SORTING TECHNIQUES:

Table of Complexity Comparison:

Name	Best Case	Average Case	Worst Case	Memory	Stable	Method Used
Quick Sort	$nlogn$	$nlogn$	n^2	$logn$	No	Partitioning
Merge Sort	$nlogn$	$nlogn$	$nlogn$	n	Yes	Merging
Heap Sort	$nlogn$	$nlogn$	$nlogn$	1	No	Selection
Insertion Sort	n	n^2	n^2	1	Yes	Insertion
Tim Sort	n	$nlogn$	$nlogn$	n	Yes	Insertion & Merging
Selection Sort	n^2	n^2	n^2	1	No	Selection
Shell Sort	$nlogn$	$n^{4/3}$	$n^{3/2}$	1	No	Insertion
Bubble Sort	n	n^2	n^2	1	Yes	Exchanging

Q: SELECTION SORT : Always bring minimum element at front in each iteration.

TIME COMPLEXITY: O(N^2) in all cases, best, avg, worst

```python
from typing import List

def selectionSort(arr: List[int]) -> None:
    for i in range(len(arr)):
        mini= i
        for j in range(i,len(arr)):
            if arr[j]<=arr[mini]:
                arr[mini],arr[j]=arr[j], arr[mini]
```

Q: BUBBLE SORT: Always bring maximum element at end. TC: O(N^2)

```python
from typing import List

def bubbleSort(arr: List[int], n: int):
    for i in range(n-1,-1,-1):
        for j in range(i):
            if arr[j]>arr[j+1]:
                arr[j],arr[j+1]= arr[j+1],arr[j]
```

OPTIMIZE THE BUBBLE SORT

We can optimize this for test case, if already sorted by using didswap flag. This can do it in O(n) if arry is already sorted.

```python
from typing import List

def bubbleSort(arr: List[int], n: int):
    for i in range(n-1,-1,-1):
        didswap=1
        for j in range(i):
            if arr[j]>arr[j+1]:
                arr[j],arr[j+1]= arr[j+1],arr[j]
        if didswap==0:
            break
```

Recursive bubble sort:

```python
def bubbleSort(arr: List[int], n: int):
    if n==1:
        return
    count=0
    #simple one pass of bubble sort to move largest element at end
    for i in range(0,n-1):
        if arr[i]>arr[i+1]:
            #swap arr[i] and arr[i+1]
            arr[i],arr[i+1]= arr[i+1],arr[i]
            count+=1
    if count==0:
        return #coz already sorted
    #recursively do bubble sort on n-1 element
    bubbleSort(arr,n-1)
```

Q:INSERTION SORT:

Insert the element into its correct position by shifting elements,

```python
from typing import List

def insertionSort(a: List[int], n: int) -> None:
    for i in range(1,n):
        j=i
        while(j>0):
            if a[j-1]>a[j]:
                a[j-1],a[j]= a[j],a[j-1]
                j-=1
            else:
                break
```

RECURSIVE INSERTION SORT:

How to implement it recursively?

Recursive Insertion Sort has no performance/implementation advantages, but can be a good question to check one's understanding of Insertion Sort and recursion.

If we take a closer look at Insertion Sort algorithm, we keep processed elements sorted and insert new elements one by one in the sorted array.

```python
from typing import List

def insertionSort(a: List[int], n: int) -> None:
    if n==1:
        return
    #sort first (n-1) elements
    insertionSort(a,n-1)
    #insert last element at its correct position
    last=a[n-1]
    j=n-2
    # Move elements of arr[0..i-1], that are greater than key, to one position ahead
        # of their current position
    # the position after j is vaccent where a[j]<last because we shift all the elements to one right
    while(j>=0 and a[j]>last):
        a[j+1]=a[j]
        j=j-1
    a[j+1]= last
```

(DIVIDE & CONQUER ALGORITHMS)

