COMPSCI 101

Principles of Programming

Lecture 21 – Python dictionaries 1

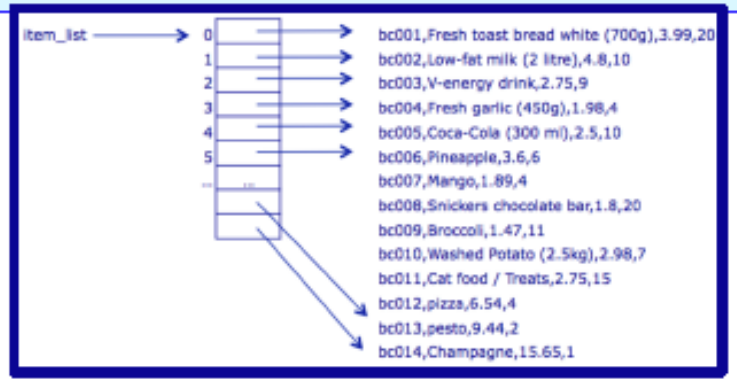
Learning outcomes

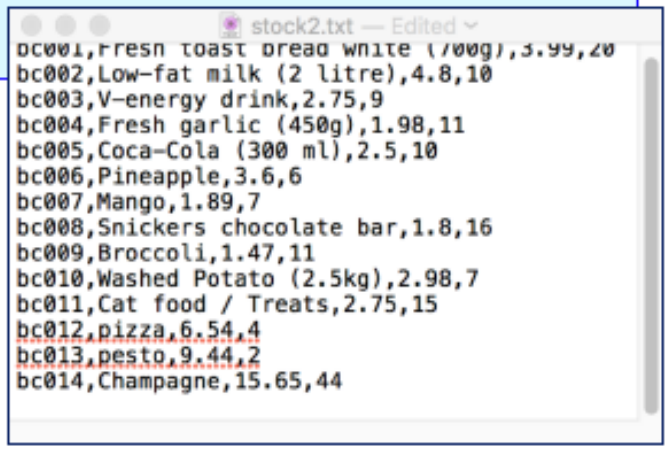
- At the end of this lecture, students should be able to:
  - understand what a dictionary is
  - create a dictionary object
  - add items to a dictionary
  - retrieve items from a dictionary
  - traverse the pairs in a dictionary

Recap

- Exercise from lecture 21 on input and output:

```
def save_stock(filename, list_of_items):
    outfile = open(filename, "w")
    for item in list_of_items:
        outfile.write(item + "\n")
    outfile.close()

def main():
    …
    save_stock("stock2.txt", items_list)

main()
```





Python dictionaries

- A dictionary is a mapping from a key to its associated data value.
  - Each key maps to a value.
  - The key has to be unique and an **immutable** object.
  - A phone book is an example of a mapping: the key is the person's name (plus address) and the associated value is their phone number.

You can think of a dictionary as a list of pairs, where the first element of the pair, the key, is used to retrieve the second element, the corresponding value.

- The key and its associated value is called a **key-value pair** or it can be called an **item**.

Creating an object of type dict

- Curly braces are used for dictionaries and {} is a dictionary which contains no key-value pairs, i.e., an empty dictionary:

```
def main():
   english_italian = {}
   print(english_italian)
   print(type(english_italian))

main()
```

```
{}
<class 'dict'>
```

- Another way to create an empty dictionary object is (does exactly the same thing as the code above):

```
def main():
   english_italian = dict()
   print(english_italian)
   print(type(english_italian))

main()
```

```
{}
<class 'dict'>
```

dict is a Python type

- Note that the name, `dict`, is a Python type and should not be used as a variable name.

```
def main():
   english_italian = dict()

main()
```

Creating a dictionary which contains pairs

- A dictionary object can be initialised with key-value pairs:
  - Each associated pair is separated by ':' and the pairs are separated by commas.

```
def main():
  english_italian = {"yes":"si", "bye":"ciao", "no":"no",
                     "maybe":"forse", "thank you":"grazie"}
  print(english_italian)

  contacts = {"Jill": 3456, "James": 3456, "Yi": 7654,
                                           "Syed": 6754}
  print(contacts)

main()
```

