Python

Experiment No. 3

Aim: Write Python programs to demonstrate different Control Structures (Decision making & Loop).

Problem Statements:

A) Implement a program to display the geometric sequence as given below for a given value n, where n is the number of elements in the sequence.
1, 2, 4, 8, 16, 32, 64, ..., 1024

Sample Input and Output:

Sample Input	Expected Output
5	1, 2, 4, 8, 16
8	1, 2, 4, 8, 16, 32, 64, 128

B) Implement a program to check whether a given number is a palindrome.
C) Write a program to display all prime numbers within a range
D) Count the total number of digits in a number
E) To display following pattern using nested for loop

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

Theory :

Types of Control Flow in Python

In Python programming language, the type of control flow statements is as follows:

1. The if statement
2. The if-else statement
3. The nested-if statement
4. The if-elif-else ladder

Python if statement

The if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not.

Syntax:

```
if condition:
  # Statements to execute if condition is true
```

Here, the condition after evaluation will be either true or false. if the statement accepts boolean values – if the value is true then it will execute the block of statements below it otherwise not.

As we know, python uses indentation to identify a block. So the block under an if statement will be identified as shown in the below example:

```
if condition:
  statement1
statement2
# Here if the condition is true, if block  will consider only statement1 to be inside its block.
```

Python If-Else Statement

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But if we want to do something else if the condition is false, we can use the else statement with if statement to execute a block of code when the if condition is false.

Syntax:

```
if (condition):
   # Executes this block if condition is true
else:
   # Executes this block if condition is false
```

Nested-If Statement in Python

A nested if is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement. Yes, Python allows us to nest if statements within if statements. i.e., we can place an if statement inside another if statement.

Syntax:

```
if (condition1):
  # Executes when condition1 is true
  if (condition2):
    # Executes when condition2 is true
  # if Block is end here
# if Block is end here
```

Python if-elif-else Ladder

Here, a user can decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

Syntax:

```
if (condition):
   statement
elif (condition):
   statement
.
.
else:
   statement
```

Short Hand if statement Whenever there is only a single statement to be executed inside the if block then shorthand if can be used. The statement can be put on the same line as the if statement.

Syntax:

```
if condition: statement
```

Short Hand if-else statement

This can be used to write the if-else statements in a single line where only one statement is needed in both the if and else blocks.

Syntax:

```
statement_when_True if condition else statement_when_False

# Python program to illustrate short hand if-else

i = 10
```

```
print(True) if i< 15 else print(False)
```

Chaining comparison operators in Python

In Python, there is a better way to write this using the Comparison operator Chaining. The chaining of operators can be written as follows:

```
if a < b < c :
   {.....}
```

According to associativity and precedence in Python, **all comparison operations** in Python have the **same priority**, **which is lower than that of any arithmetic, shifting, or bitwise operation**. Also unlike C, expressions like a < b < c have an interpretation that is conventional in mathematics. List of comparison operators in Python:

">" | "<" | "==" | ">=" | "<=" | "!=" | "is" ["not"] | ["not"] "in"

Chaining in Comparison Operators:

Comparisons yield boolean values: **True or False.**

Comparisons can be chained arbitrarily.

For example:

x < y <= z is equivalent to x < y and y <= z,

except that y is evaluated only once. (but in both cases z is not evaluated at all when x < y is found to be false).

Example:

```
x=5

y=10

z=15

 if (x < y or y < z) and z < x:

print("This will be printed as expected"
```

For Loops in Python

Python For loop is used for sequential traversal i.e. it is used for iterating over an iterable like String, Tuple, List, Set, or Dictionary.

Note: In Python, for loops only implement the collection-based iteration.

For Loops Syntax

```
for var in iterable:
   # statements
```

Here the iterable is a collection of objects like lists, and tuples. The indented statements inside the for loops are executed once for each item in an iterable. The variable var takes the value of the next item of the iterable each time through the loop

Python For Loop with List

This code uses a for loop to iterate over a list of strings, printing each item in the list on a new line. The loop assigns each item to the variable I and continues until all items in the list have been processed.

```
# Python program to illustrate# Iterating over a list

l = ["geeks", "for", "geeks"]

for i in l:

        print(i)
```

Python For Loop in Python Dictionary

This code uses a for loop to iterate over a dictionary and print each key-value pair on a new line. The loop assigns each key to the variable i and uses string formatting to print the key and its corresponding value.

```
# Iterating over dictionary

print("Dictionary Iteration")

d = dict()

d['xyz'] = 123

d['abc'] = 345

for i in d:

        print( (i, d[i]))
```

