Savitribai Phule Pune University

S. Y. B. C. A. (Science) Semester-IV

Lab Course – III

(BCA-246) Python Programming Laboratory Work Book

Name: ____

CollegeName:____

RollNo.: ____________________ **Division:** ________________________

AcademicYear: _______________________________

INTRODUCTION

1.About the work book:

This workbook is intended to be used by **S.Y.B.C.A. (Science)** students for the Python andSoftware Engineering Assignments in **Semester–IV**. This workbook is designed by considering all the practical concepts / topics mentioned in syllabus.

2. The objectives of this workbook are:

1) Defining the scope of the course.
2) To bring the uniformity in the practical conduction and implementation in all colleges affiliated to SPPU.
3) To have continuous assessment of the course and students.
4) Providing ready reference for the students during practical implementation.
5) Provide more options to students so that they can have good practice before facing the examination.
6) Catering to the demand of slow and fast learners and accordingly providing the practice assignments to them.

3. How to use this workbook:

1) The workbook is divided into two sections. Section-I is related to Python assignments and Section-II is related to Software Engineering.
2) The **Section-I (Python)** is divided into **06 assignments.** Each Python assignment has three SETs. It is mandatory for students to complete the SET A and SET B in given slot.
3) The **Section-II (Software Engineering)** is divided into **03 assignments**.

2.1Instructions to the students

Please read the following instructions carefully and follow them.

1) Students are expected to carry this book every time they come to the lab for practical.
2) Students should prepare oneself beforehand for the Assignment by reading the relevant
3) Material.
4) Instructor will specify which problems to solve in the lab during the allotted slot andstudent should complete them and get verified by the instructor. However student should spend additional hours in Lab and at home to cover as many problems as possiblegivenin this work book.
5) Studentswillbeassessedforeachexerciseonascalefrom0to5.

Not Done	**0**
Incomplete	**1**
LateComplete	**2**
Needs Improvement	**3**
Complete	**4**
WellDone	**5**

2.2 Instruction to theInstructors

1) Explainthe assignment andrelatedconcepts inaroundtenminutes usingwhiteboardif required or by demonstrating thesoftware.
2) You should evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriatebox.
3) ThevalueshouldalsobeenteredonassignmentcompletionpageoftherespectiveLab course.

Table of Contents of workbook

<u>Assignment Completion Sheet</u>

Assignment No	Description	Marks(out of 5)
1	Basic Python	
2	Python Strings	
3	Python Tuple	
4	Python Set	
5	Python Dictionary	
6	Functions in Python	
7	Problem definition & Scope of the Problem	
8	Prepare SRS for a given problem	
9	Design Data flow diagrams for the Problem	
	Total (out of 45)	
	Total (out of 15)	

ASSIGNMENT NO.1:-BASIC PYTHON

Python is a general purpose, dynamic, high level and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn. It also provides high level data structures.

The implementation of Python was started in the December 1989 by Guido Van Rossum at CWI in Netherland**.**

Starting the Interpreter

After installation, the python interpreter lives in the installed directory. By default it is /usr/local/bin/pythonX.X in Linux/Unix and C:\PythonXX in Windows, where the 'X' denotes the version number. To invoke it from the shell or the command prompt we need to add this location in the search path.
Search path is a list of directories (locations) where the operating system searches for executables. For example, in Windows command prompt, we can type set path=%path%;c:\python33 (python33 means version 3.3, it might be different in your case) to add the location to path for that particular session.

Use of Python

Python is used by hundreds of thousands of programmers and is used in many places. Python has many standard libraries which is made up of many functions that come with Python when it is installed. On the Internet there are many other libraries available that make it possible for the Python language to do more things. These libraries make it a powerful language; it can do many different things.

Some things that Python is often used for are:

- Web development
- AI & Machine learning
- Game programming
- Desktop GUI Applications
- Software Development
- Business Applications
- 3D CAD Applications
- Scientific programming
- Network programming.

First Python Program

This is a small example of a Python program. It shows "Hello World!" on the screen.
Type the following code in any text editor or an IDE and save as *helloWorld.py*

```
print ("Hello , world!")
```

Now at the command window, go to the location of this file. You can use the cd command to change directory. To run the script, type **python helloWorld.py** in the command window. We should be able to see the output as follows:

Hello, world!
If you are using PyScripter, there is a green arrow button on top. Press that button or press Ctrl+F9 on your keyboard to run the program.
In this program we have used the built-in function **print()**, to print out a string to the screen. String is the value inside the quotation marks, i.e. Hello world!. Now try printing out your name by modifying this program.

1. Immediate/Interactive mode

Typing python in the command line will invoke the interpreter in immediate mode. We can directly type in Python expressions and press enter to get the output.
>>>
is the Python prompt. It tells us that the interpreter is ready for our input. Try typing in 1 + 1 and press enter. We get 2 as the output. This prompt can be used as a calculator. To exit this mode type exit() or quit() and press enter.
Type the following text at the Python prompt and press the Enter –

```
>>> print "Hello World"
```

2. Script Mode Programming

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension **.py**. Type the following source code in a *test.py* file −

```
print"Hello World"
```

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows −

```
$ python test.py
```

This produces the following result −

```
Hello, Python!
```

3. Integrated Development Environment (IDE)

We can use any text editing software to write a Python script file.

We just need to save it with the .py extension. But using an IDE can make our life a lot easier. IDE is a piece of software that provides useful features like code hinting, syntax highlighting and checking, file explorers etc. to the programmer for application development.

Using an IDE can get rid of redundant tasks and significantly decrease the time required for application development.

IDEL is a graphical user interface (GUI) that can be installed along with the Python programming language and is available from the official website.

We can also use other commercial or free IDE according to our preference, the PyScripter IDE can be used for testing. It is free and open source.

Python Comments

In Python, there are two ways to annotate your code.

Single-line comment – Comments are created simply by beginning a line with the hash (#) character, and they are automatically terminated by the end of line.
For example:

```
#This would be a comment in Python
```

Multi-line comment- Comments that span multiple lines – used to explain things in more detail – are created by adding a delimiter (“””) on each end of the comment.

```
””” This would be a multiline comment
In Python that spans several lines and
Describes your code, your day, or anything you want it to
…
“””
```

Indentation

The enforcement of indentation in Python makes the code look neat and clean.

For example:

```
if True:
        print(‘Hello’)
        a=5
```

Incorrect indentation will result into Indentation Error.

Standard Data Types

Python has five standard data types −

- Numbers
- String
- List

- Tuple
- Dictionary

Python Numbers: Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as int, float and complex class in Python.

Python supports four different numerical types −

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Python Strings: Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes.

Example: str='Hello all'

Python Lists :

Lists are the most versatile of Python's compound data types. A list contains items can be of different data types separated by commas and enclosed within square brackets ([]).

```
list_obj=['table', 59 ,2.69,“chair”]
```

Python Tuples:

A tuple is another sequence immutable data type that is similar to the list. A tuple consists of a number of values separated by commas and enclosed in parentheses (( )).

```
Example:

tuple_obj=(786,2.23, “college” )
```

Python Dictionary

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs.

Dictionaries are enclosed by curly braces ({ })

```
Example:dict_obj={'roll_no': 15,'name':'xyz','per': 69.88}
```

Python Operators:

Python language supports the following types of operators.

- Arithmetic Operators

- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic, logical, Relational operators supported by Python language are same as other languages like C,C++.

i. **<u>Arithmetic Operators</u>**:

The new arithmetic operators in python are,

a) ** (Exponent)- Performs exponential (power) calculation on operators

Example: a**b =10 to the power 20

b) // (Floor Division) - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity)

Example: 9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

ii. **<u>Logical operators</u> :**

Logical operators are the and, or, not operators.

a) and - True if both the operands are true
b) or - True if either of the operands is true
c) not - True if operand is false (complements the operand)

iii. **<u>Relational/Comparison operators</u> :**

== (equal to), != (not equal to), < (less than),<= (**Less than or equal to**), > (greater than) and >= (**Greater than or equal to**) are same as other language relational operators.

The new relational operator in python is,

<>- If values of two operands are not equal, then condition becomes true.

Example: (a <> b) is true. This is similar to != operator.

iv. **<u>Assignment Operators</u>:** The following are assignment operators in python which are same as in C,C++.

=, +=, -=, *=, /=, %=, **=, //=

v. **<u>Bitwise Operators</u>: The following are bitwise operators in python which are same as in C,C++.**

&(bitwise AND), |(bitwise OR) ,^ (bitwise XOR),~ (bitwise NOT),<<(bitwise left shift), >>(bitwise right shift)

vi. **<u>Membership operators</u>**:

in and **not in** are the membership operators; used to test whether a value or variable is in a sequence.

in - True if value is found in the sequence
not in - True if value is not found in the sequence

vii. **<u>Identity operators</u>:**

is and **is not** are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

is - True if the operands are identical
is not - True if the operands are not identical

<u>Decision making Statement</u>

Python programming language provides following types of decision making statements.

i. **If statement**: It is similar to that of other languages

Syntax

```
if expression:
        statement(s)
```

ii. **IF...ELIF...ELSE Statements:**

Syntax

```
if expression:
        statement(s)
else:
        statement(s)
```

iii. **nested IF statements:**

In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

Syntax

```
if expression1:
        statement(s)
if expression2:
        statement(s)
elif expression3:
        statement(s)
else:
        statement(s)
elif expression4:
        statement(s)
else:
        statement(s)
```

Python – Loops

i. while loop:

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax-

```
while expression:
    statement(s)
```

Example:

```
count=0
while(count <3):
    print'The count is:', count
    count= count +1
```

ii. for loop:

It has the ability to iterate over the items of any sequence, such as a list or a string.

Syntax

```
foriterating_var in sequence:
    statements(s)
```

Example:

```
for x in'Hi':

    print x
```

Command Line Arguments

You can get access to the command line parameters using the sys module. len(sys.argv) contains the number of arguments. To print all of the arguments simply execute str(sys.argv)

```
import sys
print('Arguments:', len(sys.argv))
print('List:', str(sys.argv))
```

Storing command line arguments

```
import sys
print('Arguments:', len(sys.argv))
print('List:', str(sys.argv))
if sys.argv< 2:
   print('To few arguments, please specify a filename')
```

```
filename = sys.argv[1]
print('Filename:', filename)
```

Set A]

1) Write a Python Program to Calculate the Average of Numbers in a Given List.

2) Write a program which accepts 6 integer values and prints “DUPLICATES” if any of the values entered are duplicates otherwise it prints “ALL UNIQUE”.
   Example: Let 5 integers are (32, 10, 45, 90, 45, 6) then output “DUPLICATES” to be printed.

3) Write a program which accepts an integer value as command line and print “Ok” if value is between 1 to 50 (both inclusive) otherwise it prints” Out of range”

4) Write a program which finds sum of digits of a number.
   Example n=130 then output is 4 (1+3+0).

5) Write a program which prints Fibonacci series of a number.

Set B]

1) Write a program which accept an integer value ‘n’ and display all prime numbers till ‘n’.

2) Write a program that accept two integer values and if both are equal then prints “SAME identity” otherwise prints, “DIFFERENT identity”.

3) Write a program to display following pattern.

```
1    2    3    4
1    2    3
1    2
1
```

4) Write a program to reverse a given number.

Set C]

1) Write a Sequential search function which searches an item in a sorted list. The function should return the index of element to be searched in the list.

Assignment Evaluation

0: Not Done [ ]	1 : Incomplete [ ]	2 : Late Complete [ ]
3 : Needs Improvement [ ]	4 : Complete [ ]	5 : Well Done [ ]

Signature of Instructor

ASSIGNMENT NO.2:- PYTHON STRING

Python string is a built-in type text sequence. It is used to handle **textual data** in python.

A string is a sequence of characters. A character is simply a symbol. For example, the English language has 26 characters. Computers do not deal with characters; they deal with numbers (binary). Even though you may see characters on your screen, internally it is stored and manipulated as a combination of 0's and 1's.

This conversion of character to a number is called encoding, and the reverse process is decoding. ASCII and Unicode are some of the popular encoding used.

In Python, string is a sequence of Unicode character. Unicode was introduced to include every character in all languages and bring uniformity in encoding.

We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes. Creating strings is as simple as assigning a value to a variable. For example –

```
var1 = 'Hello World!'
var2 = "Python Programming"
```

Accessing Values in Strings

Python does not support a character type; these are treated as strings of length one, thus also considered a substring.

To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring. For **Example:1)**
–

```
#!/usr/bin/python

var1 = 'Hello World!'
var2 = "Python Programming"

print "var1[0]: ", var1[0]
print "var2[1:5]: ", var2[1:5]
```

Updating Strings

You can "update" an existing string by *re*assigning a variable to another string. The new value can be related to its previous value or to a completely different string altogether. For example −

```
#!/usr/bin/python

var1 = 'Hello World!'

print "Updated String: - ", var1[:6] + 'Python'
```

When the above code is executed, it produces the following result −

```
Updated String: -        Hello Python
```

Example:2)a = “Pravin”
b = “Kulkarni”
print (a+” “+b)
output: Pravin Kulkarni

Escape Characters

Following table is a list of escape or non-printable characters that can be represented with backslash notation.

An escape character gets interpreted; in a single quoted as well as double quoted strings.

BackslashNotation	Description
\a	Bell or alert
\b	Backspace
\cx or \C-x	Control-x
\e	Escape
\f	Formfeed
\M-\C-x	Meta-Control-x
\n	Newline
\nnn	Octal notation, where n is in the range 0.7
\r	Carriage return
\s	Space
\t	Tab
\v	Vertical tab
\x	Character x

String Special Operators

Assume string variable **a** holds 'Hello' and variable **b** holds 'Python', then −

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello

[]	Slice - Gives the character from the given Index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
In	Membership - Returns true if a character exists in the given string	H in a will give 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give 1
%	Format - Performs String formatting	See at next section

String Formatting Operator

One of Python's coolest features is the string format operator %. This operator is unique to strings and makes up for the pack of having functions from C's printf family. Following is a simple example –

```
#!/usr/bin/python
print "My name is %s and weight is %d kg!" % ('Kamil ', 65)
```

When the above code is executed, it produces the following result −

```
My name is Kamil and weight is 65 kg!
```

Here is the list of complete set of symbols which can be used along with % −

FormatSymbol	Conversion
%c	Character
%s	string conversion via str prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer *lowercaseletters*
%X	hexadecimal integer *UPPERcaseletters*
%e	exponential notation *withlowercase'e'*
%E	exponential notation *withUPPERcase'E'*
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

Other supported symbols and functionality are listed in the following table –

Symbol	Functionality
*	argument specifies width or precision
-	left justification
+	display the sign
<sp>	leave a blank space before a positive number
#	add the octal leading zero ′0′ or hexadecimal leading '0x' or '0X',
	depending on whether 'x' or 'X' were used.
0	pad from left with zeros *insteadofspaces*
%	'%%' leaves you with a single literal '%'
Var	mapping variable *dictionary arguments*
m.n.	m is the minimum total width and n is the number of digits to display after the decimal point *ifappl*.

Triple Quotes

Python's triple quotes comes to the rescue by allowing strings to span multiple lines, including verbatim NEWLINEs, TABs, and any other special characters.

The syntax for triple quotes consists of three consecutive **single or double** quotes.

```
#!/usr/bin/python
para_str = """this is a long string that is made up of several lines and non-printable characters such
as TAB ( \t ) and they will show up that way when displayed. NEWLINEs within the string, whether
explicitly given like this within the brackets [ \n ], or just a NEWLINE within the variable
assignment will also show up. """
print para_str
```

When the above code is executed, it produces the following result. Note how every single special character has been converted to its printed form, right down to the last NEWLINE at the end of the string between the "up." and closing triple quotes. Also note that NEWLINEs occur either with an explicit carriage return at the end of a line or its escape code \n −

```
this is a long string that is made up of
several lines and non-printable characters such as
TAB ( ) and they will show up that way when displayed. NEWLINEs within the string, whether
explicitly given like this within the brackets [
], or just a NEWLINE within
the variable assignment will also show up.
```

Raw strings do not treat the backslash as a special character at all. Every character you put into a raw string stays the way you wrote it −

```
#!/usr/bin/python
print 'C:\\nowhere'
```

When the above code is executed, it produces the following result −

```
C:\nowhere
```

Built-in String Methods

Python includes the following built-in methods to manipulate strings −

Sr.No.	Methods	Description
1	**Capitalize()**	Capitalizes first letter of string
2	**count(str, beg= 0,end=len(string))**	Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
3	**endswith(suffix, beg=0, end=len(string))**	Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.
4	**isalnum()**	Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
5	**isalpha()**	Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
6	**isdigit()**	Returns true if string contains only digits and false otherwise.
7	**islower()**	Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
8	**isnumeric()**	Returns true if a unicode string contains only numeric characters and false otherwise.
9	**isspace()**	Returns true if string contains only whitespace characters and false otherwise.
10	**istitle()**	Returns true if string is properly "titlecased" and false otherwise.
11	**isupper()**	Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
12	**join(seq)**	Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
13	**len(string)**	Returns the length of the string
14	**ljust(width[, fillchar])**	Returns a space-padded string with the original string left-justified to a total of width columns.
15	**lower()**	Converts all uppercase letters in string to

		lowercase.
16	**lstrip()**	Removes all leading whitespace in string.
17	**maketrans()**	Returns a translation table to be used in translate function.
18	**max(str)**	Returns the max alphabetical character from the string str.
19	**min(str)**	Returns the min alphabetical character from the string str.
20	**replace(old, new [, max])**	Replaces all occurrences of old in string with new or at most max occurrences if max given.
21	**rfind(str, beg=0,end=len(string))**	Same as find(), but search backwards in string.
22	**rjust(width,[, fillchar])**	Returns a space-padded string with the original string right-justified to a total of width columns.
23	**rstrip()**	Removes all trailing whitespace of string.
24	**split(str="", num=string.count(str))**	Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.
25	**splitlines(num=string.count('\n'))**	Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed.
26	**swapcase()**	Inverts case for all letters in string.
27	**title()**	Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase.
28	**translate(table, deletechars="")**	Translates string according to translation table str(256 chars), removing those in the del string.
29	**upper()**	Converts lowercase letters in string to uppercase.
30	**zfill (width)**	Returns original string leftpadded with zeros to a total of width characters;

		intended for numbers, zfill() retains any sign given (less one zero).
31	**isdecimal()**	Returns true if a unicode string contains only decimal characters and false otherwise.

SET A]

1. Write a program to replace all occurrences of 'a' with $ in a String. (Ex. apple then output is $pple).

2. Write a Python program to count the number of characters (character frequency) in a string.
Sample String: google.com'
Expected Result : {'o': 3, 'g': 2, '.': 1, 'e': 1, 'l': 1, 'm': 1, 'c': 1}

3. Write a Python program to get a string made of the first 2 and the last 2 chars
from a given a string. If the string length is less than 2, return instead of the empty string.
Sample String : 'General12'
Expected Result : 'Ge12'
Sample String : 'Ka'
Expected Result : 'KaKa'
Sample String : ' K'
Expected Result : Empty String

4. Write a Python program to calculate the Length of a String without using a Library Function.

5. Write a Python program to get a single string from two given strings, separated by a space and swap the first two characters of each string.
Sample String: 'ppk', 'abc'
Expected Result: 'abkppc'

SET B]

1**.** Write a python program to check if a string is a Palindrome or Not.

2. Write a Python program to calculate the Number of Digits and Letters in a string.

3. Write a Python program to remove the characters which have odd index values of a given string.

4. Write a Python program to count the occurrences of each word in a given sentence.

SET C]

1. Remove special symbols/Punctuation from a given string.

Given:

```
str1 = "/*Sachin is @Cricketer& kind person"
```

Expected Output:

“Sachin is Cricketerkind person”

Assignment Evaluation

0: Not Done [ ]   1: Incomplete [ ]   2: Late Complete [ ]
3: Needs Improvement [ ]   4: Complete [ ]   5: Well Done [ ]

Signature of Instructor

ASSIGNMENT NO. 3 : PYTHON TUPLE

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists.Tuples use **parentheses {}**, whereas lists use **square []** brackets.
Tuple is similar to list. Only the difference is tat list is enclosed between square bracket, tuple between parenthesis and List has **mutable** objects where as Tuple has **an immutable object**.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example −

```
Tuple = () # empty tuple
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containing nothing −

```
tup1 = ();
```

To write a tuple containing a single value you have to include a comma, even though there is only one value −

```
tup1 = (50,);
```

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Accessing Values in Tuple:

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

```
#!/usr/bin/python

tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5, 6, 7 );

print "tup1[0]: ", tup1[0]
print "tup2[1:5]: ", tup2[1:5]
```

When the above code is executed, it produces the following result −

```
tup1[0]:      physics
tup2[1:5]:      [2, 3, 4, 5]
```

Updating Tuple

Tuples are immutable which means you cannot update or change the values of tuple elements.
You are able to take portions of existing tuples to create new tuples as the following example

demonstrates –

```
#!/usr/bin/python

tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');

#  Following action is not valid for tuples
#  tup1[0] = 100;

#  So let's create a new tuple as follows
tup3 = tup1 + tup2;
print tup3
```

When the above code is executed, it produces the following result −

```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement. For example:

```
#!/usr/bin/python

tup = ('physics', 'chemistry', 1997, 2000);

print tup
del tup;
print "After deleting tup : "
print tup
```

This produces the following result. Note an exception raised, this is because after **del tup** tuple does not exist any more −

```
('physics', 'chemistry', 1997, 2000)
After deleting tup :
Traceback (most recent call last):
   File "test.py", line 9, in <module>
      print tup;
NameError: name 'tup' is not defined
```

Basic Tuple Operations

Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.

Python Expression	Results	Description
len(1, 2, 3)	3	Length
1, 2, 3 + 4, 5, 6	1, 2, 3, 4, 5, 6	Concatenation
′*Hi*!′,* 4	′*Hi*!′,′*Hi*!′,′*Hi*!′,′*Hi*!′	Repetition
3 in 1, 2, 3	True	Membership
for x in 1, 2, 3: print x,	1 2 3	Iteration

Indexing, Slicing, and Matrixes

Because tuples are sequences, indexing and slicing work the same way for tuples as they do for strings. Assuming following input −

L = ('spam', 'Spam', 'SPAM!')

Python Expression	**Results Description**	
L[2]	'SPAM!'	Offsets start at zero
L[-2]	'Spam'	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

No Enclosing Delimiters

Any set of multiple objects, comma-separated, written without identifying symbols, i.e., brackets for lists,parentheses for tuples, etc., default to tuples, as indicated in these short examples −

```
#!/usr/bin/python

print 'abc', -4.24e93, 18+6.6j, 'xyz'
x, y = 1, 2;
print "Value of x , y : ", x,y
```

When the above code is executed, it produces the following result −

```
abc -4.24e+93 (18+6.6j) xyz
Value of x , y : 1 2
```

Built-in Tuple Functions
Python includes the following tuple functions

Function	**Description**
all()	Return True if all elements of the tuple are true (or if the tuple is empty).
any()	Return True if any element of the tuple is true. If the tuple is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of tuple as pairs.
len()	Return the length (the number of items) in the tuple.
max()	Return the largest item in the tuple.
min()	Return the smallest item in the tuple
sorted()	Take elements in the tuple and return a new sorted list (does not sort the tuple itself).
sum()	Retrun the sum of all elements in the tuple.
tuple()	Convert an iterable (list, string, set, dictionary) to a tuple.

SET A]

1. Reverse the following tuple
   aTup = (10, 20, 30, 40, 50)
2. Write a Python program to create a list of tuples with the first element as the number and second element as the square of the number.
3. Write a Python program to create a tuple with numbers and print one item.
4. Write a Python program to unpack a tuple in several variables.
5. Write a Python program to add an item in a tuple.
6. Copy element 44 and 55 from the following tuple into a new tuple
   tuple1 = (11, 22, 33, 44, 55, 66)

SET B]

1. Write a Python program to convert a tuple to a string.
2. Sort the tuple - Tuple=(2, 4, 6, 1, 4, 7.8, 2.7)
3. Write a Python program to get the 5th element from front and 5th element from last of a tuple.
4. Write a Python program to find the repeated items of a tuple.
5. Write a Python program to check whether an element exists within a tuple.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor:

ASSIGNMENT NO. 4:- PYTHON SET

A set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable (which cannot be changed).

However, the set itself is mutable. We can add or remove items from it.

Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

How to create a set?

A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set().

It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like list, set or dictionary, as its element.

Example

```
# set of integers
my_set = {1, 2, 3}
print(my_set)
# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```

Output:
```
{1, 2, 3}
{1.0, (1, 2, 3), 'Hello'}
```

Creating an empty set is a bit tricky.

Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements we use the set() function without any argument.

```
# initialize a with {}
a = {}
# check data type of a
# Output: <class 'dict'>
```

```
print(type(a))
# initialize a with set()
a = set()
# check data type of a
# Output: <class 'set'>
print(type(a))
```

How to change a set in Python?

Sets are mutable. But since they are unordered, indexing have no meaning.
We cannot access or change an element of set using indexing or slicing. Set does not support it.
We can add single element using the add() method and multiple elements using the update() method. The update() method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

```
# initialize my_set
my_set = {1,3}
print(my_set)

# if you uncomment line 9,
# you will get an error
# TypeError: 'set' object does not support indexing

#my_set[0]

# add an element
# Output: {1, 2, 3}
my_set.add(2)
print(my_set)

# add multiple elements
# Output: {1, 2, 3, 4}
my_set.update([2,3,4])
print(my_set)

# add list and set
# Output: {1, 2, 3, 4, 5, 6, 8}
my_set.update([4,5], {1,6,8})
print(my_set)
```

How to remove elements from a set?

A particular item can be removed from set using methods, discard() and remove().

The only difference between the two is that, while using discard() if the item does not exist in the set, it remains unchanged. But remove() will raise an error in such condition.

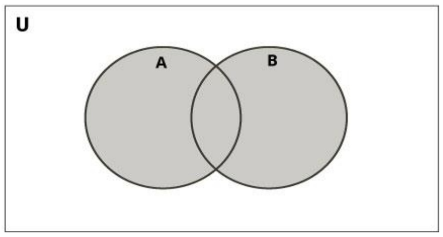
```
my_set = {1, 3, 4, 5, 6}
print(my_set)
# discard an element
# Output: {1, 3, 5, 6}
my_set.discard(4)
print(my_set)
# remove an element
# Output: {1, 3, 5}
my_set.remove(6)
print(my_set)
# discard an element
# not present in my_set
# Output: {1, 3, 5}
my_set.discard(2)
print(my_set)
```

Python Set Operations

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods.Let us consider the following two sets for the following operations.

```
>>> A ={1,2,3,4,5}
>>> B ={4,5,6,7,8}
```

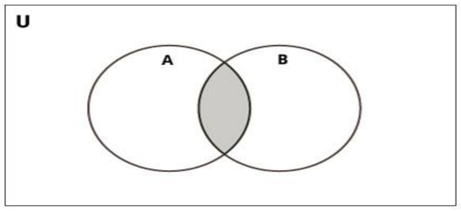
Set Union



Union of *A* and *B* is a set of all elements from both sets.

Union is performed using | operator. Same can be accomplished using the method union()

Set Intersection



Intersection of *A* and *B* is a set of elements that are common in both sets.

Intersection is performed using & operator. Same can be accomplished using the method intersection().

```
A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

#union

print(A | B)

# InterSection

# Output: {4, 5}

print(A& B)
```

Method	**Description**
add()	Add an element to a set
clear()	Remove all elements form a set
copy()	Return a shallow copy of a set
difference()	Return the difference of two or more sets as a new set
difference_update()	Remove all elements of another set from this set

discard()	Remove an element from set if it is a member. (Do nothing if the element is not in set)
intersection()	Return the intersection of two sets as a new set
intersection_update()	Update the set with the intersection of itself and another
isdisjoint()	Return True if two sets have a null intersection
issubset()	Return True if another set contains this set
issuperset()	Return True if this set contains another set
pop()	Remove and return an arbitary set element. Raise KeyErrorif the set is empty
remove()	Remove an element from a set. If the element is not a member, raise a KeyError
symmetric_difference()	Return the symmetric difference of two sets as a new set
symmetric_difference_update()	Update a set with the symmetric difference of itself and another
union()	Return the union of sets in a new set
update()	Update a set with the union of itself and others

Built-in Functions with Set

Built-in functions like all(), any(), enumerate(), len(), max(), min(), sorted(), sum() etc. are commonly used with set to perform different tasks.

Function	Description
all()	Return True if all elements of the set are true (or if the set is empty).
any()	Return True if any element of the set is true. If the set is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of set as a pair.

len()	Return the length (the number of items) in the set.
max()	Return the largest item in the set.
min()	Return the smallest item in the set.
sorted()	Return a new sorted list from elements in the set(does not sort the set itself).
sum()	Retrun the sum of all elements in the set.

SET A]

1. What is the output of following program:

   ```
   sets = {1, 2, 3, 4, 4}
   print(sets)
   ```
2. Write a python program to remove and return an arbitrary set element.Raise KeyErrorif the set is empty
3. Write a Python program to do iteration over sets.
4. Write a Python program to add and remove operation on set.

SET B]

1. Write a Python program to accept the strings which contains all vowels .
2. Write a Python program to create a union of sets.
3. Write a Python program to create an intersection of sets.
4. Write a Python program to find maximum and the minimum value in a set.
5. Write a Python program to create set difference and a symmetric difference
6. Write a Python program to find the length of a set.

SET C]

1. Write a Python program to perform different set operations.
2. Write a Python program to create a shallow copy of sets.

Note: Shallow copy is a bit-wise copy of an object. A new object is created that has an exact copy of the values in the original object.

Assignment Evaluation

0: Not Done [ ]   1: Incomplete [ ]   2: Late Complete [ ]

3: Needs Improvement [ ]   4: Complete [ ]   5: Well Done [ ]

Signature of Instructor

ASSIGNMENT NO.5:- PYTHON DICTIONARY

Definition:
Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.
Dictionaries are optimized to retrieve values when the key is known.

How to create a dictionary?

Creating a dictionary is as simple as placing items inside curly braces {} separated by comma.

An item has a key and the corresponding value expressed as a pair, key: value.

While values can be of any data type and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.

```
# empty dictionary
my_dict = {}
# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}
# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}
# using dict()
my_dict = dict({1:'apple', 2:'ball'})
# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
```

How to access elements from a dictionary?

While indexing is used with other container types to access values, dictionary uses keys. Key can be used either inside square brackets or with the get() method.

The difference while using get() is that it returns None instead of KeyError, if the key is not found.

```
my_dict = {'name':'Jack', 'age': 26}

# Output: Jack
print(my_dict['name'])

# Output: 26
print(my_dict.get('age'))
```

How to change or add elements in a dictionary?

Dictionary is mutable. We can add new items or change the value of existing items using assignment operator.

If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

```
my_dict = {'name':'aliza', 'age': 16}

# update value
my_dict['age'] = 17

#Output: {'age': 17, 'name': 'aliza'}
print(my_dict)

# add item
my_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 17, 'name': aliza'}
print(my_dict)
```

Python Dictionary Methods

Methods that are available with dictionary are tabulated below. Some of them have already been used in the above examples.

Method	**Description**
clear()	Remove all items from the dictionary.
copy()	Return a shallow copy of the dictionary.
fromkeys(seq[, v])	Return a new dictionary with keys from seq and value equal to v(defaults to None).
get(key[,d])	Return the value of key. If key doesnot exit, return d (defaults to None).
items()	Return a new view of the dictionary's items (key, value).
keys()	Return a new view of the dictionary's keys.
pop(key[,d])	Remove the item with key and return its value or d if key is not found.

	If d is not provided and key is not found, raises KeyError.
popitem()	Remove and return an arbitary item (key, value). Raises KeyError if the dictionary is empty.
setdefault(key[,d])	If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None).
update([other])	Update the dictionary with the key/value pairs from other, overwriting existing keys.
values()	Return a new view of the dictionary's values

SET A]

1. Write a Python script to access the value of a key from a dictionary.
2. Write a Python script to concatenate following dictionaries to create a new one.
   Sample Dictionary:
   dic1={1:10,2:20}
   dic2={3:30,4:40}
   dic3={5:50,6:60}
   Expected Result : {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
3. Write a Python program to iterate over dictionaries using for loops.
4. Write a Python program to sum all the items in a dictionary.
   Sample Dictionary: my_dict=**{'data1':100,'data2':-54,'data3':247}**
   Expected Result: 293
5. Write a Python program to remove a key from a dictionary.
   Sample Dictionary: myDict=**{'a':1,'b':2,'c':3,'d':4}**
   Sample Output:
   {'c': 3, 'b': 2, 'd': 4, 'a': 1}
   {'c': 3, 'b': 2, 'd': 4}

SET B]

1. Write a Python program to sort a dictionary by key.
Sample Dictionary:
   color_dict = {'red':'#FF0000','green':'#008000','black':'#000000','white':'#FFFFFF'}
   Expected Output:
   black: #000000
   green: #008000
   red: #FF0000
   white: #FFFFFF
2. Write a Python program to combine two dictionary adding values for common keys.

Sample Dictionary:
d1={'a':100,'b':200,'c':300}
d2={'a':300,'b':200,'d':400}
Sample output: Counter({'a': 400, 'b': 400, 'd': 400, 'c': 300})

3. Write a Python script to generate and print a dictionary that contains a number

   (Between 1 and n) in the form (x, x*x).
   Sample Dictionary (n = 5) :
   Expected Output : {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
4. Write a Python program to create a dictionary from a string.
   Sample-String:'W3resource'
   Expected output: {'3': 1, 's': 1, 'r': 2, 'u': 1, 'w': 1, 'c': 1, 'e': 2, 'o': 1}

SET C]

1. Write a Python program to create a dictionary from two lists without losing duplicate values.
   Sample lists: ['Class-V', 'Class-VI', 'Class-VII', 'Class-VIII'], [1, 2, 2, 3]
   Expected Output: defaultdict(<class 'set'>, {'Class-VII': {2}, 'Class-VI': {2}, 'Class-VIII': {3}, 'Class-V': {1}})
2. Write a Python program to create a dictionary of keys x, y, and z where each key has as value a list from 11-20, 21-30, and 31-40 respectively. Access the fifth value of each key from the dictionary.

Expected Output:

```
{'x': [11, 12, 13, 14, 15, 16, 17, 18, 19],
'y': [21, 22, 23, 24, 25, 26, 27, 28, 29],
'z': [31, 32, 33, 34, 35, 36, 37, 38, 39]}
15
25
35
x has value [11, 12, 13, 14, 15, 16, 17, 18, 19]
y has value [21, 22, 23, 24, 25, 26, 27, 28, 29]
z has value [31, 32, 33, 34, 35, 36, 37, 38, 39]
```

Assignment Evaluation

0: Not Done [ ] 1: Incomplete [ ] 2: Late Complete [ ]

3: Needs Improvement [ ] 4: Complete [ ] 5: Well Done [ ]

Signature of Instructor

ASSIGNMENT NO.6:-PYTHON FUNCTIONS

Function:
A function is a block of organized, reusable code that is used to perform a single, related action. Functions give modularity and reusing of code in a program.
There are many built-in functions like in Python but you can also create your own functions. These functions are called user-defined functions.

How to define a function
SYNTAX:

```
                    Def functionname( parameters ):
"function_docstring"
                    function_body
return [expression]
```

The function blocks starts with the keyword ***def*** followed by the function name and parentheses and colon(:).
The input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
The first statement of a function is an optional statement - the documentation string of the function or docstring.
The code block within every function starts with a colon (:) and is indented.
The statement return [expression] exits a function, the return statement with no arguments means return Nothing.

Consider the following function:

```
def revno(n):
        sum=0
        while (n>0):
        rem=n%10
n=n/10
sum=(sum*10)+rem
        return sum

print "The reverse of 54321 is :",revno(54321)
```

Calling a function
Once the function is defined, you can execute it by calling it from another function or directly from the Python prompt. In the above code the function is called from print statement.

Function Arguments:
There are the following types of formal arguments:
1. Required arguments
2. Keyword arguments
3. Default arguments

4. Variable-length arguments

Required arguments
Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.

```
#!/usr/bin/python
def display( str ):
printstr;
return;
# call the function
display("hello");
```

Keyword Arguments:
Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name. In this we can change the order of arguments or may skip it.

```
#!/usr/bin/python
def display( classname, roll_no ):
print "Class: ", classname;
print "Roll_no ", roll_no;
return;
# call the function
display(roll_no=10, classname="TYBCA" );
```

Default Arguments:
A default argument is an argument that considers a default value if a value is not provided in the function call for that argument.

```
#!/usr/bin/python
def display( classname, roll_no=11 ):
print "Class: ", classname;
print "Roll_no ", roll_no;
return;
display(classname="TYBCA", roll_no=10 );
display("SYBCA");
```

output is…

```
Class:  TYBCA
Roll_no  10
Class:  SYBCA
Roll_no  11
```

Variable length arguments:
We may require more arguments than specified while defining the function. These arguments are called variable-length arguments and are not named in the function definition.
def functionname([formal_args,] *var_args_tuple):

"function_docstring"
function_body
return [expression]

An asterisk (*) is placed before the variable name that will hold the values of all non keyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call.

```
#!/usr/bin/python
def display( arg1, *varargs ):
print "Output is: "
print arg1
forvar in varargs:
printvar
return;
# call the function
display( 100 );
display( 10, 20, 30 );
Output is:
100
Output is:
10 20 30
```

Return Statement:
The statement return [expression] exits a function, passing back an expression to the caller.

```
defcheckprime(n):
if n>1 :
fori in range(2,n):
if(n%i)==0:
return 0
return 1

a=checkprime(10)
if a==1:
print " no is prime"
else:
print " no is not prime"
```

Functions returning multiple values:

```
def display(x, y):
return x * 3, y * 4
a, b = display(5, 4)
print a
print b
```

Anonymous Functions:
The anonymous functions are not declared in the standard manner by using the def keyword. The lambda keyword is used to create small anonymous functions.

- Lambda forms can take any number of arguments but return just one value in the form of an expression.
- They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to print because lambda requires an expression.
- Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.

SYNTAX:
lambda [arg1 [,arg2,.....argn]]:expression

```
#!/usr/bin/python
total = lambda arg1, arg2: arg1 + arg2;
print "Value of total : ", total( 100, 20 )
print "Value of total : ", total( 200, 10 )
#output
Value of total :  120
Value of total :  210
```

Scope   of Variables:
All variables in a program may not be accessible at all locations in that program. It depends on where you have declared a variable.The scope of a variable is the portion of the program where you can access it. Thereare two basic scopes of variables in Python:
- Global variables
- Local variables

Variables that are defined inside a function body arehaving local scope, and defined outside the function body have a global scope.

Recursive Functions
The function which calls itself is a recursive function. Python allows us to write recursive functions.
Recursive function for factorial of a number:

```
def fact(n):
if(n==1):
return 1
else:
return n* fact(n-1)
print fact(4)
```

Function ducktyping

In the Python programming, the duck type is a type of dynamic style. In this style, the effective semantics of an object are determined by the current set of methods and properties rather than inheriting from a specific class or by implementing a particular interface.
In the duck type, the concern is not the type of object itself, but how it is used.
Consider the following code in which there is fly method for Parrot and Airplane classes but not for Whale class, so error will be generated for whale object

```
class Parrot:
def fly(self):
print("Parrot flying")
class Airplane:
def fly(self):
print("Airplane flying")
class Whale:
def swim(self):
print("Whale swimming")
def action(entity):
entity.fly()
parrot = Parrot()
airplane = Airplane()
whale = Whale()
action(parrot) # prints `Parrot flying`
action(airplane) # prints `Airplane flying`
action(whale) # Throws the error `'Whale' object has no attribute 'fly'`
```

```
class Duck:
def quack(self):
   print "Quaaaaaack!"
class Bird:
def quack(self):
   print "bird imitate duck."
class Doge:
def quack(self):
   print "doge imitate duck."
defin_the_forest(duck):
duck.quack()
duck = Duck()
bird = Bird()
doge = Doge()
for x in [duck, bird, doge]:
in_the_forest(x)
```

List comprehension:

List comprehension is the way to define and create list in Python. These lists have often the qualities of sets, but are not in all cases sets.
List comprehension is a complete substitute for the lambda function as well as the functions map(), filter() and reduce(). Consider the list comprehension to convert Celsius values into Fahrenheit :

```
Celsius = [39.2, 36.5, 37.3, 37.8]
Fahrenheit = [ ((float(9)/5)*x + 32) for x in Celsius ]
print Fahrenheit
######output####
[102.56, 97.7, 99.14, 100.03999999999999]
```

Cross product of two sets:

```
colours = [ "red", "green", "yellow", "blue" ]
things = [ "house", "car", "tree" ]
coloured_things = [ (x,y) for x in colours for y in things ]
printcoloured_things
output
[('red', 'house'), ('red', 'car'), ('red', 'tree'), ('green', 'house'), ('green', 'car'), ('green', 'tree'), ('yellow',
'house'), ('yellow', 'car'), ('yellow', 'tree'), ('blue', 'house'), ('blue', 'car'), ('blue', 'tree')]
```

Unpacking argument list
Python allows us to use variable number of arguments. We can also change the values inside functions using argument unpacking
packing and unpacking allows us to do:
1. validate arguments before passing them
2. set defaults for positional arguments
3. create adaptors for different pieces of code / libraries
4. modify arguments depending on context
5. log calls to methods

Consider the following code:

```
def func1(x, y, z):
print x
print y
    print z
def func2(*args):
    # Convert args tuple to a list so we can modify it
args = list(args)
args[0] = 'Hello'
args[1] = 'Everybody'
func1(*args)
```

```
func2('Goodbye', 'Hi', 'welcome')
####output
Hello
Everybody
welcome
```

Generator function

```
def infinite_generator(start=0):
        while True:
                yield start
                start += 1

for num in infinite_generator(4):
        print num
if num> 20:
        break
```

Try this generator function:

```
def vowels():
yield "a"
yield "e"
yield "i"
yield "o"
yield "u"
for i in vowels():
print(i)
```

Consider the following example for iterator and generator

```
def  simplegenerator():
        yield 'aaa'
        yield 'bbb'
        yield 'ccc'

def list_tripler(somelist):
        for item in somelist:
                item *= 3
                yield item

def limit_iterator(somelist, max):
        for item in somelist:
                if item > max:
                        yield item
def test():
        itr = simplegenerator()
        for item in itr:
                print item


alist = range(5)
it = list_tripler(alist)
for item in it:
        print item
alist = range(8)
        it = limit_iterator(alist, 4)
for item in it:
print item
it = simplegenerator()
try:
printit.next()
     print it.next()
printit.next()
printit.next()
exceptStopIteration, exp:
     print 'reached end of sequence'
if __name__ == '__main__':
test()
```

SET A]

1. Write an anonymous function to calculate area of square.
2. Write a Python function to multiply all the numbers in a list.
   ***Sample-List* :(8,2,3,-1,7)**
   ***Expected Output* : -336**
3. Write a Python function to check whether a number is in a given range.
4. Create a function showEmployee() in such a way that it should accept employee name, and it’s salary and display both, and if the salary is missing in function call it should show it as 9000

SET B]

1. Write a Python function that takes a number as a parameter and check the number is prime or not.
2. Write a generator function that reverses a given string.
3. Write a recursive function to calculate the sum of numbers from 0 to 10.
4. Write a Python program to filter a list of integers using Lambda

SET C]

1. Create an inner function to calculate the addition in the following way

- Create an outer function that will accept two parameters a and b
- Create an inner function inside an outer function that will calculate the addition of a and b.
- At last, an outer function will add 5 into addition and return it.

Assignment Evaluation

0: Not Done [ ]   1: Incomplete [ ]   2: Late Complete [ ]

3: Needs Improvement [ ]   4: Complete [ ]   5: Well Done [ ]

Signature of Instructor

<u>Assignment No. 7:Problem Definition and Scope of the Problem</u>

Mini Project for Designing Backend using Software Engineering Techniques

The Mini project is to be carried out using following steps and document them by filling the forms provided herewith

Step 1 – Form the team of two students

Step 2 – Identify the problems that involves data processing and finalize the project after discussing with your teacher guide.

Step 3 – Understand the problem, study the existing system and prepare a problem description and present drawbacks of existing system and scope of the proposed System

Step 4- Prepare software requirement specification for a given problem.

Step 5 – Prepare data flow diagram for a given problem.

The certificate for the mini project is as follows:

S. Y. B.C.A. (Science)

Mini Project

Academic Year (2020- 2021)

Project Title:____

____

____

Team members:

1) Name : ____

Roll No . Exam Seat No:

2) Name : ____

Roll No .Exam Seat No:

Project Guide Name:____

Project Guide Signature: _______________

Problem Description

Study of Existing system(Manual or Computerized)

Drawbacks of Existing system

Scope of the Proposed System

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment no 8: Prepare SRS for a given problem

SOFWARE REQUIREMENT SPECIFICATION

A **software requirements specification** (SRS) is a detailed description of a software system to be developed with its functional and non-functional requirements. The SRS is developed based the agreement between customer and contractors.
It may include the use cases of how user is going to interact with software system.

- ✓ The software requirement specification document consistent of all necessary requirements required for project development. To develop the software system we should have clear understanding of Software system. To achieve this we need to continuous communication with customers to gather all requirements.
- ✓ A good SRS defines the how Software System will interact with all internal modules, hardware, communication with other programs and human user interactions with wide range of real life scenarios.
- ✓ Using the **Software requirements specification** ***(SRS)*** document on QA lead, managers creates test plan. It is very important that testers must be cleared with every detail specified in this document in order to avoid faults in test cases and its expected results.

- ✓ It is highly recommended to review or test SRS documents before start writing test cases and making any plan for testing. Let's see how to test SRS and the important point to keep in mind while testing it.

Table of Contents

4. Software Quality Attributes
5. Other requirements

Appendix (if any)

Sample SRS of Library Management System

1. Introduction:

A development process consist of various phases, each phase ending with a defined output. The main reason for having a phased process is that it breaks the problem of developing software into successfully performing a set of phases, each handling a different concern of software development. This ensures that the cost of development is lower than what it would have been if the whole problem was tackled together. Furthermore, a phased process allows proper checking for quality and progress at some defined points during the development (end of process).Without this one would have to wait until the end to see what software has been produced.

Any problem solving in software consist of these steps:-

Requirement Analysis:

Requirement Analysis is done in order to understand the problem thesoftware system is to solve. There are two major activities in this phase: problem understanding oranalysis and requirement specification. In problem analysis, the aim is tounderstand the problem and its context, and the requirements of the newsystem that is to be developed. Once the problem is analyzed and theessentials understood, the requirements must be specified in the requirement Specification document. The requirements document must specify allfunctional and performance requirements; the formats of inputs and outputsetc.

● **Software Design:**

The purpose of design phase is to plan a solution of the problemSpecified by the requirements document. This phase is the first step in movingfrom the problem domain to solution domain.

The design activity often results in three separate outputs: architecture design,high level design and detailed design.

● **Coding:**

The main goal of coding phase is to translate the design of theSystem into code in a given programming language. The coding phase affectsboth testing and maintenance profoundly. The goal of coding should be to reducethe testing and maintenance effort, hence during coding the focus should be ondeveloping programs that are easy to read and understand.

● **Testing:**

The function of testing is to detect the defects in theSoftware. The main goal testing is to uncover requirement, design and codingerrors in the programs. The main goal of the requirement phase is to produce the software requirementspecification (SRS), which accurately capture the client's requirements. SRS isA document that describes what the software should do. The basic purpose of SRSis to bridge the communication gap between the clients, the end users and the Software developers. Another purpose is helping user to understand their ownneeds.

1.1 Purpose:

The SRS typically contains the brief description of the project. The purposeof the requirement document is to specify all the information required to design, develop and test the software.

● The purpose of this project is to provide a friendly environment to maintainthe details of books and library members.

● The main purpose of this project is to maintain easy circulation systemusing computers and to provide different reports.

1.2 Scope:

The document only covers the requirements specifications for theLibrary Management System. This document does not provide any references tothe other component of the Library Management System. All the externalinterfaces and the dependencies are also identified in this document.

Feasibility study :

The overall scope of the feasibility study was to providesufficient information to allow a decision to be made as to whether the LibraryManagement System project should proceed and if so, its relative priority in thecontext of other existing Library Management Technology.

The feasibility study phase of this project had undergone through various stepswhich as describe as under:

● Identity the origin the information at different level.

● Identity the expectation of user from computerized system.

● Analyze the drawback of existing system(manual system)

1.3 Definition, Acronyms, Abbreviation:

● JAVA ->Java is a general-purpose computer programming language that is concurrent, class based; object oriented, and specifically designed to have as few implementation dependencies as possible.

● SQL -> Structured query Language

● DFD -> Data Flow Diagram

● ER -> Entity Relationship Diagram

● IDE -> Integrated Development Environment

● SRS -> Software Requirement Specification

1.4 Reference:

Websites:-

- www.google.com
- www.youtube.com

Reference books:-

- SQL: THE COMPLETE REFERENCE BY GROFF AND JAMES, MCGRAW HILL
- PROGRAMMING WITH JAVA - E BALGURUSAMY

1.5 Overview:
The implementation of Library Management starts with entering andupdating master records like book details, library information. Any furthertransaction like book issue, book return will automatically update the currentbooks.

2. Overall Description:

2.1 Product Perspective:
The proposed Library Management System will take care of the currentbook detail at any point of time. The book issue, book return will update thecurrent book details automatically so that user will get the update current bookdetails.

2.2 Product function:
● The main purpose of this project is to reduce the manual work.
● This software is capable of managing Book Issues, Returns, andCalculating/Managing Fine. Generating various Reports forRecord-Keeping according to end user requirements

2.3 User characteristics:
We have 2 levels of users

- User module: In the user module, user will check the availabilityof the books.
- Book return
- Administration module: The following are the sub module in theadministration module.
- Register user
- Entry book details
- Book issue

2.4 General Constraints:
Any update regarding the book from the library is to be recorded tohave update & correct values.

2.5 Assumption and dependencies:
All the data entered will be correct and up to date. This softwarepackage is developed using java as front end which is supported by sun micro system. Microsoft SQL server 2005 as the back end.

3. Specific Requirement:

3.1 External Interface Requirement:
The user should be simple and easy to understand and use. Also be aninteractive interface .The system should prompt for the user and administrator tologin to the application and for proper input criteria.

3.1.1 User Interface:
The software provides good graphical interface for the user anyadministrator can operate on the system, performing the required task such ascreate, update, viewing the details of the book.
● Allows user to view quick reports like Book Issues/Returned etc inbetween particular time.

● Stock verification and search facility based on different criteria.

3.1.2 Hardware interface:

● **Operating system:**Windows 7/ Linux
● **Hard disk:**40 GB
●**RAM:** 256 MB
● **Processor:** Pentium(R) Dual-core CPU

3.1.3 Software interface:

● Java language
● Net beans IDE 7.0.1
● MS SQL server 2005

3.1.4 Communication interface:

Linux, Java, MS-SQL Server

3.2 Functional requirements:

■ Book entry: In this module we can store the details of thebooks.
■ Register student: in this module we can keep the details of thenew student.
■ Book issue: This module is used to keep a track of book issuedetails.
■ Book return: This module enables to keep a track of return thebooks.

3.3 Performance requirements:

The capability of the computer depends on the performance of thesoftware. The software can take any number of inputs provided the database sizeis larger enough. This would depend on the available memory space.

1. Design constraints:

Each member will be having a identity card which can be used for the librarybook issue, fine payment etc. whenever library member wish to take a book, thebook issued by the library authority will be check both the book details as well asthe student details and store it in library database. In case of retrieval of bookmuch of human intervention can be eliminated.

2. System attributes:

●**Maintainability**: There will be no maintained requirement for thesoftware. The database is provided by the end user and therefore ismaintained by this user.
●**Portability**: The system is developed for secured purpose, so it is can'tbe portable.
●**Availability**: This system will available only until the system on whichit is install, is running.
● **Scalability**: Applicable.

Q.1] Solve the Following Assignment.

SET A]

1. What is functional and non-function requirements?
2. What are the user Interface requirements?
3. What is the need for SRS documents?

SET B]

1. Write SRS for Restaurant Management System**(Using Linux operating system)**
2. Write SRS for Online Shopping System?

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 9:-Design Data Flow Diagrams For the Problem

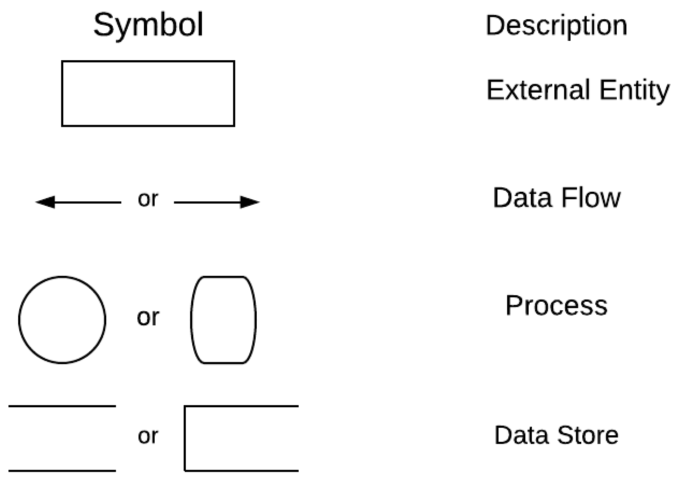
❖ What is a data flow diagram?

A data flow diagram shows the way information flows through a process or system. It includes data inputs and outputs, data stores, and the various subprocesses the data moves through. DFDs are built using standardized symbols and notation to describe various entities and their relationships.

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.It shows how data enters and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system

❖ Symbols used in Data Flow diagram

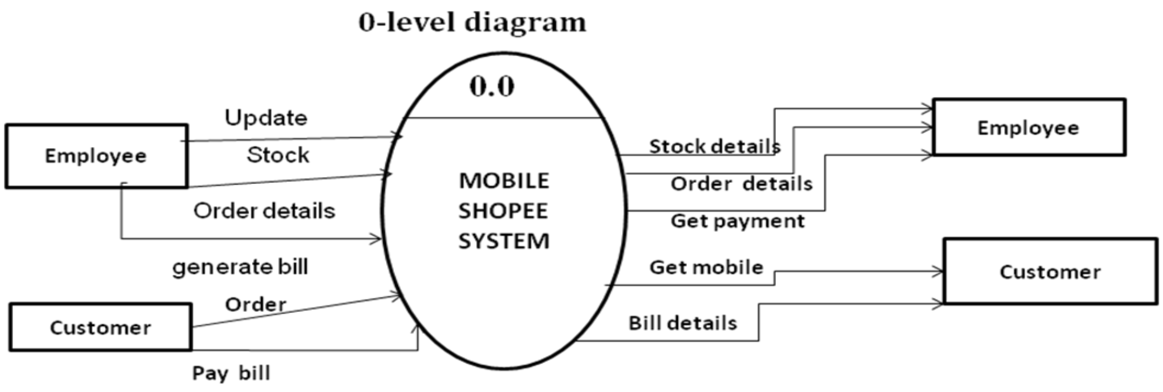


❖ Levels in Data Flow Diagrams (DFD)

- **0-level DFD(Context level Diagram)**
- **1-level DFD**
- **2-level DFD**

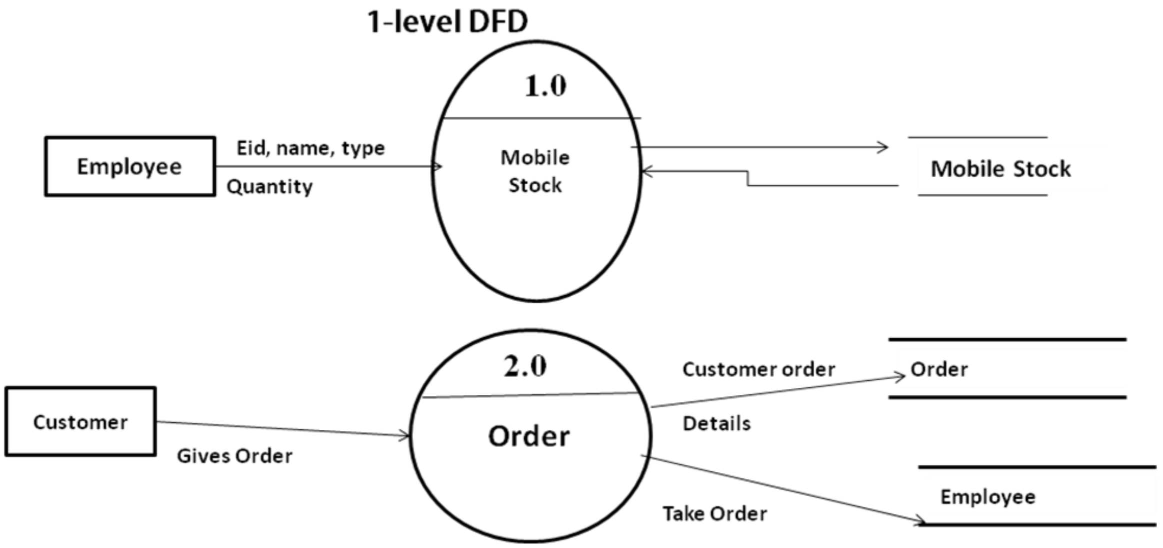
0-level DFD:

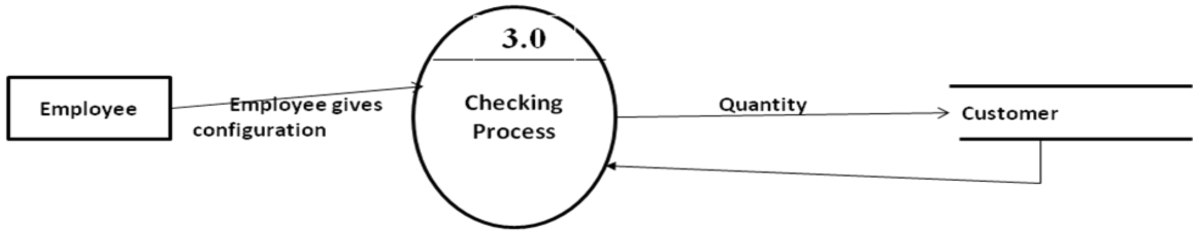
It is also known as context diagram. It’s designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as single bubble with input and output data indicated by incoming/outgoing arrows.

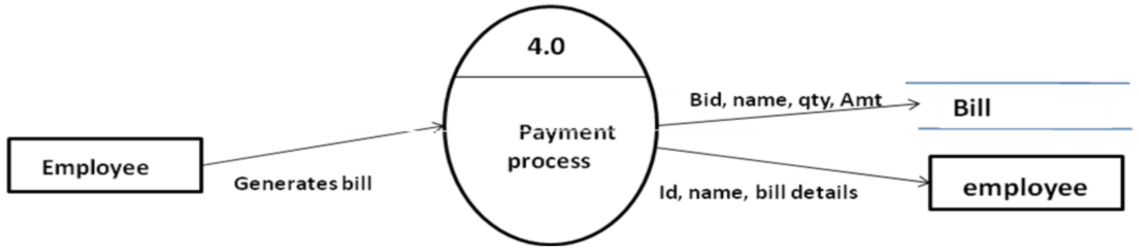


1-level DFD:

In 1-level DFD, context diagram is decomposed into multiple bubbles/processes. in this level we highlight the main functions of the system and breakdown the high level process of 0-level DFD into sub processes





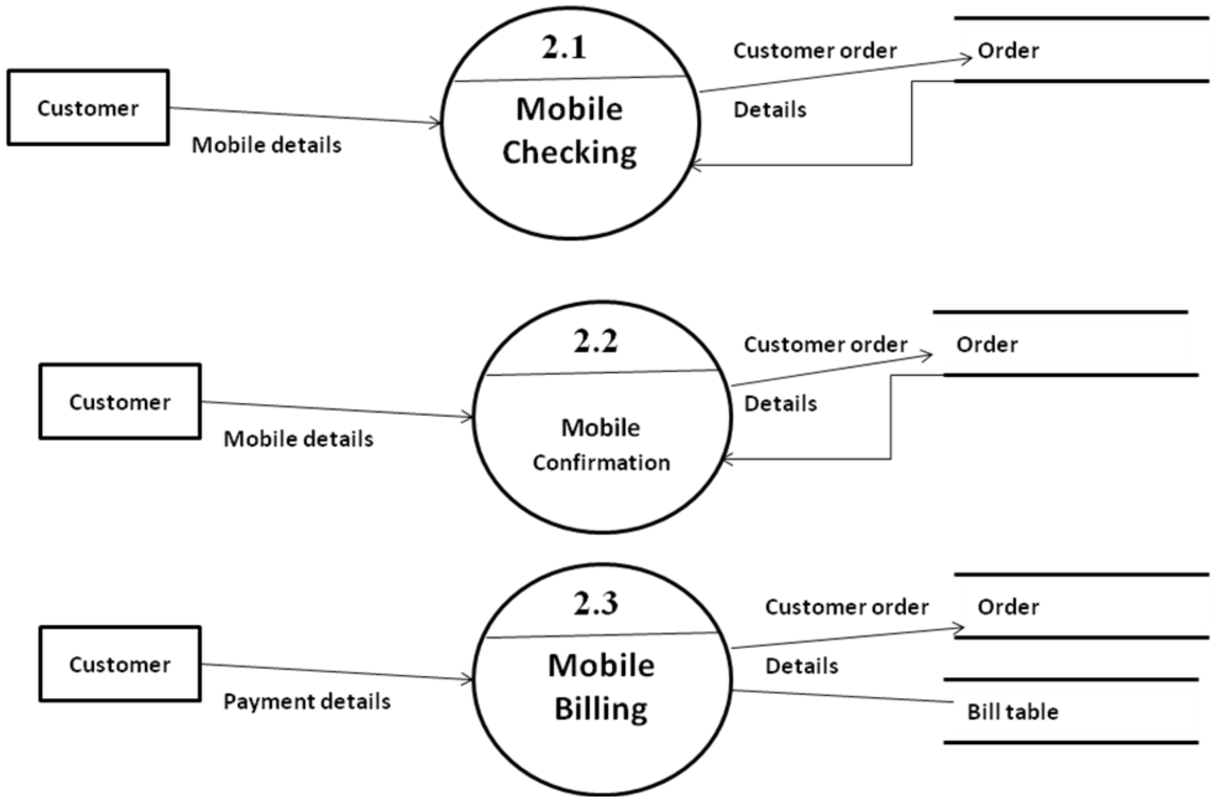


.

2-level DFD:

2-level DFD goes one step deeper into parts of 1-level DFD.It can be used to plan or record the specific/necessary detail about the system's functioning.

2-level DFD of 2nd process



Q.1] Solve the Following Assignment.

1. Consider the Savings Bank Deposit and Withdrawal System in a Nationalized Bank. Also involve calculation of Interest.
   - i. **Identify all entities.**
   - ii. **Draw context level diagram**
   - iii. **First level DFD for the system**

2. Consider a Hospital Management System in which the Hospital has InPatient Department (IPD), Outpatient Department (OPD) the system maintains patient records and bills of patient it also manages, information of various wards in the hospital like ICU, General, Private, Semi-private and Deluxe.
   - i. **Identify all entities.**
   - ii. **Draw context level diagram**
   - iii. **First level DFD for the system**

Q2.] Instructor/ Teacher can give more case studies to the students.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor