

4.2 Introduction to Using Data Sets

Self-Study Guide · About 45 minutes · With answer key

By the end you will be able to

- Explain what a data set is and recognize data sets in everyday life.
- Match a question to the kind of processing a data set needs to answer it, and tell when a data set cannot answer a question.
- Describe, in written steps, an algorithm that visits the values of a data set one at a time and processes each one.
- Show a data set as a table or chart and use a planning table to work out an algorithm before writing code.
- Trace Java code that reads a data set one value at a time and spot mistakes in where values are started, updated, or read.

What to remember from earlier

- An accumulator builds a result a little at a time inside a loop. ([Topic 2.9: Implementing Selection and Iteration Algorithms](#))
- A running total starts at 0 and adds a value on each pass. ([Topic 2.9: Implementing Selection and Iteration Algorithms, 2.9.1 Adding Up a Sum](#))
- A counter goes up by 1 only when a condition is true. ([Topic 2.9: Implementing Selection and Iteration Algorithms, 2.9.3 Counting Values That Meet a Condition](#))
- A best-so-far variable is replaced whenever a later value beats it. ([Topic 2.9: Implementing Selection and Iteration Algorithms, 2.9.6 Finding the Minimum or Maximum](#))
- A Scanner hands out its values one at a time with `nextInt()`, and a value cannot be read twice. ([Topic 1.14: Calling Instance Methods, 1.14.4 Calling Methods on a Scanner](#))

4.2.1 What a Data Set Is

Your phone counts your steps every day. A coach writes down every runner's lap time. A weather station records the temperature every hour. One number on its own says very little: "8,400 steps" does not tell you whether this was a good week. Put a whole week of step counts side by side, though, and you can start asking real questions. Was I more active on weekends? How many days did I reach my goal?

Computing is full of situations like this, where the interesting information lives in many related values taken together rather than in any single one. A **data set** is a group of particular facts or measurements about the same thing, gathered so that they can be looked at together. Each piece of information in it is a **data value**. A week of step counts is a data set; each day's count is one data value. Topic 4.1 used the term while looking at the risks of collecting data; this chapter looks closely at what a data set is and how a program works with one.

Three things make a collection a data set rather than a random pile of numbers:

Feature	What it means	In the step-count example
---------	---------------	---------------------------

The values describe one subject	Every value is the same kind of measurement or fact	Every value is a number of steps in one day
Each value is specific	Each one records a particular observation, not a guess	Tuesday: exactly 8,400 steps
The values belong together	They were collected for the same person, place, or purpose	All seven days are the same student's week

Data values do not have to be numbers. A list of the songs a radio station played this morning is a data set of words. A record of each day's weather written as one letter (S for sunny, R for rain, C for cloudy) is a data set of single characters.

KEY IDEA

A data set is a group of particular facts or measurements about one subject. Each piece is one data value, and the values are useful because they can be examined together.

WORKED EXAMPLE A week of step counts

A hypothetical student, Maya, records her steps for one week. The natural way to write a small data set down is a **table**: one row (or column) per value, with a label saying what each value describes.

Day	Steps
Monday	6,200
Tuesday	8,400
Wednesday	7,100
Thursday	9,300
Friday	5,800
Saturday	10,200
Sunday	8,900

Check it against the three features. Every value is a number of steps (one subject). Each is an exact count from her phone (specific). All seven belong to Maya's same week (they belong together). So this table holds a data set of seven data values. The day names are labels that tell you which value is which; the data values are the step counts.

Now change one thing: mix in a value that does not belong. Suppose someone adds a row "Maya's height: 160" to the table. It is a specific piece of information about Maya, but it is not a step count for a day, so it does not describe the same subject. Adding up the column would now mix steps with centimeters and give a meaningless number. A data set is only as useful as the agreement among its values.

WATCH OUT

Confusing a result with a data value. After adding Maya's steps, you get 55,900 for the week. Students sometimes write that total into the table as an eighth row. But 55,900 was not observed on any day; it was **calculated from** the data set. If it sits in the table, the next

person who adds the column counts the week twice. Keep the data set (what was recorded) separate from the results you compute from it.

PAUSE AND TRY

Question 1 Multiple choice

Which of these is a data set?

- A. The single temperature reading 72 degrees
- B. The number of points each player on a basketball team scored in Friday's game
- C. The question "Who scored the most points on Friday?"
- D. The average of the points scored on Friday

Question 2 Short answer

A hypothetical cafeteria records a table with these rows: "Monday: 212 lunches sold," "Tuesday: 198 lunches sold," "Wednesday: 230 lunches sold," and "Price of a lunch: 4 dollars." Which row does not belong in the data set, and why?

