Python Activity -3

Dictionary and Multidimensional data structure

Q1. Given DBI={'db1':'sql','db2':'sqlite','db3':'mysql'}

Write a python program to

a. Display 'db2' value

b. Replace mysql with pl/sql

c. Display list of keys from DBI

```
DBI={'db1':'sql','db2':'sqlite','db3':'mysql'}

print("db2 value:{}".format(DBI['db2']))

DBI['db3']="PL/SQL"

K=DBI.keys()
print("\nList of keys:{}".format(K))

# List of keys and values

for v in DBI.keys():
    print("Key:{}\t Value:{}".format(v,DBI[v]))
```

Q2.Write a python program

a. Create an empty dictionary

b. Display the size of dictionary

c. Read the following details from STDIN - employee name, ID, dept, place and initialize them to dictionary.

d. Using **keys()** function, display dictionary details(key, value)

```
EMP={} # empty dictionary
print("Size of d1 is:{}".format(len(d1)))
# b. display the size of dict
# read the following details from STDIN

name=input("Enter a emp name:")
ID=input("Enter {} emp id:".format(name))
dept=input("Enter {} working dept:".format(name))
place=input("Enter working place:")

# initialize input to dictionary

EMP['name']=name
EMP['ID']=ID
EMP['dept']=dept
EMP['place']=place

print("Display emp details:-")

for v in EMP.keys():
    print("{}\t{}".format(v,EMP[v]))
```

Q3. Write a python program

Given list structure

```
emp=["e123,ram,sales,pune,1000",

        "e132,kumar,prod,bglore,3423",
        "e456,arun,prod,chennai,2456",
        "e544,vijay,hr,mumbai,6500"]
```

a. create an empty dictionary and name it as EMP

b. convert the above given list into dict format.

Note:- employee id as a key, emp name as value

c. display list of key,value pairs from EMP dict

```
emp=[ "e123,ram,sales,pune,1000",
          "e132,kumar,prod,bglore,3423",
           "e456,arun,prod,chennai,2456",
           "e544,vijay,hr,mumbai,6500" ]

print("Given list:{}".format(emp))
# create an empty dictionary and name it’s as EMP
print("")
EMP={} # empty dictionary
for v in emp:
       EID,ENAME,DEPT,PLACE,COST=v.split(",")
       EMP[EID]=ENAME
for v in EMP.keys():
       print("{}\t{}".format(v,EMP[v]))
```

Q4. Write a python program

```
hosts={
“alias1”:”host1.example.com”,
“alias2”:”host2.example.com”,
“alias3”:”host3.example.com”
}
```

a. display key,value pairs to monitor.

b. read an alias name from STDIN(Keyboard)

test input key (alias) name is existing or not.

c. if key exists,update the lo (LOCAL HOST) address (127.0.0.1) to input key.

d. display key,value pairs to monitor.

```
import  sys
hosts={
"alias1": "host1.example.com",
"alias2": "host2.example.com",
"alias3": "host3.example.com"
}
# a. display key,value pairs to monitor.
for  v in hosts.keys():
    print("KEY:{}\tVALUE:{}".format(v,hosts[v]))
```

```
# b.read a alias name from STDIN( Keyboard )
# test input key (alias) name is existing or not.

var=input("Enter an alias name:")
if(hosts.get(var) == None):
        print("{} is invalid alias name".format(var))
        sys.exit()  # exit from script
else:
        hosts[v]="127.0.0.1"

# d. display key,value pairs to monitor.
for v in hosts.keys():
        print("KEY:{}\tVALUE:{}".format(v,hosts[v]))
```

Q5. Write a python program and convert below declaration into dictionary format as specified.

```
sub1="python"

sub2="ruby"

sub3="perl"

sub4="java"

sub5="oracle"

os1="OL5"

os2="OL6"

os3="OL7"
```

a. create a dictionary name as "Course"

b. create two keys named as "Subject" and "OS"

c. initialize list of subject names to "Subject" key and list of os names to "OS" key

d. display list of item pair to screen.

```
sub1="python"
sub2="ruby"
sub3="perl"
sub4="java"
sub5="oracle"
os1="OL5"
os2="OL6"
os3="OL7"

# a. create a dict name as "Course"
course={}

# b. create two keys named as "Subject" and "OS"
course["Subject"]=[sub1,sub2,sub3,sub4,sub5]
course["OS"]=[os1,os2,os3]
```

```
# c. initialize list of subject names to "Subject" key and
# list of os names to "OS" key

print("List of keys:{}".format(course.keys()))

print("\nList of keys\tvalues:")
for v in course.keys():
    print(v,course[v])
```

