Working with text

- file formats
- CSV, JSON, XML, Excel
- regular expressions
- module re, finditer

Some file formats

File extension	Content
.html	HyperText Markup Language
.mp3	Audio File
.png .jpeg .jpg	Image files
.svg	Scalable Vector Graphics file
.json	JavaScript Object Notation
.csv	Comma separated values
.xml	eXtensible Markup Language
.xlmx	Micosoft Excel 2010/2007 Workbook

File extension	Description
.exe	Windows executable file
.app	Max OS X Application
.py	Python program
.pyc	Python compiled file
.java	Java program
.cpp	C++ program
.c	C program
.txt	Raw text file

PIL – the Python Imaging Library

- pip install Pillow

rotate_image.py

```
from PIL import Image
img = Image.open("Python-Logo.png")
img_out = img.rotate(45, expand=True)
img_out.save("Python-rotated.png")
```

Python-Logo.png

Python-rotated.png

- *For many file types there exist Python packages handling such files, e.g. for images*

python-pillow.org

CSV files - Comma Separated Values

- Simple 2D tables are stored as rows in af file, with values separated by comma
- Strings stored are quoted if necessary
- Values read are strings
- The deliminator (default comma) can be changed by keyword argument `delimiter`

csv-example.py

```
import csv
FILE = 'csv-data.csv'
data = [[1, 2, 3],
        ['a', '"b"'],
        [1.0, ['x',"y"], 'd']]
with open(FILE, 'w', newline="\n") as outfile:
    csv_out = csv.writer(outfile)
    for row in data:
        csv_out.writerow(row)
with open(FILE) as infile:
    for row in csv.reader(infile):
        print(row)
```

Python shell

```
['1', '2', '3']
['a', '"b"']
['1.0', "['x', 'y']", 'd']
```

csv-data.csv

```
1,2,3
a,"""b"""
1.0,"['x', 'y']",d
```

docs.python.org/3/library/csv.html

CSV files - Tab Separated Values

tab-separated.py

```
import csv

FILE = 'tab-separated.csv'

with open(FILE) as infile:
    for row in csv.reader(infile, delimiter='\t'):
        print(row)
```

Python shell

```
['1', '2', '3']
['4', '5', '6']
['7', '8', '9']
```

tab-separated.csv

```
1	2	3
4	5	6
7	8	9
```

JSON

*"**JSON** (**J**ava**S**cript **O**bject **N**otation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is an ideal data-interchange language."*

www.json.org

- Human readable file format
- Easy way to save a Python expression to a file
- Does not support all Python types, .e.g sets are not supported, and tuples are saved (and later loaded) as lists

JSON example

json-example.py

```
import json
FILE = 'json-data.json'
data = ((None, True), (42.7, (42,)), [3,2,4], (5,6,7),
        {'b':'banana', 'a':'apple', 'c': 'coconut'})

with open(FILE, 'w') as outfile:
    json.dump(data, outfile, indent=2, sort_keys=True)

with open(FILE) as infile:
    indata = json.load(infile)

print(indata)
```

Python shell

```
[[None, True], [42.7, [42]], [3, 2, 4], [5, 6, 7], {'a':
'apple', 'b': 'banana', 'c': 'coconut'}]
```

json-data.json

```
[
  [
    null,
    true
  ],
  [
    42.7,
    [
      42
    ]
  ],
  [
    3,
    2,
    4
  ],
  [
    5,
    6,
    7
  ],
  {
    "a": "apple",
    "b": "banana",
    "c": "coconut"
  }
]
```

