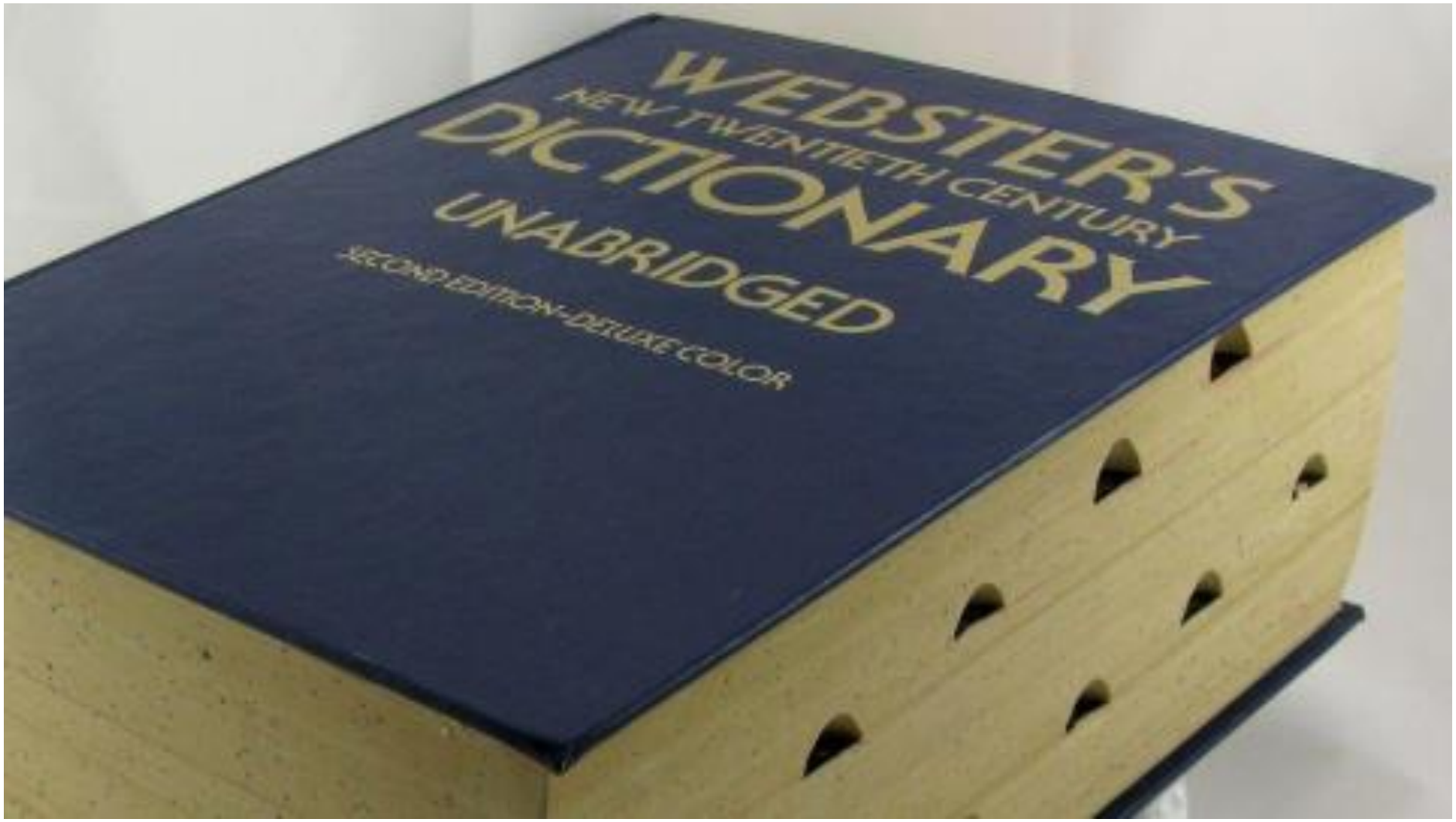


CSCI 141

Scott Wehrwein

Dictionaries

Goals

- Know the basics of how to use dictionaries (`dicts`):
  - Creation, assignment, and indexing
  - `get` method
  - `in` operator
  - `del` statement
  - Iterating over keys and values:
    - `keys`, `values`, and `items` methods

