UiT

THE ARCTIC UNIVERSITY OF NORWAY

Textparsing

Roy Dragseth <roy.dragseth@uit.no>

HPC@UiT

Overview

- Manipulating text using builtin methods
- Regexps
- Working with spreadsheets, the csv module
- XML parsing

A word about style.

Try to be as explicit as possible. Readability counts.

Example:

Prefer dict() over {}, `dict(a=1, b=2)` is better than `{"a" = 1, "b" = 2}`

A novice can search the documentation for dict, but not for {.

The same goes for list, tuple and so on.

The evil side, PERL: What the h... does `for(<>);` mean???

A word of advice from the trenches: If you think you have written a really clever piece of code, think again. Will you be able to understand this in six months?

Use builtins!

The python language has a LOT of functionality built into it. The native operations on builtins are FAST.

`tuple` -- Unmutable arrays.

`list` -- list of things (array-like)

`set` -- of unique tings.

`dict` -- Dictionary, a container with a lot of flexibility.

`tuple` and `list` have integer indexes. `dict` can have (almost) any index. `set` is a strange beast.

Also, a lot of the most common tasks are already implemented in the standard library.

A word about dicts.

`dict` is a very powerful construction and can be used to solve problems where one needs a container for unstructured data elements.

```
>>> a = {'first' : 1, 'second': 2, 'third' : 3, 'fourth' : 4}
>>> a.keys()
['second', 'fourth', 'third', 'first']
>>> a.values()
[2, 4, 3, 1]
>>> a.items()
[('second', 2), ('fourth', 4), ('third', 3), ('first', 1)]
>>> for k, v in a.items():
...     print "key=", k, "value=",v
...
key= second value= 2
key= fourth value= 4
key= third value= 3
```

```
key= first value= 1
>>> a['fifth'] = 5
>>> a
{'second': 2, 'fifth': 5, 'fourth': 4, 'third': 3, 'first': 1}
```

Manipulating text using builtin methods

The str type in python has many useful builtin methods for easy matching and manipulation.

- Splitting strings into lists
- Stripping off redundant space
- Selecting text based on substrings
- Search and replace
- Checking for numbers
- Complete example
- Further info

Splitting strings into lists

The split method can chunk up text to your liking.

```
>>> text="This is a text string in Python"
>>> print text.split()
['This', 'is', 'a', 'text', 'string', 'in', 'Python']
>>> values="one,two,three,four"
>>> print values.split(",")
['one', 'two', 'three', 'four']
>>> print values.split(",", 2)
['one', 'two', 'three,four']
```

Stripping off redundant space

Often inputs contain to much whitespace, strip is your friend.

```
>>> text="   some text      "
>>> print "|",text,"|"
|     some text       |
>>> print "|",text.strip(),"|"
| some text |
>>> print "|",text.lstrip(),"|"
| some text       |
>>> print "|",text.rstrip(),"|"
|     some text |
```

Adding space is also easy

```
>>> print text.rjust(80)
                                                                       some text
>>> print text.center(80)
                                   some text
```

Selecting text based on substrings

You can search for substrings with the find method

```
>>> text="This is a string"
>>> print text.find('s')
3
>>> text="This is a string"
>>> if "string" in text:
...     print "Found it"
...
Found it
```

Search and replace

Text strings have builtins for search and replace

```
>>> text="This is a string"
>>> print text.replace('s','x')
Thix ix a xtring
>>> text[3]='x'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Note that strings are immutable objects!

Checking for numbers

Text strings can check if they are an integer.

```
>>> anumber="1"
>>> anumber.isdigit()
True
>>> a=int(anumber)
>>> a
1
```

Converting a non-integer will throw an exception.

Complete example

Let us take a look at the participant list and display the participants with uit.no email address.

