text_mining_tutorial01

October 16, 2018

0.1 Lab class organization:

- Lab announcements can be found on the course page
- Contact information: Milad Alshomary (milad.alshomary@uni-paderborn.de)
- In total we will do 6 assignments. One assignment sheet every two weeks
  - Assignments can be found under the course page
  - Submit your assignments via email to milad.alshomary@uni-paderborn.de.
    –tm-lab.zip |____ written_part.pdf |____ programming_part.ipynb
- Prgoramming language: Python

0.1.1 Why Python:

- **Easy to learn**: Popular programming language for indroductory courses in computer science.
- **Productivity**: A lot done in a few lines of Python code
- **Interactive** programming language.
- **Libraries for Natural language processing (NLP) tasks**: tokenization, stemming, tagging, parsing, and text classification.

0.2 Introduction to python

Installing Python:

- On linux, most likely you have it. Otherwise:

```
xx install python3 # xx is (apt-get, dnf, ...) based on you linux distribution
```

- On Windows, download the installer
- Test if python working:

```
python3 --version
```

- Doesn't work? google it or simply ask us for help!

Installing Jupyter:

- For writing code and presenting it along with text in an easy and interactive way!
- Assignments are given as python notebooks templates.
- Install Jupyter via pip command:

```
pip install jupyter
```

- Run Jupyter:

```
jupyter notebook
```

- Access the notebook on under localhost:8888

0.2.1 Basic operations:

```
In [1]: print("hello students!!")
```

```
hello students!!
```

```
In [2]: import math

        # square root of a number
        sq = math.sqrt(9)
        #print
        print(sq)
```

```
3.0
```

```
In [3]: print(dir(math))
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh',
```

```
In [4]: math.log(1)
```

```
Out[4]: 0.0
```

```
In [5]: help(math.log)
```

```
Help on built-in function log in module math:

log(...)
    log(x[, base])

    Return the logarithm of x to the given base.
    If the base not specified, returns the natural logarithm (base e) of x.
```

0.2.2 Variables

- Declaring variables
- Python is typeless programming lagnuage

```
In [6]: var1 = "Hello Python world!"
        var2 = 10
        var3 = 6.8

In [7]: type(var1)

Out[7]: str

In [8]: x = var2 + var3

        print(x)

16.8
```

0.2.3 Operators

```
In [9]: print(1 + 2, 1 - 2, 1 * 2, 1 / 2)

3 -1 2 0.5


In [10]: print(True and False, False and False, True or False, not False)

False False True True
```

0.2.4 Strings

- Define a string.
- A string is sequence of characters.
- Operations on strings (concatenate, contains, title, . . .)

```
In [38]: s1 = "first name "
         s2 = "last name "

         # concatenation of two strings
         print( s1 + s2)

         # String is an array of characters..
         print(s1[0])

         # Length of a string
         print(len(s1))
```

```
        # Return first occurance in a string
        print(s2.find('name'))

        #Return the index of first occurance of letter 'e'
        print(s2.index('m'))
```

```
first name last name
f
11
5
7
```

```
In [12]: print(dir(s1))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__for
```

0.2.5 Lists and Tuples:

- Constructing a list.
- Operations on lists (concatenation, find element, length, sort, reverse, slicing . . .)
- Tuples are basically lists that can never be changed.
- Loops over lists & Tuples

```
In [13]: students = ['bernice', 'aaron', 'cody']
```

```
        print(type(students))

        # Length of the list
        print(len(students))

        #printing content of the list
        print(students)
```

```
<class 'list'>
3
['bernice', 'aaron', 'cody']
```

```
In [14]: # Accessing elements in the list
        print(students[0])

        # Slicing a list..
        print(students[0:2])

        # Adding element to string
        students.append('new student')
```

```
print(students[-1])

#Concatenating two lists
s1 = [1, 2, 3]
s2 = [4,5]

print(s1 + s2)
```

```
bernice
['bernice', 'aaron']
new student
[1, 2, 3, 4, 5]
```

```
In [15]: print(dir(s1))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '
```

Tuples: The immutable lists.

```
In [9]: tuple1 = ('RED', 'GREEN', 'BLUE')

print(tuple1[0])

#changing element of tuple is not allowed because its immutable object
tuple1[0] = 'R'
```

```
RED
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-9-2db7eb818fd1> in <module>()
  4
  5 #changing element of tuple is not allowed because its immutable object
----> 6 tuple1[0] = 'R'


TypeError: 'tuple' object does not support item assignment
```

