1. Converting an Integer into Decimals

In [ ]:
```
import decimal
integer = 10
print(decimal.Decimal(integer))
print(type(decimal.Decimal(integer)))
```

2. Converting an String of Integers into Decimals

In [ ]:
```
import decimal
string = '12345'
print(decimal.Decimal(string))
print(type(decimal.Decimal(string)))
```

3. Reversing a String using an Extended Slicing Technique

In [ ]:
```
string = "Python Programming"
print(string[::-1])
```

4. Counting VOWELS in a Given Word

In [ ]:
```
vowel = ['a', 'e', 'i', 'o', 'u']
word = "programming"
count = 0
for character in word:
    if character in vowel:
        count += 1
print(count)
```

5. Counting CONSONANTS in a Given Word

In [ ]:
```
vowel = ['a', 'e', 'o', 'i', 'u']
word = "programming"
count = 0
for character in word:
    if character not in vowel:
        count += 1
print(count)
```

6. Counting the number of occurances of a character in a String

In [ ]:
```
word = "programming"
character = 'g'
count = 0
for i in word:
    if i == character:
        count+=1
print(count)
```

7. Writing FIBONACCI Series

In [14]:
```
fib = [0,1]
# Range starts from 0 by default
n=5
for i in range(n):
    fib.append(fib[-1] + fib[-2])

#Converting the list of integers to string
print(', '.join(str(e) for e in fib))
```

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

8. Finding the Maximum Number in a list

In [ ]:
```
numberList = [12, 3, 55, 23, 6, 78, 33, 4]
max = numberList[0]
for num in numberList:
    if max < num:
        max = num
print(max)
```

9. Finding the Minimum Number in a List

In [ ]:
```
numberList = [12, 3, 55, 23, 6, 78, 33, 4]
min = numberList[0]
for num in numberList:
    if min > num:
        min = num
print(min)
```

10. Finding the middle Element in a list

In [ ]:
```
numList = [12, 3, 55, 23, 6, 78, 33, 5]
midElement = int((len(numList)/2))
print(numList[midElement])
```

11. Converting a list into a string

In [ ]:
```
list = ["P", "Y", "T", "H", "O", "N"]
string = "".join(list)

print(string)
print(type(string))
```

12. Adding Two List Elements Together

In [ ]:
```
lst1 = [1,2,3]
lst2 = [4,5,6]

res_lst = []
for i in range(0, len(lst1)):
    res_lst.append(lst1[i] + lst2[i])
print(res_lst)
```

13. Comparing Two Strings for ANAGRAMS

In [ ]:
```
str1 = "Listen"
str2 = "Silent"

str1 = str1.replace(" ","").upper()
str2 = str2.replace(" ","").upper()

if sorted(str1) == sorted(str2):
    print("True")
else:
    print("False")
```

14. Checking for PALINDROME Using Extended Slicing Technique

In [ ]:
```
str1 ="Kayak".lower()

if str1 == str1[::-1]:
    print("True")
else:
    print("False")
```

15. Counting the white spaces in a string

In [ ]:
```
string = "P r ogram in g"
print(string.count(" "))
```

16. Counting Digits, Letters, and Spaces in a String

In [ ]:
```
# importing Regular Expressions Library
import re

name = 'Python is 1'

digitCount = re.sub("[^0-9]", "",name)
letterCount = re.sub("[^a-zA-Z]", "",name)
spaceCount = re.findall("[ \s]", name)

print(len(digitCount))
print(len(letterCount))
print(len(spaceCount))
```

17. Counting Special Characters in a string

In [ ]:
```
def count_sp_char(string):
    sp_char = "!@#$%^&*()<>?/\[]{};:~`"
    count = 0
    for i in string:
        if i in sp_char:
            count+=1
    return count

text = 'Hello! How are you? #specialchars! 123'
result = count_sp_char(text)
print(count)
```

18. Removing All Whitespace in a String

In [ ]:
```
import re

string = "C O D E"
spaces = re.compile(r'\s+')
result = re.sub(spaces, "", string)
print(result)
```

In [ ]:
```
string = "C O D E"
string2 = "".join(char for char in string if char != " ")
print(string2)
```

In [ ]:
```
string = 'C O D E'
string2 = string.replace(" ","")
print(string2)
```

19. Building a Pyramid in Python

In [1]:
```
    def pyramid(n):
        for i in range(n):
            for j in range(i,n):
                print(" ",end="")
            for j in range(i+1):
                print("*",end="")
            for j in range(i):
                print("*",end="")
            print("")
    pyramid(5)
############################################################

num=int(input("enetr odd number")) #5
cnt=num//2 #2
scnt=1
for i in range(cnt+1):
    print(cnt*" "+"*"*scnt)
    cnt=cnt-1
    scnt=scnt+2
scnt=num-2
cnt=1
for i in range(num//2):
    print(cnt*" "+"*"*scnt)
    scnt=scnt-2
    cnt=cnt+1
```

```
     *
    ***
   *****
  *******
 *********
enetr odd number9
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

20. Randomizing the Items of a List in Python

In [ ]:
```
from random import shuffle

lst = ['Python', 'is', 'Easy']
shuffle(lst)
print(lst)
```

21. Create a generator to produce first n prime numbers

In [1]:
```
def isprime(num):
    for i in range(2, num):
        if num%i == 0:
            return False
    return True

def prime_generator(n):
    num = 2
    while n:
        if isprime(num):
            '''
            #yield keyword used to return a value and then the code
            is resumed inside the function unlike the return keyword end
            the code when it is called
            #yield keyword will turn any expression that is given with it
            into a generator object and return it to the caller
            '''

            yield num
            n-=1
        num+=1
x = int(input("Enter no. of prime numbers required"))
it = prime_generator(x)
for e in it:
    print(e, end=" ")
```

```
Enter no. of prime numbers required 10

2 3 5 7 11 13 17 19 23 29
```

In [1]:
```
#program to check if a given number is prime or not
def prime_no(n):
    Flag = False
    if n < 2:
        return Flag
    else:
        for i in range(2,n):
            if n%i==0:
                return Flag
            else:
                Flag = True
                return Flag

num = int(input("Enter a number:"))
prime_no(num)
```

```
Enter a number:37
```

Out[1]:
```
True
```

In [1]:
```
def prime_no(n):
    Flag = False
    if n <2 :
        return Flag
    else:
        for i in range(2,n):
            if n%i == 0:
                return Flag
        Flag = True
        return Flag
num = int(input("Enter a number:"))
prime_no(num)
```

```
Enter a number:25
```

Out[1]:
```
False
```

22. Implementing variable length arguments in python

In [4]:
```
def average(*t): # *t for tuple of variable length
    avg = sum(t)/len(t)
    return avg
result1 = average(32,5,65,22,87,34,2,57)
result2 = average(5,10,15,20,25,30,35,40,45,50)

print("Average is:",result1)
print("Average is:",result2)
```

```
Average is: 38.0
Average is: 27.5
```

In [5]:
```
def average_with_kwargs(**kwargs):  # **kwargs for a dictionary of variable lengt
    values = list(kwargs.values())
    avg = sum(values) / len(values)
    return avg

result1 = average_with_kwargs(num1=32, num2=5, num3=65, num4=22, num5=87, num6=34
resutl2 = average_with_kwargs(value1=5, value2=10, value3=15, value4=20, value5=2

print(f'Average is: {result1:.2f}')
print(f'Average is: {result2:.2f}')
```

```
Average is: 38.00
Average is: 27.50
```

23. Creating instance member variables in python

In [ ]:
```
class Test:
    def __init__(self):
        # 1st instance variable inside init function
        self.a = 5
    def f1(self):
        # 2nd instance variable inside funcition f1
        self.b = 10
t1 = Test()
t2 = Test()
t1.f1()
# 3rd instance variable
t1.c= 15
print(t1.__dict__)
print(t2.__dict__)
```

24. Addition using Lambda functions

In [ ]:
```
# lambda function to add two variables
f = lambda a,b : a+b
r = f(3,7)
print(r)
#OR
print((lambda a,b:a+b)(3,7))
```

25. Finding factorial using lambda function

In [14]:
```
f = lambda n : 1 if n==0 else n*f(n-1)
f(5)
```

Out[14]:
```
120
```

In [13]:
```
num = int(input("Enter a number: "))
# 7
factorial = 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):
       factorial = factorial*i
   print(f"The factorial of {num} is",factorial)
   # 5040
```

```
Enter a number: 3
The factorial of 3 is 6
```

26. List Compression (to create a list in single line)

In [ ]:
```
# list of even numbers
l1 = [2*e for e in range(1, 10)]
print(l1)

# to create a list of even numbers from a given list
list = [23,56,65,22,62,32,65,76,33,99]
l2 = [e for e in list if e%2==0]
print(l2)
'''
List comprehension is a concise and expressive way to create
lists in Python. It provides a more compact syntax for generating
lists compared to using traditional loops.

'''
```

27. What is the use of split and join function of str?

In [3]:
```
s = "What is right in your mind is right in your world"
#split - to convert string into list of string
s1=s.split(" ")
print(s1)
s1=s1[::-1]
print(s1)
#join - to convert the list again into string
print(" ".join(s1))
```

```
['What', 'is', 'right', 'in', 'your', 'mind', 'is', 'right', 'in', 'your', 'wor
ld']
['world', 'your', 'in', 'right', 'is', 'mind', 'your', 'in', 'right', 'is', 'Wh
at']
world your in right is mind your in right is What
```

28. Global and local variable

In [4]:
```
x = 5 # global variable
def f1():
    global x
    x = 15 # global variable updated
    y = 10 # local variable
    print("x=%d y=%d"%(x,y))
f1()
print(x)
```

```
x=15 y=10
15
```

29. Globals function

In [5]:
```
# globals function returns dictionary
#You can use the globals() function to access or modify global variables
#within a function or code block.
x = 5 #global variable
def fun():
    x = 10 #local variable
    d = globals() # d is dictionary
    # d['x'] = 15 # x is the key in dictionary d which assigns value to global va
    print( "local x=%d  global x=%d"%(x,d['x']))
fun()
#
```

```
local x=10  global x=5
```

30. Type conversion basics

In [ ]:
```
x = int('123')
a = float('123.42')
b = complex('3+4j')
c = str(12)
d = bool('True')
e = bin(25) #binary
f = oct(25)
g = hex(25)
h = ord('A') #char to unicode
i = chr(98) #unicode to character
print(x,a,b,c,d,e,f,g,h,i,sep="\n")
```

31. Type conversion

In [1]:
```
x = 5
print(type(x))
s1 = '123'
print(type(s1))
print(str(x) + s1)
print(x + int(s1))
```

```
<class 'int'>
<class 'str'>
5123
128
```

32. What is python decorator ?

In [3]:
```
#function decorator

def welcome(fx):
    def mfx(*t,**d):
        print("Before hello function")
        fx(*t,**d) #*args to take arguments as tuple, **kwargs to take arguments
        print("Thanks for using the function")
    return mfx


#decorator function without arguments
@welcome
def hello():
    print("Hello !")

