**Python session 2**

**Python for data science (class 2)**

**-----------------------------------------**

**Bharat sreeram**

**Sankara.deva2016@gmail.com**

**Whatsapp : 6309613028**

**-----------------------------------**

**Environments of python coding:**

**Interactive mode.**

**Script mode.**

**-----------------------------**

**Interactive.**

**Command prompt.**

**line by line (stmt by stmt execution)>**

**From command prompt, once developer gives a statement and**

**Defined object, when he called the statement or object name,**

**Output value will be returned on console.**

**Example:**

**>print("hello")**

**Hello**

**>a=10**

**>b=20**

**>c=a+b**

**>c <enter>**

**30**

**Advantage of interactive mode:**

**------------------------------**

**While developing logics for some programming tasks , initially**

**developer does not have entire blueprint of the program in mind.**

**Developer starts jouney of coding with basic steps.**

**Suppose, he has given statement1 and checks output.**

**Then statement2 and checks output , then he gets some clue what to do**

**In its next step. Similary after completion of 15 statements,**

**He gets clue , what to do in 16th statement.  By using this feature**

**Complex logics can be easily built in interactive way with system.**

**2.script based.**

**All statements will be kept in a file with .py extension.**

**P1.py**

**>python p1.py**

**Generally, we use interactive mode in development phase.**

**We use scripts in production.**

**----------------------------------------**

**Coding rules of python.**

**-----------------------**

**4 rules..**

**Rule1.**

**Each  statement should start with column position 1.**

**print("hello")   #invalid**

**Print("hello")  #valid**

**Rule2.**

**If a line is expecting sub statements, the line should end with**

**colon (:)**

**Ex:**

**A=10**

**B=20**

**If a>b:**

**print(a, " is big")**

**Else:**

**print(b, " is big ")**

**Ex:**

**X = [10,20,30,40]**

**For  i in x:**

**print(i)**

**Rule3:**

**------**

**sub statement should be started in farwarded position**

**To parent statement**

**# invalid**

**If a>b:**

**Print(a, " is big ")**

**Else:**

**Print(b, " is big ")**

**If a>b:**

**print(a, " is big ")**

**Else:**

**print(b, " is big ")**

**----------------------------------**

**A=10**

**B=25**

**C=30**

**If a>b:**

**if a>c:**

**print(a, " is big")**

**else:**

**:**

**:**

**Rule4:**

**All sub statements of a parent , should be in same column position.**

**A=10**

**B=20**

**If a>b:**

**print(a, " is big ")**

**print(b, " is small ")**

**Else:**

**print(b, " is big ")**

**print(a, " is small ")**

**-----------------------------------**

**Rule1.**

**Statement should start with col pos 1.**

**Rule2.**

**If line expecting sub statements, it should end with colon**

**Ex: if, for, while etc**

**Rule3.**

**Sub statement should start in forwarded pos to parent .**

**Rule4.**

**All sub statements should be in same column position.**

**----------------------------------------------------**

**Python data types.**

**---------------------**

**Python supports dynamic data typing.**

**based on assigned values,automatically data type will be constructed**

**To a variable.**

**A=10**

**#a is int**

**B=20.45**

**#b is float**

**C="hello"**

**#c is string**

**Two types .**

**Simple types.