4.2.2 Asking Questions of a Data Set

Collecting data is never the goal by itself. Maya did not record her steps so that she would have a table; she recorded them to find something out. A data set becomes useful when someone **analyzes** it: examines and works with its values to solve a problem or answer a question.

The question decides what work has to be done. Most everyday questions about a data set fall into a handful of familiar kinds, and you already wrote loops for most of them in Topic 2.9:

Kind of question	Everyday example	What the processing keeps track of
How much in all?	How many steps did Maya take this week?	A running total
What is typical?	How many steps did she average per day?	A running total, then divide
How many match?	On how many days did she reach 8,000 steps?	A counter
Which is biggest?	What was her most active day?	The best so far
Is there any?	Did she ever walk more than 10,000 steps?	A true/false flag (Topic 2.10)
Is there a pattern?	Did her steps go up from one day to the next?	The previous value, to compare

Each row asks something different of the **same** seven values, and each needs a different thing remembered while the values are examined. (A flag, from Topic 2.10, is a `boolean` variable that records whether something has happened yet.)

KEY IDEA

Data sets are analyzed to solve a problem or answer a question. The question decides what processing the values need and what has to be remembered along the way.

WORKED EXAMPLE Three questions, one data set

A hypothetical bus company records how many minutes late one bus on Route 9 was each school day for two weeks:

Day	1	2	3	4	5	6	7	8	9	10
Minutes	0	4	2	0	11	3	0	6	1	0

Look at what each question needs:

1. **"How many days was the bus on time?"** This is a "how many match?" question. The matching condition is "minutes late is 0." Going through the row, the matches are days 1, 4, 7, and 10, so the answer is 4. Only a counter is needed.
2. **"What was the worst delay?"** This is a "which is biggest?" question. Keep the largest value seen so far and replace it when a bigger one appears. The answer is 11 minutes (day 5).
3. **"What is the average delay?"** This is a "what is typical?" question. Total all ten values: $0 + 4 + 2 + 0 + 11 + 3 + 0 + 6 + 1 + 0 = 27$ minutes, then divide by 10 days for 2.7 minutes per day.

Same ten values, three different answers, because each question asks for a different kind of processing.

Now vary the question in a different way: ask "How late is the Route 4 bus usually?" The kind of question (what is typical?) is one you can process, but this data set cannot answer it. Every value describes Route 9. As you saw in Topic 4.1, a data set collected for one question may not be appropriate for a different one. Before planning any processing, check that the data set actually contains information about what is being asked.

WATCH OUT

Answering a question the data does not cover. It is tempting to stretch a data set to a question that sounds close: "Route 9 averages 2.7 minutes late, so the bus system is on time." But the data says nothing about other routes, other months, or snowy days. The calculation would run and produce a number, and the number would be wrong for that question. Ask first: does every value I need actually appear in this data set?

PAUSE AND TRY**Question 3** Multiple choice

A hypothetical library records the number of books checked out at its Main Street branch each day in March. Which question can this data set answer?

- A. How many books did the Oak Avenue branch check out in March?
- B. On how many March days did Main Street check out more than 100 books?
- C. Which book was the most popular at Main Street in March?
- D. How many books will Main Street check out next December?

Question 4 Short answer

Using the Route 9 table from the worked example, answer the question "On how many days was the bus more than 5 minutes late?" Name the kind of question and what the processing keeps track of.

4.2.3 Processing Values One at a Time

When you added up the ten Route 9 delays, you probably did not take in all ten numbers at once. You went through them in order: 0, then 4 (total 4), then 2 (total 6), and so on, keeping one number in your head. That habit is exactly how a program analyzes a data set.

Think of a cashier at a grocery store. The cashier does not lift the whole cart onto the scanner. Items go past one at a time, each one is scanned, and the register updates the total right then. At the end, the total is ready. Where the analogy stops working: a cashier can glance back at an item already bagged, but a program reading values one after another often cannot, so it must remember whatever it will need later **as it goes**.

When a data set is analyzed, its values are accessed and used **one at a time**, and each value is processed according to what the question needs. Every such algorithm has the same three-part shape, which you can write in plain language before any code exists:

1. **Set up.** Before looking at any value, give the thing you are tracking its starting value (a total of 0, a count of 0, the first value as the best so far).
2. **Visit each value, one at a time.** For each value, do the processing the question needs: add it, test it and maybe count it, compare it to the best so far.
3. **Report.** After the last value, the tracked result is the answer (or, for an average, one division away from it).

KEY IDEA

To analyze a data set, set up what you are tracking, visit the values one at a time and process each one, then report the result after the last value.