#decorator function with arguments
@welcome
def add(a,b):
    print(a+b)


hello()
add(1,3)
```

```
Before hello function
Hello !
Thanks for using the function
Before hello function
4
Thanks for using the function
```

In [7]:
```
#class Decorator

class Calculator:
    def __init__(self,func):
        self.function = func

    def __call__(self,*t,**d):
            result=self.function(*t,**d)
            return result**2

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

# add = Calculator(add)
add(10,20)  #add.__call__(a,b) since function type is callable
```

Out[7]:
```
900
```

In [2]:
```python
# Decorators are used to modify the behavior of functions or methods without
# changing their code.
# Decorator function is a function that takes another functions as there
# argument.

def decor_result(result_function):
    def distinction(marks):
        results = []  # List to store results for each element
        for m in marks:
            if m >= 75:
                print("DISTINCTION")
            results.append(result_function([m]))  # Pass a list with the current

        return results  # Return the collected results

    return distinction

@decor_result
def result(marks):
    for m in marks:
        if m >= 33:
            pass
        else:
            print("FAIL")
            return "FAIL"  # Return FAIL if any element fails

    else:
        print("PASS")
        return "PASS"  # Return PASS if all elements pass

# Get the results for each element
results = result([45, 78, 80, 34, 66, 90])

print("Results:", results)
```

```
DISTINCTION
DISTINCTION
DISTINCTION
PASS
```

In [8]:
```python

```

```
PASS
DISTINCTION
PASS
DISTINCTION
PASS
PASS
PASS
DISTINCTION
PASS
Results: ['PASS', 'PASS', 'PASS', 'PASS', 'PASS', 'PASS']
```

33. What are iterators in python?

An iterable is any Python object that can be looped over or iterated. It can be a sequence (e.g., list, tuple, string), a collection (e.g., set, dictionary), or any object that supports iteration.

An iterator used to access the objects of the iterables one by one from 1st element to last element. An iterator is an object that represents a stream of data. It provides two essential methods: **iter**() and **next**()

In [5]:
```
l1 = [23,65,22,76,34,98,43]
it = iter(l1)
while True:
    try:
        print(next(it))
    except StopIteration:
        break
```

```
23
65
22
76
34
98
43
```

33. What are generators in python? Create a generator for first n natural even numbers.

Generators in Python are a type of iterable, like lists or tuples, but they allow you to iterate over their elements lazily, one at a time, on-the-fly. This means that generators generate values as you need them, rather than storing all values in memory at once. This can be highly memory-efficient, especially when dealing with large datasets or when generating an infinite sequence.

In [8]:
```
def evenNum(n):
    i = 1
    while n:
        yield 2*i
        i+=1
        n-=1
it = evenNum(10)
even_list=[]
while True:
    try:
        even_list.append(next(it))
    except StopIteration:
        break
print(even_list)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

34. Is Function overloading allowed in python ?

In Python, function overloading as seen in languages like C++ or Java is not directly supported. However, Python provides several ways to achieve similar behavior through default arguments, variable-length arguments, and more advanced techniques like using functools.singledispatch.

Here are a few methods to achieve function overloading in Python:

1. Using Default Arguments

You can use default arguments to provide different behavior based on the number of arguments passed.

In [4]:
```
def greet(name, greeting="Hello"):
    return f"{greeting}, {name}!"
print(greet("Ashutosh"))
print(greet("Bob","Hi"))
```

```
Hello, Ashutosh!
Hi, Bob!
```

2. Using Variable-Length Arguments

You can use *args and **kwargs to accept a variable number of arguments.

In [5]:
```
def add(*t):
    return sum(t)
print(add(1,3))
print(add(1,3,5,6))
```

```
4
15
```

3. Using functools.singledispatch

The functools.singledispatch decorator allows you to create a single-dispatch generic function, which can have different implementations based on the type of the first argument.

In [14]:
```
from functools import singledispatch

@singledispatch
def process(value):
    raise NotImplementedError("Unsupported type")

@process.register(int)
def _(value):
    return f"Processing an integer: {value}"

@process.register(str)
def _(value):
    return f"Processing a string: {value}"

print(process(10))
print(process("hello"))
```

```
Processing an integer: 10
Processing a string: hello
```

4. Using Class Methods

You can also achieve overloading behavior using class methods.

In [9]:
```
class Math:
    @staticmethod
    def multiply(a,b):
        return a*b
    @staticmethod
    def multiply(a,b,c):
        return a*b*c

#python does not support method overloading, so the last defined method will be t
#To achieve overloading like behavior, you can use different method names or vari

class Math:
    @staticmethod
    def multiply(*t):
        result = 1
        for num in t:
            result *= num
        return result

print(Math.multiply(2,3))
print(Math.multiply(2,3,4,5))
```

```
6
120
```

35. What are positional arguments in python?

There two types of arguments in python, positional arguments and keyword arguments. Keyword arguments are assigned with the keywords that are defined with the function. Keyword arguments should always be declared after the positional arguments.

In [1]:
```
def f1(a, b):
    print("a=",a,", b=",b)
f1(3,5) #positional argument
#f1(b=30,5) #syntax error(compile time error)
f1(30,b=4)
# f1(2,a=10) #type error : multiple values of a (run time error)
f1(a=23,b=34) # keyword argument
```

```
a= 3 , b= 5
a= 30 , b= 4
a= 23 , b= 34
```

36. Difference between sorted and sort function in python.

Sorted is a predifined function all the iterables in python, which returns a new list in sorted form.

In [51]:
```
t1 = (34, 64, 32, 78, 88, 83, 95, 64)
s_t1 = sorted(t1)
print(s_t1)
type(s_t1)
```

```
[32, 34, 64, 64, 78, 83, 88, 95]
```

In [40]:
```
l1 = [34, 64, 32, 78, 88, 83, 95, 64]
l1.sort()
print(type(l1))   #nonetype
print(l1)  #none
```

```
<class 'list'>
[32, 34, 64, 64, 78, 83, 88, 95]
```

In [56]:
```
l1.append(45)
print(l1)
# to add element to list in sorted manner
from sortedcontainers import SortedList
l3=SortedList(l1)
l3.add(45)
print(l3)
```

```
[32, 34, 64, 64, 78, 83, 88, 95, 45, 45, 45, 45, 45, 45]
SortedList([32, 34, 45, 45, 45, 45, 45, 45, 45, 64, 64, 78, 83, 88, 95])
```

37. How to create static member variables in class?

Static variables are shared among all instances of a class and are typically used to store class-level data.

Class Method: Use @classmethod when you need to access or modify the class state or call the method on the class itself. Static Method: Use @staticmethod for utility functions that don't need to access or modify class or instance state but logically belong to the class.

Static methods are methods that are bound to a class rather than an instance and do not have access to instance-specific data. To declare a static method, use the @staticmethod decorator before defining the method.

Use Case: Static methods are typically used for utility functions that perform a task in isolation and do not need to access or modify class or instance data.

Static Variables: a is defined as a static variable directly within the class body. b, c, d, e, and f are also static variables, but they are defined within methods. g is defined as a static variable outside the class body.

Instance Variables: x is an instance variable, initialized within the **init** method.

Local Variables: y is a local variable, initialized within the **init** method. It is only accessible within this method.

In [57]:
```
class myclass:
    a = 5  #static variable
    def __init__(self):
        self.x = 10 #instance variable - can be used anywhere in class using self
        y = 4 #local variable
        myclass.b = 34 #static variable
    def f1(self):
        myclass.c = 65 #static variable
    @staticmethod
    def f2(self):
        myclass.d = 66 #static variable
    @classmethod
    def f3(cls):
        cls.e =15 #static variable
        myclass.f = 53 #static variable
myclass.g = 11 #static variable
```

38.How to use else with loop in python?

Else can be used with if and while loops in python.

After the break statement in the loop the else statement is not executed.

Else statement is only executed when the condition of the loop is false.

In [79]:
```
i = 1
while i<=10:
    print(i,end=" " )
    if i==5:
        break
    i+=1
else:
    print("You are in else ")
```

```
1 2 3 4 5
```

In [80]:
```
i = 1
while i<=10:
    print(i,end=" " )
    if i==12:
        break
    i+=1
else:
    print("\n You are in else")
```

```
1 2 3 4 5 6 7 8 9 10
 You are in else
```

In [84]:
```
for i in range(10):
    print(i,end=" ")
    if i==12:
        break
else:
    print("You are in else")
```

```
0 1 2 3 4 5 6 7 8 9 You are in else
```

39. What is name mingling in python?

When a variable is initialized with double underscore, its name is automatically changed by python to _class__variableName. This phenomenon is called Name Mingling in python.

