18ECE201J-Python & Scientific Python –Unit III

- Reading Data from File- Line by Line, Reading a Mixture of Text and Numbers
- Making Dictionaries -Dictionary Operations – Polynomials.
- File Data in Nested Dictionaries – Strings.
- Common Operations on Strings - Reading Coordinates.
- Reading Data from Web Pages- About Web Pages - Access Web Pages in Programs- Reading Pure Text Files.
- Extracting Data from an HTML Page - Writing a Table to File, Reading and Writing Spreadsheet Files.
- Representing a Function as a Class and manipulation
- Bank Accounts as class, A Class for Solving ODEs.

Courtesy and Reference: A Primer on Scientific Programming with Python by Hans Petter Langtangen

Example: reading a file with numbers

- The file data1.txt has a column of numbers:

  ```
  21.8
  18.1
  19
  23
  26
  17.8
  ```

- Goal: compute the average value of the numbers:

Example: reading a file with numbers

- We must convert strings to numbers before computing:

```
infile = open(’data1.txt’, ’r’)
lines = infile.readlines()
infile.close()
mean = 0
for line in lines:
number = float(line)
mean = mean + number
mean = mean/len(lines)
print mean
```

- A quicker and shorter variant:

```
infile = open(’data1.txt’, ’r’)
numbers = [float(line) for line in infile.readlines()]
infile.close()
mean = sum(numbers)/len(numbers)
print mean
```

Reading file Using While loop

```
infile = open(’data1.txt’, ’r’)
mean = 0
n = 0
    while True:                              # loop "forever"
           line = infile.readline()
    if not line:                             # line=’’ at end of file
           break                             # jump out of loop
    mean += float(line)
    n += 1
infile.close()
mean = mean/float(n)
print mean
```

Reading a mixture of text and numbers

- The file rainfall.dat looks like this:
- **Average rainfall (in mm) in Rome**: 1188 months between 1782 and 1970

```
Jan   81.2
Feb  63.2
Mar  70.3
Apr   55.7
May 53.0
```

- ...
- **Goal: read the numbers and compute the mean**
- Technique: for each line, split the line into words, convert the 2nd word to a number and add to sum

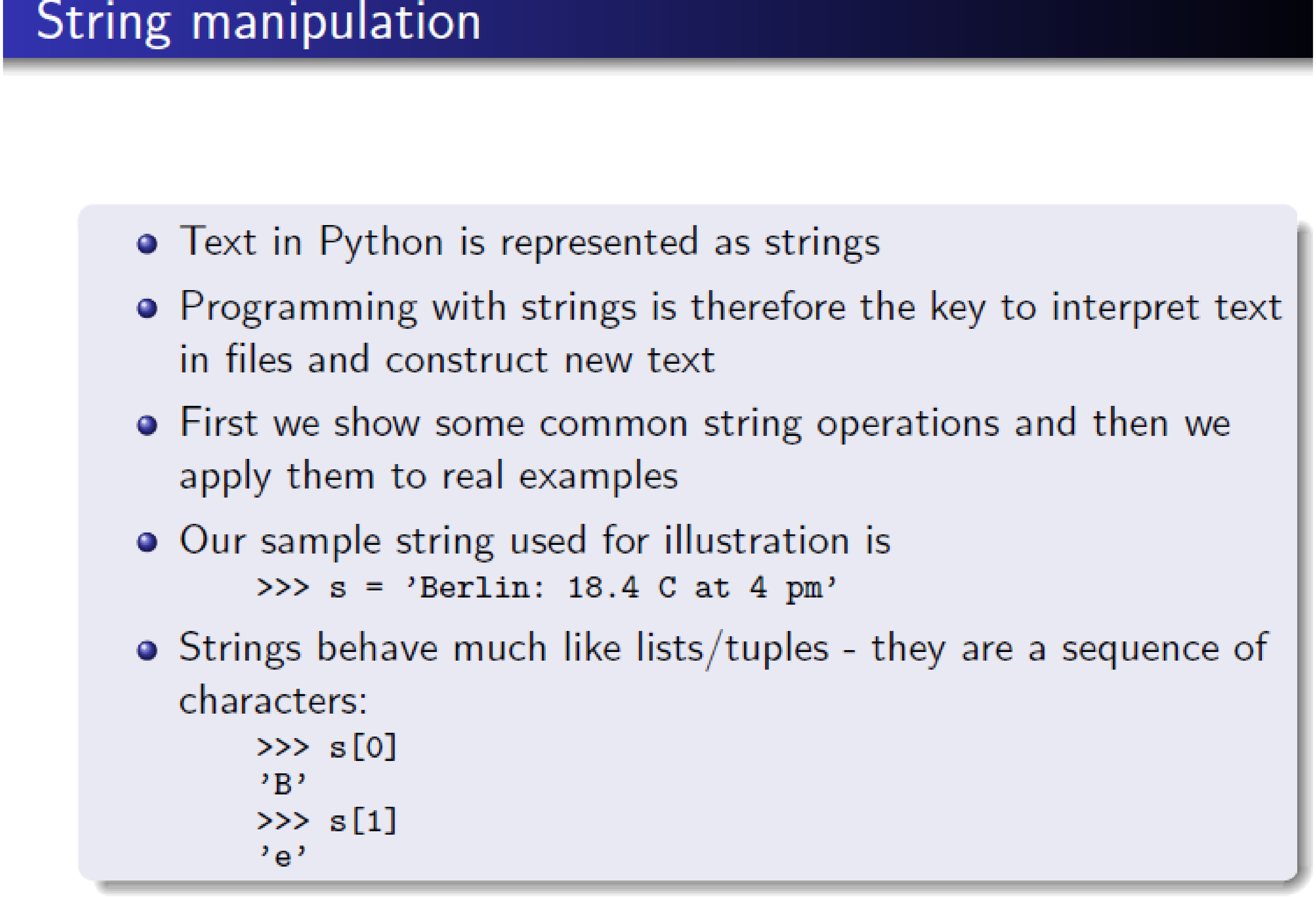
```
for line in infile:
        words = line.split()          # list of words on the line
        number = float(words[1])
```

- Note line.split(): very useful for grabbing individual words
- on a line, can split wrt any string, e.g., line.split(’;’), line.split(’:’)

Reading a mixture of text and numbers

-
  ```
  def extract_data(filename):
      infile = open(filename, ’r’)
      infile.readline() # skip the first line
      numbers = []
      for line in infile:
          words = line.split()
          number = float(words[1])
          numbers.append(number)
      infile.close()
      return numbers
  ```

- `values = extract_data(’rainfall.dat’)`
- `from scitools.std import plot`
- `month_indices = range(1, 13)`
- `plot(month_indices, values[:-1], ’o2’)`



String manipulation

- Text in Python is represented as strings
- Programming with strings is therefore the key to interpret text in files and construct new text
- First we show some common string operations and then we apply them to real examples
- Our sample string used for illustration is

  ```
  >>> s = ’Berlin: 18.4 C at 4 pm’
  ```

- Strings behave much like lists/tuples - they are a sequence of characters:

  ```
  >>> s[0]
  ’B’
  >>> s[1]
  ’e’
  ```

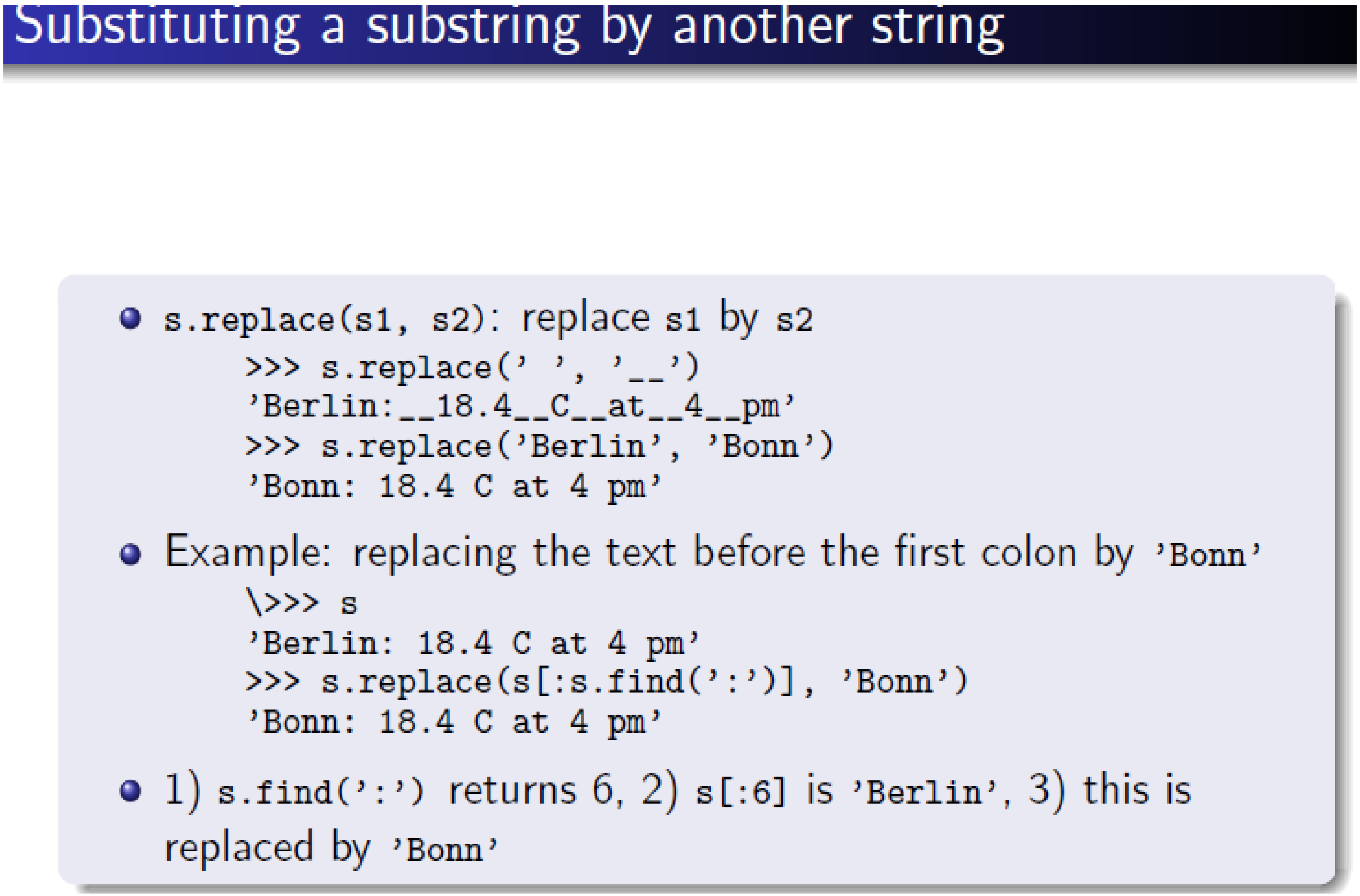
Extracting substrings

- Substrings are just as slices of lists and arrays:

```
>>> s
'Berlin: 18.4 C at 4 pm'
>>> s[8:]      # from index 8 to the end of the string
'18.4 C at 4 pm'
>>> s[8:12]    # index 8, 9, 10 and 11 (not 12!)
'18.4'
>>> s[8:-1]
'18.4 C at 4 p'
>>> s[8:-8]
'18.4 C'
```

- Find start of substring:

```
>>> s.find('Berlin')  # where does 'Berlin' start?
0                     # at index 0
>>> s.find('pm')
20
>>> s.find('Oslo')    # not found
-1
```



Substituting a substring by another string

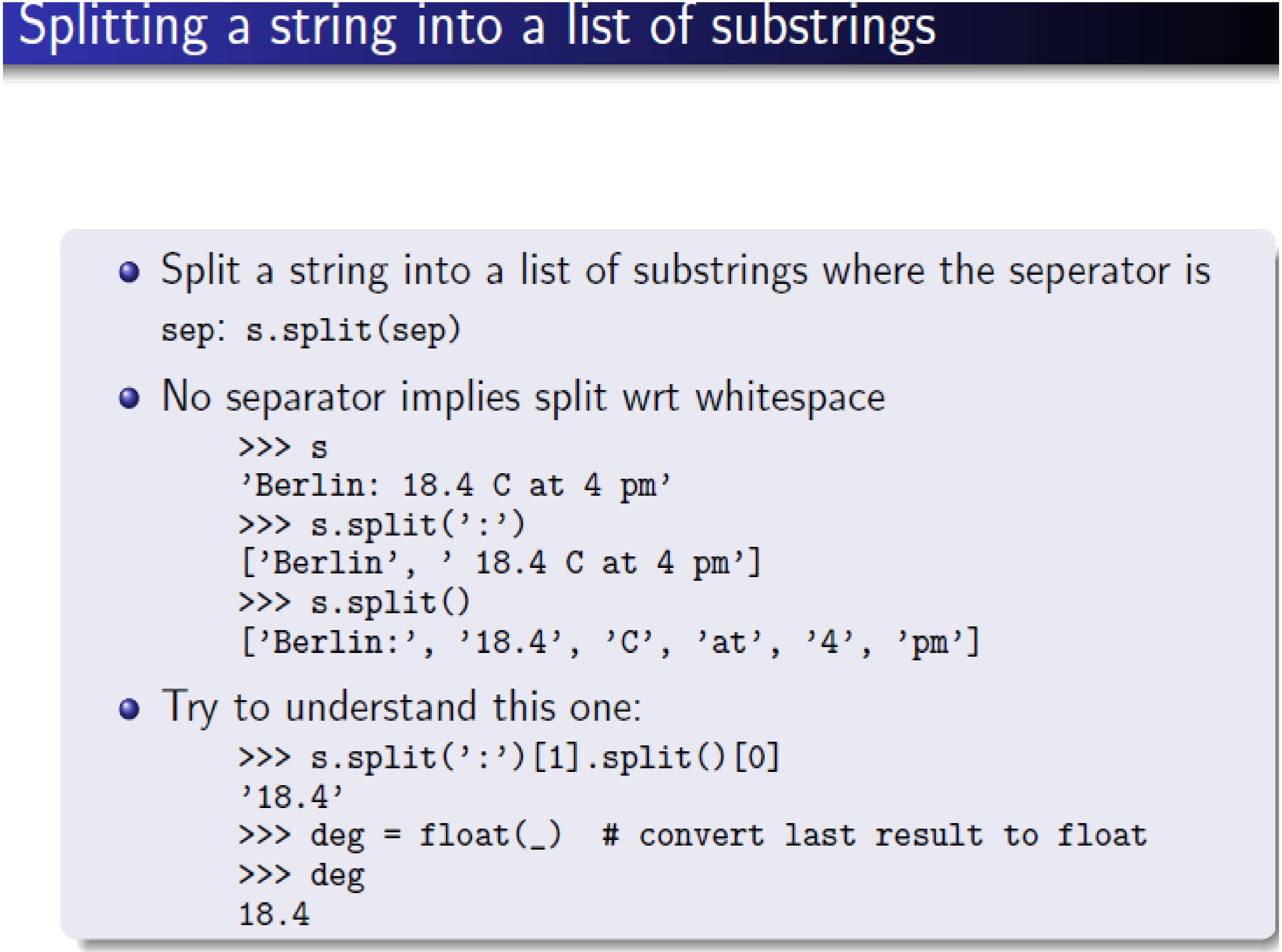
- `s.replace(s1, s2)`: replace `s1` by `s2`

```
>>> s.replace(' ', '__')
'Berlin:__18.4__C__at__4__pm'
>>> s.replace('Berlin', 'Bonn')
'Bonn: 18.4 C at 4 pm'
```

- Example: replacing the text before the first colon by `'Bonn'`

```
\>>> s
'Berlin: 18.4 C at 4 pm'
>>> s.replace(s[:s.find(':')], 'Bonn')
'Bonn: 18.4 C at 4 pm'
```

- 1) `s.find(':')` returns 6, 2) `s[:6]` is `'Berlin'`, 3) this is replaced by `'Bonn'`



Splitting a string into a list of substrings

- Split a string into a list of substrings where the seperator is sep: `s.split(sep)`
- No separator implies split wrt whitespace

```
>>> s
’Berlin: 18.4 C at 4 pm’
>>> s.split(’:’)
[’Berlin’, ’ 18.4 C at 4 pm’]
>>> s.split()
[’Berlin:’, ’18.4’, ’C’, ’at’, ’4’, ’pm’]
```

- Try to understand this one:

```
>>> s.split(’:’)[1].split()[0]
’18.4’
>>> deg = float(_)  # convert last result to float
>>> deg
18.4
```

Splitting a string into lines

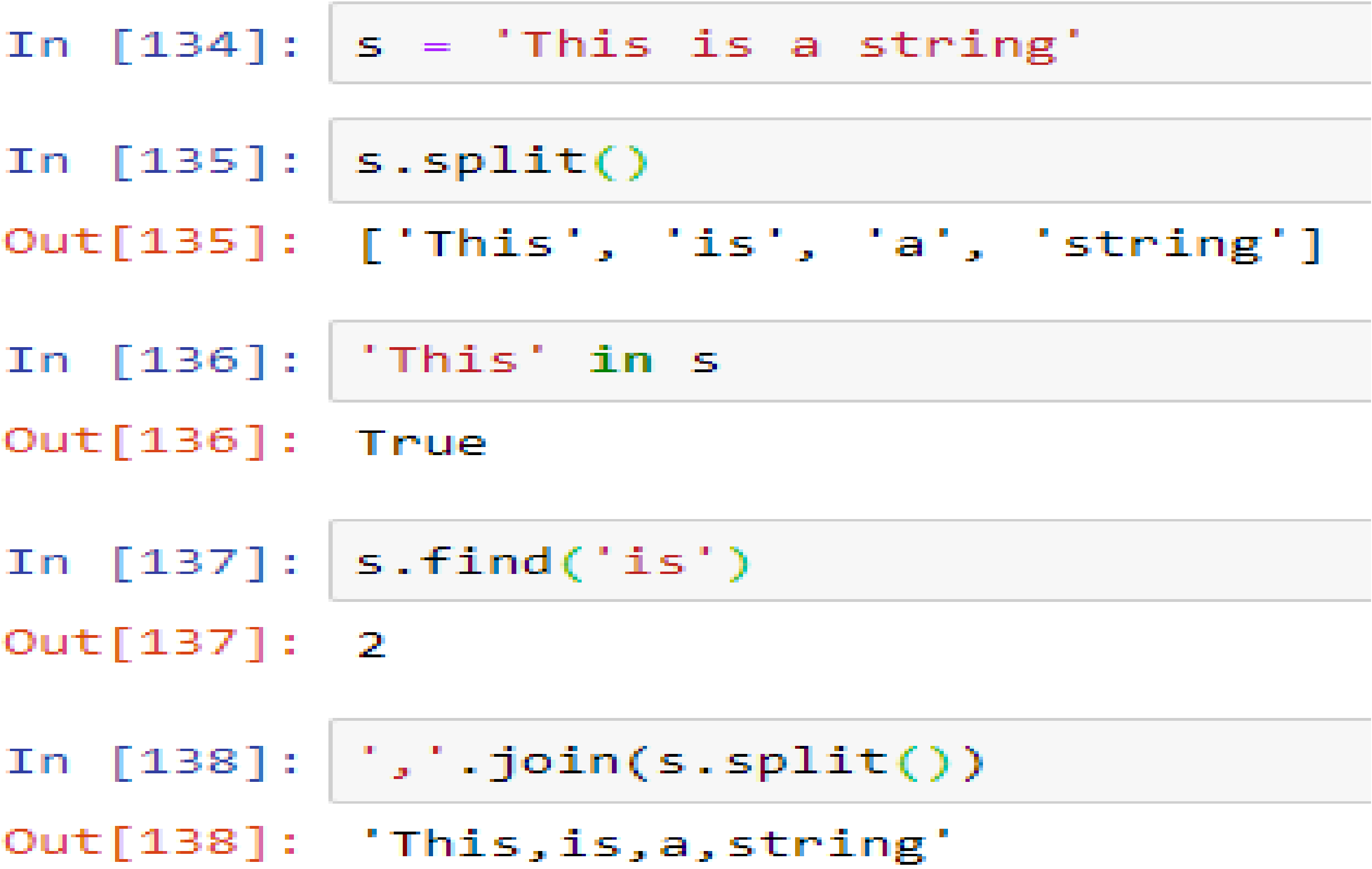
- Very often, a string contains lots of text and we want to split the text into separate lines
- Lines may be separated by different control characters on different platforms. On Unix/Linux/Mac, backslash n is used:

```
>>> t = ’1st line\n2nd line\n3rd line’
>>> print t
1st line
2nd line
3rd line
>>> t.split(’\n’)
[’1st line’, ’2nd line’, ’3rd line’]
>>> t.splitlines()  # cross platform - better!
[’1st line’, ’2nd line’, ’3rd line’]
```

String Manipulation

Text in Python is represented as strings.

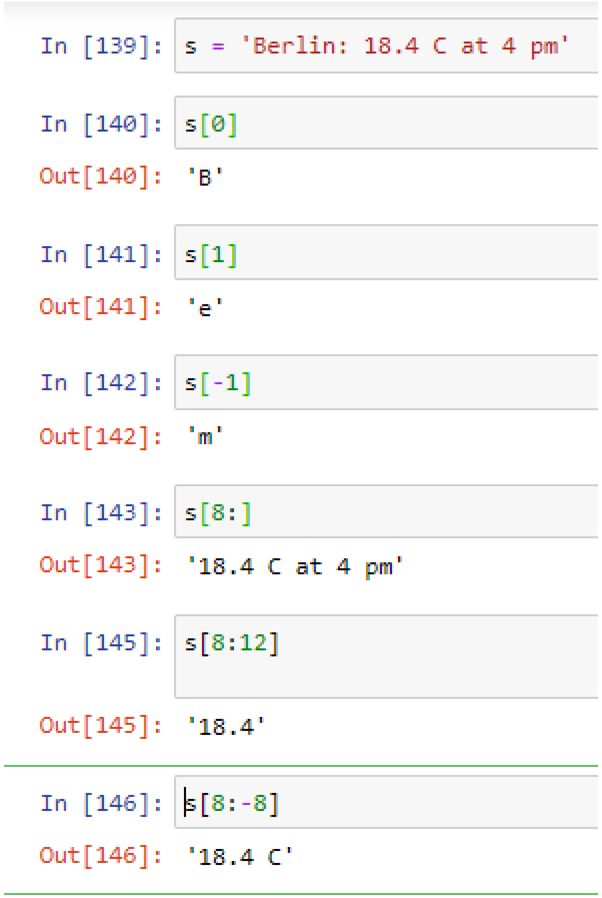
Inspecting and manipulating strings is the way we can understand the contents of files



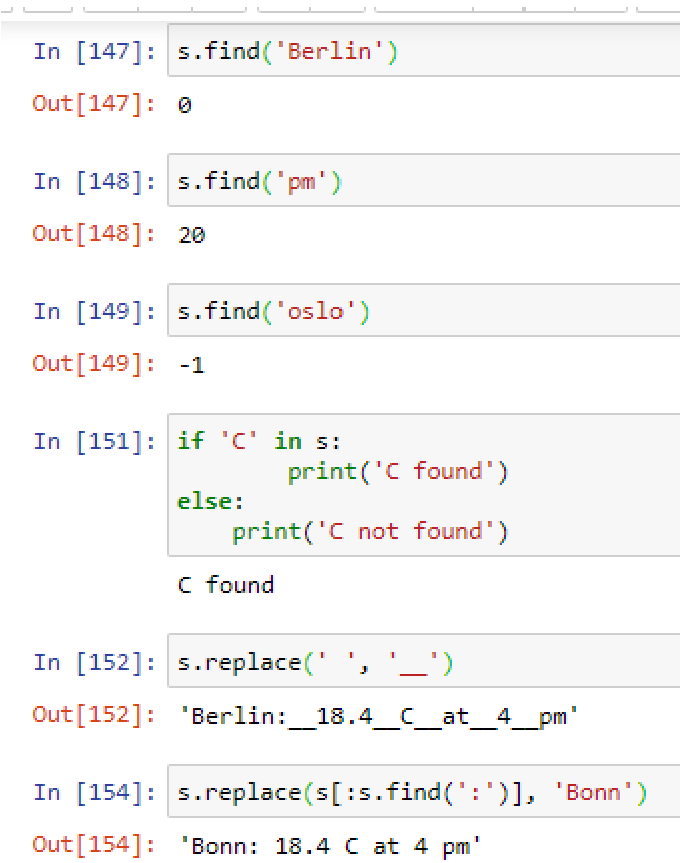
```
In [134]: s = 'This is a string'

In [135]: s.split()
Out[135]: ['This', 'is', 'a', 'string']

In [136]: 'This' in s
Out[136]: True

In [137]: s.find('is')
Out[137]: 2

In [138]: ','.join(s.split())
Out[138]: 'This,is,a,string'
```

String Manipulation

Strings behave much like lists/tuples - they are a sequence of characters



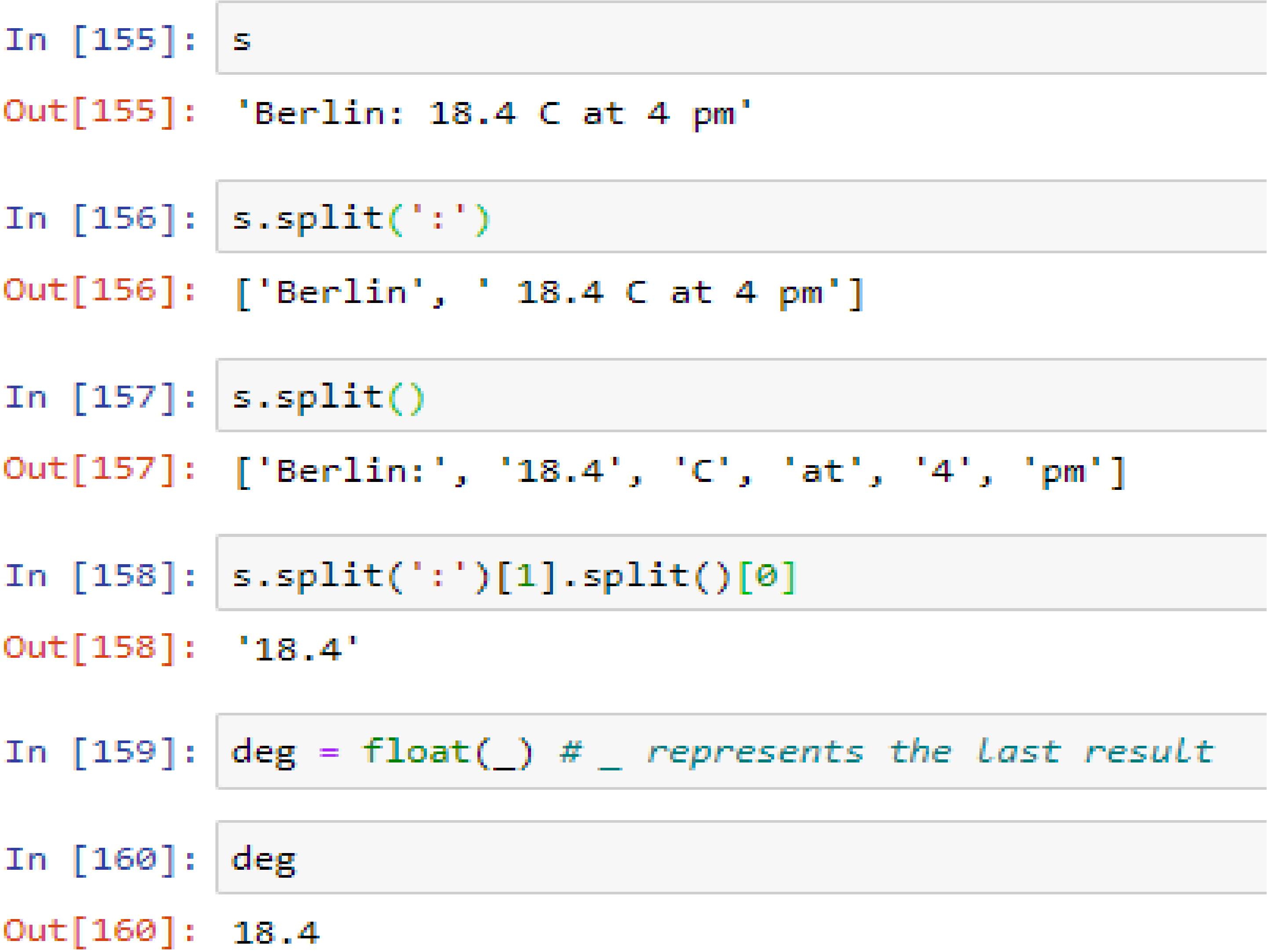
```
In [139]: s = 'Berlin: 18.4 C at 4 pm'

In [140]: s[0]
Out[140]: 'B'

In [141]: s[1]
Out[141]: 'e'

In [142]: s[-1]
Out[142]: 'm'

In [143]: s[8:]
Out[143]: '18.4 C at 4 pm'

In [145]: s[8:12]

Out[145]: '18.4'

In [146]: s[8:-8]
Out[146]: '18.4 C'
```



```
In [147]: s.find('Berlin')
Out[147]: 0

In [148]: s.find('pm')
Out[148]: 20

In [149]: s.find('oslo')
Out[149]: -1

In [151]: if 'C' in s:
                  print('C found')
          else:
              print('C not found')

          C found

In [152]: s.replace(' ', '__')
Out[152]: 'Berlin:__18.4__C__at__4__pm'

In [154]: s.replace(s[:s.find(':')], 'Bonn')
Out[154]: 'Bonn: 18.4 C at 4 pm'
```

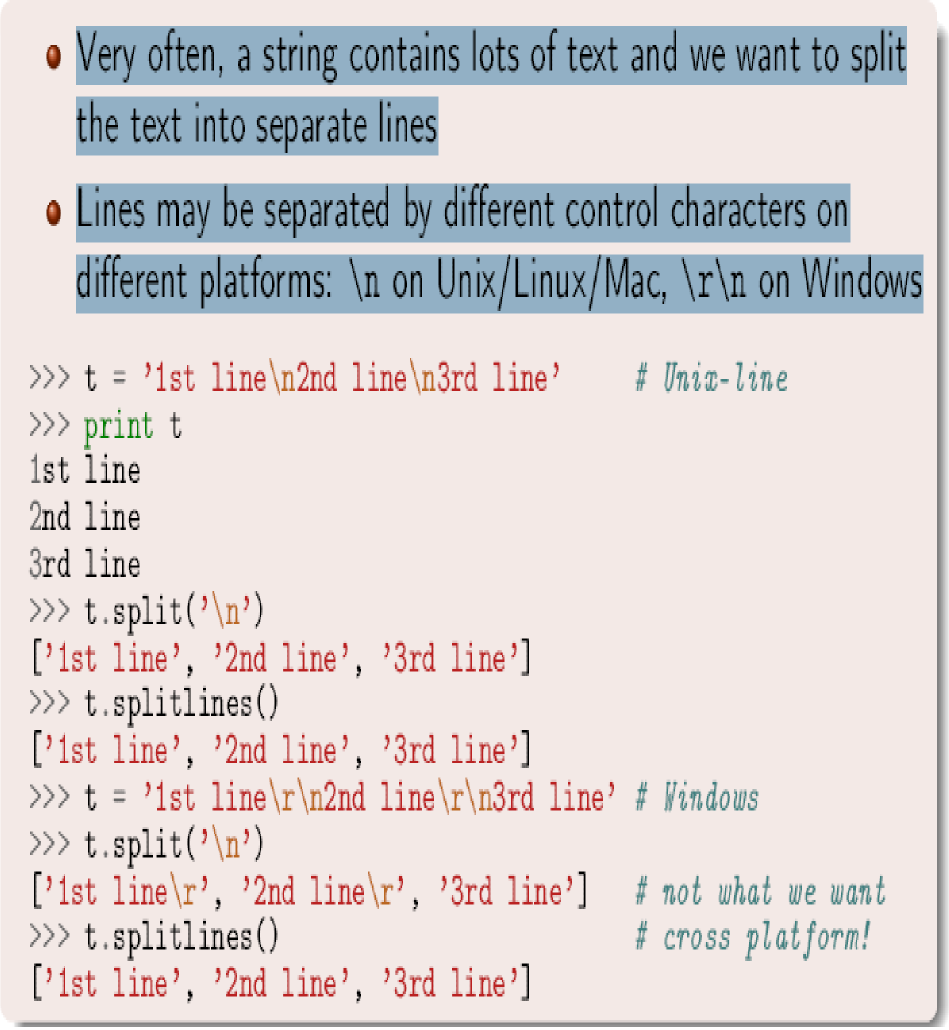
s.replace(s1, s2): replace s1 by s2
(1) s.find(':') returns 6, (2) s[:6] is 'Berlin', (3) Berlin is replaced by 'Bonn'