Q6. Given dictionary

```
conf={"f1":"/etc/passwd","f2":"/etc/group",
        "f3": "/etc/sysconfig","f4":None
}
```

a. Determine the size of conf dictionary
b. Add new configuration file (/etc/pam.d)
c. Using keys() and get() display key,value details

```
conf={
    "f1":"/etc/passwd",
    "f2":"/etc/group",
    "f3": "/etc/sysconfig",
    "f4":None
}
```

```
#a. Determine the size of conf dictionary
print("Size of dict:{}".format(len(conf)))

#b. Add new configuration file (/etc/pam.d)
conf['f4']="/etc/pam.d"

#c. Using keys() and get() display key,value details
for v in conf.keys():
        print("Key:{}\tValue:{}".format(v,conf[v]))

print("") # empty line
print("using get() function")
print("f1 value:",conf.get("f1"))
print("f2 value:",conf.get("f2"))
print("f3 value:",conf.get("f3"))
print("f4 value:",conf.get("f4"))
```

Q7. Create an empty dictionary student.

Using setdefault() function, add the following student details(student name,ID,dept,DOB to student dictionary.

a. Using keys() and get(), display the student details

b. using items() display student details

# understand the difference between return value of keys() & items()

```
STUDENT={}

STUDENT.setdefault("name","Mr.Arun")
STUDENT.setdefault("ID","MH123")
STUDENT.setdefault("dept","MECH")
STUDENT.setdefault("DOB","01/02/1990")

for v in STUDENT.keys():
	print("Key:{}\t\tValue:{}".format(v,STUDENT[v]))


print("")
print(STUDENT.items())
```

Q8. Given dict structure

```
Proc={'pid':12,'fs':'/proc','user':'root','sh':'/bin/bash'}
  using pop() - delete 'fs' and 'sh' key entries
  using del() - delete 'pid' and 'user' entries
  # what's the difference between pop() and del()
  using popitem() - delete Proc structure
```

```
Proc={'pid':12,'fs':'/proc','user':'root','sh':'/bin/bash'}

k1=Proc.pop('fs')
k2=Proc.pop('sh')

print("Removed items K1:{}\tK2:{}\n".format(k1,k2))

print("After removed 'fs' and 'sh' keys:")
print(Proc)

del(Proc['user'])
del(Proc['pid'])

print("After removed 'user' and 'pid' keys:")
print(Proc)

# Proc.popitem() # Error - dictionary is empty
```

Q9. Identify the errors

a)

car = { "brand":

"Ford" "model":

"Mustang"

"year": 1964

}

b) fs={"ftype":"ext4","proto":"tcp/ip","port":80}

>>>fs["ext4"]

a)

car = { "brand": , Comma is missing

"Ford" "model": , Comma is missing

"Mustang"

"year": 1964

}

b) fs={"ftype":"ext4","proto":"tcp/ip","port":80}

>>>fs["ext4"] ← invalid key

Q10. Identify the errors

a) import sys; sys.modules =[os] # Don’t use **=** and invalid **os** key

b) d1={"k1":"v1"}

```
d1.pop()   # dict.pop(‘key’)
           # TypeError: pop expected at least 1 arguments, got 0
```

c) d2={}

```
d2.setdefault("K1","V1","K2","V2")

# dict.setdefault(“key”,”value”) //valid
# dict.setdefault(“key”,”value”,”key”) //valid
# AttributeError: 'set' object has no attribute 'setdefault'
```

Multidimensional Data structures

Q1.

```
action_model = {
        'request': {
            'operation': 'DeleteTags',
            'params': [{
                'target': 'Resources[0]',
                'source': 'identifier',
                'name': 'Id'
            }]
        }
    }
```

a. Determine the given structure type

b. How to print 'name' value

```
print ("Given struct type is dict of dict - nested strcut is dict of list")

print ("Name value is:", action_model['request']['params'][0]['name'])
```

Q2.

```
Cloudwatch={
    AlarmName:"Web_Server_CPU_Utilization",
    ComparisonOperator:'GreaterThanThreshold',
    EvaluationPeriods:1,
    MetricName:'CPUUtilization',
    Namespace:'AWS/EC2',
    Period:60,
    Statistic:'Average',
    Threshold:70.0,
    ActionsEnabled:False,
    AlarmDescription:'Alarm when server CPU exceeds 70%',
    Dimensions:[
        {
          'Name': 'InstanceId',
          'Value': 'INSTANCE_ID'
        },
    ],
    Unit:'Seconds'
}
```

a. print structure type

b. How to display 'Namespace' and 'Threshold' values

c. How to add the following as a 'Dimensions' value{'Name1':'InstanceID1','Value1':'InstanceID2'}

d. modify 'Unit' value to 'Minits'

```
import  pprint

Cloudwatch={

    'AlarmName':"Web_Server_CPU_Utilization",

    'ComparisonOperator':'GreaterThanThreshold',

    'EvaluationPeriods':1,

    'MetricName':'CPUUtilization',

    'Namespace':'AWS/EC2',

    'Period':60,

    'Statistic':'Average',

    'Threshold':70.0,

    'ActionsEnabled':False,

    'AlarmDescription':'Alarm when server CPU exceeds 70%',

    'Dimensions':[

        {
          'Name': 'InstanceId',

          'Value': 'INSTANCE_ID'

        },

    ],

    'Unit':'Seconds'

}


print("Given struct type is: dict of dict,nested struct is dict of list and list of dict")


print("Namspace value is:",Cloudwatch['Namespace'],"\tThreshold value is:",Cloudwatch['Threshold'])

Cloudwatch['Dimensions'].append({'Name1','InstanceID1'})

Cloudwatch['Dimensions'].append({'Value1','InstanceID2'})

Cloudwatch['Unit']="Minits"

print("")

pprint.pprint(Cloudwatch)
```

Q3. Given Structure

```
S={
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:Describe*",
        "ec2:Describe*",
        "ec2:RebootInstances",
        "ec2:StopInstances*",
        "ec2:TerminateInstances"
      ],
      "Resource": [
        "*"
      ]  } ]   }
```

a. How to display the list of instances from 'Action' key?

```
print(type(S["Statement"][0]["Action"]))
print(S["Statement"][0]["Action"])
```

Q4. Given structure

```
namedtuple=(
    'ServiceContext',
    ['service_name', 'service_model', 'service_waiter_model',
     'resource_json_definitions']
)
```

Display the tuple data members

```
print(namedtuple[0])
print(namedtuple[1])


print(namedtuple[1][0])
print(namedtuple[1][1])
print(namedtuple[1][2])
print(namedtuple[1][3])
```

Q5.

```
# Create a relationship definition and attach it
# to the model, such that all identifiers must be
# supplied by the user. It will look something like:
         # d={
          #   'resource': {
          #     'type': 'ResourceName',
          #     'identifiers': [
          #       {'target': 'Name1', 'source': 'input'},
          #       {'target': 'Name2', 'source': 'input'},
          #       ...
          #     ]
          #   }
          # }
          #
```

```
  fake_has : {
                    'resource': {
                        'type': name,
                        'identifiers': []
                    }
                }
```

a. Display the 'identifiers' value.

b. Add new entries ({'target': 'Name2', 'source': 'input'}) to 'identifiers'

```
import pprint


d={
'resource': {
'type': 'ResourceName',
'identifiers': [{'target': 'Name1', 'source': 'input'},{'target': 'Name2', 'source': 'input'}],
},
'fake_has': {'resource': {'type':'name','identifiers':[]}}
}
```

```
print(type(d['resource']['identifiers']))
pprint.pprint(d['resource']['identifiers'])
# to access single data
print(d['resource']['identifiers'][0]['target'])
print(d['resource']['identifiers'][1]['target'])

# adding new entries
d['resource']['identifiers'].append({'target':'Name2'})
d['resource']['identifiers'].append({'source':'input'})
print("Updated structure:-")
pprint.pprint(d)
```

Q6.

```
my_managed_policy = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "RESOURCE_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Scan",
                "dynamodb:UpdateItem"
            ],
            "Resource": "RESOURCE_ARN"
        }
    ]
}
```

Understand the above struct.