**

**Complex types (collection types)**

**Simple:**

**-----------**

**Int**

**Float**

**String --> str**

**Boolean -->bool**

**Name--> string**

**Age --> int**

**Income   --> float**

**Areyoumarried --> bool**

**----------------------**

**Name = "ravi"**

**Or**

**Name = 'ravi'**

**-------------------------------**

**Name = "xyz"**

**<var> = input(<prompting text>)**

**Name = input("name please ")**

**#string**

**Age = int(input("age "))**

**Income =  float( input("enter monthly text "))**

**------------------------------------------**

**Inc = int(float(input("enter monthly income ")))**

**250.75# typed value**

**-------------------------------------**

**A=10**

**B=20.5**

**C = a + b**

**Type(c)**

**#float**

**-------------------------**

**Qualified = true**

**X = false**

**----------------------------------**

**S = "computer"**

**S[0]-->c**

**S[3]-->p**

**Len(s) #8**

**S[len(s)-1] --> s[7] --> r**

**S[-1] --> from last first one. ---> r**

**S[-3] --> from last third one. --> t**

**Slicing:**

**--------**

**S[startindex : endindex]**

**start index includes.**

**end index excludes..**

**S = "computer"**

**S[0:4]--> 0,1,2,3 --> comp**

**S[3:7]--> 3,4,5,6**

**S[1:-1]**

**Session 3:**

**-------------------**

**Topic: python basic collections:**

**List**

**Tuple**

**Dictionary**

**---------------------------**

**List:**

**-------------**

**collection of homogeneous items.**

**mutable object-->read and write**

**( append/update/delete etc)**

**symbol --> []**

**x = [10,20,30,40,50]**

**type(x)**

**list**

**list elememnts can be accessed using index numbers and slicing.**

**index starts from  0**

**x[0]**

**x[-1] --> 50**

**x[-2] --> 40**

**x[1:4] -->1,2,3 --> [20,30,40]**

**x[1:-1]**

**x[start:end]**

**x[:]--> all elements**

**x**

**x[:4]--> if start index missed, it takes from beginning.**

**index  0,1,2,3**

**x[1:]--> if end index is missed, it takes till last.**

**index 1,2,3,4**

**Tuple**

**---------**

**collection of heterogeneous items.**

**Difference between homogeneous and heterogeneous its**

**Qualification --> "btech","mtech","phd","datascience"**

**here each element is qualification name.  (pupose of each element is same)**

**Family --->   "ravi","venky","giri"**

**here  "ravi" is name , "venky" is father name, "giri" is grandfather name.**

**Homogeneous --> purpose of each element is same.**

**Hetrogeneous --> purpose of each element is different.**

**For all homogeneous , use list object.**

**For all heterogeneous, use tuple object.**

**Info = ("ravi",25,"hyderabad","se")**

**-------------------------------------------------**

**Price = [600,780,550,400]**

**Here each element is price. So homogeneous. Price is list here.**

**Product = ("p109","laptop","dell",80000,15)**

**1-->product id**

**2 --> pr name**

**3 --> brand**

**4 --> price**

**5 --> discount.**

**purpose of each element is different, so create it as tuple.**

**How to access tuple elements?**

**same as list.  Using index numbers , we can access tuple elements**

**Ravi, purchased 4 products.**

**each product we need following details:**

**product name, price, quantity details are needed.**

**create a data object.**

**pinfo = [ ('aa',1000,3),**

**('bb',400,7),**

**('dd',800,1),**

**('cc',900,2)]**

**pinfo -->list of tuples**

**-----------------------------------**

**Urinfo = ('giri',22,'hyd',['bt','mt','phd'])**

**------------------------------------------**

**Task:**

**I want maintain your information . Recommend me data types.**

**Name --> str**

**Age --> int**

**Weight --> float**

**Qual --> list**

**Spouse details (wife/husband) -->tuple**

**Kids information: list of tuples**

**[('tony',"male",5),("sony","female",2)]**

**Name='ravi'**

**Age=25**

**Weight = 80.23**

**Qualification = ['bt','mt']**

**Spouse=('rani',21,'delhi')**

**Kids = [('tony',"male",5),("sony","female",2)]**

**------------------------------------------**

**Dictionary:      {}**

**-------------**

**is a collection of key and value pairs.**

**key --> access identity.**

**value ---> key's associated values.**

**Rule:**

**key should be unique, value can be duplicate.**

**When you pass key, value will be retrieved.