Lecture 1: Administrata Language-Theoretic Python

PCL II, CL, UZH
February 24, 2016

PCL2

- Programming, Python, NLTK
  - encoding, file input/output
  - code organization and project management
  - recursion
    - recursion
      - recursion…
- Algorithms and typical generic approaches
  - probabilities/machine learning
  - dynamic programming
- Tasks in natural language processing
  - sequence tagging
  - sentence alignment
  - parsing
  - information extraction

Target Audience, Learning Goals

CL/IFI majors/minors, MLTA students, who

- Have the level of knowledge of PCL1
- Want to extend programming skills and practice (in Python)
  - Faster and smarter programs
  - Prettier code

Target Audience, Learning Goals

CL/IFI majors/minors, MLTA students, who

- Want to extend programming skills and practice (in Python)
  - Faster and smarter programs
  - Prettier code
- Want to get theoretical and practical experience with NLP tasks
  - Understanding, designing and implementing algorithms to solve them

Target Audience, Learning Goals

CL/IFI majors/minors, MLTA students, who

- Want to extend programming skills and practice (in Python)
  - Faster and smarter programs
  - Prettier code
- Want to get theoretical and practical experience with NLP tasks
  - Understanding, designing and implementing algorithms to solve them
- Want to get easy points but won’t >:-)

Organization

Ask questions!!! Good question examples:

- I don’t get it / we haven’t studied this
- why does it…
- what is the
- what the.../why the…
- but isn’t it the other way around?
- what if you were instead to…

Bad question examples:

People

Lectures:

- Tilia Ellendorff, ellendorff@ifi.uzh.ch
- Laura Mascarell, mascarell@ifi.uzh.ch

Labs/“Tutorat”, homework:

- Raphael Balimann
- Irene Ma
- Cazim Hysi

Course Organization

Lectures

- Once a week
- Wednesdays, 10:15—12:00, K02 F-172
  - Except March 30, Easter
- Background theoretical & practical material
- Not mandatory

Course Organization

Topic blocks:

- general intro, language-theoretic intro, code organization *(Tilia)*
- I/O, XML *(Tilia)*
- probabilities and machine learning *(Laura)*
  - document classification, sequence tagging
- external programs; testing, debugging *(Laura/Tilia)*
- dynamic programming *(Tilia/Laura/Mark)*
  - sentence alignment, parsing
- semantics and information extraction *(Laura/Tilia)*

Course Organization

Universität Zürich UZH

Programming homework

- Once every two weeks
- Total: 6 tasks
- Collaboration allowed: at most 2 people can submit solutions together
    - NB! you must inform about collaborations in your submissions

Course Organization

Programming homework topics:

● input/output, encoding, classes	*Mar 04 -- Mar 17*
● XML	*Mar 18 -- Apr 07*
● machine learning	*Apr 08 -- Apr 21*
● sequence tagging	*Apr 22 -- May 05*
● dynamic programming	*May 06 -- May 19*
● syntax	*May 20 -- May 27*

Course Organization

Tutorat Sessions

- Once a week
- Fridays 12:15—14:00, BIN 0.B.06
  - Starting **March 04, 2016**
- Supervision and help for the practical side of this course, including the programming tasks
- Not mandatory

Course Organization

Exam: 24 h exam (independent DIY)

- start at 10am on a sunny day in June
- receive a list of tasks to fulfill
  - level of difficulty: same as practical tasks
  - load: ~1.5--2 of one programming homework
- submit by 10am next morning
- you can do it at home/library/etc.

(proposed date: Wed June 15 - Thu June 16 2016)

Exam Details

- Tasks similar to the 6 practical tasks
- You may use
  - lecture slides, Python documentation, existing online resources (e.g. forum discussions)
  - someone else's code: must be shown explicitly
  - no need to mark code from lectures/Python docs
- You may not
  - collaborate, outsource, etc: do tasks individually
  - if we have doubts: oral re-examination
  - in clear plagiarism cases: Course failed, Disziplinarmassnahmen

