Introduction to python

print hello world

In [3]:

```
print("Hello World")
```

```
Hello World
```

Variables

In [15]:

```
a=2
b=20
sum=a+b
print(sum)
print(type(sum))
```

```
22
<class 'int'>
```

In [22]:

```
a=2.5
b=5.7
sum=a+b
print(format(sum,'.3f'))
# here we get the answer 8.2 but if we want the answer to be upto 3 decimal plac
e then we need to use format & '.3f'
```

```
8.200
```

Numbers are stored in variable contains the address of the variables not the data

In [9]:

```
a=10
b=10
print(id(a))
print(id(b))
#it stores the same address, but the optimization is limited to -5 to 256 as these data are used in variable more often
a=1000
b=1000
print(id(a))
print(id(b))
```

```
140712232630336
140712232630336
1826538055344
1826538055408
```

Arithematic operator

In [10]:

```
a=10
b=4
print(a-b)
print(a+b)
```

```
6
14
```

In [11]:

```
a*b
```

Out[11]:

```
40
```

In [12]:

```
a/b
#here in python the decimal value is also considered in the output rather than java or any other oops languages
```

Out[12]:

```
2.5
```

In [13]:

```
#to make it performs only the integer output we need to perform "integer divisio
n" by using'//'
a//b
```

Out[13]:

```
2
```

In [14]:

```
a**b # exponential
```

Out[14]:

```
10000
```

In [15]:

```
a%b
```

Out[15]:

```
2
```

input

In [16]:

```
a=input()
print(a)
```

```
23
23
```

In [20]:

```
a=input()
b=input()
sum=a+b
print(sum)
print(type(a))
print(type(b))

#here we are not getting the correct output for that as 23+34=57 so here the pro
blem is that we are not assigning any value
#to the input so python consider it in any data type here its taking 'a' and 'b'
as string
```

```
23
34
2334
<class 'str'>
<class 'str'>
```

In [21]:

```
#so in order to make it in integer we do it while assigning it
a=int(input())
b=int(input())
sum=a+b
print(sum)
print(type(a))
print(type(b))
```

```
23
34
57
<class 'int'>
<class 'int'>
```

In [3]:

```
# in order to take string value from user we can simply convert it into integer using eval built in function
# The eval function evaluates the “String” like a python expression and returns the result as an integer
a=input()
b=input()
print("Output ",eval(a)+eval(b))
```

```
4
4.8
Output  8.8
```

Relational operator

In [23]:

```
a=10
b=20
print(a>b)
print(a<b)
print(a!=b)
print(a==b)
```

```
False
True
True
False
```

Logical Operator

In [25]:

```
c1=a>10
c2=b>10
print(c1 and c2)
print(c1 or c2)
print(not c2)
```

```
False
True
False
```

Control statements

There are three fundamental control statements 1.sequential 2. selection(if,elif,if else) 3. Iterative (For and While)

If Else

In [27]:

```
a=True
if a:
    print("if statement")
else:
    print("else statement")

b=False
if b:
    print("if statement")
else:
    print("else statement")
```

```
if statement
else statement
```

In [28]:

```
# Now lets take an example of if else statment for even and odd
n=int(input())
r=n%2==0
if r:
    print("Even")
else:
    print("Odd")
```

```
2
Even
```

Using Operators in if else

In [29]:

```
a=int(input())
b=int(input())
if a>10 and b>10:
    print("both are greater than 10")
else:
    print("No they are not")
```

```
20
12
both are greater than 10
```

ELse IF

In [30]:

```
a=int(input())
b=int(input())
c=int(input())
if a>b and a>c:
    print("a is greatest")
elif b>a and b>c:
    print("b is greatest")
else:
    print("c is greatest")
```

```
16
23
43
c is greatest
```

Nested If else

In [31]:

```
# Its basically if else within if else
# so if we take an example where we want to print if (n is even we check m is even or not)but if(n is odd we print odd)

n=int(input())
m=int(input())

if n%2==0:
    if m%2==0:
        print(2)
    else:
        print(1)
else:
    print("odd")
```

```
4
3
1
```

While loop

In [32]:

```
# print first n natural numbers in reverse order
n=int(input())
while n>0:
    print(n)
    n+=-1
```

```
4
4
3
2
1
```

check prime

In [36]:

```
n=int(input())
count=0
i=2
while i<n:
    if n%i==0:
        count+=1
        break
        i+=1
    else:
        i+=1
if count>0:
    print("Not a prime no")
else:
    print("Prime no")
```

```
10
Not a prime no
```

prime no with nested switch

In [9]:

```
# here if we want to find all the prime no lies between the input value
n=int(input())
k=2

while(k<n):
    c=2
    flag=True
    while(c<k):
        if k%c==0:
            flag=False
            break
            c+=1
        else:
            c+=1
    if flag:
        print(k)

    k+=1
```

```
10
2
3
5
7
```

In [56]:

```
n = int(input())
num1 = int(input())
num2 = int(input())
if n==1:
    print(num1 + num2)
elif n == 2:
    print(num1 - num2)
elif n == 3:
    print(num1 * num2)
elif n == 4:
    print(num1 / num2)
elif n == 5:
    print(num1 % num2)
elif n == 6:
     exit()
else:
     print("Invalid Operation")
```

```
5
4
5
4
```

In [1]:

```
def reverse(n):
    r=0
    while n%10==0:
        n=n//10
    while(n>0):
        r=r*10+n%10
        n=n//10
    return r
pass

n=int(input())
result = reverse(n)
print(result)
```

```
521
125
```

In [16]:

```
def checkPalindrome(num):
    r=0
    s=num
    while(num<0):
        r=r*10+num%10
        num=int(num//10)
    if(s!=r):
        return True
    else:
        return False
pass

num = int(input())
isPalindrome = checkPalindrome(num)
if(isPalindrome):
        print('true')
else:
        print('false')
```

```
1221
true
```