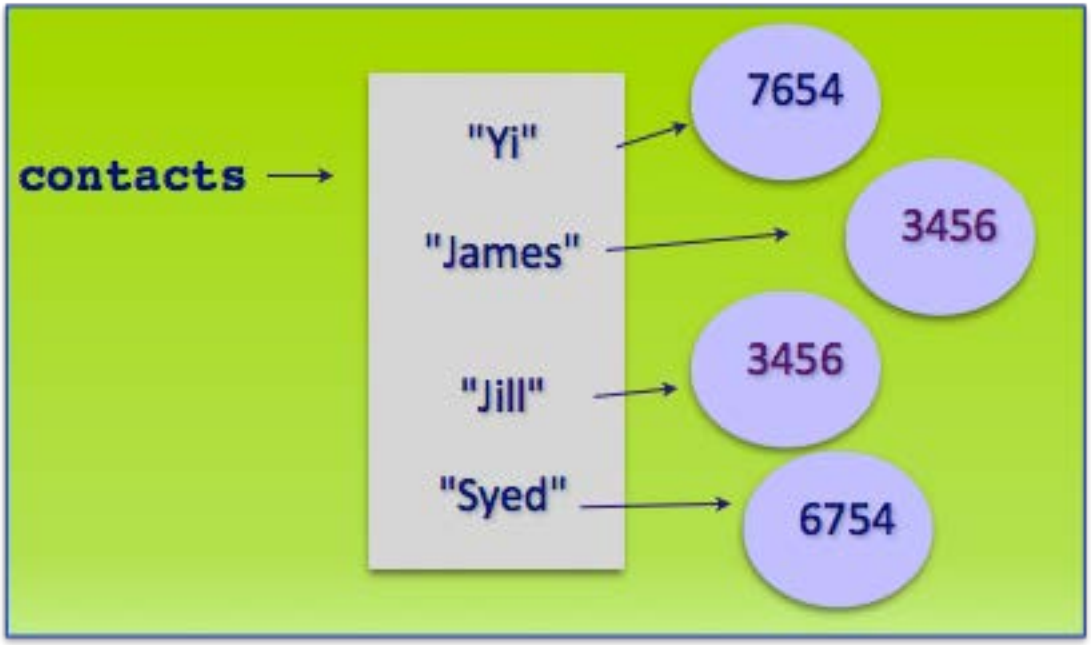
```
{'maybe': 'forse', 'bye': 'ciao', 'yes': 'si', 'no': 'no', 'thank you': 'grazie'}
{'Yi': 7654, 'Jill': 3456, 'Syed': 6754, 'James': 3456}
```

Note: the keys have to be unique but the associated values do not.

Visualising the dictionary

```
def main():
   contacts = {"Jill": 3456, "James": 3456, "Yi": 7654,
               "Syed": 6754}

   print(contacts)

main()
```

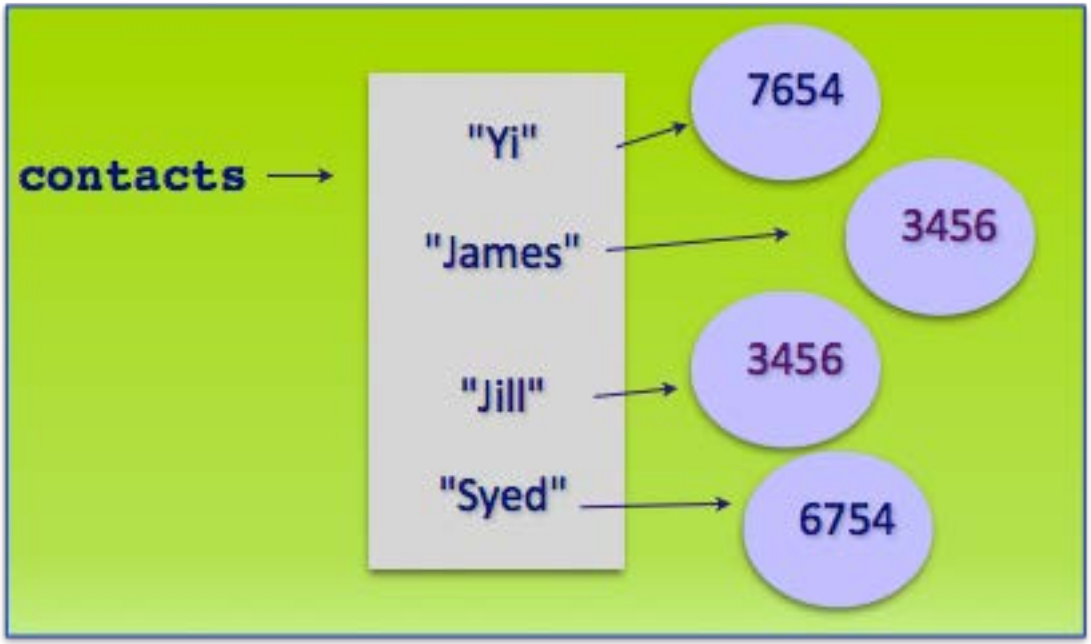
{'Jill': 3456, 'Syed': 6754, 'James': 3456, 'Yi': 7654}



Note: when the key-value pairs are printed, the order is not predictable.