Name mangling is a technique used in programming to generate unique names for entities like functions and variables to avoid name collisions. This is especially important in languages with features like function overloading and namespaces. The main purposes of name mangling are:

1. **Avoiding Name Collisions**: Ensures unique names for entities with the same name in different scopes or modules.
2. **Supporting Function Overloading**: Encodes function names with parameter types to differentiate overloaded functions.
3. **Encapsulating Namespaces and Classes**: Incorporates namespace or class names into entity names to maintain uniqueness within different contexts.
4. **Linker Compatibility**: Helps the linker correctly match references to their definitions across separate compilation units.
5. **Debugging and Maintenance**: Makes relationships between code parts clearer, aiding in debugging and maintenance.

For example, in C++, two functions `Foo::display(int)` and `Bar::display(int)` would be mangled to unique names like `_ZN3Foo7displayEi` and `_ZN3Bar7displayEi`, incorporating the namespace and parameter type information.

In [4]:
```
class world:
    x = 10
    __india= 20
print(world.x)
print(world._world__india)
```

```
10
20
```

40. What is difference between class object and instance object ?

Class object is called only once, instance object can be called any no. of times.

Class object declared with class.object

Instance object is declared with t1 = class(positional arguments) and where the first argument is instance object(t1) implicitly. Also init method is called implicitly after declaring instance object.

In [1]:
```
class test:
    x = 20
    def __init__(self, a, b):
        self.a=a   #instance member argument
        self.b=b   #instance member argument
    def show(self):
        print(self.a,self.b)

print(test.x) #class object
t1 = test(4,5) #instance object
t1.show()

# Displaying instance variables directly
print(t1.a)
print(t1.b)
```

```
20
4 5
4
5
```

41. What is init method in python?

init method is just like constructor in c++ and java. But incase of c++ and java if we don't create a constructor the program creates one implicitly. Incase of python we have create **init** method explicitly.

In [15]:
```
class test:
    def __init__(self):
        self.a = 10
        self.b = 20
t1 = test() #no need to provide the arguments
print(t1.a, t1.b)
```

```
10 20
```

42. What are default arguments in functions?

The arguments that are declared and assigned a value while creating a function are called default arguments.

It is not allowed to have non-default arguments after default arguments.

In [8]:
```
def add(a, b, c=5):  # c is default argument
    return a+b+c

s = add(2,3)
print("The addition is: ",s)
```

```
The addition is:  10
```

43. Extract int type values from a list of heterogeneous element?

In [12]:
```
l1 = ['abc',34.56,32,3+4j,'b',55,65.7,'90',180]
print(l1)
l2 =[]
for e in l1:
    if type(e)==int:
        l2.append(e)
print(l2)
```

```
['abc', 34.56, 32, (3+4j), 'b', 55, 65.7, '90', 180]
[32, 55, 180]
```

44. Does python support multiple inheritance?

When a class inherits attributes from more than one class it is called multiple inheritance.

Python supports multiple inheritance.

In [14]:
```
class A1:
    pass
class A2:
    pass
class B(A1, A2):
    pass
```

45. What is monkey patching ?

Monkey patching is replace a function object with a new function object, so that the function is now pointing to new function object. Mostly used when you want to replace a function for testing purpose.

In [5]:
```
class Test:
    def __init__(self,x):
        self.a = x
    def get_data(self):
        print("send code to fetch data from database")
    def f1(self):
        self.get_data()
    def f2(self):
        self.get_data()
t1=Test(4)
print("Before Monkey patching\n ")
t1.f1()
t1.f2()
def get_new_data(self):
    print("Some to code fetch data from test data")
Test.get_data = get_new_data
print("\nAfter Monkey Patching\n")
t1.f1()
t1.f2()
```

```
Before Monkey patching

send code to fetch data from database
send code to fetch data from database

After Monkey Patching

Some to code fetch data from test data
Some to code fetch data from test data
```

46. Accept a number user check whether it is prime or not

In [1]:
```
num=int(input("enetr number"))
for i in range(2,num):
    if num%i==0:
        print("number is not prime")
        break
else:
    print("number is prime")
```

```
enetr number47
number is prime
```

47. Write a program to print the given number is odd or even.

In [3]:
```
num = int(input("Enter a number: "))
if num%2==0:
    print(f"Entered number {num} is even")
else:
    print(f"Entered number {num} is odd")
```

```
Enter a number: 34212
Entered number 34212 is even
```

48. Write a program to find the given number is positive or negative.

In [9]:
```
num = float(input("Enter a number: "))
if num > 0:
    print("Number is positive")
elif num==0:
    print("Number is zero")
else:
    print("Number is negative")
```

```
Enter a number: 4
Number is positive
```

49. Write a program to find the sum of two numbers.

In [12]:
```
num1 = int(input("Enter 1st number: "))
num2 = int(input("Enter 2nd number: "))
print("num1 + num2 =",num1+num2)
```

```
Enter 1st number: 3333
Enter 2nd number: 666
num1 + num2 = 3999
```

50. Write a program to find GCD of two numbers.

In [18]:
```
num1 = int(input("Enter 1st number: "))
num2 = int(input("Enter 2nd number: "))

def gcd(a,b):
    if a==0 or a==b:
        return b
    elif b==0:
        return a
    elif a>b:
        return gcd(a-b,b)
    return gcd(a,b-a)

gcd(num1,num2)
```

```
Enter 1st number: 235
Enter 2nd number: 875
```

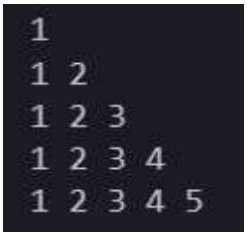
Out[18]: 5

51. Write a program to print the following pattern.

In [7]:
```
s = int(input("Enter a number"))
for i in range(1,s+1):
    print("*  "*i)
    i+=1
```

```
Enter a number5
*
*  *
*  *  *
*  *  *  *
*  *  *  *  *
```

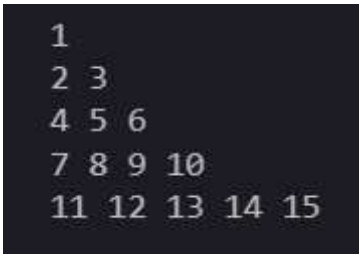
52.Write a program to print the following pattern.



In [32]:
```
s = int(input("Enter a number"))
for i in range(1,s+1):
    for j in range(1,i+1):
        print(j,end=" ")
    print()
    i+=1
```

```
Enter a number5
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

53. Write a program to print the following pattern.



In [8]:
```
def tri(n):
    num=1
    for i in range(0,n):
        for j in range(0,i+1):
            print(num, end=" ")
            num+=1
        print()

tri(5)


#### or ####

s = int(input("Enter a number: "))
p=1
for i in range(1,s+1):
    for j in range(1,i+1):
        print(p,end=" ")
        p+=1
    print()
    i+=1
```

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
Enter a number: 5
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

54. Write a program to print the following pattern.

In [9]:
```
def apha(n):
    p=65
    for i in range(n):
        for j in range(i+1):
            ch = chr(p)
            print(ch, end=" ")
        p+=1
        print()
apha(5)

### or ###

s = int(input("Enter a number: "))
p=65
for i in range(1,s+1):
    for j in range(1,i+1):
        print(chr(p),end=" ")
    p+=1
    print()
    i+=1
```

```
A
B B
C C C
D D D D
E E E E E
Enter a number: 5
A
B B
C C C
D D D D
E E E E E
```

In [13]:
```
def apha(n):
    p=65
    for i in range(n):
        for j in range(i,n):
            ch = chr(p)
            print(ch, end=" ")
        p+=1
        print()
apha(5)
```

```
A A A A A
B B B B
C C C
D D
E
```

In [23]:
```
def apha(n):
    p=65
    for i in range(n):
        for j in range(i,n):
            print(" ",end=" ")
        for j in range(i+1):
            print(chr(p), end=" ")
        for j in range(i):
            print(chr(p), end=" ")
        p+=1
        print()
apha(5)
```

```
          A
        B B B
      C C C C C
    D D D D D D D
  E E E E E E E E E
```

55. Write a program to print the following pattern.

In [3]:
```
def apha(n):
    p=65
    for i in range(n):
        for j in range(i+1):
            ch = chr(p)
            p+=1
            print(ch, end=" ")
        print()
apha(5)

### or ####

n = int(input("Enter a number: "))
p = 65
for i in range(n):
    for j in range(i+1):
        print(chr(p),end=" ")
        p+=1
    print()
```

```
A
B C
D E F
G H I J
K L M N O
```

In [8]:
```
n=5
p=65
for i in range(n):
    for j in range(i+1):
        print(" ",end=" ")
    for j in range(i,n):
        print(chr(p),end=" ")
        p+=1
    print()
```

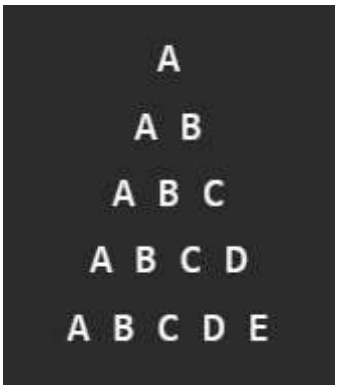
```
  A B C D E
    F G H I
      J K L
        M N
          O
```

In [12]:
```
n=5
p=65
for i in range(n):
    for j in range(i,n):
        print(" ",end=" ")
    for j in range(i+1):
        print(chr(p),end=" ")
        p+=1
    print()
```

```
          A
        B C
      D E F
    G H I J
  K L M N O
```

In [ ]:
```
def apha(n):
    p=65
    for i in range(n):
        for j in range(i,n):
            print(" ",end=" ")
        for j in range(i+1):
            print(chr(p), end=" ")
        for j in range(i):
            print(chr(p), end=" ")
        p+=1
        print()
apha(5)
```

56. Write a program to print the following pattern.



In [2]:
```
def apha(n):
    num = 65
    for i in range(n):
        for j in range(i,n):
            print(" ",end="")
        for j in range(i+1):
            print(chr(num),end =" ")
            num+=1
        num = 65
        print()

apha(5)
```

```
    A
   A B
  A B C
 A B C D
A B C D E
```

57. Write a program to print the following pattern.

In [3]:
```
def apha(n):
    num = 65
    for i in range(n+1):
        for j in range(n-i):
            print(" ",end="")
        for j in range(1,i+1):
            print(chr(num),end =" ")
            num+=1
        print("")

apha(5)
```

```
    A
   B C
  D E F
 G H I J
K L M N O
```

58. Write a program to create pyramid with 5 rows.

In [6]:
```
n=5
for i in range(n):
    for j in range(i,n):
        print(" ",end=" ")
    for j in range(i+1):
        print("*",end=" ")
    for j in range(i):
        print("*",end=" ")
    print()
```

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

In [7]:
```
n=5
for i in range(n):
    for j in range(i,n):
        print(" ",end="")
    for j in range(i+1):
        print("*",end=" ")
    print()
```

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

59. Create a sandglass pattern.

In [54]:
```
n=5
for i in range(n):
    for j in range(i+1):
        print("",end=" ")
    for j in range(i,n):
        print("*",end=" ")
    print()
for i in range(n):
    for j in range(i,n):
        print("",end=" ")
    for j in range(i+1):
        print("*",end=" ")
    print()
```

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

59. Create a butterfly pattern.

In [13]:
```
n=5
for i in range(n):
    for j in range(i+1):
        print("*",end=" ")
    for j in range(i,n-1):
        print(" ",end=" ")
    for j in range(i,n-1):
        print(" ",end=" ")
    for j in range(i+1):
        print("*",end=" ")
    print()
for i in range(n):
    for j in range(i,n-1):
        print("*",end=" ")
    for j in range(i+1):
        print(" ",end=" ")
    for j in range(i+1):
        print(" ",end=" ")
    for j in range(i,n-1):
        print("*",end=" ")
    print()
```

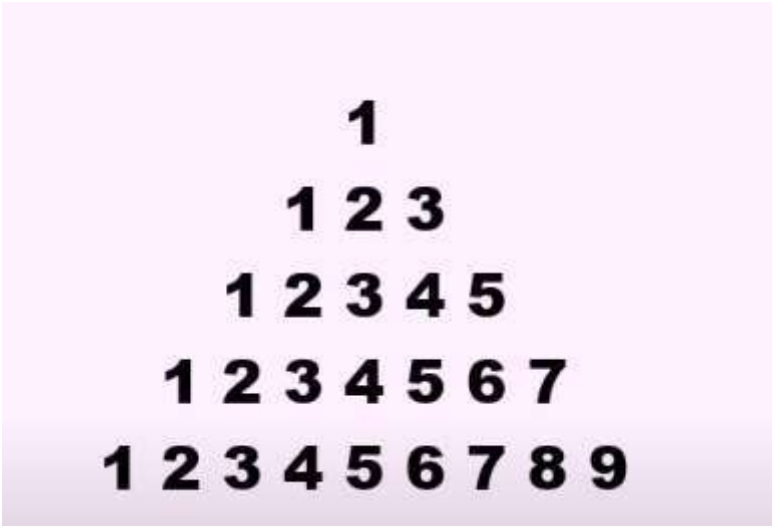
```
*                 *
* *             * *
* * *         * * *
* * * *     * * * *
* * * * * * * * * *
* * * *     * * * *
* * *         * * *
* *             * *
*                 *
```

60. Create a reverse pyramid.

In [73]:
```
n=5
for i in range(n):
    for j in range(i+1):
        print(" ",end="")
    for j in range(i,n):
        print("*",end=" ")
    print()
```

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

61. Create a hill triangle pattern.

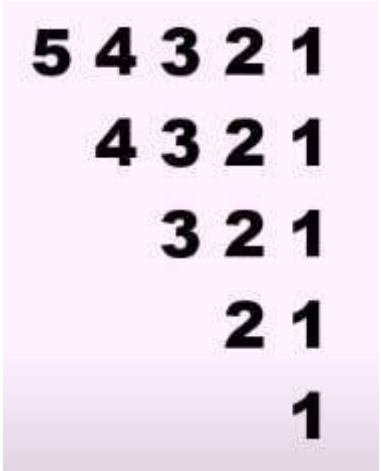


In [82]:
```
n=5

for i in range(n):
    p=1
    for j in range(i,n):
        print(" ",end=" ")
    for j in range(i):
        print(p,end=" ")
        p+=1
    for j in range(i+1):
        print(p,end=" ")
        p+=1
    print()
```

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

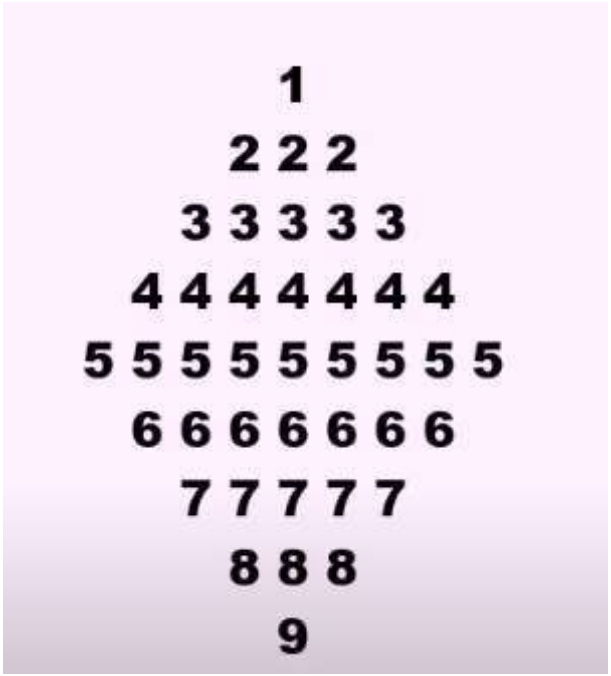
62. Create the following pattern.



In [94]:
```
n=5
p=5
for i in range(n):
    k=p
    for j in range(i+1):
        print(" ",end=" ")
    for j in range(i,n):
        print(k,end=" ")
        k-=1
    p-=1
    print()
```

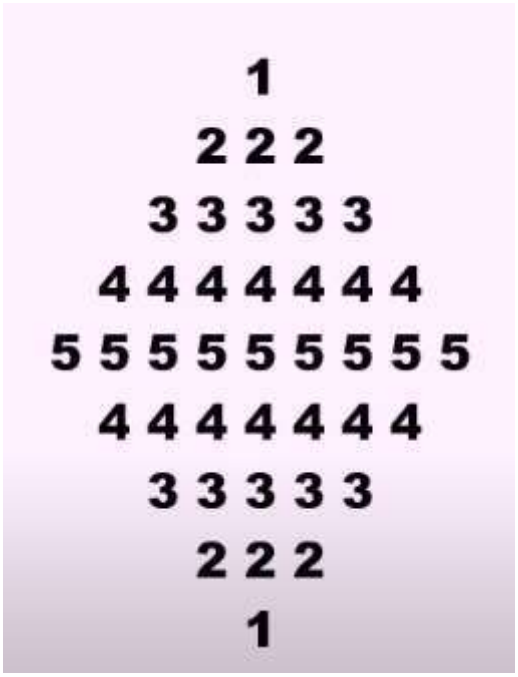
```
  5 4 3 2 1
    4 3 2 1
      3 2 1
        2 1
          1
```

63. Create the diamond patterns



In [107]:
```
n=5
p=1
for i in range(n-1):
    for j in range(i,n):
        print(" ",end=" ")
    for j in range(i+1):
        print(p,end=" ")
    for j in range(i):
        print(p,end=" ")
    p+=1
    print()
for i in range(n):
    for j in range(i+1):
        print(" ",end=" ")
    for j in range(i,n):
        print(p,end=" ")
    for j in range(i+1,n):
          print(p,end=" ")
    p+=1
    print()
```

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



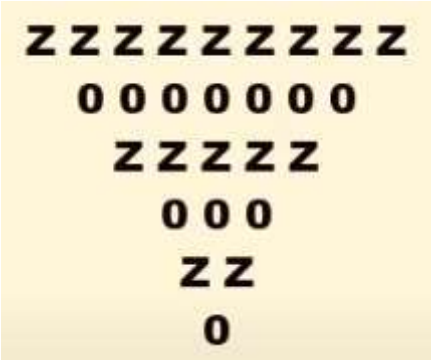
In [115]:
```
n=5
p=1
for i in range(n-1):
    for j in range(i,n):
        print(" ",end=" ")
    for j in range(i+1):
        print(p,end=" ")
    for j in range(i):
        print(p,end=" ")
    p+=1
    print()
for i in range(n):
    for j in range(i+1):
        print(" ",end=" ")
    for j in range(i,n):
        print(p,end=" ")
    for j in range(i+1,n):
          print(p,end=" ")
    p-=1
    print()
```

```
        1
      2 2 2
    3 3 3 3 3
  4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5
  4 4 4 4 4 4 4
    3 3 3 3 3
      2 2 2
        1
```

64. Print the following pattern.

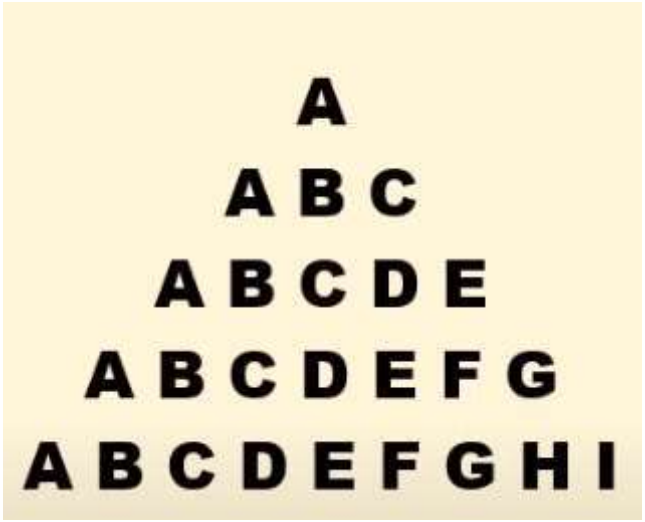
In [16]:
```
n=5
p=65
for i in range(n):
    for j in range(i+1):
        print(chr(p),end=" ")
    p+=2
    print()
```

