a0_python_intro

February 3, 2021

1 [COM6513] Introduction to Python for NLP

1.1 Instructor: Nikos Aletras

The goal of this session (**not assessed**) is to introduce you to Python 3, Jupyter notebooks and main "data science" packages that we will use throughout the course. Specifically, you will be presented to NumPy, SciPy, Pandas and Matplotlib libraries which are the backbone for data manipulation and visualisation, and scientific computing in Python. You will also be presented with essential/basic approaches to text pre-processing.

1.2 Learning objectives

By the end of this session, you will be able to:

- Setup, configure and run Python Jupyter notebooks.
- understand Python basic syntax and know where to find further help (e.g. inline help in notebooks).
- remember basic text processing tricks
- use the basic Numpy, SciPy, Pandas and Matplotlib functionalities.
- have a good overview of popular packages useful for scientific computing.

1.3 Practicalities

It is strongly recommended to use Ubuntu Linux for the labs (how to login to Ubuntu). On Ubuntu, use `/apps/anaconda/bin/python` to run python from the Anaconda distribution. You should also make sure that when using your own machine and operating system (e.g. MacOS, Linux, Windows) any Python package versions should be identical to the ones we use on the University PCs (check the Anaconda version) to avoid any issues in executing your code by the markers (that could result into losing marks). You could also work on Windows machines by using python from the Anaconda prompt but it is not recommended.

In general, for developing in Python, you could use standard IDEs like PyCharm. For all the assignments we will be using IPython and Jupyter notebooks.

1.4 Python for Natural Language Processing/Machine Learning/Data Science

1.4.1 Pros

- Open source - free to install

- Large scientific community (all new cool machine learning libraries are mostly introduced in Python)
- Easy to learn
- Widely used in industry for building data science products
- Allows interfacing with C/C++ via the Cython library (http://cython.org/)
- Easy GPU computing via CUDA + ML libraries
- Parallelisation with OpenMP

1.4.2 Cons

- Interpreted (not-compiled) language.
- Python code might be slower compared to C/C++
- Not ideal for multithread applications; the interpreter prevents from executing one Python bytecode at a time, Global Interpreter Lock (GIL)

1.5 Essential Scientific Python Libraries that we will use in the course

1.5.1 NumPy

Numerical Python (http://www.numpy.org/) is the foundational package for scientific computing in Python. Most of the other scientific computing libraries are built on top of NumPy.

- Provides a fast and efficient multidimensional array object *ndarray*
- Computations and operations between arrays
- Serialisation of array objects to disk
- Linear algebra operations and other mathematical operations
- Integrating C, C++ and Fortran code to Python (e.g. BLAS and Lapack libraries)

1.5.2 SciPy

SciPy (https://www.scipy.org/) is scientific computing and technical computing library. SciPy contains modules for optimization, linear algebra, integration, interpolation.

1.5.3 Pandas

Pandas (https://pandas.pydata.org/) is a library that provides richer data structures compared to NumPy called *DataFrames*. DataFrames are similar to the ones used in R (*data.frame*) and allow sophisticated indexing functionallity for reshaping, slicing and dicing, aggregating data sets.

1.5.4 Matplotlib

Matplotlib (https://matplotlib.org/) is the basic plotting library in Python.

1.5.5 Seaborn

Seaborn (https://seaborn.pydata.org/) is a visualization library based on matplotlib. It makes it easier for drawing graphs by providing a high-level interface to matplotlib. It can also directly plot Pandas dataframes.

1.5.6 IPython

IPython (Interactive Python https://ipython.org/) provides a platform for interactive computing (shells) that offers introspection, rich media, shell syntax, tab completion and history.

1.5.7 Jupyter Notebook

Jupyter (https://jupyter.org/) is a web-based application that allows to crerate documents (i.e. notebooks) that contain executable code, rich media and markdown inter alia. Jupyter interacts with Python via an IPython kernel.

1.5.8 Anaconda

Anaconda (https://www.anaconda.com/distribution/) is a free and open-source Python (and R) distribution that comes bundled with all the essential (and many more) packages for data sciene and machine learning. Anaconda also offers management of packages, dependencies and environments.

