**123 pattern:**

n = int(input("Enter number of rows: "))

for i in range(1, n-1):

for j in range(1, i-1):

print(j, end=" ")

print()

**student user:**

def find\_student\_users():

total\_users = int(input("Enter total users: "))

if total\_users <= 0:

print("Invalid input. Total users should be a positive integer.")

return

staff\_users = int(input("Enter staff users: "))

if staff\_users > total\_users:

print("Invalid input. Staff users cannot be more than total users.")

return

non\_teaching\_staff = staff\_users // 3

student\_users = total\_users - staff\_users - non\_teaching\_staff

print("Student Users:", student\_users)

find\_student\_users()

**decimal converter:**

def convert\_decimal():

decimal\_num = input("Enter a decimal number: ")

try:

decimal\_num = int(decimal\_num)

if decimal\_num < 0:

print("Invalid input. Decimal number should be a non-negative integer.")

return

binary\_num = bin(decimal\_num).replace("0b", "")

octal\_num = oct(decimal\_num).replace("0o", "")

print("Binary Number =", binary\_num)

print("Octal =", octal\_num)

except ValueError:

print("Invalid input. Decimal number should be a non-negative integer.")

convert\_decimal()

**pascals triangle:**

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

space = 36

# empty list containg all 0s

a = [0] \* 20

print("\n\t\t\t\t\*\*\* PASCAL TRIANGLE \*\*\*\n")

for i in range(row):

for spi in range(1,space+1):

print(" ", end="")

a[i] = 1

for j in range(i+1):

print('%6d' %(a[j]), end = "")

for j in range(i,0,-1):

a[j] = a[j] + a[j-1]

space = space - 3

print()

**Salary Grade:**

def calculate\_bonus():

grade = input("Enter the grade of the employee: ")

salary = float(input("Enter the employee salary: "))

print("Salary=", salary)

if grade.upper() == "A":

bonus\_percentage = 0.05

elif grade.upper() == "B":

bonus\_percentage = 0.10

else:

print("Invalid grade. Only A and B are valid.")

return

bonus = salary \* bonus\_percentage

if salary < 10000:

bonus += salary \* 0.02

print("Bonus=", bonus)

print("Total to be paid:", salary + bonus)

calculate\_bonus()

**perfect number:**

def is\_perfect(num):

sum = 0

for i in range(1, num):

if num % i == 0:

sum += i

return sum == num

n = int(input("Enter the number of perfect numbers to print: "))

count = 0

num = 1

while count < n:

if is\_perfect(num):

print(num, end=", ")

count += 1

num += 1

**factors:**

def count\_factors(n):

factors = []

for i in range(1, n + 1):

if n % i == 0:

factors.append(i)

return factors

def nth\_factor(factors, n):

if n > len(factors):

return "N is greater than the number of factors"

else:

return factors[n - 1]

num = int(input("Given Number: "))

n = int(input("N = "))

factors = count\_factors(num)

print("Number of factors =", len(factors))

print(f"{n}th factor of {num} =", nth\_factor(factors, n))

**simple interest:**

def calculate\_simple\_interest(principal, years, is\_senior\_citizen):

if is\_senior\_citizen.lower() == 'y':

rate = 0.12

else:

rate = 0.10

interest = (principal \* years \* rate)

return interest

principal = float(input("Enter the principal amount: "))

years = float(input("Enter the no of years: "))

is\_senior\_citizen = input("Is customer senior citizen (y/n): ")

interest = calculate\_simple\_interest(principal, years, is\_senior\_citizen)

print("Interest:", int(interest))

**Hollow Square:**

def print\_hollow\_square(size, symbol):

for i in range(size):

for j in range(size):

if i == 0 or i == size - 1 or j == 0 or j == size - 1:

print(symbol, end="")

else:

print(" ", end="")

print()

symbol = input("Enter a symbol: ")

size = int(input("Enter the size of the square: "))

print\_hollow\_square(size, symbol)

**pythagorous triple:**

def generate\_pythagorean\_triplets(limit):

triplets = []

for a in range(1, limit):

for b in range(a, limit):

c = (a\*2 + b2)\*0.5

if c <= limit and c == int(c):

triplets.append((a, b, int(c)))

return triplets

limit = int(input("Enter the limit: "))

triplets = generate\_pythagorean\_triplets(limit)

for triplet in triplets:

  print(triplet)