1. Multiplication Table Generator (Nested Loops)

Write a program that generates multiplication tables from 1 to 10. Use a nested loop: one loop for rows (1–10) and another for columns (1–10). Print each row on a new line.

```
#variable
for i in range(1,11):
    for j in range(1,11):
        print(f"{i*j:4}" , end= " " )
    print()
```

```
   1    2    3    4    5    6    7    8    9   10 
   2    4    6    8   10   12   14   16   18   20 
   3    6    9   12   15   18   21   24   27   30 
   4    8   12   16   20   24   28   32   36   40 
   5   10   15   20   25   30   35   40   45   50 
   6   12   18   24   30   36   42   48   54   60 
   7   14   21   28   35   42   49   56   63   70 
   8   16   24   32   40   48   56   64   72   80 
   9   18   27   36   45   54   63   72   81   90 
  10   20   30   40   50   60   70   80   90  100 
```

## 2. Prime Numbers Finder Write a program to display all prime numbers between 1 and 100. Use a loop to iterate through numbers. Use an inner loop or if statements to check if a number is divisible only by 1 and itself

```
for num in range(2,101):
    prime = True
    for i in range(2, int((num**0.5) + 1)):
        if num % i == 0:
            prime = False
            break
    if prime:
        print(num, end = " ")
        print()
```

```
2 
3 
5 
7 
11 
13 
17 
19 
23 
29 
31 
37 
```

```
41
43
47
53
59
61
67
71
73
79
83
89
97
```

3. Seating Arrangement Checker (Nested Loops)

Simulate a classroom seating arrangement. There are 3 rows, each with 4 seats. Display a seating chart like: Row 1: Seat 1, Seat 2, Seat 3, Seat 4 Row 2: Seat 1, Seat 2, Seat 3, Seat 4

```
#variable
seats = ["Seat 1", "Seat 2", "Seat 3", "Seat 4"]
for i in range(len(seats)):
    print(f"Row {i+1}: ", end= " ")
    for j in seats:
        print(j, end =", ")
    print()
```

```
Row 1:  Seat 1, Seat 2, Seat 3, Seat 4,
Row 2:  Seat 1, Seat 2, Seat 3, Seat 4,
Row 3:  Seat 1, Seat 2, Seat 3, Seat 4,
Row 4:  Seat 1, Seat 2, Seat 3, Seat 4,
```

4. Unique Word counter

Write a program to count unique words in a given sentence. Use a loop to split the input string into words. Use an if condition to check if a word already exists in a list.

```
str1 = "Generate random sentences from a vast database of words,
phrases, and sentence structures."
unique = []
for char in str1:

    if char not in unique:
        unique.append(char)
print(f"Lenght if unique characters in the list is {len(unique)}.")
```

```
Lenght if unique characters in the list is 21.
```

5. Library Book Management

A library system has 5 racks with up to 10 books each. Use nested loops to list the books in each rack. Ask the user to search for a book and display which rack it's in

```
#variable
library = {"Rack1": ["a1", "a2", "a3", "a4", "a5", "a6", "a7", "a8",
"a9", "a10"],
            "Rack2": ['b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7', 'b8',
'b9', 'b10'],
            "Rack3": ['c1', 'c2', 'c3', 'c4', 'c5', 'c6', 'c7', 'c8',
'c9', 'c10'],
            "Rack4": ['d1', 'd2', 'd3', 'd4', 'd5', 'd6', 'd7', 'd8',
'd9', 'd10'],
            "Rack5": ['e1', 'e2', 'e3', 'e4', 'e5', 'e6', 'e7', 'e8',
'e9', 'e10']
           }
for rack,books in library.items():
    print(f"{rack}: ")
    for book in books:
        print(f"{book}")
    print()

book_search1 = str(input("Enter book name: "))

for rack,books in library.items():
    if book_search1 in books:
        print(f"This book {book_search1} is in {rack}")
        break
```

```
Rack1:
a1
a2
a3
a4
a5
a6
a7
a8
a9
a10

Rack2:
b1
b2
b3
b4
b5
b6
```

```
b7
b8
b9
b10

Rack3:
c1
c2
c3
c4
c5
c6
c7
c8
c9
c10

Rack4:
d1
d2
d3
d4
d5
d6
d7
d8
d9
d10

Rack5:
e1
e2
e3
e4
e5
e6
e7
e8
e9
e10


Enter book name:  c5

This book c5 is in Rack3
```

6. Pattern Printing (Triangle)

Write a program to print the following triangle pattern using nested loops:

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

```python
for i in range(1,6):
    for j in range(1,i+1):
        print(j, end=" ")
    print()
```

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

7. Temperature Comparison Across Cities

Simulate a weather app that compares temperatures across 3 cities for 7 days. Use nested loops to iterate over cities and days. Display the highest and lowest temperatures for each city

```python
#variables

cities = {"Jaipur":[10,12,14,20,11,17,9],
          "Mumbai":[11,13,16,15,18,20,16],
          "Gurugram":[12,14,11,9,8,10,12]}

for city,temp in cities.items():
    print(f"Maximum temprature of {city} {max(temp)}.")
    print(f"Minimum temprature is {city} {min(temp)}.")
    print()
```

```
Maximum temprature of Jaipur 20.
Minimum temprature is Jaipur 9.

Maximum temprature of Mumbai 20.
Minimum temprature is Mumbai 11.
```

```
Maximum temprature of Gurugram 14.
Minimum temprature is Gurugram 8.
```

## 8. Password Strength Checker Write a program to check the strength of multiple passwords. Use a loop to validate a list of passwords. Use conditions to check: length, special characters, and numbers

```python
#variables

list_password = ["Abhi$1234", "zack#ujhki" , "shake4215", "ab#kb"]
special_characters = "!@#$%^&"

min_length = 8

for password in list_password:
    length_check = False
    special_characters_check = False
    num_check = False

    if len(password) >= min_length:
        length_check = True

    for char in password:
        if char in special_characters:
            special_characters_check = True
        if char.isdigit():
            num_check = True

    if length_check and special_characters and num_check:
        print(f"Your password {password} is strong.")
    elif length_check and (special_characters or num_check):
        print(f"Your password {password} is moderate.")
    else:
        print(f"Your password {password} is weak.")
```

```
Your password Abhi$1234 is strong.
Your password zack#ujhki is moderate.
Your password shake4215 is strong.
Your password ab#kb is weak.
```

9. Inventory Tracker (Nested Loops)

Create a program to manage stock levels in a warehouse with multiple sections. Each section has 5 items. Display items running low (<5 units) and their section numbers

```python
#variable
```

```
warehouse = {"section1" : [10, 20, 3, 14, 11],
             "section2" : [9, 2, 7, 16, 19],
             "section3" : [6, 4, 17, 18, 20],
             "section4" : [16, 14, 7, 1, 22]
             }

for sections ,item_list in warehouse.items():
    for item in item_list:
        if item < 5:
            print(f"{item} of {sections} is low.")
```

```
3 of section1 is low.
2 of section2 is low.
4 of section3 is low.
1 of section4 is low.
```

10. Tic-Tac-Toe Grid

Simulate a 3x3 Tic-Tac-Toe grid. Use nested loops to create and display the grid. Allow the user to mark cells (e.g., "X" or "O") by inputting row and column numbers

```
#variable

board = [[' 'for _ in range(3)] for _ in range(3)] # make the board
space
current_player = "X"
game_over = False

while not game_over:
    print("\n Let's start the game:")
    for row in board:             # make the board with rows and
columns
        print("|".join(row))
        print("-"*5)

    print(f"Current player is {current_player}.")
    row = int(input("Enter row (0,1,2)"))           # take input for
row and column
    column = int(input("Enter column (0,1,2)"))

    if board[row][column] == ' ':             # check if the space is
filled or empty
        board[row][column] = current_player
    else:
        print(f"These {row} and {column} space is filled please try on
another place.")
        continue

    # check for the winner
    winner = False
    for i in range(3):
        if board[0][i] == board[1][i] == board[2][i] != " ": # ROW
check
            winner = True
            break
        if board[i][0] == board[i][1] == board[i][2] != " ": # COLUMN
check
            winner = True
            break
        if board[0][0] == board[1][1] == board[2][2] != " " or
board[0][2] == board[1][1] == board[2][0] != " ": # DIAGONAL check
            winner = True
            break

    if winner:
        print(f"{current_player} is the winner.")
        game_over = True
        continue

    draw = True

    for row in board:
        if ' ' in row:           # check for empty space
            draw = False
            break

    if draw:
        print("The match is draw.")
        game_over = True
        continue

    current_player = 'O' if current_player == 'X' else 'X' #checking
for the player chance
```

```
 Let's start the game:
 | |
-----
 | |
-----
 | |
-----
Current player is X.

Enter row (0,1,2) 0
Enter column (0,1,2) 0


 Let's start the game:
```

```
X| |
-----
 | |
-----
 | |
-----
Current player is O.

Enter row (0,1,2) 1
Enter column (0,1,2) 1


 Let's start the game:
X| |
-----
 |O|
-----
 | |
-----
Current player is X.

Enter row (0,1,2) 1
Enter column (0,1,2) 2


 Let's start the game:
X| |
-----
 |O|X
-----
 | |
-----
Current player is O.

Enter row (0,1,2) 2
Enter column (0,1,2) 2


 Let's start the game:
X| |
-----
 |O|X
-----
 | |O
-----
Current player is X.

Enter row (0,1,2) 0
Enter column (0,1,2) 2
```

```
 Let's start the game:
X| |X
-----
 |O|X
-----
 | |O
-----
Current player is O.

Enter row (0,1,2) 2
Enter column (0,1,2) 0


 Let's start the game:
X| |X
-----
 |O|X
-----
O| |O
-----
Current player is X.

Enter row (0,1,2) 0
Enter column (0,1,2) 1

X is the winner.
```

