Soluciones a los problemas: Clase 1 – Introducción y Fundamentos

Problemas: Clase 2 – Números primos

Verificar múltiplo de 5

```
# Leer número entero
n = int(input())
# Comprobar si es divisible por 5
if n % 5 == 0:
    print(n, "es divisible por 5")
else:
    print(n, "no es divisible por 5")
```

Divisibilidad por 2 y por 3

```
# Leer número entero
n = int(input())
# Comprobar divisibilidad por 2
if n % 2 == 0:
    print(n, "es divisible por 2")
else:
    print(n, "no es divisible por 2")
# Comprobar divisibilidad por 3
if n % 3 == 0:
    print(n, "es divisible por 3")
else:
    print(n, "no es divisible por 3")
# Comprobar divisibilidad por 6
if n % 6 == 0:
    print(n, "es divisible por 6")
else:
    print(n, "no es divisible por 6")
```

Divisores de un número 2

```
# Leer número entero
n = int(input())
# Mostrar los divisores de n
for i in range(1, n + 1):
    if n % i == 0:
        # end=" " Redefine end para que, en lugar de salto de línea, imprima un
```

```
espacio
        print(i, end=" ")
```

Comprobar si un número es primo

```
# Leer número entero
n = int(input())
# Suponemos que n es primo si es mayor que 1
mar = n > 1
# Comprobar si tiene algún divisor distinto de 1 y de sí mismo
for i in range(2, n):
    if n % i == 0:
        mar = False  # Si encontramos un divisor, no es primo
        break        # Salir del bucle
# Mostrar resultado
if mar:
    print("Es primo")
else:
    print("No es primo")
```

Comprobar si un número es primo (Optimizado)

```
# Leer número entero
n = int(input())
# Suponemos que n es primo si es mayor que 1
mar = n > 1
# Comprobar divisores desde 2 hasta la raíz cuadrada de n
for i in range(2, int(n**(1/2)) + 1):
    if n % i == 0:
        mar = False  # Si encontramos un divisor, no es primo
        break        # Salimos del bucle
# Mostrar resultado final
if mar:
    print("Es primo")
else:
    print("No es primo")
```

El número mágico de Williberto

```
# Función que determina si un número es primo
def es_primo(n):
    # Comprobar divisores desde 2 hasta la raíz cuadrada de n
    for i in range(2, int(n**(1/2)) + 1):
        if n % i == 0:
            return False  # Si tiene divisor, no es primo
    return True  # Si no se encontró divisor, es primo
```

```
# Leer número entero
n = int(input())
# Buscar el siguiente número primo
n += 1
while not es_primo(n):  # Repetir mientras n no sea primo
    n += 1
# Mostrar el siguiente número primo
print(n)
```

Números primos con la Criba de Eratóstenes

```
# Leer número entero
n = int(input())
# Crear una lista para marcar si un número es primo (True) o no (False)
dp = [True] * (n + 1)
dp[0] = dp[1] = False  # 0 y 1 no son primos
# Algoritmo de la Criba de Eratóstenes
for i in range(2, int(n**0.5) + 1):
    # Marcar como no primos los múltiplos de i
    for j in range(i * i, n + 1, i):
        dp[j] = False
# Mostrar todos los números primos hasta n
for i in range(n + 1):
    if dp[i]:
        # Imprime los primos en la misma línea separados por espacio
        print(i, end=" ")
```

Factorización en primos

```
# Definir el límite máximo para generar primos
N = 1000001
# Crear una lista para marcar si un número es primo (True) o no (False)
dp = [True] * (N + 1)
dp[0] = dp[1] = False  # 0 y 1 no son primos
# Algoritmo de la Criba de Eratóstenes para marcar números no primos
for i in range(2, int(N**0.5) + 1):
    for j in range(i * i, N, i):
        dp[j] = False  # Marcar múltiplos de i como no primos
# Guardar todos los números primos en la lista p
p = []
for i in range(2, N):
    if dp[i]:
        p.append(i)
# Inicializar posición para recorrer la lista de primos
pos = 0
# Leer número a factorizar
n = int(input())
# Factorización usando los primos generados
while pos < len(p) and n >= p[pos]:
```

```
    if n % p[pos] == 0:
        con = 0
        # Contar cuántas veces el primo divide al número
        while n % p[pos] == 0:
            n //= p[pos]
            con += 1
        print(f"{p[pos]}^{con}")  # Imprimir factor primo y su exponente
    pos += 1
# Si después de dividir por todos los primos aún queda un número >1,
# significa que es un primo mayor que sqrt(n)
if n > 1:
    print(f"{n}^1")
```

Contar primos en rangos

```
# Límite máximo para generar primos
N = 1000001
# Crear lista para marcar si un número es primo (True) o no (False)
dp = [True] * (N + 1)
dp[0] = dp[1] = False  # 0 y 1 no son primos
# Criba de Eratóstenes: marcar múltiplos de primos como no primos
for i in range(2, int(N**0.5) + 1):
    for j in range(i * i, N, i):
        dp[j] = False
# Crear lista acumulativa de cantidad de primos hasta cada índice
p = [0] * N
for i in range(2, N):
    p[i] = p[i - 1]       # Empezamos con el conteo anterior
    if dp[i]:
        p[i] += 1         # Incrementamos si i es primo
# Leer número de consultas q y rango máximo n
n, q = map(int, input().split())
# Procesar cada consulta
while q > 0:
    a, b = map(int, input().split())  # Leer rango [a, b]
    print(p[b] - p[a - 1])            # Mostrar cantidad de primos en ese rango
    q -= 1                             # Disminuir contador de consultas
```