Dictionaries

Ruth Anderson
UW CSE 160
Autumn 2022

Keeping track of favorite colors

- Write a program that keeps track of the favorite color of each staff member.
- If I give you a staff member UWNetID, you should be able to tell me what their favorite color is.
  - UWNetIDs are unique
  - Favorite colors are not unique! More than one person may have the same favorite color

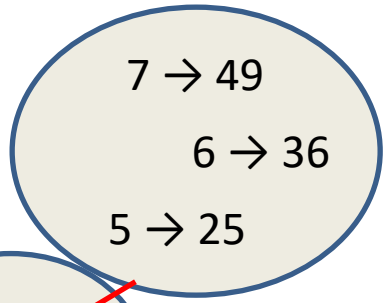
- Data structure? List of ??

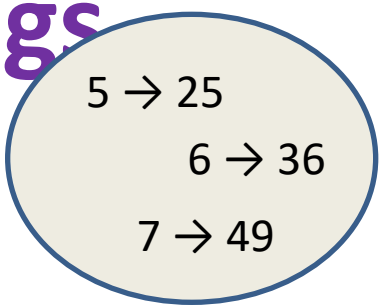
See **list of lists** example in python tutor

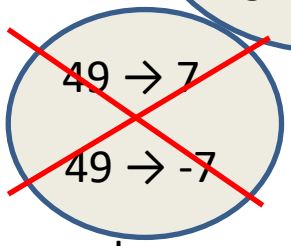
See **dictionary** example in python tutor

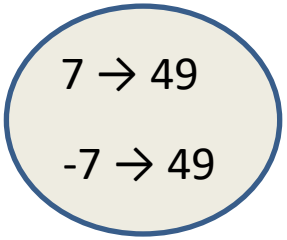
Dictionaries store mappings

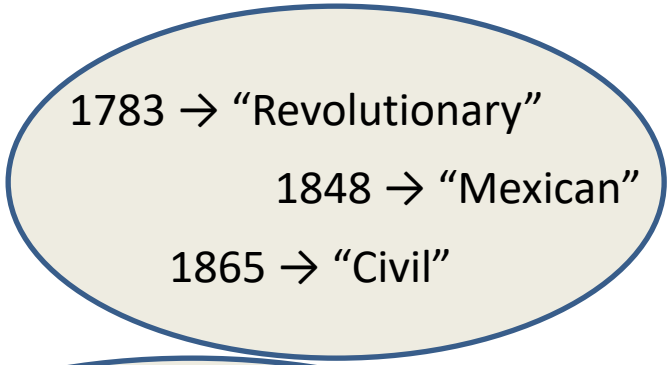
- A dictionary maps each *key* to a *value*
- Order does not matter
- Given a key, can look up a value
  - Given a value, cannot look up its key
- **No duplicate keys**
  - Two or more keys may map to the same value
- *Keys* and *values* are Python values
  - **Keys** must be **immutable** (not a list, set, or dict)
- Can add *key* → *value* mappings to a dictionary
  - Can also remove (less common)

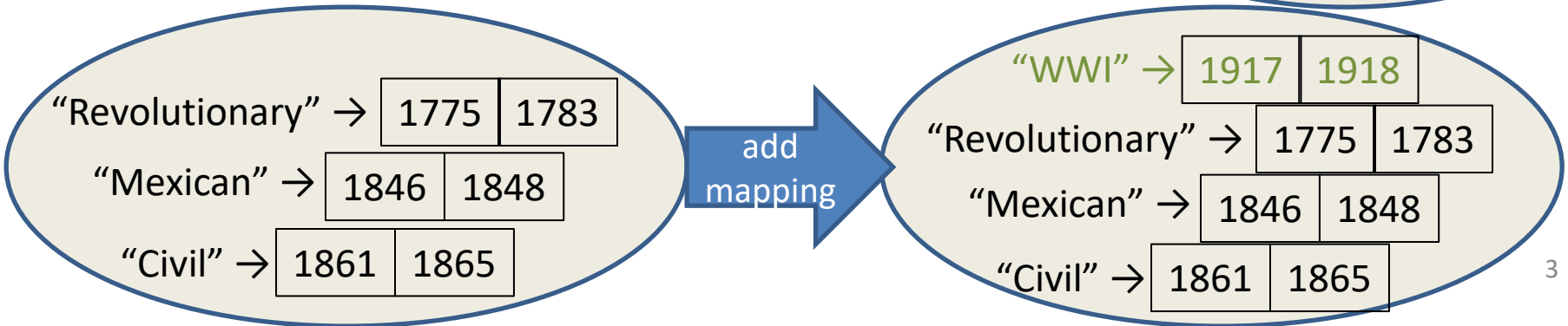












See in python tutor

Creating a dictionary

"GA" → "Atlanta"

