**PYTHON LABORATORY**

**Exercise No:** 7

**Date:** 20.11.2020

**Program:**

Fill in the missing words

primes = [2, 3, 5, 7, 11]  
print(primes) 

items = ['cake', 'cookie', 'bread']  
total\_items = items + ['biscuit', 'tart']  
print(total\_items) 

orders = ['daisies', 'periwinkle']  
orders.append('tulips')  
print(orders) 

owners\_names = ['Jenny', 'Sam', 'Alexis']  
dogs\_names = ['Elphonse', 'Dr. Doggy DDS', 'Carter']  
owners\_dogs = zip(owners\_names, dogs\_names)  
print(list(owners\_dogs)) 

items = [1, 2, 3, 4, 5, 6]  
print(items[:4])   
print(items[2:]) 

knapsack = [2, 4, 3, 7, 10]  
size = len(knapsack)   
print(size)   
cnt = knapsack.count(7)  
print(cnt) 

exampleList = [4, 2, 1, 3]  
exampleList.sort()  
print(exampleList) 

soups = ['minestrone', 'lentil', 'pho', 'laksa']  
print(soups[-1])     
print(soups[-3:])    
print(soups[:-2])  

**Exercise No: 8**

**Date :**20.11.2020

**Aim**:

To write a Python program to get a list, sorted in increasing order by the last element in each tuple from a given list of non-empty tuples.

**Program**:

def last(n):

return n[-1]

def sort\_list\_last(tuples):

return sorted(tuples, key=last)

print(sort\_list\_last([(2, 5),(1, 2),(4, 4),(2, 3),(2,1)]))

**Link:**

[**http://103.53.53.18/mod/vpl/forms/submissionview.php?id=317&userid=1780**](http://103.53.53.18/mod/vpl/forms/submissionview.php?id=317&userid=1780)

**Output:**

[(2, 5), (1, 2), (4, 4), (2, 3), (2, 1)]

 [(2, 1), (1, 2), (2, 3), (4, 4), (2, 5)]

**Result:**

Using sort\_list\_last() function we sorted the list in increasing order by the last element in each tuple from a given non-empty tuples.

**Exercise No:** 9

**Date :**20.11.2020

Fill in the missing words

print('\n-- dictionaries') 

d = {'a': 1, 'b': 2}  
print(d['a'])   
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)  

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

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

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

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

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

# 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))   


#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))  


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


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


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


**Exercise No:** 10

**Date :**20.11.2020

**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 c in s:

if c.isupper():

d["UPPER\_CASE"]+=1

elif c.islower():

d["LOWER\_CASE"]+=1

else:

pass

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=1780>

**Output:**

Sona College of Technology

No. of Upper case characters: 3

No. of Lower case Characters: 20

**Result:**

Using string\_test() function we calculated the no of upper and lower case characters from the given string.

**Exercise No**: 11

**Date :**20.11.2020

**Aim:**

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

Program:

def gcd(x,y):

gcd=1

try:

if(x%y)==0:

return y

for k in range(int(y/2),0,-1):

if x%k==0 and y%k==0:

gcd=k

break

return gcd

except:

return 0

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=1780>

**Output:**

Enter First Number:10

Enter Second Number:20

GCD is :10

**Result:**

Using gcd() recursive function we got the greatest common divisor among two numbers.

**Exercise No:** 13

**Date :**21.11.2020

**Aim:**

To write a python program to implement the class diagram for an apparel shop to manage the items which it sells.

**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=1780>

**Output:**

1320

21000

10

1524.6

20837.25

**Result:**

using class and functions we implemented the class diagram for an apparel shop to manage the items which it sells.

**Exercise No:** 14

**Date :**21.11.2020

**Aim:**

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

Program:

def areBracketsBalanced(expr):

stack=[]

open\_expressions=['{','[','(']

for i in range(len(expr)):

if(expr[i] in open\_expressions):

stack.append(expr[i])

else:

if not stack:

return False

curr=stack.pop()

if(curr=='{'and expr[i]!='}'):

return False

if(curr=='['and expr[i]!=']'):

return False

if(curr=='('and expr[i]!=')'):

return False

if stack:

return False

else:

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=1780>

**Output:**

(({}))

valid

**Result:**

Using areBracketsBalanced() functions we verified the string of parentheses are valid are not.