Final grade

- *x* = exam grade
- *y* = sum of programming task points
- Final grade:

$$\partial\left(\left(e^{-\ln x}\cdot\lceil\varphi\rceil\right)^{2e^{i\pi}}+\left(\begin{bmatrix}7&8\\4&5\end{bmatrix}-\lim_{a\to\mathrm{inf}}\frac{a+1}{a}\right)^{-1}\prod_{i\in\{x,y\}}i\right)\cdot\partial x^{-1}$$

Final grade

- 1 point is on the house
- The remaining 5 points:
  - 75% exam
  - 25% programming tasks
- E.g.
  - exam: **5.41** (out of 6), prog. tasks: **5.99** (out of 6)
  - $1 + (0.75 \times \mathbf{5.41} + 0.25 \times \mathbf{5.99}) \times \frac{5}{6} = 5.629 \rightarrow \mathbf{5.5}$
- Or
  - exam: **3.0** (out of 6), prog. tasks: **4.2** (out of 6)
  - $1 + (0.75 \times \mathbf{3.0} + 0.25 \times \mathbf{4.2}) \times \frac{5}{6} = 3.75 \rightarrow \mathbf{4.0}$

Course Organization

Use OLAT to

- See the course program
- Receive tasks, submit their solutions and receive task grades
- Receive exam task, submit its solution
- Receive final grade
- Discuss whatever you want at the PCL2 forum
- Contact Lecturers and Tutors

Suggested additional material

- Where can I see some examples of Python?
- Which functions/operators/types/... are there?

docs.python.org

- How do I...?
- Can I...?
- Why doesn't it work when I...?

www.google.com

Additional material

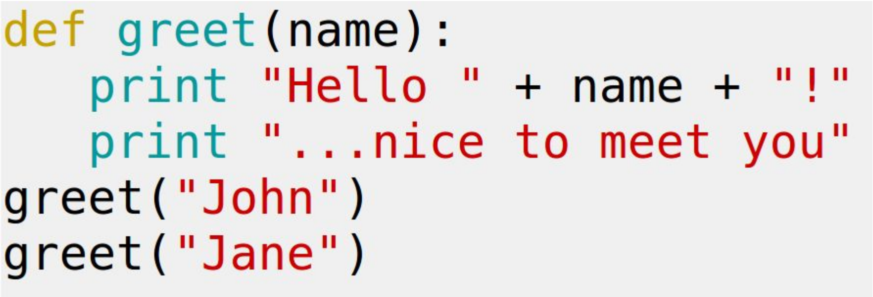
- Jurafsky & Martin: ***"Speech and Language Processing"***; library + http://www.cs.colorado.edu/~martin/slp.html
- Bird, Klein & Loper: ***"Natural Language Processing with Python"***; library + http://nltk.org/book/
- Manning & Schütze: ***"Foundations of Statistical Natural Language Processing"***; library + http://nlp.stanford.edu/fsnlp/
- ***Online NLP course from Stanford**:* https://www.coursera.org/course/nlp

Python

Outline

- Python as a programming language
  - indentation
  - programming paradigms
- Python’s type system
  - duck typing
- Exceptions, errors
  - handling exceptions/raising exceptions
  - error types
- Coding style, documentation

Python



```
def greet(name):
   print "Hello " + name + "!"
   print "...nice to meet you"
greet("John")
greet("Jane")
```

Python

```
def greet(name):
   print "Hello " + name + "!"
   print "...nice to meet you"
greet("John")
greet("Jane")
```

compare to PERL:

```
sub greet {
   my ($name) = @_;
   print "Hello $name!\n";
   print "...nice to meet you\n";
}
greet("John");
greet("Jane");
```

Python

```
def greet(name):
   print "Hello " + name + "!"
   print "...nice to meet you"
greet("John")
greet("Jane")
```

compare to PERL:

```
sub greet {     my ($name)
       = @_;
  print "Hello $name!\n";  print
             "...nice to meet you\n";
} greet("John");        greet("Jane");
```

Indentation important in Python (unlike PERL)

4 spaces per indentation level

Programming paradigms

- **Paradigm:** A style or way of doing something
- **Programming Paradigm**: A style or way of programming

- Python is a multi-paradigm programming language

Programming paradigms

Imperative programming:

- Program = list of statements that change the program state (variables, environment)
- Describe, **how to do something**
- Java, C++, ASM, PHP, Perl, VB, Python

