Zip function

- Syntax: `zip(list1, list2, …)`
- Zipping two or more lists into one list of tuples:
  - `list_1 = [1, 2, 3]`
  - `list_2 = [‘a’, ‘b’, ‘c’]`
  - `zipped_list = list(zip(list_1, list_2))`
    ☞ returns `[(1, ‘a’), (2, ‘b’), (3, ‘c’)]`
- Unzipping into separate lists:
  - `list_1, list_2 = zip(*zipped_list)`
  - `list_1 = [1, 2, 3]`
  - `list_2 = [‘a’, ‘b’, ‘c’]`

List Comprehension

- Unique way of quickly creating a list with python

Using a Loop

- List of square numbers

```
squares = []
for x in range(1, 11):
   square = x**2
   squares.append(square)
```

- Convert a list of names to uppercase:

```
names = [‘farbod', ‘maryam', ‘reza',
‘pooneh']
upper_names = []
for name in names:
    upper_names.append(name.upper())
```

Using Comprehension

- List of square numbers

```
squares = [x**2 for x in range(1, 11)]
```

- Convert a list of names to uppercase:

```
names = [‘farbod', ‘maryam', ‘reza',
‘pooneh']
upper_names = [name.upper() for name in names]
```

Functions

- A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result. Its main purpose is to prevent repetition in your code.
- Syntax:

  -
    ```
    def name_of_function(parameters):
        '''
        docstring: docstring is short for documentation
        string. It is          used to explain in brief, what
        a function does.
        '''
        # your code here
        # …
        return result
    ```

Functions

- Function with no argument:

  -
    ```
    >> def say_hello():
    ...     return ‘hello’
    >> a = say_hello()
    >> a
    >> ‘hello’
    ```
- Function with an argument:

  -
    ```
    >> def say_hello(name):
    ...     return ‘hello ’ + name
    >> a = say_hello() ☞ returns error
    >> a = say_hello(‘farbod’)
    >> a
    >> ‘hello farbod’
    ```

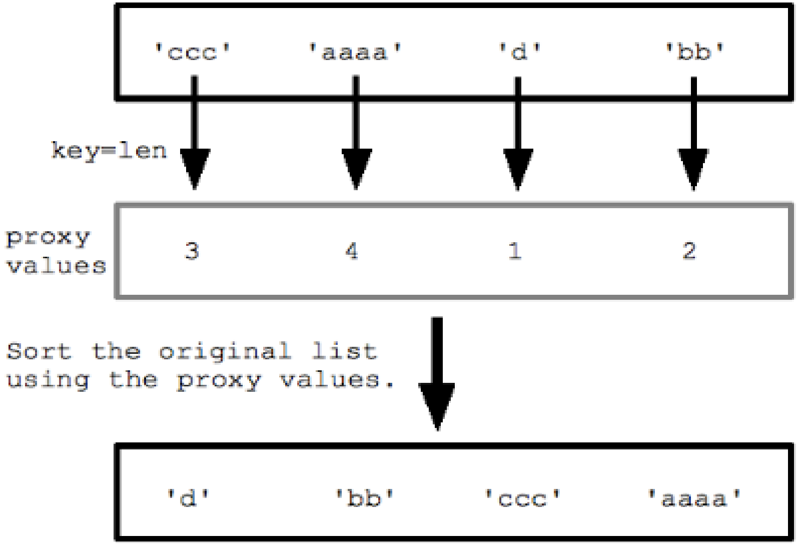
Functions

- Function with an argument with a default value:

  -
    ```
    >> def say_hello(name='farbod'):
    ...     return 'hello ' + name
    >> a = say_hello()
    >> b = say_hello('parvin')
    >> a
    >> 'hello farbod'
    >> b
    >> 'hello parvin'
    ```

List sorting

- syntax: sorted(iterable, key, reverse)
  - `a = [3, 2, 1, 4]`
  - `sorted(a)` ☞ returns `[1, 2, 3, 4]`
  - `sorted(a, reverse=True)` ☞ returns `[4, 3, 2, 1]`
- Custom sorting using *key* parameter
  - `a = ['bbb', 'cc', 'd', 'aaaa`
  - `sorted(a, key=len)` ☞ returns `['d','cc','bbb','aaaa']`
  - `sorted(a, key=len, reverse=True)` ☞ returns `['aaaa','bbb','cc','d']`



List sorting

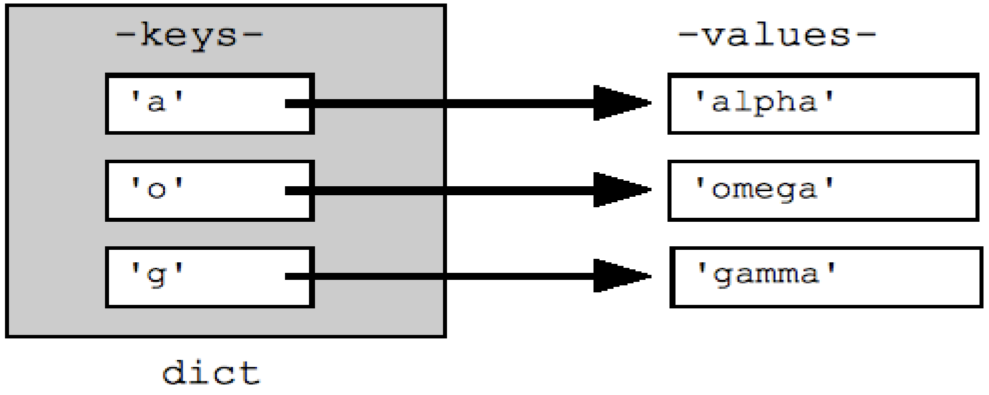
- Sorting based on your own function. for example, we want to sort a list based on the last character of the strings.
  - `a = [‘xc’, ‘zb’, ‘yd’, ‘wa’]`

  -
    ```
    def my_func(s):
        return s[-1]
    ```
  - `sorted(a, key=my_func)` ☞ returns `[‘wa’, ‘zb’, ‘xc’, ‘yd’]`
- sorting a list by some value, then by another value:
  - `a =  [‘abd’, ‘adc’, ‘ca’, ‘ba’]`

  -
    ```
    def my_func(s):
        return (s[0], s[-1])
    ```
  - `sorted(a, key=my_func)` ☞ returns `[‘adc’, ‘abd’, ‘ba’, ‘ca’]`

Dictionaries

- A **dictionary** maps a set of objects (keys) to another set of objects (values). A **Python dictionary** is a mapping of unique keys to values. **Dictionaries** are mutable, which means they can be changed. The values that the keys point to can be any **Python** value.



Dictionaries

- Creation: `new_dict={} OR new_dict=dict() OR new_dict={‘a’:1, ‘foo’:20}`
- Accessing: `new_dict[‘a’]` ☞ returns `1`
- Deleting a key-value pair: `del new_dict[‘a’]` ☞ `new_dict={‘foo’:20}`
- Adding a key-value pair: `new_dict[‘farbod’]=‘parvin’` ☞ `new_dict={‘foo’:20, ‘farbod’:’parvin’}`
- Getting a list of keys: `key_list = new_dict.keys()`
- Getting a list of values: `value_list = new_dict.values()`
- Getting a list of key-value pairs (items): `item_list = new_dict.items()`

Dictionaries

```
fav_languages =
{‘reza’:’c#’,‘maryam’:’c++’,‘farbod’:’python’,’elham’:’ruby’}
```

- Looping through the keys:

  -
    ```
    for name in fav_languages.keys():
        print(name)
    ```
- Looping through the values:

  -
    ```
    for language in fav_languages.values():
        print(language)
    ```
- Looping through the items:

  -
    ```
    for name, language in fav_languages.items():
        print(name + ‘: ’ + language)
    ```

Nesting

List of dictionaries

```
▪ car_1 = {‘brand’:’bmw’
          ‘color’:’red’
          ‘year’:2015
           }
▪ Car_2 = {‘brand’:’toyota’
          ‘color’:’blue’
          ‘year’: 2017
           }
▪ cars = [car_1, car_2]
```

Lists in a dictionary

