For Loop :-

1. Write a Python program to print numbers from 1 to 10 using a for loop.

In [4]:
```
for i in range(1,11):
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

2. Explain the difference between a for loop and a while loop in Python.

for loop: For loops are typically used when we know in advance how many times you want to repeat a block of code, or when you want to iterate over a collection of items.

syntax
for item in iterable:

```
    #code to be executed
```

while loop: A while loop, on the other hand, repeats a block of code as long as a specified condition is evaluated as True. It doesn't rely on iterating over a sequence; instead, it continuously checks the condition before each iteration

syntax:
while condition:

```
    # Code to be executed while the condition is True<br>
    #condition update<br>
```

3. Write a Python program to calculate the sum of all numbers from 1 to 100 using a for loop.

In [6]:
```
sum_dig=sum(i for i in range(1,101))
print("Sum of all numbers from 1 to 100 is :",sum_dig)
```

```
Sum of all numbers from 1 to 100 is : 5050
```

4. How do you iterate through a list using a for loop in Python?

In [7]:
```
#sample list
my_list=[1,2,3,4,5,6,7,8,9]
for i in my_list:        #iterating through the elements of list using for loop
    print(i)             #printing the elements of list one-by-one
```

```
1
2
3
4
5
6
7
8
9
```

5. Write a Python program to find the product of all elements in a list using a for loop.

In [9]:
```
#sample list
my_list=[1,2,3,4,5,6,7,8,9]
product=1
for i in my_list:
    product=i*product
print("Product of all elements in the list is:",product)
```

```
Product of all elements in the list is: 362880
```

6. Create a Python program that prints all even numbers from 1 to 20 using a for loop.

In [11]:
```
print("Even numbers from 1 to 20 are:")
for i in range(1,21):
    if i%2 == 0:
        print(i,end="   ")
```

```
Even numbers from 1 to 20 are:
2   4   6   8   10   12   14   16   18   20
```

7. Write a Python program that calculates the factorial of a number using a for loop.

In [16]:
```
num = int(input("Enter the number:"))
fact = 1
for i in range(1,num+1):
    fact = fact*i
print(f"Factorial of {num} is {fact}")
```

```
Enter the number:6
Factorial of 6 is 720
```

8. How can you iterate through the characters of a string using a for loop in Python?

In [20]:
```
#sample string
string="assignment"
for char in string:     #iterating through each character of string using for loop
    print(char)         #printing each character of the string
```

```
a
s
s
i
g
n
m
e
n
t
```

9. Write a Python program to find the largest number in a list using a for loop.

In [21]:
```
my_list=[14,52,78,59,12,32,75,41,23,65,88,81,121,35,2]
largest=my_list[0]
for i in my_list:
    if i>largest:
        largest=i
print(f"The largest number in the list {my_list} is {largest}")
```

```
The largest number in the list [14, 52, 78, 59, 12, 32, 75, 41, 23, 65, 88, 81, 12
1, 35, 2] is 121
```

10. Create a Python program that prints the Fibonacci sequence up to a specified limit using a for loop.

In [33]:
```
n=int(input("Enter the value of n:"))
a=0
b=1
if n==1:
    print(a)
elif n==2:
    print(a,b,sep='\n')
elif n>2:
    print(a,b,sep='\n')
    for i in range(3,n+1):
        a,b=b,a+b
        print(b)
else:
    print("Enter positive value on n")
```

```
Enter the value of n:10
0
1
1
2
3
5
8
13
21
34
```

11. Write a Python program to count the number of vowels in a given string using a for loop.

In [37]:

```
string=input("Enter the string:").lower()
vowels="aeiou"
count=0
for char in string:
    if char in vowels:
        count+=1
print(f"The number of vowels in the string '{string}' is {count}")
```

```
Enter the string:assignment
The number of vowels in the string 'assignment' is 3
```

12. Create a Python program that generates a multiplication table for a given number using a for loop

In [41]:

```
num = int(input("Enter the number:"))
print(f"Multiplication table for {num}:")
for i in range(1,11):
    print(f"{num} * {i} = {num*i}")
```

```
Enter the number:7
Multiplication table for 7:
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70
```

13. Write a Python program to reverse a list using a for loop.

In [55]:
```
my_list = [14, 52, 78, 59, 12, 32, 75, 2]
print("Original list:", my_list)

reversed_list = []

for i in range(len(my_list) - 1, -1, -1):
    reversed_list.append(my_list[i])

print("Reversed list:", reversed_list)
```

```
Original list: [14, 52, 78, 59, 12, 32, 75, 2]
Reversed list: [2, 75, 32, 12, 59, 78, 52, 14]
```

14. Write a Python program to find the common elements between two lists using a for loop.

In [1]:
```
list1=[1,2,3,4,5,6,7,8,9,10]
list2=[2,4,6,8,10]
common_list=[]
for i in list1:
    if i in list2:
        common_list.append(i)
print("Common elements between the 2 lists are:",common_list)
```

```
Common elements between the 2 lists are: [2, 4, 6, 8, 10]
```

15. Explain how to use a for loop to iterate through the keys and values of a dictionary in Python.

In [2]:
```
#sample dictionary
my_dict={'name':"Katappa",'age':51,'skills':['fighting','swimming','horse riding'],
```

In [3]:
```
#for loop to iterate through the keys only

for i in my_dict.keys():
    print(i)
```

```
name
age
skills
batch
```

In [4]:
```
#for loop to iterate through the values only

for i in my_dict.values():
    print(i)
```

```
Katappa
51
['fighting', 'swimming', 'horse riding']
4
```

In [5]:
```
#for loop to iterate through the items in the dictionary
```

```
for i in my_dict.items():
    print(i)
```

```
('name', 'Katappa')
('age', 51)
('skills', ['fighting', 'swimming', 'horse riding'])
('batch', 4)
```

16. Write a Python program to find the GCD (Greatest Common Divisor) of two numbers using a for loop.

In [13]:

```
num1 = int(input("Enter the first number:"))
num2 = int(input("Enter the second number:"))
gcd = 1

for i in range(1,min(num1,num2)+1):
    if num1%i == 0 and num2%i == 0:
        gcd = i
print(f"GCD of {num1} and {num2} is {gcd}")
```

```
Enter the first number:45958
Enter the second number:2348
GCD of 45958 and 2348 is 2
```

17. Create a Python program that checks if a string is a palindrome using a for loop.

In [21]:

```
def is_palindrome(string):

    start = 0
    end = len(string) - 1

    for i in range(len(string) // 2):

        if string[start] != string[end]:
            return False

        start += 1
        end -= 1

    return True


string = input("Enter a string: ")

if is_palindrome(string):
    print(f"'{string}' is palindrome.")
else:
    print(f"'{string}' is not palindrome.")
```

```
Enter a string: radar
'radar' is palindrome.
```

18. Write a Python program to remove duplicates from a list using a for loop.

In [29]:
```
sample_list=[1,2,4,52,6,1,2,47,85,1,65,2,1,5,2,1]
unique_list=[]
print("Original list:\n",sample_list)
for i in sample_list:
    if i not in unique_list:
        unique_list.append(i)
print("List after removing duplicates:\n",unique_list)
```

```
Original list:
 [1, 2, 4, 52, 6, 1, 2, 47, 85, 1, 65, 2, 1, 5, 2, 1]
List after removing duplicates:
 [1, 2, 4, 52, 6, 47, 85, 65, 5]
```

19. Create a Python program that counts the number of words in a sentence using a for loop.

In [33]:
```
sentence = input("Enter a sentence: ")
count=0
words = sentence.split()
for i in words:
    count+=1
print(f"Number of words in the sentence '{sentence}' is: {count}")
```

```
Enter a sentence: The quick brown fox jumped over a lazy dog
Number of words in the sentence 'The quick brown fox jumped over a lazy dog' is: 9
```

20. Write a Python program to find the sum of all odd numbers from 1 to 50 using a for loop

In [34]:
```
odd_sum = sum(i for i in range(1,51) if i%2 != 0)
print("Sum of odd number from 1 to 50: ",odd_sum)
```

```
Sum of odd number from 1 to 50:  625
```

21. Write a Python program that checks if a given year is a leap year using a for loop.

In [38]:
```
year = int(input("Enter a year:"))
for i in range(1):
    if year%400 == 0:
        print(year," is a leap year")
    elif year%4 == 0 and year%100 != 0:
        print(year," is a leap year")
    else:
        print(year," is not a leap year")
```

```
Enter a year:2100
2100  is not a leap year
```

22. Create a Python program that calculates the square root of a number using a for loop.

In [48]:
```python
def square_root(number, iterations=100):

    guess = number / 2.0
    for _ in range(iterations):
        guess = (guess + number / guess) / 2.0

    return guess

input_number = float(input("Enter a number: "))

result = square_root(input_number)

print(f"The square root of {input_number} is approximately {result:.3f}")
```

```
Enter a number: 45
The square root of 45.0 is approximately 6.708
```

23. Write a Python program to find the LCM (Least Common Multiple) of two numbers using a for loop.

In [60]:
```python
num1 = int(input("Enter the first number:"))
num2 = int(input("Enter the second number:"))
gcd=1
for i in range(1,min(num1,num2)+1):
    if num1%i == 0 and num2%i == 0:
        gcd=i
lcm = int(num1*num2/gcd)
print(f"LCM of {num1} and {num2} is {lcm}")
```

```
Enter the first number:25
Enter the second number:145
LCM of 25 and 145 is 725
```

If else :

1. Write a Python program to check if a number is positive, negative, or zero using an if-else statement.

In [63]:
```python
num=int(input("Enter the number:"))
if num>=0:
    if num==0:
        print(f"{num} is zero")
    else:
        print(f"{num} is positive")
```

```
else:
    print(f"{num} is negative")
```

```
Enter the number:-5
-5 is negative
```

2. Create a Python program that checks if a given number is even or odd using an if-else statement.

In [69]:
```
num = int(input("Enter the number:"))
if num%2 == 0:
    print(num,"is even")
else:
    print(num,"is odd")
```

```
Enter the number:13
13 is odd
```

3. How can you use nested if-else statements in Python, and provide an example?

In [70]:
```
'''
In case of nested if-else statement if our initial condition is true then we check
'''
#for example
num = int(input("Enter the number:"))
if num>=0:                #this is our first condition
    if num==0:            #if the previous condition is true then we copme to this co
        print(f"{num} is zero")
    else:
        print(f"{num} is positive")
else:
    print(f"{num} is negative")
```

```
Enter the number:54
54 is positive
```

4. Write a Python program to determine the largest of three numbers using if-else.

In [71]:
```
a = int(input("Enter the first number: "))
b = int(input("Enter the second number: "))
c = int(input("Enter the third number: "))

if a>=b and a>=c:
    max=a
elif b>=c and b>=a:
    max=b
else:
    max=c
print(f"\nmaximum of {a},{b},{c} is {max}")
```

```
Enter the first number: 45
Enter the second number: 21
Enter the third number: 52

maximum of 45,21,52 is 52
```

5. Write a Python program that calculates the absolute value of a number using if-else.

In [75]:
```
num = int(input("Enter the number:"))
if num<0:
    abs_value = num*-1
else:
    abs_value=num
print(f"Absolute value of {num} is {abs_value}")
```

```
Enter the number:-13
Absolute value of -13 is 13
```

6. Create a Python program that checks if a given character is a vowel or consonant using if-else.

In [82]:
```
char = input("Enter a character:")
vowels="aeiouAEIOU"
if char in vowels:
    print(f"'{char}' is a vowel")
else:
    print(f"'{char}' is a consonant")
```

```
Enter a character:e
'e' is a vowel
```

7. Write a Python program to determine if a user is eligible to vote based on their age using if-else.

In [83]:
```
age = int(input("Enter your age:"))
if age >= 18:
    print("You are eligible to vote")
else:
    print("You are not eligible to vote")
```

```
Enter your age:21
You are eligible to vote
```

8. Create a Python program that calculates the discount amount based on the purchase amount using if-else.

In [92]:
```
purchase_amt = float(input("Enter your purchase amount:₹"))
if purchase_amt >= 1000:
    discount_amt = purchase_amt-purchase_amt*.1        #10% discount above purchase
else:
    discount_amt = purchase_amt
print("your Discounted amount is:",discount_amt)
```

```
Enter your purchase amount:₹1200
your Discounted amount is: 1080.0
```

9. Write a Python program to check if a number is within a specified range using if-else.

In [97]:
```
#here we take the range to be 55-150

num = int(input("Enter a number:"))
if num in range(55,151):
    print(num,"is in range")
else:
    print(num,"is not in range")
```

```
Enter a number:148
148 is in range
```

10. Create a Python program that determines the grade of a student based on their score using if-else.

In [99]:
```
marks=float(input("Enter your marks out of 100:"))
if marks>=33:
    grade='P'
else:
    grade='F'
print("Your grade is:",grade)
```

```
Enter your marks out of 100:94
Your grade is: P
```

11. Write a Python program to check if a string is empty or not using if-else.

In [108…
```
string = input("Enter a string:")
if len(string)>0:
    print("String is not empty")
else:
    print("String is empty")
```

```
Enter a string:sample
String is not empty
```

12. Create a Python program that identifies the type of a triangle (e.g., equilateral, isosceles, or scalene) based on input values using if-else.

In [110…

```
side1 = float(input("Enter the first side:"))
side2 = float(input("Enter the second side:"))
side3 = float(input("Enter the third side:"))

if side1+side2>side3 and side1+side3>side2 and side2+side3>side1:
    if side1 == side2 == side3:
        print("This is an equilateral triangle")
    elif side1 == side2 or side2 == side3 or side3 == side1:
        print("This is an isosceles triangle")
    else:
        print("This is a scelene triangle")
else:
    print("The sides don't form a triangle")
```

```
Enter the first side:10
Enter the second side:12
Enter the third side:15
This is a scelene triangle
```

13. Write a Python program to determine the day of the week based on a user-provided number using if-else.

In [115…

```
num = int(input("Enter a number between 1-7:"))
if num == 1:
    day="Sunday"
elif num == 2:
    day="monday"
elif num == 3:
    day="wednesday"
elif num == 4:
    day="thursday"
elif num == 5:
    day="friday"
elif num == 6:
    day="saturay"
elif num == 7:
    day="saturday"
else:
    day="Invalid input"

print(f"The day of the week for {num} is {day}.")
```

```
Enter a number between 1-7:2
The day of the week for 2 is monday.
```

14. Create a Python program that checks if a given year is a leap year using both if-

else and a function.

In [120…

```
def check_year(year):
    if year%400 == 0 or (year%4 == 0 and year%100 != 0):
        print(year," is a leap year")
    else:
        print(year," is not a leap year")

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

```
Enter a year:2100
2100  is not a leap year
```

15. How do you use the "assert" statement in Python to add debugging checks within if-else blocks?

In [127…

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

if x > 5:
    assert x < 20, "x should be less than 20" #if the assertion condition is not sa
    print("x is greater than 5 but less than 20.")
else:
    print("x is not greater than 5.")
```

```
Enter a number:12
x is greater than 5 but less than 20.
```

16. Create a Python program that determines the eligibility of a person for a senior citizen discount based on age using if-else.

In [129…

```
age = int(input("Enter your age:"))
if age >= 60:
    print("You are eligible for senior citizen discount")
else:
    print("You are not eligible for senior citizen discount")
```

```
Enter your age:65
You are eligible for senior citizen discount
```

17. Write a Python program to categorize a given character as uppercase, lowercase, or neither using if-else.

In [133…

```
char = input("Enter a character:")
if char.isupper():
    print(f"'{char}' is uppercase")
elif char.islower():
    print(f"'{char}' is lowercase")
else:
    print(f"'{char}' is neither uppercase nor lowercase")
```

```
Enter a character:x
'x' is lowercase
```

18. Write a Python program to determine the roots of a quadratic equation using if-else.

In [134…

```
import math

print("Enter the coefficients 'a','b','c' of the quadratic equation of the form ax^
a=int(input("Enter a:"))
b=int(input("Enter b:"))
c=int(input("Enter c:"))
d=b**2-(4*a*c)
if d>0:
    root1=(-b+math.sqrt(d))/2*a
    root2=(-b-math.sqrt(d))/2*a
    print(f"Roots of the quadratic equation are real,unique and they are: {root1} a
elif d==0:
    root=-b/(2*a)
    print(f"Roots of the quadratic equation are real,equal and they are: {root}")
else:
    real_part = -b / (2*a)
    imaginary_part = math.sqrt(abs(d)) / (2*a)
    root1 = complex(real_part, imaginary_part)
    root2 = complex(real_part, -imaginary_part)
    print(f"The roots are complex: {root1} and {root2}")
```

```
Enter the coefficients 'a','b','c' of the quadratic equation of the form ax^2+bx+
c.
Enter a:1
Enter b:-4
Enter c:4
Roots of the quadratic equation are real,equal and they are: 2.0
```

19. Create a Python program that checks if a given year is a century year or not using if-else.

In [136…

```
year = int(input("Enter a year:"))
if year%100 == 0:
    print(year,"is a century year.")
else:
    print(year,"is not a century year")
```

```
Enter a year:2000
2000 is a century year.
```

20. Write a Python program to determine if a given number is a perfect square using if-

else.

In [148…
```
num = int(input("Enter a number:"))
sq = int(num**0.5)
if sq**2 == num:
    print(num,"is a perfect square")
else:
    print(num,"is not a perfect square")
```

```
Enter a number:27
27 is not a perfect square
```

21. Explain the purpose of the "continue" and "break" statements within if-else loops.

In [149…
```
'''
The "continue" statement is used to skip the rest of the current iteration of a loo
'''
#for example

for i in range(1,11):
    if i==5:                    #skip iteration when i is 5
        continue                      #when i is equal to 5, it will skip this iteratic
    else:
        print(i)
```

```
1
2
3
4
6
7
8
9
10
```

In [150…
```
'''
The "break" statement is used to terminate the loop prematurely. When the "break" s
the loop immediately exits, regardless of whether the loop's condition or iteration
'''

#for example

for i in range(1,11):
    if i==5:                    #it will terminate the iteration when i is 5
        break
    else:
        print(i)
```

```
1
2
3
4
```

22. Create a Python program that calculates the BMI (Body Mass Index) of a

person based on their weight and height using if-else.

In [151…

```
height = float(input("Enter your height in cm:"))
weight = float(input("Enter your weight in kg:"))
bmi = weight/((height/100)**2)
print("Your BMI is:",bmi)
if bmi<=18.5:
    print("You are underweight")
elif 18.5<bmi<25:
    print("You are normal")
elif 25<bmi<30:
    print("You are overweight")
else:
    print("You are obese")
```

```
Enter your height in cm:165
Enter your weight in kg:60
Your BMI is: 22.03856749311295
You are normal
```

23. How can you use the "filter()" function with if-else statements to filter elements from a list?

In [152…

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

# Define a filtering function
def is_even(x):
    return x % 2 == 0

# Use filter() to filter elements based on the condition
filtered_numbers = filter(is_even, numbers)


filtered_numbers_list = list(filtered_numbers)

print(filtered_numbers_list)
```

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

24. Write a Python program to determine if a given number is prime or not using if-else.

In [162…

```
num = int(input("Enter a number:"))
count=0
for i in range(1,num+1):
    if num%i == 0:
        count+=1
if count==2:
    print(num,"is prime")
else:
    print(num,"is not prime")
```

```
Enter a number:7
7 is prime
```

Map :-

1. 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.

The map function applies a given function to each element in an iterable (e.g., a list) and returns an iterable containing the results.
Syntax:

map(function, iterable)

In [1]:
```
'''
For example:
We have a list and we want to double each elements of the list
'''

numbers = [1,2,3,4,5]
doubled_numbers = list(map((lambda a:a*2),numbers))
print("Original list",numbers)
print("Doubled list",doubled_numbers)
```

```
Original list [1, 2, 3, 4, 5]
Doubled list [2, 4, 6, 8, 10]
```

2. Write a Python program that uses the `map()` function to square each element of a list of numbers.

In [2]:
```
numbers = [1,2,3,4,5]
sq_num = list(map((lambda a:a**2),numbers))
print("Original list",numbers)
print("Squared list",sq_num)
```

```
Original list [1, 2, 3, 4, 5]
Squared list [1, 4, 9, 16, 25]
```

3. How does the `map()` function differ from a list comprehension in Python, and when would you choose one over the other?

map() is a built-in Python function used for applying a specified function to each item in an iterable.

