[220 / 319] Dictionaries

Learning Objectives

Dictionaries:
- creation using { } or dict()
- lookup, insert, update, delete key-value pairs
- in operator, for loop, len built-in function
- keys() and values() methods

Applications of dictionaries
- easy and fast lookup using keys
- frequency storage

Today's Outline

Data Structures

Mappings

Dictionaries
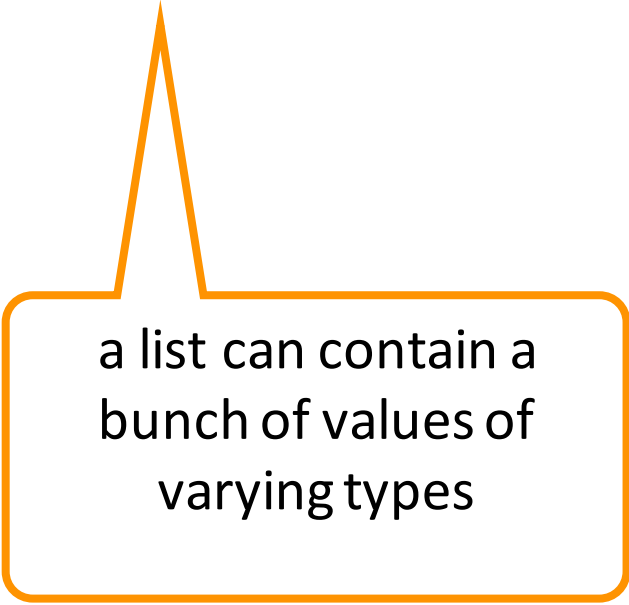
Mutations: Updates, Deletes, and Inserts

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

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

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

suggested note-taking

	values	relationships	operations
`list`	anything	ordered (0,1,...)	indexing, pop, len, index, slicing, in, iteration (for), ...
`set`	????	no ordering	`in, ==`
`dict`			
...			

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
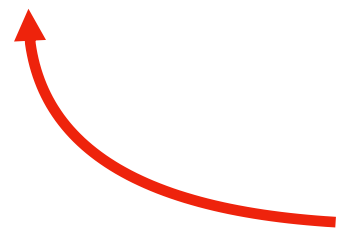
Coding examples

Mappings

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

List example:

nums = [300, 200, 400, 100]
       0   1   2   3

the "labels" are indexes, which are implicitly attached to values
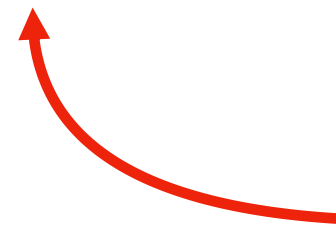
Mappings

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**

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

List example:

nums = [300, 200, 400, 100]

what if we want to use strings as labels?