1.5.9 Other popular NLP and ML Libraries

- SpaCy (https://spacy.io/) is an open-source library for Natural Language Processing.
- NLTK (https://www.nltk.org/) is a package that provides text processing libraries for classification, tokenization, stemming, tagging, parsing etc..
- Scikit-learn (https://scikit-learn.org) is an open-source machine learning library.
- Tensorflow (https://www.tensorflow.org/) and PyTorch (https://pytorch.org/) are popular open-source libraries for implementing and training neural network architectures.
- Keras (https://keras.io/) is a library that provides high level abstractions for implementing neural network architectures (built on top of Tensorflow)

Note that you are not allowed (unless it is explicitly specified) to use any of these six libraries in the assignments

1.6 Installation and Setup

Thanks to Anaconda, all above packages come in one bundle so if you use your own machine, you just need to install Anaconda for Python 3 following the instructions here: http://docs.anaconda.com/anaconda/install/ (**already installed on University's machines**)

Note that Anaconda supports Windows, MacOS and Linux. Choose your OS and follow the instructions!

You can load this notebook by running:

- `$ /apps/anaconda/bin/jupyter notebook a0_python_intro.ipynb` on uni machines

or

- `$ jupyter notebook a0_python_intro.ipynb` on your own machine

1.7 The Basics

You can skip this section if you are already familiar with basic Python functionality.

1.7.1 Help in Jupyter

```
In [ ]: ?len

        #get help for a property or a method on Jupyter/Ipython:
        # ? followed by the method or property
```

```
In [ ]: str.<TAB>

        #do not run this cell, if you hit tab after typing `str.` you will get a
        #list with all available properties and methods of an object
```

1.7.2 Basic Arithmetic

```
In [ ]: 1+1
```

```
In [ ]: 1/3
```

1.7.3 Variables

```
In [ ]: x = 1+1
```

```
In [ ]: x
```

Note that Python 3 supports dynamic typing:

```
In [ ]: y = 5.0
        y = True
        y = 'data'
        y
```

1.7.4 Whitespace Formatting

Python uses identation to delimit blocks of code.

```
In [ ]: for i in range(3):
            print(i+10)
```

Identation could be either 5 spaces or a tab, however it should be consistent throughout the code:

```
In [ ]: for i in range(3):
            print(i+10)
                print(i-10)
```

Whitespace is ignored inside paretheses and brackets:

```
In [ ]: x = [
            [1,2,3],
            [4,5,6]
        ]
        x
```

You can use a backslash to indicate that a statement continues to next line:

```
In [ ]: 1 + \
        1
```

1.7.5 Modules

Not all available functionality is loaded by default, however we can load build-in or third-party modules (packages).

```
In [ ]: import random
        random.gauss(0,10)
```

```
In [ ]: import numpy as np
        np.random.normal([0, 0], [1,10], size=[5,2])
```

1.7.6 Functions

Functions take zero or more inputs and return an output

```
In [ ]: def x_squared(x):
            return x**2 #x power of 2

        # using positional arguments
        def x_squared_new(x=3):
            return x**2 #x power of 2

        a = x_squared(2)
        b = x_squared_new()
        a, b
```

1.7.7 Strings

```
In [ ]: a = "natural"
        b = 'language'
        c = a+' '+b # concatenate strings
        c, len(c) # string length
```

1.7.8 Lists

```
In [ ]: l1 = [1,2,3,4,5] # list with 3 integers
        l2 = [1,'a'] # list with one int and one char
        l2[1] = 2 # update the value of the second element of the list
        l3 = [l1, l2, []] #list of lists
```

```
In [ ]: # use : to slice the list
        l1[:] # [1,2,3,4,5] - copy of l1
        l1[1:4] # [2,3,4]
        l1[:2] # [1,2]
        l1[2:] # [3,4,5]
```

```
In [ ]: l1[-1] # choose the last element
        l1[-3:] # last three elements
```

```
In [ ]: # check list membership
        1 in [1,2] #True
        1 in [2,3] #False
```

```
In [ ]: # list concatenation
        x = [1,2,3]
        y = [4]
        x+y
```

```
In [ ]: x.append(5) # append an item at the end of the list
        x
```

```
In [ ]: len(x) # length of the list, that's 4
```

