Dictionaries and strings (part 2)

Ole Christian Lingjærde, Dept of Informatics, UiO

20 October 2017

Today's agenda

- Quiz
- Exercise 6.7
- String manipulation

Quiz 1

Question A

```
d = {-2:-1, -1:0, 0:1, 1:2, 2:-2}
print(d[0])
# What is printed out?
```

Question B

```
d = {-2:-1, -1:0, 0:1, 1:2, 2:-2}
print(d[d[0]])
# What is printed out?
```

Question C

```
d = {-2:-1, -1:0, 0:1, 1:2, 2:-2}
print(d[-2]*d[2])
# What is printed out?
```

Quiz 2

Question A

```
table = {'age':[35,20], 'name':['Anna','Peter']}
for key in table:
    print('%s: %s' % (key,table[key]))
# What is printed out?
```

Question B

```
table = {'age':[35,20], 'name':['Anna','Peter']}
vals = list(table.values())
print(vals)
print(vals[0])
print(vals[0][0])
# What is printed out?
```

Question C

```
table = {'age':[35,20], 'name':['Anna','Peter']}
print(table['name'][1], table['age'][1])
# What is printed out?
```

Quiz 3

Question A

```
d = {3:5, 6:7}
e = {4:6, 7:8}
d.update(e)
# What is the content of dictionary d now?
```

Question B

```
d = {3:5, 6:7}
e = {4:6, 7:8}
d.update(e)
d.update(e)
# What is the content of dictionary d now?
```

Question C

```
d = {6:100}
e = {6:6, 7:8}
d.update(e)
# What is the content of dictionary d now?
```

Quiz 4

The file 'teledata.txt' gives information about mobile customers:

```
Age     Income     Gender     Monthly calls    ID
45      720k       Female     46               A001
27      440k       Male       3                A002
17      0          Male       52               A006
24      60k        Female     18               A014
...     ...        ...        ...              ...
```

- How could you store the data using five lists?
- How could you store the data using one list?
- How could you store the data in a dictionary (what information would be key and what datatype would you use for the values)?

Exercise 6.7

Make a nested dictionary from a file

The file `human_evolution.txt` holds information about various human species and their height, weight, and brain volume. Make a program that reads this file and stores the tabular data in a nested dictionary `humans`. The keys in `humans` correspond to the species name (e.g., H. erectus), and the values are dictionaries with keys 'period', 'height', 'weight', 'volume'. For example,

humans['H. habilis']['weight']

should equal '55 - 70'. Let the program print to screen the `humans` dictionary in a nice tabular form similar to that in the file.

Filename: humans

Step 1: reading the file

We first download the file and inspect it visually:

```
Species                Lived when      Adult           Adult        Brain volume
                       (mill. yrs)     height (m)      mass (kg)    (cm**3)
-------------------------------------------------------------------------------
H. habilis             2.2 - 1.6       1.0 - 1.5       33 - 55      660
H. erectus             1.4 - 0.2       1.8             60           850 (early) - 1100 (late)
H. ergaster            1.9 - 1.4       1.9                          700 - 850
H. heidelbergensis     0.6 - 0.35      1.8             60           1100 - 1400
H. neanderthalensis    0.35 - 0.03     1.6             55 - 70      1200 - 1700
H. sapiens sapiens     0.2 - present   1.4 - 1.9       50 - 100     1000 - 1850
H. floresiensis        0.10 - 0.012    1.0             25           400
-------------------------------------------------------------------------------

Source: http://en.wikipedia.org/wiki/Human_evolution
```

To read the table, we need to skip some lines at the top and bottom. How do we determine where the data start and stop?

- Solution 1: we see that the data span lines 4-10.
- Solution 2: data lines always start with 'H. '.
- Solution 3: data occur between the lines with hyphens.

All would work, but here we go for the third solution.

How to do it in Python

```
# Read all lines into a list
infile = open('human_evolution.txt', 'r')
lines = infile.readlines()

# Find first line with data
k = 0
while lines[k][0] != '-':   # When no hyphen
    k = k + 1               # ... we continue the search
first = k + 1               # First line after hyphen

# Find last line with data
k = first                   # Start point for search
while lines[k][0] != '-':   # When no hyphen
    k = k + 1               # ... we continue the search
last = k - 1                # Last line before hyphen

# Now we are ready to process the data
for i in range(first, last+1):
    # Do something with lines[i]
```

Step 2: splitting a line into columns

```
Species              Lived when      Adult        Adult        Brain volume
                     (mill. yrs)     height (m)   mass (kg)    (cm**3)
-------------------------------------------------------------------------------
H. habilis           2.2 - 1.6       1.0 - 1.5    33 - 55      660
H. erectus           1.4 - 0.2       1.8          60           850 (early) - 1100 (late)
H. ergaster          1.9 - 1.4       1.9                       700 - 850
H. heidelbergensis   0.6 - 0.35      1.8          60           1100 - 1400
H. neanderthalensis  0.35 - 0.03     1.6          55 - 70      1200 - 1700
H. sapiens sapiens   0.2 - present   1.4 - 1.9    50 - 100     1000 - 1850
H. floresiensis      0.10 - 0.012    1.0          25           400
-------------------------------------------------------------------------------

Source: http://en.wikipedia.org/wiki/Human_evolution
```

Want to split each data line into columns, for example:

```
words[0] : 'H. habilis'
words[1] : '2.2 - 1.6'
words[2] : '1.0 - 1.5'
...
```

Possible solutions:

- Split on whitespace - but how to go from there?
- Find position of each column from the header

Here we go for the second solution.

How to do it in Python

```python
# Read all lines into a list
infile = open('human_evolution.txt', 'r')
lines = infile.readlines()

# Find column positions from second line in file
s = lines[1]
start = [0, s.index('(mill. yrs)'),
               s.index('height (m)'),
               s.index('mass (kg)'),
               s.index('(cm**3)')]
stop = start[1:len(start)] + [80]

# start: [ 0, 21, 37, 50, 62]
# stop:  [21, 37, 50, 62, 80]

# The k'th column in the i'th line is now easy to find:
# words[0] = lines[i][start[0]:stop[0]]
# words[1] = lines[i][start[1]:stop[1]]
# ...etc
```

Putting step 1 and 2 together

```
infile = open('human_evolution.txt', 'r')
lines = infile.readlines()

s = lines[1]
start = [0, s.index('(mill. yrs)'), s.index('height (m)'), ...]
stop = start[1:len(start)] + [80]

k = 0
while lines[k][0] != '-':
    k = k + 1
first = k + 1
k = first
while lines[k][0] != '-':
    k = k + 1
last = k - 1

humans = {}
for i in range(first, last+1):
    species = lines[i][start[0]:stop[0]]
    period = lines[i][start[1]:stop[1]]
    height = lines[i][start[2]:stop[2]]
    weight = lines[i][start[3]:stop[3]]
    volume = lines[i][start[4]:stop[4]]
    # Store the data in a dictionary
```

Step 3: storing the data

Consider the last step in the algorithm above:

```
for i in range(first, last+1):
    species = lines[i][start[0]:stop[0]].strip()
    period = lines[i][start[1]:stop[1]].strip()
    height = lines[i][start[2]:stop[2]].strip()
    weight = lines[i][start[3]:stop[3]].strip()
    volume = lines[i][start[4]:stop[4]].strip()
    # Store the data in a dictionary
```

The variables represent one line of data from the file. We want to store it in the dictionary `humans` as one (key,value) pair.

We want the key to be `species` and the value to be another dictionary. We can achieve this as follows:

```
humans[species] = {'period': period, 'height': height,
          'weight': weight, 'volume': volume}
```

Putting step 1, 2 and 3 together

```
infile = open('human_evolution.txt', 'r')
lines = infile.readlines()

s = lines[1]
start = [0, s.index('(mill. yrs)'), s.index('height (m)'), ...]
stop = start[1:len(start)] + [80]

k = 0
while lines[k][0] != '-':
    k = k + 1
first = k + 1
k = first
while lines[k][0] != '-':
    k = k + 1
last = k - 1

for i in range(first, last+1):
    species = lines[i][start[0]:stop[0]].strip()
    period = lines[i][start[1]:stop[1]].strip()
    height = lines[i][start[2]:stop[2]].strip()
    weight = lines[i][start[3]:stop[3]].strip()
    volume = lines[i][start[4]:stop[4]].strip()
    humans[species] = {'period': period, 'height': height,
         'weight': weight, 'volume': volume}
```

Step 4: printing table on screen

```
# Print a title
s = '%-23s %-13s %-13s %-13s %-25s' % \
        ('species', 'period', 'height', 'weight', 'volume')
print(s)

# Print table contents
for sp in humans:
    d = humans[sp]
    period = d['period']
    height = d['height']
    weight = d['weight']
    volume = d['volume']
    s = '%-23s %-13s %-13s %-13s %-25s' % \
        (sp, period, height, weight, volume)
    print(s)
```

Result

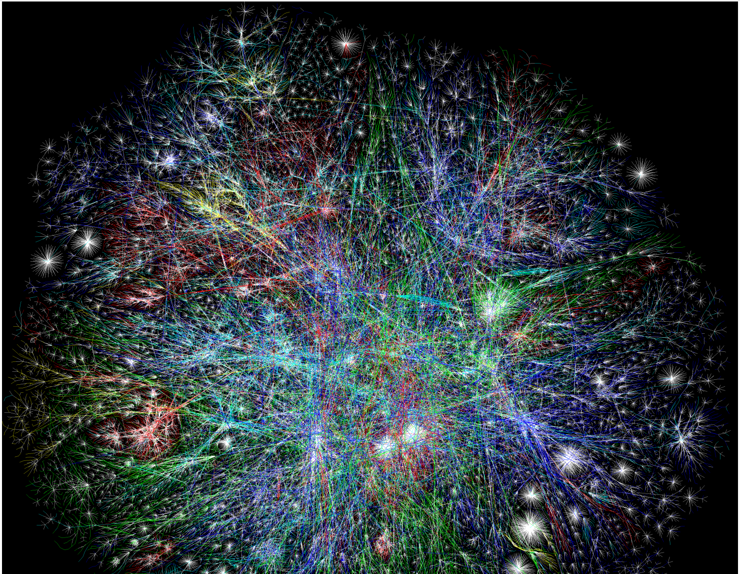
species	period	height	weight	volume
H. neanderthalensis	0.35 – 0.03	1.6	55 – 70	1200 – 1700
H. sapiens sapiens	0.2 – present	1.4 – 1.9	50 – 100	1000 – 1850
H. heidelbergensis	0.6 – 0.35	1.8	60	1100 – 1400
H. erectus	1.4 – 0.2	1.8	60	850 (early) – 1100
H. floresiensis	0.10 – 0.012	1.0	25	400
H. ergaster	1.9 – 1.4	1.9		700 – 850
H. habilis	2.2 – 1.6	1.0 – 1.5	33 – 55	660

Text processing

- We have seen that Python is well suited for mathematical calculations and visualizations.
- Python is also an efficient tool for processing of text strings. * Applications involving text processing are very common.
- Many advanced applications of text processing (e.g. web search and DNA analysis) involve mathematical and statistical computations.

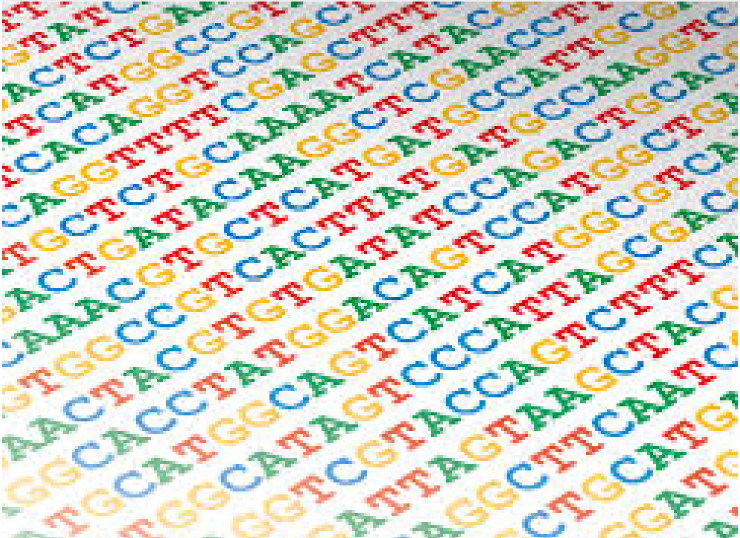
Example: web search

Google and other web search tools do advanced text processing.
Crawlers browse WWW for files and analyse their content.



Example: DNA analysis

DNA sequences are very long strings with known and undiscovered patterns. Algorithms to find and compare such patterns are very important in modern biology and medicine.



Text processing: a quick recap

```
s = 'This is a string, ok?'

# To split a string into individual words:
s.split()  # ['This', 'is', 'a', 'string,', 'ok?']

# To split a string with another delimiter
s.split(',')         # ['This is a string', ' ok?']
s.split('a string') # ['This is ', ', ok?']

# To find the location of a substring:
s.index('is')  # 2

# To check if a string contains a substring:
'This' in s  # True
'this' in s  # False

# To select a particular character in a string:
s[0]  # 'T'
s[1]  # 'h'
s[2]  # 'i'
s[3]  # 's'
```

Extracting substrings

```
s = 'This is a string, ok?'

# Remove the first character
s[1:]     # 'his is a string, ok?'

# Remove the first and the last character
s[1:-1]   # 'his is a string, ok'

# Remove the two first and two last characters
s[2:-2]   # 'is is a string, o'

# The characters with index 2,3,4
s[2:5]    # 'is '

# Select everything starting from a substring
s[s.index('is a'):]  # 'is a string, ok?'

# Remove trailing blanks
s = '   A B C   '
s.strip()    # 'A B C'
s.lstrip()   # 'A B C   '
s.rstrip()   # '   A B C'
```

Concatenating strings

```python
a = ['I', 'am', 'happy']

# Join list elements
''.join(a)    # 'Iamhappy'

# Join list elements with space between them
' '.join(a)  # 'I am happy'

# Join list elements with '%%' between them
'%%'.join(a) # 'I%%am%%happy'
```

Substituting substrings

```
s = 'This is a string, ok?'

# Replace every blank by 'X'
s.replace(' ', 'X')   # 'ThisXisXaXstring,Xok?'

# Replace one word by another
s.replace('string', 'text')   # 'This is a text, ok?'

# Replace the text before the comma by 'Fine'
s.replace(s[:s.index(',')], 'Fine')   # 'Fine, ok?'

# Replace the text from the comma by ' dummy'
s.replace(s[s.index(','):], ' dummy')   # 'This is a string dummy'
```

Line breaks in text strings

Lines are separated by different control characters on different platforms.

```
# Concatenate with Unix/Linux/Mac type line break
s1 = '\n'.join(['Line A', 'Line B', 'Line C'])

# Concatenate with Windows type line break
s2 = '\r\n'.join(['Line A', 'Line B', 'Line C'])

# Platform dependent line splitting:
s1.split('\n')     # Works: ['Line A', 'Line B', 'Line C']
s1.split('\r\n')   # FAILS: ['Line A\nLine B\nLine C']

s2.split('\n')     # FAILS: ['Line A\r', 'Line B\r', 'Line C']
s2.split('\r\n')   # Works: ['Line A', 'Line B', 'Line C']

# Better line splitting (platform independent):
s1.splitlines()    # Works: ['Line A', 'Line B', 'Line C']
s2.splitlines()    # Works: ['Line A', 'Line B', 'Line C']
```

A few more string functions

```
# Check if a string only contains digits
s = '314'
s.isdigit()    # True
s = '   314'
s.isdigit()    # False
s = '3.14'
s.isdigit()    # False

# Change to lower-case or upper-case
s = 'ABC def'
s.lower()      # 'abc def'
s.upper()      # 'ABC DEF'

# Starts with and ends with substring
s = 'This is a string'
s.startswith('This is')    # True
s.endswith('This is')      # False
```

Example

Suppose we want to read pairs of numbers (x,y) from a file.

Sample 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)
```

Algorithm:

1. Read one line at a time
2. For each line, split line into words
3. For each word, strip off parentheses and split the rest on comma

How to do it in Python

```
infile = open('pairs.dat', 'r')
pairs = []    # Create a list to hold all the pairs
for line in infile:
    words = line.split()
    for w in words:
        w = w[1:-1]    # Remove parentheses
        numbers = w.split(',')
        pair = (float(numbers[0]), float(numbers[1]))
        pairs.append(pair)
```

The pairs list

```
[(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)]
```

Alternative solution: Python syntax in file format

Suppose the file format

```
(1.3, 0)    (-1, 2)    (3, -1.5)
...
```

was slightly different:

```
[(1.3, 0),    (-1, 2),    (3, -1.5),
...
]
```

Running `eval` on the perturbed format produces the desired list!

```
text = open('read_pairs2.dat', 'r').read()
text = '[' + text.replace(')', '),') + ']'
pairs = eval(text)
```

Web pages are nothing but text files

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>
```

Programs can extract data from web pages

- A program can download a web page, as an HTML file, and extract data by interpreting the text in the file (using string operations).
- Example: climate data from the UK

Download `oxforddata.txt` to a local file `Oxford.txt`:

```
import urllib
baseurl = 'http://www.metoffice.gov.uk/climate/uk/stationdata'
filename = 'oxforddata.txt'
url = baseurl + '/' + filename
urllib.urlretrieve(url, filename='Oxford.txt')
```

The structure of the Oxfort.txt weather data 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
```

Reading the climate data

Algorithm:

1. Read the place and location in the file header
2. Skip the next 5 (for us uninteresting) lines
3. Read the column data and store in dictionary
4. Test for numbers with special annotation, "provisional" column, etc.

Program, part 1:

```
local_file = 'Oxford.txt'
infile = open(local_file, 'r')
data = {}
data['place'] = infile.readline().strip()
data['location'] = infile.readline().strip()
# Skip the next 5 lines
for i in range(5):
    infile.readline()
```

Reading the climate data - program, part 2

Program, part 2:

```
data['data'] ={}
for line in infile:
    columns = line.split()

    year = int(columns[0])
    month = int(columns[1])

    if columns[-1] == 'Provisional':
        del columns[-1]
    for i in range(2, len(columns)):
        if columns[i] == '---':
            columns[i] = None
        elif columns[i][-1] == '*' or columns[i][-1] == '#':
            # Strip off trailing character
            columns[i] = float(columns[i][:-1])
        else:
            columns[i] = float(columns[i])
```

Reading the climate data - program, part 3

Program, part 3

```
for line in infile:
    ...
    tmax, tmin, air_frost, rain, sun = columns[2:]

    if not year in data['data']:
        data['data'][year] = {}
    data['data'][year][month] = {'tmax': tmax,
                                 'tmin': tmin,
                                 'air frost': air_frost,
                                 'sun': sun}
```