In [77]:
```
1.Write a Python function that takes a list of numbers and
returns a new list with only the even numbers from the original list.

def even(n):
    even=[]
    odd=[]
    for i in n:
        if i%2==0:
            even.append(i)
        else:
            odd.append(i)
    print(odd)
    print( even)


x=[1,2,3,4,5,67,8,9]
even(x)
```

```
[1, 3, 5, 67, 9]
[2, 4, 8]
```

2.Write a Python program that checks whether a given number is prime using a for loop and if-else statements.

In [50]:
```
def is_prime(number):
    if number <= 1:
        return False
    for i in range(2, int(number ** 0.5) + 1):
        if number % i == 0:
            return False
    return True

number = int(input("Enter a number: "))
if is_prime(number):
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
```

```
4 is not a prime number.
```

3.Create a Python function that takes a dictionary as an argument and returns a new dictionary with keys and values swapped.

In [54]:
```
def swap_keys_and_values(input_dict):
    swapped_dict={value:key for key,value in input_dict.items()}
    return swapped_dict

original_dict={'a':1,'b':4,'c':8}
swapped_dict=swap_keys_and_values(original_dict)
print("Original dictionary:",original_dict)
print("Swapped dictionary:",swapped_dict)
```

```
Original dictionary: {'a': 1, 'b': 4, 'c': 8}
Swapped dictionary: {1: 'a', 4: 'b', 8: 'c'}
```

In [123…
```
def swap(input):
    swapped_dict={}
    for key,value in input.items():
        swapped_dict[value]=key
    return swapped_dict

dict={'name':"vasu",'age':21,'height':5.11}
swap(dict)
```

Out[123…
```
{'vasu': 'name', 21: 'age', 5.11: 'height'}
```

4.Write a Python program using a while loop to calculate the factorial of a given number.

In [60]:
```
n=int(input("enter a number"))
fact=1
count=n
while count>0:
    fact=fact*count
    count-=1

print(f"factorail of {n} is {fact}.")
```

```
factorail of 5 is 120.
```

In [130…
```
n=int(input("enter a number"))
fact=1
for i in range(fact,n):
    fact*=n
```

```
    n-=1

print(fact)
```

```
24
```

5.Write a Python function that takes a list of integers and returns a dictionary with the elements as keys and their frequency counts as values.

In [81]:
```
def count_frequency(n):
    frequency_dict={}
    for i in n:
        if i in frequency_dict:
            frequency_dict[i]+=1
        else:
            frequency_dict[i]=1
    return frequency_dict

list=[1,2,3,4,5,7,8,8]
count_frequency(list)
```

Out[81]:
```
{1: 1, 2: 1, 3: 1, 4: 1, 5: 1, 7: 1, 8: 2}
```

6.Create a Python program that iterates over a set of numbers and prints the numbers that are divisible by both 3 and 5. Use a for loop and if-else statements.

In [83]:
```
def set_numbers(list):
    result=[]
    for i in list:
        if i%3==0 and i%5==0:
            result.append(i)
    return result

x=[3,6,9,10,12,3,55,30,35,33,45]
set_numbers(x)
```

Out[83]:
```
[30, 45]
```

7.

Write a Python function that accepts a list of strings and returns a new list containing only the strings that start with the letter 'a'.

In [90]:
```
def strings(str):
    new_str=[]
    for i in str:
        if i[0]=='a':
            new_str.append(i)
    return new_str

list=["vasu","apple","ajay","sai"]
strings(list)
```

Out[90]:
```
['apple', 'ajay']
```

8.

Write a Python program that iterates through a list of numbers and appends the square of each number to a new list using a for loop and if statement to check if the number is positive.

In [119…
```
def square(numbers):
    squared_positives=[]
    for n in numbers:
        if n>0:
            squared_positives.append(n**2)
    return squared_positives
li=[-1,-3,-4,3,4,5,6,7,8,8]
square(li)
```

Out[119…
```
[9, 16, 25, 36, 49, 64, 64]
```

9. *Temperature Converter*:
   Write a program that converts temperature from Celsius to Fahrenheit and vice versa based on user choice.

In [146…
```
print("Temperature Converter")
print("1: Celsius to Fahrenheit")
print("2: Fahrenheit to Celsius")
```

```
choice = input("Enter your choice (1 or 2): ")

if choice == '1':
    celsius = float(input("Enter temperature in Celsius: "))
    fahrenheit = (celsius * 9/5) + 32
    print(f"{celsius}°C is equal to {fahrenheit:.2f}°F")

elif choice == '2':
    fahrenheit = float(input("Enter temperature in Fahrenheit: "))
    celsius = (fahrenheit - 32) * 5/9
    print(f"{fahrenheit}°F is equal to {celsius:.2f}°C")

else:
    print("Invalid choice. Please enter 1 or 2.")
#
```

```
Temperature Converter
1: Celsius to Fahrenheit
2: Fahrenheit to Celsius
12.0°C is equal to 53.60°F
```

In [162…
```
def celsius_to_fahrenheit(celsius):
    """Convert Celsius to Fahrenheit."""
    return (celsius * 9/5) + 32

def fahrenheit_to_celsius(fahrenheit):
    """Convert Fahrenheit to Celsius."""
    return (fahrenheit - 32) * 5/9

def main():
    print("Temperature Converter")
    print("1: Celsius to Fahrenheit")
    print("2: Fahrenheit to Celsius")

    choice = input("Enter your choice (1 or 2): ")

    if choice == '1':
        celsius = float(input("Enter temperature in Celsius: "))
        fahrenheit = celsius_to_fahrenheit(celsius)
        print(f"{celsius}°C is equal to {fahrenheit:.2f}°F")

    elif choice == '2':
        fahrenheit = float(input("Enter temperature in Fahrenheit: "))
        celsius = fahrenheit_to_celsius(fahrenheit)
        print(f"{fahrenheit}°F is equal to {celsius:.2f}°C")

    else:
        print("Invalid choice. Please enter 1 or 2.")

if __name__ == "__main__":
    main()
```

```
Temperature Converter
1: Celsius to Fahrenheit
2: Fahrenheit to Celsius
1.0°C is equal to 33.80°F
```

10. *Check Even or Odd Number*:

Write a program that asks the user to enter a number and checks if it is even or odd

In [166…
```
n=int(input("enter a number"))
if n%2==0:
    print(f"{n} is a even")
else:
    print(f"{n} is odd")
```

```
12 is a even
```

11. *Check Voting Eligibility*:

Write a program to check if a person is eligible to vote. The person must be at least 18 years old.

In [172…
```
def voting_check(age):
    if age>=18:
        print(f"{age} eligible for vote")
    else:
        print(f"{age} not eligible for vote")

voting_check(18)
```

```
18 eligible for vote
```

12. *Find the Largest Number*:

Write a program that takes three numbers as input and prints the largest one.

In [178…
```
def largest(a,b,c):
    if a>b and a>c:
        print(f"{a} is greater then {b},{c}")
    elif b>a and b>c:
        print(f"{b} is gretaer then {a},{c}")
    else:
        print(f"{c} is greater")

largest(12,12,21)
```

```
21 is greater
```

13. *Divisibility Check*:

Write a program that asks the user for a number and checks if it is divisible by both 3 and 5.

In [198…
```
n=int(input("enter a number"))
if n%3==0 and n%5==0:
    print(f"{n} is divible by both 3 and 5")
elif n%3==0 and n%5!=0:
     print(f"{n} is divible by  3 not 5")
elif n%5==0 and n%3!=0:
     print(f"{n} is divible by 5 not 3")
else:
     print(f"{n} is not divible by both 3 and 5")
```

```
30 is divible by both 3 and 5
```

12. *Leap Year Check*:

Write a program to check if a given year is a leap year.

In [204…
```
def leap_year(year):
    if year%4==0 or year%100==0:
        print(f"{year} is leap year")
    elif year%400==0:
          print(f"{year} is leap year")
    else:
        print(f"{year} is not a leap year")
leap_year(2000)
```

```
2000 is leap year
```

13. *Grade Calculator*:

Write a program that takes a student's marks and prints the grade based on the following conditions:

In [224…
```
def grade(Marks):
    if Marks>=90 and Marks<=100:
        print(f"{Marks} as Grade A")
    elif Marks>=80 and Marks<=89:
        print(f"{Marks} as Grade B")
    elif Marks>=70 and Marks<=79:
        print(f"{Marks} as Grade C")
    elif Marks>=60 and Marks<=69:
        print(f"{Marks} as Grade D")
    else:
        print(f"{Marks} as Fail")

grade(65)
```

```
65 as Grade D
```

14. *Positive, Negative, or Zero*: Write a program that asks the user to input a number and checks whether it is positive, negative, or zero.

In [241…
```
n=eval(input("enter a number"))
if n>0:
    print(f"{n} is a positive number")
elif n<0:
    print(f"{n} is Negative number")
elif n==0:
    print(f"{n} is Zero number")
else:
```

```
    print("enter the valid number")
```

```
7.666666666666667 is a positive number
```

In [3]:
```
# write a program that prints wheather given number divides by 3  and 5 if divided both prints "fizzbuzz" and
# if number divided by 3 and not 5 prints fizz and divided by 5 not 3 prints buzz then no both divides prints va

def fizzbuzz(n1):
    for n in n1:
        if n%3==0 and n%5==0:
            print("fizzbuzz")
        elif n%3==0 and n%5!=0:
            print("fizz")
        elif n%3!=0 and n%5==0:
            print("buzz")
        else:
            print(n)
x=[1,2,3,4,5,6,7,8,9,10,12,15]
y=fizzbuzz(x)
print(y)
```

```
1
2
fizz
4
buzz
fizz
7
8
fizz
buzz
fizz
fizzbuzz
None
```

In [45]:
```
# 2,sum of even numbers get it from range of values into a seperate list and from list print the sum of even va
# import numpy
def sum_even(n):
    li=[]
    for i in range(1,n):
        li.append(i)
    even=[n1 for n1 in li if n1%2==0]
    return sum(even)



sum_even(10)
```

Out[45]: 20

In [53]:
```
def sum_of_even(n):
    l1=[i for i in range(n)]
    even=[n for n in l1 if n%2==0]
    print(sum(even))

sum_of_even(10)
```

```
20
```

In [83]:
```
#print the sum of range of values divided by length of the range of values stored in perticular list from that s
def sum(n):
    list1=[]
    sum=0
    for i in range(n+1):
        list1.append(i)
    for n in list1:
        sum=sum+n
    print(list1)
    return sum/len(list1)


sum(100)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 3
0, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 5
8, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 8
6, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
```

Out[83]: 50.0

In [5]:
```
#adding character prints the trangle shape of string elemnts.
x="python"
y=""
for i in x:
    pass
    y=y+i
    print(y)
```

```
p
py
pyt
pyth
pytho
python
```

In [24]:
```
#find the largest_word from the given file usng function and try except conditions.

try:
    file=open('vasu.txt','w')
    file.write('I am the greate no one beat me')
    file.close()
except Exception as e:
    print("enter the valid values")
```

In [64]:
```
def word():
    try:
        f=open("vasu.txt","r")
        content=f.read()
        print(content)
        words=content.split()


        longest_word=max(words,key=len)
        print("longest_word:",longest_word)
    except Exception as e:
        print("checkout the files not found or exists")
print(word())
```

```
I am the greate no one beat me
longest_word: greate
None
```

In [52]:
```
def longest_word(words):
    return max(words,key=len)
longest_word(["vasu", "nagarjuna", "raviteja"])
```

Out[52]:
```
'nagarjuna'
```

In [ ]:
```
"Bank Account System"
```

In [11]:
```
class BankAccount:
    def __init__(self,balance):
        self._balance=balance
    def deposit(self,amount):
        if amount>0:
            try:
                self._balance+=amount
            except TypeError:
                print("give some valid value in numerics")
        else:
            print("kindly enter the ammount greater then 0")
    def withdraw(self,amount):
        if amount<=self._balance:
            self._balance-=amount
        else:
            print("Insufficiant funds")
    def get_balance(self):
        return self._balance

vasu=BankAccount(100)

vasu.deposit(1000)
vasu.withdraw(200)
vasu.get_balance()
```

Out[11]:
```
900
```

In [ ]:
```
"calculate the area of a circle"
o       Question: Write a program to calculate the area of a circle given its radius.
```

In [21]:
```
import math
```

```
def area_of_circle(radius):
    area=math.pi*(radius**2)
    print(f"area of circle is : {area}")

area_of_circle(10)
```

```
area of circle is : 314.1592653589793
```

In [ ]:
```
"Swap Two Numbers"
o       Question: Write a program to swap two numbers without using a temporary variable.
```

In [23]:
```
def swap_two_numbers(x,y):
    x,y=y,x
    print(f"after swapping: x={x},y={y}")

swap_two_numbers(2,4)
```

```
after swapping: x=4,y=2
```

In [ ]:
```
"check the prime number"
o       Question: Write a program to check if a number is prime.
```

In [106…
```
def prime_check(num):
    if num>1:
        for i in range(2,int(num/2)+1):
            if num%i==0:
                print(f"{num} is not a prime number")
                break
        else:
             print(f"{num} is a prime number")
    else:
        print(f"{num} is not a prime number")

prime_check(2)
```

```
2 is a prime number
```

In [75]:
```
def isprime(n):
    if n<=1:
        return False
    for i in range(2,n):
        if n%i==0:
            print("not a prime")
            break

    else:
        print("prime")

isprime(9)
```

```
not a prime
```

In [ ]:
```
"Factorial of a Number"
o       Question: Write a program to find the factorial of a number.
```

In [110…
```
num=int(input("enter a number"))
fact=1
if num<0:
    print("sorry,factorial does not exist for negative numbers")
elif num==0:
    print( "the factorial of 0 is 1")
else:
    for i in range(1,num+1):
        fact*=i
    print(f"The factorial of {num} is {fact}")
```

```
The factorial of 3 is 6
```

In [112…
```
n=int(input("enter a number"))
fact=1
for i in range(fact,n):
    fact*=n
    n-=1

print(fact)
```

```
120
```

In [ ]:
```
"reverse a string"
o       Question: Write a program to reverse a given string.
```

In [116…
```
def string(str):
```

```
    reversed_string =str[::-1]
    print(f"reversed String:{reversed_string}")

string("vasu")
```

```
reversed String:usav
```

In [ ]:
```
"Operators"
```

o Question: Write a program to perform basic arithmetic operations: addition, subtraction, multiplication, and division.

In [124…
```
a=int(input("enter first number"))
b=int(input("enter second number"))

addition=a+b
substraction=a-b
multiplication=a*b
division=a/b if b!=0 else "undefined"
print(f"Addition: {addition},Substarction : {substraction},multiplication: {multiplication},division:{division}
```

```
Addition: 18,Substarction : 6,multiplication: 72,division:2.0
```

In [ ]:
```
2.      Relational Operators
o       Question: Write a program to compare two numbers using relational operators.
```

In [128…
```
a=int(input("enter a number1"))
b=int(input("enter a number2"))
print(f"a>b:{a>b}")
print(f"a<b:{a<b}")
print(f"a==b:{a==b}")
print(f"a!=0:{a!=b}")
print(f"a>=b:{a>=b}")
print(f"a<=b:{a<=b}")
```

```
a>b:True
a<b:False
a==b:False
a!=0:True
a>=b:True
a<=b:False
```

In [ ]:
```
3.      Logical Operators
o       Question: Write a program to demonstrate the use of logical operators.
o       Answer:
```

In [134…
```
a=True
b=False
print(f"a and b :{a and b}")
print(f"a or b:{a or b}")
print(f"not a:{not a}")
```

```
a and b :False
a or b:True
not a:False
```

In [ ]:
```
4.      Bitwise Operators
o       Question: Write a program to demonstrate the use of bitwise operators.
```

In [136…
```
a=10 #1010 in binary
b=4 #0100 in binary

print(f"a & b: {a & b}")  # Bitwise AND
print(f"a | b: {a | b}")  # Bitwise OR
print(f"a ^ b: {a ^ b}")  # Bitwise XOR
print(f"~a: {~a}")        # Bitwise NOT
print(f"a << 1: {a << 1}") # Left shift
print(f"a >> 1: {a >> 1}") # Right shift
```

```
a & b: 0
a | b: 14
a ^ b: 14
~a: -11
a << 1: 20
a >> 1: 5
```

In [ ]:
```
5.      Assignment Operators
o       Question: Write a program to demonstrate the use of assignment operators.
```

In [138…
```
a=5
print(f"Initial value of a : {a}")
a+=3
print(f"a+=3: {a}")
a-=3
```

```
print(f"a-=3: {a}")
a*=3
print(f"a*=3: {a}")
a/=3
print(f"a/=3: {a}")
```

```
Initial value of a : 5
a+=3: 8
a-=3: 5
a*=3: 15
a/=3: 5.0
```

In [ ]:
```
"Conditional Statements"
1.      If Statement
o       Question: Write a program to check if a number is positive
```

In [148…
```
a=eval(input("enter a number"))
if a>0:
    print(f"yes.{a} is a positive number")
else:
    print(f"No,{a} is not a positive number")
```

```
yes.23.3 is a positive number
```

In [ ]:
```
2.      If-Else Statement
o       Question: Write a program to check if a number is odd or even.
```

In [154…
```
num=int(input("enter a number"))
if num%2==0:
    print(f"{num} is a even number")
else:
    print(f"{num} is a odd number")
```

```
321 is a odd number
```

In [ ]:
```
3.      If-Elif-Else Statement
o       Question: Write a program to check if a number is positive, negative, or zero.
```

In [164…
```
num=int(input("enter a number"))
if num>0:
    print(f"{num} is a positive number")
elif num<0:
    print(f"{num} is negative number")
else:
    print(f"{num} is a zero")
```

```
12 is a positive number
```

In [ ]:
```
4.      Nested If Statement
o       Question: Write a program to determine the largest of three numbers using nested if statements.
```

In [172…
```
a=eval(input("enter a number"))
b=eval(input("enter b number"))
c=eval(input("enter c number"))
if a>=b:
    if a>=c:
       print(f"the largest number is {a}")
    else:
        print(f"The largest number is c: {c}")
elif b>=c:
    print(f"The larsget number is {b}")
else:
    print(f" The largest number is {c}")
```

```
The larsget number is 32
```

In [ ]:
```
5.      If-Else with Multiple Conditions
o       Question: Write a program to check if a year is a leap year.
```

In [180…
```
year=int(input("enter a year"))
if year%4==0 or year%100==0:
    print(f"{year} is a leap year")
elif year%400==0:
    print(f"{year} is a leap year")

else:
    print(f"{year} is not a leap year")
```

```
2024 is a leap year
```

In [ ]:
```
"Loops"
```

In [ ]:
```
1.      For Loop
o       Question: Write a program to print the first 10 natural numbers using a for loop.
```

In [198…
```
for i in  range(1,11):
    print(i)
```

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

In [ ]:
```
2.      Nested For Loop
o       Question: Write a program to print a multiplication table up to 10 using nested for loops.
```

In [204…
```
for i in range(1,11):
    for j in range(1,11):
        print(f"{i} x {j}= {i*j}")
    print("") #blank line for better readability
```

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

2 x 1= 2
2 x 2= 4
2 x 3= 6
2 x 4= 8
2 x 5= 10
2 x 6= 12
2 x 7= 14
2 x 8= 16
2 x 9= 18
2 x 10= 20

3 x 1= 3
3 x 2= 6
3 x 3= 9
3 x 4= 12
3 x 5= 15
3 x 6= 18
3 x 7= 21
3 x 8= 24
3 x 9= 27
3 x 10= 30

4 x 1= 4
4 x 2= 8
4 x 3= 12
4 x 4= 16
4 x 5= 20
4 x 6= 24
4 x 7= 28
4 x 8= 32
4 x 9= 36
4 x 10= 40

5 x 1= 5
5 x 2= 10
5 x 3= 15
5 x 4= 20
5 x 5= 25
5 x 6= 30
5 x 7= 35
5 x 8= 40
5 x 9= 45
5 x 10= 50

6 x 1= 6
6 x 2= 12
6 x 3= 18
```

```
6 x 4= 24
6 x 5= 30
6 x 6= 36
6 x 7= 42
6 x 8= 48
6 x 9= 54
6 x 10= 60

7 x 1= 7
7 x 2= 14
7 x 3= 21
7 x 4= 28
7 x 5= 35
7 x 6= 42
7 x 7= 49
7 x 8= 56
7 x 9= 63
7 x 10= 70

8 x 1= 8
8 x 2= 16
8 x 3= 24
8 x 4= 32
8 x 5= 40
8 x 6= 48
8 x 7= 56
8 x 8= 64
8 x 9= 72
8 x 10= 80

9 x 1= 9
9 x 2= 18
9 x 3= 27
9 x 4= 36
9 x 5= 45
9 x 6= 54
9 x 7= 63
9 x 8= 72
9 x 9= 81
9 x 10= 90

10 x 1= 10
10 x 2= 20
10 x 3= 30
10 x 4= 40
10 x 5= 50
10 x 6= 60
10 x 7= 70
10 x 8= 80
10 x 9= 90
10 x 10= 100
```

In [ ]:
```
3.      For Loop with Break
o       Question: Write a program to search for an element in a list. If found, print "Element found" and stop
```

In [212…
```
list=[1,2,3,4,5,6,7,8,9]
target=int(input("enter the searching number"))
for i in list:
    if i == target:
        print("Element found")
        break
else:

    print("element not found")
```

```
Element found
```

In [ ]:
```
4.      For Loop with Continue
o       Question: Write a program to print all numbers from 1 to 10 except 5 using a for loop and continue stat
```

In [214…
```
for i in range(1,11):
    if i==5:
        continue
    print(i)
```

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

In [ ]:
```
5.      For Loop with Pass
o       Question: Write a program that iterates over a list of numbers and does nothing if the number is even u
```

In [216…
```
list=[1,2,4,5,6,7,8,9,10]
for i in list:
    if i%2==0:
       pass
    else:
        print(f"Odd number: {i}")
```

