

Data Structures in Python

2. Dictionaries

Prof. Moheb Ramzy Girgis
Department of Computer Science
Faculty of Science
Minia University

Dictionaries

- **Lists** and **tuples** are convenient for storing collections of data, where an element within a list or a tuple is located based on its position (index).
- A Python **dictionary** is an associative container which permits access based on a **key**, rather than an **index**.
- Unlike an index, a key is not restricted to an integer expression.
- The following interactive sequence creates a simple dictionary that uses string keys:

```
>>> d = {'Fred': 44, 'Tom': 56, 'Jim': 32}
>>> d
{'Fred': 44, 'Tom': 56, 'Jim': 32}
>>> d['Bob'] = 60
>>> d
{'Fred': 44, 'Tom': 56, 'Jim': 32, 'Bob': 60}
>>> d['Tom']
56
```

Data Structures in Python - Prof. Moheb Ramzy
Girgis Dept. of Computer Science - Faculty of
Science Minia University

Dictionaries

```
>>> d['Sam']
Traceback (most recent call last):
  Python Shell, prompt 6, line 1
builtins.KeyError: 'Sam'
>>> d[0]
Traceback (most recent call last):
  Python Shell, prompt 8, line 1
builtins.KeyError: 0
```

- Notice that, unlike a *list* which uses square brackets ([]), the contents of a *dictionary* appear within curly braces ({}).
- The elements of a dictionary are *key:value* pairs, i.e. every *key* has an associated *value*.
- To access an element within a dictionary, we use square brackets exactly as we would with a list.
- The dictionary `d` from the interactive sequence above pairs the key 'Fred' with the value 44, the key 'Tom' with the value 56, and so on.

Science Minia University

3

Dictionaries

- The following assignment statement associates a value with a key:

```
d['Bob'] = 60    # Associate value 60 with key 'Bob'
```
- The string 'Bob' is the key, and 60 is its associated value.
- If the key within the square brackets does not exist in the dictionary, the statement adds the key and pairs it with the value on the right of the assignment operator.
- If the key already exists in the dictionary, the statement replaces the value previously associated with the key with the new value on the right of the assignment operator.
- To access a value with a given key, this key must be a valid key or the program will raise an exception.
- A *valid key* is a key that is present in the dictionary.
- In the interaction sequence above 'Tom' is a valid key but 'Sam' and 0 are not.

Data Structures in Python - Prof. Moheb Ramzy
 Girgis Dept. of Computer Science - Faculty of
 Science Minia University

4

Dictionaries

- As we see, the interpreter generated a **KeyError** exception when we attempted to use an invalid key.
- We can check to see if a key is present in a dictionary with the **in** operator:

```
if 'Fred' in d:          # Check to see if 'Fred' is a valid key
    print(d['Fred']) # Print the value associated with key 'Fred'
else:
    print('\Fred\' is not a key in d') # Warn user of missing key
```
- A dictionary key may be of any immutable type, such as: integers, floating-point numbers, strings, Booleans, and tuples.
- Since lists are mutable objects, a list may not be a key.
- A dictionary is a **mutable** object, so a dictionary cannot use itself or another dictionary object as a key.
- A value within a dictionary may be any valid Python type, immutable or mutable

5

Dictionaries

- The keys within a given dictionary may be of mixed types.
- Consider the following interactive sequence:

```
>>> s = {}
>>> s[8] = 44
>>> s[8]
44
>>> s['Alpha'] = 'up'
>>> s['Alpha']
'up'
>>> s[True] = 'right'
>>> s[True]
'right'
>>> s[10 < 20]
'right'
>>> s['Beta'] = 100
>>> s
{8: 44, 'Alpha': 'up', True: 'right', 'Beta': 100}
>>> s[3.4] = True
```

Data Structures in Python - Prof. Moheb Ramzy
 Girgis Dept. of Computer Science - Faculty of
 Science Minia University

6

Dictionaries

```
>>> s
{8: 44, 'Alpha': 'up', True: 'right', 'Beta': 100, 3.4: True}
>>> s[2 == -2] = 'wrong'
>>> s[False]
'wrong'
>>> s
{8: 44, 'Alpha': 'up', True: 'right', 'Beta': 100, 3.4: True, False:
'wrong'}
>>> x = 8
>>> s[x]
44
>>> y = 15
>>> s[y] = 'down'
>>> lst = [1, 2, 3]
>>> s[17] = lst
>>> s
{8: 44, 'Alpha': 'up', True: 'right', 'Beta': 100, 3.4: True, False:
'wrong', 15: 'down', 17: [1, 2, 3]}
```

Girgis Dept. of Computer Science - Faculty of
Science Minia University

7

Dictionaries

- This interactive sequence reveals several dictionary characteristics:
 - The keys in a dictionary may have different types.
 - The values in a dictionary may have different types
 - The values in a dictionary may be mutable objects
- We can initialize a dictionary using the same syntax as the output that the print function displays.
- The following statement populates the dictionary d with four key:value entries:

```
d = {'Fred': 44, 'Tom': 56, 'Jim': 32, 'Bob': 60}
print(d)
```


- The code above prints: `{'Fred': 44, 'Tom': 56, 'Jim': 32, 'Bob': 60}`
- Observe that the *print* function neither lists the keys in lexicographical order nor lists the values in numerical order.
- Unlike in a list or other sequence type, the notions of order and position have no meaning within a dictionary.

