15-110 Principles of Computing – S19

Lecture 13: Dictionaries

Teacher:
Gianni A. Di Caro

Manipulating structured data

- So far we have used lists to manipulate structured data

- Example: data about people, including account number, sex, country of origin

```
accounts = [ ['J. Smith', [35672, 'M', 'USA']], ['M. Saleh', [27623, 'M', 'Jordan']],
             ['F. Dupont', [17623, 'F', 'France']] ]
```

- Example: data about animals, including name, phylum, class, order

```
animals = [ ['tiger', 'vertebrate', 'mammal', 'carnivore'],
            ['orangu-tan', 'vertebrate', 'mammal', 'primate'],
            ['falcon', 'vertebrate', 'bird', 'falconiformes'] ]
```

- Example: data about countries, including name, population, GDP per capita, S&P's rating

```
countries = [ ['USA', 324459463, 59792, 'AAA'], ['Switzerland', 8476005, 80637, 'AAA'],
              ['Qatar', 2639211, 61024, 'AA-'], ['Luxembourg', 583455, 105863, 'AAA'] ]
```

Manipulating structured data

➢ How do we access and modify these type of data?

```
accounts = [ ['J. Smith', [35672, 'M', 'USA']], ['M. Saleh', [27623, 'M', 'Jordan']],
             ['F. Dupont', [17623, 'F', 'France']] ]
```

- Example: Get the data of a specific person (e.g., J. Smith)