Python For Loop in Python String

This code uses a for loop to iterate over a string and print each character on a new line. The loop assigns each character to the variable i and continues until all characters in the string have been processed.

```
# Iterating over a String

print("String Iteration")

s = "Geeks"

for i in s:

   print(i)
```

Python For Loop with a step size

This code uses a for loop in conjunction with the range() function to generate a sequence of numbers starting from 0, up to (but not including) 10, and with a step size of 2. For each number in the sequence, the loop prints its value using the print() function. The output will show the numbers 0, 2, 4, 6, and 8.

```
for i in range(0, 10, 2):

   print(i)
```

Python For Loop inside a For Loop

This code uses nested for loops to iterate over two ranges of numbers (1 to 3 inclusive) and prints the value of i and j for each combination of the two loops. The inner loop is executed for each value of i in the outer loop. The output of this code will print the numbers from 1 to 3 three times, as each value of i is combined with each value of j.

```
for i in range(1, 4):

   for j in range(1, 4):

print(i, j)
```

Python For Loop with Tuple

This code iterates over a tuple of tuples using a for loop with tuple unpacking. In each iteration, the values from the inner tuple are assigned to variables a and b, respectively, and then printed to the console using the print() function. The output will show each pair of values from the inner tuples.

```
t = ((1, 2), (3, 4), (5, 6))
for a, b in t:
	print(a, b)
```

Else With For loop in Python

Python also allows us to use the else condition for loops. The else block just after for/while is executed only when the loop is NOT terminated by a break statement.

```
for i in range(1, 4):

   print(i)

else:  # Executed because no break in for

print("No Break\n")
```

Python While Loop

It is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.

Syntax:

```
while expression:

   statement(s)
```

While loop falls under the category of **indefinite iteration**. Indefinite iteration means that the number of times the loop is executed isn't specified explicitly in advance.

Statements represent all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements. When a while loop is executed, expr is first evaluated in a Boolean context and if it is true, the loop body is executed. Then the expr is checked again, if it is still true then the body is executed again and this continues until the expression becomes false.

Example 1: Python While Loop

```
# Python program to illustrate# while loop

count = 0

while (count < 3):

    count = count + 1

print("Hello Geek")
```

Example 2: Python while loop with list

```
# checks if list still

# contains any element

a = [1, 2, 3, 4]

 while a:

    print(a.pop())
```

Example 3: Single statement while block

Just like the if block, if the while block consists of a single statement we can declare the entire loop in a single line. If there are multiple statements in the block that makes up the loop body, they can be separated by semicolons (;).

```
# Python program to illustrate

# Single statement while block

count = 0

while (count < 5): count += 1; print("Hello Geek")
```

Loop Control structures in Python

Python break is used to terminate the execution of the loop.

Python break statement Syntax:

```
Loop{
    Condition:
        break
    }
```

break statement in Python is used to bring the control out of the loop when some external condition is triggered. break statement is put inside the loop body (generally after if condition). It terminates the current loop, i.e., the loop in which it appears, and resumes execution at the next statement immediately after the end of that loop. If the break statement is inside a nested loop, the break will terminate the innermost loop.

Example:

```
num = 0

for i in range(10):

num += 1

    if num == 8:

        break

print("The num has value:", num)

print("Out of loop")
```

Python Continue Statement skips the execution of the program block after the continue statement and forces the control to start the next iteration.

Python Continue statement is a loop control statement that forces to execute the next iteration of the loop while skipping the rest of the code inside the loop for the current iteration only, i.e. when the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped for the current iteration and the next iteration of the loop will begin.

Python continue Statement Syntax

```
while True:
    ...
    if x == 10:
        continue
    print(x)
```

Example:

```
i = 0

while i< 10:
    if i == 5:
i += 1
        continue
    print(i)
i += 1
```

The Python pass statement is a null statement. But the difference between pass and comment is that comment is ignored by the interpreter whereas pass is not ignored.

The Syntax of the pass statement

pass

What is pass statement in Python?

When the user does not know what code to write, So user simply places a pass at that line. Sometimes, the pass is used when the user doesn't want any code to execute. So users can simply place a pass where empty code is not allowed, like in loops, function definitions, class definitions, or in if statements. So using a pass statement user avoids this error.

Why Python Needs "pass" Statement?

If we do not use pass or simply enter a comment or a blank here, we will receive an IndentationError error message.

Example:

```
a =10
b =20

if(a<b):
pass
else:
print("b<a")


li =['a', 'b', 'c', 'd']

foriinli:
if(i=='a'):
pass
```

- Chaitanya Shah

```
else:
print(i)
```

Looping Techniques in Python:
Python supports various looping techniques by certain inbuilt functions, in various sequential containers. These methods are primarily very useful in competitive programming and also in various projects which require a specific technique with loops maintaining the overall structure of code. A lot of time and memory space is been saved as there is no need to declare the extra variables which we declare in the traditional approach of loops.

Where they are used?
Different looping techniques are primarily useful in the places where we don't need to actually manipulate the structure and order of the overall containers, rather only print the elements for a single-use instance, no in-place change occurs in the container. This can also be used in instances to save time.

Different looping techniques using Python data structures are:

Way 1: Using enumerate(): enumerate() is used to loop through the containers printing the index number along with the value present in that particular index.

```
# python code to demonstrate working of enumerate()

forkey, value inenumerate(['The', 'Big', 'Bang', 'Theory']):
print(key, value)
```

Way 2: Using zip(): zip() is used to combine 2 similar containers(list-list or dict-dict) printing the values sequentially. The loop exists only till the smaller container ends. A detailed explanation of zip() and enumerate() can be found here.

```
fruits = ["apple", "banana", "cherry"]
colors = ["red", "yellow", "green"]
for fruit, color in zip(fruits, colors):
print(fruit, "is", color)
```

Way 3: Using iteritem():iteritems() is used to loop through the dictionary printing the dictionary key-value pair sequentially which is used before Python 3 version.

Way 4: Using items(): items() performs the similar task on dictionary as iteritems() but have certain disadvantages when compared with iteritems().

- It is very time-consuming. Calling it on large dictionaries consumes quite a lot of time.
- It takes a lot of memory. Sometimes takes double the memory when called on a dictionary.

```
king ={'Akbar': 'The Great', 'Chandragupta': 'The Maurya',
        'Modi': 'The Changer'}
# using items to print the dictionary key-value pair
forkey, value inking.items():
print(key, value)
```

Way 5: Using sorted(): sorted() is used to print the container is sorted order. It doesn't sort the container but just prints the container in sorted order for 1 instance. The use of set() can be combined to remove duplicate occurrences.
Example:

```
lis = [1, 3, 5, 6, 2, 1, 3]

# using sorted() to print the list in sorted order
print("The list in sorted order is : ")
for i in sorted(lis):
print(i, end=" ")

print("\r")
print("The list in sorted order (without duplicates) is : ")
for i in sorted(set(lis)):
print(i, end=" ")
```

Way 6: Using reversed(): reversed() is used to print the values of the container in the reversed order. It does not reflect any changes to the original list
Example 1:

```
# python code to demonstrate working of reversed()
# initializing list
lis = [1, 3, 5, 6, 2, 1, 3]
# using reversed() to print the list in reversed order
print("The list in reversed order is : ")
for i in reversed(lis):
print(i, end=" ")
```

Q.1 Implement a program to display the geometric sequence as given below for a given value n, where n is the number of elements in the sequence.
1, 2, 4, 8, 16, 32, 64, ……, 1024

Sample Input and Output:

Sample Input	Expected Output
5	1, 2, 4, 8, 16
8	1, 2, 4, 8, 16, 32, 64, 128

Code:

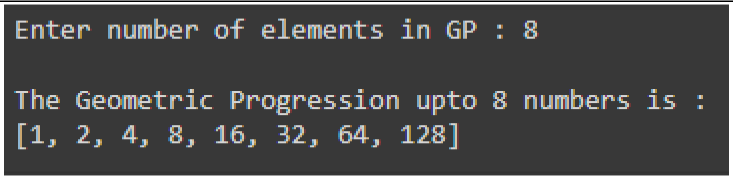
```
import math

n=int(input("Enter number of elements in GP : "))
gp=[]

foriinrange(0,n):
  gp.append(int(math.pow(2,i)))

print("\nThe Geometric Progression upto {} numbers is
:\n{}".format(n,gp))
```

Output:



```
Enter number of elements in GP : 8

The Geometric Progression upto 8 numbers is :
[1, 2, 4, 8, 16, 32, 64, 128]
```

