# Exercícios de Matemática

[**Exercícios de Matemática**](#_hiu27f3rqc92)

**1**

[**Fatoração: Dado um número inteiro, decompor o número em seus fatores primos.**](#_hjnug0qt5bl1) 1

[Máximo Divisor Comum (MDC): Encontrar o máximo divisor comum de dois ou mais números.](#_rryodsj50ama) 2

[**Mínimo Múltiplo Comum (MMC): Encontrar o mínimo múltiplo comum de dois ou mais números.**](#_30gda6k7hy2x) 3

[Teorema Fundamental da Aritmética: Trabalhar com a representação única de um número em termos de seus fatores primos.](#_yewwsaoa2up0) 3

[**Congruências: Resolver problemas de congruência modular, como encontrar o resto de uma divisão por um determinado número.**](#_8yfw80n7txw8) 4

[Números Perfeitos: Encontrar números perfeitos e verificar se um número é perfeito.](#_j9le72ruf14) 5

[**Exercício verificar se um número é primo em python:**](#_rf1qlhrvf3rf) 6

[Crivo de Eratóstenes: Implementar o Crivo de Eratóstenes para encontrar números primos em um intervalo.](#_3344v6m6ady) 6

[**Teorema do Resto Chinês: Resolver problemas envolvendo o Teorema do Resto Chinês para sistemas de congruências.**](#_dkte4645te2b) 7

[Verificar Último Teorema de Fermat](#_xy575an27621) 8

## Fatoração: Dado um número inteiro, decompor o número em seus fatores primos.

Entrada:Digite um número inteiro: 6

Saida: Fatores primos: [2, 3]

def fatoracao\_primos(numero):

fatores = []

# Encontra os fatores 2 repetidamente

while numero % 2 == 0:

fatores.append(2)

numero //= 2

# Encontra os fatores primos ímpares

for i in range(3, int(numero \*\* 0.5) + 1, 2):

while numero % i == 0:

fatores.append(i)

numero //= i

# Se o número remanescente for maior que 2, é um fator primo

if numero > 2:

fatores.append(numero)

return fatores

# Solicita a entrada do usuário

numero = int(input("Digite um número inteiro: "))

# Chama a função para realizar a fatoração

resultado = fatoracao\_primos(numero)

# Exibe o resultado

print("Fatores primos:", resultado)

## Máximo Divisor Comum (MDC): Encontrar o máximo divisor comum de dois ou mais números.

Entrada:Digite os números separados por espaço: 6 9

Saída: Máximo Divisor Comum: 3

def mdc(a, b):

while b:

a, b = b, a % b

return a

def mdc\_lista(numeros):

resultado = numeros[0]

for i in range(1, len(numeros)):

resultado = mdc(resultado, numeros[i])

return resultado

# Solicita a entrada do usuário

entrada = input("Digite os números separados por espaço: ")

numeros = list(map(int, entrada.split()))

# Chama a função para encontrar o MDC dos números

resultado = mdc\_lista(numeros)

# Exibe o resultado

print("Máximo Divisor Comum:", resultado)

## Mínimo Múltiplo Comum (MMC): Encontrar o mínimo múltiplo comum de dois ou mais números.

Entrada: Digite os números separados por espaço: 6 9

Saida: Mínimo Múltiplo Comum: 18

def mdc(a, b):

while b:

a, b = b, a % b

return a

def mmc(a, b):

return (a \* b) // mdc(a, b)

def mmc\_lista(numeros):

resultado = numeros[0]

for i in range(1, len(numeros)):

resultado = mmc(resultado, numeros[i])

return resultado

# Solicita a entrada do usuário

entrada = input("Digite os números separados por espaço: ")

numeros = list(map(int, entrada.split()))

# Chama a função para encontrar o MMC dos números

resultado = mmc\_lista(numeros)

# Exibe o resultado

print("Mínimo Múltiplo Comum:", resultado)

## Teorema Fundamental da Aritmética: Trabalhar com a representação única de um número em termos de seus fatores primos.

Digite um número inteiro: 88

Representação única em termos de fatores primos: [(2, 3), (11, 1)]

def fatoracao\_primos(numero):

fatores = []

# Encontra os fatores 2 repetidamente

while numero % 2 == 0:

fatores.append(2)

numero //= 2

# Encontra os fatores primos ímpares

for i in range(3, int(numero \*\* 0.5) + 1, 2):

while numero % i == 0:

fatores.append(i)

numero //= i

# Se o número remanescente for maior que 2, é um fator primo

if numero > 2:

fatores.append(numero)

return fatores

def representacao\_unica(numero):

fatores = fatoracao\_primos(numero)

fatores\_unicos = list(set(fatores)) # Remove duplicatas

representacao = []

for fator in fatores\_unicos:

potencia = fatores.count(fator)

representacao.append((fator, potencia))

return representacao

# Solicita a entrada do usuário

numero = int(input("Digite um número inteiro: "))

# Chama a função para obter a representação única do número em termos de fatores primos

resultado = representacao\_unica(numero)

# Exibe o resultado

print("Representação única em termos de fatores primos:", resultado)

## Congruências: Resolver problemas de congruência modular, como encontrar o resto de uma divisão por um determinado número.

