**Data Structure :** A data structure is a particular way of organizing data in a computer so that it can be used effectively.

**Array:** An array is a collection of items stored at the **contiguous**(adjacent/neighboring) memory locations. It stores multiple items of the **same data type** together. Each item of array have unique index.

**String:** Strings are defined as an array of characters. The difference between a character array and string array is the string is terminated with a special character ‘\0’.

**Linked List:** A linked list is a linear data structure, in which the elements are n**ot stored in contiguous memory locations.** The elements in a linked list are linked using pointers. Simply liked list is a node which contains a field and link to the next node in the list.

The advantage of linked list over array is :

1. Dynamic Size
2. Ease of insertion/deletion.

**Singly Linked List**

Each node has a data and pointer to the next node.

**Head → Node 1[Data | Next] → Node 2→Node 3[Data | Next] → Null**

struct node {

int data;

struct node \*next;

}

**Doubly Linked list**

We add a pointer to the previous node in a doubly linked list. Here we can go either way, forward and backward.

**Head → prev, data, next → prev, data, next ⇾ prev, data, next →Null**

**Null ← ← <-**

struct node {

int data;

struct node \*next;

struct node \*prev;

}

**Circular Linked List**

A circular linked list is a variation of a linked list in which the last element is linked to the first element.

**Head → data, next →data, next → data, next---**

**|^---------------------------------------------------|**

**Heaps**

**A Heap is a special Tree-based data structure in which the tree is a complete binary tree**

* Max-Heap: In a Max-Heap, the key present at the root node must be greatest among the keys present at all of its children. The same property must be recursively true for all subtrees in that Binary Tree.
* Min-Heap: In a Min-Heap, the key present at the root node must be minimum among the keys present at all of its children. The same property must be recursively true for all subtrees in that Binary Tree.

**Dynamic Programming**

Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for the same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of sub problems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.

**Stack**

Stack is a liner data structure which follows a particular order in which the operation are performed. The order may be **LIFO or FILO(Stack of plates).** Insertion and deletion happens on same ends.

**Queue**

A queue is a linear structure which follows a particular order in which the operations are performed. The order is **FIFO**. Insertion and deletion happens on different ends.

**Tree/Binary Tree**

A tree data structure is non-linear and hierarchical data arranged in a tree like structure. It consists of a central node, structural nodes, sub nodes(root, branches, and leaves) which are connected by edges. Tree is a non-linear and hierarchical data structure consists of nodes that each node of tree store values. **Root node is level/depth 0.**

**Binary Search Tree**

Binary Search Tree is a node-based binary tree data structure which has the following properties:

* The left subtree of a node contains only nodes with **keys lesser than the node’s key**.
* The right subtree of a node contains only nodes with **keys greater than the node’s key.**
* The left and right subtree each must also be a binary search tree.
* There must be no duplicate nodes.

**Graph**

A graph data structure is a collection of nodes that have data and are connected to other nodes through edges. Like in face book everything is a **node(anything that has a data**)user, photo, event, group, page, story, video. These nodes are connected through edges.

A graph is composed of set of vertices(node/V) and set of edges(E).

Types of graphs: Null graph, Trivial graph, undirected graph, directed graph, connected graph etc.

**Searching Algorithms**

Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored.

**Linear Searching :**

* Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
* If x matches with an element, return the index.
* If x doesn’t match with any of the elements, return -1.

**Binary Searching :**

Binary Search is a searching algorithm for finding an element's position in a **sorted array.**

* Compare x with the middle element.
* If x matches with the middle element, we return the mid-index.
* Else, If x is greater than the mid-element, then x can only lie in the right half subarray after the mid-element. So we recur for the right half.
* Else (x is smaller) recur for the left half.

**Sorting Algorithms**

A sorting algorithm is used to arrange a given array or list elements according to a comparison operator on the elements.

**==========================CODING====================================**

**Selection sort**

**Select 1st element as min number and loop through each element in the array, for i in range(len(array)) and for i in range(i+1, len(array)) #two sub arrays.**

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from the unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1) The subarray which is already sorted.

2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

# select sort

# def selectionSort(arr):

# for i in range(len(arr)):

# m=i

# for j in range(i+1, len(arr)):

# if arr[i] > arr[j]:

# m=j

# arr[m], arr[i]=arr[i],arr[m]

# arr=[4,5,7,2,9,10]

# selectionSort(arr)

# print ("Sorted array is:")

# for i in range(len(arr)):

# print (arr[i])

================================================================

**Bubble sort**

**Bubble sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order**

#bubble sort

# def bubbleSort(arr):

# n=len(arr)

# for i in range(n-1):

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

# if arr[j] > arr[j+1]:

# arr[j] ,arr[j+1] = arr[j+1], arr[j]

# arr = [4,1,9,50,34]

# bubbleSort(arr)

# print ("Sorted array is:")

# for i in range(len(arr)):

# print (arr[i])

===============================================

**Remove Duplicate character**

string="geeksforgeeks"

p=""

for char in string:

if char not in p:

p=p+char

print(p)

================================================

**Given a string, find a substring based on the following conditions:**

**The substring must be the longest one of all the possible substring in the given string.**

**There must not be any repeating characters in the substring.**

**If there is more than one substring satisfying the above two conditions, then print the substring which occurs first.**

**If there is no substring satisfying all the aforementioned conditions then print -1**

#consider point l, r

#'afgcabdfghf'

#checking in range[l to r] there are any duplicates?

#if you find repetition in range, then remove from l pointer then move pointer one step ahead.

#length =r-l+1

#ord(str) --return asci value, then compare the values eith initialised visited array

def longestsub(str):

n=len(str)

res = 0

for i in range(n):

visited =[0] \*256

for j in range(i,n):

if (visited[ord(str[j])] == True):

break

else:

res = max(res, j-i+1)

visited[ord(str[j])] = True

#remove the first char from window

visited[ord(str[i])]=False

return res

str = "geeksforgeeks"

len = longestsub(str)

print("The length of the longest "

"non-repeating character substring is ", len)

=====================================================================

**Palindrome**

def palindrome(a):

# finding the mid, start

# and last index of the string

mid = (len(a)-1)//2 #you can remove the -1 or you add <= sign in line 21

start = 0 #so that you can compare the middle elements also.

last = len(a)-1

flag = 0

# A loop till the mid of the

# string

while(start <= mid):

# comparing letters from right

# from the letters from left

if (a[start]== a[last]):

start += 1

last -= 1

else:

flag = 1

break;

# Checking the flag variable to

# check if the string is palindrome

# or not

if flag == 0:

print("The entered string is palindrome")

else:

print("The entered string is not palindrome")

==============================================================

**Symmetric**

def symmetry(a):

n = len(a)

flag = 0

# Check if the string's length

# is odd or even

if n%2:

mid = n//2 +1

else:

mid = n//2

start1 = 0

start2 = mid

while(start1 < mid and start2 < n):

if (a[start1]== a[start2]):

start1 = start1 + 1

start2 = start2 + 1

else:

flag = 1

break

# Checking the flag variable to

# check if the string is symmetrical

# or not

if flag == 0:

print("The entered string is symmetrical")

else:

print("The entered string is not symmetrical")

# Driver code

string = 'amaama'

palindrome(string)

symmetry(string)

==================================================================

**What does the if \_\_name\_\_ == “\_\_main\_\_”: do?**

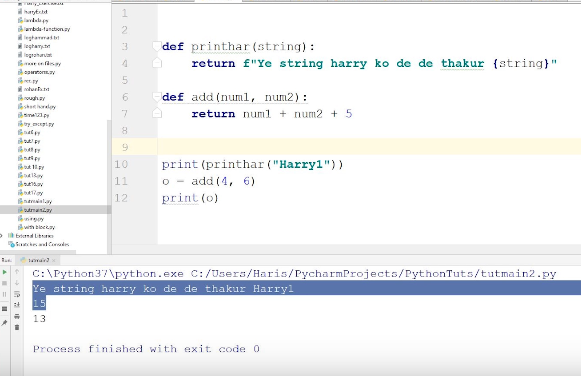
When a Python interpreter reads a Python file, it first sets a few special variables. Then it executes the code from the file.

One of those variables is called **\_\_name\_\_**

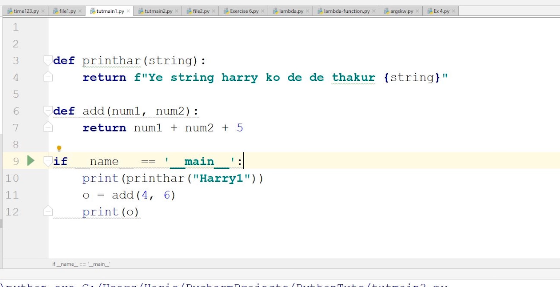
when the interpreter runs a module, the \_\_name\_\_ variable will be set as \_\_main\_\_ if the module that is being **run is the main program**.

But if the code is importing the module from **another module**, then the \_\_name\_\_ variable will be set to that **module’s name.**

If you don’t have a main function (if name==main) then It will import everything. All the print statement and all.



If you import the above module./file it will execute everything. But if you add main function..



**Then you import this file/module it will not execute the code inside main. The function logics of the above function gets executed.**

**The content it only gets executed inside the main file only( above pic) If it is imported it don’t get executed.**

**This is how you can define a function and call it.**

#!/bin/python3

import math

import os

import random

import re

import sys

def actions(n):

if(n%2!=0):

print("Weird")

elif n % 2 == 0 and 2 <= n <= 5:

print("Not Weird")

elif n % 2 == 0 and 6 <= n <= 20:

print("Weird")

else:

print("Not Weird")

if \_\_name\_\_ == '\_\_main\_\_':

n = int(input().strip())

actions(n)

**===================================================================**

**Always think about flags when you code….**

**===================================================================**

**List Comprehension.**

**A Python list comprehension consists of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element in the** [**Python list**](https://www.geeksforgeeks.org/python-list/)**.**

**newList = [ expression(element) for element in oldList if condition ]**

### **Matrix using List comprehension**

| **matrix = [[j for j in range(3)] for i in range(3)]**    **print(matrix)** |
| --- |

**Output:**

**[[0, 1, 2], [0, 1, 2], [0, 1, 2]]**

**Brackets are imp([ ])**

Let's learn about list comprehensions! You are given three integers and representing the dimensions of a cuboid along with an integer . Print a list of all possible coordinates given by on a 3D grid where the sum of is not equal to . Here, . Please use list comprehensions rather than multiple loops, as a learning exercise.

def listcom(x,y,z,n):

ans = [[i, j, k] for i in range(x + 1) for j in range(y + 1) for k in range(z + 1) if i + j + k != n]

print(ans)

if \_\_name\_\_ == '\_\_main\_\_':

x = int(input())

y = int(input())

z = int(input())

n = int(input())

listcom(x,y,z,n)

**=====================================================================**

## **Python Code:- Find first Non Repeating Characters**

**# Python implementation**

**# Python implementation**

**from collections import Counter**

**def firstnonrep(string):**

**freq=Counter(string)**

**for i in string:**

**if(freq[i]==1):**

**print(i)**

**break**

**if \_\_name\_\_ =="\_\_main\_\_":**

**print("Output:")**

**f=firstnonrep('geeksforgeeks')**

**print(f, end=" ")**

**------------------------------------------------------------------------------------------------------------------------**

## **Python Code:- Find Non Repeating Characters**

**#take user input**

**String = "prepinsta"**

**for i in String:**

**#initialize a count variable**

**count = 0**

**for j in String:**

**#check for repeated characters**

**if i == j:**

**count+=1**

**#if character is found more than 1 time**

**#brerak the loop**

**if count > 1:**

**break**

**#print for nonrepeating characters**

**if count == 1:**

**print(i)**

**=======================================**

Factorial of number

def factorial(n):

if n<0:

print("Opartion not allowed")

if n==0:

return 1

if n==1:

return 1

else:

return n\*factorial(n-1)

if \_\_name\_\_ =="\_\_main\_\_":

print("Output:")

f=factorial(4)

print(f)

=====================================================================

**# Python code**

**#reverse the string(no function and all)**

**str=" abc def hij klm"**

**s=str.split()[::-1]**

**l=[]**

**for i in s:**

**l.append(i)**

**print(" ".join(l))**

**==================================================**

# **Generators**

**Generator-Function :** A generator-function is defined like a normal function, but whenever it needs to generate a value, it does so with the [yield keyword](https://www.geeksforgeeks.org/use-yield-keyword-instead-return-keyword-python/) rather than return. If the body of a def contains yield, the function automatically becomes a generator function.

| # A generator function that yields 1 for first time,  # 2 second time and 3 third time  def simpleGeneratorFun():  yield 1  yield 2  yield 3    # Driver code to check above generator function  for value in simpleGeneratorFun():  print(value) |
| --- |

**Generator-Object : Generator functions return a generator object. Generator objects are used either by calling the next method on the generator object or using the generator object in a “for in” loop (as shown in the above program).**

| **# A Python program to demonstrate use of**  **# generator object with next()**    **# A generator function**  **def simpleGeneratorFun():**  **yield 1**  **yield 2**  **yield 3**    **# x is a generator object**  **x = simpleGeneratorFun()**    **# Iterating over the generator object using next**  **print(x.next()) # In Python 3, \_\_next\_\_()**  **print(x.next())**  **print(x.next())** |
| --- |

**So a generator function returns an generator object that is iterable, i.e., can be used as an** [**Iterators**](https://www.geeksforgeeks.org/iterators-in-python/) **.**

**===============================================**