List comprehensions

```
In [ ]: [x for x in range(5)]
```

```
In [ ]: [x+1 for x in range(50,100,10)]
```

1.7.9 Tuples

Similar to list but no element modifications are allowed

```
In [ ]: a = (1,2,3) +(4,5)
```

```
In [ ]: a
```

```
In [ ]: a[0] = 6
```

1.7.10 Dictionaries

Data structures that associate keys to values

```
In [ ]: d = {} #empty dictionary
        d = dict() #empty dictionary
```

```
In [ ]: d['Joe'] = 10
        d['Mary'] = 30
        d
```

```
In [ ]: d.keys()
```

```
In [ ]: d.values()
```

```
In [ ]: d.items()
```

```
In [ ]: 'Mary' in d
```

```
In [ ]: len(d) # size of the dictionary
```

```
In [ ]: del d['Mary'] # delete key
```

```
In [ ]: d
```

1.7.11 Sets

```
In [ ]: s = set()
        s.add(1)
        s.add(2)
        s
```

```
In [ ]: v = set([4,5,5,5,4,4,4,4,2])
        v
```

```
In [ ]: s & v # intersection
```

```
In [ ]: s | v # union
```

1.7.12 Control Flow

```
In [ ]: if 1>2:
            print("Yes!")
        else:
            print('No')
```

```
In [ ]: i=5
        if i<0:
            i=10
        elif i>0 and i<=5:
            i=20
        else:
            i=30
        i
```

```
In [ ]: x = 0
        while x < 2:
            print(x, "is less than 2")
            x+=1
```

```
In [ ]: for i in range(100):
            print('Sure!')
            break
```

1.7.13 Randomness

```
In [ ]: import random
```

```
In [ ]: #random.random() produces numbers uniformly between 0 and 1
        v = [random.random() for _ in range(10)]
```

```
In [ ]: v
```

```
In [ ]: random.seed(123) # set the random seed to get reproducible results!
        random.random()
```

```
In [ ]: random.seed(123)
        random.random()
```

```
In [ ]: random.uniform(-1, 1) # set the boundaries to sample uniformly
```

1.7.14 Object-Oriented Programming

```
In [ ]: class MLmodel:

            # member functions for a linear regression model
            # y = input * w (input: vector, w: weights)

            def __init__(self, w=None, params_size=3):
                # constructor to initialise a model
                # self. w is the weight vector (parameters)
                # if w is not set, we initialise it randomly
                # note that self.w  and w are different!
                if w == None:
                    self.w = [random.uniform(-0.1,0.1) for _ in range(params_size)]
                else:
                    self.w = w

            def predict(self,X):
                return sum([X[i]*self.w[i] for i in range(len(self.w))])

            def train(self, X, Y):
                #not implemented
                return None
```

```
In [ ]: clf = MLmodel()
```

```
In [ ]: x = [2., 5., 0]
        clf.predict(x)
```

```
In [ ]: clf = MLmodel(params_size=5)
        x = [2., 5., 0, 18, 9]
        clf.predict(x)
```

1.8 Text Processing Basics

```
In [ ]: # raw text
        d = """
            the cat sat on the mat
            """
```

```
In [ ]: d_tok = d.split() # simple whitespace tokenisation
        d_tok
```

```
In [ ]: vocab = set(d_tok) # obtain a vocabulary using a set
        vocab
```

```
In [ ]: #create a vocab_id to word dictionary
        id2word = enumerate(vocab)
        id2word = dict(id2word)
        id2word
```

Can you generate a word2id dictionary? E.g. {'on':0, 'the':2 ...}

```
In [ ]:
```

1.8.1 Regular expressions

```
In [ ]: import re

        numRE = re.compile('[0-9]+')

        numRE.findall('45 09 dfs 56352 tta& 1')
```