a. display each struct elements

b. insert following data members to 'Statement' key

```
{
"Effect":"Allow",
"Action":"logs:CreateLogMembers",
"Resource":["RESOURCE_ARN","RESOURCE_SAB","RESOURCE_AB"]}
```

```
import pprint
my_managed_policy = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "logs:CreateLogGroup",
            "Resource": "RESOURCE_ARN"
        },  {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Scan",
                "dynamodb:UpdateItem"
            ],
            "Resource": "RESOURCE_ARN"
        }  ] }
```

```
print("Given struct is dict of list of dict")

my_managed_policy['Statement'].append(
{
"Effect":"Allow",
"Action":"logs:CreateLogMembers",
"Resource":["RESOURCE_ARN","RESOURCE_SAB","RESOURCE_AB"]})

print.pprint(my_managed_policy)
```

Q7. Write a python program

Given dictionary structure

```
ResponseMetadata={
'RequestId': 'nnnnn-e323-nn-a9a3-254nnnn2c3b6',
'RetryAttempts': 0,
'HTTPHeaders': None,
'transfer-encoding': 'chunked',
'content-type': 'text/xml',
'vary': 'Accept-Encoding',
'server': 'AmazonEC2',
'HTTPStatusCode': 200
}
```

a. Display key,value details to screen.

b. Assign a value to 'HTTPHeaders'

c. Modify the value 'text/xml' into 'text/html'

Note : before modifying 'text/xml' value, test input key 'content-type' exists or NOT

(use : get() function)

d. Display key,value details to screen (compare 'a' statement)

```
ResponseMetadata={
'RequestId': 'nnnnn-e323-nn-a9a3-254nnnn2c3b6',
'RetryAttempts': 0,
'HTTPHeaders': None,
'transfer-encoding': 'chunked',
'content-type': 'text/xml',
'vary': 'Accept-Encoding',
'server': 'AmazonEC2',
'HTTPStatusCode': 200
}
# a. Display key,value details to screen.
for v in ResponseMetadata.keys():
    print("key: {}\t Value:{}".format(v,ResponseMetadata[v]))

# b. Assign a value to 'HTTPHeaders'
# ------------------------------------
ResponseMetadata['HTTPHeaders']="Apache 2.0"
```

```
# c. Modify the value 'text/xml' into 'text/html'
# Note : before modifying 'text/xml' value, test input key
'content-type' exists or NOT


if(ResponseMetadata.get('content-type')):
    ResponseMetadata['content-type']="text/html"
else:
    print("Invalid key")


print("") # empty line


# d.Display key,value details to screen.
for v in ResponseMetadata.keys():
    print("key: {}\t Value:{}".format(v,ResponseMetadata[v]))
```

Q8. Given Struct

```
stry = {
  ad_          : { 'class' : 'DBD::AnyData',       },
  ad2_         : { 'class' : 'DBD::AnyData2',      },
  ado_         : { 'class' : 'DBD::ADO',           },
  amzn_        : { 'class' : 'DBD::Amazon',        },
  best_        : { 'class' : 'DBD::BestWins',      },
  csv_         : { 'class' : 'DBD::CSV',           },
  dbi_         : { 'class' : 'DBI',                },
  dbm_         : { 'class' : 'DBD::DBM',           },
  df_          : { 'class' : 'DBD::DF',            },
  examplep_    : { 'class' : 'DBD::ExampleP',      },
}
```

a. how to add new DBD into existing dictionary (ex: db2 : DBD::DB2)

b. display list of keys from stry dictionary

```
import   pprint
stry = {
  'ad_'        : { 'class': 'DBD::AnyData',        },
  'ad2_'        : { 'class' : 'DBD::AnyData2',       },
  'ado_'        : { 'class' : 'DBD::ADO',            },
  'amzn_'       : { 'class' : 'DBD::Amazon',         },
  'best_'       : { 'class' : 'DBD::BestWins',       },
  'csv_'        : { 'class' : 'DBD::CSV',            },
  'dbi_'        : { 'class' : 'DBI',                 },
  'dbm_'        : { 'class' : 'DBD::DBM',            },
  'df_'         : { 'class' : 'DBD::DF',             },
  'examplep_'   : { 'class' : 'DBD::ExampleP',       },
}
stry['db2']="DBD::DB2"
pprint.pprint(stry)
print("\nList of keys from stry\n"+"-"*35)
pprint.pprint(stry.keys())
print("\nstry['db2'] value is:{}\n".format(stry['db2']))
```

Q9. Given structure

```
EXPORT_TAGS = {
   sql_types :[
        SQL_GUID
        SQL_WLONGVARCHAR
        SQL_WVARCHAR
        SQL_WCHAR
        SQL_BIGINT
        SQL_BIT
        SQL_TINYINT
        SQL_LONGVARBINARY
        SQL_VARBINARY
        SQL_BINARY ]
}
```

Display each key,value details to screen.

```
import   pprint
EXPORT_TAGS = {
   'sql_types' :[
        'SQL_GUID',
        'SQL_WLONGVARCHAR',
        'SQL_WVARCHAR',
        'SQL_WCHAR',
        'SQL_BIGINT',
        'SQL_BIT',
        'SQL_TINYINT',
        'SQL_LONGVARBINARY',
        'SQL_VARBINARY',
        'SQL_BINARY' ]
}
pprint.pprint(EXPORT_TAGS)
print(EXPORT_TAGS['sql_types'])
print("")
for v in EXPORT_TAGS.keys():
        print("Key:",v,"\tValues: ",EXPORT_TAGS[v])
```

```
## To access single data from EXPORT_TAGS


print(EXPORT_TAGS['sql_types'][1])
print(EXPORT_TAGS['sql_types'][2])
print(EXPORT_TAGS['sql_types'][3])
print(EXPORT_TAGS['sql_types'][4])
print(EXPORT_TAGS['sql_types'][-1])
```

Q10. Convert Given struct to dictionary of dictionary format.

```
EXPORT_TAGS={"html2":["h1":"header1","h2":"header2"],
            "html3":["cgi":"param"],
            "html5":["https":"urllib2","requests":"bs4"]
}
```

```
import pprint
EXPORT_TAGS={"html2":{"h1":"header1","h2":"header2"},
            "html3":{"cgi":"param"},
            "html5":{"https":"urllib2","requests":"bs4"}
}

print(EXPORT_TAGS['html3']['cgi'])
pprint.pprint(EXPORT_TAGS)
```

Q11.write a python program

Given list structure

emp=["e123,ram,sales,pune,1000",
"e132,kumar,prod,bglore,3423",

"e456,arun,prod,chennai,2456",

"e544,vijay,hr,mumbai,6500"]

a. create a empty dictionary name as EMP

b. convert the above given list into dict format

c. employee id as a key, rest of the details(name,dept,place,cost) as values

d. display list of key,value pairs from EMP dict

Expected struct is

EMP={"e123":["ram","sales","pune",1000],...}

```
import  pprint

emp=["e123,ram,sales,pune,1000",

"e132,kumar,prod,bglore,3423","e456,arun,prod,chennai,2456",
"e544,vijay,hr,mumbai,6500"]

EMP={} # a. create a empty dictionary name as EMP

for v in emp:

    v=v.strip()

    L=v.split(",")

    EMP[L[0]]=L[1:] # adding data to dict - single 'key' holding more than one value

pprint.pprint(EMP)


print("\n\n")

# from struct accessing single data

#

print("Employee name is:{}\t Working dept is:{}".format(EMP["e123"][0],EMP["e123"][1]))
```

Q12. Convert the given list to dictionary of list

```
OS=["OL5","RHL5","OL6","OL7","RHL7","DEB5","OL6",
      "OL7","RHL5"]

# note - ignore duplicate values
# Expected Result is
# ---------------------
OS1={"OS"=>["OL5","RHL5","OL6","OL7","RHL7","DEB5"]
}
#
import pprint
OS=["OL5","RHL5","OL6","OL7","RHL7","DEB5","OL6",
     "OL7","RHL5"]

print("Given type is:{} holding value
is:{}\n".format(type(OS),type(OS[0])))
```

```
OS1={} # create new dict

'''

To remove duplicate value from list simple way use set operation

>>> OS=["OL5","RHL5","OL6","OL7","RHL7","DEB5","OL6","OL7","RHL5"]

>>> set(OS) # typecast to set operation

{'RHL7', 'DEB5', 'OL5', 'OL6', 'OL7', 'RHL5'} <== this set result unique value.

>>>

>>> OS=list(set(OS)) # typecast to list

>>> OS

['RHL7', 'DEB5', 'OL5', 'OL6', 'OL7', 'RHL5']

>>>

'''
```

```
OS1["OS"]=OS


print("OS1 struct type is:{} holding value
is:{}\n".format(type(OS),type(OS)))

pprint.pprint(OS1)

print("--"*39+"\n")

print(OS1["OS"][0]) # Access single data from OS1 structure
## dict of list

print(OS[0]) # # Access single data from OS structure
```