- Example: Modify the data of a specific person (e.g., the account of F. Dupont)
- Example: Get the data of the citizens of a specific country (e.g., USA citizens)

➢ No built-in method does directly the job, we need to write our own function to retrieve needed data ☹

➢ *Idea:* we need to provide a **search key** and retrieve the **associate data**

Manipulating structured data

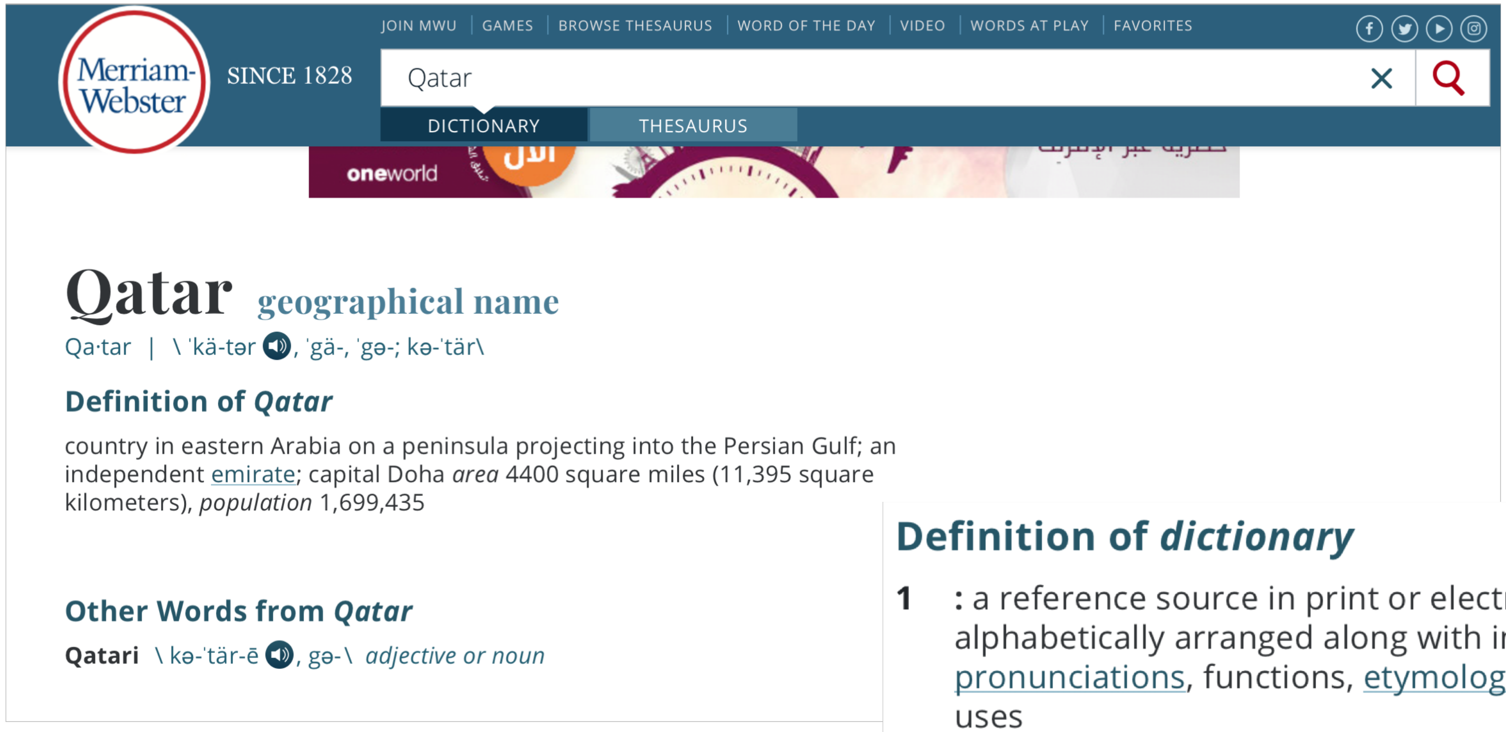
- Provide a **search key** and retrieve the **associate data**
- Let’s make it work for lists of lists, at least

```
accounts = [ ['J. Smith', [35672, 'M', 'USA']], ['M. Saleh', [27623, 'M', 'Jordan']],
             ['F. Dupont', [17623, 'F', 'France']] ]

 def find_in_list_of_lists(data, item, field):
     '''Search for item in the data, which is a list of lists.
         Item is searched in the field position of each list element.
         Return the whole list item, and its position in data, if found.
         The first item which is present in the list is returned.
         None is returned if item is not present in data at the given field.
     '''
     for i in data:
         if i[field] == item:
             index = data.index(i)
             return (i, index)
     return None
```