1.8.2 Counters

```
In [ ]: from collections import Counter


        a = Counter(['a', 'foo', 'foo', 'a', 'foo'])
        a
```

1.9 Advanced list handling

```
In [ ]: # create a new list by aligning elements from two lists
        a = list(zip([1,2,3,4], [2,3,4,5,6]))
        print(a)
```

```
In [ ]: list(zip([1,2,3,4], [2,3,4,5,6], [3,4,5,6,7]))
```

```
In [ ]: list(zip(*a)) # unzip a list
```

```
In [ ]: # enumerate elements of a list
        list(enumerate(['a','b','c']))
```

1.10 NumPy

```
In [ ]: import numpy as np
```

1.10.1 Arrays

NumPy array are similar to Python lists, except that every element of an array must be of the same type.

```
In [ ]: a = np.array([1, 2, 3], np.float32)
        type(a)
```

```
In [ ]: # equivalent to range(3) or np.arange(0,3,1)
        # last argument is the step
        np.arange(3)
```

```
In [ ]: a[:2]
```

```
In [ ]: a[0]=80
        a
```

Multidimensional arrays:

```
In [ ]: b = np.array([[1,2,3,4,5],[6,7,8,9,10]])
        b
```

Slicing across dimensions:

```
In [ ]: b[1,:]
```

```
In [ ]: b[:,3]
```

```
In [ ]: b[1:,2:4]
```

The *shape* property returns a tuple with the size of each dimension

```
In [ ]: b.shape
```

The *dtype* property returns the data type of the array

```
In [ ]: b.dtype
```

```
In [ ]: # create a copy of the array with a different data type
        c = b.astype(np.float)
        c
```

Manipulate arrays:

```
In [ ]: c.reshape((5,2)) # change shape, keep the same number of elements
```

```
In [ ]: c.transpose() # transpose an array -- same as c.T
```

```
In [ ]: c.flatten() # flatten an array, resulting to 1-D array
```

```
In [ ]: a = np.array([1,2])
        b = np.array([10,20])
        c = np.array([100, 200])
        # concatenate two or more arrays
        np.concatenate((a, b, c))
```

```
In [ ]: np.vstack((a,b)) # stack vertically
```

```
In [ ]: np.hstack((b,a)) # stack horizontally
```

EXCERCISE Form the 2-D array (without typing it in explicitly):

```
[[1,  6, 11],
 [2,  7, 12],
 [3,  8, 13],
 [4,  9, 14],
 [5, 10, 15]]
```

```
In [ ]: # Type your answer here
```

1.10.2 Array mathematics

```
In [ ]: a = np.array([1,2,3,4,5])
```

```
In [ ]: a*2
```

```
In [ ]: a+4
```

```
In [ ]: a/2
```

```
In [ ]: a**2
```

```
In [ ]: np.sqrt(a)
```

```
In [ ]: np.exp(a)
```

```
In [ ]: a.dot(np.array([1,2,3,4,5])) # dot product
```

```
In [ ]: a = np.array([[1,2],[3,4]])
        b = np.array([[0,1],[2,3]])
```

```
In [ ]: a*b #elementwise multiplication
```

```
In [ ]: a+b
```

1.10.3 Basic Array Operations

```
In [ ]: a = np.array([1,2,3,4,5])
```

```
In [ ]: a.mean()
```

```
In [ ]: a.min()
```

```
In [ ]: a.argmax()
```

```
In [ ]: b = np.array([[0,1],[2,3]])
```

```
In [ ]: b.mean(axis=0)
```

```
In [ ]: b.mean(axis=1)
```

```
In [ ]: c = np.array([100,-1,20,5.6])
        c.sort() # sort an array (inplace)
        c
```

```
In [ ]: a
```

1.10.4 Comparison operators, value testing, item selection

```
In [ ]: a = np.array([1,2,3,4,9])
        b = np.array([4,2,8,5,7])
```

```
In [ ]: a == b
```

```
In [ ]: a >= b
```

```
In [ ]: idx = np.nonzero(a>=b) # indices of the nonzero elements e.g. True
        idx
```

```
In [ ]: np.where(a>=b) # check where a>= b
```

```
In [ ]: a[idx] #select elements in a where a >= b
```

```
In [ ]: a.nonzero() # non-zero elements
```

```
In [ ]: np.isnan(a) # check for NaN values
```

```
In [ ]: idx = np.array([3,1,2]) # array of indices
```

```
In [ ]: a[idx] # subset of a containing the elements in idx
```

