Engineering Notebook

VOLUME 1

23VLS1306 Basics of Python Programming

CONTRIBUTOR

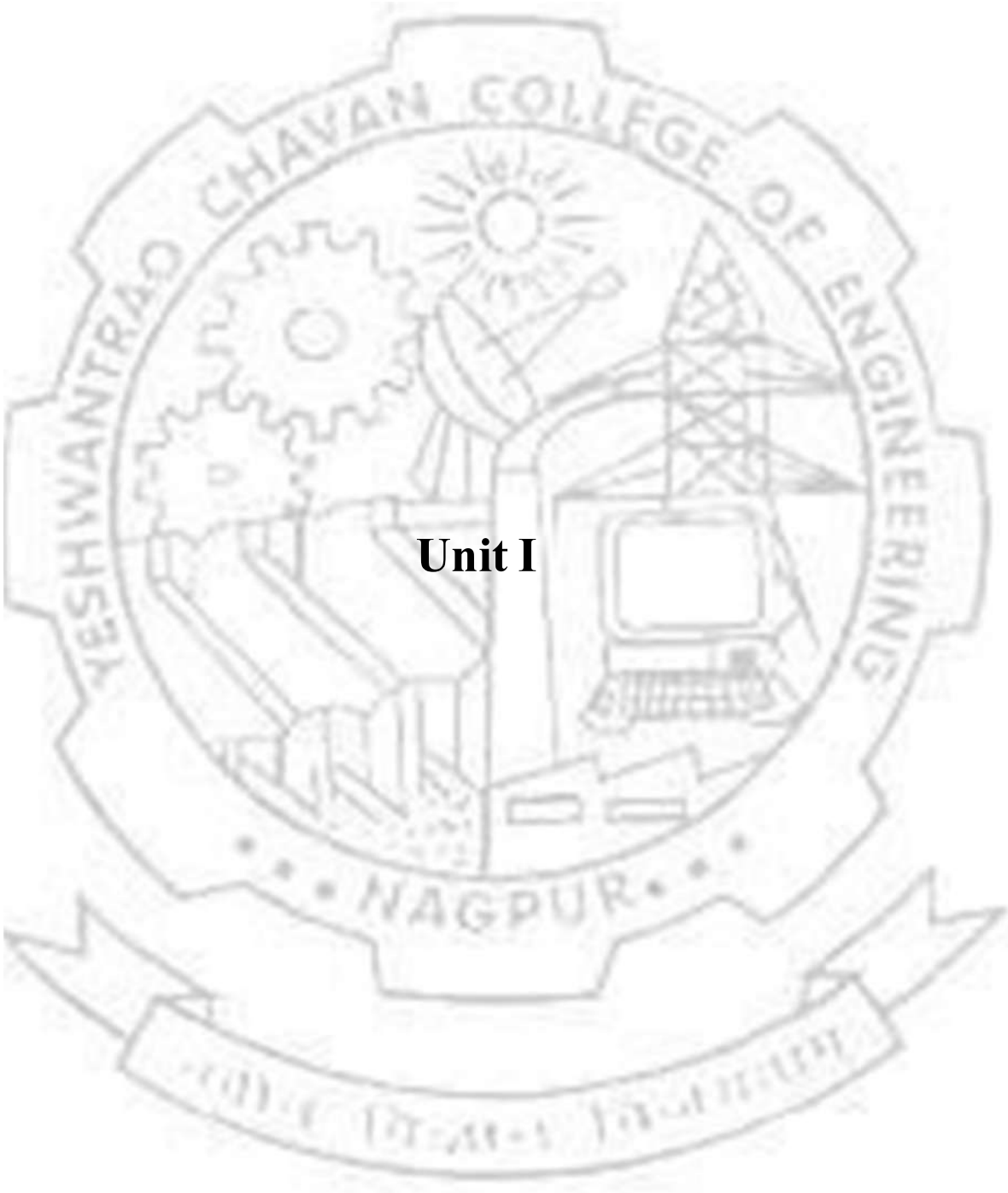
Mrs. Swati R. Nitnaware

DEPARTMENT OF ELECTRONICS ENGINEERING

YESHWANTRAO CHAVAN COLLEGE OFENGINEERING,

(An autonomous institution affiliated to Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur)

NAGPUR– 441110



Unit I

Q 1 **What are the key features of Python?**

Ans: Python is an **interpreted** language. That means that, unlike languages like *C* and its variants, Python does not need to be compiled before it is run. Other interpreted languages include *PHP* and *Ruby*.

Python is **dynamically typed**, this means that you don't need to state the types of variables when you declare them or anything like that. You can do things like x=111 and then x="I'm a string" without error

Python is well suited to **object orientated programming** in that it allows the definition of classes along with composition and inheritance. Python does not have access specifiers (like C++'s public, private).

In Python, **functions** are **first-class objects**. This means that they can be assigned to variables, returned from other functions and passed into functions. Classes are also first class objects

Writing Python code is quick but running it is often slower than compiled languages. FortunatelyPython allows the inclusion of C based extensions so bottlenecks can be optimized away and often is. The numpy package is a good example of this, it's really quite quick because a lot of the number crunching it does isn't actually done by Python. Python finds **use in many spheres** – web applications, automation, scientific modeling, big data applications and many more. It's also often used as "glue" code to get other languages and components to play nice.

Q 2 What are local variables and global variables in Python?

Global Variables:

Ans: Variables declared outside a function or in global space are called global variables. These variables can be accessed by any function in the program.

Local Variables:

Any variable declared inside a function is known as a local variable. This variable is present in the local space and not in the global space.

Example:

```
a=2
defadd():
b=3
c=a+b
```

```
print(c)
add()
```

Output: 5

When you try to access the local variable outside the function add(), it will throw an error.

Q 3 What is type conversion in Python?

Ans: Type conversion refers to the conversion of one data type iinto another.

int() – converts any data type into integer type

float() – converts any data type into float type

ord() – converts characters into integer

hex() – converts integers to hexadecimal

oct() – converts integer to octal

tuple() – This function is used to convert to a tuple.

set() – This function returns the type after converting to set.

list() – This function is used to convert any data type to a list type.

dict() – This function is used to convert a tuple of order (key,value) into a dictionary.

str() – Used to convert integer into a string.

complex(real,imag) – This functionconverts real numbers to complex(real,imag) number.

Q 4 What is the usage of help() and dir() function in Python?

Ans: Help() and dir() both functions are accessible from the Python interpreter and used for viewing a consolidated dump of built-in functions.

1. Help() function: The help() function is used to display the documentation string and also facilitates you to see the help related to modules, keywords, attributes, etc.
2. Dir() function: The dir() function is used to display the defined symbols.

Q 5 Python Program to Calculate the Average of Numbers in a Given List

Ans:
```
n=int(input("Enter the number of elements to be inserted: "))
a=[]for i in range(0,n):
elem=int(input("Enter element: "))
a.append(elem)
avg=sum(a)/nprint("Average of elements in the list",round(avg,2))
```

Q 6 Python Program to Exchange the Values of Two Numbers Without Using a Temporary Variable.

Ans:
```
a=int(input("Enter value of first variable: "))
b=int(input("Enter value of second variable: "))
a=a+b
b=a-b
a=a-b
print("a is:",a," b is:",b)
```

Q 7 **Python Program to Take in the Marks of 5 Subjects and Display the Grade**

Ans:
```
sub1=int(input("Enter marks of the first subject: "))
sub2=int(input("Enter marks of the second subject: "))
sub3=int(input("Enter marks of the third subject: "))
sub4=int(input("Enter marks of the fourth subject: "))
sub5=int(input("Enter marks of the fifth subject: "))
avg=(sub1+sub2+sub3+sub4+sub4)/5if(avg>=90):
print("Grade: A")elif(avg>=80&avg<90):
print("Grade: B")elif(avg>=70&avg<80):
print("Grade: C")elif(avg>=60&avg<70):
print("Grade: D")
```

Q 8 Python Program to Check if a Number is a Palindrome

Ans:
```
n=int(input("Enter number:"))
temp=n
rev=0while(n>0):
dig=n%10
rev=rev*10+dig
n=n//10if(temp==rev):
print("The number is a palindrome!")else:
print("The number isn't a palindrome!")
```

Q 9 What is the difference between range & xrange?

Ans: For the most part, xrange and range are the exact same in terms of functionality. They both provide a way to generate a list of integers for you to use, however you please. The only difference is that range returns a Python list object and x range returns an xrange object.

This means that xrange doesn't actually generate a static list at run-time like range does. It creates the values as you need them with a special technique called yielding. This technique is used with a type of object known as generators. That means that if you have a really gigantic range you'd like to generate a list for, say one billion, xrange is the function to use.

This is especially true if you have a really memory sensitive system such as a cell phone that you are working with, as range will use as much memory as it can to create your array of integers, which can result in a Memory Error and crash your program.

Q 10 How can you generate random numbers in Python?

Ans: Random module is the standard module that is used to generate a random number. The method is defined as:

```
import random
random.random
```

The statement random.random() method return the floating point number that is in the range of [0, 1). The function generates random float numbers. The methods that are used with the random class are the bound methods of the hidden instances. The instances of the Random can be done to show the multi-threading programs that creates a different instance of individual threads. The other random generators that are used in this are:

1. randrange(a, b): it chooses an integer and define the range in-between [a, b). It returns the elements by selecting it randomly from the range that is specified. It doesn't build a range object.
2. uniform(a, b): it chooses a floating point number that is defined in the range of [a,b).Iyt returns the floating point number
3. normalvariate(mean, sdev): it is used for the normal distribution where the mu is a mean and the sdev is a sigma that is used for standard deviation.
4. The Random class that is used and instantiated creates an independent multiple random number generators.

Q 11 What is the process of compilation and linking in python?

Ans: The compiling and linking allows the new extensions to be compiled properly without any error and the linking can be done only when it passes the compiled procedure. If the dynamic loading is used then it depends on the style that is being provided with the system. The python interpreter can be used to provide the dynamic loading of the configuration setup files and will rebuild the interpreter.

The steps that are required in this as:

1. Create a file with any name and in any language that is supported by the compiler of your system. For example file.c or file.cpp
2. Place this file in the Modules/ directory of the distribution which is getting used.
3. Add a line in the file Setup.local that is present in the Modules/ directory.
4. Run the file using spam file.o
5. After a successful run of this rebuild the interpreter by using the make command on the top-level directory.
6. If the file is changed then run rebuildMakefile by using the command as 'make Makefile'.

Q 12 Explain Inheritance in Python with an example.

Ans: Inheritance allows One class to gain all the members(say attributes and methods) of another class. Inheritance provides code reusability, makes it easier to create and maintain an application. The class from which we are inheriting is called super-class and the class that is inherited is called a derived / child class.