Dictionary data structure

- ✓ Don't we have a more structured / built-in way to provide a **search key** and retrieve the **associate data?**
- ✓ Or, more in general**, to label data and access / search data using labels?**


JOIN MWU | GAMES | BROWSE THESAURUS | WORD OF THE DAY | VIDEO | WORDS AT PLAY | FAVORITES
Merriam-Webster
SINCE 1828
Qatar
DICTIONARY
THESAURUS
oneworld

Qatar geographical name

Qa·tar | \ ˈkä-tər, ˈgä-, ˈgə-; kə-ˈtär\

Definition of *Qatar*

country in eastern Arabia on a peninsula projecting into the Persian Gulf; an independent emirate; capital Doha *area* 4400 square miles (11,395 square kilometers), *population* 1,699,435

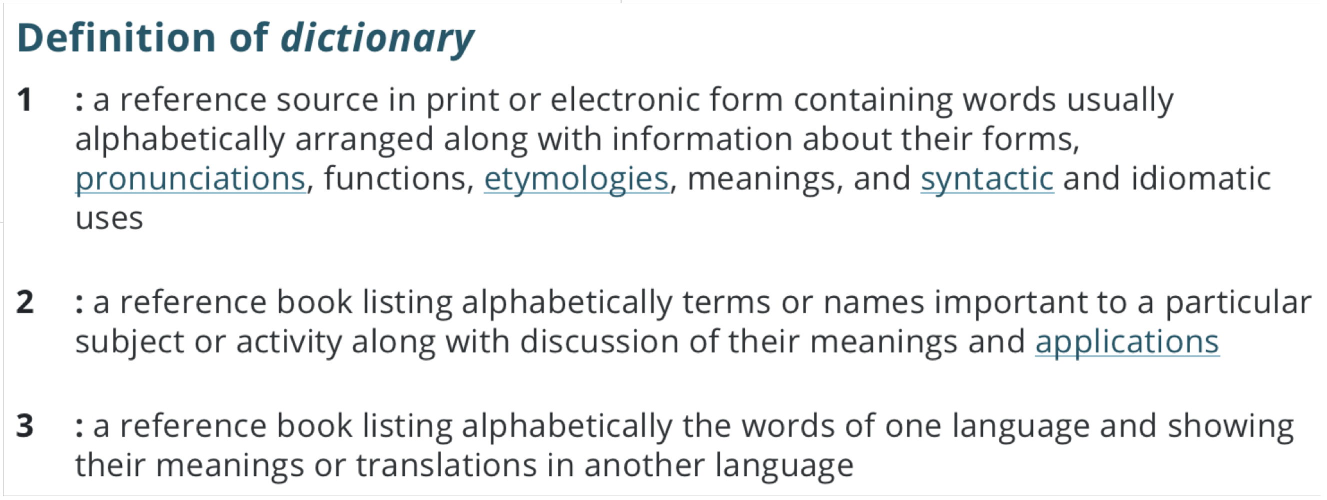
Other Words from *Qatar*

Qatari \ kə-ˈtär-ē, gə-\ *adjective or noun*

Collection of *data resources* that can be accessed through specific keyword identifiers (e.g., Qatar)

Collection of *pairs* of:

<key>, <data>



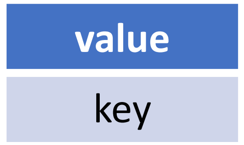
Definition of *dictionary*

1 : a reference source in print or electronic form containing words usually alphabetically arranged along with information about their forms, pronunciations, functions, etymologies, meanings, and syntactic and idiomatic uses

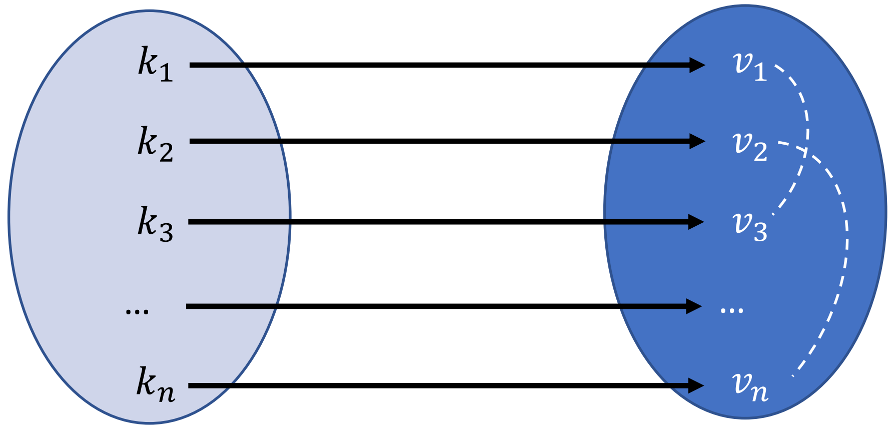
2 : a reference book listing alphabetically terms or names important to a particular subject or activity along with discussion of their meanings and applications

3 : a reference book listing alphabetically the words of one language and showing their meanings or translations in another language

Dictionary data structure: maps (associative, surjective)

