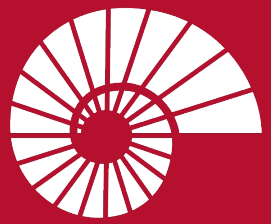


KOÇ
ÜNİVERSİTESİ

10 – Dictionaries

COMP 125 Programming with Python

Recording Disclaimer

The synchronous sessions are recorded (audiovisual recordings). The students are not required to keep their cameras on during class.

The audiovisual recordings, presentations, readings and any other works offered as the course materials aim to support remote and online learning. They are only for the personal use of the students. Further use of course materials other than the personal and educational purposes as defined in this disclaimer, such as making copies, reproductions, replications, submission and sharing on different platforms including the digital ones or commercial usages are strictly prohibited and illegal.

The persons violating the above-mentioned prohibitions can be subject to the administrative, civil, and criminal sanctions under the Law on Higher Education Nr. 2547, the By-Law on Disciplinary Matters of Higher Education Students, the Law on Intellectual Property Nr. 5846, the Criminal Law Nr. 5237, the Law on Obligations Nr. 6098, and any other relevant legislation.

The academic expressions, views, and discussions in the course materials including the audio-visual recordings fall within the scope of the freedom of science and art.

How can we store and organize data?

- **Collection**: A data structure used to store multiple values in a single unit
  - In most other programming languages, the most basic collection is an array, which stores values of the same type

- So far, we have seen **lists** and **tuples** for general data storage
  - Their items are accessed by their indices, which are **integer values from 0 to len() – 1**

```
L = [11, 22, 33, 44]
T = (1, 2, 3)
L[0], T[2]
```

- Some data types can be easily organized into related pairs.
  - e.g. phone numbers: John Doe -> 5161718
  - We want to access the phone number of a person via their name
  - But we cannot use a name (string) as an index.

Dictionaries

- A collection (sometimes called container) data type that maps "**keys**" to their associated "**values**"
- Defined using curly brackets: {key1:value1, key2:value2, …}

```
ids = {'Cigdem': 1111, 'Gunduz': 2222, 'Demir': 3333}
print(ids['Cigdem'])
print(ids)
ids['Gunduz'] =  4444
print(ids)
```

```
1111
{'Cigdem': 1111, 'Gunduz': 2222, 'Demir': 3333}
{'Cigdem': 1111, 'Gunduz': 4444, 'Demir': 3333}
```

- The values in a dictionary can be objects of any type
- The keys must be **immutable** objects (e.g., they can be strings, integers, floating point values, and tuples but cannot be lists)

Dictionaries (more examples)

1. Storing the dimensions and properties of a furniture

```
table = {'width' : 2.5, 'height' : 5, 'length' : 2, 'color' : 'brown'}
print(table)
print(table['color'])
print(table['width'])
table['color'] = 'red'
print(table['color'])
```

```
{'width': 2.5, 'height': 5, 'length': 2, 'color': 'brown'}
brown
2.5
red
```

2. Labeling points in the 2D Cartesian coordinate system

```
markers = {(0, 0.5) : '*', (1.5, 1) : '+', (4, 5) : '*', (1, -3.5) : '*'}
print(markers)
print(markers[(1.5, 1)])
markers[(1.5, 1)] = 'red'
print(markers[(1.5, 1)])
```

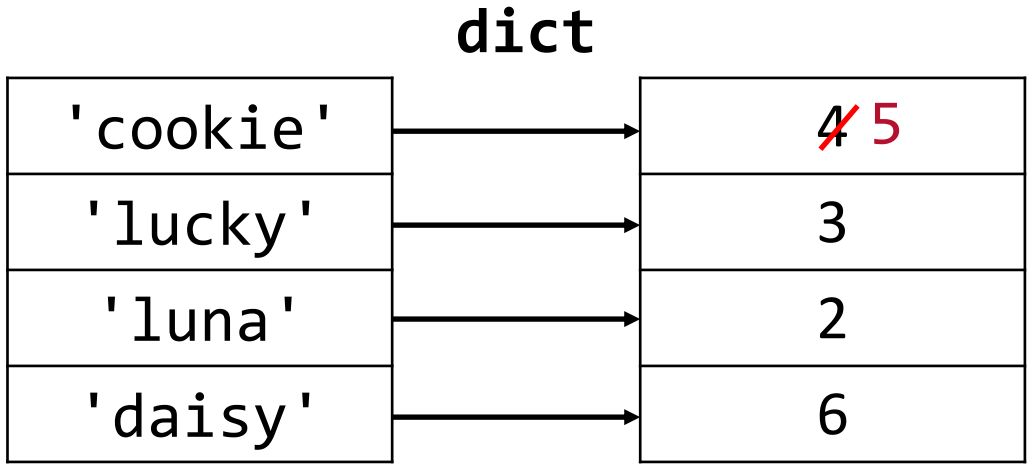
```
{(0, 0.5): '*', (1.5, 1): '+', (4, 5): '*', (1, -3.5): '*'}
+
red
```

