- store many values
- give each value a label
- use labels to lookup values
Mappings

Common data structure approach:
• store many values
• give each value a label
• use labels to lookup values

List example:

nums = [300, 200, 400, 100]

we can have many values
Mappings

Common data structure approach:
• store many values
• **give each value a label**
• use labels to lookup values

List example:

```
nums = [300, 200, 400, 100]
```

```
0 1 2 3
```

the “labels” are indexes, which are implicitly attached to values
Mappings

Common data structure approach:
• store many values
• give each value a label
• **use labels to lookup values**

List example:

```python
nums = [300, 200, 400, 100]

x = nums[2]  # x = 400
```

we use the “label” (i.e., the index) to lookup the value (here 400)
Mappings

Common data structure approach:

• store many values
• give each value a label
• use labels to lookup values

Lists are an **inflexible** mapping structure, because we don’t have control over **labels**

List example:

```python
nums = [300, 200, 400, 100]

x = nums[2]  # x=400
```

*what if we don’t want consecutive integers as labels? E.g., 0, 10, and 20 (but not between)?*

*what if we want to use strings as labels?*
Today's Outline

Data Structures

Mappings

**Dictionaries**

Mutations: Updates, Deletes, and Inserts

Coding examples
Why call it a dictionary?

this key
(the word)

break·fast
ˈbrekfəst/

maps to...

this value
(the definition)

noun
noun: breakfast; plural noun: breakfasts

1. a meal eaten in the morning, the first of the day.
   "I often have toast for my breakfast"

verb
verb: breakfast; 3rd person present: breakfasts; past tense: breakfasted; past participle: breakfasted;
gerund or present participle: breakfasting

1. eat breakfast.
   "she breakfasted on French toast and bacon"

Python dicts don't have order, though
Dictionary

Dictionaries map labels (called keys, rather than indexes) to values
- values can be anything we choose (as with lists)
- keys can be nearly anything we choose (must be immutable)

nums_list = [900, 700, 800]

nums_list[1] \rightarrow 700

a dictionary would let us give 700 a label other than its position
Dictionary

Dictionaries map labels (called keys, rather than indexes) to values

- values can be anything we choose (as with lists)
- keys can be nearly anything we choose (must be immutable)

```
nums_list = [900, 700, 800]
```

```
nums_list[1] → 700
```

```
nums_dict = {"first":900, "second":700, "third":800}
```

we have the same values
Dictionary

Dictionaries map labels (called keys, rather than indexes) to values
- values can be anything we choose (as with lists)
- keys can be nearly anything we choose (must be immutable)

```python
ums_list = [900, 700, 800]
nums_list[1] ➞ 700
```

```python
ums_dict = {"first":900, "second":700, "third":800}
```

we use **curly braces** instead of **square brackets**

careful! curly braces are for both sets and dicts
Dictionary

Dictionaries map labels (called keys, rather than indexes) to values
• values can be anything we choose (as with lists)
• keys can be nearly anything we choose (must be immutable)

```
nums_list = [900, 700, 800]

nums_list[1] → 700
```

```
nums_dict = {"first":900, "second":700, "third":800}
```

we choose the label (called a key) for each value. Here the keys are the strings “first”, “second”, and “third”

we put a colon between each key and value
Dictionary

Dictionaries map labels (called keys, rather than indexes) to values

- values can be anything we choose (as with lists)
- keys can be nearly anything we choose (must be immutable)

```
nums_list = [900, 700, 800]

nums_list[1] ➞ 700

nums_dict = {'first': 900, 'second': 700, 'third': 800}

nums_dict['second'] ➞ 700
```

lookup for a dict is like indexing for a list (label in brackets). Just use a key (that we chose) instead of an index.
Dictionary

Dictionaries map labels (called keys, rather than indexes) to values
• values can be anything we choose (as with lists)
• keys can be nearly anything we choose (must be immutable)

ums_list = [900, 700, 800]

tuple_list[1] \rightarrow 700

nums_dict = {"first":900, "second":700, "third":800}

tuple_dict["first"] \rightarrow 900

lookup for a dict is like indexing for a list (label in brackets). Just use a key (that we chose) instead of an index.
Dictionary