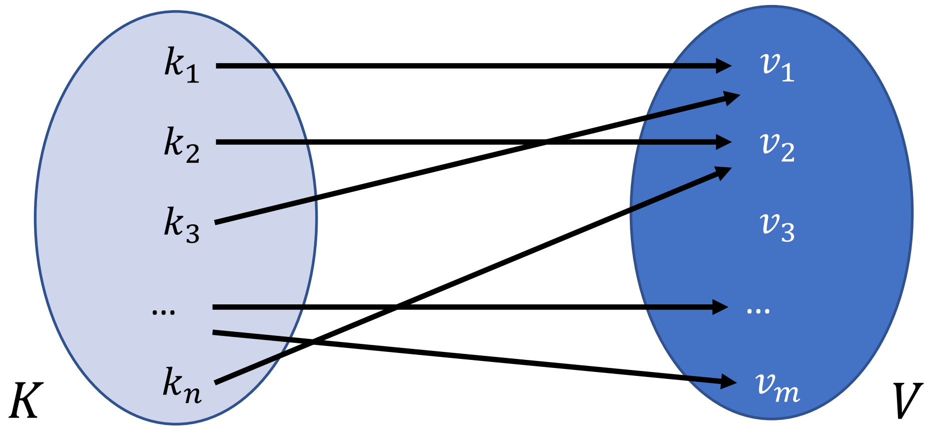


key $\mapsto$ value



- A dictionary **maps** n keys into n values
- Keys are all different / unique
- Different keys might be associated to a same value (representing however *physically different data records*)

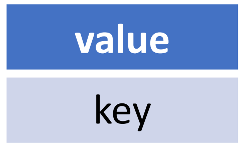
- In the example, the value v_1 associated to key k_1 is the same as the value v_3 associated to key k_3 (as shown by dashed lines), however they are physically different items
- E.g., k_1 = “John”, k_3= “Ann”, and they have the same age v_1 = 20, v_3 = 20



- Accounting for values that can be the same, we can represent a dictionary map as a **surjective map** in *mathematics*, where each element in the value set V (of size m) is associated to (is the co-image of) *at least* one element from the key set K (of size $n \geq m$), and all elements in K are associated to one element in V

$$dict : K \mapsto V$$

Dictionary data structure: associative maps



key ↦ value

Examples:

- SSNs ↦ Person information data
- Names ↦ phone numbers, email
- Usernames ↦ passwords, OS preferences
- ZIP codes ↦ Shipping costs and time
- Country names ↦ Capital, demographic info
- Sales items ↦ Quantity in stock, time to order
- Courses ↦ Student statistics
- Persons ↦ Friends in social network
- Animals ↦ Classification data
- Companies ↦ Rate, capital, investments
- …

- In all the examples, a **unique label** (***key***) can be associated to a (more or less complex) **piece of data** (the ***value***)
- This motivates the choice of a *dictionary data structure* to represent and manipulate these type of data

Dictionary data structure

- **Data type**: `dict`
- **Syntax:**

Separator between key-value entries

```
{ key_1: value_1,  key_2: value_2,  key_3: value_3 }
```

`dict` literal object with three elements

```
d = { key_1: value_1,  key_2: value_2,  key_3: value_3 }
```

definition of a `dict` variable `d` with three elements

```
d = { }
```

definition of an empty `dict` variable `d`

Delimiters for literal object definition

Key(word) identifier

Separator between key and value

Data value (information data) associated to the key

Dictionary data structure: unordered, associative array (map)

- **Unordered:** it's *not a sequence*, rather a <u>collection,</u> where items are accessed through the keys, not by their position in a sequence

```
x = [20, 22, 29, 20]
```

Value	**20**	**22**	**29**	**20**
Position index	0	1	2	3

List

`x[1]` is the value of `x` at position 1, which is 22

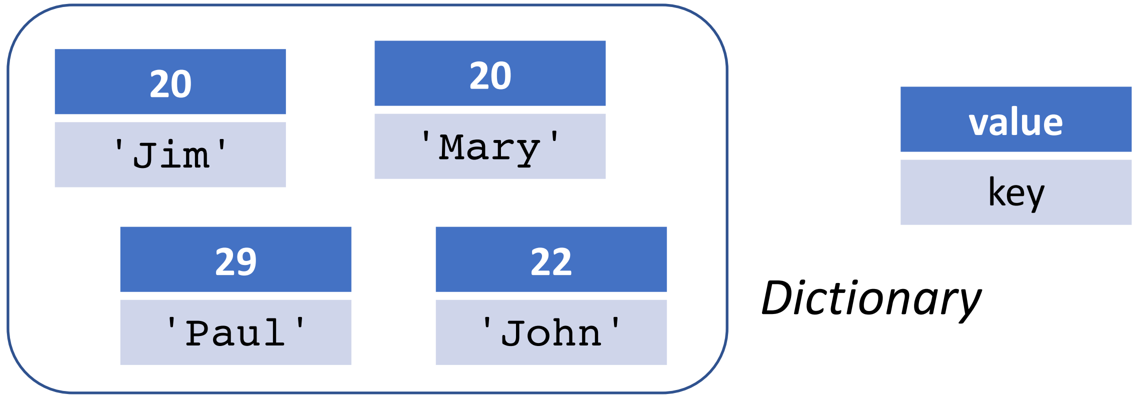
✓ A sequence type accesses values by their position in the sequence, i.e., values are *sequentially indexed*

