**PART-A**

**Python Introduction - Simple Programs in Python based on: Selection Constructs 2) Looping Constructs**

***Selection Constructs***

a=input("Enter value of a ")

print ("The value of a is",a)

b=input("Enter value of b")

print ("The value of b is",b)

if a>b:

print("a is greater")

elif b>a:

print("b is greater")

else:

print("both are equal")

***Looping Constructs***

for n in range(2, 10):

# Look for a divisor of n

for x in range(2, n):

if n % x == 0:

print(n, 'equals', x, '\*', n//x)

break

else:

# loop fell through without finding a factor

print(n, 'is a prime number')

**Using Python - Simple Problems in Python based on:  1)Lists 2) Tuples 3) Dictionary**

***List***

a=["physics","mathematics","chemistry","biology"]

#Retreiving values in list

print("Set of values in the list:",a[2:3])

#Negative index

print("Negative index",a[-2])

#change list item

a[2]="english"

print("values in the list:",a)

#delete list item

del a[2]

print("values in the list:",a)

#insert value at particular index

a.insert(1,"social studies")

print("values in the list:",a)

#reverse values

a.reverse() #returns no values

print("reversed values in the list:",a)

#sorted values

a.sort() #returns no values

print("sorted values in the list:",a)

***Tuple***

tuple1=(10,20,30,40,50)

tuple2=("a","b","c")

tuple3=tuple1+tuple2

print("Two tuples concatenated",tuple3)

del tuple3

#print("While trying to display a deleted tuple", tuple3)

#Above line will raise an exception since tuple3 no longer exists

#Few built-in functions

print("Maximum element",max(tuple1))

print("Minimum element",min(tuple1))

print("Length of tuple1",len(tuple1))

***Dictionary***

d={1:'Amazon',2:'Flipkart',3:'Snapdeal',4:'Myntra'}

#Print all values

print(d)

#Retrieving values

print(d[1])

#Deleting a key-value pair

del d[4]

print(d)

#Get the key-value pairs as list ko

print("List of key value pairs",d.items())

#Get list of keys

print("List of keys",d.keys())

#Get list of values

print("List of values",d.values())

#Shallow copy of dictionary

e=d.copy()7

print("Shallow copy of dictionary d",e)

#Clear all entries of a dictionary

d.clear()

print("Trying to print a dictionary",d)

**Combining Python Basics and its Data Structures - Programs in Python based on:  Selection, Looping with Lists, Tuples, Dictionaries**

***Selection***

num = int(input('Enter any number : '))

if num % 2 == 0:

print(f'The number {num} is a Even number')

else:

print(f'The number {num} is a Odd number')

***Looping with Lists***

def common\_data(list1, list2):

result = False

for x in list1:

for y in list2:

if x == y:

result = True

return result

print(common\_data([1,2,3,4,5], [5,6,7,8,9]))

print(common\_data([1,2,3,4,5], [6,7,8,9]))

***Looping with tuples***

T = (10,20,30,40,50)

for var in T:

print (T.index(var),var)

***Looping with dictionary***

def sum\_math\_v\_vi\_average(list\_of\_dicts):

for d in list\_of\_dicts:

n1 = d.pop('V')

n2 = d.pop('VI')

d['V+VI'] = (n1 + n2)/2

return list\_of\_dicts

student\_details= [

{'id' : 1, 'subject' : 'math', 'V' : 70, 'VI' : 82},

{'id' : 2, 'subject' : 'math', 'V' : 73, 'VI' : 74},

{'id' : 3, 'subject' : 'math', 'V' : 75, 'VI' : 86}

]

print(sum\_math\_v\_vi\_average(student\_details))

**Using Python - Programs in Python based on: 1)Functions 2) Recursive Functions 3)Class**

***Python Function***

#Sample Function

def print\_text():

print("Hello World!")

return

print\_text()

#Python Function

#Function with arguments

def print\_values(a,b):

print("The values of a and b are",a,"and",b)

return

print\_values(10,20)

#Python Function

# Function with default arguments

def print\_text(a,b=5):

print("The values of a and b are",a,"and",b)

return

print\_text(10)

#Python Function

#Function Call - Keyword Arguments

def print\_values(a,b):

print("The values of a and b are",a,"and",b)

return

print\_values(b=20,a=10)

***Recursion***

def factorial(n):

if n == 1:

return 1

else:

return n \* factorial(n-1)

a=factorial(5)

print("Result of factorial",a)

***Class***

class Complex:

def \_\_init\_\_(self, realpart, imagpart):

self.r = realpart

self.i = imagpart

x=Complex(3.5,3.5)

**Using Python - Programs in Python based on: 1) One-line comprehensions   2) Lambda, Reduce function**

***List Comprehension***

input\_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]

list\_using\_comp = [var for var in input\_list if var % 2 == 0]

print("Output List using list comprehensions:",list\_using\_comp)

***Dictionary Comprehension***

input\_list = [1, 2, 3, 4, 5, 6, 7]

output\_dict = {}

# Using loop for constructing output dictionary

for var in input\_list:

if var % 2 != 0:

output\_dict[var] = var\*\*3

print("Output Dictionary using for loop:", output\_dict )

***Lambda Function***

x = lambda a, b : a \* b

print(x(5, 6))

***Reduce***

#Importing functools module to use reduce function

from functools import reduce

li = [5, 8, 10, 20, 50, 100]

sum = reduce((lambda x, y: x + y), li)

print (sum)

