Number Theory Using Python
Lab E-Record

MAT651D

Name: Apanvi Srivastava
Reg. no.: 2240250
Class: 6 CMS

Department of Mathematics

Under the supervision of

Dr. Puneeth V.
Arkaprovo Chakraborty

LAB 1 : DIVISIBILITY

25th Novermber 2024

Apanvi Srivastava 2240250

Question 1: Write a program to find the divisors of any positive number.

In [1]:
```
n = int(input("Enter any number - "))

def find_divisors(n):
    i=2
    lis=[1,n]

    while i<=n/2:
        if n%i==0:
            lis.append(i)
        i+=1

    lis.sort()
    return lis

lis=find_divisors(n)
print(lis)
```

```
[1, 2, 3, 6]
```

Question 2: Write a program to find the divisors of any number including negatives.

In [2]:
```
n = int(input("Enter any number - "))
i=2
lis=[1,-1,-n,n]

n=abs(n)

while i<=n/2:
    if n%i==0:
        lis.append(i)
        lis.append(-i)
    i+=1

lis.sort()
print(lis)
```

```
[-20, -10, -5, -4, -2, -1, 1, 2, 4, 5, 10, 20]
```

Question 3: For any positive integer n, find $\tau(n)$ and $\sigma(n)$.

$\tau(n)$ = number of positive divisors on n

$\sigma(n)$ = sum of positive divisors on n

In [3]:
```
n=int(input("Enter any number - "))

lis=find_divisors(n)
print(lis)

print("Tau =", len(lis))
print("Sigma =", sum(lis))
```

```
[1, 2, 31, 59, 62, 118, 1829, 3658]
Tau = 8
Sigma = 5760
```

Question 4 : Find a relation between $\sigma(p)$ and p where p is a prime number

In [4]:
```
n=int(input("Enter any number - "))

lis=find_divisors(n)

print("p+1 =", n+1)
print("Sigma =", sum(lis))
```

```
p+1 = 14
Sigma = 14
```

Question 5 : Find a relation between $\sigma(p)$ and n where $n = 2^k$

In [5]:
```
n=int(input("Enter any number - "))

lis=find_divisors(n)

print("2n-1 =", 2*n - 1)
print("Sigma =", sum(lis))
```

```
2n-1 = 65
Sigma = 48
```

Question 6 : Find a relation between $\sigma(p)$ and $\tau(p)$ where p is a prime number

In [6]:
```
n=int(input("Enter any number - "))

lis=find_divisors(n)

print("Tau(p)+p-1 = ", len(lis)+n-1)
print("Sigma =", sum(lis))
```

```
Tau(p)+p-1 =  33
Sigma = 56
```

Question 7 : Write a python code to prove the division algorithm

In [7]:
```
a=int(input("Enter a number - "))
b=int(input("Enter another number - "))

q=a//b
r=a%b

print(a,"=",b,"X",q,"+",r)
```

```
10 = 3 X 3 + 1
```

Question 8 : Create a function to find the prime factorisation of an integer and use that function to find the gcd of the number

In [8]:
```
# Prime Factorisation

def isprime(n):

    for i in range(2,n):
        if n%i==0:
            return False

    return True

n=int(input("Enter a number: "))
a=int(input("Enter another number: "))
l=[]
m=[]
for i in range(2,n+1):
    while n%i==0 and isprime(i)==True:
        l.append(i)
        n//=i
for i in range(2,a+1):
    while a%i==0 and isprime(i)==True:
        m.append(i)
        a//=i

print(l)
print(m)

def product(q):
    r=1
    for i in q:
        r=r*i
    return r

def gcd(a,b):
    q = []
    used_indices = set()  # Track used indices in m
    for i in l:
        for idx, j in enumerate(m):
            if i == j and idx not in used_indices:
                q.append(i)
                used_indices.add(idx)
                break
    print("Common prime factors:", q)
    print("GCD:", product(q))

gcd(n,a)
```

```
[2, 2, 5]
[3, 5]
Common prime factors: [5]
GCD: 5
```

LAB 2 : Conversions

13th December 2024

Apanvi Srivastava 2240250

Binary to Decimal Number

Write the algorithm and program to convert a binary number to a decimal number.

algorithm

1. The binary number that is to be converted to decimal is taken as an input from the user. The data type of the input is string which makes indexing easier and preserves the preceding zeros if any. The preserving of the preceding zeros does not hold any significance in converting the number to decimal, however.
2. A variable s is initialised to 0 which stores the sum of the powers of 2 corresponding to the 1s in the binary input to convert it to decimal.
3. A for loop is created which for the number of times as is the length of the binary input. Every time the loop iterates, every digit of the binary input is addressed to. if the digit is 1, the signal is positive which raises to 2 its position and sums it up to s. By the end of the execution of the loop, the positive powers are all summed up.
4. The variable s, the decimal equivalent, is printed.

In [1]:
```
n = input("Enter the Binary Number: ")
print("Binary number : ", n)

s=0
for i in range(0,len(n)):
    if n[len(n)-i-1]=='1':
        s+=2**i

print("The Decimal of the given Binary Number is: ",s)
```

```
Binary number :  10101011
The Decimal of the given Binary Number is:  171
```

Decimal to Binary Number

In [2]:
```
n = int(input("Enter the Decimal Number: "))
print("Decimal number : ", n)
l=[]
while n>0:
    if n%2==1:
        l.append(1)
    else:
        l.append(0)
    n=int(n/2)
print("The Binary number is",str(l))
```

```
Decimal number :  9870
The Binary number is [0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1]
```

Decimal to Octal Number

In [3]:
```
n = int(input("Enter the Decimal Number: "))
print("Decimal number : ", n)

s=''
while n>0:
    s+=str(n%8)
    n=int(n/8)
print("The Octal Number is", s[::-1])
```

```
Decimal number :  3456
The Octal Number is 6600
```

Octal to Decimal Number

In [4]:
```
n = input("Enter the Octal Number: ")
print("Octal number : ", n)

s = 0
for i in range(len(n)):
    digit = int(n[len(n)-i-1])
    s += digit * (8**i)

print("The Decimal of the given Octal Number is:",s)
```

```
Octal number :  4563
The Decimal of the given Octal Number is: 2419
```

Decimal to Hexadecimal Number

In [5]:
```
n = int(input("Enter the Decimal Number: "))
print("Decimal number : ", n)

s=''
while n>0:
    d = n%16
    if d<10:
        s += str(d)
    elif d == 10:
        s += "A"
    elif d == 11:
        s += "B"
    elif d == 12:
        s += "C"
    elif d == 13:
        s += "D"
    elif d == 14:
        s += "E"
    elif d == 15:
        s += "F"
    n=int(n/16)

print("The Hexadecimal Number is", s[::-1])
```

```
Decimal number :  34532
The Hexadecimal Number is 86E4
```

Hexadecimal to Decimal Number

In [6]:
```
n = input("Enter the Hexadecimal Number: ").upper()
print("Hexadecimal number : ", n)
l = list(n)
```

```
s = 0
for i in range(len(l)):
    digit = l[len(l)-1]
    l = l[:-1]
    if digit == 'A':
        s += 10 * (16**i)
    elif digit == 'B':
        s += 11 * (16**i)
    elif digit == 'C':
        s += 12 * (16**i)
    elif digit == 'D':
        s += 13 * (16**i)
    elif digit == 'E':
        s += 14 * (16**i)
    elif digit == 'F':
        s += 15 * (16**i)
    else:
        s += int(digit) * (16**i)

print("The Decimal of the given Hexadecimal Number is:",s)
```

```
Hexadecimal number :  BDFA
The Decimal of the given Hexadecimal Number is: 48634
```

Euler's Phi Function

Let n be a positive integer, then $\phi(n) =$ number of integers that are less than or equal to n and relatively prime to n.

Example

1. $\phi(6) = 2$
2. $\phi(15) = 8$

Method 1

In [7]:
```
# Euler's Phi Function

def gcd(a,b):
    if (b == 0):
        return a
    else:
        return gcd(b,a % b)

def euler_phi(n):
    if n <= 0:
        raise ValueError("Input must be a positive integer,")

    count = 0
    for i in range(1,n+1):
        if gcd(n,i) == 1:
            count += 1
    return count

n = int(input("Enter a positive integer - "))
print("The phi function of", n ,"is given as:",euler_phi(n))
```

```
The phi function of 23425 is given as: 18720
```

Method 2 (using formula)

$n = p_1^{n_1} . p_2^{n_2} \ldots p_k^{n_k}$

$$\phi(n) = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2})\dots(1 - \frac{1}{p_k})$$

In [8]:
```
from sympy import *

n=int(input("Enter a number: "))
t = n
l=[]
for i in range(2,n+1):
    while n%i==0 and isprime(i)==True:
        l.append(i)
        n//=i

phi_val = t
print(l)
for i in set(l):
    phi_val *= 1 - (1/i)

print(int(phi_val))
```

```
[2, 5]
4
```

Question

Define a python function that accepts an integer 'n' and calculates the following:

$$R(n) = \sum_{d|n} \phi(d).\sigma(n/d)$$

where,
$\sigma(n)$ is the sum of the positive divisors of n,
$\phi(n)$ is the Euler's Phi functional value of n, and the summations runs over all the positive divisors of n.

In [9]:
```
from sympy import *

n = int(input("Enter any number - "))

def gcd(a,b):
    if (b == 0):
        return a
    else:
        return gcd(b,a % b)

def euler_phi(n):
    if n <= 0:
        raise ValueError("Input must be a positive integer.")

    count = 0
    for i in range(1,n+1):
        if gcd(n,i) == 1:
            count += 1
    return count

def find_divisors(n):
    i=1
    lis=[]

    while i<=n:
        if n%i==0:
            lis.append(i)
        i+=1

    lis.sort()
```

```
    return lis

def rn(n):
    divisors = find_divisors(n)
    rn_value = 0
    for i in divisors:
        rn_value += euler_phi(i) * sum(find_divisors(n/i))

    return int(rn_value)

print(f"R(n) = {rn(n)}")
```

```
R(n) = 3248
```

Mobius Function

For any positive integer n,

$\mu(n) = 0$, when n has a square factor
$\mu(n) = (-1)^K$, when n has K distinct prime factors
$\mu(n) = 0$, when $n = 1$

In [10]:
```
n = int(input("Enter a positive integer = "))

def prime_factors(n):
    factors = []
    p = 2

    while p * p <= n:
        while n % p == 0:
            factors.append(p)
            n //= p
        p += 1

    if n > 1:
        factors.append(n)

    return factors

def mobius(n):
    for i in prime_factors(n):
        if prime_factors(n).count(i) > 1:
            return 0
    if len(set(prime_factors(n))) == len(prime_factors(n)):
        return (-1) ** (len(prime_factors(n)))
    if n == 1:
        return 1

print("Mobius Function of",n,"is",mobius(n))
```

```
Mobius Function of 1234 is 1
```

Multiplicativity

An arithematic function f is called multiplicative if for any two co-prime numbers p and q,

$f(pq) = f(p)f(q)$

So following this definition, the functions, τ, σ, ϕ become multiplicative

Question

Check the mutiplicability of τ, σ, ϕ, μ functions

In [12]:
```
n1 = int(input("Enter the first number - "))
n2 = int(input("Enter the second number - "))
print()

def find_divisors(n):
    i=2
    lis=[1,n]

    while i<=n/2:
        if n%i==0:
            lis.append(i)
        i+=1

    lis.sort()
    return lis

def gcd(a,b):
    if (b == 0):
        return a
    else:
        return gcd(b,a % b)

def euler_phi(n):
    if n <= 0:
        raise ValueError("Input must be a positive integer")

    count = 0
    for i in range(1,n+1):
        if gcd(n,i) == 1:
            count += 1
    return count

def check():
    if gcd(n1,n2) != 1:
        print("Input numbers are not co-prime. Not checking multiplicativity\n")
        return
    else:
        print("Input numbers are co-prime. Checking multiplicativity\n")

    tau_12 = len(find_divisors(n1*n2))
    tau_1 = len(find_divisors(n1))
    tau_2 = len(find_divisors(n2))

    sigma_12 = sum(find_divisors(n1*n2))
    sigma_1 = sum(find_divisors(n1))
    sigma_2 = sum(find_divisors(n2))

    euler_12 = euler_phi(n1*n2)
    euler_1 = euler_phi(n1)
    euler_2 = euler_phi(n2)

    mu_12 = mobius(n1*n2)
    mu_1 = mobius(n1)
    mu_2 = mobius(n2)
    print("the numbers are: ")
    print(n1)
    print(n2)

    if (tau_12 == (tau_1 * tau_2)):
        print("Multiplicatibility of Tau Function is satisfied")
    else:
```

```
        print("Multiplicatibility of Tau Function is not satisfied")

    if (sigma_12 == (sigma_1 * sigma_2)):
        print("Multiplicatibility of Sigma Function is satisfied")
    else:
        print("Multiplicatibility of Sigma Function is not satisfied")

    if (euler_12 == (euler_1 * euler_2)):
        print("Multiplicatibility of Euler Phi Function is satisfied")
    else:
        print("Multiplicatibility of Euler Phi Function is not satisfied")

    if (mu_12 == (mu_1 * mu_2)):
        print("Multiplicatibility of Mobius Function is satisfied")
    else:
        print("Multiplicatibility of Mobius Function is not satisfied")

check()

# It might happen that some pairs exist for which it is multiplicative, even if one pair is
# condition doesn't hold, then it is non-multiplicative
```

```
Input numbers are co-prime. Checking multiplicativity

the numbers are:
23
4
Multiplicatibility of Tau Function is satisfied
Multiplicatibility of Sigma Function is satisfied
Multiplicatibility of Euler Phi Function is satisfied
Multiplicatibility of Mobius Function is satisfied
```

Pseudo Prime

A given positive integer n is called pseudoprime if

$2^n = 2(mod n)$, i.e., $n|(2^n - 2)$

In [13]:
```
n = int(input("Enter a positive integer - "))
print("the number: ", n)

def pseudoprime(n):
    if (2**n - 2) % n == 0:
        print(n,"is a pseudoprime number")
    else:
        print(n,"is not a pseudoprime number")

pseudoprime(n)
```

```
the number:  46
46 is not a pseudoprime number
```

Question : Print the first 10 pseudoprimes.

In [14]:
```
pseudoprimecnt = 0
n = 3
print("The first five pseudo prines are as follows:")
while (pseudoprimecnt <= 5):
    if (2**n - 2) % n == 0:
        print(n)
        pseudoprimecnt += 1
    n += 1
```

```
The first five pseudo prines are as follows:
3
5
7
11
13
17
```

LAB Assignment 1

20th December 2024

Apanvi Srivastava 2240250

Question 1: Program to check if a number is perfect

In [1]:
```
def perfnum(n):

    factors = []

    for i in range(1, (n // 2) + 1):
        if n % i == 0:
            factors.append(i)

    sum_factors = sum(factors)

    if sum_factors == n:
        return "Perfect Number"
    else:
        return "Not a Perfect Number"

n = int(input("Enter a number: "))
print("The number:", n)
print(perfnum(n))
```

```
The number: 28
Perfect Number
```

Question 2 : Write a program to find integers n in the range 2 to 15 for which the sum of HCFs of n with 1 to (n-1) is divided by n.

In [ ]:
```
from math import *

def ques2():
    l = []
    for n in range(2, 16):
        gcd_sum = sum(gcd(n, i) for i in range(1, n))
        if gcd_sum % n == 0:
            l.append(n)
    return l

print(ques2())
```

```
[4, 15]
```

Question 3 : Write a program to find the numbers in the range 1 to 50 with exactly 3 distinct prime factors

In [ ]:
```
from sympy import primerange

def prime_factors(n):
    factors = set()
    for p in primerange(2, n + 1):
        while n % p == 0:
            factors.add(p)
```

```
            n //= p
        if n == 1:
            break
    return factors

def numbers_with_three_prime_factors():
    results = []
    for num in range(1, 51):
        if len(prime_factors(num)) == 3:
            results.append(num)
    return results

print(numbers_with_three_prime_factors())
```

```
[30, 42]
```

Question 4: Write a program to find the numbers n in the range 1 to 10 such that $\phi(n)$ * $\tau(n)$ is a perfect square.

In [ ]:
```
from math import isqrt

def euler_phi(n):
    result = n
    p = 2
    while p * p <= n:
        if n % p == 0:
            while n % p == 0:
                n //= p
            result -= result // p
        p += 1
    if n > 1:
        result -= result // n
    return result

def num_divisors(n):
    count = sum(2 if i != n // i else 1 for i in range(1, isqrt(n) + 1) if n % i == 0)
    return count

def find_special_numbers():
    return [
        n for n in range(1, 11)
        if (phi := euler_phi(n)) * (t := num_divisors(n)) == isqrt(phi * t) ** 2
    ]

print(find_special_numbers())
```

```
[1, 3, 8, 10]
```

Question 5: Write a program to check whether 2 numbers are amicable or not

In [ ]:
```
def amicable(n1, n2):
    div_1 = []
    div_2 = []
    for i in (1, (n1//2)+1):
        if n1 % i == 0:
            div_1.append(i)
    for j in (1, (n2//2)+1):
        if n1 % j == 0:
            div_2.append(j)
    sum1 = 0
    sum2 = 0
    for i in range(len(div_1)+1):
        sum1 += i
```

```
    for j in range(len(div_2)+1):
        sum2 += j
    if sum1 == sum2:
        return "Amicable"
    else:
        return "Not Amicable"
n1 = int(input("Enter the first number:"))
print(n1)
n2 = int(input("Enter the second number:"))
print(n2)

amicable(n1, n2)
```

```
220
400
'Amicable'
```

LAB 3 : Modular Inverse, Femrat's Little Theorem, Chinese Remainder Theorem

25th January 2025

Apanvi Srivastava 2240250

Modular inverse

inverse of a mod m exists iff $gcd(a, m) = 1$

$1 \equiv +m \equiv +m \equiv +m$

EXAMPLES

1. 5 mod 7

$1 \equiv 8 \equiv 15 -> 5, 3$, hence 3 is the inverse

1. 19 mod 7

$1 \equiv 8 \equiv 15 \equiv 22 \equiv 29 \equiv 36 \equiv 43 \equiv 50 \equiv 57 -> 19, 3$, 3 is the inverse

19/7 ≡ reminder is 5,
hence 5 mod 7 and continue

In [1]:
```
from math import *
gcd(19,7)
```

Out[1]:
```
1
```

In [2]:
```
a = int(input("Enter a: "))
print(a)

m = int(input("Enter m: "))
print(m)
x = 1
def mod_inv(a,m):
    if(a > m):
        a = a % m
    if gcd(a,m) == 1:
        for i in range(1,m):
            if (a * i) % m == 1:
                return i

print("The inverse is:",mod_inv(a,m))
```

```
9
19
The inverse is: 17
```

Chinese Reminder Theorem

$x = a_1(modm_1), x = a_2(modm_2), \ldots, x = a_n(modm_n)$

Steps

1. If $gcd(m_i, m_j) = 1$, then a solution for x exist
2. Calculate $M = m_1 m_2 m_3$; $M_1 = \frac{M}{m_1}$ $M_2 = \frac{M}{m_2}$ $M_3 = \frac{M}{m_3}$
3. Find M_1 mod M,M_2 mod M,M_3 mod M
4. Finally the solution is given as $X = (a_1 M_1 M_1^-1 + a_2 M_2 M_2^-1 + \ldots + a_n M_n M_n^-1)(mod M)$

Question

Solve the following system in Python -

$x \equiv 2(mod 3)$
$x \equiv 3(mod 4)$
$x \equiv 1(mod 5)$

In [3]:
```
# Chinese Remainder Theorem

from math import gcd

def mod_inverse(a,m):
    a = a % m
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None

def solve_linear_congruences(congruences):
    M = 1
    for _, modulus in congruences:
        M *= modulus

    # Compute the solution using the CRT formula
    x = 0
    for remainder, modulus in congruences:
        Mi = M // modulus
        Mi_inverse = mod_inverse(Mi, modulus)
        if Mi_inverse is None:
            raise ValueError(f"Modular inverse doesn't exist")
        x += remainder * Mi * Mi_inverse

    return x % M, M
```

In [4]:
```
# Example usage
if __name__ == "__main__":
    # Example: Solve the system of congruences
    # x ≡ 2 (mod 3)
    # x ≡ 3 (mod 4)
    # x ≡ 1 (mod 5)
    congruences = [(2,3), (3,4), (1,5)]

    solution, mod_product = solve_linear_congruences(congruences)
    print(f"The solution is x ≡ {solution} (mod {mod_product})")
```

```
The solution is x ≡ 11 (mod 60)
```

Question : Obtain the inverse of $7(mod 19)$

In [5]:
```
mod_inverse(7,19)
```

Out[5]: 11

Question

Solve the following system in Python -

$4x \equiv 5(mod 9)$
$2x \equiv 6(mod 19)$

In [6]:
```
if __name__ == "__main__":
    # Example: Solve the system of congruences
    # x ≡ 35 (mod 9)
    # x ≡ 60 (mod 19)
    congruences = [(35,9), (60,19)]

    solution, mod_product = solve_linear_congruences(congruences)
    print(f"The solution is x ≡ {solution} (mod {mod_product})")
```

```
The solution is x ≡ 98 (mod 171)
```

In [7]:
```
print("The x value is",98+171)
```

```
The x value is 269
```

Fermat's Little Theorem

$a^{p-1} \equiv 1(mod p)$, for any prime p

Question

Verify Fermat's Little Theorem when $a = 2$ and $p = 7$

In [8]:
```
a = int(input("Enter the value of a - "))
p = int(input("Enter the value of p - "))
print(a)
print(p)
print()

def femrmat_little(a,p):
    if (a**(p-1) - 1) % p == 0:
        print("Fermat Little Theorem is satisfied with a =",a,"and p =",p)
    else:
        print("Fermat Little Theorem is not satisfied with a =",a,"and p =",p)

femrmat_little(a,p)
```

```
2
7

Fermat Little Theorem is satisfied with a = 2 and p = 7
```

Question

Solve the following system in Python -

$x \equiv 5(mod 9)$
$x \equiv 6(mod 20)$

In [15]:
```
if __name__ == "__main__":
    # Example: Solve the system of congruences
    # x ≡ 5 (mod 9)
    # x ≡ 6 (mod 20)
    congruences = [(5,9), (6,20)]

    solution, mod_product = solve_linear_congruences(congruences)
    print(f"The solution is x ≡ {solution} (mod {mod_product})")
```

```
The solution is x ≡ 86 (mod 180)
```

Question

Solve the following system in Python -

$x \equiv 3(mod 7)$
$x \equiv 10(mod 14)$

In [9]:
```
if __name__ == "__main__":
    # Example: Solve the system of congruences
    # x ≡ 3 (mod 7)
    # x ≡ 10 (mod 14)
    congruences = [(3,7), (10,14)]

    solution, mod_product = solve_linear_congruences(congruences)
    print(f"The solution is x ≡ {solution} (mod {mod_product})")

# This system will have a solution but it is not showing since we are using the CRT Code. I
# then it will have a solution.
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[9], line 7
      1 if __name__ == "__main__":
      2     # Example: Solve the system of congruences
      3     # x ≡ 3 (mod 7)
      4     # x ≡ 10 (mod 14)
      5     congruences = [(3,7), (10,14)]
----> 7     solution, mod_product = solve_linear_congruences(congruences)
      8     print(f"The solution is x ≡ {solution} (mod {mod_product})")
     10 # This system will have a solution but it is not showing since we are using the CRT
Code. If we change the code,
     11 # then it will have a solution.

Cell In[3], line 23, in solve_linear_congruences(congruences)
     21     Mi_inverse = mod_inverse(Mi, modulus)
     22     if Mi_inverse is None:
---> 23         raise ValueError(f"Modular inverse doesn't exist")
     24     x += remainder * Mi * Mi_inverse
     26 return x % M, M

ValueError: Modular inverse doesn't exist
```

NOTE - What happens to the solution of the following system when m's are not co-prime?

$x \equiv a_1(modm_1)$
$x \equiv a_2(modm_2)$

ANS - If $gcd(m_1, m_2)|(|a_2 - a_1|)$, then the above system has a solution

In [10]:
```
from sympy.ntheory.modular import solve_congruence
```

```
solution = solve_congruence((3, 7), (10, 14))
print("Solution to the system of equations:", sum(solution))
```

```
Solution to the system of equations: 24
```

LAB 4 : Hashing Functions

11th February 2025

Apanvi Srivastava 2240250

Hashing Functions

hash() functions are deterministic in nature. They contain immutable objects.

Introduction

- Hasing is a method of onverting input data into a fixed length string.
- Used in data structure (hash tables), cryptography and integrity check.
- Properties of a good hash functions are:
  - Deterministic: Same input -> Same hash.
  - Fast computation: Should be quick to compute.
  - Uniform Distribution: Avoids collisions.
  - Irreversability: Cannot derive the input from the hash.
- Different Hashing Alogrithms
  - Uses **MD5, SHA-1, SHA-256 and SHA-512** to hash the same input.
  - $getattr(hashlib, algo)(text.encode()).hexdigest()$ dynamically calls each hashing function
- Hashing with Salt:
  - Generates a random salt using $os.urandom(16)$.
  - Uses $hashlib.pbkdf2_hmac()$ for iterative hashing (100,000 iterations for added security)
  - Returns both salt and hashed password (as hex) for safe storage.

These are useful in cryptographic applications like password storage, digital signatures and cryptographic integrity checks.

In [1]:
```
# initializing objects tuple are immutable
tuple_val = (1, 2, 3, 4, 5)

# list are mutable
list_val = [1, 2, 3, 4, 5]

# Printing the hash values.
# Notice exception when trying to convert mutable object
print("The tuple hash value is : " + str(hash(tuple_val)))
print("The list hash value is : " + str(hash(list_val)))
```

```
The tuple hash value is : -5659871693760987716
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[1], line 10
      7 # Printing the hash values.
      8 # Notice exception when trying to convert mutable object
      9 print("The tuple hash value is : " + str(hash(tuple_val)))
---> 10 print("The list hash value is : " + str(hash(list_val)))

TypeError: unhashable type: 'list'
```

In [2]:
```
### Sample inputs
```

```
print(hash(5))
print(hash(5.5))
print(hash("hello"))
print(hash("Hello!"))
```

```
5
1152921504606846981
8304794381031063615
8384647041295593868
```

In [3]:
```
import hashlib

# Example: Hash a string using SHA-256
def hash_string(input_string):
    return hashlib.sha256(input_string.encode()).hexdigest()

text1 = "Hello, Number Theory!"
hashed_value1 = hash_string(text1)
print("Original Text:",text1)
print("SHA-256 Hash:",hashed_value1)

print()

text2 = "I will complete the work"
hashed_value2 = hash_string(text2)
print("Original Text:",text2)
print("SHA-256 Hash:",hashed_value2)
```

```
Original Text: Hello, Number Theory!
SHA-256 Hash: 2d58bac87bdb44c0b2bea056bc04502222e24c3794a6cdcc73b90bb8c4482812

Original Text: I will complete the work
SHA-256 Hash: 83fb12e3933343da8ca88f83cbc89751257076582ff8964feb7c8d31a386b6a0
```

In [4]:
```
# Exploring Different Hasing Algorithms

import hashlib

def hash_text(text):
    """Returns hashes of a given text using different hashing algorithms."""
    hash_algorithms = ["md5", 'sha1', "sha256", "sha512"]

    hashes = {algo: getattr(hashlib, algo)(text.encode()).hexdigest() for algo in hash_algo

    for algo, hash_value in hashes.items():
        print(f"{algo.upper()} Hash: {hash_value}")

# Example text
text = "Hello, Number Theory!"
print("Original Text:", text)
hash_text(text)
```

```
Original Text: Hello, Number Theory!
MD5 Hash: e37706f0e8f1b9ca67dcaeba64bb6ce5
SHA1 Hash: bf71f12c093cf7b20901aafbad86aa42ba32d757
SHA256 Hash: 2d58bac87bdb44c0b2bea056bc04502222e24c3794a6cdcc73b90bb8c4482812
SHA512 Hash: 58892b0ebe3584a2a8ca46668bf876e9ae862a7ffbe0e856c5d59d0a8cbd1791da6516adbaa0f02
18c949df26085efc7bac88b17cec9ba3c607cb171c525217e
```

In [5]:
```
# Hashing with Salts (to prevent dictionary attacks)

import os
import hashlib

def hash_with_salt(password):
    """Hashes a password with a randomly generated salt to enhance security."""
    salt = os.urandom(16)  # Generate a 16-byte salt
    hash_object = hashlib.pbkdf2_hmac('sha256', password.encode(), salt, 100000)
```

```
    # Store salt and hash together for verification
    return salt.hex(), hash_object.hex()

# Example password
password = "securepassword123"
salt, hashed_password = hash_with_salt(password)

print("\nPassword Hashing with Salt:")
print(f"Salt: {salt}")
print(f"Hashed Password: {hashed_password}")
```

```
Password Hashing with Salt:
Salt: 272c951b8627917abd96e3856eda790a
Hashed Password: cd2de302245ae746d9743e262ba18de0797c896e7fef0da8453203c90ac8b5dc
```

LAB Assignment 2

11th February 2025

Apanvi Srivastava 2240250

QUESTIONS :

Write separate Python Functinos to check divisibility of a given number by:

- 7 (Difference between twice the unit digit and the remaining part should be divisible by 7)
- 9 (Sum of digits should be divisible by 9)
- 11 (Difference between the sums of the digits in the even places and odd places should either be 0 or divisible by 11)

In [1]:
```
def seven(num):
    u=num%10
    num//=10
    if(((2*u)-num)%7==0):
        return "yes"
    else:
        return "no"

def nine(num):
    s=0
    while(num>0):
        s+=num%10
        num//=10
    if(s%9==0):
        return "yes"
    else:
        return "no"

def eleven(num):
    count=1
    ec=0
    oc=0

    while(num>0):
        if(count%2==0):
            ec+=num%10
            num//=10
            count+=1
        else:
            oc+=num%10
            num//=10
            count+=1
    if((ec-oc)==0 or (ec-oc)%11==0):
        return "yes"
    else:
        return "no"


print("14 divisible by 7", seven(14))
print("18 divisible by 7",seven(18))
print("22 divisible by 7",seven(22))

print("14 divisible by 9",nine(14))
print("18 divisible by 9",nine(18))
print("22 divisible by 9",nine(22))
```

```
print("14 divisible by 11",eleven(14))
print("18 divisible by 11",eleven(18))
print("22 divisible by 11",eleven(22))

print("70828162317 divisibile by 7 ", seven(70828162317))
print("70828162317 divisibile by 9", nine(70828162317))
print("70828162317 divisibile by 11", eleven(70828162317))
```

```
14 divisible by 7 yes
18 divisible by 7 no
22 divisible by 7 no
14 divisible by 9 no
18 divisible by 9 yes
22 divisible by 9 no
14 divisible by 11 no
18 divisible by 11 no
22 divisible by 11 yes
70828162317 divisibile by 7  no
70828162317 divisibile by 9 yes
70828162317 divisibile by 11 yes
```

LAB 5 : Pseudorandom Numbers

21st February 2025

Apanvi Srivastava 2240250

Pseudo-Random Numbers

- A random number is an unpredictable number, while a pseudorandom number is generated using deterministic algorithms that mimic randomness.
- These numbers are used in:
  - Simulations (eg. Monte Carlo Simulations, statistical sampling)
  - Cryptography (for generating encryption keys, one-time pads)
  - Machine Learning (random weight initialization in neural networks)
- Psuedorandom numbers are generated using a seed, meaning if we set the same seed, we will get the same sequence of numbers.

In [5]:
```python
# basic implementation using random module

import random

# set a seed for reproducibility
random.seed(42)

# generate a random integer between 1 and 100
print("Random Integer (1-100):",random.randint(1,100))

# generate a random floating-point number between 0 an 1
print("Random Float (0-1):",random.random())

# generate a list of 5 random numbers
random_numbers = [random.randint(1,100) for _ in range(5)]
print("Random List:",random_numbers)


# shuffle a list randomly
sample_list = [1,2,3,4,5]
random.shuffle(sample_list)
print("Shuffled list:",sample_list)
```

```
Random Integer (1-100): 82
Random Float (0-1): 0.11133106816568039
Random List: [95, 36, 32, 29, 18]
Shuffled list: [4, 2, 3, 5, 1]
```

In [3]:
```python
# secure random numbers using secrets module

import secrets

# generate a cryptographically secure random integer
secure_number = secrets.randbelow(100)
print("Secure Random Number:",secure_number)

# generate a secure random hex token
token = secrets.token_hex(16)
print("Secure Token:", token)
```

```
Secure Random Number: 59
Secure Token: 1f49180c0c18c1b97fc971ab17bc2f88
```

Question : Generate a random password of a given length using random.choice()

In [7]:
```
import random
import string

def generate_password(length):
    characters = string.ascii_letters + string.digits + string.punctuation
    password = ''.join(random.choice(characters) for _ in range(length))
    return password

length = int(input("Enter the desired password length: "))
print("Desired Lenghts: ", length)
print("Generated Password:", generate_password(length))
```

```
Desired Lenghts:  10
Generated Password: |@`(1C5.Ja
```

LAB 6 : Even/Odd Parity

4th March 2025

Apanvi Srivastava 2240250

In [1]:
```python
# Checking Even/Odd Parity Manually

def check_parity(num):
    count_ones = bin(num).count('1')
    return "Even Parity" if count_ones %2 == 0 else "Odd Parity"

print(check_parity(7)) # Binary: 0111 -> Odd Parity
print(check_parity(8)) # Binary: 1000 -> Odd Parity
print(check_parity(3)) # Binary: 0011 -> Even Parity
```

```
Odd Parity
Odd Parity
Even Parity
```

In [2]:
```python
# Check the parity and determine the parity bits for even transmission
# 1. 57
# 2. 123
# 3. 1064

# My Code

def check_parity(num):
    count_ones = bin(num).count('1')
    return "Even Parity" if count_ones %2 == 0 else "Odd Parity"

num=[57,123,1064]
for i in num:
    if check_parity(i) == 'Odd Parity':
        bin1 = bin(i)[2:] # To remobe 0b from answer
        bin1=bin1+'1'
        print("The Even Parity for",i,"is", bin1)
    else:
        bin1 = bin(i)[2:]
        bin1=bin1+'0'
        print(i," is already an Even Parity: ",bin1)
```

```
57  is already an Even Parity:  1110010
123  is already an Even Parity:  11110110
The Even Parity for 1064 is 100001010001
```

In [3]:
```python
import random

def compute_parity_bit(data, even = True):
    # Computes and returns the parity bit for a given binary string
    count_ones = sum(int(bit) for bit in data)
    if even:
        return '0' if count_ones % 2 == 0 else '1' # Even Parity
    else:
        return '1' if count_ones % 2 == 0 else '0' # Odd Parity

def add_parity_bit(data, even = True):
    # Adds a parity bit to the data and returns the new binary string
    parity_bit = compute_parity_bit(data,even)
    return data + parity_bit
```

```
def check_parity(data, even = True):
    # Checks if the received data follows the expected parity
    count_ones = sum(int(bit) for bit in data)
    return (count_ones % 2 == 0) if even else (count_ones % 2 == 1)

# Example Transmission with Even Parity
data1 = '1001'
data2 = '1011'
print("Even Parity Transmisssion")
data1_with_parity = add_parity_bit(data1, even=True)
data2_with_parity = add_parity_bit(data2, even=True)
print(f"Sent(Even Parity):{data1_with_parity},{data2_with_parity}")
print(f"Received correctly: {check_parity(data1_with_parity)},{check_parity(data2_with_pari

# Example Transmission with Odd Parity
print("Odd Parity Transmisssion")
data1_with_parity_odd = add_parity_bit(data1, even=False)
data2_with_parity_odd = add_parity_bit(data2, even=False)
print(f"Sent(Even Parity):{data1_with_parity_odd},{data2_with_parity_odd}")
print(f"Received correctly: {check_parity(data1_with_parity_odd,even=False)},{check_parity(
```

```
Even Parity Transmisssion
Sent(Even Parity):10010,10111
Received correctly: True,True
Odd Parity Transmisssion
Sent(Even Parity):10011,10110
Received correctly: True,True
```