8

Dictionaries


- The **keys** method of the **dictionary** class returns a sequence of all the keys in d.
- The following code demonstrates this:

```
d = {'Fred': 44, 'Tom': 56, 'Jim': 32, 'Bob': 60}
for k in d.keys():
    print(k, end=' ')
print()
```

 Fred Tom Jim Bob
- The following code produces the same output as iterating over d.keys():

```
for k in d:
    print(k, end=' ')
print()
```
- The **values** method of the **dictionary** class returns a sequence of all the values in d.
- The following code illustrates this:

```
for v in d.values():
    print(v, end=' ')
print()
```


 44 56 32 60

9

Dictionaries

- We can obtain a sequence of tuples of **key:value** pairs of a dictionary with the **items** method:

```
for k, v in d.items():
    print(k, v)
```



Fred	44
Tom	56
Jim	32
Bob	60
- As mentioned before, we can zip two lists together into a sequence of tuples using the **zip** function.
- We can use the **dict** function to create a dictionary of **key:value** pairs formed from the tuples, as the following interactive sequence shows:


```
>>> names = ['Fred', 'Tom', 'Jim', 'Bob']
>>> numbers = [4174, 2287, 5003, 2012]
>>> names
['Fred', 'Tom', 'Jim', 'Bob']
>>> numbers
[4174, 2287, 5003, 2012]
>>> d = dict(zip(names, numbers))
>>> d
{'Fred': 4174, 'Tom': 2287, 'Jim': 5003, 'Bob': 2012}
```

10

Dictionaries

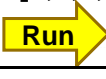
- Note that:
 - The elements of the list specified as the first actual parameter to **zip** become the dictionary keys, and the elements of the list specified as the second argument to **zip** form the values in the dictionary.
 - The first element from the **names** list is paired with the first element of the **numbers** list, the second element from **names** is paired with the second element of **numbers**, and so forth.
- The following code shows another way to display the contents of a dictionary:


```
d = {'Fred': 44, 'Tom': 56, 'Jim': 32, 'Bob': 60}
for k in d.keys():
    print ("d[" + k + "] = ", d[k], sep="")
```



d[Fred] = 44
 d[Tom] = 56
 d[Jim] = 32
 d[Bob] = 60
- We can use **dictionary comprehension** to build dictionaries:


```
myDict = {x: x**2 for x in [1,2,3,4,5]}
print(myDict)
```



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

11

Dictionaries

- A **dictionary** is sometimes called an **associative array** because its elements (values) are associated with keys instead of indices.
- The placement and lookup of an element within a dictionary uses a process known as **hashing**.
- A **hash function** maps a key to a location within the dictionary where the key's associated value resides.
- Python dictionaries are related to hash tables in computer science.
- The important thing to know about the hashing process is that it makes value lookup via a key very fast.
- When treated as a Boolean expression, the empty dictionary ({}) is interpreted as **False**, and any other dictionary is considered **True**.

Dictionaries

Dictionary Methods

Method	Description
<code>clear()</code>	Deletes all items from the dictionary.
<code>copy()</code>	Creates a copy of the dictionary.
<code>get(key [, falseValue])</code>	Returns the value associated with <i>key</i> . If <i>key</i> is not in the dictionary and if <i>falseValue</i> is specified, returns the specified value.
<code>has_key(key)</code>	Returns 1 if <i>key</i> is in the dictionary; returns 0 if <i>key</i> is not in the dictionary.
<code>items()</code>	Returns a list of tuples that are key-value pairs.
<code>keys()</code>	Returns a list of keys in the dictionary.
<code>setdefault(key [, falseValue])</code>	Behaves similarly to method <code>get</code> . If <i>key</i> is not in the dictionary and <i>falseValue</i> is specified, inserts the key and the specified value into dictionary.
<code>update(otherDictionary)</code>	Adds all key-value pairs from <i>otherDictionary</i> to the current dictionary.
<code>values()</code>	Returns a list of values in the dictionary.

