C: Dictionaries

Key properties of dictionaries:

1. Collection of **key, value** entries
2. Shown by curly brackets: **{}**
3. Key, value relationship indicated by colon: **{ key1: value1, key2: value2 }**
4. Keys are often strings (not required, though); values vary a lot
5. Getting and setting elements is similar to lists:

   Can **look up values** by key: dname[key]
   Can **set values** by key: dname[key] = newvalue

Example:

```
course_info = { "722": "Quantitative Analysis",
                "789": "Advanced Policy Analysis" }
```

course_info["722"] 👉 "Quantitative Analysis"
course_info["789"] 👉 "Advanced Policy Analysis"

Or, using a **variable** to hold the key:

course = "789"
course_info[course] 👉 "Advanced Policy Analysis"

Easy to add an additional (key, value) pair:

```
course_info["777"] = "Economics of Environmental Policy"
```

```
course_info 👉  { "722": "Quantitative Analysis",
                  "789": "Advanced Policy Analysis",
                  "777": "Economics of Environmental Policy"  }
```

Easy to change existing values:

```
course_info["777"] = "Environmental Economics"
```

```
course_info 👉  { "722": "Quantitative Analysis",
                  "789": "Advanced Policy Analysis",
                  "777": "Environmental Economics"  }
```

Elements can be heterogeneous:

```
ny = { "name": "New York",
       "admitted": 1788,
       "population": 19.3e6,
       "senators": ["Gillibrand", "Schumer"] }
```

ny["name"]	👉	"New York"	*str*
ny["admitted"]	👉	1788	*int*
ny["population"]	👉	19.3e6	*float*
ny["senators"]	👉	["Gillibrand", "Schumer"]	*list*

Use Case 1: Dictionaries as **data objects** with **attributes**:

Example: **ny**

- Data **object** describing a state
- Four **attributes**: name, admitted, population, senators

Second object for Texas with similar attributes: **tx**

```
tx = { "name": "Texas",
       "admitted": 1845,
       "population": 29.4e6,
       "senators": ["Cornyn", "Cruz"] }
```

Could use a **list** of state objects to find joint population and senators:

```
states = [ ny, tx ]

tot_pop = 0
tot_senators = []

for state in states:
   tot_pop = tot_pop + state['population']
   tot_senators.extend( state['senators'] )

print( tot_pop/1e6, "million" )
print( sorted(tot_senators) )
```

Unrolling the loop:

```
1    state = states[0]                                  (implicit)
2    tot_pop = tot_pop + state['population']
3    tot_senators.extend( state['senators'] )

4    state = states[1]                                  (implicit)
5    tot_pop = tot_pop + state['population']
```

```
6       tot_senators.extend( state['senators'] )
```

What happens at each step:

Step 1: state = states[0]

Input:
states 👉 [ny, tx]
states[0] 👉 ny

Result:
state 👉 ny

Step 2: tot_pop = tot_pop + state['population']

Input:
tot_pop 👉 0
state 👉 ny
state['population'] 👉 ny['population'] 👉 19.3e6

Result:
tot_pop 👉 19.3e6 (has been **updated**)

Step 3: tot_senators.extend(state['senators'])

Input:
tot_senators 👉 []
state 👉 ny
state['senators'] 👉 ny['senators'] 👉 ["Gillibrand", "Schumer"]

Result:
tot_senators 👉 ["Gillibrand", "Schumer"]

Step 4: state = states[1]

Input:

states 👉 [ny, tx]

states[1] 👉 tx

Result:

state 👉 tx

Step 5: tot_pop = tot_pop + state['population']

Input:

tot_pop 👉 19.3e6

state 👉 tx

state['population'] 👉 tx['population'] 👉 29.4e6

Result:

tot_pop 👉 48.7e6

Step 6: tot_senators.extend(state['senators'])

Input:

tot_senators 👉 ["Gillibrand", "Schumer"]

state 👉 tx

state['senators'] 👉 tx['senators'] 👉 ["Cornyn", "Cruz"]

Result:

tot_senators 👉 ["Gillibrand", "Schumer", "Cornyn", "Cruz"]

Ultimate result at the print statements:

48.7 million
['Cornyn', 'Cruz', 'Gillibrand', 'Schumer']

HIGHLY scalable:

Same loop code no matter how long the list of states

Use Case 2: Dictionaries for **translating** or **standardizing** data:

Example: standardizing address data with inconsistent city spellings:

```
raw_data = [ "Syr", "Cuse", "Lafayette", "Syracuse", "Cuse" ]
```

Set up dictionary with **raw names** as **keys** and **good names** as **values**:

Abstractly:

```
dname = { raw_name1: good_name1,
          raw_name2: good_name2,
          … }
```

This example:

```
fixnames = { "Syracuse":   "Syracuse",
             "Syr":        "Syracuse",
             "Cuse":       "Syracuse",
             "Lafayette":  "Lafayette" }
```

Then:

1. Loop through the raw data
2. Use raw name to look up the good name for each input name
3. Add to a new list of the good names

```
good_names = []
```

```
for r in raw_data:
   g = fixnames[ r ]
   good_names.append( g )
```

Unrolling the loop:

Step 1: r = raw_data[**0**]

Input: raw_data = ["Syr", "Cuse", "Lafayette", "Syracuse", "Cuse"]

Result: r 👉 "Syr"

Step 2: g = fixnames[r]

Input: fixnames["Syr"] 👉 "Syracuse"

Result: g 👉 "Syracuse"

Step 3: good_names.append(g)

Input: good_names 👉 []

Result: good_names 👉 ["Syracuse"]

Step 4: r = raw_data[**1**]

Input: raw_data = ["Syr", "Cuse", "Lafayette", "Syracuse", "Cuse"]

Result: r 👉 "Cuse"

Step 5: g = fixnames[r]

Input: fixnames["Cuse"] 👉 "Syracuse"

Result: g 👉 "Syracuse"

Step 6: good_names.append(g)

Input: good_names 👉 ["Syracuse"]

Result: good_names 👉 ["Syracuse", "Syracuse"]

Step 7: …

End result:

good_names = ["Syracuse", "Syracuse", "Lafayette", "Syracuse", "Syracuse"]

Note: good use for a list comprehension

HIGHLY scalable:

Easy to add corrections via additional dictionary rows

Detailed example: demo.py in g05