**

**--------------------------------------------**

**qual = {"btech":78,"mtech":80,"ds":78}**

**Qual['mtech']**

**80**

**Qual['ds']**

**-->78**

**Qual['mba']**

**-->error -->key not found error.**

**Qual = {'btech': ('ou',78,2010),**

**'mtech' : ('jntu',69,2012) }**

**Qual['mtech']**

**("jntu',69,2012)**

**-----------------------------------------------**

**Ex: bank account .**

**Account number as key**

**Balance amount as value.**

**Accinfo = {101:45000,**

**107:0,**

**109,9000,**

**110,8000}**

**When account number is passed as key into dictionary, we get**

**Balance amount.**

**Accinfo[109]**

**# 9000**

**Python session 4**

**Python session 4**

**# practice examples**

**(**[**python session 3**](https://bharatsreeram.blogspot.com/p/python-session-3.html)**)**

**# coding rules.**

**# rule1: statement should be started in column position 1.**

**Print(" hello ")**

**#rule 2. If line expecting substatements, the line should end with colon.**

**#rule 3. The sub statement, should be started in a forwarded position to parent  line**

**A=100**

**B=150**

**If  a>b:**

**print(a , " is big")**

**Else:**

**print(b, " is big ")**

**#rule4.**

**# all sub statements of a parenet  should be in same column position.**

**A = 10**

**B = 20**

**If a>b:**

**print(a, " is big")**

**print(b, " is small")**

**Else:**

**print(b, " is big")**

**print(a, " is small")**

**# python collection examples**

**# list/tuple/dictionary.**

**# list**

**A = []**

**Type(a)**

**# here a is empty list object**

**# creating list with values**

**X = [10,20,30,40,50,60]**

**Print(x)**

**#how to access list elements.**

**# using index numbers and slicing.**

**X[0] # first element**

**# check below output**

**X[2]**

**X[-1]**

**X[-2]**

**X[1:4]**

**X[3:]**

**X[-3:]**

**X[:3]**

**# modifying list values.**

**X[2] = 300**

**X[-2] =  5**

**X**

**# appending values to list**

**X.append(500)**

**Print(x)**

**X.append(450)**

**# merging list objects**

**A = [1,2,3]**

**B =  [9,10,11,5]**

**A+b**

**# appending multiple values to a list**

**X = x + [100,200,700]**

**Or**

**X += [100,200,700]**

**A = [10,20,30,40]**

**A += [9,7,0]**

**Print(a)**

**# aggregated functions on list object**

**Len(x)**

**Sum(x)**

**Max(x)**

**Min(x)**

**Avg = sum(x)/len(x)**

**# transformation : perform some action on each element of a list.**

**A = [10,20,30,40,50]**

**# add 100 to each element.(transformation. )**

**#way1**

**B = []**

**For  i in a:**

**v = i + 100**

**b.append(v)**

**Print(a)**

**Print(b)**

**# way2.**

**C = [i+100 for i in a ]**

**Print(c)**

**# inserting a value in middle of list.**

**A = a[:2]+[90]+a[2:]**

**Name = ["amar",'amala','ankit','ankita']**

**S = 'appple'**

**S.upper()**

**# convert each name into uppercase**

**Newname = [s.upper()  for s in name]**

**Newname**

**# convert first character into uppercase and remaining into lowercase**

**Name = [' amar ','kiran','manoj','venkat ']**

**# way1.**

**Nn = []**

**For n in name:**

**n=n.strip()**

**fc = n[0].upper()**

**rc = n[1:].lower()**

**nn.append(fc+rc)**

**Print(name)**

**Print(nn)**

**# create a function ,**

**# to convert first letter into uppercase**

**# and remaining into lower case  after removing whitespaces(strip)**

**Def  fupper(s):**

**s=s.strip().lower()**

**fc = s[0].upper()**

**rc = s[1:]**

**return fc + rc**

**# way2(use above function for transformation)**

**Un = [ fupper(n) for  n in name]**

**Print(un)**

**Task:**

**Add 1 to 10, 2 to 20, 3 to 30, 4 to 40, 5 to 50 of below list**

**A = [ 10,20,30,40,50]**

**B = [ i+1 for i in range(5)]**

**Ix = 0**

**Res = []**

**For  x in a:**

**res.append(x+b[ix])**

**ix +=1**

**Print(a)**

**Print(b)**

**Print(res)**

**Python session 5**

**Python session 5**

**List -->**

**How to filter elements of a list.**

**X = [10,20,30,40,50,60,90,80,120]**

**Len(x)**

**# want to take x >=50**

**# way1.