Dictionaries

- The following interactive sequence exercises these dictionary methods:

```
>>> d = {'Fred': 44, 'Tom': 56, 'Jim': 32, 'Bob': 60}
>>> c = d.copy()
>>> c
{'Fred': 44, 'Tom': 56, 'Jim': 32, 'Bob': 60}
>>> d.get('Tom')
56
>>> d.get('Sam')
>>> d.get('Sam', 66)
66
>>> d
{'Fred': 44, 'Tom': 56, 'Jim': 32, 'Bob': 60}
>>> d.setdefault('sam', 66)
66
>>> d
{'Fred': 44, 'Tom': 56, 'Jim': 32, 'Bob': 60, 'sam': 66}
```

Dictionaries

```
>>> d.setdefault('sam')
66
>>> d.setdefault('sara')
>>> d.update({'Paul':87, 'Peter':90})
>>> d
{'Fred': 44, 'Tom': 56, 'Jim': 32, 'Bob': 60, 'sam': 66, 'sara': None,
'Paul': 87, 'Peter': 90}
>>> d.keys()
dict_keys(['Fred', 'Tom', 'Jim', 'Bob', 'sam', 'sara', 'Paul', 'Peter'])
>>> d.values()
dict_values([44, 56, 32, 60, 66, None, 87, 90])
>>> d.clear()
>>> d
{}
```

Data Structures in Python - Prof. Moheb Ramzy
Girgis Dept. of Computer Science - Faculty of
Science Minia University

15

Using Dictionaries

- You should use a dictionary when you need fast and convenient access to an element of a collection based on a search key rather than an index.
- **Example 1:** Consider the problem of implementing a simple telephone contact list.
- A contact list associates a name with a telephone number.
- In this example, a person or company's name is a unique identifier for a contact. So, the name is a key to that contact.
- We will look up a number based on a name.
- A Python dictionary is the ideal data structure for mapping keys to values.
- A dictionary allows for the fast retrieval of a value given its associated key.

Data Structures in Python - Prof. Moheb Ramzy
Girgis Dept. of Computer Science - Faculty of
Science Minia University

16

Using Dictionaries

- The following program uses a Python dictionary to implement a simple telephone contact database with a simple command line interface.

```
contacts = {} # The global telephone contact list
running = True
while running:
    command = input('A)dd D)delete L)ook up Q)uit: ')
    if command == 'A' or command == 'a' :
        name = input('Enter new name:')
        print('Enter phone number for', name, end=':')
        number = input()
        contacts[name] = number
    elif command == 'D' or command == 'd':
        name = input('Enter name to delete :')
        del contacts[name]
    elif command == 'L' or command == 'l':
        name = input('Enter name :')
        print(name, contacts[name])
```

Science Minia University

17

Using Dictionaries

```
elif command == 'Q' or command == 'q':
    running = False
elif command == 'dump': # Secret command
    print(contacts)
else:
    print(command, 'is not a valid command')
```



Sample Run

```
A)dd D)delete L)ook up Q)uit: a
Enter new name:Fred
Enter phone number for Fred:123456
A)dd D)delete L)ook up Q)uit: a
Enter new name:Tom
Enter phone number for Tom:246830
A)dd D)delete L)ook up Q)uit: a
Enter new name:Bob
Enter phone number for Bob:135790
A)dd D)delete L)ook up Q)uit: l
Enter name :Tom
Tom 246830
A)dd D)delete L)ook up Q)uit: d
Enter name to delete :Tom
A)dd D)delete L)ook up Q)uit: dump
{'Fred': '123456', 'Bob': '135790'}
A)dd D)delete L)ook up Q)uit: q
```

18

Using Dictionaries

- **Example 2:** The following program uses a dictionary to assist in translating some Spanish words into English:

```
translator = {'uno':'one', 'dos':'two', 'tres':'three', 'cuatro':'four',
'cinco':'five', 'seis':'six', 'siete':'seven', 'ocho':'eight'}
word = ''
while word != "": # Loop until user presses return by itself
    # Obtain word from the user
    word = input('Enter Spanish word:')
    if word in translator:
        print(translator[word])
    else:
        print('Unknown word!!!')
```

Sample Run

```
Enter Spanish word:seis
six
Enter Spanish word:dos
two
Enter Spanish word:abc
Unknown word!!!
Enter Spanish word:
Unknown word!!!
```

19

Counting with Dictionaries

- We have seen before programs that count one kind of things at a time, e.g. counting no. of vowels in a sentence.
- Sometimes we need to count more than one kind of things at a time. In this case, we need to use a separate variable for each count.
- For example, the following code counts the number of negative and nonnegative numbers in a list of numbers and returns a tuple with the results:

```
def count_neg_nonneg(nums):
    # Initialize counters
    neg_count, nonneg_count = 0, 0
    for num in nums:
        if num < 0:
            neg_count += 1
        else:
            nonneg_count += 1
    return neg_count, nonneg_count
```

20

Counting with Dictionaries

- Since we needed to count two different kinds of things, we had to use two separate counter variables.
- What if we need to count multiple kinds of things, but we cannot know ahead of time how many kinds of things there will be to count?
- The answer is this:
If all the things we need to count are immutable objects, like numbers or strings, we can use the objects as keys in a dictionary and associate with each key a count.
- **Example:** The following program reads the content of a text file containing words, then prints a count of each word.
 - Firstly, the user supplies the text file name.
 - To simplify things, the text file contains only words with no punctuation.