Functional programming

- Program = functions, depending only on input (not on program state); order less important
- Describe, **what the program should accomplish**
- Haskell, Lisp, Scala, OCaml, Scala, Python

Programming paradigms

Imperative programming:

- Program = list of statements that change the program state (variables, environment)
- Describe, **how to do something**
- Java, C++, ASM, PHP, Perl, VB, **Python**

Functional programming

- Program = functions, depending only on input (not on program state); order less important
- Describe, **what the program should accomplish**
- Haskell, Lisp, Scala, OCaml, Scala, **Python**

Python

- Imperative:

```
result = []
for num in [3, 4, -7, 0, 2]:
    if (num > 0):
        result += [num**2 - 1]
```

- Functional:

```
result = map(computeVal,
    filter(filterNegatives,
           [3, 4, -7, 0, 2]))
```

```
def computeVal(x):
    return x**2 - 1

def filterNegatives(x):
    return x > 0
```

Python

Universität Zürich^UZH

- Imperative:

```
result = []
for num in [3, 4, -7, 0, 2]:
    if (num > 0):
        result += [num**2 - 1]
```

- Functional:

```
result = map(lambda x: x**2 - 1,
    filter(lambda x: x > 0,
           [3, 4, -7, 0, 2]))
```

Typing

- Java, C++: static, strict

```
int x = 3;
String y = "yo";
```

- PERL, PHP: dynamic, weak

```
my $x;
$x = "3" + 4;   # $x = 7
$x = "3" . 4;   # $x = "34"
```

- Python: dynamic, strict -- “Duck typing”

```
x = 5
x = "hi"
y = x + 3 #error, string + int not allowed
```

Duck typing

Duck test:

- When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.

Duck typing

In programming:

- doesn't matter if the value is of the precise expected type
- as long as it has the expected properties

- "EAFP": easier to ask forgiveness than permission

Duck typing

In programming:

- doesn't matter if the value is of the precise expected type
- as long as it has the expected properties

- "EAFP": easier to ask forgiveness than permission
  - do not test value for type (`type() is`/`isinstance`)
  - try performing the required operation with the value
  - handle any possible errors

Duck typing

```
def average(numList):
    sum = 0
    for num in numList:
        sum = sum + num

    return sum / len(numList)


print average([1, 5, 2, 4])    #ok
print average([5, 2, "x"])
#TypeError: unsupported operand type(s) for +: 'int' and 'str'
print average(3)
#TypeError: 'int' object is not iterable
print average("string of misfortunes")
#TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Exceptions

```
def average(numList):
    sum = 0
    for num in numList: # assume numList is a list
        sum = sum + num  # assume num is numeric

    return sum / len(numList)
```

Exceptions

```
def average(numList):
    sum = 0
    for num in numList: # assume numList is a list
        sum = sum + num  # assume num is numeric

    return sum / len(numList) # assume len(numList) is not 0
```

Exceptions

```
def average(numList):
    sum = 0

    for num in numList: # assume numList is a list

        sum = sum + num  # assume num is numeric



    return sum / len(numList) # assume len(numList) is not 0
```

Exceptions

```
def average(numList):
    sum = 0

    for num in numList: # assume numList is a list
        try:
            sum = sum + num
        except TypeError:
            print ‘skipping non-numeric list member’



    return sum / len(numList) # assume len(numList) is not 0
```

Exceptions

```
def average(numList):
    sum = 0
    try:
        for num in numList:
            try:
                sum = sum + num
            except TypeError:
                print ‘skipping non-numeric list member’
    except TypeError:
        print ‘non-list given’
    else:
        return sum / len(numList)# assume len(numList) is not 0
```

Exceptions

```
def average(numList):
    sum = 0
    try:
        for num in numList:
            try:
                sum = sum + num
            except TypeError:
                print ‘skipping non-numeric list member’
    except TypeError:
        print ‘non-list given’
    else:
        try:
            return sum / len(numList)
        except ZeroDivisionError:
            print ‘list empty’
```

Exceptions

```
try:
    ...
    ...
except ExceptionClass:
    ...
except OtherExceptionClass as excVar:
    ... (raise excVar)
except:
    ...
else:
    ...
finally:
    ...
```

Exceptions

Universität Zürich^UZH

- You can raise them yourself:

```
if (... data is not good):
    raise Exception
```

or

```
if (... data is not good):
    raise Exception('The data has been very, very naughty')
```

Errors

- A (programming/natural/formal/...) language is a set of “allowed” meaningful expressions

- If a program code does not conform to the Python standards, the interpreter will complain about it – **meaningfully**

  The interpreter does its best to tell you, where and what type of error there is
- **Read the error messages!**

  They are frequently hard to understand – Google is your friend!

Error types

- Compile-time errors

```
print x
```

- Run-time errors

```
x = 0
print 1/x
```

- Logical errors

```
x = 5
print x/2
```

Errors and Exceptions

http://docs.python.org/tutorial/errors.html

Coding style

```
a=raw_input()
b=raw_input()
m=[]
for i in range(0,len(a)+1):
    m+=[[]]
    for j in range(0,len(b)+1):
        m[i]+=[0]
        if (i>0 and j>0):
            m[i][j]=min(m[i-1][j]+1,
                m[i][j-1]+1,m[i-1][j-1]+
                (0 if (a[i-1]==b[j-1]) else 1))
print m[len(a)][len(b)]
```

Coding style

```
a=raw_input()
b=raw_input()

m=[]

for i in range(0,len(a)+1):
    m+=[[]]

    for j in range(0,len(b)+1):
        m[i]+=[0]

        if (i>0 and j>0):
            m[i][j]=min(m[i-1][j]+1,
                m[i][j-1]+1,m[i-1][j-1]+
                (0 if (a[i-1]==b[j-1]) else 1))

print m[len(a)][len(b)]
```

Coding style

```
a = raw_input()
b = raw_input()

m = []

for i in range(0, len(a) + 1):
    m += [[]]

    for j in range(0, len(b) + 1):
        m[i] += [0]

        if (i > 0 and j > 0):
            m[i][j] = min(m[i - 1][j] + 1, m[i][j - 1] + 1,
                m[i - 1][j - 1] +
                    (0 if (a[i - 1] == b[j - 1]) else 1))

print m[len(a)][len(b)]
```

Coding style

```
#input a and b
a = raw_input()
b = raw_input()

m = []

#i goes from 0 to len(a)
for i in range(0, len(a) + 1):
     #add empty row
     m += [[]]

     #j goes from 0 to len(b)
     for j in range(0, len(b) + 1):
          #add empty cell
          m[i] += [0]
...

#print result
print m[len(a)][len(b)]
```

Coding style

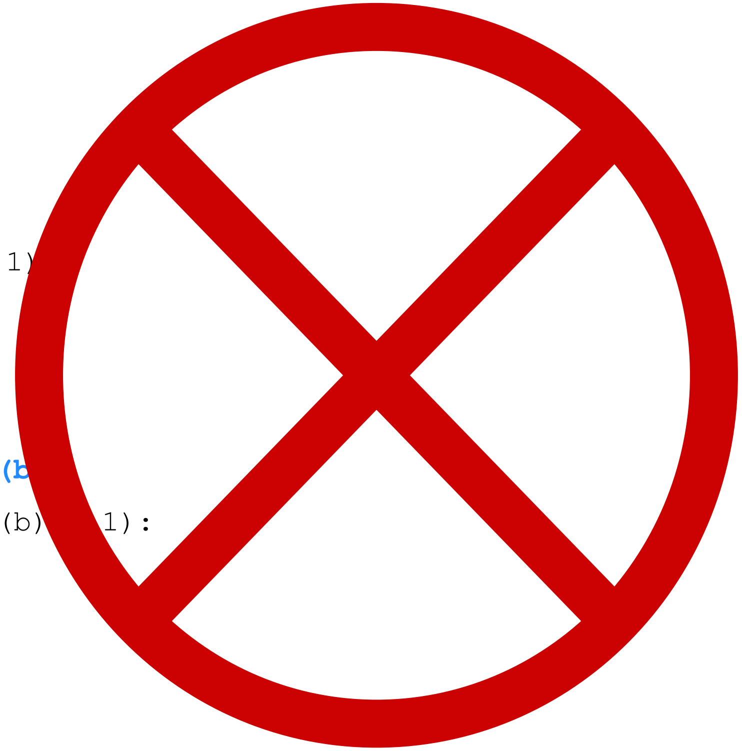
```
#input a and b
a = raw_input()
b = raw_input()

m = []

#i goes from 0 to len(a)
for i in range(0, len(a) + 1)
     #add empty row
     m += [[]]

     #j goes from 0 to len(b
     for j in range(0, len(b)   1):
          #add empty cell
          m[i] += [0]
...