```
A
C C
E E E
G G G G
I I I I I
```



In [24]:
```
n=6
for i in range(n):
    for j in range(i+1):
        print(" ",end=" ")
    for j in range(i,n):
        if i%2==0:
            print("z",end=' ')
        else:
            print("o",end=" ")
    for j in range(i,n-1):
        if i%2==0:
            print("z",end=' ')
        else:
            print("o",end=" ")
    print()
```

```
  z z z z z z z z z z z
    o o o o o o o o o
      z z z z z z z
        o o o o o
          z z z
            o
```

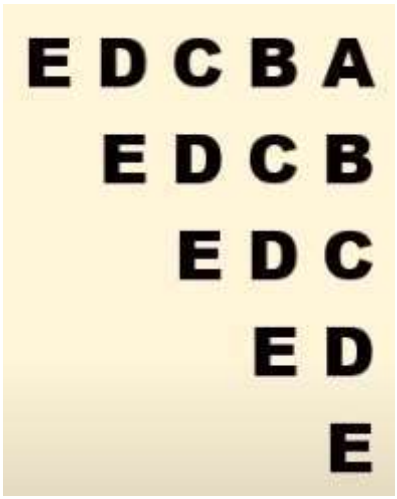


In [33]:
```
n=5
for i in range(n):
    p=65
    for j in range(i,n):
        print(" ",end=" ")
    for j in range(i):
        print(chr(p),end=" ")
        p+=1
    for j in range(i+1):
        print(chr(p),end=" ")
        p+=1
    print()
```

```
        A
      A B C
    A B C D E
  A B C D E F G
A B C D E F G H I
```

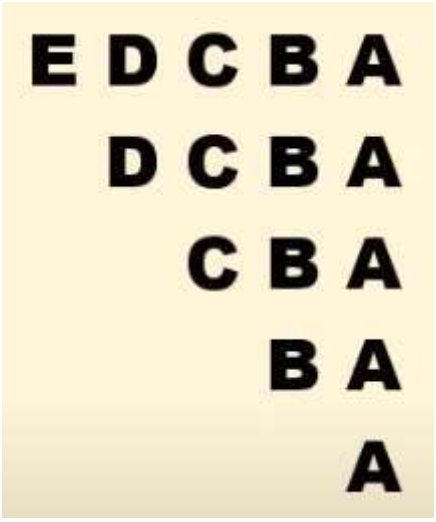
In [35]:
```
n=5
p=65
for i in range(n):
    for j in range(i,n):
        print(" ",end=" ")
    for j in range(i):
        print(chr(p),end=" ")
        p+=1
    for j in range(i+1):
        print(chr(p),end=" ")
        p+=1
    print()
```

```
          A
        B C D
      E F G H I
    J K L M N O P
  Q R S T U V W X Y
```



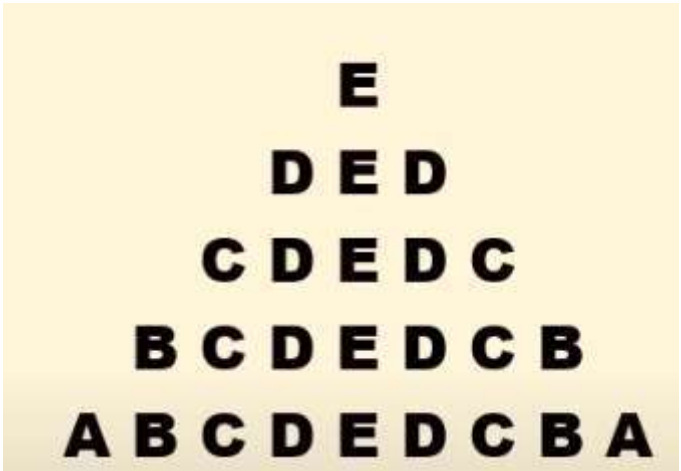
In [42]:
```
n=5
for i in range(n):
    p=69
    for j in range(i+1):
        print(" ",end=" ")
    for j in range(i,n):
        print(chr(p),end=" ")
        p-=1
    print()
```

```
  E D C B A
    E D C B
      E D C
        E D
          E
```



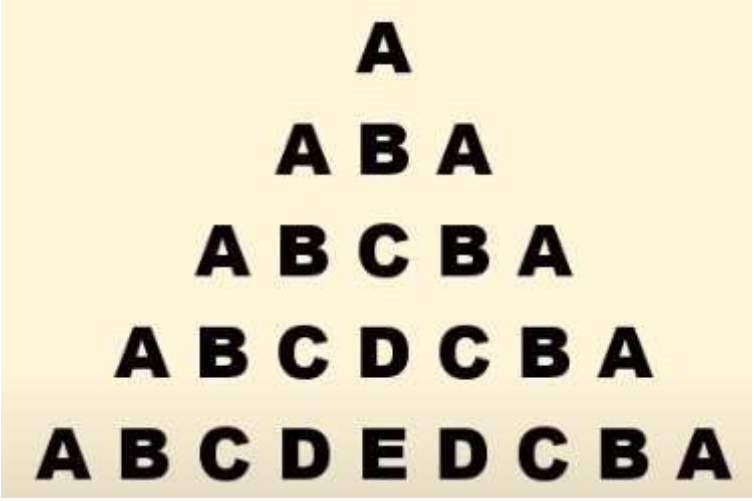
In [49]:
```
n=5
k=69
for i in range(n):
    p=k
    for j in range(i+1):
        print(" ",end=" ")
    for i in range(i,n):
        print(chr(p),end=" ")
        p-=1
    k-=1
    print()
    p=69
    p-=1
```

```
  E D C B A
    D C B A
      C B A
        B A
          A
```



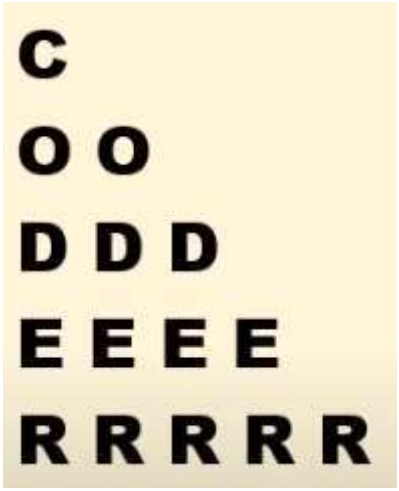
In [71]:
```
n=5
k=69
for i in range(n):
    p=k
    for j in range(i,n):
        print(" ",end=" ")
    for j in range(i):
        print(chr(p),end=" ")
        p+=1
    for j in range(i+1):
        print(chr(p),end=" ")
        p-=1
    k-=1
    print()
```

```
          E
        D E D
      C D E D C
    B C D E D C B
  A B C D E D C B A
```



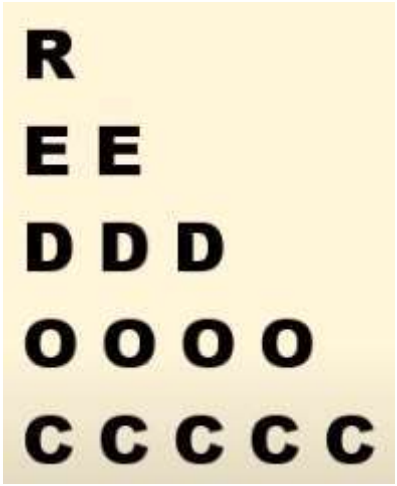
In [69]:
```
n=5
for i in range(n):
    p=65
    for j in range(i,n):
        print(" ",end=" ")
    for j in range(i):
        print(chr(p),end=" ")
        p+=1
    for j in range(i+1):
        print(chr(p),end=" ")
        p-=1
    print()
```

```
        A
      A B A
    A B C B A
  A B C D C B A
A B C D E D C B A
```



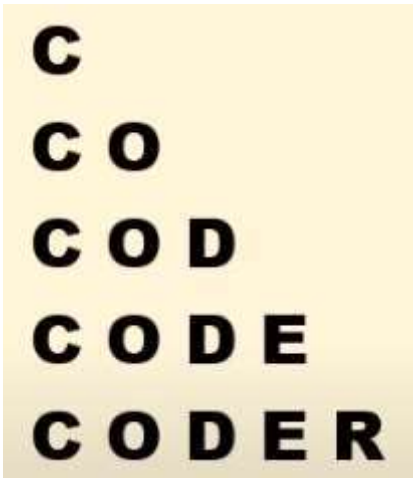
In [73]:
```
s="CODER"
n=len(s)
p=0
for i in range(n):
    for j in range(i+1):
        print(s[p],end=" ")
    p+=1
    print()
```

```
C
O O
D D D
E E E E
R R R R R
```



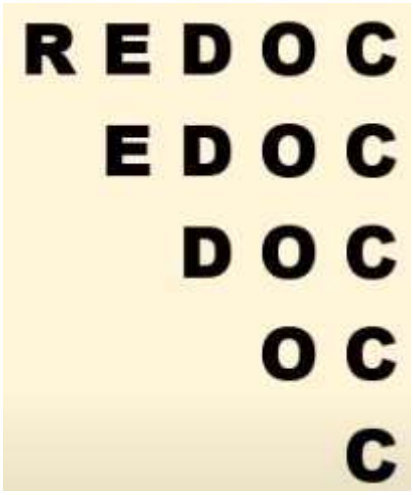
In [74]:
```
s="CODER"
n=len(s)
p=n-1
for i in range(n):
    for j in range(i+1):
        print(s[p],end=" ")
    p-=1
    print()
```