Syntax: It takes two arguments - the function and the iterable.

Output: map() returns a map object, which is an iterator. To obtain a list or another iterable, you need to convert it explicitly using list(), tuple(), or another constructor.

List comprehensions are a concise and readable way to create new lists by applying expressions to each element in an iterable.

Syntax: They use square brackets [ ] and can include optional conditionals for filtering elements.

Output: List comprehensions directly produce a new list without the need for additional conversion.

List comprehensions are often preferred in Python for their simplicity and readability, while map() is a better choice when memory efficiency is a concern or when working with more complex transformations that are better represented as functions.

4. Create a Python program that uses the `map()` function to convert a list of names to uppercase.

In [3]:
```
my_list = ["salman","akshay","ajay","shahrukh","aamir","saif"]
upper_list = list(map((lambda a:a.upper()),my_list))
print("Original list:",my_list)
print("Uppercase list:",upper_list)
```

```
Original list: ['salman', 'akshay', 'ajay', 'shahrukh', 'aamir', 'saif']
Uppercase list: ['SALMAN', 'AKSHAY', 'AJAY', 'SHAHRUKH', 'AAMIR', 'SAIF']
```

5. Write a Python program that uses the `map()` function to calculate the length of each word in a list of strings.

In [4]:
```
my_list = ["salman","akshay","ajay","shahrukh","aamir","saif"]
length_list = list(map((lambda a:len(a)),my_list))
print("Length of each word in the list:",length_list)
```

```
Length of each word in the list: [6, 6, 4, 8, 5, 4]
```

6. How can you use the `map()` function to apply a custom function to elements of multiple lists simultaneously in Python?

```
In [9]: '''
        we will define a custom function for adding 2 numbers
        and add elements of 2 lists.
        '''

        list1 = [10,20,30,40,50]
        list2 = [5,4,3,2,1]
        sum_list = list(map(lambda a,b:a+b,list1,list2)) #using map function on 2 lists sim
        print("sum of elements of the 2 list:",sum_list)
```

```
sum of elements of the 2 list: [15, 24, 33, 42, 51]
```

7. Create a Python program that uses `map()` to convert a list of temperatures from Celsius to Fahrenheit.

```
In [10]: celsius_list = [0,30,40,100,50]
         fahrenheit_list = list(map((lambda a:(9*a/5)+32),celsius_list))
         print("Celsius list:",celsius_list)
         print("Fahrenheit list:",fahrenheit_list)
```

```
Celsius list: [0, 30, 40, 100, 50]
Fahrenheit list: [32.0, 86.0, 104.0, 212.0, 122.0]
```

8. Write a Python program that uses the `map()` function to round each element of a list of floating-point numbers to the nearest integer.

```
In [11]: float_list = [12.5,13.8,45.21,32.1,11.98,42.6,4.2]
         nearest_int_list = list(map((lambda a:round(a)),float_list))
         print("Float list:",float_list)
         print("Nearest integer list:",nearest_int_list)
```

```
Float list: [12.5, 13.8, 45.21, 32.1, 11.98, 42.6, 4.2]
Nearest integer list: [12, 14, 45, 32, 12, 43, 4]
```

Reduce :-

1. 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 function applies a given function cumulatively to the elements of an iterable, reducing it to a single value.

We should import functools module to use it.

syntax :

from functools import reduce

```
In [15]:
'''
For example:
we can find the sum of all elements in a list
'''
from functools import reduce

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

sum_num = reduce((lambda a,b:a+b),l)

print("sum of all the numbers in the list:",sum_num)
```

```
sum of all the numbers in the list: 55
```

2. Write a Python program that uses the `reduce()` function to find the product of all elements in a list.

```
In [14]:
my_list = [1,2,3,4,5,6,7]
product = reduce((lambda a,b:a*b),my_list)
print("Product of all elements in the list is:",product)
```

```
Product of all elements in the list is: 5040
```

3. Create a Python program that uses `reduce()` to find the maximum element in a list of numbers.

```
In [17]:
my_list = [2,44,52,41,63,20,-85,63,45,95,231,12,41,121]

max_ele = reduce((lambda a,b:a if a>b else b),my_list)

print("Maximum element in the list is:",max_ele)
```

```
Maximum element in the list is: 231
```

4. How can you use the `reduce()` function to concatenate a list of strings into a single string?

```
In [4]:
from functools import reduce
my_list = ['This', 'is', 'a', 'sample', 'string']
```

```
string = reduce((lambda a,b:a+" "+b),my_list)
print("String list:",my_list)
print("concatenated list of string:",string)
```

```
String list: ['This', 'is', 'a', 'sample', 'string']
concatenated list of string: This is a sample string
```

5. Write a Python program that calculates the factorial of a number using the `reduce()` function.

In [31]:
```
from functools import reduce

n = int(input("Enter the number whose factorial is to be found:"))

factorial = reduce(lambda a,b:a*b,range(1,n+1))
print(f"Factorial of {n} is {factorial}")
```

```
Enter the number whose factorial is to be found:4
Factorial of 4 is 24
```

6. Create a Python program that uses `reduce()` to find the GCD (Greatest Common Divisor) of a list of numbers.

In [50]:
```
from functools import reduce

my_list = [56,42,58,96,108]

#defining a function to find gcd of 2 numbers
def find_gcd(a,b):
    gcd = 1
    for i in range(1,min(a,b)+1):
        if a%i == 0 and b%i == 0:
            gcd=i
    return gcd

gcd = reduce(find_gcd,my_list)

print(f"GCD of the elements of the list {my_list} is {gcd}")
```

```
GCD of the elements of the list [56, 42, 58, 96, 108] is 2
```

7. Write a Python program that uses the `reduce()` function to find the sum of the digits of a given number.

In [51]:
```
from functools import reduce
num = input("Enter the number whose sum of digits is required:")

dig_sum = reduce(lambda a,b:int(a)+int(b),num)
```

```
print(f"Sum of the digits of the number {int(num)} is {dig_sum}")
```

```
Enter the number whose sum of digits is required:45
Sum of the digits of the number 45 is 9
```

Filter :-

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.

The filter function filters elements from an iterable based on a provided function's condition and returns an iterable containing the filtered elements.

