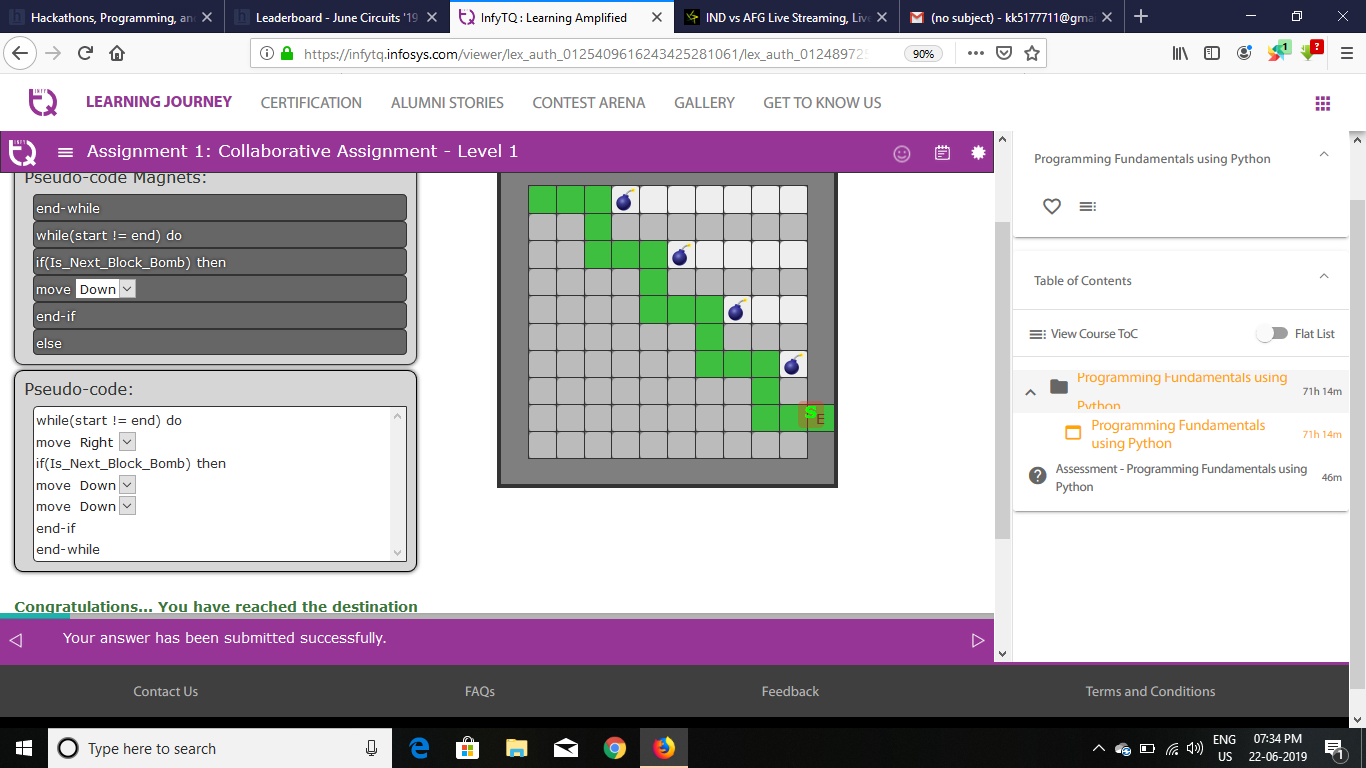
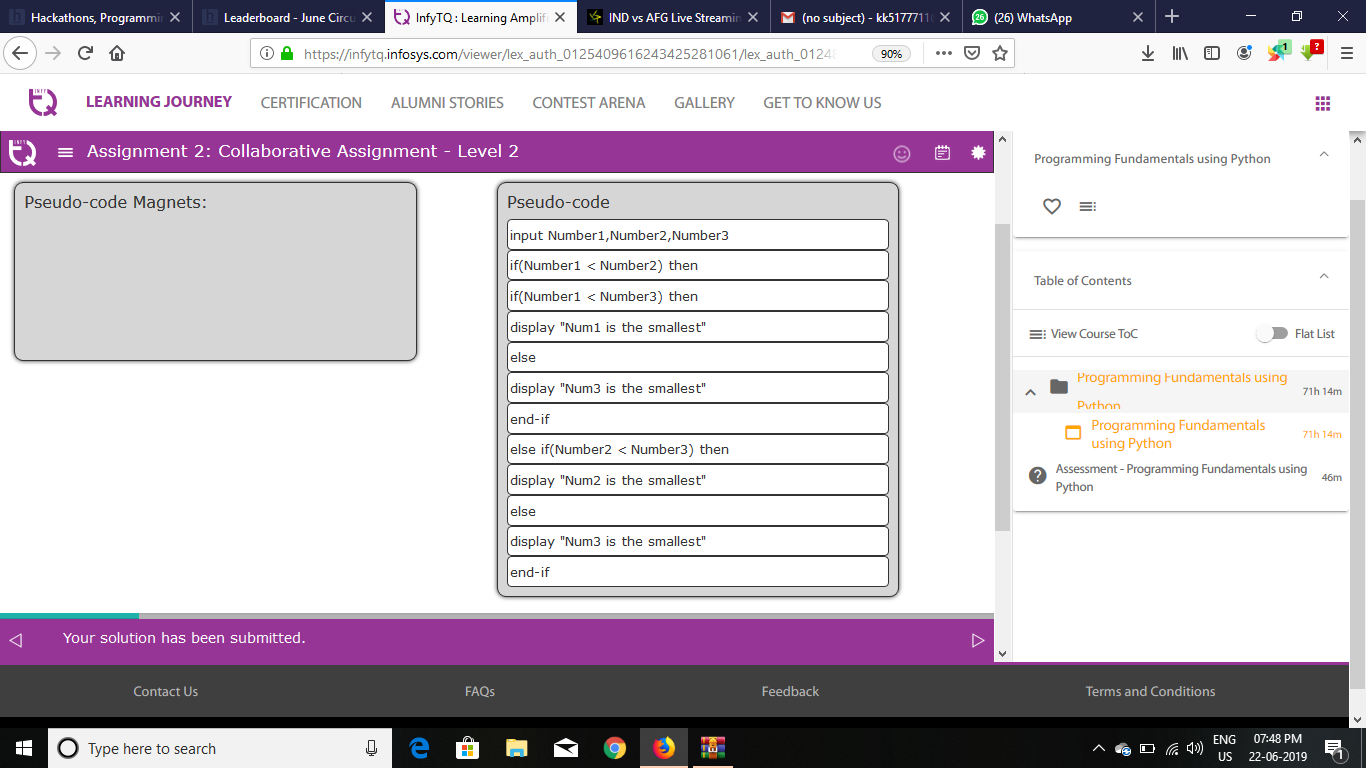
#Assignment - 1