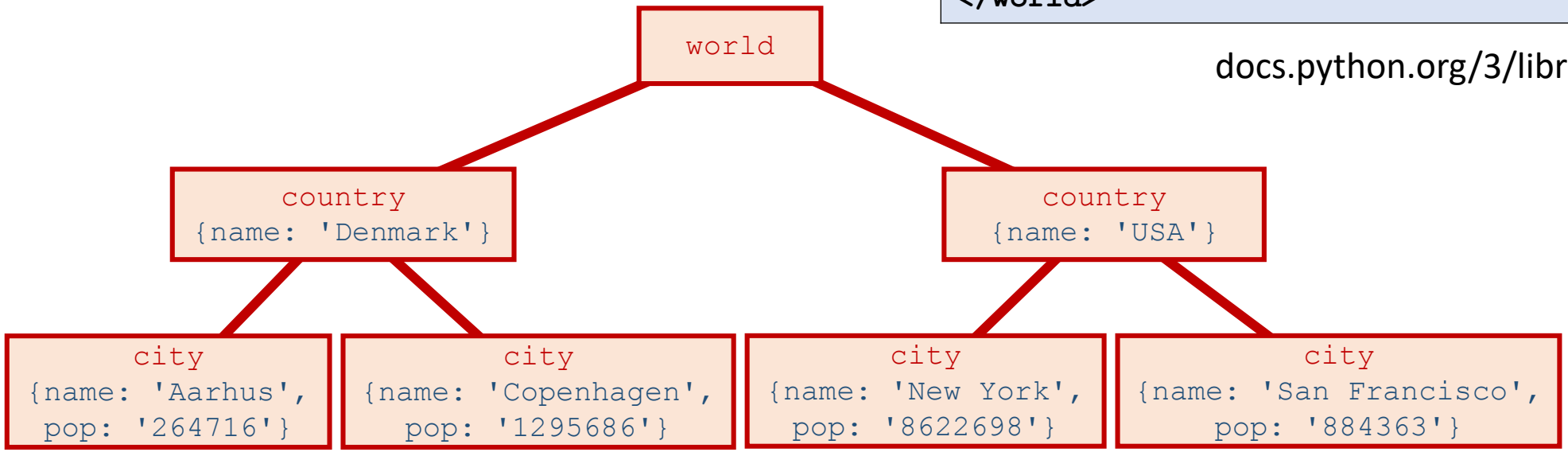
XML - eXtensible Markup Language

- XML is a widespread used data format to store hierarchical data with tags and attributes

cities.xml

```
<?xml version="1.0"?>
<world>
  <country name="Denmark">
    <city name="Aarhus" pop="264716"/>
    <city name="Copenhagen" pop="1295686"/>
  </country>
  <country name="USA">
    <city name="New York" pop="8622698"/>
    <city name="San Francisco" pop="884363"/>
  </country>
</world>
```

docs.python.org/3/library/xml.html



xml-example.py

```
import xml.etree.ElementTree as ET
FILE = 'cities.xml'
tree = ET.parse(FILE)  # parse XML file to internal representation
root = tree.getroot()  # get root element
for country in root:
    for city in country:
        print(city.attrib['name'],  # get value of attribute for an element
              'in',
              country.attrib['name'],
              'has a population of',
              city.attrib['pop'])
print(root.tag, root[0][1].attrib)  # the tag & indexing the children of an element
print([city.attrib['name'] for city in root.iter('city')])  # .iter finds elements
```

Python shell

```
Aarhus in Denmark has a population of 264716
Copenhagen in Denmark has a population of 1295686
New York in USA has a population of 8622698
San Francisco in USA has a population of 884363
world {'name': 'Copenhagen', 'pop': '1295686'}
['Aarhus', 'Copenhagen', 'New York', 'San Francisco']
```

XML tags with text

city-descriptions.xml

```
<?xml version="1.0"?>
<world>
  <country name="Denmark">
    <city name="Aarhus" pop="264716">The capital of Jutland</city>
    <city name="Copenhagen" pop="1295686">The capital of Denmark</city>
  </country>
  <country name="USA">
    <city name="New York" pop="8622698">Known as Big Apple</city>
    <city name="San Francisco" pop="884363">Home of the Golden Gate Bridge</city>
  </country>
</world>
```

xml-descriptions.py

```
import xml.etree.ElementTree as ET
FILE = 'city-descriptions.xml'
tree = ET.parse(FILE)
root = tree.getroot()

for city in root.iter('city'):
    print(city.get('name'), "-", city.text)
```

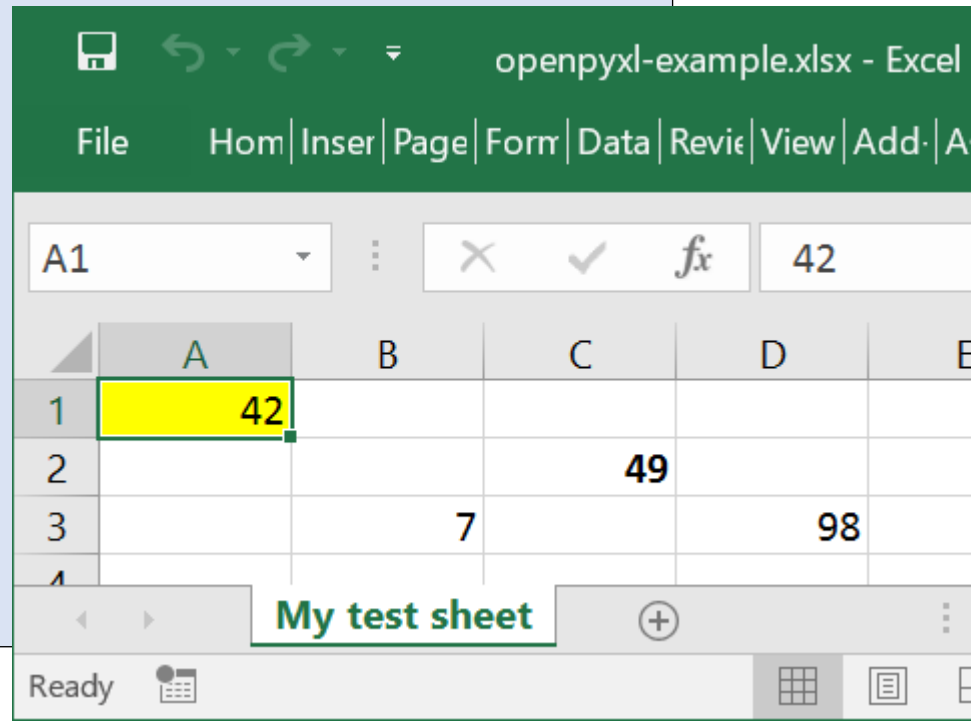
Python shell

```
Aarhus - The capital of Jutland
Copenhagen - The capital of Denmark
New York - Known as Big Apple
San Francisco - Home of the Golden Gate Bridge
```