Syntax:

filter(function, iterable)

In [52]:
```
'''
For example:
we have a list of numbers we will filter out the odd numbers from the list using fi
'''

my_list = [1,2,3,4,5,6,7,8,9,10]
odd_list = list(filter((lambda a:a%2!=0),my_list))
print("Original list:",my_list)
print("Filtered list with only odd elements",odd_list)
```

```
Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Filtered list with only odd elements [1, 3, 5, 7, 9]
```

2. Write a Python program that uses the `filter()` function to select even numbers from a list of integers.

In [53]:
```
my_list = [1,2,3,4,5,6,7,8,9,10]
even_list = list(filter((lambda a:a%2==0),my_list))
print("Original list:",my_list)
print("Filtered list with only even elements",even_list)
```

```
Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Filtered list with only even elements [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.

In [58]:
```python
my_list = ["ram","ramayan","ravan","shyam","sameer","sundar","noni","karn","thala"]
ch = input("Enter the initial character:").lower()
filtered_list = list(filter((lambda l:l[0]==ch),my_list))
if len(filtered_list)>0:
    print(f"Names starting with '{ch}' are {filtered_list}")
else:
    print(f"No names in the list start with '{ch}'")
```

```
Enter the initial character:r
Names starting with 'r' are ['ram', 'ramayan', 'ravan']
```

4. Write a Python program that uses the `filter()` function to select prime numbers from a list of integers.

In [92]:
```python
my_list = [1,2,3,4,5,6,7,8,9,10]

#defining function to find prime number
def is_prime(num):
    count=0
    for i in range(1,num+1):
        if num%i ==0:
            count+=1
    if count == 2:
        return True
    else:
        return False
prime_list = list(filter(is_prime,my_list))
print("original list:",my_list)
print("Filtered list with prime numbers:",prime_list)
```

```
original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Filtered list with prime numbers: [2, 3, 5, 7]
```

5. How can you use the `filter()` function to remove None values from a list in Python?

In [114…
```python
my_list = [1,2,3,None,5,8,None,85,None]

filtered_list = list(filter((lambda l:l is not None),my_list))

print("original list:",my_list)
print("Filtered list without None values:",filtered_list)
```

```
original list: [1, 2, 3, None, 5, 8, None, 85, None]
Filtered list without None values: [1, 2, 3, 5, 8, 85]
```

In [113…
```python
filtered_list
```

Out[113]:
```
[1, 2, 3, 5, 8, 85]
```

6. Create a Python program that uses `filter()` to select words longer than a certain length from a list of strings.

In [117…
```
my_list = ["ram","ramayan","ravan","shyam","sameer","sundar","noni","karn","thala"]
filtered_list = list(filter((lambda l:len(l)>5),my_list))    #selecting words longe
print("original list:",my_list)
print("\nFiltered list with words of length greater than 5:",filtered_list)
```

```
original list: ['ram', 'ramayan', 'ravan', 'shyam', 'sameer', 'sundar', 'noni', 'k
arn', 'thala']

Filtered list with words of length greater than 5: ['ramayan', 'sameer', 'sundar']
```

7. Write a Python program that uses the `filter()` function to select elements greater than a specified threshold from a list of values.

In [121…
```
my_list = [20,52,4,5,75,85,6,2,4,36,45,2,5,36,5,2,65,3,5,32,6,2,58,36,5,3,96,45,36,
print("Original list:\n",my_list)
value = int(input("Enter the threshold value:"))
filtered_list = list(filter((lambda l:l>value),my_list))
if len(filtered_list)>0:
    print(f"Filtered list with values greater than {value} is {filtered_list}")
else:
    print(f"No element in the list is greater than {value}")
```

```
Original list:
 [20, 52, 4, 5, 75, 85, 6, 2, 4, 36, 45, 2, 5, 36, 5, 2, 65, 3, 5, 32, 6, 2, 58, 3
6, 5, 3, 96, 45, 36, 96, 3, 5, 2, 6]
Enter the threshold value:60
Filtered list with values greater than 60 is [75, 85, 65, 96, 96]
```

Recursion:-

1. Explain the concept of recursion in Python. How does it differ from iteration?

Recursion in Python:

Recursion is a programming technique where a function calls itself to solve a problem. In Python, a recursive function is a function that performs a task in part and delegates the remaining part of the task to itself. Recursive functions typically have two parts:

Base Case(s):

These are the conditions under which the function returns a result directly without making any further recursive calls. Base cases are essential to prevent infinite recursion and provide a stopping point for the recursion.

Recursive Case(s):

These are the conditions under which the function calls itself with modified arguments to solve a smaller or simpler version of the original problem. The function keeps making these recursive calls until it reaches the base case(s) and then works its way back up, combining the results to solve the original problem.

Differences between Recursion and iteration are:

Control Flow:

Recursion: Control flow is managed by making recursive function calls, where each call is a new invocation of the function. Iteration: Control flow is managed using loops (e.g., for and while loops) that repeatedly execute a block of code until a specified condition is met.

Termination Condition:

Recursion: Requires a base case or termination condition that specifies when the recursion should stop. The function returns a result when the base case is met. Iteration: Requires a loop condition that determines when the loop should terminate. The loop continues until the condition is false.

Method of Repeating:

Recursion: Achieves repetition by making a function call to itself with modified arguments, solving a smaller or simpler instance of the problem in each call. Iteration: Achieves repetition by repeatedly executing a block of code within a loop without making new function calls.

2. Write a Python program to calculate the factorial of a number using recursion

In [135…

```
def fact(num):
    if num == 0:
        return 1
    else:
        return num*fact(num-1)

num = int(input("Enter the number whose factorial is to be found:"))
print(f"factorial of {num} is {fact(num)}")
```

```
Enter the number whose factorial is to be found:6
factorial of 6 is 720
```

3. Create a recursive Python function to find the nth Fibonacci number.

In [137…

```
def fibo(n):
    if n <= 1:
        return n
    else:
        return fibo(n-1)+fibo(n-2)

num = int(input("Enter the value of n:"))
print(f"{num}th term of the fibonacci series is {fibo(num)}")
```

```
Enter the value of n:10
10th term of the fibonacci series is 55
```

4. Write a recursive Python function to calculate the sum of all elements in a list.

In [144…

```
my_list = [12,45,12,10,32,56,85]

def ele_sum(my_list):
    if len(my_list)==1:
        return my_list[0]
    else:
        return my_list[0] + ele_sum(my_list[1::])

print(f"Sum of all the elements in the list {my_list} is {ele_sum(my_list)}")
```

```
Sum of all the elements in the list [12, 45, 12, 10, 32, 56, 85] is 252
```

5. How can you prevent a recursive function from running indefinitely, causing a stack overflow error?

Preventing a recursive function from running indefinitely and causing a stack overflow error is essential to ensure the proper execution of your program. Here are several strategies to prevent infinite recursion:

1-Base Case:

Ensure that your recursive function has a well-defined base case. The base case defines when the recursion should stop. When the input to the function reaches this base case, the function should return a result or perform a specific action without making a recursive call.

2-Progress Toward Base Case:

Make sure that the recursive function makes progress towards the base case in each recursive call. In other words, ensure that the input to the function gets closer to the base case with each recursive call.

3-Conditional Checks:

Use conditional checks to limit recursion based on certain conditions. For example, you can check if the input meets certain criteria and decide whether to continue with the recursive call or not.

6. Create a recursive Python function to find the greatest common divisor (GCD) of two numbers using the Euclidean algorithm.

In [147…