Digite o valor: 5

Digite o módulo: 2

O resto da congruência 5 % 2 é 1.

def resolver\_congruencia(valor, modulo):

resto = valor % modulo

return resto

# Solicita a entrada do usuário

valor = int(input("Digite o valor: "))

modulo = int(input("Digite o módulo: "))

# Chama a função para resolver a congruência

resultado = resolver\_congruencia(valor, modulo)

# Exibe o resultado

print(f"O resto da congruência {valor} % {modulo} é {resultado}.")

## Números Perfeitos: Encontrar números perfeitos e verificar se um número é perfeito.

Digite um limite superior: 50

Números perfeitos encontrados até 50 : [6, 28]

def calcular\_divisores(numero):

divisores = [1] # Começamos com 1, pois todo número é divisível por 1

for i in range(2, int(numero \*\* 0.5) + 1):

if numero % i == 0:

divisores.append(i)

if i != numero // i: # Evita adicionar o mesmo divisor duas vezes para números quadrados perfeitos

divisores.append(numero // i)

return divisores

def eh\_numero\_perfeito(numero):

divisores = calcular\_divisores(numero)

return sum(divisores) == numero

def encontrar\_numeros\_perfeitos(limit):

numeros\_perfeitos = []

for i in range(2, limit + 1):

if eh\_numero\_perfeito(i):

numeros\_perfeitos.append(i)

return numeros\_perfeitos

# Solicita a entrada do usuário

limite = int(input("Digite um limite superior: "))

# Encontra números perfeitos até o limite fornecido

numeros\_perfeitos = encontrar\_numeros\_perfeitos(limite)

# Exibe os números perfeitos encontrados

print("Números perfeitos encontrados até", limite, ":", numeros\_perfeitos)

## Exercício verificar se um número é primo em python:

def is\_prime(number):

if number <= 1:

return False

elif number <= 3:

return True

elif number % 2 == 0 or number % 3 == 0:

return False

i = 5

while i \* i <= number:

if number % i == 0 or number % (i + 2) == 0:

return False

i += 6

return True

# Exemplo de uso

num = int(input("Digite um número: "))

if is\_prime(num):

print(f"{num} é um número primo.")

else:

print(f"{num} não é um número primo.")

## Crivo de Eratóstenes: Implementar o Crivo de Eratóstenes para encontrar números primos em um intervalo.

Digite o limite superior: 50

Números primos até 50 : [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]

def crivo\_eratostenes(limite):

primos = [True] \* (limite + 1)

primos[0] = primos[1] = False

for num in range(2, int(limite\*\*0.5) + 1):

if primos[num]:

for mult in range(num\*num, limite + 1, num):

primos[mult] = False

numeros\_primos = [num for num, primo in enumerate(primos) if primo]

return numeros\_primos

# Solicita a entrada do usuário

limite = int(input("Digite o limite superior: "))

# Chama a função para encontrar números primos usando o Crivo de Eratóstenes

numeros\_primos = crivo\_eratostenes(limite)

# Exibe os números primos encontrados

print("Números primos até", limite, ":", numeros\_primos)

## Teorema do Resto Chinês: Resolver problemas envolvendo o Teorema do Resto Chinês para sistemas de congruências.

Digite o número de congruências: 2

Digite o 1º residuo: 5

Digite o 1º módulo: 6

Digite o 2º residuo: 4

Digite o 2º módulo: 2

A solução do sistema de congruências é: 10

def mdc\_estendido(a, b):

if a == 0:

return b, 0, 1

else:

mdc, x, y = mdc\_estendido(b % a, a)

return mdc, y - (b // a) \* x, x

def teorema\_resto\_chines(residuos, modulos):

n = len(modulos)

# Calcula o produto de todos os módulos

N = 1

for modulo in modulos:

N \*= modulo

# Calcula as constantes de cada equação

constantes = []

for modulo in modulos:

Ni = N // modulo

\_, inverso, \_ = mdc\_estendido(Ni, modulo)

constantes.append(inverso \* Ni)

# Calcula a solução usando o Teorema do Resto Chinês

x = sum(residuos[i] \* constantes[i] for i in range(n)) % N

return x

# Solicita a entrada do usuário

n = int(input("Digite o número de congruências: "))

residuos = []

modulos = []

for i in range(n):

residuo = int(input(f"Digite o {i+1}º residuo: "))

modulo = int(input(f"Digite o {i+1}º módulo: "))

residuos.append(residuo)

modulos.append(modulo)

# Chama a função para resolver o sistema de congruências usando o Teorema do Resto Chinês

resultado = teorema\_resto\_chines(residuos, modulos)

# Exibe o resultado

print("A solução do sistema de congruências é:", resultado)

## Verificar Último Teorema de Fermat

A equação 3^2 + 4^2 = 5^2 é verdadeira.

def verifica\_teorema\_fermat(a, b, c, n):

if a\*\*n + b\*\*n == c\*\*n:

return True

else:

return False

# Valores a serem verificados

a = 3

b = 4

c = 5

n = 2

# Verifica a equação para os valores dados

resultado = verifica\_teorema\_fermat(a, b, c, n)

# Exibe o resultado

if resultado:

print(f"A equação {a}^{n} + {b}^{n} = {c}^{n} é verdadeira.")

else:

print(f"A equação {a}^{n} + {b}^{n} = {c}^{n} não é verdadeira.")