

2.7 while Loops

Lesson Plan

Pacing: 3 class periods of 45 minutes

Learning objectives: 2.7.A, 2.7.B

Essential knowledge: 2.7.A.1, 2.7.A.2, 2.7.A.3, 2.7.A.4, 2.7.B.1

Materials

- Projector and the Loop the Loop slide deck
- Whiteboard and markers
- Mini whiteboards or scrap paper (trace tables)
- IDE on every computer (Days 1 and 3)
- Printed Loop the Loop puzzle worksheet (answer key for you) and exit ticket

Agenda

Minutes	Activity
5	Day 1 warm-up — one word, two different jobs (slide 2)
15	Day 1 direct instruction — the shape of a while loop and tracing it (slides 3–7)
5	Day 1 direct instruction — if checks once, while keeps checking (slide 8)
15	Day 1 guided practice — trace it (slide 9) and worksheet questions 1–2
5	Day 1 closure — five checks, four passes
5	Day 2 warm-up — trace by memory
12	Day 2 direct instruction — zero iterations and initial conditions (slides 10–11)
10	Day 2 direct instruction — infinite loops (slide 12)
10	Day 2 guided practice — does it end? (slide 13)
5	Day 2 independent practice — worksheet question 3
3	Day 2 closure — two questions about any loop
5	Day 3 warm-up — how many laps?
12	Day 3 direct instruction — off-by-one errors and where the update goes (slides 14–16)
10	Day 3 direct instruction — loops that stop on a goal, and the sentinel (slides 18–20)
8	Day 3 guided practice — fix the off-by-one (slide 17) and worksheet question 7
5	Day 3 closure — will it stop? (slide 22)
5	Day 3 exit ticket

Overview

Day 1 is the loop itself: a `while` loop repeats its body as long as its Boolean condition is true, and the condition is checked before every iteration, including the first (2.7.A.1, 2.7.B.1). Students trace

loops with a table and see that a loop with 4 passes makes 5 checks. Day 2 covers the edges: a loop whose condition is false at the first check runs zero times (2.7.A.3), the starting values a loop needs (initial conditions), and an infinite loop, where the condition is true at every check (2.7.A.2). Day 3 is off-by-one errors, the effect of putting the update before or after the statement that uses the variable, and loops that run until a goal or a stop value is reached.

Students know `if` (Topic 2.3), Boolean expressions with relational and logical operators (Topics 2.2 and 2.5), String comparison with `equals` (Topic 2.6), and `+=` and `++` (Topic 1.6). Stay inside the topic boundary: only `while` loops today. `for` loops are Topic 2.8, so do not write a `for` header even as a shortcut. The standard algorithms (running totals, counting, digits, minimum and maximum) are Topic 2.9; the Scanner sentinel loop on slide 20 is shown only as a loop that stops on a goal. The CED does not ask for `break`, `continue`, or `do-while`; do not teach them.

TEACHER NOTE

The slides are the Loop the Loop deck (22 slides, with speaker notes); slide numbers below refer to it. Day 1 uses slides 1–9, Day 2 uses slides 10–13, and Day 3 uses slides 14–22. The Loop the Loop puzzle worksheet (10 questions, about 35 minutes) is split across the three days: questions 1 and 2 in class on Day 1, question 3 in class on Day 2, questions 4 and 6 as Day 2 homework, and questions 5, 7, 8, 9, and 10 after Day 3 (question 7 is started in class).

Files whose names end in `.snippet.java` hold only the statements that go inside `main`; paste them into the `main` of a scratch class to run them. All files are in the topic's `examples/` folder.

- **Day 1 demonstrations:** `Countdown`, `BallBounce`, `TicketsTrace`, `IfNotWhile`, `CountToFive`.
- **Day 2 demonstrations:** `CupsSkipped`, `MissingUpdateFixed`, `OddStepsFixed`, `ReuseCounter`, `ReuseCounterFixed`. An infinite loop in the IDE needs a stop button; run one only after students have predicted it, and stop it with the IDE's red square.
- **Day 3 demonstrations:** `OffByOneShort`, `OffByOneLong`, `UpdateFirstWrong`, `UpdateFirstRight`, `CountdownShort`, `ScoresReadFirstWrong`, `ScoresUntilStop`.
- **Printed only (exit ticket):** `WhileExitTrace` and `WhileExitPages`.

Day 1 warm-up: one word, two different jobs

Show slide 2. A plate needs 3 scrubs. Students write how many scrubs each instruction gives it: "If the plate is dirty, scrub it" gives 1; "While the plate is dirty, scrub it" gives 3. Take answers without confirming, then ask what the second instruction keeps doing between scrubs: asking again. Leave the word "checks" on the board; the lesson is about when and how often a loop checks.

Day 1 direct instruction: the shape of a while loop and tracing it

1. **Goals** (slide 3). Read the five goals; goals 1–3 are today's, 4 and 5 are Days 2 and 3.
2. **The shape** (2.7.A.1, 2.7.B.1, slide 5). A `while` loop is an iterative statement: `while (condition)` followed by a body in braces. The **condition** is a Boolean expression. If it is true, the body runs; then Java goes back and checks again. If it is false, the loop ends and the statements after the braces run. Each run of the body is an **iteration**. Stress that the check happens before every iteration, including the first (2.7.A.2).

3. **Pour three cups** (slide 6). The condition `cups < 3` is checked 4 times, true 3 times and false once. The body runs 3 times. `cups++` is the **update**: it moves `cups` toward making the condition false.
4. **Trace it with a table** (slide 7). Each row of the table is one check, with the variable values at the moment of the check. Fill it in with the class: the loop checks 5 times and runs the body 4 times, and the last check (`63 > 70` is false) is the one that ends the loop. Run `Countdown` and `BallBounce` to confirm a traced result.
5. **Predict: TicketsTrace**. Students trace `tickets` starting at 13 with `tickets >= 5` on mini whiteboards. Output: 8, 3, then Done. Ask how many checks it made (3) and why Done prints only once.

MISCONCEPTION Putting if where you meant while

Students write `if` and expect the body to repeat. An `if` never goes back to its condition. Run `IfNotWhile` (output: Lap 1, Race over), then change `if` to `while`: it now prints Lap 1 to Lap 5. Ask what would happen without the `lap++` line (slide 12 returns to this). Slide 8 shows the same contrast with `dirt`: `if` leaves 2, `while` leaves 0.

MISCONCEPTION The body runs as many times as the condition is checked

Students say 4 passes means 4 checks. The condition is always checked one more time than the body runs, because the last check is the false one that ends the loop. Make them count both columns on the slide 7 table.

Day 1 direct instruction: if checks once, while keeps checking

Show slide 8 and have students predict each output before revealing it: 2 with `if` and 0 with `while`. Give them one sentence to copy: an `if` is a decision made once; a `while` is a step that repeats until its condition is false.

Day 1 guided practice: trace it

Students work slide 9 on paper: `a` starts at 1, `a = a * 3` while `a < 40`. They fill in the table (check 1: `a` is 1; check 2: 3; check 3: 9; check 4: 27; check 5: 81, false) and write the output, 81, and the number of body runs, 4. Then they do worksheet questions 1 and 2 in pairs. Question 1 (answer A, 4 3) is a trace where the loop divides; question 2 is the table with a doubling step, 5 body runs and 6 checks. Question 1's "Think first" box is settled by question 2.

TEACHER NOTE

Question 1: `n` takes the values 50, 25, 12, 6, 3, and the check `3 > 5` ends the loop after 4 passes. Choice A is correct; have students who pick another choice find the row of their trace where it went wrong. Circulate and look for students who skip the table and trace only the variables they think matter.

Day 1 closure

Ask: "A loop's body ran 4 times. How many times was its condition checked?" Students write 5 and one sentence saying why.

Day 2 warm-up: trace by memory

Put the slide 7 code on the board without the table. Students recreate the table's first and last rows from memory on a sticky note: `temp` 95 and 63, `minutes` 0 and 4. This reactivates the idea that the last check is false.

Day 2 direct instruction: zero iterations and initial conditions

1. **Zero iterations** (2.7.A.3, slide 11). If the condition is false at the very first check, the body never runs. The `tickets` program prints only `Sold out`. Run `CupsSkipped`: `cups` starts at 10, `cups < 8` is false, so nothing is poured and the output is `Cups ready: 10`. A loop that runs zero times is not an error; it compiles and runs.
2. **Initial conditions**. The values the loop's variables hold before the loop starts are its **initial conditions**, and the loop only works as intended if they are right. Ask what `lap` must start as for "Lap 1 through Lap 3" and what the program does if it starts at 4.
3. **Reusing a counter**. Run `ReuseCounter`: a second loop with the same counter `count` never runs, because the first loop left `count` at 4. The cool-downs never print. Run `ReuseCounterFixed` and name the fix: set the variable back to its starting value before the second loop.

MISCONCEPTION Every loop runs at least once

Students assume the body always runs once and then the condition decides. With `while`, the check comes first, so a false first check means zero iterations. Point at the first row of the slide 7 table: the check is row 1, before any body.

MISCONCEPTION Reusing a counter without resetting it

After one loop finishes, its variable still holds the value that made the condition false. A second loop that uses the same variable starts from that value (`ReuseCounter`). Have students say, for each loop they write, what the variable holds at its first check.

Day 2 direct instruction: infinite loops

1. **Definition** (2.7.A.2, slide 12). An **infinite loop** is a loop whose condition is true at every check, so the program never gets past it.
2. **Three ways to get one** (slide 12). No update (`n` never changes); an update that skips the target (`k != 10` with `k` going 3, 5, 7, 9, 11, never 10); an update placed outside the loop (the `t--` after the closing brace).

3. **Two questions for any loop.** Is the condition true at the start? Does the body move the variable toward making it false? Show `OddStepsFixed: distance != 0` would never end with steps of 2 from 7, so the program uses `> 0` and finishes with `-1`.
4. **Fixing the skip.** Replace `!=` with an inequality (`<` or `>`) so a value that jumps past the target still ends the loop. Show `MissingUpdateFixed` as the repair for the first case.

MISCONCEPTION Thinking every bad loop is infinite

A loop whose condition is false at the start does not run forever; it runs zero times. An infinite loop needs a condition that is true at every check. Ask both questions before calling a loop infinite.

MISCONCEPTION The loop stops in the middle of the body

The condition is checked only at the top. If the body makes the condition false partway through, the rest of that iteration still runs, so a `println` after the update still shows the value that would have failed the test. Show the effect with `UpdateFirstWrong` on Day 3.

Day 2 guided practice: does it end?

Slide 13 has four loops. Students write "ends after __ passes" or "never ends" for each, then compare with a partner and explain. Loop A (12, 8, 4, 0) ends after 3 passes. Loop B (10, 6, 2, -2, ...) never ends because `n` skips 0. Loop C never ends because nothing changes `n`. Loop D ends after 0 passes: `n > 10` is false at the start. Insist that students write why, not just the answer.

Day 2 independent practice

Students do worksheet question 3 (code tiles: rebuild the countdown) alone. The decoy tile checks for `count >= 0`, which would print `0...` as well. Questions 4 (snippet pool: weeks to save \$100, answer `savings < 100, += 15, weeks++`) and 6 (you're the compiler) are homework.

TEACHER NOTE

Question 6 has one infinite loop (snippet 5: `x` goes 7, 5, 3, 1, -1 and never equals 0), one zero-iteration loop (snippet 2, output 10), and two compile problems: `=` in the condition (snippet 1) and a local variable `count` read before it is given a value (snippet 3). Snippet 4 uses `String +=` and prints `hop!!!`; students have seen it in Topic 1.15.

Day 2 closure

Students write two questions they would ask about any `while` loop before running it: "Is the condition true at the start?" and "Does the body move toward making it false?"

Day 3 warm-up: how many laps?

Write on the board: `int lap = 1; while (lap < 5) printing "Lap " + lap`. Ask how many laps print and which is the last one. Students answer on whiteboards: 4 laps, the last is Lap 4. Then ask how to print Lap 5 as well.

Day 3 direct instruction: off-by-one errors and where the update goes

1. **Off-by-one** (2.7.A.4, slide 15). An off-by-one error runs a loop one time too many or one time too few. Fill in the slide 15 table: `k < 5` from 1 prints 1–4 (4 passes); `k <= 5` prints 1–5 (5 passes); starting at 0 with `k < 5` prints 0–4 (5 passes). Run `OffByOneShort` and `OffByOneLong`.
2. **Check the first and last value** the body sees. Say it as a routine: the starting value, whether the test includes the stop value, and the last value used.
3. **Update first or print first?** (slide 16). With the print first, `pts` goes 0, 10, 20; with the update first, it prints 10, 20, 30. Run `UpdateFirstWrong` (prints Lap 2 to Lap 6) and `UpdateFirstRight`. The statements before the update see the old value; the statements after see the new one.
4. **Fix it** (slide 17). Students do the "Fix the off-by-one" task in the guided practice below.

MISCONCEPTION Using `<` when you needed `<=`

The stop value is left out. When a task says "through 5", the last value must be allowed to pass the test. Have students test their condition on the last value the output needs, and on the value just after it.

MISCONCEPTION Moving the update changes the output

Students move `lap++` above the print to "tidy up" and the output starts at 2 and ends one too high. Any statement that uses the variable behaves according to whether the update ran before it.

Day 3 direct instruction: loops that stop on a goal

1. **When to loop** (slide 19). A loop fits when a step must repeat, especially when the number of repetitions depends on the data. "Warn once if the battery is low" is an `if`. "Double until at least 500" is a `while`: the followers count goes 30, 60, 120, 240, 480, 960, so the loop makes 5 passes and prints 5 weeks, 960 followers.
2. **The sentinel** (slide 20). A **sentinel** is a value that marks the end of the data. The program reads the first score before the loop, then reads the next value as the last statement of the body, so the condition checks every value before the body uses it. Run `ScoresUntilStop`: it prints each score and ends at `-1` without printing it.
3. **Read at the wrong point**. Run `ScoresReadFirstWrong`: with the read at the top of the body, the first score is lost and `-1` is printed as a score. Say plainly that the position of the read is the whole algorithm.

TEACHER NOTE

Do not teach `Scanner` itself; Topic 1.4 introduced input. The examples use a `Scanner` over a `String` so they run without typing. The sentinel idea matters here; the standard algorithms built on it (totals, averages, minimum, maximum) are Topic 2.9.

Day 3 guided practice: fix the off-by-one

Slide 17: the loop should print 2 4 6 8 10 but `even < 10` stops at 8. Students write what it prints (2 4 6 8) and change one thing (`<=>`). Then they start worksheet question 7 (one-token fix: `lap < 5` should be `lap <= 5`) and say aloud which first and last values the body sees before and after the fix.

Day 3 independent practice

Questions 5, 7, 8, 9, and 10 of the worksheet are homework. Question 5 matches four near-identical loops to their output (1 2 3, 1 2, 2 3 4, nothing); question 8 is two truths and a lie about the energy loop (the lie is claim 3: the body always finishes, so the last pass does reach `steps++`).

Day 3 closure

Slide 22. Partners explain how to tell, before running it, whether a `while` loop will stop, using the starter: "A while loop stops when the condition becomes false, so the body has to change a variable in the condition."

Day 3 exit ticket

Give the exit ticket for the last 5 minutes. Collect it as students leave.