MATHS

Logarithm

log of value with the rerspect to base is a power

Python functions for computing log:

- math.log(x) -> returns log(x) with base=e
- math.log(x,a) -> returns log(x) with base=a
- math.log2(x) -> returns log(x) with base=2
- math.log10(x) -> returns log(x) with base=10

In [21]:
```
import math
```

In [2]:
```
math.log(10)
```

Out[2]:
```
2.302585092994046
```

In [3]:
```
math.log(343,7)
```

Out[3]:
```
3.0
```

In [4]:
```
math.log2(8)
```

Out[4]:
```
3.0
```

In [5]:
```
n = 100

while n > 1:
    print(n)
    n = n/2
```

```
100
50.0
25.0
12.5
6.25
3.125
1.5625
```

How many iterations would above code snippet take?

(in term of n)

here we can find by every time loop divide the value i two parts

approx log n

In [ ]:

GCD - Euclid's Algo

math module method - math.gcd(*integers)

gcd(a, b) = gcd(b%a, a) if a > 0
gcd(a, b) = b if a == 0

In [6]:
```
math.gcd(2,4)
```

Out[6]: 2

In [7]:
```
math.gcd(35,50)
```

Out[7]: 5

pseudocode - Iterative

In [8]:
```
a = int(input())
b = int(input())

while a != 0:
    a,b,= b%a, a

print("GCD is", b)
```

```
50
45
GCD is 5
```

In [9]:
```
math.gcd(10,18,60,40,12)
```

Out[9]: 2

Pseudocode - Recursive

In [2]:
```
def gcd(a, b):
    if a == 0:
        return b
    return gcd(b%a, b)
```

In [3]:
```
gcd(14,2)
```

Out[3]: 2

LCM

math module method - math.lcm(*integers)

In [10]:
```
math.lcm(15,10,40)
```

Out[10]: 120

Compute GCD using Euclid's algo and find LCM with the belove formula.

> gcd(a, b) * lcm(a, b) = a * b

Pseudocode

In [6]:
```
def lcm(a, b):
    return (a * b) / gcd(a, b)
```

In [7]:
```
lcm(4,2)
```

Out[7]: 4.0

In [ ]:
```

```

Check prime number

let take number 36

N = 36 -> 1,2,3,4,6,9,12,18,36

N = 40 -> 1,2,4,5,8,10,20,40

all devisors of a no exists in pairs in these all devisors some of all are small numbers and some of these are large numbers

here in 40:

1,2,4,5 are small numbers that are less than root n

8,10,20,40 are large numbers that are greater than root n

In [7]:
```
def isPrime(n):
    '''
    This function retuens True or False based on primality check of n.
    Time complexity - o(sqrt(n))
    '''

    if n == 1:
        return False

    sq = int(math.sqrt(n))

    for i in range(2,sq+1):
        if n%i == 0:
            return False
    return True
```

In [6]:
```
isPrime(6)
```

Out[6]:
```
False
```

In [ ]:
```

```

AP - Arithmatic Progression

An arithmetic progression(AP),also called an arithmetic sequence of numbers which differ from each other by a common difference. For example,the sequence 2,4,6,8,... is an arithmetic sequence with the common diff 2.

let d = common difference let a = first term

SUM FORMULA:

```
T(n) = a + (n-1)d
```

SUM TILL n TERNS:

```
T(n) = n/2(2a+(n-1)d
```

Geometric Progression - GP

A geometric progression (GP), also called a geometric sequence, is a sequence of numbers which differ from each other by a common ratio. For example, the sequence 2, 4, 8, 16, ... is a geometric sequence with common ratio 2.

Let a = Initial Term
Let r = Common ratio

Nth term:

> T(n) = a * pow(r,n-1)

Sum of GP:

> S(n) = a * ((pow(r,n) - 1)/(r - 1)) if r != 1
> S(n) = a * n if r == 1

Combinations

The no. of ways to choose r objrcts out of n total objects.

```
n C r = C(n,r)
```

math module method-
math.combs(n, r)

In [3]:
```
math.comb(10,4)
```

Out[3]:
```
210
```

In [5]:
```
def combination(n,r):
    num = math.factorial(n)
    den = math.factorial(r) * math.factorial(n-r)
    return num/den
combination(10,4)
```

Out[5]:
```
210.0
```

Arrangments

if you have n objects then you can arrange that n objets in n! ways

In [6]:
```
math.factorial(11)
```

Out[6]:
```
39916800
```

Permutations

choose and arrange them

ex: if there is 20 players in team and you have to choose 11 players and arrange them...then it is permutation problem

math module method-
math.perm(n,r)

In [7]:
```
math.perm(20,11)
```

Out[7]:
```
6704425728000
```

In [8]:
```
def permutation(n, r):
    return math.factorial(n) / math.factorial(n-r)
```

In [9]:
```
permutation(11,4)
```

Out[9]:
```
7920.0
```

Decimal to Binary

In order to convert a decimal number to binary, repeatedly divide by two until you reach 0. Store the remainders seperately and the resultant string of those remainders in reverse order is the binary equivalant.

Using python inbuilt functionality:

```
def decimalTobinary(n):
    return bunn(n).replace("0b", " ")
```

In [1]:
```
def decimalToBinary(n):
    return bin(n).replace("0b", "")
```

In [2]:
```
bin(20)
```

Out[2]:
```
'0b10100'
```

In [9]:
```
def decimalToBinary(n):
    l = []
    while n>0:
        remainder = n % 2
        l.append(str(remainder))
        n = n//2

    l = list(reversed(l))

    return "".join(l)
```

In [10]:
```
decimalToBinary(20)
```

Out[10]:
```
'10100'
```

Binary to Decimal

To convert a binary string to decimal integer, start iterating from the right and multiply each digit

In [17]:
```
def binaryToDecimal(s):
    i = 0
    result = 0

    for digit in s[::-1]:
        result += int(digit) * (2**i)
        i = i + 1

    return result
```

In [19]:
```
binaryToDecimal("90")
```

Out[19]:
```
18
```

Trailing Zeros in factorial number

solution link : https://youtu.be/zpvDctj8mM4?si=UuEBFj2-Dpa-6oha (https://youtu.be/zpvDctj8mM4?si=UuEBFj2-Dpa-6oha)

In [23]:
```
math.factorial(20)
```

Out[23]:
```
2432902008176640000
```

In [24]:
```
math.factorial(30)
```

Out[24]:
```
265252859812191058636308480000000
```

In [25]:
```
math.factorial(100)
```

Out[25]:
```
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000
```

In [26]:
```
100//5
```

Out[26]:
```
20
```

In [29]:
```
def trailingZeroes(n):
    result = 0
    den = 5
    x = n // den

    while x >= 1:
        result += x
        den *=5
        x = n // den
    return result
```

In [30]:
```
trailingZeroes(100)
```

Out[30]: 24

In [ ]:
```

```