#Assignment-2



#PF-Assgn-3

#This verification is based on string match.

mileage=12

amount\_per\_litre=40

distance\_one\_way=190

per\_head\_cost=0

divisible\_by\_five=False

#Start writing your code from here

#Populate the variables: per\_head\_cost and divisible\_by\_five

per\_head\_cost = (distance\_one\_way/mileage)\*amount\_per\_litre/2

if per\_head\_cost%5==0:

divisible\_by\_five=True

#Do not modify the below print statements for verification to work

print(per\_head\_cost)

print(divisible\_by\_five)

#PF-Assgn-4

#This verification is based on string match.

principal=7800

rate\_of\_interest=7.7

time=26

interest=0

#Start writing your code from here

#Populate the variable: interest

interest=(principal \*rate\_of\_interest\* time)/100

#Do not modify the below print statement for verification to work

print(interest)

#PF-Exer-7

def calculate\_total\_ticket\_cost(no\_of\_adults, no\_of\_children):

total\_ticket\_cost=0

#Write your logic here

return total\_ticket\_cost

#Provide different values for no\_of\_adults, no\_of\_children and test your program

total\_ticket\_cost=calculate\_total\_ticket\_cost(1,2)

print("Total Ticket Cost:",total\_ticket\_cost)

//PF-Exer-8

//This verification is based on string match.

package main

import ("fmt")

func main(){

var finalFee int

//Write your program logic here

//Populate the variable: finalFee

var marks int=70

var fee int=25000

var extra int=1500

finalFee=fee-marks\*fee/200 + extra

//Do not modify the below print statement for verification to work

fmt.Println(finalFee)

}

//PF-Exer-9

noOfFlightsTakeOff=100

noOfFlightsLanded=110

seatingCapacityTakeOff=150

seatingCapacityLanded=185

totalCookies=0

//Write your code here

//Populate the variable: totalCookies

totalCookies = noOfFlightsLanded\*seatingCapacityLanded+2\*noOfFlightsTakeOff\*seatingCapacityTakeOff

//Do not modify the below print statement for verification to work

console.log(totalCookies)

import turtle

wn = turtle.Screen() # creates a graphics window

wn.setup(540,508) # set window dimension

alex = turtle.Turtle() # create a turtle named alex

alex.shape("turtle") # alex looks like a turtle

alex.color("blue") # alex has a color

'''

alex.backward(50) # alex moves 50 positions backward

alex.forward(50) # alex moves 50 positions forward

alex.right(60) # alex turns 60 degrees right

alex.left(60) # alex turns 60 degrees left

alex.write("Hello") # alex says "Hello"

'''

#Write the logic to take the turtle to its destination

#Refer the statements given above to draw the pattern

#Provide different values and test your program

destination="west"

if destination=="south":

alex.right(90)

alex.forward(100)

elif destination=="north":

alex.left(90)

alex.forward(100)

elif destination=="west":

alex.left(180)

alex.forward(100)

else:

alex.forward(100)

#PF-Exer-11

def display(num):

message=""

#write your logic here

if num%15==0:

message="zoom"

elif num%3==0:

message="zip"

elif num%5==0:

message="zap"

else:

message="Invalid"

return message

#Provide different values for num and test your program

message=display(9)

print(message)

//PF-Exer-12

//This verification is based on string match.

package main

import ("fmt")

func main(){

var a int = 3

var b int = 4

var c int = 1

var m int = 0

if(a>m){

m=a

}

if(b>m){

m=b

}

if(c>m){

m=c

}

fmt.Println(m)

}

//PF-Exer-13

num1=5

num2=5

//Write your code here

if(num1==num2)

{

console.log(num1+num2)

}

else

{

console.log(2\*(num1+num2))

}

score = 55

grade = ""

if score>0 and score<64:

grade="D"

elif score>65 and score<72:

grade="C"

elif score>73 and score<79:

grade="B"

elif score>80 and score<100:

grade="A"

else:

grade="Z"

print(grade)

#PF-Assgn-15

def find\_product(num1,num2,num3):

product=1

#write your logic here

if num3==7:

product=-1

elif num2==7:

product=num3

elif num1==7:

product=num2\*num3

else:

product=num1\*num3\*num2

return product

#Provide different values for num1, num2, num3 and test your program

product=find\_product(7,6,2)

print(product)

#PF-Assgn-16

def make\_amount(rupees\_to\_make,no\_of\_five,no\_of\_one):

five\_needed=0

one\_needed=0

#Start writing your code here

#Populate the variables: five\_needed and one\_needed

five\_needed = rupees\_to\_make//5

one\_needed = rupees\_to\_make%5

# Use the below given print statements to display the output

# Also, do not modify them for verification to work

if five\_needed<=no\_of\_five and one\_needed<=no\_of\_one:

print("No. of Five needed :", five\_needed)

print("No. of One needed :", one\_needed)

elif five\_needed>no\_of\_five:

x = (five\_needed - no\_of\_five)\*5

if x<=(no\_of\_one - one\_needed):

print("No. of Five needed :", no\_of\_five)

print("No. of One needed :", one\_needed+x)

else:

print(-1)

else:

print(-1)

#Provide different values for rupees\_to\_make,no\_of\_five,no\_of\_one and test your program

make\_amount(28,8,5)

#PF-Tryout

def find\_new\_salary(current\_salary,job\_level):

# write your logic here

if job\_level==3:

new\_salary=current\_salary+current\_salary\*15/100

elif job\_level=="4":

new\_salary=current\_salary+current\_salary\*7/100

elif job\_level=="5":

new\_salary=current\_salary+current\_salary\*5/100

else:

new\_salary=current\_salary

return new\_salary

# provide different values for current\_salary and job\_level and test yor program

new\_salary=find\_new\_salary(15000,3)

print(new\_salary)

#PF-Tryout

def convert\_currency(amount\_needed\_inr,current\_currency\_name):

current\_currency\_amount=0

#write your logic here

if current\_currency\_name=="Euro":

d=0.01417

elif current\_currency\_name=="British Pound":

d=0.0100

elif current\_currency\_name=="Australian Dollar":

d=0.02140

elif current\_currency\_name=="Canadian Dollar":

d=0.02027

else:

return -1

current\_currency\_amount = d\*amount\_needed\_inr

return current\_currency\_amount

#Provide different values for amount\_needed\_inr,current\_currency\_name and test your program

currency\_needed=convert\_currency(2000,"Euro")

if(currency\_needed!= -1):

print(currency\_needed )

else:

print("Invalid currency name")

#PF-Assgn-19

def calculate\_bill\_amount(food\_type,quantity\_ordered,distance\_in\_kms):

bill\_amount=0

if (food\_type=="N" or food\_type=="V" )and quantity\_ordered>0 and distance\_in\_kms>0:

if food\_type == "N":

bill\_amount = 150\*quantity\_ordered

else:

bill\_amount = 120\*quantity\_ordered

if distance\_in\_kms<=3:

distance\_in\_kms=0

else:

distance\_in\_kms-=3

if distance\_in\_kms<=3:

charge = 3\*distance\_in\_kms

distance\_in\_kms=0

else:

charge = 3\*3

distance\_in\_kms-=3

if distance\_in\_kms>0:

charge+=distance\_in\_kms\*(3+3)

return bill\_amount+charge

else:

return -1

#Provide different values for food\_type,quantity\_ordered,distance\_in\_kms and test your program

bill\_amount=calculate\_bill\_amount("N",0,5)

print(bill\_amount)

#PF-Assgn-20

def calculate\_loan(account\_number,salary,account\_balance,loan\_type,loan\_amount\_expected,customer\_emi\_expected):

eligible\_loan\_amount=0

bank\_emi\_expected=0

eligible\_loan\_amount=0

a=b=c=0

if account\_number//1000==1:

a=1

if account\_balance>=100000:

b=1

if loan\_type=="Car" and salary>25000:

eligible\_loan\_amount = 500000

bank\_emi\_expected = 36

elif loan\_type=="House" and salary>50000:

eligible\_loan\_amount = 6000000

bank\_emi\_expected = 60

elif loan\_type=="Business" and salary>75000:

eligible\_loan\_amount = 7500000

bank\_emi\_expected = 84

else:

c=1

#Use the below given print statements to display the output, in case of success

if a==1 and b==1 and eligible\_loan\_amount>=loan\_amount\_expected and customer\_emi\_expected<=bank\_emi\_expected:

print("Account number:", account\_number)

print("The customer can avail the amount of Rs.", eligible\_loan\_amount)

print("Eligible EMIs :", bank\_emi\_expected)

print("Requested loan amount:", loan\_amount\_expected)

print("Requested EMI's:",customer\_emi\_expected)

#Use the below given print statements to display the output, in case of invalid data.

if a==0:

print("Invalid account number")

elif c==1:

print("Invalid loan type or salary")

elif b==0:

print("Insufficient account balance")

elif eligible\_loan\_amount<loan\_amount\_expected or customer\_emi\_expected>bank\_emi\_expected:

print("The customer is not eligible for the loan")

# Also, do not modify the above print statements for verification to work

#Test your code for different values and observe the results

calculate\_loan(1001,40000,250000,"Car",300000,30)

#PF-Assgn-21

#PF-Tryout

def generate\_next\_date(day,month,year):

#Start writing your code here

if (year % 400 == 0):

leap\_year = True

elif (year % 100 == 0):

leap\_year = False

elif (year % 4 == 0):

leap\_year = True

else:

leap\_year = False

if month in (1, 3, 5, 7, 8, 10, 12):

month\_length = 31

elif month == 2:

if leap\_year:

month\_length = 29

else:

month\_length = 28

else:

month\_length = 30

if day < month\_length:

day += 1

else:

day = 1

if month == 12:

month = 1

year += 1

else:

month += 1

print(str(day)+"-"+str(month)+"-"+str(year))

generate\_next\_date(27,2,1996)

#PF-Assgn-22

def find\_leap\_years(given\_year):

list\_of\_leap\_years = []

count=1

while count<=15:

if given\_year%100==0 and given\_year%400!=0:

given\_year+=1

continue

if given\_year%4==0:

count+=1

list\_of\_leap\_years.append(given\_year)

given\_year+=1

return list\_of\_leap\_years

list\_of\_leap\_years=find\_leap\_years(1580)

print(list\_of\_leap\_years)

#PF-Assgn-23

def calculate\_bill\_amount(gems\_list, price\_list, reqd\_gems,reqd\_quantity):

bill\_amount=0

for i in range(len(reqd\_gems)):

reqd = reqd\_gems[i]

if reqd not in gems\_list:

return -1

index = gems\_list.index(reqd)

bill\_amount += reqd\_quantity[i]\*price\_list[index]

if bill\_amount>30000:

bill\_amount -=0.05\*bill\_amount

return bill\_amount

#List of gems available in the store

gems\_list=["Emerald","Ivory","Jasper","Ruby","Garnet"]

#Price of gems available in the store. gems\_list and price\_list have one-to-one correspondence

price\_list=[1760,2119,1599,3920,3999]

#List of gems required by the customer

reqd\_gems=["Ivory","Emerald","Garnet"]

#Quantity of gems required by the customer. reqd\_gems and reqd\_quantity have one-to-one correspondence

reqd\_quantity=[3,10,12]

bill\_amount=calculate\_bill\_amount(gems\_list, price\_list, reqd\_gems, reqd\_quantity)

print(bill\_amount)

#PF-Assgn-24

def form\_triangle(num1,num2,num3):

#Do not change the messages provided below

success="Triangle can be formed"

failure="Triangle can't be formed"

#Write your logic here

if num1>=num2+num3 or num2>=num1+num3 or num3>=num1+num2:

return failure

#Use the following messages to return the result wherever necessary

return success

#Provide different values for the variables, num1, num2, num3 and test your program

num1=3

num2=3

num3=5

form\_triangle(num1, num2, num3)

#PF-Assgn-25

#debug the below code

counter1=0

counter2=5

while(counter1 < 5):

star=""

while(counter2>counter1):

star=star+ "\*"

counter2-=1

print(star)

counter2=5

counter1+=1

#PF-Assgn-26

def solve(heads,legs):

error\_msg="No solution"

chicken\_count=0

rabbit\_count=0

#Start writing your code here

#Populate the variables: chicken\_count and rabbit\_count

if (4\*heads-legs)%2==0:

chicken\_count = (4\*heads-legs)//2

rabbit\_count = heads - chicken\_count

if rabbit\_count>=0 and chicken\_count>=0:

print(chicken\_count,rabbit\_count)

else:

print(error\_msg)

else:

print(error\_msg)

# Use the below given print statements to display the output

# Also, do not modify them for verification to work

#Provide different values for heads and legs and test your program

solve(38,131)

#PF-Assgn-26-level 2

import turtle # allows us to use the turtles library

wn = turtle.Screen() # creates a graphics window

wn.setup(500,500) # set window dimension

alex = turtle.Turtle() # create a turtle named alex

alex.shape("turtle") # alex looks like a turtle

alex.color("Green") # alex has a color

#alex.right(60) # alex turns 60 degrees left

#draws circles

for counter in range(1,5):

alex.circle(20\*counter)

alex.color("Blue") # alex has a color

alex.right(120) # alex turns 60 degrees right

#alex.left(60) # alex turns 60 degrees left

#alex.circle(50) # draws a circle of radius 50

#draws circles

for counter in range(1,5):

alex.circle(20\*counter)

alex.color("Red") # alex has a color

alex.right(120) # alex turns 60 degrees right

#alex.left(60) # alex turns 60 degrees left

#alex.circle(50) # draws a circle of radius 50

#draws circles

for counter in range(1,5):

alex.circle(20\*counter)

#Write the logic to create the given pattern

#Refer the statements given above to draw the pattern

#PF-Assgn-28

def find\_max(num1, num2):

max\_num=-1

for i in range(num1+1,num2+1):

j = i

if i%5==0:

c = 0

sum=0

while i>0:

sum+=i%10

i//=10

c+=1

if c==2 and sum%3==0:

max\_num = j

return max\_num

#Provide different values for num1 and num2 and test your program.

max\_num=find\_max(10,15)

print(max\_num)

#PF-Exer-22

def generate\_ticket(airline,source,destination,no\_of\_passengers):

ticket\_number\_list=[]

#Write your logic here

ticket = airline+":"+source[:3]+":"+destination[:3]+":"

for i in range(no\_of\_passengers):

ticket1 = ticket+str(101+i)

ticket\_number\_list.append(ticket1)

#Use the below return statement wherever applicable

if no\_of\_passengers>5:

return ticket\_number\_list[-5:]

return ticket\_number\_list

#Provide different values for airline,source,destination,no\_of\_passengers and test your program

print(generate\_ticket("AI","Bangalore","London",7))

#PF-Exer-23

def translate(bilingual\_dict,english\_words\_list):

#Write your logic here

swedish\_words\_list = []

for i in english\_words\_list:

swedish\_words\_list.append(bilingual\_dict[i])

return swedish\_words\_list

bilingual\_dict= {"merry":"god", "christmas":"jul", "and":"och", "happy":"gott", "new":"nytt", "year":"ar"}

english\_words\_list=["merry","christmas"]

print("The bilingual dict is:",bilingual\_dict)

print("The english words are:",english\_words\_list)

swedish\_words\_list=translate(bilingual\_dict, english\_words\_list)

print("The equivalent swedish words are:",swedish\_words\_list)

//PF-Exer-24

//This verification is based on string match.

package main

import ("fmt")

func find\_square(val int) int {

return (val\*val)

}

func main() {

var n int = 3;

var sqr int = find\_square(n)

fmt.Println(sqr)

}

//JS-Exer-25

//Start writing your code here

function find\_sum(n){

var sum=0;

for(i=1;i<=n;i++){

sum+=i;

}

return sum;

}

console.log(find\_sum(10))

//console.log(5)

#PF-Assgn-29

def calculate(distance,no\_of\_passengers):

#Remove pass and write your logic here

profit = no\_of\_passengers\*80-(distance/10)\*70

if profit>=0:

return profit

return -1

#Provide different values for distance, no\_of\_passenger and test your program

distance=20

no\_of\_passengers=50

print(calculate(distance,no\_of\_passengers))

#PF-Assgn-30

def encode(message):

message+="a"

c=1

st=''

for i in range(len(message)-1):

if message[i+1]==message[i]:

c+=1

else:

st+=str(c)+message[i]

c=1

return st

#Remove pass and write your logic here

#Provide different values for message and test your program

encoded\_message=encode("ABBBBCCCCCCCCAB")

print(encoded\_message)

#PF-Assgn-31

def check\_palindrome(word):

i=0

j=len(word)-1

while i<=j:

if word[i]!=word[j]:

return False

i+=1

j-=1

return True

status=check\_palindrome("malayalam")

if(status):

print("word is palindrome")

else:

print("word is not palindrome")

#PF-Assgn-32

def max\_visited\_speciality(patient\_medical\_speciality\_list, medical\_speciality):

counts = {}

for \_, speciality in zip(patient\_medical\_speciality\_list[::2], patient\_medical\_speciality\_list[1::2]):

counts[speciality] = counts.get(speciality, 0) + 1

most\_visited\_speciality = max(medical\_speciality, key=lambda e: counts.get(e, 0))

return medical\_speciality[most\_visited\_speciality]

# provide different values in the list and test your program

patient\_medical\_speciality\_list = [301, 'P', 302, 'P', 305, 'P', 401, 'E', 656, 'E']

medical\_speciality = {"P": "Pediatrics", "O": "Orthopedics", "E": "ENT"}

speciality = max\_visited\_speciality(patient\_medical\_speciality\_list, medical\_speciality)

print(speciality)

#PF-Assgn-33

def find\_common\_characters(msg1,msg2):

st =""

for i in msg1:

if i in msg2 and i not in st:

st+=i

if len(st)>0:

return st

return -1

msg1="I like Python"

msg2="Java is a very popular language"

common\_characters=find\_common\_characters("MOTO","apple")

print(common\_characters)

