

# [301] Dictionaries

Tyler Caraza-Harter

# 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

Updates and Deletes

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

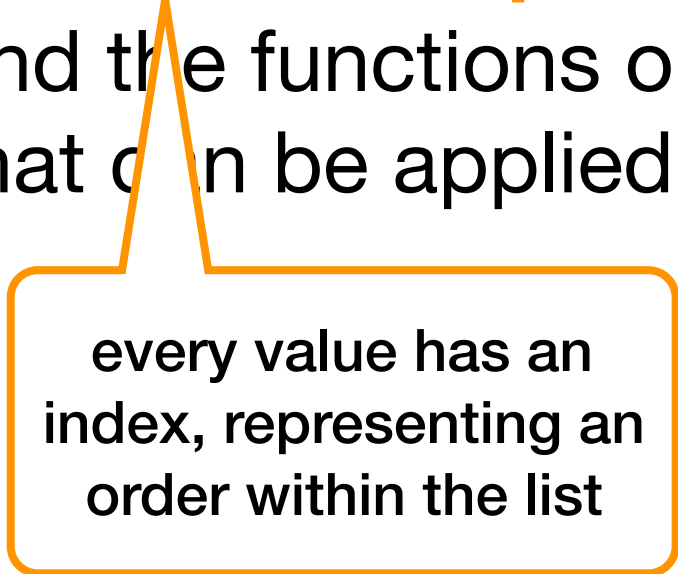


a list can contain a  
bunch of values of  
varying types


# 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

# 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

`L.sort()`, `len(L)`, `L.pop(0)`, `L.append(x)`,  
update, iterate (for loop), etc



Why do we need data structures to organize values?

Instead of just creating lots of variables?

# Motivation

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

Updates and Deletes

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]
```

# 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:

```
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:

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

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

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

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

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

```
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**

Updates and Deletes

Coding examples

# 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
```

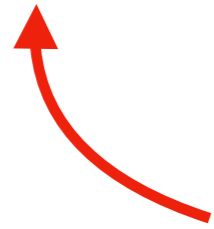
# Dictionary

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
```



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}
```

# 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)

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



```
nums_list[1] → 700
```

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



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

# 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 are keys are the strings "first", "second", and "third"

# 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 are 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
```

# Dictionary

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

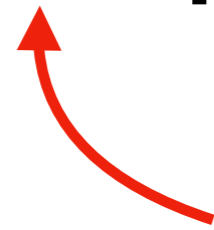
- values can be anything we choose (as with lists)
- keys can be nearly anything we choose (must be immutable)

```
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)

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

```
nums_list[1] → 700
```

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

```
nums_dict["first"] → 900
```

# 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
```

# Why call it a dictionary?

## break·fast

*/ˈbrekfəst/* 

*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"



# Why call it a dictionary?

this key  
(the word)

**break·fast**

*/ˈbrekfəst/* 

*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"

# Why call it a dictionary?

this key  
(the word)

break·fast

*/ˈbrekfəst/* 

*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"

maps to...

# Why call it a dictionary?

this key  
(the word)

**break·fast**

*/ˈbrekfəst/* 

maps to...

*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**

this value  
(the definition)

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

# Today's Outline

Data Structures

Mappings

Dictionaries

**Updates and Deletes**

Coding examples

# Dictionary Updates

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

# Dictionary Updates

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

# Dictionary Updates

```
>>> lst = ["zero", "ten", "not set"]
```

```
>>> lst[2] = "twenty"
```

```
>>> lst
```

```
['zero', 'ten', 'twenty']
```

```
>>> d = {0: "zero", 10: "ten", 20: "not set"}
```

```
>>> d[20] = "twenty"
```

dictionary updates look like list updates

# Dictionary Updates

```
>>> 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', 10: 'ten', 20: 'twenty'}
```

dictionary updates look like list updates



# Dictionary Deletes

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

# Dictionary Deletes

```
>>> lst = ["zero", "ten", "not set"]
```

```
>>> lst.pop(-1)
```

```
'not set'
```

```
>>> lst
```

```
['zero', 'ten']
```



**"not set" isn't in the list**

# Dictionary Deletes

```
>>> 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'
```

dictionary deletes look like list deletes

# Dictionary Deletes

```
>>> 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'}
```



**“not set” isn’t in the dict**

dictionary deletes look like list deletes

# Today's Outline

Data Structures

Mappings

Dictionaries

Updates and Deletes

**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
```

# 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

## 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
```