The keys of the dictionary must be immutable

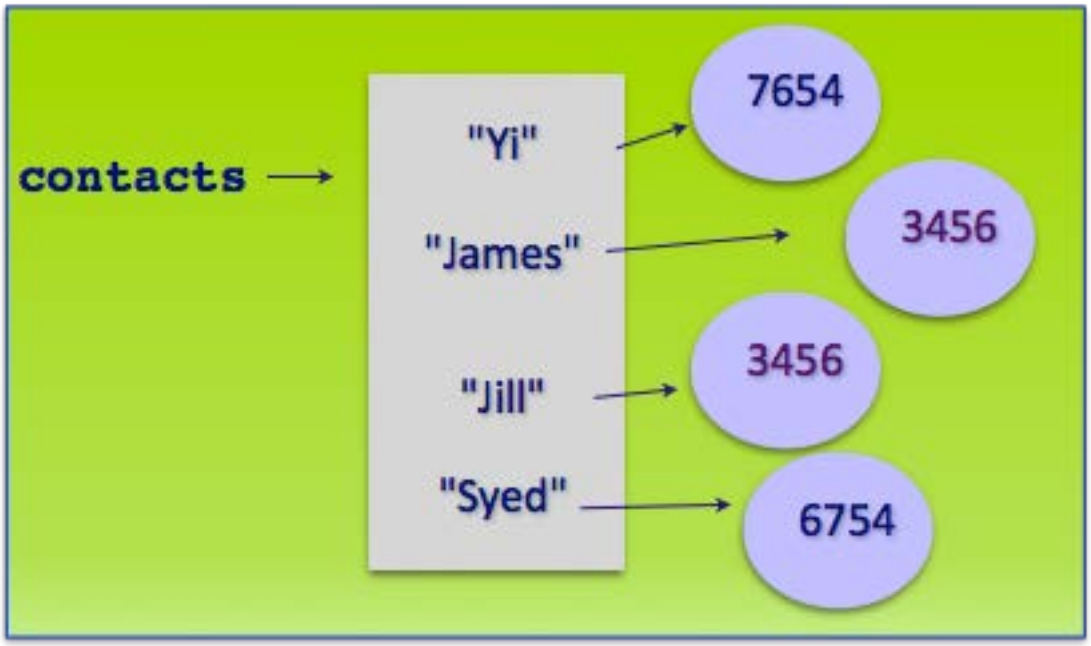
- The **keys** of a dictionary must be of a type which is **immutable** such as: string, int, tuple.
- The **keys** of a dictionary must be **unique**.
- The values can be of any type and they do not need to be unique.



Remember that lists are mutable and therefore dictionary keys cannot be of type list.

Dictionaries are not ordered structures

- Dictionary elements cannot be accessed using the index value. A dictionary is a collection of key:value pairs.
- There is no predictable order to the key:value pairs in a dictionary (the order may change as new pairs are added and removed).

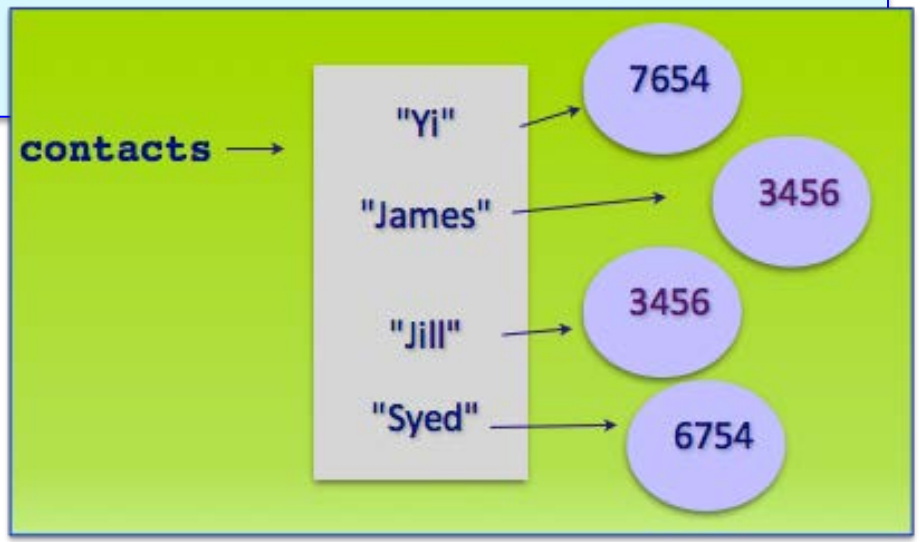


Access the value associated with a key

- The value associated with a certain key can be accessed using square brackets (enclosing the key):

```
def main():
    contacts = {"Jill": 3456, "James": 3456, "Yi": 7654,
                "Syed": 6754}
    name1 = "Jill"
    name2 = "James"
    print(name1, "is at extension:", contacts[name1])
    if contacts[name1] == contacts[name2]:
        print(name2, "has the same extension")

main()
```

```
Jill is at extension: 3456
James has the same extension
```

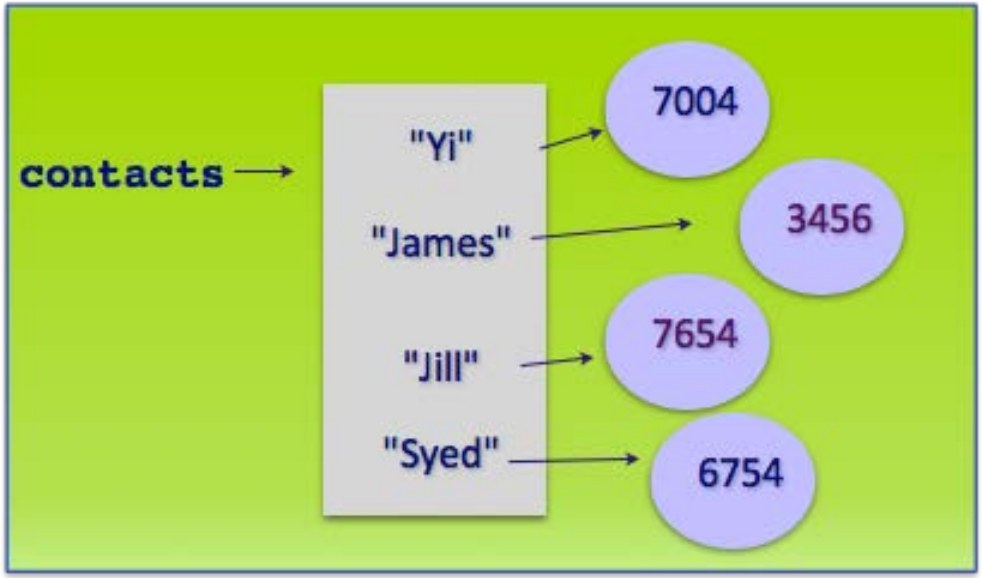


Changing the associated value in a dictionary

- The associated value of a pair can be changed by assigning a different value to the dictionary key. This replaces the old value.

```
def main():
    contacts = {"Jill": 3456, "James": 3456, "Yi": 7654,
                "Syed": 6754}
    contacts["Jill"] = 7654
    contacts["Yi"] = 7004
    print(contacts)

main()
```

```
{'Syed': 6754, 'Yi': 7004, 'James': 3456, 'Jill': 7654}
```



Adding a pair to the dictionary

- Key-value pairs can be added to the dictionary using assignment statements:

```
def main():
   contacts = {"Jill": 3456, "James": 3456, "Yi": 7654,
               "Syed": 6754}
   contacts["Mark"] = 7654
   contacts["Jerry"] = 7004

   print(contacts)

main()
```

```
{'Jerry': 7004, 'Syed': 6754, 'Yi': 7654, 'Mark': 7654,
'Jill': 3456, 'James': 3456}
```

Note: when the key-value pairs are printed, the order is not predictable.