"WA" → "Olympia"

```
state_capitals = {"GA" : "Atlanta", "WA": "Olympia" }
```

One way to create an empty dictionary

```
phonebook = dict()
phonebook["Alice"] = "206-555-4455"
phonebook["Bob"] = "212-555-2211"
```

"Alice" → "206-555-4455"

"Bob" → "212-555-1212"

Another way to create an empty dictionary

```
atomic_number = {}
atomic_number["H"] = 1
atomic_number["Fe"] = 26
atomic_number["Au"] = 79
```

"H" → 1

"Fe" → 26

"Au" → 79

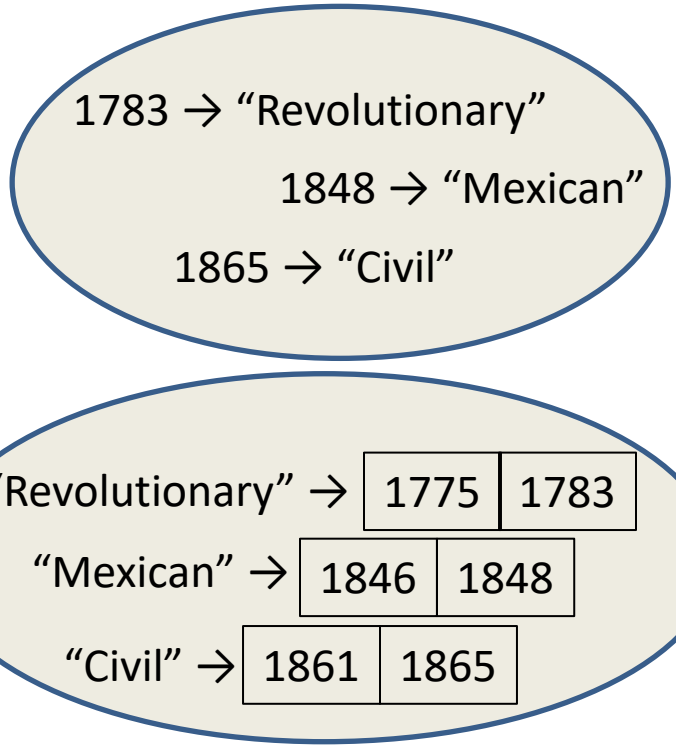
See in python tutor

Dictionary syntax in Python

```
d = {}
d = dict()
```

Two different ways to create an empty dictionary

```
us_wars_by_end = {
   1783: "Revolutionary",
   1848: "Mexican",
   1865: "Civil" }
us_wars_by_name = {
   "Civil": [1861, 1865],
   "Mexican": [1846, 1848],
   "Revolutionary": [1775, 1783]
}
```



- Syntax just like lists, for accessing and setting:

```
us_wars_by_end[1783]    ⇒
us_wars_by_end[1783][1:10]    ⇒
us_wars_by_name["WWI"] = [1917, 1918]
```

Showing typing into the **python interpreter,** not a program

[See in python tutor]

Accessing a dictionary

```
>>> atomic_number = {"H":1, "Fe":26, "Au":79}
>>> atomic_number["Au"]
79
>>> atomic_number["B"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'B'
>>> "Au" in atomic_number
True
>>> list(atomic_number.keys())
['H', 'Au', 'Fe']
>>> list(atomic_number.values())
[1, 79, 26]
>>> list(atomic_number.items())
[('H', 1), ('Au', 79), ('Fe', 26)]
```

"H" → 1

"Fe" → 26

"Au" → 79

This is a **tuple**, not a list.
Note: Uses parentheses

Good for iteration (for loops)

```
for key in my_map.keys():
  val = my_map[key]
  … use key and val

for key in my_map:
  val = my_map[key]
  … use key and val

for (key,val) in my_map.items():
  … use key and val
```