```
List of participants
=====================

.. csv-table:: Workshop 2012
    :header: #, Name, E-mail address, Affiliation
    :widths: 5, 16, 30, 20

    1 , Heike Fliegl       , heike.fliegl@kjemi.uio.no           , CTCC/UiO
    2 , Maarten Beerepoot  , m.t.p.beerepoot@umail.leidenuniv.nl , CTCC/UiT
    3 , Arne Bunkan        , a.j.c.bunkan@kjemi.uio.no           , CTCC/UiO
    4 , Ole Jacob Broch    , olejacob.broch@sintef.no            , SINTEF
    5 , Karel Viaene       , karel.viaene@ugent.be               , Universiteit Gent
    6 , Lisette de Hoop    , L.deHoop@science.ru.nl              , Radboud University
```

```
    7 , Thomas Beka        , thomas.beka@uit.no                  , IFT/UiT
    8 , Alexander Tveit    , alexander.t.tveit@uit.no            , Dept. for Arctic and Marine Biology
    9 , Nicolas Horne      , nicolas.horne@uit.no                , Kunstakademiet
    10, Elena Malkin        , elena.malkin@gmail.com              , CTCC?
    11, Stian Sjøli        , stian.sjoli@uit.no                  , Norstruct
```

```
    12, Geir Isaksen       , geir.isaksen@uit.no                 , Norstruct
    13, Stanislav Komorovsky, stanislav.komorovsky@uit.no         , CTCC
    14, Tan Thi Nguyen     , tan.t.nguyen@uit.no                   , BFE
    15, Espen Robertsen    , espen.m.robertsen@uit.no              , Norstruct
    16, Taye B.            , sene3095@gmail.com                    , CTCC/UiT

.. csv-table:: Workshop 1/2013 (pending)
    :header: #, Name, E-mail address, Affiliation
    :widths: 5, 16, 30, 20

    1,   Marcin Pierechod    , marcin.m.pierechod@uit.no            , Norstruct
    2,   Cecilie Bækkedal    , cecilie.baekkedal@googlemail.com     , Norstruct
    3,   Ilona Urbarova      , ilona.urbarova@uit.no                , Norstruct
    4,   Laura Liikanen      , liikanenlaura@googlemail.com         , Norstruct
    5,   Ravna Sarre         , ravna_s@hotmail.com                  , Norstruct
    6,   Espen Åberg         , espen.aberg@uit.no                   , Norstruct
    7,   Hege Bøvelstad      , hege.m.bovelstad@uit.no              , ISM
    8,   Glenn B.S. Miller   , g.b.miller@kjemi.uio.no              , CTCC/UiO
    9,   Elina Halttunen     , elina.halttunen@uit.no               , BFE
    10,  Kåre Olav Holm      , kare.olav.holm@uit.no                , Norstruct
    11,  Davide Michetti     , dmichetti@gmail.com                  , CTCC/UiT
    12,  Thomas Kræmer       , thomakra@gmail.com                   , IFT/UiT
    13,  Myrland Øystein     , oystein.myrland@uit.no               , HHT/UiT
    14,  Inaki Rodriguez     , inaki.rodriguez@uit.no               , HHT/UiT
    ~15, Ralph Kube          , ralph.kube@uit.no                    , Nordlysobservatoriet
    ~16, Alexander Kashulin , aleksandr.kashulin@uit.no            , Norstruct
```

The program

participants.py

```
fd = file("Python-workshop-participants.rst", 'r')

participants = list()
for line in fd.readlines():
    if '@' in line:
        _, name, email, affiliation = map(str.strip, line.split(','))
        p = {'name' : name, 'email' : email, 'affiliation' : affiliation}
        participants.append(p)

    if 'pending' in line:
        break

for p in participants:
    if p['email'].endswith('uit.no'):
        print "name=%(name)25s email=%(email)20s affiliation=%(affiliation)20s" % p
```

There are a couple of WTFs here: map??, string formatting.

The output

