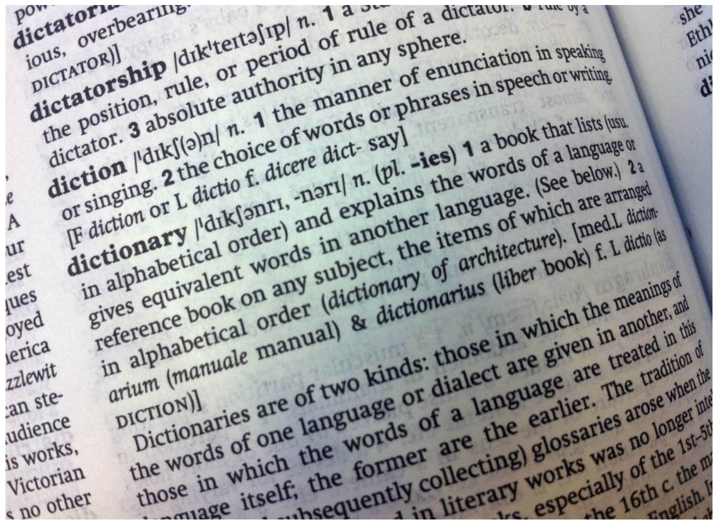
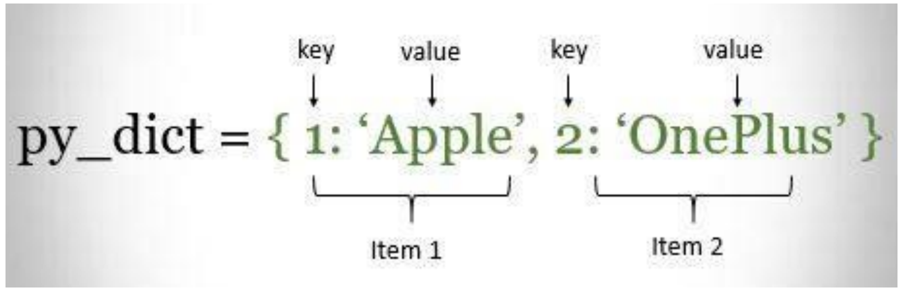
COMP 125 Programming with Python
Dictionaries



dictatorial ... ious, overbearing. ... DICTATOR)]
dictatorship /dɪkˈteɪtəʃɪp/ n. 1 a st... the position, rule, or period of rule of a dictator. 3 absolute authority in any sphere.
diction /ˈdɪkʃ(ə)n/ n. 1 the manner of enunciation in speaking or singing. 2 the choice of words or phrases in speech or writing. [F diction or L dictio f. dicere dict- say]
dictionary /ˈdɪkʃənrɪ, -nərɪ/ n. (pl. -ies) 1 a book that lists (usu. in alphabetical order) and explains the words of a language or gives equivalent words in another language. (See below.) 2 a reference book on any subject, the items of which are arranged in alphabetical order (dictionary of architecture). [med.L dictionarium (manuale manual) & dictionarius (liber book) f. L dictio (as DICTION)]
Dictionaries are of two kinds: those in which the meanings of the words of one language or dialect are given in another, and those in which the words of a language are treated in this language itself; the former are the earlier. The tradition of ... subsequently collecting) glossaries arose when the ... in literary works was no longer inte... especially of the 1st–5t... the 16th c. the m... English. I...



```
py_dict = { 1: 'Apple', 2: 'OnePlus' }
```

Mehmet Sayar

Koç University

How can we store and organize data?

- **Collection**: A data structure used to store values as a single unit

- So far, we have seen lists and tuples for general data storage
- The items were accessed by their indices
- What if we want to use more generic indices, such as a string?
- Or if we have mappings such as key -> value

Organizing Data

- Dictionaries

```
Turkish2English['bir'] = 'one'
```

- Using name and surname to get the student identification numbers

```
StudentIDs['John Doe'] = '00123456'
```

- Storing the measurements of a furniture

```
table['width'] = 1.2,table['length'] = 2.0
```

- Sparse arrays (where a lot entries are non-existent)

```
array[1] = 1.48,array[141] = -7.92,array[186219] = 0.3
```

- And many more!