WORKED EXAMPLE A written algorithm for total steps

Here is the "how much in all?" algorithm for Maya's week written in words:

1. Set **total** to 0.
2. For each day's step count, in order: add the step count to **total**.
3. After the last day, report **total**.

Follow it through the data set. The table shows what **total** holds after each value is processed:

Value visited	What step 2 does	total after
(start)	Set to 0	0
6,200	$0 + 6,200$	6,200
8,400	$6,200 + 8,400$	14,600

7,100	14,600 + 7,100	21,700
9,300	21,700 + 9,300	31,000
5,800	31,000 + 5,800	36,800
10,200	36,800 + 10,200	47,000
8,900	47,000 + 8,900	55,900

Step 3 reports 55,900. At every moment, the algorithm held just one number, and each value was used once, at its turn.

Now change the question but keep the shape: "On how many days did Maya reach 8,000 steps?"

1. Set **goalDays** to 0.
2. For each day's step count, in order: if the step count is at least 8,000, add 1 to **goalDays**.
3. After the last day, report **goalDays**.

Visiting 6,200, 8,400, 7,100, 9,300, 5,800, 10,200, 8,900 in turn, **goalDays** goes 0, 1, 1, 2, 2, 3, 4. The answer is 4. Setup and reporting look the same as before; only the processing inside step 2 changed. That is the point of the shape: once you know it, a new question means deciding what to set up and what to do with each value.

WATCH OUT

Putting the setup inside the "for each" step. A plan that says "For each day's step count: set goalDays to 0, then if the count is at least 8,000 add 1" looks almost right. But the reset happens again at every value, wiping out what was counted before. At the end, **goalDays** describes only the last day (1, because Sunday's 8,900 reached the goal). Setup happens once, before the first value; only the processing repeats.

PAUSE AND TRY

Question 5 Tracing

A hypothetical student saves money each week: 12, 0, 7, and 15 dollars. Follow this written algorithm and give the final values of **total** and **bigWeeks**.

1. Set **total** to 0 and **bigWeeks** to 0.
2. For each week's savings, in order: add it to **total**; then, if it is at least 10, add 1 to **bigWeeks**.
3. After the last week, report **total** and **bigWeeks**.

Question 6 Multiple choice

Which written algorithm correctly reports how many test scores in a data set are at least 90?

- A. Set count to 0 → for each score, if it is at least 90, add 1 to count → after the last score, report count
- B. For each score, set count to 0, then if it is at least 90 add 1 to count → after the last score, report count
- C. Set count to 0 → for each score, add 1 to count → after the last score, report count
- D. Set count to 0 → if the first score is at least 90, add 1 to count → report count

4.2.4 Planning an Algorithm With a Table

Some questions are harder to plan in words alone. "What was the warmest night?" or "Did the temperature keep rising?" require remembering something and comparing against it, and it is easy to get the details wrong in your head. Programmers plan these on paper first, using a picture of the data.

A data set can be shown in a diagram such as a **table** or a **chart**. A table lists the values exactly, which is what you need to follow an algorithm step by step. A chart, such as a bar chart, turns each value into a shape whose size shows the value, so patterns jump out at a glance. Here is Maya's week as a simple text bar chart, where each # stands for 1,000 whole steps:

```
Mon  6,200  #####
Tue  8,400  #####
Wed  7,100  #####
Thu  9,300  #####
Fri  5,800  #####
Sat 10,200  #####
Sun  8,900  #####
           ^ goal: 8,000
```

The chart answers "which days reached the goal?" visually (the bars that pass the 8 mark), and it suggests the algorithm: check each bar against the goal line, one bar at a time.

To plan the details of an algorithm, add columns to the table for everything the algorithm must remember. This **planning table** has one row per data value and one column per tracked result. Filling it in, row by row, is running the algorithm by hand before it becomes code. If the last row holds the right answer, the plan works; if not, the table shows exactly which row went wrong.

KEY IDEA

Showing a data set as a table or chart helps plan the algorithm. A planning table adds a column for each thing the algorithm remembers and a row for each value, so you can run the plan by hand and check it.

WORKED EXAMPLE Planning the warmest night

A hypothetical town records its overnight low temperatures (in degrees Fahrenheit) for five winter nights: -3, -8, -1, -6, -4. The question is "What was the warmest of these lows?", a "which is biggest?" question. The plan: start **highest** at the first value, then for each remaining value, if it is greater than **highest**, replace **highest**.

Night	Value	Value greater than highest?	highest after
1	-3	(first value, used to start)	-3
2	-8	-8 > -3? no	-3
3	-1	-1 > -3? yes	-1
4	-6	-6 > -1? no	-1
5	-4	-4 > -1? no	-1

The last row holds -1 , which is the warmest night. The table also shows why each step happened: on night 3 the comparison was true, so the value was replaced; on every other night it was false, so **highest** stayed.

Now vary the question to a pattern: "On how many nights was it warmer than the night before?" The best so far is no longer the right thing to remember; each value must be compared with the one just **before** it. So the planning table needs a column for the **previous** value, and after each comparison, the current value becomes the new previous value.

Night	Value	previous	Warmer than previous?	rises after
1	-3	(none yet)	(first value)	0
2	-8	-3	no	0
3	-1	-8	yes	1
4	-6	-1	no	1
5	-4	-6	yes	2

Two nights (3 and 5) were warmer than the night before. Drawing the table forced a decision you might have missed in words: the first night has nothing before it, so it can only become the first **previous** value.

WATCH OUT

Starting the best so far at 0. A plan that says "Set highest to 0, then for each value, if it is greater than highest, replace highest" works for step counts, which are never negative. Run it in a planning table on the winter lows: $-3 > 0$? no. $-8 > 0$? no. $-1 > 0$? no. $-6 > 0$? no. $-4 > 0$? no. Nothing ever replaces the 0, so the plan reports 0 degrees, a temperature that is not in the data set at all. Start the best so far at the first value, which is guaranteed to be real data. The planning table catches this mistake before any code is written.

PAUSE AND TRY

Question 7 Tracing

A hypothetical video game costs 24, 19, 27, 19, and 22 dollars at five stores (stores 1 through 5). The plan: start **lowest** at store 1's price and **bestStore** at 1; for each later store, if its price is **less than lowest**, set **lowest** to that price and **bestStore** to that store's number. Fill in a planning table and give the final **lowest** and **bestStore**. Then say what **bestStore** would be if the comparison were "less than or equal to" instead.

Question 8 Short answer

For the store prices in the previous question, a friend's plan starts **lowest** at 0 and replaces it whenever a price is less than **lowest**. What does the friend's plan report, and why? What should the plan start with instead?

4.2.5 Processing a Data Set in Java

A planning table is only useful if it turns into a working program. You already have the tools. In Topics 1.13 and 1.14, a `Scanner` built from a string handed out its values one at a time, and each `nextInt()` call moved it forward so the same value could not be read twice. In Topic 2.9, accumulator variables built up a result inside a loop. Put those together and you have a program that processes a data set exactly the way this chapter has described: one value at a time, never all at once.

The step from plan to code is direct. Each column you tracked in the planning table becomes a variable. The setup becomes statements before the loop. The "for each value" step becomes the loop body, with one `nextInt()` call to get the current value. The report becomes a print statement after the loop. (Later in this unit you will learn ways to store a whole data set inside a program so that its values can be visited again. For now, each value is read, used, and left behind.)

KEY IDEA

Each column of a planning table becomes a variable. Setup goes before the loop, processing one value goes inside it, and the report goes after it.

WORKED EXAMPLE The warmest night, from plan to code

This program is the warmest-night plan from 4.2.4. The five lows are in a `Scanner` built from a string.

HighestTemp.java

```

1  import java.util.Scanner;
2
3  public class HighestTemp
4  {
5      public static void main(String[] args)
6      {
7          Scanner lows = new Scanner("-3 -8 -1 -6 -4");
8          int highest = lows.nextInt();
9          for (int night = 2; night <= 5; night++)
10         {
11             int temp = lows.nextInt();
12             if (temp > highest)
13             {
14                 highest = temp;
15             }
16         }
17         System.out.println("Warmest low: " + highest);
18     }
19 }
```

Match each part to the plan:

Plan	Code
Start highest at the first value	Line 8: <code>int highest = lows.nextInt();</code> reads -3
For each remaining value (nights 2 to 5)	Line 9: the loop runs with <code>night = 2, 3, 4, 5</code>
Get the current value	Line 11: <code>int temp = lows.nextInt();</code>
If it is greater than highest , replace	Lines 12–15

Report **highest**

Line 17, after the loop

The variable `highest` takes the values `-3`, `-3`, `-1`, `-1`, `-1`, exactly as in the planning table's last column, and the program prints:

```
Warmest low: -1
```

The loop starts at night 2 because line 8 already read the first value. A version that starts `highest` at 0 and loops over all five nights prints `Warmest low: 0`, the mistake the planning table caught in 4.2.4.

Now vary the processing. Here is the "how many days reached 8,000 steps?" plan from 4.2.3 in Java:

StepGoalCount.java

```
1 import java.util.Scanner;
2
3 public class StepGoalCount
4 {
5     public static void main(String[] args)
6     {
7         Scanner weekSteps = new Scanner("6200 8400 7100 9300 5800 10200 8900");
8         int goalDays = 0;
9         for (int day = 1; day <= 7; day++)
10        {
11            int steps = weekSteps.nextInt();
12            if (steps >= 8000)
13            {
14                goalDays++;
15            }
16        }
17        System.out.println("Days at goal: " + goalDays);
18    }
19 }
```

```
Days at goal: 4
```

The setup (line 8) is a counter starting at 0, not a value read from the data, so the loop reads all seven values, with `day` running from 1 to 7. Inside the loop, line 11 reads the current day's steps and line 12 tests it; `goalDays` goes 0, 1, 1, 2, 2, 3, 4, matching the written trace. A data set does not have to be numbers, either. The same visit-and-process shape works on a string of single-letter weather codes using the character loop from Topic 2.10, as you will see in the questions below.

WATCH OUT

Reading more values than the data set has. The loop in this program runs while `day <= 8`, but the week has only seven values:

StepTooMany.java

```
1 import java.util.Scanner;
2
3 public class StepTooMany
4 {
5     public static void main(String[] args)
6     {
7         Scanner weekSteps = new Scanner("6200 8400 7100 9300 5800 10200 8900");
8         int total = 0;
9         for (int day = 1; day <= 8; day++)
10        {
11            int steps = weekSteps.nextInt();
12            total += steps;
13        }
14        System.out.println("Total steps: " + total);
15    }
16 }
```

Seven passes read all seven values. On the eighth pass, line 11 calls `nextInt()` when nothing is left, and the program crashes before printing anything:

```
Exception in thread "main" java.util.NoSuchElementException
    at StepTooMany.main(StepTooMany.java:11)
```

(The output is shortened here: Java also prints several lines about where inside `Scanner` the error happened. The exact wording can vary between Java versions.) Students make this mistake by guessing the loop's bound instead of counting the data. Every value in the data set is read exactly once, so the loop must run once per remaining value: change `day <= 8` to `day <= 7`.

PAUSE AND TRY

Question 9 Tracing

A hypothetical weather log records one letter per day: S for sunny, R for rain, C for cloudy. What does this code print? Show which characters are counted.

```
1 String weather = "SSRCSRS";
2 int rainyDays = 0;
3 for (int i = 0; i < weather.length(); i++)
4 {
5     if (weather.substring(i, i + 1).equals("R"))
6     {
7         rainyDays++;
8     }
9 }
10 System.out.println("Rainy days: " + rainyDays);
```

Question 10 Multiple choice

In the step total program below, the loop was changed to run while `day <= 6`. The data set still has seven values. What does the program print?

StepTotalSixDays.java

```
1 import java.util.Scanner;
2
3 public class StepTotalSixDays
4 {
5     public static void main(String[] args)
6     {
7         Scanner weekSteps = new Scanner("6200 8400 7100 9300 5800 10200 8900");
8         int total = 0;
9         for (int day = 1; day <= 6; day++)
10        {
11            int steps = weekSteps.nextInt();
12            total += steps;
13        }
14        System.out.println("Total steps: " + total);
15        System.out.println("Average per day: " + total / 7);
16    }
17 }
```

- A. Total steps: 47000 then Average per day: 6714
- B. Total steps: 55900 then Average per day: 7985
- C. Nothing; it crashes with a `NoSuchElementException`
- D. Total steps: 47000 then Average per day: 7833

Putting it together

A hypothetical student, Maya, wants to know two things about her week: how many steps she took in all, and how many she averaged per day. Follow the whole process.

The data set (4.2.1). Seven specific values, one step count per day, all from the same week: 6,200, 8,400, 7,100, 9,300, 5,800, 10,200, 8,900.

The questions (4.2.2). "How much in all?" needs a running total. "What is typical?" needs the same running total, divided by the number of days. Both are about Maya's steps this week, which is exactly what the data records, so the data set can answer them.

The plan in words (4.2.3). Set **total** to 0. For each day's steps, add them to **total**. After the last day, report **total**, and report **total** divided by 7.

The plan in a table (4.2.4). One row per value, one column for **total**: 0, 6,200, 14,600, 21,700, 31,000, 36,800, 47,000, 55,900. The last row gives 55,900 steps, and $55,900 \div 7$ is about 7,985.7.

The code (4.2.5). The column **total** becomes the variable `total`, set up before the loop. The loop runs once per value and reads each with `nextInt()`. The two reports go after the loop.

StepTotal.java

```

1  import java.util.Scanner;
2
3  public class StepTotal
4  {
5      public static void main(String[] args)
6      {
7          Scanner weekSteps = new Scanner("6200 8400 7100 9300 5800 10200 8900");
8          int total = 0;
9          for (int day = 1; day <= 7; day++)
10         {
11             int steps = weekSteps.nextInt();
12             total += steps;
13         }
14         System.out.println("Total steps: " + total);
15         System.out.println("Average per day: " + total / 7);
16     }
17 }

```

Total steps: 55900
Average per day: 7985

The total matches the planning table exactly. The average prints 7985, not 7985.7, because `total / 7` divides two `int` values and drops the decimal part (Topic 1.3). For step counts, a whole number of steps is a sensible answer; if the decimal mattered, you would divide by `7.0` instead, so that the division uses `double` values (Topic 1.3). Each idea in the chapter did one job: the data set held the facts, the question chose the processing, the written plan and the table worked out the details, and the code carried them out one value at a time.

Vocabulary

Term	Definition	Example
data set	A group of particular facts or measurements about the same thing	Maya's seven daily step counts
data value	One piece of information in a data set	Tuesday's 8,400 steps
analyze	Examine and work with a data set's values to solve a problem or answer a question	Finding how many days reached 8,000 steps
table	A diagram that lists a data set's values exactly, with labels	Day and Steps columns
chart	A diagram that shows each value as a shape whose size shows the value	A bar per day, longer for more steps
planning table	A table with a row per data value and a column per tracked result, used to run a plan by hand	Value, highest after

Chapter summary

- A **data set** is a group of particular facts or measurements about one subject. Each **data value** is one of those pieces, and results calculated from the data are kept separate from it.
- A data set is **analyzed** to solve a problem or answer a question. The question decides the processing (a total, a count, a best so far, a comparison with the previous value), and the data must actually cover what is asked.
- Analysis visits the values **one at a time**: set up what you track once, process each value at its turn, and report after the last value.
- A data set can be shown as a **table** or **chart**, and a **planning table** with a column for each tracked result lets you run and check an algorithm before coding it.
- In Java, each planning-table column becomes a variable, setup goes before the loop, one value is read and processed per pass, and the loop must run exactly once per value to read.

Review questions

Question 11 Multiple choice

A hypothetical community center records how many people visited each day for a month. The director asks, "Was there any day with more than 300 visitors?" Which plan answers the question?

- A. Keep a running total of all visitors → report whether the total is more than 300
- B. Set found to false → for each day, if visitors are more than 300, set found to true → after the last day, report found
- C. Set found to false → for each day, set found to true if visitors are more than 300 and false otherwise → after the last day, report found
- D. Find the average number of visitors per day → report whether the average is more than 300

Question 12 Tracing

A hypothetical classroom thermometer records the temperature each hour: 41, 44, 47, 46, 50. Trace the program and give its output. What question about the data set does it answer?

RiseCount.java

```

1  import java.util.Scanner;
2
3  public class RiseCount
4  {
5      public static void main(String[] args)
6      {
7          Scanner readings = new Scanner("41 44 47 46 50");
8          int previous = readings.nextInt();
9          int rises = 0;
10         for (int hour = 2; hour <= 5; hour++)
11         {
12             int current = readings.nextInt();
13             if (current > previous)
14             {
15                 rises++;
16             }
17             previous = current;
18         }
19         System.out.println("Rises: " + rises);
20     }
21 }

```

Question 13 Debugging

This program should report how many days in Maya's week reached 8,000 steps (the correct answer is 4), but it prints Days at goal: 1. Find the error and correct it.

StepGoalCountReset.java

```

1  import java.util.Scanner;
2
3  public class StepGoalCountReset
4  {
5      public static void main(String[] args)
6      {
7          Scanner weekSteps = new Scanner("6200 8400 7100 9300 5800 10200 8900");
8          int goalDays = 0;
9          for (int day = 1; day <= 7; day++)
10         {
11             goalDays = 0;
12             int steps = weekSteps.nextInt();
13             if (steps >= 8000)
14             {
15                 goalDays++;
16             }
17         }
18         System.out.println("Days at goal: " + goalDays);
19     }
20 }

```

Question 14 Tracing

A hypothetical student logs whether she finished her homework each day: Y, Y, N, Y, Y, Y, N. She wants the **longest streak** of Y days in a row. Her plan:

1. Set **streak** to 0 and **longest** to 0.

2. For each day, in order: if it is Y, add 1 to **streak**, and then if **streak** is greater than **longest**, set **longest** to **streak**. If it is N, set **streak** to 0.
3. After the last day, report **longest**.

Fill in a planning table and give the final value of **longest**.

Question 15 Short answer

A classmate says, "To find the average of a data set, a program has to look at all the values at the same time. Processing them one at a time can't work, because you don't know the average until you've seen everything." Explain what is wrong with this reasoning, using the idea of what the program remembers as it goes.

Reflection

Check each statement. If the answer is "not yet," reread the linked subsection.

- ☐ I can explain what a data set and a data value are, and tell a data value from a calculated result. Reread: 4.2.1 What a Data Set Is
- ☐ I can match a question to the processing it needs and tell when a data set cannot answer it. Reread: 4.2.2 Asking Questions of a Data Set
- ☐ I can describe in written steps an algorithm that visits a data set one value at a time. Reread: 4.2.3 Processing Values One at a Time
- ☐ I can use a table or chart, and a planning table, to plan and check an algorithm. Reread: 4.2.4 Planning an Algorithm With a Table
- ☐ I can trace Java code that reads a data set one value at a time and find setup and loop-bound mistakes. Reread: 4.2.5 Processing a Data Set in Java

Check Your Reasoning

Answers and explanations for every question, numbered to match.

Question 1

B. It is a collection of specific values (one point total per player) that all describe the same thing: points scored in Friday's game.

- **A** is wrong: one value on its own is a single data value, not a collection.
- **C** is wrong: a question is something you ask **of** a data set. It contains no data.
- **D** is wrong: an average is a result calculated from a data set, not the data set itself.

Reread 4.2.1 *What a Data Set Is* if you chose differently.

Question 2

Model answer: "Price of a lunch: 4 dollars" does not belong. The other rows are all the same kind of value (number of lunches sold on one day), but the price is a different kind of fact. If you added up the column to find total lunches sold, the 4 would be counted as if it were a day's sales and the total would be wrong.

A complete answer must:

- name the price row, and
- explain that its value describes a different subject from the other values.

Reread 4.2.1 *What a Data Set Is* if you picked a different row.

Question 3

B. This is a "how many match?" question about exactly the values recorded: Main Street, March, number of books per day. Count the days whose value is greater than 100.

- **A** is wrong: no value describes the Oak Avenue branch.
- **C** is wrong: the data records **how many** books went out each day, not **which** books. Titles are not in the data set.
- **D** is wrong: the data covers March only. Guessing about December stretches the data to a question it was not collected for.

Reread 4.2.2 *Asking Questions of a Data Set* if you chose differently.

Question 4

Model answer: It is a "how many match?" question, so the processing keeps a **counter** that goes up by 1 for each value greater than 5. Going through the row: 0, 4, 2, 0 do not match; 11 matches (count 1); 3 and 0 do not; 6 matches (count 2); 1 and 0 do not. The answer is **2 days**.

A complete answer must:

- give the answer 2,
- identify it as a counting ("how many match?") question, and
- name a counter as the thing remembered.

Values such as 4 are close but do not count, because 4 is not more than 5. Reread 4.2.2 *Asking Questions of a Data Set* if you named a different kind of processing.

Question 5

Value visited	total after	At least 10?	bigWeeks after
(start)	0		0
12	12	yes	1
0	12	no	1
7	19	no	1
15	34	yes	2

The algorithm reports **total = 34** and **bigWeeks = 2**. One pass through the values can update two tracked results at once, because each value is processed in every way the question needs at its turn. Reread 4.2.3 *Processing Values One at a Time* if your table differed.

Question 6

A. Setup happens once, every score is visited, and the count goes up only when the score matches.

- **B** is wrong: the reset is inside the "for each" step, so the count is wiped at every score. It reports 1 or 0 depending only on the last score.
- **C** is wrong: it adds 1 for every score, so it reports how many scores there are, not how many are at least 90. The condition is missing.
- **D** is wrong: it looks at only the first score. Analyzing a data set means visiting every value.

Reread 4.2.3 *Processing Values One at a Time* if you chose differently.

Question 7

Store	Price	Price less than lowest?	lowest after	bestStore after
1	24	(first value, used to start)	24	1
2	19	19 < 24? yes	19	2
3	27	27 < 19? no	19	2
4	19	19 < 19? no	19	2
5	22	22 < 19? no	19	2

The plan reports **lowest = 19** at **store 2**. Store 4 ties at 19, but "less than" is false for a tie, so the first cheapest store is kept.

With "less than or equal to," row 4 becomes $19 \leq 19$? yes, so **bestStore** becomes **4**: the last of the tied stores. The lowest price is still 19 either way; only which tied store gets reported changes.

Reread 4.2.4 *Planning an Algorithm With a Table* if your table differed.

Question 8

Model answer: It reports **0**. Every price is positive, so "price less than 0" is false at every store and **lowest** is never replaced. Zero is not even one of the prices. The plan should start **lowest** at the first price (24), which is real data, and then compare the rest against it.

A complete answer must:

- say the plan reports 0,
- explain that no price is less than 0, so nothing ever replaces it, and
- say to start with the first value in the data set.

This is the same trap as starting a highest temperature at 0, turned upside down. Reread 4.2.4 *Planning an Algorithm With a Table* if your answer missed a point.

Question 9

The loop visits each character of "SSRCSRS" one at a time, from index 0 to index 6:

i	Character	Equals "R" ?	rainyDays after
0	S	no	0
1	S	no	0
2	R	yes	1
3	C	no	1
4	S	no	1
5	R	yes	2
6	S	no	2

Rainy days: 2

This is a "how many match?" question on a data set of characters: the counter is set up before the loop, each character is processed at its turn, and the result is printed after the last one. Reread 4.2.5 *Processing a Data Set in Java* if your count differed.

Question 10

A. The loop runs six times, so it reads and adds only the first six values: $6,200 + 8,400 + 7,100 + 9,300 + 5,800 + 10,200 = 47,000$. Sunday's 8,900 is never read. Line 15 still divides by 7, and $47000 / 7$ in `int` division is 6714 (the decimal part is dropped).

- **B** is wrong: that would need all seven values. A loop that runs too few times silently leaves values out.
- **C** is wrong: the crash happens when a program reads **more** values than exist. Reading fewer is allowed; the extra value just goes unused. That is what makes this mistake dangerous: there is no error message, only a wrong answer.
- **D** is wrong: 7833 is $47000 / 6$, but line 15 divides by 7, not by the number of values actually read.

Reread 4.2.5 *Processing a Data Set in Java* if you chose differently.

Question 11

B. The question is whether **any** value matches. The tracked result starts as "no match yet" and changes to true the first time a matching day is processed. Once true, it stays true, so the report after the last day is correct.

- **A** is wrong: the total for the month is far bigger than any single day. A month of 50-visitor days would have a total over 300 without any single day passing 300.
- **C** is wrong: setting found to false for non-matching days undoes an earlier match. The result describes only the last day, the same problem as resetting a counter inside the "for each" step.
- **D** is wrong: an average can be under 300 even when one busy day is over 300. A question about any single day needs each value checked, not a summary.

Reread 4.2.2 *Asking Questions of a Data Set* and 4.2.3 *Processing Values One at a Time* if you chose differently.

Question 12

Line 8 reads the first value into `previous`. The loop then handles hours 2 to 5:

hour	current	current > previous?	rises after	previous after line 17
(start)			0	41
2	44	44 > 41: yes	1	44
3	47	47 > 44: yes	2	47
4	46	46 > 47: no	2	46
5	50	50 > 46: yes	3	50

Rises: 3

It answers "How many times was the temperature higher than the hour before?", a pattern question. Line 17 is what makes it work: after each comparison, the current value becomes the previous one for the next pass. Without line 17, every value would be compared with 41. Reread 4.2.4 *Planning an Algorithm With a Table* and 4.2.5 *Processing a Data Set in Java* if your table differed.

Question 13

The error is line 11, `goalDays = 0;`, inside the loop. It resets the counter at the start of every pass, so whatever earlier days counted is erased. Only the last day survives: Sunday's 8,900 reaches the goal, so the counter ends at 1. The program compiles and runs without crashing; it is a logic error.

Fix: delete line 11. The setup on line 8, `int goalDays = 0;`, already happens once before the loop, which is where setup belongs. With line 11 removed, the program is the same as `StepGoalCount.java` and prints:

Days at goal: 4

Reread 4.2.3 *Processing Values One at a Time* if you looked for the error elsewhere.

Question 14

Day	Value	streak after	longest after
(start)		0	0
1	Y	1	1
2	Y	2	2
3	N	0	2
4	Y	1	2
5	Y	2	2
6	Y	3	3
7	N	0	3

The plan reports **longest = 3** (days 4 to 6). Two tracked results do different jobs: **streak** is about the current run and drops to 0 on an N, while **longest** is a best so far that never goes down. Day 5 shows why both are needed: **streak** is 2, which ties but does not beat **longest**, so **longest** stays 2 until day 6. Reread *4.2.4 Planning an Algorithm With a Table* if your table differed.

Question 15

Model answer: The program does not need all the values at once; it needs to remember only a **running total** (and know how many values there are). It visits the values one at a time, adding each to the total at its turn. It is true that the average is not known until the end, but that is fine: the average is computed in the report step, after the last value, by dividing the total by the number of values. At every moment, the program holds one number, not the whole data set.

A complete answer must:

- say the program keeps a running total as it visits each value,
- say the division happens after the last value, and
- explain that it never needs all the values at the same moment.

Reread *4.2.3 Processing Values One at a Time* if your answer missed a point.