```
$ python -i participants.py Python-workshop-participants.rst
name=             Thomas Beka email=  thomas.beka@uit.no affiliation=             IFT/UiT
name=         Alexander Tveit email=alexander.t.tveit@uit.no affiliation=Dept. for Arctic and Marine Biology
name=           Nicolas Horne email=nicolas.horne@uit.no affiliation=      Kunstakademiet
name=            Stian Sjøli email=  stian.sjoli@uit.no affiliation=           Norstruct
name=            Geir Isaksen email= geir.isaksen@uit.no affiliation=           Norstruct
name=     Stanislav Komorovsky email=stanislav.komorovsky@uit.no affiliation=                CTCC
name=          Tan Thi Nguyen email= tan.t.nguyen@uit.no affiliation=                 BFE
name=         Espen Robertsen email=espen.m.robertsen@uit.no affiliation=           Norstruct
```

Exercises

1. List the non-uit emails.
2. Count the number of participants based from each email domain.
3. Split the name into firstname and surname.

- Further info

The inline docs on strings is pretty extensive, `help(str)` in ipython should be a good start.

Sorting.

Sorting stuff is often needed and Python provides a lot of neat things right out of the box.

The `list` has a builtin sort-functionality

```
>>> a = [1,3,5,7,9,2,4,6,8]
>>> a.sort()
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a.sort(reverse=True)
>>> a
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Remark: `list().sort()` is done in place!

The builtin function `sorted()` can produce a sorted copy of a list.

```
>>> a = [1,3,5,7,9,2,4,6,8]
>>> sorted(a)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a
[1, 3, 5, 7, 9, 2, 4, 6, 8]
```

More on sorting in Python: https://wiki.python.org/moin/HowTo/Sorting/

Regexps

For more advanced text manipulations one needs to use regexps (REGular EXPressions).

- Regexps is a language for matching text.
- The syntax is really cryptic
- Example regexp matching dates in format `yyyy-mm-dd`, `yyyy/mm/dd` or `yyyy.mm.dd`.

```
(19|20)\d\d([- /.])(0[1-9]|1[012])\2(0[1-9]|[12][0-9]|3[01])
```

- Do a google search for regex cheat-sheet, print it out and tape it to the wall.

Basics

http://www.regular-expressions.info/reference.html

Python Regular Expression HowTo

http://docs.python.org/2/howto/regex.html

Python has its own regexp library, re

Search and grab

Find all emails

findemails.py

```
import re

emails=re.compile("([^ ,]+@[^ ,]+)")

text = file("Python-workshop-participants.rst", "r").read()
for m in emails.findall(text):
  print m
```

The regexp says, find the widest substring that contain @, but not space or comma.

- () marks a pattern group that can be referenced later.
- [] denotes character classes, [a-z]= all lowercase chars. [^] not, [^a-z] anything except the lowercase chars.

- . any character, + one or more matches, * zero or more matches -> .* will match any string.
- if you want to match a . (a dot) you need to quote it with backslash, \.
- if you want to match a backslash you need to quote it with a backslash, \\

Note that this is not the way to match email addresses in general. To match a RFC822 compliant email adress you need to do this

```
\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}\b
```

(YIKES!!!)

```
$ python  findemails.py
heike.fliegl@kjemi.uio.no
m.t.p.beerepoot@umail.leidenuniv.nl
a.j.c.bunkan@kjemi.uio.no
olejacob.broch@sintef.no
karel.viaene@ugent.be
```

Search and replace

Scramble all emails

Add "-nospam" to all emails to prevent spam-bots to get hold of published emails.

scrambleemails.py

```
import re

emails=re.compile("([^ ,]+)(@[^ ,]+)")

text = file("Python-workshop-participants.rst", "r").read()
newtext = emails.sub(r"\1-nospam\2", text)
print newtext
```

- \1 contain the part before @, \2 contains the part after @.