```
R
E E
D D D
O O O O
C C C C C
```



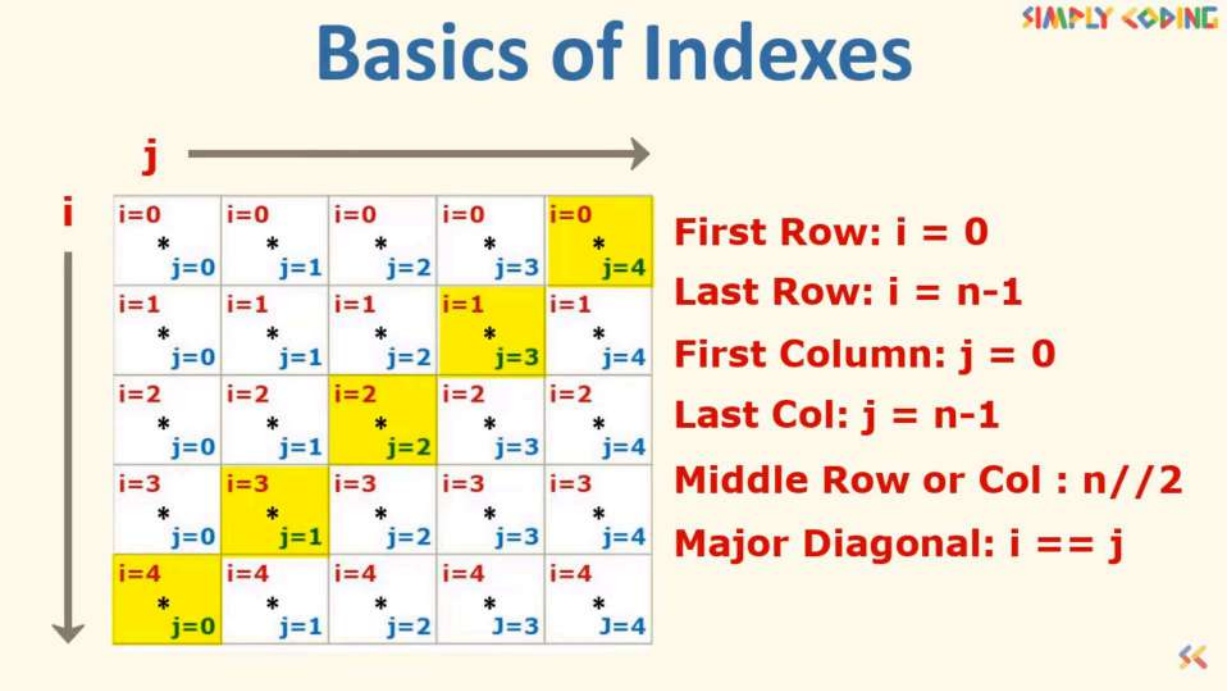
In [76]:
```
s="CODER"
n=len(s)
for i in range(n):
    p=0
    for j in range(i+1):
        print(s[p],end=" ")
        p+=1
    print()
```

```
C
C O
C O D
C O D E
C O D E R
```



In [80]:
```
s="CODER"
n=len(s)
k=n-1
for i in range(n):
    p=k
    for j in range(i+1):
        print(" ",end=" ")
    for j in range(i,n):
        print(s[p],end=" ")
        p-=1
    print()
    k-=1
```

```
  R E D O C
    E D O C
      D O C
        O C
          C
```



In [18]:
```
# square matrix
n=9
for i in range(n):
    for j in range(n):
        print("*",end=" ")
    print()
```

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

In [16]:
```
# parallel vertical lines
n=9
for i in range(n):
    for j in range(n):
        if j==0 or j==n-1:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    print()
```

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

In [9]:
```
# square plus pattern
n=9
for i in range(n):
    for j in range(n):
        if i==n//2 or j==n//2:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    print()
```

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

In [10]:
```
# cross pattern
n=9
for i in range(n):
    for j in range(n):
        if i==j or i+j==n-1:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    print()
```

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

In [13]:
```
#hollow square pattern:
n=9
for i in range(n):
    for j in range(n):
        if i==0 or i==n-1 or j==0 or j==n-1:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    print()
```

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

In [15]:
```
#hollow increasing triangle
n=9
for i in range(n):
    for j in range(n):
        if j==0 or i==n-1 or i==j:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    print()
```

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

In [20]:
```
#hollow decreasing triangle
n=9
for i in range(n):
    for j in range(n):
        if i==0 or j==0 or i+j==n-1:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    print()
```

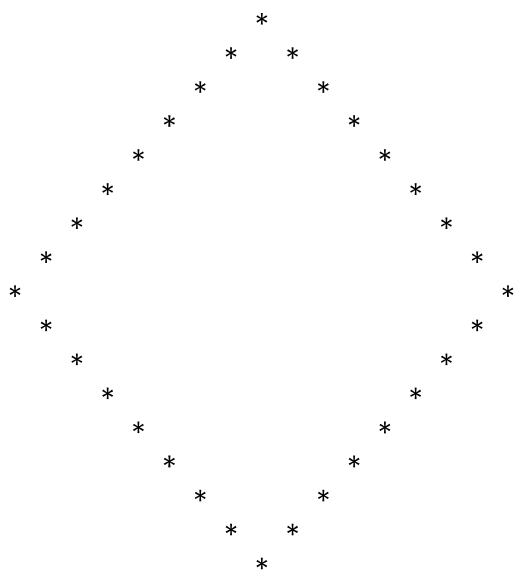
```
* * * * * * * * *
*             *
*           *
*         *
*       *
*     *
*   *
* *
*
```

In [1]:
```
# hollow hill pattern
n=9
for i in range(n):
    for j in range(i,n):
        print(" ",end=" ")
    for j in range(i):
        if j==0 or i==n-1:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    for j in range(i+1):
        if i==j or i==n-1:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    print()
```

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

In [2]:
```
# hollow diamond pattern
n=9
for i in range(n-1):
    for j in range(i,n-1):
        print(" ",end=" ")
    for j in range(i):
        if j==0:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    for j in range(i+1):
        if i==j:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    print()
for i in range(n):
    for j in range(i):
        print(" ",end=" ")
    for j in range(i,n):
        if j==i:
            print("*",end=" ")
        else:
            print(" ",end=" ")
    for j in range(i,n-1):
        if j==n-2:
            print("*",end=" ")
        else:
            print(" ",end=" ")

    print()
```



64. Check if a number is palindrome or not.

In [1]:
```
#initialization
n=1001
m=n
sum=0
while m!=0:
    #logic
    d=m%10
    sum=sum*10+d
    m=m//10
#check if true or not
if sum==n:
    print("Yes")
else:
    print("No")
```

```
Yes
```

In [ ]:
```
# Take input from the user
n = int(input("Enter a number: "))

# Store the original number
original_number = n

# Initialize sum to build the reversed number
reversed_number = 0

# Reverse the number
while n != 0:
    digit = n % 10
    reversed_number = reversed_number * 10 + digit
    n = n // 10

# Check if the reversed number matches the original number
if reversed_number == original_number:
    print("Yes")
else:
    print("No")
```

65. Check if a number is spy number. Means sum of its digits equal to the product of its digits.

In [8]:
```
n=132
m=n
sum=0
prod=1
while m!=0:
    d=m%10
    m=m//10
    sum=sum+d
    prod=prod*d
if sum==prod:
    print("Yes")
else:
    print("No")
```

```
Yes
```

66. Check if a number is a special number. If sum of the digits plus product of the digits is equal to the original number.

In [11]:
```
n=59
m=n
sum=0
prod=1
while m!=0:
    d=m%10
    m=m//10
    sum=sum+d;prod=prod*d
if sum+prod==n:
    print("Yes")
else:
    print("No")
```

```
Yes
```

In [ ]:
```
n = int(input("Enter a number: "))
sum_digits = 0
prod_digits = 1
original_number = n

while n != 0:
    digit = n % 10
    n = n // 10
    sum_digits += digit
    prod_digits *= digit

if sum_digits == prod_digits:
    print(f"{original_number} is a spy number")
else:
    print(f"{original_number} is not a spy number")
```

67. Check if a number is Harshad/Niven number. If the number is divisible by the sum of its digits.

In [15]:
```
n=156
m=n
sum=0
prod=1
while m!=0:
    d=m%10
    m=m//10
    sum=sum+d
if n%sum==0:
    print("Yes")
else:
    print("No")
```

```
Yes
```

In [2]:
```
n= int(input("Enter a number: "))
sum = 0
org = n
while n!= 0:
    digit = n%10
    n=n//10
    sum += digit
if org%sum == 0:
    print("Harshad number")
else:
    print("Not a Harshad number")
```

```
Enter a number: 24
Harshad number
```

68. Check if a number is duck number or not. Number which has zeros present in it.

In [27]:
```
n=1560000
m=n
count=0
while m!=0:
    d=m%10
    m=m//10
    if d==0:count+=1
if count>0:
    print("Yes")
else:
    print("No")
print("No of zeros in number:",count)
```

```
Yes
No of zeros in number: 4
```

69. Check if the number is Neon number. Number, digits of whose square are equal to the number itself.

In [30]:
```
n=9
m=n**2
sum=0
while m!=0:
    d=m%10
    sum=sum+d
    m=m//10
if sum==n:
    print("Yes")
else:
    print("No")
```

```
Yes
```

70. Check if a number is automorphic number or not. It is a number which is contained in the last digit(s) of its square. eg. 25 in 625.

In [39]:
```
#n=25
n=9
flag=0
m=n
q=n*n
while m!=0:
    d=m%10;d1=q%10
    if d!=d1:flag=1
    m=m//10;q=q//10
if flag==0:
    print("Yes")
else:
    print("No")
```

```
No
```

71. Check if a number is Krishna Murthy special number or not. A number which is equal to the sum of factorial of its digits. eg. 145 = 1!+4!+5!

In [3]:
```
import math
n=145
m=n
sum=0
while m!=0:
    d=m%10
    sum=sum+math.factorial(d)
    m=m//10
if sum==n:
    print("Yes")
else:
    print("No")
```

```
No
```

72. Find the factors of a number.

In [7]:
```
n=int(input("Enter a number: "))
for i in range(1,n+1):
    if n%i==0:
        print(i,end=" ")
```

```
Enter a number: 45
1 3 5 9 15 45
```

73. Find if a number is prime or not.

In [17]:
```
n=int(input("Enter a number: "))
count=0
for i in range(1,n+1):
    if n%i==0:count+=1
if count==2:
    print("Prime")
else:
    print("Not Prime OR composite")
```