```
Odd number: 1
Odd number: 5
Odd number: 7
Odd number: 9
```

In [ ]:
```
"while loop"
1.      Simple While Loop
o       Question: Write a program to print numbers from 1 to 5 using a while loop.
```

In [220…
```
i=1
while i<6:
    print(i)
    i+=1
```

```
1
2
3
4
5
```

In [ ]:
```
2.      While Loop with Break
o       Question: Write a program to repeatedly take user input until the user enters 0.
```

In [228…
```
while True:
    num=int(input("Enter a number (0 to stop):"))
    if num==0:
        break
    print(f"you enetered :{num}")
```

```
you enetered :2
you enetered :9
```

In [ ]:
```
3.      While Loop with Continue
o       Question: Write a program to print only odd numbers from 1 to 10 using a while loop and continue statem
```

In [242…
```
i=1
while i<=10:
    if i%2==0:
        i+=1
        continue
    print(i)
    i=i+1
```

```
1
3
5
7
9
```

In [ ]:
```
4.      While Loop with Else
o       Question: Write a program to search for an element in a list using a while loop with an else statement.
```

In [257…
```
list=[1,2,3,4,5,6,7,8]
i=0
target=int(input("enter a search number"))
while i<len(list):
    if list[i]==target:
        print("Element found")
        break
    i+=1
else:
```

```
    print("not found")
```

```
not found
```

In [ ]:
```
5.      While Loop for Factorial Calculation
o       Question: Write a program to calculate the factorial of a number using a while loop.
```

In [275…
```
num=int(input("enter a number"))
fact=1
i=1
while i<=num:
    fact*=i
    i+=1
print(f" the fcatoraila of {num} is {fact}")
```

```
 the fcatoraila of 6 is 720
```

In [ ]:
```
"Type Casting"
1.      String to Integer Conversion
o       Question: Write a program to convert a string containing a number into an integer.
```

In [5]:
```
str_num=eval(input("enter a string"))
int_num=int(str_num)
print(f"String to Integer: {int_num}")
```

```
String to Integer: 3556
```

In [ ]:
```
2.      Integer to Float Conversion
o       Question: Write a program to convert an integer into a float.
```

In [7]:
```
int_num=int(input("enter a integer value"))
float_num=float(int_num)
print(f"integer to float: {float_num}")
```

```
integer to float: 34.0
```

In [ ]:
```
3.      Float to String Conversion
o       Question: Write a program to convert a float into a string.
o       Answer
```

In [5]:
```
float_num=23.23
str_num=str(float_num)
print(f"float to string: {str_num}")
```

```
float to string: 23.23
```

In [ ]:
```
4.      List to Tuple Conversion
o       Question: Write a program to convert a list into a tuple.
```

In [1]:
```
my_list=[1,2,3,4,5]
my_tuple=tuple(my_list)
print(f"List to tuple: {my_tuple}")
```

```
List to tuple: (1, 2, 3, 4, 5)
```

In [ ]:
```
5.      String to List Conversion
o       Question: Write a program to convert a string into a list of characters.
```

In [5]:
```
string1="srinivasulu"
my_list=list(string1)
print(f"string1 to list: {my_list}")
```

```
string1 to list: ['s', 'r', 'i', 'n', 'i', 'v', 'a', 's', 'u', 'l', 'u']
```

In [ ]:
```
"List Operations"
```

In [ ]:
```
1.      List Append
o       Question: Write a program to append an element to a list.
```

In [13]:
```
my_list=[2,3,4,5,2]
my_list.append(23)
print(f"List after append :{my_list}")
```

```
List after append :[2, 3, 4, 5, 2, 23]
```

In [ ]:
```
2.      List Insert
o       Question: Write a program to insert an element at the second position in a list.
```

In [17]:
```
my_list=[23,34,2,4,4]
my_list.insert(2,"vasu")
print(f" my_list after insert: {my_list}")
```

```
 my_list after insert: [23, 34, 'vasu', 2, 4, 4]
```

In [ ]:
```
3.      Nested List
o       Question: Write a program to create a nested list and access an element from
```

In [25]:
```
My_nested_list=[[1,2,4],["vasu","srinu","sai"],[23,32,3]]
print(f"Element at nested_list[1][2]: {My_nested_list[1][1]}")
```

```
Element at nested_list[1][2]: srinu
```

In [ ]:
```
4.      List Pop
o       Question: Write a program to remove and return the last element from a list using pop().
```

In [41]:
```
my_list=[1,2,4,5,6]
popped_element=my_list.pop()
print(f"popped element:{popped_element}")
print(f"List after pop: {my_list}")
```

```
popped element:6
List after pop: [1, 2, 4, 5]
```

In [ ]:
```
5.      List with For Loop
o       Question: Write a program to iterate over a list using a for loop and print each element.
```

In [43]:
```
My_list=[2,3,4,"vasu","sai"]
for i in My_list:
    print(i)
```

```
2
3
4
vasu
sai
```

In [ ]:
```
"Tuple Operations"
1.      Tuple Creation
o       Question: Write a program to create a tuple and print its elements.
```

In [47]:
```
my_tuple=(12,21,"vasu","sai")
print(f"my_tuple:{my_tuple}")
```

```
my_tuple:(12, 21, 'vasu', 'sai')
```

In [ ]:
```
2.      Tuple Operations
o       Question: Write a program to perform concatenation and repetition operations on tuples
```

In [53]:
```
tuple1=(23,2,"vasu","sai34",33)
tuple2=(12,33)
concatenated_tuple=tuple1+tuple2
repeated_tuple=tuple1*2
print(f"Conactenated tuple:{concatenated_tuple}")
print(f"repeated_tuple:{repeated_tuple}")
```

```
Conactenated tuple:(23, 2, 'vasu', 'sai34', 33, 12, 33)
repeated_tuple:(23, 2, 'vasu', 'sai34', 33, 23, 2, 'vasu', 'sai34', 33)
```

In [ ]:
```
3.      Tuple with For Loop
o       Question: Write a program to iterate over a tuple using a for loop and print each element.
```

In [55]:
```
my_tuple=(23,332,"vasu","srinu")
for i in my_tuple:
    print(i)
```

```
23
332
vasu
srinu
```

In [ ]:
```
4.      Tuple Indexing
o       Question: Write a program to access elements from a tuple using positive and negative indexing.
```

In [61]:
```
my_tuple=(23,332,"vasu","srinu")
print(f"Element at index 1:{my_tuple[0]}")
print(f"Element at index -1:{my_tuple[-1]}")
```

```
Element at index 1:23
Element at index -1:srinu
```

In [ ]:
```
"Dictionary Operations"
1.      Dictionary Creation and Access
o       Question: Write a program to create a dictionary and access its elements.
```

In [69]:
```
My_dict={'name':'vasu','age':12,'gender':'male'}
print(f"Name :{My_dict['name']}")
print(f"age: {My_dict['age']}")
print(f"gender:{My_dict['gender']}")
```

```
Name :vasu
age: 12
gender:male
```

In [ ]:
```
2.      Dictionary Insertion
o       Question: Write a program to add a new key-value pair to an existing dictionary.
```

In [73]:
```
My_dict={'name':'vasu','age':12,'gender':'male'}
My_dict['city']='India'
print(f"Updated dictionary:{My_dict}")
```

```
Updated dictionary:{'name': 'vasu', 'age': 12, 'gender': 'male', 'city': 'India'}
```

In [ ]:
```
3.      Dictionary Deletion
o       Question: Write a program to delete a key-value pair from a dictionary.
o       Answer:
```

In [75]:
```
My_dict={'name':'vasu','age':12,'gender':'male'}
del My_dict['age']
print(f"Dictionary after deletion:{My_dict}")
```

```
Dictionary after deletion:{'name': 'vasu', 'gender': 'male'}
```

In [ ]:
```
4.      Dictionary with For Loop
o       Question: Write a program to iterate over a dictionary and print each key-value pair.
```

In [87]:
```
my_dict={'name': 'vasu', 'age': 12, 'gender': 'male', 'city': 'India'}
for key,value in my_dict.items():
    print(f"{key}:{value}")
```

```
name:vasu
age:12
gender:male
city:India
```

In [ ]:
```
5.      Check Key in Dictionary
o       Question: Write a program to check if a specific key exists in a dictionary.
```

In [93]:
```
my_dict={'name': 'vasu', 'age': 12, 'gender': 'male', 'city': 'India'}
key_to_check='age'
if key_to_check in my_dict:
    print(f"'{key_to_check}' exists in the dictionary with value:{my_dict[key_to_check]}")

else:
    print(f"{key_to_check} does not exist in the dictionary.")
```

```
'age' exists in the dictionary with value:12
```

In [ ]:
```
"Set Operations"
1.      Set Creation and Access
o       Question: Write a program to create a set and print its elements.
```

In [95]:
```
my_set={1,2,3,4}
print(f"My_set:{my_set}")
```

```
My_set:{1, 2, 3, 4}
```

In [ ]:
```
2.      Set Union and Intersection
o       Question: Write a program to perform union and intersection operations on two sets.
```

In [107…
```
set1={1,2,4,5}
set2={3,4,5,3}
union_set=set1.union(set2)
intersection_set=set1.intersection(set2)
print(f"Union :{union_set}")
print(f"Intersection:{intersection_set}")
```

```
Union :{1, 2, 3, 4, 5}
Intersection:{4, 5}
```

In [ ]:
```
3.      Set Difference
o       Question: Write a program to find the difference between two sets.
```

In [121…
```
set1={1,2,4,5}
set2={3,4,5,3}
difference=set1.difference(set2)

print(f"difference:{difference}")
```

```
difference:{1, 2}
```

In [ ]:
```
4.      Set Symmetric Difference
o       Question: Write a program to find the symmetric difference between two sets.
```

In [117…
```
set1={1,2,4,5}
set2={3,4,5,3}
sym_diff=set1.symmetric_difference(set2)
print(f"symmteric_difference:{sym_diff}")
```

```
symmteric_difference:{1, 2, 3}
```

In [ ]:
```
5.      Set Update
o       Question: Write a program to update a set with another set using the update() method.
```

In [123…
```
set1={1,2,4,5}
set2={3,4,5,3}
set1.update(set2)
print(f"updated set:{set1}")
```

```
updated set:{1, 2, 3, 4, 5}
```

In [ ]:
```
"functions"
```

In [ ]:
```
1.      Function Definition
o       Question: Write a simple function to calculate the square of a number.
```

In [23]:
```
def square(num):
    return num*num
result=square(12)
print(f"square of 5:{result}")
```

```
square of 5:144
```

In [ ]:
```
2.      Function with Arguments
o       Question: Write a function that takes two numbers and returns their sum.
```

In [27]:
```
def sum(num1,num2):
    return num1+num2

result=sum(12,32)
print(f"sum  is :{result}")
```

```
sum  is :44
```

In [ ]:
```
3.      Function with Default Arguments
o       Question: Write a function that greets a person. If no name is provided, it should greet with "Hello, W
```

In [35]:
```
def greet(name="World"):
    print(f"Hello,{name}!")

greet() #default greeting
greet("Alice") #custome greeting
```

```
Hello,World!
Hello,Alice!
```

In [ ]:
```
4.      Function with Return Values
o       Question: Write a function that returns the factorial of a given number.
```

In [57]:
```
def fact(n):
    fact=1
    if n==0 or n==1:
        return 1
    else:
           for i in range(1,n+1):
               fact=fact*i
           return fact

fact(3)
```

Out[57]:
```
6
```

In [53]:
```
def factorial(n):
    if n==0 or n==1:
        return 1
    else:
        return n*factorial(n-1)

result=factorial(5)
print(f"factorial of 5 : {result}")
```

```
factorial of 5 : 120
```

In [ ]:
```
5.      Function with Variable Length Arguments
o       Question: Write a function that takes a variable number of arguments and returns their sum.
```

In [2]:
```
def custom_sum(*args):
```

```
    return sum(args)

# Call the function
result = custom_sum(1, 2, 3, 4, 5)
print(f"Sum: {result}")
```

```
Sum: 15
```

In [ ]:
```
"Exception Handling"
1.      Basic Try-Except
o       Question: Write a program that handles division by zero using try-except.
```

In [6]:
```
try:
    result=10/0
except ZeroDivisionError as e:
    print("Cannot divide by zero!",e)
```

```
Cannot divide by zero! division by zero
```

In [ ]:
```
2.      Multiple Exceptions
o       Question: Write a program that handles both ValueError and ZeroDivisionError.
```

In [18]:
```
try:
    num=int(input("enter a number:"))
    result=10/0
except ValueError as v:
    print("Invalid value given !Please enter a number.")

except ZeroDivisionError:
    print("denominator cannot be zero")
```

```
denominator cannot be zero
```

In [ ]:
```
3.      Finally Clause
o       Question: Write a program that demonstrates the use of the finally clause.
```

In [35]:
```
try:
    result=12/2
    print(result)
except ZeroDivisionError:
    print("denominator cannot be a zero")

finally:
    print("executed succesfully")
```

```
6.0
executed succesfully
```

In [ ]:
```
4.      Custom Exception
o       Question: Write a program that raises a custom exception if a number is negative.
```

In [65]:
```
class NegativeNumberError(Exception):
    pass

def check_positive(number):
    if number < 0:
        raise NegativeNumberError("Negative number detected!")
    else:
        return number

try:
    check_positive(-5)
except NegativeNumberError as e:
    print(e)
```

```
Negative number detected!
```

In [ ]:
```
5.      Raise Exception
o       Question: Write a program that raises a ValueError if the input is not an integer.
```

In [82]:
```
def check_integer(num):
    if not isinstance(num,int):
        raise ValueError("Input is not an integer!")
    return num
try:
    check_integer("abc")

except ValueError as e:
    print(e)
```

```
Input is not an integer!
```

In [ ]:
```
File Handling
1.      File Read
o       Question: Write a program to read the content of a text file and print it.
```

In [88]:
```
with open('vasu.txt','r') as file:
    content=file.read()
    print(content)
```

```
I am the greate no one beat me
```

In [ ]:
```
2.      File Write
o       Question: Write a program to write a string to a text file.
o       Answer:
```

In [92]:
```
with open('vasu.txt','w') as file:
    file.write("I am your big fan")
```

In [ ]:
```
3.      File Append
o       Question: Write a program to append a string to an existing text file.
```

In [98]:
```
with open('vasu.txt','a') as file:
    file.write("\nHi I love")
```

In [ ]:
```
4.      File Read Line by Line
o       Question: Write a program to read a file line by line and print each line.
```

In [119…
```
with open('vasu.txt','r') as file:

    for line in file:
       print(line,end='')
```

```
I am your big fanHi I love
Hi I love
```

In [ ]:
```
5.      File Handling with Exception
o       Question: Write a program to handle file not found exception while trying to open a file.
```

In [145…
```
try:
    with open('file.txt') as file:
        file.read()
except (SyntaxError,FileNotFoundError):
    print("file is found")
```

In [ ]:
```
"Class and Objects"

1.      Basic Class
o       Question: Write a class Person with attributes name and age. Create an object and print its attributes.
```

In [7]:
```
class Person:
    def __init__(self,name,age):
        self.name=name
        self.age=age
obj=Person("vasu",21)
print(f"Name: {obj.name}")
print(f"age:{obj.age}")
```

```
Name: vasu
age:21
```

In [ ]:
```
2.      Class with Method
o       Question: Write a class Circle with a method to calculate the area of the circle.
```

In [11]:
```
class Circle:
    def __init__(self,radius):
        self.radius=radius
    def area(self):
        return 3.14*self.radius*self.radius

circle=Circle(2)
print(f"Area of circle :{circle.area()}")
```

```
Area of circle :12.56
```

In [ ]:
```
3.      Encapsulation
o       Question: Write a class BankAccount with private attributes balance. Provide methods to deposit and wit
```

In [43]:
```
class BankAccount:
    def __init__(self,balance):
        self._balance=balance
    def deposit(self,amount):
```

```
        self._balance+=amount
    def withdraw (self,amount):
        if self._balance>=amount:
            self._balance-=amount
            return amount
        else:
            return "Insufficiant balance"
    def get_balance(self):
        return self._balance

account=BankAccount(100)
account.deposit(10000)
print(f"Balance: {account.get_balance()}")
account.withdraw(100)
print(f"Balance:{account.get_balance()}")
```

```
Balance: 10100
Balance:10000
```

4. Polymorphism

o Question: Write a program to demonstrate polymorphism with a method named area in different classes.

In [47]:
```
class Rectangle:
    def __init__(self,length,breadth):
        self.length=length
        self.breadth=breadth
    def area(self):
        return self.length*self.breadth
class Circle:
    def __init__(self,radius):
        self.radius=radius
    def area(self):
        return 3.14*self.radius**2

shapes=[Rectangle(10,5),Circle(7)]
for shape in shapes:
    print(f"Area:{shape.area()}")
```

```
Area:50
Area:153.86
```

In [ ]:
```
5.      Inheritance (Single Inheritance)
o       Question: Write a program to demonstrate single inheritance with a base class Animal and a derived class
```

In [61]:
```
class Animal:
    def __init__(self,name):
        self.name=name
    def speak(self):
        return "Animal sound"
class Dog(Animal):
    def speak(self):
        return "Bark"

dog=Dog("Buddy")
print(f"{dog.name} says {dog.speak()}")
```

```
Buddy says Bark
```

6. Inheritance (Multiple Inheritance)

o Question: Write a program to demonstrate multiple inheritance with classes Parent1, Parent2, and Child.

In [73]:
```
class Parent1:
    def function1(self):
        return "function 1"
class Parent2:
    def function2(self):
        return "function2"

class Child(Parent1,Parent2):
    pass

child=Child()
print(f"child from Perent1 ,{child.function1()}")
print(f"child from Perent2 ,{child.function2()}")
```

```
child from Perent1 ,function 1
child from Perent2 ,function2
```

In [ ]:
```
7.      Inheritance (Multilevel Inheritance)
o       Question: Write a program to demonstrate multilevel inheritance with classes Grandparent, Parent, and Cl
```

In [87]:
```
class GrandParent:
    def grandparent_function(self):
        return "I a grandfather for you"
class Parent(GrandParent):
    def parent(self):
        return "I am a Parent for you"

class Child(Parent):
    def child(self):
        return "I am children for you both"

child=Child()
print(child.parent())
print(child.grandparent_function())
print(child.child())
```

```
I am a Parent for you
I a grandfather for you
I am children for you both
```

8. Inheritance (Hierarchical Inheritance)

o Question: Write a program to demonstrate hierarchical inheritance with a base class Vehicle and derived classes Car and Bike.

In [109…
```
class vehicle:
    def __init__(self,make):
        self.make=make
    def info(self):
        return f"Make: {self.make}"
class Car(vehicle):
    def car(self,name):
        return f"make:{self.make},{name}"
class Bike(vehicle):
    def bike(self):
        return f"{self.make}"

car=Car("Toyota")
bike=Bike("yamaha")

print(car.info())
print(car.car("benz"))

print(bike.info())
print(bike.bike())
```

```
Make: Toyota
make:Toyota,benz
Make: yamaha
yamaha
```

9. Inheritance (Hybrid Inheritance)

o Question: Write a program to demonstrate hybrid inheritance involving multiple base and derived classes.

In [129…
```
class Base1:
    def base1_function(self):
        return "Base1 function"

class Base2:
    def base2_function(self):
        return "Base2_function"

class Derived1(Base1,Base2):
    def derived1_function(self):
        return "Derived1 Function"

class Derived2(Derived1):
    def derived2_function(self):
        return "Derived1 Function"
derived=Derived2()
print(derived.base1_function())
print(derived.base2_function())
print(derived.derived1_function())
print(derived.derived2_function())
```

```
Base1 function
Base2_function
Derived1 Function
Derived1 Function
```

In [ ]:
```
1.Single Inheritence
```

Example 1: Basic Inheritance

In [131…
```
class Animal:
    def sound(self):
        return "some sound"

class Dog(Animal):
    def sound(self):
        return "Bow Bow"

dog=Dog()
dog.sound()
```

Out[131… 'Bow Bow'

In [ ]: Example 2: Inheriting Methods

In [133…
```
class  vehicle:
    def start(self):
        return "vehicle started"

class Car(vehicle):
    def accelerate(self):
        return "car is accelerating"

car=Car()
print(car.start())
print(car.accelerate())
```

```
vehicle started
car is accelerating
```

In [ ]: Example 3: Overriding Methods

In [153…
```
class Bird:
    def fly(self):
        return "Flying"

class Penguine(Bird):
    def fly(self):

        return "Cannot Fly"

penguine=Penguine()
print(penguine.fly())
```

```
Cannot Fly
```

In [ ]: Example 4: Using super()

In [143…
```
class Person:
    def greet(self):
        return "Hello"

class Employee(Person):
    def greet(self):
        original_greet=super().greet()
        return f"{original_greet},I am an employee"

employee=Employee()
print(employee.greet())
```

```
Hello,I am an employee
```

In [ ]: Example 5: Real-world Example - Bank Account

In [161…
```
class Account:
    def __init__(self,owner,balance=0):
        self.owner=owner
        self.balance=balance
    def deposit(self,amount):
        self.balance+=amount
        return self.balance
class SavingsAccount(Account):
    def add_interest(self,rate):
        self.balance+=self.balance*rate
        return self.balance

savings=SavingsAccount("vasu",10000)
savings.deposit(500)
print(savings.add_interest(0.05))
```

```
11025.0
```

In [ ]:
```
2.Multiple Inheritence

Example 1: Basic Multiple Inheritence
```

In [3]:
```
class Father:
    def tall(self):
        return "Tall"

class Mother:
    def beautiful(self):
        return "Beautiful"

class Child(Father,Mother):
    pass

child=Child()
print(child.tall())
print(child.beautiful())
```

```
Tall
Beautiful
```

In [ ]:
```
Example2.Method Resolution Order(MRO)
```

In [9]:
```
class A:
    def method(self):
        return "A"

class B(A):
    def method(self):
        return "B"

class C(A):
    def method(self):
        return "C"

class D(B,C):
    pass

d=D()
print(d.method())
print(D.__mro__)
print(D.mro())
```

```
B
(<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class '__main__.A'>, <class 'object'>)
[<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class '__main__.A'>, <class 'object'>]
```

In [ ]:
```
Example 3. Inheriting from Multiple Parents.
```

In [15]:
```
class Writer:
    def write(self):
        return "Writing"

class Painter:
    def paint(self):
        return "Painter"

class Artist(Writer,Painter):
    pass

artist=Artist()
print(artist.write())
print(artist.paint())
```

```
Writing
Painter
```

In [ ]:
```
Example 4: Diamond Problem and super()
```

In [23]:
```
class A:
    def __init__(self):
        print("A's init called")

class B(A):
    def __init__(self):
        super().__init__()
        print("B's init called")

class C(A):
    def __init__(self):
        super().__init__()
        print("C's init called")
```

```
class D(B,C):
    def __init__(self):
        super().__init__()
        print("D's init called")

d=D()
```

```
A's init called
C's init called
B's init called
D's init called
```

In [ ]:
```
Example 5: Real-world Example - Multi-skilled Worker
```

In [25]:
```
class Electrician :
    def work(self):
        return "Fixing electrical issues"

class Plumber:
    def work(self):
        return "Fixing plumbing issues"

class Handyman(Electrician,Plumber):
    def work(self):
        return f"{Electrician.work(self)},{Plumber.work(self)}"

handyman=Handyman()
print(handyman.work())
```

```
Fixing electrical issues,Fixing plumbing issues
```

In [ ]:
```
"3.Multileval Inheritence"
```

In [ ]:
```
Example 1.Basic Multiple Inheritence
```

In [31]:
```
class Animal:
    def eat(self):
        return "Eating"

class Manmmal(Animal):
    def drink(self):
        return "Drinking"

class Dog(Manmmal):
    def bark(self):
        return "Barking"

dog=Dog()
print(dog.eat())
print(dog.drink())
print(dog.bark())
```

```
Eating
Drinking
Barking
```

In [ ]:
```
Example 2: Overriding in Multilevel Inheritance
```

In [35]:
```
class A:
    def show(self):
        return "A's show"
class B(A):
    def show(self):
        return "B's show"
class C(B):
    def show(self):
        return "C's show"

c=C()
print(c.show())
```

```
C's show
```

In [ ]:
```
Example 3: Calling Parent Method with super()
```

In [41]:
```
class Vehicle:
    def move(self):
        return "Vehicle moving"
```

```
class Car(Vehicle):
    def move(self):
        return super().move()+" on raod"

class SportsCar(Car):
    def move(self):
        return super().move()+" with speed"

sports_car=SportsCar()
print(sports_car.move())
```

```
Vehicle moving on raod with speed
```

In [ ]:
```
Example 4: Multilevel Inheritance with Init Method
```

In [43]:
```
class A:
    def __init__(self):
        print("A's init")

class B(A):
    def __init__(self):
        super().__init__()
        print("B's init")

class C(B):
    def __init__(self):
        super().__init__()
        print("C's init")

c=C()
```

```
A's init
B's init
C's init
```

In [ ]:
```
Example 5: Real-world Example - Organization Hierarchy
```

In [55]:
```
class Employee:
    def __init__(self,name):
        self.name=name

class Manager(Employee):
    def __init__(self,name,department):
        super().__init__(name)
        self.department=department

class Director(Manager):
    def __init__(self,name,department,region):
        super().__init__(name,department)
        self.region=region
director=Director("vasu","IT","Banglore")
print(director.name)
print(director.department)
print(director.region)
```

```
vasu
IT
Banglore
```

In [ ]:
```
"4.Hybrid Inheritence"
```

In [ ]:
```
Example 1. Combination of Single and Multiple Inheritence
```

In [67]:
```
class A:
    def method_A(self):
       return "A"

class B(A):
    def method_B(self):
        return "B"

class C(A):
    def method_C(self):
        return "C"

class D(B,C):
    def method_D(self):
        return "D"

d=D()
print(d.method_A())
print(d.method_B())
print(d.method_C())
```

```
print(d.method_D())
```

```
A
B
C
D
```

In [ ]:
```
Example 2: Hybrid with Multiple Parents
```

In [73]:
```
class X:
    def x_method(self):
        return "Method from X"

class Y:
    def y_method(self):
        return "Method from Y"

class Z(X,Y):
    def z_method(self):
        return "Method from Z"

class A(Z):
    def a_method(self):
        return "Method from A"

a=A()
print(a.x_method())
print(a.y_method())
print(a.z_method())
print(a.a_method())
```

```
Method from X
Method from Y
Method from Z
Method from A
```

In [ ]:
```
Example 3: Combining Hierarchical and Multiple Inheritance
```

In [79]:
```
class A:
    def method_A(self):
        return "A"

class B(A):
    def method_B(self):
        return "B"

class C(A):
    def method_C(self):
        return "C"

class D(B,C):
    def method_D(self):
        return "D"

class E(D):
    def method_E(self):
        return "E"

e=E()
print(e.method_A())
print(e.method_B())
print(e.method_C())
print(e.method_D())
print(e.method_E())
```

```
A
B
C
D
E
```

In [ ]:
```
4. Hybrid Inheritance (continued)
Example 4: Real-world Example - Educational Structure
```

In [43]:
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

class Student(Person):
    def __init__(self, name, age, student_id):
        super().__init__(name, age)
        self.student_id = student_id
```

```
class Teacher(Person):
    def __init__(self, name, age, employee_id):
        super().__init__(name, age)
        self.employee_id = employee_id

class TeachingAssistant(Student, Teacher):
    def __init__(self, name, age, student_id, employee_id):
        Student.__init__(self, name, age, student_id)
        Employee.__init__(self, name, age, employee_id)

ta = TeachingAssistant("John Doe", 24, "S123", "T456")
print(f"Name: {ta.name}, Age: {ta.age}, Student ID: {ta.student_id}, Employee ID: {ta.employee_id}")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[43], line 21
     18         Student.__init__(self, name, age, student_id)
     19         Employee.__init__(self, name, age, employee_id)
---> 21 ta = TeachingAssistant("John Doe", 24, "S123", "T456")
     22 print(f"Name: {ta.name}, Age: {ta.age}, Student ID: {ta.student_id}, Employee ID: {ta.employee_id}")

Cell In[43], line 18, in TeachingAssistant.__init__(self, name, age, student_id, employee_id)
     17 def __init__(self, name, age, student_id, employee_id):
---> 18     Student.__init__(self, name, age, student_id)
     19     Employee.__init__(self, name, age, employee_id)

Cell In[43], line 8, in Student.__init__(self, name, age, student_id)
      7 def __init__(self, name, age, student_id):
----> 8     super().__init__(name, age)
      9     self.student_id = student_id

TypeError: Teacher.__init__() missing 1 required positional argument: 'employee_id'
```

In [ ]:
```
Example 5: Complex Hybrid Structure
```

In [9]:
```
class Engine:
    def engine_type(self):
        return "V8"

class Car(Engine):
    def wheels(self):
        return 4

class Boat(Engine):
    def propeller(self):
        return 1

class AmphibiousVehicle(Car,Boat):
    def vehicle_type(self):
        return "Amphibious Vehicle"

vehicle=AmphibiousVehicle()
print(vehicle.engine_type())
print(vehicle.wheels())
print(vehicle.propeller())
print(vehicle.vehicle_type())
```

```
V8
4
1
Amphibious Vehicle
```

In [ ]:
```
"5.Heirarachical Inheritence"
```

In [ ]:
```
Example 1: Basic Hierarchical Inheritance
```

In [19]:
```
class Animal:
    def sound(self):
        return "some sound"

class Dog(Animal):
    def sound(self):
        return "Bark"

class Cat(Animal):
    def sound(self):
        return "Meow"
dog=Dog()
cat=Cat()
print(dog.sound())
print(cat.sound())
```

```
Bark
Meow
```

In [ ]:
```
Example 2: Shared Method in Parent Class
```

In [21]:
```
class Person:
    def greet(self):
        return "Hello"

class Student(Person):
    def study(self):
        return "Studying"

class Teacher(Person):
    def teach(self):
        return "Teaching"

student=Student()
teacher=Teacher()
print(student.greet())
print(student.study())
print(teacher.greet())
print(teacher.teach())
```

```
Hello
Studying
Hello
Teaching
```

In [ ]:
```
Example 3: Adding More Children
```

In [23]:
```
class Device:
    def turn_on(self):
        return "Device is now on"

class Phone(Device):
    def call(self):
        return "Calling"

class Laptop(Device):
    def code(self):
        return "Coding"

class Tablet(Device):
    def draw(self):
        return "Drawing"

phone=Phone()
laptop=Laptop()
tablet=Tablet()
print(phone.turn_on())
print(phone.call())
print(laptop.turn_on())
print(laptop.code())
print(tablet.turn_on())
print(tablet.draw())
```

```
Device is now on
Calling
Device is now on
Coding
Device is now on
Drawing
```

In [ ]:
```
Example 4: Real-world Example - Employee Types
```

In [33]:
```
class Employee:
    def __init__(self,name,emp_id):
        self.name=name
        self.emp_id=emp_id
    def work(self):
        return "Working"

class Developer(Employee):
    def code(self):
        return "Writing Code"

class Designer(Employee):
    def design(self):
        return "Designing graphics"

class Manager(Employee):
    def manage(self):
```

```
        return "Managing team"

developer=Developer("vasu","12BJ")
designer=Designer("sai","13BJ")
manager=Manager("mani","14BJ")

print(developer.work())
print(developer.code())
print(designer.work())
print(designer.design())
print(manager.work())
print(manager.manage())
```

```
Working
Writing Code
Working
Designing graphics
Working
Managing team
```

In [ ]:
```
Example 5: Hierarchical Inheritance with Additional Methods
```

In [35]:
```
class Vehicle:
    def start(self):
        return "Starting engine"

class Car(Vehicle):
    def drive(self):
        return "Car Driving"

class Bike(Vehicle):
    def ride(self):
        return "Bike Riding"

car=Car()
bike=Bike()
print(car.start())
print(car.drive())
print(bike.start())
print(bike.ride())
```

```
Starting engine
Car Driving
Starting engine
Bike Riding
```

In [ ]:
```
"6.Datetime functions"
```

In [ ]:
```
Example 1: Getting Current Date and Time
```

In [9]:
```
from datetime import datetime
now=datetime.now()
print("Current Date and Time: ",now)
```

```
Current Date and Time:  2024-09-10 11:15:25.167724
```

In [ ]:
```
Example 2: Formatting Date and Time
```

In [15]:
```
from datetime import datetime
now=datetime.now()
formatted_date=now.strftime("%Y/%m/%d %H:%M:%S")
print("Formatted Date and Time :",formatted_date)
```

```
Formatted Date and Time : 2024/09/10 11:17:07
```

In [ ]:
```
Example 3: Parsing Date Strings
```

In [23]:
```
from datetime import datetime
date_string = "2024-08-31 10:30:00"
date_object = datetime.strptime(date_string, "%Y-%m-%d %H:%M:%S")
print("Parsed Date Object:", date_object)
```

```
Parsed Date Object: 2024-08-31 10:30:00
```

In [ ]:
```
Example 4: Date Arithmetic
```

In [25]:
```
from datetime import datetime,timedelta

now=datetime.now()
future_date=now+timedelta(days=10)
print("current date:",now)
print("future_date:",future_date)
```

```
current date: 2024-09-10 11:22:12.809198
future_date: 2024-09-20 11:22:12.809198
```

In [ ]:
```
Example 5: Past date
```

In [31]:
```
now=datetime.now()
past_date=now-timedelta(days=10)
print("current date:",now)
print("future_date:",past_date)
```

```
current date: 2024-09-10 11:23:31.659230
future_date: 2024-08-31 11:23:31.659230
```

In [ ]:
```
Example 5: Real-world Example - Time Difference Calculation
```

In [47]:
```
from datetime import datetime

start_time=datetime(2024,8,31,9,0,0)
end_time=datetime(2024,8,31,20,0,0)
work_duration=(end_time-start_time)
print("work_duration:",work_duration)
```

```
work_duration: 11:00:00
```

In [ ]:
```

```

In [ ]:
```
"7.Iterator"
```

In [ ]:
```
Example 1: Custom Iteration Class
```

In [55]:
```
class MyNumbers:
    def __iter__(self):
        self.a=1
        return self
    def __next__(self):
        x=self.a
        self.a+=1
        return x

mynum=MyNumbers()
my_iter=iter(mynum)
print(next(my_iter))
print(next(my_iter))
print(next(my_iter))
```

```
1
2
3
```

In [ ]:
```
Example 2: Iterating Over a String
```

In [63]:
```
my_string="srinuvasulu"
my_iter=iter(my_string)

print(next(my_iter))
print(next(my_iter))
print(next(my_iter))
print(next(my_iter))
print(next(my_iter))
```

```
s
r
i
n
u
```

In [ ]:
```
Example 3: Creating a Reverse Iterator
```

In [69]:
```
class Reverse:
    def __init__(self,data):
        self.data=data
        self.index=len(data)
    def __iter__(self):
        return self
    def __next__(self):
        if self.index==0:
            raise StopIteration
        self.index-=1
        return self.data[self.index]

rev=Reverse('Python')
for char in rev:
    print(char,end=" ")
```

```
n o h t y P
```

In [ ]:
```
Example 4: Real-world Example - Fibonacci Iterator
```

In [77]:
```
class Fibonacci:
    def __init__(self,max):
        self.max=max
    def __iter__(self):
        self.a,self.b=0,1
        return self
    def __next__(self):
        fib=self.a
        if fib>self.max:
            raise StopIteration
        self.a,self.b=self.b,self.a+self.b
        return fib

fib_iter=iter(Fibonacci(10))
for num in fib_iter:
    print(num,end=" ")
```

```
0 1 1 2 3 5 8
```

In [ ]:
```
Example 5: Using iter() with Sentinel
```

In [81]:
```
with open('vasu.txt','r') as file:
    for line in iter(file.readline,''):
        print(line,end=" ")
```

```
I am your big fanHi I love
 Hi I love
```

In [ ]:
```
8.Generator
```

In [ ]:
```
Example 1.Basic Generator Function
```

In [83]:
```
def simple_generator():
    yield 1
    yield 2
    yield 3

for i in simple_generator():
    print(i)
```

```
1
2
3
```

In [ ]:
```
Example 2: Generator for Fibonacci Sequence
```

In [91]:
```
def fibonacci(max_value):
    a,b=0,1
    while a<=max_value:
        yield a
        a,b=b,a+b


for i in fibonacci(10):
    print(i,end=" ")
```

```
0 1 1 2 3 5 8
```

In [ ]:
```
Example 3: Infinite Generator
```

In [95]:
```
def infinite_generator():
    i=0
    while True:
        yield i
        i+=1
gen=infinite_generator()
print(next(gen))
print(next(gen))
print(next(gen))
print(next(gen))
```

```
0
1
2
3
```

In [ ]:
```
Example 4: Generator Expression
```

In [109…
```
squares=(x*x for x in range(5))

for sqaure in squares:
    print(sqaure,end=" ")
```

```
0 1 4 9 16
```

In [ ]:
```
Example 5: Real-world Example - Reading Large File
```

In [117…
```
def read_large_file(file_path):
    with open(file_path,'r') as file:
        while True:
            data=file.readline()
            if not data:
                  break
            yield data
for line in read_large_file('vasu.txt'):
    print(line,end=" ")
```

```
I am your big fanHi I love
 Hi I love
```

In [ ]:
```
"9.Decorator"
```

In [ ]:
```
Example 1: Basic Decorator
```

In [126…
```
def my_decorator(func):
    def wrapper():
        print("Hi ,Hello Namsthey vanakkam nee mee srinivas")
        func()
        print("I am data science ,am i data analytics ")
    return wrapper

@my_decorator
def say_hello():
    print("Hello")

say_hello()
```

```
Hi ,Hello Namsthey vanakkam nee mee srinivas
Hello
I am data science ,am i data analytics
```

In [ ]:
```
Example 2: Decorator with Arguments
```

In [128…
```
def my_decorator(func):
    def wrapper(*args,**kwargs):
        print("Arguments pass:",args,kwargs)
        return func(*args,**kwargs)
    return wrapper

@my_decorator
def add(a,b):
    return a+b

print(add(23,43))
```

```
Arguments pass: (23, 43) {}
66
```

In [ ]:
```
Example 3: Chaining Decorators
```

In [132…
```
def uppercase_decorator(func):
    def wrapper():
        result=func()
        return result.upper()
    return wrapper

def split_string_decorator(func):
    def wrapper():
        result=func()
        return result.split()
    return wrapper
@split_string_decorator
@uppercase_decorator
def say_hello():
    return "Hello WOrld Namasthey"

print(say_hello())
```

```
['HELLO', 'WORLD', 'NAMASTHEY']
```

In [ ]:
```
Example 4: Real-world Example - Timing Function Execution
```

In [138…
```
import time
def timer_decorator(func):
    def wrapper(*args,**kwargs):
        start_time=time.time()
        result=func(*args,**kwargs)
        end_time=time.time()
        print(f"Execution time: {end_time-start_time} seconds")
        return result
    return wrapper

@timer_decorator
def slow_function():
    time.sleep(2)
    print("Function complete")

slow_function()
```

```
Function complete
Execution time: 2.0076091289520264 seconds
```

In [ ]:
```
Example 5: Caching Results with Decorator
```

In [148…
```
def cache_decorator(func):
    cache={}
    def wrapper(*args):
        if args in cache:
            return cache[args]
        result=func(*args)
        cache[args]=result
        return result
    return wrapper

@cache_decorator
def slow_addition(a,b):
    time.sleep(1)
    return a+b

print(slow_addition(3,5))
print(slow_addition(3,5))
```

```
8
8
```

In [ ]:
```
"10.Deque(Double-Ended Queue)
Example 1. Basic opeartions with deque
```

In [156…
```
from collections import deque
d=deque([])
d.append(1)
d.append(2)
d.appendleft(3)
print(d)
```

```
deque([3, 1, 2])
```

In [158…
```
d.pop()
```

Out[158… 2

In [160…
```
d.popleft()
```

Out[160… 3

In [162…
```
print(d)
```

```
deque([1])
```

In [ ]:
```
Example 2: Rotating Elements in Deque
```

In [224…
```
from collections import deque
d=deque([1,2,3,4,5,6])
d.rotate(2)
print(d) #two values comes to left side
```

```
deque([5, 6, 1, 2, 3, 4])
```

In [214…
```
d.rotate(-1)
print(d) #randomly splitting moving
```

```
deque([1, 2, 3, 4, 5, 6])
```

In [ ]:
```
Example 3: Limiting Deque Size
```

In [230…
```
from collections import deque
d=deque(maxlen=3)
d.append(1)
d.append(2)
d.append(3)

print(d)
```

```
deque([1, 2, 3], maxlen=3)
```

In [232…
```
from collections import deque
d=deque(maxlen=3)
d.append(1)
d.append(2)
d.append(3)
d.append(4)
print(d)
```

```
deque([2, 3, 4], maxlen=3)
```

In [ ]:
```
Example 4: Real-world Example - Browser History
```

In [3]:
```
from collections import deque

class BrowserHistory:
    def __init__(self,capacity=5):
        self.history=deque(maxlen=capacity)
    def visit(self,site):
        self.history.append(site)
        print(f"visisted {site}")
    def back(self):
        if self.history:
            site=self.history.pop()
            print(f"Back to {site}")
        else:
            print("No history")

browser=BrowserHistory()
browser.visit("google.com")
browser.visit("stackoverflow.com")
browser.visit("w3schoold.com")
browser.visit("github.com")
browser.visit("Linkedlin.com")
browser.visit("youtube.com")
browser.back()
```

```
visisted google.com
visisted stackoverflow.com
visisted w3schoold.com
visisted github.com
visisted Linkedlin.com
visisted youtube.com
Back to youtube.com
```

In [260…
```
browser.back()
```

```
Back to Linkedlin.com
```

In [262…
```
browser.back()
```

```
Back to github.com
```

In [264…
```
browser.back()
```

```
Back to w3schoold.com
```

In [ ]:
```
Example 5: Implementing a Queue with Deque
```

In [5]:
```
class Queue:
    def __init__(self):
        self.queue=deque()
    def enqueue(self,item):
        self.queue.append(item)
    def dequeue(self):
        if not self.is_empty():
            return self.queue.popleft()
        return "Queue is empty"
    def is_empty(self):
        return len(self.queue)==0

q=Queue()
q.enqueue(1)
q.enqueue(2)
q.enqueue(3)
```

```
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.dequeue())
print(q.is_empty())
```

```
1
2
3
Queue is empty
True
```