index ↦ value

✓ A dictionary represents data values by *using key labels*, and then accesses values by their keys, i.e., *associates* key labels to values (<u>associative memory</u>):

key ↦ value

```
d = {'John': 22, 'Jim': 20,
     'Mary': 20, 'Paul': 29}
```



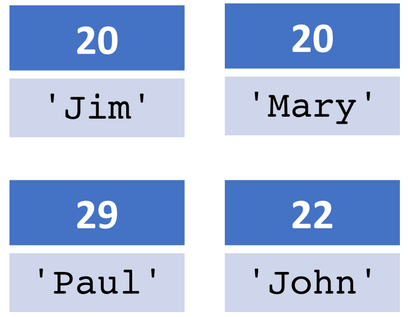
Dictionary

- `d['John']` is the value of `x` **associated** to the keyword `'John'`, which is 22
- `d[1]` throws an error: there's no a key with value 1 in the dictionary

Dictionary data structure: non-scalar, mutable

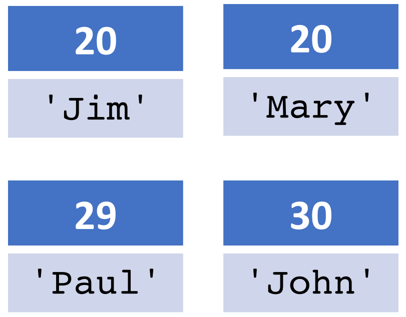
- **Non-scalar**: it’s a composite data type, it has *internal structure*
- **Mutable:** values of dictionary’s entries can be *updated* and items can be added and deleted (without changing dictionary identity), *aliases* can be created between variables

```
d = {'John': 22, 'Jim': 20, 'Mary': 20, 'Paul': 29}
```



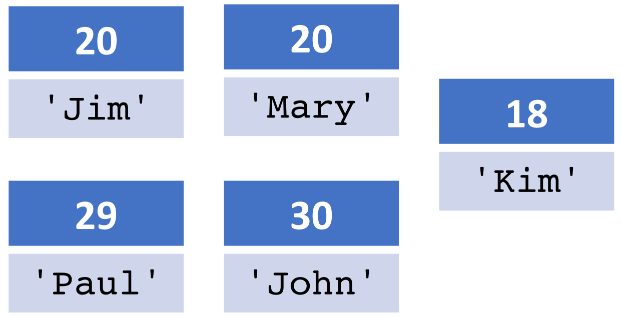
✓ Update value of existing keys

```
d['John'] = 30
```



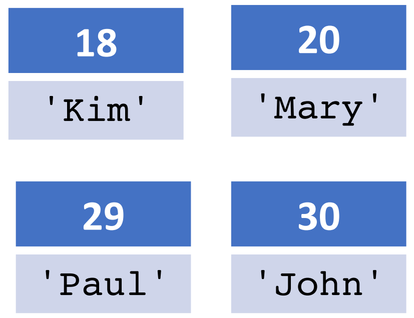
✓ Add a new key-value pair:

```
d['Kim'] = 18
```



