**Ex No: 7**

**Date: 22.11.2020**

**AIM:**

Fill the missing words

**PROGRAM:**

|  |
| --- |
| primes = [2, 3, 5, 7, 11]  print(primes)  **# Output: [2, 3, 5, 7, 11]** |
|  |
|  |
| items = ['cake', 'cookie', 'bread'] total\_items = items + ['biscuit', 'tart']  print(total\_items)  # **Output:['cake', 'cookie', 'bread', 'biscuit', 'tart']** |
| orders = ['daisies', 'periwinkle']  orders.append('tulips') print(orders) **# Output: ['daisies', 'periwinkle', 'tulips']**  owners\_names = ['Jenny', 'Sam', 'Alexis'] dogs\_names = ['Elphonse', 'Dr. Doggy DDS', 'Carter'] owners\_dogs = zip(owners\_names, dogs\_names) print(list(owners\_dogs)) **# Output: [('Jenny', 'Elphonse'), ('Sam', 'Dr.Doggy DDS'), ('Alexis', 'Carter')]**  items = [1, 2, 3, 4, 5, 6] print(items[:4]) **#Output: [1, 2, 3, 4]** print(items[2:]) **#Output: [3, 4, 5, 6]**  knapsack = [2, 4, 3, 7, 10] size = len(knapsack) print(size) **# Output: 5**  cnt = knapsack.count(7)  print(cnt) **# Output: 1**  exampleList = [4, 2, 1, 3] exampleList.sort() print(exampleList) **# Output: [1, 2, 3, 4]**  soups = ['minestrone', 'lentil', 'pho', 'laksa'] soups[-1] **# output: laksa** soups[-3:] **# output: [ 'lentil', 'pho', 'laksa' ]** soups[:-2] **# output: [ 'minestrone', 'lentil' ]** |

**RESULT:**

The above program has been successfully verified.

**Ex No: 8**

**Date: 22.11.2020**

**AIM:**

To Write a Python program to get a list of numbers of list and create a list of tuples having first element as the number and second element as the cube of the number. Next the list of tuples should be sorted in increasing order by the last element in each tuple from a given list of non-empty tuples.

**PROGRAM:**

l=[]

n=input()

n=int(n)

for i in range(n):

num=int(input())

l.append(num)

a=[(j,pow(j,3)) for j in l]

print(a)

def first(n):

return n[0]

a.sort(key=first)

print(a)

**LINK:**

http://103.53.53.18/mod/vpl/forms/submissionview.php?id=317&userid=1781

**OUTPUT:**

3

1

3

2

[(1, 1), (3, 27), (2, 8)]

[(1, 1), (2, 8), (3, 27)]

**RESULT:**

The list of tuples are sorted in increasing order by the last element in each tuple from a given list of non-empty tuples.

**Ex No: 9**

**Date: 22.11.2020**

**AIM:**

Fill the missing words

**PROGRAM:**

print('\n—dictionaries') **#Output: -- dictionaries**

d = {'a': 1, 'b': 2}

print(d['a']) **#Output: 1**

del d['a']

# iterate  
d = {'a': 1, 'b': 2}  
for key, value in d.items():  
 print(key, ':', value)

for key in d:  
 print(key, d[key])

# d.fromkeys(iterable[,value=None]) -> dict: with keys from iterable and all same value  
d = d.fromkeys(['a', 'b'], 1)  
print(d) **#Output: {'a'**

# d.clear() -> removes all items from d  
d = {'a': 1, 'b': 2}  
d.clear()  
print(d) **#Output: {}**

# d.items() -> list: copy of d's list of (key, item) pairs  
d = {'a': 1, 'b': 2}  
print(d.items()) **#Output: [('a', 1), ('b', 2)]**

# d.keys() -> list: copy of d's list of keys  
d = {'a': 1, 'b': 2}  
print(d.keys()) **#Output: ['a', 'b']**

# d.values() -> list: copy of d's list of values  
d = {'a': 1, 'b': 2}  
print(d.values()) **#Output: [1, 2]**

# d.get(key,defval) -> value: d[key] if key in d, else defval  
d = {'a': 1, 'b': 2}  
print(d.get("c", 3)) **#Output: 3**

print(d) **#Output: {'a'**

# d.setdefault(key[,defval=None]) -> value: if key not in d set d[key]=defval, return d[key]  
d = {'a': 1, 'b': 2}  
print('d.setdefault("c", []) returns ' + str(d.setdefault("c", 3)) + ' d is now ' + str(d))

#**Output: d.setdefault("c", []) returns 3 d is now {'a': 1, 'b': 2, 'c': 3}**

#d.pop(key[,defval]) -> value: del key and returns the corresponding value. If key is not found, defval is returned if given, otherwise KeyError is raised  
d = {'a': 1, 'b': 2}  
print('d.pop("b", 3) returns ' + str(d.pop("b", 3)) + ' d is now ' + str(d))

**#Output: d.pop("b", 3) returns 2 d is now {'a': 1}**

print('d.pop("c", 3) returns ' + str(d.pop("c", 3)) + ' d is still ' + str(d))

**#Output: d.pop("c", 3) returns 3 d is still {'a': 1}**

# sort on values  
import operator  
x = {1: 4, 5: 4, 4: 4}  
sorted\_x = sorted(x.items(), key=operator.itemgetter(1), reverse=True)

**#Output: print('sorted(x.items(), key=operator.itemgetter(1)) sorts on values ' + str(sorted\_x))**

# max of values  
d = {'a':1000, 'b':3000, 'c': 100}  
print('key of max value is ' + max(d.keys(), key=(lambda key: d[key])))

**#Output: key of max value is b**

**RESULT:**

The above program has been successfully verified.