```python
def gcd(a, b):
    # Base case: GCD(a, 0) = a, where a is the greater number.
    if b == 0:
        return a
    else:
        # Recursive case: GCD(a, b) is equivalent to GCD(b, a % b).
        return gcd(b, a % b)


num1 = int(input("Enter the first number:"))
num2 = int(input("Enter the second number:"))
result = gcd(num1, num2)
print(f"The GCD of {num1} and {num2} is {result}")
```

```
Enter the first number:48
Enter the second number:52
The GCD of 48 and 52 is 4
```

7. Write a recursive Python function to reverse a string

In [23]:

```python
def rev_str(string):
    if len(string)<=1:
        return string
    else:
        return string[-1]+rev_str(string[:-1])
string = input("Enter a string:")
print("Original string:",string)
print("reversed string:",rev_str(string))
```

```
Enter a string:assignment
Original string: assignment
reversed string: tnemngissa
```

8. Create a recursive Python function to calculate the power of a number (x^n).

In [156…

```
def power(x,n):
    if n == 0:
        return 1
    else:
        return x * power(x,n-1)

x = int(input("Enter number(x):"))
n = int(input("Enter power(n)"))
print(f"{x}^{n} is {power(x,n)}")
```

```
Enter number(x):2
Enter power(n)5
2^5 is 32
```

9. Write a recursive Python function to find all permutations of a given string.

In [5]:

```
def find_permutations(input_str):
    if len(input_str) == 1:
        return [input_str]

    all_permutations = []

    for i in range(len(input_str)):
        first_char = input_str[i]
        remaining_chars = input_str[:i] + input_str[i + 1:]
        remaining_permutations = find_permutations(remaining_chars)

        for perm in remaining_permutations:
            all_permutations.append(first_char + perm)

    return all_permutations


input_string = "xyz"
permutations = find_permutations(input_string)
print(f"Permutations of '{input_string}':")
for perm in permutations:
    print(perm)
```

```
Permutations of 'xyz':
xyz
xzy
yxz
yzx
zxy
zyx
```

10. Write a recursive Python function to check if a string is a palindrome.

In [22]:
```python
def is_palindrome(string):

    if len(string) <= 1:
        return True
    else:
        if string[0] != string[-1]:
            return False
        else:
            return is_palindrome(string[1:-1])


string = input("Enter a string:")

if is_palindrome(string):
    print(f"'{string}' is a palindrome string")
else:
    print(f"'{string}' is not a palindrome string")
```

```
Enter a string:radar
'radar' is a palindrome string
```

11. Create a recursive Python function to generate all possible combinations of a list of elements.

In [24]:
```python
def generate_combinations(elements, k):
    # Base case: If k is 0, return an empty list (no combination).
    if k == 0:
        return [[]]

    # Base case: If there are no elements, return an empty list (no combination).
    if not elements:
        return []

    # Recursive case:
    # Generate combinations with the first element and without the first element.
    first_element = elements[0]
    combinations_without_first = generate_combinations(elements[1:], k)
    combinations_with_first = generate_combinations(elements[1:], k - 1)

    # Add the first element to combinations with the first element.
    combinations_with_first = [[first_element] + combo for combo in combinations_wi

    # Combine both sets of combinations.
    return combinations_without_first + combinations_with_first

# Example usage:
elements = [1, 2, 3]
k = 2
combinations = generate_combinations(elements, k)
print("Combinations of", elements, "choose", k, "are:")
for combo in combinations:
    print(combo)
```

```
Combinations of [1, 2, 3] choose 2 are:
[2, 3]
[1, 3]
[1, 2]
```

Basics of Functions:

1. What is a function in Python, and why is it used?

A function is a block of reusable code that performs a specific task or set of tasks.

Syntax:

def functionname(parameter):

    #code

2. How do you define a function in Python? Provide an example.

We define a function in python by the key word 'def'.

In [25]:
```
'''
for example:
'''

def square(x):        #we are defining a function named square using keyword def tha
    return x*x        #the function returns the square of the number x
```

In [26]:
```
sq = square(5)
```

In [27]:
```
sq
```

Out[27]:
```
25
```

3. Explain the difference between a function definition and a function call.

Function Definition:

A function definition is the code block where we define what the function does and how it should behave. It specifies the name of the function, the parameters it accepts, and the actions it performs when called.

In [28]:
```
# example

def add(a, b):                #defining a function named add that takes asks for 2 pa
    return a + b
```

Function Call:

A function call is the act of using a defined function to perform a specific task or computation. When we call a function, we execute the code inside the function definition and potentially pass values (arguments) to it.

In [30]:
```
#example

s = add(2,3)   #we are calling the function add and passing 2 arguments 2 and 3 and
print("sum =",s)
```

```
sum = 5
```

4. Write a Python program that defines a function to calculate the sum of two numbers and then calls the function.

In [31]:
```
def add(a, b):                #defining a function named add that takes asks for 2 pa
    return a + b

num1 = int(input("Enter first number:"))
num2 = int(input("Enter second number:"))

s = add(num1,num2)        # calling the function add and passing 2 arguments to cal

print(f"Sum of {num1} and {num2} is {s}")
```

```
Enter first number:5
Enter second number:4
Sum of 5 and 4 is 9
```

5. What is a function signature, and what information does it typically include?

The signature of a function in Python is a description of the function's parameters, their types, and the type of the function's return value. The signature of a function is important for a number of reasons. It helps the reader of the function's code understand what arguments the function expects and what type of value it returns

syntax:

def function_name(argument1: type1, argument2: type2, ...) -> return_type:

```
# Function body
```

In [32]:
```python
#example:
def calculate_area(length: float, width: float) -> float:
    """
    Calculate the area of a rectangle.

    Args:
        length (float): The length of the rectangle.
        width (float): The width of the rectangle.