**

**Y = []**

**For i in x:**

**if i>=50:**

**y.append(i)**

**Print(x)**

**Print(y)**

**# way 2.**

**#  [expression    <for loop >   <if condtion>]**

**Z = [ v  for v in x  if v>=50 ]**

**Print(z)**

**Info = [('ravi','m',25),**

**('rani','f',24),**

**('giri','m',29),**

**('gita','f',30)]**

**#  seperate males and females into seperate ist objets.**

**#way1**

**Male = []**

**Fem = []**

**For  s in info:**

**sex = s[1]**

**if sex=='m':**

**male.append(s)**

**else:**

**fem.append(s)**

**Print(male)**

**Print(fem)**

**# way2.**

**M = [s for s in info  if s[1]=='m']**

**F = [s for s in info  if s[1]=='f']**

**Print("males ", m)**

**Print("females ", f)**

**----------------------------------------------**

**Combination of transformations and filters.**

**Info = [('ravi','m',11,80000),**

**('rani','f',12,90000),**

**('giri','m',13,30000),**

**('gita','f',14,55555),**

**('raj','m',11,45000),**

**('raji','f',12,56000),**

**('ranjith','m',13,25555)]**

**Name -->first letter uppercase, remaining lower case.**

**Sex --> m as male, f as female.**

**Dno -->  11 as 'marketing, 12 as 'hr',13 as 'finance',**

**remaining as 'other'.**

**Salary -->   >=70k --> a**

**50 to 70k --> b**

**30 to 50k --> c**

**<30k ---> d**

**Seperate male and  female profiles.**

**Def fupper(s):**

**s = s.strip().lower()**

**fc = s[0].upper()**

**rc = s[1:]**

**return  fc+rc**

**Def gend(s):**

**s= s.lower()**

**if s=='m':**

**return 'male'**

**else:**

**return 'female'**

**Def dept(d):**

**dname = 'other'**

**if d==11:**

**dname='marketing'**

**elif d==12:**

**dname='hr'**

**elif d==13:**

**dname='finance'**

**return dname**

**Def grade(s):**

**grd = 'd'**

**if s>=70000:**

**grd = 'a'**

**elif s>=50000:**

**grd = 'b'**

**elif s>=30000:**

**grd = 'c'**

**return grd**

**Info = [('ravi','m',11,80000),**

**('rani','f',12,90000),**

**('giri','m',13,30000),**

**('gita','f',14,55555),**

**('raj','m',11,45000),**

**('raji','f',12,56000),**

**('ranjith','m',13,25555)]**

**Expected o/p:**

**[('ravi','male','marketing',80000,'a'),**

**:**

**:**

**:**

**]**

**#way1.**

**Newinfo = []**

**For  r in info:**

**name = fupper(r[0])**

**g = gend(r[1])**

**dn = dept(r[2])**

**sal = r[-1]**

**grd = grade(sal)**

**rec = (name,g,dn,sal,grd)**

**newinfo.append(rec)**

**Print(newinfo)**

**#way2**

**Ninfo = [ (fupper(r[0]),gend(r[1]),dept(r[2]),r[-1],grade(r[-1]))**

**for r in info]**

**[expr for loop  if cond]**

**Males = [ v for v in ninfo if v[1]=='male']**

**Fems = [v for v in ninfo if v[1]=='female']**

**Print("males ", males)**

**Print("\_"\*40)**

**Print("fmales ", fems)**

**Python session 6**

**How to run(iterate) a loop  across multiple list objects.**

**-----------------------------------------------------------**

**A = [10,20,30,40,50]**

**B = [1,2,3,4,5]**

**C = [100,200,300,400,500]**

**# task: corresponding elements should be summed (any arithmetic operationn )**

**# d --> [10+1+100,......]**

**#way1**

**I=0**

**D = []**

**For  v  in a:**

**tot = v + b[i] + c[i]**

**d.append(tot)**

**i+=1**

**Print(a)**

**Print(b)**

**Print(c)**

**Print(d)**

**#zip() ---> corresponding elements will be mapped as tuple object.**

**X = [1,2,3]**

**Y = [7,8,9]**

**Z = [100,200,300]**

**Print( list(zip(x,y,z)))**

**#o/p:  [(1, 7, 100), (2, 8, 200), (3, 9, 300)]**

**#way2.**

**A = [10,20,30,40,50]**

**B = [1,2,3,4,5]**

**C = [100,200,300,400,500]**

**D = []**

**For v in  list(zip(a,b,c)):**

**tot = v[0] + v[1] + v[2]**

**d.append(tot)**

**Print(d)**

**#way3**

**A = [10,20,30,40,50]**

**B = [1,2,3,4,5]**

**C = [100,200,300,400,500]**

**D = []**

**For x,y,z in  list(zip(a,b,c)):**

**tot = x+y+z**

**d.append(tot)**

**Print(d)**

**#way4**

**A = [10,20,30,40,50]**

**B = [1,2,3,4,5]**

**C = [100,200,300,400,500]**

**D = [ x+y+z  for x,y,z in list(zip(a,b,c))]**

**Print(d)**

**#way5**

**A = [10,20,30,40,50]**

**B = [1,2,3,4,5]**

**C = [100,200,300,400,500]**

**D = [sum(v) for v in list(zip(a,b,c))]**

**Print(d)**

**#task:**

**Name = ['ravi','mani','girija','veni']**

**Sex = ['m','m','f','f']**

**Age = [25,35,24,32]**

**# form these 3 list object into list of tuples.