Dictionaries

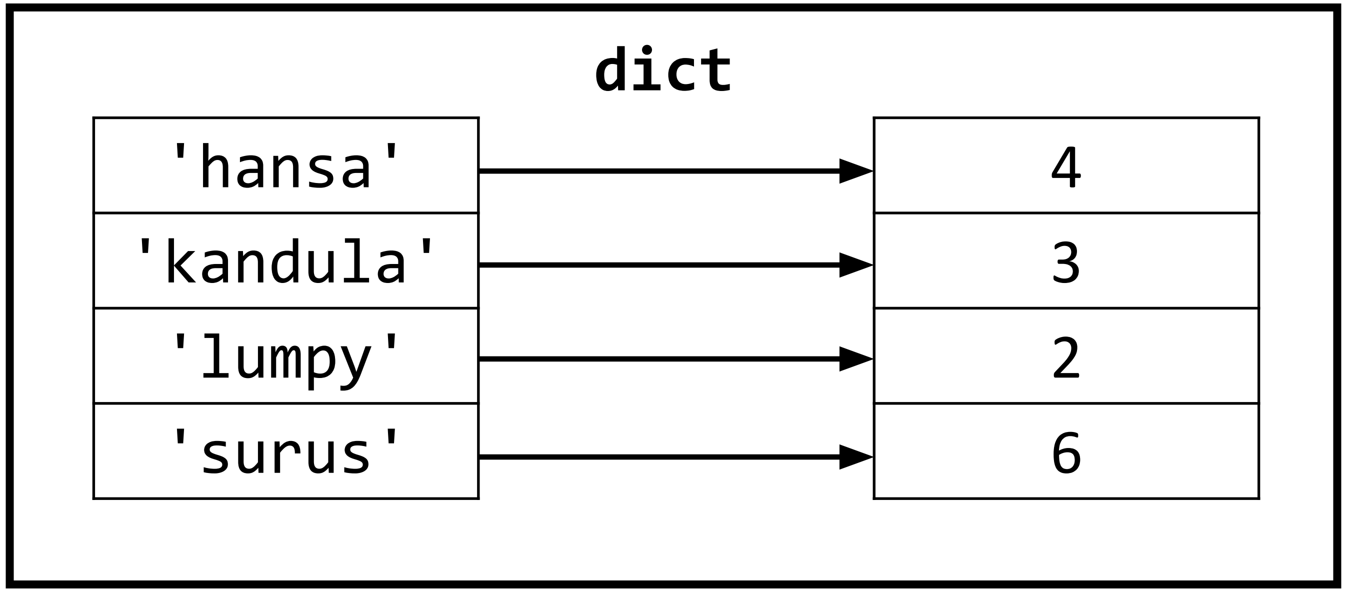
- A collection (sometimes called container) data type that <u>maps</u> "**keys**" to their associated "**values**"
- Defined using curly brackets: `{key1:value1, key2:value2, ...}`
- Called maps in most languages, associative arrays in some others
- Values can be any Python object
- Keys can be any "hashable" Python object (more on this in a little bit)
  - Strings, integers, floats work
  - Booleans and functions work too
  - Tuples work but not lists

Anatomy of a Dictionary

- Let’s define a dictionary of pet’s and the number of times they were fed

```
d = {'hansa': 4, 'kandula': 3, 'lumpy': 2, 'surus': 6}
```

- Let’s visualize:



Anatomy of a Dictionary

- Can “get” (retrieve) the value

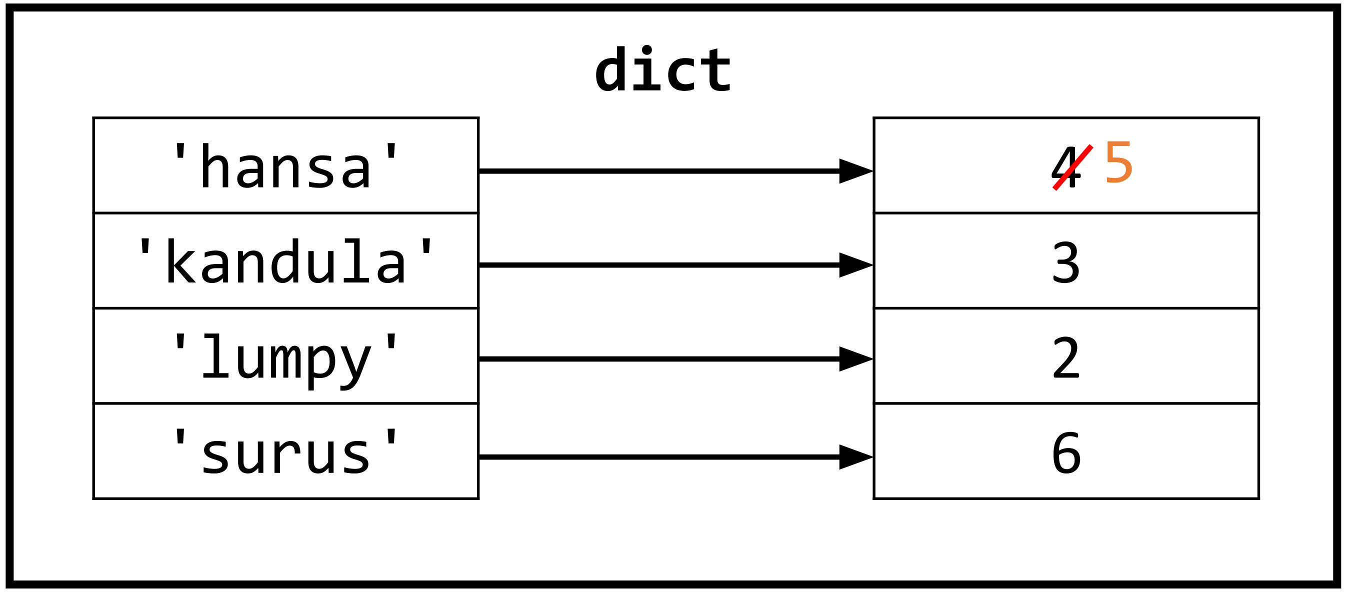
```
a = d['hansa']
a → 4
```

- Can “set” the values

```
d['hansa'] = 5
```

- Error if key not in the dictionary for get

```
b = d['karzo']
```



Keys are unique, each key stores a single value
(this value can be a list, tuple or another collection)

Anatomy of a Dictionary

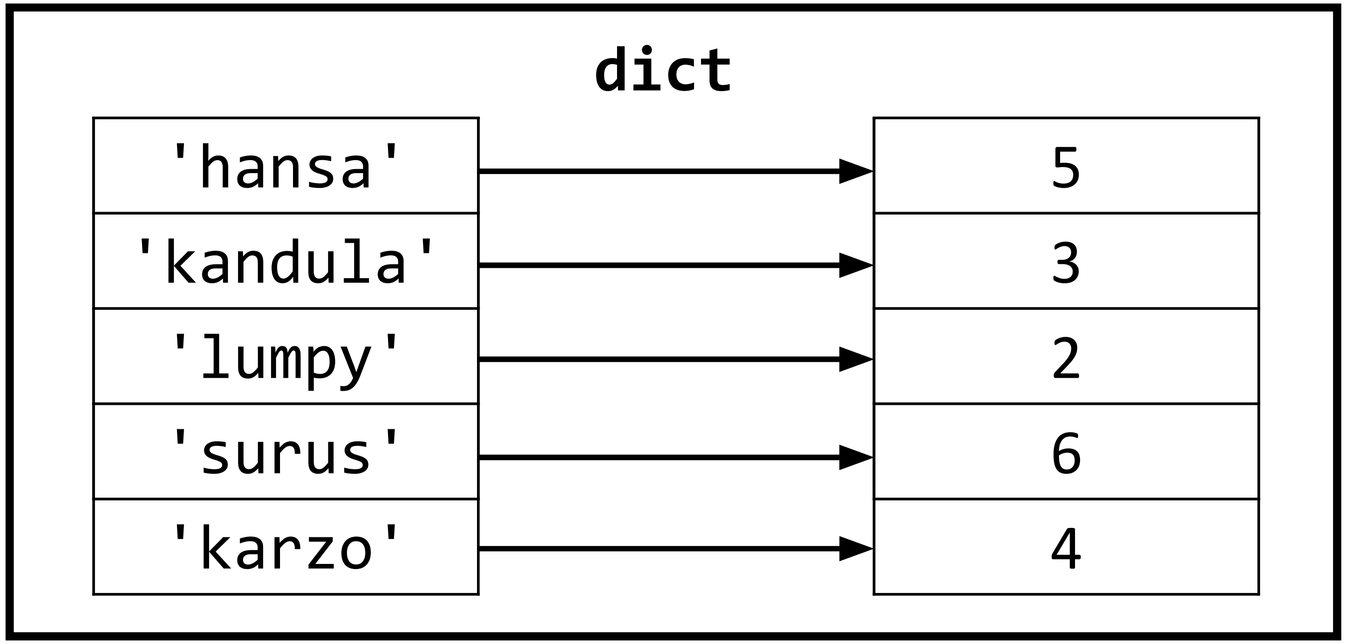
- Can create a new key-value pair

```
d['karzo'] = 4
```