In [ ]:
```
"11.namedtuple"
```

In [ ]:
```
Example 1: Basic Usage of namedtuple
```

In [11]:
```
from collections import namedtuple
```

In [13]:
```
Point=namedtuple('Point','x y')
p=Point(1,2)
print(p.x)
print(p.y)
```

```
1
2
```

In [ ]:
```
Example 2: namedtuple with Default Values
```

In [19]:
```
from collections import namedtuple
Point=namedtuple('Point','x y z',defaults=[0,0])
p1=Point(1,2)
p2=Point(1,2,3)
print(p1)
print(p2)
```

```
Point(x=1, y=2, z=0)
Point(x=1, y=2, z=3)
```

In [ ]:
```
Example 3: Unpacking namedtuple
```

In [ ]:
```
from collections import namedtuple
```

In [31]:
```
Point=namedtuple('Point','x y z')
p=Point(1,2,3)
x,y,z=p
print(p)
print(p.x)
print(p.y)
print(p.z+p.x)
print(x,y,z)
```

```
Point(x=1, y=2, z=3)
1
2
4
1 2 3
```

In [ ]:
```
Example 4: Real-world Example - Employee Record
```

In [33]:
```
from collections import namedtuple
Employee =namedtuple('Employee','name age department')
emp1=Employee('Alice','30','HR')
emp2=Employee('vasu','21','datascience')
print(emp1)
print(emp2)
```

```
Employee(name='Alice', age='30', department='HR')
Employee(name='vasu', age='21', department='datascience')
```

In [ ]:
```
Example 5: Using namedtuple for 3D Coordinates
```

In [37]:
```
from collections import namedtuple
Coordinate=namedtuple('Coordinate','x y z')
point1=Coordinate(3,4,5)
point2=Coordinate(-1,4,-3)
def euclidean_distance(p1,p2):
    return ((p1.x-p2.x)**2+(p1.y-p2.y)**2+(p1.z-p2.z)**2)**0.5
print(euclidean_distance(point1,point2))
```

```
8.94427190999916
```

In [ ]:
```
"12.ChainMap"
```

In [ ]:
```
Example 1: Basic Usage of ChainMap
```

In [45]:
```
from collections import ChainMap

dict1={'a':1,'b':2}
dict2={'b':3,'c':4}
chain =ChainMap(dict1,dict2)
print(chain['a'])
print(chain['b'])#Output: 2 (from dict1, since it appears first)
print(chain['c'])
```

```
1
2
4
```

In [ ]:
```
Example 2: Updating ChainMap
```

In [59]:
```
from collections import ChainMap

dict1={'a':1,'b':2}
dict2={'b':3,'c':4}
chain=ChainMap(dict1,dict2)
chain['a']=10
chain['c']=40
print(dict1)
print(dict2)
```

```
{'a': 10, 'b': 2, 'c': 40}
{'b': 3, 'c': 4}
```

In [61]:
```
dict1={'a':1,'b':2}
dict2={'b':3,'c':4}
chain=ChainMap(dict2,dict1)
chain['a']=10
chain['c']=40
print(dict1)
print(dict2)
```

```
{'a': 1, 'b': 2}
{'b': 3, 'c': 40, 'a': 10}
```

In [ ]:
```
Example 3: Adding a New Dictionary to ChainMap
```

In [ ]:
```
from collections import ChainMap
```

In [65]:
```
dict1={'a':1,'b':5}
dict2={'b':3,'c':4}
dict3={'d':5,'e':8}

chain=ChainMap(dict1,dict2)
chain=chain.new_child(dict3)
print(chain['d'])
print(chain['b']) #(from dict1, since it appears first)
```

```
5
5
```

In [ ]:
```
Example 4: Real-world Example - Configurations Management
```

In [69]:
```
from collections import ChainMap

default_config={'theme':'light','language':'Telugu','show_tips':True}
user_config={'theme':'dark','show_tips':False}
config=ChainMap(user_config,default_config) #here first preference is user_default dict
print(config['theme'])# Output: dark (user preference overrides default)
print(config['language'])
print(config['show_tips'])# Output: False (user preference overrides default)
```

```
dark
Telugu
False
```

In [ ]:
```
Example 5: Merging Dictionaries with ChainMap
```

In [ ]:
```
from collections import ChainMap
```

In [75]:
```
dict1={'x':10,'y':30}
dict2={'y':39,'z':53}
dict3={'z':50,'w':90}
chain=ChainMap(dict1,dict2,dict3)
merged=dict(chain)
print("normal chain_dict:",chain)
print("merged_chian:",merged)
```

```
normal chain_dict: ChainMap({'x': 10, 'y': 30}, {'y': 39, 'z': 53}, {'z': 50, 'w': 90})
merged_chian: {'z': 53, 'w': 90, 'y': 30, 'x': 10}
```

In [ ]:
```
"13.Counter"
```

In [ ]:
```
Example 1: Basic Counter Usage
```

In [77]:
```
from collections import Counter
data=[1,2,4,5,3,2,4,6,3,2,4,5,3]
counter=Counter(data)
print(counter)
```

```
Counter({2: 3, 4: 3, 3: 3, 5: 2, 1: 1, 6: 1})
```

In [ ]:
```
Example 2: Counting Characters in a String
```

In [81]:
```
string="abcbacebaceb"
counter=Counter(string)
print(counter)
```

```
Counter({'b': 4, 'a': 3, 'c': 3, 'e': 2})
```

In [ ]:
```
Example 3: Finding the Most Common Elements
```

In [89]:
```
data=['apple','banana','apple','orenge','banana','apple']
counter=Counter(data)
print(counter.most_common(2))
```

```
[('apple', 3), ('banana', 2)]
```

In [ ]:
```
Example 4: Real-world Example - Word Frequency in Text
```

In [91]:
```
from collections import Counter

text="I am srinivasulu and evryone used to call me vasu and I am data scinece student"
words=text.split()
counter=Counter(words)
print(counter)
```

```
Counter({'I': 2, 'am': 2, 'and': 2, 'srinivasulu': 1, 'evryone': 1, 'used': 1, 'to': 1, 'call': 1, 'me': 1, 'vas
u': 1, 'data': 1, 'scinece': 1, 'student': 1})
```

In [ ]:
```
Example 5: Subtracting Counters
```

In [107…
```
from collections import Counter

inventory=Counter(apples=10,orenges=5,banana=7)
sold=Counter(apples=4,orenges=1,banana=6)
inventory.subtract(sold)
print(inventory)
```

```
Counter({'apples': 6, 'orenges': 4, 'banana': 1})
```

In [ ]:
```
"14 OrderedDict"
```

In [ ]:
```
Example 1: Maintaining Insertion Order
```

In [121…
```
from collections import OrderedDict
od=OrderedDict()
od['one']=1
od['two']=2
od['three']=3
print(od)
```

```
OrderedDict({'one': 1, 'two': 2, 'three': 3})
```

In [ ]:
```
Example 2: Comparing OrderedDicts
```

In [127…
```
from collections import OrderedDict

od1=OrderedDict({'a':1,'b':3,'c':4})
od2=OrderedDict({'a':1,'c':4,'b':3})
print(od1==od2)# Output: False (order matters)
```

```
False
```

In [133…
```
od1=OrderedDict({'a':1,'b':3,'c':4})
od2=OrderedDict({'a':1,'b':3,'c':4,})
print(od1==od2)
```

```
True
```

In [135…
```
od1=OrderedDict({'a':2,'b':3,'c':4})
od2=OrderedDict({'a':1,'b':4,'c':4,})
print(od1==od2) #values matters
```

```
False
```

In [ ]:
```
Example 3: Reordering OrderedDict
```

In [145…
```
od1=OrderedDict({'one':1,'two':2,'three':3})
od1.move_to_end('one')
print(od1)
```

```
OrderedDict({'two': 2, 'three': 3, 'one': 1})
```

In [149…
```
dir(OrderedDict)
```

Out[149…
```
['__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__ior__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__or__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__ror__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'clear',
 'copy',
 'fromkeys',
 'get',
 'items',
 'keys',
 'move_to_end',
 'pop',
 'popitem',
 'setdefault',
 'update',
 'values']
```

In [ ]:
```
Example 4: Real-world Example - LRU Cache
```

In [184…
```
from collections import OrderedDict

class LRUCache:
    def __init__(self,capacity: int):
        self.cache=OrderedDict()
        self.capacity=capacity
    def get(self,key:int)->int:
        if key not in self.cache:
            return -1
        self.cache.move_to_end(key)
        return self.cache[key]
    def put(self,key:int,value:int) -> None:
        if key in self.cache:
            self.cache.move_to_end(key)
        self.cache[key]=value
        if len(self.cache)>self.capacity:
```

```
            self.cache.popitem(last=False)


lru_cache=LRUCache(2)
lru_cache.put(1,1)
lru_cache.put(2,2)
print(lru_cache.get(1))
lru_cache.put(3,3)
print(lru_cache.get(2))#(evicted)
```

```
1
-1
```

In [ ]:
```
Example 5: Sorting an OrderedDict
```

In [188…
```
from collections import OrderedDict

od=OrderedDict({'banana':3,'apple':4,'pear':1,'orenge':2})
sorted_od=OrderedDict(sorted(od.items(),key=lambda t: t[1]))
print(sorted_od)
```

```
OrderedDict({'pear': 1, 'orenge': 2, 'banana': 3, 'apple': 4})
```

In [192…
```
sorted_od=OrderedDict(sorted(od.items()))
sorted_od
```

Out[192…
```
OrderedDict([('apple', 4), ('banana', 3), ('orenge', 2), ('pear', 1)])
```

In [198…
```
sorted_od=OrderedDict(sorted(od.items(),key=lambda t:t[0]))
sorted_od
```

Out[198…
```
OrderedDict([('apple', 4), ('banana', 3), ('orenge', 2), ('pear', 1)])
```

In [ ]:
```
"15 defaultdict"
```

In [ ]:
```
Example 1: defaultdict with List
```

In [200…
```
from collections import defaultdict

d=defaultdict(list)
d['fruits'].append('apple')
d['fruits'].append('banana')
print(d)
```

```
defaultdict(<class 'list'>, {'fruits': ['apple', 'banana']})
```

In [ ]:
```
Example 2: defaultdict with int for Counting
```

In [202…
```
from collections import defaultdict

word_count=defaultdict(int)
text="hello world hello world"
for word in text.split():
    word_count[word]+=1
print(word_count)
```

```
defaultdict(<class 'int'>, {'hello': 2, 'world': 2})
```

In [ ]:
```
Example 3: Grouping Items by Key
```

In [208…
```
from collections import defaultdict

names=[('john','A'),('Jane','B'),('Jack','A'),('Jill','B')]
grade_group=defaultdict(list)
for name,grade in names:
    grade_group[grade].append(name)
print(grade_group)
```

```
defaultdict(<class 'list'>, {'A': ['john', 'Jack'], 'B': ['Jane', 'Jill']})
```

In [ ]:
```
Example 4: Real-world Example - Creating an Adjacency List
```

In [210…
```
from collections import defaultdict
edges=[('A','B'),('A','C'),('B','D'),('C','D')]
adjacency_list=defaultdict(list)
for start,end in edges:
    adjacency_list[start].append(end)
print(adjacency_list)
```

```
defaultdict(<class 'list'>, {'A': ['B', 'C'], 'B': ['D'], 'C': ['D']})
```

In [ ]:
```
Example 5: defaultdict with a Custom Factory Function
```

In [214…
```python
from collections import defaultdict

def default_factory():
    return "Unknown"
city_country=defaultdict(default_factory)
city_country['Paris']='France'
city_country['India']='Germany'
print(city_country)
print(city_country['Paris'])
print(city_country['Londan'])
```

```
defaultdict(<function default_factory at 0x00000267036B9C60>, {'Paris': 'France', 'India': 'Germany'})
France
Unknown
```

In [ ]:
```python
"16.Datetime functions"
```

In [ ]:
```python
Example 1: Getting current date and time
```

In [226…
```python
from datetime import datetime
current_date=datetime.now()
print(current_date)
```

```
2024-09-11 16:15:37.146240
```

In [ ]:
```python
Example 2: Formatting Dates
```

In [222…
```python
from datetime import datetime

current_datetime=datetime.now()
formatted_date=current_datetime.strftime("%Y-%m-%d %H:%M:%S")
print(formatted_date)
```

```
2024-09-11 16:14:33
```

In [ ]:
```python
Example 3: Parsing Strings into Dates
```

In [232…
```python
from datetime import datetime

date_str="2024-08-11 14:30:00"

parsed_date=datetime.strptime(date_str,"%Y-%m-%d %H:%M:%S")
print(parsed_date)
```

```
2024-08-11 14:30:00
```

In [ ]:
```python
Example 4: Calculating Time Differences
```

In [238…
```python
from datetime import datetime,timedelta
now=datetime.now()
future_date=now+timedelta(days=5)
difference=future_date-now
print(f"Days untill future date: {difference.days}")
```

```
Days untill future date: 5
```

In [ ]:
```python
"17.Iterator"
```

In [ ]:
```python
Example 1: Creating Custom Iterator
```

In [246…
```python
class Counter:
    def __init__(self,start,end):
        self.current=start
        self.end=end
    def __iter__(self):
        return self
    def __next__(self):
        if self.current>self.end:
            raise StopIteration
        else:
            self.current+=1
            return self.current-1

counter=Counter(1,5)

for num in counter:
    print(num)
```

```
1
2
3
4
5
```

In [ ]:
```
Example 2: Implementing a Reverse Iterator
```

In [256…
```
class Reverse:
    def __init__(self,data):
        self.data=data
        self.index=len(data)
    def __iter__(self):
        return self
    def __next__(self):
        if self.index==0:
            raise StopIteration
        self.index-=1
        return self.data[self.index]
rev=Reverse('Python')
for char in rev:
    print(char,end=" ")
```

```
n o h t y P
```

In [ ]:
```
Example 3: Iterating Over a Dictionary
```

In [262…
```
my_dict={'a':1,'b':2,'c':3}
for key in my_dict:
    print(key,my_dict[key])
```

```
a 1
b 2
c 3
```

In [ ]:
```
Example 4: Real-world Example - File Line Iterator
```

In [280…
```
class FileLineIterator:
    def __init__(self,file_name):
        self.file=open(file_name,'r')
    def __iter__(self):
        return self
    def __next__(self):
        line=self.file.readline()
        if not line:
            self.file.close()
            raise StopIteration
        return line.strip()

file_iter=FileLineIterator('example.txt')
for line in file_iter:
    print(line)
```

```
hello world,namstheyhello 'world',namsthey
```

In [ ]:
```
Example 5: Using Iterator to Iterate Over Two Lists Simultaneously
```

In [282…
```
list1=[1,2,3]
list2=['a','b','c']

iter1=iter(list1)
iter2=iter(list2)

for i in range(len(list1)):
    print(next(iter1),next(iter2))
```

```
1 a
2 b
3 c
```

In [ ]:
```
"18.Multilevel Inheritence"
```

In [ ]:
```
Example 1 : Basic Multilevel Inheritence
```

In [288…
```
class GrandParent:
    def __init__(self):
        print("GrandParent Constructer")

class Parent(GrandParent):
    def __init__(self):
        super().__init__()
        print("Parent Constructer")

class Child(Parent):
    def __init__(self):
       super().__init__()
       print("Child Constructer")
```

```
child=Child()
```

```
GrandParent Constructer
Parent Constructer
Child Constructer
```

In [ ]:
```
Example 2: Overriding Methods in Multilevel Inheritence
```

In [290…
```
class A:
    def show(self):
        print('A')
class B:
    def show(self):
        print('B')

class C:
    def show(self):
        print('C')
obj=C()
obj.show()
```

```
C
```

In [ ]:
```
Example 3: Calling Parent Class Methods in Multilevel Inheritance
```

In [296…
```
class Animal:
    def sound(self):
        print("Animal makes sound")

class Mammal(Animal):
    def sound(self):
        super().sound()
        print("Mammal makes sound")

class Dog(Mammal):
    def sound(self):
        super().sound()
        print("Dog makes sound")

dog=Dog()
dog.sound()
```

```
Animal makes sound
Mammal makes sound
Dog makes sound
```

In [ ]:
```
Example 4: Real-world Example -Company Heirarchy
```

In [5]:
```
class Company:
    def __init__(self,name):
        self.name=name
class Department(Company):
    def __init__(self,name,department_name):
        super().__init__(name)
        self.department_name=department_name
class Employee(Department):
    def __init__(self,name,department_name,employee_name):
        super().__init__(name,department_name)
        self.employee_name=employee_name
    def get_details(self):
        return f"Employee: {self.employee_name},Department_name: {self.department_name},Company: {self.name}"
emp=Employee("data science","IT","vasu")
print(emp.get_details())
```

```
Employee: vasu,Department_name: IT,Company: data science
```

In [ ]:
```
Example 5 : Multilevel inheritence with init Method
```

In [7]:
```
class Base:
    def __init__(self,name):
        self.name=name
        print("Base class constructer")

class Intermediate(Base):
    def __init__(self,name,age):
        super().__init__(name)
        self.age =age
        print("Intermadite claa constrcuter")

class Derived(Intermediate):
    def __init__(self,name,age,role):
        super().__init__(name,age)
        self.role=role
        print("Derived class constructor")
```

```
derived=Derived("vasu",21,"data scientist")
```

```
Base class constructer
Intermadite claa constrcuter
Derived class constructor
```

In [ ]:
```
8.Ordered Dictionary
```

In [ ]:
```
Example 1: Preserving Insertion OrderedDict
```

In [9]:
```
from collections import OrderedDict

#creating an OrederedDict
order_dict=OrderedDict()

#Adding items
order_dict['apple']=2
order_dict['banana']=3
order_dict['orenge']=2
#Iterating over the OrederedDict
for key,value in order_dict.items():
    print(key,value)
```

```
apple 2
banana 3
orenge 2
```

In [ ]:
```
Example 2: Comparing Two Dictionaries
```

In [11]:
```
from collections import OrderedDict

dict1=OrderedDict([('apple',2),('banana',4),('orenge',1)])
dict2=OrderedDict([('banana',2),('apple',3),('orenge',4)])
print(dict1==dict2)# False, order matters
```

```
False
```

In [ ]:
```
Example 3: Move Elements to the End or Start
```

In [19]:
```
from collections import OrderedDict

order_dict=OrderedDict([('apple',2),('banana',3),('orenge',4)])

order_dict.move_to_end('apple')

print(order_dict)
```

```
OrderedDict({'banana': 3, 'orenge': 4, 'apple': 2})
```

In [23]:
```
dir(OrderedDict)
```

Out[23]:
```
['__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__ior__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__or__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__ror__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'clear',
 'copy',
 'fromkeys',
 'get',
 'items',
 'keys',
 'move_to_end',
 'pop',
 'popitem',
 'setdefault',
 'update',
 'values']
```

In [ ]:
```
Example 4: Implementing a Simple LRU cache
```

In [29]:
```
from collections import OrderedDict
class LRUCache:
    def __init__(self,capacity):
        self.cache=OrderedDict()
        self.capacity=capacity
    def get(self,key):
        if key in self.cache:
            self.cache.move_to_end(key)
            return self.cache[key]
        return -1
    def put(self,key,value):
        if key in self.cache:
            self.cache.move_to_end(key)
        self.cache[key]=value
        if len(self.cache)>self.capacity:
            self.cache.popitem(last=False)

lru=LRUCache(2)
lru.put(1,1)
lru.put(2,2)
print(lru.get(1))
lru.put(3,3)
print(lru.get(2))
```

```
1
-1
```

In [ ]:
```
Example 5: Maintaining a Sorted Dictionary
```

In [31]:
```
unsorted_dict={'banana':3,'apple':2,'orange':1}
sorted_dict=OrderedDict(sorted(unsorted_dict.items()))
print(sorted_dict)
```

```
OrderedDict({'apple': 2, 'banana': 3, 'orange': 1})
```

In [ ]:
```
9.Default Dictionary
```

In [ ]:
```
Example 1: Counting Frequrncies
```

In [33]:
```
from collections import defaultdict

#creating a defaultdict for counting
freq=defaultdict(int)
for char in "banana":
    freq[char]+=1
print(freq)
```

```
defaultdict(<class 'int'>, {'b': 1, 'a': 3, 'n': 2})
```

In [ ]:
```
Example 2: Grouping Items
```

In [ ]:
```
from collections import defaultdict
```

In [35]:
```
#creating a defaultdict for grouping

grouped=defaultdict(list)
#grouping items
items=[('apple',2),('banana',3),('apple',5)]
for key,value in items:
    grouped[key].append(value)

print(grouped)
```

```
defaultdict(<class 'list'>, {'apple': [2, 5], 'banana': [3]})
```

In [ ]:
```
Example 3: Building a Multi-Value Dictionary
```

In [39]:
```
from collections import defaultdict
#creating a defaultdict
multi_dict=defaultdict(set)
#Adding multiple values to key
multi_dict['fruits'].add('apple')
multi_dict['fruits'].add('banana')
print(multi_dict)
```

```
defaultdict(<class 'set'>, {'fruits': {'apple', 'banana'}})
```

In [ ]:
```
Example 4: Missing Key Handling
```

In [41]:
```
from collections import defaultdict

zero_dict=defaultdict(lambda:0)
print(zero_dict['missing'])
```

```
0
```

In [ ]:
```
Example 5: Accumulating Results
```

In [43]:
```
from collections import defaultdict

accum=defaultdict(int)
transactions=[('apple',10),('banana',5),('apple',-3)]
for fruit,amount in transactions:
    accum[fruit]+=amount

print(accum)
```

```
defaultdict(<class 'int'>, {'apple': 7, 'banana': 5})
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```