#print result
print m[len(a)][len(b)]
```



Coding style

```
#let the user enter two strings
a = raw_input()
b = raw_input()

#initialize a matrix for computing the Levenshtein distance
m = []

#go through every cell of the matrix starting with the 1st string
for i in range(0, len(a) + 1):
    #add an empty row to the matrix
    m += [[]]

    #go through the 2nd string
    for j in range(0, len(b) + 1):
        #add an empty cell to the current row
        m[i] += [0]
...
#get the total Levenshtein distance from the bottom-right cell
print m[len(a)][len(b)]
```

Coding style

- naming variables, functions, classes, modules
  - descriptive variable names
  - cmpTokTxtLen **vs** compareTokenizedTextLength
  - NewTmpClassDef3 **vs** SntPnktTok **vs** SentPunktTokenizer
- spacing (lines, operators, variables, functions)
  - sparse is better than dense

Coding style

- aim: improve code readability
- main rule: be consistent
- see
    - style guide: http://www.python.org/dev/peps/pep-0008/
    - the Zen of Python

      http://www.python.org/dev/peps/pep-0020/

      or `>>> import this`

Documentation

- Inline documentation = code comments
  - "Non-transparent" code
  - workarounds (read: ugly hacks)
  - TODO, FIXME, etc.

Documentation

- Module/Function documentation
  - general functionality
    - not implementation details
    - those go into inline documentation, if necessary
  - functions, classes, modules
    - functionality
    - expected parameters and their types
    - return values
    - thrown exceptions
    - ways of calling or using it

Documentation

```
def retrieveArticle(url, asNltkText = False):
    rawHtmlCode = unicode(urlopen(url).read(), "utf-8")
...
```

Documentation

```
def retrieveArticle(url, asNltkText = False):
    """Import and clean an article text
    from the web, based on its URL

    @param url: the URL of the article
    @param asNltkText: if True, function returns
        an NLTK.Text object as a result;
        otherwise a list of sentences is returned
    """

    rawHtmlCode = unicode(urlopen(url).read(), "utf-8")
...
```

Documentation

```
def retrieveArticle(url, asNltkText = False):
    """Import and clean an article text
    from the web, based on its URL

    @param url: the URL of the article
    @param asNltkText: if True, function returns
        an NLTK.Text object as a result;
        otherwise a list of sentences is returned
    """

    rawHtmlCode = unicode(urlopen(url).read(), "utf-8")
...

>>> import mymodule
>>> help(mymodule)
>>> help(mymodule.retrieveArticle)
```

To summarize

- Python is a nice programming language
- Read the documentation
- Become friends with error messages
- Try to relax and enjoy :)

Bonus: TextBlob

- like NLTK, only simpler

```
from textblob import TextBlob

text = "The titular threat of The Blob has always..."

blob = TextBlob(text)

blob.tags           # [('The', 'DT'), ('titular', 'JJ'),
                    #  ('threat', 'NN'), ('of', 'IN'), ...]

blob.noun_phrases   # WordList(['titular threat', 'blob',
                    #             'ultimate movie monster',
                    #             'amoeba-like mass', ...])

for sentence in blob.sentences:
    print(sentence.sentiment.polarity)
# 0.060
# -0.341

blob.translate(to="ru")  # 'Титульная угроза...'
```

That's it

Multilayer Perceptron

- Neuron output = synaptic activation values weighted by the synapse weights:

$f(x) = g(\mathbf{w}^T\mathbf{x})$

- Error function

$E = \Sigma_i (y_i - f(x_i))^2$

- Learning the synaptic weights:

$\Delta\mathbf{w} = \partial E / \partial\mathbf{w}$

- Iterative forward-backward passes to learn