Dictionaries

- Lists, tuples, strings are all **sequences** (their contents are ordered)

- Python also has some types that handle non-sequential collections, including dictionaries (type `dict`):

- A dictionary is a collection of **key-value mappings**

Dictionaries

Another way to think about **lists**:

A **list** is a **mapping**

from *integer indices*

to *arbitrary values*.

Example:

```
["B", "A", 7]
```

represents the following mapping:

0: "B"
1: "A"
2: 7

the index `0` maps to the value `"B"`

Dictionaries

Another way to think about **lists**:

A **list** is a **mapping**

from *integer indices*

to *arbitrary values*.

Example:

```
["B", "A", 7]
```

represents the following mapping:

0: "B" → the index `0` maps to the value `"B"`
1: "A"
2: 7

A **dictionary** is a **mapping**

from *arbitrary immutable keys*

to *arbitrary values*.

```
{"B": 6, "A": 7}
```

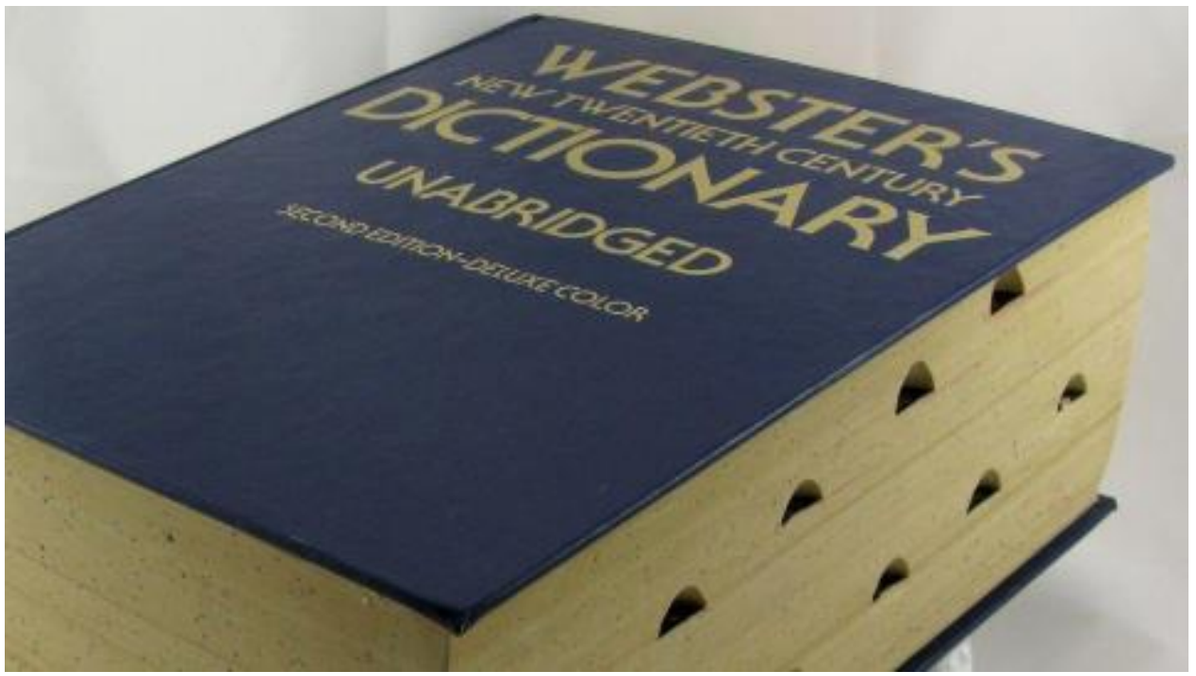
represents the following mapping:

"B": 6 → the key `B` maps to the value 6
"A": 7

Dictionaries

Why do we want this?