```
Enter a number: 23
Prime
```

74. Find if a number is a perfect number. A number is equal to sum of its factors or divisors, other than the number itself.

In [19]:
```
n=int(input("Enter a number: "))
sum=0
for i in range(1,n-1):
    if n%i==0:
        sum=sum+i
if n==sum:
    print("Yes")
else:
    print("No")
```

```
Enter a number: 6
Yes
```

75. Find if a number is composite number or not. A number having more than one factor other than 1 and itself.

In [11]:
```
n=int(input("Enter a number: "))
count=0
for i in range(1,n+1):
    if n%i==0:count+=1
if count>2:
    print("Composite number")
else:
    print("Not a Composite number")
```

```
Enter a number: 34
Composite number
```

76. Find if the number is an abundant number or deficient . Here the sum of factors of the number

is greater than the number itself.

In [1]:
```
n=int(input("Enter a number: "))
sum=0
for i in range(1,n):
    if n%i==0:
        sum=sum+i
if sum>n:
    print("Abundant number")
else:
    print("Deficient number")
```

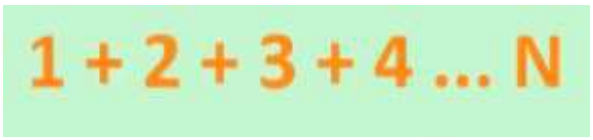
```
Enter a number: 34
Deficient number
```

77. Find if the number is pronic or not. Number which is the product of two consecutive numbers.

In [12]:
```
n=int(input("Enter a number: "))
fact=0
for i in range(1,n+1):
    if(i*(i+1)==n):
        print(f"{i} x {i+1} = {n}")
        fact=i
if fact>0:
    print("Pronic number")
else:
    print("Not a Pronic number number")
```

```
Enter a number: 56
7 x 8 = 56
Pronic number
```

78. Arithematic series - sequence where the difference between two consecutive terms are the same.

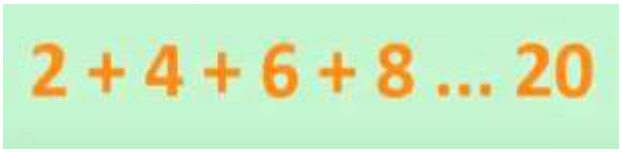


In [7]:
```
N=int(input(" Enter N: "))
sum=0
for i in range(1,N+1):
    sum=sum + i
print("Sum of series: ",sum)
```

```
 Enter N: 20
Sum of series:  210
```

In [6]:
```
N=int(input(" Enter N: "))
sum=0
a=9
for i in range(1,N+1):
    sum=sum + a
    a+=4
print("Sum of series: ",sum)
```

```
 Enter N: 20
Sum of series:  940
```



In [8]:
```
N=int(input(" Enter N: "))
sum=0
a=2
for i in range(1,N+1):
    sum=sum + a
    a+=2
print("Sum of series: ",sum)
```

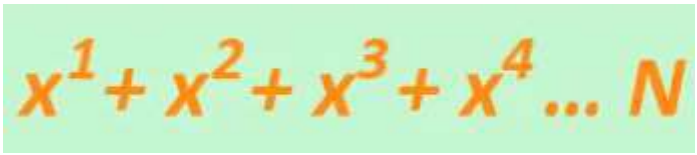
```
 Enter N: 20
Sum of series:  420
```

In [9]:
```
N=int(input(" Enter N: "))
sum=0
a=1
for i in range(1,N+1):
    sum=sum + a
    a+=2
print("Sum of series: ",sum)
```

```
 Enter N: 20
Sum of series:  400
```

In [12]:
```
N=int(input(" Enter N: "))
sum=0
a=10
for i in range(1,N+1):
    sum=sum + a
    a-=1
print("Sum of series: ",sum)
```

```
 Enter N: 10
Sum of series:  55
```



In [13]:
```
N=int(input(" Enter N: "))
X=int(input("Enter X: "))
sum=0
a=10
for i in range(1,N+1):
    sum=sum + X**i
print("Sum of series: ",sum)
```

```
 Enter N: 10
Enter X: 3
Sum of series:  88572
```

In [15]:
```
import math as m
N=int(input(" Enter N: "))
X=int(input("Enter X: "))
sum=0
a=9
for i in range(1,N+1):
    sum=sum+m.factorial(a)/X
    a+=4
print("Sum of series: ",sum)
```

```
 Enter N: 10
Enter X: 2
Sum of series:  5.981112715901029e+55
```

$$2^x + 4^x + 6^x + 8^x \ldots 20$$

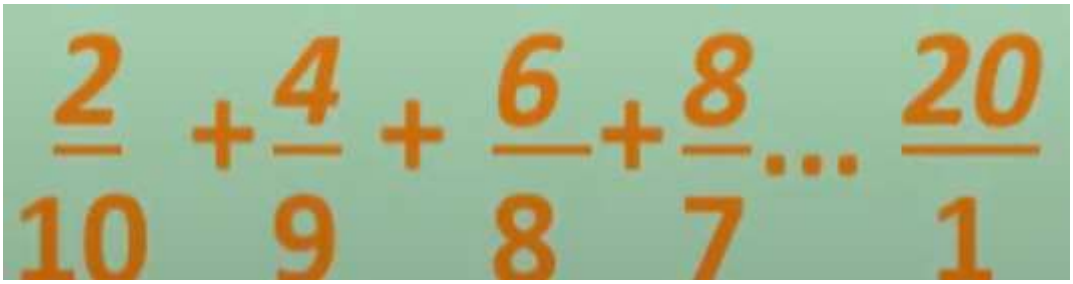
In [16]:
```
N=int(input(" Enter N: "))
X=int(input("Enter X: "))
sum=0
a=2
for i in range(1,N+1):
    sum=sum + a**X
    a+=2
print("Sum of series: ",sum)
```

```
 Enter N: 20
Enter X: 2
Sum of series:  11480
```

$$\frac{1^3}{x} + \frac{3^3}{x} + \frac{5^3}{x} + \frac{7^3}{x} \ldots N$$

In [17]:
```
N=int(input(" Enter N: "))
X=int(input("Enter X: "))
sum=0
a=1
for i in range(1,N+1):
    sum=sum + (a**3)/X
    a+=2
print("Sum of series: ",sum)
```

```
 Enter N: 20
Enter X: 2
Sum of series:  159800.0
```



In [19]:
```
N=10
sum=0
a=2
b=10
for i in range(1,N+1):
    sum=sum + a/b
    a+=2;b-=1
print("Sum of series: ",sum)
```

```
Sum of series:  44.43730158730159
```

79. Geometric Series - Every term is derived by multiplying previous term by a fixed number.

In [5]:
```
N=int(input("Enter N: "))
sum=0
a=2
for i in range(1,N+1):
    sum=sum + a*2
    a=a*2
print("Sum of series: ",sum)
```

```
Enter N: 20
Sum of series:  4194300
```

In [8]:
```
N=int(input("Enter N: "))
sum=0
a=2
for i in range(1,N+1):
    sum=sum + a
    a=a*3
print("Sum of series: ",sum)
```

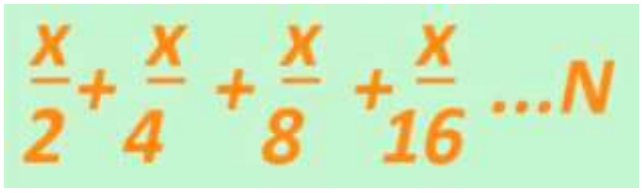
```
Enter N: 20
Sum of series:  3486784400
```

In [9]:
```
N=int(input("Enter N: "))
sum=0
a=10
for i in range(1,N+1):
    sum=sum + a
    a=a*3
print("Sum of series: ",sum)
```

```
Enter N: 20
Sum of series:  17433922000
```

In [11]:
```
N=int(input("Enter N: "))
sum=0
a=5
for i in range(1,N+1):
    sum=sum + a
    a=a*3
print("Sum of series: ",sum)
```

```
Enter N: 10
Sum of series:  147620
```



In [12]:
```
N=int(input("Enter N: "))
X=int(input("Enter X: "))
sum=0
a=2
for i in range(1,N+1):
    sum=sum + X/a
    a=a*2
print("Sum of series: ",sum)
```

```
Enter N: 10
Enter X: 2
Sum of series:  1.998046875
```

$$2 - 6 + 18 - 54 \ldots N$$

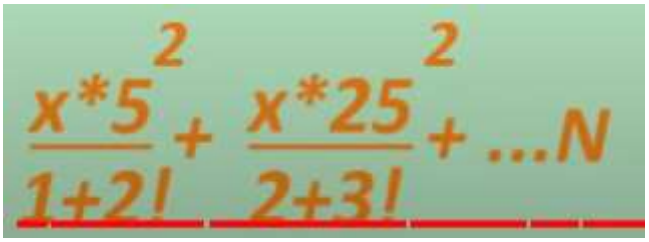
In [16]:
```
N=int(input("Enter N: "))
sum=0
a=2
for i in range(1,N+1):
    if i%2==0:
        sum=sum-a
    else:
        sum=sum+a
    a=a*3
print("Sum of series: ",sum)
```

```
Enter N: 20
Sum of series:  -1743392200
```

$$\frac{x+2}{10} + \frac{x+4}{30} + \frac{x+6}{90} + ..N$$

In [17]:
```
N=int(input("Enter N: "))
X=int(input("Enter X: "))
sum=0
a=2
b=10
for i in range(1,N+1):
    sum=sum+(X+a)/b
    a+=2
    b=b*3
print("Sum of series: ",sum)
```

```
Enter N: 10
Enter X: 2
Sum of series:  0.7499364934207186
```



In [1]:
```
import math as m
N=int(input("Enter N: "))
X=int(input("Enter X: "))
sum=0;a=5;b=2
for i in range(1,N+1):
    sum=sum+(X*a**2)/(i+m.factorial(b))
    a=a*5;b+=1
print("Sum of series: ",round(sum,2))
```