- Chaitanya Shah

Q.2 Implement a program to check whether a given number is a palindrome

Code:

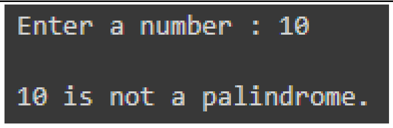
```
n=int(input("Enter a number : "))

temp=n
sum=0

whilen!=0:
  digit=n%10
  sum=sum*10 + digit
  n=n//10

if sum==temp:
  print("\n{} is a palindrome.".format(temp))
else:
  print("\n{} is not a palindrome.".format(temp))
```

Output:



```
Enter a number : 10

10 is not a palindrome.
```

- Chaitanya Shah

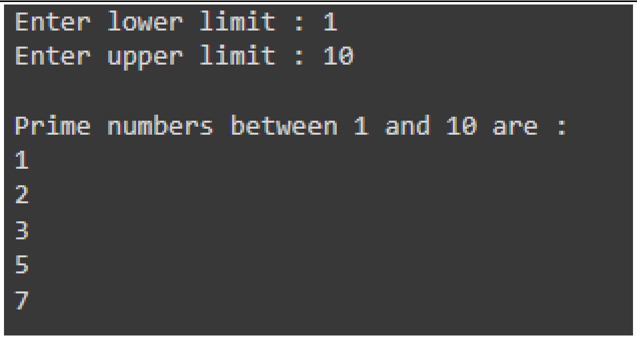
Q.3 Write a program to display all prime numbers within a range

Code:

```
n1=int(input("Enter lower limit : "))
n2=int(input("Enter upper limit : "))

print("\nPrime numbers between {} and {} are : ".format(n1,n2))
for n inrange(n1,n2+1):
  foriinrange(2,int(n**0.5)+1):
    ifn%i==0:
      break
  else:
      print(n,"\t")
```

Output:



```
Enter lower limit : 1
Enter upper limit : 10

Prime numbers between 1 and 10 are :
1
2
3
5
7
```

- Chaitanya Shah

Q.4 Count the total number of digits in a number

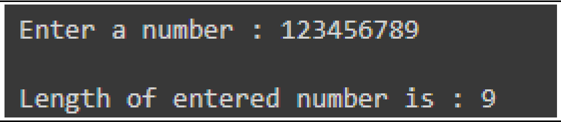
Code:

```
n=int(input("Enter a number : "))

l=0
whilen!=0:
  n//=10
  l+=1

print("\nLength of entered number is :",l)
```

Output:



```
Enter a number : 123456789

Length of entered number is : 9
```

- Chaitanya Shah

Q.5 To display following pattern using nested for loop

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

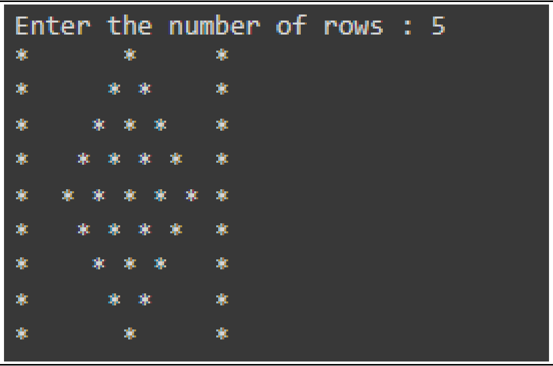
Code:

```
row=int(input("Enter the number of rows : "))

foriinrange(0,row):
  print("*",end="")
  for j inrange(0,row-i+1):
    print(" ",end="")
  for kinrange(0, i+1):
    print("* ",end="")
  for l inrange(row - i - 1):
    print(" ", end="")
  print("*")

foriinrange(0,row-1):
  print("*",end="")
  for j inrange(0,i+3):
    print(" ",end="")
  for kinrange(0, row-i-1):
    print("* ",end="")
  for l inrange(row - i - 2,row-1):
    print(" ", end="")
  print("*")
```

Output:



```
Enter the number of rows : 5
*      *     *
*     * *    *
*    * * *   *
*   * * * *  *
*  * * * * * *
*   * * * *  *
*    * * *   *
*     * *    *
*      *     *
```

- Chaitanya Shah

Lab Outcome:

Through the implementation of this experiment, we have understood how the python make decision using statement like if, if-else, nested if-else and statements to execute some functionality through loop and iterative statements.