```
▪ car_1 = {‘brand’:’bmw’
          ‘color’:’red’
          ‘year’:2015
          ‘features’=[‘cruise control’,
          ‘sunroof’],
           }
```

Working with files (IO basics)

- In python, there are five options available for working with files:
  - `my_file = open('test.txt', mode='r')`

read	write	append	read and write	write and read
`mode='r'`	`mode='w'`	`mode='a'`	`mode='r+'`	`mode='w+'`

- Two ways of opening a file:

File should be closed:

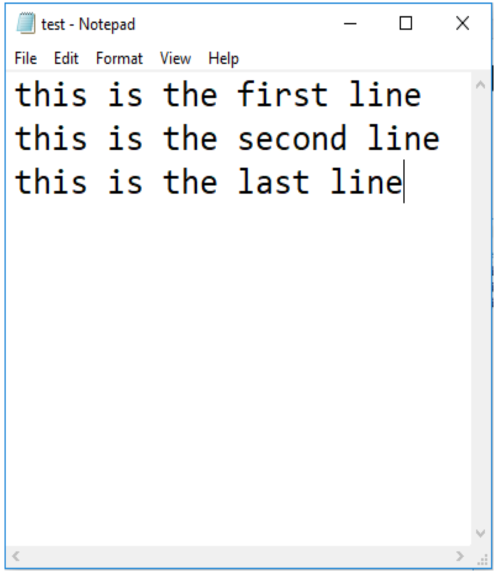
```
myfile = open('test.txt', mode='r')
text = myfile.read()
myfile.close()
```

File will be closed when you exit the inner block:

```
with open('test.txt', mode='r') as myfile:
    text = f.read()
print(text) # at this line, file is closed
```

Working with files (read)

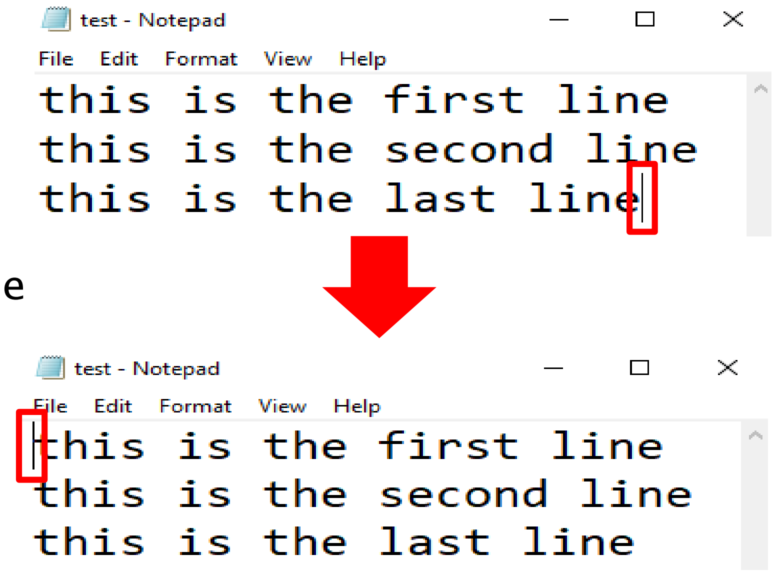
- Opening a file
  - `my_file = open(‘test.txt’, mode=‘r’)`
    `#if the file is in your current directory`
- Storing the content as a single <u>string</u>:
  - `my_file.read()` ☞ returns
    `'this is the first line\nthis is the second line\nthis is the last line‘`
- Storing the content as a <u>list of lines</u>:
  - `my_file.readlines()` ☞ returns
    `['this is the first line\n', 'this is the second line\n', 'this is the last line']`
- Closing the file:
  - `my_file.close()`



```
this is the first line
this is the second line
this is the last line
```

Working with files (read)

- NOTE: you should reset the cursor every time you read a file, so that it starts from the beginning!!!
- Example:
  - `my_file = open('test.txt')`
  - `my_file.read()` ☞ returns `'this is the first line\nthis is the second line\nthis is the last line'`
  - `my_file.read()` ☞ returns `''`
- Resetting the cursor:
  - `my_file.seek(0)`
  - `my_file.read()` ☞ returns `'this is the first line\nthis is the second line\nthis is the last line'`



```
this is the first line
this is the second line
this is the last line
```

```
this is the first line
this is the second line
this is the last line
```

Working with files (write)

- Creating a new file:
  - `new_file = open(‘new.txt’, mode=‘w’)`
- Writing some text in the file:
  - `new_file.write(‘this is a text file’)`
- Text will be shown when you close the file:
  - `new_file.close()`
- NOTE: if you open the file again, all of the previous text will be deleted.