Ejercicio: Generador de Informes de Ventas

Escenario Empresarial: Imagina que trabajas para una empresa minorista que vende gadgets electrónicos. Tu tarea es crear un programa en Python que genere informes de ventas basados en varios parámetros. Necesitas definir una función que llame a varias funciones auxiliares para procesar y devolver un resumen detallado de los datos de ventas.

Instrucciones:

1. Define una función llamada `generate_sales_report` con los siguientes parámetros:
   - `store_name` (str): El nombre de la tienda minorista.
   - `date` (str): La fecha del informe de ventas.
   - `*sales_data` (tuplas): Un número variable de tuplas donde cada tupla representa la venta de un gadget electrónico y contiene los siguientes elementos:
     - Nombre del artículo (str)
     - Precio unitario (float)
     - Cantidad vendida (int)
2. Utiliza **anotaciones de tipo de datos** para especificar los tipos de datos esperados para cada parámetro.
3. Incluye un **docstring** que explique qué hace la función y describa cada parámetro.
4. Implementa la función `generate_sales_report` de manera que llame a las siguientes funciones auxiliares para calcular y devolver la siguiente información:
   - **`calculate_total_sales(sales_data: List[Tuple[str, float, int]], include_tax: bool) -> float`**: Calcula el total de ventas de todos los artículos y, opcionalmente, incluye un impuesto del 10%.
   - **`calculate_average_price(sales_data: List[Tuple[str, float, int]]) -> float`**: Calcula el precio unitario promedio de los artículos vendidos.
   - **`count_total_items(sales_data: List[Tuple[str, float, int]]) -> int`**: Cuenta el número total de artículos vendidos.
   - **`find_min_max_price_item(sales_data: List[Tuple[str, float, int]]) -> Tuple[Tuple[str, float], Tuple[str, float]]`**: Encuentra el artículo más caro y el más barato vendido, devolviendo sus nombres y precios.
   - **`generate_report(store_name: str, date: str, total_sales: float, average_price: float, total_items: int, min_item: Tuple[str, float], max_item: Tuple[str, float], currency: str) -> str`**: Genera el informe final de ventas en formato de texto.
5. Usa el parámetro `**kwargs` en `generate_sales_report` para permitir argumentos de palabra clave opcionales. Incluye los siguientes argumentos de palabra clave opcionales:

- `include_tax` (bool, predeterminado False): Si es True, calcula la cantidad total de ventas incluyendo un impuesto sobre las ventas del 10%.
- `currency` (str, predeterminado 'USD'): El símbolo de moneda a usar en el informe.

6. Crea un informe de ventas de ejemplo usando la función `generate_sales_report`. Incluye una mezcla de diferentes gadgets, cantidades y precios en los datos de ventas.

7. Imprime el informe de ventas, incluyendo toda la información calculada, en un formato amigable para el usuario.

8. Prueba la función con diferentes conjuntos de datos de ventas y argumentos de palabra clave opcionales para asegurarte de que funcione como se espera.

Ejemplo de Uso:

```
sales_report = generate_sales_report(
    "Gadget Store",
    "2023-09-15",
    ("Phone", 499.99, 10),
    ("Tablet", 299.99, 5),
    ("Laptop", 899.99, 3),
    ("Smart Watch", 199.99, 15),
    ("Headphones", 149.99, 20),
    ("Camera", 499.99, 2),
    ("Drone", 1599.99, 1),
    ("Speaker", 99.99, 7),
    include_tax=True,
    currency='EUR'
)
print(sales_report)
```

Salida Esperada:

```
Informe de Ventas de Gadget Store
Fecha: 2023-09-15
Total de Ventas: EUR20349.31
Precio Promedio: EUR293.64
Total de Artículos Vendidos: 63
Artículo Más Barato: Speaker a EUR99.99
Artículo Más Caro: Drone a EUR1599.99

from typing import List, Tuple

# Función para calcular el total de ventas
def calculate_total_sales(sales_data: List[Tuple[str, float, int]],
include_tax: bool) -> float:
    """
    Calcula el total de ventas de todos los artículos y,
```

```
opcionalmente, incluye un impuesto del 10%.

    Argumentos:
        sales_data (List[Tuple[str, float, int]]): Datos de ventas
donde cada tupla contiene el nombre del artículo,
            el precio unitario y la cantidad vendida.
        include_tax (bool): Si True, incluye un impuesto del 10% en el
total.

    Devuelve:
        float: El total de ventas (con o sin impuesto).
    """
    total = sum(price * quantity for _, price, quantity in sales_data)
    if include_tax:
        total *= 1.10  # Agrega el 10% de impuestos
    return total

# Función para calcular el precio promedio
def calculate_average_price(sales_data: List[Tuple[str, float, int]])
-> float:
    """
    Calcula el precio unitario promedio de los artículos vendidos.

    Argumentos:
        sales_data (List[Tuple[str, float, int]]): Datos de ventas
donde cada tupla contiene el nombre del artículo,
            el precio unitario y la cantidad vendida.

    Devuelve:
        float: El precio promedio de los artículos.
    """
    total_items = sum(quantity for _, _, quantity in sales_data)
    if total_items == 0:
        return 0.0
    total_price = sum(price * quantity for _, price, quantity in
sales_data)
    return total_price / total_items

# Función para contar el total de artículos
def count_total_items(sales_data: List[Tuple[str, float, int]]) ->
int:
    """
    Cuenta el número total de artículos vendidos.

    Argumentos:
        sales_data (List[Tuple[str, float, int]]): Datos de ventas
donde cada tupla contiene el nombre del artículo,
            el precio unitario y la cantidad vendida.

    Devuelve:
```

```
        int: El total de artículos vendidos.
    """
    return sum(quantity for _, _, quantity in sales_data)

# Función para encontrar el artículo más caro y el más barato
def find_min_max_price_item(sales_data: List[Tuple[str, float, int]])
-> Tuple[Tuple[str, float], Tuple[str, float]]:
    """
    Encuentra el artículo más caro y el más barato vendido.

    Argumentos:
        sales_data (List[Tuple[str, float, int]]): Datos de ventas
donde cada tupla contiene el nombre del artículo,
            el precio unitario y la cantidad vendida.

    Devuelve:
        Tuple[Tuple[str, float], Tuple[str, float]]: Tuplas que
contienen el nombre y precio del artículo más barato
            y del más caro.
    """
    min_item = min(sales_data, key=lambda x: x[1])  # Artículo más
barato
    max_item = max(sales_data, key=lambda x: x[1])  # Artículo más
caro
    return (min_item[0], min_item[1]), (max_item[0], max_item[1])

# Función para generar el informe
def generate_report(store_name: str, date: str, total_sales: float,
average_price: float,
                    total_items: int, min_item: Tuple[str, float],
max_item: Tuple[str, float],
                    currency: str) -> str:
    """
    Genera el informe final de ventas en formato de texto.

    Argumentos:
        store_name (str): El nombre de la tienda.
        date (str): La fecha del informe.
        total_sales (float): El total de ventas.
        average_price (float): El precio promedio.
        total_items (int): El total de artículos vendidos.
        min_item (Tuple[str, float]): El artículo más barato.
        max_item (Tuple[str, float]): El artículo más caro.
        currency (str): El símbolo de la moneda.

    Devuelve:
        str: Informe de ventas formateado.
    """
    report = f"Informe de Ventas de {store_name}\n"
    report += f"Fecha: {date}\n"
```

```
    report += f"Total de Ventas: {currency}{total_sales:.2f}\n"
    report += f"Precio Promedio: {currency}{average_price:.2f}\n"
    report += f"Total de Artículos Vendidos: {total_items}\n"
    report += f"Artículo Más Barato: {min_item[0]} a {currency}
{min_item[1]:.2f}\n"
    report += f"Artículo Más Caro: {max_item[0]} a {currency}
{max_item[1]:.2f}\n"
    return report

# Función principal para generar el informe de ventas
def generate_sales_report(store_name: str, date: str, *sales_data:
Tuple[str, float, int],
                          **kwargs) -> str:
    """
    Genera un informe de ventas basado en los datos proporcionados.

    Argumentos:
        store_name (str): El nombre de la tienda minorista.
        date (str): La fecha del informe de ventas.
        *sales_data (Tuple[str, float, int]): Tuplas con información
de ventas.
        **kwargs: Argumentos de palabra clave opcionales.
            include_tax (bool): Si es True, incluye un impuesto sobre
las ventas.
            currency (str): El símbolo de moneda a usar en el informe.

    Devuelve:
        str: Informe de ventas.
    """
    include_tax = kwargs.get('include_tax', False)
    currency = kwargs.get('currency', 'USD')

    total_sales = calculate_total_sales(sales_data, include_tax)
    average_price = calculate_average_price(sales_data)
    total_items = count_total_items(sales_data)
    min_item, max_item = find_min_max_price_item(sales_data)

    return generate_report(store_name, date, total_sales,
average_price, total_items, min_item, max_item, currency)

# Ejemplo de uso
sales_report = generate_sales_report(
    "Gadget Store",
    "2023-09-15",
    ("Phone", 499.99, 10),
    ("Tablet", 299.99, 5),
    ("Laptop", 899.99, 3),
    ("Smart Watch", 199.99, 15),
    ("Headphones", 149.99, 20),
    ("Camera", 499.99, 2),
```

```
    ("Drone", 1599.99, 1),
    ("Speaker", 99.99, 7),
    include_tax=True,
    currency='EUR'
)

print(sales_report)
```

```
Informe de Ventas de Gadget Store
Fecha: 2023-09-15
Total de Ventas: EUR20349.31
Precio Promedio: EUR293.64
Total de Artículos Vendidos: 63
Artículo Más Barato: Speaker a EUR99.99
Artículo Más Caro: Drone a EUR1599.99
```

Ejercicio: Generador de Informes de Registro de Estudiantes

Escenario Escolar: Imagina que trabajas en una escuela y tu tarea es crear un programa en Python que genere informes de registro de estudiantes basados en varios parámetros. Necesitas definir una función que llame a varias funciones auxiliares para procesar y devolver un resumen detallado de los datos de los estudiantes registrados.

Instrucciones:

1. Define una función llamada `generate_student_report` con los siguientes parámetros:
   - `school_name` (str): El nombre de la escuela.
   - `date` (str): La fecha del informe de registro.
   - `*student_data` (tuplas): Un número variable de tuplas donde cada tupla representa a un estudiante y contiene los siguientes elementos:
     - Nombre del estudiante (str)
     - Edad del estudiante (int)
     - Grado (str)
2. Utiliza **anotaciones de tipo de datos** para especificar los tipos de datos esperados para cada parámetro.
3. Incluye un **docstring** que explique qué hace la función y describa cada parámetro.
4. Implementa la función `generate_student_report` de manera que llame a las siguientes funciones auxiliares para calcular y devolver la siguiente información:

- **`calculate_total_students(student_data: List[Tuple[str, int, str]]) -> int`**:
  - Cuenta el número total de estudiantes registrados.
  - Debe incluir gestión de errores usando `assert` para asegurarse de que `student_data` no esté vacío.
- **`calculate_average_age(student_data: List[Tuple[str, int, str]]) -> float`**:
  - Calcula la edad promedio de los estudiantes.
  - Debe manejar excepciones utilizando `try`, `except`, `else` y `finally`.
- **`find_min_max_age_student(student_data: List[Tuple[str, int, str]]) -> Tuple[Tuple[str, int], Tuple[str, int]]`**:
  - Encuentra el estudiante más joven y el más viejo, devolviendo sus nombres y edades.
  - Debe lanzar una excepción si no se encuentran estudiantes usando `raise`.
- **`generate_report(school_name: str, date: str, total_students: int, average_age: float, min_student: Tuple[str, int], max_student: Tuple[str, int]) -> str`**:
  - Genera el informe final de registro de estudiantes en formato de texto.
  - Debe incluir gestión de errores para asegurar que los datos sean válidos, usando `try` y `except`.

5. Usa el parámetro `**kwargs` en `generate_student_report` para permitir argumentos de palabra clave opcionales. Incluye los siguientes argumentos de palabra clave opcionales:
   - `grade_filter` (str, predeterminado None): Si se proporciona, filtra el informe para incluir solo estudiantes de ese grado.
6. Crea un informe de registro de ejemplo usando la función `generate_student_report`. Incluye una mezcla de diferentes estudiantes, edades y grados en los datos.
7. Imprime el informe de registro, incluyendo toda la información calculada, en un formato amigable para el usuario.
8. Prueba la función con diferentes conjuntos de datos de estudiantes y argumentos de palabra clave opcionales para asegurarte de que funcione como se espera.

Ejemplo de Uso:

```
student_report = generate_student_report(
    "Escuela Primaria",
    "2023-09-15",
    ("Juan Pérez", 10, "5to"),
    ("María López", 9, "4to"),
    ("Carlos Gómez", 11, "5to"),
    ("Ana Torres", 8, "3ro"),
    ("Luis Martínez", 10, "5to"),
```

```
    ("Laura Jiménez", 7, "2do"),
    ("Pedro González", 12, "6to"),
    ("Sofía Fernández", 9, "4to"),
    grade_filter="5to"
)
print(student_report)
```

Salida Esperada:

```
Informe de Registro de Estudiantes de Escuela Primaria
Fecha: 2023-09-15
Total de Estudiantes: 3
Edad Promedio: 10.0
Estudiante Más Joven: Ana Torres, 8 años
Estudiante Más Viejo: Carlos Gómez, 11 años

from typing import List, Tuple, Optional

def generate_student_report(school_name: str, date: str,
*student_data: Tuple[str, int, str], **kwargs) -> str:
    """
    Genera un informe de registro de estudiantes.

    Args:
        school_name (str): El nombre de la escuela.
        date (str): La fecha del informe de registro.
        *student_data (Tuple[str, int, str]): Datos de los estudiantes
en tuplas.
        **kwargs: Palabras clave opcionales.
            - grade_filter (str, predeterminado None): Filtro para
incluir solo estudiantes de un grado específico.

    Returns:
        str: Informe de registro de estudiantes.
    """

    # Filtrar por grado si se proporciona
    grade_filter = kwargs.get('grade_filter', None)
    if grade_filter:
        student_data = [student for student in student_data if
student[2] == grade_filter]

    total_students = calculate_total_students(student_data)
    average_age = calculate_average_age(student_data)
    min_student, max_student = find_min_max_age_student(student_data)

    report = generate_report(school_name, date, total_students,
average_age, min_student, max_student)
    return report
```

```python
def calculate_total_students(student_data: List[Tuple[str, int, str]])
-> int:
    """
    Cuenta el número total de estudiantes registrados.

    Args:
        student_data (List[Tuple[str, int, str]]): Datos de los
estudiantes.

    Returns:
        int: Número total de estudiantes.

    Raises:
        AssertionError: Si student_data está vacío.
    """
    assert student_data, "No hay datos de estudiantes para procesar."
    return len(student_data)

def calculate_average_age(student_data: List[Tuple[str, int, str]]) ->
float:
    """
    Calcula la edad promedio de los estudiantes.

    Args:
        student_data (List[Tuple[str, int, str]]): Datos de los
estudiantes.

    Returns:
        float: Edad promedio de los estudiantes.

    Raises:
        ValueError: Si no se puede calcular la edad promedio.
    """
    try:
        if not student_data:
            raise ValueError("No hay estudiantes para calcular la edad
promedio.")
        total_age = sum(student[1] for student in student_data)
        average_age = total_age // len(student_data)
    except ZeroDivisionError:
        raise ValueError("No hay estudiantes para calcular la edad
promedio.")
    return average_age

def find_min_max_age_student(student_data: List[Tuple[str, int, str]])
-> Tuple[Tuple[str, int], Tuple[str, int]]:
    """
    Encuentra el estudiante más joven y el más viejo.

    Args:
```

```
        student_data (List[Tuple[str, int, str]]): Datos de los
estudiantes.

    Returns:
        Tuple[Tuple[str, int], Tuple[str, int]]: Estudiante más joven
y más viejo.

    Raises:
        ValueError: Si no hay estudiantes en student_data.
    """
    if not student_data:
        raise ValueError("No se encontraron estudiantes.")

    min_student = min(student_data, key=lambda x: x[1])  # estudiante
más joven
    max_student = max(student_data, key=lambda x: x[1])  # estudiante
más viejo
    return min_student, max_student


def generate_report(school_name: str, date: str, total_students: int,
average_age: float,
                    min_student: Tuple[str, int], max_student:
Tuple[str, int]) -> str:
    """
    Genera el informe final de registro de estudiantes.

    Args:
        school_name (str): Nombre de la escuela.
        date (str): Fecha del informe.
        total_students (int): Número total de estudiantes.
        average_age (float): Edad promedio de los estudiantes.
        min_student (Tuple[str, int]): Estudiante más joven.
        max_student (Tuple[str, int]): Estudiante más viejo.

    Returns:
        str: Informe de registro de estudiantes.

    Raises:
        ValueError: Si los datos no son válidos.
    """
    try:
        if total_students < 0:
            raise ValueError("El número total de estudiantes no puede
ser negativo.")

        report = f"""Informe de Registro de Estudiantes de
{school_name}
Fecha: {date}
Total de Estudiantes: {total_students}
```

```
Edad Promedio: {average_age:.2f}
Estudiante Más Joven: {min_student[0]}, {min_student[1]} años
Estudiante Más Viejo: {max_student[0]}, {max_student[1]} años
"""
    except ValueError as e:
        return str(e)

    return report

# Ejemplo de uso
student_report = generate_student_report(
    "Escuela Primaria",
    "2023-09-15",
    ("Juan Pérez", 10, "5to"),
    ("María López", 9, "4to"),
    ("Carlos Gómez", 11, "5to"),
    ("Ana Torres", 8, "3ro"),
    ("Luis Martínez", 10, "5to"),
    ("Laura Jiménez", 7, "2do"),
    ("Pedro González", 12, "6to"),
    ("Sofía Fernández", 9, "4to"),
    grade_filter="5to"
)

print(student_report)
```

```
Informe de Registro de Estudiantes de Escuela Primaria
Fecha: 2023-09-15
Total de Estudiantes: 3
Edad Promedio: 10.00
Estudiante Más Joven: Juan Pérez, 10 años
Estudiante Más Viejo: Carlos Gómez, 11 años
```