- Can check if a key exists

```
'hansa ' in d
→ True
'pasa' in d
→ False
'pasa' not in d
→ True
```

- Common pattern: Check if key is present. If it is, do something. If it isn't, do something else.



More Dictionary Examples

```
# Create an empty dictionary:
d = {}

# Add the key-value pair
d['hansa'] = 1

# Update the key-value pair (note that get and set done at the same line)
d['hansa'] += 2
d['hansa'] → 3

# Keys and values do not have to be of the same type
d[1.34] = 'Random Float'
```

$$\texttt{d} \rightarrow \{\underbrace{\texttt{'hansa':3}}_{\substack{\text{string key}\\ \text{integer value}}},\ \underbrace{\texttt{1.34:'Random Float'}}_{\substack{\text{float key}\\ \text{string value}}}\}$$

Built-in Methods for Dictionaries

- `len`: Returns the number of key-value pairs inside the dictionary
- `del` statement: removes key-value pair
  - General format: `del dictionary[key]`
- `min` and `max`: They work on keys, but the keys must be comparable (e.g. if you have both string and integer type keys it won’t work)

Dictionary Methods: Reference Slide

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys(keys)	Returns a dictionary with the specified keys and value
get(key, default_value)	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop(key, default_value)	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
update(pair_iterable)	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Some Useful Dictionary Methods

- `keys()`: Returns an iterable data type that holds the **keys**
- `values()`: Returns an iterable data type that holds the **values**
- `items()`: Returns an iterable data type that holds the **key-value pairs** as tuples

Accessing Keys

```
d = {'Koc': 27, 'Bogazici': 157, 'METU': 64}
d.keys() → dict_keys(['Koc', 'Bogazici', 'METU'])
```

iterable collection of all the keys.
iterable means it can be used in foreach

```
list(d.keys()) → ['Koc', 'Bogazici', 'METU']
```

You can use list() to convert d.keys() into a list

Iterating over the Keys

```
d = {'Koc': 27, 'Bogazici': 157, 'METU': 64}

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

Output:

```
Koc
Bogazici
METU
```

Accessing Values and the Key-Value Pairs

```
d = {'Koc': 27, 'Bogazici': 157, 'METU': 64}
d.values() → dict_values([27, 157, 64])

list(d.values()) → [27, 157, 64]

d.items()
→ dict_items([('Koc', 27), ('Bogazici', 157), ('METU', 64)])
```

Iterating over the Values

```
d = {'Koc': 27, 'Bogazici': 157, 'METU': 64}

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

Output:

```
27
157
64
```

Iterating over the Items

```
d = {'Koc': 27, 'Bogazici': 157, 'METU': 64}
```

Iterating over tuples and unpacking

```
for university, age in d.items():
    print(university, 'is', age, 'years old')
```

Output:

```
Koc is 27 years old
Bogazici is 157 years old
METU is 64 years old
```

Side note: Print automatically concatenates, with a single space in between, the inputs separated by commas

Sorting Keys and Values

```
d = {'Koc': 27, 'Bogazici': 157, 'METU': 64}

sorted(d.keys()) → ['Bogazici', 'Koc', 'METU']

sorted(d.values()) → [27, 64, 157]
```