1.10.5 Vector and matrix mathematics - Basic Linear Algebra

```
In [ ]: a = np.array([[1, 2, 3],[2,3,4],[5,6,7]], np.float)
        b = np.array([0, 1, 1], np.float)
```

```
In [ ]: np.dot(a,b) # dot product
```

```
In [ ]: np.dot(a.T,b)
```

```
In [ ]: np.inner(a, b) # inner product
```

```
In [ ]: np.outer(a,b) # outer product
```

```
In [ ]: np.cross(a, b) #cross product
```

```
In [ ]: np.linalg.det(a) # determinant of a
```

```
In [ ]: vals, vecs = np.linalg.eig(a) # eigenvalues and eigenvectors
        vals, vecs
```

```
In [ ]: b = np.linalg.inv(a) # invert a matrix
        b
```

```
In [ ]: U, s, Vh = np.linalg.svd(a) # Singular Value Decomposition
        U
```

1.11 Scipy

1.11.1 Statistics

```
In [ ]: import numpy as np
        from scipy import stats
```

```
In [ ]: x1 = np.random.uniform(-1,1, size=5)
        x2 = np.random.uniform(-1,1, size=5)

        #  t-test to test whether the mean of two samples are statistical significant.
        stats.ttest_ind(x1, x2)
```

1.11.2 Sparse Matrices

Sometimes our data might be so large that cannot fit memory but contain many zeros that we do not need to store. In that case SciPy provides memory efficient sparse matrix data structures.

```
- csc_matrix: Compressed Sparse Column format

- csr_matrix: Compressed Sparse Row format

- bsr_matrix: Block Sparse Row format

- lil_matrix: List of Lists format
```

```
- dok_matrix: Dictionary of Keys format

- coo_matrix: COOrdinate format (aka IJV, triplet format)

- dia_matrix: DIAgonal format
```

See https://docs.scipy.org/doc/scipy/reference/sparse.html for more details.

```
In [ ]: import numpy as np
        from scipy.sparse import *
```

```
In [ ]: A = csr_matrix([[1, 2, 0], [0, 0, 3], [4, 0, 5]])
        # 5 non-zero elements
        A
```

```
In [ ]: v = np.array([1, 0, -1])
        A.dot(v)
```

1.12 Pandas

```
In [ ]: import pandas as pd
```

1.12.1 Object Creation

Series is a one-dimensional *ndarray* with axis labels.

```
In [ ]: s = pd.Series([1,2,3,np.nan,6,100])
```

```
In [ ]: s
```

```
In [ ]: # series can contain any data type casted to an object
        s = pd.Series([10,True,5,'test',6,8])
```

```
In [ ]: s
```

DataFrame is a Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes.

```
In [ ]: df = pd.DataFrame(np.random.randn(6,4), index=None, columns=list('ABCD'))
```

```
In [ ]: df
```

```
In [ ]: # Specify an index
        df = pd.DataFrame(np.random.randn(6,4),
                          index=pd.date_range('20180101', periods=6),
                          columns=list('ABCD'))
```

```
In [ ]: df
```

1.12.2 Viewing Data

In [ ]:
```
df.head()
```

In [ ]:
```
df.tail(2)
```

In [ ]:
```
df.describe()
```

In [ ]:
```
df.T
```

In [ ]:
```
df.sort_index(axis=1, ascending=False)
```

In [ ]:
```
df.sort_values(by='D')
```

1.12.3 Data Selection

In [ ]:
```
df['A'] # selecting a single column
```

In [ ]:
```
df[0:3] # slicing rows by index
```

In [ ]:
```
df['20180101':'20180102'] # slicing with a labelled index
```

In [ ]:
```
df.loc['20180101'] # selecting by label
```

In [ ]:
```
df.loc[:,['A','D']] # selecting multiple columns
```

In [ ]:
```
df.loc['20180102':'20180105',['C','A']] # row and column slicing
```

In [ ]:
```
df.loc['20180102','A'] # access to scalar values
```

In [ ]:
```
df.iloc[3] # selecting rows using numeric indices
```

In [ ]:
```
# selecting rows and columns using numeric indices
df.iloc[2:4,0:2]
```