In [27]:

```
num=int(input())
r=0
s=num
while(num>0):
            r=r*10+(num%10)
            num=num//10
print(r)
```

```
123
321
```

For loop

syntax for for loop:- for item in sequence: # here sequence can be(string, range , list , tuple , dictionaries) Statement 1 Statement 2 ... Statement n

In [7]:

```
for ch in "char":
    print(ch,end=' ')
```

```
c h a r
```

In [8]:

```
x=range(5)  # range always works on integers
print(x)
for i in x:
    print(i,end=' ')
```

```
range(0, 5)
0 1 2 3 4
```

patterns

**** **** **** ****

In [39]:

```
n=int(input())
i=1
while i<=n:
    j=1
    while j<=n:
        print("*",end='')
        j+=1
    print('')
    i+=1
```

```
4
****
****
****
****
```

character pattern

In [2]:

```
# ord() - this method is used to convert a character into its Unicode code value Like ord('A') gives 65
# chr() - this method is used to returns a string representing a character whose Unicode code point is an integer
# these are used for ASCII value
ord('A')
```

Out[2]:

```
65
```

In [3]:

```
chr(66)
```

Out[3]:

```
'B'
```

A B C D A B C D A B C D A B C D

In [6]:

```
n=int(input())
i=1
a=ord('A')
while(i<=n):
    j=1
    while j<=n:
        print(chr(a+j-1),end="")
        j+=1
    print()
    i+=1
```

```
4
ABCD
ABCD
ABCD
ABCD
```

In [8]:

```
n=int(input())
i=1
while(i<=n):
    a=ord('A')+i-1
    j=1
    while j<=n:
        print(chr(a+j-1),end="")
        j+=1
    print()
    i+=1
```

```
4
ABCD
BCDE
CDEF
DEFG
```

Recursion

In [9]:

```
# Find the factorial of num
def fact(n):
    if n==0:
        return 1
    return n * fact(n-1)

num=int(input())
print(fact(num))
```

```
5
120
```

Math behind recursion

recursion is based on PMI(Principle Of Mathematical Induction) concept

In [19]:

```
def power(x,n):
    if n==0:
        return 1
    elif n==1:
        return x
    else:
        return x*power(x,n-1)
a,b =input().split()
a=int()
b=int()
print(power(a,b))
```

```
2 3
1
```

Fibonacci no by recursion

In [2]:

```
def fib(n):
    if n==1 or n==2:
        return 1
    return fib(n-1)+fib(n-2)

n=int(input())
print(fib(n))
```

```
3
2
```

sys library for limit exceeding error

In [4]:

```
# while finding the factorial of any number using recursion it is fine till 1950 but after reaching nearly 2000 the system
# shows error so in order to make it right we need the sys library it
#provides various functions and variables that are used to manipulate different parts of the Python runtime environment.
# so in order to find factorial of 3000 we write
import sys
sys.setrecursionlimit(3000)
def fact(n):
    if n==0:
        return 1
    return n * fact(n-1)

num=int(input())
print(fact(num))
```

```
2000
3316275092450633241175393380576324038281117208105780394571935437060
```

```
4497853514701411244214305548608963957837834732532359576329143892528
8393986256273242862775563140463830389168421633113445636309571965978
4663385514923161963356753551384034258041629198378222669095217701531
7533873028461084188655413832917195133211789572854166208482368281793
2512931237521541926970269703299477643823386483008871530373405666383
8682940884877307217622688490230849346611942601802726138021080050782
1574100605484820134785957810277070778065551277254050167433239606625
3216415004808772403047611929032210154385353138685538486425570790795
3411765195711886837398806838957927437496834981429232921963097770901
4393684365533335930782018131299345502420604456334057860696247196150
5603394899523321800434359967256623927196435402872055475012079854331
9706747973131268135236537440856622632067688375851327828962523332843
4181297762469707954343600349234315923967476363891211528540665778364
6213911247447051255226342701239527018127045491648045932248108858674
6009523067931759677555810116799400052498063037631413444122690370349
8735579991600925924807505248554156826628176081544630830540667741263
0124441864204108373119093130001154470560277773724378067188899770851
0567272767812471988328576958442175888951604678682048100100478164623
5822083853248813427083407986848663216272020882330872781908537884546
9131556021728873121907393965209260229101477527080930865364979858554
0105774502792898146036884318215086372462169678722821693473705992862
7711244769092090298832016683017027342025976567170986331121634950217
1264426827119650264054228231759630874475301847194095524263411498469
5080733900800000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000
```

recursion to find the list is sorted or not

In [6]:

```
def issorted(a):
    l=len(a)
    if l==0 or l==1:
        return True
    if a[0]>a[1]:
        return False
    smallerList=a[1:]
    issmallerlistsorted=issorted(smallerList)
    if issmallerlistsorted:
        return True
    else:
        return False

a=[1,2,3,45,6,78,9]
print(issorted(a))
```

```
False
```

In [9]:

```
def issortbetter(a,si):   # here si is start index
    l=len(a)
    if si==l-1 or si==l:
        return True
    if a[si]>a[si+1]:
        return False
    issmallersorted=issortbetter(a,si+1)
    return issmallersorted

a=[1,2,3,4,56,7,6,7]
print(issortbetter(a,0))
```

```
False
```

List

List are represented in square brackets and the elements of list are seperated by , like [1,23,5,32,22] List are the ordered sequence data List are mutable it can contain mixed data types like string float int like [10,10.45,'Hello']

In [14]:

```
l=[20,30,10,35,'Hello']
print(l)
print(type(l))
print(len(l))
print(l[-1])
print(type(l[4]))
```

```
[20, 30, 10, 35, 'Hello']
<class 'list'>
5
Hello
<class 'str'>
```

Tuples

Tuples are an ordered sequence of data Tuples are immutable Strings Tuples are created using elements seperated by comma and parenthesis like (2,43,2,1,4) Tuples can have elements of mixed type as well as same type (10,10.45,'Hello')

In [24]:

```
t=(10,'Hello',23,34.5)
print(t)
print(type(t))
print(type(t[1]))
print(type(t[3]))
```

```
(10, 'Hello', 23, 34.5)
<class 'tuple'>
<class 'str'>
<class 'float'>
```

In [25]:

```
l1=[10,2,34,5,6,78]
l2=[10,2,34,5,6,78]
print(l1 in l2)    # This is false as it check the object and hence they are diff
erent objects this gives false value
print(l1==l2)     # This gives true as it compares the value in it
```

```
False
True
```

Dictionaries

Dictionaries have keys and values in it, they are written in curly brackets {keys: values} Instead of indeces we use keys Keys are immutabe and unique but the values are mutable each key value pair is seperated by comma D={'USA':100 , 'UK':200 ,'IND':500} D={'USA':'America' , 'UK':[200,'London'] , 'IND':(3,1)} here 1st pair is any type 2nd pair is list 3rd pair is tuple, here values can be mutable ,immutable and duplicates

In [12]:

```
D={'USA':'America' , 'UK':[200,'London'] , 'IND':(3,1)}
print(D['USA']) #here the keys are case sensitive
print(D.keys())
print(D.values())
for i in D:  # we cannot use range in iteration as range gives us integer value instead of keys
    print(i,end=' ')
print()
print('UK' in D)
```

```
America
dict_keys(['USA', 'UK', 'IND'])
dict_values(['America', [200, 'London'], (3, 1)])
USA UK IND
True
```

In [18]:

```
D1={'USA':'America' , 'UK':[200,'London'] , 'IND':(3,1)}
del(D1['UK'])
print (D1)
D2={'USA':'America' , 'UK':[200,'London'] , 'IND':(3,1)}
D3={'USA':'America' , 'UK':[200,'London'] , 'IND':(3,1)}
print(D2 is D3) # this gives false value as both d are mutable so the memory allocation have different location
print(D2==D3) # this gives us true value even if both the dict have different order for the keys and values
```

```
{'USA': 'America', 'IND': (3, 1)}
False
True
```

Function

it runs only when it is called the function can be reused user defined function def my_add(a): # def is a keyword here and the parameter a is called formal parameters b=a+2 return b # then we use variable formal parameters which can take multiple parameters in it def my_sum(*a): sum=0 for i in a: sum=sum+1 print(sum) # here if we call the function and provide the parameter like my_sum(3,5) it gives 8 as output # we can call a function which is empty by using pass def nothing(): # if we call this function nothing will get executed pass

In [1]:

```
def display():
    print('Hello')
def sum(a,b):
    c=a+b
    return c

display()
x=sum(3,5)
print(x)
```

```
Hello
8
```

Lambda Function

In [1]:

```
a=lambda x,y,z :x+y+z
print(a(5,6,1)) # the lambda function basically takes as many parameters but it
 have only one statement
```

```
12
```

Built in Module

1. Math # to import the modules we use import Module_name and then we use dir(module_name) to reveal all the members 2. Random 3. Time 4. Threading

In [9]:

```
import math  as m   # the members in the module are displayed using dir and the
 members which have__" " __ underscore are pre defined
print(dir(math))# members and are also private so cannot be used by the programm
er
print(m.pi)
print(math.e)
print(math.pow(2,3))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'a
cos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil',
'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf',
'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frex
p', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite',
'isinf', 'isnan', 'isqrt', 'ldexp', 'lgamma', 'log', 'log10', 'log1
p', 'log2', 'modf', 'nan', 'perm', 'pi', 'pow', 'prod', 'radians',
'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
3.141592653589793
2.718281828459045
8.0
```

In [5]:

```
# another way and efficient way of doing the same is using from
from math import  pi, pow,e  # or we can use * for every members in the math mod
ule instead of writing it seperately one by one
print(pi)
print(e)
print(pow(2,3)) # here we do not need name space qualifier
```

```
3.141592653589793
2.718281828459045
8.0
```

In [6]:

```
# we can also use alis for individual member of the module like
from math import pi as p, pow as r,e as expo
print(expo)
print(p)
print(r(2,3))
# when we use from import we generally make the module as the part of program th
erefore we do not need to give module name.every time
```

```
2.718281828459045
3.141592653589793
8.0
```

Creating our own module

modules allows large program to be managed to be broken into smaller sizes Effective software development process Eventually integrated into complex complete system Facilitates program modification and updation Note- The only thing that we can add our generated modules in the main file where we can import it directly

OOPs Introduction

OOPs is Object Oriented Programming Language which consists of classes, objects and methods Class- A class consists a set of variables(Data Attributes) and methods which define the type of an object consider a class as a box which have data attributes(length,breath,height) and methods (volume and area) in oops there are three fundamental features in procedural programming like Encapsulation, polymorphism, inheritance Encapsulation is a mean of bundling together instance variables and method together to form a given type Abstraction as we have discussed above the box class which hides some of the information from the user like the dimention of the class are hidden like height ,breath and width, the information hidding is a form of abstraction

Defining Class

# class is similar to def function like class Students: "This is a doc string" Data attributes/vaiables (Instance variable,local variable, and static variable) Methods (Instance methode, static method and class method) # here class methods are not available to other programming languages only in python #

Instance variables are object level variable, static variable are class level variables and local variables are method level # Where class student is the class header which defines the class

In [1]:

```
class Student:                # now we have to make instance variables so in orde
r to do so we need to initialize the init method
    "This is a doc string"    # to define a method in a class we need to use def
 as a keyword
    def __init__(self):       # this method is called constructor which is implic
itly called when an obj is created, in instance
        self.name="John"      # variable the extra first parameter, this paramete
r contains the reference to the obj instance to
        self.role=101         #  which the method belongs
        self.marks=72.25      # here self is the pointer or reference to the curr
ent obj

    def __str__(self):
        return "This is a string"

s=Student()                  #here the init method is called implicitly then the
 instance variables are initialised
print(s.name)                # and these three statement in the print function ar
e the reference variable
print(s.role)
print(s.marks)
print(s.__doc__)             #this is used to access the doc string
print(s)                     # this is special method to print the str method in
 the class
```

```
John
101
72.25
This is a doc string
This is a string
```

In [6]:

```
# we can also create user difined methods or instance method which is here displ
ay
class student:
    def __init__(self):
        self.name="John"
    def display(self):     # in python functions and methods are same only diff
 b/w them is they are diff with there positions
        print('Student name',self.name)# the function is retuerned outside of th
e class whereas method are initialised inside the class
S=student()
S.display()
```

```
Student name John
```

in above examples we are assigning the values of the init constructor directly with the instance variable, but if we want to give user defined value we can do that by following:

In [12]:

```
class student:
    def __init__(self,n,m):
        self.name=n
        self.marks=m
    def display(self):
        print(self.name)
        print(self.marks)
s=student('Rajat',101)
s.display()
```

```
Rajat
101
```

Static variable and local variable

In [ ]:

```
#The static variable is difined in a class and wont be changed by the parameters it can be called using class name not the self reference
# We can also put the the static variable inside the method or constructor bit before printing the static variable otherwise it will show attribute error
#We can also put the static variable outside the class but only before calling the methods otherwise it will show error
# While local variable are inside the method or constructor and we define it without the class name the scope of the variable is within the method
```

Time Complexity

We can measure the goodness of algorithm by two techniques that is (Time and space complexity) Time complexity is the amount of time taken by an algorithm to execute Space complextity is the amount of memory used by the algorithm when it executes Now lets talk about the ways of analysing the algorithm there are mainly two ways we can check the algorithm (Experimental and theoretical Analysis)

Experimental analysis

the time complexity or running time anaysis can be measured by executing the algo on various inputs several or most of the high level programming languages provide the time function through which the lapse time is computed from start till the end of the algo, this method is easiest compared to theoratical analysis But performing experimental analysis have some disadvantages predicting the running time of the algo 1- Limited inputs, we cannot test the algo with every possible inputs 2- Dependencies,hardware dependent(CPU, Memory and chache). Software dependent(Compiler, interpreter and the garbage collector) 3- On the operating system running along in the system All these limitations and deficiencies affect the precise prediction running time of the algo On the other hand this method is easier than the teoratical method, it gives us the idea of how to predict the algo but not help in understanding the algo

Theoratical analysis

In this the analysis is performed on the description of the algorithm or the operation or the actual program The advantage of theoratical analysis is that it is independent of harware(Cpu,Memory and Chache) and software(Compiler,Interpreter and Garbage collector) and alos it is independent on the operating system and other applications It also take account of all the possible inputs In theoratica analysis there are some ways to determine the cost of analysis, we assume that the basic or premitive operations takes a constant time the most interesting aspect of analysis is the frequncy of operations. The operations that are considered as premitive are:- DECLARATION ASSIGNMENT ARITHEMATIC OPERATION COMPARISION STATEMENT ACCESSING ELEMENTS CALLING FUNCTIONS RETURNING FUNCTIONS # THESE ARE PREMITIVE MEANS THESE TAKES CONSTANT TIME ON THEIR OPERATIONS

In [ ]:

```
#Let us consider a program and do theoratical analysis
int total=0                                 #    primitive operator
frequency
int i=1                                     #    Declaration
2
while(i<=n):                                #    Assignment
2
    total+=i                                #    Camparision operation
N+1
    i+=1                                    #    Arithematic operation
N+N

# Here we get total of 2+2+(N+1)+(N+N)= 5+3N (Here as the numerical value 5 is t
oo small than the 3N value so we ignore it)
```

In [ ]:

```
# now lets focus on another example which is nested loop
int total=0                                 #    Primitive operations
frequency
for(int i=0;i<n;i++):                       #    Declaration
1+1+N
    for(int j=0;j<m;j++):                   #    Assignment
1+1+N
        total+=a[i][j]                      #    Comparision operator
 (N+1)+N*(M+1)
                                            #    Accessing array
N*M
                                            #    Increment Operation
 (N+1)+N*(M+1)
                                            #    Arithematic operation
N*M
# Here we get total of 2(1+1+N)+2((N+1)+N*(M+1))+2(N*M) = 6+6N+4NM since this NM
is very large we assume it same numbers then we
# get 6+6N+4N*N
```

Asymptotic analysis

there are mainly three cases in this Best case (Lower bound), Worset case (Upper bound), and Average case (B/W Upper and Lower) In terms of performance of algo we do not need to be concerned about the best case, we should be more bothered about the worst case senario that guarantee that this is the extreme but in some cases the upper bound and the lower bound are the same like if we want to get the sum of a list or array we need to go through all the elements of list but in this case also we chech the upper bound Now in order to chech the performance we need Asymptotic notations:- Big-oh O() (It chechs the upper bound) Big-Omega Ω() (It checks the lower bound) Big-theta Θ() (It measures between Upper and the Lower bound) when big-o and big-omega are same then we use the big_theta notation and we consider it as Θ(n/2) as this is mainly the middle value

Order of Growth

Performance Classification Constant O(1) Logarithmic O(log(n)) Linear O(n) n-log-n O(n(log(n))) Quadratic O(n^2) Cubic O(n^3) Exponential O(2^n)

Memory or space complexity

This simply tells us how many bits or bytes of memory is allocated for the program to run or during execution Primitive data types array-1d bytes Boolean 1 char[] 2*n Byte 1 int[] 4*n char 2 double[] 8*n int 4 float 4 long 8 double 8 now lets look at some examples

In [ ]:

```
int total=0
int i=1
while i<=n: # let us assume that the n is global variable or function parameter
 which is of type double then the memory consumed
    total+=i  # will be 2*4+8 =16 bytes this memory the program will occupy at t
he execution time
    i+=1

int total=0                # here let us assume the type of data for m and n is
 4+4 bytes(int) so the total memory is
for(int i=0;i<n;i++):
    for(int j=0;j<m;j++):  # 4+4+4*n   (4*n is the value as j is counted n times
in the nested loop)+ adding value of m and n
        total+=A[i][j]     # which is 4+4=8 we get 4+4+4+4+4*n+(the value of the
array A and for the array it type*m*n)here lets
                           # assume the type is double for the array then the va
lue is 8*n*m
                           # now adding this we get 4n+8nm+16
```

In [1]:

```
#to calculate the time complexity of a recursive function we need to know about recurrence relation
#although the time complexity is O(n) for normal recursion like this
def rec(n):
    if n>0:
        print(n)
        rec(n-1)
rec(5)
```

```
5
4
3
2
1
```

Types of recursion

There are mainly four types of recursion:- Tail recursion Head recursion Tree recursion Indirect recursion

Tail Recursion

In [3]:

```
# When the recursive function is at the end of the function and nothing else after that than it is called (tail recursion)
def calculate(n):
    if n>0:
        k=n**2
        print(k)
        calculate(n-1)
calculate(4)   # here the recursion takes place after the print statement is executed
```

```
16
9
4
1
```

Head Recursion

In [4]:

```
# When the recursive function is at the begining of the function and nothing els
e before that than it is called (Head recursion)
def calc(n):
    if n>0:
        calc(n-1)
        k=n**2
        print(k)
calc(4)       # Here the recursion is called first and when it ends after that t
he print of the value is done
```

```
1
4
9
16
```

Tree Recursion

In [5]:

```
# When a fuction calls itself more than once then it is called tree recursion
def calc(n):
    if n>0:
        calc(n-1)
        k=n**2
        print(k)
        calc(n-1)
calc(3)   # here the tree is made while calling the multiple recursion and it is
called 15 times so the time complex is O(2^n)
# which is exponential for the tree recursion
```

```
1
4
1
9
1
4
1
```

Indirect Recursion

In indirect recursion the function did not call itself but it calls the other function and the other calls the other function and this continues till the last call is for the first function def calc_A(n): def calc_A(n): if n>0: if n>0: .... calc_B(n-1) calc_A(n-1) ....

Searching

Lenear Search or sequential search

In [55]:

```
def lenear(A,key):
    i=0
    while i<len(A):
        if A[i]==key:
            return i+1
        i+=1
    return -1


list=[2,3,4,5,6]
a=lenear(list,2)
print(a)
```

```
1
```

Binary search

It only works if the list is in sorted order, then we start with the middle element if it matches then voala we found the element If key is less than middle element than the key is present in the lower half of the array and vice versa this process continues till the element is not found

In [4]:

```
# Binary search iterative algorithm
def binary(A,key):
    l=0
    r=len(A)-1
    while l<=r:
        m=(l+r)//2
        if key==A[m]:
            return m
        elif key<A[m]:
            r=m-1
        elif key>A[m]:
            l=m+1
    return -1
a=[1,2,4,4,5]
r=binary(a,4)
print(r)
```

```
2
```

In [ ]:

```
# Now writing binary search using recursion
def binary_rec(A,key,l,r):
    if l>r:
        return -1
    else:
        m=(l+r)//2
        if A[m]==key:
            return m
        elif A[m]>key:
            return binary_rec(A,key,l,m-1)
        elif A[m]<key:
            return binary_rec(A,key,m+1,r)

a=[4,6,8,9,12,45]
r=binary_rec(a,4,0,5)
print(r)
```

Sorting

Sorting is a technique to make the array or list into ascending or desending order, tere are various sorting algorithm to approach sorting 1. Selection 6. Shell 2. Insersion 7. Heap 3. Bubble 8. Count 4. Merge 9. Bucket 5. Quick 10. Radix These are the main sorting techniqes that we will discover later Till 7th place the sorting is done with the comparison approach and after that they are using index approach

Stable and unstable sorting

The method of sorting elements may differ if we conider a collection of elements where there is duplicate values then if they are arranged in ascending order and the duplicates are adjecnt to each other but there position to the ramdom collection is same than they are called stable and if the left one goes to the right side they are considered same but there position is changed therefore they are called unstable

Selection sort

in this the basic idea is that we take a random array and fetch the min element of the array,place it in the appropriate position and apply this techqnic in all the respective elements in this after placing the first element then we chech the min value in the remaining places and finds and use swaping tecnique on that to change the position selection sort take a loop and checks the elements in length -1 and in the secong for loop we need to traverse the element till nth place As there is swaping included in the selection sort therefore this is unstable algorithm Time compextity of selection sort is Comparisions :- 1+2+3+ ... +n-1 = n(n-1)/2 which is O(n^2) Swapping :- 1+1+1+1+1+1+1+1 = (n-1) which is O(n) this is the only sorting tecqnic which uses O(n) least time in swapping than the others

In [17]:

```
def selection_sort(a):
    n=len(a)
    for i in range(n):
        j=1
        for j in range(n):
            if a[i]<a[j]:
                temp=a[i]
                a[i]=a[j]
                a[j]=temp
    return a
a=[4,3,5,7,2]
r=selection_sort(a)
print(r)
```

```
[2, 3, 4, 5, 7]
```

Insertion sort

Here we select one element at the time from the left of the collection Insert the element at the proper position after insertion every element to its left will be sorted also insersion sort is stable sorting tecnique Time compextity of selection sort is Comparisions :- 1+2+3+4+....+n-1 =n(n-1)/2 which is O(n^2) Swapping :- 1+2+3+4+....+n-1 =n(n-1)/2 which is O(n^2)

In [4]:

```
def insersion_sort(a):
    n=len(a)
    for i in range(1,n-1):
        cvalue=a[i]
        while a[i]>0 and a[i-1]>a[i]:
            a[i]=a[i-1]
            i=i-1
        a[i]=cvalue
    return a
a=[1,2,34,5,7,3,21,23,4]
r=insersion_sort(a)
print(r)
```

```
[1, 2, 5, 7, 3, 21, 23, 34, 4]
```

Bubble sort

it compares the consecutive elements If the element to the left is greater than the right element than swap them Continue this to the end of the collection and perform several passes to sort the element Bubble sort tecnique is also stable sort Time compextity of selection sort is Comparisions :- n-1+....+3+2+1 =n(n-1)/2 which is O(n^2) Swapping :- n-1+....+3+2+1 =n(n-1)/2 which is O(n^2)

In [5]:

```
def bubble_sort(a):
    n=len(a)
    for i in range(n-1,0,-1):
        for j in range(i):
            if a[j]>a[j+1]:
                temp=a[j]
                a[j]=a[j+1]
                a[j+1]=temp
    return a
a=[2,3,46,8,8,4,2,1,233,90]
r=bubble_sort(a)
print(r)
```

```
[1, 2, 2, 3, 4, 8, 8, 46, 90, 233]
```

Shell sort

It select an element and compare element after a gap similar to insertion sort and it insert selected element from the gap at its proper position Let us assume that the collection have 6 elements in it then the it divides by 2 and get 3 as the gap, now it compares every 3 gap index on the left hand side for the first index but at left there is nothing so it compares the right hand side with the 3 gap in the index so with the 4th element after the process the gap is again divided by 2 then 3/2 we get 1 now it do the same process over and over till we get the sorted result and till the gap value becomes zero Time compextity of selection sort is Gaps :- n/2,gap/2,gap/2,....,0 = log2(n) which is O(nlog(n)) It is not hard an fast rule that we only divide the gap by 2 we can also divide it by 3 or any number usually we use 2 or any prime number which is less than the collection

In [16]:

```
def shell_sort(a):
    n=len(a)
    gap=n//2
    while gap>0:
        i=gap
        while i<n:
            temp=a[i]
            j=i-gap
            while j>=0 and a[j]>temp:
                a[j+gap]=a[j]
                j=j-gap
            a[j+gap]=temp
            i+=1
        gap//=2
    return a
a=[1,3,4,6,7,2,23,1,2]
r=shell_sort(a)
print(r)
```

```
[1, 1, 2, 2, 3, 4, 6, 7, 23]
```

Merge sort

In this we divide the collection of elements in smaller subsets Recursively sort the subset Combine or merge the solution into one result Divide and conquer approach if let us assume that there is a collection of 6 elements in an array then we divide this collection by half and thus getting two smaller subsets then there is left half and the right half first of all let us assume the left half we further divide the left half into two parts but the left half contains only 3 members so we cannot divide it equally so we consider the first element be one half and ther other two as the other half and again we divide the two element into half then we merge the sorted array same things we do for the other right half after the bot half are sorted then the easy part is done now the conquer part which is difficult now we take the first sorted element on each half and compares them Time compextity of selection sort is Comparisions :- n1+n2 which is O(n1+n2) = O(n) Subset at levels :- 1+2+4+16+.... 2^0+2^1+2^2+2^3+....=log2(n) Therefore running time of merge : O(n) Total time spent : O(n)*log2(n) which is O(nlog(n))

In [3]:

```
def merge_sort(a,left,right):
    if left<right:
        mid=(left+right)//2
        merge_sort(a,left,mid)
        merge_sort(a,mid+1,right)
        merge(a,left,mid,right)
    return a

def merge(a,left,mid,right):
    i=left
    j=mid+1
    k=left
    b=[0]*(right+1) # this is the size of the array
    while i<=mid and j<=right:
        if a[i]<a[j]:
            b[k]=a[i]
            i+=1
        else:
            b[k]=a[j]
            j+=1
        k+=1
    while i<=mid:
        b[k]=a[i]
        i+=1
        k+=1
    while j<=right:
        b[k]=a[j]
        j+=1
        k+=1
    for x in range(left,right+1):
        a[x]=b[x]

a=[2,45,6,8,9,12,3,43,65,78,44,1]
l=len(a)-1
r=merge_sort(a,0,l)
print(r)
```

```
[1, 2, 3, 6, 8, 9, 12, 43, 44, 45, 65, 78]
```

Quick sort

It basically divides the collection of elements into subset or partitions Partition based on pivot Recursivly sort the partition using quick sort divide and conquer approach here pivot basically is a point from which the left hand of that point all the elements are smaller than the right side elements, if we cant find the pivot than we arrange them which is called partioning and then we get the pivot point Time compextity of selection sort is O(n^2)

In [14]:

```
def quick_sort(a,low,high):
    if low<high:
        pi=partition(a,low,high)
        quick_sort(a,low,pi-1)
        quick_sort(a,pi+1,high)

def partition(a,low,high):
    pivot=a[low]
    i=low+1
    j=high
    while True:
        while i<=j and a[i]<=pivot:
            i+=1
        while i<=j and a[j]>pivot:
            j-=1
        if i<=j:
            a[i],a[j]=a[j],a[i]
        else:
            break
    a[low],a[j]=a[j],a[low]
    return a

a=[1,34,5,2,54]
l=len(a)-1
r=quick_sort(a,0,l)
print(r)
```

```
-------------------------------------------------------------------
--------
TypeError                                 Traceback (most recent ca
ll last)
<ipython-input-14-3c4da0c89284> in <module>
     23 a=[1,34,5,2,54]
     24 l=len(a)-1
---> 25 r=quick_sort(a,0,l)
     26 print(r)

<ipython-input-14-3c4da0c89284> in quick_sort(a, low, high)
      2     if low<high:
      3         pi=partition(a,low,high)
----> 4         quick_sort(a,low,pi-1)
      5         quick_sort(a,pi+1,high)
      6 

TypeError: unsupported operand type(s) for -: 'list' and 'int'
```

Python built in sort function

In [15]:

```
a=[1,3,4,52,2,1,43,45,6,9] # built in sort function default it sorts in ascending order
b=sorted(a)
print(b)
```

```
[1, 1, 2, 3, 4, 6, 9, 43, 45, 52]
```

In [16]:

```
a=[1,3,4,52,2,1,43,45,6,9]  # built in sort method to do sort
a.sort()
print(a)
```

```
[1, 1, 2, 3, 4, 6, 9, 43, 45, 52]
```

In [17]:

```
a=[1,3,4,52,2,1,43,45,6,9] # use reverse=True for printing in descending order
a.sort(reverse=True)
print(a)
```

```
[52, 45, 43, 9, 6, 4, 3, 2, 1, 1]
```

Linked List

what is array array is a sequential memory list, python uses low level concept of arrays to represent the sequenced we use list,tuple, and str as a built in sequential classes but there are some limitations of array particularly arrays that all the data stored must be of same integer type, secondly an array must be of fixed size (immutable), once the array is created it cant be changed and also the array stores the data sequntially Now let us talk about linked list:- It is a collection of elements, where each element is represented as node, and each node contains the data along with the link of the other node here the link stores the address of the other nodes and the other stores the address of the next node and the process is not sequntial like for arrays there may be the address of the next node be way different memory than the previous one, and the last element of the linked list stores null advantage of linked list is that they can store only that elements which we basically need and that would save the memory too, also it is more flexible than the array while creating a node we need two things element and the link or here we are going to call it as next, here in node the element and next are called instance variables So in order t do so we need dictionary for that to store the instantaneous variables, but here to avoid the dict we need to use a built-in class called (__slot__) by this we can create the linked list, we use it as class Node: __slot__ = '_element', '_next' def __init__(self,element,next): now we need to initialize the class so to do so we need the init method self._element=element self._next=next here self is describing the instance class method and the (element and the next) is different from the (_element and _next) the _ variables are instance variable which is visible throughout the class inside the init method we are going to give the values to the instance variables Let us now create the first node by an object N1 as the node n1=Node(7,None) here the n1 is the object and the node is the class and the arguments are element as 7 and next as none None is a premitie value in python and now the first node is created, to access the elements of the first node we use dot like n1._element n1._next After this another node is created n2=Node(4,None) now we have to link these two so we need to take the memory address of n2 to n1 so to do so we need to so in order to do so we write n1._next=n2 here remember the dot is used to access the members of the class here each node is a unique object, the first and the last node of a linked list is called a head and tail respectively without the reference of head there will be no first list we may or may not locate the tail reference so we allocate head to a variable x=head here head is pointing the first node value the time complexity of linked list is O(1)

In [9]:

```
#traversing a linked list when we move from head to tail then we get the linked
 list and the method is called traversing

class _node:
    __slots__='_element','_next'    # here we are using slots to efficiently all
ocate the memory for the instance variable
    def __init__(self,element,next):# and to overcome the problem of implicit cl
ass which require additional data usage and
        self._element=element       # memory and is not suitable for data struct
ure , now we need to initialize the node class
        self._next= next

class linkedlist:                # this class is used to define various operation
 to create the linked list
    def __init__(self):          #init method has only one parameter since this is
an instance method of the linked list class
        self._head= None
        self._tail=None
        self._size= 0

    def __len__(self):          # this method is used to store the size of the lin
ked list
        return self._size       # this will store the linked list

    def isempty(self):          # here we are not preceding or succeding with __ a
s this is user defined method but init and len
        return self._size==0    # are class level built in methods, this method ch
ecks wheather the link is empty or not

    def addlast(self,e):        # here the patameter e contains the value of the l
ist
        newest=_node(e,None)
        if self.isempty():
            self._head=newest
        else:
            self._tail._next=newest
        self._tail=newest
        self._size+=1

    def addfirst(self,e):
        newest=_node(e,None)
        if self.isempty():
            self._head=newest
            self._tail=newest
        else:
            newest._next=self._head
            self._head=newest
        self._size+=1

    def addany(self,e,position):
        newest=_node(e,None)
        p=self._head
        i=1
        while i<position-1:
            p=p._next
            i+=1
        newest._next=p._next
        p._next=newest
        self._size+=1

    def display(self):        # for traversing the linked list
        p=self._head          # here p is the pointer
        while p:
            print(p._element,end='-->')
            p=p._next
        print()

    def search(self,key):
        p=self._head
        index=0
        while p:
            if p._element==key:
                return index
            p=p._next
            index+=1
        return -1
    def removelast(self):
        if self.isempty():
            print("List is empty")
            return
        p=self._head
        i=1
        while i<len(self)-1:
            p=p._next
            i+=1
        self._tail=p
        p=p._next
        e=p._element
        self._tail._next=None
        self._size-=1
        return e

    def removeany(self,pos):
        if self.isempty():
            print("The list is empty")
            return
        p=self._head
        i=1
        while i<pos-1:
            p=p._next
            i+=1
        e=p._next._element
        p._next=p._next._next
        self._size-=1
        return e



L=linkedlist()               # here L is the object of linkedlist class
L.addfirst(15)
```

```
L.addfirst(4)
L.addfirst(3)
L.addlast(25)
L.addfirst(1)
L.addany(2,3)
L.removeany(3)
L.display()
print("size:",len(L))
i=L.search(15)
print(i)
```

```
1-->3-->4-->15-->25-->
size: 5
3
```

Delete a node in the Linked list

Here in order to delete a node at the tail of the linked list we need to travese the linked list from the head as we cannot delete the tail node as we cannot traverse it backward cause the next only goes forward only so we always goes from head to tail if we want to delete the last node you can find the delete function in the above example

Stack

In [10]:

```
def StringChallenge(str):
  l=len(str)
  str2='passWord'
  bool=False
  if str2 in str:
        return bool

  for i in str:
        if(ord(i)>=33 and ord(i)<=126 and l>7 and l<31):
            return True
  return str

# keep this function call here
print(StringChallenge(input()))
```

```
passWord123!!!!
False
```

Practice

In [65]:

```
def ArrayChallenge(arr):
  l=len(arr)
  dict={"count":[],"value":[]}
  for i in range(0,l,1):
        j=0
        count=0
        for j in range(0,l,1):
            if arr[j]==arr[i]:
                count+=1
        dict["count"].append(count)
        dict["value"].append(arr[i])
  return dict

# keep this function call here
print(ArrayChallenge(input()))
```

```
2 3 4 2 3 4 3 3
{'count': [2, 7, 4, 7, 2, 7, 2, 7, 4, 7, 2, 7, 4, 7, 4], 'value':
['2', ' ', '3', ' ', '4', ' ', '2', ' ', '3', ' ', '4', ' ', '3', '
', '3']}
```

In [6]:

```
import numpy as np
a=np.array([[2,3,4,51,2],[3,5,6,7,30]])
print(a)
```

```
[[ 2  3  4 51  2]
 [ 3  5  6  7 30]]
```

In [10]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data_df=pd.read_excel('DATA.xlsx','DATA',index_col=None,na_value=['NA'])
data_df.head
```

```
-------------------------------------------------------------------
--------
ValueError                                Traceback (most recent ca
ll last)
~\anaconda3\lib\site-packages\xlrd\book.py in sheet_by_name(self, s
heet_name)
    473         try:
--> 474             sheetx = self._sheet_names.index(sheet_name)
    475         except ValueError:

ValueError: 'DATA' is not in list

During handling of the above exception, another exception occurred:

XLRDError                                 Traceback (most recent ca
ll last)
<ipython-input-10-f4635ed18cd8> in <module>
      2 import pandas as pd
      3 import matplotlib.pyplot as plt
----> 4 data_df=pd.read_excel('DATA.xlsx','DATA',index_col=None,na_
value=['NA'])
      5 data_df.head

~\anaconda3\lib\site-packages\pandas\io\excel\_base.py in read_exce
l(io, sheet_name, header, names, index_col, usecols, squeeze, dtyp
e, engine, converters, true_values, false_values, skiprows, nrows,
 na_values, keep_default_na, verbose, parse_dates, date_parser, tho
usands, comment, skipfooter, convert_float, mangle_dupe_cols, **kwd
s)
    309         )
    310 
--> 311     return io.parse(
    312         sheet_name=sheet_name,
    313         header=header,

~\anaconda3\lib\site-packages\pandas\io\excel\_base.py in parse(sel
f, sheet_name, header, names, index_col, usecols, squeeze, converte
rs, true_values, false_values, skiprows, nrows, na_values, parse_da
tes, date_parser, thousands, comment, skipfooter, convert_float, ma
ngle_dupe_cols, **kwds)
    866             )
    867 
--> 868         return self._reader.parse(
    869             sheet_name=sheet_name,
    870             header=header,

~\anaconda3\lib\site-packages\pandas\io\excel\_base.py in parse(sel
f, sheet_name, header, names, index_col, usecols, squeeze, dtype, t
rue_values, false_values, skiprows, nrows, na_values, verbose, pars
e_dates, date_parser, thousands, comment, skipfooter, convert_floa
t, mangle_dupe_cols, **kwds)
    437 
    438             if isinstance(asheetname, str):
--> 439                 sheet = self.get_sheet_by_name(asheetname)
    440             else:  # assume an integer if not a string
    441                 sheet = self.get_sheet_by_index(asheetname)
```

```
~\anaconda3\lib\site-packages\pandas\io\excel\_xlrd.py in get_sheet
_by_name(self, name)
     41 
     42     def get_sheet_by_name(self, name):
---> 43         return self.book.sheet_by_name(name)
     44 
     45     def get_sheet_by_index(self, index):

~\anaconda3\lib\site-packages\xlrd\book.py in sheet_by_name(self, s
heet_name)
    474             sheetx = self._sheet_names.index(sheet_name)
    475         except ValueError:
--> 476             raise XLRDError('No sheet named <%r>' % sheet_n
ame)
    477         return self.sheet_by_index(sheetx)
    478 

XLRDError: No sheet named <'DATA'>
```

In [35]:

```python
import numpy as np

def findarr(arr,k):
    for i in arr:
        if(any(k for j in i)==True):
            print("Yes")
            break
        else:
            print("No")

t=int(input())
for i in range(t):
    k,n,m=map(int, input().split())
    arr=np.array([input().split() for i in range(n)], dtype=int)
    findarr(arr,k)
```

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

In [51]:

```
import numpy as np

def arr(arr,x):
    n=len(arr)
    m=len(arr[0])
    print(n)
    print(m)
    for i in range(n):
        for j in range(m):
            if(arr[i][j]==x):
                return True

    return False

a=np.array([[2,3],[4,3],[4,5]])
print(arr(a,5))
```

```
3
2
True
```

In [50]:

```
a.shape
```

Out[50]:

```
(3, 2)
```