**HTML Basics Introduction - Creating Static Pages with different HTML Components**

<!--Form Example-->

<html>

<head>

<title>

Customer Feedback

</title>

</head>

<body>

<h2>Enter your feedback</h2>

<form action="thanks.html">

<!--Labels can be mentioned as such without need of any tag-->

Name <br>

<!--Text Box-->

<input type="text" name="Customer\_name" value="Customer Name"/><br><br>

**HTML Basics Introduction - Creating Static Pages with hyperlinks and images**

<html>

<head>

<title>

Lists, Tables, Images and Links

</title>

</head>

<body>

<h3>Python Datatypes</h3>

<ul>

<li>Strings</li>

<li>Numbers</li>

<ol>

<li>Integer</li>

<li>Floating Point</li>

<li>Complex Number </li>

<li>Long</li>

</ol>

<li>Tuples</li>

<li>Dictionary</li>

<h3> Image Example </h3>

<img src="Desert.jpg" alt="Desert" height="200" weight="200"/>

<h3>Link Example</h3>

<a href="https://www.w3schools.com">Visit W3Schools.com!</a>

<h3>Table Example</h3>

<table style="width:100%" border="1px">

<tr>

<th>Number</th>

<th>Example Values</th>

</tr>se

<tr>

<td>Integer</td>

<td>10</td>

</tr>

<tr>

<td>Long</td>

<td>9000000L</td>

</tr>

<tr>

<td>Float</td>

<td>1.5</td>

</tr>

<tr>

<td>Complex</td>

<td>1.j7.8</td>

</tr>

</table>

</body>

</html>

**Javascript Introduction - Basics of client side scripting**

<html>

<head>

<title>

Factorial

</title>

</head>

<body>

<script>

document.writeln(" Factorial Calculation<br>")

var i, fact=1;

for (i = 1; i <= 5; i++)

fact = fact \* i;

document.writeln ("Factorial of 5 is",fact) </body>

</html>

**Using Javascript - HTML along with Javascript on client side**

<!--Simple Javascript program using number inputs-->

<html>

<head>

<title>

Number Inputs

</title>

</head>

<body>

<form>

<p> Enter two numbers </p> <br><br>

Enter Number1

<input type="text" id="num1"/> <br><br>

Enter Number2

<input type="text" id="num2"/> <br><br>

<input type="button" value="Press me to view the result" onclick="myFunction()"/>

</form>

<p id="result"> </p> <br><br>

<script>

function myFunction()

{

n1=document.getElementById("num1").value;

n2=document.getElementById("num2").value;

if(n1 && n2 && n1>0 && n2>0)

{

document.getElementById("result").innerHTML=n1/n2;

}

else

{

document se.getElementById("result").innerHTML="Please enter correct positive numbers";

}

}

</script>

</body>

</html>

**Combining HTML & Javascript - Programs with Client side event handling scenarios**

***Mouse Event***

<!DOCTYPE html>

<html>

<body>

<img onmouseover="bigImg(this)" onmouseout="normalImg(this)" border="0" src="smiley.gif" alt="Smiley" width="32" height="32">

<p>The function bigImg() is triggered when the user moves the mouse pointer over the image.</p>

<p>The function normalImg() is triggered when the mouse pointer is moved out of the image.</p>

<script>

function bigImg(x) {

x.style.height = "64px";

x.style.width = "64px";

}

function normalImg(x) {

x.style.height = "32px";

x.style.width = "32px";

}

</script>

</body>

</html>

***OnClick Event***

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to add two click events to the same button.</p>

<button id="myBtn">Try it</button>

<script>

var x = document.getElementById("myBtn");

x.addEventListener("click", myFunction);

x.addEventListener("click", someOtherFunction);

function myFunction()

{

alert ("Hello World!");

}

function someOtherFunction()

{

alert ("This function was also executed!");

}

</script>

</body>

</html>

**Using Javascript - Loading JSON data dynamically on the client side.**

***abc.json***

data = '[{"name" : "Riyaz"},{"name" : "Javed"},{"name" : "Arun"},{"name" : "Sunil"},{"name" : "Rahul"},{"name" : "Anita"}]';

***sample.html***

<!doctype html>

<html>

<head>

<script type="text/javascript" src="abc.json"></script>

<script type="text/javascript" >

function load()

{

var mydata = JSON.parse(data);

alert(mydata.length);

var div = document.getElementById('data');

for(var i = 0;i < mydata.length; i++)

{

div.innerHTML = div.innerHTML + "<p class='inner' id="+i+">"+ mydata[i].name +"</p>" + "<br>";

}

}

</script>

</head>

<body onload="load()">

<div id= "data">

</div>

</body>

</html>

**PART-B**

1. **Introduction to Python: Write Python programs to do the following:**
2. **Read a list of elements. Create a new list having all the elements minus the duplicates (Use functions). Use one-line comprehensions of create a new list of even numbers. Create another list reversing the elements.**
3. **Write a python program to count the frequency of words in a given file.**
4. **Read a list of numbers. Uses a recursive function to find the maximum of ‘n’ numbers.**

***1a.py***

S = [x\*\*2 for x in range(10)] # read elements to list

M = [x for x in S if x % 2 == 0]

M.reverse()

***1b.py es***

from collections import Counter

def word\_count(fname):

with open(fname) as f:

return Counter(f.read().split())

print("Number of words in the file :",word\_count("test.txt"))

***1c.py***

def Max(list):

if len(list) == 1:

return list[0]

else:

m = Max(list[1:])

return m if m > list[0] else list[0]

def main():

try:

list = eval(input("Enter a list of numbers: "))

print ("The largest number is: ", Max(list))

except SyntaxError:

print ("Please enter comma separated numbers")

except:

print ("Enter only numbers")

main()

**Introduction to Python Functions:** Write a temperature converter python program, which is menu driven. Each such conversion logic should be defined in separate functions. The program should call the respective function based on the user’s requirement. The program should run as long as the user wishes so. Provide an option to view the conversions stored as list of tuples with attributes - from unit value, to unit value sorted by the user’s choice (from-value or to-value).

