

```
In [3]: ▶ #Lab:7
        #Pgm:1

        #i)
        fruits={'apples':20,'bananas':50,'oranges':100}
```

```
In [4]: ▶ #ii)
        print(fruits)

        {'apples': 20, 'bananas': 50, 'oranges': 100}
```

```
In [5]: ▶ #iii)
        print("There are",fruits.get('bananas'),'bananas')

        There are 50 bananas
```

```
In [6]: ▶ #iv)
        print("No. of keys:",len(fruits))

        No. of keys: 3
```

```
In [7]: ▶ #v)
        if 'graphs' in fruits:
            print("Graphs is Available")
        else :
            print("Graphs is NOT Available")

        Graphs is NOT Available
```

```
In [8]: ▶ #vi)
        if 'pears' in fruits:
            print("Pears is Available")
        else :
            fruits['pears']=10
            print(fruits)

        {'apples': 20, 'bananas': 50, 'oranges': 100, 'pears': 10}
```

```
In [9]: ▶ #vii)
        print("Asending Order :")
        for i in sorted(fruits):
            print(i)

        Asending Order :
        apples
        bananas
        oranges
        pears
```

```
In [10]: ▶ #viii)
print("Desending Order :")
for i in reversed(fruits):
    print(i)
```

```
Desending Order :
pears
oranges
bananas
apples
```

```
In [20]: ▶ #ix)
fruits={'apples': 20, 'bananas': 50, 'oranges': 100, 'pears': 10}
del fruits["pears"]
print(fruits)
```

```
{'apples': 20, 'bananas': 50, 'oranges': 100}
```

```
In [21]: ▶ #x)
def show():
    print(f'{fruits}')

#main:
show()
```

```
{'apples': 20, 'bananas': 50, 'oranges': 100}
```

```
In [27]: ▶ #xi)
def add_fruits(fruits,name,quantity):
    fruits[name]=fruits.get(name)+quantity

#main:
add_fruits(fruits,'apples',40)
show()
```

```
{'apples': 100, 'bananas': 150, 'oranges': 100}
```

```
In [23]: ▶ #xii)
#main:
add_fruits(fruits,'bananas',100)
print(fruits)
```

```
{'apples': 60, 'bananas': 150, 'oranges': 100}
```

```
In [24]: ▶ #xiii)
show()
```

```
{'apples': 60, 'bananas': 150, 'oranges': 100}
```

```
In [25]: ▶ #xiv)
import pickle
fruits={'apples':60,'bananas':150,'oranges':100}
file=open("mypicklefile","wb")
pickle.dump(fruits,file)
file.close()
```

```
In [26]: ▶ import pickle
frut_prc=open("mypicklefile","rb")
fruits=pickle.load(frut_prc)
print(fruits)

{'apples': 60, 'bananas': 150, 'oranges': 100}
```

```
In [ ]: ▶
```

```
In [ ]: ▶
```

```
In [31]: ▶ #Lab:7
#Pgm:2

customers={}
while True:
    a=input("Name: ")
    b=int(input("Phone No.: "))
    c=input("Emailid: ")
    d=input("Continue or '(Type Done)' Over: ")
    if d=='done':
        break
    key=a
    contacts=[b,c]
    customers[key]=contacts
    print('\n',customers)
```

Name: arul
Phone No.: 8870245563
Emailid: arulkumarark1924@gmail.com
Continue or '(Type Done)' Over: no

```
{'arul': [8870245563, 'arulkumarark1924@gmail.com']}
```

Name: asha
Phone No.: 9595675455
Emailid: asharajkumar@gmai.com
Continue or '(Type Done)' Over: no

```
{'arul': [8870245563, 'arulkumarark1924@gmail.com'], 'asha': [9595675455, 'asharajkumar@gmai.com']}
```

Name: raj
Phone No.: 9442555105
Emailid: rajasha@hotmail.com
Continue or '(Type Done)' Over: no

```
{'arul': [8870245563, 'arulkumarark1924@gmail.com'], 'asha': [9595675455, 'asharajkumar@gmai.com'], 'raj': [9442555105, 'rajasha@hotmail.com']}
```

Name: swathi
Phone No.: 9488220700
Emailid: swathi2220@gmail.com
Continue or '(Type Done)' Over: done

```
In [38]: ▶ if "rex" in customers:
print(customers.get("rex"))
else:
    print("Not exists")
```

Not exists

```
In [39]: ▶ customers.update({"rex":[9942002764,"rajkumar@bhc.edu"]})
```

In [40]: `print(customers)`

```
{'arul': [8870245563, 'arulkumarark1924@gmail.com'], 'asha': [9595675455, 'asharajkumar@gmai.com'], 'raj': [9442555105, 'rajasha@hotmail.com'], 'rex': [9942002764, 'rajkumar@bhc.edu']}
```

In [41]: `for i in customers:
 print("Name :",i,"\t","Contacts :",customers[i])`

```
Name : arul      Contacts : [8870245563, 'arulkumarark1924@gmail.com']  
Name : asha     Contacts : [9595675455, 'asharajkumar@gmai.com']  
Name : raj      Contacts : [9442555105, 'rajasha@hotmail.com']  
Name : rex      Contacts : [9942002764, 'rajkumar@bhc.edu']
```

In [47]: `print("Asending Order :")
for j in sorted(customers):
 print(j)
print()
print("Count of Customers :",len(customers))`

```
Asending Order :  
arul  
asha  
raj  
rex
```

```
Count of Customers : 4
```

In [37]: `del customers["rex"]
print(customers)`

```
{'arul': [8870245563, 'arulkumarark1924@gmail.com'], 'asha': [9595675455, 'asharajkumar@gmai.com'], 'raj': [9442555105, 'rajasha@hotmail.com']}
```

In []:


```

In [56]: #Lab:7
#Pgm:3

f=open("D:\PSPR\Python Coding\dict_word.txt")
r=f.read()
words = [word.lower() for word in r.split()]
words.sort(reverse=True)
alphabet=str(words).split()

print("Desensing Order :")
for word in words:
    print(word,end=" ")
print()
print()
print()
for word in words:
    for j in word:
        print(j,end=" ")
alphabets = 0
print()
print()
for i in range(len(r)):
    if(r[i].isalpha()):
        alphabets = alphabets + 1

print()
print()
print("Total Number of Alphabets :",alphabets)

```

Desensing Order :

written without with with will which when when well way very very v
 ariables, use typing trace. together. to to to to to to time, through th
 is therefore there the the the the the the the the the the the t
 he the testifying syntax supports structures, stepping step, statements
 standard stack source: source source so since simple, simple setting sem
 antics. segmentation scripting reuse. reduces readability rapid raises q
 uickest python's python's python python python python python python prov
 ides. programs programming programmers program program program program p
 roductivity prints print power. platforms, packages, other or or or on.
 on often, often of of of of object-oriented, no never modules modularity
 makes make major maintenance. love local line library level learn langua
 ge language itself, its it it it is is is is is is introspective interpr
 eter interpreter interpreter interpreted, instead, inspection input incr
 edibly increased in in in in high-level high-level hand, glue global fre
 ely form for for for few fault. fast. fast fall extensive expressions, e
 xisting exception. exception, evaluation error, encourages emphasizes ef
 fective. edit-test-debug edit-test-debug easy: easy dynamic dynamic dyna
 mic doesn't distributed. discovers development, debugging debugger debug
 ger debug data cycle cycle cost connect components compilation combined
 code code charge cause catch can built bug breakpoints, binding, binary
 because be bad available attractive at as as as as are arbitrary approach a
 pplication and and and and and and and and and an an an allows all add a a a
 a a a a a a

w r i t t e n w i t h o u t w i t h w i t h w i t h w i l l w h i c h w

```

henwhenwellwayveryveryvariables,uset
ypingtrace.together.tototototototime
,throughthisthereforetheretheethet
hethethethethethethethethethethet
estifyingssyntaxsupportstructures,st
eppingsstep,statementsstandardstackso
urce:sourcesourcesosincesimple,simpl
esettingssemantics.segmentationscript
ingreuse.reducesreadabilityrapidrais
esquickestpython'spython'spythonpyth
onpythonpythonpythonpythonprovides.p
rogramsprogrammingprogrammersprogram
programprogramprogramproductivitypri
ntsprintpower.platforms,packages,oth
erororororon.onoften,oftenofofofofobje
ct-oriented,nonevermodulesmodularity
makesmakemajormaintenance.lovelocall
inelibrarylevelleearnlanguagelanguage
itself,itsititititisisisisisisintrospe
ctiveinterpreteinterinterpreteinterpret
erinterpreted,instead,inspectioninpu
tincrediblyincreasedininininhigh-lev
elhigh-levelhand,glueglobalfreelyfor
mforforforfewfault.fast.fastfallexte
nsiveexpressions,existingexception.e
xception,evaluationerror,encouragese
mphasizeseffective.edit-test-debuged
it-test-debugeasy:easydynamicdynamic
dynamicdoesn'tdistributed.discoversd
evelopment,debuggingdebuggerdebugger
debugdatacyclecyclecostconnectcompon
entscompilationcombinedcodecodecharg
ecausecatchcanbuiltbugbreakpoints,bi
nding,binarybecausebebadavailableatt
ractiveatasasasarearbitraryapproach
applicationandandandandandandandand
ananaallowsalladdaaaaaa

```

Total Number of Alphabets : 1313

In []: ▶