Day 3:-

print(10*4/6+3-1.1/.2)

in python => 8.66

in C => 8.00

C	Python	Java	
&	and	&	(and)
|	|	|	(or)
~	~	~	(Negatition) (or) (Bit not)
!			
(1's complementary)			
^	^	^	(XOR symbol) Eschar

Tilt symbol = ~

Backlake = `

int = 32 bits (store)

7 + 244 + 349.

8 4 2 1

0111
0010
0010
0100
~~0000~~
~~0011~~
0000
1001
0000

0110
0011
0010
1001

0011
1001
0001

0010
0100
0000

1

=> highest => + -

=> 244
0010
0100
0000

=> 349.
0011
1001
1000

7 0111
244 0110
=> 0110
349 1010
0010
4
2

(→ print (7&2&4&(3&9)) 84 &1

^ → 11 → 1
00 → 1
4 → 10 → 0
01 → 0

0010
0100

0000

0011
1001

0001

And (&)

0	0	0
0	1	0
1	0	0
1	1	1

Or (|)

0	0	0
0	1	1
1	0	1
1	1	1

0111
0000

0111

→ 0111
0001

1

2&4

0010
0100

0000

3&9

0011
1001

0001

0111 → 7
0000

0111
0001

0001
→ 1

Highest priority:-

→ && → high precendenc
|| → low precendence

→ print (10 % 3&4) → Ans: 0

print (0 | 3&4)
→ 1

& — (2) " Multiple operator
| — (4) "
~ — (1) → works with one operator → Ans → Negative: n = −(n+1)
^ — (3) → multiple.

formula:-
−(n+1)
n = 6
→ −(6+1)
= −7

~7 = −8

① valid

② polarity

③ system o/p

① 10+4∧2 → Invalid.

② ∧9+4+6 → valid → 2

③ 6!3+9+6 → valid

④ 2∧413*2* → Invalid

⑤ 9∧ → Invalid

→ starts with Negation ∧ is only valid ex:- ∧9.

XOR - ∧

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Concept =>

0	1	
0	1	
0	0	

Types :-

- Inclusive XOR
- Exclusive XOR

→ shift ← 1) left shift << => Right - left.

2) Right shift >> → toward Rights.

→ left → Right

* Bit manipulation Tricks:-

xor ^

even 1's : 0
odd 1's : 1

ex:-

xor of number itself is 0 => 5^5=>0

xor of number with 0 is number itself

ex:-

ex:- => 5^5^5

=> 0^5

=> 5

=> 1^2^3

=> 3^3 = 0

=> 4^6^5

0 1 0 0
0 1 1 0
0 1 0 1
0 1 1 1

=> 7

① Right shift

5 >> 1

0101 => >> 0010

5 2

0101
0010
0000

② 5 >> 2

0101 => 0001 => 1

③ 6 >> 3

0110 => 0000

0

④ 14 >> 4

1110 => 0000

0

=> left bits
int 32 bits

1) 5 - 0101

5<<2

5 * power(2,2) => 5 x 4 = 20

2) 10<<3

10 * power(2x3) => 10 x 2^3 => 10 x 8 = 80

3) 9<<4

9 * power(2x4) -> 9 x 2^4 => 144.

=> In the given array every number occurs twice only one number occurs once. find the number which is occurance once

```
def findsingle(ar,n):
    res=ar[0]
    for i in range(1,n):
        res=res^ar[i]
    return res
ar=[2,3,4,5,3,4,2]
print(findsingle(ar,len(ar)))
```

Swap 2 number using xor

```
a=100
b=200
print("a:",a,"b:",b)
```

a=a^b a=100^200
b=a^b b=100^200^200 ans 200 200 cancelled b=100
a=a^b a=100^200^100 ans 100 gets cancelled a=200

```
print("a:",a,"b:",b)
```

> For the given number n check the kth bit is set (or) not.

```
n=13
k=2
if n&(1<<k-1):
    print("set")
else:
    print("not set")
```

o/p:- set

→ for given number n print the xor of all numbers

i/p = 5

ans should be 1^2^3^4^5

o/p = 1

Bi

```
n=12
xor=0
for i in range(1,n+1):
    xor=xor^i
print(xor)
```

O(n)

optimize

n	
1	1
2	3
4	0
4	4
5	1
6	7
7	0
8	8
9	1

```
n=7
xor=0
if n%4==0:
    print(n)
elif n%4==1:
    print(1)
elif n%4==2:
    print(n+1)
elif n%4==3:
    print(0)
```

=> above is O(1) no loop

---

* Given a range l-r print xor

l=2; r=4

2^3^4

ans=5

let say

l=2 r=5

2^3^4^5

↓

1

0

Lets say ⇒ we can find 1 to any number.

4 to 9

xor(1 to 9) ^ xor(1 to 3)

(1^2^3^4^5^6^7^8^9) ^ (1^2^3)

in above 123 xors will get cancelled

generalise xor(n) ^ xor(n-1)

* check number even (or) odd.

```
n=13
if (n & 1 == 0):
    print("even")
else:
    ("odd")
```

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

nterms = int(input("how many terms ?"))
if nterms <= 0:
    print("enter integer")
else:
    print("fib seq:")
    for i in range(nterms):
        print(fib(i))
```

```
if:
    n==1
    return
else
    (n*recur_factorial(n-1)
    num=5
elif num==0:
else:
```

Time Complexity : ex:- n=10

```
for(i=0; i<n, i++){}
```

TC = O(n)

masters theorem

=> n=10

```
f(i=0; i<n; i+=2)
```

n = 10 => 5

n = 50 -> 25

TC = O(n/2)

ignore constants

TC = O(n/2) = 1/2 x O(n) => O(n)

→ for(i=0; i>0, i--=10) → (increament & decrement)
⇒ T.C ⇒ O(n)

n=100 → to=n/10

TC = O(n/10) = O(n)

Rule 2:

⇒ for(i=0; i<100; i*=2) → O(1)

T.C = O(log(n))

⇒ for(i=0; i<=100; i/=2)

T.C = O(log(n))

⇒ for(i=0; i<n; i++) ⟺ O(n)

{

for(j=0; j<n; j*=2) ⇒ O(log(n))

{

//code

}

}

T.C ⇒ O(n log n)

⇒ for(i=0; i<n; i++) → O(n)

{

for(j=0; j<n; j+=2) → O(n)

{

//code

}

}

TC ⇒ O(n²)

Masters

TC(n):

2.

3.

Space

ex:- in

in

q

Masters theorm

T(n) = 1

2.

3.

Space Complexity

ex: int a=10
int b=20
int c=30
print(a,b,c)

} 3 variable → Constant

n=100
a[n] = 
print(arr)
⇒ O(n)

⇒ temp=arr
printf(temp) ⇒ O(2n) ⇒ 2x(o(n)) =

O(n)