1. QUICK SORT:

```python
#User function Template for python3

class Solution:
    #Function to sort a list using quick sort algorithm.
    def quickSort(self,arr,low,high):
        if low<high:
            pi= self.partition(arr,low,high)   #partition index in pi
            self.quickSort(arr,low,pi-1)
            self.quickSort(arr,pi+1,high)


    def partition(self,arr,low,high):
        #let suppose leftmost element as pivot
        pivot= arr[low]
        i=low    #i pointer finds the index from leftmost which is greater than pivot
        j=high    #j pointer finds the index from rightmost which is less than pivot
        while(i<j):
            while(arr[i]<=pivot and i<high):
                i+=1
            #when arr[i]>pivot we have to swap that index to index which is less than pivot from right
            while(arr[j]>pivot and j>low):
                j-=1

            if i<j:
                #swap i and j
                arr[i], arr[j]= arr[j], arr[i]
        #swap pivot and arr[j]
        arr[low], arr[j] = arr[j], arr[low]
        return j
```

Time Complexity: O(N*logN), where N = size of the array.

Reason: At each step, we divide the whole array, for that logN and n steps are taken for the partition() function, so overall time complexity will be N*logN.

The following recurrence relation can be written for Quick sort :

$F(n) = F(k) + F(n-1-k)$

Here k is the number of elements smaller or equal to the pivot and n-1-k denotes elements greater than the pivot.

There can be 2 cases :

Worst Case – This case occurs when the pivot is the greatest or smallest element of the array. If the partition is done and the last element is the pivot, then the worst case would be either in the increasing order of the array or in the decreasing order of the array.

Recurrence:
$F(n) = F(0)+F(n-1)$ or $F(n) = F(n-1) + F(0)$

Worst Case Time complexity: $O(n^2)$

Best Case – This case occurs when the pivot is the middle element or near to middle element of the array.
Recurrence :
$F(n) = 2F(n/2)+ n$

Time Complexity for the best and average case: O(N*logN)

Space Complexity: O(1) + O(N) auxiliary stack space.

MERGE SORT:

```python
class Solution:
    def merge(self,arr, l, m, r):
        i=l
        j=m+1
        temp=[]
        while(i<=m and j<=r):
            if arr[i]<=arr[j]:
                temp.append(arr[i])
                i+=1
            else:
                temp.append(arr[j])
                j+=1

        while(i<=m):
            temp.append(arr[i])
            i+=1
        while(j<=r):
            temp.append(arr[j])
            j+=1

        for i in range(l,r+1):      #copy back to original array
            arr[i]= temp[i-l]

    def mergeSort(self,arr, l, r):
        if l>=r:
            return
        mid = (l+r)//2
        self.mergeSort(arr,l,mid)
        self.mergeSort(arr,mid+1,r)
        self.merge(arr,l,mid,r)
```

Time complexity: O(nlogn)

T(n)= 2 T(N/2) +N

Reason: At each step, we divide the whole array, for that logn and we assume n steps are taken to get sorted array, so overall time complexity will be nlogn

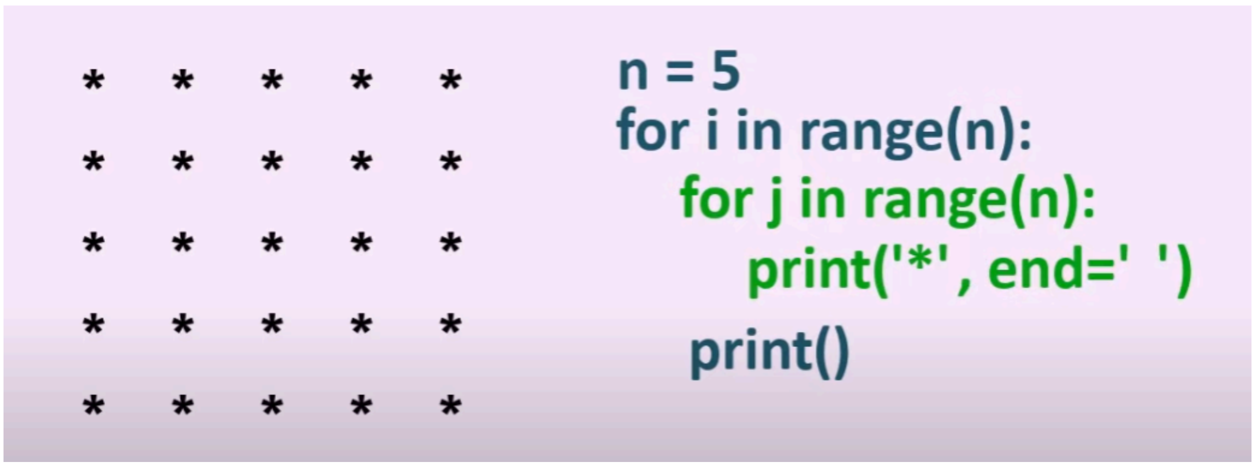
Space complexity: O(n)

Reason: We are using a temporary array to store elements in sorted order.

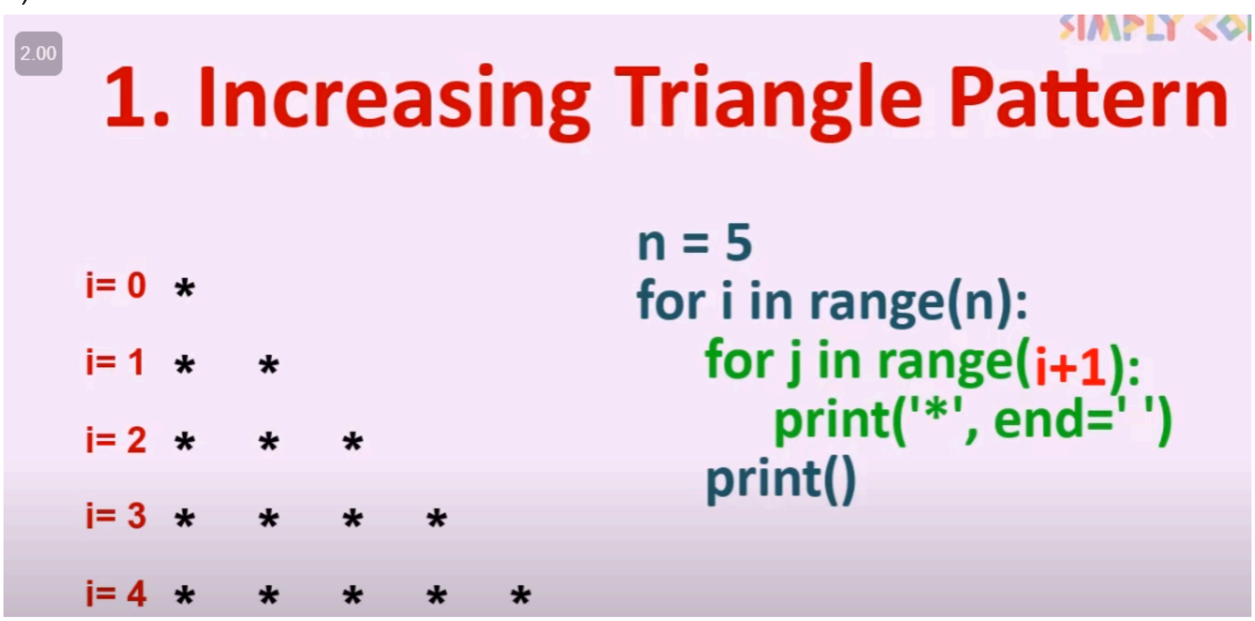
Auxiliary Space Complexity: O(n) , used during merge procedure.

Pattern Programming:

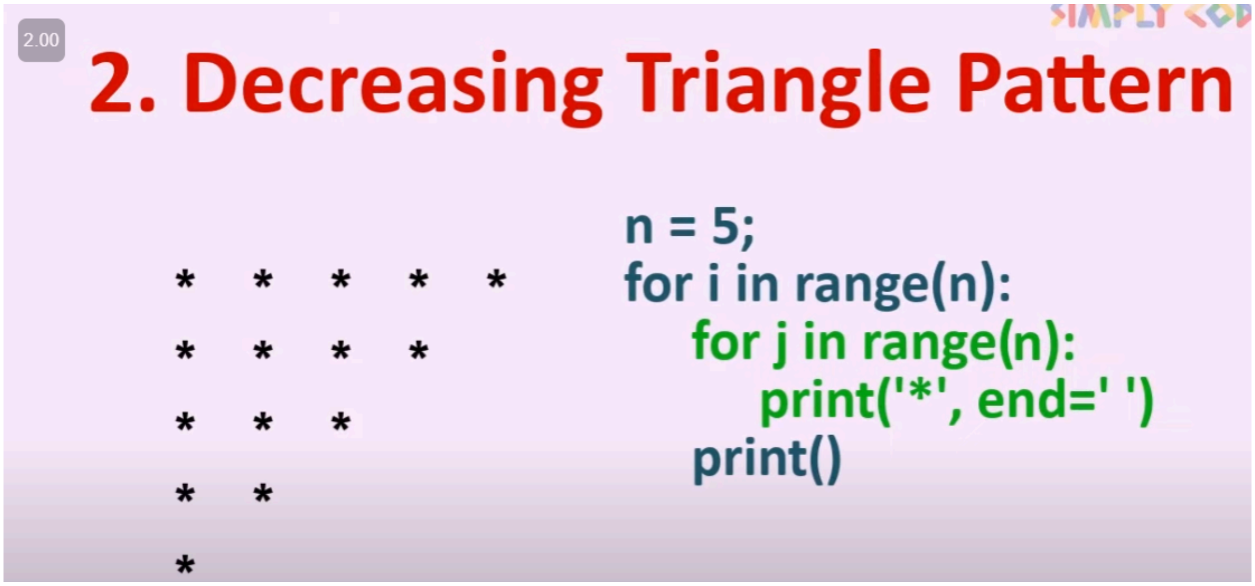
1)Square Pattern



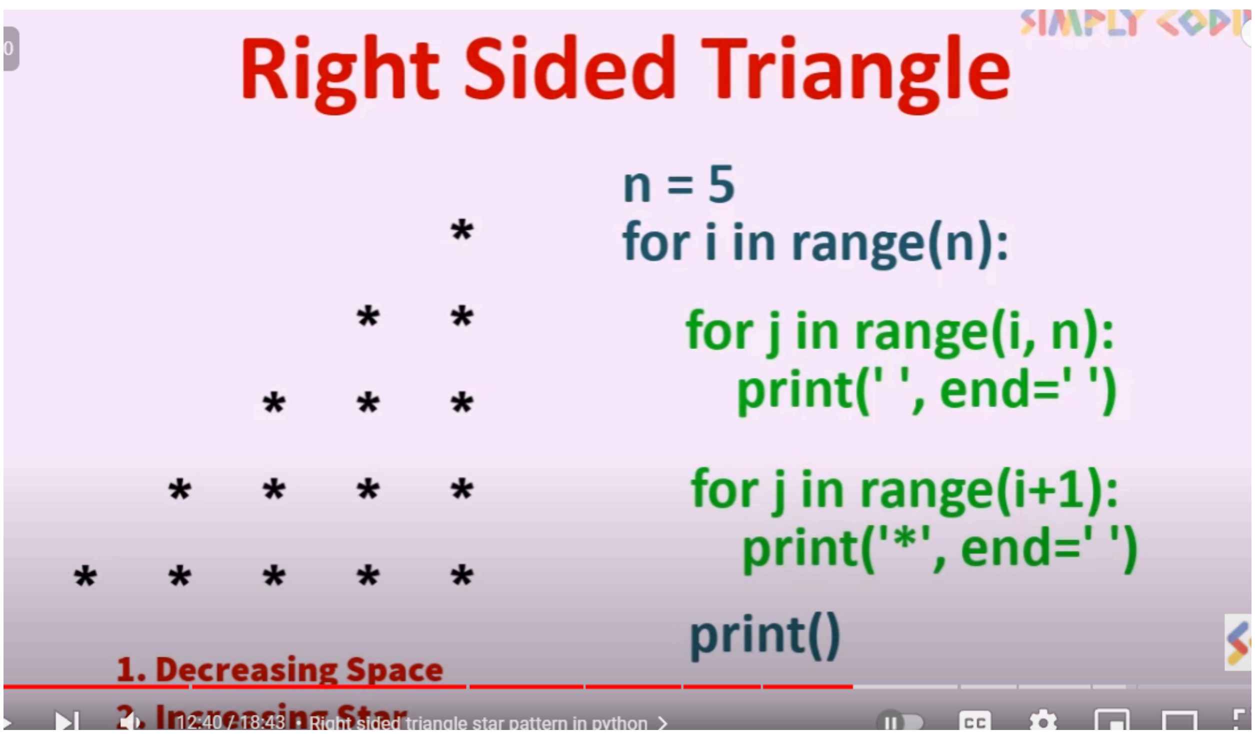
2)



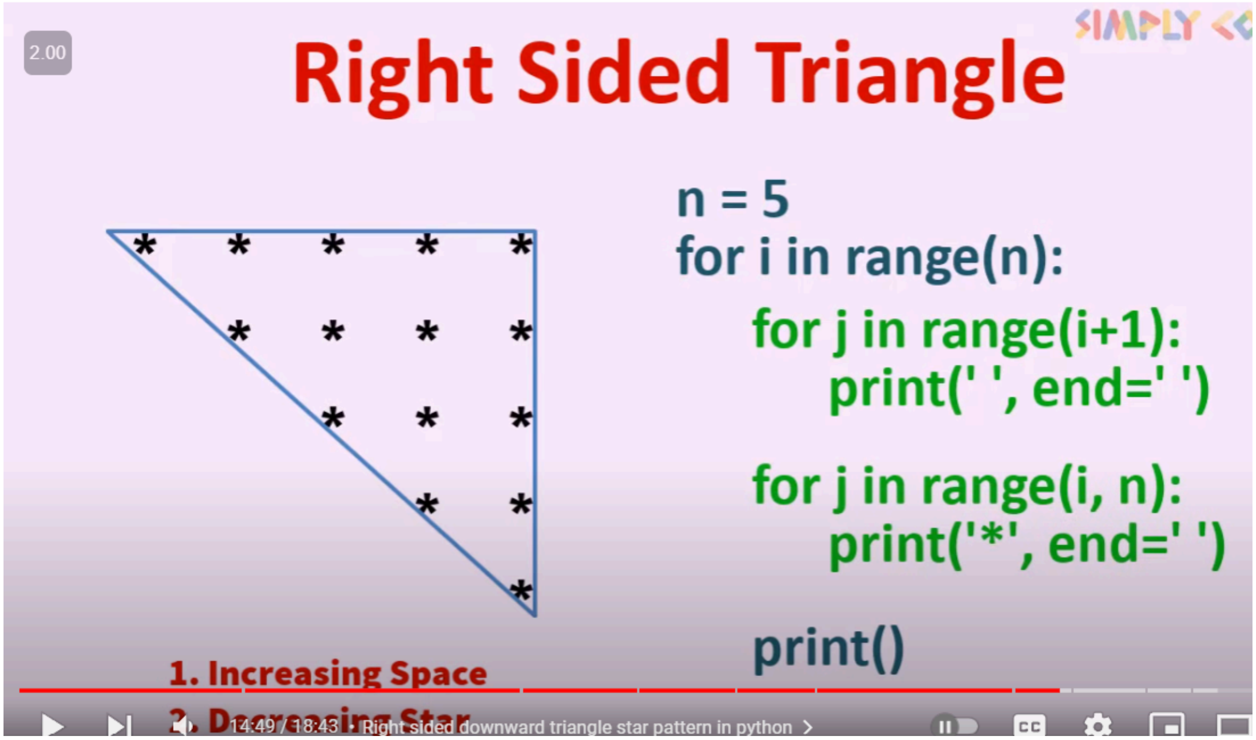
3)


2.00
2. Decreasing Triangle Pattern
* * * * *
* * * *
* * *
* *
*
n = 5;
for i in range(n):
for j in range(n):
print('*', end=' ')
print()

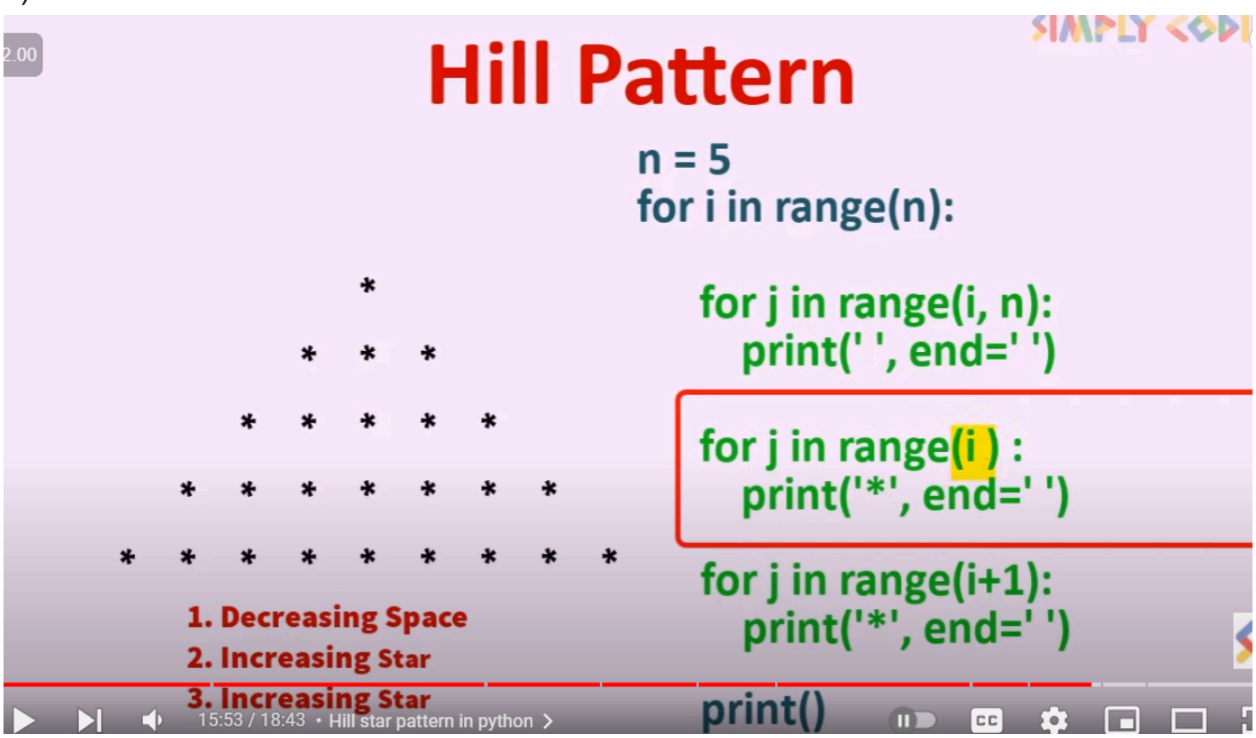
4)


Right Sided Triangle
*
* *
* * *
* * * *
* * * * *
n = 5
for i in range(n):
for j in range(i, n):
print(' ', end=' ')
for j in range(i+1):
print('*', end=' ')
print()
1. Decreasing Space
12:40 / 18:43 • Right sided triangle star pattern in python

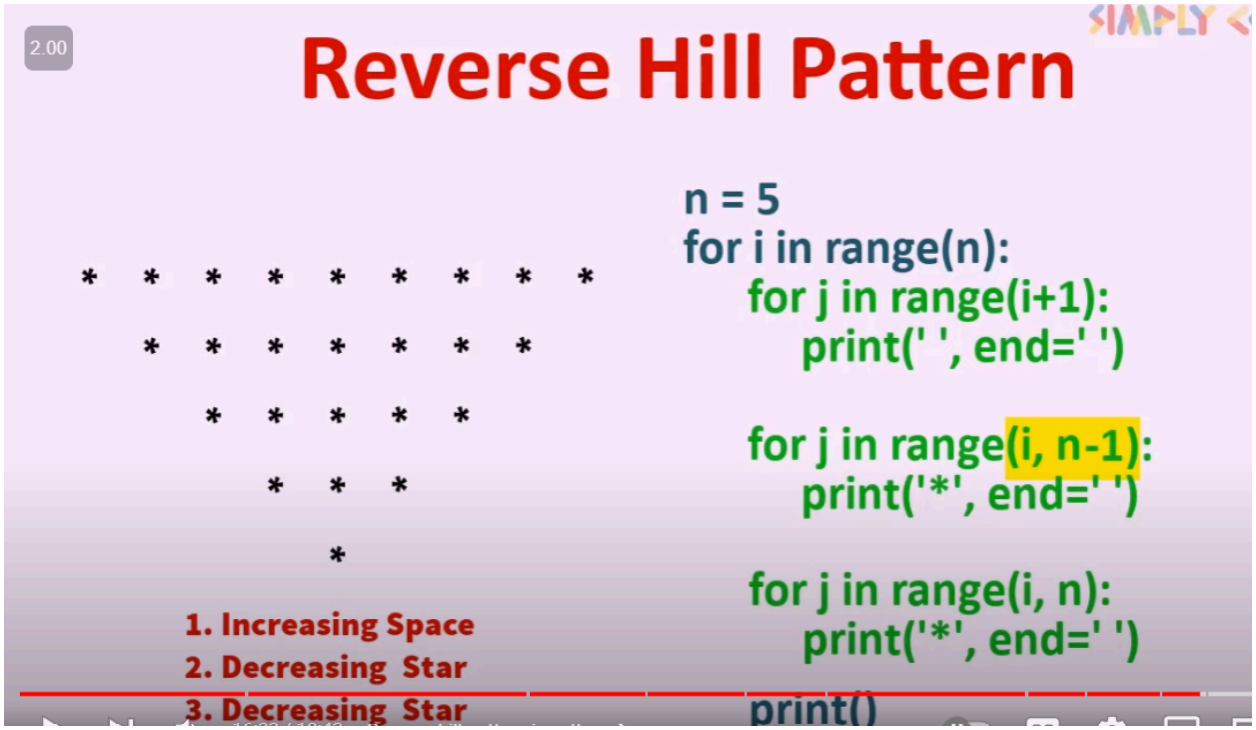
5)


2.00
Right Sided Triangle
n = 5
for i in range(n):
for j in range(i+1):
print(' ', end=' ')
for j in range(i, n):
print('*', end=' ')
print()
1. Increasing Space
14:49 / 18:43 • Right sided downward triangle star pattern in python

6)


2.00
Hill Pattern
n = 5
for i in range(n):
for j in range(i, n):
print(' ', end=' ')
for j in range(i) :
print('*', end=' ')
for j in range(i+1):
print('*', end=' ')
print()
1. Decreasing Space
2. Increasing Star
3. Increasing Star
15:53 / 18:43 • Hill star pattern in python

7)


2.00
Reverse Hill Pattern
* * * * * * * * *
* * * * * * *
* * * * *
* * *
*
1. Increasing Space
2. Decreasing Star
3. Decreasing Star
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):
print('*', end=' ')
print()

8)

Diamond PAttern

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

```
n=5

for i in range(n):
    for j in range(i,n):
        print(' ',end='')
    for j in range(i):
        print('*',end='')
    for j in range(i+1):
        print('*',end='')
    print()

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):
        print('*',end='')

    print()
```