# 17th_September_Python_Assignment

September 23, 2023

[1]:
```
#Write a Python program to print numbers from 1 to 10 using a for loop.
for num in range(1, 11):
    print(num)


print(list(range(1, 11)))
```

```
1
2
3
4
5
6
7
8
9
10
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[2]:
```
#Explain the difference between a for loop and a while loop in Python.
(1)For loop

For loop is used to iterate over a sequence of items.

For loops are designed for iterating over a sequence of items. Eg. list, tuple,␣
 ↪etc.

For loop require a sequence to iterate over.

For loop is typically used for iterating over a fixed sequence of items

For loop is more efficient than a while loop when iterating over sequences,␣
 ↪since the number of iterations is predetermined and the loop can be␣
 ↪optimized accordingly.

(2)While loop:
```

```
While loop is used to repeatedly execute a block of statements while a␣
 ↪condition is true.

While loop is used when the number of iterations is not known in advance or␣
 ↪when we want to repeat a block of code until a certain condition is met.

While the loop requires an initial condition that is tested at the beginning of␣
 ↪the loop.

While loop is used for more complex control flow situations.

While a loop may be more efficient in certain situations where the condition␣
 ↪being tested can be evaluated quickly.
```

[2]:
```
#Write a Python program to calculate the sum of all numbers from 1 to 100 using␣
 ↪a for loop.
n = input("Enter value of n: ")
try:
    n = int(n)
    total_sum = 0
    # sum of n numbers in python using for loop
    for i in range(1, n+1):
        total_sum = total_sum + i
    print("Total sum is: ", total_sum)
except:
    print("Please enter a natural number")
```

```
Enter value of n:  100

Total sum is:  5050
```

[3]:
```
#How do you iterate through a list using a for loop in Python?
list = [1, 3, 5, 7, 9]

# Using for loop
for i in list:
    print(i)
```

```
1
3
5
7
9
```

[4]:
```
#Write a Python program to find the product of all elements in a list using a␣
 ↪for loop.
def multiply_numbers(list):
```

```
    prod = 1
    for i in list:
        prod = prod*i
    return prod
given_list = [2,1,3,7,4,85,3]
print('The list is:',given_list)
print("The product is: ")
print(multiply_numbers(given_list))
```

```
The list is: [2, 1, 3, 7, 4, 85, 3]
The product is:
42840
```

[5]:
```
#Create a Python program that prints all even numbers from 1 to 20 using a for␣
 ↪loop.
start = int(input("Enter the start of range: "))
end = int(input("Enter the end of range: "))

# iterating each number in list
for num in range(start, end + 1):

    # checking condition
    if num % 2 == 0:
        print(num, end=" ")
```

```
Enter the start of range:  1
Enter the end of range:  20
2 4 6 8 10 12 14 16 18 20
```

[6]:
```
#How can you iterate through the characters of a string using a for loop in␣
 ↪Python?
str = "SparkBy"
print("String is:", str)

# Iterate over the string by index
print("Characters of string:")
for i in range(0, len(str)):
        print(i, str[i])
```

```
String is: SparkBy
Characters of string:
0 S
1 p
2 a
3 r
4 k
5 B
```

```
6 y
```

[7]:
```python
#Write a Python program to find the largest number in a list using a for loop.
numbers = [1, 2, 3, 5, 9, 6, 101, 88, 66, 6, 101, 55, -1001]
maxi = numbers[0]

for i in numbers:
    if i > maxi:
        maxi = i

print("Greatest number: ", maxi)
```

```
Greatest number:  101
```

[8]:
```python
#Create a Python program that prints the Fibonacci sequence up to a specified␣
 ↪limit using a for loop.
def fib(n):
    a = 0
    b = 1
    if n == 1:
        print(a)
    else:
        print(a)
        print(b)
        for i in range(2,n):
            c = a + b
            a = b
            b = c
            print(c)
fib(10)
```

```
0
1
1
2
3
5
8
13
21
34
```

[14]:
```python
#Write a Python program to count the number of vowels in a given string using a␣
 ↪for loop.
string = "GeekforGeeks!"
vowels = "aeiouAEIOU"
```

```
count = sum(string.count(vowel) for vowel in vowels)
print(count)
```

```
5
```

[17]:
```
#Create a Python program that generates a multiplication table for a given␣
 ↪number using a for loop.
ourNum = int(input("Enter the number you want to generate a multiplication␣
 ↪table for, then hit the `enter` key: "))
ourRange = range(1,11)
for x in ourRange:
    result = ourNum * x
    print(ourNum," * ",x," = ",result)
```

```
Enter the number you want to generate a multiplication table for, then hit the
`enter` key:  12

12  *  1  =  12
12  *  2  =  24
12  *  3  =  36
12  *  4  =  48
12  *  5  =  60
12  *  6  =  72
12  *  7  =  84
12  *  8  =  96
12  *  9  =  108
12  *  10  =  120
```

[18]:
```
#Write a Python program to reverse a list using a for loop.
original_list = [1, 2, 3, 4, 5]
print("List before reverse : ",original_list)
reversed_list = []
for value in original_list:
  reversed_list = [value] + reversed_list
print("List after reverse : ", reversed_list)
```

```
List before reverse :  [1, 2, 3, 4, 5]
List after reverse :  [5, 4, 3, 2, 1]
```

[5]:
```
#Write a Python program to find the common elements between two lists using a␣
 ↪for loop.
a=[2,3,4,5]
b=[3,5,7,9]
def common(a,b):
    c = [value for value in a if value in b]
    return c
d=common(a,b)
print(d)
```

```
[3, 5]
```

[6]:
```
#Explain how to use a for loop to iterate through the keys and values of a␣
 ↪dictionary in Python.
states_tz_dict = {
    'Florida': 'EST and CST',
    'Hawaii': 'HST',
    'Arizona': 'DST',
    'Colorado': 'MST',
    'Idaho': 'MST and PST',
    'Texas': 'CST and MST',
    'Washington': 'PST',
    'Wisconsin': 'CST'
}

for k in states_tz_dict.keys():
    print(k)
```

```
Florida
Hawaii
Arizona
Colorado
Idaho
Texas
Washington
Wisconsin
```

[7]:
```
#Write a Python program to find the GCD (Greatest Common Divisor) of two␣
 ↪numbers using a for loop.
x = 50
y = 100
if x > y:
  x, y = y, x
for i in range(1,x+1):
  if x%i == 0 and y%i == 0:
    gcd = i

print("GCD of", x, "and", y, "is:", gcd)
```

```
GCD of 50 and 100 is: 50
```

[13]:
```
#Create a Python program that checks if a string is a palindrome using a for␣
 ↪loop.
def isPalindrome(s):
    return s == s[::-1]
```