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | select = "y" |
|  | op1=[] |
|  | op2=[] |
|  | op3=[] |
|  | op4=[] |
|  | op5=[] |
|  | op6=[] |
|  | while(select=="y" or select == "Y"): |
|  | op = input("Enter the selection: 1.C->F 2.F->C 3.C->K 4.K->C 5.F->K 6.K->F \n") |
|  | if(op=="1" or op=="2" or op=="3" or op=="4" or op=="5" or op=="6"): |
|  | if(op=="1"): |
|  | c = float(input("Enter the temperature in celsius: ")) |
|  | op1 = op1 +[(c,celciustofarenhiet(c))] |
|  | print("Celsius to Farenhiet\n") |
|  | print(sorted(op1)) |
|  | elif(op=="2"): |
|  | p = float(input("Enter the temperature in farenhiet: ")) |
|  | op2=op2 + [(p,farenhiettocelcius(p))] |
|  | print("Farenhiet to Celsius\n") |
|  | print(sorted(op2)) |
|  | elif(op=="3"): |
|  | x = float(input("Enter the temperature in celsius: ")) |
|  | op3=op3 + [(x,celciustokelvin(x))] |
|  | print("Celsius to Kelvin\n") |
|  | print(sorted(op3)) |
|  | elif(op=="4"): |
|  | y = float(input("Enter the temperature in kelvin: ")) |
|  | op4 = op4+[(y,kelvintocelcius(y))] |
|  | print("Kelvin to Celsius\n") |
|  | print(sorted(op4)) |
|  | elif(op=="5"): |
|  | a = float(input("Enter the temperature in farenhiet: ")) |
|  | op5=op5+[(a,farenhiettokelvin(a))] |
|  | print("Farenhiet to Kelvin\n") |
|  | print(sorted(op5)) |
|  | elif(op=="6"): |
|  | b = float(input("Enter the temperature in kelvin: ")) |
|  | op6=op6+[(b,kelvintofarenhiet(b))] |
|  | print("Kelvin to Farenhiet\n") |
|  | print(sorted(op6)) |
|  | else: |
|  | print("Invalid Input") |
|  | select = input("Enter Y to continue and N to terminate\n") |

|  |  |
| --- | --- |
|  | def celciustofarenhiet(c): |
|  | return (c\*(9/5) + 32) |
|  |  |
|  | def farenhiettocelcius(p): |
|  | return ((p-32)\*(5/9)) |
|  |  |
|  | def celciustokelvin(x): |
|  | return (x+273.15) |
|  |  |
|  | def kelvintocelcius(y): |
|  | return (y-273.15) |
|  |  |
|  | def farenhiettokelvin(a): |
|  | return celciustokelvin(farenhiettocelcius(a)) |
|  |  |
|  | def kelvintofarenhiet(b): |
|  | return celciustofarenhiet(kelvintocelcius(b)) |

**Python Classes:** Write a python class to reverse a sentence (initialized via constructor) word by word. Example:  “I am here” should be reversed as “here am I”. Create instances of this class for each of the three strings input by the user and display the reversed string for each, in descending order of number of vowels in the string.

class SentenceReverser:

vowels = ["a","e","i","o","u"]

sentence = ""

reverse = ""

vowelCount = 0

def \_\_init\_\_(self,sentence):

self.sentence = sentence

self.reverseSentence()

def reverseSentence(self):

self.reverse = " ".join(reversed(self.sentence.split()))

def getVowelCount(self):

self.vowelCount = sum(s in self.vowels for s in self.sentence.lower())

return self.vowelCount

def getReverse(self):cd ..

return self.reverse

items = []

for i in range(3):

sentence = input("Enter a phrase : ")

reverser = SentenceReverser(sentence.strip())

items.append(reverser)

print()

sortedItems = sorted(items, key=lambda item: item.getVowelCount(), reverse=True)

print ("Sorted on vowel count (descending) : \n")

for i in range(len(sortedItems)):

print ("Reverse : ", sortedItems[i].getReverse(), ", Vowel Count : ", sortedItems[i].getVowelCount())

**Python for Data Science:** Load *Titanic* *Dataset* into one of the data structures (*NumPy* or *Pandas*). Perform data pre-processing on this dataset. Create dataframes, perform computations and visualize the results appropriately.

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

titanic\_df = pd.read\_csv('train.csv')

# Convert the survived column to strings for easier reading

titanic\_df ['Survived'] = titanic\_df ['Survived'].map({

0: 'Died',

1: 'Survived'

})

print("======Data Headers Before Dropping Columns=======")

print(titanic\_df.head(5))

print("\*\*\*\* \n\nDATA TRANSFORMATION \*\*\*\*\*\n")

print("======Data Headers After Dropping Columns - First Way=======")

titanic\_df.drop(['Parch','PassengerId','Name','Ticket'], axis=1, inplace=True)

#axis =1 in drop method shows you are dropping a column

#inplace=True means you are editing original dataframe

print(titanic\_df.head(5))

print("======Data Headers After Dropping Columns - Second Way =======")

titanic\_df = titanic\_df.drop(['SibSp','Fare'], axis=1)

print(titanic\_df.head(5))

# Convert the Class column to strings for easier reading

titanic\_df ['Pclass'] = titanic\_df ['Pclass'].map({

1: 'Luxury Class',

2: 'Economy Class',

3: 'Lower Class'

})

print("======Data Headers After Transforming Class Column =======")

print(titanic\_df.head(5))

titanic\_df["Embarked"] = titanic\_df["Embarked"].fillna("S")

print("======Data Headers After Filling with default value for Embarked Column =======")

print(titanic\_df.head(5))

# Convert the Embarked column to strings for easier reading

titanic\_df ['Embarked'] = titanic\_df ['Embarked'].map({

'C':'Cherbourg',

'Q':'Queenstown',

'S':'Southampton'

})

print("======Data Headers After Transforming Embarked Column =======")

print(titanic\_df.head(5))

print("\n\n\n\*\*\*\* DATA VISUALIZATIONS\*\*\*\*\n\n")

print("Visualization #1 : Survival Rate Based on Passenger Sitting Class")

ax = sns.countplot(x = 'Pclass', hue = 'Survived', palette = 'Set1',data = titanic\_df)

ax.set(title = 'Passenger status (Survived/Died) against Passenger Class',

xlabel = 'Passenger Class', ylabel = 'Total')

plt.show()