Result

```
$ python scrambleemails.py
```

```
List of participants
=====================

.. csv-table:: Workshop 2012
    :header: #, Name, E-mail address, Affiliation
    :widths: 5, 16, 30, 20

    1 , Heike Fliegl       , heike.fliegl-nospam@kjemi.uio.no          , CTCC/UiO
    2 , Maarten Beerepoot  , m.t.p.beerepoot-nospam@umail.leidenuniv.nl , CTCC/UiT
    3 , Arne Bunkan        , a.j.c.bunkan-nospam@kjemi.uio.no          , CTCC/UiO
    .
    .
    .
```

(OK, this might not fool even the stupidest spam-bot.)

Further info

Do a google search on "python regexp" and you will find more than you ever need.

Using a regexp editor is often a good help: http://myregexp.com/

A word of caution from the Python Regexp Howto:

> Sometimes using the re module is a mistake.

In many cases the builtin string methods are easier to use and you can actually understand what you were doing six months from now on.

Working with spreadsheets

The simplest way to work with data from spreadsheets is to go via the csv format (Comma Separated Values).

The csv format is just a textfile with lines of data-entries separated by a common character. (Not neccessarily a comma.)

```
1 , Heike Fliegl       , heike.fliegl@kjemi.uio.no            , CTCC/UiO
2 , Maarten Beerepoot  , m.t.p.beerepoot@umail.leidenuniv.nl , CTCC/UiT
3 , Arne Bunkan        , a.j.c.bunkan@kjemi.uio.no            , CTCC/UiO
```

All spreadsheet applications and databases can export to csv format. (At least the ones I've heard of.)

Basics

Python has a csv library in its standard distribution.

The main parts are the writer and reader objects.

Writing csv files

The writer has a method, writerow, that will take a list and create one line in the csv file.

writeparticipants2csv.py

```
import csv

fd = file("Python-workshop-participants.rst", "r")

participants = list()
for line in fd.readlines():
    if '@' in line:
        _, name, email, affiliation = map(str.strip, line.split(','))
        p = dict(name=name, email=email, affiliation=affiliation)
        participants.append(p)
    if 'pending' in line:
```

```
        break
```

```python
csvfile = file("participants.csv", "w")
writer = csv.writer(csvfile)
writer.writerow(["Name", "Email", "Affiliation"])
for p in participants:
    writer.writerow([p["name"], p["email"], p["affiliation"]])
```