    Returns:
        float: The area of the rectangle.
    """
    return length * width
```

6. Create a Python function that takes two arguments and returns their product.

In [33]:
```python
def product(a,b):
    return a*b

num1 = int(input("Enter first number:"))
num2 = int(input("Enter second number:"))

p = product(num1,num2)
print(f"Product of {num1} and {num2} is {p}")
```

```
Enter first number:5
Enter second number:3
Product of 5 and 3 is 15
```

Function Parameters and Arguments:

1. Explain the concepts of formal parameters and actual arguments in Python functions.

Formal Parameters:

Formal parameters are placeholders or variables defined in the function's header or signature. They serve as a way to receive input values or arguments from the caller when the function is called. Formal parameters are used to specify what kind of data the function expects and how many arguments it requires. These parameters are local to the function and have scope only within the function block. Formal parameters are defined inside parentheses following the function name, separated by commas.

In [1]:
```python
#example
def add_numbers(x, y):  # 'x' and 'y' are formal parameters.
    result = x + y
    return result
```

Actual Arguments:

Actual arguments, also known as arguments or function arguments, are the values or expressions provided by the caller when calling a function. They are the real data that gets passed into the function and assigned to the corresponding formal parameters. Actual arguments can be variables, constants, or expressions, and they must match the number and order of formal parameters defined in the function. Actual arguments are enclosed in parentheses when calling the function.

In [2]:
```
result = add_numbers(3, 5)  # '3' and '5' are actual arguments.
```

2. Write a Python program that defines a function with default argument values

In [5]:
```
def num_sum(a = 10, b = 5):
    return a+b

'''
Here we define a function num_sum with 2 parametres with default argument
values a=10,b=5 .While calling the function if no argument is passed to the functic
it takes the default values of a and b and performs the operation.
and when the arguments are passed during the function call then the default values
with the passed arguments.
'''
sum1 = num_sum() #no arguments are passed during the call , so the default values c
sum2 = num_sum(2,5) #arguments are passed during the function call and so the defau
print("sum1=",sum1)
print("sum2=",sum2)
```

```
sum1= 15
sum2= 7
```

3. How do you use keyword arguments in Python function calls? Provide an example.

When calling a function, we specify arguments using the parameter names as keywords, followed by an equal sign and the corresponding values. This approach allows us to pass arguments in any order and is especially helpful when dealing with functions that have many parameters

In [6]:
```
#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')
```

```
First Name: Eric
Last Name: Cartman
```

4. Create a Python function that accepts a variable number of arguments and calculates their sum.

In [10]:
```python
def calc_sum(*nums):   #in parameter we use*args for variable number of arguments
    return sum(nums)

sum1 = calc_sum(1,2,3,4,5,6,7,8,9,10)
sum2 = calc_sum(1,2,4,5,7)

print("sum1=",sum1)
print("sum2=",sum2)
```

```
sum1= 55
sum2= 19
```

5. What is the purpose of the `*args` and `**kwargs` syntax in function parameter lists?

*args:

This is useful when we want to pass an arbitrary number of arguments without knowing in advance how many there will be.

In [11]:
```python
#eg:
def calc_sum(*nums):   #in parameter we use*args for variable number of arguments
    return sum(nums)

sum1 = calc_sum(1,2,3,4,5,6,7,8,9,10)
sum2 = calc_sum(1,2,4,5,7)

print("sum1=",sum1)
print("sum2=",sum2)
```

```
sum1= 55
sum2= 19
```

**kwargs

If we do not know how many keyword arguments that will be passed into our function, add two asterisk: ** before the parameter name in the function definition.

In [13]:
```python
#eg:

def create_person(**kwargs):
    person = {}
    if "first_name" in kwargs:
        person["first_name"] = kwargs["first_name"]
    if "last_name" in kwargs:
        person["last_name"] = kwargs["last_name"]
    if "age" in kwargs:
        person["age"] = kwargs["age"]
    return person

person1 = create_person(first_name="Alice", last_name="Johnson", age=30)
person2 = create_person(first_name="Bob", age=25)
person3 = create_person(last_name="Smith")

print(person1)
print(person2)
print(person3)
```

```
{'first_name': 'Alice', 'last_name': 'Johnson', 'age': 30}
{'first_name': 'Bob', 'age': 25}
{'last_name': 'Smith'}
```

Return Values and Scoping:

1. Describe the role of the `return` statement in Python functions and provide examples.

The return statement in Python functions is used to specify what value or values the function should give back as its result when it is called. It marks the end of the function's execution and passes control and data back to the caller.

In [15]:
```
#for example

def product(a,b):
    return a*b

#this function named product takes 2 arguments and resurns their product as result

result = product(13,5) #the value returned by the function is stored in the variabl
print("Result=",result)
```

```
Result= 65
```

2. Explain the concept of variable scope in Python, including local and global variables.

Variable scope in Python refers to the region or context in which a variable is defined and can be accessed.

Local variable

Local variables are declared and used within a specific function or block of code. They have limited visibility and exist only within the scope of the function or block where they are defined. Local variables are created when the function is called and destroyed when the function exits. They are not accessible from outside the function.

In [16]:
```
#example

def my_function():
    x = 10  #  here 'x' is a local variable
    print(x)

my_function()
```

```
10
```

In [18]:
```
print(x)  #if we try to access x outside the function we get an error
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_3392\2239527823.py in <module>
----> 1 print(x)  #if we try to access x outside the function we get an error

NameError: name 'x' is not defined
```

Global Variables:

Global variables are defined at the top level of a script or module, outside of any function. They have a global scope, which means they can be accessed from any part of the code, including inside functions. Global variables persist throughout the entire program's execution. If a function wants to access a global variable, it can do so without needing to declare it as global explicitly.

In [19]:
```
#example

global_var = 20  # 'global_var' is a global variable

def my_function():
    print(global_var)  # Accessing the global variable 'global_var' inside the func

my_function()
print(global_var)  # Accessing the global variable outside the function
```

```
20
20
```

3. Write a Python program that demonstrates the use of global variables within functions.

In [25]:
```
# Define a global variable
global_var = 10

# Define a function that uses the global variable
def modify_global():
    global global_var  # Declare 'global_var' as global within the function
    global_var += 5    # Modify the global variable
    print("Inside the function: global_var =", global_var)

# Call the function
modify_global()

# Print the global variable outside the function
print("Outside the function: global_var =", global_var)
```

```
Inside the function: global_var = 15
Outside the function: global_var = 15
```

4. Create a Python function that calculates the factorial of a number and returns it.

In [28]:
```
def factorial(num):
    fact = 1
    for i in range(1,num+1):
        fact = fact*i
    return fact

num = int(input("Enter number whose factorial is to be found:"))
fact = factorial(num)
print(f"Factorial of {num} is {fact}")
```

```
Enter number whose factorial is to be found:6
Factorial of 6 is 720
```

5. How can you access variables defined outside a function from within the function?

We can access variables defined outside a function from within the function using global keyword.

In [33]:
```
#eg:

x = 10

def test():
    global x
    x = x+15
    print(x)

test()

print("x =",x)
```

```
25
x = 25
```

Lambda Functions and Higher-Order Functions:

1. What are lambda functions in Python, and when are they typically used?

Lambda functions in Python, also known as anonymous functions or function without name, are small, inline functions that are defined without a name. They are typically used for short, simple operations that can be expressed concisely in a single line of code. Lambda functions are created using the lambda keyword and have the following syntax:

lambda arguments: expression

arguments are optional.

2. Write a Python program that uses lambda functions to sort a list of tuples based on the second element.

In [34]:
```
data = [(6, 5), (1, 4), (9, 10), (5, 13)]

sorted_data = sorted(data, key=lambda x: x[1])


print(sorted_data)
```

```
[(1, 4), (6, 5), (9, 10), (5, 13)]
```

3. Explain the concept of higher-order functions in Python, and provide an example.

Higher-order functions in Python are functions that can take one or more functions as arguments or return a function as their result.

In [35]:
```
# for example
def apply_operation(operation, operand):
    return operation(operand)