Dictionaries

- Let's define a dictionary of pets and the number of times they are fed in a day

```
d = {'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6}
```

- Let's visualize



Can **"get"** (retrieve) the value

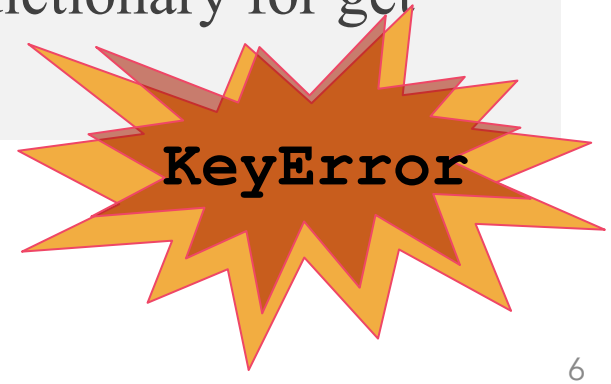
```
a = d['cookie']
a → 4
```

Can **"set"** the values

```
d['cookie'] = 5
```

Error if key not in the dictionary for get

```
b = d['bailey']
```



Dictionaries

- Can create a new key-value pair

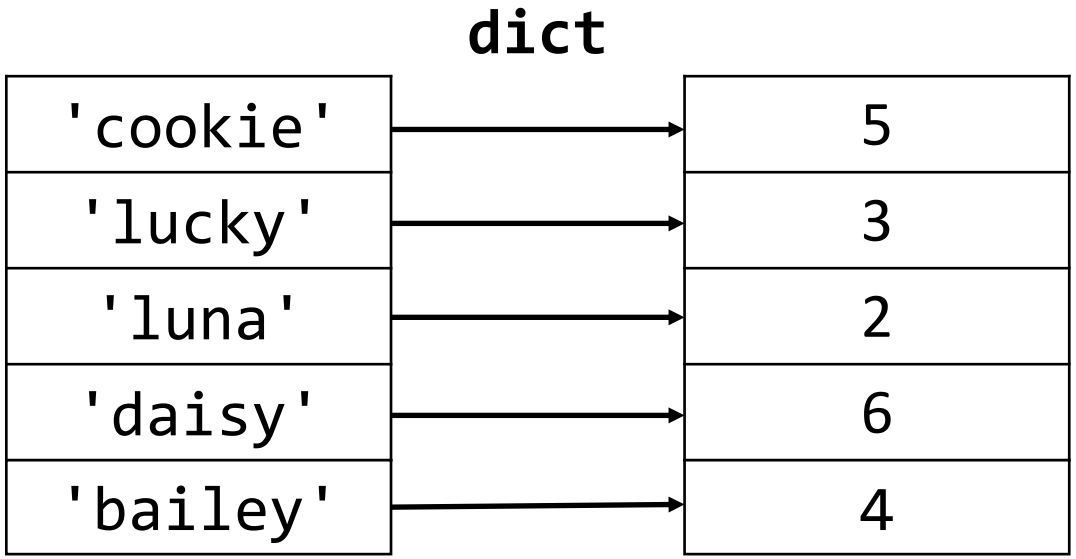
```
d['bailey'] = 4
```

- Can check if a key exists

```
'cookie' in d
→ True

'hunter' in d
→ False

'hunter' not in d
→ True
```



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

```
table = {'width' : 2.5, 'height' : 5, 'length' : 2, 'color' : 'brown'}
print(table)
print(table.keys())
print(table.values())
print(table.items())
```

```
{'width': 2.5, 'height': 5, 'length': 2, 'color': 'brown'}
dict_keys(['width', 'height', 'length', 'color'])
dict_values([2.5, 5, 2, 'brown'])
dict_items([('width', 2.5), ('height', 5), ('length', 2), ('color', 'brown')])
```

Key selection

- We strongly recommend you to use unique keys (case sensitive) in a single dictionary to prevent unwanted side effects
- However, you may use the same keys in different dictionaries

```
table = {'color' : 'red', 'height' : 10}
chair = {'color' : 'green', 'COLOR' : 14}

print(table)
print(table.keys())
print(table.values())
print(table.items())

print(chair)
print(chair.keys())
print(chair.values())
print(chair.items())
```

```
{'color': 'red', 'height': 10}
dict_keys(['color', 'height'])
dict_values(['red', 10])
dict_items([('color', 'red'), ('height', 10)])

{'color': 'green', 'COLOR': 14}
dict_keys(['color', 'COLOR'])
dict_values(['green', 14])
dict_items([('color', 'green'), ('COLOR', 14)])
```

