# **Table of Contents**

[Table of Contents 1](#_Toc501604390)

[1. Python Learning 3](#_Toc501604391)

[1.1 Programming, Data Structures AND Algorithms 3](#_Toc501604392)

[1.1.1 GCD - Greatest Common Divisor Algorithm 3](#_Toc501604393)

[1.1.2 Python Arthematic & Boolen 3](#_Toc501604394)

[1.1.3 Python Strings 3](#_Toc501604395)

[1.1.4 Python Lists 4](#_Toc501604396)

[2. Python For Data Science 4](#_Toc501604397)

[2.1 Python FRom Data Camp 4](#_Toc501604398)

[2.1.1 Numpy 4](#_Toc501604399)

[2.1.2 Pandas 6](#_Toc501604400)

[2.1.3 SQL 6](#_Toc501604401)

[2.2 Python Data Exploration 8](#_Toc501604402)

[2.2.1 Boolean Indexing 8](#_Toc501604403)

[2.2.2 Apply Function 8](#_Toc501604404)

[3. IIT Kharagpur - NLP 9](#_Toc501604405)

[3.1 Mayank Code 9](#_Toc501604406)

[3.1.1 Numpy 10](#_Toc501604407)

# Python Learning

## Programming, Data Structures AND Algorithms

### GCD - Greatest Common Divisor Algorithm

gcd(m,n)

1. Find the facotrs of m

2. Find the factors of n

3. Find the largest of factor common to m,n

Algorithm as the property of

1. Description of what to do must be written down in finite way.

2.Terminates after finite number of steps

### Python Arthematic & Boolen

* Python is a interpreter and function needs to defined before execution
* In Numerical operations you have
* 9//4 is 1 and 9%5 is 4 and 3\*\*4 is 81
* For more advanced operation you include math "library"

*from math import \**

* Names inherit their type from their current value.
* Type of a name is not fixed,.
* Unlike languages like C,C++,Java where each name is "declared" in advance with its type.

i=5

j=i/3

i=2\*j

* type(e) returns type of the expression e.
* Not good style to assign values of mixed types to same name.

Names inherit type from currently assigned value, Value have types that determine what operations are allowed.

### Python Strings

* Type string, "str", a sequence of characters.
* A single character is a string of length of 1, no separate type "Char".
* You can have enclose it with single or double quotes or triple Quotes.
* its sequence or list of characters.
* positions 0,1,2..n-1 for a string of length n.
* positions -1,-2... count backwards from end
* Concatenation operator is +
* len(ur\_string) returns the length of string
* s[1:4], starts at 1 and ends at 3, s[i:],s[:j]
* cannot update a string "in place"
* Instead, use slices and concatenation

s=s[0:3] + "p!"

* Note we are modifying, we have created new value for s. Strings are immutable values.
* For list, a single position returns a value, a slice returns a list. where as in string both return string

fac[0] == 'S'

fac[0,1] == 'S'

### Python Lists

* Sequences of values, type need not be uniform.
* Unlike strings list are "in place " updateable.
* len(ur\_list) returns the length of string
* For list, a single position returns a value, a slice returns a list. where as in string both return string

fac[0] == 1

fac[0,1] == [1]

* you can have nested list

nested = [[2, [37]],4,["hello"]]

nested[0] = [2,[37]]

nested[2][0][3] is 'l'

nested[0][1:2] is [[37]]

# Python For Data Science

## Python FRom Data Camp

### Numpy

List are great, very powerful however when it comes mathematical calculation they are inefficient and very cumbersome. To calculate BMI (=weight/Height) python throws error where you need to go each element in list and find the BMI value.

Same operation will be super fast with Numpy array,

*height = [1.73,1.68,1.71,1.89,1.79]*

*weight = [65.4,59.2,63.6,88.4,68.7]*

*import numpy as np*

*np\_height = np.array(height)*

*np\_weight = np.array(weight)*

*bmi = np\_weight / np\_height \*\* 2*

*print bmi*

1. Numpy holds array of single type value, if you try to create numpy with different type then you will have numpy created with single type of string.
2. Numpy is again a type in python like list, dictionary. for example if you '+' two list than it concatenates but in numpy if the type numberic it will add the corresponding elements.
3. Subsetting numpy is pretty much same as list. You can also subset through boolean

*print bmi > 23*

*[False False False True False]*

*print bmi[bmi > 23]*

*[ 24.7473475]*

1. When you type "type(np\_height)" you will have numpy.ndarray, meaning its numpy type with 'n' dimensional array. np\_height is a one dimensional array but you can create 2, 3..7,... dimensional arrays.

*type(np\_height)*

*numpy.ndarray*

1. Creating 2 dimensional array, shape gives the dimension of array 2row,5columns

*np\_2d = np.array([[1.73,1.68,1.71,1.89,1.79],*

*[65.4,59.2,63.6,88.4,68.7]])*

*type(np\_2d)*

*np\_2d.shape*

*(2L, 5L)*

1. Subsetting Numpy array,

*print np\_2d[0]*

*[ 1.73 1.68 1.71 1.89 1.79]*

*print np\_2d[0][3]*

*1.89*

*print np\_2d[0,3]*

*1.89*

*print np\_2d[ : ,1:3]*

*[[ 1.68 1.71]*

*[ 59.2 63.6 ]]*

1. Summary Statistics

*np.mean(np\_2d[0, : ])*

*np.median(np\_2d[0, : ])*

*np.corrcoef(np\_2d[0, : ],np\_2d[1, : ])*

*np.std(np\_2d[0, : ])*

*height = np.round(np.random.normal(1.75,0.20,5000),2)*

*weight = np.round(np.random.normal (60.32,15,5000),2)*

*np\_city = np.column\_stack((height, weight))*

*print np\_city*

1. Creating 2 dimensional array, shape gives the dimension of array 2row,5columns
2. Importing flat files using Numpy

* Numpy arrays are the standard for storing numerical data.
* Essential for other packages like scikit-Learn, You can use

1. loadtxt()

*data = np.loadtxt(filename, delimiter=',',skiprows=1,usecols=[0,2])*

// numpy can handle different datatypes but dataframes will be the way to go.

2. genfromtxt()

### Pandas

Numpy uses structure of nd-array, where as in pandas we have two structures

1. Series, array like structure it can store any type of data but homogenous. Every column is series.
2. DataFrame, collection of series stacked together.

*data = pd.read\_csv(filename,index\_col=0)*

*data.head*

//convert the dataframe to numpy by calling values

*data\_array = data.values*

//select a column

*data["col\_name"]*

*data.col\_name*

// add a column

*data["new\_colname"]=[val1,val2,val3,....valn]*

//derive new column from other

*data["density"]= data["population"]/data["area"]\*1000000*

// toaccess the column you use [] for accessing row we use "Loc"

*data.loc["row\_index"]*

*data.loc["In"]*

// to get one element from the dataframe

*data.loc["row\_index", "col\_name"]*

or

*data.loc["row\_index"]["col\_name"]*

or

*data["capital"].loc["row\_index"]*

### SQL

*from sqlalchemy import create\_engine*

*engine = create\_engine('sqlite:///census\_nyc\_sqlite')*

*connection = engine.connect()*

*stmt = 'select \* from people'*

*result\_proxy = connection.execute(stmt)*

*results = result\_proxy.fetchall()*

# to get the first row

*first\_row = results[0]*

*print(first\_row)*

#to get the columns name use keys method

*print(first\_row.keys())*

#print only one column

print(first\_row.column\_name)

**SQLAlchemy to build queries**

* Provides a Pythonic way to build SQL statements
* Hides difference between backend database types

*from sqlalchemy import Table, MetaData*

*metadata = MetaData()*

*Census = Table('census' , metadata, autoload=true, autoload\_with=engine)*

*stmt = select([census])*

*print(stmt)*

*results = connection.execute(stmt).fetchall()*

## Python Data Exploration

https://www.analyticsvidhya.com/blog/2016/01/12-pandas-techniques-python-data-manipulation/

import pandas as pd

import numpy as np

data = pd.read\_csv("train.csv", index\_col="Loan\_ID")

### Boolean Indexing

What do you do, if you want to filter values of a column based on conditions from another set of columns? For instance, we want a list of all females who are not graduate and got a loan. Boolean indexing can help here. You can use the following code

#Applying per column:

print "Missing values per column:"

print data.apply(num\_missing, axis=0) #axis=0 defines that function is to be applied on each column

### Apply Function

It is one of the commonly used functions for playing with data and creating new variables. Apply returns some value after passing each row/column of a data frame with some function. The function can be both default or user-defined. For instance, here it can be used to find the #missing values in each row and column.

#Create a new function:

def num\_missing(x):

return sum(x.isnull())

#Applying per column:

print "Missing values per column:"

print data.apply(num\_missing, axis=0)

#axis=0 defines that function is to be applied on each column

#Applying per row:

print "\nMissing values per row:"

print data.apply(num\_missing, axis=1).head()

#axis=1 defines that function is to be applied on each row

# IIT Kharagpur - NLP

## Mayank Code

*import nltk*

*from nltk.corpus import brown*

#what tokens are present in the brown Corpus

*tokenList = brown.words()*

*print (tokenList)*

#what is the size of Brown Corpus

*print("Total Token List",len(tokenList))*

#what are the names of this categories

*print(brown.categories())*

#what are the tokens present in adventure category

*adventureTokenList = brown.words(categories='adventure')*

#what is the size of adventure token in Brown Corpus

*print("Total adventure Token List",len(adventureTokenList))*

# to find the unique token list, use "set" function to convert the list into set, in set all the elements are unique

*print("Total vocabulary count",len(set(tokenList)))*

# Compute TTR, the count of unique words by total number of words in vocabulary.

*print("TTR for brown corpus",len(set(tokenList))/len(tokenList)\*1.0)*

# How many times each token appears in the corpus

*tokenCount = nltk.FreqDist(tokenList)*

*print(tokenCount)*

*print(tokenCount.most\_common(50))*

# Other example

*sentence = "This is a basic tutorial on POS tagging."*

*tokenizedSent = nltk.word\_tokenize(sentence)*

*print tokenizedSent*

# Social Network - IIT Roopur

## Week1

### Part 1

#Modeling road network of Indian Cities

import networkx as nx

import matplotlib.pyplot as plt

import random as rn

G = nx.Graph() # undirected graph

#G = nx.DiGraph() # directed graph

city\_set = ['Delhi','Bangalore','Hydrabed','Ahmedabad','Chennai','Kolkata','Surat','Pune','Jaipur']

print city\_set

for each in city\_set:

G.add\_node(each)

cost=[]

value = 100

while (value <= 2000):

cost.append(value)

value=value+100

print cost

#we are going to add edges

while(G.number\_of\_edges()<15):

c1=rn.choice(G.nodes())

c2=rn.choice(G.nodes())

if c1 != c2 and G.has\_edge(c1,c2)==0:

w=rn.choice(cost)

G.add\_edge(c1,c2,weight=w)

# Check if graph is connected

print nx.is\_connected(G)

# find the routes between nodes

for u in G.nodes():

for v in G.nodes():

print u,v,nx.has\_path(G,u,v)

# find the shortest distance

#pos=nx.spectral\_layout(G)

pos=nx.circular\_layout(G)

nx.draw(G,pos)

nx.draw\_networkx\_edge\_labels(G,pos)

plt.show()

### Part 2

#Modeling road network of Indian Cities

import networkx as nx

import matplotlib.pyplot as plt

import random as rn

import sys # for exceptional handling

def create\_network(city\_set,cost,num\_edges):

G = nx.Graph() # undirected graph

#G = nx.DiGraph() # directed graph

for each in city\_set:

G.add\_node(each)

#we are going to add edges

while(G.number\_of\_edges()<num\_edges):

c1=rn.choice(G.nodes())

c2=rn.choice(G.nodes())

if c1 != c2 and G.has\_edge(c1,c2)==0:

w=rn.choice(cost)

G.add\_edge(c1,c2,weight=w)

return G

# define nodes i:e is cities

city\_set = ['Delhi','Bangalore','Hydrabed','Ahmedabad','Chennai','Kolkata','Surat','Pune','Jaipur']

# define costs

cost=[]

value = 100

while (value <= 2000):

cost.append(value)

value=value+100

# create the graph

G = create\_network(city\_set,cost,14)

# what is the distance between two nodes if no edges then exception

'''G = create\_network(city\_set,cost,0)

try:

l=nx.dijkstra\_path\_length(G,var1,var2)

except:

#print 'error'

l=10000

print 'path length = ',l '''

# if the path is not there then represent infinity say 10000

# Check if graph is connected

print nx.is\_connected(G)

var1='Delhi'

var2='Kolkata'

# shortest path between 2 nodes

print nx.dijkstra\_path(G,var1,var2)

# find the shortest length

print nx.dijkstra\_path\_length(G,var1,var2)

#pos=nx.spectral\_layout(G)

pos=nx.circular\_layout(G)

nx.draw(G,pos)

nx.draw\_networkx\_edge\_labels(G,pos)

plt.show()

## Week4

### Part 1

*import networkx as nx*

*import matplotlib.pyplot as plt*

*import random*

*def create\_graph():*

*g = nx.Graph()*

*#g.add\_nodes\_from(range(1,101))*

*for i in range (1,101):*

*g.add\_node(i)*

*return g*

*def visualize(g,labeldict,nodesize):*

#def visualize(g,labeldict,nodesize,color):

#nx.draw(g,labels=labeldict,node\_size=nodesize,node\_color=color)

*nx.draw(g,labels=labeldict,node\_size=nodesize)*

*plt.show()*

*def assign\_bmi(g):*

*for each in g.nodes():*

#add attribute name

*g.node[each]['name'] = random.randint(15,40)*

#add attribute type for foci

*g.node[each]['person'] = 'person'*

# create dictnoary for label

*def get\_labels(g):*

*dict1 = {}*

*for each in g.nodes():*

*dict1[each] = g.node[each]['name']*

*return dict1*

# array of size of different nodes

*def get\_size(g):*

*array1 = []*

*for each in g.nodes():*

*if g.node[each]['type'] == 'person':*

*array1.append(g.node[each]['name']\*20)*

*else:*

*array1.append(1000)*

*return array1*

*def add\_foci\_nodes(g):*

*n=g.number\_of\_nodes()*

*i=n+1*

*foci\_nodes=['gym','eatou','movieclub','karateclub','yogaclub']*

*for j in range (0,5):*

*g.add\_node(i)*

*g.node[i]['name']=foci\_nodes[j]*

*g.node[i]['type']='foci'*

*i=i+1*

*def get\_colors(g):*

*c=[]*

*for each in g.nodes():*

*if g.nodes[each]['type']=='person':*

*c.append('blue')*

*else:*

*c.append('red')*

*return c*

*g = create\_graph()*

*assign\_bmi(g)*

*add\_foci\_nodes(g)*

*labeldict=get\_labels(g)*

#nodesize = get\_size(g)

#color\_array=get\_colors(g)

*visualize(g,labeldict,nodesize)*

#visualize(g,labeldict,nodesize,color)