#PF-Exer-26

def factorial(number):

fact = 1

if number==0:

return 1

for i in range(number):

fact\*=(i+1)

return fact

def find\_strong\_numbers(num\_list):

ls = []

for i in num\_list:

j = i

s = 0

while i>0:

s += factorial(i%10)

i//=10

if s==j and j!=0:

ls.append(j)

return ls

num\_list=[145, 375, 100, 2, 10, 40585, 0]

strong\_num\_list=find\_strong\_numbers(num\_list)

print(strong\_num\_list)

#PF-Tryout

account\_list=[1001,1002,1003,1004,1005]

balance\_list=[2500,10000,7000,1500,500]

def Transaction(account\_list,balance\_list,account\_number,transaction\_type,amount):

flag=None

if(transaction\_type=="Withdraw"):

for index in range(0,len(account\_list)):

if(account\_list[index]==account\_number):

flag=True

value=index

if(flag==True):

balance=balance\_list[value]

new\_balance=balance-amount

if(new\_balance >= 500):

balance\_list[value]=new\_balance

print("Transaction completed successfully")

print("Balance Amount :", new\_balance)

else:

print("Insufficient Balance")

else:

print("Invalid Account number")

elif(transaction\_type=="Deposit"):

for index in range(0,len(account\_list)):

if(account\_list[index]==account\_number):

flag=True

value=index

if(flag==True):

balance=balance\_list[value]

new\_balance=balance+amount

balance\_list[value]=new\_balance

print("Transaction completed successfully")

print("Balance Amount :", new\_balance)

else:

print("Invalid Account number")

elif(transaction\_type=="Balance Enquiry"):

for index in range(0,len(account\_list)):

if(account\_list[index]==account\_number):

flag=True

value=index

if(flag==True):

balance=balance\_list[value]

print(balance)

else:

print("Invalid Account number")

else:

print("Invalid Transaction Type")

amount=1000

account\_number=1003

transaction\_type="Withdraw"

Transaction(account\_list,balance\_list,account\_number,transaction\_type,amount)

#PF-Exer-28

#This method accepts the name of winner of each match of the day

def find\_winner\_of\_the\_day(\*match\_tuple):

c1 = 0

c2 = 0

for i in match\_tuple:

if i=="Team1":

c1+=1

else:

c2+=1

if c1 >c2:

return "Team1"

elif c1<c2:

return "Team2"

else:

return "Tie"

#Invoke the function with each of the print statements given below

#print(find\_winner\_of\_the\_day("Team1","Team2","Team1"))

print(find\_winner\_of\_the\_day("Team1","Team2","Team1","Team2","Team2"))

#PF-Exer-29

def merge\_lists(list1,list2):

return list1+list2

return new\_merge\_list

def sort\_list(merged\_list):

#Write your logic here

merged\_list.sort()

return merged\_list

#Provide different values for list1 and list2 and test your program

merged\_list=merge\_lists(list1=[1,2,3,4,1] ,list2=[2,3,4,5,4,6])

print(merged\_list)

sorted\_merged\_list=sort\_list(merged\_list)

print(sorted\_merged\_list)

#PF-Assgn-34

def find\_pairs\_of\_numbers(num\_list,n):

c = 0

#Remove pass and write your logic here

l = len(num\_list)

for i in range(l):

for j in range(i+1,l):

if num\_list[i]+num\_list[j]==n:

c+=1

return c

num\_list=[1, 2, 4, 5, 6]

n=6

print(find\_pairs\_of\_numbers(num\_list,n))

#PF-Assgn-35

#Global variable

list\_of\_marks=(12,18,25,24,2,5,18,20,20,21)

def find\_more\_than\_average():

s = sum(list\_of\_marks)

n = len(list\_of\_marks)

avg = s/n

s=0

for i in list\_of\_marks:

if i>avg:

s+=1

return (s/n)\*100

#Remove pass and write your logic here

def sort\_marks():

return sorted(list\_of\_marks)

#Remove pass and write your logic here

def generate\_frequency():

ls = [0 for i in range(25+1)]

for i in list\_of\_marks:

ls[i]+=1

return ls

#Remove pass and write your logic here

print(find\_more\_than\_average())

print(generate\_frequency())

print(sort\_marks())

#PF-Assgn-36

def create\_largest\_number(number\_list):

number\_list.sort(reverse=True)

st = ""

for i in number\_list:

st+=str(i)

return int(st)

number\_list=[23,45,67]

largest\_number=create\_largest\_number(number\_list)

print(largest\_number)

#PF-Assgn-37

#Global variables

child\_id=(10,20,30,40,50)

chocolates\_received=[12,5,3,4,6]

def calculate\_total\_chocolates():

s = 0

for i in chocolates\_received:

s+=i

return s

def reward\_child(child\_id\_rewarded,extra\_chocolates):

if extra\_chocolates<1:

print("Extra chocolates is less than 1")

return

if child\_id\_rewarded not in child\_id:

print("Child id is invalid")

return

index = child\_id.index(child\_id\_rewarded)

chocolates\_received[index] = chocolates\_received[index]+extra\_chocolates

print(chocolates\_received)

print(calculate\_total\_chocolates())

#Test your code by passing different values for child\_id\_rewarded,extra\_chocolates

reward\_child(20,2)

#PF-Assgn-38

def check\_double(number):

n=number

tot=0

tot1=0

t=n\*2

while(n>0 and t>0):

dig=n%10

tot+=dig

n//=10

dig1=t%10

tot1+=dig1

t//=10

if(tot==tot1):

return True

else:

return False

print(check\_double(125874))

#PF-Assgn-39

#This verification is based on string match.

#This verification is based on string match.