#crosstab - Cross tabulation of two or more factors

print("Visualization #2 : Survival Rate Based on Gender")

print(pd.crosstab(titanic\_df["Sex"],titanic\_df.Survived))

ax = sns.countplot(x = 'Sex', hue = 'Survived', palette = 'Set2', data = titanic\_df)

ax.set(title = 'Total Survivors According to Sex', xlabel = 'Sex', ylabel='Total')

plt.show()

print("Visualization #3 : Survival Rate Based on Passenger Age Group")

# We look at Age column and set Intevals on the ages and the map them to their categories as

# (Children, Teen, Adult, Old)

interval = (0,18,35,60,120)

categories = ['Children','Teens','Adult', 'Old']

#cut - Segment and sort data values into bins

titanic\_df['Age\_cats'] = pd.cut(titanic\_df.Age, interval, labels = categories)

ax = sns.countplot(x = 'Age\_cats', data = titanic\_df, hue = 'Survived', palette = 'Set3')

ax.set(xlabel='Age Categorical', ylabel='Total',

title="Age Categorical Survival Distribution")

plt.show()

print("Visualization #4 : Survival Rate Based on Passenger Embarked Port")

print(pd.crosstab(titanic\_df['Embarked'], titanic\_df.Survived))

ax = sns.countplot(x = 'Embarked', hue = 'Survived', palette = 'Set1', data = titanic\_df)

ax.set(title = 'Survival distribution according to Embarking place')

plt.show()

**Python for Data Science:** Load *Student Performance Dataset* into one of the data structures (*NumPy* or *Pandas*). Perform data pre-processing on this dataset. Create dataframes, perform computations and visualize the results appropriately.

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns

studperf\_df=pd.read\_csv('StudentsPerformance.csv')

studperf\_df.info()

studperf\_df.info(verbose=False) # short summary

studperf\_df.describe()

print(studperf\_df.head(15))

#Remove unnecessary features (E.g. drop unwanted columns) from the dataset such as â€˜lunchâ€™ and â€˜test preparation courseâ€™

stuperf\_df = studperf\_df.drop(['lunch'], axis=1,inplace = False)

print('\\n====Understanding Inplace False : The Copied Dataframe====')

print(stuperf\_df.head(3))

#Manipulate data by replacing empty column values in â€˜parental level of educationâ€™ with a default value.

stuperf\_df["parental level of education"] =stuperf\_df["parental level of education"].fillna("Not applicable")

print(stuperf\_df.head(5))

print(stuperf\_df.head(5))

#Convert the attribute â€˜race/ethnicityâ€™ to have â€˜groupAâ€™ to be â€˜Asian Studentsâ€™, â€˜groupBâ€™ to be â€˜African Studentsâ€™ , â€˜groupCâ€™ to be â€˜Afro-Asian #Studentsâ€™, â€˜groupDâ€™ to be â€˜American Studentsâ€™ and â€˜groupEâ€™ to be â€˜European Studentsâ€™

stuperf\_df["race/ethnicity"]=stuperf\_df["race/ethnicity"].map({"group A" : "Asian students",

"group B" : "African students",

"group C" : "Afro-Asian students",

"group D" : "American students",

"group E" : "European students"})

print(stuperf\_df.head(2))

#Tally of the Number of Male & Female students who took up the â€˜test preparation courseâ€™ and those who did not.

ax = sns.countplot(x="test preparation course",hue='gender',palette='Set3',data=stuperf\_df)

ax.set(title="Course completion based on gender", xlabel='Course', ylabel='Total')

plt.show()

#Total Number of Male & Female Students belonging to each student group

ax = sns.countplot(x="race/ethnicity",hue="gender",palette="Set2",data=stuperf\_df)

ax.set(title="Total number of male and female students belonging to each group", xlabel="Groups", ylabel="Total")

plt.show()

#No of students who â€˜failedâ€™(less than 40), â€˜second classâ€™(between 40 & 50).

#'first classâ€™(between 60 & 75) and â€˜distinctionâ€™(above 75) in â€˜Mathsâ€™,

#â€˜Readingâ€™ and â€˜Writingâ€™.

interval=(0,40,50,60,75)

categories = ["Fail", "2nd class","1st class","Distinction"]

stuperf\_df["Marks\_cats"]=pd.cut(stuperf\_df.mathscore,interval,labels=categories)

ax=sns.countplot(x="Marks\_cats",hue="gender",palette="Set1",data=stuperf\_df)

ax.set(title="Marks categorisation for math",xlabel="Categories",ylabel="Number of students")

plt.show()

stuperf\_df["Marks\_Cats"]=pd.cut(stuperf\_df.readingscore,interval,labels=categories)

ax=sns.countplot(x="Marks\_Cats",hue="gender",palette="Set1",data=stuperf\_df)

ax.set(title="Marks categorisation for reading",xlabel="Categories",ylabel="Number of students")

plt.show()

stuperf\_df["Marks\_Cats"]=pd.cut(stuperf\_df.writingscore,interval,labels=categories)

ax=sns.countplot(x="Marks\_Cats",hue="gender",palette="Set1",data=stuperf\_df)

ax.set(title="Marks categorisation for writing",xlabel="Categories",ylabel="Number of students")

plt.show()

**Python for Data Science:** Load *Iris Dataset* into one of the data structures (*NumPy* or *Pandas*). Perform data pre-processing on this dataset. Create dataframes, perform computations and visualize the results appropriately.