See in python tutor

Iterating through a dictionary

```
atomic_number = {"H":1, "Fe":26, "Au":79}

# Print out all the keys:
for element_name in atomic_number.keys():
    print(element_name)

# Another way to print out all the keys:
for element_name in atomic_number:
    print(element_name)

# Print out all the values:
for element_number in atomic_number.values():
    print(element_number)

# Print out the keys and the values
for (element_name, element_number) in atomic_number.items():
    print("name:", element_name, "number:", element_number)
```

Often you do **not need** to iterate through a dictionary! If you have a key and you want the value associated with it, just ask for it! E.g. `print(atomic_number["Fe"])`

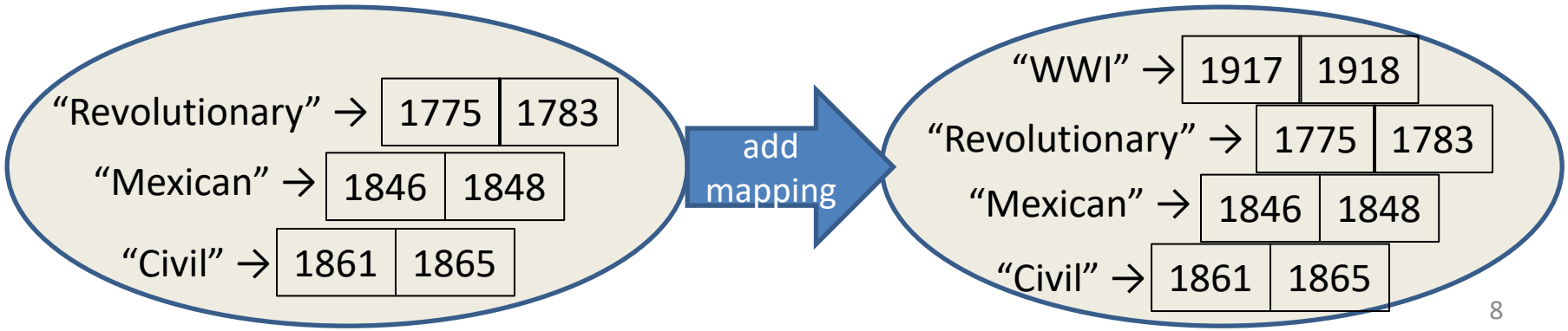
If you need to print or otherwise examine **all** of the keys or the values in a dictionary, then iterate.

See in python tutor

Modifying a dictionary

```
us_wars1 = {
    "Revolutionary": [1775, 1783],
    "Mexican": [1846, 1848],
    "Civil": [1861, 1865] }
```

```
us_wars1["WWI"] = [1917, 1918]   # add mapping
del us_wars1["Civil"]   # remove mapping
```



See in python tutor

Dictionary Exercises

Given: `squares = {1: 1, 2: 4, 3: 9, 4: 16}`

What do these expressions evaluate to?

```
squares[3] + squares[3]
squares[3 + 3]
squares[2] + squares[2]
squares[2 + 2]
```

- Write code to convert a list to a dictionary where each item in the list will be a key, and whose value is the square of the key:
  - For example, given [5, 6, 7], produce {5: 25, 6: 36, 7: 49}
- Write code to reverse the key with the value in a dictionary:
  - For example, given {5: 25, 6: 36, 7: 49}, produce {25: 5, 36: 6, 49: 7}

See in python tutor

Dictionary Exercise (Answers)

- Convert a list to a dictionary:
  - E.g. Given [5, 6, 7], produce {5: 25, 6: 36, 7: 49}

```
d = {}
for i in [5, 6, 7]:  # or range(5, 8)
    d[i] = i * i
```

- Reverse key with value in a dictionary:
  - E.g. Given {5: 25, 6: 36, 7: 49}, produce {25: 5, 36: 6, 49: 7}

```
k ={}
for i in d.keys():
    k[d[i]] = i
```

Aside: A list is like a dictionary

- A list maps an integer index to a value
  - The integers must be a continuous range 0..*i*

```
my_list = ['a', 'b', 'c']
my_list[1] ⇒ 'b'
my_list[3] = 'c'  # error!
```

- In what ways is a list more convenient than a dictionary?
- In what ways is a list less convenient than a dictionary?

Not every type is allowed to be a key in a dictionary

- Dictionaries hold **key: value** pairs
- **Keys** must be **immutable**
  - int, float, bool, string, *tuple of immutable types*
  - *not*: list, set, dictionary
- **Values** in a dictionary can be anything