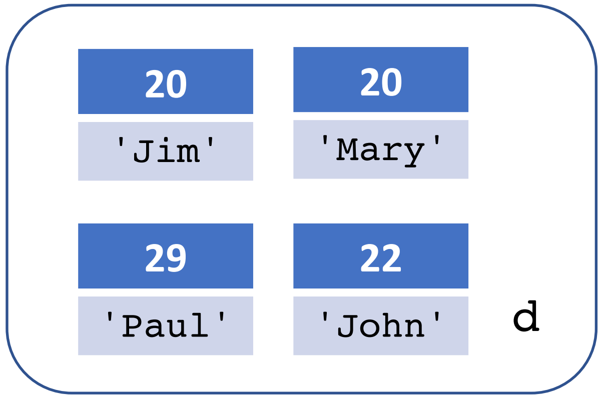
✓ Delete an existing item:

```
del d['Jim']
```



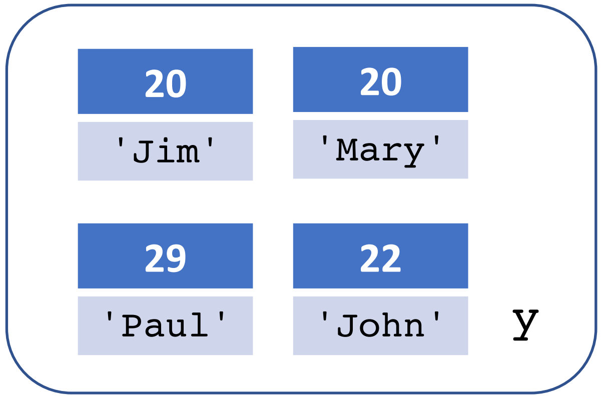
Dictionary data structure: mutable

```
d = {'John': 22, 'Jim': 20, 'Mary': 20, 'Paul': 29}
```



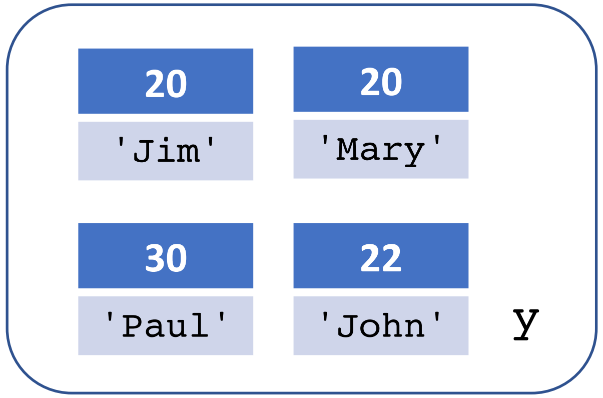
✓ Create an alias:

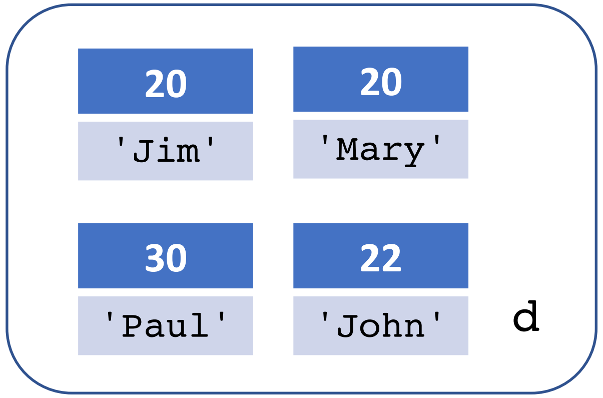
```
y = d
```



- Changing `y` changes `d` and vice versa:

```
y['Paul'] = 30
```





- The two dictionaries have the same identity:

`id(y)` is the same as `id(d)`

Restrictions and freedom on data types for keys and values

key ↦ value

- A **key** can only contain **immutable** data types: `int, float, bool, str, tuple`
- A **value** can be of **any type**
- **Keys and values** of the same dictionary can be of any (allowed) mixed type

```
d = {'John': 22, 'Jim': 20, 'Mary': 20, 'Paul': 29}
```

- ✓ `d[12] = 'New data value'`
- ✓ `d['Jim'] = True`
- ✓ `d['List data'] = [ [1,2,3], [4,5,6] ]`
- ✓ `d[(2,3)] = 7.8`
- ❖ `d[[2,3]] = 'Incorrect'`
- ❖ `d[([1,3], 2)] = 'Incorrect'`