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

df = pd.read\_csv('iris.csv')

print("======Data Headers Before Dropping Columns=======")

print(df.head(5))

df.describe()

df.info()

print("======Data Headers after Dropping Columns=======")

df.drop(['Sepal\_Length'], inplace=True,axis=1)

print(df.head(5))

"""ndf = pd.DataFrame(columns=['Class','Petal\_Width'])

ndf.groupby(['Class'],as\_index=False).mean()"""

plt.figure(figsize=[12,6]) # to create a wider graph\n",

ax = sns.countplot(data = df,hue = 'Class',palette='Set1',x = ' Sepal\_Width')

ax.set(title='Flowers of each specie',xlabel='Sepal Width',ylabel='No. of flowers')

plt.tight\_layout()

plt.show()

interval = (0,1,2,4)

category = ['<1','1 to 2','>2']

df['Petal\_Catg'] = pd.cut(df[' Petal\_Width'],interval,labels=category)

ax = sns.countplot(data = df,x = 'Petal\_Catg',hue='Class',palette='YlOrRd')

ax.set(title='Petal Width',xlabel='Category of Petals',ylabel='No. of flowers')

plt.show()

plt.figure(figsize=[12,6])

ax = sns.countplot(data = df[df['Class'] == 'Iris-setosa'],x = ' Sepal\_Width',palette='Set1')

ax.set(title='Iris-setosa',xlabel='Sepal Width',ylabel='No. of flowers')

plt.show()

**Python File Handling & List Comprehension**: Write a python program to read contents of a file (filename as argument) and store number of occurrences of each word in a dictionary. Display the top 10 words with most number of occurrences in descending order. Store the length of each of these words in a list and display the list. Write a one-line reduce function to get the average length and one-line list comprehension to display squares of all odd numbers and display both.

#!/usr/bin/env python

# Note : Handle all other corner cases which are not handled here

import sys

import os

from functools import reduce

# Write a python program to read contents of a file (filename as argument)

# and store number of occurrences of each word in a dictionary.

dict = {}

wordLen = []

if(len(sys.argv) != 2):

print ("Invalid Arguments")

sys.exit()

if(not(os.path.exists(sys.argv[0]))):

print ("Invalid File Path")

sys.exit()

if(sys.argv[1].split('.')[-1] != "txt"):

print ("Invalid File Format. Only TXT files allowed")

sys.exit()

with open(sys.argv[1]) as file:

for line in file:

for word in line.split():

dict[word] = dict.get(word,0) + 1

# print dict

# Display the top 10 words with most number of occurrences in descending order.

# Food for thought - Does a dictionary maintain order? How to print 10 words with most frequency?

# Ans - extract dict items as Tuples and sort them based on value in dictionary

#(second item of the tuple / index 1)

sortedDict = sorted(dict.items(), key=lambda dictItem: dictItem[1], reverse=True)

for i in range(10):

try:

wordTuple = sortedDict[i]

wordLen.append(len(wordTuple[0]))

print (wordTuple[0], ", Frequency: " , wordTuple[1] , ", Length " , len(wordTuple[0]))

except IndexError:

print ("File has less than 10 words")

break

print ("Lengths of 10 most frequently occuring words:")

print (wordLen)

# Write a one-line reduce function to get the average length

sum = reduce(lambda x,y: x+y, wordLen)

print ("Average length of words: " , sum\*1.0/len(wordLen)\*1.0)

# Write a one-line list comprehension to display squares of all odd numbers

squares = [x\*\*2 for x in wordLen if x%2 != 0]

print ("Squres of odd word lengths: ")

print (squares)

**Python for Data Science:** Load *Black Friday Dataset* into one of the data structures (*NumPy* or *Pandas*). Perform data pre-processing on this dataset. Create dataframes, perform computations and visualize the results appropriately.

"""a)Load the â€˜Black Fridayâ€™ dataset into one of the data structures (NumPy or Pandas).\n",

"b)Display header rows and description of the loaded dataset.\n",

"""

import pandas as pd

import numpy as np

df = pd.read\_csv("blackfri.csv")

print(\"<-----Data Information----->\")

print("Head of Dataset")

print(df.head(5))

print("Head of Dataset")

print(df.describe())

print(df.info())

"""c) Remove unnecessary features (E.g. drop unwanted columns) from the dataset such as â€˜User\_IDâ€™, â€˜Product\_ID â€˜ â€˜Stay\_In\_Current\_City\_Yearsâ€™"""

df.drop(['User\_ID','Product\_ID','Stay\_In\_Current\_City\_Years'], axis=1, inplace=True)

print(df.head(5))

"""d) Manipulate data by replacing empty column values in â€˜City\_Categoryâ€™ with a default value for the city. """

print("Filling empty values")

df['City\_Category'] = df['City\_Category'].fillna(\"A\")

print(df.head(5))

"""e) Convert the attribute â€˜City\_Categoryâ€™ to have â€˜Aâ€™ to be â€˜Metro Citiesâ€™, â€˜Bâ€™ to be â€˜Small Townsâ€™ , â€˜Câ€™ to be â€˜Villagesâ€™."""

print("Mapping values/attributes in City\_Category to types")

df['City\_Category'] = df['City\_Category'].map({'A':'Metro cities','B':'Small Towns','C':'Villages'})

print(df.head(5))

"""f) Rename the attribute â€˜Product\_Category\_1â€™ to have â€˜Baseball Capsâ€™, \n",

"â€˜Product\_Category\_2â€™ to have â€˜Wine Tumblersâ€™ and â€˜Product\_Category\_3â€™ to \n",

"have â€˜Pet Raincoatsâ€™\n",

"""

print("Renaming the column names")

df.rename(columns={'Product\_Category\_1':'Baseball\_Caps','Product\_Category\_2':'Wine\_Tumblers','Product\_Category\_3':'Pet\_Raincoats'},inplace=True)

print(df.head(5))

"""g) Convert the attribute â€˜Marital\_Statusâ€™ to have â€˜1:Marriedâ€™ and â€˜0:Un-Marriedâ€™\n","""

print("Mapping values/attributes in Marital Status to types")

df['Marital\_Status'] = df['Marital\_Status'].map({1:'Married',0:'Un-Married'})

print(df.head(5))

"""h) Perform the following visualizations on the loaded dataset:\n",

"i) Tally of the Number of Male & Female who bought â€˜Product\_Category\_3(Pet\_Raincoats)â€™. \n",