0.2.6 Dictionaries

- Defining dictionaries.
- Operations on dictionaries.

```
In [16]: params = {"parameter1" : 1.0,
                   "parameter2" : 2.0,
                   "parameter3" : 3.0,}

         print(type(params))
         print(params)
```

```
<class 'dict'>
{'parameter1': 1.0, 'parameter2': 2.0, 'parameter3': 3.0}
```

```
In [17]: params["parameter1"] = "A"
         params["parameter2"] = "B"

         # add a new entry
         params["parameter4"] = "D"

         print("parameter1 = " + str(params["parameter1"]))
         print("parameter2 = " + str(params["parameter2"]))
         print("parameter3 = " + str(params["parameter3"]))
         print("parameter4 = " + str(params["parameter4"]))
```

```
parameter1 = A
parameter2 = B
parameter3 = 3.0
parameter4 = D
```

0.2.7 Control Flow

- Using while and if statments

```
In [18]: statement1 = False
         statement2 = False

         if statement1:
             print("statement1 is True")
         elif statement2:
             print("statement2 is True")
         else:
             print("statement1 and statement2 are False")
```

```
statement1 and statement2 are False
```

```
In [20]: mylist = ["scientific", "computing", "with", "python"]

         for word in mylist:
             print(word)
```

```
scientific
computing
with
python
```

```
In [21]: i = 0
         while i < 5:
             print(i)
             i = i + 1
         print("done")
```

```
0
1
2
3
4
done
```

0.2.8 Functions:

- Defining a function..
- Calling a function..
- Anonymous (lambda) functions for functional programming..

```
In [22]: # include a docstring
         def func(s):
             """
             Print a string 's' and tell how many characters it has
             """

             print(s + " has " + str(len(s)) + " characters")
```

```
In [23]: help(func)
```

```
Help on function func in module __main__:

func(s)
    Print a string 's' and tell how many characters it has
```

```
In [24]: func('hello!')
```

```
hello! has 6 characters
```

lambda operator or lambda function is used for creating small, one-time and anonymous function objects in Python.

```
In [41]: # Defining anonymous functions
         # Functions are like variables can be passed to other functions..
         def get_square_func():
             #Defined an anonymous function and save it in variable f1
             f1 = lambda x: x**2
             return f1

         squre_fun = get_square_func()
         squre_fun(2)
```

```
Out[41]: 4
```

0.2.9 Classes:

- Defining a class
- Initializing an object from a class..
- calling a method on the class

```
In [44]: class Point:
             def __init__(self, x, y):
                 self.x = x
                 self.y = y

             def translate(self, dx, dy):
                 self.x += dx
                 self.y += dy

             def __str__(self):
                 return("Point at [%f, %f]" % (self.x, self.y))
```

```
In [45]: p1 = Point(0, 0.5)
         print(p1)
```

```
Point at [0.000000, 0.500000]
```

```
In [46]: p1.translate(0.25, 1.5)

         print(p1)
```

```
Point at [0.250000, 2.000000]
```

0.2.10 Exercise:

Given a list of tuples (city, temperature in celsius) `cities` we want to apply two operations:

- Transform the degrees from celsius to fahrenheit
- Remove cities that have fahrenheit less than 50

map functions expects a function object and any number of iterables like list, dictionary, etc. It executes the function_object for each element in the sequence and returns a list of the elements modified by the function object

filter function expects two arguments, function_object and an iterable. function_object returns a boolean value. function_object is called for each element of the iterable and filter returns only those element for which the function_object returns true.

```
In [4]: cities = [('Berlin', 14), ('Leipzig', 8), ('Paderborn', 17)]

        # 1. Method one:
        def fahrenheit(city):
            city_name, temp = city
            temp_fahrenheit = ((float(9)/5)*temp + 32)
            return (city_name, temp_fahrenheit)
        def less_than_50(city):
            return city[1] >= 50

        cities = map(fahrenheit, cities)
        cities = filter(less_than_50, cities)
        print(list(cities))
```

```
[('Berlin', 57.2), ('Paderborn', 62.6)]
```

```
In [6]: cities = [('Berlin', 14), ('Leipzig', 8), ('Paderborn', 17)]

        # 2. Method two:
        #map degree from celcius to fahrenheit
        cities = map(lambda x: (x[0], (float(9)/5) * x[1] + 32), cities)
        #filter out cities with fahrenheit less than 50
        cities = filter(lambda x: x[1] >= 50, cities)
        print(list(cities))
```

```
[('Berlin', 57.2), ('Paderborn', 62.6)]
```