- s.split(sep): split s into a list of substrings separated by sep (no separator implies split wrt whitespace):



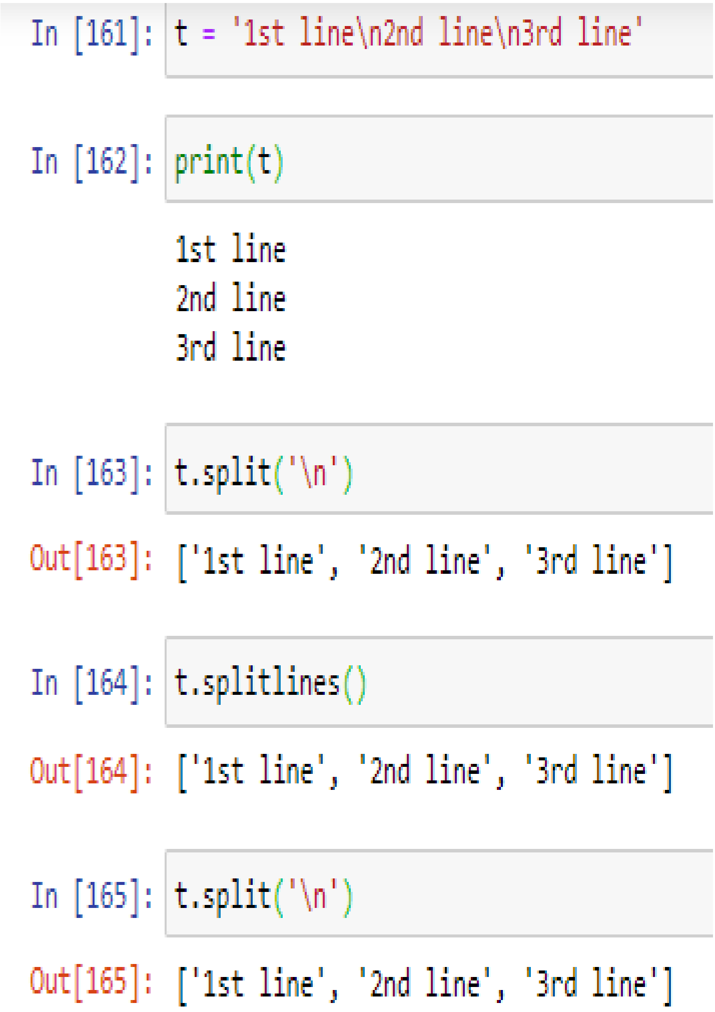
```
In [155]: s

Out[155]: 'Berlin: 18.4 C at 4 pm'

In [156]: s.split(':')

Out[156]: ['Berlin', ' 18.4 C at 4 pm']

In [157]: s.split()

Out[157]: ['Berlin:', '18.4', 'C', 'at', '4', 'pm']

In [158]: s.split(':')[1].split()[0]

Out[158]: '18.4'

In [159]: deg = float(_) # _ represents the last result

In [160]: deg

Out[160]: 18.4
```

Splitting a string into lines



- Very often, a string contains lots of text and we want to split the text into separate lines
- Lines may be separated by different control characters on different platforms: \n on Unix/Linux/Mac, \r\n on Windows

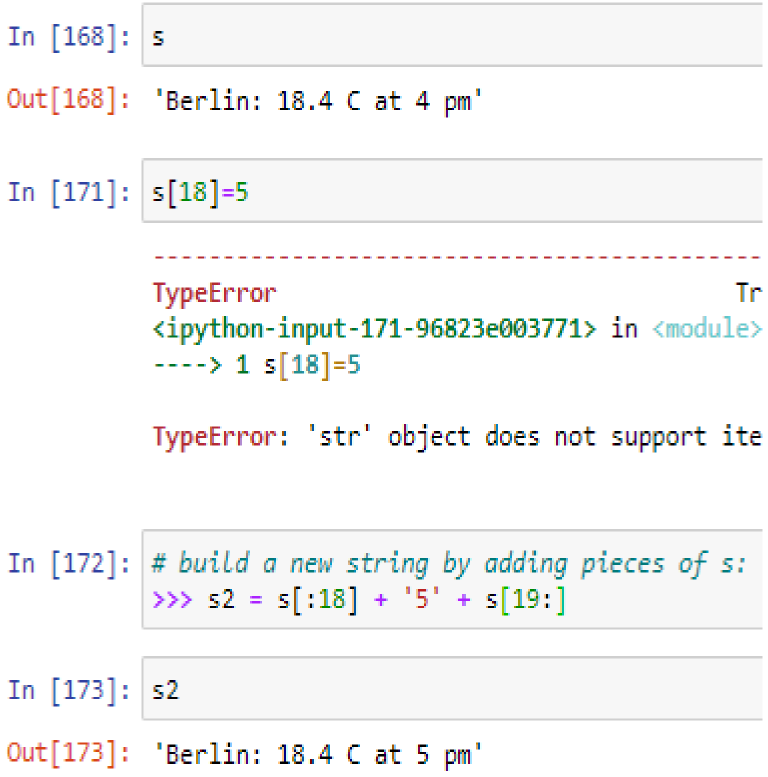
```
>>> t = '1st line\n2nd line\n3rd line'      # Unix-line
>>> print t
1st line
2nd line
3rd line
>>> t.split('\n')
['1st line', '2nd line', '3rd line']
>>> t.splitlines()
['1st line', '2nd line', '3rd line']
>>> t = '1st line\r\n2nd line\r\n3rd line' # Windows
>>> t.split('\n')
['1st line\r', '2nd line\r', '3rd line']    # not what we want
>>> t.splitlines()                           # cross platform!
['1st line', '2nd line', '3rd line']
```



```
In [161]: t = '1st line\n2nd line\n3rd line'

In [162]: print(t)

1st line
2nd line
3rd line

In [163]: t.split('\n')

Out[163]: ['1st line', '2nd line', '3rd line']

In [164]: t.splitlines()

Out[164]: ['1st line', '2nd line', '3rd line']

In [165]: t.split('\n')

Out[165]: ['1st line', '2nd line', '3rd line']
```

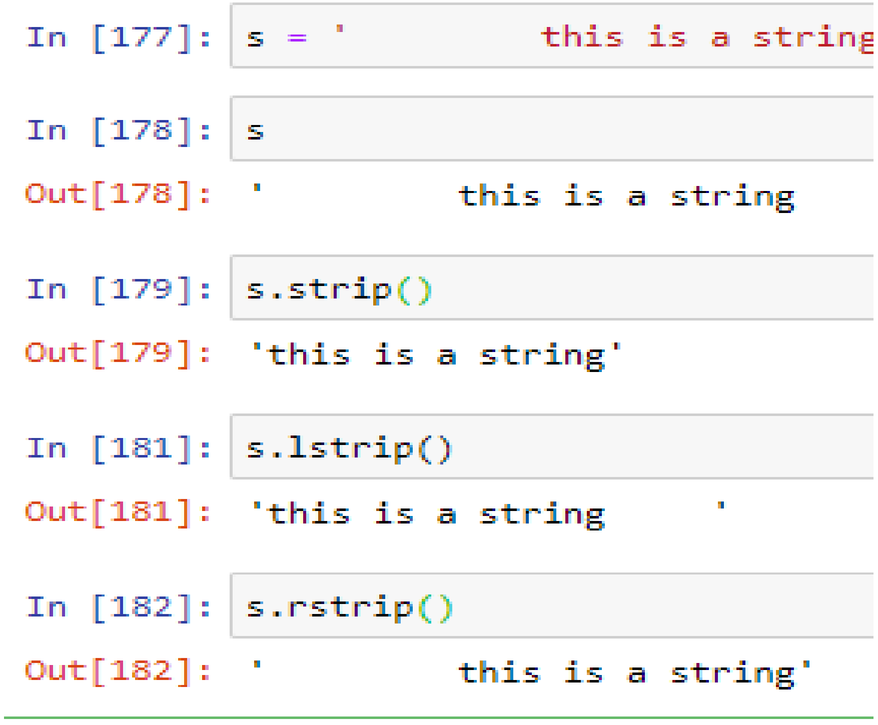
Changing a string / stripping off whitespaces

- all changes of a strings results in a new string



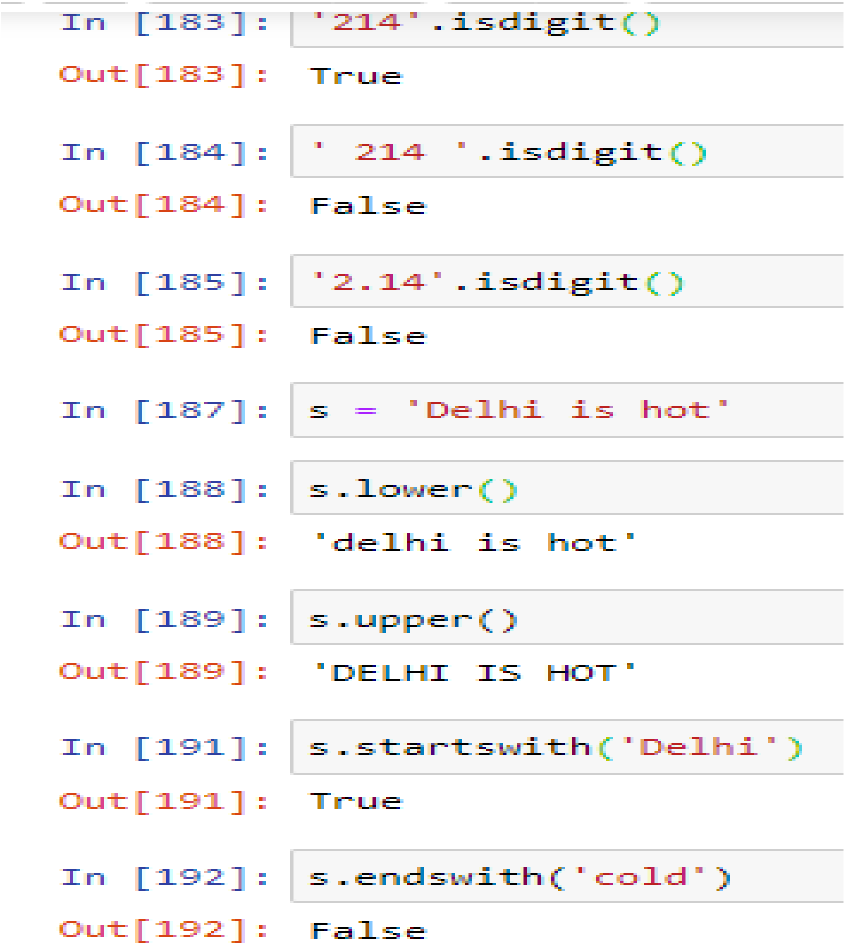
```
In [168]: s
Out[168]: 'Berlin: 18.4 C at 4 pm'

In [171]: s[18]=5

          ---------------------------------------------
          TypeError                                  Tr
          <ipython-input-171-96823e003771> in <module>
          ----> 1 s[18]=5

          TypeError: 'str' object does not support ite


In [172]: # build a new string by adding pieces of s:
          >>> s2 = s[:18] + '5' + s[19:]

In [173]: s2
Out[173]: 'Berlin: 18.4 C at 5 pm'
```

Stripping off leading/trailing whitespace

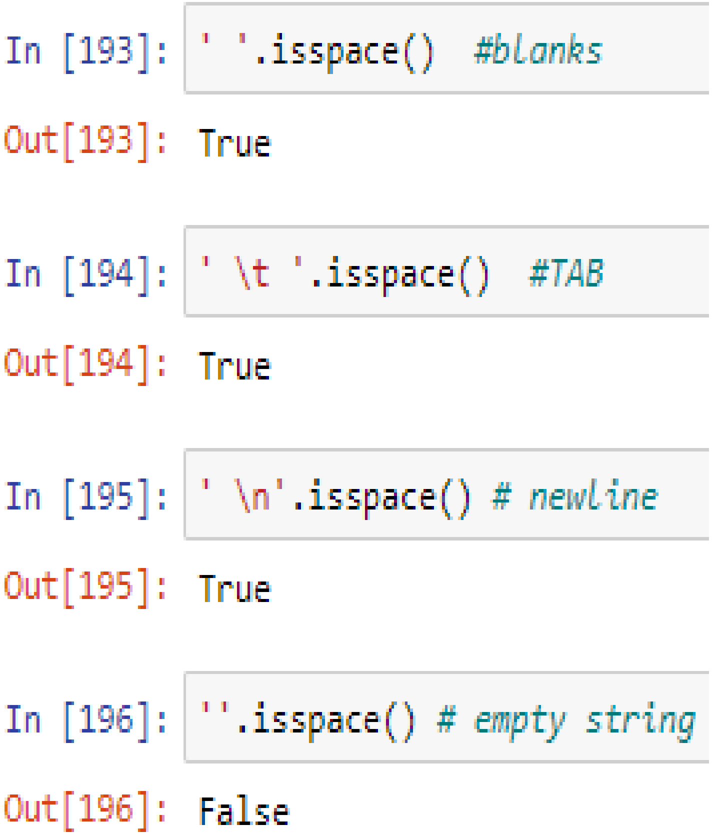


```
In [177]: s = '          this is a string

In [178]: s
Out[178]: '          this is a string

In [179]: s.strip()
Out[179]: 'this is a string'

In [181]: s.lstrip()
Out[181]: 'this is a string      '

In [182]: s.rstrip()
Out[182]: '          this is a string'
```

String functions

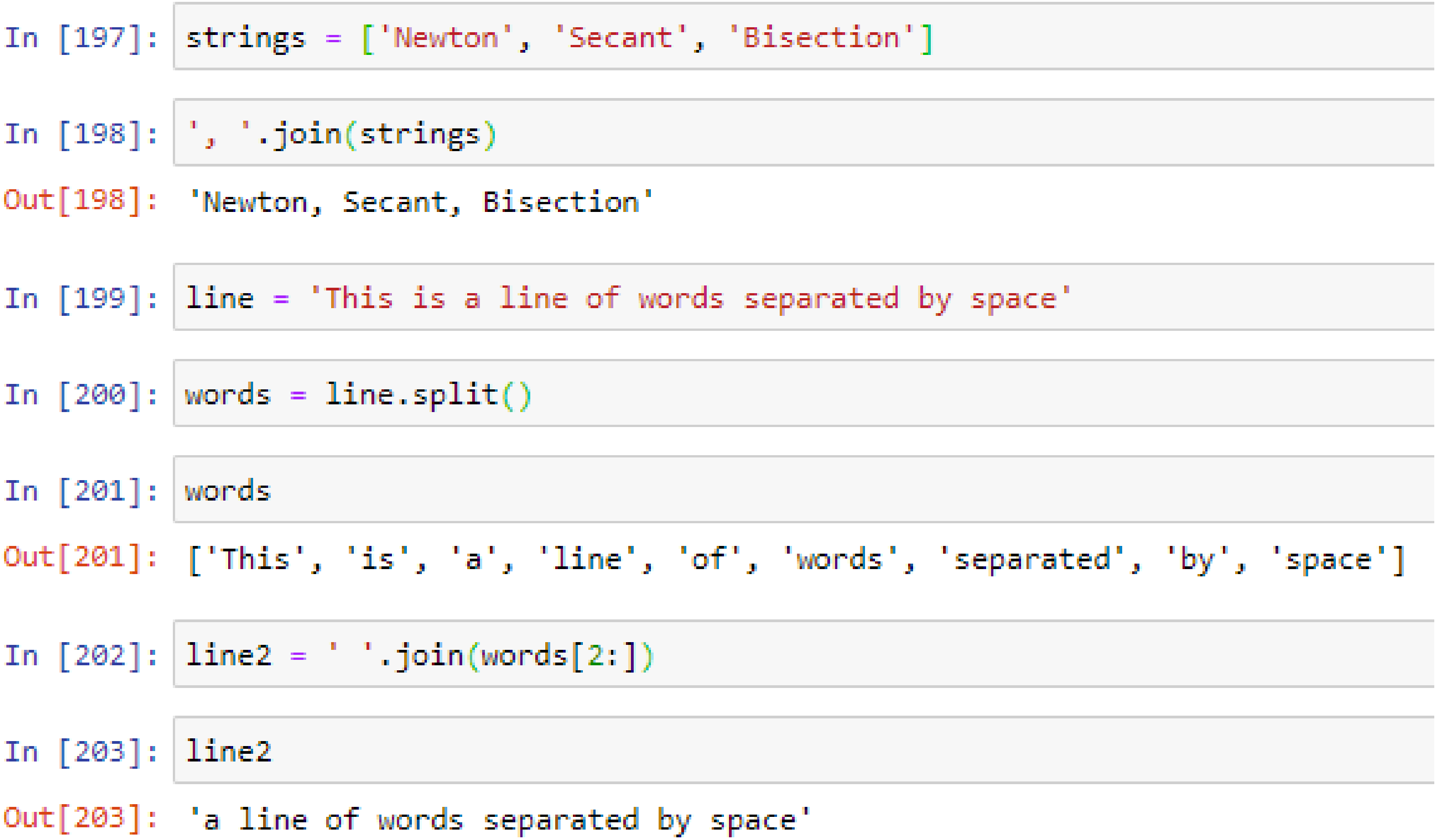


```
In [183]: '214'.isdigit()
Out[183]: True

In [184]: ' 214 '.isdigit()
Out[184]: False

In [185]: '2.14'.isdigit()
Out[185]: False

In [187]: s = 'Delhi is hot'

In [188]: s.lower()
Out[188]: 'delhi is hot'

In [189]: s.upper()
Out[189]: 'DELHI IS HOT'

In [191]: s.startswith('Delhi')
Out[191]: True

In [192]: s.endswith('cold')
Out[192]: False
```



```
In [193]: ' '.isspace()  #blanks

Out[193]: True


In [194]: ' \t '.isspace()  #TAB

Out[194]: True


In [195]: ' \n'.isspace() # newline

Out[195]: True


In [196]: ''.isspace() # empty string

Out[196]: False
```

Joining a list of substrings to a new string



```
In [197]: strings = ['Newton', 'Secant', 'Bisection']

In [198]: ', '.join(strings)

Out[198]: 'Newton, Secant, Bisection'

In [199]: line = 'This is a line of words separated by space'

In [200]: words = line.split()

In [201]: words

Out[201]: ['This', 'is', 'a', 'line', 'of', 'words', 'separated', 'by', 'space']

In [202]: line2 = ' '.join(words[2:])

In [203]: line2

Out[203]: 'a line of words separated by space'
```

Python Dictionary

Dictionary in Python is an ordered collection of data values, used to store data values like a map.
Dictionary holds **key:value** pair. Key-value is provided in the dictionary to make it more optimized.

Creating a Dictionary
a Dictionary can be created by placing a sequence of elements within curly {} braces, separated by ‘comma’.

Dictionary holds a pair of values, one being the Key and the other corresponding pair element being its Key:value.

Values in a dictionary can be of any datatype and can be duplicated, whereas keys can’t be repeated and must be immutable.

Note – Dictionary keys are case sensitive

Example on a dictionary

- Suppose we need to store the temperatures in Oslo, London and Paris
- List solution:

```
temps = [13, 15.4, 17.5]
# temps[0]: Oslo
# temps[1]: London
# temps[2]: Paris
```

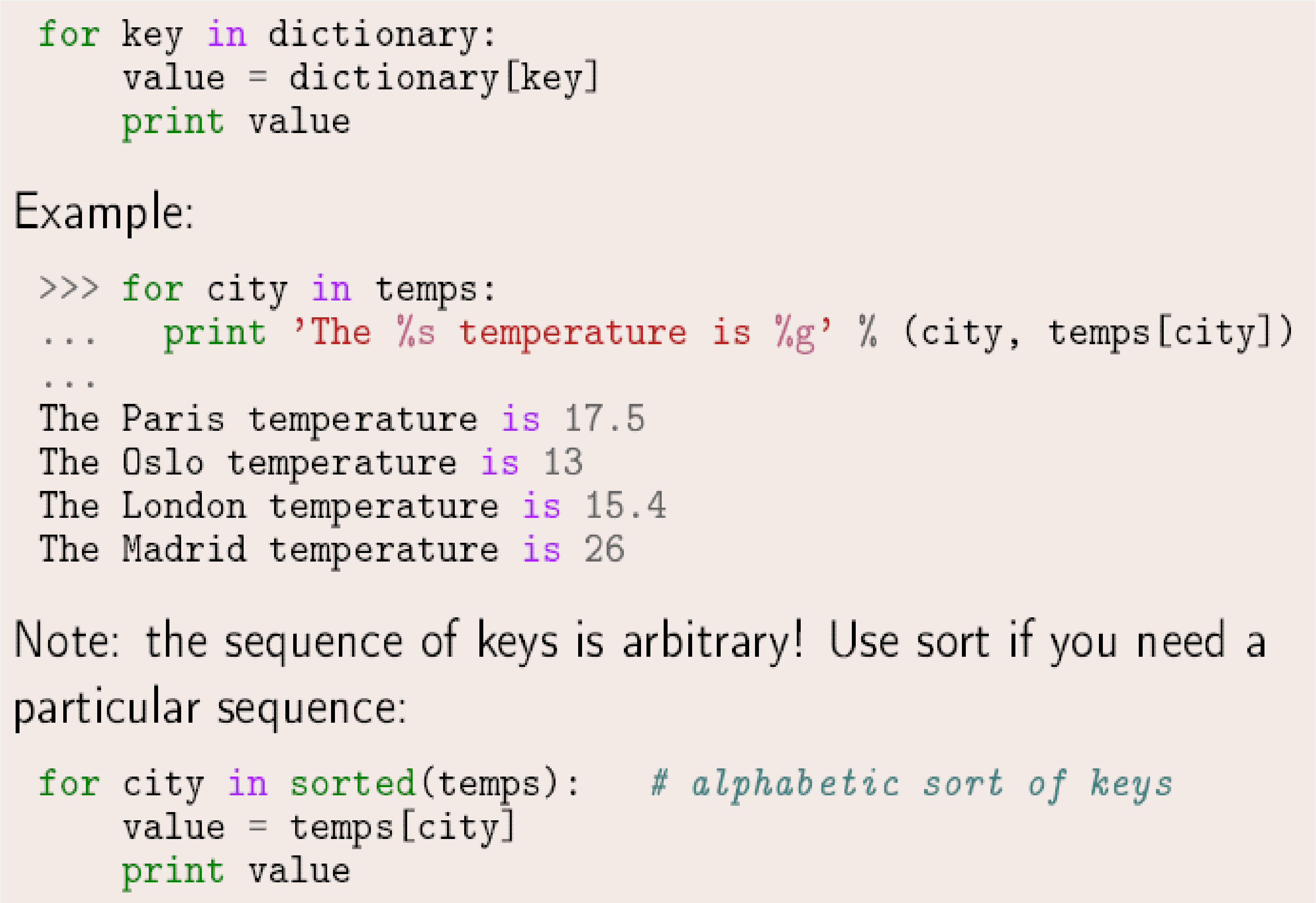
- We need to remember the mapping between the index and the
- city name – with a dictionary we can index the list with the
- city name directly (e.g., temps["Oslo"]):

```
temps = {’Oslo’: 13, ’London’: 15.4, ’Paris’: 17.5}
# or
temps = dict(Oslo=13, London=15.4, Paris=17.5)
# application:
print (“The temperature in London is”, temps[’London’])
>>> temps.keys()
[’Paris’, ’Oslo’, ’London’, ’Madrid’]
>>> temps.values()
[17.5, 13, 15.4, 26.0]
```

Add a new element to the dictionary

```
temps[“madird”]=26.0
```

Print dictionary



```
for key in dictionary:
    value = dictionary[key]
    print value
```

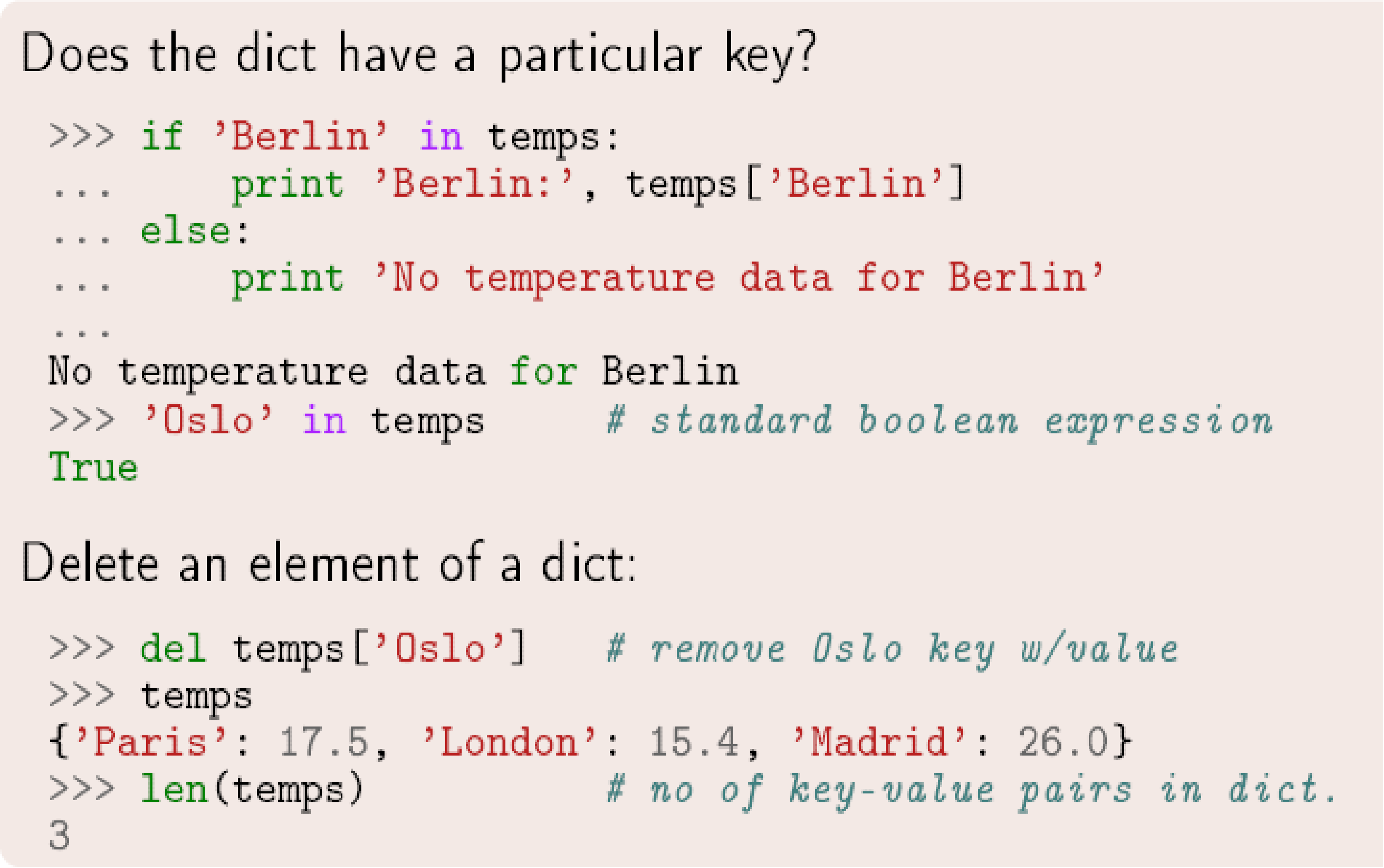
Example:

```
>>> for city in temps:
...     print ’The %s temperature is %g’ % (city, temps[city])
...
The Paris temperature is 17.5
The Oslo temperature is 13
The London temperature is 15.4
The Madrid temperature is 26
```

Note: the sequence of keys is arbitrary! Use sort if you need a particular sequence:

```
for city in sorted(temps):    # alphabetic sort of keys
    value = temps[city]
    print value
```

Delete and Check the particular Key



Does the dict have a particular key?

```
>>> if 'Berlin' in temps:
...     print 'Berlin:', temps['Berlin']
... else:
...     print 'No temperature data for Berlin'
...
No temperature data for Berlin
>>> 'Oslo' in temps         # standard boolean expression
True
```

Delete an element of a dict:

```
>>> del temps['Oslo']       # remove Oslo key w/value
>>> temps
{'Paris': 17.5, 'London': 15.4, 'Madrid': 26.0}
>>> len(temps)              # no of key-value pairs in dict.
3
```

OrderedDict in Python

An **OrderedDict** is a dictionary subclass that remembers the order that keys were first inserted.

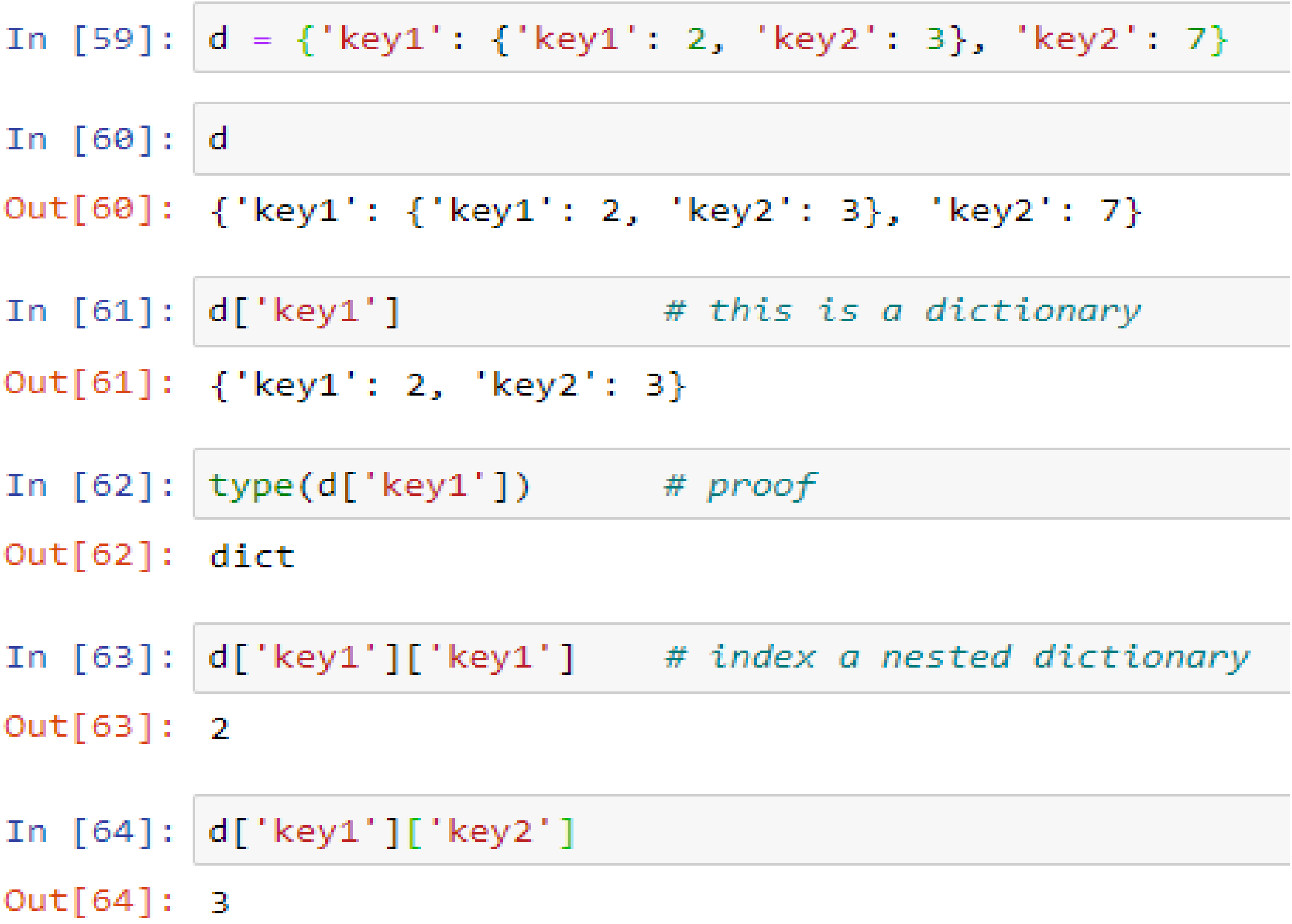
The only difference between dict() and OrderedDict() is that:

OrderedDict **preserves the order** in which the keys are inserted. A regular dict doesn’t track the insertion order, and iterating it gives the values in an arbitrary order. By contrast, the order the items are inserted is remembered by OrderedDict.

```python
from collections import OrderedDict


print("This is a Dict:\n")
d = {}
d['a'] = 1
d['b'] = 2
d['c'] = 3
d['d'] = 4


for key, value in d.items():
    print(key, value)


print("\nThis is an Ordered Dict:\n")
od = OrderedDict()
od['a'] = 1
od['b'] = 2
od['c'] = 3
od['d'] = 4


for key, value in od.items():
    print(key, value)
```

Nested Dictionaries

- dictionaries of dictionaries.



```
In [59]: d = {'key1': {'key1': 2, 'key2': 3}, 'key2': 7}

In [60]: d

Out[60]: {'key1': {'key1': 2, 'key2': 3}, 'key2': 7}

In [61]: d['key1']                # this is a dictionary

Out[61]: {'key1': 2, 'key2': 3}

In [62]: type(d['key1'])          # proof

Out[62]: dict

In [63]: d['key1']['key1']        # index a nested dictionary

Out[63]: 2

In [64]: d['key1']['key2']

Out[64]: 3
```

Summary:

- Dictionaries in a programming language is a type of data-structure used to store information connected in some way.

- Python Dictionary are defined into two elements Keys and Values.
- Dictionaries do not store their information in any particular order, so you may not get your information back in the same order you entered it.

- Keys will be a single element

- Values can be a list or list within a list, numbers, etc.

- More than one entry per key is not allowed (no duplicate key is allowed)

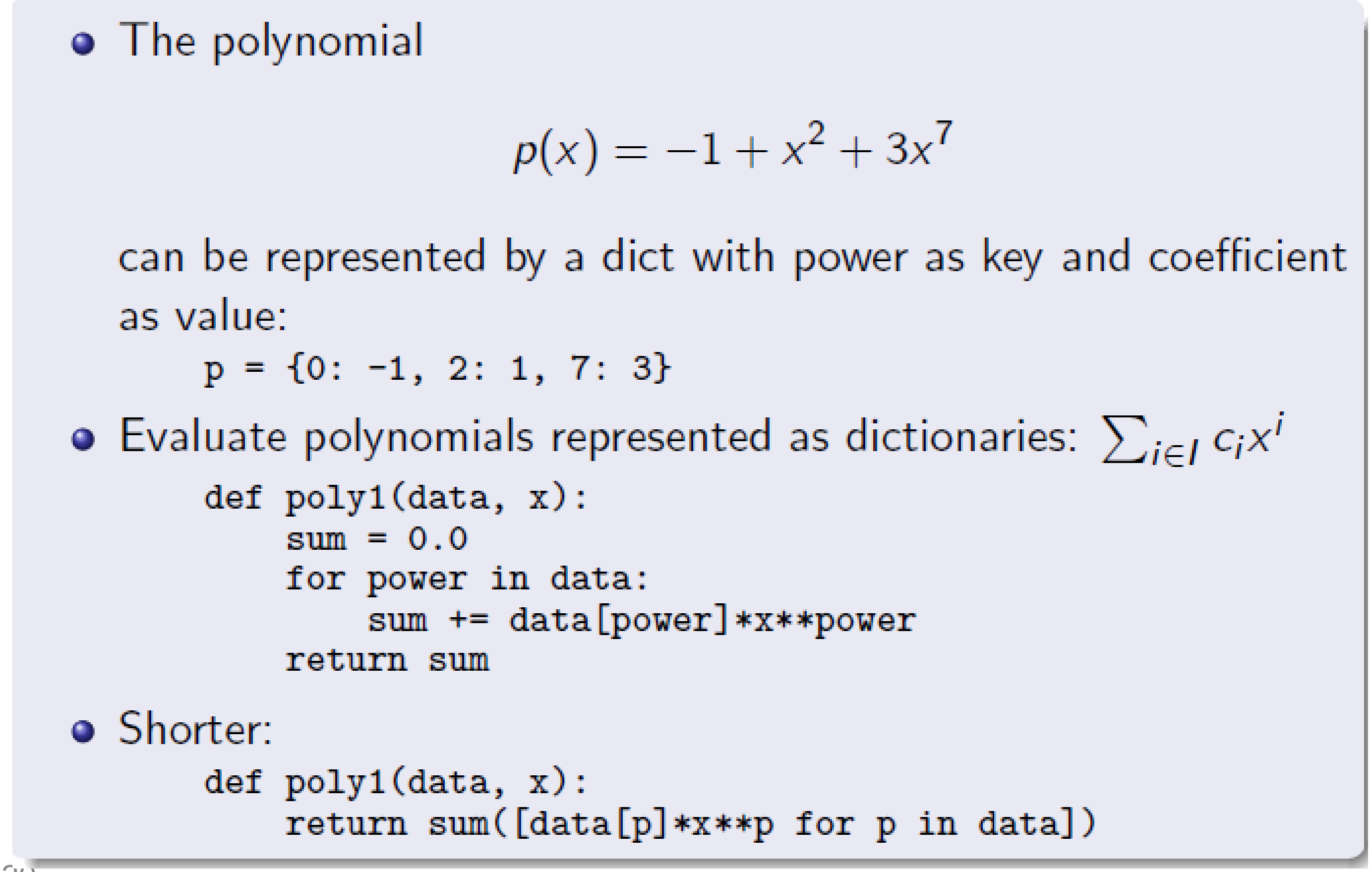
- The values in the dictionary can be of any type, while the keys must be immutable like numbers, tuples, or strings.

- Dictionary keys are case sensitive- Same key name but with the different cases are treated as different keys in Python dictionaries.

Examples: polynomials represented by dictionaries

Python objects that cannot change their contents are known as *immutable* data types and consist of int, float, complex, str, and tuple. Lists and dictionaries can change their contents and are called *mutable* objects

The keys in a dictionary are not restricted to be strings. In fact, any immutable Python object can be used as key.



- The polynomial

$$p(x) = -1 + x^2 + 3x^7$$

  can be represented by a dict with power as key and coefficient as value:

```
p = {0: -1, 2: 1, 7: 3}
```

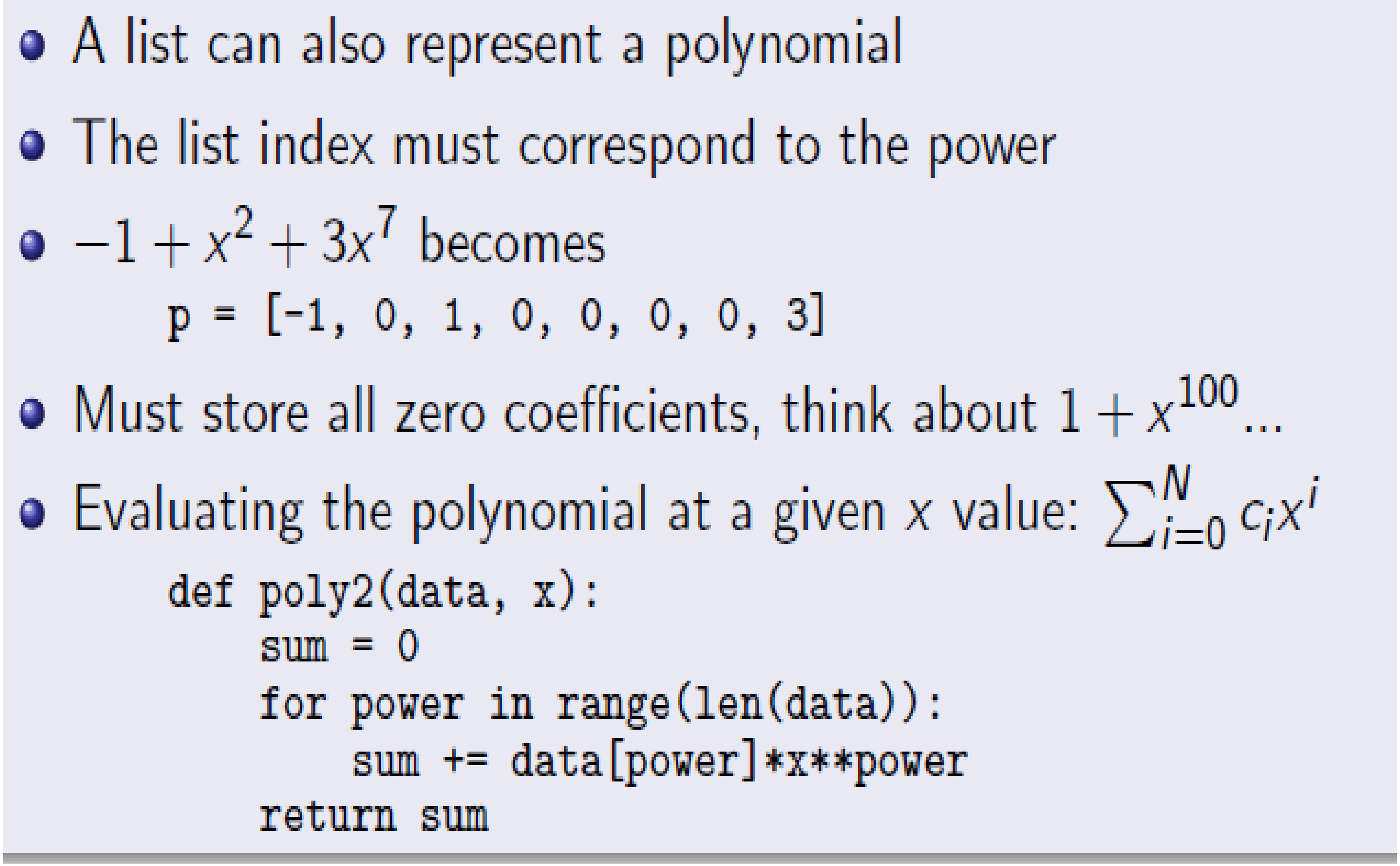
- Evaluate polynomials represented as dictionaries: $\sum_{i \in I} c_i x^i$

```
def poly1(data, x):
    sum = 0.0
    for power in data:
        sum += data[power]*x**power
    return sum
```

- Shorter:

```
def poly1(data, x):
    return sum([data[p]*x**p for p in data])
```

Lists as Dictionaries



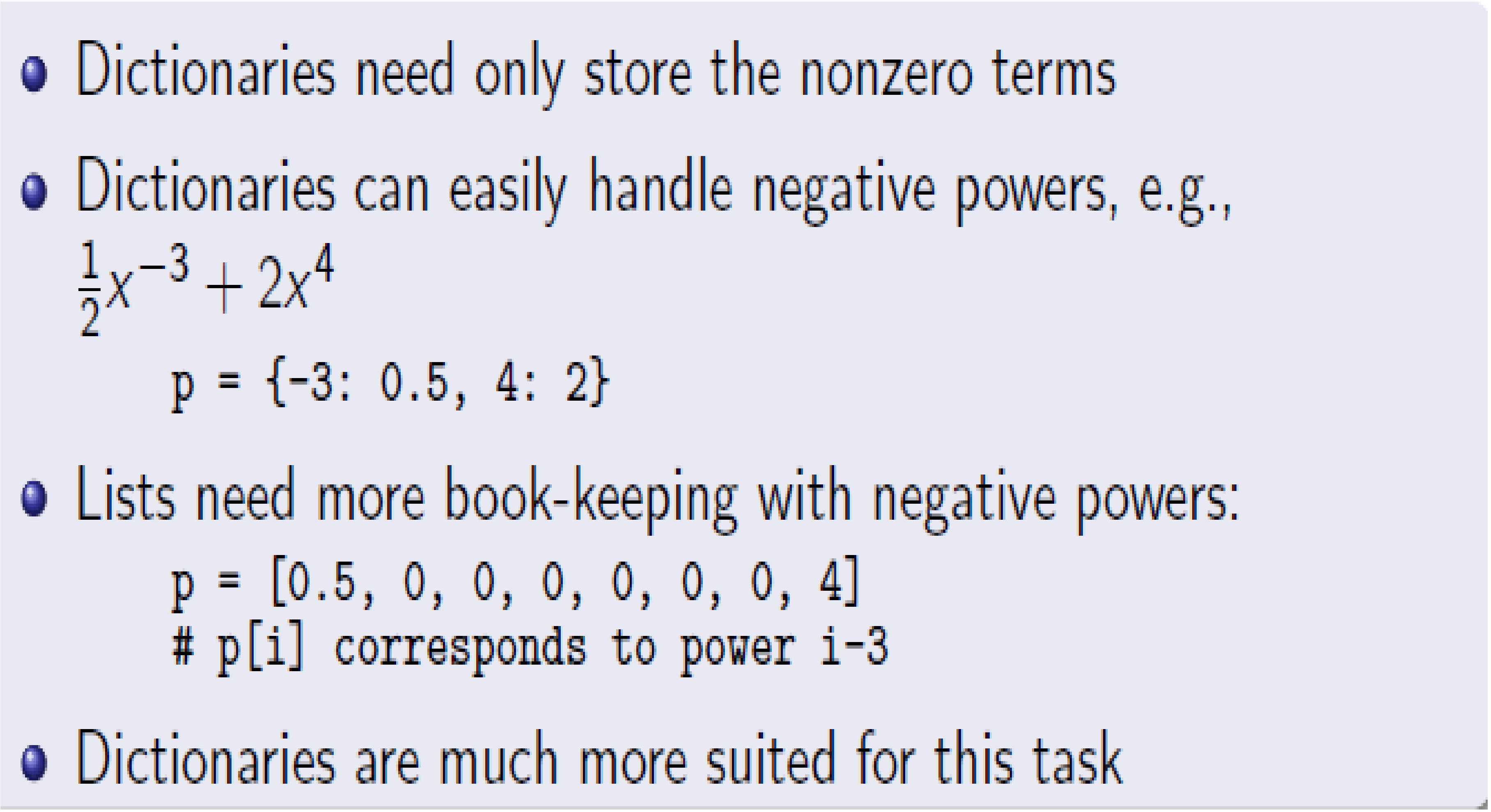
- A list can also represent a polynomial
- The list index must correspond to the power
- $-1 + x^2 + 3x^7$ becomes

```
p = [-1, 0, 1, 0, 0, 0, 0, 3]
```

- Must store all zero coefficients, think about $1 + x^{100}$...
- Evaluating the polynomial at a given x value: $\sum_{i=0}^{N} c_i x^i$

```
def poly2(data, x):
    sum = 0
    for power in range(len(data)):
        sum += data[power]*x**power
    return sum
```

WHAT IS BEST FOR ploynomial?



- Dictionaries need only store the nonzero terms
- Dictionaries can easily handle negative powers, e.g., $\frac{1}{2}x^{-3} + 2x^4$

```
    p = {-3: 0.5, 4: 2}
```

- Lists need more book-keeping with negative powers:

```
    p = [0.5, 0, 0, 0, 0, 0, 0, 4]
    # p[i] corresponds to power i-3
```

- Dictionaries are much more suited for this task

Reading file data into nested dictionaries

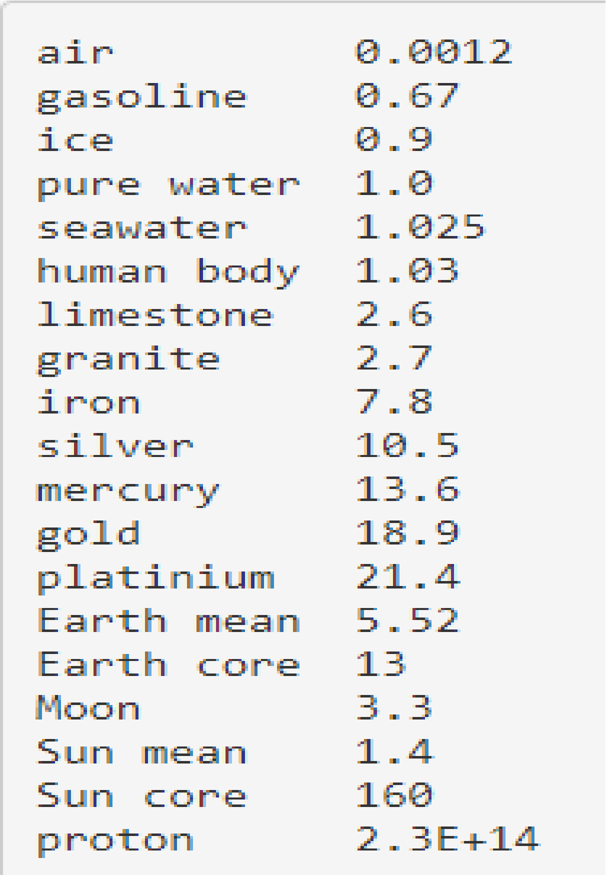
- Data file `table.dat` with measurements of four properties:

```
           A        B        C       D
1       11.7    0.035     2017    99.1
2        9.2    0.037     2019   101.2
3       12.2      no       no    105.2
4       10.1    0.031      no    102.1
5        9.1    0.033     2009   103.3
6        8.7    0.036     2015   101.9
```

- Create a dict `data[p][i]` (dict of dict) to hold measurement no. `i` of property `p` (`"A"`, `"B"`, etc.)
- Examine the first line: split it into words and initialize a dictionary with the property names as keys and empty dictionaries ({}) as values
- For each of the remaining lines: split line into words
- For each word after the first: if word is not "no", convert to float and store
- See the book for implementation details!

Read file data in dictionaries

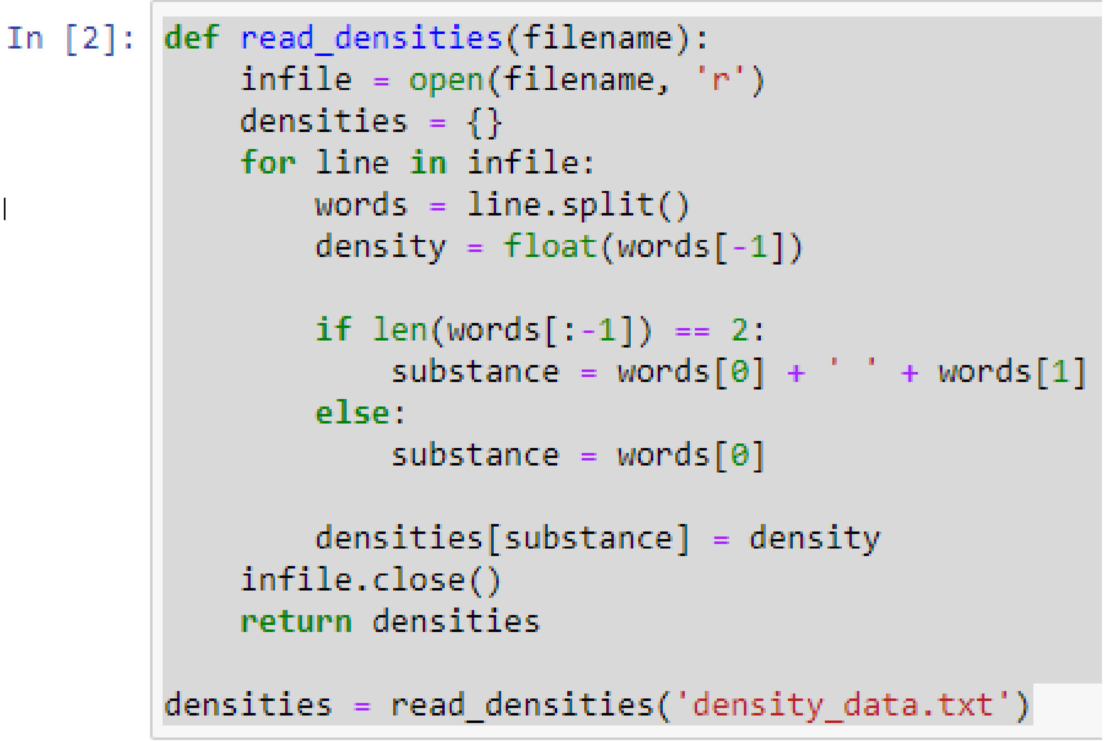
- **Density_data.txt**



```
air           0.0012
gasoline      0.67
ice           0.9
pure water    1.0
seawater      1.025
human body    1.03
limestone     2.6
granite       2.7
iron          7.8
silver        10.5
mercury       13.6
gold          18.9
platinium     21.4
Earth mean    5.52
Earth core    13
Moon          3.3
Sun mean      1.4
Sun core      160
proton        2.3E+14
```

Read file data in dictionaries

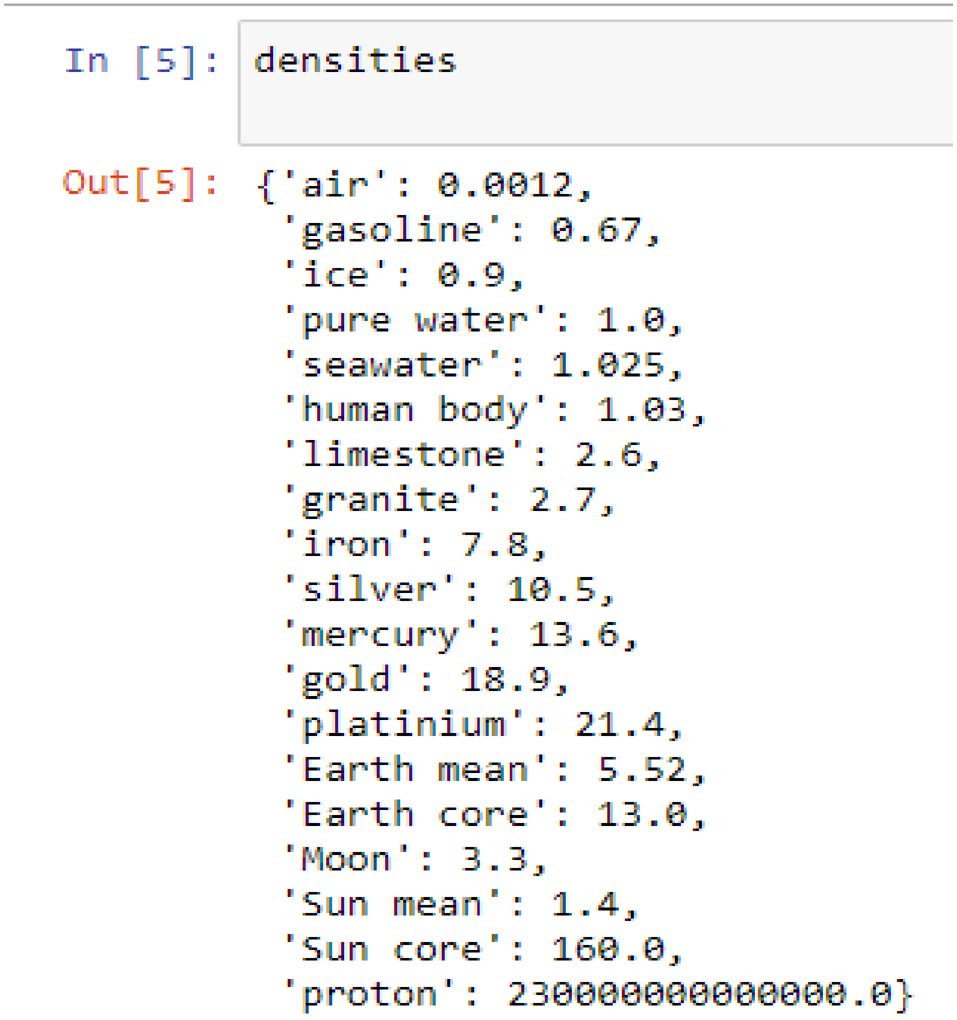
- We can read the density_data.txt file line by li
- split each line into words,
- Use a float conversion of the last word as density value,
- The remaining one or two words key in the dictionary.



In [2]:

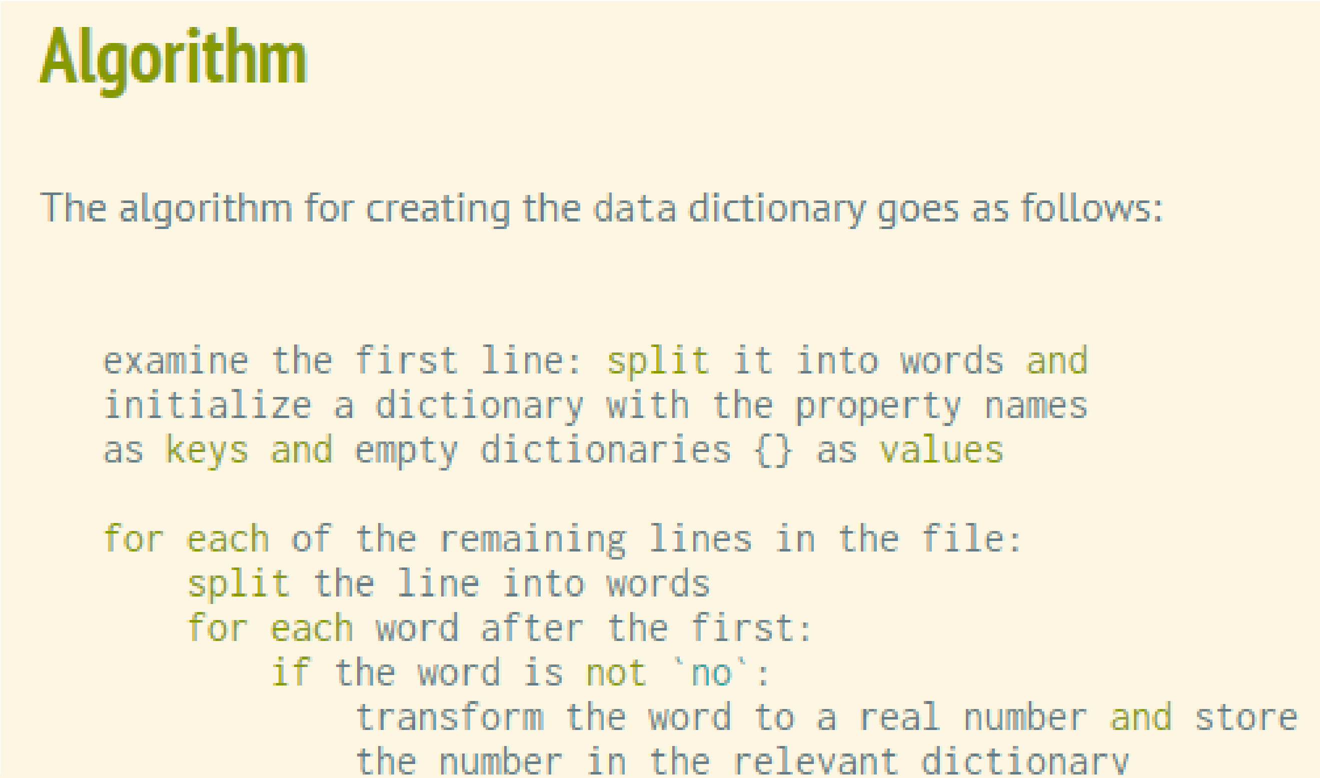
```
def read_densities(filename):
    infile = open(filename, 'r')
    densities = {}
    for line in infile:
        words = line.split()
        density = float(words[-1])

        if len(words[:-1]) == 2:
            substance = words[0] + ' ' + words[1]
        else:
            substance = words[0]

        densities[substance] = density
    infile.close()
    return densities

densities = read_densities('density_data.txt')
```

Read file data in dictionaries



```
In [5]: densities

Out[5]: {'air': 0.0012,
         'gasoline': 0.67,
         'ice': 0.9,
         'pure water': 1.0,
         'seawater': 1.025,
         'human body': 1.03,
         'limestone': 2.6,
         'granite': 2.7,
         'iron': 7.8,
         'silver': 10.5,
         'mercury': 13.6,
         'gold': 18.9,
         'platinium': 21.4,
         'Earth mean': 5.52,
         'Earth core': 13.0,
         'Moon': 3.3,
         'Sun mean': 1.4,
         'Sun core': 160.0,
         'proton': 230000000000000.0}
```

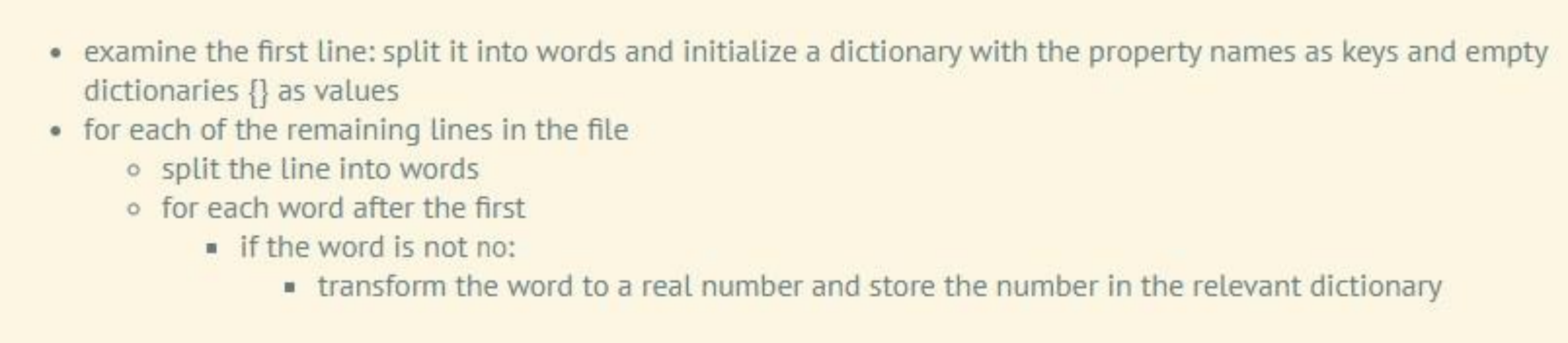
Read tabular file data into a dictionary



Algorithm

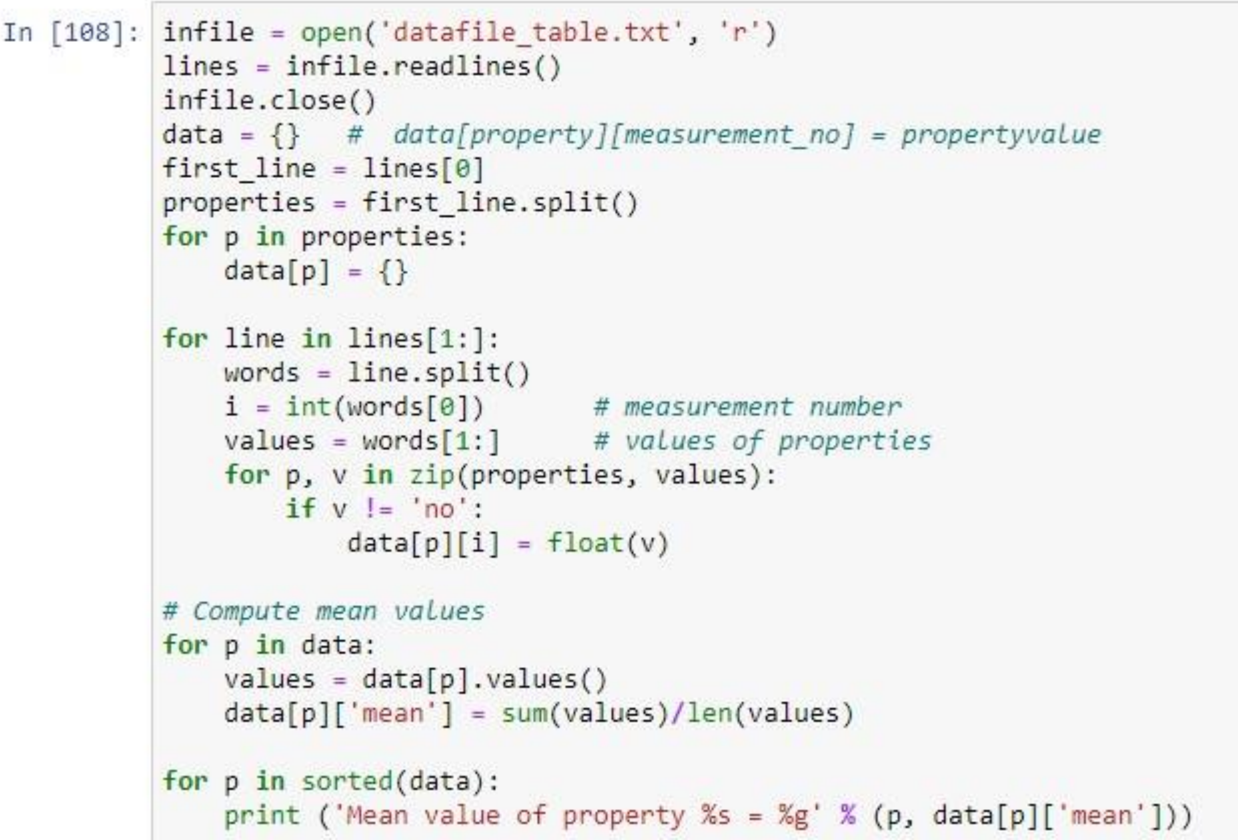
The algorithm for creating the data dictionary goes as follows:

```
examine the first line: split it into words and
initialize a dictionary with the property names
as keys and empty dictionaries {} as values

for each of the remaining lines in the file:
    split the line into words
    for each word after the first:
        if the word is not `no`:
            transform the word to a real number and store
            the number in the relevant dictionary
```

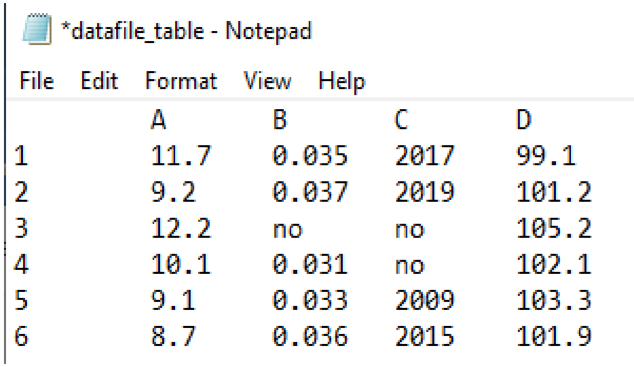


- examine the first line: split it into words and initialize a dictionary with the property names as keys and empty dictionaries {} as values
- for each of the remaining lines in the file
  - split the line into words
  - for each word after the first
    - if the word is not no:
      - transform the word to a real number and store the number in the relevant dictionary

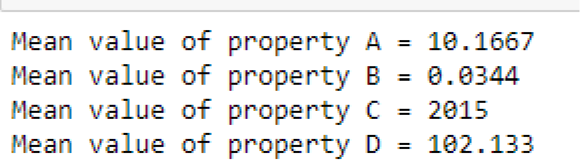
Read tabular file data into a dictionary



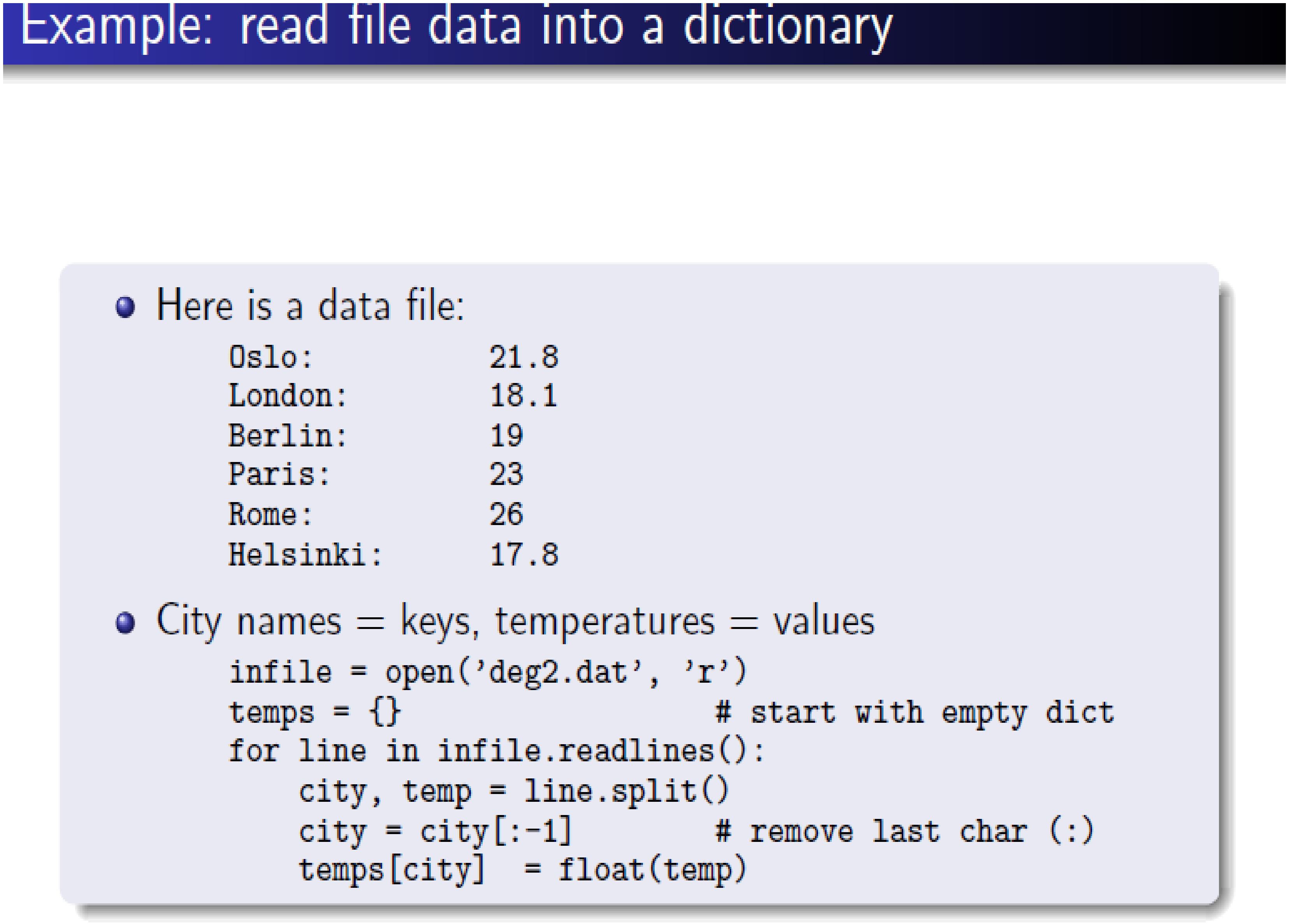
```
infile = open('datafile_table.txt', 'r')
lines = infile.readlines()
infile.close()
data = {}   #  data[property][measurement_no] = propertyvalue
first_line = lines[0]
properties = first_line.split()
for p in properties:
    data[p] = {}

for line in lines[1:]:
    words = line.split()
    i = int(words[0])       # measurement number
    values = words[1:]      # values of properties
    for p, v in zip(properties, values):
        if v != 'no':
            data[p][i] = float(v)

# Compute mean values
for p in data:
    values = data[p].values()
    data[p]['mean'] = sum(values)/len(values)

for p in sorted(data):
    print ('Mean value of property %s = %g' % (p, data[p]['mean']))
```



```
        A       B       C       D
1       11.7    0.035   2017    99.1
2       9.2     0.037   2019    101.2
3       12.2    no      no      105.2
4       10.1    0.031   no      102.1
5       9.1     0.033   2009    103.3
6       8.7     0.036   2015    101.9
```



```
Mean value of property A = 10.1667
Mean value of property B = 0.0344
Mean value of property C = 2015
Mean value of property D = 102.133
```



Example: read file data into a dictionary

- Here is a data file:

```
Oslo:          21.8
London:        18.1
Berlin:        19
Paris:         23
Rome:          26
Helsinki:      17.8
```

- City names = keys, temperatures = values

```
infile = open('deg2.dat', 'r')
temps = {}                         # start with empty dict
for line in infile.readlines():
    city, temp = line.split()
    city = city[:-1]               # remove last char (:)
    temps[city]  = float(temp)
```

Comparing stock prices (part 1)

- Problem: we want to compare the stock prices of Microsoft, Sun, and Google over a long period
- finance.yahoo.com offers such data in files with tabular form

```
Date,Open,High,Low,Close,Volume,Adj Close
2008-06-02,28.24,29.57,27.11,28.35,79237800,28.35
2008-05-01,28.50,30.53,27.95,28.32,69931800,28.32
2008-04-01,28.83,32.10,27.93,28.52,69066000,28.42
2008-03-03,27.24,29.59,26.87,28.38,74958500,28.28
2008-02-01,31.06,33.25,27.02,27.20,122053500,27.10
...
```

- Columns are separated by comma
- First column is the date, the final is the price of interest
- We can compare Microsoft and Sun from e.g. 1988 and add in Google from e.g. 2005
- For comparison we should normalize prices: Microsoft and Sun start at 1, Google at the max Sun/Microsoft price in 2005

Comparing stock prices (part 2)

Algorithm for file reading:

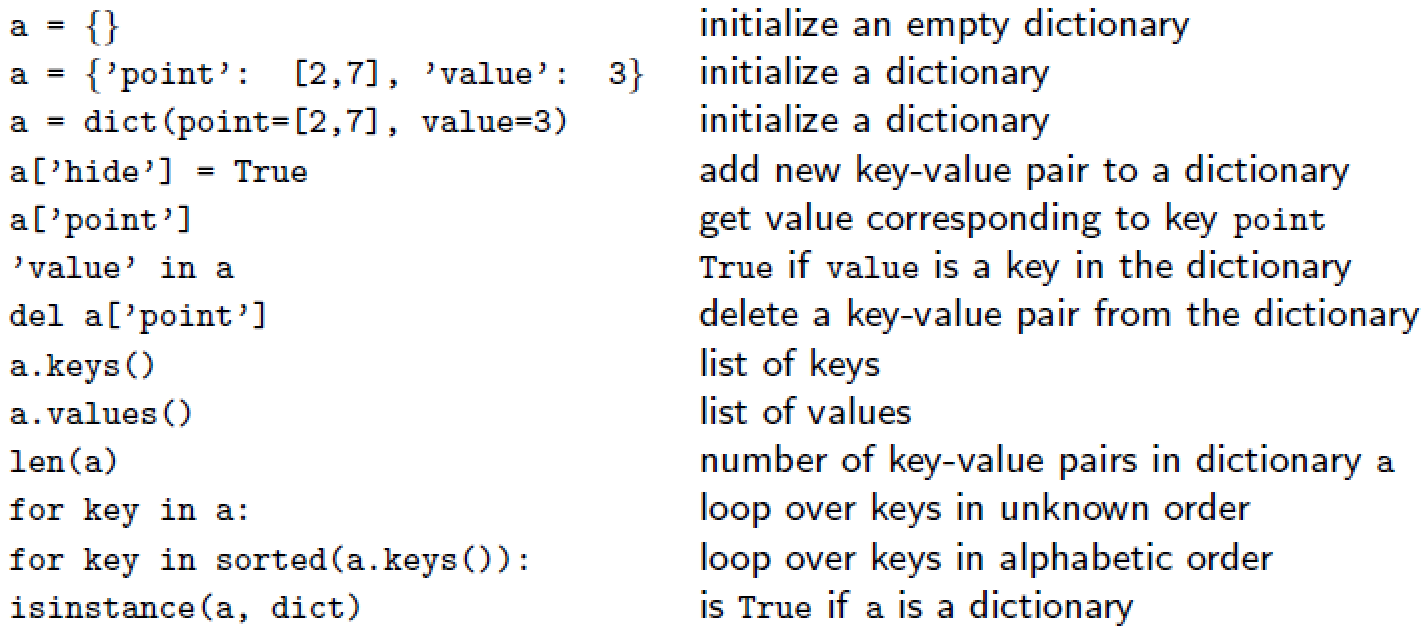
- Skip first line, read line by line, split line wrt. colon, store first "word" in a list of dates, final "word" in a list of prices; collect lists in dictionaries with company names as keys; make a function so it is easy to repeat for the three data files

Algorithm for file plotting:

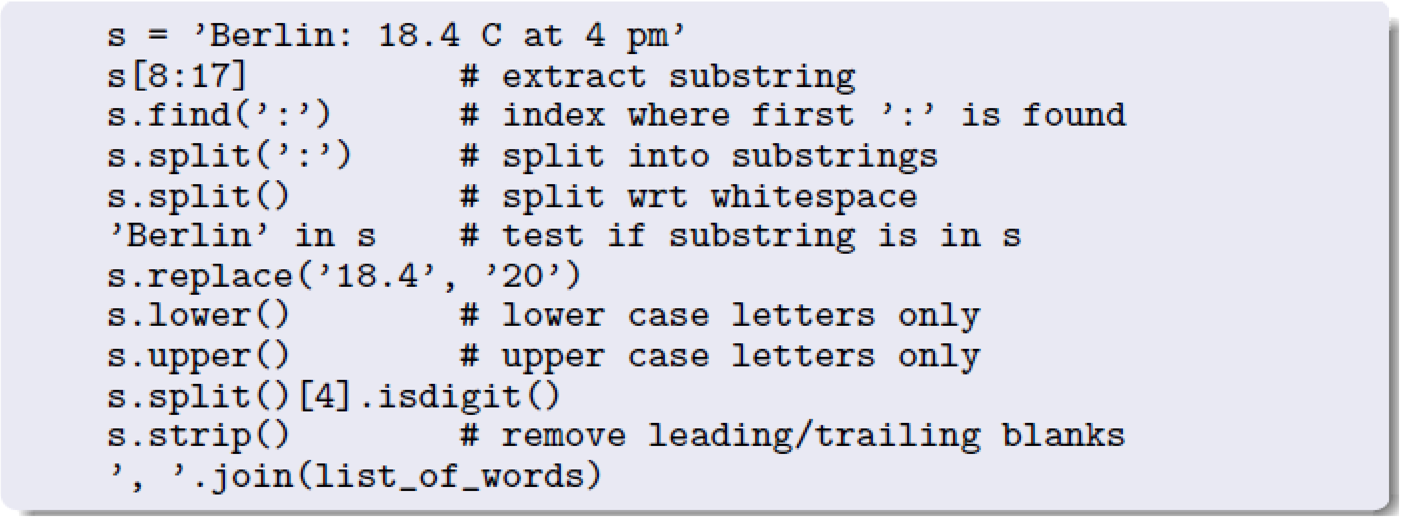
- Convert year-month-day time specifications in strings into coordinates along the x axis (use month indices for simplicity), Sun/Microsoft run 0,1,2,... while Google start at the Sun/Microsoft index corresponding to Jan 2005

See the book for all details. If you understand this example, you know and understand a lot!

Dictionary functionality



`a = {}`	initialize an empty dictionary
`a = {’point’: [2,7], ’value’: 3}`	initialize a dictionary
`a = dict(point=[2,7], value=3)`	initialize a dictionary
`a[’hide’] = True`	add new key-value pair to a dictionary
`a[’point’]`	get value corresponding to key point
`’value’ in a`	True if value is a key in the dictionary
`del a[’point’]`	delete a key-value pair from the dictionary
`a.keys()`	list of keys
`a.values()`	list of values
`len(a)`	number of key-value pairs in dictionary a
`for key in a:`	loop over keys in unknown order
`for key in sorted(a.keys()):`	loop over keys in alphabetic order
`isinstance(a, dict)`	is True if a is a dictionary



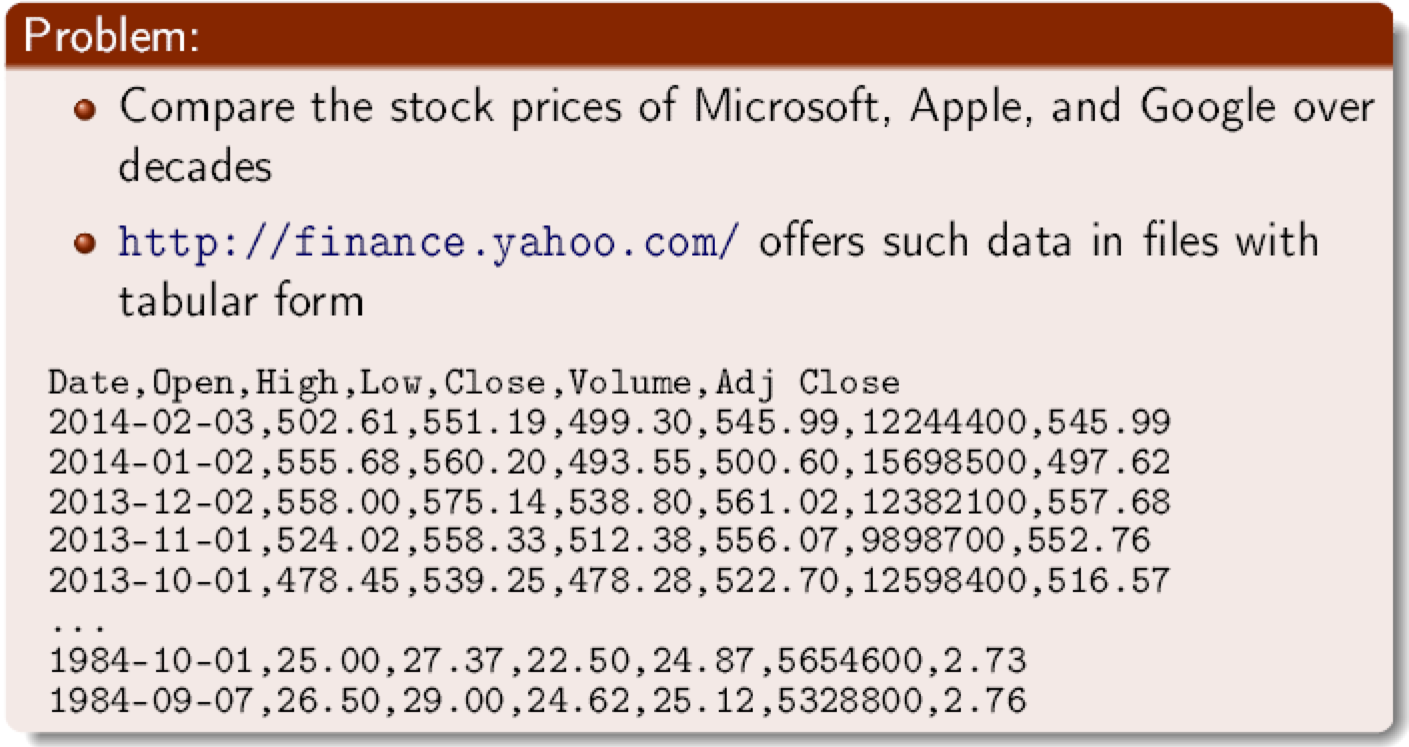
```
s = ’Berlin: 18.4 C at 4 pm’
s[8:17]          # extract substring
s.find(’:’)      # index where first ’:’ is found
s.split(’:’)     # split into substrings
s.split()        # split wrt whitespace
’Berlin’ in s    # test if substring is in s
s.replace(’18.4’, ’20’)
s.lower()        # lower case letters only
s.upper()        # upper case letters only
s.split()[4].isdigit()
s.strip()        # remove leading/trailing blanks
’, ’.join(list_of_words)
```

What are web pages?

- Web pages are nothing but text files
- Commands in the text files tell the browser that this is a headline, this is boldface text, here is an image, etc.
- The commands are written in the HTML language

```
<html>
<body bgcolor="orange">
<h1>A Very Simple Web Page</h1> <!-- headline -->
Ordinary text is written as ordinary text, but when we
need headlines, lists,
<ul>
<li><em>emphasized words</em>, or
<li> <b>boldfaced words</b>,
</ul>
we need to embed the text inside HTML tags. We can also
insert GIF or PNG images, taken from other Internet sites,
if desired.
<hr> <!-- horizontal line -->
<img src="http://www.simula.no/simula_logo.gif">
</body>
</html>
```

Download data from the web and vizualise

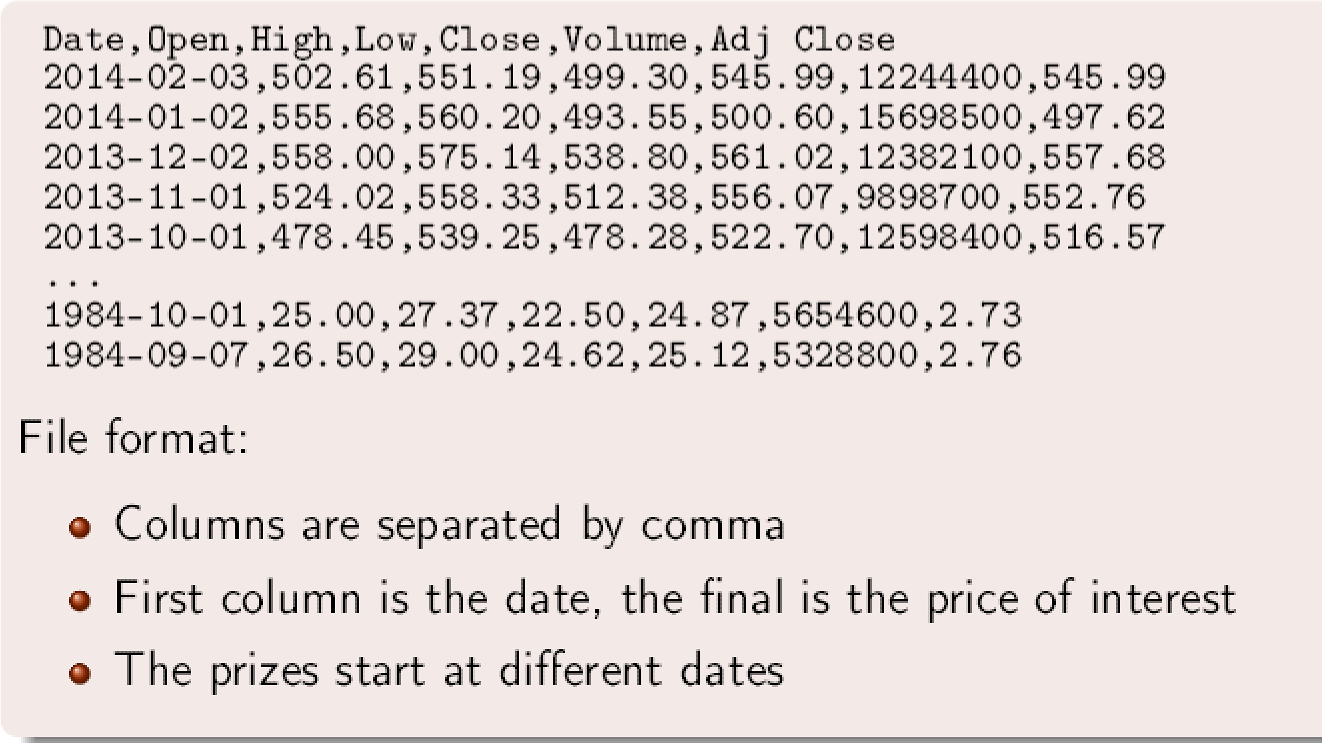


Problem:

- Compare the stock prices of Microsoft, Apple, and Google over decades
- http://finance.yahoo.com/ offers such data in files with tabular form

```
Date,Open,High,Low,Close,Volume,Adj Close
2014-02-03,502.61,551.19,499.30,545.99,12244400,545.99
2014-01-02,555.68,560.20,493.55,500.60,15698500,497.62
2013-12-02,558.00,575.14,538.80,561.02,12382100,557.68
2013-11-01,524.02,558.33,512.38,556.07,9898700,552.76
2013-10-01,478.45,539.25,478.28,522.70,12598400,516.57
...
1984-10-01,25.00,27.37,22.50,24.87,5654600,2.73
1984-09-07,26.50,29.00,24.62,25.12,5328800,2.76
```

Download data from the web and vizualise

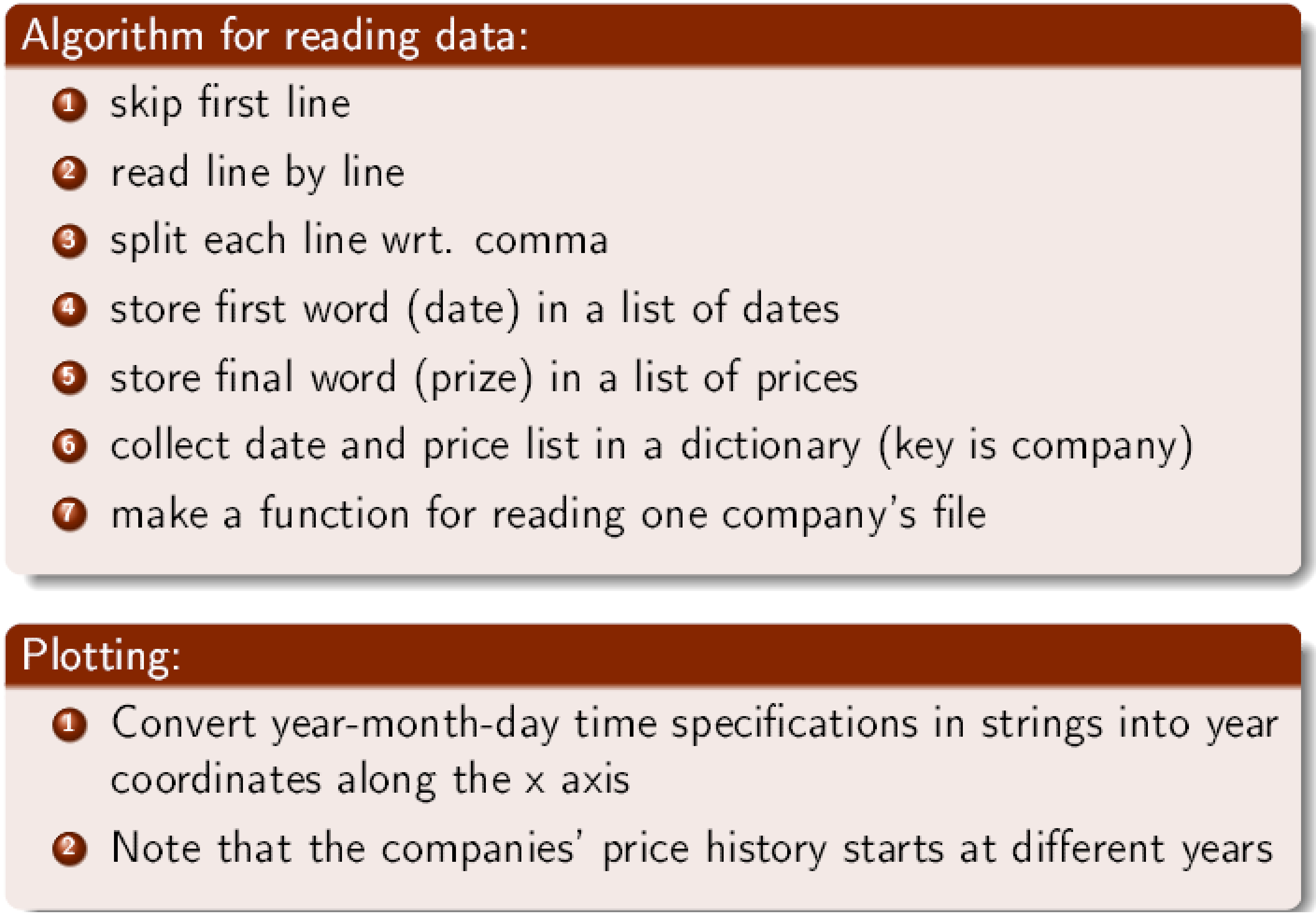


```
Date,Open,High,Low,Close,Volume,Adj Close
2014-02-03,502.61,551.19,499.30,545.99,12244400,545.99
2014-01-02,555.68,560.20,493.55,500.60,15698500,497.62
2013-12-02,558.00,575.14,538.80,561.02,12382100,557.68
2013-11-01,524.02,558.33,512.38,556.07,9898700,552.76
2013-10-01,478.45,539.25,478.28,522.70,12598400,516.57
...
1984-10-01,25.00,27.37,22.50,24.87,5654600,2.73
1984-09-07,26.50,29.00,24.62,25.12,5328800,2.76
```

File format:

- Columns are separated by comma
- First column is the date, the final is the price of interest
- The prizes start at different dates

Download data from the web and vizualise



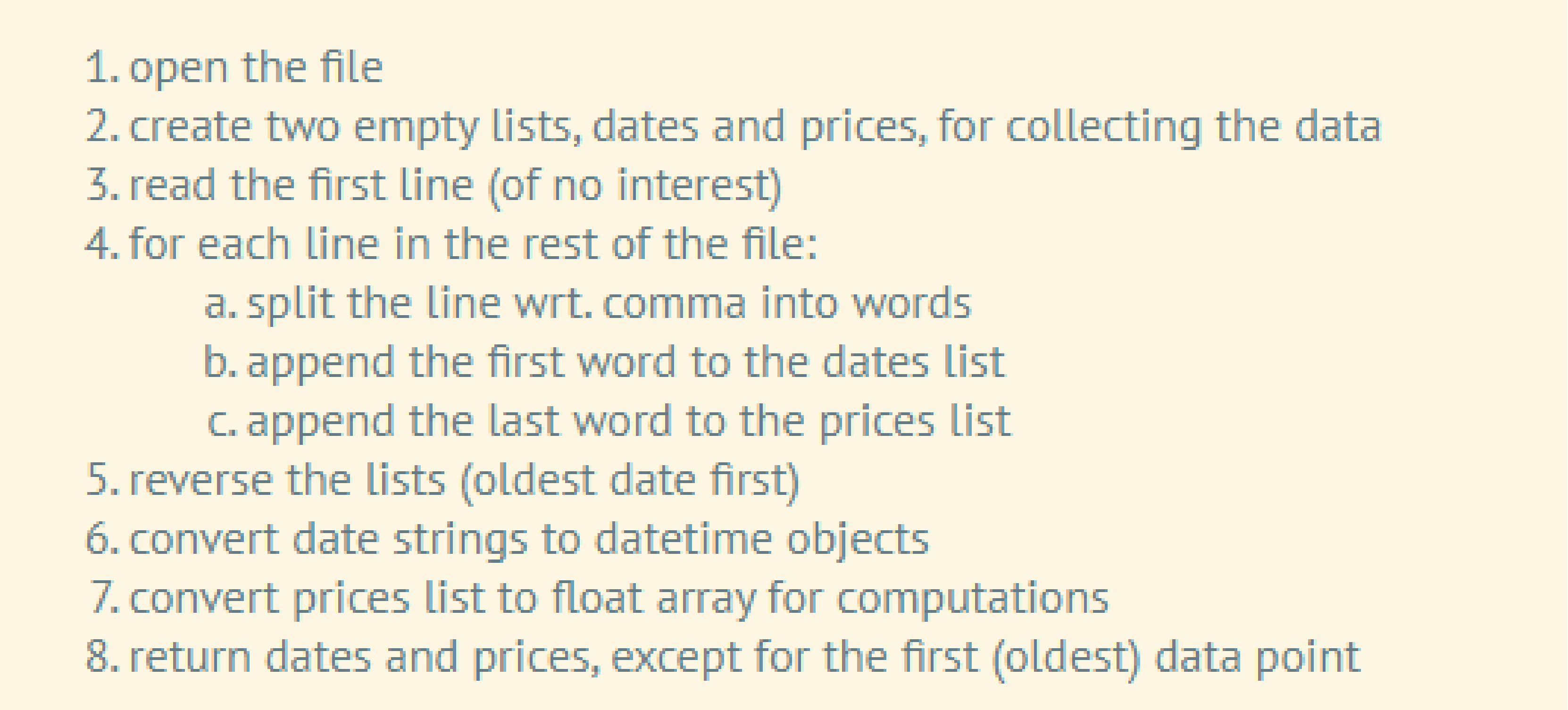
Algorithm for reading data:

1. skip first line
2. read line by line
3. split each line wrt. comma
4. store first word (date) in a list of dates
5. store final word (prize) in a list of prices
6. collect date and price list in a dictionary (key is company)
7. make a function for reading one company's file

Plotting:

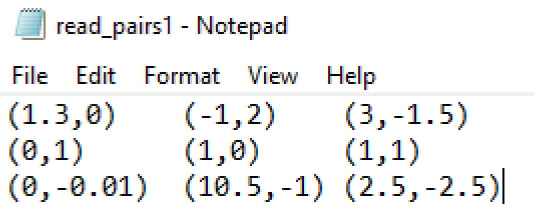
1. Convert year-month-day time specifications in strings into year coordinates along the x axis
2. Note that the companies' price history starts at different years

Algorithm

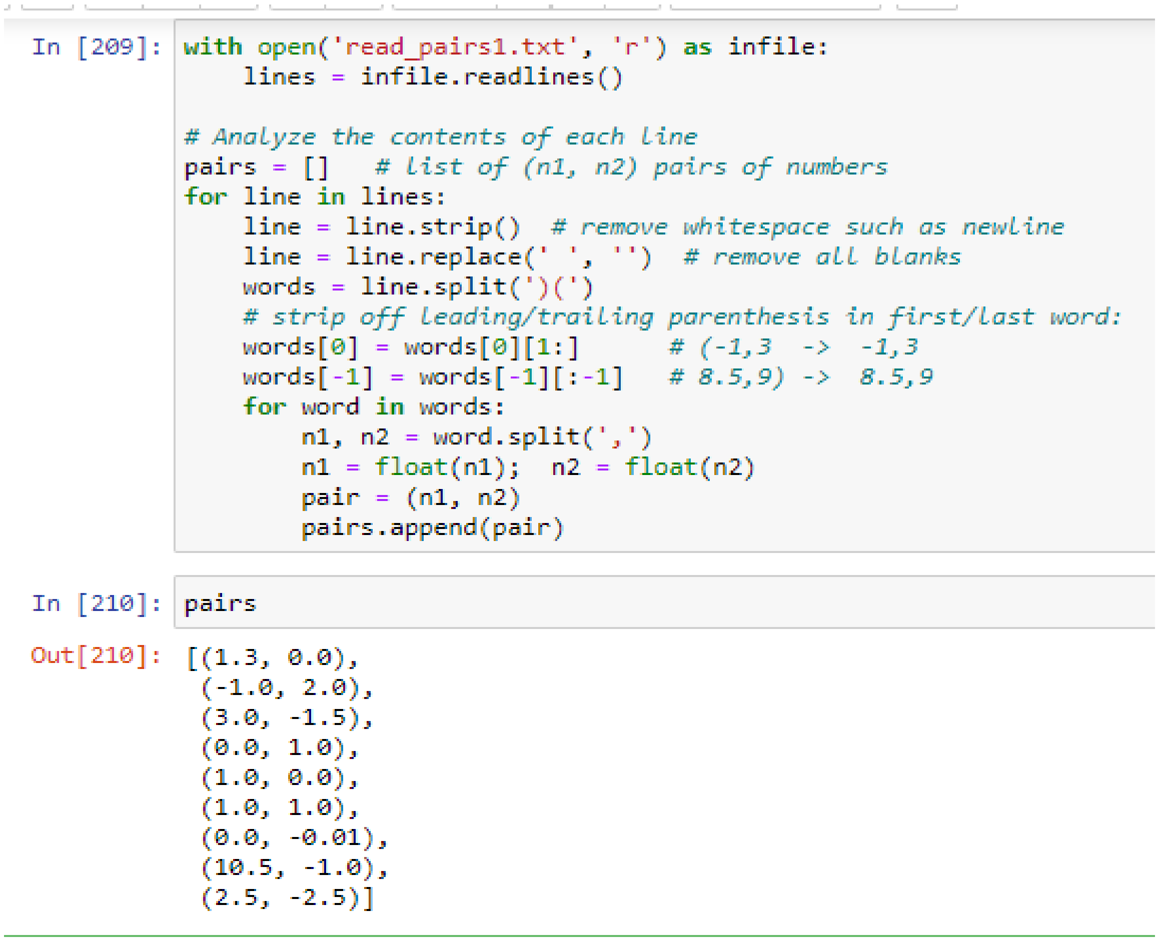


```
1. open the file
2. create two empty lists, dates and prices, for collecting the data
3. read the first line (of no interest)
4. for each line in the rest of the file:
        a. split the line wrt. comma into words
        b. append the first word to the dates list
        c. append the last word to the prices list
5. reverse the lists (oldest date first)
6. convert date strings to datetime objects
7. convert prices list to float array for computations
8. return dates and prices, except for the first (oldest) data point
```

Read pairs of numbers (x,y) from a file



```
(1.3,0)     (-1,2)     (3,-1.5)
(0,1)       (1,0)      (1,1)
(0,-0.01)   (10.5,-1)  (2.5,-2.5)
```



```
In [209]: with open('read_pairs1.txt', 'r') as infile:
              lines = infile.readlines()

          # Analyze the contents of each line
          pairs = []   # list of (n1, n2) pairs of numbers
          for line in lines:
              line = line.strip()  # remove whitespace such as newline
              line = line.replace(' ', '')  # remove all blanks
              words = line.split(')(')
              # strip off leading/trailing parenthesis in first/last word:
              words[0] = words[0][1:]       # (-1,3  ->  -1,3
              words[-1] = words[-1][:-1]   # 8.5,9) ->  8.5,9
              for word in words:
                  n1, n2 = word.split(',')
                  n1 = float(n1);  n2 = float(n2)
                  pair = (n1, n2)
                  pairs.append(pair)
```

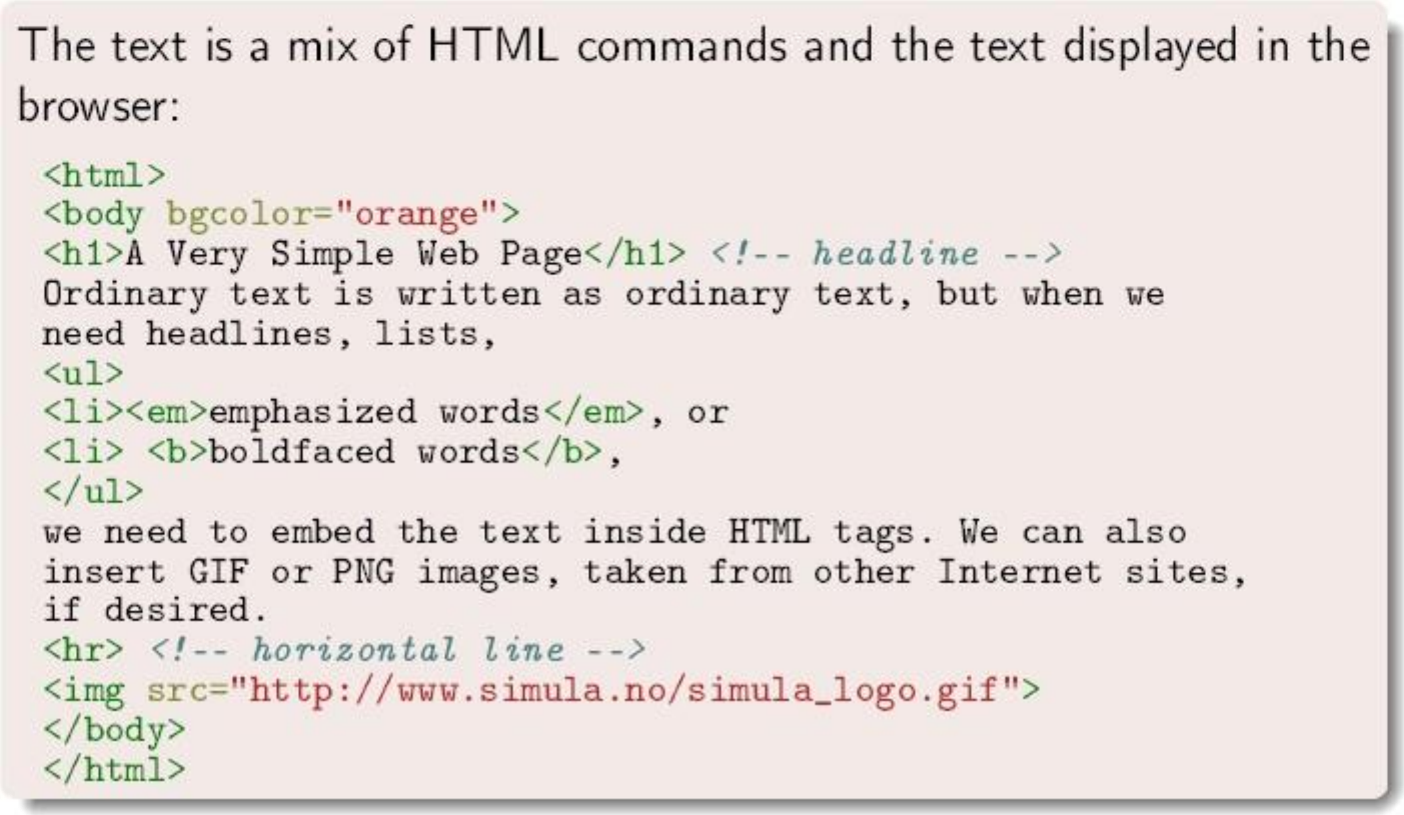
```
In [210]: pairs
```

```
Out[210]: [(1.3, 0.0),
           (-1.0, 2.0),
           (3.0, -1.5),
           (0.0, 1.0),
           (1.0, 0.0),
           (1.0, 1.0),
           (0.0, -0.01),
           (10.5, -1.0),
           (2.5, -2.5)]
```

Reading data from web pages

- This type of web address is called a URL (Uniform Resource Locator) or URI (Uniform Resource Identifier).
- The graphics you see in a web browser, i.e., the web page you see with your eyes, is produced by a series of commands that specifies the text on the page, the images, buttons to be pressed, etc.
- Roughly speaking, these commands are like statements in computer programs.
- The commands are stored in a text file and follow rules in a language, exactly as you are used to when writing statements in a programming language.
- The common language for defining web pages is HTML.
- A web page is then simply a text file with text containing HTML commands.
- Instead of a physical file, the web page can also be the output text from a program.
- In that case the URL is the name of the program file.

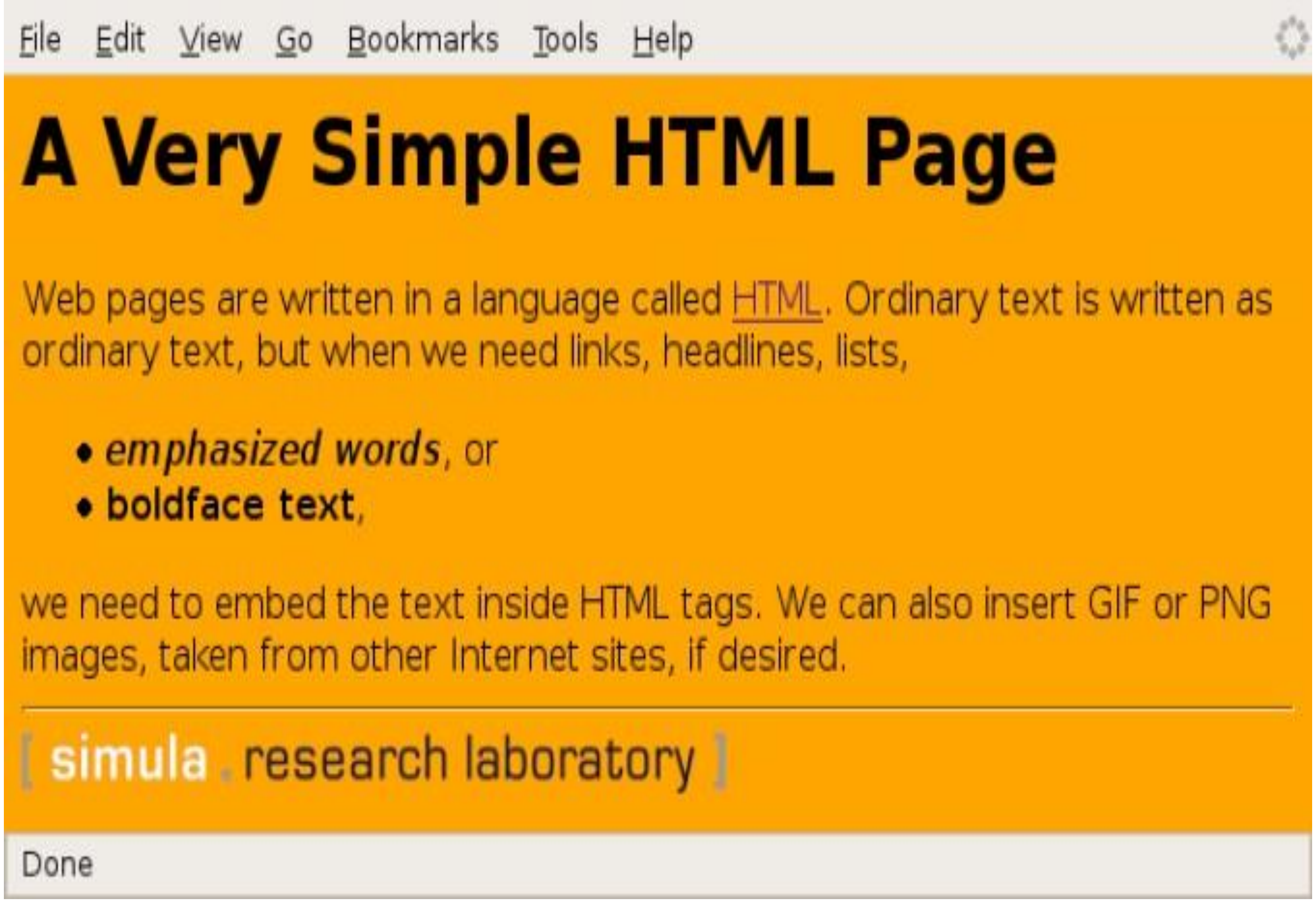
Web page is text file of HTML commands



The text is a mix of HTML commands and the text displayed in the browser:

```
<html>
<body bgcolor="orange">
<h1>A Very Simple Web Page</h1> <!-- headline -->
Ordinary text is written as ordinary text, but when we
need headlines, lists,
<ul>
<li><em>emphasized words</em>, or
<li> <b>boldfaced words</b>,
</ul>
we need to embed the text inside HTML tags. We can also
insert GIF or PNG images, taken from other Internet sites,
if desired.
<hr> <!-- horizontal line -->
<img src="http://www.simula.no/simula_logo.gif">
</body>
</html>
```

The web page generated by HTML code from the previous slide



File Edit View Go Bookmarks Tools Help

A Very Simple HTML Page

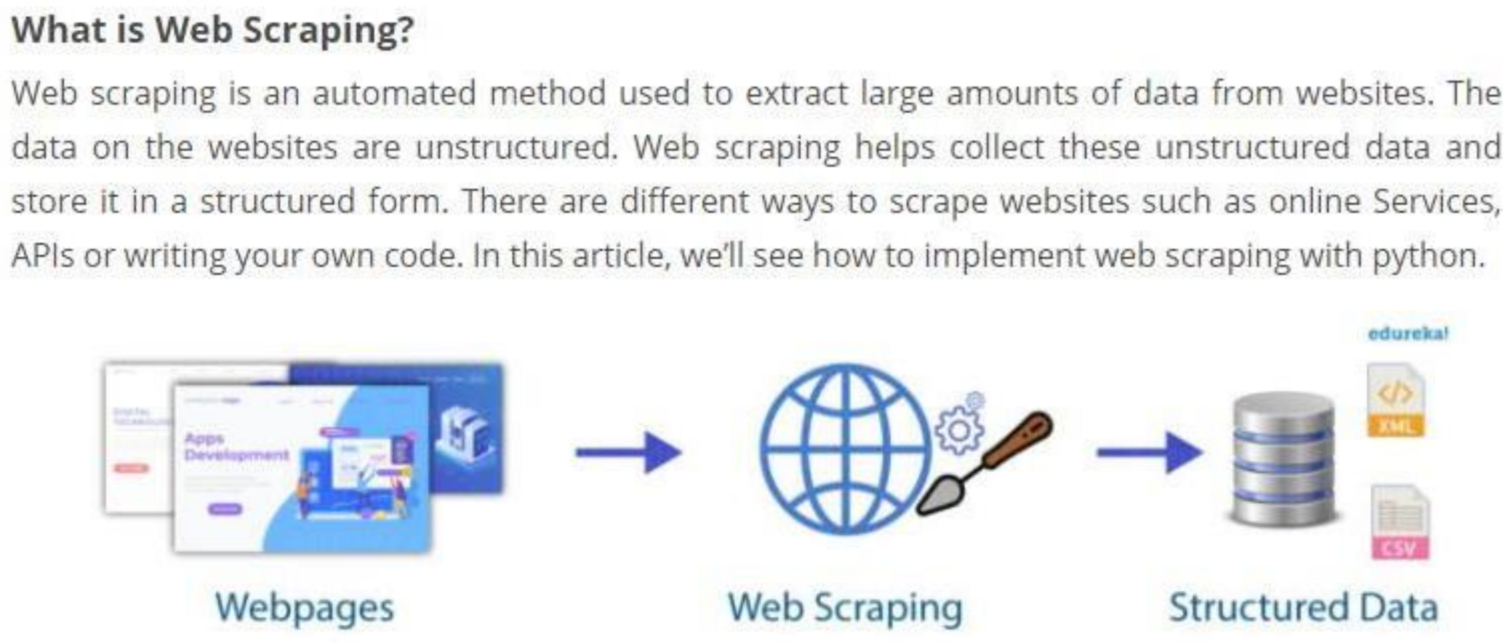
Web pages are written in a language called HTML. Ordinary text is written as ordinary text, but when we need links, headlines, lists,

- *emphasized words*, or
- **boldface text**,

we need to embed the text inside HTML tags. We can also insert GIF or PNG images, taken from other Internet sites, if desired.

[simula . research laboratory]
Done

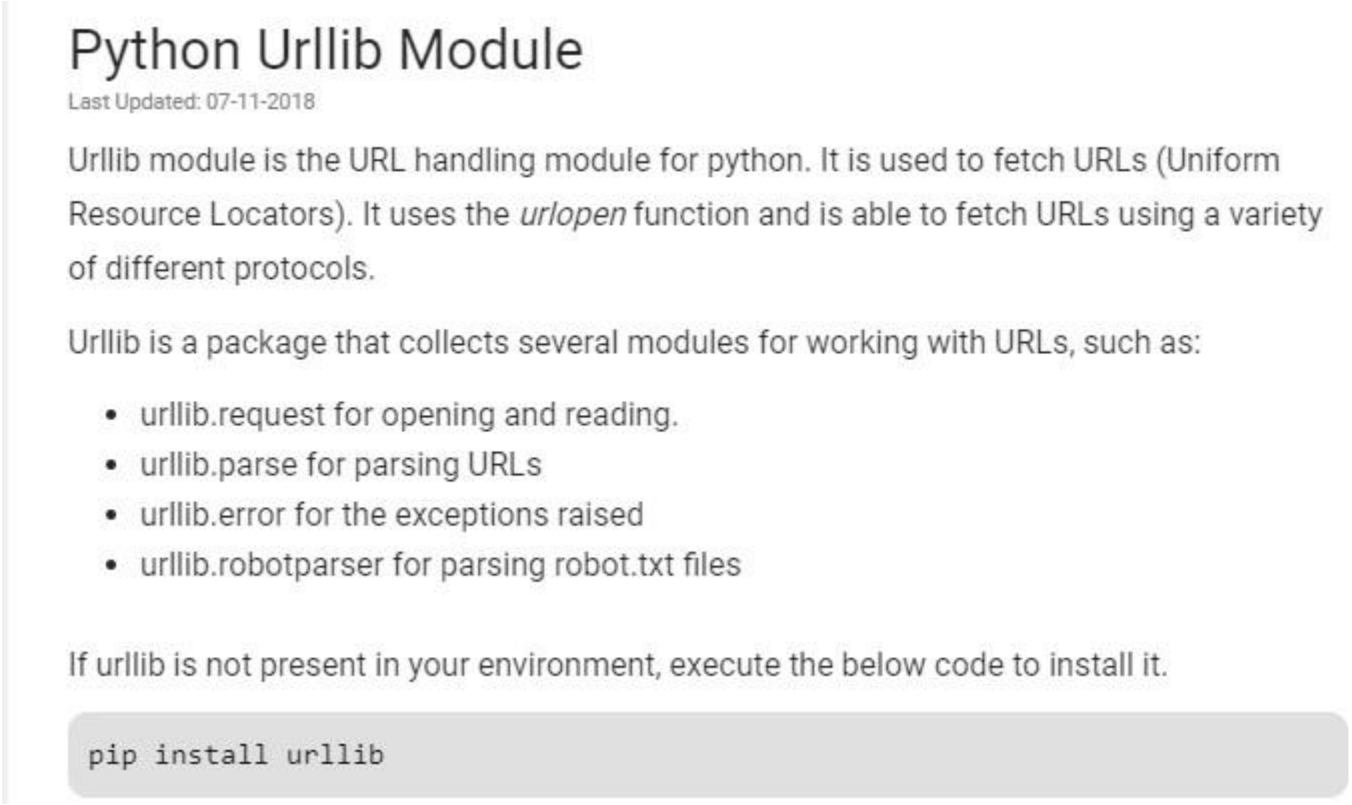
Programs can extract data from web pages



What is Web Scraping?

Web scraping is an automated method used to extract large amounts of data from websites. The data on the websites are unstructured. Web scraping helps collect these unstructured data and store it in a structured form. There are different ways to scrape websites such as online Services, APIs or writing your own code. In this article, we'll see how to implement web scraping with python.

Reading data from web pages



Python Urllib Module

Last Updated: 07-11-2018

Urllib module is the URL handling module for python. It is used to fetch URLs (Uniform Resource Locators). It uses the *urlopen* function and is able to fetch URLs using a variety of different protocols.

Urllib is a package that collects several modules for working with URLs, such as:

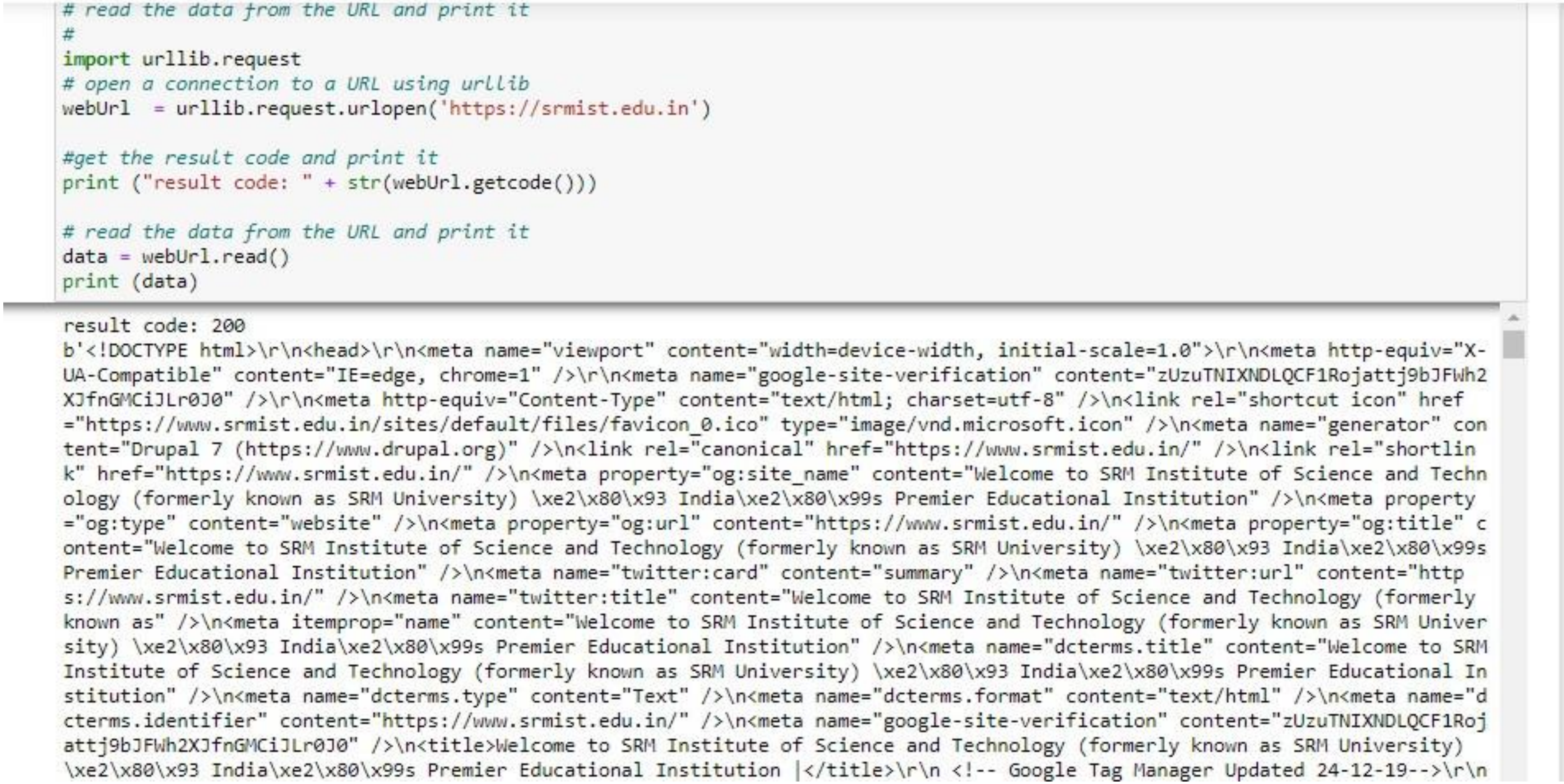
- urllib.request for opening and reading.
- urllib.parse for parsing URLs
- urllib.error for the exceptions raised
- urllib.robotparser for parsing robot.txt files

If urllib is not present in your environment, execute the below code to install it.

```
pip install urllib
```

- This module helps to define functions and classes to open URLs (mostly HTTP).
- One of the most simple ways to open such URLs is :
  urllib.request.urlopen(url)

Reading data from web pages



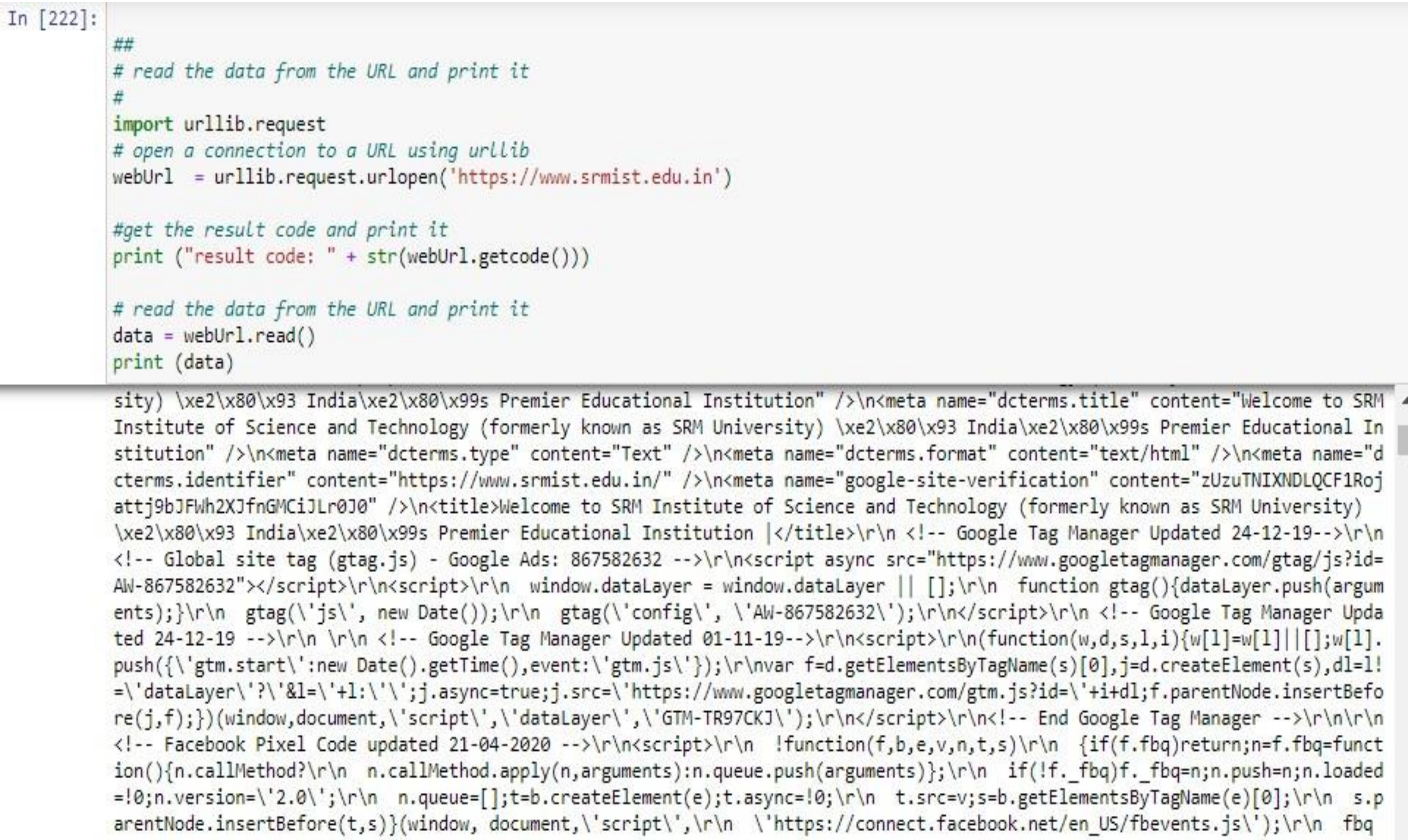
```
# read the data from the URL and print it
#
import urllib.request
# open a connection to a URL using urllib
webUrl  = urllib.request.urlopen('https://srmist.edu.in')

#get the result code and print it
print ("result code: " + str(webUrl.getcode()))

# read the data from the URL and print it
data = webUrl.read()
print (data)
```

```
result code: 200
b'<!DOCTYPE html>\r\n<head>\r\n<meta name="viewport" content="width=device-width, initial-scale=1.0">\r\n<meta http-equiv="X-
UA-Compatible" content="IE=edge, chrome=1" />\r\n<meta name="google-site-verification" content="zUzuTNIXNDLQCF1Rojattj9bJFWh2
XJfnGMCiJLr0J0" />\r\n<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />\n<link rel="shortcut icon" href
="https://www.srmist.edu.in/sites/default/files/favicon_0.ico" type="image/vnd.microsoft.icon" />\n<meta name="generator" con
tent="Drupal 7 (https://www.drupal.org)" />\n<link rel="canonical" href="https://www.srmist.edu.in/" />\n<link rel="shortlin
k" href="https://www.srmist.edu.in/" />\n<meta property="og:site_name" content="Welcome to SRM Institute of Science and Techn
ology (formerly known as SRM University) \xe2\x80\x93 India\xe2\x80\x99s Premier Educational Institution" />\n<meta property
="og:type" content="website" />\n<meta property="og:url" content="https://www.srmist.edu.in/" />\n<meta property="og:title" c
ontent="Welcome to SRM Institute of Science and Technology (formerly known as SRM University) \xe2\x80\x93 India\xe2\x80\x99s
Premier Educational Institution" />\n<meta name="twitter:card" content="summary" />\n<meta name="twitter:url" content="http
s://www.srmist.edu.in/" />\n<meta name="twitter:title" content="Welcome to SRM Institute of Science and Technology (formerly
known as" />\n<meta itemprop="name" content="Welcome to SRM Institute of Science and Technology (formerly known as SRM Univer
sity) \xe2\x80\x93 India\xe2\x80\x99s Premier Educational Institution" />\n<meta name="dcterms.title" content="Welcome to SRM
Institute of Science and Technology (formerly known as SRM University) \xe2\x80\x93 India\xe2\x80\x99s Premier Educational In
stitution" />\n<meta name="dcterms.type" content="Text" />\n<meta name="dcterms.format" content="text/html" />\n<meta name="d
cterms.identifier" content="https://www.srmist.edu.in/" />\n<meta name="google-site-verification" content="zUzuTNIXNDLQCF1Roj
attj9bJFWh2XJfnGMCiJLr0J0" />\n<title>Welcome to SRM Institute of Science and Technology (formerly known as SRM University)
\xe2\x80\x93 India\xe2\x80\x99s Premier Educational Institution |</title>\r\n <!-- Google Tag Manager Updated 24-12-19-->\r\n
```

Programs can extract data from web pages

- Import urllib
- Define your main function
- Declare the variable webUrl
- Then call the urlopen function on the URL lib library
- The URL we are opening is srm university home page
- Next, we going to print the result code
- Result code is retrieved by calling the getcode function on the webUrl variable we have created
- We going to convert that to a string, so that it can be concatenated with our string "result code"
- This will be a regular HTTP code "200", indicating http request is processed successfully

How to get HTML file from URL in Python



```
In [222]:
##
# read the data from the URL and print it
#
import urllib.request
# open a connection to a URL using urllib
webUrl  = urllib.request.urlopen('https://www.srmist.edu.in')

#get the result code and print it
print ("result code: " + str(webUrl.getcode()))

# read the data from the URL and print it
data = webUrl.read()
print (data)
```

```
sity) \xe2\x80\x93 India\xe2\x80\x99s Premier Educational Institution" />\n<meta name="dcterms.title" content="Welcome to SRM
Institute of Science and Technology (formerly known as SRM University) \xe2\x80\x93 India\xe2\x80\x99s Premier Educational In
stitution" />\n<meta name="dcterms.type" content="Text" />\n<meta name="dcterms.format" content="text/html" />\n<meta name="d
cterms.identifier" content="https://www.srmist.edu.in/" />\n<meta name="google-site-verification" content="zUzuTNIXNDLQCF1Roj
attj9bJFWh2XJfnGMCiJLr0J0" />\n<title>Welcome to SRM Institute of Science and Technology (formerly known as SRM University)
\xe2\x80\x93 India\xe2\x80\x99s Premier Educational Institution |</title>\r\n <!-- Google Tag Manager Updated 24-12-19-->\r\n
<!-- Global site tag (gtag.js) - Google Ads: 867582632 -->\r\n<script async src="https://www.googletagmanager.com/gtag/js?id=
AW-867582632"></script>\r\n<script>\r\n  window.dataLayer = window.dataLayer || [];\r\n  function gtag(){dataLayer.push(argum
ents);}\r\n  gtag(\'js\', new Date());\r\n  gtag(\'config\', \'AW-867582632\');\r\n</script>\r\n <!-- Google Tag Manager Upda
ted 24-12-19 -->\r\n \r\n <!-- Google Tag Manager Updated 01-11-19-->\r\n<script>\r\n(function(w,d,s,l,i){w[l]=w[l]||[];w[l].
push({\'gtm.start\':new Date().getTime(),event:\'gtm.js\'});\r\nvar f=d.getElementsByTagName(s)[0],j=d.createElement(s),dl=l!
=\'dataLayer\'?\'&l=\'+l:\'\';j.async=true;j.src=\'https://www.googletagmanager.com/gtm.js?id=\'+i+dl;f.parentNode.insertBefo
re(j,f);})(window,document,\'script\',\'dataLayer\',\'GTM-TR97CKJ\');\r\n</script>\r\n<!-- End Google Tag Manager -->\r\n\r\n
<!-- Facebook Pixel Code updated 21-04-2020 -->\r\n<script>\r\n  !function(f,b,e,v,n,t,s)\r\n  {if(f.fbq)return;n=f.fbq=funct
ion(){n.callMethod?\r\n  n.callMethod.apply(n,arguments):n.queue.push(arguments)};\r\n  if(!f._fbq)f._fbq=n;n.push=n;n.loaded
=!0;n.version=\'2.0\';\r\n  n.queue=[];t=b.createElement(e);t.async=!0;\r\n  t.src=v;s=b.getElementsByTagName(e)[0];\r\n  s.p
arentNode.insertBefore(t,s)}(window, document,\'script\',\r\n  \'https://connect.facebook.net/en_US/fbevents.js\');\r\n  fbq
```

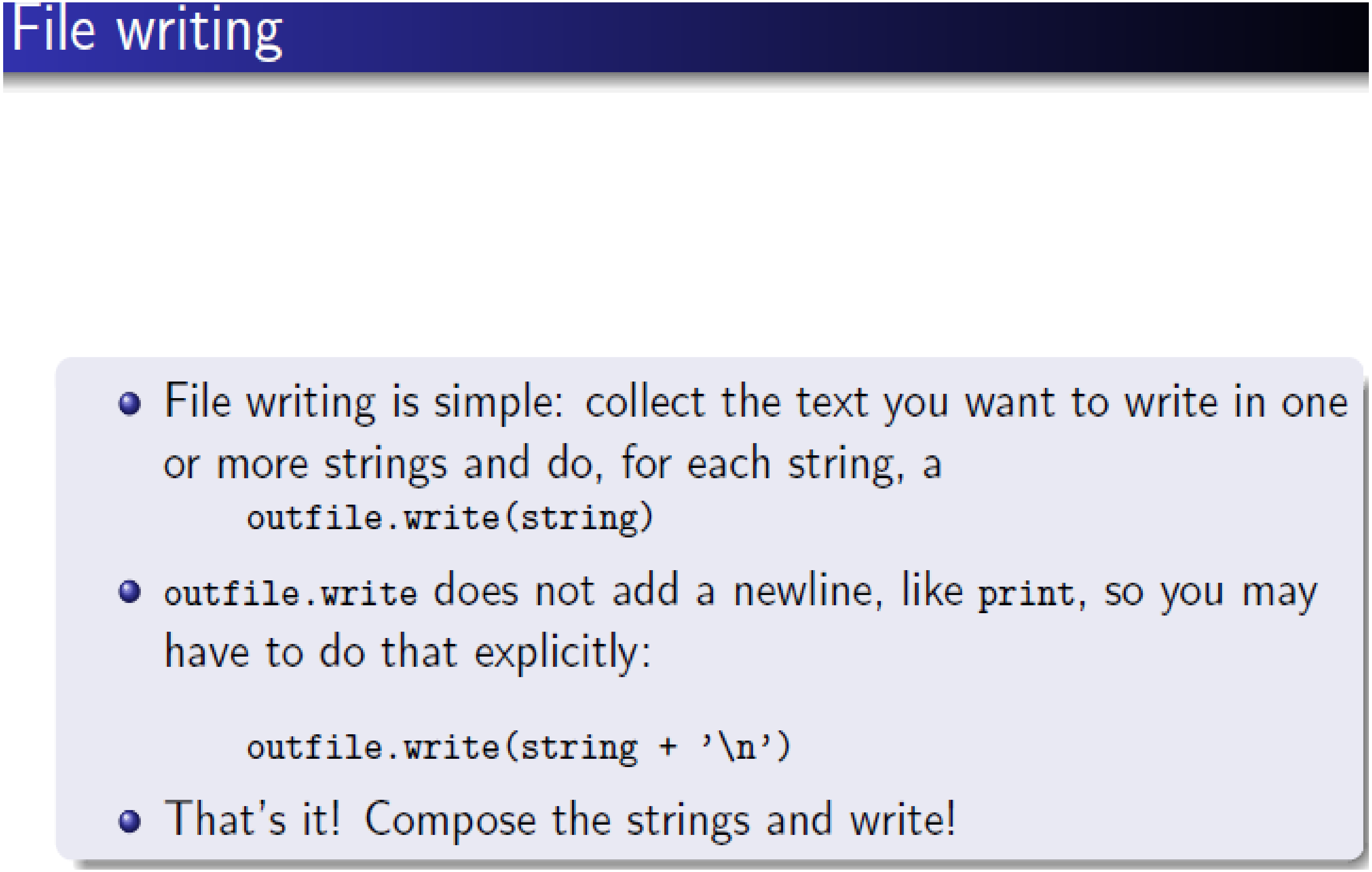
The structure of the Yahoo! weather forecast file

```
Oxford
Location: 4509E 2072N, 63 metres amsl
Estimated data is marked with a * after the value.
Missing data (more than 2 days missing in month) is marked by  ---.
Sunshine data taken from an automatic ...
   yyyy  mm   tmax    tmin      af    rain    sun
              degC    degC    days      mm  hours
   1853   1    8.4     2.7       4    62.8    ---
   1853   2    3.2    -1.8      19    29.3    ---
   1853   3    7.7    -0.6      20    25.9    ---
   1853   4   12.6     4.5       0    60.1    ---
   1853   5   16.8     6.1       0    59.5    ---

...

   2010   5   17.6     7.3       0    28.6  207.4
   2010   6   23.0    11.1       0    34.5  230.5
   2010   7   23.3*   14.1*      0*   24.4*  184.4*  Provisional
   2010  10   14.6     7.4       2    43.5  128.8   Provisional
```

We want to extract the weather conditions and the temperature

- Read the file line by line
- If a line contains `Current conditions`, grab the text between the `<h3>` tags on the next line
- If a line contains `forecast-temperature`, grab the temperature between the `<h3>` tags on the next line

```
lines = infile.readlines()
for i in range(len(lines)):
    line = lines[i]  # short form
    if ’Current conditions’ in line:
        weather = lines[i+1][4:-6]
    if ’forecast-temperature’ in line:
        temperature = float(lines[i+1][4:].split(’&’)[0])
        break  # everything is found, jump out of loop
```



File writing

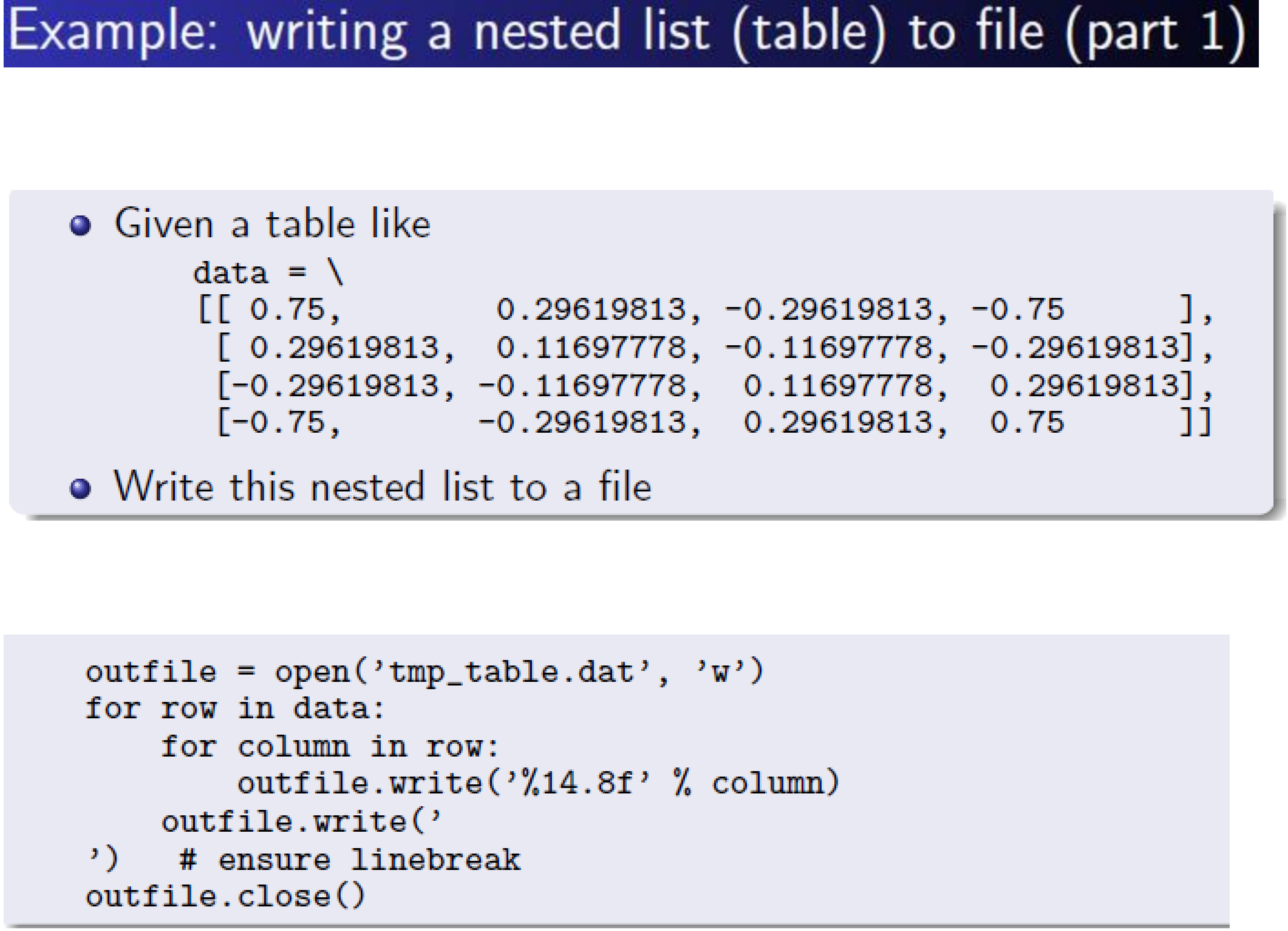
- File writing is simple: collect the text you want to write in one or more strings and do, for each string, a

```
    outfile.write(string)
```

- `outfile.write` does not add a newline, like `print`, so you may have to do that explicitly:

```
    outfile.write(string + '\n')
```

- That's it! Compose the strings and write!



Example: writing a nested list (table) to file (part 1)

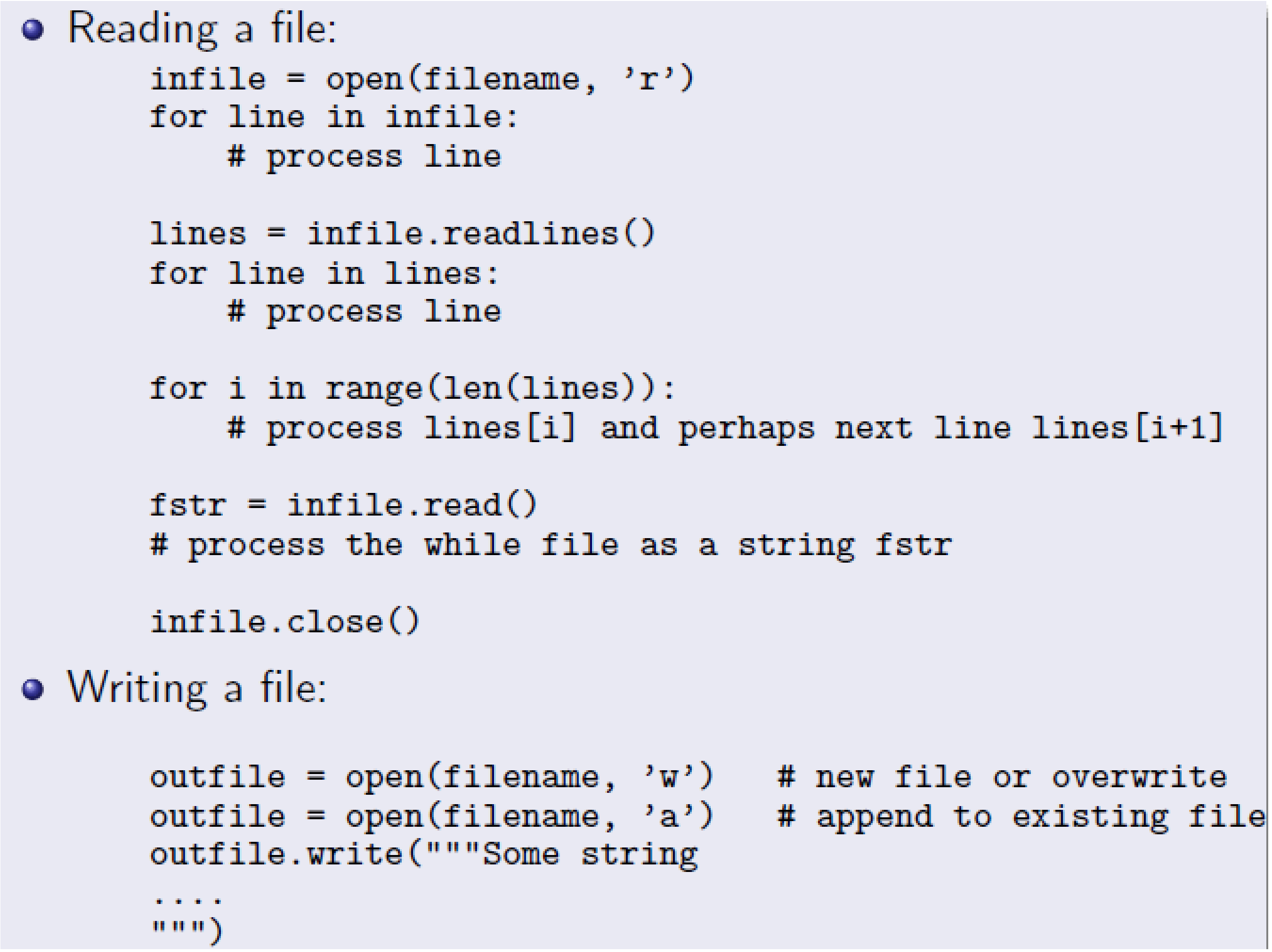
- Given a table like

```
data = \
[[ 0.75,        0.29619813, -0.29619813, -0.75      ],
 [ 0.29619813,  0.11697778, -0.11697778, -0.29619813],
 [-0.29619813, -0.11697778,  0.11697778,  0.29619813],
 [-0.75,       -0.29619813,  0.29619813,  0.75      ]]
```

- Write this nested list to a file

```
outfile = open(’tmp_table.dat’, ’w’)
for row in data:
    for column in row:
        outfile.write(’%14.8f’ % column)
    outfile.write(’
’)   # ensure linebreak
outfile.close()
```

Summary of file reading and writing

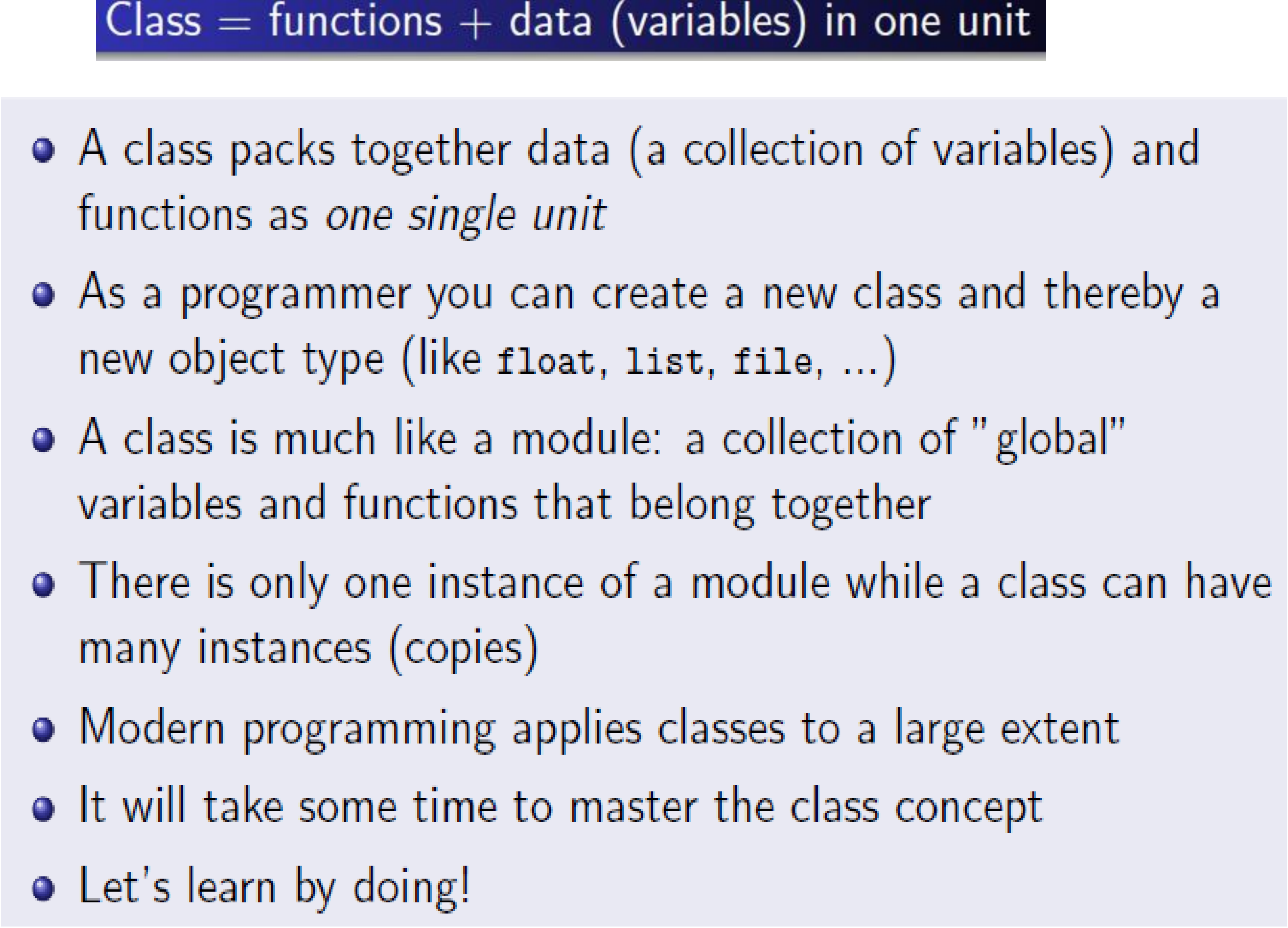


- Reading a file:

```
infile = open(filename, ’r’)
for line in infile:
    # process line

lines = infile.readlines()
for line in lines:
    # process line

for i in range(len(lines)):
    # process lines[i] and perhaps next line lines[i+1]

fstr = infile.read()
# process the while file as a string fstr

infile.close()
```

- Writing a file:

```
outfile = open(filename, ’w’)   # new file or overwrite
outfile = open(filename, ’a’)   # append to existing file
outfile.write("""Some string
....
""")
```



Class = functions + data (variables) in one unit

- A class packs together data (a collection of variables) and functions as *one single unit*
- As a programmer you can create a new class and thereby a new object type (like `float`, `list`, `file`, ...)
- A class is much like a module: a collection of "global" variables and functions that belong together
- There is only one instance of a module while a class can have many instances (copies)
- Modern programming applies classes to a large extent
- It will take some time to master the class concept
- Let's learn by doing!

Representing a function by a class; background

- Consider a function of t with a parameter v_0:

$$y(t; v_0) = v_0 t - \frac{1}{2} g t^2$$

- We need both v_0 and t to evaluate y (and $g = 9.81$)
- How should we implement this?

```
def y(t, v0):
    g = 9.81
    return v0*t - 0.5*g*t**2

# or v0 global?

def y(t):
    g = 9.81
    return v0*t - 0.5*g*t**2
```

- It is best to have `y` as function of `t` only (`y(t)`, see the book for a thorough discussion)
- Two possibilites for `y(t)`: `v0` as global variable (bad solution!) or `y` as a class (good solution!)

__init__ method ,self in Class

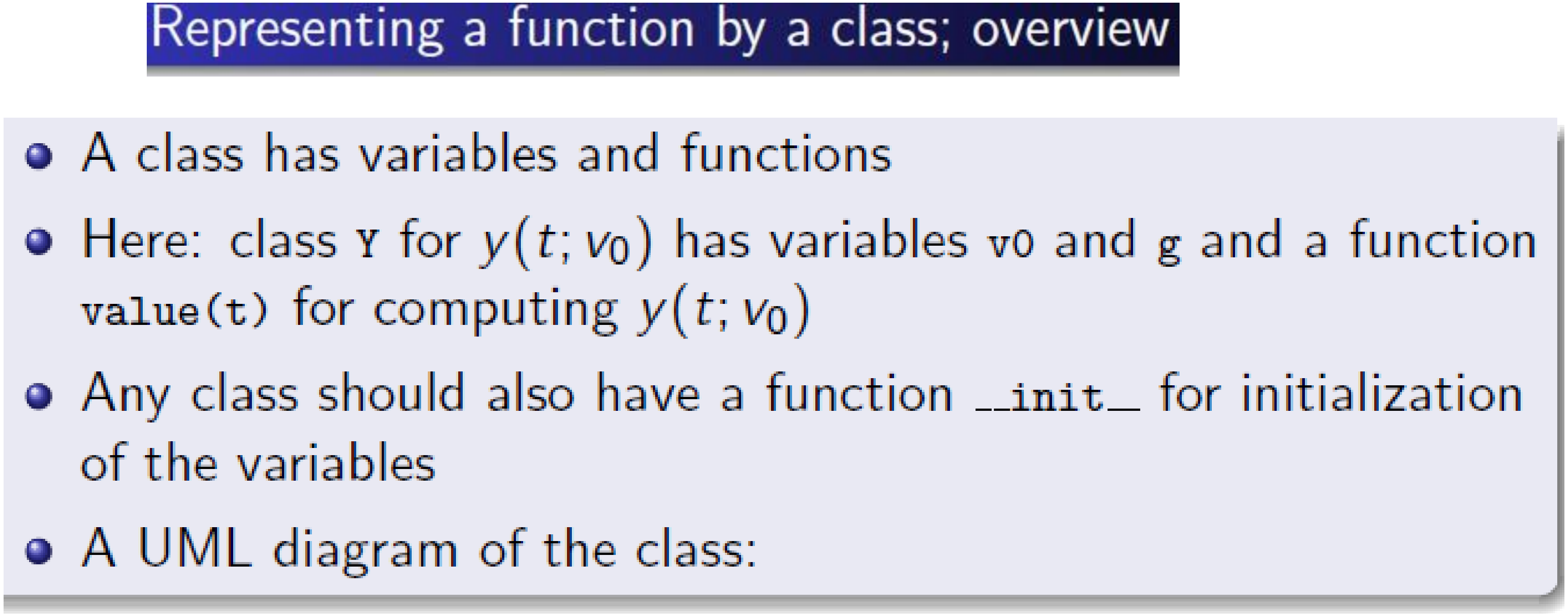
Whenever we create a class in Python, the programmer needs a way to access its *attributes* and *methods*. In most languages, there is a fixed syntax assigned to refer to attributes and methods; for example, C++ uses this for reference.

In Python, the word self is the first parameter of methods that represents the instance of the class. Therefore, in order to call attributes and methods of a class, the programmer needs to use self

The __init__ method is similar to **constructors** in C++ and Java. Constructors are used to initialize the object’s state.

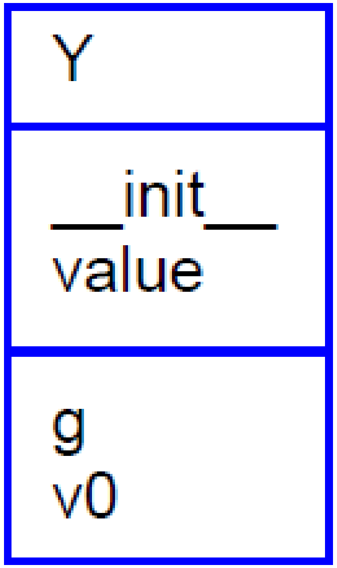
The task of constructors is to initialize(assign values) to the data members of the class when an object of class is created.

Like methods, a constructor also contains of statements(i.e. instructions) that are executed at time of Object creation. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.



Representing a function by a class; overview

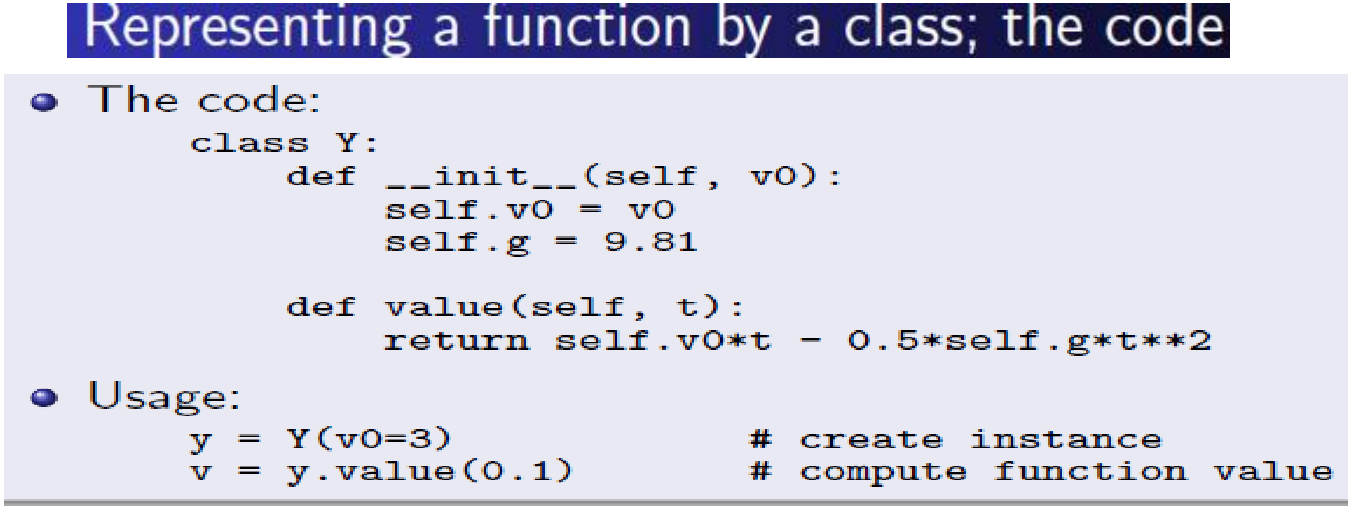
- A class has variables and functions
- Here: class `Y` for $y(t; v_0)$ has variables `v0` and `g` and a function `value(t)` for computing $y(t; v_0)$
- Any class should also have a function `__init__` for initialization of the variables
- A UML diagram of the class:



Y

__init__
value

g
v0



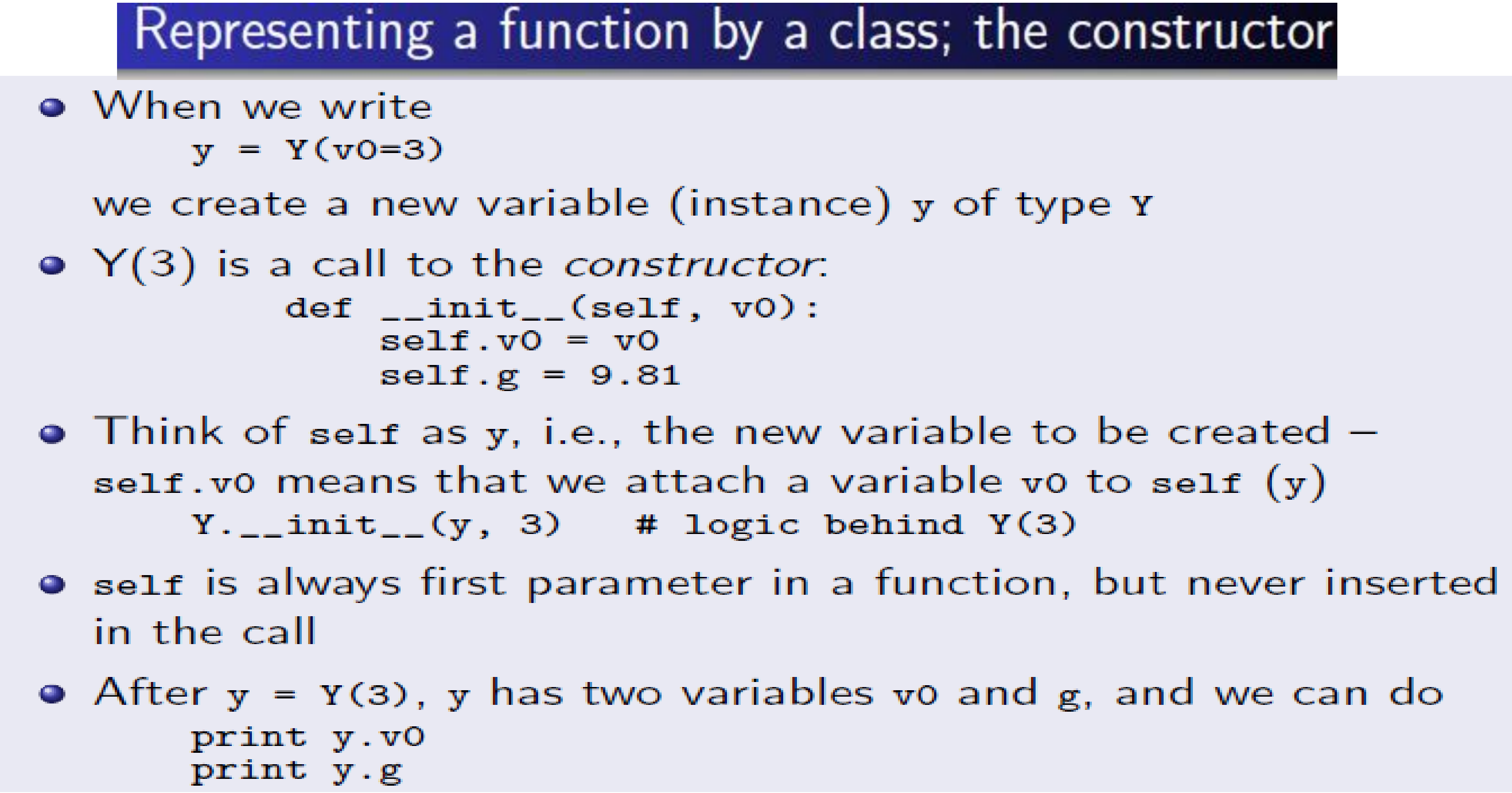
Representing a function by a class; the code

- The code:

```
class Y:
    def __init__(self, v0):
        self.v0 = v0
        self.g = 9.81

    def value(self, t):
        return self.v0*t - 0.5*self.g*t**2
```

- Usage:

```
y = Y(v0=3)                 # create instance
v = y.value(0.1)            # compute function value
```



Representing a function by a class; the constructor

- When we write

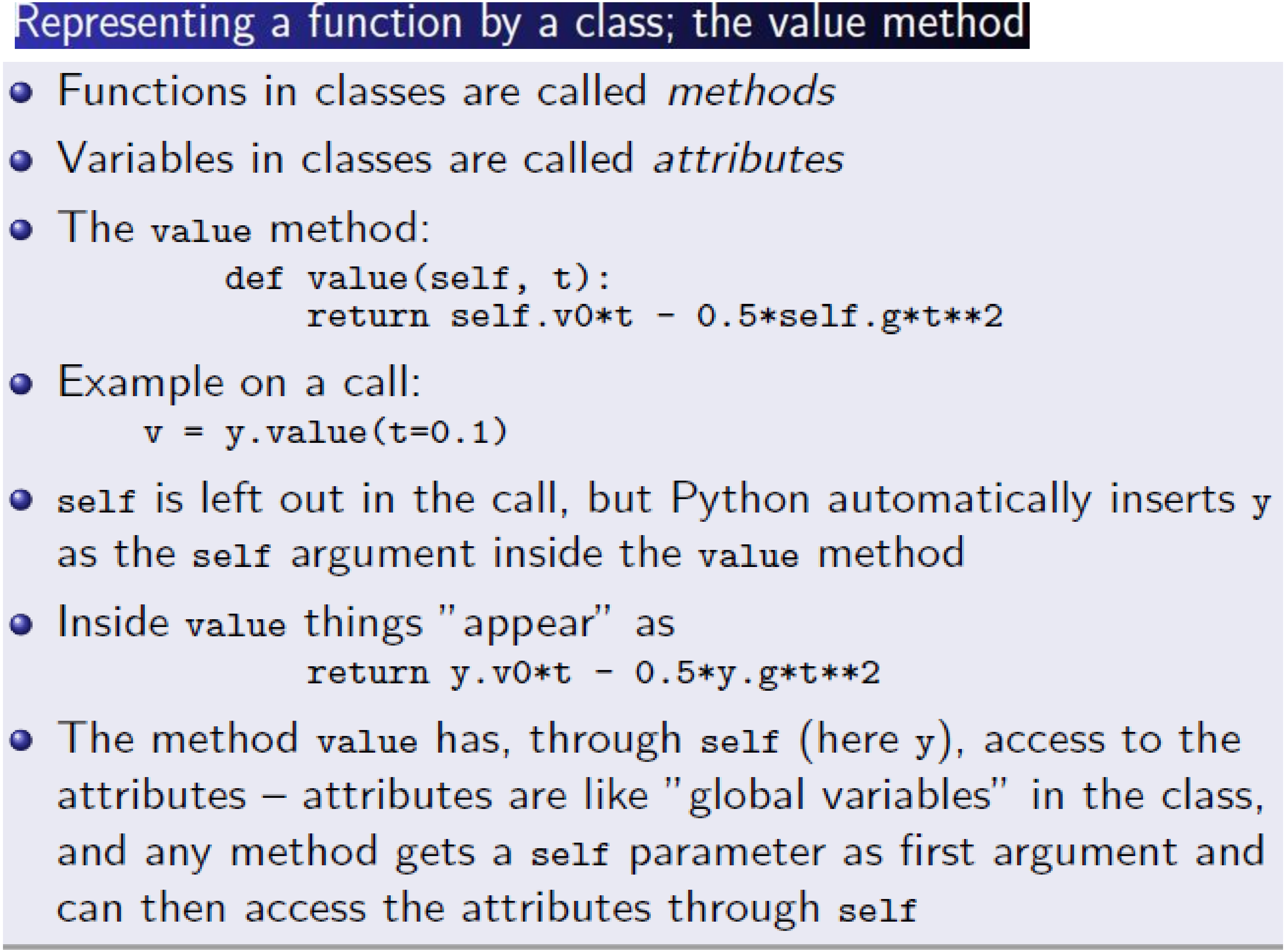
  ```
  y = Y(v0=3)
  ```

  we create a new variable (instance) `y` of type `Y`
- Y(3) is a call to the *constructor*:

  ```
  def __init__(self, v0):
      self.v0 = v0
      self.g = 9.81
  ```
- Think of `self` as `y`, i.e., the new variable to be created – `self.v0` means that we attach a variable `v0` to `self` (`y`)

  ```
  Y.__init__(y, 3)    # logic behind Y(3)
  ```
- `self` is always first parameter in a function, but never inserted in the call
- After `y = Y(3)`, `y` has two variables `v0` and `g`, and we can do

  ```
  print y.v0
  print y.g
  ```



Representing a function by a class; the value method

- Functions in classes are called *methods*
- Variables in classes are called *attributes*
- The `value` method:

```
        def value(self, t):
            return self.v0*t - 0.5*self.g*t**2
```

- Example on a call:

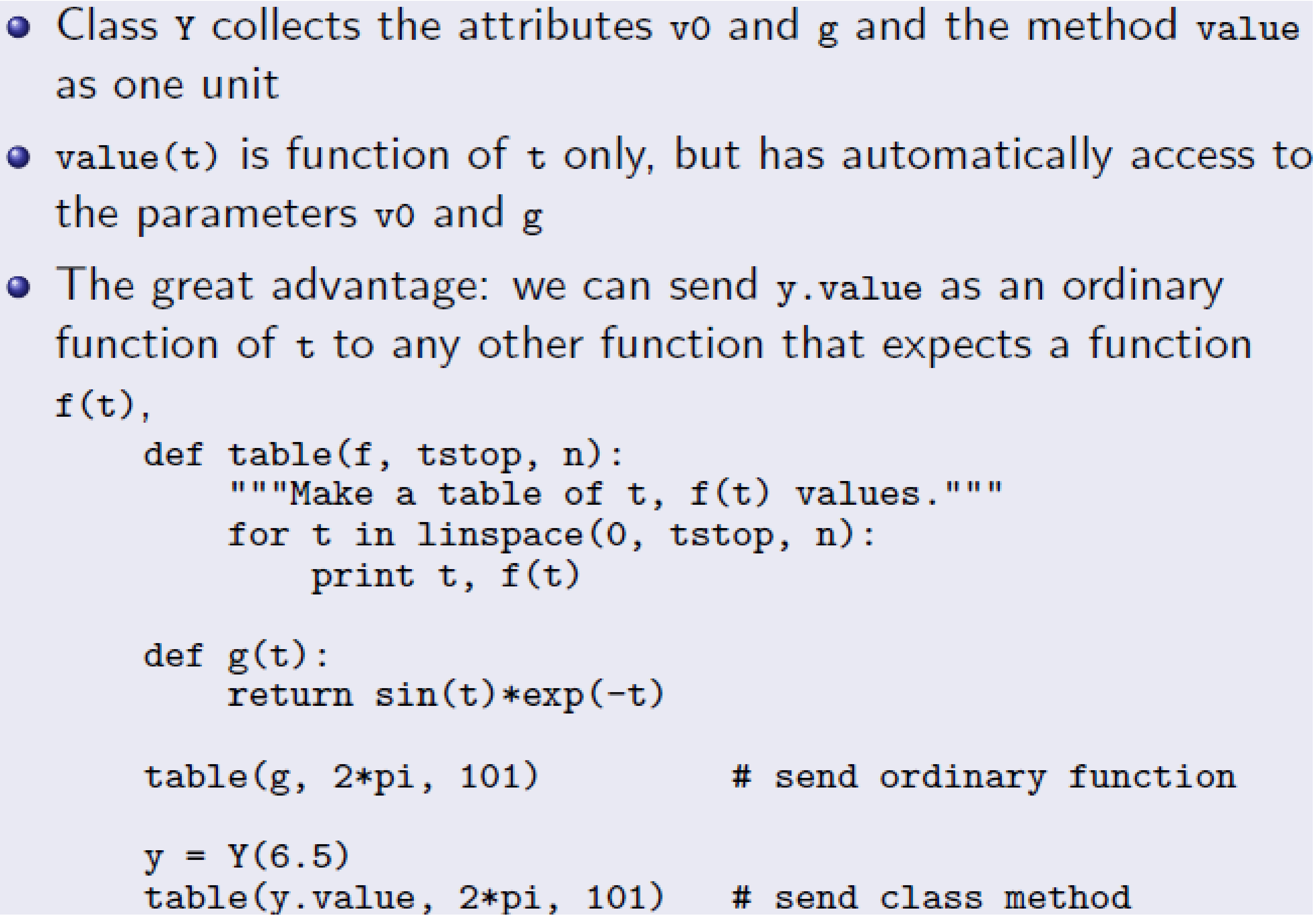
```
    v = y.value(t=0.1)
```

- `self` is left out in the call, but Python automatically inserts `y` as the `self` argument inside the `value` method
- Inside `value` things "appear" as

```
            return y.v0*t - 0.5*y.g*t**2
```

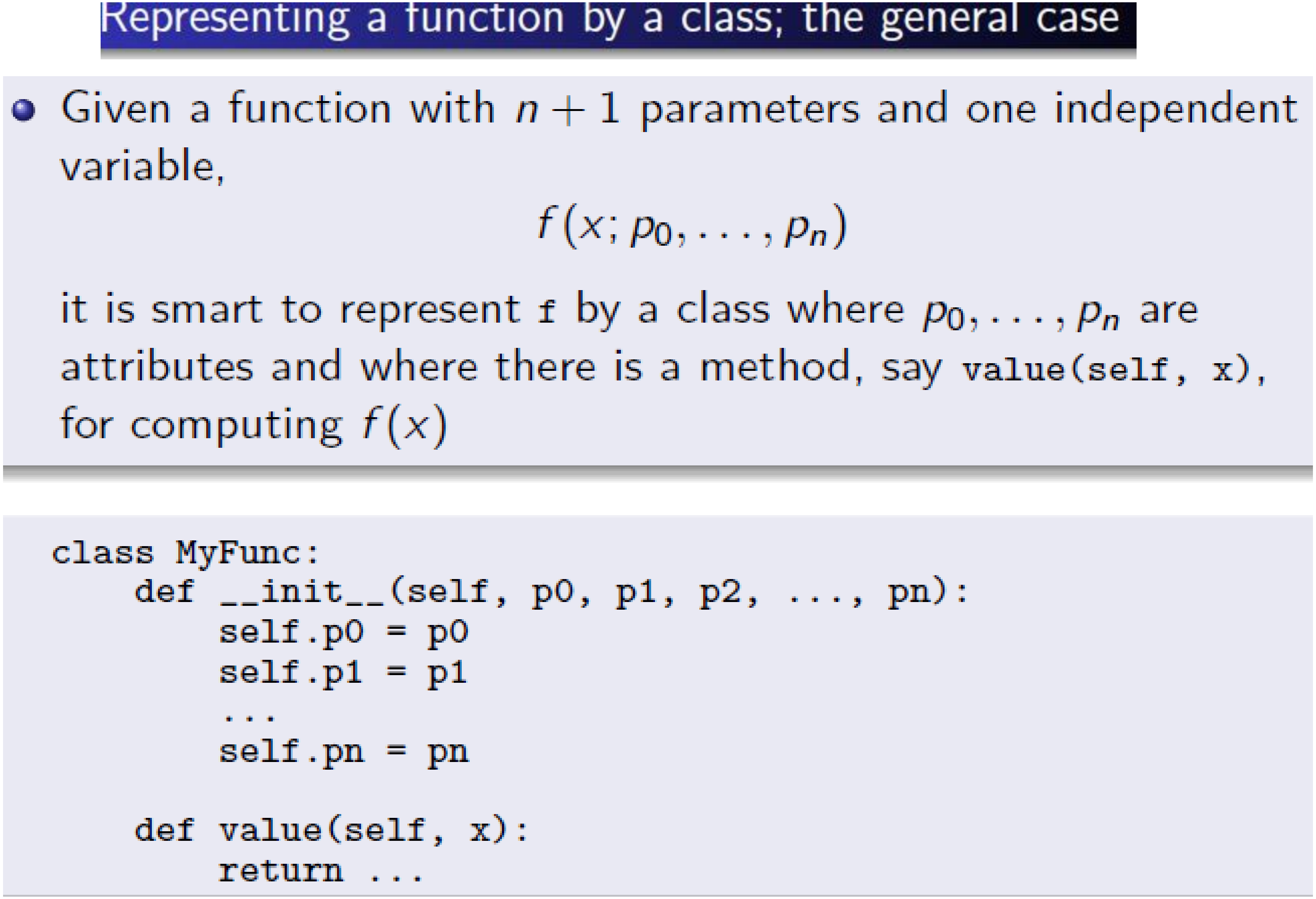
- The method `value` has, through `self` (here `y`), access to the attributes – attributes are like "global variables" in the class, and any method gets a `self` parameter as first argument and can then access the attributes through `self`

Representing a function by a class; summary



- Class `Y` collects the attributes `v0` and `g` and the method `value` as one unit
- `value(t)` is function of `t` only, but has automatically access to the parameters `v0` and `g`
- The great advantage: we can send `y.value` as an ordinary function of `t` to any other function that expects a function `f(t)`,

```
def table(f, tstop, n):
    """Make a table of t, f(t) values."""
    for t in linspace(0, tstop, n):
        print t, f(t)

def g(t):
    return sin(t)*exp(-t)

table(g, 2*pi, 101)         # send ordinary function

y = Y(6.5)
table(y.value, 2*pi, 101)   # send class method
```



Representing a function by a class; the general case

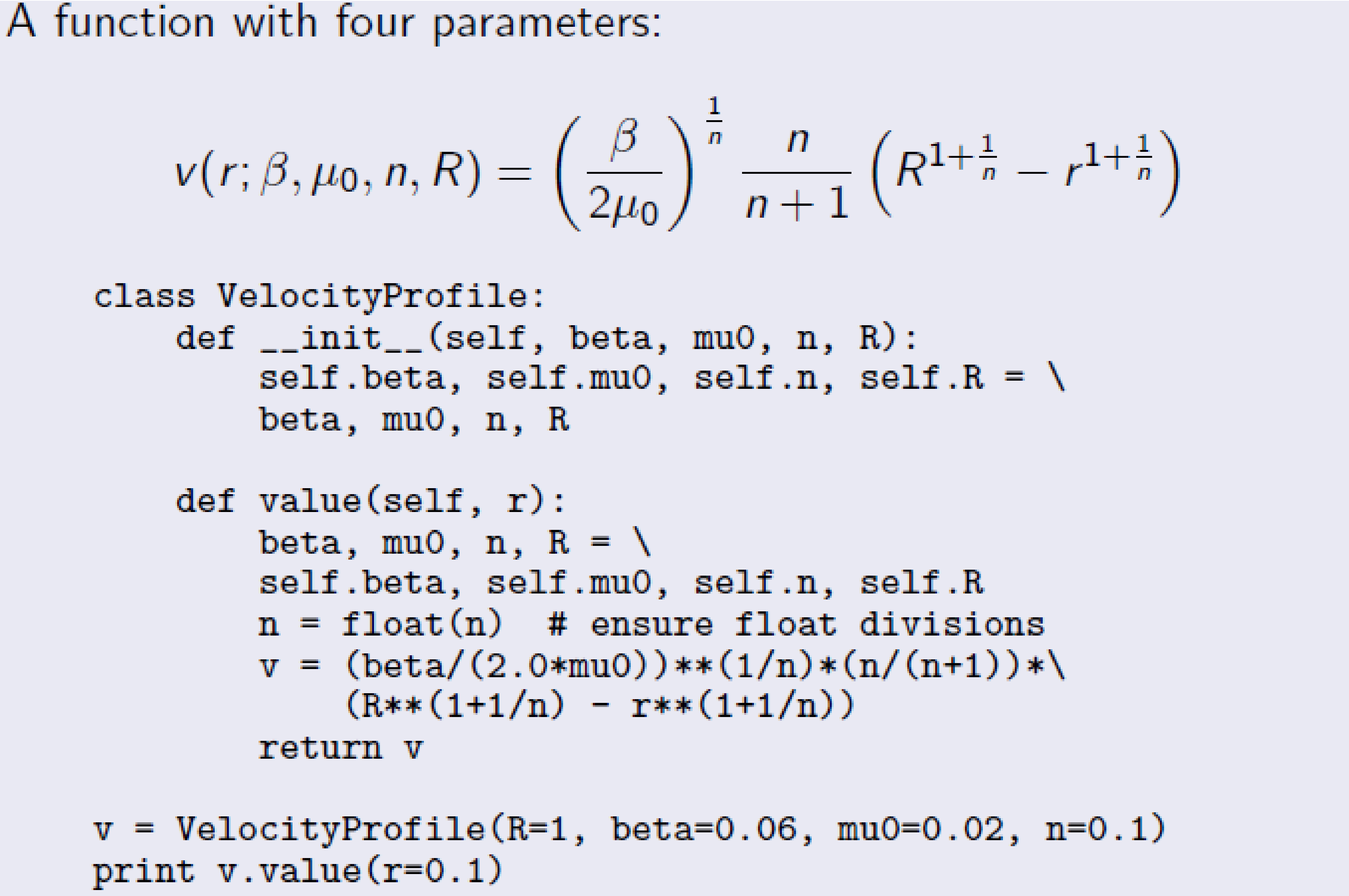
- Given a function with $n+1$ parameters and one independent variable,

$$f(x; p_0, \ldots, p_n)$$

it is smart to represent `f` by a class where $p_0, \ldots, p_n$ are attributes and where there is a method, say `value(self, x)`, for computing $f(x)$

```
class MyFunc:
    def __init__(self, p0, p1, p2, ..., pn):
        self.p0 = p0
        self.p1 = p1
        ...
        self.pn = pn

    def value(self, x):
        return ...
```

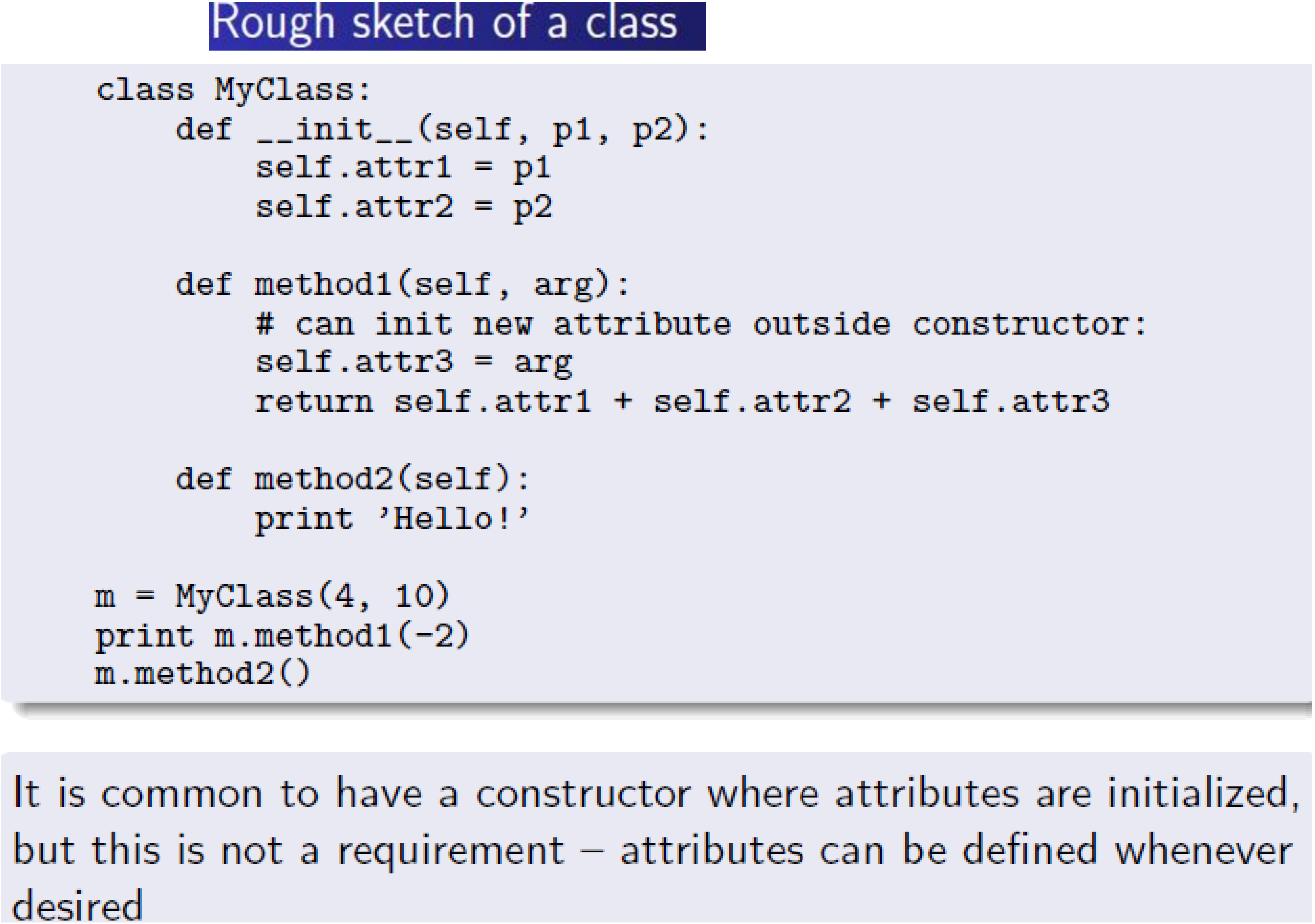
Representing a function by a class; another example



A function with four parameters:

$$v(r;\beta,\mu_0,n,R)=\left(\frac{\beta}{2\mu_0}\right)^{\frac{1}{n}}\frac{n}{n+1}\left(R^{1+\frac{1}{n}}-r^{1+\frac{1}{n}}\right)$$

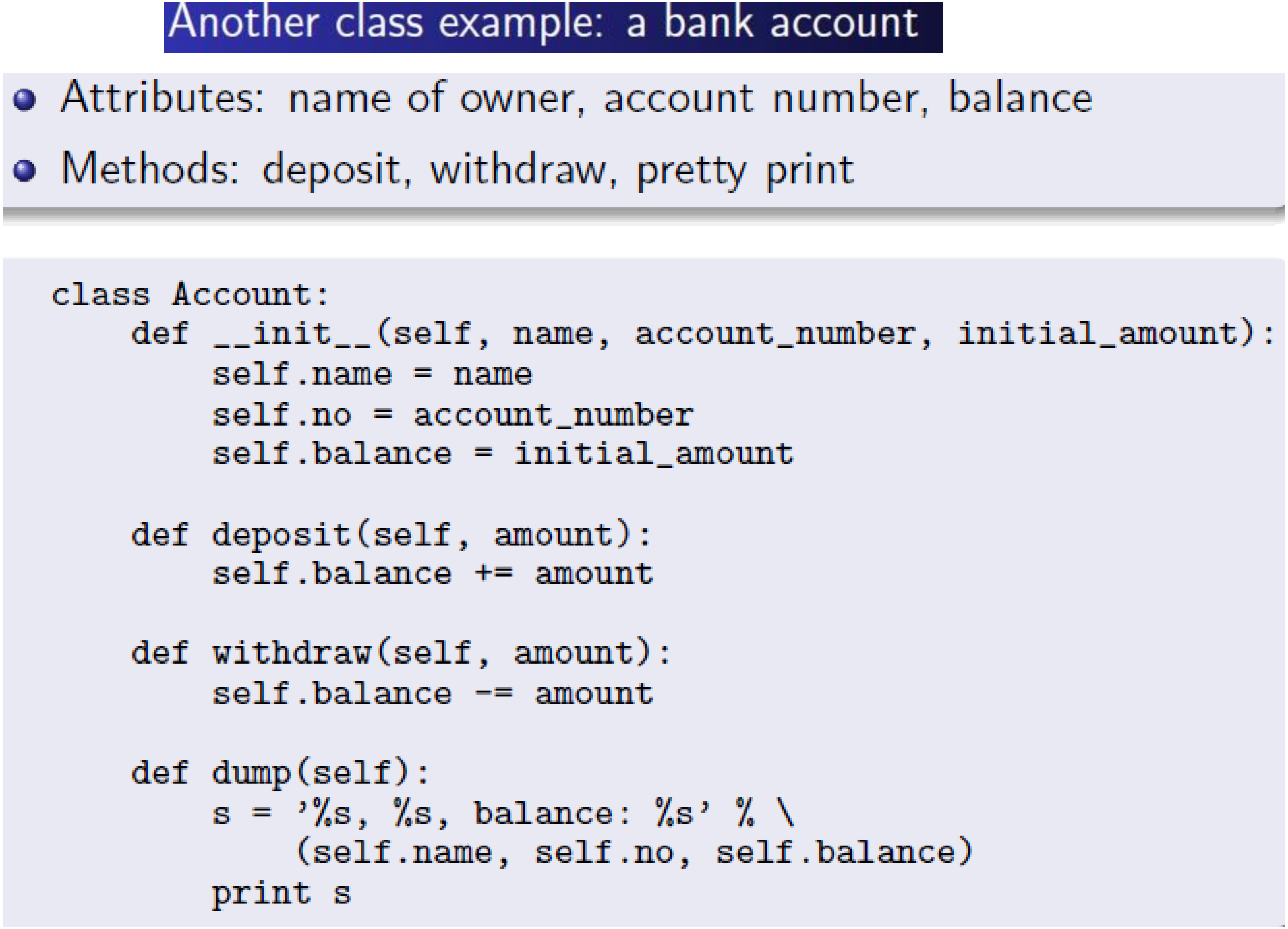
```
class VelocityProfile:
    def __init__(self, beta, mu0, n, R):
        self.beta, self.mu0, self.n, self.R = \
        beta, mu0, n, R

    def value(self, r):
        beta, mu0, n, R = \
        self.beta, self.mu0, self.n, self.R
        n = float(n)  # ensure float divisions
        v = (beta/(2.0*mu0))**(1/n)*(n/(n+1))*\
            (R**(1+1/n) - r**(1+1/n))
        return v

v = VelocityProfile(R=1, beta=0.06, mu0=0.02, n=0.1)
print v.value(r=0.1)
```



Rough sketch of a class

```
class MyClass:
    def __init__(self, p1, p2):
        self.attr1 = p1
        self.attr2 = p2

    def method1(self, arg):
        # can init new attribute outside constructor:
        self.attr3 = arg
        return self.attr1 + self.attr2 + self.attr3

    def method2(self):
        print 'Hello!'

m = MyClass(4, 10)
print m.method1(-2)
m.method2()
```

It is common to have a constructor where attributes are initialized, but this is not a requirement – attributes can be defined whenever desired

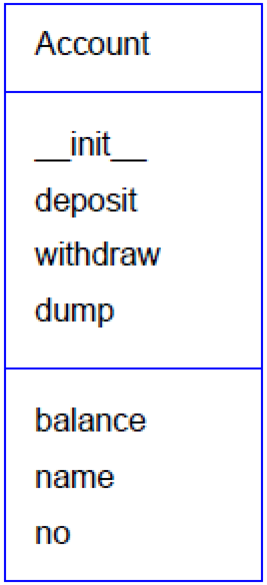


Another class example: a bank account

- Attributes: name of owner, account number, balance
- Methods: deposit, withdraw, pretty print

```
class Account:
    def __init__(self, name, account_number, initial_amount):
        self.name = name
        self.no = account_number
        self.balance = initial_amount

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        self.balance -= amount

    def dump(self):
        s = ’%s, %s, balance: %s’ % \
            (self.name, self.no, self.balance)
        print s
```

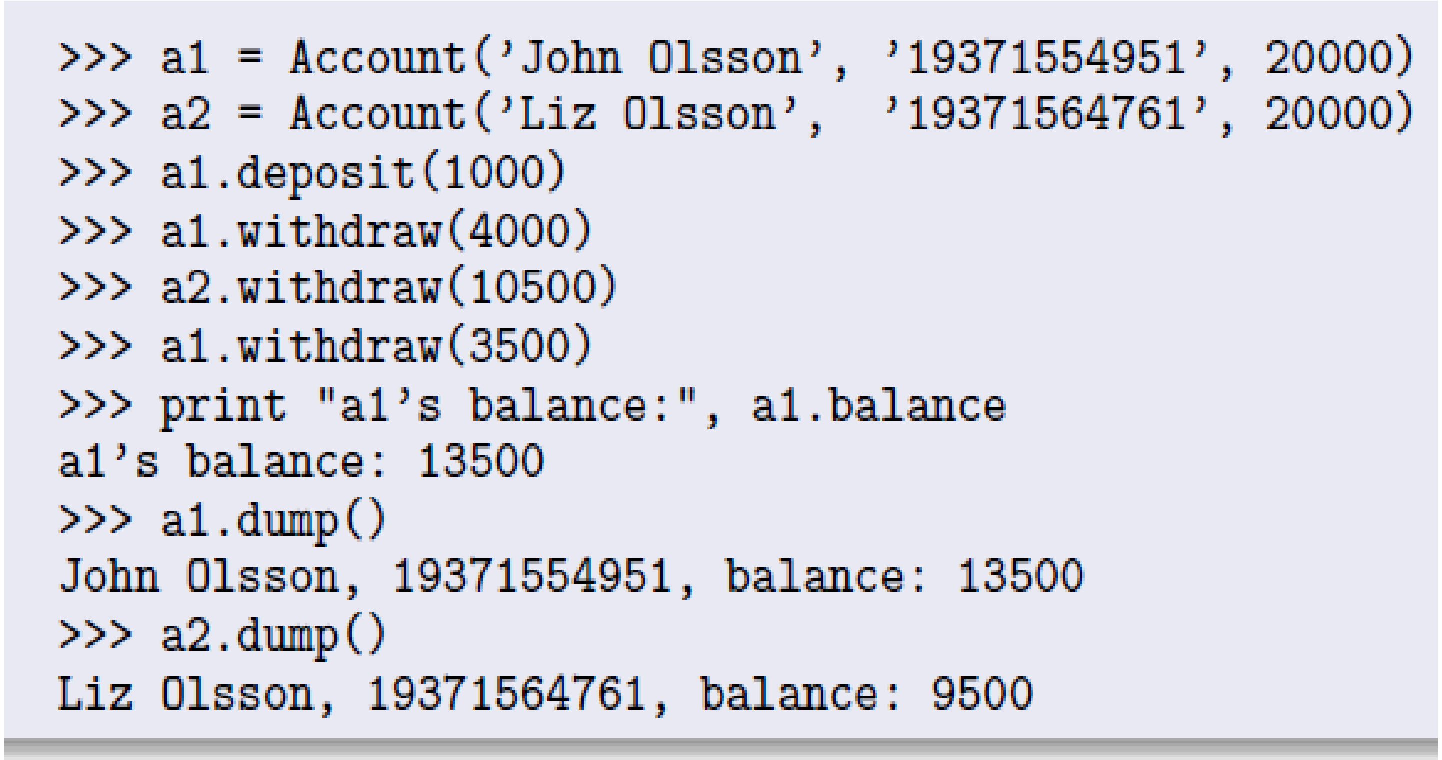
UML diagram of class Account


Account

__init__
deposit
withdraw
dump

balance
name
no

Example on using class Account



```
>>> a1 = Account('John Olsson', '19371554951', 20000)
>>> a2 = Account('Liz Olsson',  '19371564761', 20000)
>>> a1.deposit(1000)
>>> a1.withdraw(4000)
>>> a2.withdraw(10500)
>>> a1.withdraw(3500)
>>> print "a1's balance:", a1.balance
a1's balance: 13500
>>> a1.dump()
John Olsson, 19371554951, balance: 13500
>>> a2.dump()
Liz Olsson, 19371564761, balance: 9500
```

Protected names for avoiding misuse

Possible, but not intended:

```
>>> a1.name = 'Some other name'
>>> a1.balance = 100000
>>> a1.no = '19371564768'
```

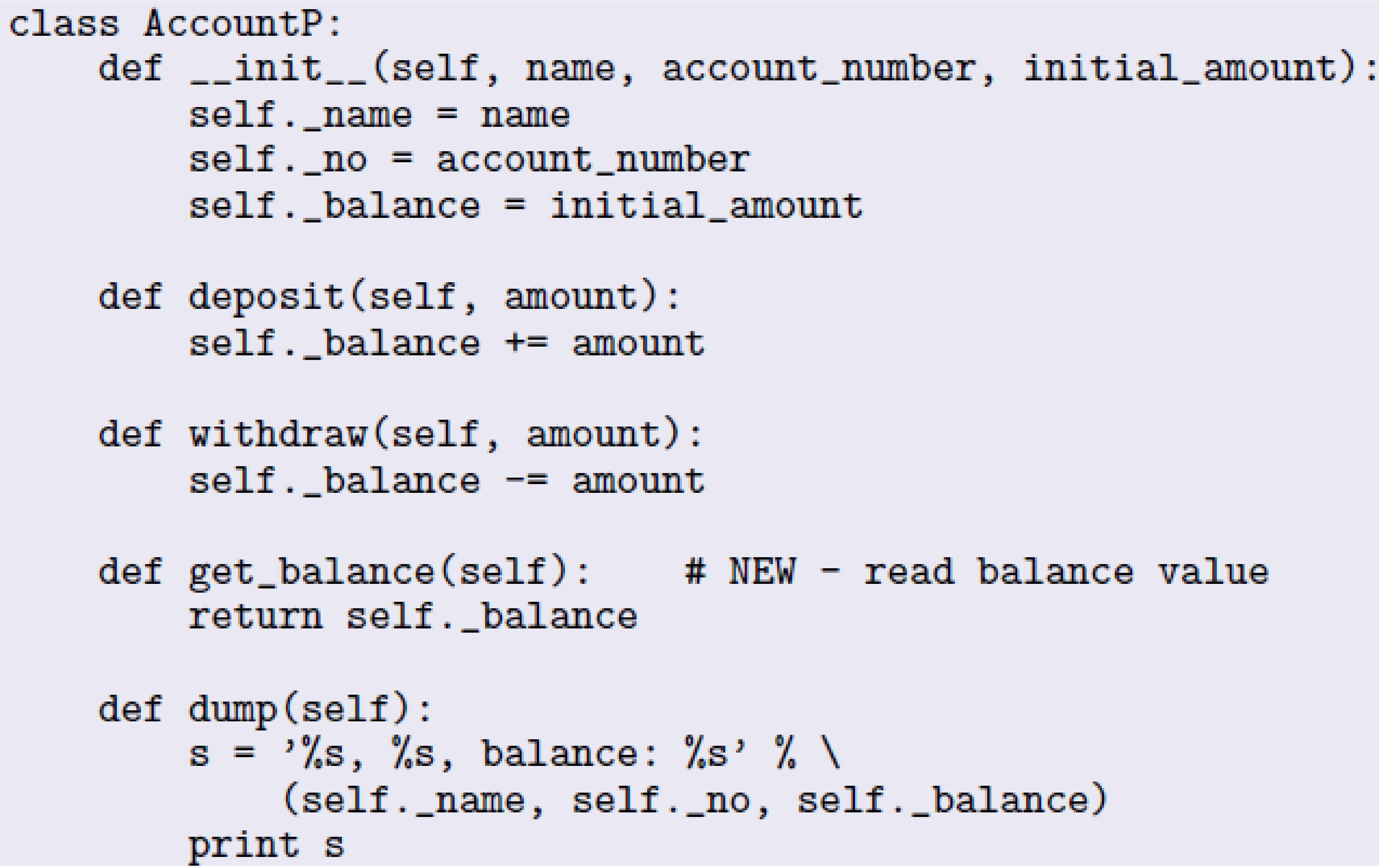
The assumptions on correct usage:

- The attributes should *not* be changed!
- The `balance` attribute can be viewed
- Changing `balance` is done through `withdraw` or `deposit`

Remedy:

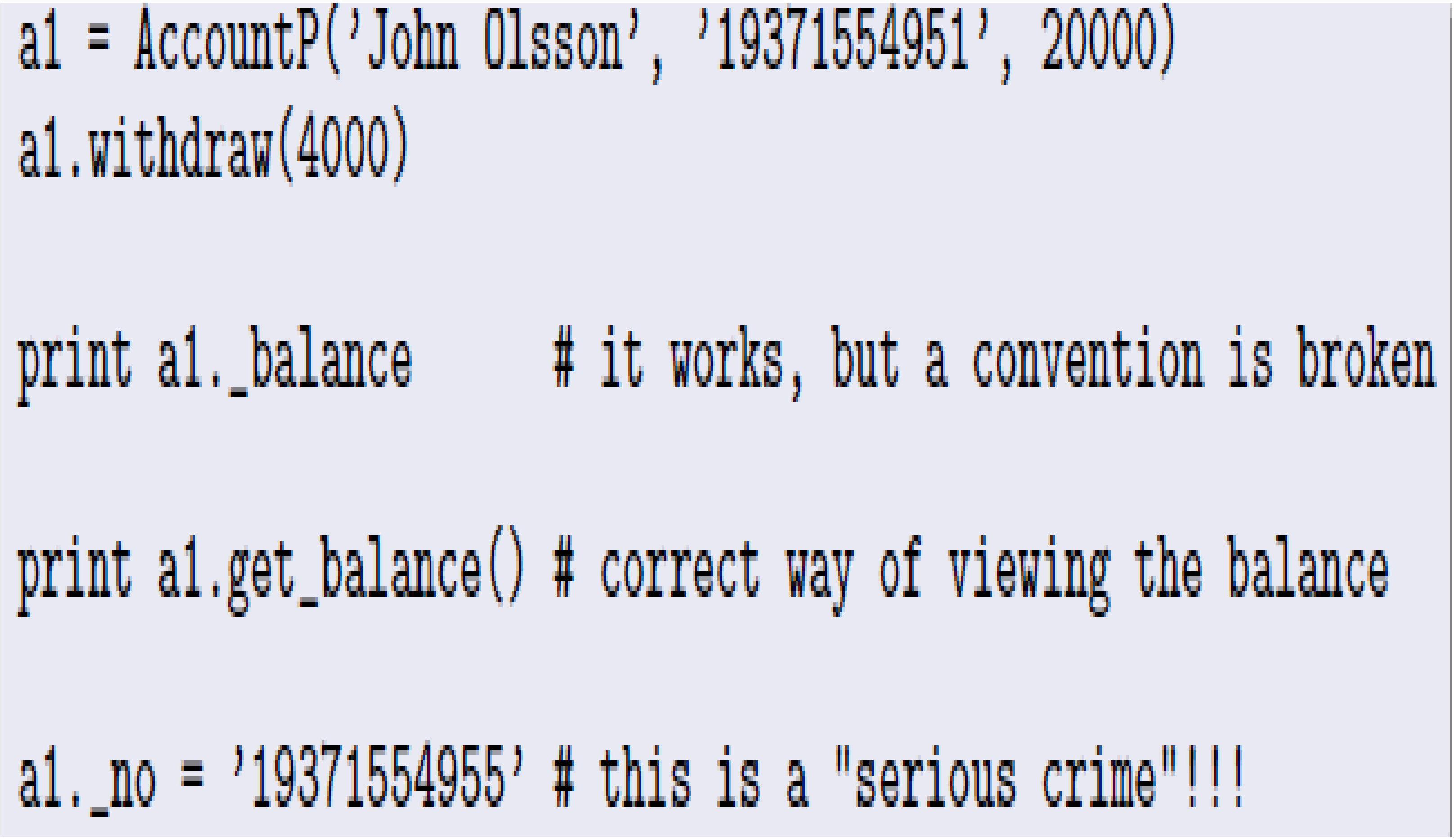
Attributes and methods not intended for use outside the class can be marked as *protected* by prefixing the name with an underscore (e.g., `_name`). This is just a convention – and no technical way of avoiding attributes and methods to be accessed.

Improved class with attribute protection (underscore)

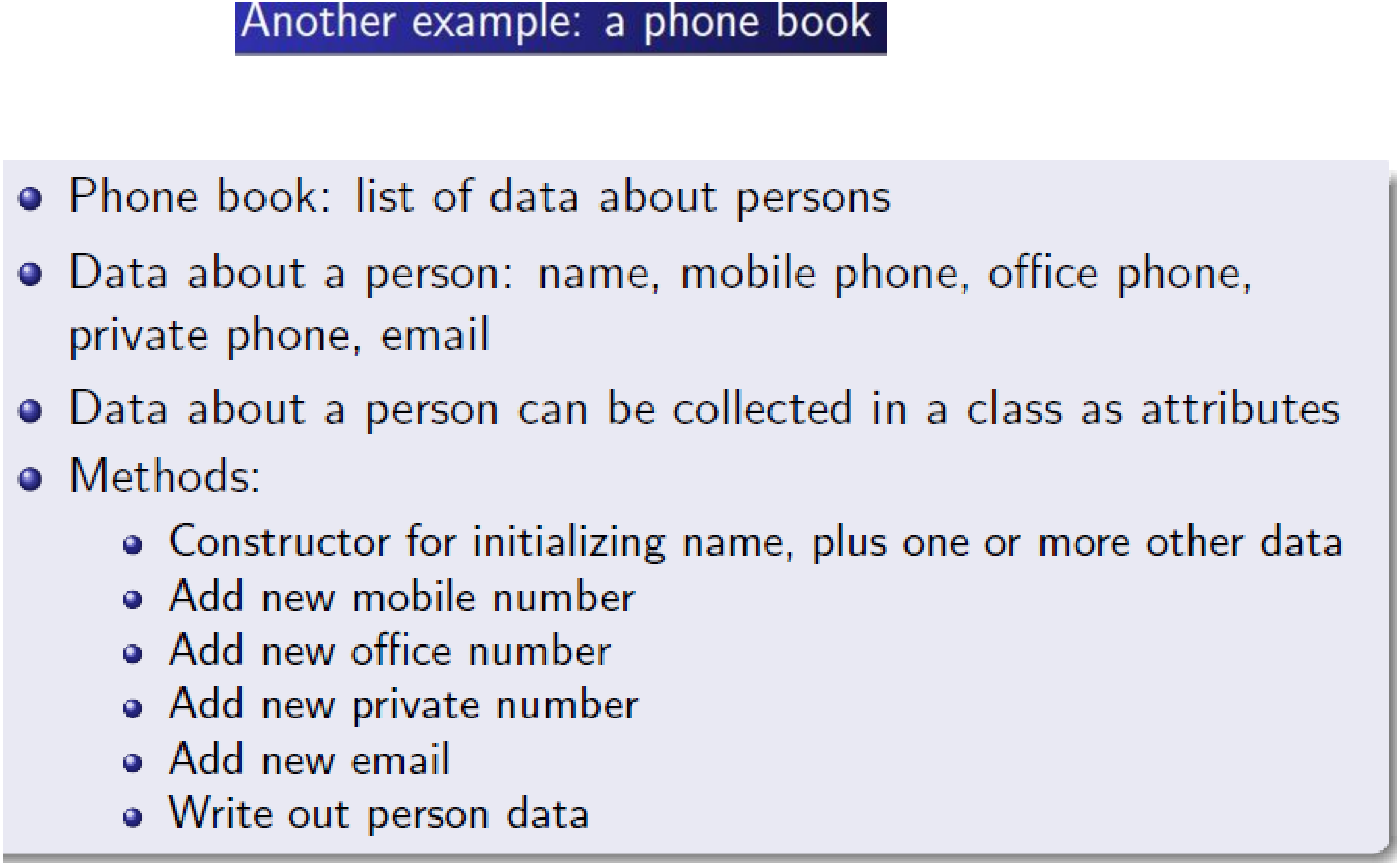


```
class AccountP:
    def __init__(self, name, account_number, initial_amount):
        self._name = name
        self._no = account_number
        self._balance = initial_amount

    def deposit(self, amount):
        self._balance += amount

    def withdraw(self, amount):
        self._balance -= amount

    def get_balance(self):    # NEW - read balance value
        return self._balance

    def dump(self):
        s = ’%s, %s, balance: %s’ % \
            (self._name, self._no, self._balance)
        print s
```

Usage of improved class AccountP



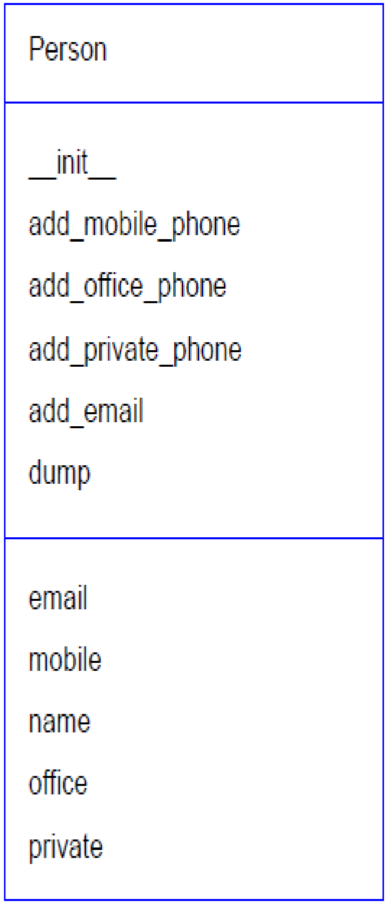
```
a1 = AccountP('John Olsson', '1937155495l', 20000)
a1.withdraw(4000)

print a1._balance      # it works, but a convention is broken

print a1.get_balance() # correct way of viewing the balance

a1._no = '1937154955' # this is a "serious crime"!!!
```

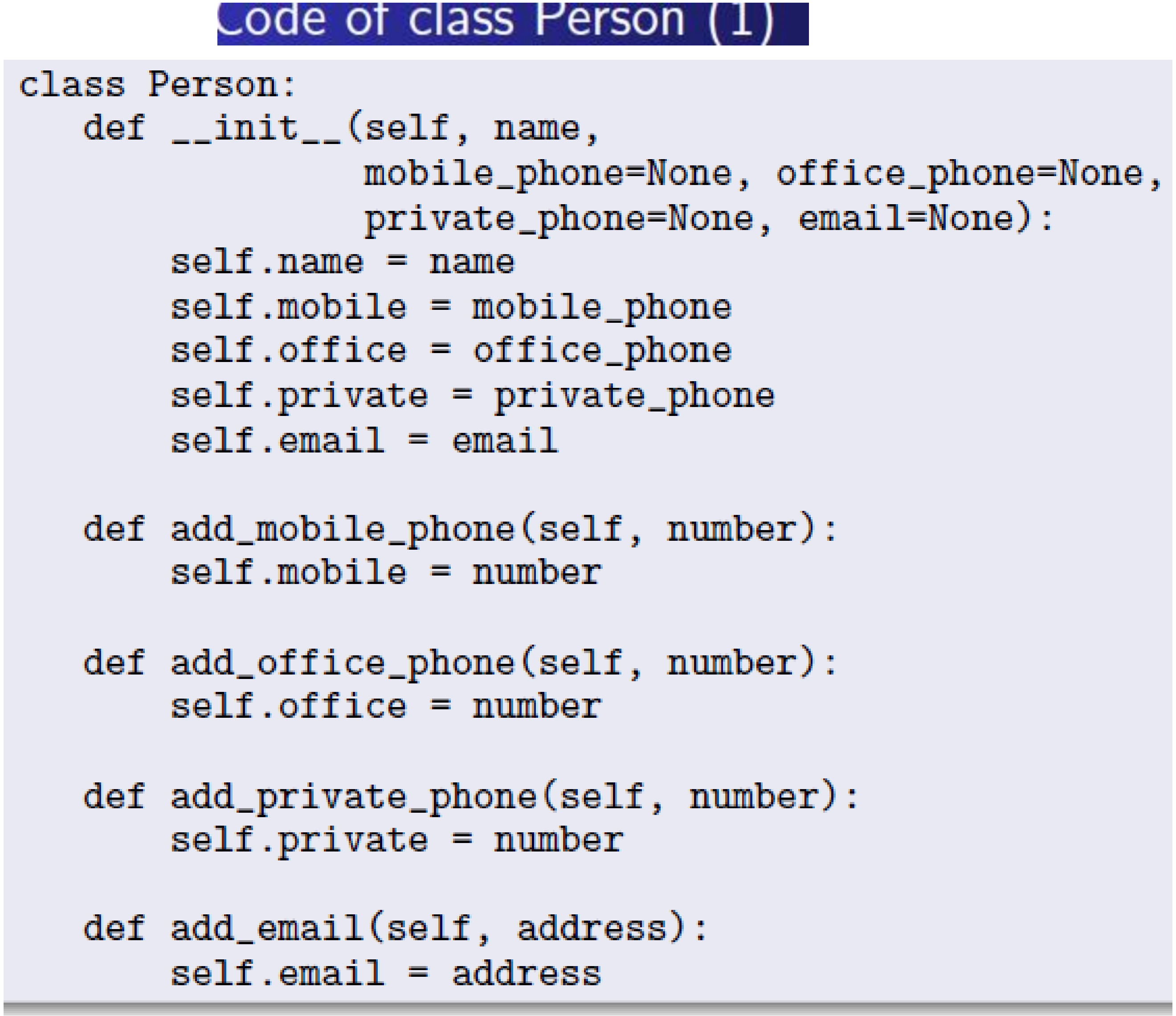


Another example: a phone book

- Phone book: list of data about persons
- Data about a person: name, mobile phone, office phone, private phone, email
- Data about a person can be collected in a class as attributes
- Methods:
  - Constructor for initializing name, plus one or more other data
  - Add new mobile number
  - Add new office number
  - Add new private number
  - Add new email
  - Write out person data

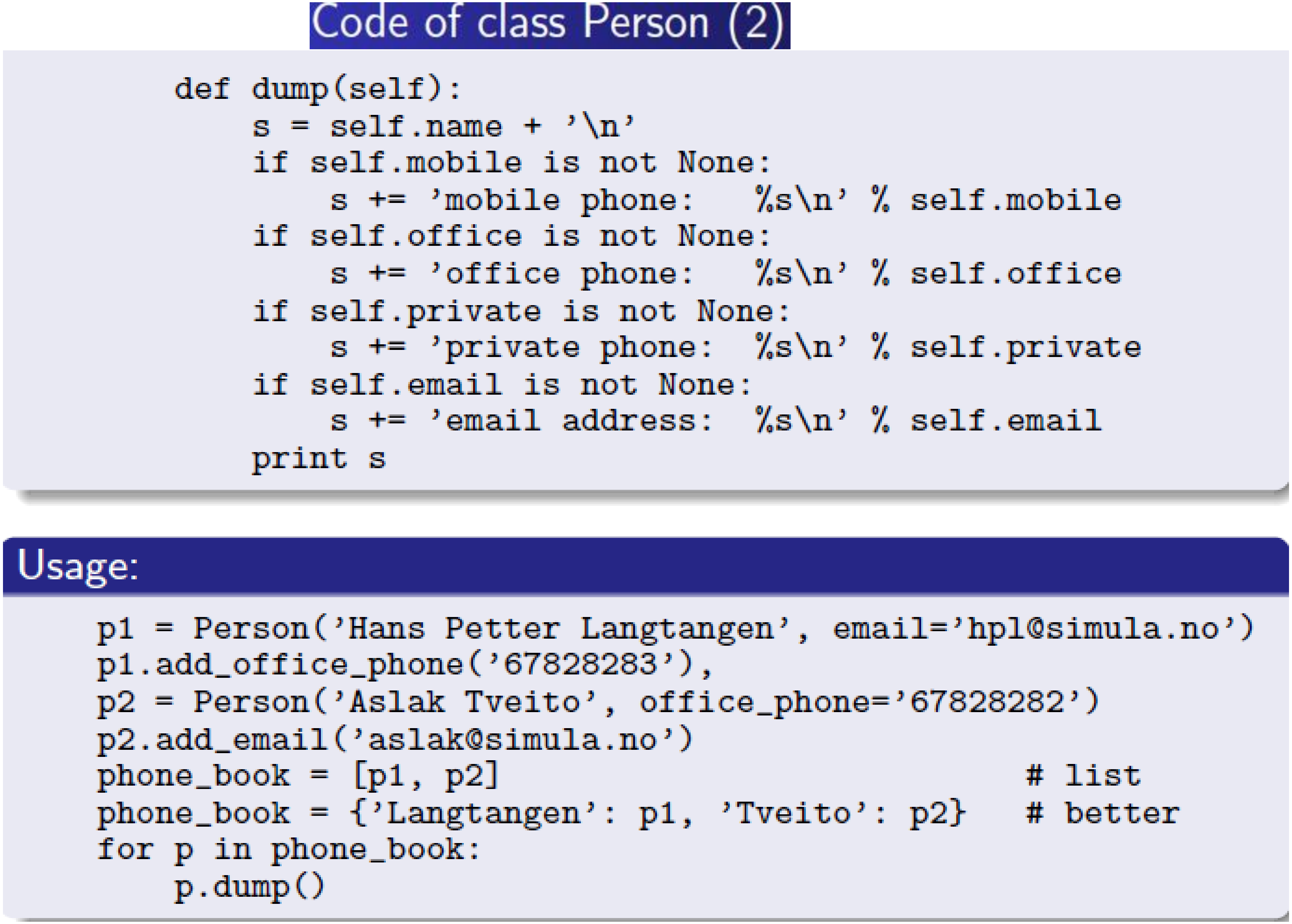
UML diagram of class Person





Code of class Person (1)

```
class Person:
   def __init__(self, name,
                mobile_phone=None, office_phone=None,
                private_phone=None, email=None):
       self.name = name
       self.mobile = mobile_phone
       self.office = office_phone
       self.private = private_phone
       self.email = email

   def add_mobile_phone(self, number):
       self.mobile = number

   def add_office_phone(self, number):
       self.office = number

   def add_private_phone(self, number):
       self.private = number

   def add_email(self, address):
       self.email = address
```



Code of class Person (2)

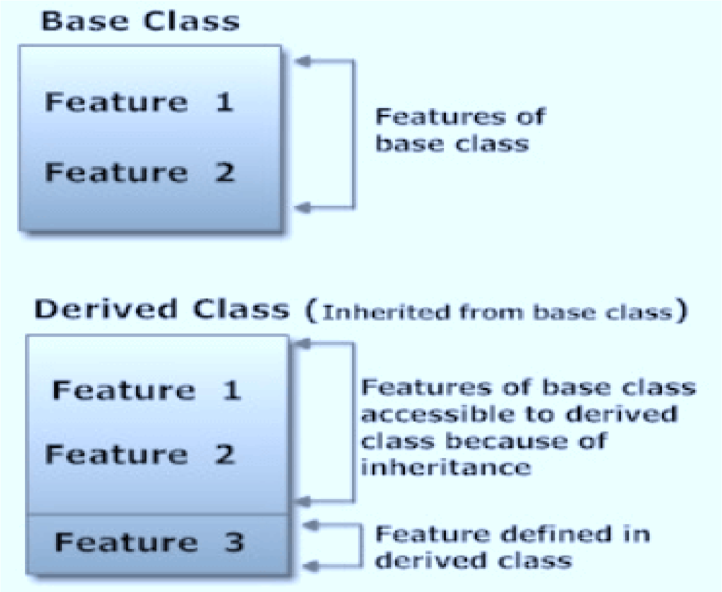
```
    def dump(self):
        s = self.name + ’\n’
        if self.mobile is not None:
            s += ’mobile phone:   %s\n’ % self.mobile
        if self.office is not None:
            s += ’office phone:   %s\n’ % self.office
        if self.private is not None:
            s += ’private phone:  %s\n’ % self.private
        if self.email is not None:
            s += ’email address:  %s\n’ % self.email
        print s
```

Usage:

```
p1 = Person(’Hans Petter Langtangen’, email=’hpl@simula.no’)
p1.add_office_phone(’67828283’),
p2 = Person(’Aslak Tveito’, office_phone=’67828282’)
p2.add_email(’aslak@simula.no’)
phone_book = [p1, p2]                           # list
phone_book = {’Langtangen’: p1, ’Tveito’: p2}   # better
for p in phone_book:
    p.dump()
```

Inheritance in Python

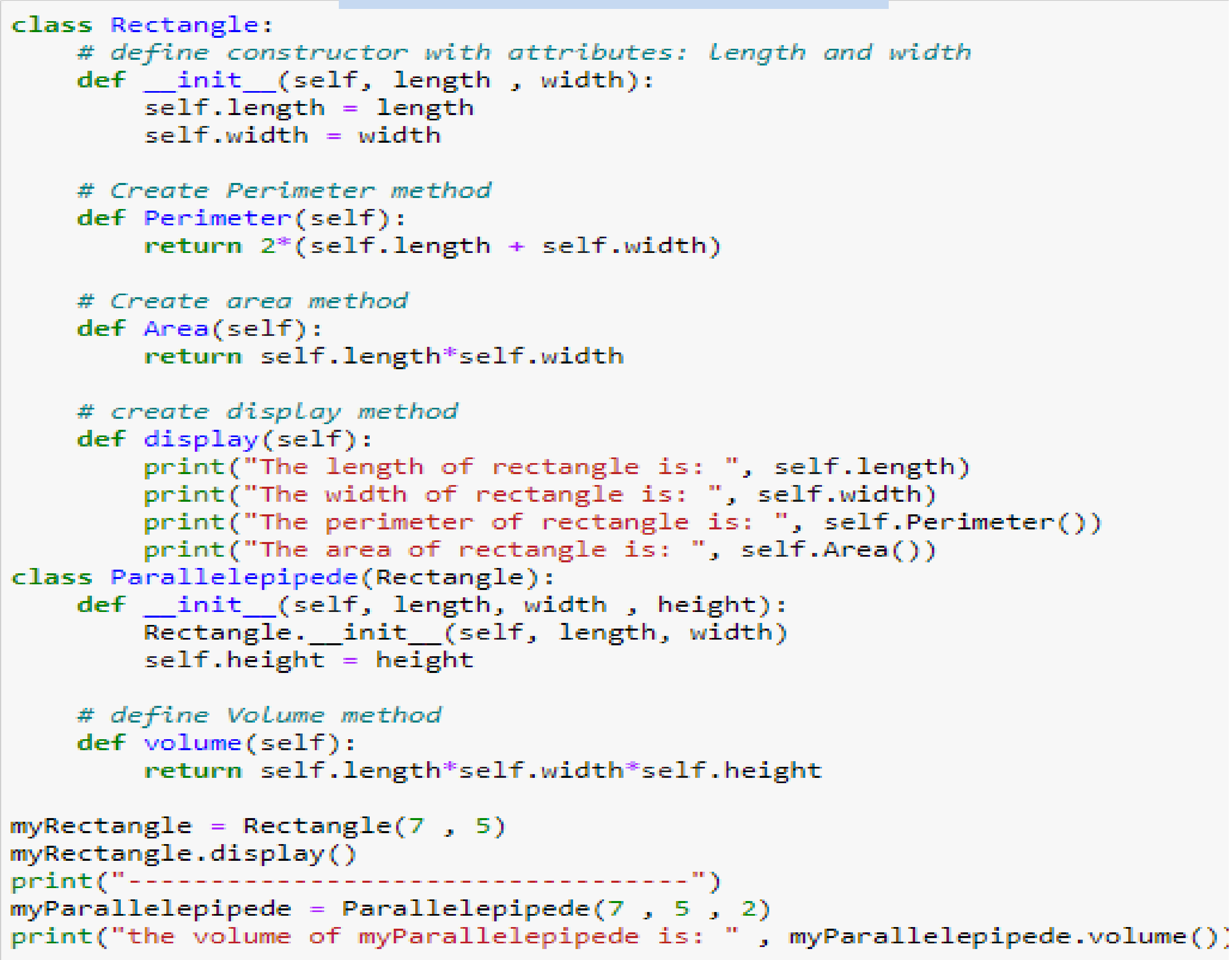
inheritance is a feature used in object-oriented programming; it refers to defining a new class with less or no modification to an existing class. The new class is called **derived class** and from one which it inherits is called the **base**. Python supports inheritance;


Base Class
Feature 1
Feature 2
Features of base class
Derived Class (Inherited from base class)
Feature 1
Feature 2
Feature 3
Features of base class accessible to derived class because of inheritance
Feature defined in derived class

inheritance are:

1.It represents real-world relationships well.
2.It provides **reusability** of a code. We don’t have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
3.It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Inheritance in Python-example



```python
class Rectangle:
    # define constructor with attributes: length and width
    def __init__(self, length , width):
        self.length = length
        self.width = width

    # Create Perimeter method
    def Perimeter(self):
        return 2*(self.length + self.width)

    # Create area method
    def Area(self):
        return self.length*self.width

    # create display method
    def display(self):
        print("The length of rectangle is: ", self.length)
        print("The width of rectangle is: ", self.width)
        print("The perimeter of rectangle is: ", self.Perimeter())
        print("The area of rectangle is: ", self.Area())
class Parallelepipede(Rectangle):
    def __init__(self, length, width , height):
        Rectangle.__init__(self, length, width)
        self.height = height

    # define Volume method
    def volume(self):
        return self.length*self.width*self.height

myRectangle = Rectangle(7 , 5)
myRectangle.display()
print("-----------------------------------")
myParallelepipede = Parallelepipede(7 , 5 , 2)
print("the volume of myParallelepipede is: " , myParallelepipede.volume()
```