"""

import matplotlib.pyplot as plt

import seaborn as sns

print("<-------Data Visualisation------->")

print(pd.crosstab(df.Gender,df.Baseball\_Caps))

print(pd.crosstab(df.Gender,df.Pet\_Raincoats))

ax = sns.countplot(data=df,x='Gender',hue='Pet\_Raincoats',palette='Set2')

ax.set(title='Male and Female who bought Pet\_Raincoats',xlabel='Gender',ylabel='Count')

plt.show()

"""h) Perform the following visualizations on the loaded dataset:\n",

"ii) Total Number of Male & Female persons belonging to each city category\n",

"""

ax = sns.countplot(data=df,x='Gender',hue='City\_Category',palette='Set1')

ax.set(title='Male and Female belonging to each city',xlabel='Gender',ylabel='Count')

plt.show()

1. **a) JavaScript – Client Side Validation:** Design a case study for a Bakery that creates and validates a HTML form at the client side using Javascript. Bakery Menu & Price calculation of items bought should be the result. Perform the necessary Client Side validation using JavaScript.

<!DOCTYPE html>

<html>

<head>

<title>Shopping Cart</title>

<script type="text/javascript">

function submit() {

var cakeQuantity = document.getElementById("cakeQuantity").value;

var cookieQuantity = document.getElementById("cookieQuantity").value;

var cakeSelected = document.getElementById("cake").checked;

var cookieSelected = document.getElementById("cookies").checked;

if(cakeSelected && cakeQuantity && isNormalInteger(cakeQuantity)){

alert("Bill for cake = "+(parseInt(cakeQuantity)\*100))

}else if(cookieSelected && cookieQuantity && isNormalInteger(cookieQuantity)){

alert("Bill for cookies = "+(parseInt(cookieQuantity)\*50))

}else if(!cakeSelected && !cookieQuantity){

alert("Select an item");

}else{

alert("Invalid a valid positive integer");

}

}

function isNormalInteger(str) {

var n = Math.floor(Number(str));

return String(n) === str && n >= 0;

}

</script>

</head>

<body>

<input type="radio" name="item" id="cake" class="input-hidden" />

<label for="cake">

<img src="cake.png" width="100px" />

</label>

<br>

<p> Cost = Rs. 100 </p>

<p>Quantity:<input type="number" id="cakeQuantity"></p>

<br>

<input type="radio" name="item" id="cookies" class="input-hidden" />

<label for="happy">

<img src="cookie.png" alt="Cookies" width="100px" />

</label>

<br>

<p> Cost = Rs. 50 </p>

<p>Quantity:<input type="number" id="cookieQuantity"></p>

<br> <br> <br>

<input type="submit" onClick="submit()" value="Order">

</body>

</html>

1. **a) JavaScript (JSON):** Dynamically loading JSON data - Implement a HTML+JS application that has a JSON Array with details of different kinds of data. Example: Model, Name, Price, Year. Display details of each vehicle dynamically by only showing details of the vehicle that the user has selected (via mouse-over).

cars.js

window.onload = function() {

var teslaModels = [

{

"model": "modelS",

"name": "Model S",

"price": 69200,

"year": 2016

},

{

"model": "modelX",

"name": "Model X",

"price": 83700,

"year": 2017

},

{

"model": "model3",

"name": "Model 3",

"price": 35000,

"year": 2017

},

];

//Generate HTML for Menu Bar

teslaModels.forEach(function(item,index){

listElemet = document.createElement("th")

listElemet.id = item.model

listElemet.innerHTML = item.name

document.getElementById("menu").appendChild(listElemet)

})

// add event handler for mouseover on each img:

teslaModels.forEach(mouseOverHandler);

function mouseOverHandler(item, index)

{

var elem = document.getElementById(item.model);

elem.onmouseover = function(){

var details = item;

// if found, update DOM with information from array

if(details!=null)

{

document.getElementById("data-table").removeAttribute('hidden');

document.getElementById("model").innerHTML = details.name;

document.getElementById("picture").innerHTML = '<img src="img/'+details.model+'.png"/>';

document.getElementById("price").innerHTML = "$"+details.price;

document.getElementById("year").innerHTML = details.year;

}

}

}

};

index.html

<!DOCTYPE html>

<html>

<head>

<title>Tesla Catalog</title>

<script type="text/javascript" src="cars.js"></script>

<style>

table {

text-align: center;

width: 100%;

}

td {

font-size: xx-large;

}

th {

font-size: x-large;

background-color: #4CAF50;

color: white;

}

th:hover {

background-color: #4CA;

}

</style>

</head>

<body>

<h3 style="text-align:center">Hover a Tesla Type</h3><br>

<table>

<tr id="menu">

</tr>

</table><br>

<hr><br>

<table class="table" id="data-table" hidden>

<tr><td>Name</td><td id="model"></td></tr>

<tr><td>Look</td><td id="picture"></td></tr>

<tr><td>Price</td><td id="price"></td></tr>

<tr><td>Year</td><td id="year"></td></tr>

</table>

</body>

</html>

**b) JavaScript (JSON):** Create a JSON object to store *either* the colour names *or* font names and its numerical values. Use this colour *or* font object to display details on a web page.

**Python and JavaScript - Form Validation:** Design any case study that creates and validates a HTML form. On the server side, use Python to display the details entered by the user at the client side along with appropriate validation messages. Use Display appropriate messages if error occurs, and redirect to the other HTML page if successful.

loginTest.py

from flask import Flask, redirect, url\_for, render\_template, request

# Initialize the Flask application

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

return render\_template('log\_in.html')

@app.route('/log\_in',methods = ['POST','GET'])

def log\_in():

if request.method == 'POST':

if request.form['usn'] == 'admin' and request.form['pass'] == 'admin' :

return redirect(url\_for('success'))

else:

return render\_template('log\_in.html', msg='incorrect entry')

return redirect(url\_for('index'))

@app.route('/success')

def success():

return 'logged in successfully'

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

login\_in.html

<!DOCTYPE html>

<html>

<head>

<title>Form Details</title>

</head>

<body>

<h1> \*\*\*Login Details\*\*\* </h1>

<h5> {{msg}} </h5> <!-- Error Message displayed here -->

<!-- Form for details -->

<form action="{{url\_for('log\_in')}}" method="POST" >

USERNAME : <input name = "usn" type="text" /><br>

PASS : <input name = "pass" type="text" /><br>

<input name = "\*\*\*Login Details\*\*\*" type = "submit" />

</form>

</body>

</html>

1. **Python and JavaScript - ATM Application:** Design a HTML form that displays user’s current balance, an input field to enter amount and buttons to withdraw or deposit money. Validate the form such that

**i)** Negative amount cannot be entered and Users cannot withdraw more than 5000 at one time.

**ii)** Users cannot withdraw an amount greater than their balance and cannot deposit more than 10000 at one time. Also users can perform at most 5 transactions. Update the balance accordingly and ensure relevant data is not lost on closing the browser.

**Index.html**

<!DOCTYPE html>

<html>

<head>

<title>ATM</title>

</head>

<body>

<h1> ATM </h1>

<h3> Your balance is : {{balance}} </h3> <!-- Account Balance displayed here -->

<h5>{{msg}}</h5> <!-- Error Message displayed here -->

<!-- Form for amount details, with buttons to Deposit or Withdraw -->

<form action="{{url\_for('index')}}" method="POST" id="atm">

Amount : <input name = "amount" type="number" required /><br>

<input type="submit" name="action" value="Withdraw" onclick="return validateForm(this)">

<input type="submit" name="action" value="Deposit" onclick="return validateForm(this)">

</form>

<!-- Javascript to validate amount -->

<script>

function validateForm(button)

{

var form = document.getElementById("atm");

// Checks if amount field is empty

if (form.amount.value == "" ) {

alert("Amount required");

return false;

}

// Checks if amount entered is negative

if (parseInt(form.amount.value) < 0) {

alert("Cannot enter negative amount");

return false;

}

// Checks if user clicked on Withdraw and amount is greater than 5000

if(button.value == "Withdraw" && parseInt(form.amount.value) > 5000)

{

alert("Cannot withdraw more than 5000");

return false;

}

// Checks if user clicked on Deposit and amount is greater than 10000

if(button.value == "Deposit" && parseInt(form.amount.value) > 10000)

{

alert("Cannot deposit more than 10000");

return false;

}

// Returns true if amount is valid

return true;

}

</script>

</body>

</html>

**Application.py**

from flask import Flask, redirect, render\_template, request, url\_for, session

import time

import re

app = Flask(\_\_name\_\_)

# Secret key for sessions

app.secret\_key = "secret"

@app.route("/", methods=['GET', 'POST'])

def index():

# Initialises balance and transaction count from session variables

# If session keys do not exist, KeyError raised - (occurs the first time program is run)

# Session variables initialised

# To clear session - session.clear()

try:

balance = session["balance"]

count = session["count"]

except KeyError:

count = session["count"] = 0

balance = session["balance"] = 8000

if request.method == "GET":

return render\_template("index.html", balance=balance, msg="")

if request.method == "POST":

# Checks if amount field is empty

if request.form["amount"] == "" :

msg = "Amount is required"

return render\_template("index.html", balance=balance, msg=msg)

# Checks if amount entered is negative

if int(request.form["amount"]) < 0 :

msg = "Cannot enter negative amount"

return render\_template("index.html", balance=balance, msg=msg)

# Checks if number of transactions is equal to 5

if session["count"] == 5:

msg = "5 transactions complete"

return render\_template("index.html", balance=balance, msg=msg)

# Checks if user clicked on Withdraw

if request.form["action"] == 'Withdraw':

# Checks if amount is greater than balance

if int(request.form["amount"]) > session["balance"] :

msg = "Cannot withdraw amount greater than balance"

return render\_template("index.html", balance=balance, msg=msg)

# Checks if amount is greater than 5000

elif int(request.form["amount"]) > 5000 :

msg = "Cannot withdraw amount greater than 5000"

return render\_template("index.html", balance=balance, msg=msg)

# Deducts amount entered from balance and stores in session

# Updates the number of transactions by one

else:

balance = balance - int(request.form["amount"])

session["balance"] = balance

session["count"] = session["count"] + 1

msg = "Money Withdrawn"

return render\_template("index.html", balance=balance, msg=msg)

# Checks if user clicked on Deposit

elif request.form["action"] == 'Deposit':

# Checks if amount is greater than 10000

if int(request.form["amount"]) > 10000 :

msg = "Cannot deposit amount greater than 10000"

return render\_template("index.html", balance=balance, msg=msg)

# Adds amount entered to balance and stores in session

# Updates the number of transactions by one

else:

balance = balance + int(request.form["amount"])

session["balance"] = balance

session["count"] = session["count"] + 1

msg = "Money Deposited"

return render\_template("index.html", balance=balance, msg=msg)

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

**Python and JavaScript - Shopping Cart Application:** Design a simple Shopping Cart application which allows users to add items to their cart from a list of products. Allow users to view their cart (items and quantities of each). Ensure that items in the cart persist even after closing the application. On selecting buy, print out a bill of items in the cart. Perform any necessary validation. Demonstrate data persistence even after the browser is closed.

**bill.html**

<!DOCTYPE html>

<html>

<head>

<title>Shopping Cart</title>

</head>

<body>

<h1> Bill </h1>

<!-- Table displaying item, quantity and price -->

<table border="1" width="50%">

<tr>

<td>Item</td>

<td>Quantity</td>

<td>Price</td>

</tr>

{% for item in cart %}

<tr>

<td>{{item["name"]}}</td>

<td>{{item["quantity"]}}</td>

<td>{{item["price"]}}</td>

</tr>

{% endfor %}

</table>

<!-- Total Bill Amount -->

<p>Total Bill Amount = Rs. {{amount}} </p>

<!-- Link to view store -->

<a href="{{url\_for('store')}}"> Continue Shopping </a><br>

</body>

</html>

**cart.html**

<!DOCTYPE html>

<html>

<head>

<title>Shopping Cart</title>

</head>

<body>

<h1> Items in Cart </h1>

<!-- View Shopping Cart - item and quantity -->

{% for item in cart %}

{{item["name"]}} : {{item["quantity"]}}<br>

{% endfor %}

<br>

<!-- Link to view store -->

<a href="{{url\_for('store')}}"> Continue Shopping </a><br><br>

<!-- Link to view bill -->

<a href="{{url\_for('buy')}}"> View Bill </a>

</body>

</html>

**store.html**

<!DOCTYPE html>

<html>

<head>

<title>Add to Cart</title>

</head>

<body>

<h1> Store Items </h1>

<!-- Form containing Items in Store -->

<!-- Add items to cart by entering quantity for item (no negative numbers) -->

<form action = "{{url\_for('store')}}" method="POST">

<input name = "eggs" type = "number" min="0" value="0"/>Eggs - Rs.5 <br>

<input name = "milk" type = "number" min="0" value="0"/>Milk - Rs.12 <br>

<input name = "bread" type = "number" min="0" value="0"/>Bread - Rs.22 <br><br>

<input value = "Add to Cart" type = "submit"/>

</form><br>

<!-- Link to view cart -->

<a href="{{url\_for('cart')}}"> View your Shopping Cart </a>

</body>

</html>

**application.py**

from flask import Flask, redirect, render\_template, request, session, url\_for

app = Flask(\_\_name\_\_)

# Secret key for sessions

app.secret\_key = "secret"

@app.route("/", methods=["GET", "POST"])

def store():

if request.method == "GET":

return render\_template("store.html")

# For each item in store, checks if item is in session

# If item is in session, increments the count by value entered in the form

# If item is not in session, initialises the count by value entered in the form

# Redirects to the HTML page to view Shopping Cart

if request.method == "POST":

for item in ["eggs", "milk", "bread"]:

if item not in session :

session[item] = int(request.form[item])

else:

session[item] += int(request.form[item])

return redirect(url\_for("cart"))

@app.route("/cart", methods=["GET", "POST"])

def cart():

# Creates a list of dictionaries containing each it sem in cart and its quantity

# Displays this list in a HTML page

cart = []

for item in ["eggs", "milk", "bread"]:

cart.append({"name":item.capitalize(), "quantity":session[item]})

return render\_template("cart.html", cart=cart)

@app.route("/buy", methods=["GET", "POST"])

def buy():

# Total amount initialised to 0

amount = 0

# Index for list containing prices of items

index = 0

# Prices of every item

prices = [5, 12, 22]

# Creates a list of dictionaries containing each item in cart, its quantity, and cost(price\*qty)

# Calculates total bill amount

# Displays the bill in a HTML page

cart = []

for item in ["eggs", "milk", "bread"]:

row = {}

row["name"] = item.capitalize()

row["quantity"] = session[item]

row["price"] = prices[index] \* session[item]

amount = amount + row["price"]

cart.append(row)

index = index + 1

return render\_template("bill.html", cart=cart, amount=amount)

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

**Python and JavaScript - Student Registration Application:** Design a HTML form that displays:

* Two text fields to input the user’s USN and Date of Birth.
* Three text boxes to input three marks.

Validate the data entry on the *client side* *using Javascrip*t so that null values are not accepted for all the five text boxes.

Validate the entry on *server-side using Pytho*n to ensure that USN is accepted in a proper pattern as well as date validations are done. Calculate the average using Python on server-side and display the result.

**index.html**

<!DOCTYPE html>

<html>

<head>

<title>Form Details</title>

</head>

<body>

<h1> \*\*\*Register here\*\*\* </h1>

<h5> {{msg}} </h5> <!-- Error Message displayed here -->

<!-- Form for details -->

<form action="{{url\_for('indexJS')}}" method="POST" >

USN (Eg. 1MS02IS001)\*\*\*: <input name = "usn" type="text" /><br>

D.O.B (Eg. dd/mm/yyyy)\*\*\* : <input name = "dob" type="text" /><br>

<input name = "\*\*\*Register\*\*\*" type = "submit" />

</form>

</body>

</html>

**success.html**

<!DOCTYPE html>

<html>

<head>

<title>Form Details</title>

</head>

<body>

<h1> Registration Successful\*\*\*</h1>

</body>

</html>

**application.py**

from flask import Flask, redirect, render\_template, request, url\_for

import time #time and date related functions

import re #regular expressions

app = Flask(\_\_name\_\_)

@app.route("/", methods=['GET', 'POST'])

def indexJS():

if request.method == "GET":

msg="\*\*\*hello user enter your details\*\*\*"

return render\_template("indexJS.html",msg=msg)

if request.method == "POST":

#Check if form fields are empty

if request.form["usn"] == "" or request.form["dob"] == "" :

msg = "\*\*\*All form fields are required\*\*\*"

return render\_template("indexJS.html", msg=msg)

#Check if date entered in dd/mm/yyyy format and is not an invalid date Eg. 31/11/2016

#Use strptime() function which raises an exception if date is invalid

try:

time.strptime(request.form["dob"],"%d/%m/%Y")

except ValueError:

msg = "\*\*\*Date is invalid\*\*\*"

return render\_template("indexJS.html", msg=msg)

#Regex for USN

usn\_pattern = "^[1][A-se 3Z][A-Z][0-9][0-9][A-Z][A-Z][0-9][0-9][0-9]$"

#Check if entered USN matches Regex

if not re.match(usn\_pattern, request.form["usn"]) :

msg = "\*\*\*USN format invalid\*\*\*"

return render\_template("indexJS.html", msg=msg)

#If form fields are valid return success HTML page

l=[request.form["usn"],request.form["dob"]]

return render\_template("success.html", l=l)

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

# app.run(host='192.168.43.177', port='8000', debug=True)