x = nums[2]   # x=400
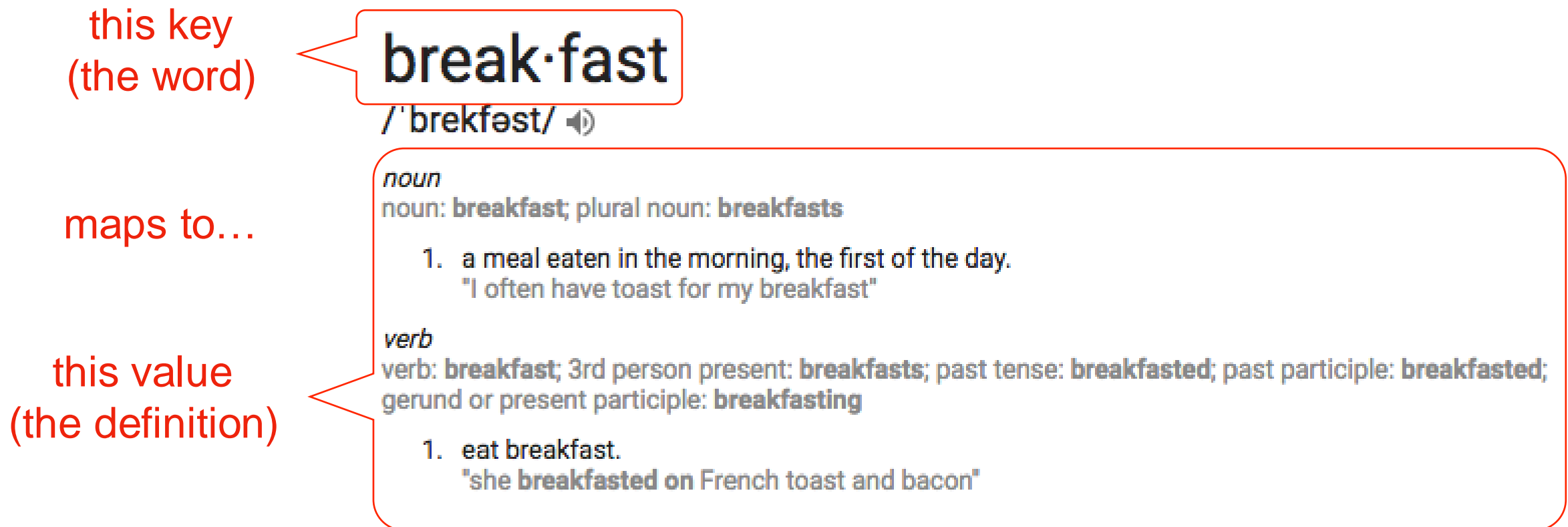
Today's Outline

Data Structures

Mappings

Dictionaries

Mutations: Updates, Deletes, and Inserts

Coding examples

Why call it a dictionary?

this key
(the word)

break·fast
/ˈbrekfəst/ 🔊

noun
noun: **breakfast**; plural noun: **breakfasts**

maps to…

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"

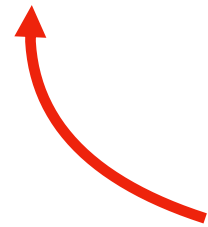Python dicts have insertion-based order (Python version > 3.6)

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 it's 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, "third":700, "second":800}
```

we have the same values

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

careful! curly braces are for both `set`s and `dict`s

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)

```
            0      1      2
nums_list = [900, 700, 800]
```

nums_list[1] ➡ 700

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

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

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, "third":700, "second":800}
```

nums_dict["second"] ➡ 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)
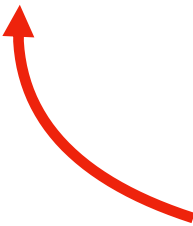to values
- values can be anything we choose (as with lists)
- keys can be nearly anything we choose (must be immutable)