Data Structures in Python - Prof. Moheb Ramzy
Girgis Dept. of Computer Science - Faculty of
Science Minia University

21

Counting with Dictionaries

```

""" Uses a dictionary to count the number of occurrences of
each word in a text file. """
def main():
    """ Counts the words in a text file. """
    filename = input('Enter the name of a text file: ')
    counters = {} # Initialize counting dictionary
    with open(filename, 'r') as f: # Open the file for reading
        content = f.read() # Read in content of the entire file
        words = content.split() # Make list of individual words
    for word in words:
        word = word.upper() # Make the word all caps
        if word not in counters:
            counters[word] = 1 # First occurrence, add the counter
        else:
            counters[word] += 1 # Increment existing counter
    # Report the counts for each word
    for word, count in counters.items():
        print(word, count)
if __name__ == '__main__':
    main()

```

22

Counting with Dictionaries

- Assume the text file contains the following text:

The cat sat on the mat

Run

Enter the name of a text file: text.txt

```
THE 2
CAT 1
SAT 1
ON 1
MAT 1
```

- In the above program, since we cannot predict what words will appear in the document, we cannot use a separate variable for each counter.
- Instead, we use the user's words as keys in a dictionary. For each key in the dictionary we associate an integer value that keeps track of the number of times the word appears in the file.
- The following expression in the above program:
`content.split()`
exercises the *split* method of the *str* class that separates the long string composed of all the words in the file into separate strings.

Science Minia University

23

Counting with Dictionaries

- The *split* method divides the string based on whitespace (spaces, tabs, and newlines) and returns the individual words in a list.
- The following interactive sequence shows how the *split* method works:

```
>>> s = ' ABC def GHI JKLM-nop, aaa '
>>> s.split()
['ABC', 'def', 'GHI', 'JKLM-nop,', 'aaa']
```
- With no arguments, the *split* method splits the string based on whitespace (spaces, tabs, and newlines).
- The split function accepts an optional string parameter that contains the characters used to separate the words (or tokens); for example,

```
>>> x = 'ABC:xyz.122:prst'
>>> x.split(':')
['ABC', 'xyz.122', 'prst']
```

Eng. Dept. of Computer Science - Faculty of
Science Minia University

24

Counting with Dictionaries

- Suppose we have string containing comma-separated integer data. The string could contain spaces as well.
- The interactive interpreter reveals how we can split such a string into its comma-separated components:

```
s = " 85, 54 , 13,   17, 44 ,31  , 80, 35,   30, 54, 78  "
s.split(',')
[' 85', ' 54 ', ' 13', '   17', ' 44 ', '31  ', ' 80', ' 35', '   30', ' 54', ' 78  ']
```
- This leaves us a list of strings, many containing extraneous leading or trailing spaces (or both).
- We need to remove the spaces with the string *strip* method and then convert the results to integers.
- The following interactive session shows how to use *a one-line list comprehension* to do exactly what we need:

```
int_list = [int(x.strip()) for x in s.split(",")]
int_list
[85, 54, 13, 17, 44, 31, 80, 35, 30, 54, 78]
```

Girgis Dept. of Computer Science - Faculty of
Science Minia University

25

Grouping with Dictionaries

- Dictionaries are useful for grouping items.
- **Example:** The following program groups the words into lists based on the number of letters in the word. All the words containing only one letter are in one list, all the words containing two letters are in another list, etc.

```
""" Uses a dictionary to group the words in a text file according
to their length (number of letters). """
def main():
    """ Counts the words in a text file. """
    filename = input('Enter the name of a text file: ')
    groups = {} # Initialize grouping dictionary
    with open(filename, 'r') as f: # Open the file for reading
        content = f.read() # Read in content of the entire file
        words = content.split() # Make list of individual words
        for word in words:
            word = word.upper() # Make the word all caps
            # Compute the word's length
            size = len(word)
```

26

Grouping with Dictionaries

```

if size in groups:
    if word not in groups[size]: # Avoid duplicates
        groups[size] += [word] # Add the word to its group
    else:
        groups[size] = [word] # Add the word to a new group
# Show the groups
for size, group in groups.items():
    print(size, ': ', group)
if __name__ == '__main__':
    main()

```

- Assume the text file contains the following text:

The cat sat on a mat

Run

Enter the name of a text file: text.txt
 3 : ['THE', 'CAT', 'SAT', 'MAT']
 2 : ['ON']
 1 : ['A']

- Each key represents the length of all the strings in the list it oversees.