Dictionaries (more examples)

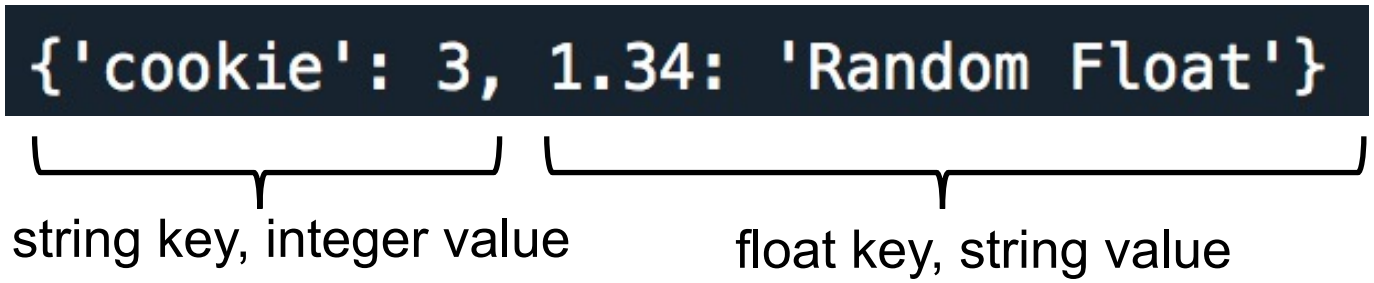
```
# Create an empty dictionary (both methods work)
d = {}
d = dict()

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

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

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

print(d)
```



Iterating over keys

```
how_old = {'Koc': 28, 'Bogazici': 158, 'METU': 65}

# keys() method returns an iterable collection of all keys
# iterable means it can be used in a for loop
how_old.keys()
→ dict_keys(['Koc', 'Bogazici', 'METU'])

# You can ase use list() to convert d.keys() into a list
list(how_old.keys())
→ ['Koc', 'Bogazici', 'METU']
```

Iterating over keys

```
how_old = {'Koc': 28, 'Bogazici': 158, 'METU': 65}

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

Output:

```
Koc
Bogazici
METU
```

Iterating over values

```
how_old = {'Koc': 28, 'Bogazici': 158, 'METU': 65}

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

Output:

```
28
158
65
```

Iterating over items

```
how_old = {'Koc': 28, 'Bogazici': 158, 'METU': 65}

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

Output:

```
Koc is 28 years old
Bogazici is 158 years old
METU is 65 years old
```

Sorting keys, values, and items

```
how_old = {'Koc': 28, 'Bogazici': 158, 'METU': 65}

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

sorted(how_old.values())
→ [28, 65, 158]

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

Built-in methods for dictionaries

- **`len()`:** Returns the number of key-value pairs inside the dictionary

- **`del statement`:** Removes a key-value pair

```
del dictionary[key]
```

```
d = {'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6.3}
print(len(d))
print(d)

del d['lucky']
print(len(d))
print(d)
```

```
4
{'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6.3}
3
{'cookie': 4, 'luna': 2, 'daisy': 6.3}
```

Built-in methods for dictionaries

- **`min()`** and **`max()`**: Find minimum and maximum keys/values/items, which must be comparable

```
d = {'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6.3}
print(max(d))
print(max(d.keys()))
print(max(d.values()))
print(max(d.items()))
```

```
luna
luna
6.3
('luna', 2)
```

Built-in methods for dictionaries

- **`min()`** and **`max()`**: Find minimum and maximum keys/values/items, which must be comparable

```
table = {'width' : 2.5, 'height' : 5, 'color' : 'brown'}
another = {'width' : 2.5, 34 : 'istanbul'}

print(max(table))
print(max(table.keys()))
print(max(table.items()))

# All of the following will give an error
# TypeError: '>' not supported between instances of 'str' and 'int'

print(max(table.values()))
print(max(another))
print(max(another.keys()))
print(max(another.items()))
```

```
width
width
('width', 2.5)
```

More built-in methods (summary)

Method	Description
`clear()`	Removes all the elements from the dictionary
`copy()`	Returns a copy of the dictionary
`get(key)`	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
`values()`	Returns a list of all the values in the dictionary
`pop(key)`	Removes the element with the specified key
`popitem()`	Removes the last inserted key-value pair
`dict.fromkeys(keys, value)`	Returns a dictionary with the specified keys and value
`update(pair_iterable)`	Updates the dictionary with the specified key-value pairs

get() and clear() methods

get()

```
d = {'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6}
v1 = d.get('cookie')
v2 = d.get('oscar')
v3 = d.get('cookie', -100)
v4 = d.get('oscar', 0)
print(v1)
print(v2)
print(v3)
print(v4)
```

Default value to return if the key is not found.

```
4
None
4
0
```

clear()

```
d = {'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6}
d.clear()
print(d)
```

```
{}
```

pop() method

pop()

```
d = {'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6}

v1 = d.pop('lucky')
print('Removed value: ', v1)
print('After pop: ', d)

v2 = d.pop('luna', 0)
v3 = d.pop('bailey', 0)
print(v2)
print(v3)

v4 = d.pop('bailey')
```

```
Removed value:  3
After pop:  {'cookie': 4, 'luna': 2, 'daisy': 6}
2
0
```

```
    v4 = d.pop('bailey')

KeyError: 'bailey'
```

update() method

update()

```
d1 = {'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6}
d2 = {'bailey': 4, 'oscar':5, 'cookie': 99}
d1.update(d2)
print(d1)
```

```
{'cookie': 99, 'lucky': 3, 'luna': 2, 'daisy': 6, 'bailey': 4, 'oscar': 5}
```

fromkeys() method

fromkeys()

```
keys = ['a', 'e', 'i', 'o', 'u']
value1 = 1
vowels1 = dict.fromkeys(keys, value1)

value2 = 'next'
vowels2 = dict.fromkeys(keys, value2)

value3 = (1, 2)
vowels3 = dict.fromkeys(keys, value3)

print(vowels1)
print(vowels2)
print(vowels3)
```

```
{'a': 1, 'e': 1, 'i': 1, 'o': 1, 'u': 1}
{'a': 'next', 'e': 'next', 'i': 'next', 'o': 'next', 'u': 'next'}
{'a': (1, 2), 'e': (1, 2), 'i': (1, 2), 'o': (1, 2), 'u': (1, 2)}
```

Copying dictionaries

- The `copy()` function and the `=` operator are different

```
d = {'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6}
dcopy1 = d
print(dcopy1)
dcopy1['oscar'] = 5
print(d)
print(dcopy1)
```

```
{'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6}
{'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6, 'oscar': 5}
{'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6, 'oscar': 5}
```

```
d = {'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6}
dcopy2 = d.copy()
print(dcopy2)
dcopy2['oscar'] = 5
print(d)
print(dcopy2)
```

```
{'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6}
{'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6}
{'cookie': 4, 'lucky': 3, 'luna': 2, 'daisy': 6, 'oscar': 5}
```

Example

- Write a function that finds all unique words in a given text. This function should store each unique word and its count as an item in a dictionary.

For example, for the following text

‘In midterm one I lost two points from question one and one point from question two’

It will produce the following dictionary

```
{'In': 1,         'midterm': 1, 'one': 3,     'I': 1,
 'lost': 1,       'two': 2,     'points': 1,  'from': 2,
 'question': 2,   'and': 1,     'point': 1}
```

```
def unique_words(text):
    s = text.split()
    d = {}

    for word in s:
        # Check if the word is already in dictionary
        if word in d:
            # Increment count of word by 1
            d[word] += 1
        else:
            # Add the word to dictionary with count 1
            d[word] = 1
    return d

def main():
    text = '''In midterm one I lost two points from
           question one and one point from question two'''

    d = unique_words(text)
    print(d)

main()
```

Example

- Now, add another function that returns the most frequent word in a given text. You may assume that the given text is not empty.

```
def find_most_frequent(text):
    d = unique_words(text)
    max_count = 0
    for i in d.keys():
        if d[i] > max_count:
            max_count = d[i]
            most_frequent = i
    return most_frequent

def main():
    text = '''In midterm one I lost two points from
           question one and one point from question two'''

    most_frequent = find_most_frequent(text)
    print(most_frequent)

main()
```

Example

- How many movies did you watch last year?
- Collect the data from all students (or we can just generate some random data)
- Analyze it by calculating the frequencies
- Store in a dictionary {number of movies watched: frequency}
  - See solution: 10-movies.py

Dictionary Comprehension

- An efficient and compact way to create dictionaries

  Syntax:

  new_dict = {*key:value_expr* for *member* in *iterable (if condition)*}

```
squares = {n : n**2 for n in range(1,6)}
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

- Dict comprehension with conditional

```
even_squares = {n : n**2 for n in range(1,6) if n % 2 == 0}
```

```
{2: 4, 4: 16}
```

Dictionary Comprehension

Building a dictionary of month days:

```
mnames = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
          'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
mdays = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

months = {}
for name, days in zip(mnames, mdays):
    months[name] = days
```

Alternative way with comprehension:

```
mnames = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
          'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
mdays = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

months = {name:days for name, days in zip(mnames, mdays)}
```

months =

```
{'Jan': 31, 'Feb': 28, 'Mar': 31, 'Apr': 30, 'May': 31, 'Jun': 30, 'Jul':
31, 'Aug': 31, 'Sep': 30, 'Oct': 31, 'Nov': 30, 'Dec': 31}
```