**Ex No: 10**

**Date: 22.11.2020**

**AIM:**

To write a Python function that accepts a string and calculate the number of upper case letters and lower case letters .

**PROGRAM:**

def string\_test(s):

d={"UPPER\_CASE":0, "LOWER\_CASE":0}

for let in s:

if let.isupper():

d["UPPER\_CASE"]=d["UPPER\_CASE"]+1

elif let.islower():

d["LOWER\_CASE"]=d["LOWER\_CASE"]+1

else:

continue

print("No. of Upper case characters: ",d["UPPER\_CASE"])

print("No. of Lower case characters: ",d["LOWER\_CASE"])

s=input()

string\_test(s)

**LINK:**

http://103.53.53.18/mod/vpl/forms/submissionview.php?id=325&userid=1781

**OUTPUT:**

Sona College of Technology

No. of Upper case characters: 3

No. of Lower case Characters: 20

**RESULT:**

The number of upper case letters and lower case letters for a input string is obtained successfully.

**Ex No: 11**

**Date: 22.11.2020**

**AIM:**

To write a Python program to find the greatest common divisor (gcd) of two integers using recursion.

**PROGRAM:**

def gcd(a,b):

if(b==0):

return a

else:

return gcd(b,a%b)

a=int(input("Enter first number:"))

b=int(input("Enter second number:"))

GCD=gcd(a,b)

print("GCD is: ")

print(GCD)

**LINK:**

http://103.53.53.18/mod/vpl/forms/submissionview.php?id=326&userid=1781

**OUTPUT:**

Enter first number:10

Enter second number:12

GCD is:

2

**RESULT:**

The greatest common divisor (gcd) of two integers is obtained successfully using recursion.

**Ex No: 13**

**Date: 22.11.2020**

**AIM:**

To write a python program to implement the class diagram .

**PROGRAM:**

class Apparel:

counter=100

def \_\_init\_\_(self,price,item\_type):

Apparel.counter+=1

self.\_\_item\_id=item\_type[0]+str(Apparel.counter)

self.\_\_price=price

self.\_\_item\_type=item\_type

def calculate\_price(self):

self.\_\_price+=self.\_\_price\*0.05

def get\_item\_id(self):

return self.\_\_item\_id

def get\_price(self):

return self.\_\_price

def get\_item\_type(self):

return self.\_\_item\_type

def set\_price(self,price):

self.\_\_price=price

return self.\_\_price

class Cotton(Apparel):

def \_\_init\_\_(self,price,discount):

super().\_\_init\_\_(price,'Cotton')

self.\_\_discount=discount

def calculate\_price(self):

super().calculate\_price()

price=self.get\_price()

price-=price\*(self.\_\_discount/100)

price+=price\*0.05

self.set\_price(price)

return price

def get\_discount(self):

return self.\_\_discount

class Silk(Apparel):

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

super().\_\_init\_\_(price,'Silk')

self.\_\_points=None

def calculate\_price(self):

super().calculate\_price()

if(self.get\_price()>10000):

self.\_\_points=10

else:

self.\_\_points=3

return self.set\_price(self.get\_price()+(self.get\_price()\*0.1))

def get\_points(self):

return self.\_\_points

silk=int(input())

cotton=int(input())

discount=int(input())

a=Silk(silk)

print(a.calculate\_price())

b=Cotton(cotton,discount)

print(b.calculate\_price())

**LINK:**

http://103.53.53.18/mod/vpl/forms/submissionview.php?id=328&userid=1781

**OUTPUT:**

1320

21000

10

1524.6000000000001

20837.25

**RESULT:**

The output for the given class diagram is obtained successfully.

**Ex No: 14**

**Date: 22.11.2020**

**AIM:**

To Write a Python class to find validity of a string of parentheses, '(', ')', '{', '}', '[' and '].

**PROGRAM:**

def areBracketsBalanced(expr):

stack=[]

for brac in expr:

if brac in ['[','{','(']:

stack.append(brac)

else:

if not stack:

return False

cur\_brac=stack.pop()

if cur\_brac=='[':

if brac!=']':

return False

if cur\_brac=='{':

if brac!='}':

return False

if cur\_brac=='(':

if brac!=')':

return False

if stack:

return False

return True

if \_\_name\_\_ == "\_\_main\_\_":

expr=input()

if areBracketsBalanced(expr):

print("valid")

else:

print("invalid")

**LINK:**

http://103.53.53.18/mod/vpl/forms/submissionview.php?id=327&userid=1781

**OUTPUT:**

input: ()[]{()}

output: valid

**RESULT:**

The validity of a string of parenthesis is successfully verified.