They are different types of inheritance supported by Python:

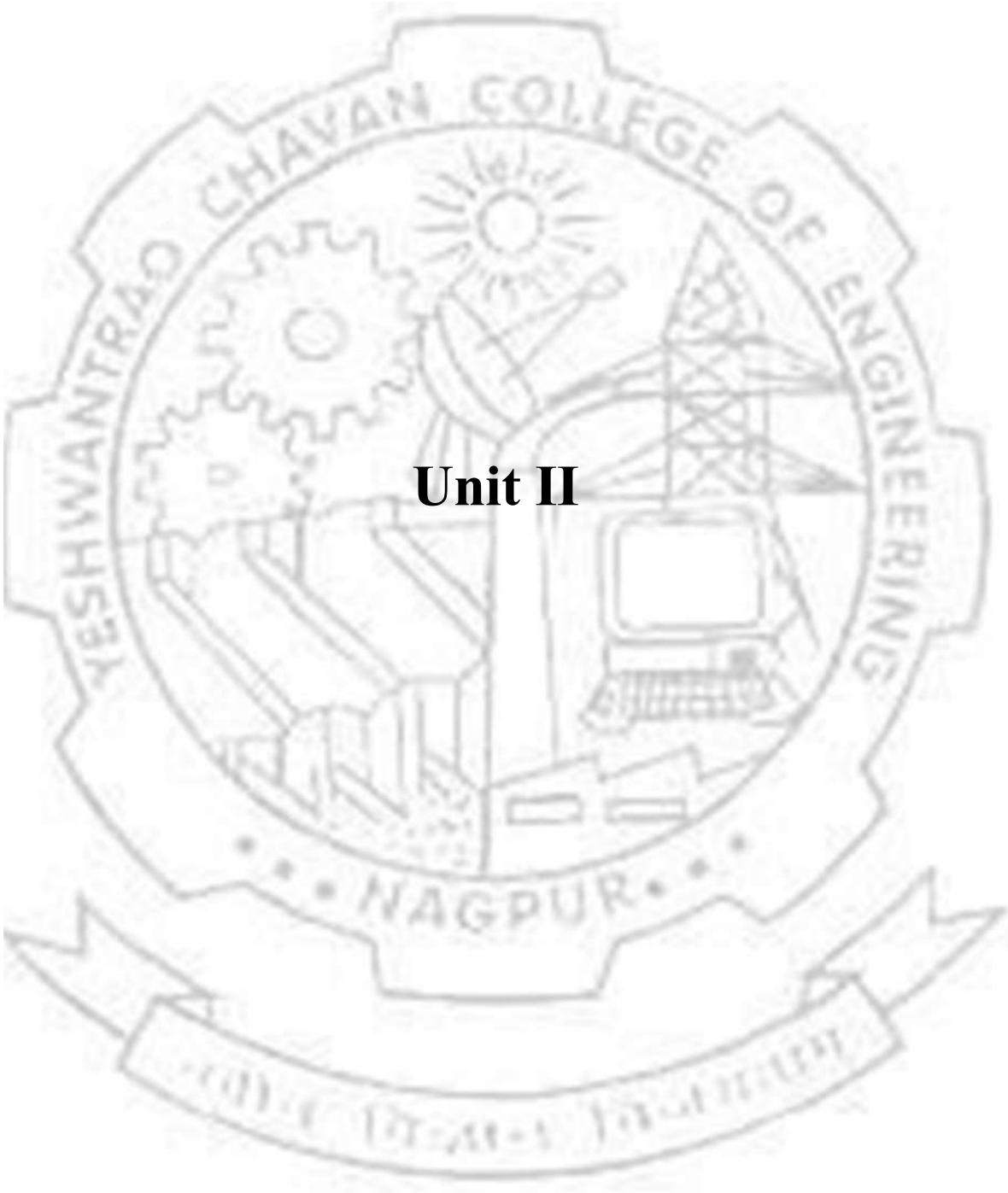
1. Single Inheritance – where a derived class acquires the members of a single super class.
2. Multi-level inheritance – a derived class d1 in inherited from base class base1, and d2 are inherited from base2.
3. Hierarchical inheritance – from one base class you can inherit any number of child classes
4. Multiple inheritance – a derived class is inherited from more than one base class.
5.

Q 13 Python Program to Check if a Date is Valid and Print the Incremented Date

Ans:

```
date=input("Enter the date: ")
dd,mm,yy=date.split('/')
dd=int(dd)
mm=int(mm)
yy=int(yy)
```

```
if(mm==1 or mm==3 or mm==5 or mm==7 or mm==8 or mm==10 or mm==12):
   max1=31
elif(mm==4 or mm==6 or mm==9 or mm==11):
   max1=30
elif(yy%4==0 and yy%100!=0 or yy%400==0):
  max1=29
else:
   max1=28
if(mm<1 or mm>12):
print("Date is invalid.")
elif(dd<1 or dd>max1):
print("Date is invalid.")
elif(dd==max1 and mm!=12):
  dd=1
   mm=mm+1
print("The incremented date is: ",dd,mm,yy)
elif(dd==31 and mm==12):
   dd=1
   mm=1
   yy=yy+1
print("The incremented date is: ",dd,mm,yy)
else:
   dd=dd+1
print("The incremented date is: ",dd,mm,yy)
```



Unit II

Q 1 What are the different types of Python operators?

Ans:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	**Name**	**Example**
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y

**	Exponentiation	x ** y
//	Floor division	x // y

Q 2 Explain the Membership operators

Ans: Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Q 3 Python Program to Check If Two Numbers are Amicable Numbers

Ans:

```
x=int(input('Enter number 1: '))
y=int(input('Enter number 2: '))
sum1=0
sum2=0
for i in range(1,x):
if x%i==0:
    sum1+=i
for j in range(1,y):
```

```
if y%j==0:
      sum2+=j
if(sum1==y and sum2==x):
print('Amicable!')
else:
print('Not Amicable!')
```

Program Explanation

1. User must enter both the numbers and store it in separate variables.
2. Using a for loop and an if statement, find the proper divisors of both the numbers.
3. Find the sum of the proper divisors of both the numbers.
4. The if statement is used to check if the sum of proper divisors of the number is equal to the other number and vice-versa.
5. If it is equal, both the numbers are amicable numbers.
6. The final result is printed.

Runtime Test Cases

```
Case 1:
Enter number 1: 220
Enter number 2: 284
Amicable!

Case 2:
Enter number 1: 349
Enter number 2: 234
Not Amicable!
```

Q 4 **Python Program to Find the Gravitational Force Acting Between Two Objects**

Ans:

```
m1=float(input("Enter the first mass: "))
m2=float(input("Enter the second mass: "))
r=float(input("Enter the distance between the centres of the masses: "))
G=6.673*(10**-11)
f=(G*m1*m2)/(r**2)
print("Hence, the gravitational force is: ",round(f,2),"N")
```

Program Explanation

1. User must enter the values for both the masses and the distance between the masses and store it in separate variables.

2. One of the variables is initialised to the value of gravitational constant (G) which is equal to 6.673*(10**-11).
3. Then the formula: f=(G*m1*m2)/(r**2), where m1 and m2 are the masses and r is the distance between them, is used to determine the magnitude of force acting between the masses.
4. The force calculated is rounded up-to 2 decimal places and printed.
Runtime Test Cases

Case 1:
Enter the first mass: 1000000
Enter the second mass: 500000
Enter the distance between the centres of the masses: 20
Hence, the gravitational force is: 0.08 N

Case 2:
Enter the first mass: 90000000
Enter the second mass: 7000000
Enter the distance between the centres of the masses: 20
Hence, the gravitational force is: 105.1 N

Q 5 Python Program to Find the Sum of the Series: 1 + x^2/2 + x^3/3 + … x^n/n

Ans:

```
n=int(input("Enter the number of terms:"))
x=int(input("Enter the value of x:"))
sum1=1
for i in range(2,n+1):
   sum1=sum1+((x**i)/i)
print("The sum of series is",round(sum1,2))
```

Program Explanation
1. User must enter the number of terms to find the sum of.
2. The sum variable is initialized to 0.
3. The for loop is used to find the sum of the series and the number is incremented for each iteration.
4. The numbers are added to the sum variable and this continues till the value of i reaches the number of terms.
5. Then the sum of the series is printed.
Runtime Test Cases

Case 1:
Enter the number of terms:3
Enter the value of x:1
The sum of series is 1.83

```
Case 2:
Enter the number of terms:5
Enter the value of x:2
The sum of series is 16.07
```

Q 6 Python Program to Compute the Value of Euler's Number e. Use the Formula: e = 1 + 1/1! + 1/2! + …… 1/n!

Ans:

1. Take in the number of terms to find the sum of the series for.
2. Initialize the sum variable to 1.
3. Use a for loop ranging from 1 to the number and find the sum of the series.
4. Print the sum of the series after rounding it off to two decimal places.
5. Exit.

Program/Source Code

Here is source code of the Python Program to find the sum of series: 1 + 1/2 + 1/3 +..... + 1/N. The program output is also shown below.

```
import math
n=int(input("Enter the number of terms: "))
sum1=1
for i in range(1,n+1):
    sum1=sum1+(1/math.factorial(i))
print("The sum of series is",round(sum1,2))
```

Program Explanation

1. User must enter the number of terms to find the sum of.
2. The sum variable is initialized to 1.
3. The for loop is used to find the sum of the series and the number is incremented for each iteration.
4. The numbers are added to the sum variable and this continues till the value of i reaches the number of terms.
5. Then the sum of the series is printed.

Runtime Test Cases

```
Case 1:
Enter the number of terms: 5
The sum of series is 2.72

Case 2:
Enter the number of terms: 20
```

The sum of series is 2.72

Q 7. Python Program to Determine all Pythagorean Triplets in the Range

Ans:

```
limit=int(input("Enter upper limit:"))

c=0
m=2
while(c<limit):
for n in range(1,m+1):
        a=m*m-n*n
        b=2*m*n
        c=m*m+n*n
if(c>limit):
break
if(a==0 or b==0 or c==0):
break
print(a,b,c)
   m=m+1
```

Program Explanation

1. User must enter the upper limit and store it in a variable.
2. The while and for loop is used to find the value of the Pythagorean triplets using the formula.
3. If the value of a side is greater than the upper limit or if any of the side is 0, the loop is broken out of.
4. Then the triplets are printed.

Runtime Test Cases

```
Case 1:
Enter upper limit:20
3 4 5
8 6 10
5 12 13
15 8 17
12 16 20

Case 2:
Enter upper limit:35
3 4 5
8 6 10
5 12 13
15 8 17
12 16 20
7 24 25
```

```
24 10 26
21 20 29
16 30 34
```

Q 8 Python Program to Search the Number of Times a Particular Number Occurs in a List

Ans:

```
a=[]

n=int(input("Enter number of elements:"))
for i in range(1,n+1):
    b=int(input("Enter element:"))
    a.append(b)
k=0
num=int(input("Enter the number to be counted:"))
for j in a:
 if(j==num):
        k=k+1
print("Number of times",num,"appears is",k)
```

Program Explanation

1. User must enter the number of elements for the first list and store it in a variable.
2. User must then enter the elements of the list one by one using a for loop and store it in a list.
3. User must also enter the number to be search the list for.
4. A for loop is used to traverse through the elements in the list and an if statement is used to count a particular number.
5. The total count of a particular number in the list is printed.

Runtime Test Cases

```
Case 1:
Enter number of elements:4
Enter element:23
Enter element:45
Enter element:23
Enter element:67
Enter the number to be counted:23
Number of times 23 appears is 2

Case 2:
```

```
Enter number of elements:7
Enter element:12
Enter element:45
Enter element:67
Enter element:45
Enter element:67
Enter element:67
Enter element:67
Enter the number to be counted:67
Number of times 67 appears is 4
```

Q 9 Python Program to test Collatz Conjecture for a Given Number

Ans:

Problem Solution

1. Create a function collatz that takes an integer n as argument.
2. Create a loop that runs as long as n is greater than 1.
3. In each iteration of the loop, update the value of n.
4. If n is even, set n to n/2 and if n is odd, set it to 3n + 1.
5. Print the value of n in each iteration.

Program/Source Code

Here is the source code of a Python program to test Collatz conjecture for a given number. The program output is shown below.

```
def collatz(n):
while n > 1:
print(n, end=' ')
if (n % 2):
# n is odd
        n = 3*n + 1
else:
# n is even
        n = n//2
print(1, end='')



n = int(input('Enter n: '))
print('Sequence: ', end='')
collatz(n)
```

Program Explanation

1. The user is asked to input n.
2. The sequence is printed by calling collatz on n.

Runtime Test Cases

```
Case 1:
Enter n: 11
Sequence: 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

Case 2:
Enter n: 5
Sequence: 5 16 8 4 2 1

Case 3:
Enter n: 1
Sequence: 1
```

Q 10 **Python Program to Generate Gray Codes using Recursion**

Ans:

Problem Solution

1. The function get_gray_codes is defined.
2. It takes the number of bits n as argument.
3. It returns n-bit Gray code in a list.
4. The function works by first obtaining the (n – 1)-bit Gray code.
5. The first half of the n-bit Gray codewords are simply the (n – 1)-bit Gray codewords prepended by a 0.
6. The second half of the n-bit Gray codewords are (n – 1)-bit Gray codewords listed in reverse and prepended with a 1.
7. The 0-bit Gray code simply consists of the empty string.