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

nums_list[1] ➡ 700

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

nums_dict["first"] ➡ 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, "third":700, "second":800}
```

nums_dict["third"] ➡ 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

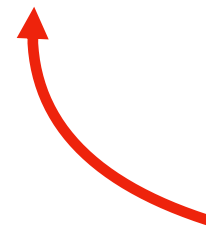Dictionaries map labels (called keys, rather than indexes) to values

- 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

index labels	values
0	900
1	700
2	800

ordered

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

nums_dict["third"] ➡ 700

key labels	values
"first"	900
"third"	700
"second"	800

insertion order (Python > 3.6)

A note on parenthetical characters

common structures   **uses**

parentheses: **(and)**

specifying order: `(1+2) * 3`

function invocation
or function definition: `f()`

brackets: **[and]**

list creation: `l = [1, 2, 3]`

sequence indexing: `l[-1]`

sequence slicing: `l[1:-2]`

dict lookup: `d["one"]`

braces: **{ and }**

dict creation: `d = {"one": 1, "two": 2}`
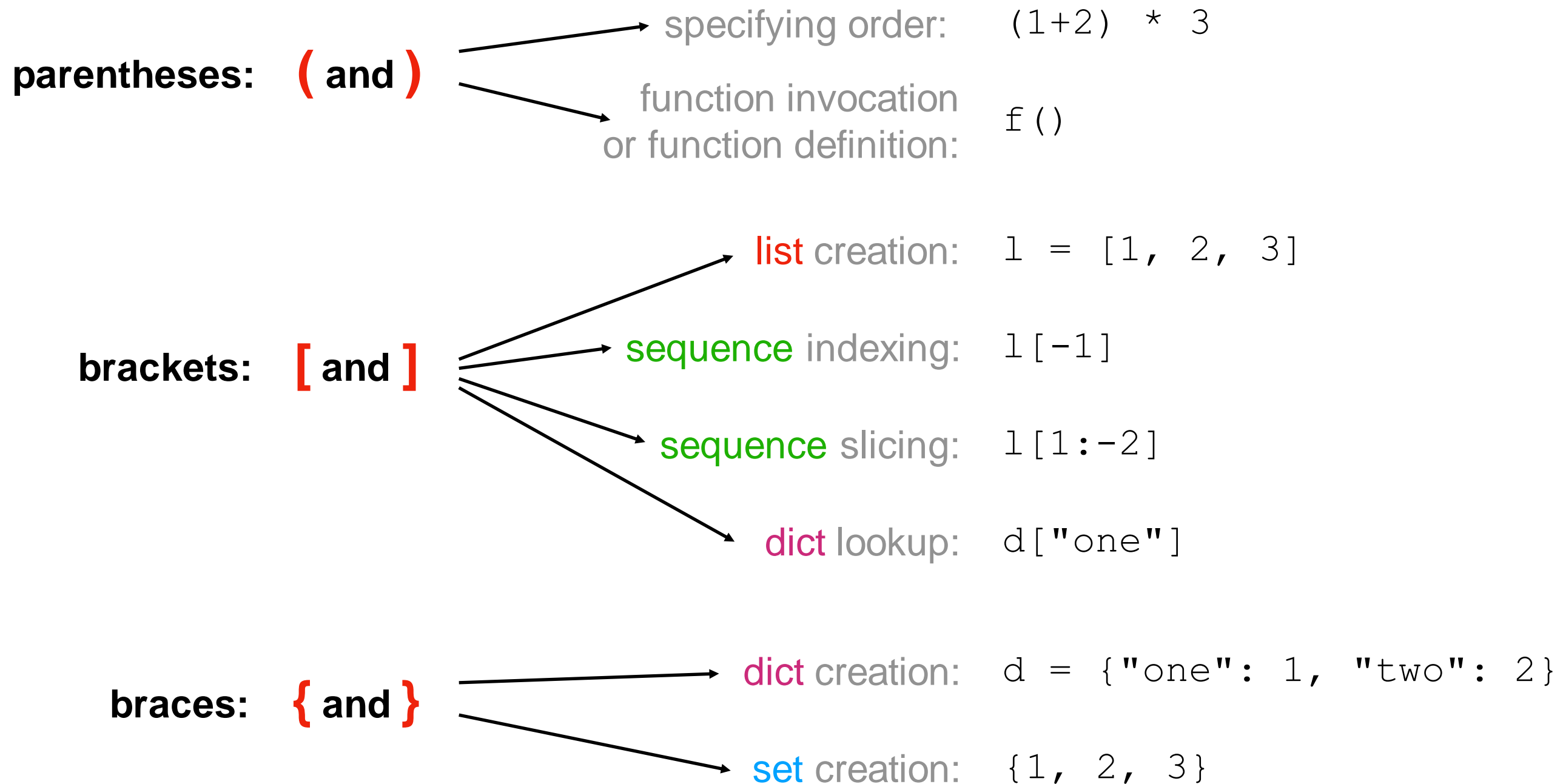
set creation: `{1, 2, 3}`

Empty set, list, and dict

braces: **{ and }**

dict creation: or

set creation: `s = set()`

```
d = {}

d = dict()
```

brackets: **[and]**

list creation: or

```
l = list()

l = []
```

Today's Outline

Data Structures

Mappings

Dictionaries

Mutations: Updates, Deletes, and Inserts

Coding examples

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", "twenty"]
>>> lst.pop(-1)
'twenty'
>>> lst
['zero', 'ten']

>>> d = {0: "zero", 10: "ten", 20: "twenty"}
>>> d.pop(20)
'twenty'
>>> d
{0: 'zero', 10: 'ten'}
```

"twenty" isn't in the list

"twenty" isn't in the dict

dictionary deletes look like list deletes

Dictionary Inserts

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

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

Example: Print Major Count

Goal: given a CSV of CS220 survey data, print each major's frequency

Input:
- A CSV

Output:
- count per major

Example output (not actual count):

Computer Science: 40
Engineering: 50
Business: 20

ALL COURSES. ALL DEGREES. ALL MAJORS.

Challenge: 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

https://en.wikipedia.org/wiki/The_Wizard_of_Oz_(1939_film)