```
# Driver code
s = "malayalam"
ans = isPalindrome(s)

if ans:
    print("Yes")
else:
    print("No")
```

```
Yes
```

[15]:
```
#Write a Python program to remove duplicates from a list using a for loop.
def Remove(duplicate):
    final_list = []
    for num in duplicate:
        if num not in final_list:
            final_list.append(num)
    return final_list

# Driver Code
duplicate = [2, 4, 10, 20, 5, 2, 20, 4]
print(Remove(duplicate))
```

```
[2, 4, 10, 20, 5]
```

[17]:
```
#Create a Python program that counts the number of words in a sentence using a␣
 ↪for loop.
str1 = input("Please Enter your Own String : ")
total = 1

for i in range(len(str1)):
    if(str1[i] == ' ' or str1 == '\n' or str1 == '\t'):
        total = total + 1

print("Total Number of Words in this String = ", total)
```

```
Please Enter your Own String :  I am Abhishek
Total Number of Words in this String =  3
```

[14]:
```
#Write a Python program to find the sum of all odd numbers from 1 to 50 using a␣
 ↪for loop.
n = int(input("Enter the end number : "))
oddsum = []
for i in range(1, n+1):
  if i % 2 != 0:
    oddsum.append(i)
print(sum(oddsum))
```

```
Enter the end number :  50
```

```
625
```

[2]:
```
#Create a Python program that calculates the square root of a number using a␣
 ↪for loop.
import math
# Iterate through numbers from 0 to 29 and print their square roots
for i in range(30):
    # Use the format() method to insert the values of i and its square root␣
 ↪into the string
    print(" Square root of a number {0} = {1} ".format( i, math.sqrt(i)))
```

```
 Square root of a number 0 = 0.0
 Square root of a number 1 = 1.0
 Square root of a number 2 = 1.4142135623730951
 Square root of a number 3 = 1.7320508075688772
 Square root of a number 4 = 2.0
 Square root of a number 5 = 2.23606797749979
 Square root of a number 6 = 2.449489742783178
 Square root of a number 7 = 2.6457513110645907
 Square root of a number 8 = 2.8284271247461903
 Square root of a number 9 = 3.0
 Square root of a number 10 = 3.1622776601683795
 Square root of a number 11 = 3.3166247903554
 Square root of a number 12 = 3.4641016151377544
 Square root of a number 13 = 3.605551275463989
 Square root of a number 14 = 3.7416573867739413
 Square root of a number 15 = 3.872983346207417
 Square root of a number 16 = 4.0
 Square root of a number 17 = 4.123105625617661
 Square root of a number 18 = 4.242640687119285
 Square root of a number 19 = 4.358898943540674
 Square root of a number 20 = 4.47213595499958
 Square root of a number 21 = 4.58257569495584
 Square root of a number 22 = 4.69041575982343
 Square root of a number 23 = 4.795831523312719
 Square root of a number 24 = 4.898979485566356
 Square root of a number 25 = 5.0
 Square root of a number 26 = 5.0990195135927845
 Square root of a number 27 = 5.196152422706632
 Square root of a number 28 = 5.291502622129181
 Square root of a number 29 = 5.385164807134504
```

[19]:
```
#Write a Python program to find the LCM (Least Common Multiple) of two numbers␣
 ↪using a for loop.
num1 = 12
num2 = 14
```

```
for i in range(max(num1, num2), 1 + (num1 * num2)):
    if i % num1 == i % num2 == 0:
        lcm = i
        break
print("LCM of", num1, "and", num2, "is", lcm)
```

```
LCM of 12 and 14 is 84
```

[3]:
```
#Write a Python program to check if a number is positive, negative, or zero␣
 ↪using an if-else statement.
num = float(input("Enter a number: "))
if num > 0:
   print("Positive number")
elif num == 0:
   print("Zero")
else:
   print("Negative number")
```

```
Enter a number:  -23
```

```
Negative number
```

[7]:
```
#Create a Python program that checks if a given number is even or odd using an␣
 ↪if-else statement.
num = int (input ("Enter any number to test whether it is odd or even: "))

if (num % 2) == 0:

              print ("The number is even")

else:

              print ("The provided number is odd")
```

```
Enter any number to test whether it is odd or even:  45
```

```
The provided number is odd
```

[8]:
```
#How can you use nested if-else statements in Python, and provide an example?
# Simpler example of a nested if-else statement
num = 10

if num > 0:
    if num % 2 == 0:
        print("The number is positive and even.")
    else:
        print("The number is positive but odd.")
else:
```

```
    print("The number is not positive.")
```

```
The number is positive and even.
```

[9]:
```
#Write a Python program to determine the largest of three numbers using if-else.
num1 = 10
num2 = 14
num3 = 12

# uncomment following lines to take three numbers from user
#num1 = float(input("Enter first number: "))
#num2 = float(input("Enter second number: "))
#num3 = float(input("Enter third number: "))

if (num1 >= num2) and (num1 >= num3):
   largest = num1
elif (num2 >= num1) and (num2 >= num3):
   largest = num2
else:
   largest = num3

print("The largest number is", largest)
```

```
The largest number is 14
```

[10]:
```
#Write a Python program that calculates the absolute value of a number using␣
 ↪if-else.