Program/Source Code

Here is the source code of a Python program to generate all gray codes using recursion. The program output is shown below.

```
def get_gray_codes(n):
"""Return n-bit Gray code in a list."""
if n == 0:
return ['']
    first_half = get_gray_codes(n - 1)
    second_half = first_half.copy()

    first_half = ['0' + code for code in first_half]
    second_half = ['1' + code for code in reversed(second_half)]
```

```
return first_half + second_half


n = int(input('Enter the number of bits: '))
codes = get_gray_codes(n)
print('All {}-bit Gray Codes:'.format(n))
print(codes)
```

Program Explanation

1. The user is prompted to enter the number of bits.
2. get_gray_codes is called on the number of bits.
3. The returned list is displayed.

Runtime Test Cases

```
Case 1:
Enter the number of bits: 3
All 3-bit Gray Codes:
['000', '001', '011', '010', '110', '111', '101', '100']

Case 2:
Enter the number of bits: 1
All 1-bit Gray Codes:
['0', '1']

Case 3:
Enter the number of bits: 4
All 4-bit Gray Codes:
['0000', '0001', '0011', '0010', '0110', '0111', '0101', '0100', '1100', '1101', '1111', '1110', '1010', '1011',
'1001', '1000']
```

Q 11 Python Program to Convert Gray Code to Binary

Ans:

Problem Solution

1. The function gray_to_binary is defined.
2. It takes the Gray codeword string as argument.
3. It returns its associated binary number as a string.
4. If g(i) is the ith bit in the Gray codeword and b(i) is the ith bit in its associated binary number where

the 0th bit is the MSB, it can be shown g(0) = b(0) and b(i) = g(i) XOR b(i – 1) for i > 0.
5. From the above, it follows that b(i) = g(i) XOR g(i – 1) XOR … XOR g(0).
6. Thus a Gray codeword g can be converted to its associated binary number by performing (g XOR (g >> 1) XOR (g >> 2) XOR … XOR (g >> m)) where m is such that g >> (m + 1) equals 0.

Program/Source Code

Here is the source code of a Python program to convert Gray code to binary. The program output is shown below.

```
def gray_to_binary(n):
"""Convert Gray codeword to binary and return it."""
    n = int(n, 2) # convert to int

    mask = n
while mask != 0:
        mask >>= 1
        n ^= mask

# bin(n) returns n's binary representation with a '0b' prefixed
# the slice operation is to remove the prefix
return bin(n)[2:]


g = input('Enter Gray codeword: ')
b = gray_to_binary(g)
print('In binary:', b)
```

Program Explanation

1. The user is prompted to enter the Gray codeword.
2. gray_to_binary is called on the Gray codeword.
3. The returned string which is its associated binary number is displayed.

Runtime Test Cases

```
Case 1:
Enter Gray codeword: 111
In binary: 101

Case 2:
Enter Gray codeword: 1101
```

```
In binary: 1001

Case 3:
Enter Gray codeword: 10
In binary: 11
```

Unit III

Q 1 Explain different comparison operators in Python.

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Q 2 Explain conditional execution in detail with example.

In order to write useful programs, we almost always need the ability to check conditions and change the behavior of the program accordingly. Conditional statements give us this ability. The simplest form is the if statement:

```
if x > 0 :
print('x is positive')
```

The boolean expression after the if statement is called the condition. We end the if statement with a colon character (:) and the line(s) after the if statement are indented.

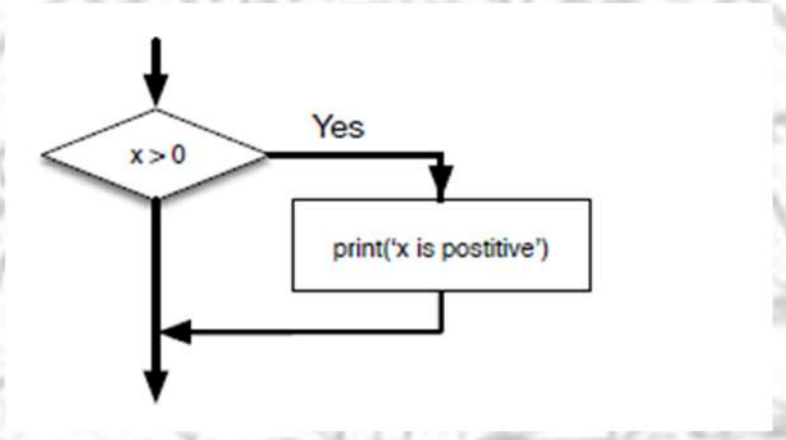


Fig No.1

If the logical condition is true, then the indented statement gets executed. If the logical condition is false, the indented statement is skipped. if statements have the same structure as function definitions or for loops1. The statement consists of a header line that ends with the colon character (:) followed by an indented block. Statements like this are called compound statements because they stretch across more than one line. There is no limit on the number of statements that can appear in the body, but there must be at least one. Occasionally, it is useful to have a body with no statements (usually as a place holder for code you haven't written yet). In that case, you can use the pass statement, which does nothing.

```
if x < 0 :
pass            # need to handle negative values!
```

If you enter an if statement in the Python interpreter, the prompt will change from three chevrons to three dots to indicate you are in the middle of a block of statements, as shown below:

```
>>> x = 3
>>> if x < 10:
... print('Small')
...
Small
>>>
```

When using the Python interpreter, you must leave a blank line at the end of a block, otherwise Python will return an error:

```
>>> x = 3
>>> if x < 10:
... print('Small')
... print('Done')
File "<stdin>", line 3
print('Done')
^
SyntaxError: invalid syntax
```

A blank line at the end of a block of statements is not necessary when writing and executing a script, but it may improve readability of your code.

A second form of the if statement is alternative execution, in which there are two possibilities and the condition determines which one gets executed. The syntax looks like this:

```
if x%2 == 0 :
print('x is even')
else :
print('x is odd')
```

If the remainder when x is divided by 2 is 0, then we know that x is even, and the program displays a message to that effect. If the condition is false, the second set of statements is executed.

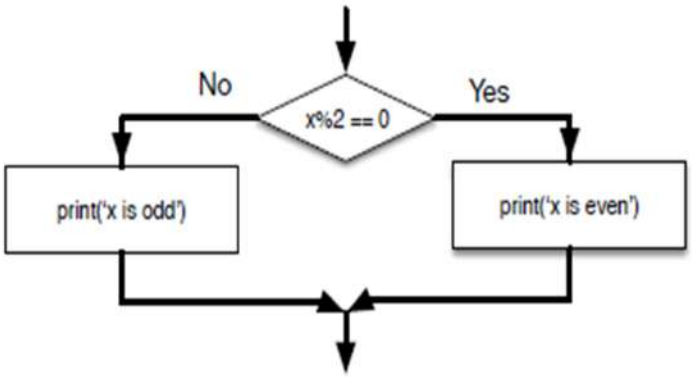


Fig No.2

Since the condition must either be true or false, exactly one of the alternatives will be executed. The alternatives are called branches, because they are branches in the flow of execution.

Q 3 What are chained conditionals?
Ans:

Sometimes there are more than two possibilities and we need more than two branches. One way to express a computation like that is a chained conditional:

```
if x < y:
print('x is less than y')
elif x > y:
print('x is greater than y')
else:
print('x and y are equal')
```

elif is an abbreviation of "else if." Again, exactly one branch will be executed. There is no limit on the number of elif statements. If there is an else clause, it has to be at the end, but there doesn't have to be one.

```
if choice == 'a':
print('Bad guess')
elif choice == 'b':
print('Good guess')
elif choice == 'c':
print('Close, but not correct')
```

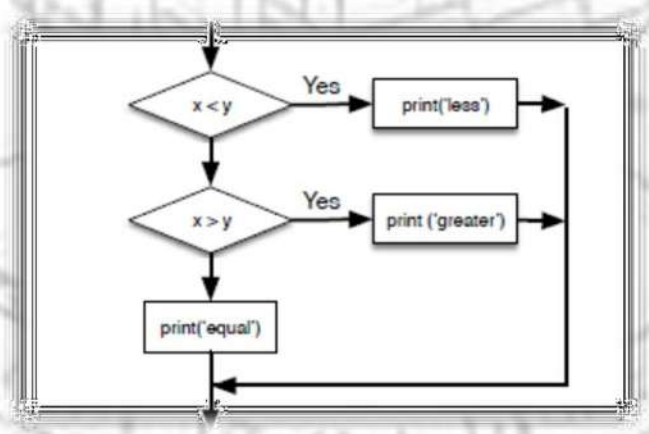


Fig No.3

Each condition is checked in order. If the first is false, the next is checked, and so on. If one of them is true, the corresponding branch executes, and the statement ends. Even if more than one condition is true, only the first true branch executes.

Q 4 What are nested conditionals?

Ans: One conditional can also be nested within another. We could have written the three-branch example like this:

```
if x == y:
print('x and y are equal')
else:
if x < y:
print('x is less than y')
else:
print('x is greater than y')
```

The outer conditional contains two branches. The first branch contains a simple statement. The second branch contains another if statement, which has two branches of its own. Those two branches are both simple statements, although they could have been conditional statements as well.

Although the indentation of the statements makes the structure apparent, nested conditionals become difficult to read very quickly. In general, it is a good idea to avoid them when you can. Logical operators often provide a way to simplify nested conditional statements.

For example, we can rewrite the following code using a single conditional:

```
if 0 < x:
if x < 10:
print('x is a positive single-digit number.')
```

The print statement is executed only if we make it past both conditionals, so we can get the same effect with the and operator:

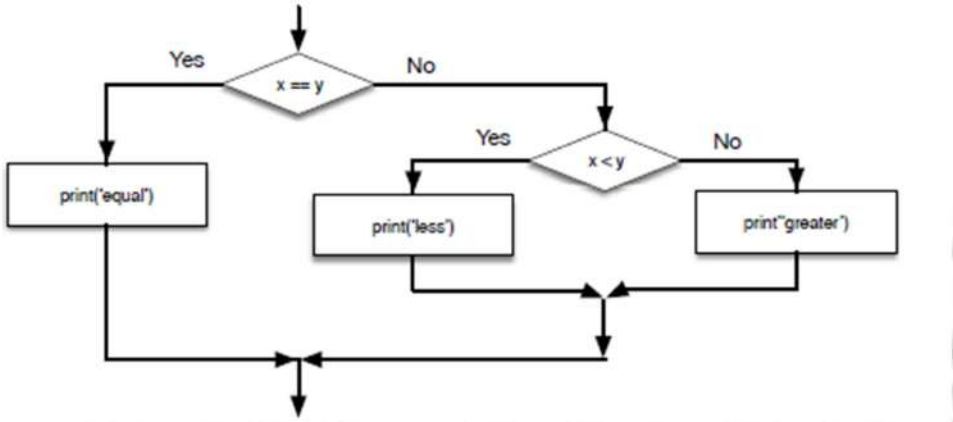


Fig No.4

if 0 < x and x < 10:
print('x is a positive single-digit number.')

Q 5 Explain the concept of try and except in Python

Ans:

>>> prompt = "What...is the airspeed velocity of an unladen swallow?\n"
>>> speed = input(prompt)
What...is the airspeed velocity of an unladen swallow?
What do you mean, an African or a European swallow?
>>> int(speed)
ValueError: invalid literal for int() with base 10:
>>>

When we are executing these statements in the Python interpreter, we get a new prompt from the interpreter, think "oops", and move on to our next statement. However if you place this code in a Python script and this error occurs, your script immediately stops in its tracks with a traceback. It does not execute the following statement.