#Global variables

menu=('Veg Roll','Noodles','Fried Rice','Soup')

quantity\_available=[2,200,3,0]

'''This method accepts the item followed by the quantity required by a customer in the format item1, quantity\_required, item2, quantity\_required etc.'''

def place\_order(\*item\_tuple):

#Remove pass and write your logic here

global quantity\_available

for i in range(0,len(item\_tuple),2):

item = item\_tuple[i]

index = -1

for j in range(len(menu)):

if item==menu[j]:

index = j

break

if index!=-1:

if check\_quantity\_available(index,item\_tuple[i+1]):

print(item,"is available")

else:

print(item,"stock is over")

else:

print(item,"is not available")

'''This method accepts the index position of the item requested by the customer in the quantity\_available list, and the requested quantity of the item.'''

def check\_quantity\_available(index,quantity\_requested):

global quantity\_available

if(quantity\_available[index]>=quantity\_requested):

quantity\_available[index]=quantity\_available[index]-quantity\_requested

return True

else:

return False

#Provide different values for items ordered and test your program

place\_order("Fried Rice",2,"Soup",1 )

#PF-Exer-30

def find\_average(mark\_list):

total = 0

for i in range(0,len(mark\_list)):

try:

total+=mark\_list[i]

except NameError:

print("Name Error")

marks\_avg=total/len(mark\_list)

return marks\_avg

m\_list=[1,2,3,4]

print("Average marks:", find\_average(m\_list))

#PF-Exer-32

def human\_pyramid(no\_of\_people):

if(no\_of\_people==1):

return 1\*(50)

else:

return no\_of\_people\*(50)+human\_pyramid(no\_of\_people-2)

def find\_maximum\_people(max\_weight):

no\_of\_people=1

while max\_weight>=human\_pyramid(no\_of\_people):

no\_of\_people+=2

return no\_of\_people-2

#Provide different values for max\_weight and test your program

max\_people=find\_maximum\_people(1000)

print(max\_people)

#PF-Assgn-40

def is\_palindrome(word):

l = len(word)

if l==0 or l==1:

return True

if (word[0].lower())==(word[-1].lower()):

return is\_palindrome(word[1:l-1])

else:

return False

#Provide different values for word and test your program

result=is\_palindrome("MadAMa")

if(result):

print("The given word is a Palindrome")

else:

print("The given word is not a Palindrome")

#PF-Assgn-41

def find\_ten\_substring(num\_str):

ls = []

j = 0

sum = 0

for i in range(len(num\_str)):

sum += int(num\_str[i])

if sum==10:

ls.append(num\_str[j:i+1])

elif sum>10:

while j<i:

st = int(num\_str[j])

sum -=st

j+=1

if sum==10:

ls.append(num\_str[j:i+1])

if sum<10:

break

return ls

num\_str="234123"

print("The number is:",num\_str)

result\_list=find\_ten\_substring(num\_str)

print(result\_list)

#PF-Assgn-42

def find\_factors(num):

#Accepts a number and returns the list of all the factors of a given number

factors = []

for i in range(2,(num+1)):

if(num%i==0):

factors.append(i)

return factors

def is\_prime(num, i):

#Accepts the number num and num/2 --> i and returns True if the number is prime ,else returns False

if(i==1):

return True

elif(num%i==0):

return False;

else:

return(is\_prime(num,i-1))

def find\_largest\_prime\_factor(list\_of\_factors):

mx = 0

for i in list\_of\_factors:

if is\_prime(i,i//2):

mx = i

return mx

def find\_f(num):

ls = find\_factors(num)

return find\_largest\_prime\_factor(ls)

def find\_g(num):

sum = 0

for i in range(9):

sum+=find\_f(num+i)

return sum

print(find\_g(10))

#PF-Assgn-43

def find\_smallest\_number(num):

#start writing your code here

for value in range(1,1000):

list=[]

for n in range(1,value+1):

if value%n ==0:

list.append(n)

if len(list) == num:

return value

break

num=16

print("The number of divisors :",num)

result=find\_smallest\_number(num)

print("The smallest number having",num," divisors:",result)

#PF-Assgn-44

def find\_duplicates(list\_of\_numbers):

d = []

ls = []

for i in list\_of\_numbers:

if i not in d:

d.append(i)

elif i not in ls:

ls.append(i)

ls.sort()

return ls

list\_of\_numbers=[1,2,2,3,3,3,4,4,4,4]

list\_of\_duplicates=find\_duplicates(list\_of\_numbers)

print(list\_of\_duplicates)

#PF-Assgn-45

#PF-Tryout

def find\_armstrong(number):

temp=0

j = number

while number>0:

remainder=number%10

number=number//10

temp+=(remainder\*remainder\*remainder)

if(j==temp):

return True

return False

number=371

if(find\_armstrong(number)):

print(number,"is an Armstrong number")

else:

print(number,"is not an Armstrong number")

#PF-Assgn-46

def nearest\_palindrome(number):

while True:

number+=1

n = str(number)

j = 0

i = len(n)-1

c = 0

while j<=i:

if n[i]!=n[j]:

c = -1

break

j+=1

i-=1

if c==0:

break

return number

number=12300

print(nearest\_palindrome(number))

#PF-Assgn-47

def encrypt\_sentence(sentence):

vowels = ['a','e','i','o','u','A','E','I','O','U']

ls = sentence.split(" ")

st = ""

for i in range(len(ls)):

st+=" "

a = []

for j in ls[i]:

a.append(j)

if i%2==0:

a.reverse()

for c in a:

st+=c

else:

for c in a:

if c not in vowels:

st+=c

for c in a:

if c in vowels:

st+=c

return st[1:]

sentence="good day"

encrypted\_sentence=encrypt\_sentence(sentence)

print(encrypted\_sentence)

#PF-Assgn-48

def find\_correct(word\_dict):

d = word\_dict

c=0

ac=0

wr=0

for i in d:

if i==d[i]:

c+=1

elif len(i)!=len(d[i]):

wr+=1

else:

k = 0

l = len(i)

for j in range(l):

if i[j]!=d[i][j]:

k+=1

if k>2:

break

if k>2:

wr+=1

else:

ac+=1

return [c,ac,wr]

word\_dict={"THEIR": "THEIR","BUSINESS":"BISINESS","WINDOWS":"WINDMILL","WERE":"WEAR","SAMPLE":"SAMPLE"}

print(find\_correct(word\_dict))

#PF-Tryout

#Start writing your code here

import random

TotalHeads = 0

TotalTails = 0

i = 0

Probability = 70

NumberOfTrials = 1000

def biasedflip():

global TotalTails

global TotalHeads

if random.randint(1,100) < Probability:

TotalHeads += 1

else:

TotalTails += 1

while i < NumberOfTrials:

biasedflip()

i += 1

print(TotalHeads)

print(TotalTails)

#PF-Assgn-50

def sms\_encoding(data):

vowels = ['a','e','i','o','u','A','E','I','O','U']

ls = data.split(' ')

st = ""

for s in ls:

st+=" "

c=0

for i in s:

if i not in vowels:

c=-1

break

if c==0:

st+=s

else:

for i in s:

if i not in vowels:

st+=i

return st[1:]

data="I love Python"

print(sms\_encoding(data))

#PF-Exer-33

#PF-Tryout

import random

def guess\_number(number\_in\_mind):

n=random.randrange(1,11)

if n>number\_in\_mind:

print ('Number is low')

elif n<number\_in\_mind:

print ('Number is high')

else:

print ('You have got it right!!!')

#Provide different values for number\_in\_mind and test your program

guess\_number(4)

#PF-Exer-34

import math

def find\_number\_of\_combination(number\_of\_flavours):

total\_combination=2\*\*number\_of\_flavours

return total\_combination

#Provide different values for number\_of\_flavours and test your program

number\_of\_combination=find\_number\_of\_combination(6)

print(number\_of\_combination)

#PF-Exer-35

def count\_names(name\_list):

count1=0

count2=0

for i in name\_list:

if len(i)==3 and i.endswith("at"):

count1+=1

for j in range(len(i)-1):

if i[j]=="a" and i[j+1]=="t":

count2+=1

#start writing your code here

#Populate the variables: count1 and count2

# Use the below given print statements to display the output

# Also, do not modify them for verification to work

print("\_at -> ",count1)

print("%at% -> ",count2)

#Provide different names in the list and test your program

name\_list=["Hat","Cat","rabbit","matter"]

count\_names(name\_list)

#PF-Tryout

import re

flight\_details="Good Evening, Welcome to British Airways. Your flight number is ba8004. Flight departure time is 16:45"

#This function returns the values in the search result

def printout(search\_result):

if(search\_result!=None):

return search\_result.group()

else:

return "No Output"

search\_result = re.search(r"British Airways",flight\_details)

#This will invoke the printout() and displays the search result values

print(printout(search\_result))

search\_result = re.search(r"16:45$",flight\_details)

#Write your regular expression here for question 2

print(printout(search\_result))

search\_result = re.search(r"^Good",flight\_details)

print(printout(search\_result))

search\_result = re.search(r"F.....",flight\_details)

#Write your regular expression here for question 4

print(printout(search\_result))

if re.search(r"ba(\d{4})",flight\_details):

print(re.sub(r"ba",r"BA",flight\_details))

#PF-Exer-38

#This verification is based on string match.

num1=36

num2=7

num3=18

calc = lambda x,y : x%y + x-y

#Write the lambda expression for question1

print(calc(num1,num2))

square\_root = lambda x:x\*\*0.5

print(square\_root(num3))

square\_root2= lambda x:x\*\*(0.5)

#Write the lambda expression for question3

print(square\_root2(num3))

#PF-Exer-39

#This verification is based on string match.

principal\_amount=4000

duration=12

rate\_of\_interest=13

simple\_interest = lambda x,y,z : (x\*y\*z)/100

if(simple\_interest(principal\_amount,duration,rate\_of\_interest)>1000):

print("Platinum Member")

else:

print("Gold Member")

#write your logic here

#PF-Exer-40

#This verification is based on string match.

num1=20

num2=30

div = lambda x,y : x+y

if(div(num1,num2)%10)==0:

print("Divisible by 10")

else:

print("Not Divisible by 10")#write your logic here

#PF-Exer-41

#This verification is based on string match.

def sum\_all(function, data):

sum = 0

for i in data:

if function(i):

sum+=i

return sum

#list\_of\_nos=[1,3,4,5,6,7,8,9,10,15,20,30,110]

#list\_of\_nos = [25,26,27,28,29,30,147,187]

list\_of\_nos = [100,200,300,500,1040]

greater = lambda x: x>10

divide = lambda x: x<=100 and x%10==0

range\_of\_values = lambda x:x>=25 and x<=50

#Use the below given print statements to display the output

# Also, do not modify them for verification to work

print(sum\_all(greater,list\_of\_nos))

print(sum\_all(divide,list\_of\_nos))

print(sum\_all(range\_of\_values,list\_of\_nos))

#PF-Assgn-52

sample\_data = range(1,2)

list\_of\_oddnum=[]

list\_of\_evennum=[]

def even(sample\_data):

for i in sample\_data:

if(i%2==0):

list\_of\_evennum.append(i)

return list\_of\_evennum

def odd(sample\_data):

for i in sample\_data:

if(i%2!=0):

list\_of\_oddnum.append(i)

return list\_of\_oddnum

def sum\_of\_numbers(list\_of\_num,filter\_func=None):

if filter\_func:

return sum(filter\_func(list\_of\_num))

return sum(list\_of\_num)

print(sum\_of\_numbers(sample\_data,None))

#PF-Assgn-53

#This verification is based on string match.

import re

poem='''

It takes strength for being certain,

It takes courage to have doubt.

It takes strength for challenging alone,

It takes courage to lean on another.

It takes strength for loving other souls,

It takes courage to be loved.

It takes strength for hiding our own pain,

It takes courage to help if it is paining for someone.

'''

#Note: Triple quotes can be used to enclose Strings which has lines of text.

#Write your logic here for question 1

c=0

x = re.findall("v", poem)

for i in x:

c=c+1

print(c)

print()

x1 = poem.replace('\n',' ')

l = len(x1)

print(x1[1:l-1])

print()

x = re.sub("co", "Co", poem)

x = re.sub("ch", "Ch", x)

print(x)

print(re.sub(r"([ah]i).{3}", r"\1\*/\*", poem))

#PF-Assgn-54

def check\_anagram(data1,data2):

#start writing your code he

n1 = len(data1)

n2 = len(data2)

count=0

# If lenght of both strings is not same, then

# they cannot be anagram

if n1 != n2:

return False

for i in range(len(data1)):

if(data1[i]==data2[i]):

count+=1

if(count==1):

return False

# Sort both strings

data1 = sorted(data1.lower())

data2 = sorted(data2.lower())

# Compare sorted strings

for i in range(0, n1):

if data1[i] != data2[i]:

return False

return True

print(check\_anagram("eat","tea"))

#PF-Assgn-55

#Sample ticket list - ticket format: "flight\_no:source:destination:ticket\_no"

#Note: flight\_no has the following format - "airline\_name followed by three digit number

#Global variable

ticket\_list=["AI567:MUM:LON:014","AI077:MUM:LON:056", "BA896:MUM:LON:067", "SI267:MUM:SIN:145","AI077:MUM:CAN:060","SI267:BLR:MUM:148","AI567:CHE:SIN:015","AI077:MUM:SIN:050","AI077:MUM:LON:051","SI267:MUM:SIN:146"]

def find\_passengers\_flight(airline\_name="AI"):

#This function finds and returns the number of passengers tra

count=0

for i in ticket\_list:

string\_list=i.split(":")

if(string\_list[0].startswith(airline\_name)):

count+=1

return count

def find\_passengers\_destination(destination):

#Write the logic to find and return the number of passengers traveling to the specified destination

count=0

for i in ticket\_list:

if i[10:13]==destination:

count+=1

return count

def find\_passengers\_per\_flight():

'''Write the logic to find and return a list having number of passengers traveling per flight based on the details in the ticket\_list

In the list, details should be provided in the format:

[flight\_no:no\_of\_passengers, flight\_no:no\_of\_passengers, etc.].'''

d = {}

for i in ticket\_list:

k = i[:5]

if k in d:

d[k]+=1

else:

d[k]=1

ls = []

for i in d:

st = i+":"+str(d[i])

ls.append(st)

return ls

#Remove pass and write your logic here

def sort\_passenger\_list():

ls = find\_passengers\_per\_flight()

return sorted(ls,key=lambda x:int(x[-1]),reverse=True)

#Write the logic to sort the list returned from find\_passengers\_per\_flight() function in the descending order of number of passengers

#Remove pass and write your logic here

#Provide different values for airline\_name and destination and test your program.

#print(find\_passengers\_flight("AI"))

#print(find\_passengers\_destination("LON"))

#print(sort\_passenger\_list())

#find\_passengers\_per\_flight()

print(find\_passengers\_flight("AI"))

print(find\_passengers\_destination("LON"))

print(sort\_passenger\_list())

#PF-Assgn-56

def max\_frequency\_word\_counter(data):

word=""

frequency=0

ls = data.split(" ")

d = {}

for x in ls:

x = x.lower()

if x in d:

d[x]+=1

else:

d[x]=1

if d[x]>frequency:

frequency = d[x]

word = x

#start writing your code here

#Populate the variables: word and frequency

# Use the below given print statements to display the output

# Also, do not modify them for verification to work

print(word,frequency)

#Provide different values for data and test your program.

data="Work like you do not need money, love like you have never been hurt, and dance like no one is watching"

max\_frequency\_word\_counter(data)

#PF-Assgn-57

def check\_prime(number):

num=number

if num==1:

return False

if num >1:

for i in range(2,num):

if (num % i) == 0:

return False

return True

def rotations(num):

ls = []

while num>0:

ls.append(num%10)

num//=10

ls1 = []

l = len(ls)

ls.reverse()

for i in range(l):

m = 0

for j in range(l):

m = 10\*m

m += ls[(i+j)%l]

ls1.append(m)

return ls1

def get\_circular\_prime\_count(limit):

ls = []

for i in range(1,limit):

rot = rotations(i)

c = 0

for j in rot:

if check\_prime(j):

continue

else:

c=-1

break

if c==0:

ls.append(i)

return len(ls)

#Provide different values for limit and test your program

#print(rotations(197))

print(get\_circular\_prime\_count(100))

#print(check\_prime(197))

#PF-Assgn-58

def validate\_credit\_card\_number(card\_number):

sum = 0

i = 0

while(card\_number>0):

n = card\_number%10

if i%2==0:

sum+=n

else:

n = 2\*n

while(n>0):

sum+=n%10

n//=10

card\_number//=10

i+=1

if sum%10==0:

return True

return False

card\_number= 1456734512345698 #4539869650133101 #1456734512345698 # #5239512608615007

result=validate\_credit\_card\_number(card\_number)

if(result):

print("credit card number is valid")

else:

print("credit card number is invalid")

#PF-Assgn-59

def check\_perfect\_number(number):

s = 0

for i in range(1,number//2+1):

if number%i==0:

s+=i

if s==number:

return True

return False

def check\_perfectno\_from\_list(no\_list):

ls = []

for i in no\_list:

if check\_perfect\_number(i) and i!=0:

ls.append(i)

return ls

perfectno\_list=check\_perfectno\_from\_list([18,6,4,2,1,28])

print(perfectno\_list)

#PF-Assgn-60

def remove\_duplicates(value):

st = ""

for i in value:

if i in st:

continue

else:

st+=i

return st

print(remove\_duplicates("11223445566666ababzzz@@@123#\*#\*"))

#PF-Assgn-61

def validate\_name(name):

if len(name)==0 or len(name)>15 or not name.isalpha():

return False

return True

def validate\_phone\_no(phno):

pho = str(phno)

if len(pho)!=10 or not pho.isdigit():

return False

c=0

for i in range(len(pho)-1):

if pho[i]!=pho[i+1]:

c=-1

break

if c==0:

return False

return True

def validate\_email\_id(email\_id):

l = len(email\_id)

c = 0

for i in range(l):

if email\_id[i]=="@":

c=-1

break

if c==0 or not email\_id.endswith(".com"):

return False

j = i

if i>4:

for i in range(j-3):

if email\_id[i:i+4]==".com":

return False

st = email\_id[j+1:]

for i in range(len(st)):

if st[i]==".":

break

st = st[:i]

mail = ["gmail","yahoo","hotmail"]

if st not in mail:

return False

return True

def validate\_all(name,phone\_no,email\_id):

#Start writing your code here

# Use the below given print statements to display appropriate messages

if not validate\_name(name):

print("Invalid Name")

elif not validate\_phone\_no(phone\_no):

print("Invalid phone number")

elif not validate\_email\_id(email\_id):

print("Invalid email id")

else:

print("All the details are valid")

#Provide different values for name, phone\_no and email\_id and test your program

validate\_all("Tina", "9999999999", "tina@yahoo.com")