```
Enter N: 5
Enter X: 2
Sum of series:  34570.38
```

80. Convert Numbered string by reversing and converting to character string using ascii values [A-Z]=65-90 and [a-z]=97-122.

In [18]:
```
s=input('Enter the string:') #796115110113721110141108
s= s[::-1]
print(s)
i=0
res=""

while(i<len(s)-1):
    x=s[i]+s[i+1]
    if x=="32":
        res=res+" "
    elif int(x) in range(65,91) or int(x) in range(97,123):
        res=res+chr(int(x))
    elif i+2<len(s):
        x=x+s[i+2]
        res=res+chr(int(x))
        i+=1
    i+=2
print(res)
```

```
Enter the string:796115110113721110141108
80114101112731101151697
PrepInsta
```

81. What are the commonly used decorators in python ?

In [12]:
```
#Some commonly used decorators that are built into Python are @classmethod , @sta

#@classmethod decorator often used for operations that involve the class and its
class Myclass:
    a = "I am a class variable"

    def __init__(self,len):
        self.length = len

    @classmethod
    def class_method(cls):
        print("This is a class method.")
        print(f"using class variable:{cls.a}")

#using class method without creating an instance
Myclass.class_method()
```

```
This is a class method.
using class variable:I am a class variable
```

In [ ]:
```
#classmethod example

class InstanceCounter:
    count = 0

    def __init__(self):
        # Each time an instance is created, increment the count
        InstanceCounter.count += 1

    @classmethod
    def get_instance_count(cls):
        return cls.count

# Creating instances of the class
instance1 = InstanceCounter()
instance2 = InstanceCounter()
instance3 = InstanceCounter()

# Using the class method to get the instance count
total_instances = InstanceCounter.get_instance_count()

print(f"Total number of instances created: {total_instances}")
```

In [ ]:
```python
#@staticmethod decorator is used to define a static method within a class.
#A static method is a method that belongs to a class rather than an instance of t
#Unlike regular methods, static methods don't have access to the instance or its
#they are bound to the class itself.

class MyClass:
    class_variable = "I am a class variable"

    def __init__(self, instance_variable):
        self.instance_variable = instance_variable

    @staticmethod
    def static_method():
        print("This is a static method.")
        print("It doesn't have access to instance variables or self.")

# Using the static method without creating an instance
MyClass.static_method()
```

In [ ]:
```python
#static method calculator code
class Calculator:
    @staticmethod
    def add(x, y):
        return x + y

    @staticmethod
    def subtract(x, y):
        return x - y

    @staticmethod
    def multiply(x, y):
        return x * y

    @staticmethod
    def divide(x, y):
        if y != 0:
            return x / y
        else:
            print("Cannot divide by zero.")
            return None

# Using the static methods without creating an instance
result_add = Calculator.add(5, 3)
print(f"Addition: {result_add}")

result_subtract = Calculator.subtract(8, 4)
print(f"Subtraction: {result_subtract}")

result_multiply = Calculator.multiply(2, 6)
print(f"Multiplication: {result_multiply}")

result_divide = Calculator.divide(10, 2)
print(f"Division: {result_divide}")
```

In [1]:
```python
# In Python, the @property decorator is used to define a property on a class.
# Properties allow you to define getter, setter, and deleter methods for a class
# making it possible to encapsulate the access and modification of attributes wit

class Circle:
    def __init__(self,radius):
        self._radius = radius

    @property
    def radius(self):
        """Getter method for radius"""
        return self._radius

    @radius.setter
    def radius(self,value):
        """setter method for radius"""
        if value < 0:
            raise ValueError("Value cannot be negative")
        self._radius=value

    @property
    def area(self):
        """method to calculate the area"""
        return 3.14 * self._radius**2

#creating instance of Circle class
obj = Circle(radius=5)

print(f"radius:{obj.radius}")

print(f"area:{obj.area}")
```

```
radius:5
area:78.5
```

82. different types of parameters in python.

In [1]:
```
def print_params(x,y,z=3,*pospar,**keypar):
    print(x,y,z)
    print(pospar)
    print(keypar)

print_params(1,2,3,5,6,7, foo=1, bar=2)
print_params(1,2)
print_params(1,2,3,'Testing', foo=1, bar=2)
```

```
1 2 3
(5, 6, 7)
{'foo': 1, 'bar': 2}
1 2 3
()
{}
1 2 3
('Testing',)
{'foo': 1, 'bar': 2}
```

83. Operator overloading in python.

In [9]:
```
# The __add__ method allows you to use the + operator between two Point instances
# The __mul__ method allows you to use the * operator between two Point instances
# The __rmul__ method is called when the left operand of * is not an instance of
# The __repr__ method provides a string representation of your Point object.
class Point:
    def __init__(self,x=0,y=0):
        self.x = x
        self.y = y
    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)
    def __mul__(self, other):
        if isinstance(other, Point): # isinstance used to check if object is inst
            return Point(self.x * other.x, self.y * other.y)
        else:
            return Point(self.x * other, self.y * other)
    def __rmul__(self,other):
        return Point(self.x * other, self.y * other)
    def __repr__(self):
        return ("{0},{1}".format(self.x,self.y))

p1 = Point(2,3)
p2 = Point(3,4)

print(p1 + p2)
print(p1 * p2)
```

```
5,7
6,12
```

In [1]:
```
#Opening a file
try:
    fhand = open('jatin pg.txt')
    print(fhand)
except FileNotFoundError:
    print("Abe sahi file name enter kar le...")
```

```
<_io.TextIOWrapper name='jatin pg.txt' mode='r' encoding='cp1252'>
```

83. File Handling in python

In [5]:
```
# reading a file
fhand = open('mbox-short.txt')
inp = fhand.read()
print(len(inp))
print(inp[:20])
```

```
94626
From stephen.marquar
```

In [14]:
```
# reading a file
fhand = open('mbox-short.txt')
for line in fhand:
    if line.startswith("From:"):
        print(line)
```

```
From: stephen.marquard@uct.ac.za

From: louis@media.berkeley.edu

From: zqian@umich.edu

From: rjlowe@iupui.edu

From: zqian@umich.edu

From: rjlowe@iupui.edu

From: cwen@iupui.edu

From: cwen@iupui.edu

From: gsilver@umich.edu

From: gsilver@umich.edu

From: zqian@umich.edu

From: gsilver@umich.edu

From: wagnermr@iupui.edu

From: zqian@umich.edu

From: antranig@caret.cam.ac.uk

From: gopal.ramasammycook@gmail.com

From: david.horwitz@uct.ac.za

From: david.horwitz@uct.ac.za

From: david.horwitz@uct.ac.za

From: david.horwitz@uct.ac.za

From: stephen.marquard@uct.ac.za

From: louis@media.berkeley.edu

From: louis@media.berkeley.edu

From: ray@media.berkeley.edu

From: cwen@iupui.edu

From: cwen@iupui.edu

From: cwen@iupui.edu
```

In [15]:
```
# reading a file
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.startswith("From:"):
        print(line)
```

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: cwen@iupui.edu
From: cwen@iupui.edu
From: gsilver@umich.edu
From: gsilver@umich.edu
From: zqian@umich.edu
From: gsilver@umich.edu
From: wagnermr@iupui.edu
From: zqian@umich.edu
From: antranig@caret.cam.ac.uk
From: gopal.ramasammycook@gmail.com
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: louis@media.berkeley.edu
From: ray@media.berkeley.edu
From: cwen@iupui.edu
From: cwen@iupui.edu
From: cwen@iupui.edu
```

In [17]:
```
# reading a file
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith("From:"):
        continue
    print(line)
```

```
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: cwen@iupui.edu
From: cwen@iupui.edu
From: gsilver@umich.edu
From: gsilver@umich.edu
From: zqian@umich.edu
From: gsilver@umich.edu
From: wagnermr@iupui.edu
From: zqian@umich.edu
From: antranig@caret.cam.ac.uk
From: gopal.ramasammycook@gmail.com
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: david.horwitz@uct.ac.za
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: louis@media.berkeley.edu
From: ray@media.berkeley.edu
From: cwen@iupui.edu
From: cwen@iupui.edu
From: cwen@iupui.edu
```

In [2]:
```
# input file name to read and handle exceptions
fname = input("Enter the file name:")
try:
    fhand = open(fname)
except:
    print("File cannot be opened:",fname)
    exit()
count = 0
for line in fhand:
    if line.startswith('Subject:'):
        count = count + 1
print('There were', count, 'subject lines in',fname)
```

```
Enter the file name:mbox-short.txt
There were 27 subject lines in mbox-short.txt
```

In [7]:
```
# writing files
fout = open('output.txt', 'w')
print(fout)

line1 = "This is the first line\n"
fout.write(line1)

line2 = "This is the second line\n"
fout.write(line2)

# Close the file after writing
fout.close()

# Open the file in read mode
fout = open('output.txt', 'r')

# Read the contents of the file
inp = fout.read()
print(inp)

# Close the file after reading
fout.close()
```

```
<_io.TextIOWrapper name='output.txt' mode='w' encoding='cp1252'>
This is the first line
This is the second line
/n
```

84. When you encounter a line that starts with "X-DSPAM-Confidence:" pull apart the line to extract the floating-point number on the line. Count these lines and then compute the total of the spam confidence values from these lines. When you reach the end of the file, print out the average spam confidence.

In [5]:
```
try:
    fname = input("Enter file name:")
    fhand = open(fname)
except:
    if fname == 'na na boo boo':
        print("NA NA BOO BOO TO YOU - You have been punk'd!")
    else:
        print("Please enter correct file name:",fname)
count = 0
total = 0
try:
    for line in fhand:
        if line.startswith("X-DSPAM-Confidence:"):
            count = count + 1
            start = line.find(" ")
            num = float(line[start:])
            total += num
    avg = total/count
    print("Average spam confidence:",avg)
except ZeroDivisionError:
    print("C")
```

```
Enter file name:na na boo boo
NA NA BOO BOO TO YOU - You have been punk'd!
```

In [ ]: