Set:

Sets are used to store multiple items in a single variable. A set is a collection which is *unordered*, *unchangeable**, and *unindexed*. Set *items* are unchangeable, but we can remove items and add new items.

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

1. Sets are unordered, so you cannot be sure in which order the items will appear.
2. We cannot change the item after set has been created.
3. It does not allow duplicate values.

Dictionary:

Dictionaries are used to store data values in key. A dictionary is a collection which is ordered*, changeable and does not allow duplicates. Dictionaries are written with curly brackets, and have keys and values:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

- In the latest versions of python compilers, dictionaries are ordered.
- We can change, add or remove items after the dictionary has been created.
- Dictionaries cannot have two items with the same key.

Exception Handling:

If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a *try: block*. After the try: block, include an *except: statement*, followed by a block of code which handles the problem as elegantly as possible.

Syntax

Here is simple syntax of *try....except...else* blocks −

```
try:
   You do your operations here;
   ......................
except ExceptionI:
   If there is ExceptionI, then execute this block.
except ExceptionII:
   If there is ExceptionII, then execute this block.
   ......................
```

```
else:
   If there is no exception then execute this block.
```

A list of various exceptions are as follows:

Exception Name	Exception Description
ArithmeticError	Base class for all errors that occur for numeric calculation.
OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
FloatingPointError	Raised when a floating point calculation fails.
ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.
AssertionError	Raised in case of failure of the Assert statement.
AttributeError	Raised in case of failure of attribute reference or assignment.
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
ImportError	Raised when an import statement fails.
KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.
IndexError	Raised when an index is not found in a sequence.
KeyError	Raised when the specified key is not found in the dictionary.
NameError	Raised when an identifier is not found in the local or global namespace.
UnboundLocalError	Raised when trying to access a local variable in a function or method but no value has been assigned to it.

EnvironmentError	Base class for all exceptions that occur outside the Python environment.
IOError	Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
SyntaxError	Raised when there is an error in Python syntax.
IndentationError	Raised when indentation is not specified properly.
SystemError	Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
SystemExit	Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.
ValueError	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
RuntimeError	Raised when a generated error does not fall into any category.
NotImplementedError	Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

Problems:

1. What will be the output of the following code:

```
dict1={"a":10,"b":2,"c":3}
str1=""
for i in dict1:
            str1=str1+str(dict1[i])+" "
            str2=str1[:-1]
print(str2[::-1])
```

Ans: 3 2 01

2. What will be the output of the following code:

```
total={}
```

```
def insert(items):
    if items in total:
        total[items] += 1
    else:
        total[items] = 1
insert('Apple')
insert('Ball')
insert('Apple')
print (len(total))
```

Ans: 2

3. What will be the output of the following code:

```
def foo():
    try:
        return 1
    finally:
        return 2
k = foo()
print(k)
```

Ans: 2

4. What will be the output of the following code:

```
names = {'Janice': 5, 'Emily': 3, 'John': 7, 'Eleanor': 2}
list_o_names = []
for name in names:
    if names[name] > 5:
        list_o_names.append(name)
print(list_o_names)
```

Ans: ['John']

5. Guess the output of the following code:

```
word = 'brontosaurus'
diction = {}
for letter in word:
    if letter not in diction.keys():
        diction[letter] = 0
    diction[letter] += 1
print(diction.get('o', 0) + 4)
```

Ans: 6

6. A software company is going to develop software for analyzing text of a book. They need a function to calculate the occurrence of each word in the book. Write a function for doing this. The final result should be stored in a dictionary. Print the occurrence of a particular word given by user. If word is not there, print appropriate message.

```
word=input("Enter word to search for: ")
```

```
fptr=open("D:\python_programs\python_course_bennett\lab_preparation\\test.tx
t","r")
contents=fptr.read()

wordCount={}
for i in contents.split():
    if i in wordCount:
        wordCount[i]=int(wordCount[i])+1
    else:
        wordCount[i]=1

if word in wordCount:
    print("{}: {}".format(word,wordCount[word]))
else:
    print("{} is not there.".format(word))
```

7. You have been given a dictionary as follows:

```
sampleDictionary = {
    'first' : 500,
    'bikes' : ['pulsar', 'unicorn', 'gixxer'],
    'sedans' : ['amaze','dzire', 'verna','city']
}
```

   a. Add a key to sampleDictionary called 'hatchbacks' and Set the value of 'hatchbacks' to be a list consisting of the strings 'altroz', 'swift', and 'polo'.

```
sampleDictionary ['pocket']=[ 'altroz', 'swift', 'polo']
```

   b. Sort the items in the list stored under the 'sedans' key.

```
sampleDictionary['sedans'].sort()
```

   c. Then remove 'dzire' from the list of items stored under the 'sedans' key.

```
sampleDictionary['sedans'].remove(“desire”)
```

   d. Add 50 to the number stored under the 'first' key.

```
sampleDictionary['first']= sampleDictionary['first']+50
```

8. Make a list named *shoppingList* with the values "banana","orange", and "apple". Now create two dictionaries for storing stock of different fruits and prices of different fruits. For each item in the food list, calculate the total amount to pay. Ignore whether or not the item you're billing for is in stock. Only add the price of the item to total if the item's stock count is greater than zero. If the item is in stock and after you add the price to the total,

subtract one from the item's stock count. Finally print the bill and the current stock of fruits.

```
shopping_list = ["banana", "orange", "apple"]

stock = {
    "banana": 6,
    "apple": 0,
    "orange": 32,
    "pear": 15
}

prices = {
    "banana": 4,
    "apple": 2,
    "orange": 1.5,
    "pear": 3
}


def compute_bill(food):
    total=0
    for x in food:
        price= prices[x]
        if stock[x]>0:
           total=total +price
           stock[x]=stock[x] -1
    print(total)

compute_bill(shopping_list)
```

9. You are given a list of numbers. But somebody inserted some strings also in that list. You need to calculate the square of the numbers in the list. Write a program which will be able to perform desired operation with the numbers. In case of exception print appropriate message and continue with the execution.

```
inputList=[3,4,5,"a",6]
for n in inputList:
    try:
        print("Square of {} is: {}".format(n,n*n))
    except:
        print("{} is not a number".format(n))
```

10. Write a function to subtract second number from first number. If first number is less than the second number then raise exception and handle it in your codes. Print appropriate messages.

```
def subtract(a,b):
    if a<b:
        raise
        #return 0
    else:
        return a-b

try:
    print(subtract(1, 3))
except:
    print('The subtracted number cannot be less than the subtracted number')
```

11. Write a function to check whether the age entered by user is even or odd. Raise exception if age is negative. Otherwise print appropriate message for even and odd age. Handle the exception in your code.

```
def enterage(age):
    if age < 0:
        raise

    if age % 2 == 0:
        print("age is even")
    else:
        print("age is odd")

try:
    num = int(input("Enter your age: "))
    enterage(num)
except:
    print("Only positive ages are allowed")
```

12. Suppose you have two variables and after adding these variables, it is giving type error. Write a python program to handle this exception.

```
try:
    a = 10
    b = "hello"
    c = a + b
except TypeError:
    print ('TypeError Exception Raised and successfully handled')
else:
    print ('no error!')
```

13. You have given a list of numbers. Exception will be raised if you try to assign value to a nonexistent index. Write a python code to print exception stack trace in this case.

```
import traceback
A = [1, 2, 3, 4]
try:
    value = A[5]
except:
    traceback.print_exc()
print("end of program")
```