Dictionaries map labels (called keys, rather than indexes) to values

- values can be anything we choose (as with lists)
- keys can be nearly anything we choose (must be immutable)

```
nums_list = [900, 700, 800]

nums_list[1] → 700

nums_dict = {"first":900, "second":700, "third":800}

nums_dict["third"] → 800
```

lookup for a dict is like indexing for a list (label in brackets). Just use a key (that we chose) instead of an index.
Dictionary

Dictionaries map labels (called keys, rather than indexes) to values

- values can be anything we choose (as with lists)
- keys can be nearly anything we choose (must be immutable)

```python
nums_list = [900, 700, 800]
nums_list[1] ➔ 700

nums_dict = {"first":900, "second":700, "third":800}
nums_dict["third"] ➔ 800
```

(For Python <= 3.6)
A note on parenthetical characters

**common structures**

- **Parentheses:** `( and )
- **Brackets:** `[ and ]
- **Braces:** `{ and }

**Uses**

- *Specifying order:* $(1+2)\times3$
- *Function invocation:* $f()$
- *List creation:* $s = [1,2,3]$
- *Sequence indexing:* $s[-1]$
- *Sequence slicing:* $s[1:-2]$
- *Dictionary lookup:* $d["one"]$
- *Dictionary creation:* $d = \{"one":1, "two":2\}$
- *Set creation:* ${1,2,3}$
Today's Outline

Data Structures

Mappings

Dictionaries

Mutations: Updates, Deletes, and Inserts

Coding examples
Dictionary Updates

```python
>>> lst = ['zero', 'ten', 'not set']
>>> lst[2] = 'twenty'
>>> lst
['zero', 'ten', 'twenty']

>>> d = {0: 'zero', 10: 'ten', 20: 'not set'}
>>> d[20] = 'twenty'
>>> d
{0: 'zero', 20: 'twenty', 10: 'ten'}
```

dictionary updates look like list updates
Dictionary Deletes

```python
>>> lst = ["zero", "ten", "not set"]
>>> lst.pop(-1)
'not set'
>>> lst
['zero', 'ten']

>>> d = {0: "zero", 10: "ten", 20: "not set"}
>>> d.pop(20)
'not set'
>>> d
{0: 'zero', 10: 'ten'}
```

dictionary deletes look like list deletes

"not set" isn't in the list

"not set" isn't in the dict
Dictionary Inserts

```python
>>> lst = ['zero', 'ten']
>>> lst.append('twenty')  # doesn't work: lst[2] = ...
>>> lst
['zero', 'ten', 'twenty']

>>> d = {0: 'zero', 10: 'ten'}
>>> d[20] = 'twenty'
>>> d
{0: 'zero', 20: 'twenty', 10: 'ten'}
```

with a dict, if you try to set a value at a key, it automatically creates it (doesn't work w/ lists)
Today's Outline

Data Structures

Mappings

Dictionaries

Mutations: Updates, Deletes, and Inserts

Coding examples
Demo 1: Score Keeping App

Goal: let users enter scores for various players

**Input:**
- Commands: set score, lookup score, get highest

**Output:**
- The champion and their score

**Example:**

```
prompt> python scores.py
enter a cmd (type "help" for descriptions): set alice 10
enter a cmd (type "help" for descriptions): high
Alice: 10
enter a cmd (type "help" for descriptions): q
exiting
```
Demo 2: Print Tornados per Year

Goal: given a CSV of tornados, print how many occurred per year

Input:
• A CSV

Output:
• number per year

Example:

prompt> python tornados.py
...
2015: 9
2016: 2
2017: 4

https://en.wikipedia.org/wiki/Tornado
Demo 3: Wizard of Oz

Goal: count how often each word appears in the Wizard of Oz

**Input:**
- Plaintext of book (from Project Gutenberg)

**Output:**
- The count of each word