The number of key-value pairs in a dictionary

- The len() function can be used with a dictionary object to find out how many key-value pairs are currently in the dictionary:

```
def main():
  contacts = {"Jill": 3456, "James": 3456, "Yi": 7654,
              "Syed": 6754}
  print(len(contacts), "in dictionary")
  contacts["Yi"] = 7654
  contacts["Jerry"] = 7004
  print(len(contacts), "in dictionary")

main()
```

```
4 in dictionary
5 in dictionary
```

Check if a key is in the dictionary

- The '**in**' operator can be used to check if a **key** is in the dictionary:

```
def main():
   contacts = {"Jill": 3456, "James": 3456, "Yi": 7654,
               "Syed": 6754}
   name = "Jack"
   if name in contacts:
      print(name, "is at extension:", contacts[name])
   else:
      contacts[name] = 0

   if name in contacts:
      print(name, "is at extension:", contacts[name])

   print(contacts)

main()
```

```
Jack is at extension: 0
{'Jill': 3456, 'James': 3456, 'Yi': 7654, 'Syed': 6754, 'Jack': 0}
```

The in operator with dictionaries

- An error is raised when accessing a key which is not in the dictionary. Usually you test before accessing a key-value pair.

```
1 def main():
2   contacts = {"Jill": 3456, "James": 3456, "Yi": 7654,
                "Syed": 6754}
3   if "Jill" in contacts:                  #Test first
4      print("Jill", "-", contacts["Jill"])

5   print(contacts["Izzy"])
6 main()
```

```
Jill - 3456
Traceback (most recent call last):
  File "LectureCode.py", line 5, in <module>
    print(contacts["Izzy"])
KeyError: 'Izzy'
```

Traversing the pairs in the dictionaries

- Use a **for … in** loop to traverse (visit) each key in the dictionary:

```
def main():
   contacts = {"Jill": 3456, "James": 3456, "Yi": 7654,
               "Syed": 6754}
   for name in contacts:
      print(name , "-", contacts[name])

main()
```

Same code

```
Yi - 7654
Jill - 3456
Syed - 6754
James - 345
```

```
def main():
   contacts = {"Jill": 3456, "James": 3456, "Yi": 7654,
               "Syed": 6754}
   for key in contacts:
      print(key, "-", contacts[key])

main()
```

Exercise

- "Story.txt" is a text file. The following program reads the text from the file, converts it to lower case, and creates a dictionary of all the unique words which start with a vowel ("a", "e", "i","o", "u"). Note: the key is the vowel and each word is added to an associated list (the list grows as the text is processed).

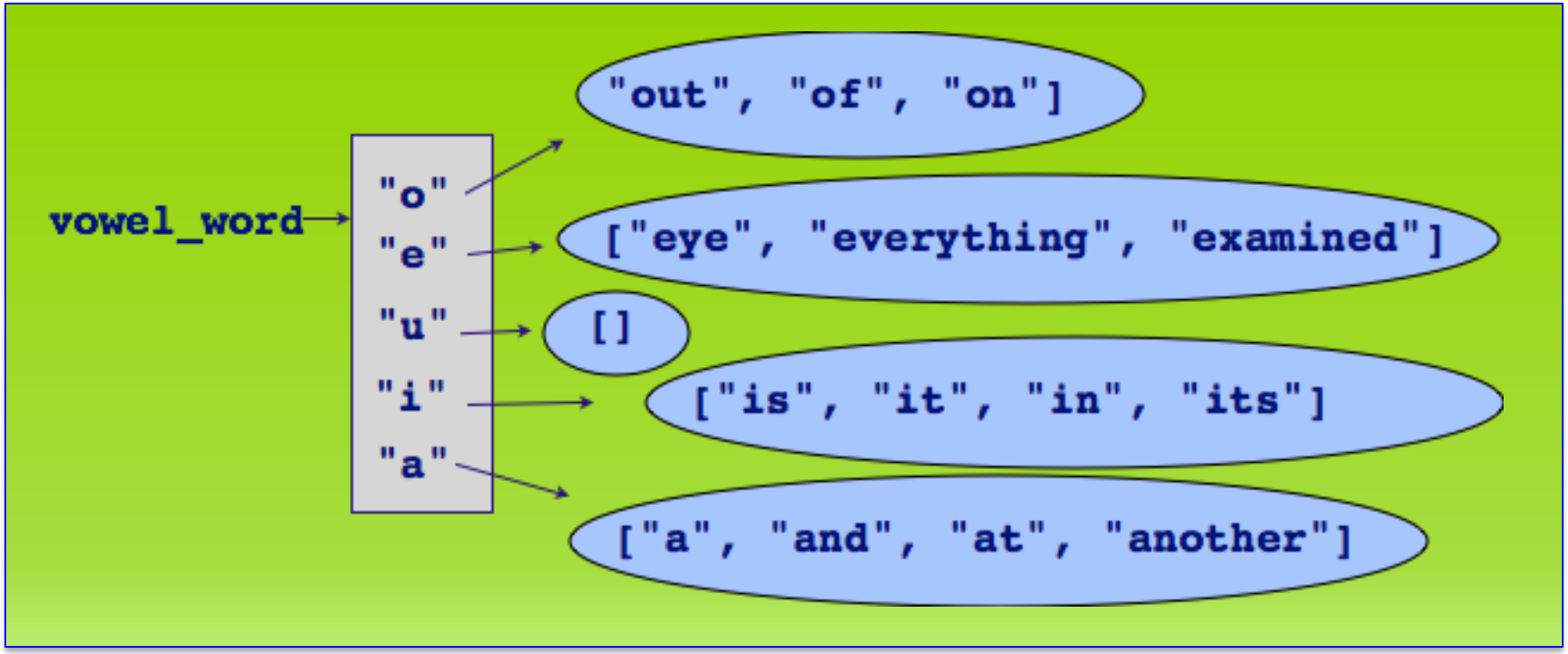
```
def main():
   vowel_words_dict = get_dictionary_from_file_words("Story.txt")
   display_results(vowel_words_dict)

def get_dictionary_from_file_words(filename): #complete the code
def display_results(vowel_words): #complete the code

main()
```

```
o - ['on', 'one', 'old', 'only', 'of', 'opportunity', 'official', 'out']
e - ['elder', 'excited', "elder's"]
u - []
i - ['indian', 'in', 'if']
a - ['apollo', 'astronaut', 'a', 'and', 'across', 'asked', 'are', 'astronauts', 'after', 'an']
```

Exercise

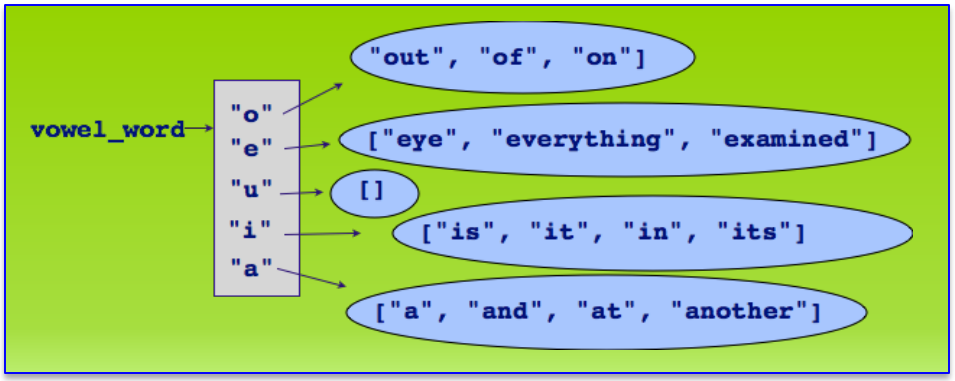
Story.txt

A small trouble is like a pebble. Hold it too close to your eye, and it fills the whole world and puts everything out of focus. Hold it at the proper distance, and it can be examined and properly classified. Throw it at your feet and it can be seen in its true setting, just another tiny bump on the pathway of life.



Exercise

```
def get_dictionary_from_file_words(file_name):
```



Story.txt

A small trouble is like a pebble. Hold it too close to your eye, and it fills the whole world and puts everything out of focus. Hold it at the proper distance, and it can be examined and properly classified. Throw it at your feet and it can be seen in its true setting, just another tiny bump on the pathway of life.

Exercise

```
def display_results(vowel_words_dict):
```

```
o - ['on', 'one', 'old', 'only', 'of', 'opportunity', 'official', 'out']
e - ['elder', 'excited', "elder's"]
u - []
i - ['indian', 'in', 'if']
a - ['apollo', 'astronaut', 'a', 'and', 'across', 'asked', 'are', 'astronauts', 'after', 'an']
```

Summary

- In Python:
  - dictionaries are used to store key:value pairs (items)
  - a dictionary object can be created in two ways
  - items can be added to a dictionary
  - Items can be retrieved from the dictionary
  - the pairs in a dictionary can be traversed using for … in

Python features used in this lecture

```
english_italian = {"yes":"si", "bye":"ciao", "no":"no", "maybe":"forse",
                                                    "thank you":"grazie"}
english_italian["never"] = "mai"
print(english_italian["bye"] )

for word in english_italian:
    print(english_italian[word])

print(len(english_italian))
```