# Data Science notes

# Python for Data Science (Class 2)

# -----------------------------------------

# Bharat Sreeram

# sankara.deva2016@gmail.com

# whatsapp : 6309613028

# -----------------------------------

# environments of python coding:

# 1. interactive mode.

# 2. script mode.

# -----------------------------

# 1. 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 .

# 1. simple types.

# 2. complex types (collection types)

# 1. 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]

# More on Strings and String Handling we will see in upcoming classes.

# session 3:

# -------------------

# Topic: Python basic Collections:

# 1. List

# 2. tuple

# 3. dictionary

# ---------------------------

# 1. 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

# 2. 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

# ---------------------------------------------

# More information and examples on list/tuple/dictionary we will see

# in next class.

# -----------------------------------------------

# 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)

# Comment

# 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)

# Comment