Section 1: Setup & First RDD

TASK 1.1 Solution:

```
words_list = ["Spark", "RDD", "HandsOn"]
words_rdd = sc.parallelize(words_list)
print("Count:", words_rdd.count())  # 3
```

Section 2: Transformations

TASK 2.1 Solution (cubes >100):

```
cubes = rdd.map(lambda x: x**3)
big_cubes = cubes.filter(lambda x: x > 100)
print("Big cubes:", big_cubes.collect())  # [125, 216, 343, 512,
        729, 1000]
```

TASK 2.2 Solution:

```
total = rdd.reduce(lambda a, b: a + b)
print("Sum:", total)  # 55
```

Section 3: Text Processing

Expected lines RDD:

```
lines = [
    "Apache Spark processes data fast",
    "RDD is resilient distributed dataset",
    "Hands-on practice makes perfect",
    "Transformations are lazy evaluations"
]
text_rdd = sc.parallelize(lines)
```

TASK 3.1 Solution (clean words):

```
words = text_rdd.flatMap(lambda line: line.lower().split())
words_clean = words.filter(lambda w: w != "data" and len(w) > 3)
print("Clean words:", words_clean.collect())
# ['apache', 'spark', 'processes', 'fast', 'resilient',
        'distributed', ...]
```

TASK 3.2 Solution:

```
long_words = words.filter(lambda w: len(w) > 4)
print("Words >4 chars:", long_words.count())  # 12 (depends on
        exact text)
```

Section 4: Pair RDDs & Word Count

Complete pipeline:

```
word_pairs = words_clean.map(lambda w: (w, 1))
counts = word_pairs.reduceByKey(lambda a, b: a + b)
print("All counts:", counts.take(10))

top_words = counts.map(lambda kv: (kv[^1],
        kv[^0])).sortByKey(ascending=False)
print("Top 3:", top_words.take(3))
```

TASK 4.1 Solution (≥2 times):

```
frequent = counts.filter(lambda kv: kv[^1] >= 2)
print("Frequent words:", frequent.collect())  # [('is', 1),
        ('practice', 1)] → only ≥2
```

TASK 4.2 Solution:

```
counts.saveAsTextFile("wordcount_output")
print("Saved to wordcount_output/")
```

Section 5: MINI-CHALLENGE (Transactions)

Complete solution:

```
transactions = [
    ("A", 100), ("B", 200), ("A", 50),
    ("C", 70), ("B", 30), ("A", 150)
]
tx_rdd = sc.parallelize(transactions)

# 1. Total per customer
totals = tx_rdd.reduceByKey(lambda a,b: a+b)
print("Totals:", totals.collect())  # [('A', 300), ('B', 230),
        ('C', 70)]

# 2-3. Sort DESC + top 2
top_customers = totals.map(lambda kv: (kv[^1],
        kv[^0])).sortByKey(ascending=False)
print("Top 2 customers:", top_customers.take(2))  # [('A', 300),
        ('B', 230)]
```

BONUS: With Cache (Performance Demo)

```
tx_rdd.cache()
print("Count:", tx_rdd.count())
print("Totals (fast!):", tx_rdd.reduceByKey(lambda
        a,b:a+b).count())
tx_rdd.unpersist()
```