In [ ]:
```
df.iloc[1,0] # getting a value
```

In [ ]:
```
df[df['A'] > 0] # boolean indexing
```

In [ ]:
```
df[df > 0]
```

In [ ]:
```
df.iat[0,1] = 0 # setting one value

# setting the values of an entire column using a numpy array
df.loc[:,'C'] = np.ones(len(df))
```

In [ ]:
```
df
```

1.12.4 Handling Missing Data

```
In [ ]: df1 = df[df > -0.5]
        df1
```

```
In [ ]: # drop any rows that have missing data
        df1.dropna(how='any')
```

```
In [ ]: df1.fillna(value=0) # filling missing data
```

1.12.5 Operations

```
In [ ]: df.mean() # Performing a descriptive statistic
```

```
In [ ]: # Performing a descriptive statistic on the axis 1
        df.mean(1)
```

```
In [ ]: df.apply(np.cumsum) # Applying functions to the data
```

```
In [ ]: df['D'].apply(np.exp) # Applying a function to a single column
```

```
In [ ]: df.values # get DataFrame values as a numpy array
```

1.12.6 Grouping

- Splitting the data into groups based on some criteria
- Applying a function to each group independently
- Combining the results into a data structure

```
In [ ]: df = pd.DataFrame({'A' : ['red', 'blue', 'blue', 'red',
                                  'red', 'red', 'blue', 'blue'],
                           'B' : ['red', 'blue', 'green', 'green',
                                  'blue', 'blue', 'red', 'green'],
                           'C' : np.random.randn(8),
                           'D' : np.random.randn(8)})
        df
```

```
In [ ]: # group by A then apply sum (think it as a SQL operation)
        df.groupby('A').sum()
```

```
In [ ]: df.groupby(['A','B']).sum() # hierachical index
```

```
In [ ]: pd.pivot_table(df, values='D', index=['A'], columns=['B'])
```

```
In [ ]:
```

1.12.7 Data I/O

CSV

```
In [ ]: df.to_csv('foo.csv') # write to CSV
```

```
In [ ]: df = pd.read_csv('foo.csv')
```

```
In [ ]: df
```

```
In [ ]: df = pd.read_csv('foo.csv', index_col=0)
```

```
In [ ]: df
```

HDF5

```
In [ ]: df.to_hdf('foo.h5','df')
        pd.read_hdf('foo.h5','df')
```

1.13 Plotting with MatplotLib

```
In [ ]: #allow plotting inline
        %matplotlib inline

        import matplotlib.pyplot as plt
```

```
In [ ]: loss = [4.2, 3.8, 3.7, 3.6, 3.55]
        epochs = [1,2,3,4,5]
```

```
In [ ]: plt.plot(epochs,loss);
```

```
In [ ]: plt.plot(epochs,loss,color='g',
                 linestyle='dashdot',
                 label='validation loss');

        plt.title("loss monitoring")
        plt.xlabel("epochs")
        plt.ylabel("loss");
        plt.legend();
```

Excercise: Plot a figure containing two lines

```
In [ ]:
```

1.14 Wrap Up

In this session, you saw:

- How to setup, configure and run Python Jupyter notebooks.
- How Python differs from other languages; its basic syntax and know where to find further help (e.g. inline help in notebooks).

- Remember basic text processing tricks.
- Use of basic Numpy, SciPy, Pandas and Matplotlib functionalities.

More Practice:

- Go through this Python tutorial and this one ("1. Getting started with Python for science" section) on NumPy, SciPy and matplotlib libriaries.

Extras:

- Jupyter tutorial
- Introduction to Unix/Linux

```
In [ ]:
```