# input numbers(integers)
num_1 = 4
num_2 = -6
num_3 = 0
num_4 = -875
# calculating absolute values of input integers
print("absolute value of 4 = ", abs(num_1))
print("absolute value of -6 = ", abs(num_2))
print("absolute value of 0 = ", abs(num_3))
print("absolute value of -875 = ", abs(num_4))
```

```
absolute value of 4 =  4
absolute value of -6 =  6
absolute value of 0 =  0
absolute value of -875 =  875
```

[11]:
```
#Create a Python program that checks if a given character is a vowel or␣
 ↪consonant using if-else.
from operator import countOf
```

```
# Function to check whether the given character is a vowel or not
def isVowel(char):

    # A string of vowels
    vowels = "aeiouAEIOU"

    # Checking whether the given character is a vowel or a consonant
    if countOf(vowels, char) > 0:
        print(f"The character '{char}' is a vowel!")
    else:
        print(f"The character '{char}' is a consonant!")

# Get an input character from the user
character = input("Enter a character: ")

# Calling the function
isVowel(character)
```

```
Enter a character:  E
```

```
The character 'E' is a vowel!
```

[13]:
```
#Write a Python program to determine if a user is eligible to vote based on␣
 ↪their age using if-else.
age = int(input("Enter age : "))
if age >= 18:
    print("Eligible for Voting!")
else:
    print("Not Eligible for Voting!")
```

```
Enter age :  23
```

```
Eligible for Voting!
```

[14]:
```
#Create a Python program that calculates the discount amount based on the␣
 ↪purchase amount using if-else.

amt = int(input("Enter Purchase Amount: "))

# checking conditions and calculating discount
if(amt>0):
    if amt<=5000:
       disc = amt*0.05
    elif amt<=15000:
        disc=amt*0.12
    elif amt<=25000:
        disc=0.2 * amt
```

```
    else:
         disc=0.3 * amt

    print("Discount : ",disc)
    print("Net Pay  : ",amt-disc)
else:
    print("Invalid Amount")
```

```
Enter Purchase Amount:  100000

Discount :  30000.0
Net Pay  :  70000.0
```

[15]:
```
#Write a Python program to check if a number is within a specified range using␣
 ↪if-else.
lower_bound = 10
upper_bound = 20

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

if lower_bound <= number <= upper_bound:
    print("The number is within the specified range.")
else:
    print("The number is outside the specified range.")
```

```
Enter a number:  21

The number is outside the specified range.
```

[16]:
```
#Create a Python program that determines the grade of a student based on their␣
 ↪score using if-else.
print("Enter Marks Obtained in 5 Subjects: ")
markOne = int(input())
markTwo = int(input())
markThree = int(input())
markFour = int(input())
markFive = int(input())

tot = markOne+markTwo+markThree+markFour+markFive
avg = tot/5

if avg>=91 and avg<=100:
    print("Your Grade is A1")
elif avg>=81 and avg<91:
    print("Your Grade is A2")
elif avg>=71 and avg<81:
    print("Your Grade is B1")
elif avg>=61 and avg<71:
```

```
    print("Your Grade is B2")
elif avg>=51 and avg<61:
    print("Your Grade is C1")
elif avg>=41 and avg<51:
    print("Your Grade is C2")
elif avg>=33 and avg<41:
    print("Your Grade is D")
elif avg>=21 and avg<33:
    print("Your Grade is E1")
elif avg>=0 and avg<21:
    print("Your Grade is E2")
else:
    print("Invalid Input!")
```

```
Enter Marks Obtained in 5 Subjects:

 12
 12
 13
 56
 67

Your Grade is E1
```

[17]:
```
#Write a Python program to check if a string is empty or not using if-else.
# Using len() To Check if a String is Empty
string = ''

if len(string) == 0:
    print("Empty string!")
else:
    print("Not empty string!")

# Returns
# Empty string!
```

```
Empty string!
```

[20]:
```
#Create a Python program that identifies the type of a triangle (e.g.,␣
 ↪equilateral, isosceles, or scalene) based on input values using if-else.
x = int(input("x: "))
y = int(input("y: "))
z = int(input("z: "))

if x == y == z:
        print("Equilateral triangle")
elif x==y or y==z or z==x:
        print("isosceles triangle")
```

```
else:
        print("Scalene triangle")
```

```
x:  89
y:  81
z:  89
```

```
isosceles triangle
```

[21]:
```
#Write a Python program to determine the day of the week based on a␣
 ↪user-provided number using if-else.
weekday = int(input("Enter weekday number (1-7) : "))

if weekday == 1 :
    print("\nMonday");

elif weekday == 2 :
    print("\nTuesday")

elif(weekday == 3) :
    print("\nWednesday")

elif(weekday == 4) :
    print("\nThursday")

elif(weekday == 5) :
    print("\nFriday")

elif(weekday == 6) :
    print("\nSaturday")

elif (weekday == 7) :
    print("\nSunday")

else :
    print("\nPlease enter any weekday number (1-7)")
```

```
Enter weekday number (1-7) :  3
```

```
Wednesday
```

[ ]:
```
#Create a Python program that checks if a given year is a leap year using both␣
 ↪if-else and a function.
def check_leap(year):
    return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

year = int(input("Enter a year: "))
```

```
print(f"{year} is leap year" if check_leap(year) else f"{year} is not leap␣
 ↪year")
```

[ ]:
```
#Create a Python program that determines the eligibility of a person for a␣
 ↪senior citizen discount based on age using if-else.
age=71
state=Florida
if age >= 70 and state == "Florida":
    print("You qualify for a discount!")
else:
    print("You do not qualify for a discount!")
```

[ ]:
```
#Write a Python program to categorize a given character as uppercase,␣
 ↪lowercase, or neither using if-else
def check(ch):

    if (ch >= 'A' and ch <= 'Z'):
        print(ch,"is an UpperCase character");

    elif (ch >= 'a' and ch <= 'z'):
        print(ch,"is an LowerCase character");
    else:
        print(ch,"is not an alphabetic character");

# Driver Code

# Get the character
ch = 'A';

# Check the character
check(ch);

# Get the character
ch = 'a';

# Check the character
check(ch);

# Get the character
ch = '0';

# Check the character
check(ch);
```

[ ]:
```
#Write a Python program to determine the roots of a quadratic equation using␣
 ↪if-else.
import cmath
a = float(input('Enter a: '))
b = float(input('Enter b: '))
c = float(input('Enter c: '))