Here is a sample program to convert a Fahrenheit temperature to a Celsius temperature:

np = input('Enter Fahrenheit Temperature: ')
fahr = float(inp)
cel = (fahr - 32.0) * 5.0 / 9.0
print(cel)

If we execute this code and give it invalid input, it simply fails with an unfriendly error message:
python fahren.py

Enter Fahrenheit Temperature:72

22.22222222222222

Enter Fahrenheit Temperature:fred
Traceback (most recent call last):
File "fahren.py", line 2, in <module>
fahr = float(inp)
ValueError: could not convert string to float: 'fred'

There is a conditional execution structure built into Python to handle these types of expected and unexpected errors called "try / except". The idea of try and except is that you know that some sequence of instruction(s) may have a problem and you want to add some statements to be executed if an error occurs. These extra statements (the except block) are ignored if there is no error. You can think of the try and except feature in Python as an "insurance policy" on a sequence of statements.
We can rewrite our temperature converter as follows:

```
inp = input('Enter Fahrenheit Temperature:')
try:
fahr = float(inp)
cel = (fahr - 32.0) * 5.0 / 9.0
print(cel)
except:
print('Please enter a number')
```

Python starts by executing the sequence of statements in the try block. If all goes well, it skips the except block and proceeds. If an exception occurs in the try block, Python jumps out of the try block and executes the sequence of statements in the except block.

Enter Fahrenheit Temperature:72
22.22222222222222

Enter Fahrenheit Temperature:fred
Please enter a number

Handling an exception with a try statement is called catching an exception. In this example, the except clause prints an error message. In general, catching an exception gives you a chance to fix the problem, or try again, or at least end the program gracefully.

Q 6 What is short circuit evaluation of expressions? Explain with proper examples.

Ans:

When Python is processing a logical expression such as x >= 2 and (x/y) > 2, it evaluates the expression from left to right. Because of the definition of and, if x is less than 2, the expression x >= 2 is False and so the whole expression is False regardless of whether (x/y) > 2 evaluates to True or False. When Python detects that there is nothing to be gained by evaluating the rest of a logical expression, it stops its evaluation and does not do the computations in the rest of the logical expression. When the evaluation of a logical expression stops because the overall value is already known, it is called short-circuiting the evaluation.

While this may seem like a fine point, the short-circuit behavior leads to a clever technique called the guardian pattern. Consider the following code sequence in the Python interpreter:

```
>>> x = 6
>>> y = 2
>>> x >= 2 and (x/y) > 2
True
>>> x = 1
>>> y = 0
>>> x >= 2 and (x/y) > 2
False
>>> x = 6
>>> y = 0
>>> x >= 2 and (x/y) > 2
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
```

The third calculation failed because Python was evaluating (x/y) and y was zero, which causes a runtime error. But the second example did not fail because the first part of the expression x >= 2 evaluated to False so the (x/y) was not ever executed due to the short-circuit rule and there was no error.

We can construct the logical expression to strategically place a guard evaluation just before the evaluation that might cause an error as follows:

```
>>> x = 1
>>> y = 0
>>> x >= 2 and y != 0 and (x/y) > 2
False
>>> x = 6
>>> y = 0
>>> x >= 2 and y != 0 and (x/y) > 2
```

False
>>> x >= 2 and (x/y) > 2 and y != 0
Traceback (most recent call last):

File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
In the first logical expression, x >= 2 is False so the evaluation stops at the and. In the second logical expression, x >= 2 is True but y != 0 is False so we never reach (x/y). In the third logical expression, the y != 0 is after the (x/y) calculation so the expression fails with an error. In the second expression, we say that y != 0 acts as a guard to insure that we only execute (x/y) if y is non-zero.

Q 7 Write a Python program to convert temperatures to and from celsius, Fahrenheit.
[Formula : c/5 = f-32/9 [where c = temperature in celsius and f = temperature in fahrenheit]
***Expected Output* :**
60°C is 140 in Fahrenheit
45°F is 7 in Celsius

Ans:

```
temp = input("Input the  temperature you like to convert? (e.g., 45F, 102C etc.) : ")

degree = int(temp[:-1])

i_convention = temp[-1]



if i_convention.upper() == "C":

  result = int(round((9 * degree) / 5 + 32))

  o_convention = "Fahrenheit"

elif i_convention.upper() == "F":

  result = int(round((degree - 32) * 5 / 9))

  o_convention = "Celsius"

else:

  print("Input proper convention.")
```

```
quit()
```

print("The temperature in", o_convention, "is", result, "degrees."

Q 8 Write a Python program to guess a number between 1 to 9.

Ans:

```
 import random
target_num, guess_num = random.randint(1,10),0
while target_num != guess_num:
   guess_num =int(input('Guess a number between 1 and 10 until you get it right : '))
print('Well guessed!')
```

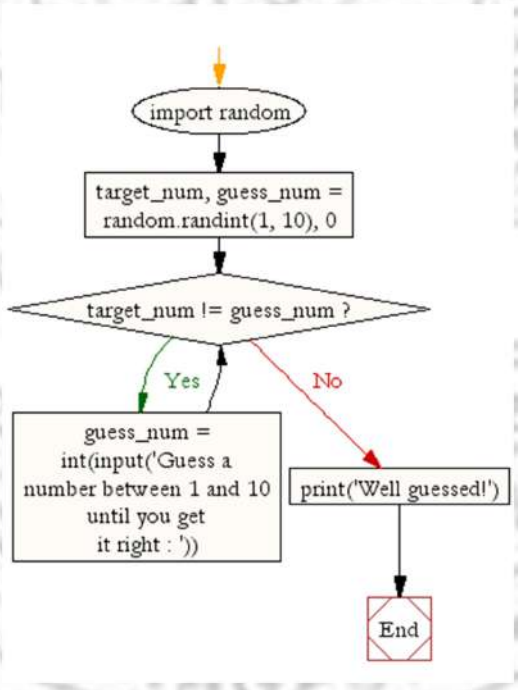


Fig No: 5

Q 9 Write a Python program to construct the following pattern, using a nested for loop.

```
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```

Ans:

```
n=5;for i inrange(n):
for j inrange(i):
print('* ', end="")
print('')
for i inrange(n,0,-1):
for j inrange(i):
print('* ', end="")
print('')
```

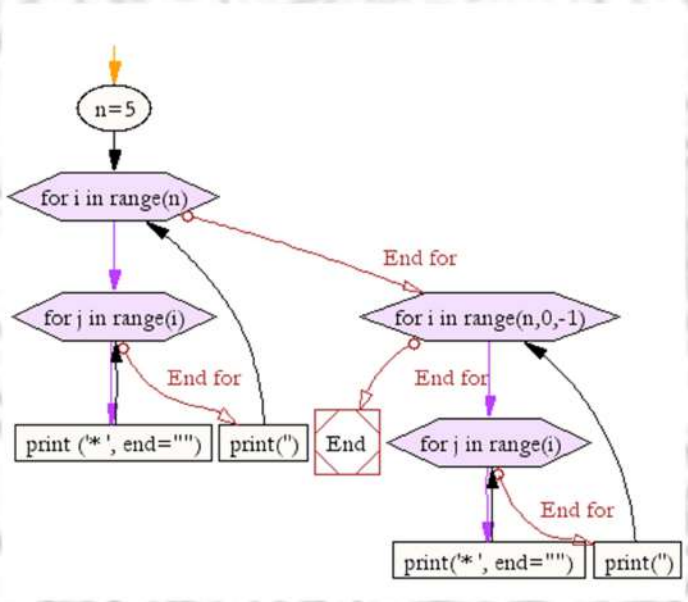


Fig No: 6

Q 10 Write a Python program to count the number of even and odd numbers from a series of numbers.

Ans: *Sample numbers* : numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9)
Expected Output :
Number of even numbers : 5
Number of odd numbers : 4

```
numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9) # Declaring the tuple
count_odd = 0
count_even = 0
for x in numbers:
    if not x % 2:
        count_even+=1
    else:
```

```
        count_odd+=1
print("Number of even numbers :",count_even)
print("Number of odd numbers :",count_odd)
```

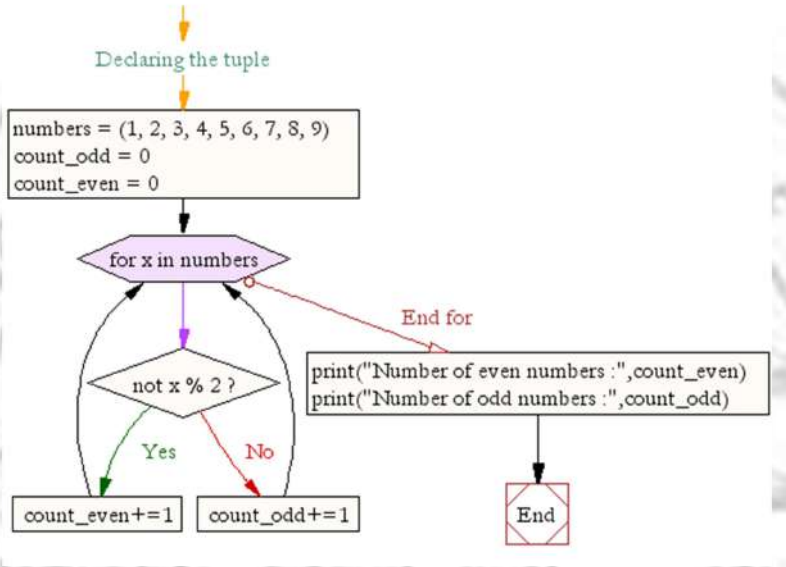


Fig No: 7

Q 11 Write a Python program which takes two digits m (row) and n (column) as input and generates a two-dimensional array. The element value in the i-th row and j-th column of the array should be i*j.

Note :
i = 0,1.., m-1
j = 0,1, n-1.

Test Data : Rows = 3, Columns = 4
Expected Result : [[0, 0, 0, 0], [0, 1, 2, 3], [0, 2, 4, 6]]

Ans:

```
row_num = int(input("Input number of rows: "))

col_num = int(input("Input number of columns: "))

multi_list = [[0 for col in range(col_num)] for row in range(row_num)]
```

```
for row in range(row_num):
    for col in range(col_num):
        multi_list[row][col]= row*col
print(multi_list)
```

Unit IV

Q 1 Write a Python program that accepts a string and calculate the number of digits and letters.

Sample Data : Python 3.2
Expected Output :
Letters 6
Digits 2

Ans:

```
s = input("Input a string")
d=l=0
for c in s:
if c.isdigit():
    d=d+1
  elif c.isalpha():
    l=l+1
  else:
    pass
print("Letters", l)
print("Digits", d)
```

Q 2 Write a Python program to check the validity of password input by users.

Validation :

At least 1 letter between [a-z] and 1 letter between [A-Z].

At least 1 number between [0-9].

At least 1 character from [$#@].

Minimum length 6 characters.

Maximum length 16 characters.

Ans:

```
import re
p= input("Input your password")
x = True
while x:
  if (len(p)<6 or len(p)>12):
      break
  elif not re.search("[a-z]",p):
      break
  elif not re.search("[0-9]",p):
      break
   elif not re.search("[A-Z]",p):
      break
  elif not re.search("[$#@]",p):
      break
  elif re.search("\s",p):
      break
   else:
      print("Valid Password")
      x=False
      break

if x:
    print("Not a Valid Password")
```

Q 3 Write a Python program to calculate the length of a string.

```
def string_length(str1):
    count = 0
    for char in str1:
        count += 1
    return count
print(string_length('w3resource.com'))
```

Q 4 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}

Ans:

```
def char_frequency(str1):
    dict = {}
    for n in str1:
        keys = dict.keys()
        if n in keys:
            dict[n] += 1
        else:
            dict[n] = 1
    return dict
print(char_frequency('google.com'))
```

Q 5 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 : 'w3resource'
Expected Result : 'w3ce'

Sample String : 'w3'
Expected Result : 'w3w3'
Sample String : ' w'
Expected Result : Empty String

Ans:

```
def string_both_ends(str):
  if len(str) < 2:
    return ''
  return str[0:2] + str[-2:]

print(string_both_ends('w3resource'))
print(string_both_ends('w3'))
print(string_both_ends('w'))
```

Q 6 Write a Python program to get a string from a given string where all occurrences of its first char have been changed to '$', except the first char itself.

Sample String : 'restart'
Expected Result : 'resta$t'

Ans:

```
def change_char(str1):
  char = str1[0]
  str1 = str1.replace(char, '$')
  str1 = char + str1[1:]
  return str1

print(change_char('restart'))
```

Q 7 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 : 'abc', 'xyz'
Expected Result : 'xyc abz'

Ans:

```
def chars_mix_up(a, b):
  new_a = b[:2] + a[2:]
  new_b = a[:2] + b[2:]

  return  new_a  +  '  '  +  new_b
print(chars_mix_up('abc', 'xyz'))
```

Q 8 Write a Python program to add 'ing' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add 'ly' instead. If the string length of the given string is less than 3, leave it unchanged.

Sample String : 'abc'
Expected Result : 'abcing'
Sample String : 'string'
Expected Result : 'stringly'

```
def add_string(str1):
  length = len(str1)

  if length > 2:
   if str1[-3:] == 'ing':
     str1 += 'ly'
   else:
     str1 += 'ing'

  return str1
```

```
print(add_string('ab'))

print(add_string('abc'))

print(add_string('string'))
```

Q 9 Write a Python program to find the first appearance of the substring 'not' and 'poor' from a given string, if 'not' follows the 'poor', replace the whole 'not'...'poor' substring with 'good'. Return the resulting string.

Sample String : 'The lyrics is not that poor!'
'The lyrics is poor!'
Expected Result : 'The lyrics is good!'
'The lyrics is poor!'

Ans:

```
def not_poor(str1):
  snot = str1.find('not')
  spoor = str1.find('poor')



  if spoor > snot and snot>0 and spoor>0:
   str1 = str1.replace(str1[snot:(spoor+4)], 'good')
    return str1
  else:
    return str1
print(not_poor('The lyrics is not that poor!'))

print(not_poor('The lyrics is poor!'))
```

Q 10 Python Program to Find the Largest Number in a List

Problem Solution

1. Take in the number of elements and store it in a variable.
2. Take in the elements of the list one by one.
3. Sort the list in ascending order.
4. Print the last element of the list.
5. Exit.

Program/Source Code

```
a=[]
n=int(input("Enter number of elements:"))for i inrange(1,n+1):
   b=int(input("Enter element:"))
   a.append(b)
a.sort()print("Largest element is:",a[n-1])
```

Program Explanation

1. User must enter the number of elements and store it in a variable.
2. User must then enter the elements of the list one by one using a for loop and store it in a list.
3. The list should then be sorted.
4. Then the last element of the list is printed which is also the largest element of the list.

Runtime Test Cases

```
Case 1:
Enter number of elements:3
Enter element:23
Enter element:567
Enter element:3
Largest element is: 567

Case 2:
```

```
Enter number of elements:4
Enter element:34
Enter element:56
Enter element:24
Enter element:54
Largest element is: 56
```

Q 11 Python Program to Put Even and Odd elements in a List into Two Different Lists

Problem Description

The program takes a list and puts the even and odd elements in it into two separate lists.

Problem Solution

1. Take in the number of elements and store it in a variable.
2. Take in the elements of the list one by one.
3. Use a for loop to traverse through the elements of the list and an if statement to check if the element is even or odd.
4. If the element is even, append it to a separate list and if it is odd, append it to a different one.
5. Display the elements in both the lists.
6. Exit.

Program

```
a=[]
n=int(input("Enter number of elements:"))for i inrange(1,n+1):
   b=int(input("Enter element:"))
   a.append(b)
even=[]
odd=[]for j in a:
if(j%2==0):
      even.append(j)
else:
```

```
odd.append(j)print("The even list",even)print("The odd list",odd)
```

Program Explanation

1. User must enter the number of elements and store it in a variable.
2. User must then enter the elements of the list one by one using a for loop and store it in a list.
3. Another for loop is used to traverse through the elements of the list.
4. The if statement checks if the element is even or odd and appends them to separate lists.
5. Both the lists are printed.

Runtime Test Cases

```
Case 1:
Enter number of elements:5
Enter element:67
Enter element:43
Enter element:44
Enter element:22
Enter element:455
The even list [44, 22]
The odd list [67, 43, 455]

Case 2:
Enter number of elements:3
Enter element:23
Enter element:44
Enter element:99
The even list [44]
The odd list [23, 99]
```

Q 12 Python Program to Merge Two Lists and Sort it

Problem Description

The program takes two lists, merges them and sorts the merged list.

Problem Solution

1. Take in the number of elements for the first list and store it in a variable.
2. Take in the elements of the list one by one.
3. Similarly, take in the elements for the second list also.
4. Merge both the lists using the '+' operator and then sort the list.
5. Display the elements in the sorted list.
6. Exit.

Program/Source Code

Here is source code of the Python Program to merge two lists and sort it. The program output is also shown below.

```
a=[]
c=[]
n1=int(input("Enter number of elements:"))for i inrange(1,n1+1):
  b=int(input("Enter element:"))
  a.append(b)
n2=int(input("Enter number of elements:"))for i inrange(1,n2+1):
  d=int(input("Enter element:"))
  c.append(d)new=a+cnew.sort()print("Sorted list is:",new)
```

Program Explanation

1. User must enter the number of elements for the first list and store it in a variable.
2. User must then enter the elements of the list one by one using a for loop and store it in a list.
3. User must similarly enter the elements of the second list one by one.
4. The '+' operator is then used to merge both the lists.
5. The sort function then sorts the list in ascending order.
6. The sorted list is then printed.

Runtime Test Cases

```
Case 1:
Enter number of elements:5
Enter element:10
Enter element:20
Enter element:35
Enter element:55
Enter element:71
Enter number of elements:2
Enter element:5
Enter element:37
Sorted list is: [5, 10, 20, 35, 37, 55, 71]

Case 2:
Enter number of elements:3
Enter element:2
Enter element:4
Enter element:8
Enter number of elements:3
Enter element:2
Enter element:4
Enter element:0
Sorted list is: [0, 2, 2, 4, 4, 8]
```

Q 13 Python Program to Sort the List According to the Second Element in Sublist

Problem Description

The program takes a list of lists and sorts the list according to the second element in the sublist.

Problem Solution

1. Take in a list containing sublists.
2. Using two for loops, use bubble sort to sort the sublists based on the second value of the sublist.
3. If the second element of the first sublist is greater than the second element of the second sublist, exchange the entire sublist.
4. Print the sorted list.
5. Exit.

Program/Source Code

Here is source code of the Python Program to sort the list according to the second element in the sublist. The program output is also shown below.

```
a=[['A',34],['B',21],['C',26]]for i inrange(0,len(a)):
for j inrange(0,len(a)-i-1):
if(a[j][1]>a[j+1][1]):
        temp=a[j]
        a[j]=a[j+1]
        a[j+1]=temp
 print(a)
```

Program Explanation

1. User must enter a list containing several sublists.
2. Then bubble sort is implemented to sort the list according to the second element in the sublist.
3. If the second element of the first sublist is greater than the second element of the second sublist, then the entire sublist is switched.
4. This process continues till the entire list has been sorted.
5. The sorted list is then printed.

Runtime Test Cases

```
Case 1:
[['B', 21], ['C', 26], ['A', 34]]
```

Q 14 Python Program to Find the Second Largest Number in a List Using Bubble Sort

Problem Description

The program takes a list and finds the second largest number in the list using bubble sort.

Ans:

Problem Solution

1. Take in the number of elements for the list and store it in a variable.
2. Take in the elements of the list one by one.
3. Then sort the list using bubble sort.
4. Print the second last element of the sorted list which is the second largest number.
5. Exit.

Program

```
a=[]
n=int(input("Enter number of elements:"))for i inrange(1,n+1):
   b=int(input("Enter element:"))
   a.append(b)for i inrange(0,len(a)):
for j inrange(0,len(a)-i-1):
if(a[j]>a[j+1]):
        temp=a[j]
        a[j]=a[j+1]
        a[j+1]=temp print('Second largest number is:',a[n-2])
```

Program Explanation

1. User must enter the number of elements for the list and store it in a variable.
2. User must then enter the elements of the list one by one using a for loop and store it in a list.
3. Then bubble sort algorithm may be used to sort the elements of the list.
4. The second last element of the sorted list is then printed which is basically the second largest element in the list.

Runtime Test Cases

```
Case 1:
Enter number of elements:4
Enter element:56
Enter element:43
Enter element:78
Enter element:12
('Second largest number is:', 56)

Case 2:
Enter number of elements:6
Enter element:23
Enter element:45
Enter element:94
Enter element:10
Enter element:34
Enter element:95
('Second largest number is:', 94)
```

Q 15 Python Program to Sort a List According to the Length of the Elements

Ans:

Problem Description

The program takes a list and sorts the list according to the length of the elements.

Problem Solution

1. Take in the number of elements for the first list and store it in a variable.
2. Take in the elements of the list one by one.
3. Then sort the list giving the length of the elements as the key.
4. Display the sorted list.
5. Exit.

Program

Here is source code of the Python Program to sort a list according to the length of the elements. The program output is also shown below.

```
a=[]
n=int(input("Enter number of elements:"))for i inrange(1,n+1):
    b=input("Enter element:")
    a.append(b)
a.sort(key=len)print(a)
```

Program Explanation

1. User must enter the number of elements for the first list and store it in a variable.
2. User must then enter the elements of the list one by one using a for loop and store it in a list.
3. Then the list is sorted using the length of the elements as the key.
4. The sorted list is then printed.

Runtime Test Cases

```
Case 1:
Enter number of elements:4
Enter element:"Apple"
Enter element:"Ball"
Enter element:"Cat"
Enter element:"Dog"
['Cat', 'Dog', 'Ball', 'Apple']
```

Case 2:

Enter number of elements:4

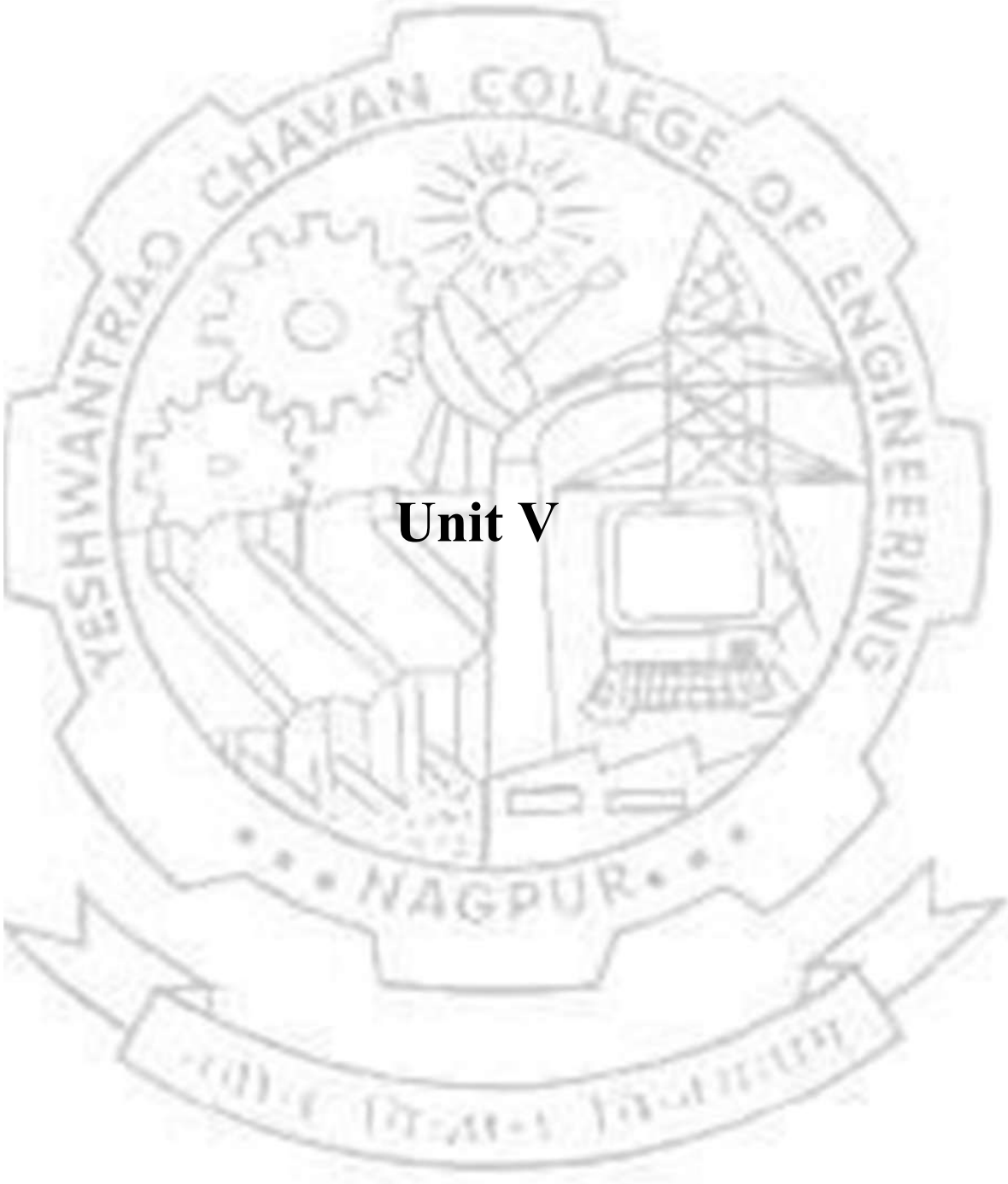
Enter element:"White"

Enter element:"Red"

Enter element:"Purple"

Enter element:"Orange"

['Red', 'White', 'Purple', 'Orange']



Unit V

Q 1 Python Program to Find the Union of two Lists

Ans: Problem Description

The program takes two lists and finds the unions of the two lists.

Problem Solution

1. Define a function which accepts two lists and returns the union of them.
2. Declare two empty lists and initialise to an empty list.
3. Consider a for loop to accept values for two lists.
4. Take the number of elements in the list and store it in a variable.
5. Accept the values into the list using another for loop and insert into the list.
6. Repeat 4 and 5 for the second list also.
7. Find the union of the two lists.
8. Print the union.
9. Exit.

Program

Here is source code of the Python Program to find the union of two lists. The program output is also shown below.

```
l1 =[]
num1 =int(input('Enter size of list 1: '))for n inrange(num1):
    numbers1 =int(input('Enter any number:'))
    l1.append(numbers1)

l2 =[]
num2 =int(input('Enter size of list 2:'))for n inrange(num2):
    numbers2 =int(input('Enter any number:'))
    l2.append(numbers2)
```

```
union =list(set().union(l1,l2))
 print('The Union of two lists is:',union)
```

Program Explanation

1. User must enter the number of elements in the list and store it in a variable.
2. User must enter the values to the same number of elements into the list.
3. The append function obtains each element from the user and adds the same to the end of the list as many times as the number of elements taken.
4. The same of 2 and 3 is done for the second list also.
5. The union function accepts two lists and returns the list which is the union of the two lists, i.e, all the values from list 1 and 2 without redundancy.
6. The set function in the union function accepts a list and returns the list after elimination of redundant values.
7. The lists are passed to the union function and the returned list is printed.

Runtime Test Cases

```
Case 1:
Enter size of list 1: 4
Enter any number: 20
Enter any number: 40
Enter any number: 30
Enter any number: 60
Enter size of list 2: 4
Enter any number: 10
Enter any number: 20
Enter any number: 50
Enter any number: 40
The Union of two lists is: [40, 10, 50, 20, 60, 30]

Case 2:
```

```
Enter size of list 1: 3
Enter any number: 5
Enter any number: 6
Enter any number: 7
Enter size of list 2: 2
Enter any number: 8
Enter any number: 9
The Union of two lists is: [8, 9, 5, 6, 7]
```

Q 2 Python Program to Find the Intersection of Two Lists

Ans:

Problem Description

The program takes two lists and finds the intersection of the two lists.

Problem Solution

1. Define a function which accepts two lists and returns the intersection of them.
2. Declare two empty lists and initialize them to an empty list.
3. Consider a for loop to accept values for the two lists.
4. Take the number of elements in the list and store it in a variable.
5. Accept the values into the list using another for loop and insert into the list.
6. Repeat 4 and 5 for the second list also.
7. Find the intersection of the two lists.
8. Print the intersection.
9. Exit.

Program

Here is source code of the Python Program to find the intersection of two lists. The program output is also shown below.

```
def intersection(a, b):
returnlist(set(a)&set(b))
```

```
def main():
   alist=[]
   blist=[]
   n1=int(input("Enter number of elements for list1:"))
   n2=int(input("Enter number of elements for list2:"))
print("For list1:")
for x inrange(0,n1):
      element=int(input("Enter element" + str(x+1) + ":"))
      alist.append(element)
print("For list2:")
for x inrange(0,n2):
      element=int(input("Enter element" + str(x+1) + ":"))
      blist.append(element)
print("The intersection is :")
print(intersection(alist, blist))
main()
```

Program Explanation

1. User must enter the number of elements in the list and store it in a variable.
2. User must enter the values to the same number of elements into the list.
3. The append function obtains each element from the user and adds the same to the end of the list as many times as the number of elements taken.
4. The same of 2 and 3 is done for the second list also.
5. The intersection function accepts two lists and returns the list which is the intersection of the two lists, i.e, all the common values from list 1 and 2.
6. The set function in the intersection function accepts a list and returns the list which contains the common elements in both the lists.
7. The lists are passed to the intersection function and the returned list is printed.

Runtime Test Cases

Case 1:

Enter number of elements for list1:3

Enter number of elements for list2:4

For list1:

Enter element1:34

Enter element2:23

Enter element3:65

For list2:

Enter element1:33

Enter element2:65

Enter element3:23

Enter element4:86

The intersection is :

[65, 23]

Case 2:

Enter number of elements for list1:2

Enter number of elements for list2:4

For list1:

Enter element1:3

Enter element2:4

For list2:

Enter element1:12

Enter element2:34

Enter element3:4

Enter element4:6

The intersection is :

[4]

Q 3 Python Program to Create a List of Tuples with the First Element as the Number and Second Element as the Square of the Number

Ans:

Problem Description

The program takes a range and creates a list of tuples within that range with the first element as the number and the second element as the square of the number.

Problem Solution

1. Take the upper and lower range for the numbers from the user.
2. A list of tuples must be created using list comprehension where the first element is the number within the range and the second element is the square of the number.
3. This list of tuples is then printed.
4. Exit.

Program

Here is source code of the Python Program to create a list of tuples with the first element as the number and the second element as the square of the number. The program output is also shown below.

```
l_range=int(input("Enter the lower range:"))
u_range=int(input("Enter the upper range:"))
a=[(x,x**2)for x inrange(l_range,u_range+1)]print(a)
```

Program Explanation

1. User must enter the upper and lower range for the numbers.
2. List comprehension must be used to create a list of tuples where the first number is the number itself from the given range and the second element is a square of the first number.
3. The list of tuples which is created is printed.

Runtime Test Cases

```
Case 1:
Enter the lower range:1
```

```
Enter the upper range:4
[(1, 1), (2, 4), (3, 9), (4, 16)]


Case 2:
Enter the lower range:45
Enter the upper range:49
[(45, 2025), (46, 2116), (47, 2209), (48, 2304), (49, 2401)]
```

Q 4 Python Program to Find all Numbers in a Range which are Perfect Squares and Sum of all Digits in the Number is Less than 10

Ans:

Problem Description

The program takes a range and creates a list of all numbers in the range which are perfect squares and the sum of the digits is less than 10.

Problem Solution

1. User must enter the upper and lower range for the numbers.
2. A list must be created using list comprehension where the element is a perfect square within the range and the sum of the digits of the number is less than 10.
3. This list must then be printed.
4. Exit.

Program

Here is source code of the Python Program to create a list of of all numbers in a range which are perfect squares and the sum of the digits of the number is less than 10. The program output is also shown below.

```
l=int(input("Enter lower range: "))
```

```
u=int(input("Enter upper range: "))
a=[]
a=[x for x inrange(l,u+1)if(int(x**0.5))**2==x andsum(list(map(int,str(x))))<10]print(a)
```

Program Explanation

1. User must enter the upper and lower range for the numbers.
2. List comprehension must be used to create a list of numbers which are perfect squares and sum of the digits is less than 10.
3. The second part of the list comprehension first maps separate digits of the number into a list and finds the sum of the elements in the list.
3. The list which is created is printed.

Runtime Test Cases

Case 1:

Enter lower range: 1

Enter upper range: 40

[1, 4, 9, 16, 25, 36]

Case 2:

Enter lower range: 50

Enter upper range: 100

[81, 100]

Q 5 Python Program to Find all Numbers in a Range which are Perfect Squares and Sum of all Digits in the Number is Less than 10

Ans:

Problem Description

The program takes a range and creates a list of all numbers in the range which are perfect squares and the sum of the digits is less than 10.

Problem Solution

1. User must enter the upper and lower range for the numbers.
2. A list must be created using list comprehension where the element is a perfect square within the range and the sum of the digits of the number is less than 10.
3. This list must then be printed.
4. Exit.

Program/Source Code

Here is source code of the Python Program to create a list of of all numbers in a range which are perfect squares and the sum of the digits of the number is less than 10. The program output is also shown below.

```
l=int(input("Enter lower range: "))
u=int(input("Enter upper range: "))
a=[]
a=[x for x inrange(l,u+1)if(int(x**0.5))**2==x andsum(list(map(int,str(x))))<10]print(a)
```

Program Explanation

1. User must enter the upper and lower range for the numbers.
2. List comprehension must be used to create a list of numbers which are perfect squares and sum of the digits is less than 10.
3. The second part of the list comprehension first maps separate digits of the number into a list and finds the sum of the elements in the list.
3. The list which is created is printed.

Runtime Test Cases

```
Case 1:
Enter lower range: 1
Enter upper range: 40
[1, 4, 9, 16, 25, 36]

Case 2:
```

```
Enter lower range: 50
Enter upper range: 100
[81, 100]
```

Q 6 Python Program to Remove the ith Occurrence of the Given Word in a List where Words can Repeat

Ans:

Problem Description

The program takes a list and removes the ith occurrence of the given word in the list where words can repeat.

Problem Solution

1. Take the number of elements in the list and store it in a variable.
2. Accept the values into the list using a for loop and insert them into the list.
3. Use a for loop to traverse through the elements in the list.
4. Then use an if statement to check if the word to be removed matches the element and the occurrence number and otherwise it appends the element to another list.
5. The number of repetitions along with the updated list and distinct elements is printed.
6. Exit.

Program

Here is source code of the Python Program to remove the ith occurrence of the given word in list where words can repeat. The program output is also shown below.

```
a=[]
n=int(input("Enter the number of elements in list:"))for x inrange(0,n):
    element=input("Enter element" + str(x+1) + ":")
    a.append(element)print(a)
c=[]
count=0
b=input("Enter word to remove: ")
```

```
n=int(input("Enter the occurrence to remove: "))for i in a:
if(i==b):
    count=count+1
if(count!=n):
        c.append(i)
else:
    c.append(i)if(count==0):
print("Item not found ")else:
print("The number of repetitions is: ",count)
print("Updated list is: ",c)
print("The distinct elements are: ",set(a))
```

Program Explanation

1. User must enter the number of elements in the list and store it in a variable.
2. User must enter the values of elements into the list.
3. The append function obtains each element from the user and adds the same to the end of the list as many times as the number of elements taken.
4. User must enter the word and the occurrence of the word to remove.
5. A for loop is used to traverse across the elements in the list.
6. An if statement then checks whether the element matches equal to the word that must be removed and whether the occurrence of the element matches the occurrence to be removed.
7. If both aren't true, the element is appended to another list.
8. If only the word matches, the count value is incremented.
9. Finally the number of repetitions along with the updated list and the distinct elements is printed.

Runtime Test Cases

```
Case 1:
Enter the number of elements in list:5
Enter element1:"apple"
Enter element2:"apple"
Enter element3:"ball"
```

```
Enter element4:"ball"
Enter element5:"cat"
['apple', 'apple', 'ball', 'ball', 'cat']
Enter word to remove: "ball"
Enter the occurence to remove: 2
('The number of repitions is: ', 2)
('Updated list is: ', ['apple', 'apple', 'ball', 'cat'])
('The distinct elements are: ', set(['ball', 'apple', 'cat']))

Case 2:
Enter the number of elements in list:6
Enter element1:"A"
Enter element2:"B"
Enter element3:"A"
Enter element4:"A"
Enter element5:"C"
Enter element6:"A"
['A', 'B', 'A', 'A', 'C', 'A']
Enter word to remove: "A"
Enter the occurence to remove: 3
('The number of repitions is: ', 4)
('Updated list is: ', ['A', 'B', 'A', 'C', 'A'])
('The distinct elements are: ', set(['A', 'C', 'B']))
```

Q 7Python Program to Remove All Tuples in a List of Tuples with the USN Outside the Given Range

Ans:

Problem Description

The program removes all tuples in a list of tuples with the USN outside the given range.

Problem Solution

1. Take in the lower and upper roll number from the user.
2. Then append the prefixes of the USN's to the roll numbers.
3. Using list comprehension, find out which USN's lie in the given range.
4. Print the list containing the tuples.
5. Exit.

Program

Here is source code of the Python Program to remove all tuples in a list of tuples with the USN outside the given range. The program output is also shown below.

```
y=[('a','12CS039'),('b','12CS320'),('c','12CS055'),('d','12CS100')]
low=int(input("Enter lower roll number (starting with 12CS):"))
up=int(input("Enter upper roll number (starting with 12CS):"))
l='12CS0'+str(low)
u='12CS'+str(up)
p=[x for x in y if x[1]>l and x[1]<u]print(p)]
```

Program Explanation

1. User must enter the upper and lower roll number.
2. The prefixes are then appended to the roll numbers using the '+' operator to form the USN.
3. List comprehension is then used to find the USN's within the upper and lower range.
4. The list containing USN's in the given range is then printed.

Runtime Test Cases

```
Case 1:
Enter lower roll number (starting with 12CS):50
Enter upper roll number (starting with 12CS):150
[('c', '12CS055'), ('d', '12CS100')]
```

Q 8 Python Program to solve Maximum Subarray Problem using Divide and Conquer

Ans:

Problem Description

The program finds a subarray that has the maximum sum within the given array.

Problem Solution

1. Define the function find_max_subarray that takes a list as argument and two indexes, start and end. It finds the maximum subarray in the range [start, end – 1].
2. The function find_max_subarray returns the tuple (l, r, m) where l, r are the left and right indexes of the maximum subarray and m is its sum.
3. The base case is when the input list is just one element (i.e. start == end – 1).
4. Otherwise, the list is divided into two and find_max_subarray is called on both the halves.
5. The maximum subarray is either the maximum subarray of one of the halves or the maximum subarray crossing the midpoint of the two halves.
6. The function find_max_crossing_subarray takes a list as argument and three indexes, start, mid and end and finds the maximum subarray containing mid.

Program/Source Code

Here is the source code of a Python program to find the subarray with maximum sum. The program output is shown below.

```
def find_max_subarray(alist, start, end):
"""Returns (l, r, m) such that alist[l:r] is the maximum subarray in
   A[start:end] with sum m. Here A[start:end] means all A[x] for start <= x <
   end."""
# base case
if start == end - 1:
return start, end, alist[start]
else:
      mid =(start + end)//2
      left_start, left_end, left_max = find_max_subarray(alist, start, mid)
```

```
        right_start, right_end, right_max = find_max_subarray(alist, mid, end)
        cross_start, cross_end, cross_max = find_max_crossing_subarray(alist, start, mid, end)
if(left_max > right_max and left_max > cross_max):
return left_start, left_end, left_max
elif(right_max > left_max and right_max > cross_max):
return right_start, right_end, right_max
else:
return cross_start, cross_end, cross_max
 def find_max_crossing_subarray(alist, start, mid, end):
"""Returns (l, r, m) such that alist[l:r] is the maximum subarray within
    alist with start <= l < mid <= r < end with sum m. The arguments start, mid,
    end must satisfy start <= mid <= end."""
    sum_left =float('-inf')
    sum_temp =0
    cross_start = mid
for i inrange(mid - 1, start - 1, -1):
        sum_temp = sum_temp + alist[i]
if sum_temp > sum_left:
            sum_left = sum_temp
            cross_start = i

    sum_right =float('-inf')
    sum_temp =0
    cross_end = mid + 1
for i inrange(mid, end):
        sum_temp = sum_temp + alist[i]
if sum_temp > sum_right:
            sum_right = sum_temp
            cross_end = i + 1
```

```
return cross_start, cross_end, sum_left + sum_right


alist =input('Enter the list of numbers: ')
alist = alist.split()
alist =[int(x)for x in alist]
start, end, maximum = find_max_subarray(alist,0,len(alist))print('The maximum subarray starts at index {}, ends at index {}'
' and has sum {}.'.format(start, end - 1, maximum))
```

Program Explanation

1. The user is prompted to enter a list of numbers.
2. The list is passed to find_max_subarray and the returned tuple is stored.
3. The start and end indexes and the sum of the maximum subarray is printed.

Runtime Test Cases

```
Case 1:
Enter the list of numbers: 3 4 -2 3 -10 32 4 -11 7 -3 2
The maximum subarray starts at index 5, ends at index 6 and has sum 36.

Case 2:
Enter the list of numbers: 4 -2 3 4 1 4 -5
The maximum sub array starts at index 0, ends at index 5 and has sum 14.

Case 3:
Enter the list of numbers: 2
The maximum sub array starts at index 0, ends at index 0 and has sum 2.
```

Q 9Python Program to Find Element Occurring Odd Number of Times in a List

Ans:

Problem Description

A list is given in which all elements except one element occurs an even number of times. The problem is to find the element that occurs an odd number of times.

Problem Solution

1. The function find_odd_occurring is defined.
2. It takes a list as argument which has only one element that occurs an odd number of times.
3. The function returns that element.
4. It finds the element by XORing all elements in the list.
5. Since the XOR operation is commutative and associative and it satisfies p XOR p = 0 and p XOR 0 = p, the element that occurs an odd number of times is the result.

Program

Here is the source code of a Python program to find element that occurs odd number of times in a list. The program output is shown below.

```
def find_odd_occurring(alist):
"""Return the element that occurs odd number of times in alist.

   alist is a list in which all elements except one element occurs an even
   number of times.
"""
   ans =0

for element in alist:
       ans ^= element

return ans
```

```
alist =input('Enter the list: ').split()
alist =[int(i)for i in alist]
ans = find_odd_occurring(alist)print('The element that occurs odd number of times:', ans)
```

Program Explanation

1. The user is prompted to enter the list.
2. find_odd_occurring is called on the list.
3. The result is then displayed.

Runtime Test Cases

```
Case 1:
Enter the list: 15 22 10 33 22 33 15 1 1 15 15
The element that occurs odd number of times: 10

Case 2:
Enter the list: 3
The element that occurs odd number of times: 3

Case 3:
Enter the list: 1 2 3 1 2 3 4 1 2 3 1 2 4 3 4
The element that occurs odd number of times: 4
```

Q 10 Python Program to Add a Key-Value Pair to the Dictionary

Ans:

Problem Description

The program takes a key-value pair and adds it to the dictionary.

Problem Solution

1. Take a key-value pair from the user and store it in separate variables.
2. Declare a dictionary and initialize it to an empty dictionary.
3. Use the update() function to add the key-value pair to the dictionary.
4. Print the final dictionary.
5. Exit.

Program

Here is source code of the Python Program to add a key-value pair to a dictionary. The program output is also shown below.

```
key=int(input("Enter the key (int) to be added:"))
value=int(input("Enter the value for the key to be added:"))
d={}
d.update({key:value})print("Updated dictionary is:")print(d)
```

Program Explanation

1. User must enter a key-value pair and store it in separate variables.
2. A dictionary is declared and initialized to an empty dictionary.
3. The update() function is used to add the key-value pair to the dictionary.
4. The final dictionary is printed.

Runtime Test Cases

```
Case 1:
Enter the key (int) to be added:12
Enter the value for the key to be added:34
Updated dictionary is:
{12: 34}

Case 2:
Enter the key (int) to be added:34
Enter the value for the key to be added:29
```

```
Updated dictionary is:
{34: 29}
```

Q 11 Python Program to Concatenate Two Dictionaries Into One
Ans:

Problem Description

The program takes two dictionaries and concatenates them into one dictionary.

Problem Solution

1. Declare and initialize two dictionaries with some key-value pairs
2. Use the update() function to add the key-value pair from the second dictionary to the first dictionary.
3. Print the final dictionary.
4. Exit.

Program/Source Code

Here is source code of the Python Program to add a key-value pair to a dictionary. The program output is also shown below.

```
d1={'A':1,'B':2}
d2={'C':3}
d1.update(d2)print("Concatenated dictionary is:")print(d1)
```

Program Explanation

1. User must enter declare and initialize two dictionaries with a few key-value pairs and store it in separate variables.
2. The update() function is used to add the key-value pair from the second to the first dictionary.
3. The final updated dictionary is printed.

Runtime Test Cases

```
Case 1:
Concatenated dictionary is:
{'A': 1, 'C': 3, 'B': 2}
```

Q 12 Python Program to Check if a Given Key Exists in a Dictionary or Not

Ans:

Problem Description

The program takes a dictionary and checks if a given key exists in a dictionary or not.

Problem Solution

1. Declare and initialize a dictionary to have some key-value pairs.
2. Take a key from the user and store it in a variable.
3. Using an if statement and the in operator, check if the key is present in the dictionary using the dictionary.keys() method.
4. If it is present, print the value of the key.
5. If it isn't present, display that the key isn't present in the dictionary.
6. Exit.

Program/Source Code

Here is source code of the Python Program to check if a given key exists in a dictionary or not. The program output is also shown below.

```
d={'A':1,'B':2,'C':3}
key=raw_input("Enter key to check:")if key in d.keys():
print("Key is present and value of the key is:")
print(d[key])else:
print("Key isn't present!")
```

Program Explanation

1. User must enter the key to be checked and store it in a variable.
2. An if statement and the in operator is used check if the key is present in the list containing the keys of the dictionary.
3. If it is present, the value of the key is printed.
4. If it isn't present, "Key isn't present!" is printed.
5. Exit.

Runtime Test Cases

```
Case 1:
Enter key to check:A
Key is present and value of the key is:
1

Case 2:
Enter key to check:F
Key isn't present!
```

Q 13 Python Program to Generate a Dictionary that Contains Numbers (between 1 and n) in the Form (x,x*x).

Ans:

Problem Description

The program takes a number from the user and generates a dictionary that contains numbers (between 1 and n) in the form (x,x*x).

Problem Solution

1. Take a number from the user and store it in a separate variable.
2. Declare a dictionary and using dictionary comprehension initialize it to values keeping the number between 1 to n as the key and the square of the number as their values.
3. Print the final dictionary.
4. Exit.

Program

Here is source code of the Python Program to generate a dictionary that contains numbers (between 1 and n) in the form (x,x*x). The program output is also shown below.

```
n=int(input("Enter a number:"))
d={x:x*x for x in range(1,n+1)}print(d)
```

Program Explanation

1. User must enter a number and store it in a variable.
2. A dictionary is declared and initialized to values using dictionary comprehension.
3. The numbers between 1 to n are kept as keys while the squares of the numbers are made their values.
4. The final dictionary is printed.

Runtime Test Cases

```
Case 1:

Enter a number:5

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Q 14 Python Program to Remove the Given Key from a Dictionary

Ans:

Problem Description

The program takes a dictionary and removes a given key from the dictionary.

Problem Solution

1. Declare and initialize a dictionary to have some key-value pairs.
2. Take a key from the user and store it in a variable.
3. Using an if statement and the in operator, check if the key is present in the dictionary.
4. If it is present, delete the key-value pair.
5. If it isn't present, print that the key isn't found and exit the program.
6. Exit.

Program

Here is source code of the Python Program to remove the given key from a dictionary. The program output is also shown below.

```
d ={'a':1,'b':2,'c':3,'d':4}print("Initial dictionary")print(d)

key=raw_input("Enter the key to delete(a-d):")if key in d:

del d[key]else:

print("Key not found!")
```

```
exit(0)print("Updated dictionary")print(d)
```

Program Explanation

1. User must enter the key to be checked and store it in a variable.
2. An if statement and the in operator is used check if the key is present in the dictionary.
3. If it is present, the key-value pair is deleted.
4. If it isn't present, "Key not found!" is printed and the program is exited.

Runtime Test Cases

```
Case 1:
Initial dictionary
{'a': 1, 'c': 3, 'b': 2, 'd': 4}
Enter the key to delete(a-d):c
Updated dictionary
{'a': 1, 'b': 2, 'd': 4}

Case 2:
Initial dictionary
{'a': 1, 'c': 3, 'b': 2, 'd': 4}
Enter the key to delete(a-d):g
Key not found!
```

Q 15 Python Program to Map Two Lists into a Dictionary

Ans:

Problem Description

The program takes two lists and maps two lists into a dictionary.

Problem Solution

1. Declare two empty lists and initialize them to an empty list.
3. Consider a for loop to accept values for the two lists.
4. Take the number of elements in the list and store it in a variable.
5. Accept the values into the list using another for loop and insert into the list.
6. Repeat 4 and 5 for the values list also.
7. Zip the two lists and use dict() to convert it into a dictionary.
8. Print the dictionary.
9. Exit.

Program

Here is source code of the Python Program to map two lists into a dictionary. The program output is also shown below.

```
keys=[]
values=[]
n=int(input("Enter number of elements for dictionary:"))print("For keys:")for x inrange(0,n):
    element=int(input("Enter element" + str(x+1) + ":"))
    keys.append(element)print("For values:")for x inrange(0,n):
    element=int(input("Enter element" + str(x+1) + ":"))
    values.append(element)
d=dict(zip(keys,values))print("The dictionary is:")print(d)
```

Program Explanation

1. User must enter the number of elements in the list and store it in a variable.
2. User must enter the values to the same number of elements into the list.
3. The append function obtains each element from the user and adds the same to the end of the list as many times as the number of elements taken.
4. The same of 2 and 3 is done for the second values list also.
5. The two lists are merged together using the zip() function.
6. The zipped lists are then merged to form a dictionary using dict().
7. The dictionary formed from the two lists is then printed.

Runtime Test Cases

```
Case 1:
```

```
Enter number of elements for dictionary:3
For keys:
Enter element1:1
Enter element2:2
Enter element3:3
For values:
Enter element1:1
Enter element2:4
Enter element3:9
The dictionary is:
{1: 1, 2: 4, 3: 9}

Case 2:
Enter number of elements for dictionary:2
For keys:
Enter element1:23
Enter element2:46
For values:
Enter element1:69
Enter element2:138
The dictionary is:
{46: 138, 23: 69}
```

Q 16 Python Program to Create a Dictionary with Key as First Character and Value as Words Starting with that Character

Ans:

Problem Description

The program takes a string and creates a dictionary with key as first character and value as words starting with that character.

Problem Solution

1. Enter a string and store it in a variable.
2. Declare an empty dictionary.
3. Split the string into words and store it in a list.
4. Using a for loop and if statement check if the word already present as a key in the dictionary.
5. If it is not present, initialize the letter of the word as the key and the word as the value and append it to a sublist created in the list.
6. If it is present, add the word as the value to the corresponding sublist.
7. Print the final dictionary.
8. Exit.

Program

Here is source code of the Python Program to create a dictionary with key as first character and value as words starting with that character. The program output is also shown below.

```
test_string=raw_input("Enter string:")
l=test_string.split()
d={}for word in l:
if(word[0]notin d.keys()):
    d[word[0]]=[]
    d[word[0]].append(word)
else:
if(word notin d[word[0]]):
```

```
        d[word[0]].append(word)for k,v in d.items():
print(k,":",v)
```

Program Explanation

1. User must enter a string and store it in a variable.
2. An empty dictionary is declared.
3. The string is split into words and is stored in a list.
4. A for loop is used to traverse through the words in the list.
5. An if statement is used to check if the word already present as a key in the dictionary.
6. If it is not present, the letter of the word is initialized as the key and the word as the value and they are append to a sublist created in the list.
7. If it is present, the word is added as the value to the corresponding sublist.
8. The final dictionary is printed.

Runtime Test Cases

```
Case 1:
Enter string:Hello world this is a test string sanfoundry
('a', ':', ['a'])
('i', ':', ['is'])
('H', ':', ['Hello'])
('s', ':', ['sanfoundry', 'string'])
('t', ':', ['test', 'this'])
('w', ':', ['world'])

Case 2:
Enter string:python is my most favourite programming language in the entire world
('e', ':', ['entire'])
('f', ':', ['favourite'])
('i', ':', ['in', 'is'])
('m', ':', ['most', 'my'])
('l', ':', ['language'])
('p', ':', ['programming', 'python'])
```

```
('t', ':', ['the'])
('w', ':', ['world'])
```

Q 17 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.

```
{'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]
```

Program:

```
from pprint import pprint
dict_nums = dict(x=list(range(11, 20)), y=list(range(21, 30)), z=list(range(31, 40)))
pprint(dict_nums)
print(dict_nums["x"][4])
print(dict_nums["y"][4])
print(dict_nums["z"][4])
for k,v in dict_nums.items():
  print(k, "has value", v)
```

Unit VI

Q 1 What do you mean by functions in Python.

Ans: In the context of programming, a *function* is a named sequence of statements that performs a computation. When you define a function, you specify the name and the sequence of statements. Later, you can "call" the function by name. One example of a *function call*:

```
>>> type(32)
<class 'int'>
```

The name of the function is type. The expression in parentheses is called the *argument* of the function. The argument is a value or variable that we are passing into the function as input to the function. The result, for the type function, is the type of the argument.
It is common to say that a function "takes" an argument and "returns" a result.
The result is called the *return value*.

Q 2 What do you mean by Type conversion functions in Python.

Ans: Python provides built-in functions that convert values from one type to another. The int function takes any value and converts it to an integer, if it can, or complains otherwise:

```
>>> int('32')
32
>>> int('Hello')

ValueError: invalid literal for int() with base 10: 'Hello'
```

int can convert floating-point values to integers, but it doesn't round off; it chops off the fraction part:

```
>>> int(3.99999)
3
>>> int(-2.3)
-2
float converts integers and strings to floating-point numbers:
>>> float(32)
32.0
>>> float('3.14159')
3.14159
Finally, str converts its argument to a string:
>>> str(32)
'32'
>>> str(3.14159)
'3.14159'
```

Q 3 Write a Python function to find the Max of three numbers.

Ans:

```
def max_of_two( x, y ):
    if x > y:
        return x
    return y
def max_of_three( x, y, z ):
    return max_of_two( x, max_of_two( y, z ) )
print(max_of_three(3, 6, -5))
```

Q 4 Write a Python function to sum all the numbers in a list.

Ans:

Sample List : (8, 2, 3, 0, 7)
Expected Output : 20

```
Sol: def sum(numbers):
    total = 0
    for x in numbers:
        total += x
    return total
print(sum((8, 2, 3, 0, 7)))
```

Q 5 Write a Python function to multiply all the numbers in a list.

***Sample List* : (8, 2, 3, -1, 7)**
***Expected Output* : -336**

Ans:

```
Sol: def multiply(numbers):
    total = 1
    for x in numbers:
        total *= x
    return total
```

```
print(multiply((8, 2, 3, -1, 7)))
```

Q 6 Write a Python program to reverse a string.
***Sample String* : "1234abcd"**
Expected Output : "dcba4321"

Ans:

```
def string_reverse(str1):


   rstr1 = ''

   index = len(str1)

   while index > 0:

      rstr1 += str1[ index - 1 ]

      index = index - 1

  return rstr1

print(string_reverse('1234abcd'))
```

Q 7 Write a Python function to calculate the factorial of a number (a non-negative integer). The function accepts the number as an argument.

Ans:

```
def factorial(n):

   if n == 0:

      return 1

   else:

      return n * factorial(n-1)

n=int(input("Input a number to compute the factiorial : "))

print(factorial(n))
```

Q 8 Write a Python function that accepts a string and calculate the number of upper case letters and lower case letters.

***Sample String* : 'The quick Brow Fox'**
***Expected Output* :**
No. of Upper case characters : 3
No. of Lower case Characters : 12

Ans:

```
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 ("Original String : ", s)
   print ("No. of Upper case characters : ", d["UPPER_CASE"])
   print ("No. of Lower case Characters : ", d["LOWER_CASE"])


string_test('The quick Brown Fox')
```

Q 9 Write a Python function that takes a list and returns a new list with unique elements of the first list.

***Sample List :* [1,2,3,3,3,3,4,5]**
***Unique List :* [1, 2, 3, 4, 5]**

Ans:

```
def unique_list(l):
  x = []
  for a in l:
```

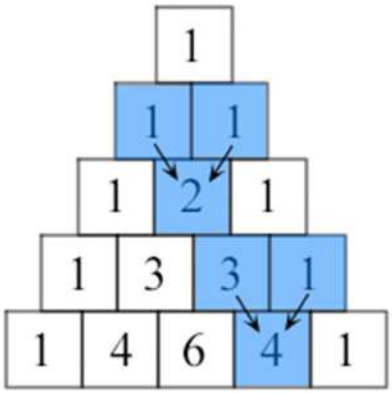
```
    if a not in x:
        x.append(a)
    return x


print(unique_list([1,2,3,3,3,3,4,5]))
```

Q 10 Write a Python function that prints out the first n rows of Pascal's triangle.

Note : Pascal's triangle is an arithmetic and geometric figure first imagined by Blaise Pascal.

Sample Pascal's triangle:




```
def pascal_triangle(n):
    trow = [1]
    y = [0]
    for x in range(max(n,0)):
        print(trow)
        trow=[l+r for l,r in zip(trow+y, y+trow)]
    return n>=1
pascal_triangle(6)
```

Q 11 Write a Python function to check whether a string is a pangram or not.

Note : Pangrams are words or sentences containing every letter of the alphabet at least once. For example : "The quick brown fox jumps over the lazy dog"

Ans:

```
import string, sys

def ispangram(str1, alphabet=string.ascii_lowercase):

   alphaset = set(alphabet)

   return alphaset <= set(str1.lower())

print ( ispangram('The quick brown fox jumps over the lazy dog'))
```

Q 12 Write a Python program to make a chain of function decorators (bold, italic, underline etc.).

Ans:

```
def make_bold(fn):

   def wrapped():

      return "<b>" + fn() + "</b>"

   return wrapped


def make_italic(fn):

   def wrapped():

      return "<i>" + fn() + "</i>"

   return wrapped


def make_underline(fn):

   def wrapped():
```

```
        return "<u>" + fn() + "</u>"
    return wrapped
@make_bold
@make_italic
@make_underline
def hello():
    return "hello world"
print(hello()) ## returns "<b><i><u>hello world</u></i></b>"
```

Q 13 Write a Python program to execute a string containing Python code.

Ans:

```
mycode = 'print("hello world")'
code = """
def mutiply(x,y):
    return x*y


print('Multiply of 2 and 3 is: ',mutiply(2,3))
"""
exec(mycode)
exec(code)
```

Q 14 Write a Python program to access a function inside a function.

Ans:

```
def test(a):
    def add(b):
        nonlocal a
        a += 1
```

```
        return a+b
    return add
func= test(4)
print(func(4))
```

Q 15 Write a Python program to detect the number of local variables declared in a function.

Ans:

```
def abc():
  x = 1
  y = 2
  str1= "w3resource"
  print("Python Exercises")

print(abc.__code__.co_nlocals)
```