**

**Info = list(zip(name,sex,age))**

**Print(info)**

**#task:  seperate below object into 4 seperate list objects**

**Info = [('ravi','m',11,80000),**

**('rani','f',12,90000),**

**('giri','m',13,30000),**

**('gita','f',14,55555),**

**('raj','m',11,45000),**

**('raji','f',12,56000),**

**('ranjith','m',13,25555)]**

**Name = []**

**Sex = []**

**Dno = []**

**Sal = []**

**For n,s,d,sl in info:**

**name.append(n)**

**sex.append(s)**

**dno.append(d)**

**sal.append(sl)**

**Print(name)**

**Print(sex)**

**Print(dno)**

**Print(sal)**

**-------------------------------------------**

**Dictionary:**

**-----------**

**# is a collection of key value pairs.**

**# rules:**

**# key should be unique, value can be duplicated.**

**# when key is passed, value will be returned.**

**Qual = {'btech':2010,'mtech':2012}**

**Qual['btech']**

**#2010**

**Qual['mtech']**

**#2012**

**Qual['phd']**

**# key error**

**# how to insert a pair into dictionary.**

**Qual['phd']=2018**

**# how to update a value of a  key.**

**Qual['btech'] =2009**

**Print(qual)**

**# {'btech':2009, 'mtech':2012, 'phd':2018}**

**# observation:**

**"""**

**during assignment, if given key existed , the associated value will**

**be updated. If given key not existed, a new pair will be inserted**

**into dictionary.**

**"""**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**# bank accounts**

**Acno = {101,102,103}**

**Balance = {30000,10000,5000}**

**# form above list objects into dictionary. Acno as key, balance as value**

**#way1**

**Accs = {}**

**For a,b  in list(zip(acno,balance)):**

**accs[a]=b**

**Print(accs)**

**#way2**

**Accs = dict(zip(acno,balance))**

**Print(accs)**

**# {101: 30000, 102: 10000, 103: 5000}**

**Task:**

**# 101 deposited 5000**

**# 102 withdrawn 2000**

**# 104 joined as new account with 10000 deposit.**

**Accs[101]+= 5000**

**Accs[102]-= 2000**

**Accs[104]=10000**

**Print(accs)**

**# {101: 35000, 102: 8000, 103: 5000, 104: 10000}**

**Ac = 102**

**If accs.get(ac)==none:**

**print("invalid account number ")**

**Else:**

**print(ac, " is valid account and balance is ", accs[ac])**

**<dict>.get()?**

**-------------**

**if given key existed, it returns value.**

**if not existed, it return  " none " object.**

**Python session 7**

**# more on dictionary.**

**Info = [('ravi','m',11,80000),**

**('rani','f',12,90000),**

**('giri','m',13,30000),**

**('gita','f',14,55555),**

**('raj','m',11,45000),**

**('raji','f',12,56000),**

**('ranjith','m',13,25555)]**

**{m:4,f:3}**

**# grouping aggregations .**

**# performing aggregations seperately on each data group.**

**Ex:**

**Task: foreach sex group, what is count.**

**Sql:   select sex,count(\*) from info group by sex;**

**Scnt = {}**

**For v in info:**

**sex = v[1].lower()**

**if scnt.get(sex)==none:**

**scnt[sex]=1**

**else:**

**scnt[sex]+=1**

**Print(scnt)**

**Task: foreach sex group, what is total salary?**

**Sql:  select sex, sum(sal) from info group by sex:**

**Info = [('ravi','m',11,80000),**

**('rani','f',12,90000),**

**('giri','m',13,30000),**

**('gita','f',14,55555),**

**('raj','m',11,45000),**

**('raji','f',12,56000),**

**('ranjith','m',13,25555)]**

**Stot = {}**

**For v in info:**

**s = v[1].lower()**

**sal = v[-1]**

**if stot.get(s)==none:**

**stot[s]=sal**

**else:**

**stot[s]+=sal**

**Print(stot)**

**Task: perform multiple aggregations , on each each data group.**

**# single grouping and multiple aggregations.