# calculate the discriminant
d = (b**2) - (4*a*c)

# find two solutions
sol1 = (-b-cmath.sqrt(d))/(2*a)
sol2 = (-b+cmath.sqrt(d))/(2*a)
print('The solution are {0} and {1}'.format(sol1,sol2))
```

[ ]:
```
#Create a Python program that checks if a given year is a century year or not␣
 ↪using if-else.
def check_leap_year(year):
    """
    Check if the given year is a leap year or not.
    """
    if year % 4 == 0:
        if year % 100 == 0:
            if year % 400 == 0:
                print(f"{year} is a leap year")
            else:
                print(f"{year} is not a leap year")
        else:
            print(f"{year} is a leap year")
    else:
        print(f"{year} is not a leap year")


check_leap_year(2000)  # This 2000 is a leap year
check_leap_year(2016)  # This 2016 is a leap year
check_leap_year(2100)  # This 2100 is not a leap year

Output:
```

[ ]:
```
#Write a Python program to determine if a given number is a perfect square␣
 ↪using if-else.
import math
num = 25
sqrt_num = math.sqrt(num)
if sqrt_num.is_integer():
    print("The number is a perfect square")
else:
```

```
    print("The number is not a perfect square")
```

[ ]:
```
#Explain the purpose of the "continue" and "break" statements within if-else␣
 ↪loops.

1.Break

The break statement is used to terminate the loop immediately.

break keyword is used to indicate break statements in java programming.

We can use a break with the switch statement.

2.Continue

The continue statement is used to skip the current iteration of the loop.

continue keyword is used to indicate continue statement in java programming.

We can not use a continue with the switch statement.
```

[40]:
```
#Create a Python program that calculates the BMI (Body Mass Index) of a person␣
 ↪based on their weight and height using if-else.
weight=float(input())
height=float(input())

bmi= weight/(height)**2

if bmi<18.5:
    print("Underweight")
elif bmi>=18.5 and bmi<25:
    print("Normal")
elif bmi>=25 and bmi<30:
    print("Overweight")
else:
    print("Obesity")
```

[41]:
```
#How can you use the "filter()" function with if-else statements to filter␣
 ↪elements from a list?
scores = [70, 60, 80, 90, 50]

filtered = []

for score in scores:
    if score >= 70:
        filtered.append(score)
```

```
print(filtered)
```

[42]:
```
#Write a Python program to determine if a given number is prime or not using␣
 ↪if-else.
flag = False

if num == 1:
    print(num, "is not a prime number")
elif num > 1:
    # check for factors
    for i in range(2, num):
        if (num % i) == 0:
            # if factor is found, set flag to True
            flag = True
            # break out of loop
            break

    # check if flag is True
    if flag:
        print(num, "is not a prime number")
    else:
        print(num, "is a prime number")
```

[43]:
```
#Explain the purpose of the `map()` function in Python and provide an example␣
 ↪of how it can be used to apply a function to each element of an iterable.
map() function returns a map object(which is an iterator) of the results after␣
 ↪applying the given function to each item of a given iterable (list, tuple␣
 ↪etc.)

# Python program to demonstrate working
# of map.

# Return double of n
def addition(n):
    return n + n

# We double all numbers using map()
numbers = (1, 2, 3, 4)
result = map(addition, numbers)
print(list(result))
```

[1]:
```
#Write a Python program that uses the `map()` function to square each element␣
 ↪of a list of numbers.
def square_num(n):
  return n * n
nums = [4, 5, 2, 9]
print("Original List: ",nums)
```

```
result = map(square_num, nums)
print("Square the elements of the said list using map():")
print(list(result))
```

```
Original List:  [4, 5, 2, 9]
Square the elements of the said list using map():
[16, 25, 4, 81]
```

[45]:
```
#How does the `map()` function differ from a list comprehension in Python, and␣
 ↪when would you choose one over the other?
List comprehension has a simpler configuration than the map function.
List comprehension can be used together with if condition as replacement of␣
 ↪filter method. Map function has no such functionality. However, we can feed␣
 ↪the map function output to the filter function.
List comprehension returns a list, whereas the map function returns an object␣
 ↪of Iterable.
List comprehension execution is faster than that of map function when the␣
 ↪formula expression is huge and complex.
Map function is faster than list comprehension when the formula is already␣
 ↪defined as a function earlier. So, that map function is used without lambda␣
 ↪expression.
```

[5]:
```
#Create a Python program that uses the `map()` function to convert a list of␣
 ↪names to uppercase.
def change_cases(s):
  return str(s).upper()

chars = {'a', 'b', 'E', 'f', 'a', 'i', 'o', 'U', 'a'}
print("Original Characters:\n",chrars)

result = map(change_cases, chars)
print("\nAfter converting above characters in upper and eliminating duplicate␣
 ↪letters:")
print(set(result))
```

```
Original Characters:
 {'b', 'f', 'U', 'i', 'a', 'o', 'E'}

After converting above characters in upper and eliminating duplicate letters:
{'O', 'A', 'U', 'F', 'I', 'B', 'E'}
```

[6]:
```
#Write a Python program that uses the `map()` function to calculate the length␣
 ↪of each word in a list of strings.
test_string = "Geeksforgeeks is best Computer Science Portal"

# Printing original string
print("The original string is : " + test_string)
```

```
# Words lengths in String
# using split() method
res = list(map(len, test_string.split()))

# Printing result
print("The list of words lengths is : " + str(res))
```

```
The original string is : Geeksforgeeks is best Computer Science Portal
The list of words lengths is : [13, 2, 4, 8, 7, 6]
```

[12]:
```
#Create a Python program that uses `map()` to convert a list of temperatures␣
 ↪from Celsius to Fahrenheit.
places = [('Nashua',32),("Boston",12),("Los Angelos",44), ("Miami",29)]

newplaces = list(map(lambda c: (c[0],  (9/5) * c[1]+ 32), places))
print(newplaces)
```

```
[('Nashua', 89.6), ('Boston', 53.6), ('Los Angelos', 111.2), ('Miami', 84.2)]
```

[13]:
```
#Write a Python program that uses the `map()` function to round each element of␣
 ↪a list of floating-point numbers to the nearest integer.
list_of_floats = [1.23, 3.45, 5.67]

result = [int(item) for item in list_of_floats]
print(result)
```

```
[1, 3, 5]
```

[51]:
```
#What is the `reduce()` function in Python, and what module should you import␣
 ↪to use it? Provide an example of its basic usage.
The reduce(fun,seq) function is used to apply a particular function passed in␣
 ↪its argument to all of the list elements mentioned in the sequence passed␣
 ↪along.This function is defined in "functools" module.

import functools

# initializing list
lis = [1, 3, 5, 6, 2]

# using reduce to compute sum of list
print("The sum of the list elements is : ", end="")
print(functools.reduce(lambda a, b: a+b, lis))

# using reduce to compute maximum element from list
print("The maximum element of the list is : ", end="")
print(functools.reduce(lambda a, b: a if a > b else b, lis))
```

[1]:
```
#Write a Python program that uses the `reduce()` function to find the product␣
 ↪of all elements in a list.
def multiplyList(myList):

    # Multiply elements one by one
    result = 1
    for x in myList:
        result = result * x
    return result


# Driver code
list1 = [1, 2, 3]
list2 = [3, 2, 4]
print(multiplyList(list1))
print(multiplyList(list2))
```

```
6
24
```

[2]:
```
#Create a Python program that uses `reduce()` to find the maximum element in a␣
 ↪list of numbers
from functools import reduce
lst = [20, 10, 20, 4, 100]
largest_elem = reduce(max, lst)
print(largest_elem)
```

```
100
```

[3]:
```
#How can you use the `reduce()` function to concatenate a list of strings into␣
 ↪a single string?
from functools import reduce

# Method 3: Using the reduce() function
fruits = ['apple', 'banana', 'orange', 'grape']

def concatenate_strings(str1, str2):
    return str1 + ', ' + str2

result = reduce(concatenate_strings, fruits)

print(result)
```

```
apple, banana, orange, grape
```

[5]:
```
#Write a Python program that calculates the factorial of a number using the␣
 ↪`reduce()` function.
```

```
import functools

def factorial(n):
    if n == 0:
        return 1
    else:
        return functools.reduce(lambda x,y: x*y, range(1,n+1))

print(factorial(3))
```

```
6
```

[12]:
```
#Create a Python program that uses `reduce()` to find the GCD (Greatest Common Divisor) of a list of numbers.

from functools import reduce
def gcd(a,b):
    if a==0:
        return b
    else:
        return gcd(b%a,a)
A = [12, 24, 27, 30, 36]
gcdp = reduce(lambda x,y:gcd(x,y),A)
print(gcdp)
```

```
3
```

[15]:
```
#Write a Python program that uses the `reduce()` function to find the sum of the digits of a given number.
def getSum(n):

    sum = 0
    while (n != 0):

        sum = sum + (n % 10)
        n = n//10

    return sum

n = 569
print(getSum(12))
```

```
3
```

[1]:
```
#Explain the purpose of the `filter()` function in Python and provide an example of how it can be used to filter elements from an iterable.
def check_even(number):
```

```
    if number % 2 == 0:
          return True

    return False


numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# if an element passed to check_even() returns True, select it
even_numbers_iterator = filter(check_even, numbers)

# converting to list
even_numbers = list(even_numbers_iterator)

print(even_numbers)
```

```
[2, 4, 6, 8, 10]
```

[2]:
```
#Write a Python program that uses the `filter()` function to select even␣
 ↪numbers from a list of integers.
def check_even(number):
    if number % 2 == 0:
          return True

    return False


numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# if an element passed to check_even() returns True, select it
even_numbers_iterator = filter(check_even, numbers)

# converting to list
even_numbers = list(even_numbers_iterator)

print(even_numbers)
```

```
[2, 4, 6, 8, 10]
```

[3]:
```
#Create a Python program that uses the `filter()` function to select names that␣
 ↪start with a specific letter from a list of strings.
test_list = ['sapple', 'orange', 'smango', 'grape']

# initializing start Prefix
start_letter = 's'

# printing original list
```

```
print("The original list : " + str(test_list))

# using list comprehension + startswith()
# Prefix Separation
with_s = [x for x in test_list if x.startswith(start_letter)]

# print result
print("The list with prefix s : " + str(with_s))
```

```
The original list : ['sapple', 'orange', 'smango', 'grape']
The list with prefix s : ['sapple', 'smango']
```

[5]:
```
#Write a Python program that uses the `filter()` function to select prime␣
 ↪numbers from a list of integers.
import math

def return_primes(arr):
    return list(filter(lambda x : is_prime(x), arr))

def is_prime(n):
    for i in range(2, int(math.sqrt(n))):
        if n % i == 0:
            return False
    return True

print(return_primes([10,21,3,8,9,11,44,62,100,19]))
```

```
[3, 8, 9, 11, 19]
```

[6]:
```
#How can you use the `filter()` function to remove None values from a list in␣
 ↪Python?
my_list = [1, None, 3, None, 8, None]

new_list = list(filter(lambda x: x is not None, my_list))

print(new_list)
```

```
[1, 3, 8]
```

[8]:
```
#Create a Python program that uses `filter()` to select words longer than a␣
 ↪certain length from a list of strings.
data = ['hello', 'communication', 'be', 'dog', 'test']
filtered_list = [x for x in data if len(x) > 4 and len(x) < 8]
print(filtered_list)
```

```
['hello']
```

[9]:
```
#Write a Python program that uses the `filter()` function to select elements␣
 ↪greater than a specified threshold from a list of values.
test_list = ['gfg', 'is', 'best', 'for', 'geeks']

# Printing original list
print("The original list : " + str(test_list))

# Initialize Threshold
thres = 4

# Filter above Threshold size Strings
# using list comprehension + len()
res = [ele for ele in test_list if len(ele) >= thres]

# Printing result
print("The above Threshold size strings are : " + str(res))
```

```
The original list : ['gfg', 'is', 'best', 'for', 'geeks']
The above Threshold size strings are : ['best', 'geeks']
```

[66]:
```
#Explain the concept of recursion in Python. How does it differ from iteration?
Recursion is when a function calls itself within its code, thus repeatedly␣
 ↪executing the instructions present inside it.
Iteration is when a loop repeatedly executes the set of instructions like "for"␣
 ↪loops and "while" loops.

(1)Recursion:
    Recursion uses the selection structure.

    Infinite recursion occurs if the step in recursion doesn't reduce the␣
 ↪problem to a smaller problem. It also becomes infinite recursion if it␣
 ↪doesn't convert on a specific condition. This specific condition is known as␣
 ↪the base case.

    The system crashes when infinite recursion is encountered.

  (2)Iteration:
    Iteration uses the repetition structure.

    An infinite loop occurs when the condition in the loop doesn't become False␣
 ↪ever.

    Iteration uses the CPU cycles again and again when an infinite loop occurs.
```

[10]:
```
#Write a Python program to calculate the factorial of a number using recursion.
def recur_factorial(n):
```

```
    if n == 1:
        return n
    else:
        return n*recur_factorial(n-1)

num = 7

# check if the number is negative
if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
    print("The factorial of", num, "is", recur_factorial(num))
```

```
The factorial of 7 is 5040
```

[11]:
```
#Create a recursive Python function to find the nth Fibonacci number.
def fibonacci(n):
    a = 0
    b = 1
    if n < 0:
        print("Incorrect input")
    elif n == 0:
        return a
    elif n == 1:
        return b
    else:
        for i in range(2, n+1):
            c = a + b
            a = b
            b = c
        return b
print(fibonacci(9))
```

```
34
```

[1]:
```
#How can you prevent a recursive function from running indefinitely, causing a␣
 ↪stack overflow error?
import sys


#setting recursion limit
sys.setrecursionlimit(10**6)


def fact(n):
```

```
    if(n == 0):
        return 1

    return n * fact(n - 1)

if __name__ == '__main__':

    # taking input
    f = int(input('Enter the number: \n'))

    print(fact(f))
```

```
Enter the number:
 23
```

```
25852016738884976640000
```

[2]:
```
#Create a recursive Python function to find the greatest common divisor (GCD)␣
 ↪of two numbers using the Euclidean algorithm.
def gcd(a, b):

    # Everything divides 0
    if (a == 0):
        return b
    if (b == 0):
        return a

    # base case
    if (a == b):
        return a

    # a is greater
    if (a > b):
        return gcd(a-b, b)
    return gcd(a, b-a)

# Driver program to test above function
a = 98
b = 56
if(gcd(a, b)):
    print('GCD of', a, 'and', b, 'is', gcd(a, b))
else:
    print('not found')
```

```
GCD of 98 and 56 is 14
```

[3]:
```
#Write a recursive Python function to reverse a string.
def reverse(s):
    if len(s) == 0:
        return s
    else:
        return reverse(s[1:]) + s[0]


s = "Geeksforgeeks"

print("The original string is : ", end="")
print(s)

print("The reversed string(using recursion) is : ", end="")
print(reverse(s))
```

```
The original string is : Geeksforgeeks
The reversed string(using recursion) is : skeegrofskeeG
```

[4]:
```
#Create a recursive Python function to calculate the power of a number (x^n).
def power(N, P):


    if P == 0:
        return 1

    # Recurrence relation
    return (N*power(N, P-1))


# Driver code
if __name__ == '__main__':
    N = 5
    P = 2

    print(power(N, P))
```

```
25
```

[5]:
```
#Write a recursive Python function to find all permutations of a given string.
import itertools

if __name__ == '__main__':
    s = 'ABC'

    nums = list(s)
    permutations = list(itertools.permutations(nums))
```

```
    # Output: ['ABC', 'ACB', 'BAC', 'BCA', 'CAB', 'CBA']
    print([''.join(permutation) for permutation in permutations])
```

```
['ABC', 'ACB', 'BAC', 'BCA', 'CAB', 'CBA']
```

[7]:
```
#Write a recursive Python function to check if a string is a palindrome.
def is_palindrome(s):
    if len(s) < 1:
        return True
    else:
        if s[0] == s[-1]:
            return is_palindrome(s[1:-1])
        else:
            return False
a=str(input("Enter string:"))
if(is_palindrome(a)==True):
    print("String is a palindrome!")
else:
    print("String isn't a palindrome!")
```

```
Enter string: string
String isn't a palindrome!
```

[8]:
```
#Create a recursive Python function to generate all possible combinations of a␣
 ↪list of elements.
def combinations(l):
    if l:
      result = combinations(l[:-1])
      return result + [c + [l[-1]] for c in result]
    else:
      return [[]]

print(combinations([1,2,3]))
```

```
[[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]]
```

[76]:
```
#What is a function in Python, and why is it used?
Python Functions is a block of statements that return the specific task. The␣
 ↪idea is to put some commonly or repeatedly done tasks together and make a␣
 ↪function so that instead of writing the same code again and again for␣
 ↪different inputs, we can do the function calls to reuse code contained in it␣
 ↪over and over again.

Some Benefits of Using Functions

Increase Code Readability
```

```
Increase Code Reusability
```

[77]:
```
#How do you define a function in Python? Provide an example.

Python Functions is a block of statements that return the specific task. The␣
 ↪idea is to put some commonly or repeatedly done tasks together and make a␣
 ↪function so that instead of writing the same code again and again for␣
 ↪different inputs, we can do the function calls to reuse code contained in it␣
 ↪over and over again.

The syntax to declare a function is:

def function_name(arguments):
    # function body

    return
Here,

def - keyword used to declare a function
function_name - any name given to the function
arguments - any value passed to function
return (optional) - returns value from a function
Let's see an example,

def greet():
    print('Hello World!')
```

[78]:
```
#Explain the difference between a function definition and a function call.
Basic Syntax for Defining a Function in Python
In Python, you define a function with the def keyword, then write the function␣
 ↪identifier (name) followed by parentheses and a colon.

The next thing you have to do is make sure you indent with a tab or 4 spaces,␣
 ↪and then specify what you want the function to do for you.

def functionName():
    # What to make the function do
Basic Examples of a Function in Python
Following the basic syntax above, an example of a basic Python function␣
 ↪printing "Hello World" to the terminal looks like this:

def myfunction():
    print("Hello World")
To call this function, write the name of the function followed by parentheses:
myfunction()
```

[9]:
```
#Write a Python program that defines a function to calculate the sum of two␣
 ↪numbers and then calls the function.
def add_num(a,b):#function for addition
    sum=a+b;
    return sum; #return value
num1=25  #variable declaration
num2=55
print("The sum is",add_num(num1,num2))#call the function
```

```
The sum is 80
```

[80]:
```
#What is a function signature, and what information does it typically include?
A function signature is its declaration, parameters, and return type.

def func(params):

    return modified_params

When you call it is an instance of that function.
var = func(parameters)
```

[10]:
```
#Create a Python function that takes two arguments and returns their product.
def multiply(a, b):
  return a*b

print(multiply(2,5))
```

```
10
```

[82]:
```
#Explain the concepts of formal parameters and actual arguments in Python␣
 ↪functions.
 An argument is a variable (which contains data) or a parameter that is sent to␣
 ↪the function as input. Before getting into argument types, let's get␣
 ↪familiar with words formal and actual arguments.

Formal arguments: When a function is defined it (may) has (have) some␣
 ↪parameters within the parentheses. These parameters, which receive the␣
 ↪values sent from the function call, are called formal arguments.
Actual arguments: The parameters which we use in the function call or the␣
 ↪parameters which we use to send the values/data during the function call are␣
 ↪called actual arguments.
Example: formal and actual function arguments in python (Demo15.py)
def sum(a, b):
   c = a + b            # a and b are formal arguments
   print(c)
# call the function
x = 10
```

```
y = 15
sum(x, y)              # x and y are actual arguments
```

[83]:
```
#Write a Python program that defines a function with default argument values.
The arguments that take default values when no explicit values are supplied to␣
 ↪them from the function call are known as default arguments in Python␣
 ↪functions.

Let us look at a simple python function as an example that uses a default␣
 ↪argument.

def greet(name="world"):
   print("Hello,", name)
greet()
```

[84]:
```
#How do you use keyword arguments in Python function calls? Provide an example.
In keyword arguments, arguments are assigned based on the name of arguments.␣
 ↪For example,

def display_info(first_name, last_name):
    print('First Name:', first_name)
    print('Last Name:', last_name)

display_info(last_name = 'Cartman', first_name = 'Eric')

Here, notice the function call,

display_info(last_name = 'Cartman', first_name = 'Eric')
Here, we have assigned names to arguments during the function call.

Hence, first_name in the function call is assigned to first_name in the␣
 ↪function definition. Similarly, last_name in the function call is assigned␣
 ↪to last_name in the function definition.

In such scenarios, the position of arguments doesn't matter.
```

[11]:
```
#Create a Python function that accepts a variable number of arguments and␣
 ↪calculates their sum.
def summ(num1, *args):
    total = num1
    for num in args:
        total = total + num
    return total

summ(1, 2, 3, 4, 5, 6, 7)
```

[11]: 28

[86]:
```
#What is the purpose of the `*args` and `**kwargs` syntax in function parameter␣
 ↪lists?

(1)The special syntax *args in function definitions in Python is used to pass a␣
 ↪variable number of arguments to a function. It is used to pass a␣
 ↪non-keyworded, variable-length argument list.

def myFun(*argv):
    for arg in argv:
        print(arg)


myFun('Hello', 'Welcome', 'to', 'GeeksforGeeks')

(2)The special syntax **kwargs in function definitions in Python is used to␣
 ↪pass a keyworded, variable-length argument list. We use the name kwargs with␣
 ↪the double star. The reason is that the double star allows us to pass␣
 ↪through keyword arguments (and any number of them).

def myFun(arg1, **kwargs):
    for key, value in kwargs.items():
        print("%s == %s" % (key, value))


# Driver code
myFun("Hi", first='Geeks', mid='for', last='Geeks')
```

[87]:
```
#Describe the role of the `return` statement in Python functions and provide␣
 ↪examples.

A return statement is used to end the execution of the function call and␣
 ↪"returns" the result (value of the expression following the return keyword)␣
 ↪to the caller. The statements after the return statements are not executed.␣
 ↪If the return statement is without any expression, then the special value␣
 ↪None is returned. A return statement is overall used to invoke a function so␣
 ↪that the passed statements can be executed.

def add(a, b):

    # returning sum of a and b
    return a + b

def is_true(a):

    # returning boolean of a
```

```
    return bool(a)

# calling function
res = add(2, 3)
print("Result of add function is {}".format(res))

res = is_true(2<5)
print("\nResult of is_true function is {}".format(res))
```

[88]:
```
#Explain the concept of variable scope in Python, including local and global␣
 ↪variables.
Python Global variables are those which are not defined inside any function and␣
 ↪have a global scope whereas Python local variables are those which are␣
 ↪defined inside a function and their scope is limited to that function only.␣
 ↪In other words, we can say that local variables are accessible only inside␣
 ↪the function in which it was initialized whereas the global variables are␣
 ↪accessible throughout the program and inside every function.

Local variables in Python are those which are initialized inside a function and␣
 ↪belong only to that particular function. It cannot be accessed anywhere␣
 ↪outside the function.
```

[13]:
```
#Write a Python program that demonstrates the use of global variables within␣
 ↪functions.
_my_global = 5

def func1():
    _my_global = 42

def func2():
    print(_my_global)

func1()
func2()
```

```
5
```

[14]:
```
#Create a Python function that calculates the factorial of a number and returns␣
 ↪it.
def recur_factorial(n):
   if n == 1:
       return n
   else:
       return n*recur_factorial(n-1)
num = 5
# check if the number is negative
if num < 0:
```

```
   print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
   print("The factorial of 0 is 1")
else:
   print("The factorial of", num, "is", recur_factorial(num))
```

```
The factorial of 5 is 120
```

[15]:
```
#How can you access variables defined outside a function from within the␣
 ↪function?
x = 10

def showX():
    print("The value of x is", x)

showX()
```

```
The value of x is 10
```

[92]:
```
#What are lambda functions in Python, and when are they typically used?
Python Lambda Functions are anonymous function means that the function is␣
 ↪without a name. As we already know that the def keyword is used to define a␣
 ↪normal function in Python. Similarly, the lambda keyword is used to define␣
 ↪an anonymous function in Python.


Syntax: lambda arguments : expression

This function can have any number of arguments but only one expression, which␣
 ↪is evaluated and returned.
One is free to use lambda functions wherever function objects are required.
```

[16]:
```
#Write a Python program that uses lambda functions to sort a list of tuples␣
 ↪based on the second element.

subject_marks = [('English', 88), ('Science', 90), ('Maths', 97), ('Social␣
 ↪sciences', 82)]
print("Original list of tuples:")
print(subject_marks)
subject_marks.sort(key = lambda x: x[1])
print("\nSorting the List of Tuples:")
print(subject_marks)
```

```
Original list of tuples:
[('English', 88), ('Science', 90), ('Maths', 97), ('Social sciences', 82)]

Sorting the List of Tuples:
```

```
[('Social sciences', 82), ('English', 88), ('Science', 90), ('Maths', 97)]
```

[94]:
```
#Explain the concept of higher-order functions in Python, and provide an␣
 ↪example.

Higher-order functions are functions that take a function as a parameter and/or␣
 ↪return a function as an output.

A few useful higher-order functions are map(), filter(), and reduce(). map()␣
 ↪and filter() are built-in functions, whereas reduce() is contained in␣
 ↪functools() module.

num=[1,2,3,4,5]
square=map(lambda x: x**2 , num)
print (square)
```

[17]:
```
#Create a Python function that takes a list of numbers and a function as␣
 ↪arguments, applying the function to each element in the list.
def addition(n):
    return n + n

# We double all numbers using map()
numbers = (1, 2, 3, 4)
result = map(addition, numbers)
print(list(result))
```

```
[2, 4, 6, 8]
```

[96]:
```
#Describe the role of built-in functions like `len()`, `max()`, and `min()` in␣
 ↪Python.

(1)len()        Returns the length of an object

(2)max()        Returns the largest item in an iterable

(3)min()        Returns the smallest item in an iterable
```

[97]:
```
#Write a Python program that uses the `map()` function to apply a function to␣
 ↪each element of a list.

def addition(n):
    return n + n

# We double all numbers using map()
numbers = (1, 2, 3, 4)
result = map(addition, numbers)
print(list(result))
```

[98]:
```
#How does the `filter()` function work in Python, and when would you use it?
As the name suggests filter extracts each element in the sequence for which the␣
 ↪function returns True


Example:
a = [1,2,3,5,7,9]
b = [2,3,5,6,7,8]
print filter(lambda x: x in a, b)  # prints out [2, 3, 5, 7]
```

[99]:
```
#Create a Python program that uses the `reduce()` function to find the product␣
 ↪of all elements in a list.

import functools

# importing operator for operator functions
import operator

# initializing list
lis = [1, 3, 5, 6, 2]

# using reduce to compute product
# using operator functions
print("The product of list elements is : ", end="")
print(functools.reduce(operator.mul, lis))
```

[100]:
```
#Explain the purpose of docstrings in Python functions and how to write them.

Python documentation string or commonly known as docstring, is a string␣
 ↪literal, and it is used in the class, module, function, or method definition.
 ↪ Docstrings are accessible from the doc attribute (__doc__) for any of the␣
 ↪Python objects and also with the built-in help() function. An object's␣
 ↪docstring is defined by including a string constant as the first statement␣
 ↪in the object's definition.

Docstrings are great for understanding the functionality of the larger part of␣
 ↪the code, i.e., the general purpose of any class, module, or function,␣
 ↪whereas the comments are used for code, statements, and expressions, which␣
 ↪tend to be small. They are descriptive text written by a programmer mainly␣
 ↪for themselves to know what the line of code or expression does and also for␣
 ↪the developer who wishes to contribute to that project. It is an essential␣
 ↪part that documenting your Python code is going to serve well enough for␣
 ↪writing clean code and well-written programs.
```

```
Docstrings help you understand the capabilities of a module or a function. For␣
 ↪example, let's say you installed the scikit-learn library, and you would␣
 ↪like to know all about the sklearn package like description, package␣
 ↪modules, etc., you could simply use the help function to get all the␣
 ↪information.

def square(a):
    '''Returned argument a is squared.'''
    return a**a
```

[13]:
```
#Describe some best practices for naming functions and variables in Python,␣
 ↪including naming conventions and guidelines.
1. Naming Convention for Variables
JavaScript variable names are case-sensitive. Lowercase and uppercase letters␣
 ↪are distinct. For example, you can define three unique variables to store a␣
 ↪dog name, as follows.

var DogName = 'Scooby-Doo';
var dogName = 'Droopy';
var DOGNAME = 'Odie';
console.log(DogName); // "Scooby-Doo"
console.log(dogName); // "Droopy"
console.log(DOGNAME); // "Odie"
However, the most recommended way to declare JavaScript variables is with camel␣
 ↪case variable names. You can use the camel case naming convention for all␣
 ↪types of variables in JavaScript, and it will ensure that there aren't␣
 ↪multiple variables with the same name.

// bad
var dogname = 'Droopy';
// bad
var dog_name = 'Droopy';
// bad
var DOGNAME = ‘Droopy’;
// bad
var DOG_NAME = 'Droopy';
// good
var dogName = 'Droopy';
The names of variables should be self-explanatory and describe the stored value.
 ↪ For example, if you need a variable to store a dog's name, you should use␣
 ↪dogName instead of just Name since it is more meaningful.

// bad
var d = 'Scooby-Doo';
// bad
var name = 'Scooby-Doo';
// good
```

```
var dogName = 'Scooby-Doo';

2.Naming Convention for Functions
JavaScript function names are also case-sensitive. So, similar to variables,␣
 ↪the camel case approach is the recommended way to declare function names.

In addition to that, you should use descriptive nouns and verbs as prefixes.␣
 ↪For example, if we declare a function to retrieve a name, the function name␣
 ↪should be getName.

// bad
function name(dogName, ownerName) {
  return '${dogName} ${ownerName}';
}

// good
function getName(dogName, ownerName) {
  return '${dogName} ${ownerName}';
}
```

[14]:
```
#Write a python program to recursively find sum of digits of a List

myList=[23,4,2,6,7]

def sumOfList(myList, nSum):
    if len(myList):
        return sumOfList(myList[1:], nSum+myList[0])
    else:
        return nSum

print(sumOfList(myList, 0))
```

```
42
```

[ ]: