

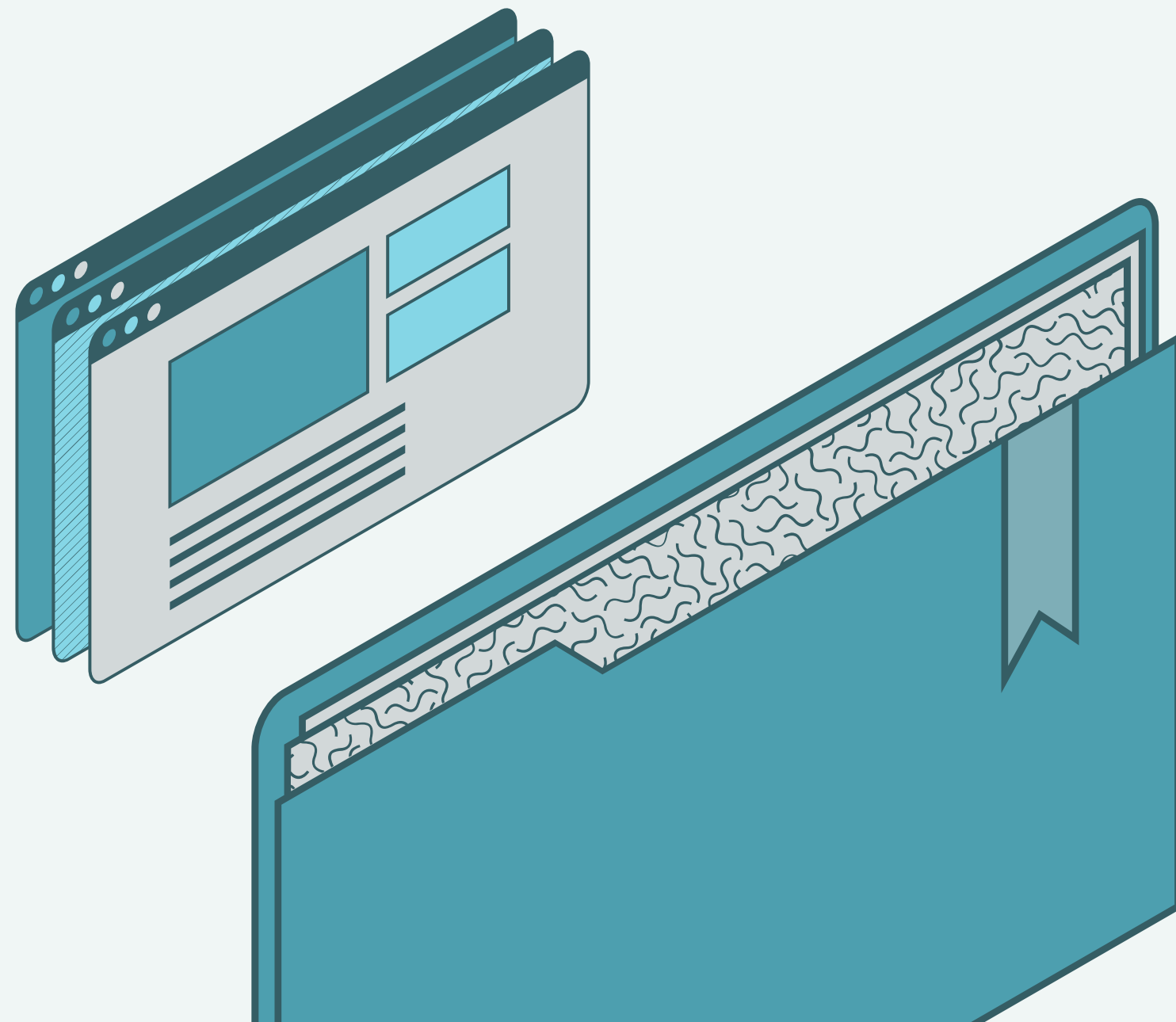
## TEMA 4

# APLICACIONES DE FÍSICA Y MATEMÁTICA CON PYTHON

## PROGRAMACIÓN BÁSICA



# ÍNDICE



---

## 1. Introducción

---

### 1.1 Equilibrio de Fuerzas

---

### 1.2 Movimiento Rectilíneo Uniforme (MRU)

---

### 1.3 Movimiento Rectilíneo Uniformemente Acelerado (MRUA)

---

### 1.4 Densidad

---

### 1.5 Calor Específico

---

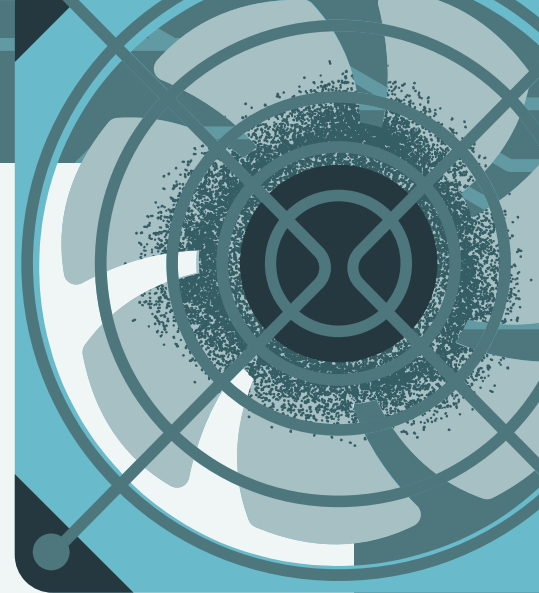
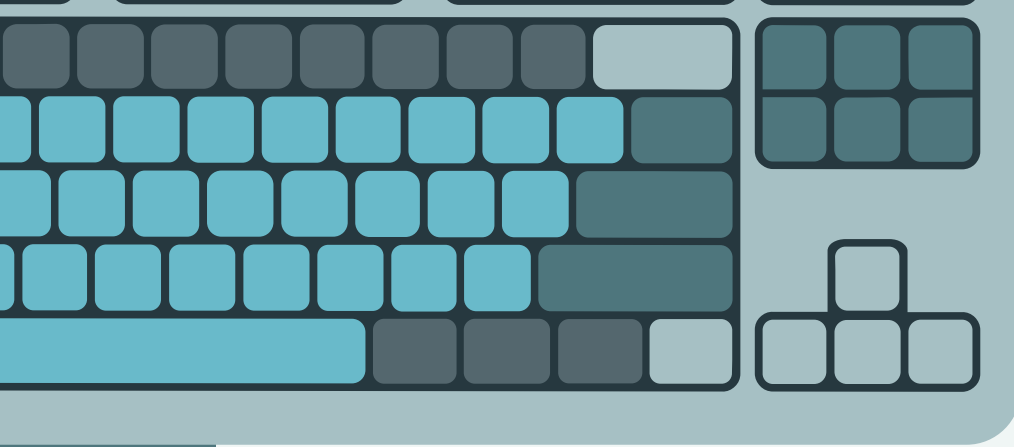
### 1.6 Equilibrio Térmico

---

## 2. Programas de Matemáticas

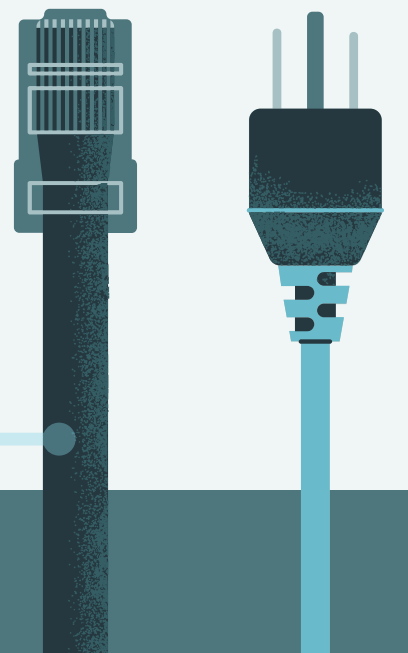
---

### 2.1 Secuencias Matemáticas: Fibonacci



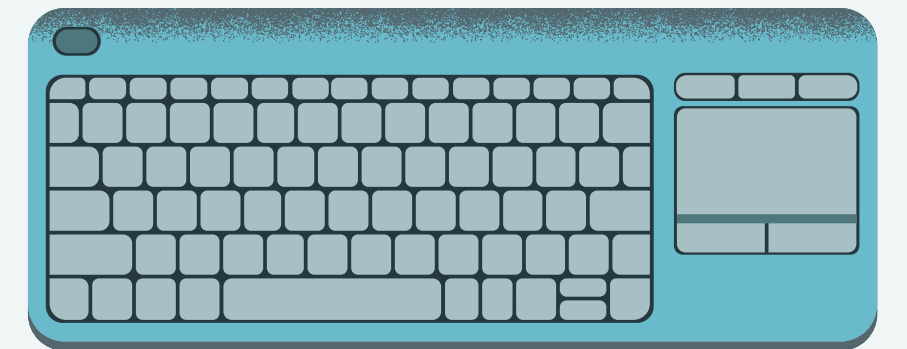
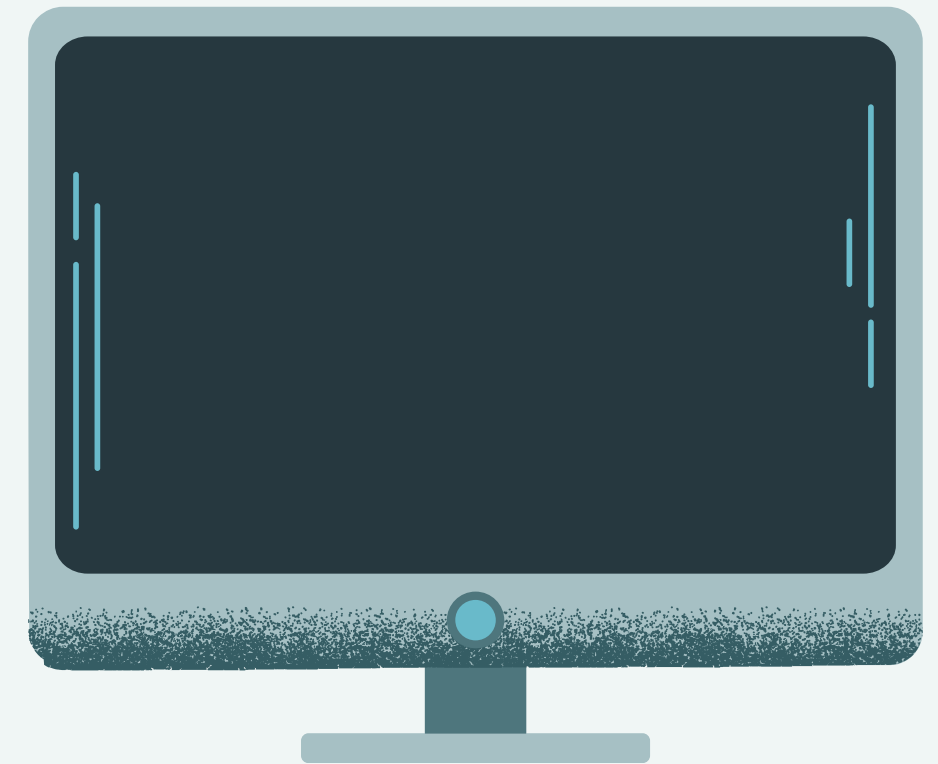
# OBJETIVO

Este material tiene como objetivo que el alumno construya programas para resolver problemas típicos de física y matemáticas, utilizando Python como lenguaje de programación.



# INTRODUCCIÓN

Python es una herramienta poderosa para aplicar conceptos de Física y Matemática de forma práctica. Gracias a su sencillez y a sus bibliotecas científicas, permite resolver ecuaciones, simular fenómenos y visualizar resultados. En esta presentación se mostrarán algunas aplicaciones que demuestran cómo la programación facilita el análisis y la comprensión de estos temas..



# 1.1- EQUILIBRIO DE FUERZAS

## Ejercicio

Realizar un programa que permita analizar sistemas de fuerzas actuando sobre un punto material. Calcula la fuerza resultante mediante descomposición vectorial y determina si el sistema esta en equilibrio estático.

## Ejemplo Práctico

Se analizan tres fuerzas de 10 N separadas en 120° (0°, 120° y 240°). El programa calcula que la fuerza resultante es cero, confirmando un equilibrio perfecto.

## Fundamento Teórico

Para que un cuerpo esté en equilibrio estático, la suma vectorial de todas las fuerzas debe ser cero:

$$\sum Fx = 0, \sum Fy = 0$$

Las componentes rectangulares de una fuerza se obtienen mediante:

$$Fx = F \cos(\theta), Fy = F \sin(\theta)$$

Magnitud de la fuerza resultante:

$$FR = \sqrt{(\sum Fx)^2 + (\sum Fy)^2}$$

## Resultados obtenidos

Al finalizar el programa debe ser capaz de realizar el cálculo para resolver problemas de ejercicios de equilibrio de fuerzas, teniendo como referencia el ejemplo práctico planteado.

## Salida del programa

El programa debe mostrar un resultado como el siguiente:

```
Fuerza resultante: 0.00 N
Angulo: 0.00 grados
Sistema en equilibrio: True
```

```
# Programa para calcular el equilibrio de fuerzas
import math

def equilibrio_fuerzas(fuerzas):
    # Parametros: lista de tuplas (magnitud, angulo_en_grados)
    # Retorna: fuerza_resultante, angulo, en_equilibrio

    suma_fx = 0 # Suma de componentes horizontales
    suma_fy = 0 # Suma de componentes verticales

    # Descomponer cada fuerza en componentes
    for magnitud, angulo_grados in fuerzas:
        angulo_rad = math.radians(angulo_grados) # Convertir a radianes
        fx = magnitud * math.cos(angulo_rad) # Componente horizontal
        fy = magnitud * math.sin(angulo_rad) # Componente vertical

        suma_fx += fx # Acumular componente x
        suma_fy += fy # Acumular componente y

    # Calcular magnitud de la fuerza resultante
    fuerza_resultante = math.sqrt(suma_fx**2 + suma_fy**2)

    # Calcular direccion de la fuerza resultante
    if fuerza_resultante > 0.01: # Si hay fuerza resultante
        angulo_resultante = math.degrees(math.atan2(suma_fy, suma_fx))
    else:
        angulo_resultante = 0

    # Verificar equilibrio (tolerancia de 0.01 N)
    en_equilibrio = fuerza_resultante < 0.01

    return fuerza_resultante, angulo_resultante, en_equilibrio

# Ejemplo: tres fuerzas de 10 N separadas 120 grados
fuerzas = [
    (10, 0), # 10 N hacia la derecha
    (10, 120), # 10 N a 120 grados
    (10, 240) # 10 N a 240 grados
]

Fr, angulo, equilibrio = equilibrio_fuerzas(fuerzas)
print(f"Fuerza resultante: {Fr:.2f} N")
print(f"Angulo: {angulo:.2f} grados")
print(f"Sistema en equilibrio: {equilibrio}")
```

## 1.2 - MOVIMIENTO RECTILÍNEO UNIFORME (MRU)

### Ejercicio

Realizar un programa que permita simular el movimiento de un objeto con velocidad constante y genera una tabla posición-tiempo.

### Ejemplo Práctico

Un automóvil viaja a 20 m/s durante 10 s. Se calcula y muestra la posición cada 0.5 s.

### Fundamento Teórico

El Movimiento Rectilíneo Uniforme (MRU) se caracteriza por una velocidad constante. La ecuación fundamental que relaciona posición, velocidad y tiempo es:

$$x = x_0 + vt$$

La velocidad se define como:

$$v = \frac{\Delta x}{\Delta t} = \textit{constante}$$

## Resultados obtenidos

Al finalizar el programa debe ser capaz de simular el movimiento de un objeto con velocidad constante y genera una tabla posición-tiempo, teniendo como referencia el ejemplo práctico planteado.

## Salida del programa

El programa debe mostrar un resultado como el siguiente:

| Tiempo (s) | Posicion (m) |
|------------|--------------|
| 0.0        | 0.0          |
| 0.5        | 10.0         |
| 1.0        | 20.0         |
| 1.5        | 30.0         |
| 2.0        | 40.0         |
| 2.5        | 50.0         |
| 3.0        | 60.0         |
| 3.5        | 70.0         |
| 4.0        | 80.0         |
| 4.5        | 90.0         |
| 5.0        | 100.0        |
| 5.5        | 110.0        |
| 6.0        | 120.0        |
| 6.5        | 130.0        |
| 7.0        | 140.0        |
| 7.5        | 150.0        |
| 8.0        | 160.0        |
| 8.5        | 170.0        |
| 9.0        | 180.0        |
| 9.5        | 190.0        |
| 10.0       | 200.0        |

```
# Programa para calcular MRU
def movimiento_rectilineo_uniforme(x0, v, t):
    # Parametros: x0=posicion inicial, v=velocidad, t=tiempo
    # Retorna: posicion final

    # Ecuacion del MRU:  $x = x_0 + v \cdot t$ 
    posicion = x0 + v * t

    return posicion

def simular_mru(x0, v, tiempo_total, dt=0.5):
    # Simula el MRU durante un intervalo de tiempo
    # Parametros: x0=posicion inicial, v=velocidad,
    # tiempo_total=tiempo de simulacion, dt=paso de tiempo

    datos = [] # Lista para almacenar resultados
    t = 0      # Tiempo inicial

    # Simular hasta el tiempo total
    while t <= tiempo_total:
        x = movimiento_rectilineo_uniforme(x0, v, t) # Calcular posicion
        datos.append((t, x)) # Guardar tiempo y posicion
        t += dt              # Incrementar tiempo

    return datos

# Ejemplo: Un auto que viaja a 20 m/s
x_inicial = 0 # posicion inicial en metros
velocidad = 20 # velocidad en m/s
tiempo = 10 # tiempo total en segundos

# Simular el movimiento
trayectoria = simular_mru(x_inicial, velocidad, tiempo)

print("Tiempo (s)\tPosicion (m)")
for t, x in trayectoria:
    print(f"{t:.1f}\t{x:.1f}")
```

## 1.3 - MOVIMIENTO RECTILÍNEO UNIFORME-MENTE ACCELERADO (MRUA)

### Ejercicio

Realizar un programa que permita calcular la posición y velocidad para un movimiento con aceleración constante.

### Ejemplo Práctico

Un vehículo acelera con  $a = 2 \text{ m/s}^2$  desde una velocidad inicial de  $v_0 = 5 \text{ m/s}$ .

El programa calcula la posición y la velocidad cada 2 s durante 10 s.

### Fundamento Teórico

El Movimiento Rectilíneo Uniformemente Acelerado (MRUA) se caracteriza por una aceleración constante. Las ecuaciones fundamentales son:

Ecuación de velocidad:

$$v = v_0 + at$$

Ecuación de posición:

$$x = x_0 + v_0t + \frac{1}{2}at^2$$

Ecuación independiente del tiempo:

$$v^2 = v_0^2 + 2a(x - x_0)$$

## Resultados obtenidos

Al finalizar el programa debe ser capaz de realizar el cálculo para resolver problemas de cálculo la posición y velocidad para un movimiento con aceleración constante, teniendo como referencia el ejemplo práctico planteado.

## Salida del programa

El programa debe mostrar un resultado como el siguiente:

| Tiempo (s) | Posicion (m) | Velocidad (m/s) |
|------------|--------------|-----------------|
| 0          | 0.00         | 5.00            |
| 2          | 14.00        | 9.00            |
| 4          | 36.00        | 13.00           |
| 6          | 66.00        | 17.00           |
| 8          | 104.00       | 21.00           |
| 10         | 150.00       | 25.00           |

```
# Programa para calcular MRUA
def mrua(v0, a, t):
    # Parametros: v0=velocidad inicial, a=aceleracion, t=tiempo
    # Retorna: posicion y velocidad finales

    # Ecuacion de posicion:  $x = v_0 * t + (1/2) * a * t^2$ 
    posicion = v0 * t + 0.5 * a * t**2

    # Ecuacion de velocidad:  $v = v_0 + a * t$ 
    velocidad = v0 + a * t

    return posicion, velocidad

# Ejemplo: Un coche que acelera desde 5 m/s
v_inicial = 5 # velocidad inicial en m/s
aceleracion = 2 # aceleracion en m/s^2

print("Tiempo (s)\tPosicion (m)\tVelocidad (m/s)")
for tiempo in range(0, 11, 2): # De 0 a 10 segundos cada 2 s
    x, v = mrua(v_inicial, aceleracion, tiempo)
    print(f"{tiempo}\t\t{x:.2f}\t\t{v:.2f}")
```

## 1.4 - DENSIDAD

### Ejercicio

Realizar un programa que permita calcular la densidad a partir de masa y volumen, e identifica el material comparándolo con una base de datos.

### Ejemplo Práctico

Un objeto de 2.7 kg y volumen 0.001 m<sup>3</sup> tiene densidad 2700 kg/m<sup>3</sup>, coincidiendo con el aluminio.

### Datos:

- Agua: 1000 kg/m<sup>3</sup>
- Aluminio: 2700 kg/m<sup>3</sup>
- Hierro: 7874 kg/m<sup>3</sup>
- Oro: 19320 kg/m<sup>3</sup>

### Fundamento Teórico

La densidad es una propiedad intensiva de la materia que relaciona la masa de un objeto con el volumen que ocupa. Se define como:

$$\rho = \frac{m}{V}$$

p = velocidad final (m/s)

m = masa del objeto (kg)

V = volumen del objeto (m<sup>3</sup>)

## Resultados obtenidos

Al finalizar el programa debe ser capaz de permitira calcular la densidad a partir de masa y volumen, e identifica el material, teniendo como referencia el ejemplo práctico planteado.

## Salida del programa

El programa debe mostrar un resultado como el siguiente:

```
Densidad calculada: 2700.00 kg/m^3
Posible material: Aluminio
```

```
# Programa para calcular densidad
def calcular_densidad(masa, volumen):
    # Parametros: masa (kg), volumen (m^3)
    # Retorna: densidad (kg/m^3)

    # Formula de densidad: rho = m / V
    densidad = masa / volumen

    return densidad

def identificar_material(densidad):
    # Identifica material segun su densidad
    # Parametros: densidad (kg/m^3)
    # Retorna: nombre del material mas cercano

    # Diccionario de densidades de materiales comunes
    materiales = {
        'Aluminio': 2700,
        'Hierro': 7874,
        'Cobre': 8960,
        'Plomo': 11340,
        'Agua': 1000,
        'Madera': 600
    }

    # Buscar el material mas cercano
    material_cercano = None
    diferencia_minima = float('inf')

    for nombre, dens_ref in materiales.items():
        diferencia = abs(densidad - dens_ref) # Calcular diferencia
        if diferencia < diferencia_minima: # Si es mas cercano
            diferencia_minima = diferencia
            material_cercano = nombre

    return material_cercano

# Ejemplo: Calcular densidad de un objeto
masa_obj = 2.7 # kg
volumen_obj = 0.001 # m^3 (1 litro)

dens = calcular_densidad(masa_obj, volumen_obj)
material = identificar_material(dens)

print(f"Densidad calculada: {dens:.2f} kg/m^3")
print(f"Posible material: {material}")
```

# 1.5 - CALOR ESPECÍFICO

## Ejercicio

Realizar un programa que permitan calcular el calor necesario para modificar la temperatura de una sustancia.

## Ejemplo Práctico

Realizar los cálculos necesarios para calentar 2 kg de agua de 20°C a 100°C, requiere aproximadamente 670 kJ.

## Datos:

- Agua:  $c = 4186 \text{ J/(kg} \cdot \text{K)}$
- Hierro:  $c = 450 \text{ J/(kg} \cdot \text{K)}$
- Cobre:  $c = 385 \text{ J/(kg} \cdot \text{K)}$
- Aluminio:  $c = 900 \text{ J/(kg} \cdot \text{K)}$

## Fundamento Teórico

El calor específico es la cantidad de energía necesaria para elevar la temperatura de 1 kg de una sustancia en 1 grado Celsius (o Kelvin). La ecuación fundamental de calorimetría es:

$$Q = mc\Delta T$$

$Q$  = calor transferido (J)

$m$  = masa de la sustancia (kg)

$c$  = calor específico (J/(kg · K))

$\Delta T = T_f - T_i$  = cambio de temperatura (K o °C)

## Resultados obtenidos

Al finalizar el programa debe ser capaz de realizar el cálculo para resolver problemas de ejercicios de equilibrio de fuerzas, teniendo como referencia el ejemplo práctico planteado.

## Salida del programa

El programa debe mostrar un resultado como el siguiente:

```
Masa de agua: 2.0 kg
Cambio de temperatura: 80 grados C
Calor necesario: 669760.00 J
Calor necesario: 669.76 kJ
```

```
# Programa para calcular calor especifico
def calcular_calor(masa, calor_especifico, delta_T):
    # Parametros: masa (kg), calor_especifico (J/(kg*K)), delta_T (K)
    # Retorna: calor transferido (J)

    # Formula: Q = m * c * delta_T
    calor = masa * calor_especifico * delta_T

    return calor

# Ejemplo: Calentar agua
masa_agua = 2.0      # kg
c_agua = 4186        # J/(kg*K) - calor especifico del agua
temp_inicial = 20    # grados Celsius
temp_final = 100     # grados Celsius

# Calcular cambio de temperatura
delta_temperatura = temp_final - temp_inicial

# Calcular calor necesario
Q = calcular_calor(masa_agua, c_agua, delta_temperatura)

print(f"Masa de agua: {masa_agua} kg")
print(f"Cambio de temperatura: {delta_temperatura} grados C")
print(f"Calor necesario: {Q:.2f} J")
print(f"Calor necesario: {Q/1000:.2f} kJ")
```

# 1.6 EQUILIBRIO TÉRMICO

## Ejercicio

Realiza un programa que permita calcular la temperatura final de dos cuerpos que se mezclan térmicamente.

## Ejemplo Práctico

1 kg de agua a 80°C se mezcla con 2 kg a 20°C

Temperatura de equilibrio:

40°C

## Fundamento Teórico

Cuando dos cuerpos a diferentes temperaturas se ponen en contacto, hay transferencia de calor del cuerpo más caliente al más frío hasta alcanzar el equilibrio térmico. Por el principio de conservación de la energía:

$$Q_{cedido} = Q_{ganado}$$

Para dos cuerpos en contacto:

$$T_{eq} = \frac{m_1 c_1 T_1 + m_2 c_2 T_2}{m_1 c_1 + m_2 c_2}$$

$T_{eq}$  = temperatura de equilibrio (K o °C)

$m_1, m_2$  = masas de los cuerpos (kg)

$c_1, c_2$  = calores específicos (J/(kg · K))

$T_1, T_2$  = temperaturas iniciales (K o °C)

## Resultados obtenidos

Al finalizar el programa debe ser capaz de calcular la temperatura final de dos cuerpos que se mezclan térmicamente, dados los datos del ejemplo práctico.

## Salida del programa

El programa debe mostrar un resultado como el siguiente:

```
Temperatura de equilibrio: 40.00 grados C
Calor cedido: 167440.00 J
Calor ganado: 167440.00 J
Verificación (deben ser iguales): 0.00 J
```

```
# Programa para calcular equilibrio termico
def temperatura_equilibrio(m1, c1, T1, m2, c2, T2):
    # Calcula temperatura de equilibrio entre dos cuerpos
    # Parámetros: masas (m1, m2), calores específicos (c1, c2),
    # temperaturas iniciales (T1, T2)
    # Retorna: temperatura de equilibrio

    # Formula: m1*c1*(Teq - T1) + m2*c2*(Teq - T2) = 0
    # Despejando Teq:
    numerador = m1 * c1 * T1 + m2 * c2 * T2
    denominador = m1 * c1 + m2 * c2

    T_eq = numerador / denominador # Temperatura de equilibrio

    return T_eq

def suma_calores(m1, c1, T1, m2, c2, T2):
    # Calcula intercambio de calor entre dos cuerpos
    # Retorna: calor_cedido, calor_ganado, temperatura_equilibrio

    # Calcular temperatura de equilibrio
    T_eq = temperatura_equilibrio(m1, c1, T1, m2, c2, T2)

    # Calcular calor cedido (cuerpo caliente)
    if T1 > T2:
        Q_cedido = m1 * c1 * (T1 - T_eq)
        Q_ganado = m2 * c2 * (T_eq - T2)
    else:
        Q_cedido = m2 * c2 * (T2 - T_eq)
        Q_ganado = m1 * c1 * (T_eq - T1)

    return Q_cedido, Q_ganado, T_eq

# Ejemplo: Mezclar agua caliente y fría
m_caliente = 1.0 # kg de agua caliente
T_caliente = 80 # grados Celsius
m_fria = 2.0 # kg de agua fría
T_fria = 20 # grados Celsius
c_agua = 4186 # J/(kg*K)

# Calcular equilibrio
Q_ced, Q_gan, T_final = suma_calores(
    m_caliente, c_agua, T_caliente,
    m_fria, c_agua, T_fria
)

print(f"Temperatura de equilibrio: {T_final:.2f} grados C")
print(f"Calor cedido: {Q_ced:.2f} J")
print(f"Calor ganado: {Q_gan:.2f} J")
print(f"Verificación (deben ser iguales): {abs(Q_ced - Q_gan):.2f} J")
```

## 2.1 SECUENCIAS MATEMATICAS: FIBONACCI

### Problema

Realiza un programa que permita generar los primeros  $n$  términos de la secuencia de Fibonacci y calcula la razón entre términos consecutivos.

### Ejemplo Práctico

El programa genera los primeros 15 términos y muestra la aproximación al número áureo.

### Fundamento Teórico

La sucesión de Fibonacci es una secuencia matemática definida por la siguiente relación de recurrencia:

$$F(n) = F(n - 1) + F(n - 2)$$

con valores iniciales:

$$F(0) = 0, F(1) = 1$$

Los primeros términos son: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Una propiedad interesante es que la razón entre términos consecutivos converge al número áureo:

$$\lim_{n \rightarrow \infty} \frac{F(n+1)}{F(n)} = \varphi = \frac{1 + \sqrt{5}}{2} \approx 1,618$$

## Resultados obtenidos

Al finalizar el programa debe ser capaz de realizar el cálculo para generar los primeros n términos de la secuencia de Fibonacci y calcular la razón entre términos consecutivos.

## Salida del programa

El programa debe mostrar un resultado como el siguiente:

```
Primeros 15 numeros de Fibonacci:
F(0) = 0
F(1) = 1
F(2) = 1
F(3) = 2
F(4) = 3
F(5) = 5
F(6) = 8
F(7) = 13
F(8) = 21
F(9) = 34
F(10) = 55
F(11) = 89
F(12) = 144
F(13) = 233
F(14) = 377

Razon aurea aproximada: 1.618026
Valor exacto: 1.618034
```

```
# Secuencia de Fibonacci
def fibonacci(n):
    # Genera los primeros n numeros de Fibonacci
    # Parametros: n = cantidad de terminos a generar
    # Retorna: lista con los numeros de Fibonacci

    secuencia = [] # Lista para guardar la secuencia

    # Casos base
    if n >= 1:
        secuencia.append(0) # F(0) = 0
    if n >= 2:
        secuencia.append(1) # F(1) = 1

    # Generar terminos siguientes
    for i in range(2, n):
        # Cada termino es la suma de los dos anteriores
        siguiente = secuencia[i-1] + secuencia[i-2]
        secuencia.append(siguiente)

    return secuencia

# Ejemplo
n_terminos = 15
fib_seq = fibonacci(n_terminos)

print(f"Primeros {n_terminos} numeros de Fibonacci:")
for i, valor in enumerate(fib_seq):
    print(f"F({i}) = {valor}")

# Calcular razon aurea aproximada
if n_terminos > 2:
    razon = fib_seq[-1] / fib_seq[-2]
    print(f"\nRazon aurea aproximada: {razon:.6f}")
    print(f"Valor exacto: {(1 + 5**0.5) / 2:.6f}")
```