Reading csv files

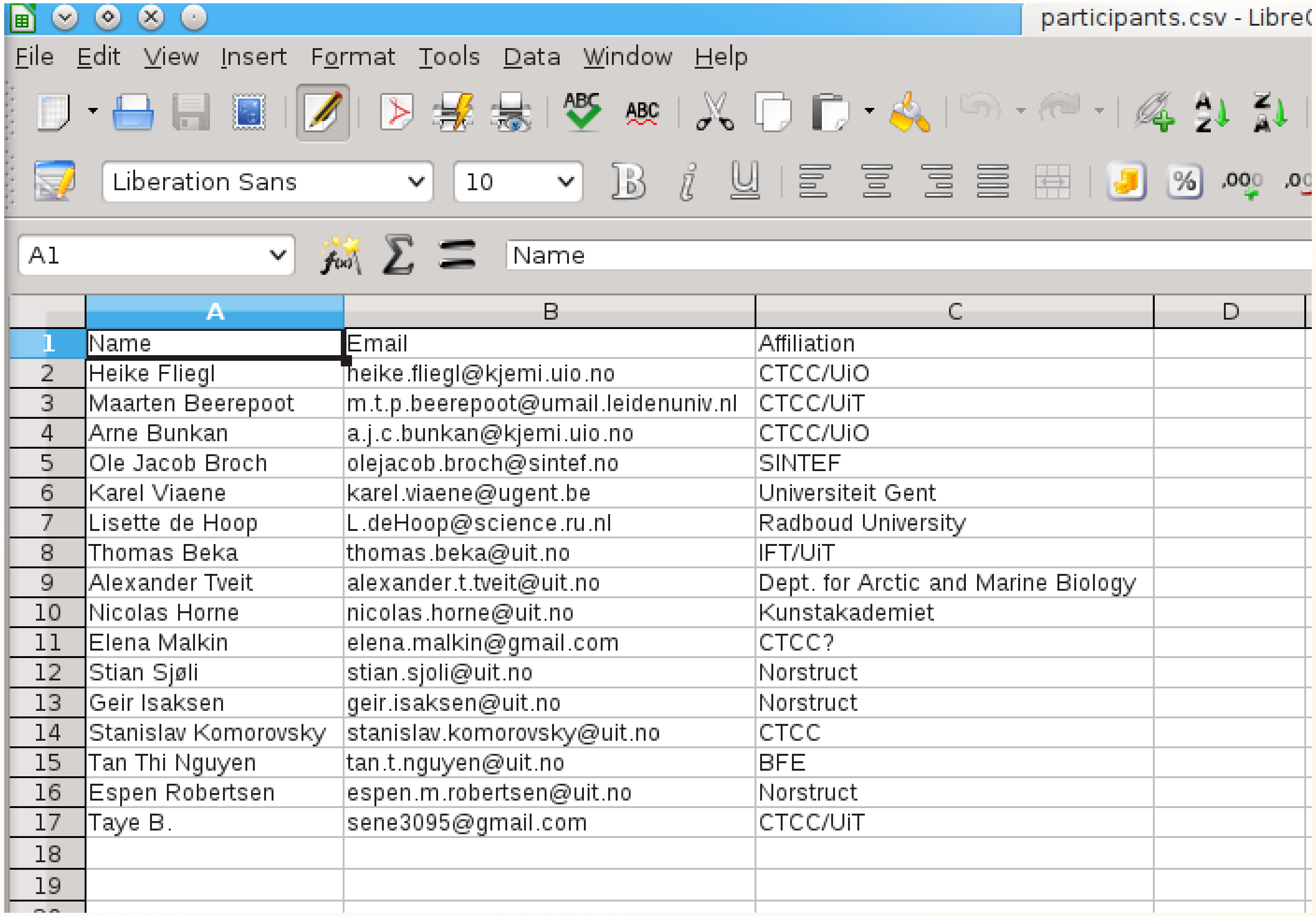
Reading csv files is equally simple.

readparticipantsfromcsv.py

```
import csv

csvfile = file("participants.csv", "r")
reader = csv.reader(csvfile)
for row in reader:
    print row
```

Or view it in a spreadsheet



	A	B	C	D
1	Name	Email	Affiliation	
2	Heike Fliegl	heike.fliegl@kjemi.uio.no	CTCC/UiO	
3	Maarten Beerepoot	m.t.p.beerepoot@umail.leidenuniv.nl	CTCC/UiT	
4	Arne Bunkan	a.j.c.bunkan@kjemi.uio.no	CTCC/UiO	
5	Ole Jacob Broch	olejacob.broch@sintef.no	SINTEF	
6	Karel Viaene	karel.viaene@ugent.be	Universiteit Gent	
7	Lisette de Hoop	L.deHoop@science.ru.nl	Radboud University	
8	Thomas Beka	thomas.beka@uit.no	IFT/UiT	
9	Alexander Tveit	alexander.t.tveit@uit.no	Dept. for Arctic and Marine Biology	
10	Nicolas Horne	nicolas.horne@uit.no	Kunstakademiet	
11	Elena Malkin	elena.malkin@gmail.com	CTCC?	
12	Stian Sjøli	stian.sjoli@uit.no	Norstruct	
13	Geir Isaksen	geir.isaksen@uit.no	Norstruct	
14	Stanislav Komorovsky	stanislav.komorovsky@uit.no	CTCC	
15	Tan Thi Nguyen	tan.t.nguyen@uit.no	BFE	
16	Espen Robertsen	espen.m.robertsen@uit.no	Norstruct	
17	Taye B.	sene3095@gmail.com	CTCC/UiT	
18				
19				

Working with xls files.

If you have a ton of EXCEL files, you do not need to manually convert them into csv to be able to work with them in python.

There are several module available for working directly with xls files.

http://www.python-excel.org/

is a good start.

XML data format

XML (eXtended Markup Language) is a standard for transporting data between different systems.

It is widely used in many large projects, but can be very complex to deal with.

Due to its flexible format it is prohibitively hard to use regular string methods or regexps to parse XML files.

One needs a full-blown parser, and again, python has a module for it, lxml.

XML resembles HTML, everything is embedded in tags, <tag>data</tag>.

The lxml module

Extensive documentation can be found at http://lxml.de/tutorial.html

The lxml module can be used to both write and read xml files.

A special submodule, lxml.html, can be used to parse and create html.

Writing an XML file

writeparticipants2xml.py

```
import csv
fd = file("Python-workshop-participants.rst", "r")
participants = list()
for line in fd.readlines():
    if '@' in line:
        _, name, email, affiliation = map(str.strip, line.split(','))
        p = dict(name=name, email=email, affiliation=affiliation)
        participants.append(p)
    if 'pending' in line:
        break

from lxml.builder import E
from lxml import etree
```

```
participantelements = list()
for p in participants:
    element = E.participant(E.name(p["name"]),
                                E.email(p["email"]),
                                E.affiliation(p["affiliation"]))
     participantelements.append(element)

print etree.tostring(E.participants(*participantelements), pretty_print=True)
```

Results in this xml output.

```
<participants>
  <participant>
    <name>Heike Fliegl</name>
    <email>heike.fliegl@kjemi.uio.no</email>
    <affiliation>CTCC/UiO</affiliation>
  </participant>
  <participant>
    <name>Maarten Beerepoot</name>
    <email>m.t.p.beerepoot@umail.leidenuniv.nl</email>
    <affiliation>CTCC/UiT</affiliation>
  </participant>
  .
```

```
    .
    .
</participants>
```

Parsing an XML file

lxml provides several ways of parsing xml data, xpath, objectify, events.

Eventbased parsing.

readparticipantsfromxml.py

```python
from lxml import etree

events = ("start", )
context = etree.iterparse(file("participants.xml", "r"), events=events)
for a, ele in context:
  print  ele.tag, ele.text
```

Result:

```
participants

participant

name Heike Fliegl
```

```
email heike.fliegl@kjemi.uio.no
affiliation CTCC/UiO
participant

name Maarten Beerepoot
email m.t.p.beerepoot@umail.leidenuniv.nl
affiliation CTCC/UiT
participant

name Arne Bunkan
email a.j.c.bunkan@kjemi.uio.no
affiliation CTCC/UiO
```

A concrete example: Numbergrabbing.

Sometimes you need to make sense of a heap of rubbish...

The `TextParsing/manyfiles` catalog contain a zip-file with a lot of text data, 300 files. Grab the relevant number from each file and find the max and min value.

(Walk through interactive example)

The code: grabnumbers.py

```
import os

filenames = os.listdir(".")

#print filenames

runtimes = list()
for f in filenames:
    for line in file(f, "r").readlines():
        if line.startswith("WR12L2C2"):
            #print line

            linetokens = line.strip().split()
            #print linetokens[5]
            runtimes.append(float(linetokens[5]))

runtimes.sort()

print "min=", runtimes[0], "max=", runtimes[-1]
```

Exercises

Modify `grabnumbers.py` to

1. Find the slowest compute node. Hint: the computer name is the second entry on the first line of each file.
2. Find all compute nodes with more than 10% more runtime than the fastest one.

Summary

- Python have many ways to parse text.
- Use the simplest method you can, that is, prefer string builtins over regexps.