**

**Sql: select sex, sum(sal), count(), avg(sal),max(sal), min(sal)**

**from info group by sex:**

**Res = {}**

**For v in info:**

**s=v[1]**

**sal = v[-1]**

**if res.get(s)==none:**

**res[s]= [sal]**

**else:**

**res[s].append(sal)**

**Print(res)**

**For x in res:**

**print(x, res[x])**

**Res = {m:[90,80,67,45], f: [45,56,23]}**

**Report = {}**

**For x in res:**

**slist=res[x]**

**tot = sum(slist)**

**cnt = len(slist)**

**avg = tot/cnt**

**mx = max(slist)**

**mn = min(slist)**

**agr = (tot,cnt,avg,mx,mn)**

**report[x]=agr**

**For  k in report:**

**print(k,  report[k])**

**#  multiple grouping columns and single aggregation**

**# select dno,sex,sum(sal) from info group by dno, sex**

**Res = {}**

**For v in info:**

**s = v[1]**

**d = v[2]**

**k = (d,s)**

**sal = v[-1]**

**if res.get(k)==none:**

**res[k]=sal**

**else:**

**res[k]+=sal**

**Print(res)**

**Python session 9**

**Numpy:**

**-------**

**Import numpy as np**

**A = np.array(1,2,3,4,5)**

**# error: the argument should be a tuple or list object or numpy array.**

**B = np.array((1,2,3))**

**Print(b)**

**Print(type(b))**

**C = np.array([1,3,4,5,7])**

**Print(c)**

**#-----------------------**

**# sequence of  numbers**

**X = np.arange(5) # 0 to 4**

**Print(x)**

**# arithmetic operations over numpy array.**

**A = np.array([10,20,30,40])**

**B = a\*5**

**Print(b)**

**# when you perform arithmetic operation on numpy array, the operation**

**Will be done for each element.**

**--------------------------------------------------**

**# creating 2d.**

**Argument should be  tuple of tuple or list of list.**

**A = np.array([[1,2,3],**

**[5,6,7]])**

**# to get dimension number.**

**Np.ndim(a)**

**Or**

**A.ndim**

**# to get shape of array.**

**Np.shape(a)**

**# (2,3)**

**Or**

**A.shape**

**A.shape[0]**

**# rows**

**A.shape[1]**

**# number of cols.**

**# converting single dimension to n dimensions.**

**A = np.arange(30)+1**

**B = a.reshape((5,6))**

**C = a.reshape((6,5))**

**# ravel() --> it flattens multi dimension into single dimension.**

**D = np.ravel(c)**

**D.ndim**

**#1**

**# creating 3 dimensional object (list of list of list).**

**Dim1 --> course style (normal or fasttrack)**

**Dim2 --> course name**

**Dim3 --> training mode (online/classroom)**

**Fee = np.array([**

**[[10,30],**

**[20,40]] ,**

**[[30,60],**

**[40,80]]**

**])**

**Fee.shape**

**(2,2,2)**

**Fee.ravel()**

**# converts 3d into single d.**

**# veg market.**

**# 3 markets. Hyd,del,pune.**

**# we have two shifts. Morning,evening.**

**# 3 vegitables.  Tomato , onion, carrot**

**# each vegitable price varies from day to day (mon to fri).**

**# shape --> (3,2,3,5)**

**Prices = np.arange(90)+100**

**Prices**

**P = prices.reshape((3,2,3,5))**

**Print(p)**

**# what is price of onion @ del,2nd shift,thursday**

**P[1,-1,1,-2]**

**# price of onion, in 1 st shift , in all markets, in all days.**

**P[:,0,1]**

**#**

**What are price of onion and carrot on hyd and del market**

**for 2 shift on wednesday.**

**P[:2,1,1:,2]**

**Python session 10**

**Import numpy as np**

**A = np.array([[1,2,3],**

**[4,5,6]])**

**Len(a)**

**# number of elements higher dimension.**

**A.size**

**# total number of elements in n-dimensional array.**

**Sum(a)**

**# column sum of a matrix.**

**Sum(sum(a))**

**#total of nd-array**

**Np.sum(a)**

**# sum of all elements.**

**Np.sum(a,axis=0)**

**# sum of each column.**

**Np.sum(a,axis=1)**

**#sum of each row.**

**Np.mean(a)**

**A.max()**

**A.min(axis=1)**

**#-------------**

**If artithmetic operation applied between numpy arrays.**

**Corresponding elements will be arithmetically operated.**

**A= np.arange(5)**

**B = np.ones(5)**

**C = np.zeros(5)**

**D = a + b+ c**

**Print(a)**

**Print(b)**

**Print(c)**

**Print(d)**

**# computations on matrices.**

**B = (np.arange(6)+10).reshape(a.shape)**

**Print(a)**

**Print(b)**

**C = a+b**

**Print("\_"\*40)**

**Print(c)**

**A\*b**

**# its element multiplication.**

**#matrix multiplication.**

**M = np.array([[1,4],**

**[2,1],**

**[1,3]])**

**A.shape**

**# (2,3)**

**M.shape**

**# (3,2)**

**# multiplication rule: number of cols of left side, number of rows**

**# of right side should be equal.**

**P = np.dot(a,m)**

**P.shape**

**# (2,2)**

**# matrix transpose.**

**C = np.transpose(b)**

**Print(b.shape)**

**Print(c.shape)**

**N = a.dot(b.t)**

**Print(n)**

**# matrix inversion.**

**Rule: 1. Matrix should be square.**

**Det should not be zero.**

**Q = np.array([[1,2],**

**[3,4]])**

**From  numpy.linalg import inv**

**I = inv(q)**

**Print(q)**

**Print("\*"\*10)**

**Print(i)**

**#---------------------------------**

**# develop statistical functions using numpy**

**Def sd(x):**

**num = ((x - x.mean())\*\*2).sum()**

**den = x.size -1**

**return (num/den)\*\*0.5**

**Def cov(x,y):**

**dx = x - x.mean()**

**dy = y - y.mean()**

**num = sum(dx\*dy)**

**den = x.size - 1**

**return num/den**

**T = np.array([10,12,14,17,15,13])**

**Ice = np.array([20,34,40,52,45,38])**

**Swetter = np.array([60,55,40,30,35,58])**

**Print(" tmpr and ice ", cov(t,ice))**

**Print(" tmpr and swetter sales ", cov(t,swetter))**

**Python session 11**

**More on statistical functions with numpy.**

**-------------------------------------------**

**Sd()**

**Covariance --> cov()**

**used to identify dependency between variables.**

**covariance can explain dependency briefly (not deeply).**

**it can explain 3 ways.**

**if cov==0 --> no dependency.**

**if cov >0 --> positive dependency**

**if cov <0 --> negative dependency.**

**--------------------------**

**Correlation.**

**------------**

**used to identify dependency between variables.**

**correlation always between  -1 to +1.**

**symbol--> r.**

**if r==0, there is no dependency**

**if r>0 and r<=0.5 --> weak positive dependency.**

**if r>0.5 and r<1 --> strong positive dependency.**

**if r==1--> perfect positive dependency.**

**if r>=-0.5  and r<0 --> weak negative dependency.**

**if r>-1 and r<-0.5 --> strong negative dependency.**

**if r==-1 ---> perfect negative dependency.**

**Def corr(x,y):**

**n = x.size**

**nleft= n\*sum(x\*y)**

**nright= sum(x)\*sum(y)**

**dleft= n \* sum(x\*\*2) - sum(x)\*\*2**

**dright=  n \* sum(y\*\*2) - sum(y)\*\*2**

**num = nleft - nright**

**den = (dleft\*dright)\*\*0.5**

**return num/den**

**Python session 12**

**Predictive models with python (in statistics style)**

**--------------------------**

**Simple linear regression.**

**Multiple linear regression.**

**Logistic regression.**

**1.**

**Task:**

**based on age , predict weight of a person.**

**age--> input variable.**

**weight --> target variable.**

**simple linear regression:-> 1 input variable 1 or more target variables.**

**Phase1-->build model**

**Phase2--> accuracy testing.**

**Phase3 --> apply predictions for new data.**

**# build  model.**

**Age = [1,3,6,9,10,12,15,20]**

**Wgt = [3,10,19,28,31,38,43,59]**

**For v in list(zip(age,wgt)):**

**print(v)**

**# need to find weights(co-efficients) between age, wgt**

**#  co-efficients construct relationship bw i/p and o/p variable.**

**# predictive equation : y = a + bx**

**# wgt = a + b. Age**

**# a --> intercept**

**# b --> slope**

**Def sd(x):**

**num =  ((x-x.mean())\*\*2).sum()**

**den = x.size - 1**

**return (num/den)\*\*0.5**

**Def corr(x,y):**

**n = x.size**

**nleft = n\* sum(x\*y)**

**nright = sum(x) \* sum(y)**

**num = nleft - nright**

**dleft = n \*sum(x\*\*2) - sum(x)\*\*2**

**dright = n\*sum(y\*\*2) - sum(y)\*\*2**

**den = (dleft \* dright)\*\*0.5**

**return num/den**

**Def slope(x,y):**

**return corr(x,y) \* sd(y)/ sd(x)**

**Def intercept(x,y):**

**return  y.mean() - slope(x,y)\*x.mean()**

**Def weights(x,y):**

**b = slope(x,y)**

**a = intercept(x,y)**

**return (a,b)**

**Import numpy as np**

**Age = np.array(age)**

**Wgt = np.array(wgt)**

**W = weights(age,wgt)**

**Print(w)**

**(1.3, 3)**

**# wgt = 1.3 + 3\*age**

**#phase2: accuracy testing.**

**# its based on closeness between actual and predicted values.**

**Y = wgt**

**Ycap =  w[0] + w[1]\* age**

**Dist = np.abs(y-ycap)/y \* 100**

**Print(np.c\_[y,ycap,dist])**

**Def distance(y,ycap):**

**return np.abs(y-ycap)/y \* 100**

**Def accuracy(y,ycap,closeness):**

**d = distance(y,ycap)**

**de = 100-closeness**

**pcnt = 0**

**for v in d:**

**if v<=de:**

**pcnt+=1**

**n = y.size**

**acc =  pcnt/n \* 100**

**return acc**

**Accuracy(y,ycap,90)**

**# 87.5**

**#phase3: apply prediction on new data.**

**Newage = [24,20,30]**

**P = np.array(newage)**

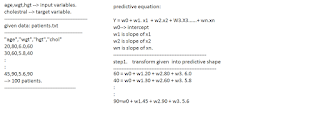
**Pwgts =  w[0] + w[1]\*p**

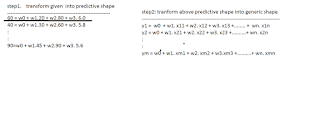
**# predictions:**

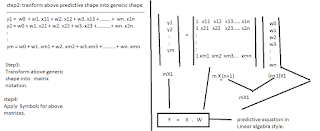
**For v in list(zip(p,pwgts)):**

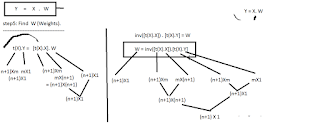
**pritn(v)**

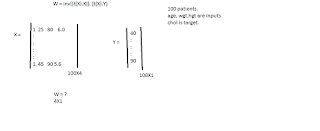
**Python session 13**

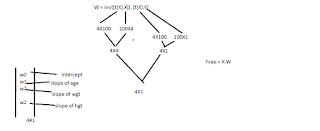
[](https://1.bp.blogspot.com/-aeLY720r0mY/XubkxtBIsII/AAAAAAAAAAU/Sfuxtj15UN0mRFzAR46Zw07e3BF4LJuHACK4BGAsYHg/s1348/mlr1.png)

[](https://1.bp.blogspot.com/-VyxmcTZ-fx4/Xubk4SRZBoI/AAAAAAAAAAg/vqFgyfzNMRAf4fNoEqy6Hr_EydpXBn-igCK4BGAsYHg/s1348/mlr2.png)

[](https://1.bp.blogspot.com/-dMi488yHdv0/Xubk9gV_SkI/AAAAAAAAAAs/0cM2EF3yEDAresKgBBDYBGZRxcfb6CfKgCK4BGAsYHg/s1348/mlr3.png)

[](https://1.bp.blogspot.com/-BFQT8MdjeIQ/XublDSG4wyI/AAAAAAAAAA8/UzrnlitRLI4aGQWuGq7lVcTyCKLFgOMQgCK4BGAsYHg/s1348/mlr4.png)

[](https://1.bp.blogspot.com/-ESU9F3d-jcM/XublJZ8G1rI/AAAAAAAAABM/Ttge0Xeb4PIs4odvjAD_EsxQsOoftMSRwCK4BGAsYHg/s1348/mlr5.png)

[](https://1.bp.blogspot.com/-0afKfWvtYvI/XublOT8QGMI/AAAAAAAAABc/SU-4yHg6RRI-lmk35sO72DG9XPslJQqawCK4BGAsYHg/s1348/mlr6.png)

**Python session 14**

**Multiple linear regression implementation with python.(numpy)**

**----------------------------------------------------------**

**# function to derive weights.**

**Import numpy as np**

**Def weights(x,y):**

**t = np.transpose(x)  # x.t**

**from numpy.linalg import inv**

**left =   inv(t.dot(x))**

**right =  t.dot(y)**

**w = left.dot(right)**

**return w**

**# function for prediction.**

**Def  predict(x,w):**

**ycap = x.dot(w)**

**return ycap**

**# function for accuracy testing.**

**Def  accuracy(y,ycap,closeness):**

**d =  np.abs(y-ycap)/y \* 100**

**de = 100 -closeness**

**pcnt = d[d<=de].size**

**n = y.size**

**acc = pcnt/n \* 100**

**return acc**

**"""**

**Age,wgt,hgt are input variables.**

**Colestral is target variable.**

**"""**

**Age = np.array([10,15,20,25,35,30,45,50])**

**Wgt = np.array([20,32,41,48,72,65,91,95])**

**Hgt = np.array([3,4,6,5.8,5.5,5.9,5.8,5.9])**

**Chol = np.array([ 53.4,83.7, 106.8,128.54, 191.9,  169.92, 244.54, 259.92])**

**# step1. Prepare input matrix. (x)**

**Ones = np.ones(len(age))**

**Print(ones)**

**X = np.c\_[ones,age,wgt,hgt]**

**Print(x)**

**# step2. Prepare output matrix (y)#**

**#way1.**

**Y = chol.reshape((len(chol),1))**

**Print(y)**

**#way2**

**Y = np.c\_[chol]**

**Print(y)**

**#step3. Find relationship bw features(inputs) and labels(target).**

**# weights will construct relationship bw inputs and targets.**

**W = weights(x,y)**

**Print(w)**

**#step4. Accuracy testing with 90% closeness bw actual and predicted.**

**Ycap = predict(x,w)**

**Print(ycap)**

**Acc = accuracy(y,ycap,90)**

**Print(acc)**

**# step5. Apply predictions on new data (patients).**

**A=[23,42,25]**

**Wt=[47,80,53]**

**H=[5.7,5.8,6]**

**= np.ones(len(a))**

**P = np.c\_[o,a,wt,h]**

**Print(p)**

**Cholestrals = predict(p,w)**

**Print(cholestrals)**

**Print(np.c\_[a,wt,h,cholestrals])**