Suppose I want to store...




```
english = {}
english["aardvark"] = """a nocturnal burrowing
mammal with long ears, a tubular snout, and a
long extensible tongue, feeding on ants and
termites. Aardvarks are native to Africa and have
no close relatives."""
```

Dictionaries

Why do we want this?

Suppose I want to store...

A list of W#s of all the students in each of the lab sections.

```
sections = {}
sections[20769] = ["W0183782", "W0243810", # ...
sections[23512] = ["W0184582", "W0182368", # ...
# ...
```

Dictionaries

Why do we want this?

Suppose I want to store...

A bunch of different information about a WWU employee:

```
employee = {"First": "Scott",
            "Last": "Wehrwein",
            "Type": "Faculty",
            "W#": 98765438,
            # ...}
```

Dictionaries

Why do we want this?

Suppose I want to store...

The number of students with each letter grade in my class:

```
grade_counts = {"A": 6, "B": 12, "C": 8, "D": 2}
```

Dictionaries: Let's play

Dictionaries: Let's play

```
# create a dict:
grades = {"A": 10, "B": 18, "C": 6, "D": 2}
grades["A"] # => 10
grades["B"] # => 18
grades["E"] # KeyError
grades["E"] = "Huh?" # new mapping
grades["A"] = 12 # overwrites existing value
"F" in grades # => False
"E" in grades # => True
del grades["E"] # removes "E" and its value
"E" in grades # => False
```

Dictionaries: Let's play

```
# several ways to access values:
grades["A"] # => 12
grades.get("A") # => 12

# get method never causes an error
grades["Q"] # KeyError
grades.get("Q") # => None (no error!)

# get can take a default value to
# return if the key isn't found:
grades.get("A", 0) # => 12
grades.get("Q", 0) # => 0
```

Dictionaries: Cheat Sheet

- Creation:
  `d = {key1: value1, key2: value2, ...}`

- Access:
  `d[key] # => value`, or **error** if key not in d
  `d.get(key) # => value`, or ***`None`*** *if key not in d*
  `d.get(key, alt) # => value`, or ***`alt`*** *if key not in d*

- Assignment:
  `d[key] = new_value` → if key exists: overwrite old value
  → otherwise: add new key-value mapping

- Membership:
  `key in d # => True` if `d[key]` exists

- Removal:
  `del d[key] # deletes key and its associated value`

Iterating over Dictionaries? Demo

```
pop = {"WWU": 16121, "UW": 47899, "WSU": 24470}
```

Iterating over Dictionaries? Demo

```
pop = {"WWU": 16121, "UW": 47899, "WSU": 24470}
```

- for key in d
- d.keys(); list(d.keys())
- for val in d.values()
- key, value in d.items()
- list(d.items())

Dictionaries: Cheat Sheet

- Creation:
  `d = {key1: value1, key2: value2, ...}`

- Access:
  `d[key] # => value`, or **error** if key not in d
  `d.get(key) # => value`, or ***`None`*** *if key not in d*
  `d.get(key, alt) # => value`, or ***`alt`*** *if key not in d*

- Assignment:
  `d[key] = new_value` → if key exists: overwrite old value
  → otherwise: add new key-value mapping

- Membership:
  `key in d # => True` if `d[key]` exists

- Removal:
  `del d[key] # deletes key and its associated value`

Dictionary Iteration: Cheat Sheet

```
d = {key1: value1, key2: value2, ...}
```

```
for key in d:
    print(key)
```

```
for key in d.keys():
    print(key)
```

```
for val in d.values():
    print(val)
```

```
for (key, val) in d.items():
    print(key, val, sep=": ")
```

Note 1: Like `range`, these methods return sequences that are not lists. To get a list of values use `list(d.values())`.

Note 2: In Python <3.7, you can't rely on the key ordering being the same. In 3.7+, the order matches insertion order.