```python
from pyspark.sql import SparkSession
from pyspark import SparkContext

# Crear una sesión de Spark
spark = SparkSession.builder \
    .appName("RDDExample") \
    .master("local[*]") \
    .getOrCreate()

sc = spark.sparkContext

# Crear un RDD a partir de una colección
rdd = sc.parallelize(range(1, 1000001), 100)

# Aplicar transformaciones complejas
resultRDD = rdd \
    .map(lambda x: (x % 10, x)) \
    .reduceByKey(lambda a, b: a + b) \
    .mapValues(lambda x: x * x)

# Acción para materializar el resultado
result = resultRDD.collect()

# Imprimir los primeros 10 elementos del resultado
for item in result[:10]:
    print(item)

# Detener la sesión de Spark
spark.stop()
```

El código que has proporcionado es un ejemplo completo de cómo crear y trabajar con un Resilient Distributed Dataset (RDD) en PySpark. Vamos a analizarlo paso a paso:

1. **`from pyspark.sql import SparkSession`**: Importa la clase `SparkSession`, que es el punto de entrada para la funcionalidad de Spark SQL y DataFrames. Aunque este ejemplo se centra en RDDs, es una buena práctica iniciar siempre con una `SparkSession` ya que proporciona acceso al `SparkContext` subyacente.
2. **`from pyspark import SparkContext`**: Importa la clase `SparkContext`. El `SparkContext` es la entrada principal para la funcionalidad de Spark "core", incluyendo RDDs. En versiones más recientes de Spark, el `SparkContext` se crea automáticamente cuando se inicializa una `SparkSession`, por lo que esta importación no siempre es estrictamente necesaria si ya estás creando una `SparkSession`.
3. **`spark = SparkSession.builder \`**: Inicia la construcción de una `SparkSession`.
    - **`.appName("RDDExample") \`**: Establece el nombre de la aplicación Spark, que será visible en la interfaz web de Spark.

    - **`.master("local[*]") \`**: Configura la conexión al cluster Spark. `"local[*]"` indica que Spark se ejecutará en modo local, utilizando todos los núcleos disponibles de tu máquina. Esto es útil para desarrollo y pruebas.
    - **`.getOrCreate()`**: Obtiene una `SparkSession` existente o crea una nueva si no existe.
4. **`sc = spark.sparkContext`**: Obtiene el `SparkContext` asociado con la `SparkSession` que acabas de crear. Ahora puedes usar `sc` para operaciones de bajo nivel de Spark, como la creación de RDDs.
5. **`rdd = sc.parallelize(range(1, 1000001), 100)`**: Crea un RDD llamado `rdd`.
    - **`range(1, 1000001)`**: Genera una secuencia de números enteros desde 1 hasta 1,000,000 (inclusive).
    - **`sc.parallelize(..., 100)`**: Distribuye esta secuencia de números en 100 particiones dentro del RDD. El número de particiones puede afectar el paralelismo de las operaciones posteriores.
6. **`resultRDD = rdd \`**: Comienza una cadena de transformaciones en el `rdd`.
    - **`.map(lambda x: (x % 10, x)) \`**: Aplica una función a cada elemento del `rdd`. La función `lambda x: (x % 10, x)` toma un número `x` y devuelve una tupla donde el primer elemento es el resto de `x` al dividirlo por 10, y el segundo elemento es el propio `x`. Esto crea un RDD de pares clave-valor.
    - **`.reduceByKey(lambda a, b: a + b) \`**: Agrupa los elementos con la misma clave (el resto de la división por 10) y aplica una función de reducción a los valores asociados. La función `lambda a, b: a + b` suma los valores para cada clave.
    - **`.mapValues(lambda x: x * x) \`**: Aplica una función a cada *valor* del RDD de pares clave-valor (sin modificar la clave). La función `lambda x: x * x` eleva al cuadrado el valor.
7. **`result = resultRDD.collect()`**: Realiza una acción que materializa el resultado del `resultRDD` y lo trae de vuelta al programa del driver como una lista de Python llamada `result`. **Advertencia:** `collect()` puede ser costoso si el RDD resultante es muy grande, ya que todos los datos se transfieren a la memoria del driver.
8. **`for item in result[:10]: \`**: Itera a través de los primeros 10 elementos de la lista `result` (si hay al menos 10 elementos).
    - **`print(item)`**: Imprime cada uno de estos elementos.
9. **`spark.stop()`**: Detiene la `SparkSession` y libera los recursos asociados. Es importante detener la sesión al finalizar tu trabajo para evitar el consumo innecesario de recursos.

En resumen, este código realiza las siguientes operaciones:

- Crea un RDD con números del 1 al 1,000,000.
- Transforma el RDD para crear pares clave-valor donde la clave es el último dígito del número.
- Suma todos los números que tienen el mismo último dígito.
- Eleva al cuadrado la suma resultante para cada último dígito.
- Recopila los resultados en una lista de Python y muestra los primeros 10.
- Finalmente, detiene la sesión de Spark.

Este es un buen ejemplo para entender las transformaciones (`map`, `reduceByKey`, `mapValues`) y las acciones (`collect`) básicas en PySpark RDDs. Recuerda que las transformaciones son "lazy" (se ejecutan solo cuando se necesita un resultado, por ejemplo, con una acción como `collect`).