Openpyxl - Microsoft Excel 2010 manipulation

openpyxl-example.xml

```
from openpyxl import Workbook
from openpyxl.styles import Font, PatternFill

wb = Workbook() # create workbook
ws = wb.active  # active worksheet

ws['A1'] = 42
ws['B3'] = 7
ws['C2'] = ws['A1'].value + ws['B3'].value
ws['D3'] = '=A1+B3+C2'

ws.title = 'My test sheet'

ws['A1'].fill = PatternFill('solid', fgColor='ffff00')
ws['C2'].font = Font(bold=True)

wb.save("openpyxl-example.xlsx")
```



String searching using `find`

- Search for first occurence of *substring* in *str*[*start*, *end*]

str.find(*substring*[, *start*[, *end*]])

string-search.py

```
text = 'this is a string - a list of characters'
pattern = 'is'

idx = text.find(pattern)
while idx >= 0:
    print(idx)
    idx = text.find(pattern, idx + 1)
```

Python shell

```
2
5
22
```

docs.python.org/3/library/stdtypes.html#textseq

Regular expression
– A powerful language to describe sets of strings

- **Examples**
  - `abc` denotes a string of letters
  - `ab*c` any string starting with `a`, followed by an arbitrary number of `b`s and terminated by `c`, i.e. {`ac`, `abc`, `abbc`, `abbbc`, `abbbbc`, ...}
  - `ab+c` as `ab*c`, except that there must be at least one `b`
  - `a\wc` any three letter string, starting with `a` and ending with `c`, where second character is any character in `[a-zA-Z0-9_]`
  - `a[xyz]c` any three letter string, starting with `a` and ending with `c`, where second character is either `x`, `y` or `z`
  - `a[^xyz]c` any three letter string, starting with `a` and ending with `c`, where second character is none of `x`, `y` or `z`
  - `^xyz` match at start of string
  - `xyz$` match at end of string
  - ...
- See docs.python.org/3.6/library/re.html, Section 6.2.1, for more

String searching using regular expressions

- `re.search(`*`pattern, text`*`)`
  - find the first occurence of *`pattern`* in *`text`* – returns `None` or a *match object*
- `re.findall(`*`pattern, text`*`)`
  - returns a list of non-overlapping occurence of *`pattern`* in *`text`* – returns a list of substrings
- `re.finditer(`*`pattern, text`*`)`
  - iterator returning a match object for each non-overlapping occurence of *`pattern`* in *`text`*

Python shell

```
> text = 'this is a string - a list of characters'
> re.findall(r'i\w*', text)
| ['is', 'is', 'ing', 'ist']
> for m in re.finditer(r'a[^at]*t', text):
      print("text[%s, %s] = %s" % (m.start(), m.end(), m.group()))
| text[8, 12] = a st
  text[19, 25] = a list
  text[33, 36] = act
```

docs.python.org/3.6/library/re.html

Substitution and splitting using regular expressions

- `re.sub(`*`pattern`*`, `*`replacement`*`, `*`text`*`)`
  - replace any occurence of the *`pattern`* in *`text`* by *`replacement`*
- `re.split(`*`pattern`*`, `*`text`*`)`
  - split *`text`* at all occurences of *`patern`*

Python shell

```
> text = 'this is a string - a list of characters'
> re.sub(r'\w*i\w*', 'X', text)
| 'X X a X - a X of characters'
> re.split(r'[^\w]+a[^\w]+', text)
| ['this is', 'string', 'list of characters']
```

docs.python.org/3.6/library/re.html