11. Palindrome Checker

Write a program to check if multiple strings from a list are palindromes. Use a loop to iterate through the list. Use if to compare each string with its reverse

```python
#variable
palindrome = ["madam", "racecar", "hello"]

for char in palindrome:
    if char == char[::-1]:
        print("True")
    else:
        print("False")
```

```
True
True
False
```

## 12. Employee Attendance Tracker Simulate tracking attendance for 10 employees across 5 days. Use nested loops to record attendance (P for present, A for absent). Display a summary of each employee’s attendance at the end

```python
#variable

attendence = employees_attendance = {
    "emp1": ["P", "A", "P", "P", "A"],
    "emp2": ["P", "P", "A", "P", "P"],
    "emp3": ["A", "P", "P", "A", "P"],
    "emp4": ["P", "P", "P", "P", "A"],
    "emp5": ["A", "P", "A", "P", "P"],
    "emp6": ["P", "A", "P", "P", "A"],
    "emp7": ["P", "P", "A", "P", "P"],
    "emp8": ["A", "P", "P", "A", "P"],
    "emp9": ["P", "P", "P", "P", "A"],
    "emp10": ["A", "P", "A", "P", "P"]
}

for employee,attendence in attendence.items():
    print(f"Summary of {employee} attendence is {attendence}.")
```

```
Summary of emp1 attendence is ['P', 'A', 'P', 'P', 'A'].
Summary of emp2 attendence is ['P', 'P', 'A', 'P', 'P'].
Summary of emp3 attendence is ['A', 'P', 'P', 'A', 'P'].
Summary of emp4 attendence is ['P', 'P', 'P', 'P', 'A'].
Summary of emp5 attendence is ['A', 'P', 'A', 'P', 'P'].
Summary of emp6 attendence is ['P', 'A', 'P', 'P', 'A'].
Summary of emp7 attendence is ['P', 'P', 'A', 'P', 'P'].
Summary of emp8 attendence is ['A', 'P', 'P', 'A', 'P'].
Summary of emp9 attendence is ['P', 'P', 'P', 'P', 'A'].
Summary of emp10 attendence is ['A', 'P', 'A', 'P', 'P'].
```

13. Grade Categorizer

Write a program to categorize student grades. Input grades for 5 students using a loop. Use if conditions to assign letter grades (A, B, C, etc.) and display the results

```python
#variable

student_marks = {"stu1": 88, "stu2": 90, "stu3": 75, "stu4": 67,
"stu5": 58}
report = {}
for student,marks in student_marks.items():
    if marks>=90:
        report[student] = "A"
    elif 90>= marks >=80:
        report[student] = "B"
    elif 80> marks >70:
        report[student] = "C"
    elif 70> marks >60:
        report[student] = "D"
    elif 60> marks >50:
        report[student] = "E"
    elif 50> marks >40:
        report[student] = "F"

print(f"Grade of all students is {report}")
```

```
Grade of all students is {'stu1': 'B', 'stu2': 'A', 'stu3': 'C',
'stu4': 'D', 'stu5': 'E'}
```

14. Pyramid Pattern (Numbers)

Write a program to print the following pattern:

1

1 2

1 2 3

1 2 3 4

```
num = 4

for i in range(1,num+1):
    print(" "*(num-i), end = " ")
    for j in range(1,i+1):
         print(j, end = " ")
    print()
```

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

15. Sales Data Analysis

- Analyze sales data for 3 products over 4 months.
- Use nested loops to store and display sales data.
- Highlight the product with the highest sales each month

```
#variable

sales_data = {"Product1":[120000, 32000, 25000, 10000],
              "Product2":[10000, 25000, 45000, 40500],
              "Product3":[100000, 18000, 35000, 20000]
             }

months = ["January" , "February", "March", "April"]
```

```
#product and total sales
for i in sales_data.keys():
    for j in sales_data.values():
        print(f"{i}:{j}")

#highest sales each month
print("\nHighest Sales Each Month:")
for k in range(len(months)):
    max_month_sale = 0
    for i,j in sales_data.items():
        if j[k] > max_month_sale:
            max_month_sale = j[k]
            print(f"{months[k]}:{max_month_sale} of product {i}.")
```

```
Product1:[120000, 32000, 25000, 10000]
Product1:[10000, 25000, 45000, 40500]
Product1:[100000, 18000, 35000, 20000]
Product2:[120000, 32000, 25000, 10000]
Product2:[10000, 25000, 45000, 40500]
Product2:[100000, 18000, 35000, 20000]
Product3:[120000, 32000, 25000, 10000]
Product3:[10000, 25000, 45000, 40500]
Product3:[100000, 18000, 35000, 20000]

Highest Sales Each Month:
January:120000 of product Product1.
February:32000 of product Product1.
March:25000 of product Product1.
March:45000 of product Product2.
April:10000 of product Product1.
April:40500 of product Product2.
```

16. Palindrome Numbers in a Range

Write a program to find all palindrome numbers between 10 and 200. Use a loop to iterate through the range. Use if conditions to check if a number reads the same backward as forward

```
#variable

for i in range(10,201):
    if str(i) == str(i)[::-1]:
        print(i)
```

```
11
22
33
44
55
66
```

```
77
88
99
101
111
121
131
141
151
161
171
181
191
```

17. Magic Square Validator

Write a program to check if a given 3x3 matrix is a magic square (all rows, columns, and diagonals add up to the same number). Use nested loops to iterate through the matrix. Use if conditions to validate the sums

```python
#variable
matrix = [[8,1,6],
          [3,5,7],
          [4,9,2]
        ]

magic_square = False
total = 15

for i in range(len(matrix)):
    if matrix[0][i] + matrix[1][i] + matrix[2][i] == total: #Row check
        magic_square = True

    if matrix[i][0] + matrix[i][1] + matrix[i][2] == total: #Column
check
        magic_square = True

    if matrix[0][0] == matrix[1][1] == matrix[2][2] == total or
matrix[0][2] == matrix[1][1] == matrix[2][0] == total: # DIAGONAL
check
        magic_square = True

if magic_square == True:
    print("It is a magic s")
```

18. Meal Planner

- Simulate a meal planner for a week.
- Use a nested loop to list 7 days and 3 meals (breakfast, lunch, dinner).

- Allow the user to input or modify meals for each day

```
#variable

days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday","Sunday"]
meals = ["Breakfast", "Lunch", "Dinner"]

meal_planner = {}

for day in days:
    meal_planner[day] = {}
    print(f"{day}:")
    for meal in meals:
        meals_input = str(input(f"Enter meal for {meal}:"))
        if meals_input:
            meal_planner[day][meal] = meals_input

print("\n Meal plan for 7 days:-")
for day in days:
    print(f"\n{day}:")
    for meal in meals:
        print(f"  {meal}: {meal_planner[day][meal]}")

Monday:

Enter meal for Breakfast: bb
Enter meal for Lunch: ll
Enter meal for Dinner: dd

Tuesday:

Enter meal for Breakfast: kk
Enter meal for Lunch: jj
Enter meal for Dinner: ll

Wednesday:

Enter meal for Breakfast: qq
Enter meal for Lunch: ww
Enter meal for Dinner: ee

Thursday:

Enter meal for Breakfast: rr
Enter meal for Lunch: tt
Enter meal for Dinner: yy

Friday:

Enter meal for Breakfast: uu
Enter meal for Lunch: ii
Enter meal for Dinner: oo
```

```
Saturday:

Enter meal for Breakfast: pp
Enter meal for Lunch: zz
Enter meal for Dinner: xx

Sunday:

Enter meal for Breakfast: cc
Enter meal for Lunch: vv
Enter meal for Dinner: nn


 Meal plan for 7 days:-

Monday:
  Breakfast: bb
  Lunch: ll
  Dinner: dd

Tuesday:
  Breakfast: kk
  Lunch: jj
  Dinner: ll

Wednesday:
  Breakfast: qq
  Lunch: ww
  Dinner: ee

Thursday:
  Breakfast: rr
  Lunch: tt
  Dinner: yy

Friday:
  Breakfast: uu
  Lunch: ii
  Dinner: oo

Saturday:
  Breakfast: pp
  Lunch: zz
  Dinner: xx

Sunday:
  Breakfast: cc
  Lunch: vv
  Dinner: nn
```