# Define two simple operations as functions
def double(x):
    return x * 2

def square(x):
    return x ** 2


result1 = apply_operation(double, 5)
result2 = apply_operation(square, 4)

print("Result 1:", result1)
print("Result 2:", result2)
```

```
Result 1: 10
Result 2: 16
```

4. Create a Python function that takes a list of numbers and a function as arguments, applying the function to each element in the list

In [42]:
```
#function that takes a function and a list of numbers as arguments
def combo(func,l):
    l1=[]
    for i in l:
        l1.append(func(i))
    return l1
#function to double a number
def double(num):
    return num*2

l=[1,2,3,4,5,6,7,8]

my_list = combo(double,l)
print("Doubled list=",my_list)
```

```
Doubled list= [2, 4, 6, 8, 10, 12, 14, 16]
```

Built-in Functions:

1. Describe the role of built-in functions like `len()`, `max()`, and `min()` in Python.

len():

It returns the length or number of elements in an object such as list, tuple, string,etc.

In [1]:
```
#example
text = "Assignment"
length = len(text)
print(f"Length of the string is {length}")
```

```
Length of the string is 10
```

max():

The max() function is used to find the maximum value from a collection of values. It can be used with various data types, including numbers and strings.

In [2]:
```
#example:

numbers = [5, 7, 8, 9, 2]
```

```
max_value = max(numbers)
print(f"Maximum value of the list is {max_value}")
```

```
Maximum value of the list is 9
```

min():

The min() function is the counterpart of max(). It is used to find the minimum value from a collection of values.

In [3]:
```
#example:
numbers = [5, 7, 8, 9, 2]
min_value = min(numbers)
print(f"Minimum value of the list is {min_value}")
```

```
Minimum value of the list is 2
```

2. Write a Python program that uses the `map()` function to apply a function to each element of a list.

In [5]:
```
my_list = [3,4,2,56,4,5,64,2,5,76] #sample list
#we will double each element of the list using map() function
doubled_list = list(map(lambda a:a*2,my_list))

print("Original list:",my_list)
print("Doubled list:",doubled_list)
```

```
Original list: [3, 4, 2, 56, 4, 5, 64, 2, 5, 76]
Doubled list: [6, 8, 4, 112, 8, 10, 128, 4, 10, 152]
```

3. How does the `filter()` function work in Python, and when would you use it?

The filter() function in Python is used to filter a sequence (such as a list, tuple, or iterable) based on a specified condition or function. It returns an iterator containing the elements from the original sequence that satisfy the given condition.

syntax:

filter(function,iterable)

The filter() function applies the provided function to each element in the sequence. If the function returns True for an element, that element is included in the resulting iterator; otherwise, it is excluded.

In [6]:
```
#for example

'''
we have a list of numbers we will filter out the even numbers from the
list using filter() function.
```

```
'''
my_list = [1,3,43,54,756,34,32,7,23,87,56,2,8,32,86,23]

even_list = list(filter(lambda x:x%2 == 0,my_list))

print("Original list:",my_list)
print("even list:",even_list)
```

```
Original list: [1, 3, 43, 54, 756, 34, 32, 7, 23, 87, 56, 2, 8, 32, 86, 23]
even list: [54, 756, 34, 32, 56, 2, 8, 32, 86]
```

4. Create a Python program that uses the `reduce()` function to find the product of all elements in a list.

In [12]:
```
from functools import reduce

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

product = reduce(lambda a,b:a*b,my_list)

print(f"Product of all the elements in the list {my_list} is {product}")
```

```
Product of all the elements in the list [2, 3, 4, 5, 6, 7, 8, 9, 10] is 3628800
```

Function Documentation and Best Practices:

1. Explain the purpose of docstrings in Python functions and how to write them.

Docstrings in Python functions serve as documentation strings that provide information about the purpose and usage of a function. They are used to document our code, making it more understandable and accessible to others (including your future self).

Methods to write docstring:

Docstrings are placed immediately after the function definition and are enclosed in triple-quotes (single or double). They can also be placed within classes and modules to document classes and modules.

A typical docstring consists of a one-line summary followed by a more detailed description. It can also include sections for parameters, return values, and examples.

In [14]:
```
#for example

def add(a, b):
    """
    Adds two numbers.
    This function takes two numbers as input and returns their sum.

    Args:
        a (int): The first number.
        b (int): The second number.

    Returns:
        int: The sum of 'a' and 'b'.

    Example:
        >>> add(2, 3)
        5
    """
    return a + b
```

2. Describe some best practices for naming functions and variables in python, including conventions and guidelines.

For Function Names:

Using Descriptive Names:

Choosing function names that clearly and concisely describe the purpose or action of the function. Names like calculate_total or validate_email are more informative than generic names like func1 or process_data.

Using lowercase with Underscores:

For function names, following the lowercase_with_underscores naming convention (also known as snake_case). This convention is widely accepted in Python. For example, calculate_total_amount, not calculateTotalAmount.

Verb-Noun Pairing:

Beginning function names with a verb or verb phrase that describes the action the function performs, followed by a noun or noun phrase. For example, get_user_info, compute_average_score.

Avoid Using Names of Built-in Functions:

Do not use names of built-in functions or Python keywords as function names to avoid conflicts and confusion.

Be Consistent:

Maintain consistency in your naming conventions across your codebase. If you use a specific naming style for functions, stick with it throughout your project.

For Variable Names:

convention for naming a variable:

A variable name must contain only letters, digits and underscore. A variable name cannot start with a digit

Use Descriptive Names:

Variable names should be clear and descriptive. Names like customer_age or product_price are better than generic names like x or temp.

Use lowercase with Underscores:

Similar to function names, use snake_case for variable names. This convention is the standard in Python.

Choose Meaningful Names:

The variable name should indicate the purpose of the variable. For example, use total_sales instead of t to represent total sales.

Be Explicit:

Avoid single-letter variable names unless they represent an iterator in a loop (e.g., i, j). Even in loops, try to use descriptive names like index or element.

Use Constants in UPPERCASE:

If you define constants, use UPPERCASE_WITH_UNDERSCORES to differentiate them from regular variables. For example, MAX_LENGTH, PI.

Avoid Acronyms and Abbreviations:

Prefer full words over acronyms or abbreviations unless they are widely understood and accepted in the domain. Clarity is essential.

Avoid Reusing Built-in Names:

Do not use the names of built-in functions or Python keywords as variable names to prevent conflicts.