Chapter 4.2: Flowcharts

Computer Fundamentals and Introduction to Programming

Symbols | Sequence | Decision | Loop | Verification

Sercan KÜLCÜ · Giresun University

© Sercan Külcü. All rights reserved. · Interactive HTML version of these slides: https://sercankulcu.github.io/files/computer/slides/Chapter_04_02_Flowcharts.html

INTRODUCTION

Learning Outcomes

Symbols

Recognize the basic flowchart symbols.

Flow

Order the steps with arrows.

Control

Show decisions and loops.

Translation

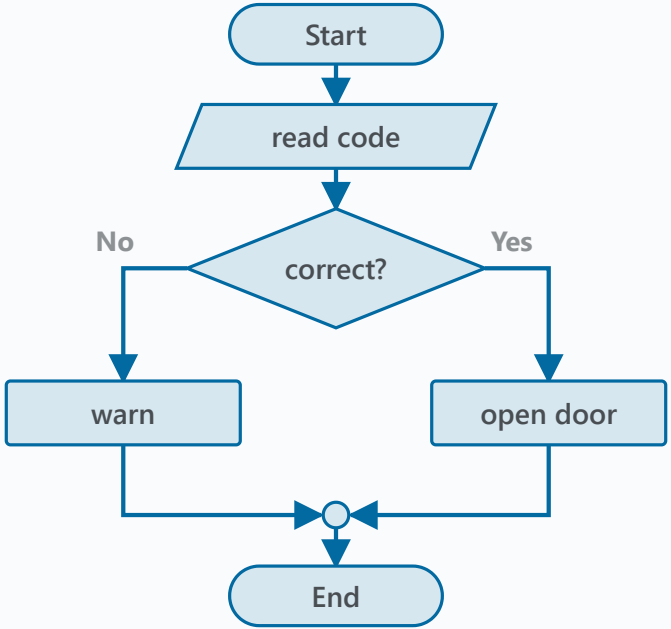
Turn a flowchart into pseudocode and a program.

BASICS

What Is a Flowchart?

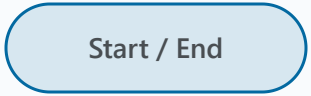
- A diagram of the order in which an algorithm's steps run.
- It lets us discuss complex logic apart from code.
- Symbols show the **kind** of each step; arrows show the **direction** of execution.

The example: an everyday task (a door code) as an algorithm. Two paths leave the decision box and join again.



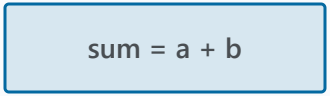
SYMBOLS

Basic Symbols



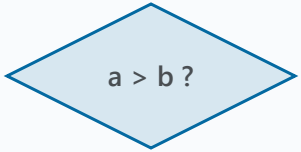
Terminal

One start and at least one end per chart



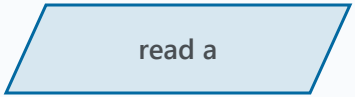
Process

Calculation, assignment



Decision

Two exits: Yes / No



Input / output

Get data (read), show it (print)



Flow line

Direction of execution; the arrowhead matters



Connector

Paths merge / page break

The **shape** tells you the kind of step; inside, write only a short phrase for that step.

Symbol and Role

Symbol	Role	Example
 Oval	Start or end	Start
 Rectangle	Process (calculation, assignment)	sum = a + b
 Diamond	Decision (condition)	a > b ?
 Parallelogram	Input / output	read a
 Arrow	Direction of execution	go to the next step

A decision holds a **question** (a > b ?); a process holds a **command** (sum = a + b).

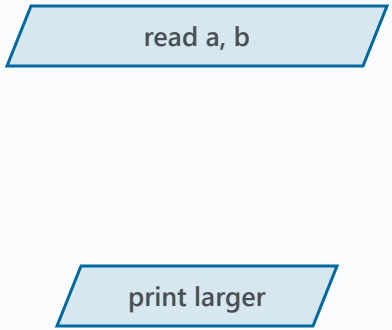
More Symbols

Symbol	Role	Example
 Off-page connector	The chart continues on another page; label it with page and point.	2-A
 Predefined process	Calls a subprocess drawn separately.	Compute average
 Preparation	Gives a loop counter's start and limit in one symbol.	i = 1 → n
 Document	Printed output or a report.	Grade sheet
 Data store	Data kept on disk or in a database.	Student records
 Annotation	Attaches a note to a symbol with a dashed line; not part of the flow.	grade 0–100

These symbols come from ISO 5807 and appear in common drawing tools. Five basic symbols are enough for a small algorithm; the extra ones help when a chart grows across pages or must show external media such as files and reports.

Building a Flowchart: The Larger of Two Numbers

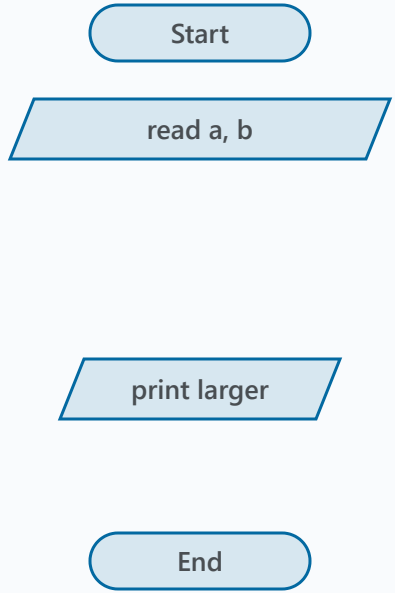
1. Fix input and output
2. Order the steps
3. Separate decisions
4. Connect arrows
5. Trace an example



Problem: print the larger of two numbers. **Input** a and b (parallelogram: read), **output** the larger one (parallelogram: print).

Building a Flowchart: The Larger of Two Numbers

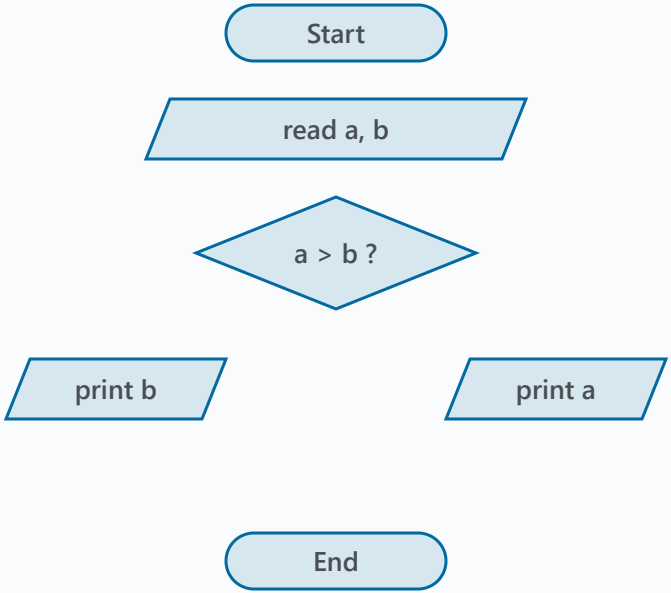
1. Fix input and output
2. Order the steps
3. Separate decisions
4. Connect arrows
5. Trace an example



Add Start and End; order the steps top to bottom: read first, print last.

Building a Flowchart: The Larger of Two Numbers

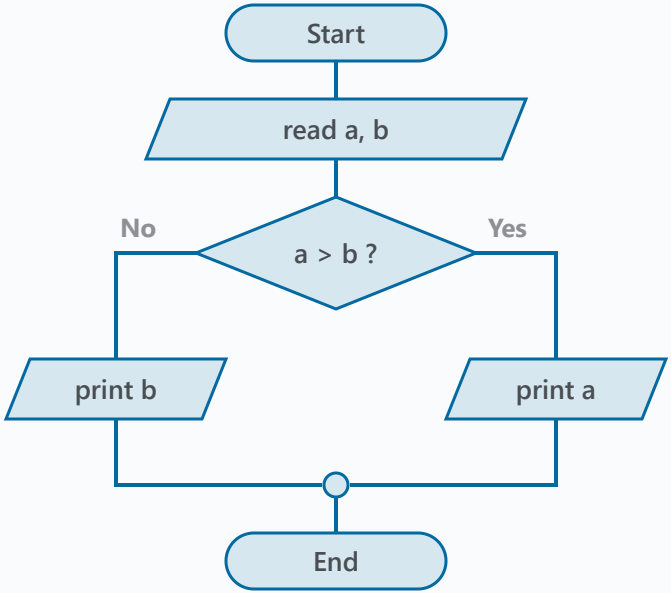
1. Fix input and output
2. Order the steps
3. Separate decisions
4. Connect arrows
5. Trace an example



Which number to print depends on a question: add the decision **a > b ?** and split the output into two boxes.

Building a Flowchart: The Larger of Two Numbers

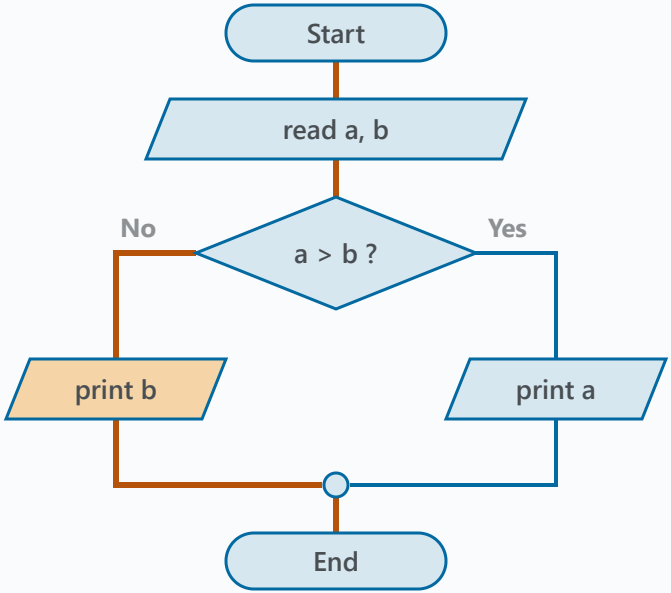
1. Fix input and output
2. Order the steps
3. Separate decisions
4. Connect arrows
5. Trace an example



Arrows give the direction of execution. Label the decision exits **Yes/No** and merge the paths at a connector.

Building a Flowchart: The Larger of Two Numbers

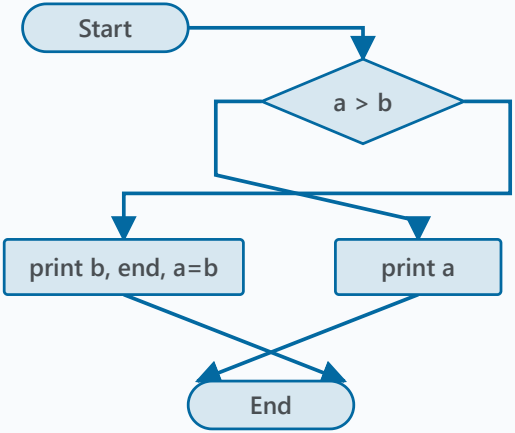
1. Fix input and output
2. Order the steps
3. Separate decisions
4. Connect arrows
5. Trace an example



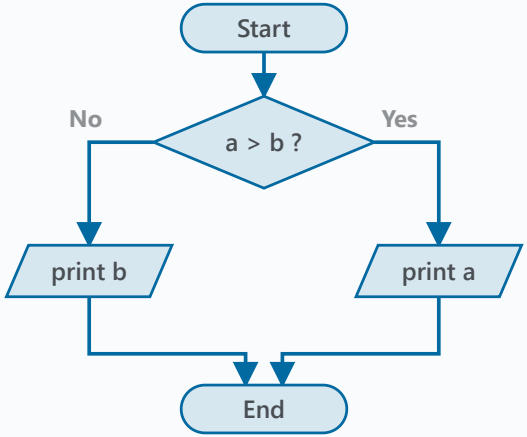
Trace an example: **a = 4, b = 9** → 4 > 9 is false → No path → "print b" → output **9**. The orange path is the one traced.

DESIGN

Readability Rules



✗ Crossing arrows, unlabeled decision, three steps in one box



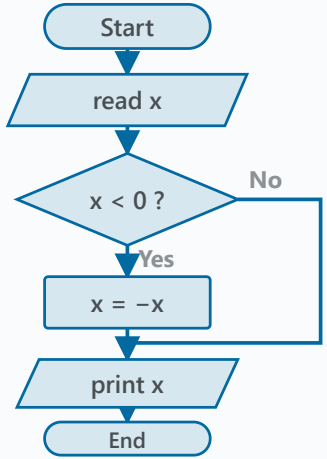
✓ Top to bottom, Yes/No labels, one step per box

- Flow should run top to bottom and left to right wherever possible.
- Decision exits need meaningful labels such as Yes/No; keep crossing arrows to a minimum.
- Put one basic step in each symbol; split large charts into subprocesses.

Flowchart or Pseudocode?

	Flowchart	Pseudocode
Form	Symbols and arrows	Short indented lines
Strength	Branches and loops are visible at a glance	Quick to write, very close to code
Weakness	Outgrows the page; tedious to change	Structure depends on indentation
Best for	Learning, presentations, business processes	Long algorithms, right before coding

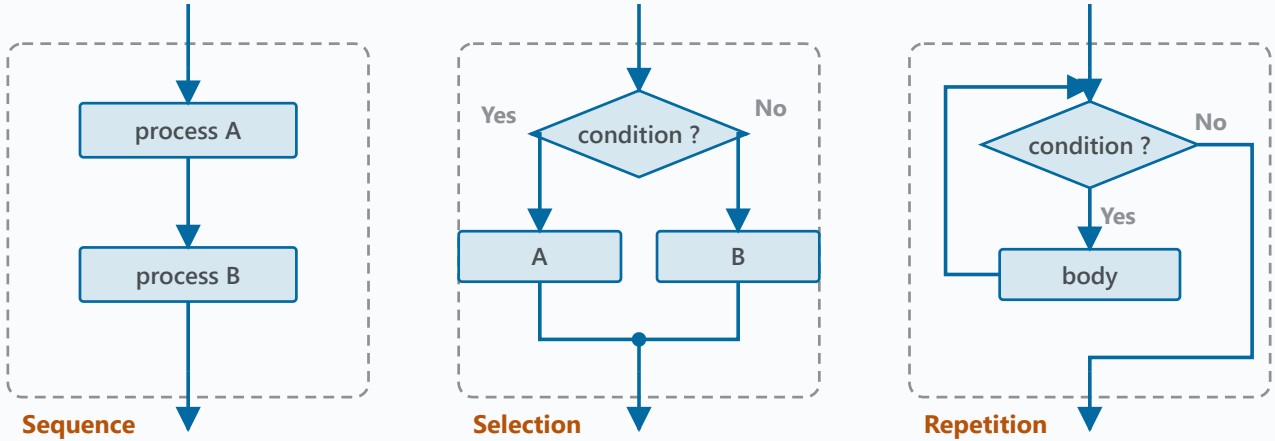
Both describe the same algorithm, and each translates line by line into the other. Pseudocode has no fixed standard; clarity is what counts.




```
read x
if x < 0 then
    x = -x
print x
```

DESIGN

The Three Structures of Structured Programming



- Böhm and Jacopini (1966): every algorithm can be written with just **sequence**, **selection** and **repetition**.
- Each structure has **one entry and one exit**, so a structure can replace any box inside another one (nesting).
- Arrows that jump to arbitrary points (**spaghetti** flow) make charts hard to read and test. In Java, `goto` is a reserved word but cannot be used.

DESIGN

Flowchart Drawing Tools

Pen and paper

Fastest for a first draft; easy to erase and redraw. Exam charts are drawn by hand too.

Word and PowerPoint

The **Flowchart** group under **Insert** → **Shapes**; connector arrows follow when a box moves.

draw.io (diagrams.net)

Free and browser-based; ready-made flowchart shapes and export to PNG, SVG and PDF.

Microsoft Visio

A commercial tool for business process and workflow diagrams.

Flowgorithm

Runs the chart, shows variables step by step and converts it to code in many languages, including Java.

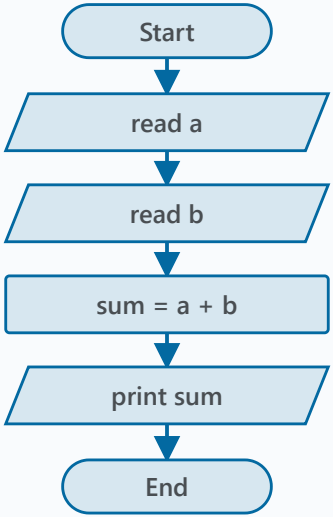
Mermaid

You describe the chart in text and the tool draws it; common in documents and code repositories.

Whatever the tool, the meaning of the symbols and the readability rules stay the same.

KINDS OF FLOW

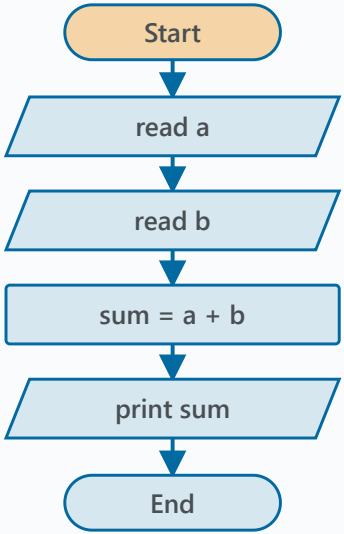
Sequential Flow




```
Start
  read a
  read b
  sum = a + b
  print sum
End
```

- Each step runs after the previous one finishes.
- No branches or repetition: one entry, one exit.
- Arrows always point down.
- Order matters: `sum` cannot be computed before a and b are read.

Sequential Flow Step by Step (a = 4, b = 7)




```
▶ Start
2   read a
3   read b
4   sum = a + b
5   print sum
6 End
```

VARIABLES

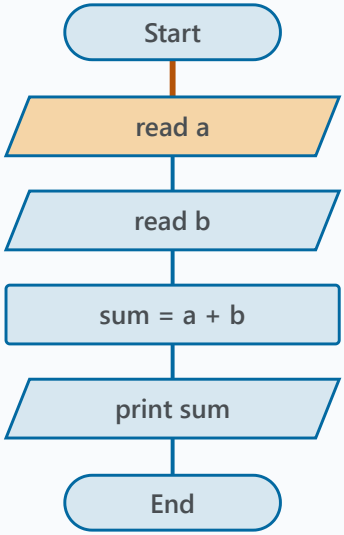
—

OUTPUT

Execution begins at the **Start** box. No variables exist yet.

Step 1 / 6

Sequential Flow Step by Step (a = 4, b = 7)




```
1 Start
▶   read a
3   read b
4   sum = a + b
5   print sum
6 End
```

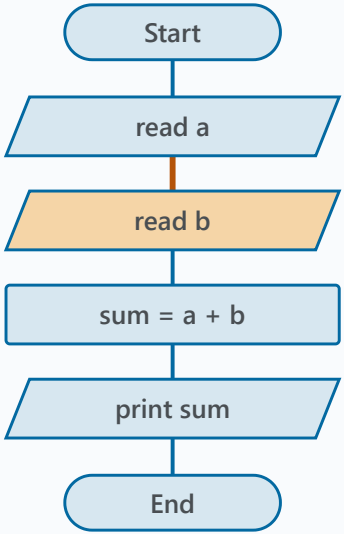
VARIABLES

a = 4

OUTPUT

Input box: the user enters **4** → **a = 4**.

Sequential Flow Step by Step (a = 4, b = 7)



The arrow leads to the next box: read **b = 7**.

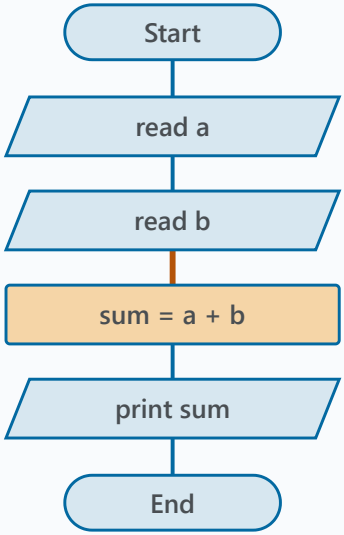
```
1 Start
2   read a
▶   read b
4   sum = a + b
5   print sum
6 End
```

VARIABLES

a = 4 b = 7

OUTPUT

Sequential Flow Step by Step (a = 4, b = 7)



Process box: **sum = 4 + 7 = 11**.

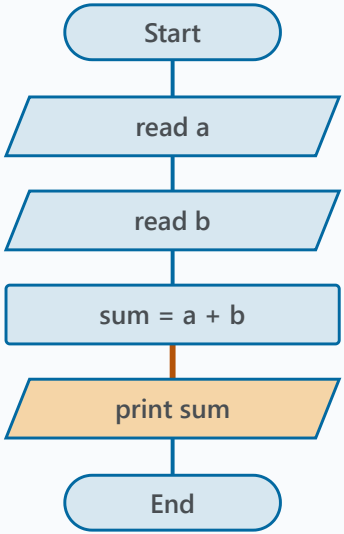
```
1 Start
2   read a
3   read b
▶   sum = a + b
5   print sum
6 End
```

VARIABLES

a = 4 | b = 7 | sum = 11

OUTPUT

Sequential Flow Step by Step (a = 4, b = 7)



Output box: **11** is printed.

```
1 Start
2   read a
3   read b
4   sum = a + b
▶   print sum
6 End
```

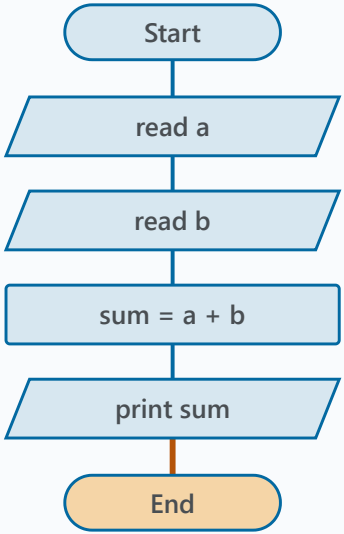
VARIABLES

a = 4 | b = 7 | sum = 11

OUTPUT

```
11
```

Sequential Flow Step by Step (a = 4, b = 7)




```
1 Start
2   read a
3   read b
4   sum = a + b
5   print sum
▶ End
```

VARIABLES

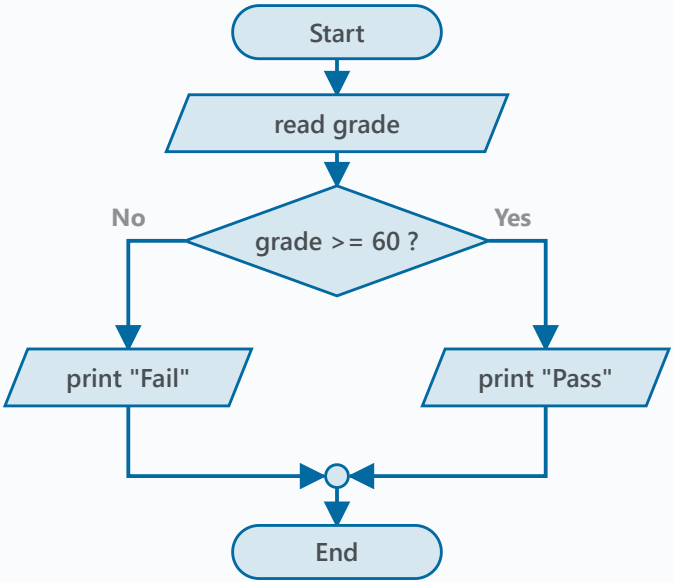
a = 4 | b = 7 | sum = 11

OUTPUT

```
11
```

We reach End. Every box ran **exactly once**: the mark of sequential flow.

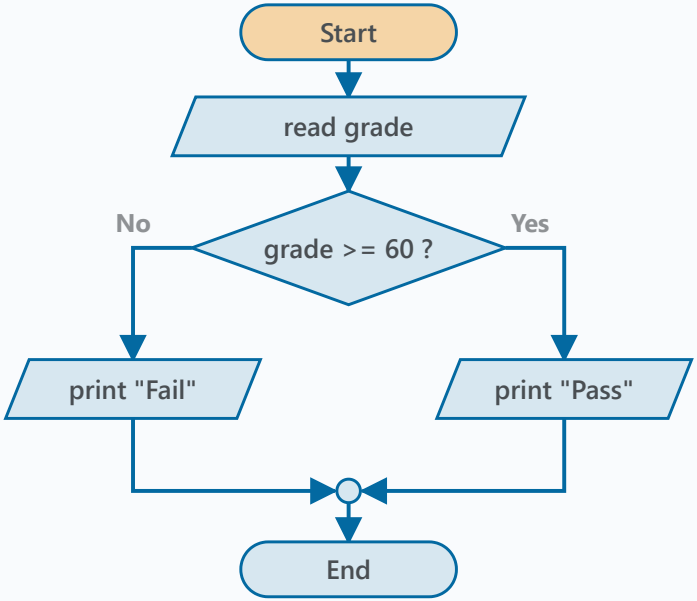
Conditional Flow




```
Start
  read grade
  grade >= 60 ?
    Yes → print "Pass"
    No → print "Fail"
End
```

- Two paths leave the decision; each run follows **only one** of them.
- The paths merge again at the connector (●) below.
- Test the boundary separately: with grade = 60, `>=` gives “Pass” but `>` gives “Fail”.

Conditional Flow Step by Step (grade = 72, then 45)




```
▶ Start
2   read grade
3   grade >= 60 ?
4     Yes → print "Pass"
5     No → print "Fail"
6 End
```

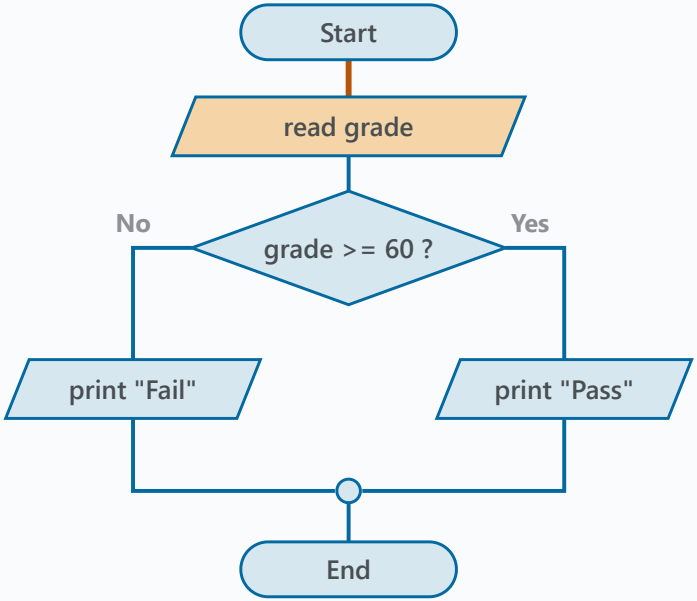
VARIABLES

—

OUTPUT

Run 1 (grade = 72): execution begins at Start.

Conditional Flow Step by Step (grade = 72, then 45)




```
1 Start
▶   read grade
3   grade >= 60 ?
4     Yes → print "Pass"
5     No → print "Fail"
6 End
```

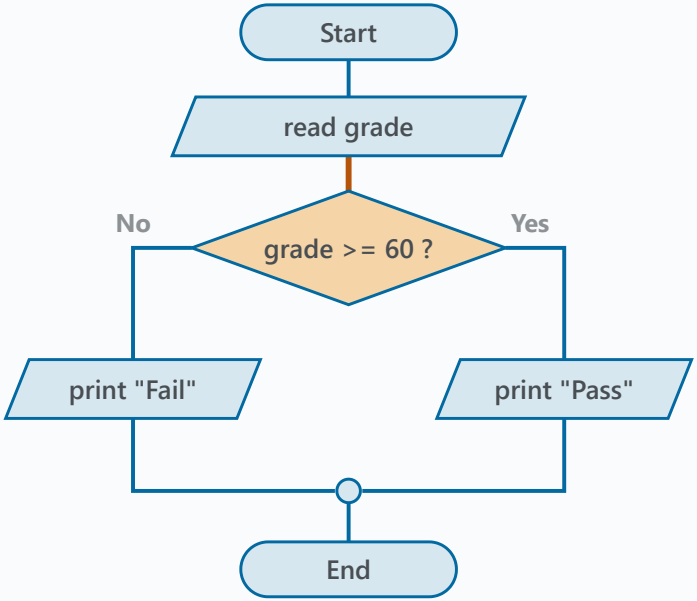
VARIABLES

`grade = 72`

OUTPUT

The user enters **72** → **grade = 72**.

Conditional Flow Step by Step (grade = 72, then 45)




```
1 Start
2   read grade
▶   grade >= 60 ?
4     Yes → print "Pass"
5     No → print "Fail"
6 End
```

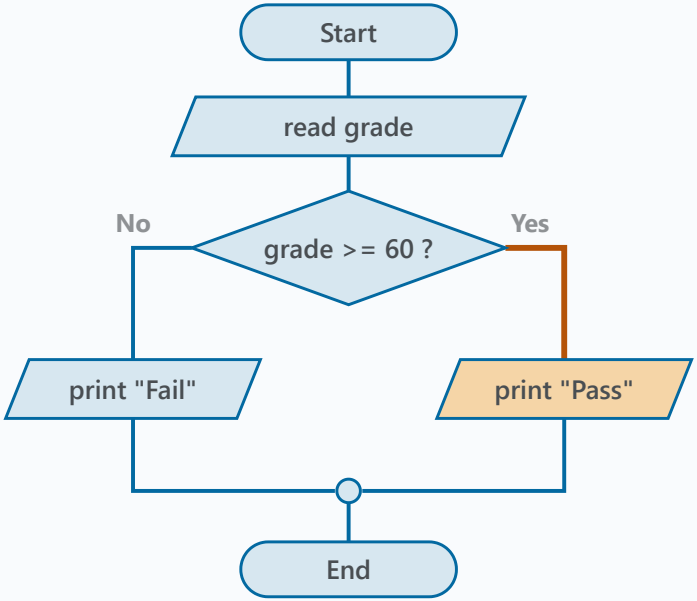
VARIABLES

`grade = 72` `cond = true`

OUTPUT

Decision: **72 >= 60** → **true**. Flow continues along the **Yes** arrow.

Conditional Flow Step by Step (grade = 72, then 45)




```
1 Start
2   read grade
3   grade >= 60 ?
▶     Yes → print "Pass"
5     No → print "Fail"
6 End
```

VARIABLES

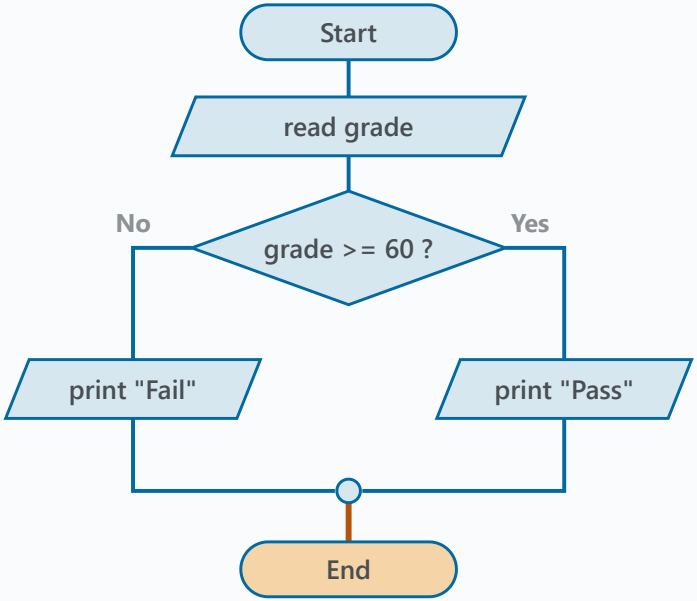
`grade = 72` `cond = true`

OUTPUT

```
Pass
```

"Pass" is printed. The box on the No branch **never runs** this time.

Conditional Flow Step by Step (grade = 72, then 45)




```
1 Start
2   read grade
3   grade >= 60 ?
4     Yes → print "Pass"
5     No → print "Fail"
▶ End
```

VARIABLES

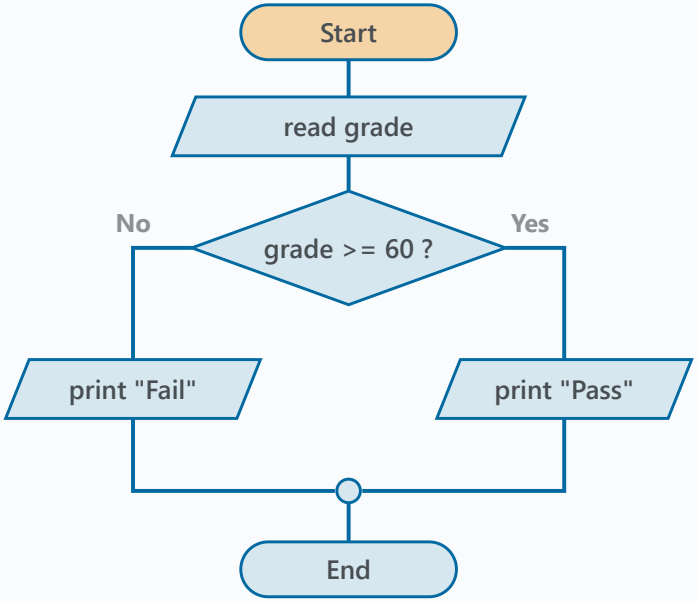
`grade = 72` `cond = true`

OUTPUT

```
Pass
```

The two paths merge at the connector and reach End. Now run again with grade = 45 to see the other branch.

Conditional Flow Step by Step (grade = 72, then 45)




```
▶ Start
2   read grade
3   grade >= 60 ?
4     Yes → print "Pass"
5     No → print "Fail"
6 End
```

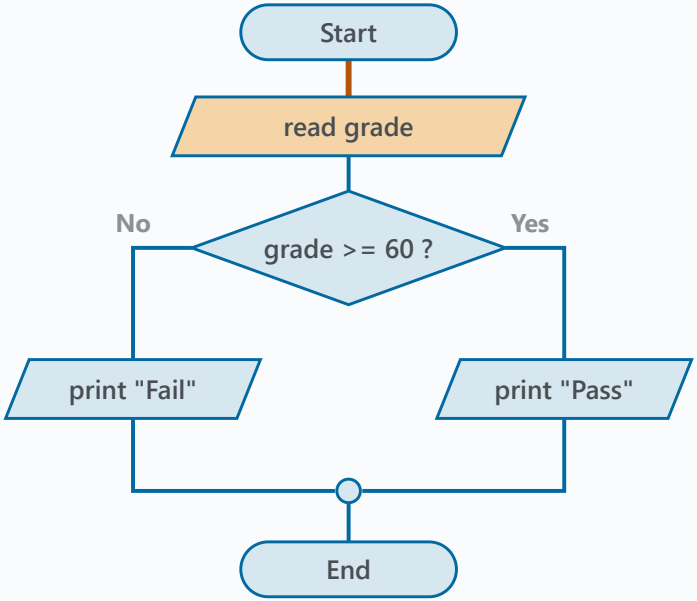
VARIABLES

—

OUTPUT

Run 2 (grade = 45): execution begins at Start.

Conditional Flow Step by Step (grade = 72, then 45)




```
1 Start
▶   read grade
3   grade >= 60 ?
4     Yes → print "Pass"
5     No → print "Fail"
6 End
```

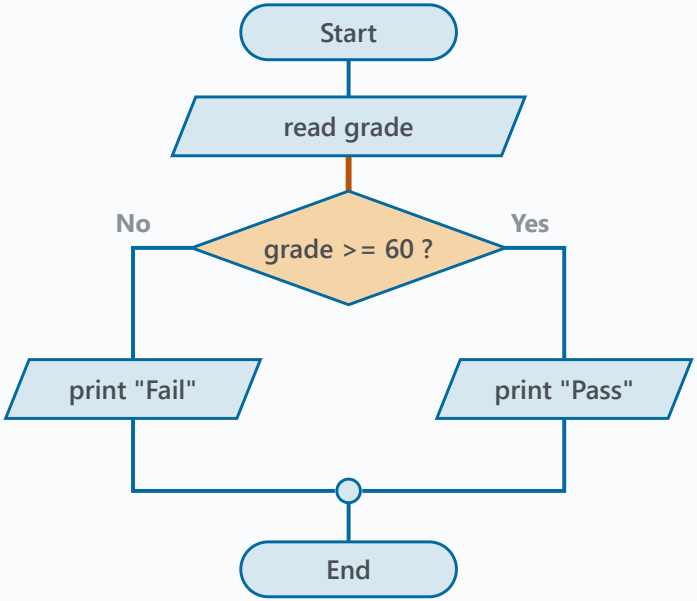
VARIABLES

`grade = 45`

OUTPUT

The user enters **45** → **grade = 45**.

Conditional Flow Step by Step (grade = 72, then 45)




```
1 Start
2   read grade
▶   grade >= 60 ?
4     Yes → print "Pass"
5     No → print "Fail"
6 End
```

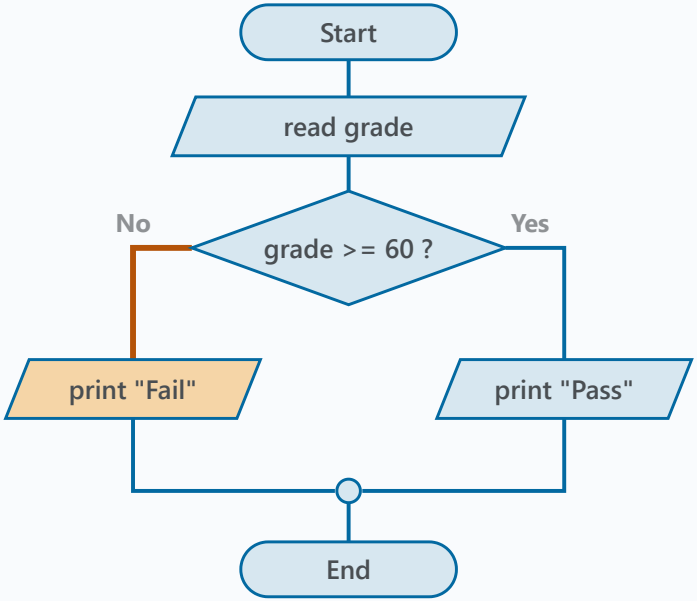
VARIABLES

`grade = 45` `cond = false`

OUTPUT

Decision: **45 >= 60** → **false**. Flow continues along the **No** arrow.

Conditional Flow Step by Step (grade = 72, then 45)




```
1 Start
2   read grade
3   grade >= 60 ?
4     Yes → print "Pass"
▶     No → print "Fail"
6 End
```

VARIABLES

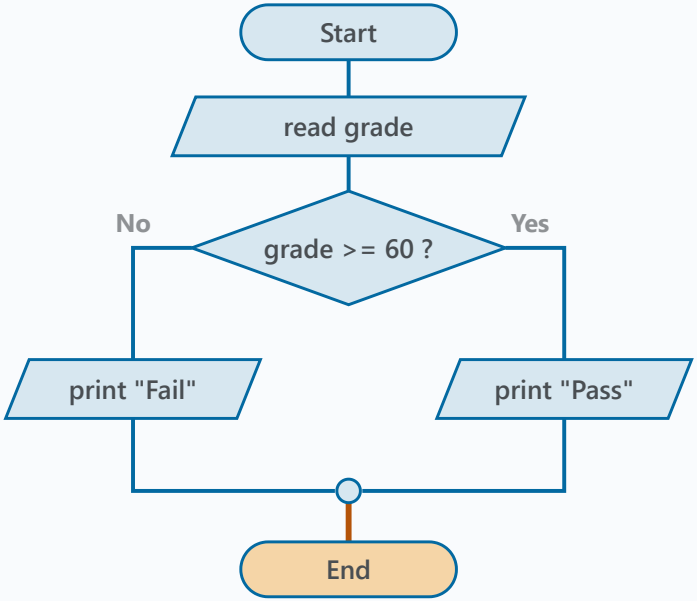
`grade = 45` `cond = false`

OUTPUT

```
Fail
```

"Fail" is printed. The box on the Yes branch **never runs** this time.

Conditional Flow Step by Step (grade = 72, then 45)




```
1 Start
2   read grade
3   grade >= 60 ?
4     Yes → print "Pass"
5     No → print "Fail"
▶ End
```

VARIABLES

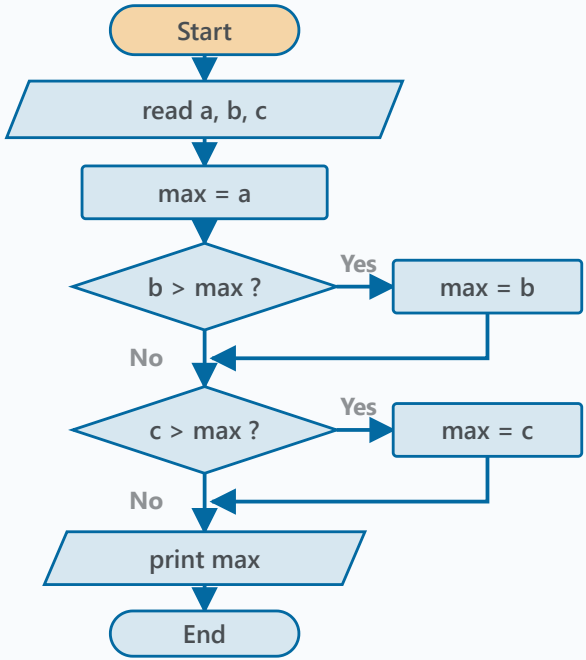
`grade = 45` `cond = false`

OUTPUT

```
Fail
```

The two paths merge at the connector and reach End. Two inputs exercised both branches.

Largest of Three Numbers Step by Step




```
▶ Start
2   read a, b, c
3   max = a
4   b > max ?  Yes → max = b
5   c > max ?  Yes → max = c
6   print max
7 End
```

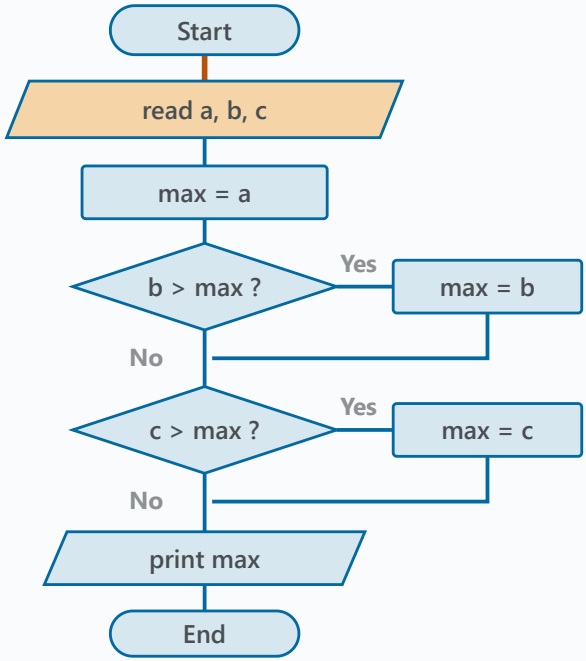
VARIABLES

—

OUTPUT

Run 1 (4, 9, 7): Start.

Largest of Three Numbers Step by Step




```
1 Start
▶   read a, b, c
3   max = a
4   b > max ?  Yes → max = b
5   c > max ?  Yes → max = c
6   print max
7 End
```

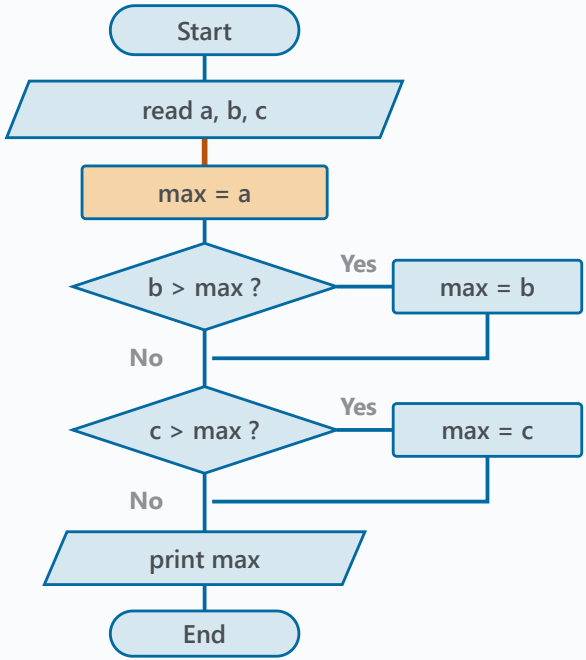
VARIABLES

a = 4 b = 9 c = 7

OUTPUT

Read three numbers: a = 4, b = 9, c = 7.

Largest of Three Numbers Step by Step




```
1 Start
2   read a, b, c
▶   max = a
4   b > max ?  Yes → max = b
5   c > max ?  Yes → max = c
6   print max
7 End
```

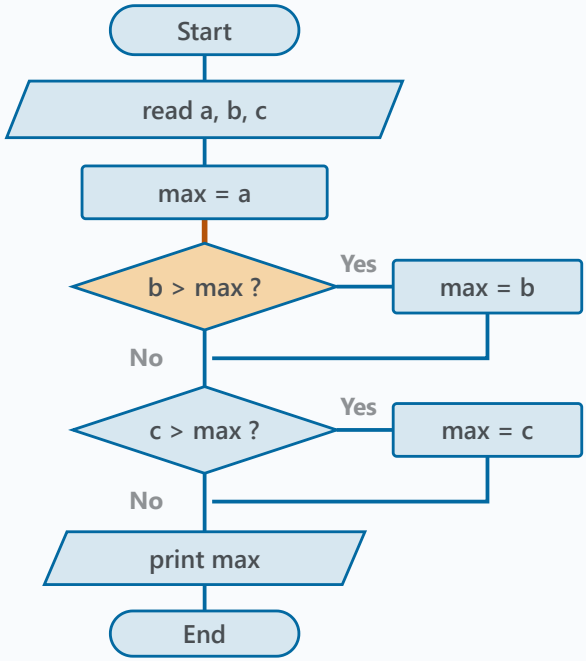
VARIABLES

a = 4 | b = 9 | c = 7 | max = 4

OUTPUT

First candidate for the largest: **max = a = 4**.

Largest of Three Numbers Step by Step




```
1 Start
2   read a, b, c
3   max = a
▶   b > max ?  Yes → max = b
5   c > max ?  Yes → max = c
6   print max
7 End
```

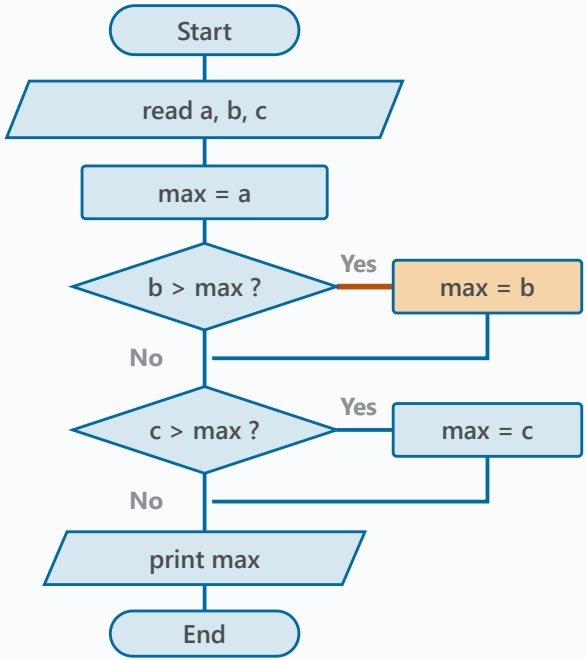
VARIABLES

a = 4 | b = 9 | c = 7 | max = 4

OUTPUT

Decision: **9 > 4** → **true** → Yes.

Largest of Three Numbers Step by Step




```
1 Start
2   read a, b, c
3   max = a
▶   b > max ?  Yes → max = b
5   c > max ?  Yes → max = c
6   print max
7 End
```

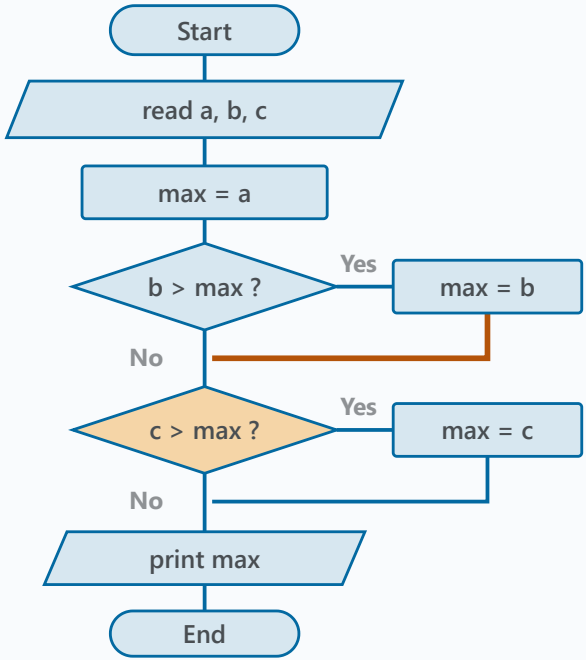
VARIABLES

a = 4 | b = 9 | c = 7 | max = 9

OUTPUT

max = b = 9. The path merges below.

Largest of Three Numbers Step by Step




```
1 Start
2   read a, b, c
3   max = a
4   b > max ?  Yes → max = b
▶   c > max ?  Yes → max = c
6   print max
7 End
```

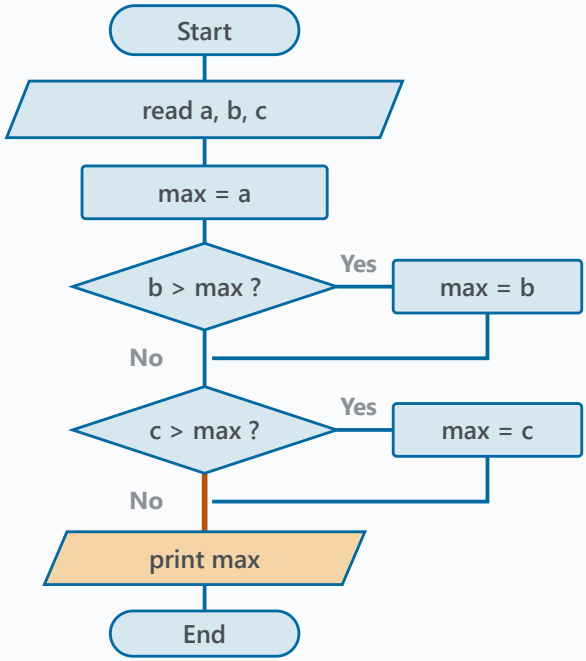
VARIABLES

a = 4 | b = 9 | c = 7 | max = 9

OUTPUT

Decision: **7 > 9** → **false** → No.

Largest of Three Numbers Step by Step




```
1 Start
2   read a, b, c
3   max = a
4   b > max ?  Yes → max = b
5   c > max ?  Yes → max = c
▶   print max
7 End
```

VARIABLES

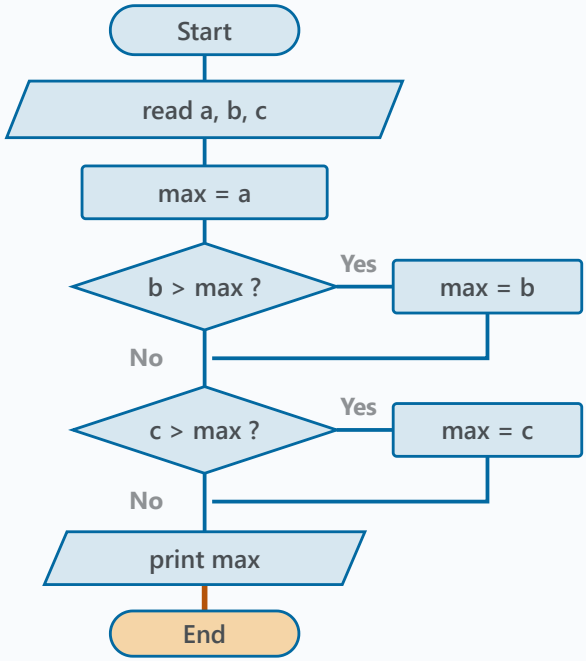
a = 4 | b = 9 | c = 7 | max = 9

OUTPUT

```
9
```

9 is printed.

Largest of Three Numbers Step by Step




```
1 Start
2   read a, b, c
3   max = a
4   b > max ?  Yes → max = b
5   c > max ?  Yes → max = c
6   print max
▶ End
```

VARIABLES

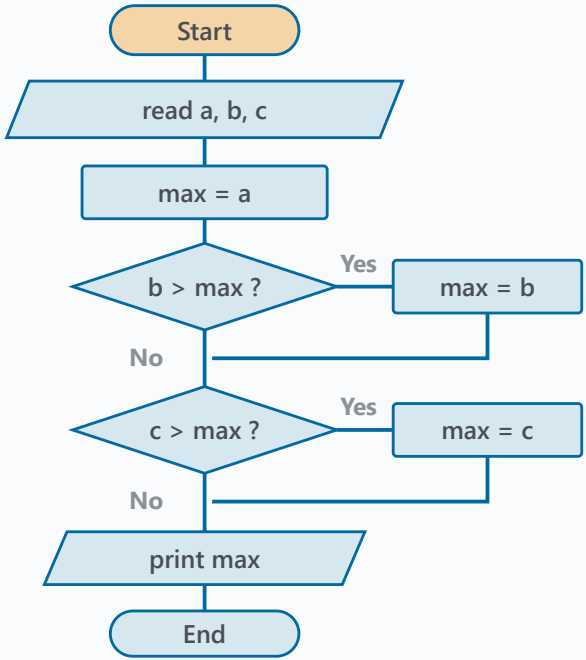
a = 4 | b = 9 | c = 7 | max = 9

OUTPUT

```
9
```

End. Now try an input where c is the largest.

Largest of Three Numbers Step by Step




```
▶ Start
2   read a, b, c
3   max = a
4   b > max ?  Yes → max = b
5   c > max ?  Yes → max = c
6   print max
7 End
```

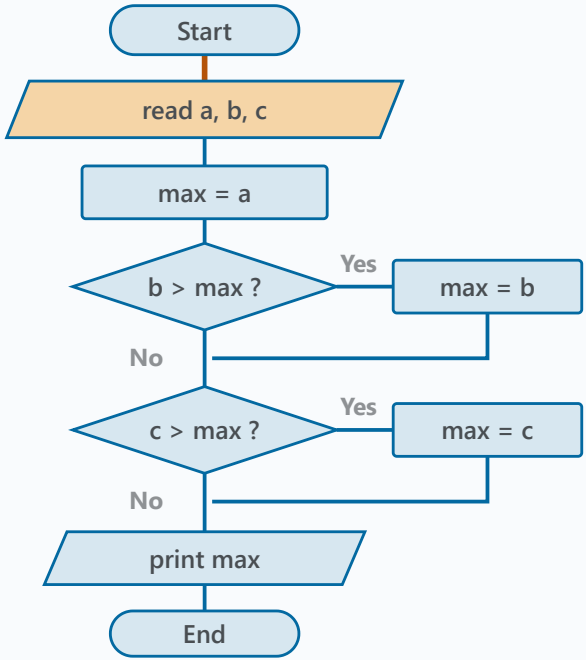
VARIABLES

—

OUTPUT

Run 2 (3, 2, 8): Start.

Largest of Three Numbers Step by Step




```
1 Start
▶   read a, b, c
3   max = a
4   b > max ?  Yes → max = b
5   c > max ?  Yes → max = c
6   print max
7 End
```

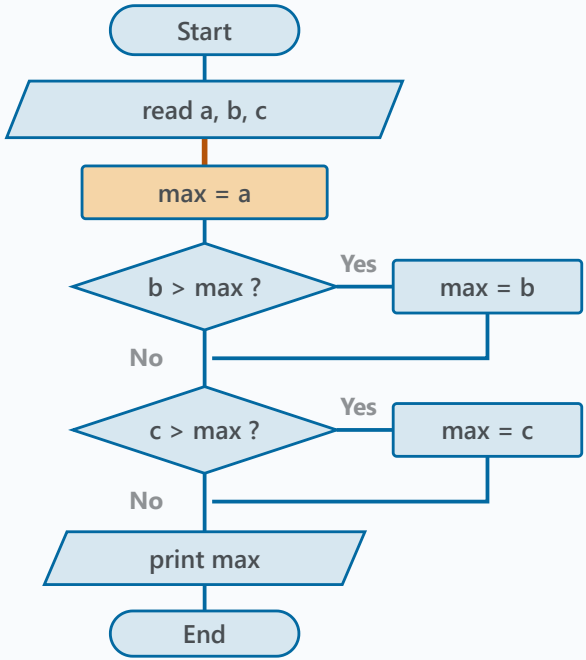
VARIABLES

a = 3 b = 2 c = 8

OUTPUT

Read three numbers: a = 3, b = 2, c = 8.

Largest of Three Numbers Step by Step




```
1 Start
2   read a, b, c
▶   max = a
4   b > max ?  Yes → max = b
5   c > max ?  Yes → max = c
6   print max
7 End
```

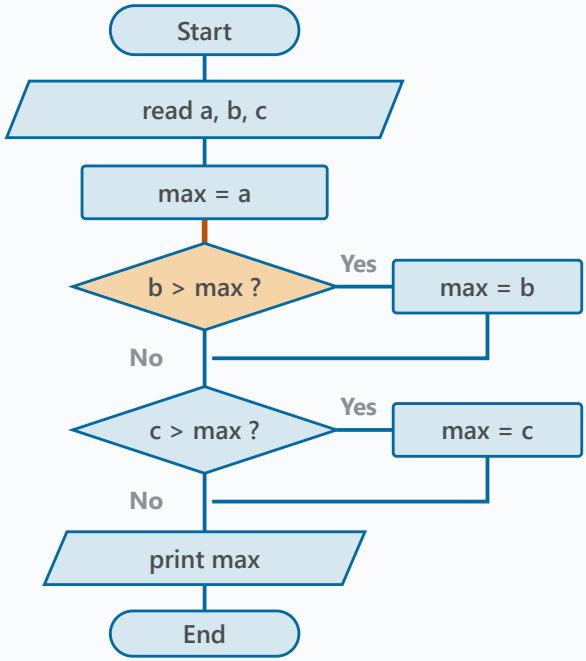
VARIABLES

a = 3 | b = 2 | c = 8 | max = 3

OUTPUT

First candidate for the largest: **max = a = 3**.

Largest of Three Numbers Step by Step




```
1 Start
2   read a, b, c
3   max = a
▶   b > max ?  Yes → max = b
5   c > max ?  Yes → max = c
6   print max
7 End
```

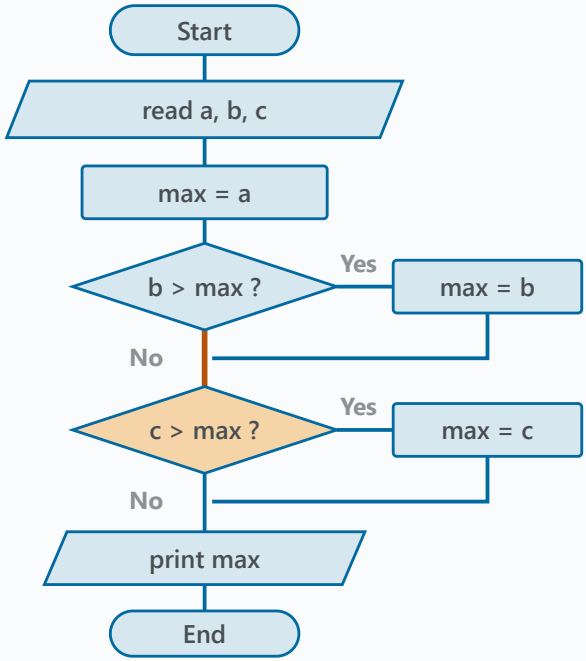
VARIABLES

a = 3 | b = 2 | c = 8 | max = 3

OUTPUT

Decision: **2 > 3** → **false** → No; max stays.

Largest of Three Numbers Step by Step




```
1 Start
2   read a, b, c
3   max = a
4   b > max ?  Yes → max = b
▶   c > max ?  Yes → max = c
6   print max
7 End
```

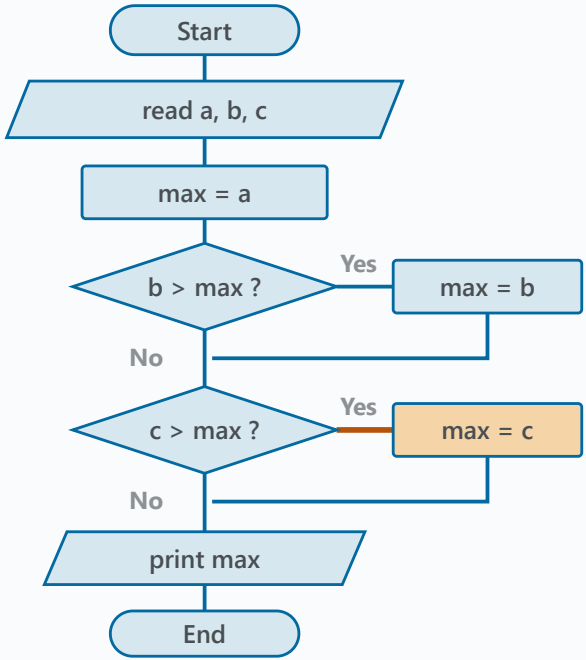
VARIABLES

a = 3 | b = 2 | c = 8 | max = 3

OUTPUT

Decision: **8 > 3** → **true** → Yes.

Largest of Three Numbers Step by Step




```
1 Start
2   read a, b, c
3   max = a
4   b > max ?  Yes → max = b
▶   c > max ?  Yes → max = c
6   print max
7 End
```

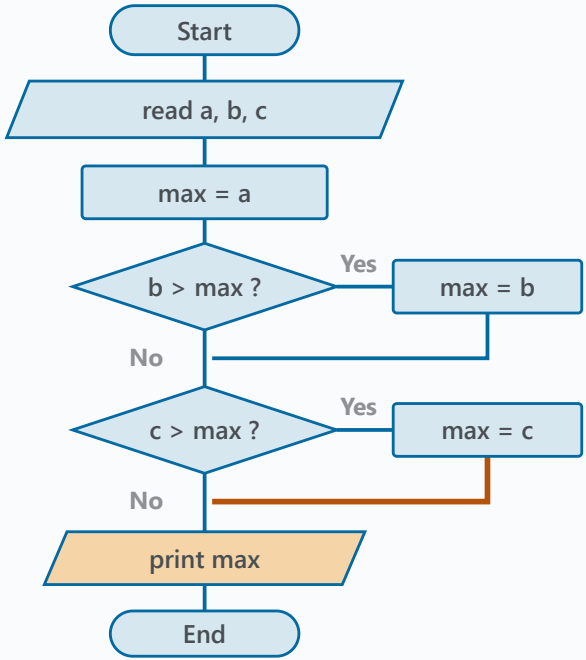
VARIABLES

a = 3 | b = 2 | c = 8 | max = 8

OUTPUT

max = c = 8.

Largest of Three Numbers Step by Step




```
1 Start
2   read a, b, c
3   max = a
4   b > max ?  Yes → max = b
5   c > max ?  Yes → max = c
▶   print max
7 End
```

VARIABLES

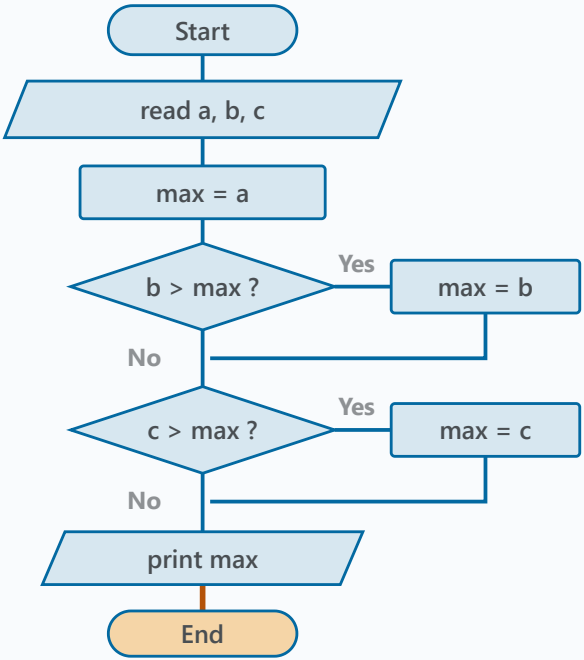
a = 3 | b = 2 | c = 8 | max = 8

OUTPUT

```
8
```

8 is printed.

Largest of Three Numbers Step by Step



End. Each decision is asked only once; the method stays the same with more numbers.

```
1 Start
2   read a, b, c
3   max = a
4   b > max ?  Yes → max = b
5   c > max ?  Yes → max = c
6   print max
▶ End
```

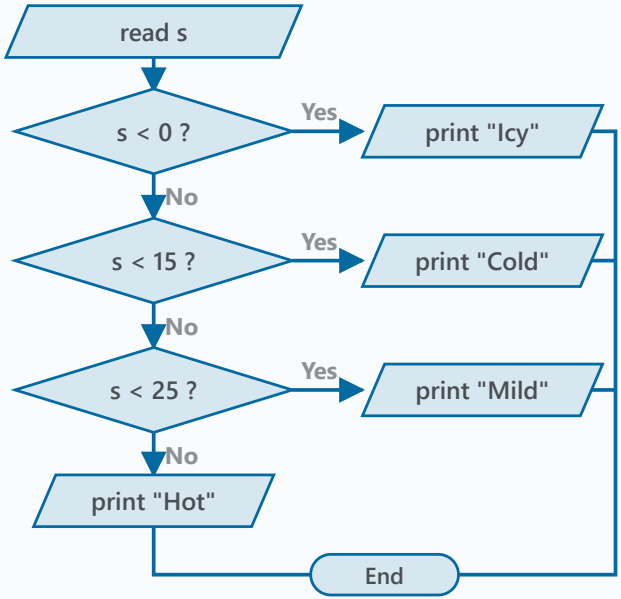
VARIABLES

a = 3 | b = 2 | c = 8 | max = 8

OUTPUT

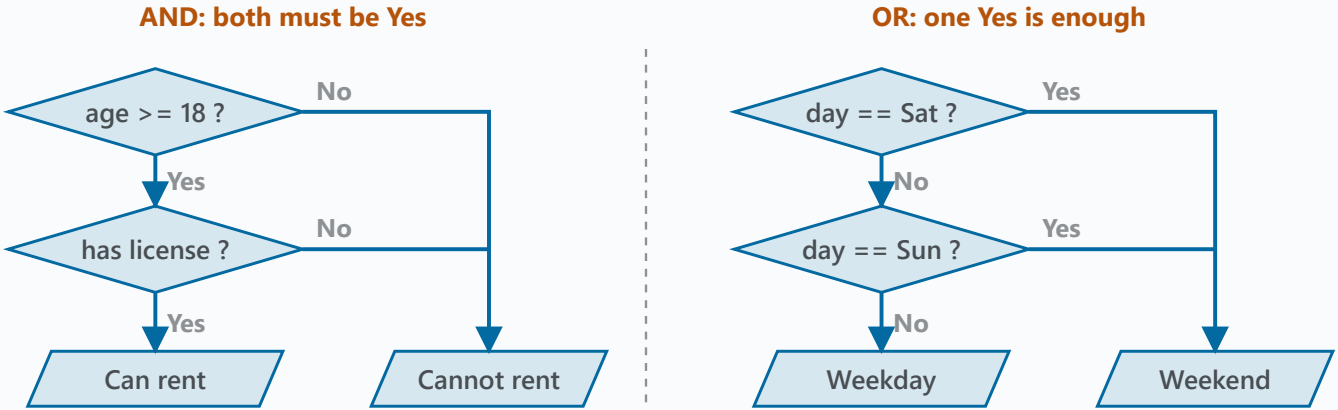
```
8
```

Multi-Way Decision: A Decision Chain



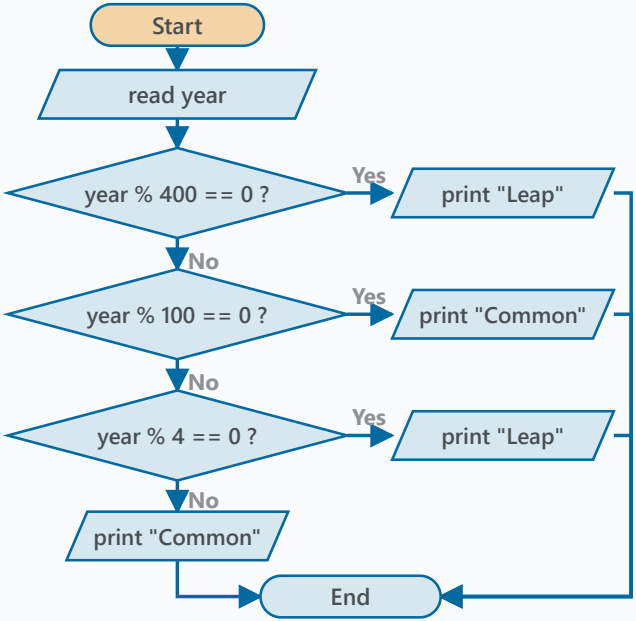
- With more than two options, decisions form a **chain**: each one sits on the **No** branch of the previous one.
- The branch of the first true condition runs; later decisions are never asked. For s = 18 three decisions are asked: 18 < 0 and 18 < 15 are false, 18 < 25 is true → **Mild**.
- Each decision checks only the upper bound; the lower bound is already known (when s < 15 is asked, s ≥ 0 is certain).
- **Order matters:** if s < 25 came first, s = −5 would also print “Mild”.
- n options need n − 1 decisions. The Java equivalent is an `if … else if … else` chain.

Compound Conditions: AND / OR



- A compound condition can also go in a single decision box: **age >= 18 AND has license ?**. Expand it into two decisions when the chart needs detail.
- **AND**: if the first decision is No, the second is **never asked**; the result is already No. **OR**: if the first is Yes, the second is unnecessary. Java's `&&` and `||` work the same way: short-circuit evaluation.
- To test, exercise both exits of every decision: in the AND example, (20, yes), (20, no) and (16, —) run three different paths.

Leap Year Step by Step (2024, 1900, 2000)




```
▶ Start
2   read year
3   year % 400 == 0 ?  Yes → print "Leap"
4   year % 100 == 0 ?  Yes → print "Common"
5   year % 4 == 0 ?    Yes → print "Leap"
6   No → print "Common"
7 End
```

VARIABLES

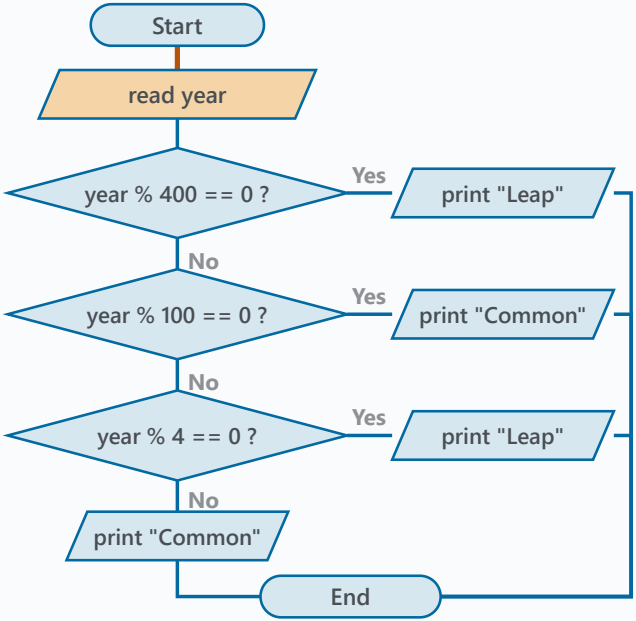
—

OUTPUT

The leap-year rule: years divisible by 4 are leap years, except those divisible by 100, unless also divisible by 400. A chain of three decisions. **Run 1: 2024.**

Step 1 / 16

Leap Year Step by Step (2024, 1900, 2000)




```
1 Start
▶   read year
3   year % 400 == 0 ?  Yes → print "Leap"
4   year % 100 == 0 ?  Yes → print "Common"
5   year % 4 == 0 ?    Yes → print "Leap"
6   No → print "Common"
7 End
```

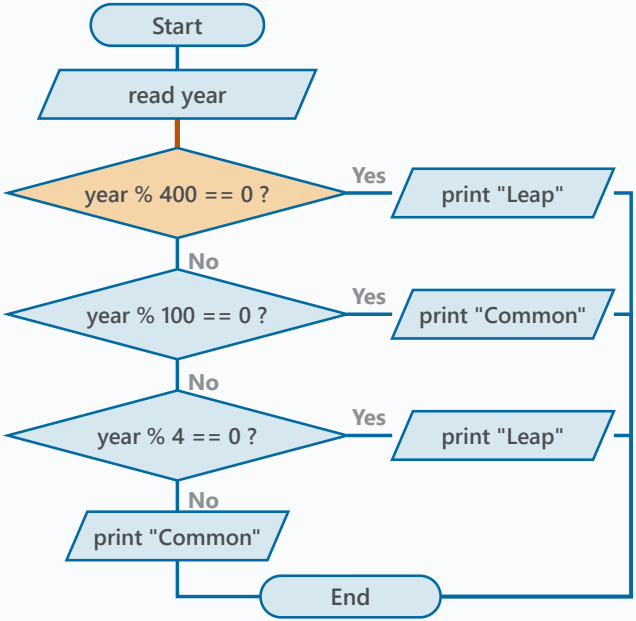
VARIABLES

`year = 2024`

OUTPUT

Read year = **2024**.

Leap Year Step by Step (2024, 1900, 2000)




```
1 Start
2   read year
▶   year % 400 == 0 ?  Yes → print "Leap"
4   year % 100 == 0 ?  Yes → print "Common"
5   year % 4 == 0 ?    Yes → print "Leap"
6   No → print "Common"
7 End
```

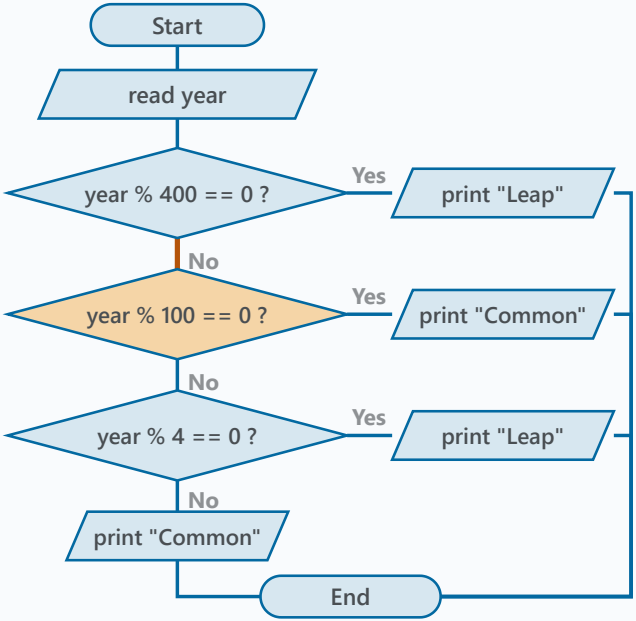
VARIABLES

`year = 2024`

OUTPUT

2024 % 400 = 24 → **false** → No.

Leap Year Step by Step (2024, 1900, 2000)




```
1 Start
2   read year
3   year % 400 == 0 ?  Yes → print "Leap"
▶   year % 100 == 0 ?  Yes → print "Common"
5   year % 4 == 0 ?    Yes → print "Leap"
6   No → print "Common"
7 End
```

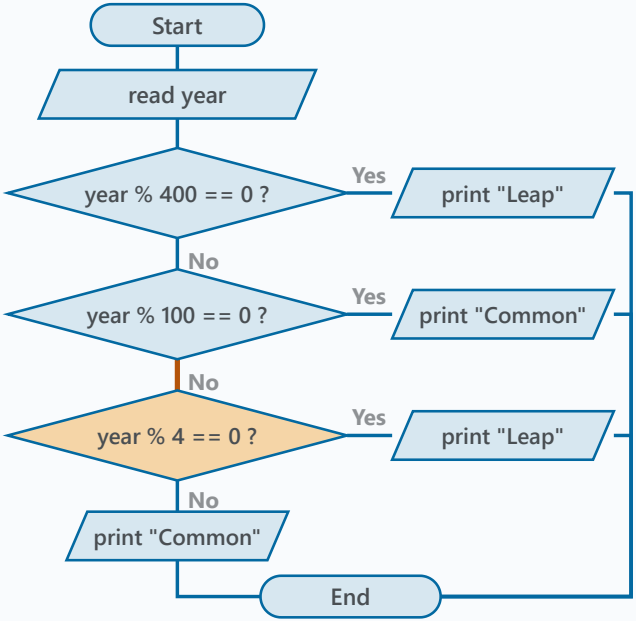
VARIABLES

`year = 2024`

OUTPUT

2024 % 100 = 24 → **false** → No.

Leap Year Step by Step (2024, 1900, 2000)




```
1 Start
2   read year
3   year % 400 == 0 ?   Yes → print "Leap"
4   year % 100 == 0 ?   Yes → print "Common"
▶   year % 4 == 0 ?     Yes → print "Leap"
6   No → print "Common"
7 End
```

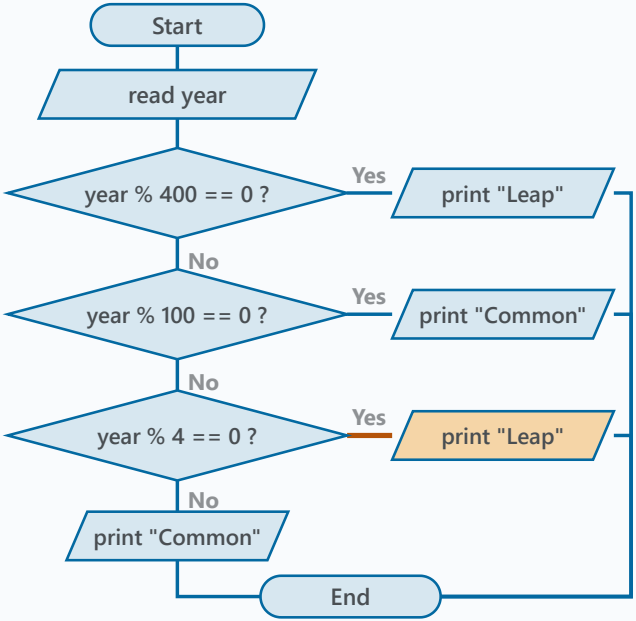
VARIABLES

`year = 2024`

OUTPUT

2024 % 4 = 0 → **true** → Yes.

Leap Year Step by Step (2024, 1900, 2000)




```
1 Start
2   read year
3   year % 400 == 0 ?   Yes → print "Leap"
4   year % 100 == 0 ?   Yes → print "Common"
▶   year % 4 == 0 ?     Yes → print "Leap"
6   No → print "Common"
7 End
```

VARIABLES

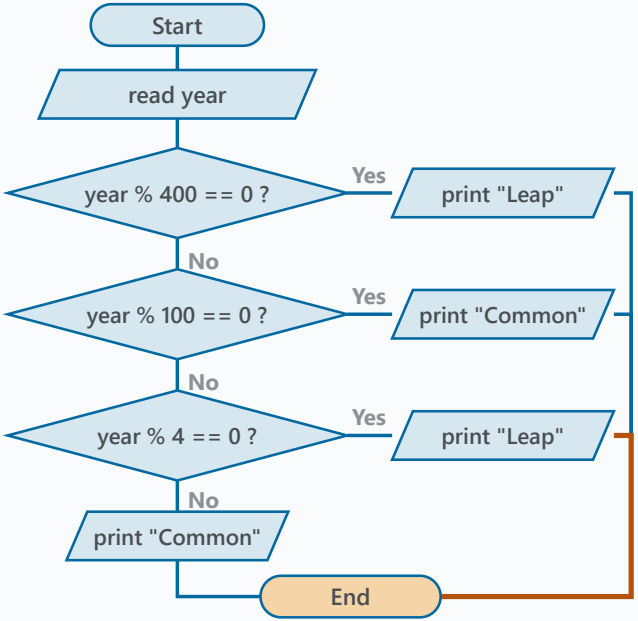
`year = 2024`

OUTPUT

```
Leap
```

"Leap" is printed. 2024 is a leap year (February has 29 days).

Leap Year Step by Step (2024, 1900, 2000)




```
1 Start
2   read year
3   year % 400 == 0 ?  Yes → print "Leap"
4   year % 100 == 0 ?  Yes → print "Common"
5   year % 4 == 0 ?    Yes → print "Leap"
6   No → print "Common"
▶ End
```

VARIABLES

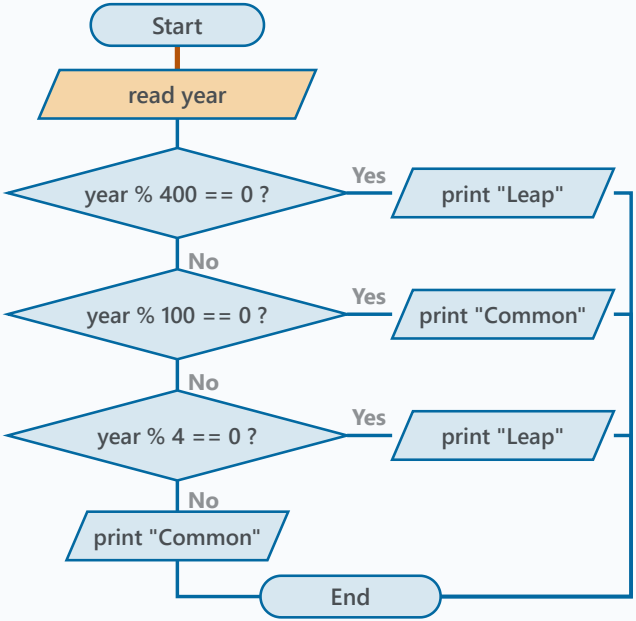
`year = 2024`

OUTPUT

```
Leap
```

End. **Run 2: 1900.**

Leap Year Step by Step (2024, 1900, 2000)




```
1 Start
▶   read year
3   year % 400 == 0 ?  Yes → print "Leap"
4   year % 100 == 0 ?  Yes → print "Common"
5   year % 4 == 0 ?    Yes → print "Leap"
6   No → print "Common"
7 End
```

VARIABLES

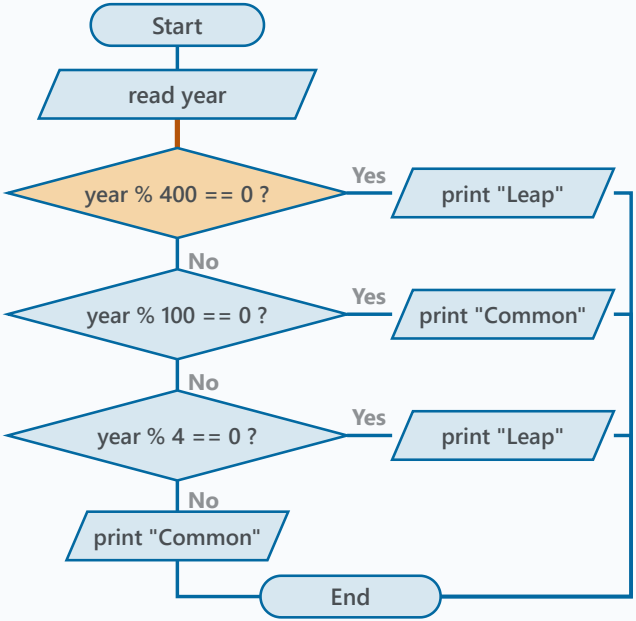
`year = 1900`

OUTPUT

```
Leap
```

Read year = **1900**.

Leap Year Step by Step (2024, 1900, 2000)




```
1 Start
2   read year
▶   year % 400 == 0 ?  Yes → print "Leap"
4   year % 100 == 0 ?  Yes → print "Common"
5   year % 4 == 0 ?    Yes → print "Leap"
6   No → print "Common"
7 End
```

VARIABLES

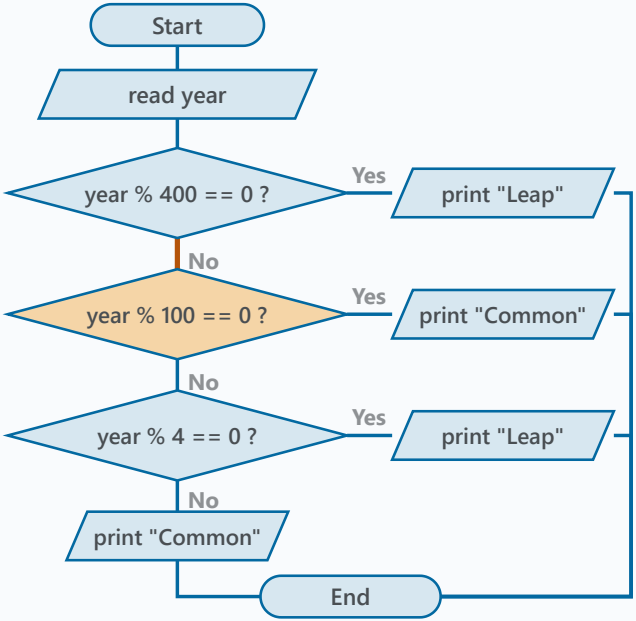
`year = 1900`

OUTPUT

```
Leap
```

1900 % 400 = 300 → **false** → No.

Leap Year Step by Step (2024, 1900, 2000)




```
1 Start
2   read year
3   year % 400 == 0 ?   Yes → print "Leap"
▶   year % 100 == 0 ?   Yes → print "Common"
5   year % 4 == 0 ?     Yes → print "Leap"
6   No → print "Common"
7 End
```

VARIABLES

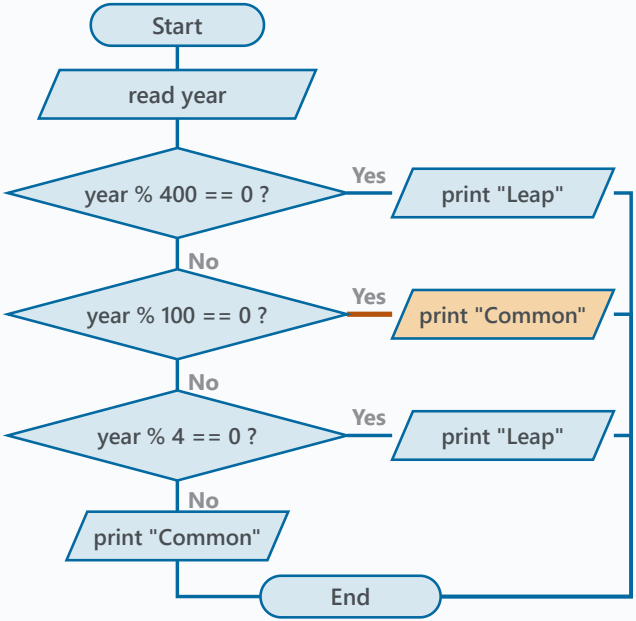
`year = 1900`

OUTPUT

```
Leap
```

1900 % 100 = 0 → **true** → Yes. The third decision is never asked.

Leap Year Step by Step (2024, 1900, 2000)



"Common" is printed. 1900 is divisible by 4 but is **not** a leap year; a chart that checked only % 4 would get this wrong.

```
1 Start
2   read year
3   year % 400 == 0 ?   Yes → print "Leap"
▶   year % 100 == 0 ?   Yes → print "Common"
5   year % 4 == 0 ?     Yes → print "Leap"
6   No → print "Common"
7 End
```

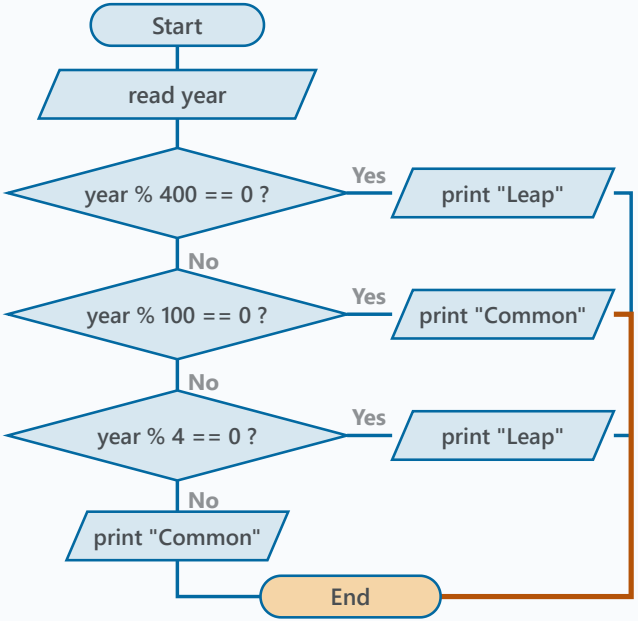
VARIABLES

`year = 1900`

OUTPUT

```
Leap
Common
```

Leap Year Step by Step (2024, 1900, 2000)




```
1 Start
2   read year
3   year % 400 == 0 ?  Yes → print "Leap"
4   year % 100 == 0 ?  Yes → print "Common"
5   year % 4 == 0 ?    Yes → print "Leap"
6   No → print "Common"
▶ End
```

VARIABLES

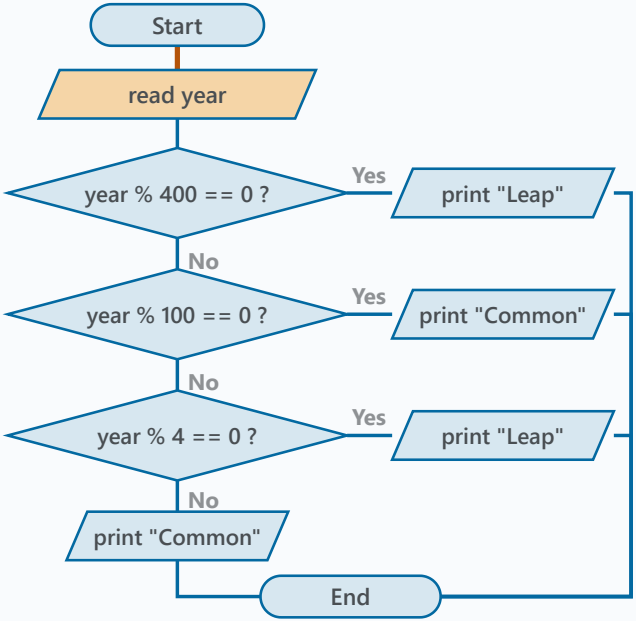
`year = 1900`

OUTPUT

```
Leap
Common
```

End. **Run 3: 2000.**

Leap Year Step by Step (2024, 1900, 2000)




```
1 Start
▶   read year
3   year % 400 == 0 ?  Yes → print "Leap"
4   year % 100 == 0 ?  Yes → print "Common"
5   year % 4 == 0 ?    Yes → print "Leap"
6   No → print "Common"
7 End
```

VARIABLES

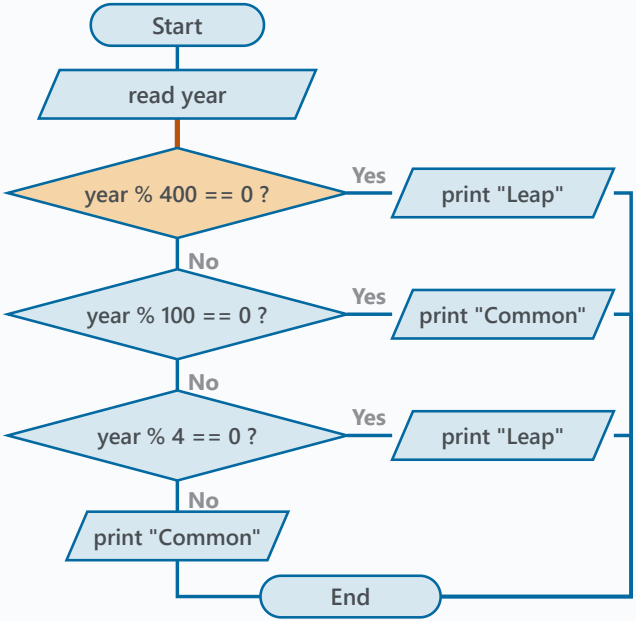
`year = 2000`

OUTPUT

```
Leap
Common
```

Read year = **2000**.

Leap Year Step by Step (2024, 1900, 2000)



2000 % 400 = 0 → **true** → Yes. The other two decisions are skipped.

```
1 Start
2   read year
▶   year % 400 == 0 ?  Yes → print "Leap"
4   year % 100 == 0 ?  Yes → print "Common"
5   year % 4 == 0 ?    Yes → print "Leap"
6   No → print "Common"
7 End
```

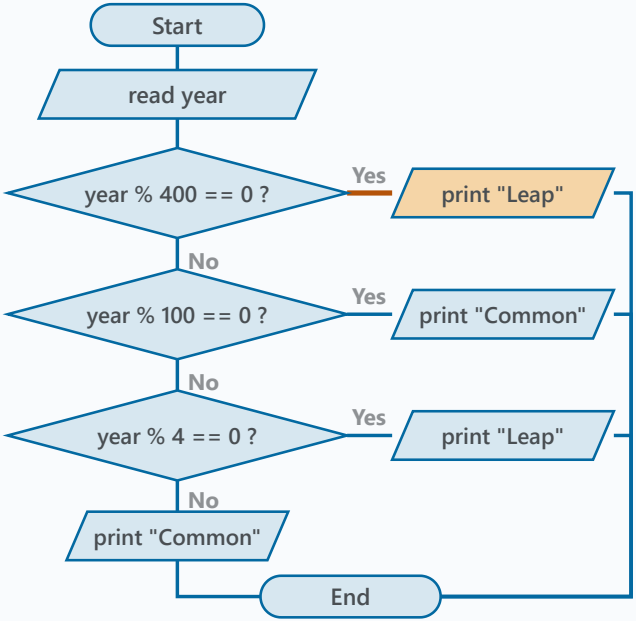
VARIABLES

`year = 2000`

OUTPUT

```
Leap
Common
```

Leap Year Step by Step (2024, 1900, 2000)




```
1 Start
2   read year
▶   year % 400 == 0 ?  Yes → print "Leap"
4   year % 100 == 0 ?  Yes → print "Common"
5   year % 4 == 0 ?    Yes → print "Leap"
6   No → print "Common"
7 End
```

VARIABLES

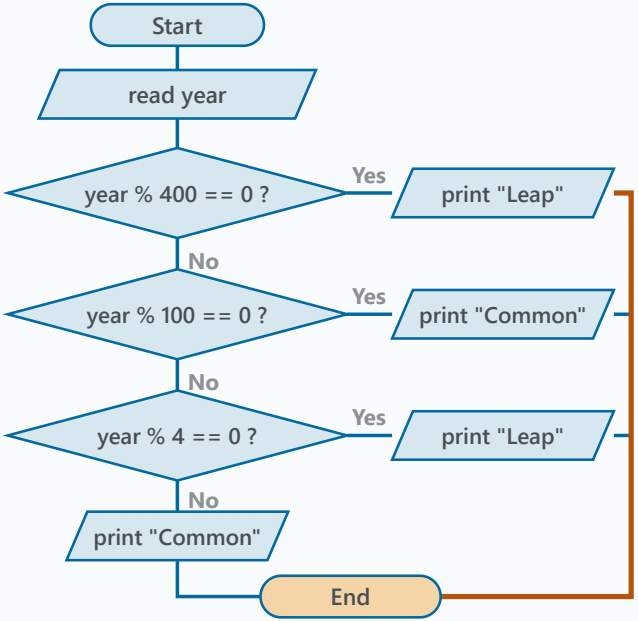
`year = 2000`

OUTPUT

```
Leap
Common
Leap
```

"Leap" is printed. 2000 is divisible by 100, but also by 400, so it is a leap year.

Leap Year Step by Step (2024, 1900, 2000)



End. Three inputs exercised three of the four exits; a year like 2023 tests the last No branch. The **order** of the decisions handles the rule's exceptions correctly.

```
1 Start
2   read year
3   year % 400 == 0 ?  Yes → print "Leap"
4   year % 100 == 0 ?  Yes → print "Common"
5   year % 4 == 0 ?    Yes → print "Leap"
6   No → print "Common"
▶ End
```

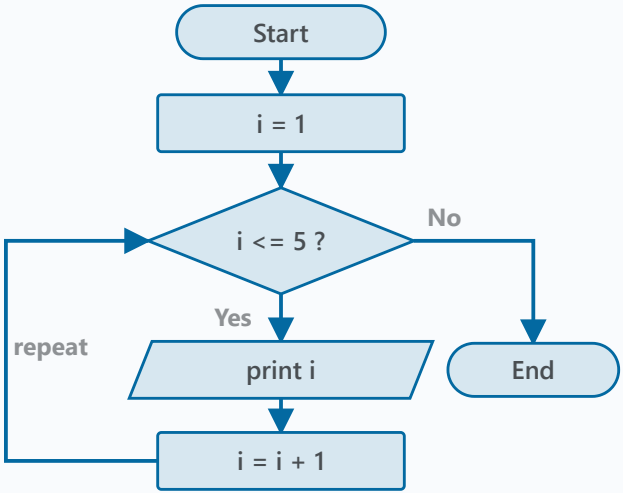
VARIABLES

`year = 2000`

OUTPUT

```
Leap
Common
Leap
```

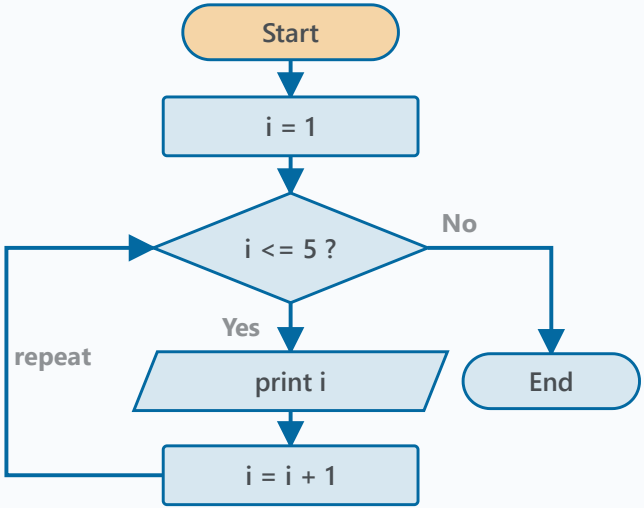
Repetition




```
Start
  i = 1
  i <= 5 ?
    Yes → print i
            i = i + 1
            back to the decision
    No → End
```

- The arrow going back (repeat) forms the loop; the condition is checked before the body (a pre-test loop), and the loop ends when it is false.
- The repeat path must contain a step that changes the condition (**i = i + 1**).

Repetition Step by Step



Start.

```
▶ Start
2   i = 1
3   i <= 5 ?
4     Yes → print i
5               i = i + 1
6               back to the decision
7     No → End
```

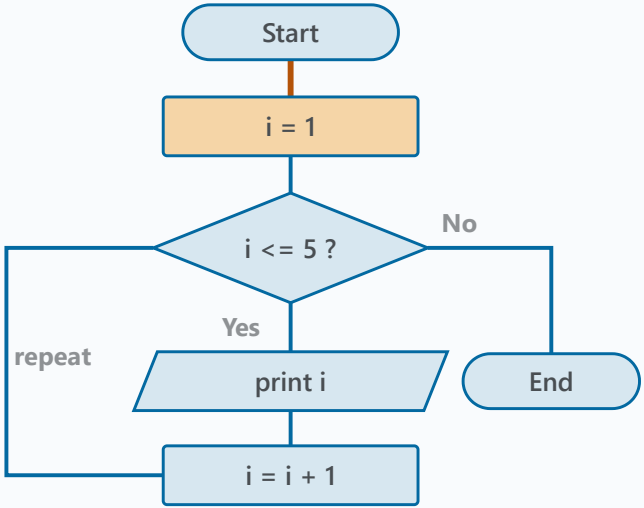
VARIABLES

—

OUTPUT

Step 1 / 19

Repetition Step by Step



The counter gets its first value: **i = 1**.

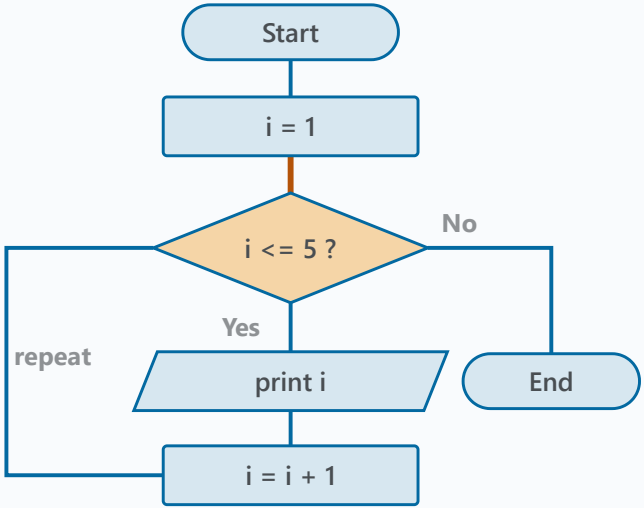
```
1 Start
▶   i = 1
3   i <= 5 ?
4     Yes → print i
5               i = i + 1
6               back to the decision
7     No → End
```

VARIABLES

`i = 1`

OUTPUT

Repetition Step by Step



Decision: **1 <= 5** → **true** → Yes, enter the loop body.

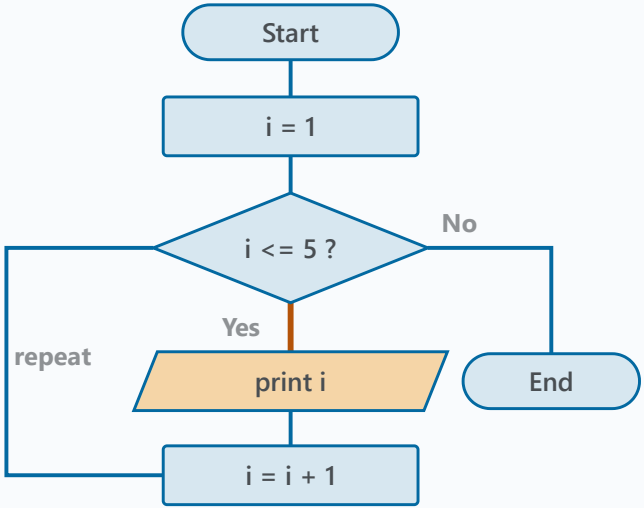
```
1 Start
2   i = 1
▶   i <= 5 ?
4     Yes → print i
5             i = i + 1
6             back to the decision
7     No → End
```

VARIABLES

`i = 1`

OUTPUT

Repetition Step by Step



1 is printed.

```
1 Start
2   i = 1
3   i <= 5 ?
▶     Yes → print i
5             i = i + 1
6             back to the decision
7     No → End
```

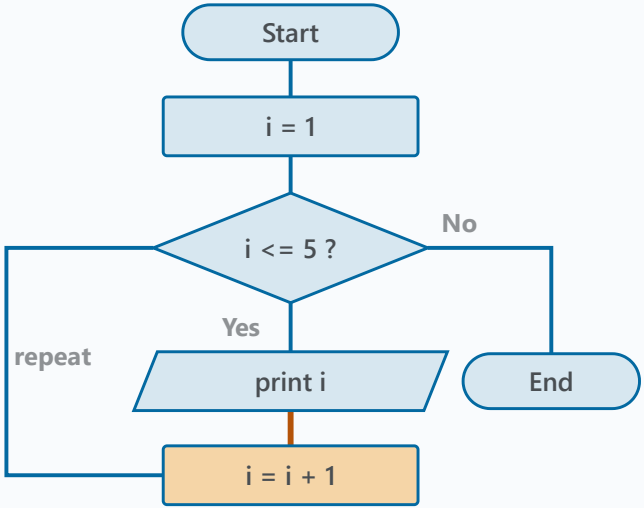
VARIABLES

`i = 1`

OUTPUT

```
1
```

Repetition Step by Step



i = 1 + 1 = 2. The back arrow returns to the decision.

```
1 Start
2   i = 1
3   i <= 5 ?
4     Yes → print i
5           i = i + 1
6           back to the decision
7     No → End
```

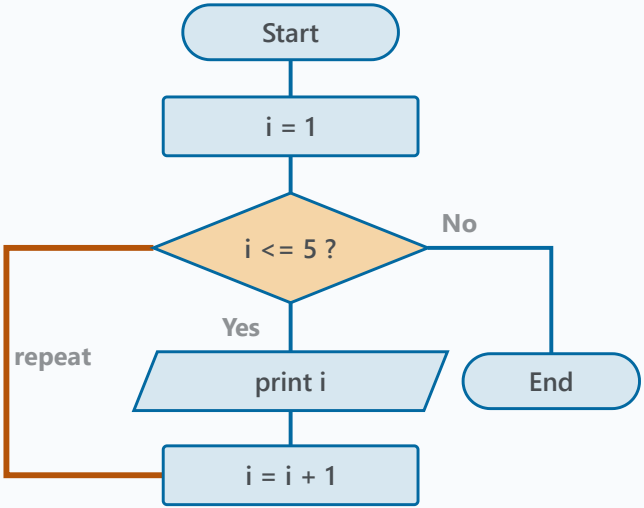
VARIABLES

`i = 2`

OUTPUT

```
1
```

Repetition Step by Step



Decision: **2 <= 5** → **true** → Yes, enter the loop body.

```
1 Start
2   i = 1
▶   i <= 5 ?
4     Yes → print i
5             i = i + 1
6             back to the decision
7     No → End
```

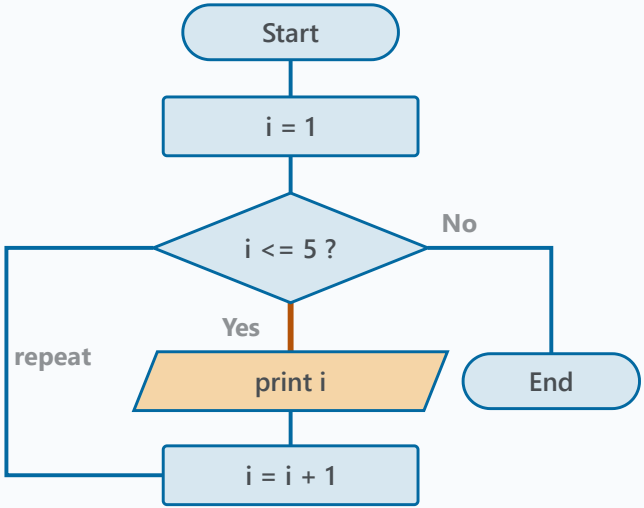
VARIABLES

`i = 2`

OUTPUT

```
1
```

Repetition Step by Step



2 is printed.

```
1 Start
2   i = 1
3   i <= 5 ?
      Yes → print i
5             i = i + 1
6             back to the decision
7     No → End
```

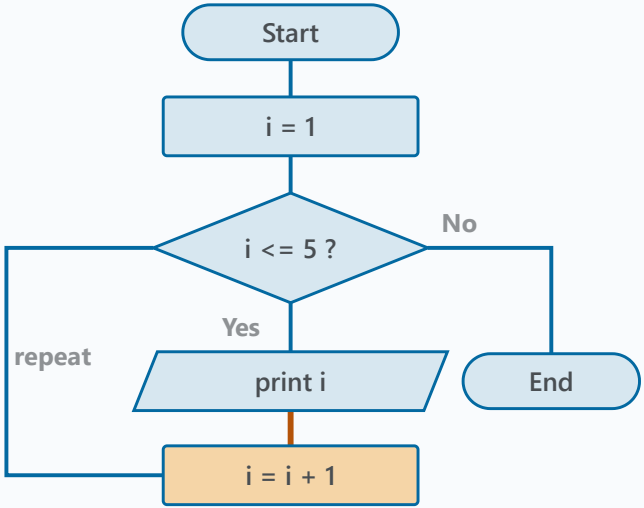
VARIABLES

`i = 2`

OUTPUT

```
1 2
```

Repetition Step by Step



i = 2 + 1 = 3. The back arrow returns to the decision.

```
1 Start
2   i = 1
3   i <= 5 ?
4     Yes → print i
5           i = i + 1
6           back to the decision
7     No → End
```

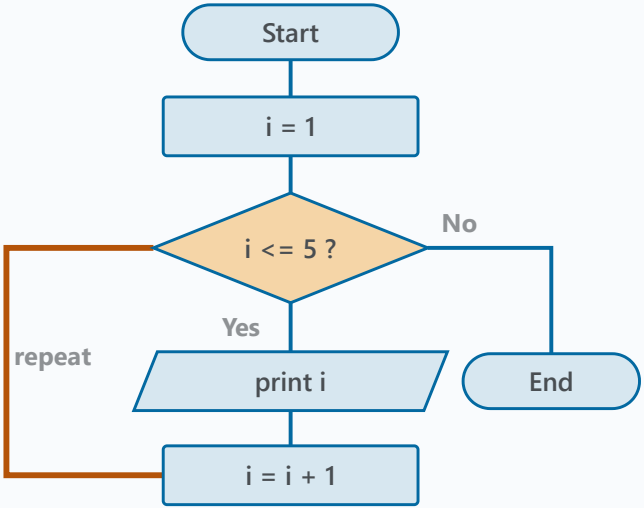
VARIABLES

`i = 3`

OUTPUT

```
1 2
```

Repetition Step by Step



Decision: **3 <= 5** → **true** → Yes, enter the loop body.

```
1 Start
2   i = 1
▶   i <= 5 ?
4     Yes → print i
5             i = i + 1
6             back to the decision
7     No → End
```

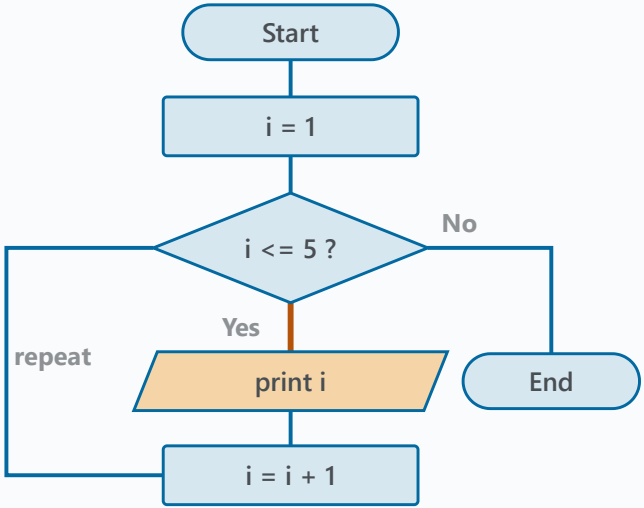
VARIABLES

`i = 3`

OUTPUT

```
1 2
```

Repetition Step by Step



3 is printed.

```
1 Start
2   i = 1
3   i <= 5 ?
      Yes → print i
5               i = i + 1
6               back to the decision
7     No → End
```

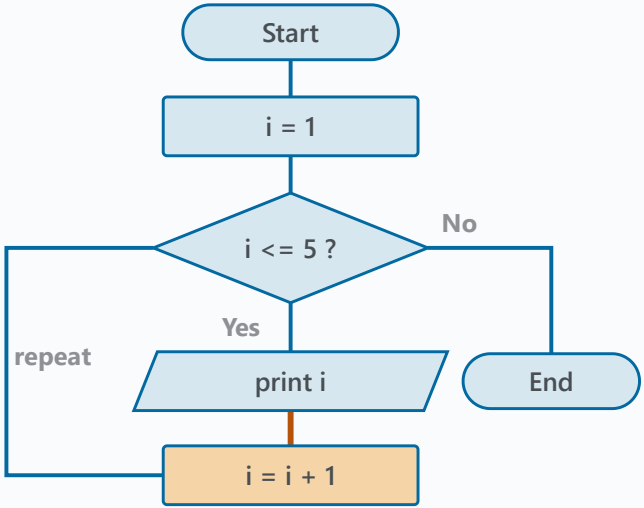
VARIABLES

`i = 3`

OUTPUT

```
1 2 3
```

Repetition Step by Step



i = 3 + 1 = 4. The back arrow returns to the decision.

```
1 Start
2   i = 1
3   i <= 5 ?
4     Yes → print i
5           i = i + 1
6           back to the decision
7     No → End
```

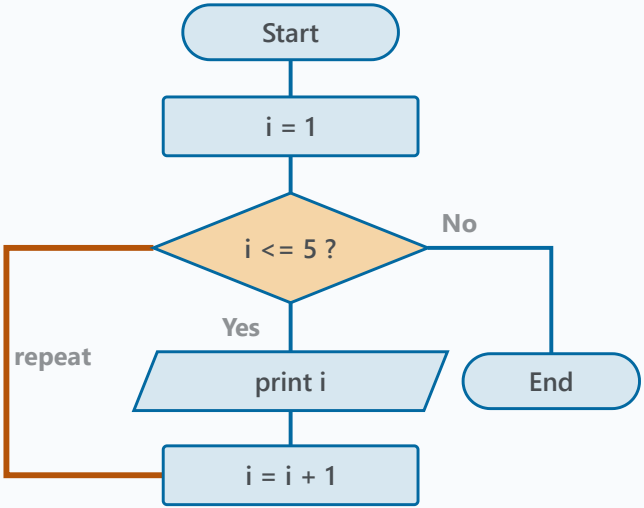
VARIABLES

`i = 4`

OUTPUT

```
1 2 3
```

Repetition Step by Step



Decision: **4 <= 5** → **true** → Yes, enter the loop body.

```
1 Start
2   i = 1
▶   i <= 5 ?
4     Yes → print i
5             i = i + 1
6             back to the decision
7     No → End
```

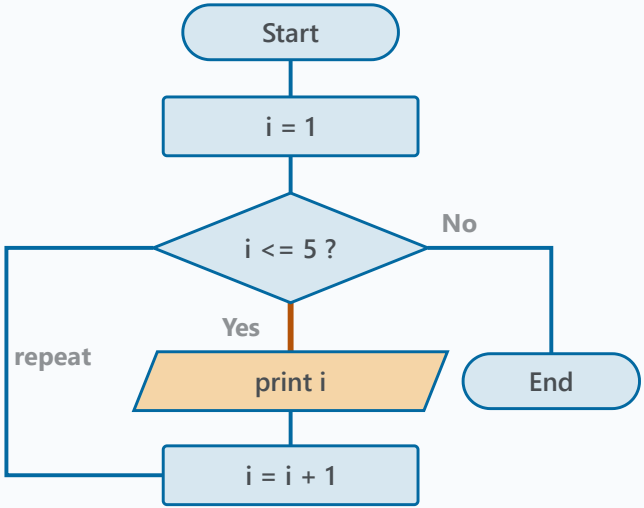
VARIABLES

`i = 4`

OUTPUT

```
1 2 3
```

Repetition Step by Step



4 is printed.

```
1 Start
2   i = 1
3   i <= 5 ?
▶     Yes → print i
5             i = i + 1
6             back to the decision
7     No → End
```

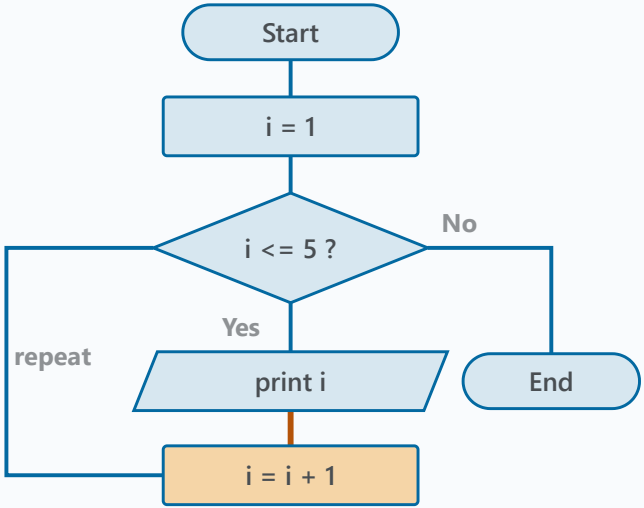
VARIABLES

`i = 4`

OUTPUT

```
1 2 3 4
```

Repetition Step by Step



i = 4 + 1 = 5. The back arrow returns to the decision.

```
1 Start
2   i = 1
3   i <= 5 ?
4     Yes → print i
▶               i = i + 1
6               back to the decision
7     No → End
```

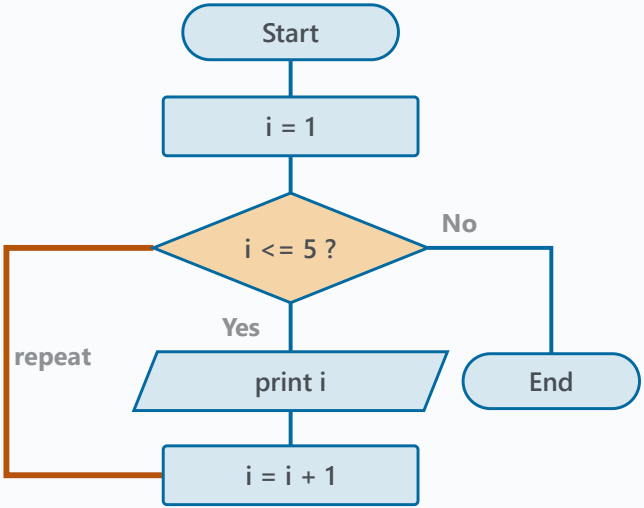
VARIABLES

`i = 5`

OUTPUT

```
1 2 3 4
```

Repetition Step by Step



Decision: **5 <= 5** → **true** → Yes, enter the loop body.

```
1 Start
2   i = 1
▶   i <= 5 ?
4     Yes → print i
5             i = i + 1
6             back to the decision
7     No → End
```

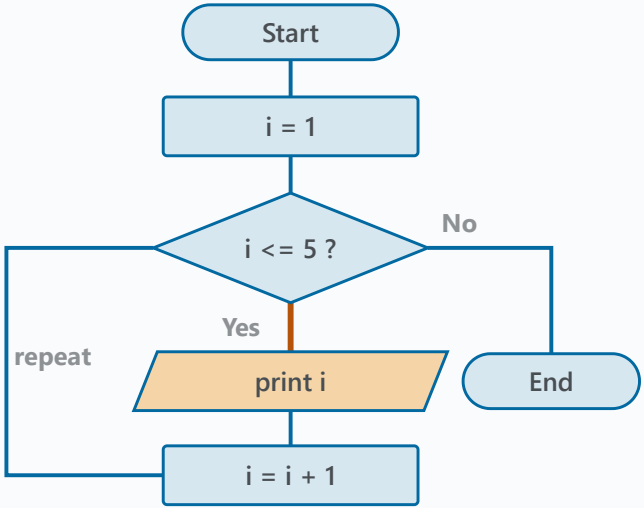
VARIABLES

`i = 5`

OUTPUT

```
1 2 3 4
```

Repetition Step by Step



5 is printed.

```
1 Start
2   i = 1
3   i <= 5 ?
▶     Yes → print i
5             i = i + 1
6             back to the decision
7     No → End
```

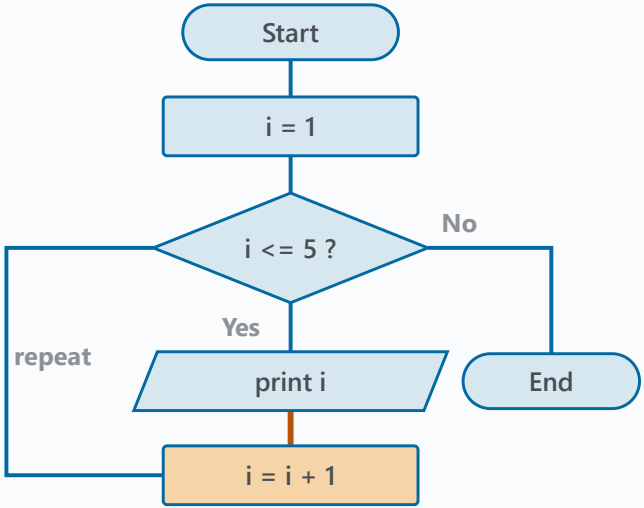
VARIABLES

`i = 5`

OUTPUT

```
1 2 3 4 5
```

Repetition Step by Step



i = 5 + 1 = 6. The back arrow returns to the decision.

```
1 Start
2   i = 1
3   i <= 5 ?
4     Yes → print i
5           i = i + 1
6           back to the decision
7     No → End
```

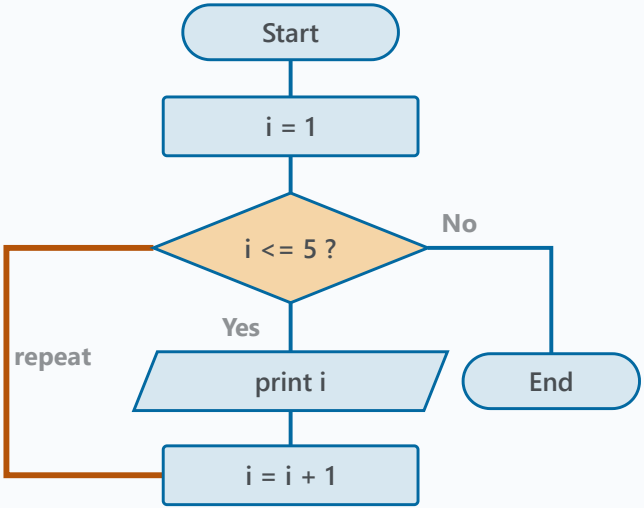
VARIABLES

`i = 6`

OUTPUT

```
1 2 3 4 5
```

Repetition Step by Step



Decision: **6 <= 5** → **false** → No, leave the loop.

```
1 Start
2   i = 1
▶   i <= 5 ?
4     Yes → print i
5             i = i + 1
6             back to the decision
7     No → End
```

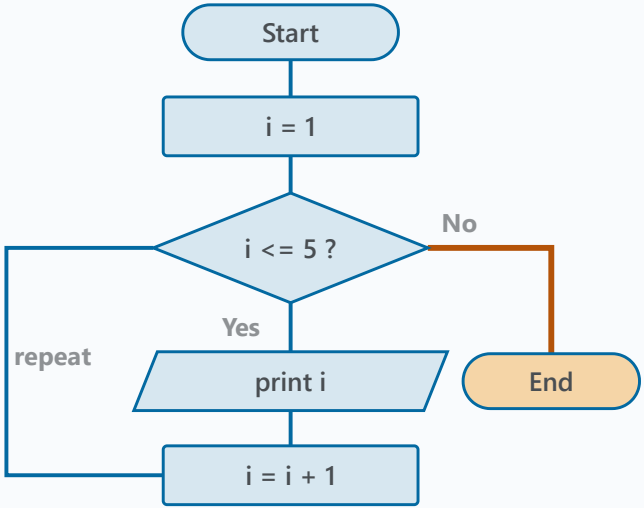
VARIABLES

`i = 6`

OUTPUT

```
1 2 3 4 5
```

Repetition Step by Step



End. The body ran **5 times**; the condition was tested **6 times** (the last test, with i = 6, was false).


```
1 Start
2   i = 1
3   i <= 5 ?
4     Yes → print i
5               i = i + 1
6               back to the decision
      No → End
```

VARIABLES

`i = 6`

OUTPUT

```
1 2 3 4 5
```

Counters and Accumulators

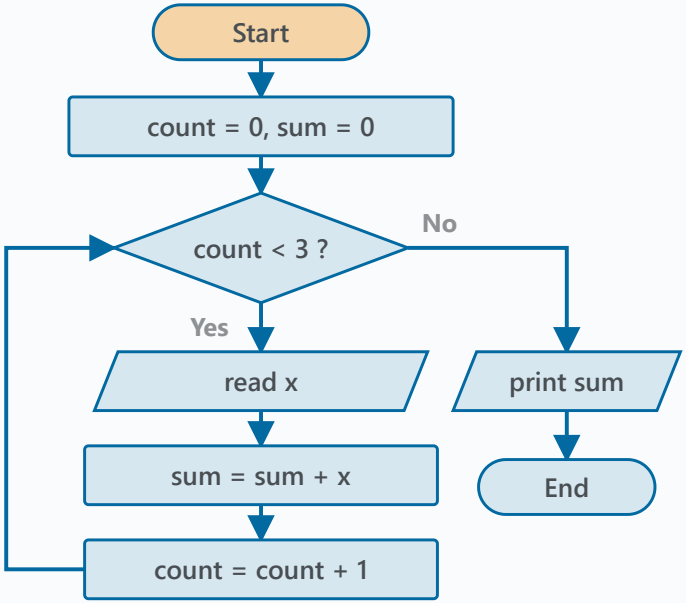
- A **counter** tracks how many repetitions have run: it grows by 1 each pass.
- An **accumulator** holds a partial result: each pass adds a new value.
- Pick the starting value to suit the operation: **0** for a sum, **1** for a product.



Pass-by-pass values for inputs 4, 7, 2:

Step	x	count	sum
start	—	0	0
pass 1	4	1	4
pass 2	7	2	11
pass 3	2	3	13

Counter and Accumulator Step by Step (4, 7, 2)



Start. Three numbers will be read and added; the inputs are 4, 7, 2.

```
▶ Start
2   count = 0, sum = 0
3   count < 3 ?
4     Yes → read x
5               sum = sum + x
6               count = count + 1
7               back to the decision
8     No → print sum
9 End
```

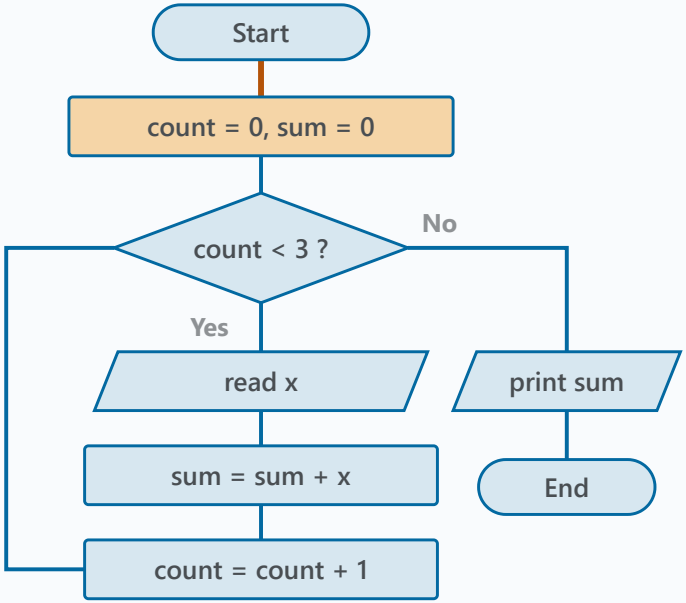
VARIABLES

—

OUTPUT

Step 1 / 17

Counter and Accumulator Step by Step (4, 7, 2)



Counter and accumulator start at **0**.

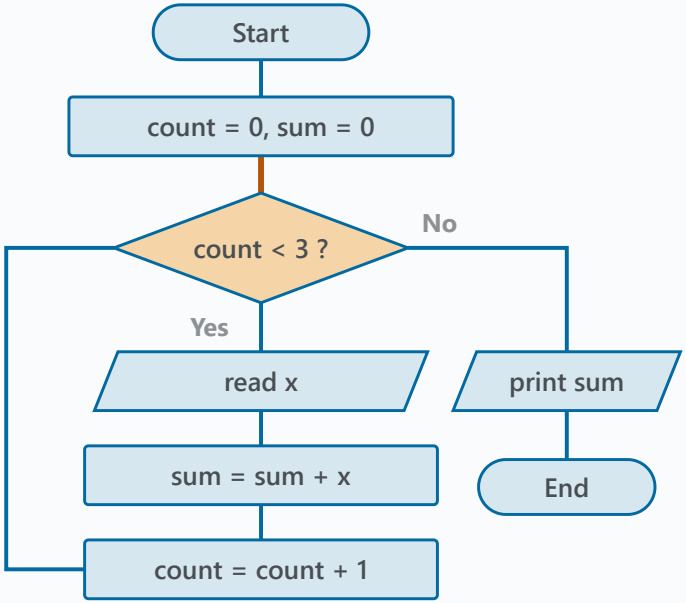
```
1 Start
▶   count = 0, sum = 0
3   count < 3 ?
4     Yes → read x
5               sum = sum + x
6               count = count + 1
7               back to the decision
8     No → print sum
9 End
```

VARIABLES

`count = 0` `sum = 0`

OUTPUT

Counter and Accumulator Step by Step (4, 7, 2)



Decision: **0 < 3** → **true** → Yes.

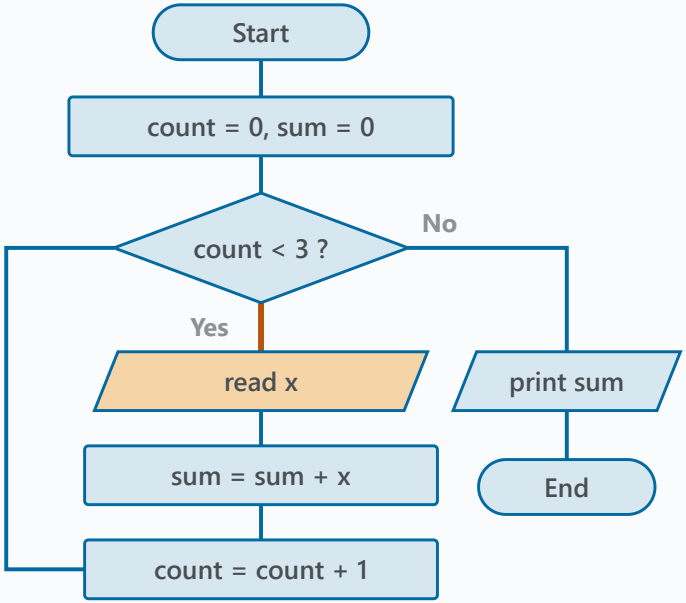
```
1 Start
2   count = 0, sum = 0
▶   count < 3 ?
4     Yes → read x
5               sum = sum + x
6               count = count + 1
7               back to the decision
8     No → print sum
9 End
```

VARIABLES

`count = 0` `sum = 0`

OUTPUT

Counter and Accumulator Step by Step (4, 7, 2)



Read **x = 4**.

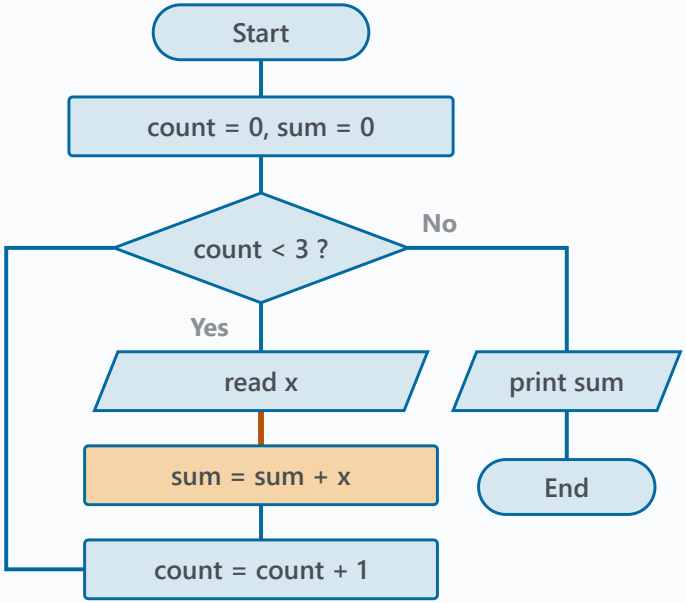
```
1  Start
2    count = 0, sum = 0
3    count < 3 ?
▶      Yes → read x
5              sum = sum + x
6              count = count + 1
7              back to the decision
8      No → print sum
9  End
```

VARIABLES

`count = 0` `sum = 0` `x = 4`

OUTPUT

Counter and Accumulator Step by Step (4, 7, 2)



Accumulator: **sum = 0 + 4 = 4**.

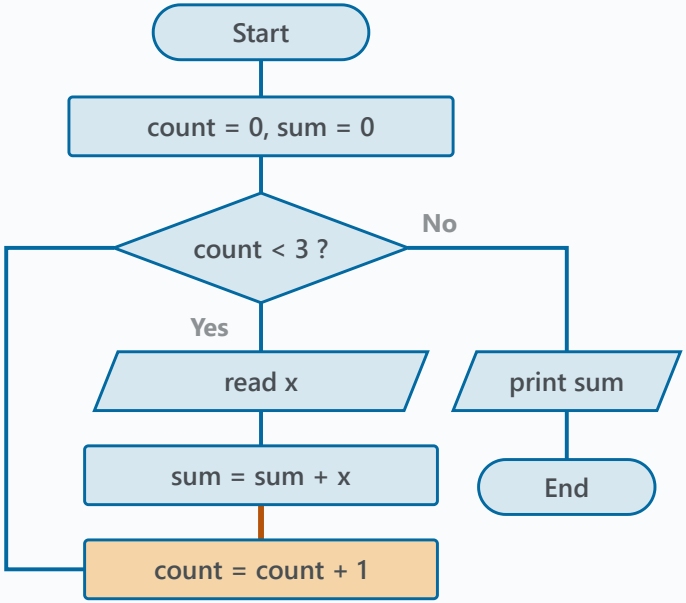
```
1  Start
2    count = 0, sum = 0
3    count < 3 ?
4      Yes → read x
▶              sum = sum + x
6              count = count + 1
7              back to the decision
8      No → print sum
9  End
```

VARIABLES

`count = 0` `sum = 4` `x = 4`

OUTPUT

Counter and Accumulator Step by Step (4, 7, 2)



Counter: **count = 0 + 1 = 1**. Go back.

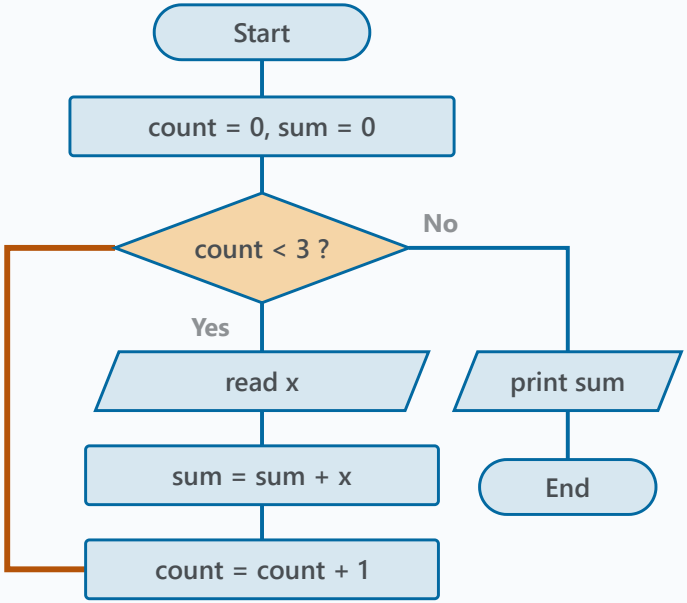
```
1 Start
2   count = 0, sum = 0
3   count < 3 ?
4     Yes → read x
5             sum = sum + x
▶             count = count + 1
7             back to the decision
8     No → print sum
9 End
```

VARIABLES

`count = 1` `sum = 4` `x = 4`

OUTPUT

Counter and Accumulator Step by Step (4, 7, 2)



Decision: **1 < 3** → **true** → Yes.

```
1 Start
2   count = 0, sum = 0
▶   count < 3 ?
4     Yes → read x
5             sum = sum + x
6             count = count + 1
7             back to the decision
8     No → print sum
9 End
```

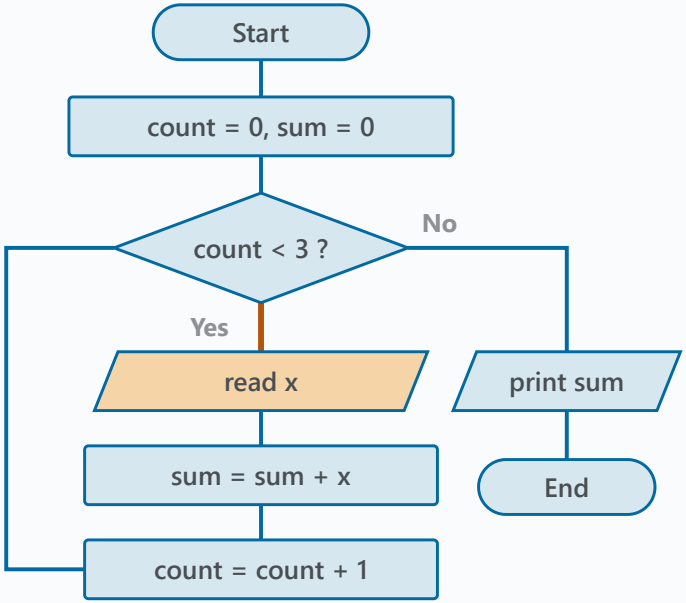
VARIABLES

`count = 1` `sum = 4` `x = 4`

OUTPUT

Step 7 / 17

Counter and Accumulator Step by Step (4, 7, 2)



Read **x = 7**.

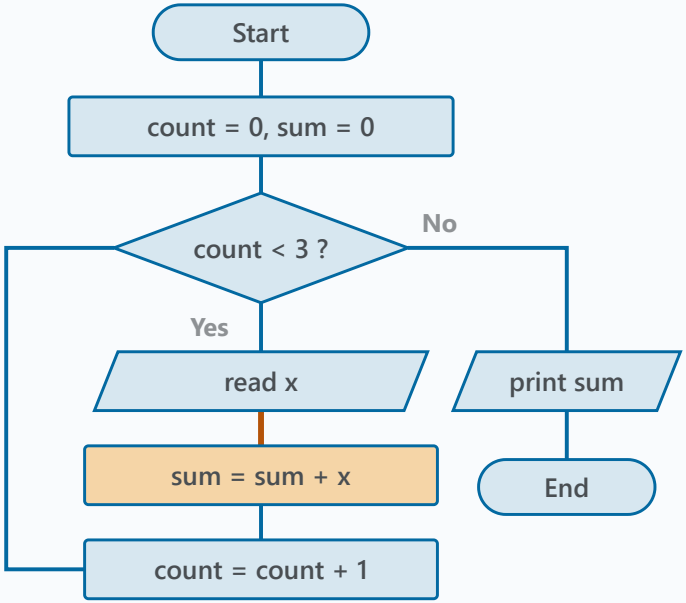
```
1  Start
2    count = 0, sum = 0
3    count < 3 ?
▶      Yes → read x
5              sum = sum + x
6              count = count + 1
7              back to the decision
8      No → print sum
9  End
```

VARIABLES

`count = 1` `sum = 4` `x = 7`

OUTPUT

Counter and Accumulator Step by Step (4, 7, 2)



Accumulator: **sum = 4 + 7 = 11**.

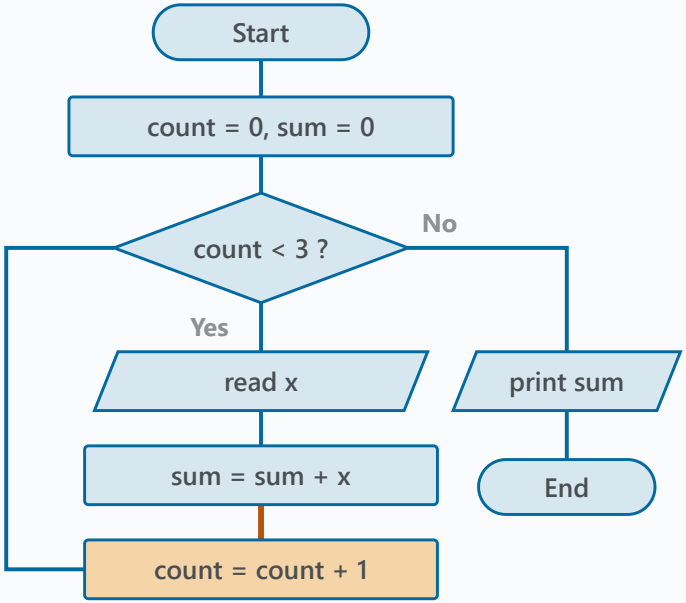
```
1 Start
2   count = 0, sum = 0
3   count < 3 ?
4     Yes → read x
▶               sum = sum + x
6               count = count + 1
7               back to the decision
8     No → print sum
9 End
```

VARIABLES

`count = 1` `sum = 11` `x = 7`

OUTPUT

Counter and Accumulator Step by Step (4, 7, 2)



Counter: **count = 1 + 1 = 2**. Go back.

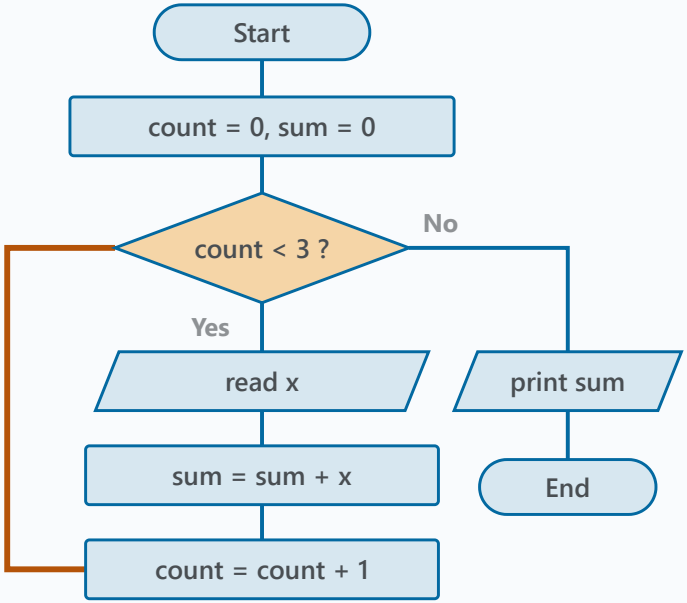
```
1 Start
2   count = 0, sum = 0
3   count < 3 ?
4     Yes → read x
5             sum = sum + x
▶             count = count + 1
7             back to the decision
8     No → print sum
9 End
```

VARIABLES

`count = 2` `sum = 11` `x = 7`

OUTPUT

Counter and Accumulator Step by Step (4, 7, 2)



Decision: **2 < 3** → **true** → Yes.

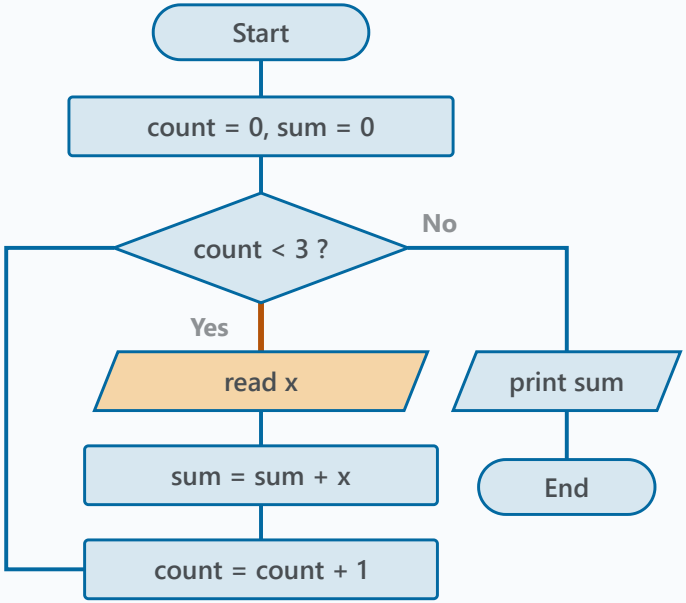
```
1 Start
2   count = 0, sum = 0
▶   count < 3 ?
4     Yes → read x
5             sum = sum + x
6             count = count + 1
7             back to the decision
8     No → print sum
9 End
```

VARIABLES

`count = 2` `sum = 11` `x = 7`

OUTPUT

Counter and Accumulator Step by Step (4, 7, 2)



Read **x = 2**.

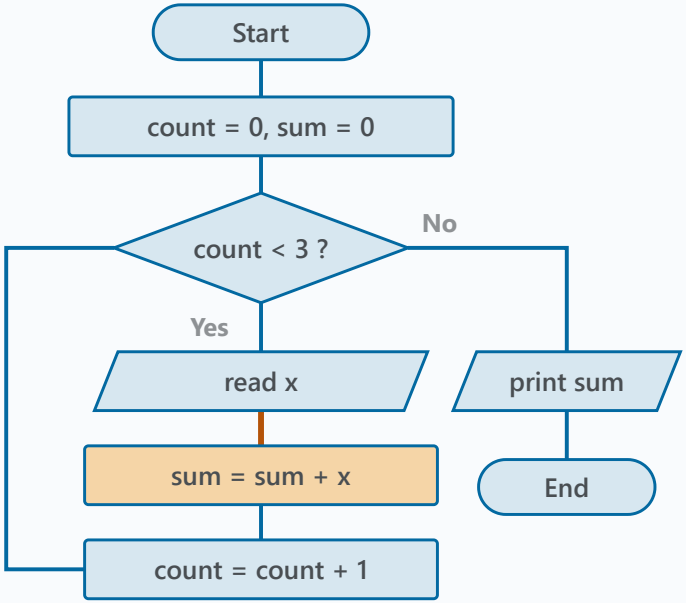
```
1 Start
2   count = 0, sum = 0
3   count < 3 ?
▶     Yes → read x
5               sum = sum + x
6               count = count + 1
7               back to the decision
8     No → print sum
9 End
```

VARIABLES

`count = 2` `sum = 11` `x = 2`

OUTPUT

Counter and Accumulator Step by Step (4, 7, 2)



Accumulator: **sum = 11 + 2 = 13**.

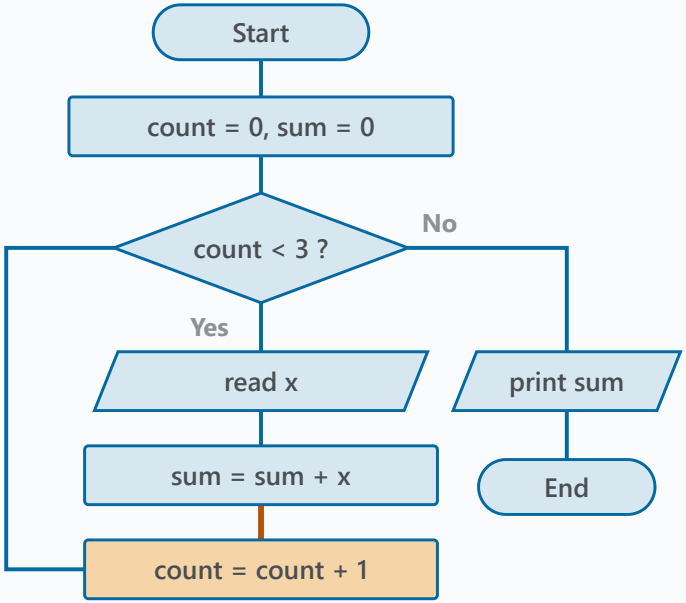
```
1  Start
2    count = 0, sum = 0
3    count < 3 ?
4      Yes → read x
▶              sum = sum + x
6              count = count + 1
7              back to the decision
8      No → print sum
9  End
```

VARIABLES

`count = 2` `sum = 13` `x = 2`

OUTPUT

Counter and Accumulator Step by Step (4, 7, 2)



Counter: **count = 2 + 1 = 3**. Go back.

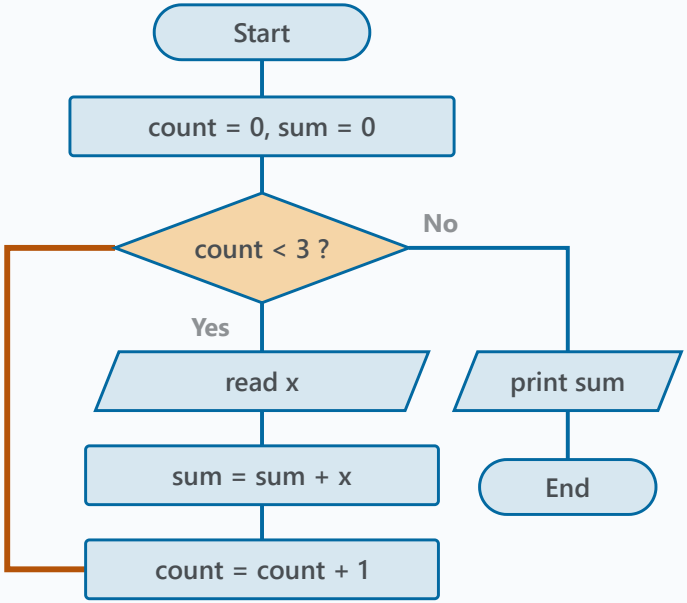
```
1 Start
2   count = 0, sum = 0
3   count < 3 ?
4     Yes → read x
5             sum = sum + x
▶             count = count + 1
7             back to the decision
8     No → print sum
9 End
```

VARIABLES

`count = 3` `sum = 13` `x = 2`

OUTPUT

Counter and Accumulator Step by Step (4, 7, 2)



Decision: **3 < 3** → **false** → No, the loop ends.

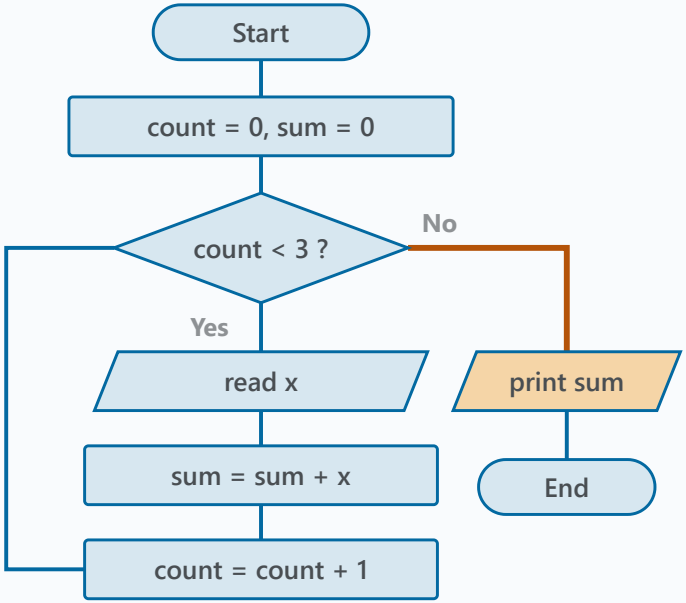
```
1 Start
2   count = 0, sum = 0
▶   count < 3 ?
4     Yes → read x
5             sum = sum + x
6             count = count + 1
7             back to the decision
8     No → print sum
9 End
```

VARIABLES

`count = 3` `sum = 13` `x = 2`

OUTPUT

Counter and Accumulator Step by Step (4, 7, 2)




```
1 Start
2   count = 0, sum = 0
3   count < 3 ?
4     Yes → read x
5               sum = sum + x
6               count = count + 1
7               back to the decision
▶     No → print sum
9 End
```

VARIABLES

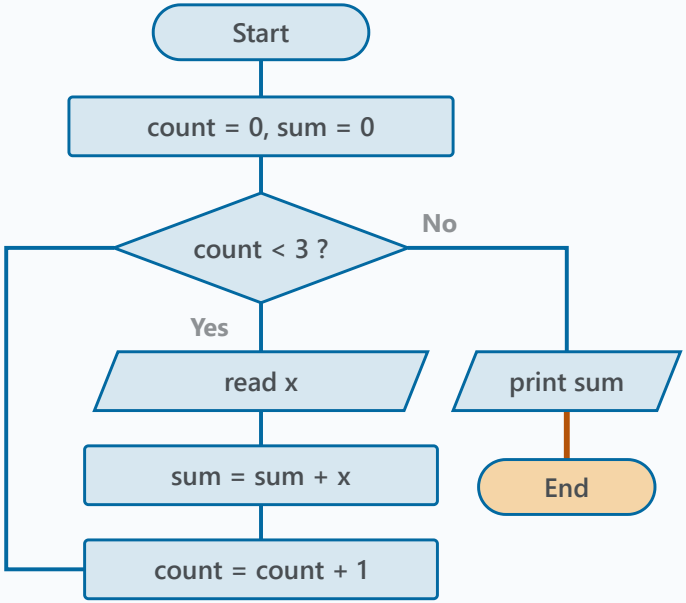
`count = 3` `sum = 13` `x = 2`

OUTPUT

```
13
```

Print the sum: **13**.

Counter and Accumulator Step by Step (4, 7, 2)



End. The counter recorded **how many times** the loop ran; the accumulator held the **result**.

```
1 Start
2   count = 0, sum = 0
3   count < 3 ?
4     Yes → read x
5               sum = sum + x
6               count = count + 1
7               back to the decision
8     No → print sum
▶ End
```

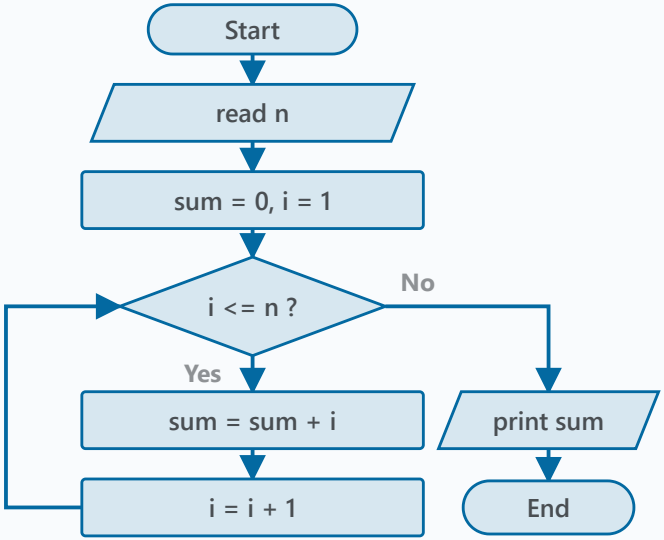
VARIABLES

`count = 3` `sum = 13` `x = 2`

OUTPUT

```
13
```

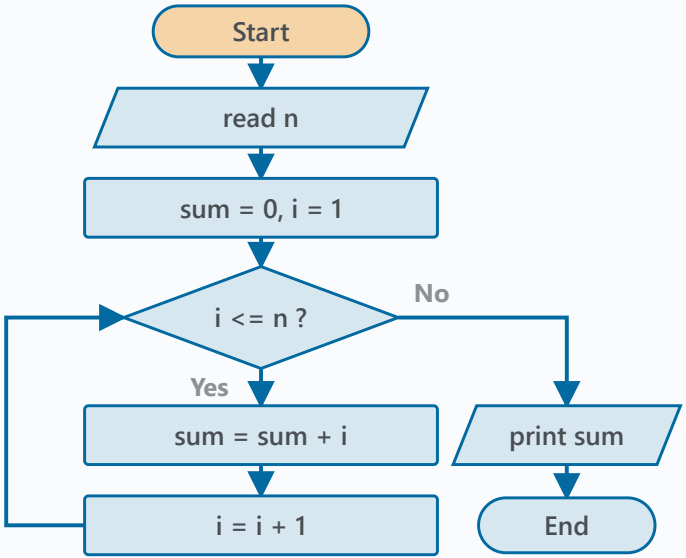
Example: Sum from 1 to n




```
Start
  read n
  sum = 0, i = 1
  i <= n ?
    Yes → sum = sum + i
            i = i + 1
            back to the decision
    No → print sum
End
```

- **i** is the counter, **sum** the accumulator.
- For n = 0 the condition is false on the first test, and 0 is printed.

Sum from 1 to n Step by Step (n = 4)



Start. Input: n = 4. Expected result 1 + 2 + 3 + 4 = 10.

Step 1 / 18

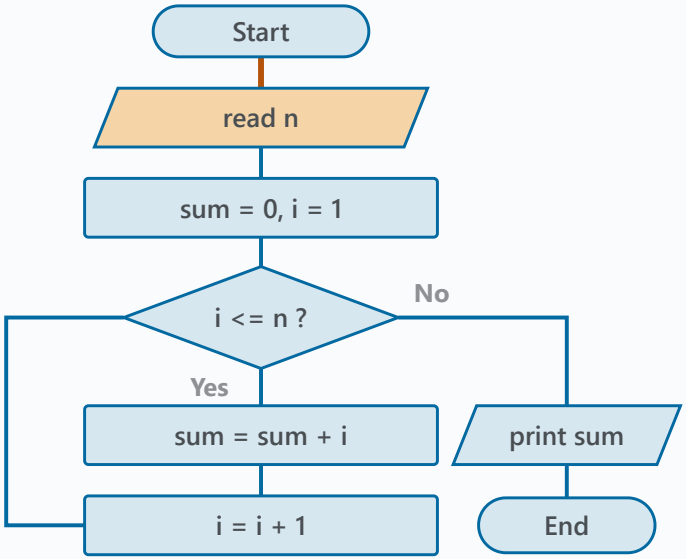
```
▶ Start
2   read n
3   sum = 0, i = 1
4   i <= n ?
5     Yes → sum = sum + i
6             i = i + 1
7             back to the decision
8     No → print sum
9 End
```

VARIABLES

—

OUTPUT

Sum from 1 to n Step by Step (n = 4)



Read **n = 4**.

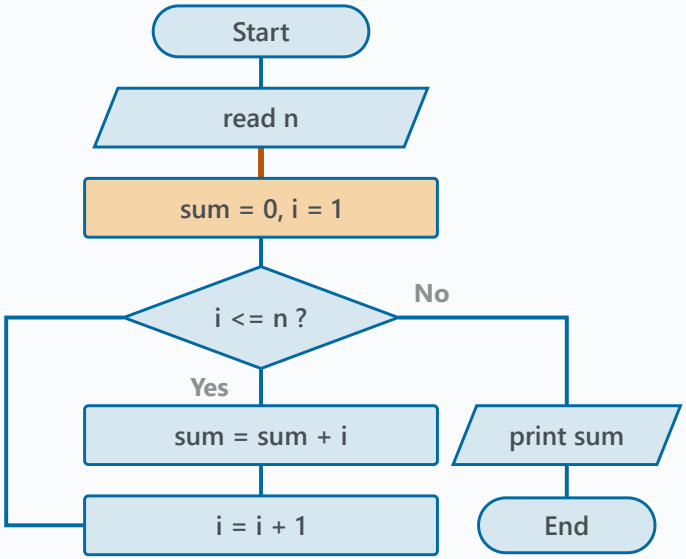
```
1  Start
▶    read n
3    sum = 0, i = 1
4    i <= n ?
5      Yes → sum = sum + i
6              i = i + 1
7              back to the decision
8      No → print sum
9  End
```

VARIABLES

n = 4

OUTPUT

Sum from 1 to n Step by Step (n = 4)



Accumulator **sum = 0**, counter **i = 1**.

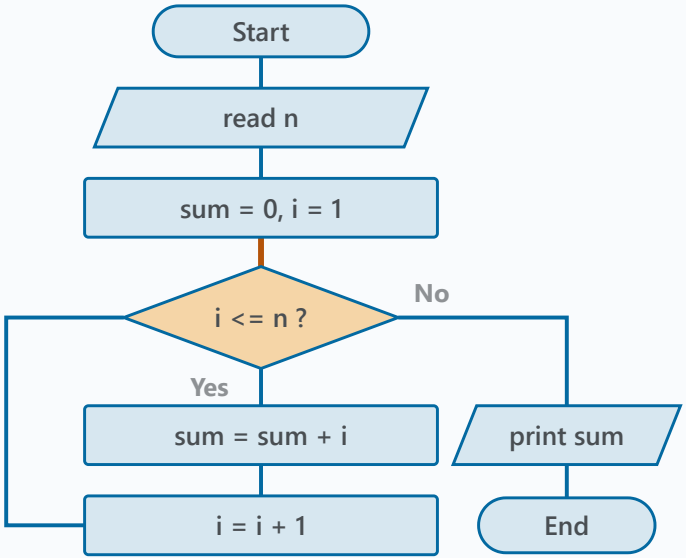
```
1 Start
2   read n
▶   sum = 0, i = 1
4   i <= n ?
5     Yes → sum = sum + i
6             i = i + 1
7             back to the decision
8     No → print sum
9 End
```

VARIABLES

`n = 4` `sum = 0` `i = 1`

OUTPUT

Sum from 1 to n Step by Step (n = 4)



Decision: **1 <= 4** → **true** → Yes.

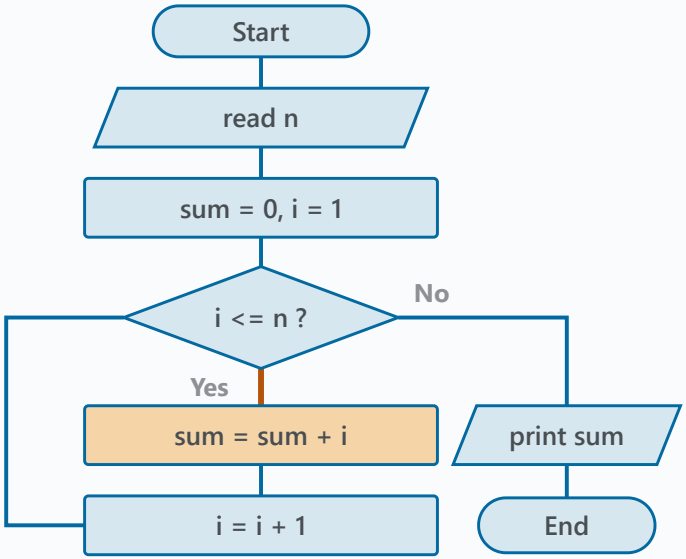
```
1 Start
2   read n
3   sum = 0, i = 1
▶   i <= n ?
5     Yes → sum = sum + i
6             i = i + 1
7             back to the decision
8     No → print sum
9 End
```

VARIABLES

`n = 4` `sum = 0` `i = 1`

OUTPUT

Sum from 1 to n Step by Step (n = 4)



sum = 0 + 1 = 1.

```
1 Start
2   read n
3   sum = 0, i = 1
4   i <= n ?
▶     Yes → sum = sum + i
6             i = i + 1
7             back to the decision
8     No → print sum
9 End
```

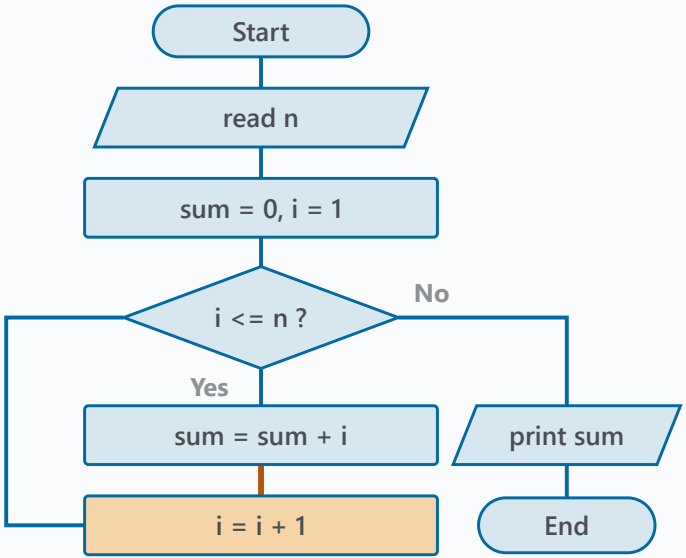
VARIABLES

`n = 4` `sum = 1` `i = 1`

OUTPUT

Step 5 / 18

Sum from 1 to n Step by Step (n = 4)



i = 2; back to the decision.

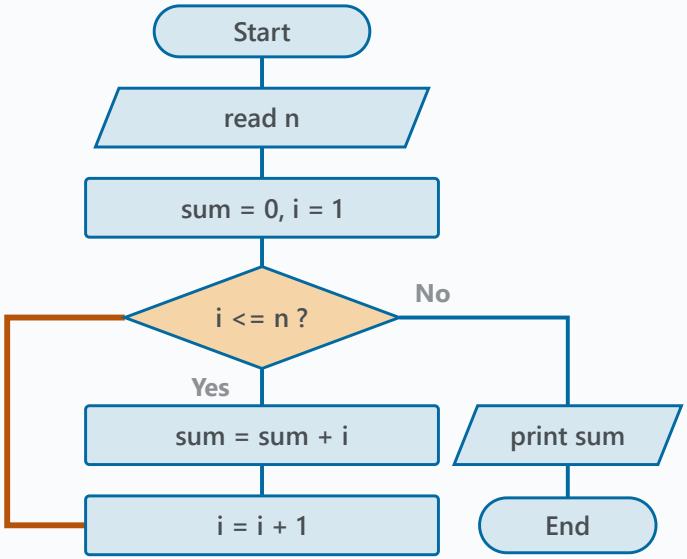
```
1 Start
2   read n
3   sum = 0, i = 1
4   i <= n ?
5     Yes → sum = sum + i
▶               i = i + 1
7               back to the decision
8     No → print sum
9 End
```

VARIABLES

`n = 4` `sum = 1` `i = 2`

OUTPUT

Sum from 1 to n Step by Step (n = 4)



Decision: **2 <= 4** → **true** → Yes.

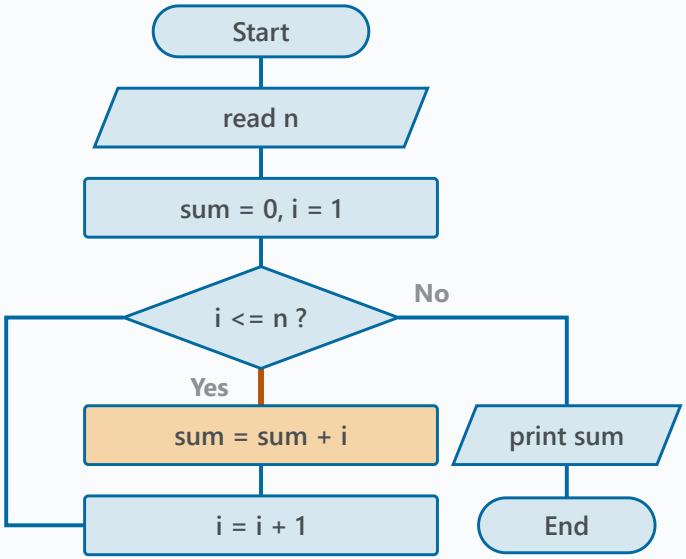
```
1 Start
2   read n
3   sum = 0, i = 1
▶   i <= n ?
5     Yes → sum = sum + i
6             i = i + 1
7             back to the decision
8     No → print sum
9 End
```

VARIABLES

`n = 4` `sum = 1` `i = 2`

OUTPUT

Sum from 1 to n Step by Step (n = 4)



sum = 1 + 2 = 3.

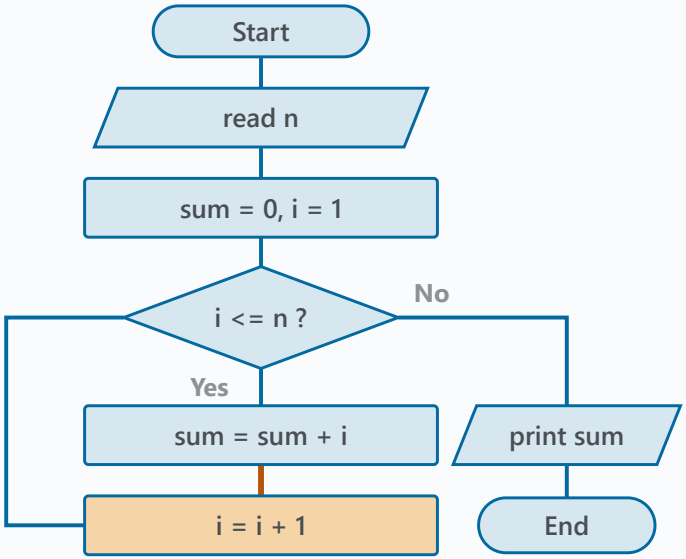
```
1  Start
2    read n
3    sum = 0, i = 1
4    i <= n ?
▶      Yes → sum = sum + i
6              i = i + 1
7              back to the decision
8      No → print sum
9  End
```

VARIABLES

`n = 4` `sum = 3` `i = 2`

OUTPUT

Sum from 1 to n Step by Step (n = 4)



i = 3; back to the decision.

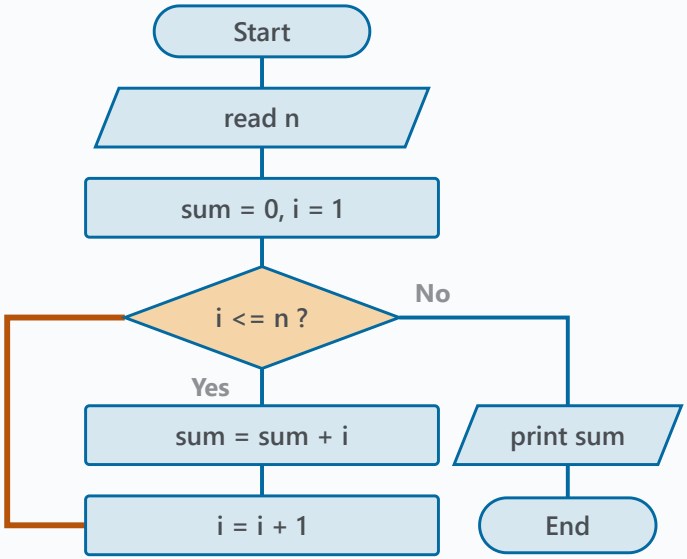
```
1 Start
2   read n
3   sum = 0, i = 1
4   i <= n ?
5     Yes → sum = sum + i
▶               i = i + 1
7               back to the decision
8     No → print sum
9 End
```

VARIABLES

`n = 4` `sum = 3` `i = 3`

OUTPUT

Sum from 1 to n Step by Step (n = 4)



Decision: **3 <= 4** → **true** → Yes.

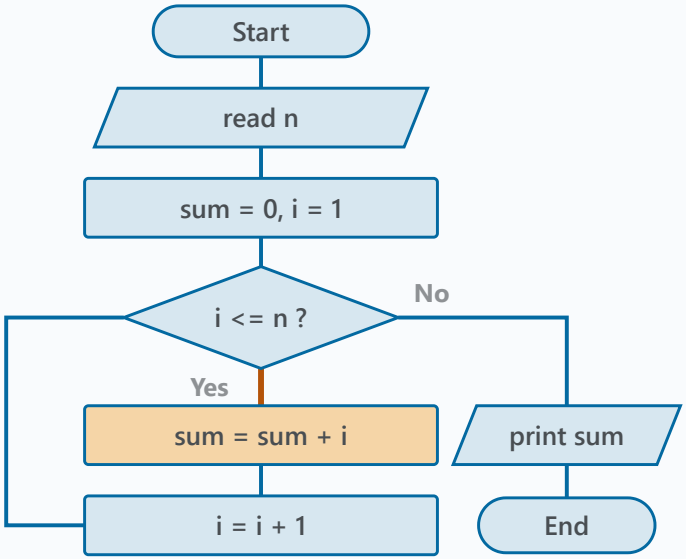
```
1 Start
2   read n
3   sum = 0, i = 1
▶   i <= n ?
5     Yes → sum = sum + i
6             i = i + 1
7             back to the decision
8     No → print sum
9 End
```

VARIABLES

`n = 4` `sum = 3` `i = 3`

OUTPUT

Sum from 1 to n Step by Step (n = 4)



sum = 3 + 3 = 6.

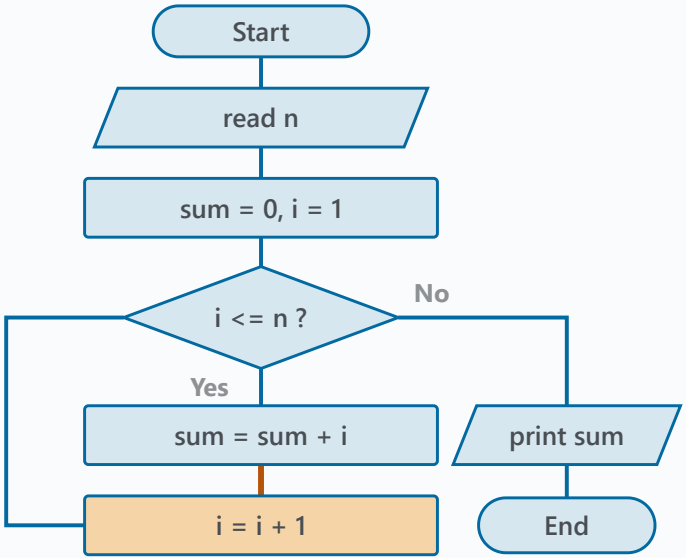
```
1  Start
2    read n
3    sum = 0, i = 1
4    i <= n ?
▶      Yes → sum = sum + i
6              i = i + 1
7              back to the decision
8      No → print sum
9  End
```

VARIABLES

`n = 4` `sum = 6` `i = 3`

OUTPUT

Sum from 1 to n Step by Step (n = 4)



i = 4; back to the decision.

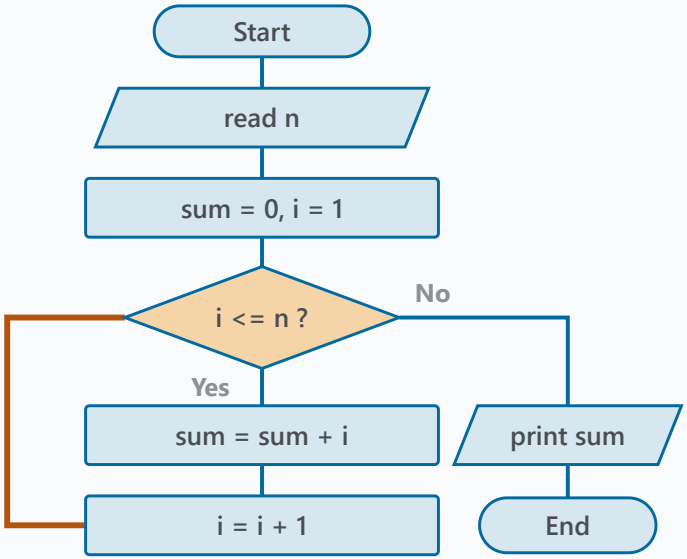
```
1 Start
2   read n
3   sum = 0, i = 1
4   i <= n ?
5     Yes → sum = sum + i
▶               i = i + 1
7               back to the decision
8     No → print sum
9 End
```

VARIABLES

`n = 4` `sum = 6` `i = 4`

OUTPUT

Sum from 1 to n Step by Step (n = 4)



Decision: **4 <= 4** → **true** → Yes.

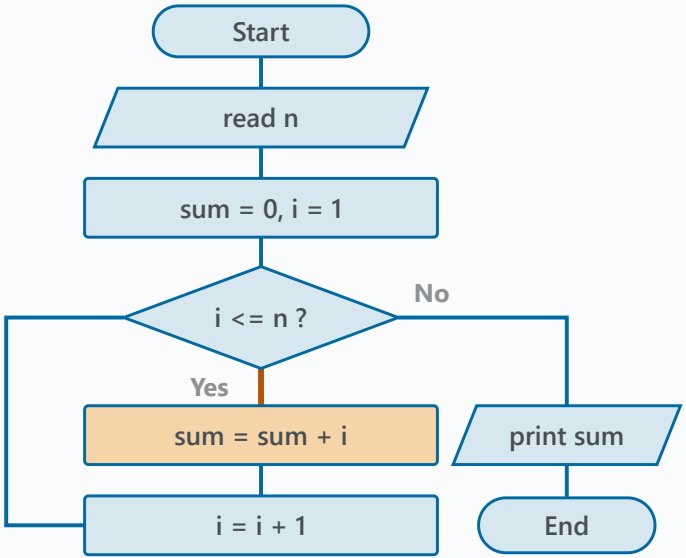
```
1 Start
2   read n
3   sum = 0, i = 1
▶   i <= n ?
5     Yes → sum = sum + i
6             i = i + 1
7             back to the decision
8     No → print sum
9 End
```

VARIABLES

`n = 4` `sum = 6` `i = 4`

OUTPUT

Sum from 1 to n Step by Step (n = 4)



sum = 6 + 4 = 10.

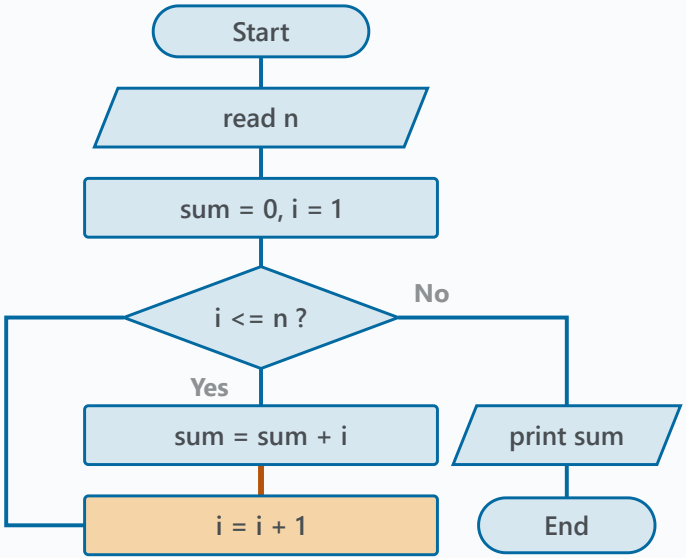
```
1  Start
2    read n
3    sum = 0, i = 1
4    i <= n ?
▶      Yes → sum = sum + i
6              i = i + 1
7              back to the decision
8      No → print sum
9  End
```

VARIABLES

`n = 4` `sum = 10` `i = 4`

OUTPUT

Sum from 1 to n Step by Step (n = 4)



i = 5; back to the decision.

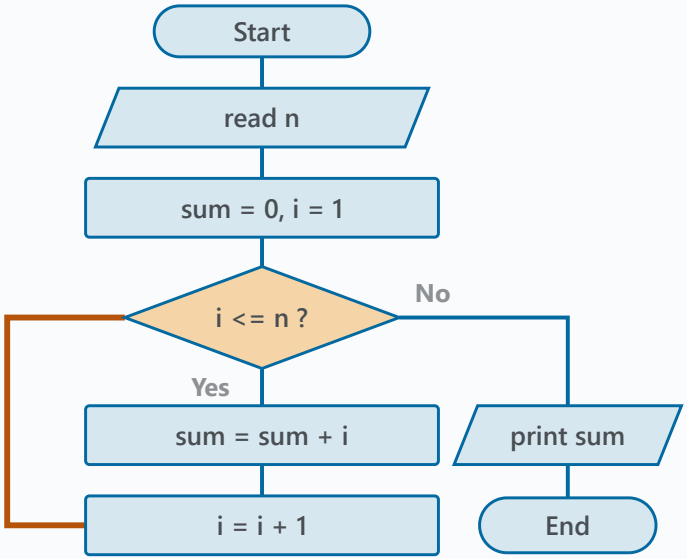
```
1 Start
2   read n
3   sum = 0, i = 1
4   i <= n ?
5     Yes → sum = sum + i
▶               i = i + 1
7               back to the decision
8     No → print sum
9 End
```

VARIABLES

`n = 4` `sum = 10` `i = 5`

OUTPUT

Sum from 1 to n Step by Step (n = 4)



Decision: **5 <= 4** → **false** → No, leave the loop.

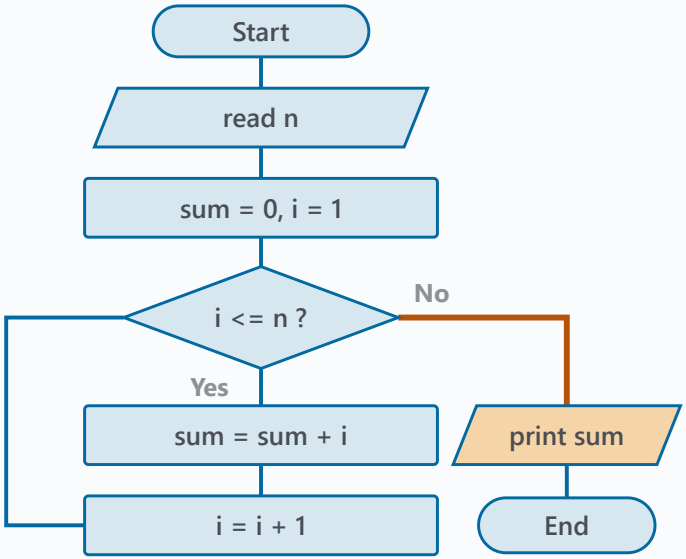
```
1 Start
2   read n
3   sum = 0, i = 1
▶   i <= n ?
5     Yes → sum = sum + i
6             i = i + 1
7             back to the decision
8     No → print sum
9 End
```

VARIABLES

`n = 4` `sum = 10` `i = 5`

OUTPUT

Sum from 1 to n Step by Step (n = 4)



10 is printed.

```
1 Start
2   read n
3   sum = 0, i = 1
4   i <= n ?
5     Yes → sum = sum + i
6             i = i + 1
7             back to the decision
▶     No → print sum
9 End
```

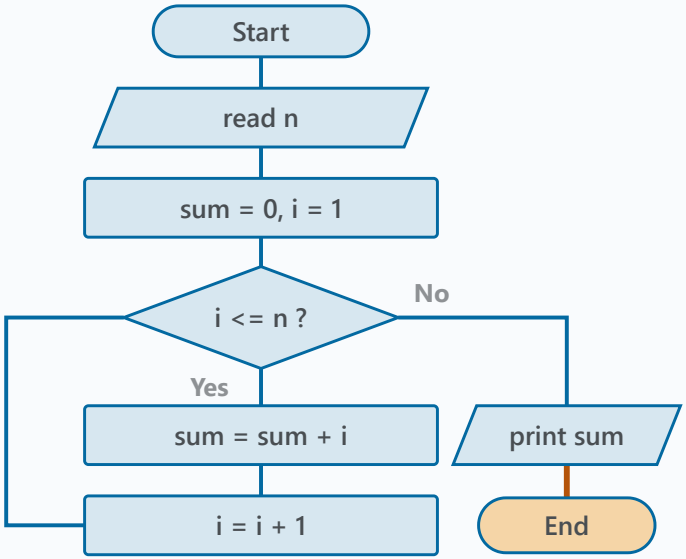
VARIABLES

`n = 4` `sum = 10` `i = 5`

OUTPUT

```
10
```

Sum from 1 to n Step by Step (n = 4)



End. The body ran n = 4 times; the formula confirms it: n(n+1)/2 = 10.

```
1 Start
2   read n
3   sum = 0, i = 1
4   i <= n ?
5     Yes → sum = sum + i
6               i = i + 1
7               back to the decision
8     No → print sum
▶ End
```

VARIABLES

`n = 4` `sum = 10` `i = 5`

OUTPUT

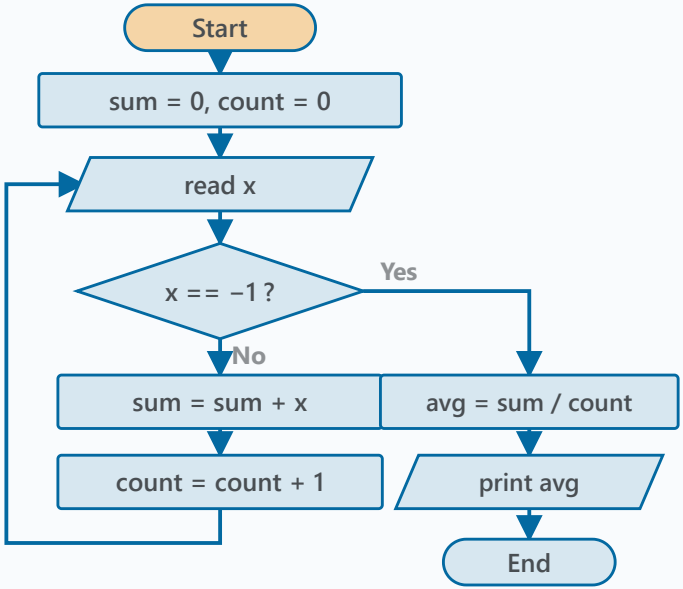
```
10
```

Counter-Controlled and Sentinel Loops

	Counter-controlled	Sentinel-controlled
Number of repetitions	Known in advance (n times)	Unknown; stops when a special value arrives
Decision	i <= n ?	x == −1 ? (sentinel yet?)
Example	Read the grades of 30 students in a class	Items scanned at a checkout, until the “done” key
Usually in Java	`for`	`while`

- Choose a **sentinel** that cannot occur in real data: if grades are 0–100, −1 works and 0 does not.
- The sentinel must not join the data: ask the decision **before** adding and counting; the back arrow returns to the **read** box.
- If the first input is the sentinel, the body never runs and the counter stays 0; before dividing for an average, ask **count > 0 ?**.

Sentinel Loop Step by Step (80, 64, 90, −1)



Start. The user enters grades and types **−1** when done.
Inputs: 80, 64, 90, −1.

```
▶ Start
2   sum = 0, count = 0
3   read x
4   x == -1 ?  Yes → go to line 7
5   sum = sum + x
6   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

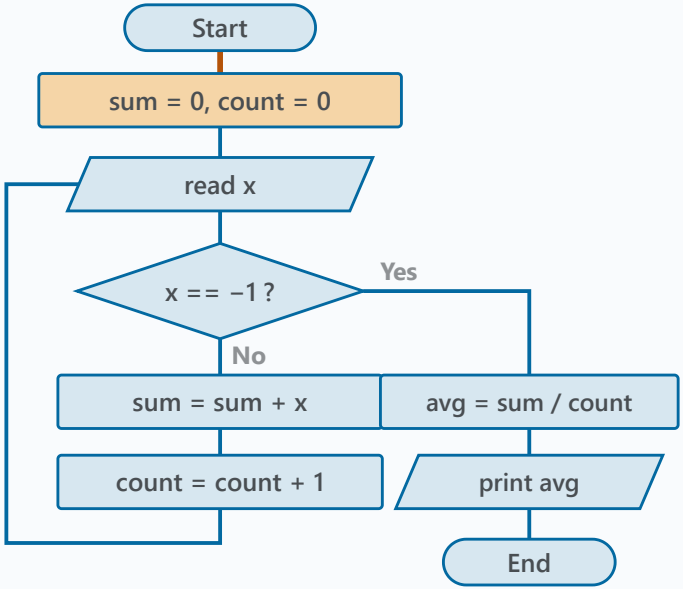
VARIABLES

—

OUTPUT

Step 1 / 19

Sentinel Loop Step by Step (80, 64, 90, −1)



Accumulator and counter start at 0.

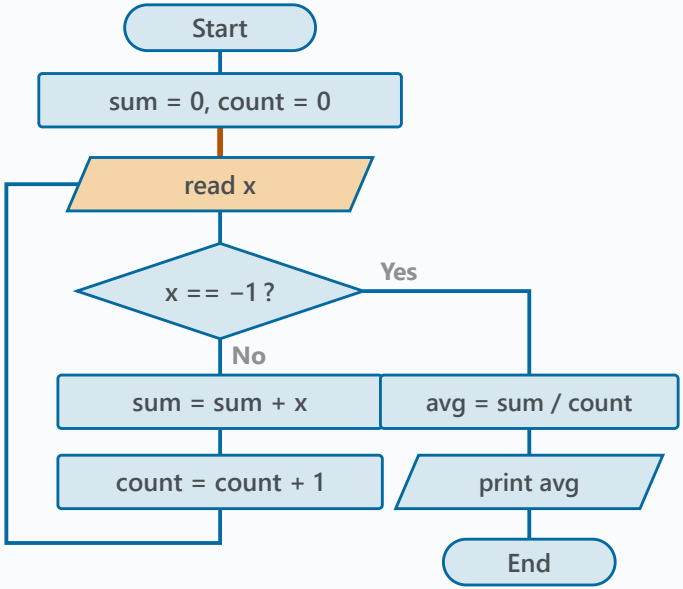
```
1 Start
▶   sum = 0, count = 0
3   read x
4   x == -1 ?  Yes → go to line 7
5   sum = sum + x
6   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

VARIABLES

`sum = 0` `count = 0`

OUTPUT

Sentinel Loop Step by Step (80, 64, 90, −1)



Read x = **80**.

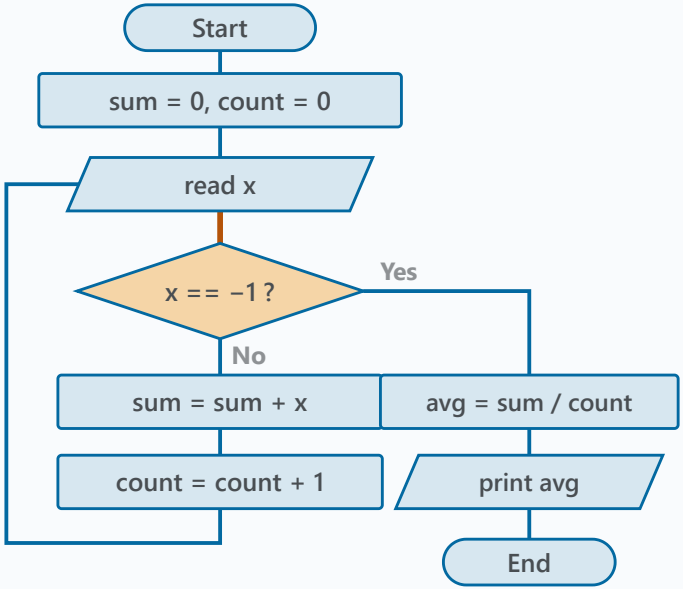
```
1 Start
2   sum = 0, count = 0
▶   read x
4   x == -1 ?  Yes → go to line 7
5   sum = sum + x
6   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

VARIABLES

`sum = 0` `count = 0` `x = 80`

OUTPUT

Sentinel Loop Step by Step (80, 64, 90, −1)




```
1 Start
2   sum = 0, count = 0
3   read x
▶   x == -1 ?  Yes → go to line 7
5   sum = sum + x
6   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

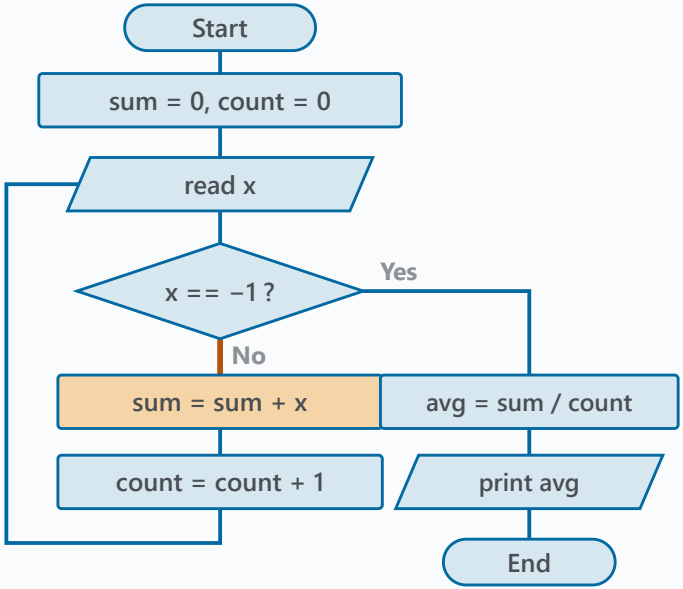
VARIABLES

`sum = 0` `count = 0` `x = 80`

OUTPUT

80 == −1 → **false** → No: this is a real grade.

Sentinel Loop Step by Step (80, 64, 90, −1)



sum = 0 + 80 = **80**.

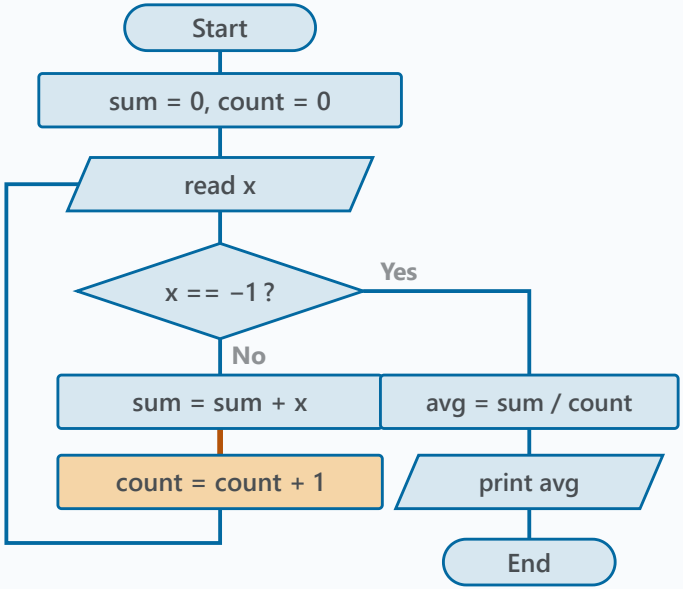
```
1 Start
2   sum = 0, count = 0
3   read x
4   x == -1 ?  Yes → go to line 7
▶   sum = sum + x
6   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

VARIABLES

`sum = 80` `count = 0` `x = 80`

OUTPUT

Sentinel Loop Step by Step (80, 64, 90, −1)



count = **1**.

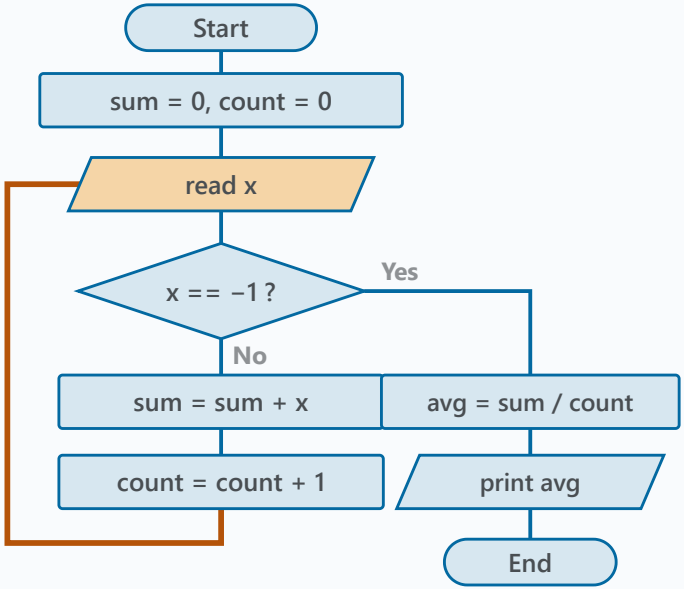
```
1 Start
2   sum = 0, count = 0
3   read x
4   x == -1 ?  Yes → go to line 7
5   sum = sum + x
▶   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

VARIABLES

`sum = 80` `count = 1` `x = 80`

OUTPUT

Sentinel Loop Step by Step (80, 64, 90, −1)



Read x = **64**. The back arrow returns to the **read** box.

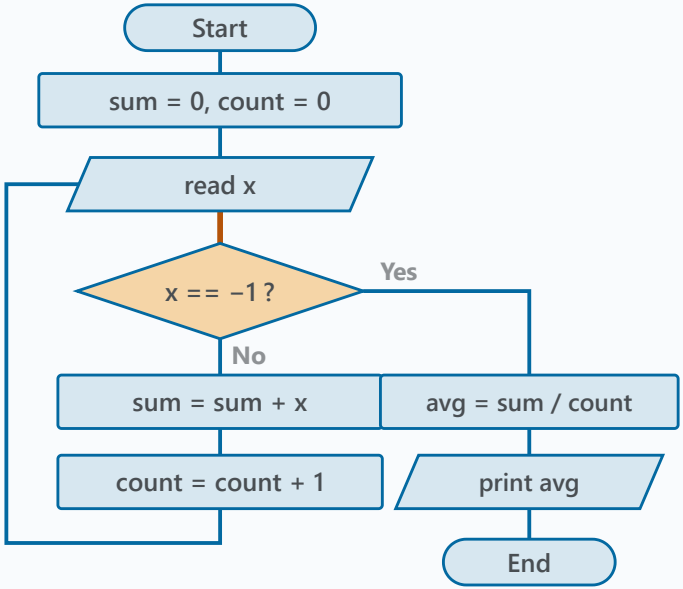
```
1 Start
2   sum = 0, count = 0
▶   read x
4   x == -1 ?  Yes → go to line 7
5   sum = sum + x
6   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

VARIABLES

`sum = 80` `count = 1` `x = 64`

OUTPUT

Sentinel Loop Step by Step (80, 64, 90, −1)



64 == −1 → **false** → No: this is a real grade.

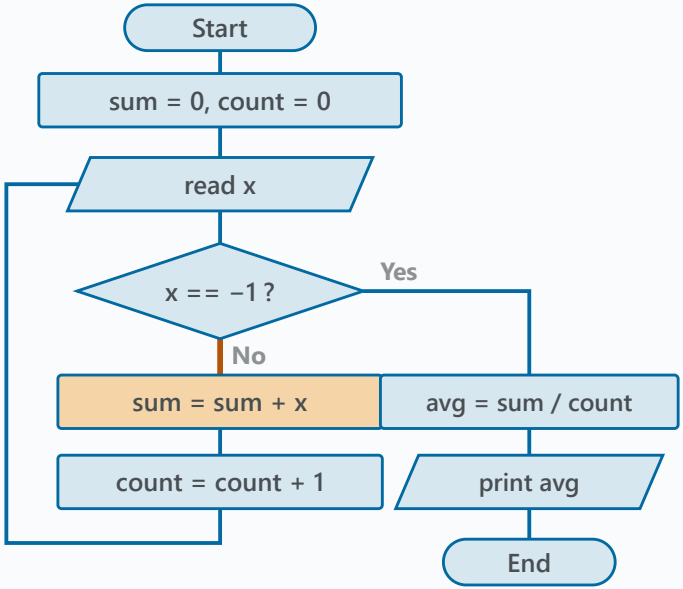
```
1 Start
2   sum = 0, count = 0
3   read x
▶   x == -1 ?  Yes → go to line 7
5   sum = sum + x
6   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

VARIABLES

`sum = 80` `count = 1` `x = 64`

OUTPUT

Sentinel Loop Step by Step (80, 64, 90, −1)



sum = 80 + 64 = **144**.

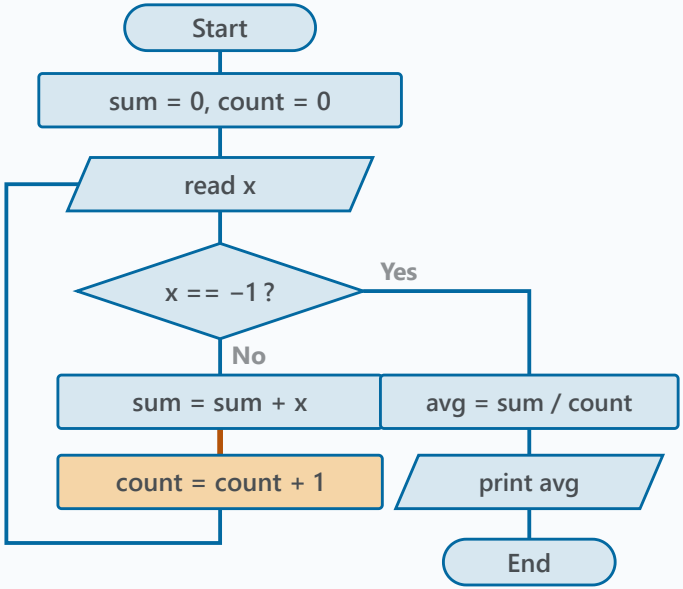
```
1 Start
2   sum = 0, count = 0
3   read x
4   x == -1 ?  Yes → go to line 7
▶   sum = sum + x
6   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

VARIABLES

`sum = 144` `count = 1` `x = 64`

OUTPUT

Sentinel Loop Step by Step (80, 64, 90, −1)



count = **2**.

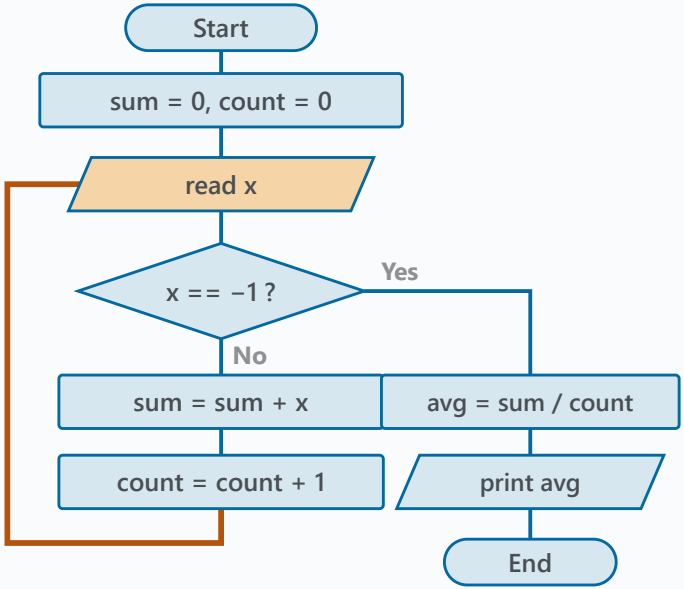
```
1 Start
2   sum = 0, count = 0
3   read x
4   x == -1 ?  Yes → go to line 7
5   sum = sum + x
▶   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

VARIABLES

`sum = 144` `count = 2` `x = 64`

OUTPUT

Sentinel Loop Step by Step (80, 64, 90, −1)



Read x = **90**. The back arrow returns to the **read** box.

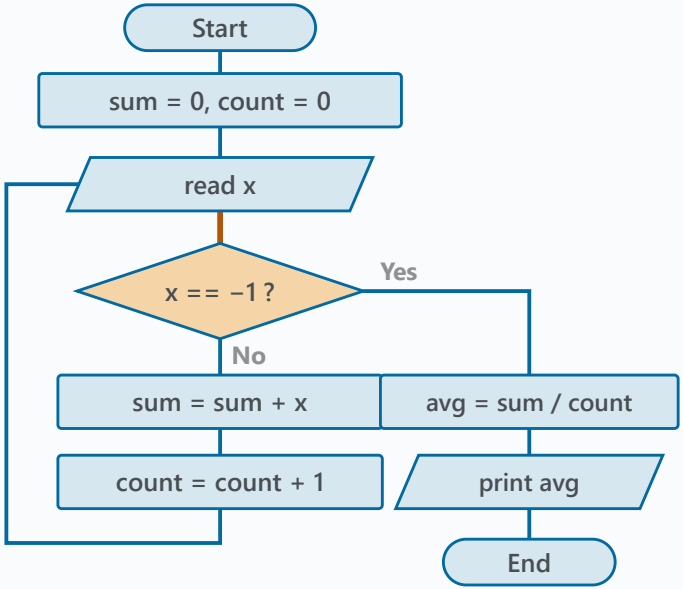
```
1 Start
2   sum = 0, count = 0
▶   read x
4   x == -1 ?  Yes → go to line 7
5   sum = sum + x
6   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

VARIABLES

`sum = 144` `count = 2` `x = 90`

OUTPUT

Sentinel Loop Step by Step (80, 64, 90, −1)



90 == −1 → **false** → No: this is a real grade.

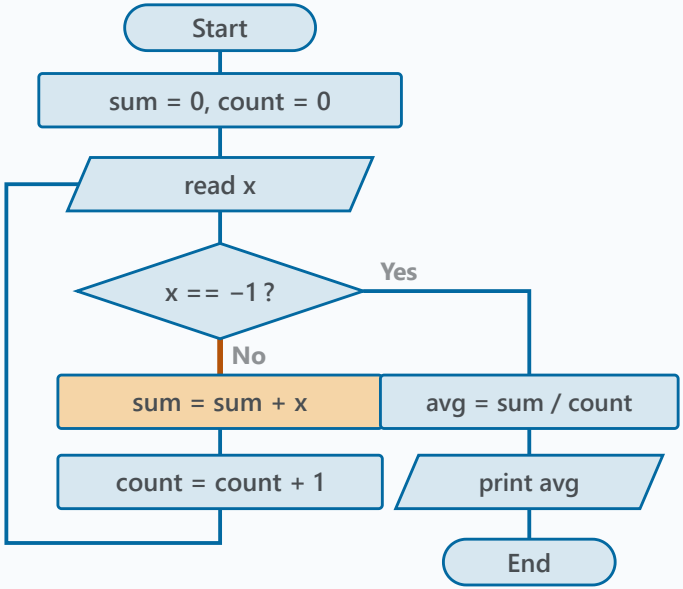
```
1 Start
2   sum = 0, count = 0
3   read x
▶   x == -1 ?  Yes → go to line 7
5   sum = sum + x
6   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

VARIABLES

`sum = 144` `count = 2` `x = 90`

OUTPUT

Sentinel Loop Step by Step (80, 64, 90, −1)



sum = 144 + 90 = **234**.

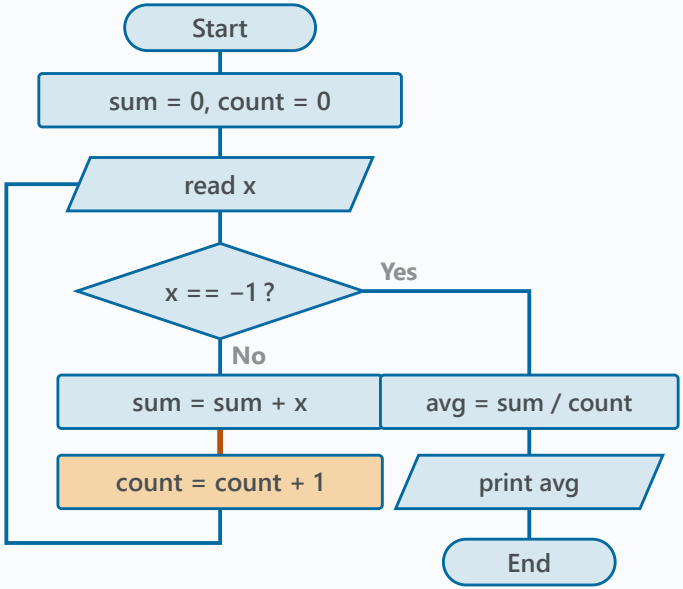
```
1 Start
2   sum = 0, count = 0
3   read x
4   x == -1 ?  Yes → go to line 7
▶   sum = sum + x
6   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

VARIABLES

sum = **234** | count = **2** | x = **90**

OUTPUT

Sentinel Loop Step by Step (80, 64, 90, −1)



count = **3**.

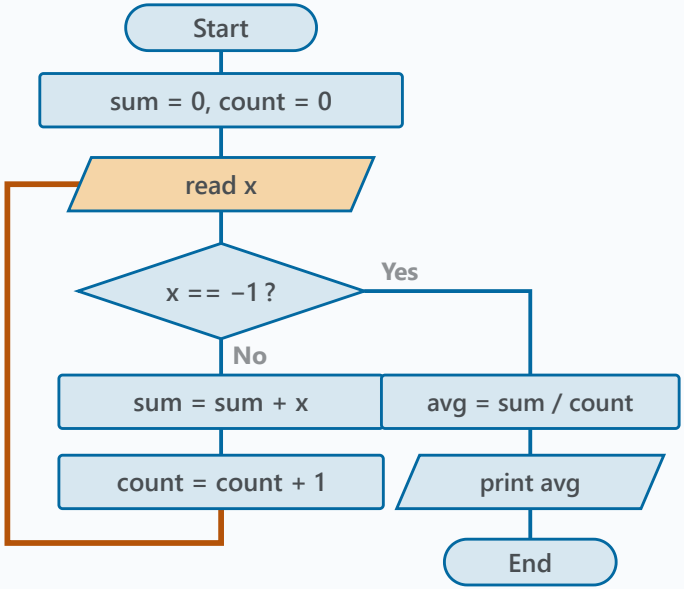
```
1 Start
2   sum = 0, count = 0
3   read x
4   x == -1 ?  Yes → go to line 7
5   sum = sum + x
▶   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

VARIABLES

`sum = 234` `count = 3` `x = 90`

OUTPUT

Sentinel Loop Step by Step (80, 64, 90, −1)



Read x = **−1**.

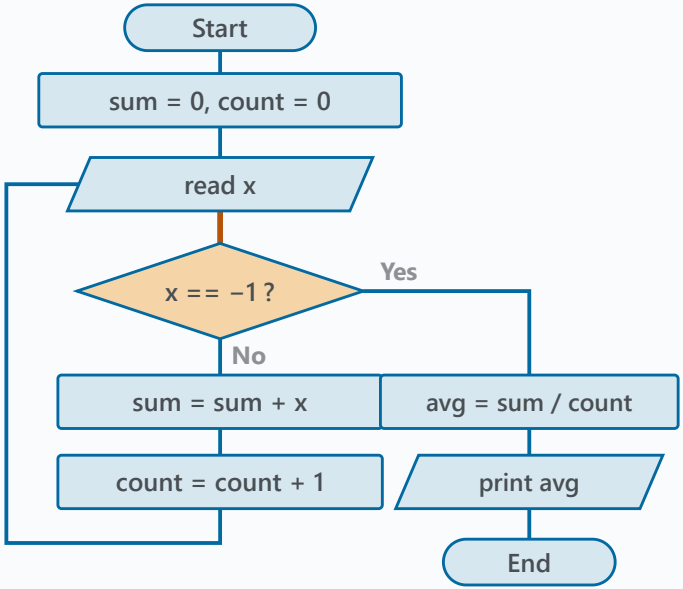
```
1 Start
2   sum = 0, count = 0
▶   read x
4   x == -1 ?  Yes → go to line 7
5   sum = sum + x
6   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

VARIABLES

`sum = 234` `count = 3` `x = −1`

OUTPUT

Sentinel Loop Step by Step (80, 64, 90, −1)




```
1 Start
2   sum = 0, count = 0
3   read x
▶   x == -1 ?  Yes → go to line 7
5   sum = sum + x
6   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
9 End
```

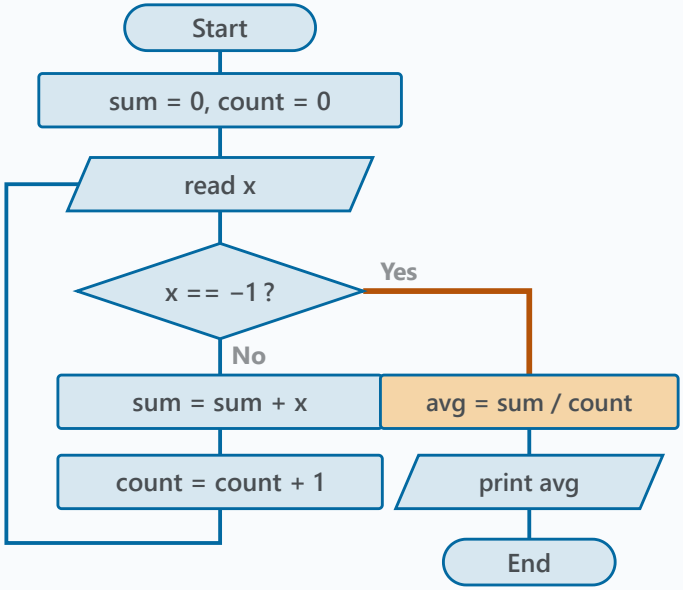
VARIABLES

`sum = 234` `count = 3` `x = −1`

OUTPUT

−1 == −1 → **true** → Yes: the sentinel arrived, so the loop ends. −1 was **not added** to the sum or the count.

Sentinel Loop Step by Step (80, 64, 90, −1)



avg = 234 / 3 = **78**.

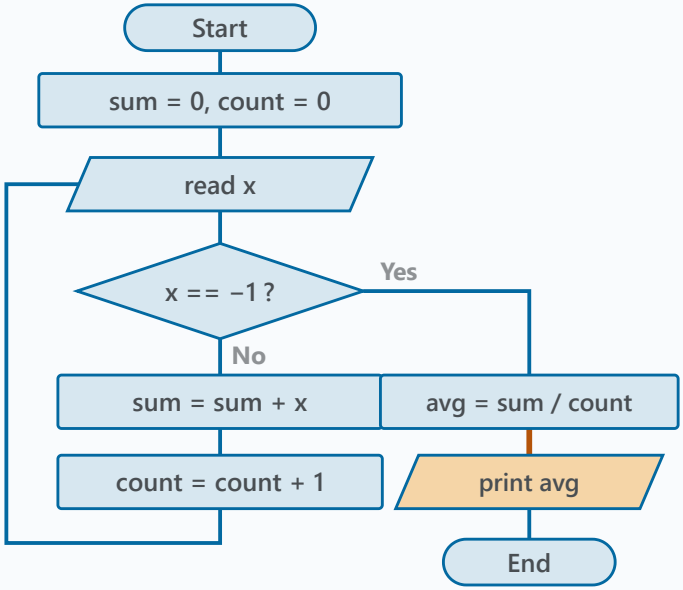
```
1 Start
2   sum = 0, count = 0
3   read x
4   x == -1 ?  Yes → go to line 7
5   sum = sum + x
6   count = count + 1;  back to line 3
▶   avg = sum / count
8   print avg
9 End
```

VARIABLES

`sum = 234` `count = 3` `x = −1` `avg = 78`

OUTPUT

Sentinel Loop Step by Step (80, 64, 90, −1)



The average is printed.

```
1 Start
2   sum = 0, count = 0
3   read x
4   x == -1 ?  Yes → go to line 7
5   sum = sum + x
6   count = count + 1;  back to line 3
7   avg = sum / count
▶   print avg
9 End
```

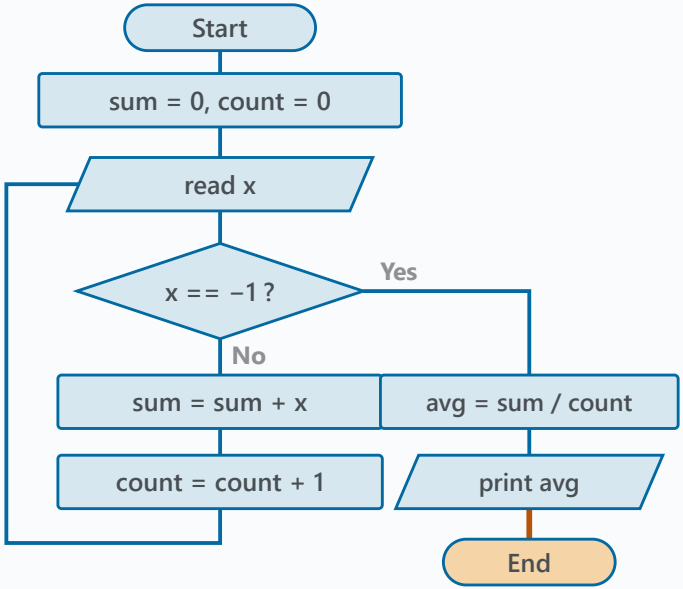
VARIABLES

sum = 234 | count = 3 | x = −1 | avg = 78

OUTPUT

```
78
```

Sentinel Loop Step by Step (80, 64, 90, −1)



End. The body ran 3 times, the decision 4 times. The number of grades was unknown in advance; the **data** stopped the loop.

```
1 Start
2   sum = 0, count = 0
3   read x
4   x == -1 ?  Yes → go to line 7
5   sum = sum + x
6   count = count + 1;  back to line 3
7   avg = sum / count
8   print avg
▶ End
```

VARIABLES

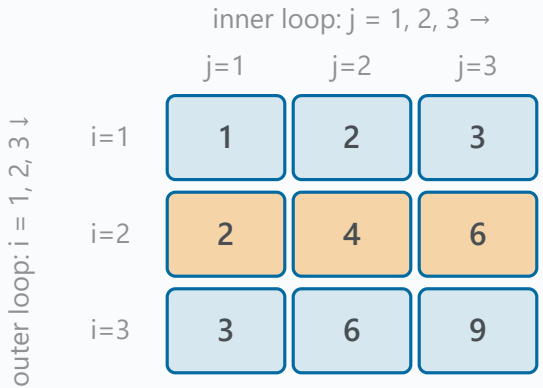
`sum = 234` `count = 3` `x = −1` `avg = 78`

OUTPUT

```
78
```

Nested Structures

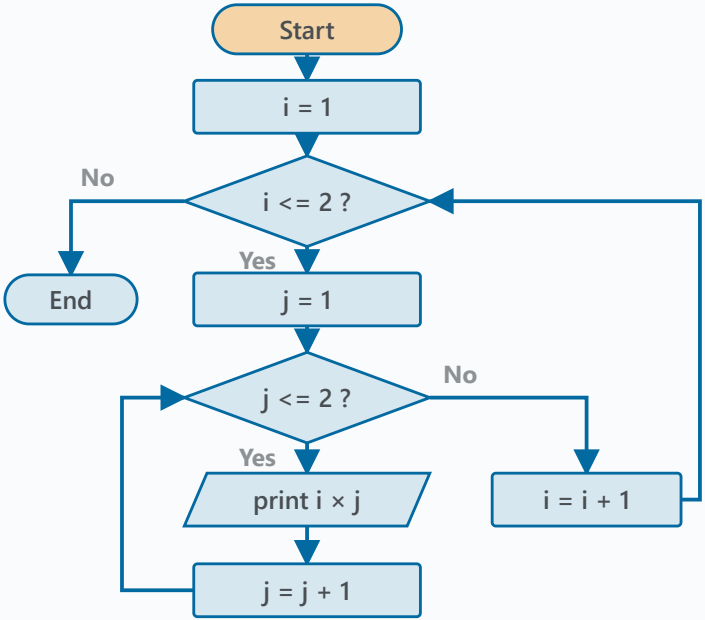
- A decision's branch can contain another decision.
- A loop can contain a second loop: on **every pass** of the outer loop, the inner loop runs from start to finish.
- The entry and exit of each structure should be visually distinct.



3 × 3 = 9 cells: the inner body runs 9 times. Orange row: the three passes of j while i = 2.

Deep nesting may be a sign that the algorithm should be split into subprocesses.

Nested Loop Step by Step (i = 1)



Start. Outer loop i = 1, 2; inner loop j = 1, 2. Each row prints i × j.

```
▶ Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

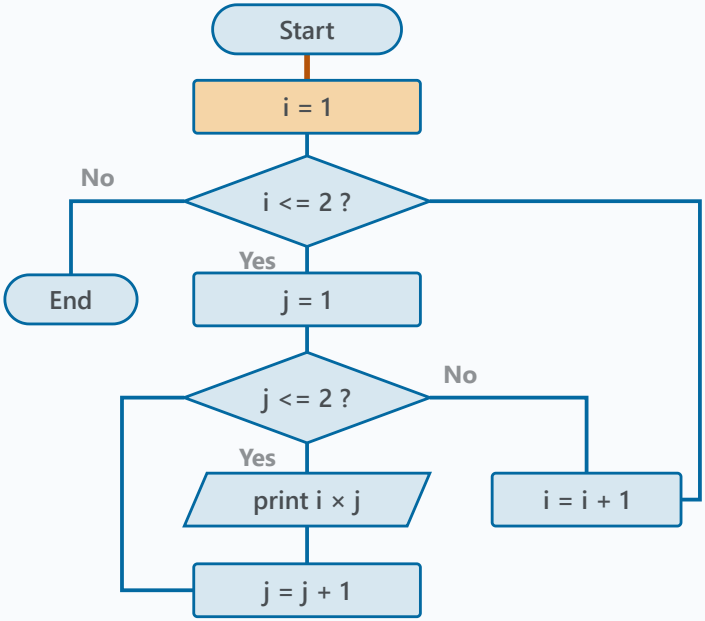
VARIABLES

—

OUTPUT

Step 1 / 12

Nested Loop Step by Step (i = 1)



i = 1: the outer counter starts.

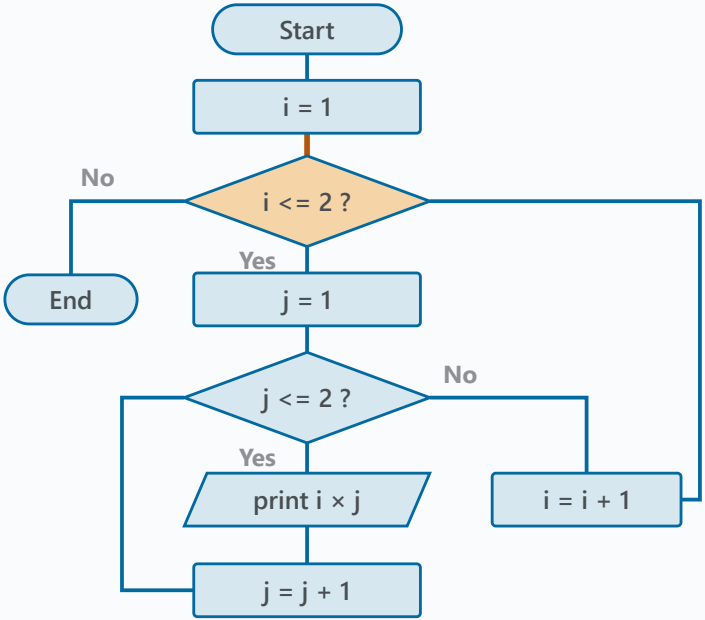
```
1 Start
▶   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

VARIABLES

`i = 1`

OUTPUT

Nested Loop Step by Step (i = 1)



Outer decision: **1 <= 2** → **true** → enter the inner loop.

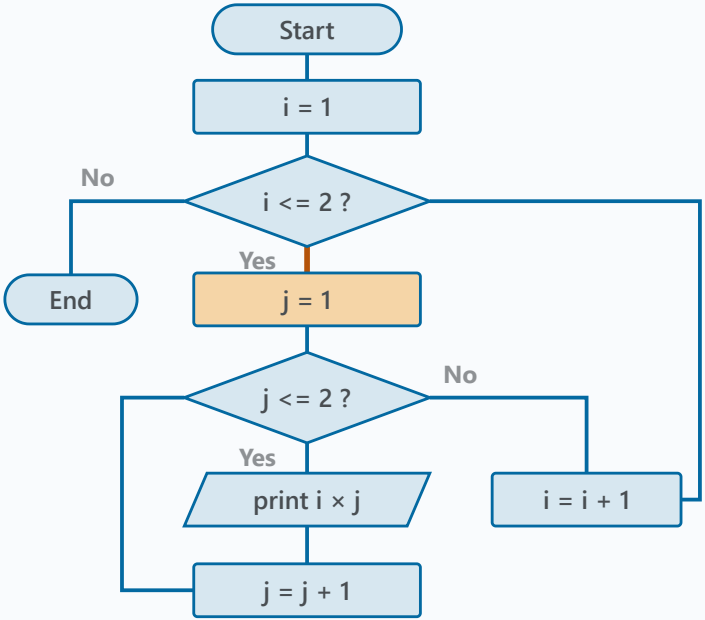
```
1 Start
2   i = 1
▶   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

VARIABLES

`i = 1`

OUTPUT

Nested Loop Step by Step (i = 1)



j = 1: the inner counter is reset to 1 **on every** outer pass.

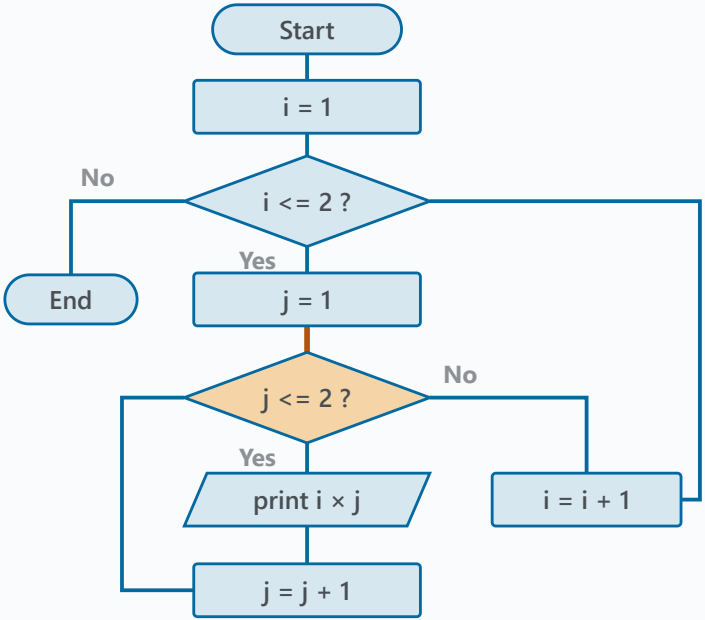
```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
▶     j = 1
5     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

VARIABLES

`i = 1` `j = 1`

OUTPUT

Nested Loop Step by Step (i = 1)



Inner decision: **1 <= 2 → true**

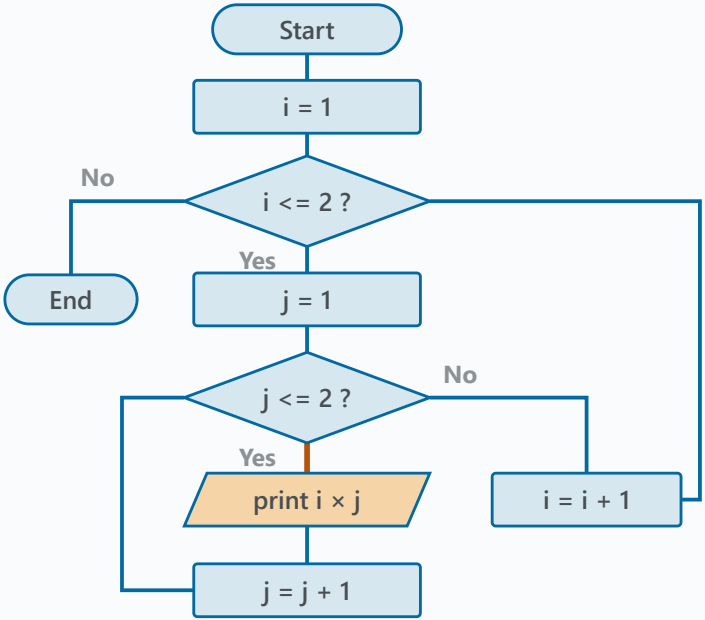
```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
▶     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

VARIABLES

`i = 1` `j = 1`

OUTPUT

Nested Loop Step by Step (i = 1)



1 × 1 = 1 is printed.

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
▶       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

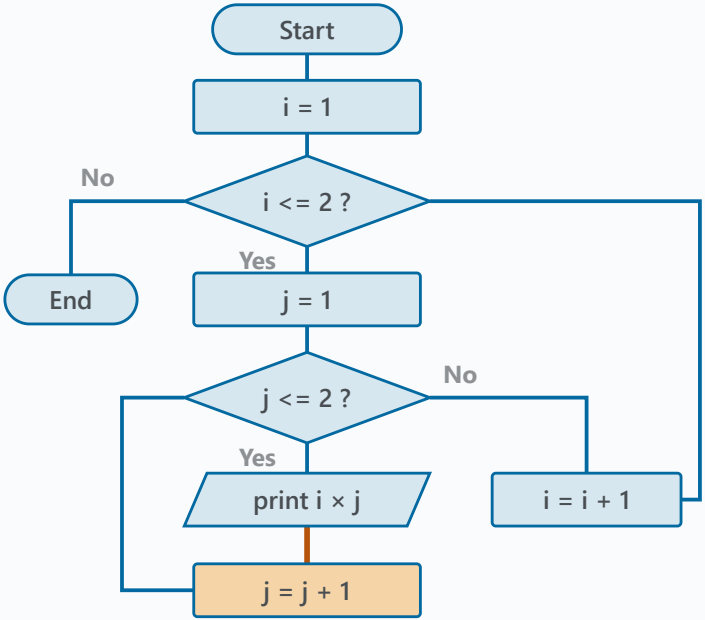
VARIABLES

`i = 1` `j = 1`

OUTPUT

```
1
```

Nested Loop Step by Step (i = 1)



j = 2; back to the inner decision.

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
6       print i × j
▶       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

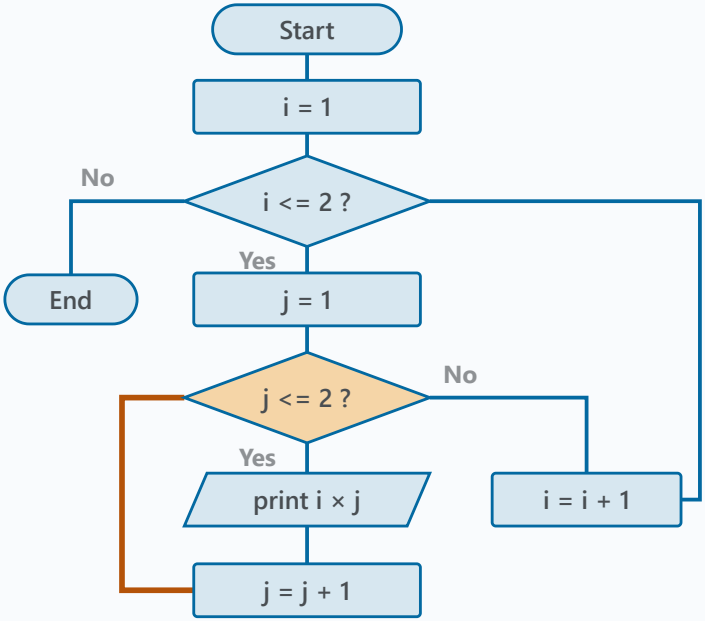
VARIABLES

`i = 1` `j = 2`

OUTPUT

```
1
```

Nested Loop Step by Step (i = 1)



Inner decision: **2 <= 2 → true**

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
▶     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

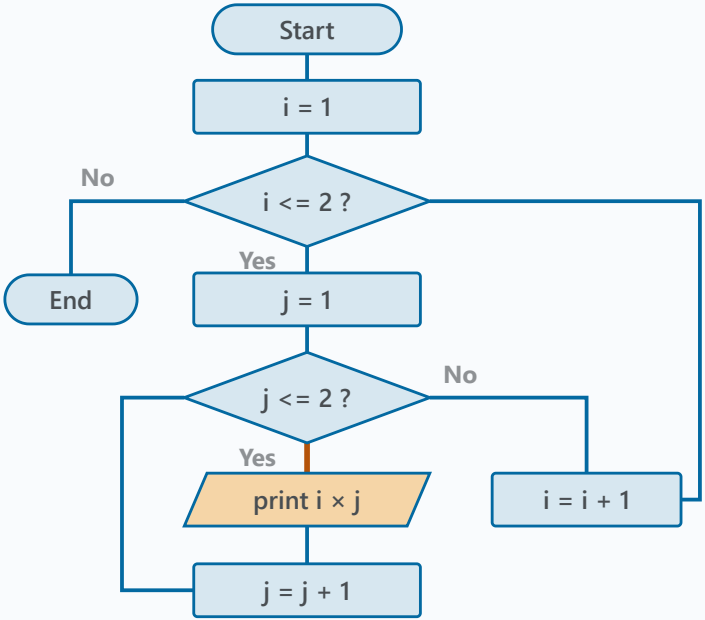
VARIABLES

`i = 1` `j = 2`

OUTPUT

```
1
```

Nested Loop Step by Step (i = 1)



1 × 2 = 2 is printed.

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
▶       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

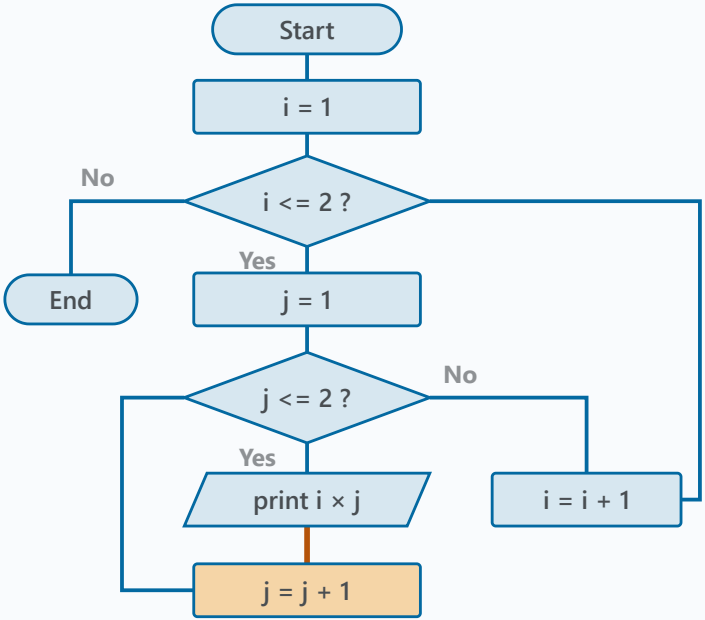
VARIABLES

`i = 1` `j = 2`

OUTPUT

```
1 2
```

Nested Loop Step by Step (i = 1)



j = 3; back to the inner decision.

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
6       print i × j
▶       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

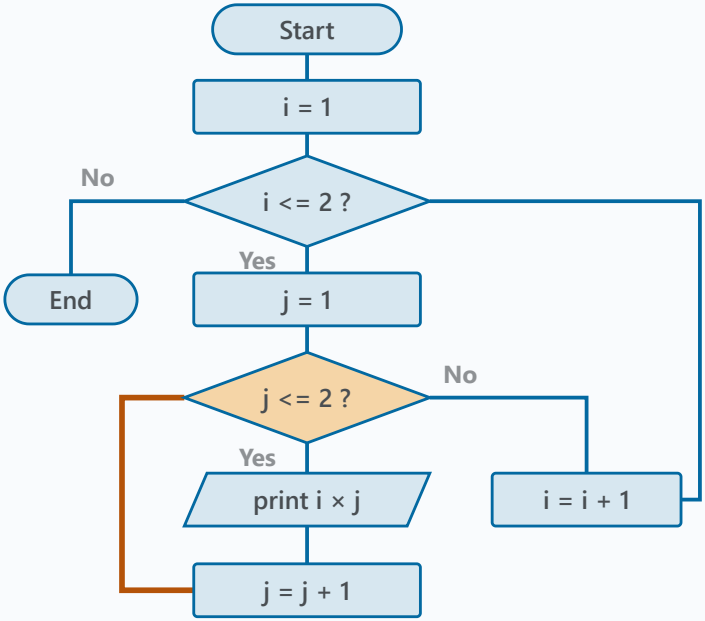
VARIABLES

`i = 1` `j = 3`

OUTPUT

```
1 2
```

Nested Loop Step by Step (i = 1)



Inner decision: **3 <= 2** → **false** → No, the inner loop ends.

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
▶     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back to outer decision
9 End
```

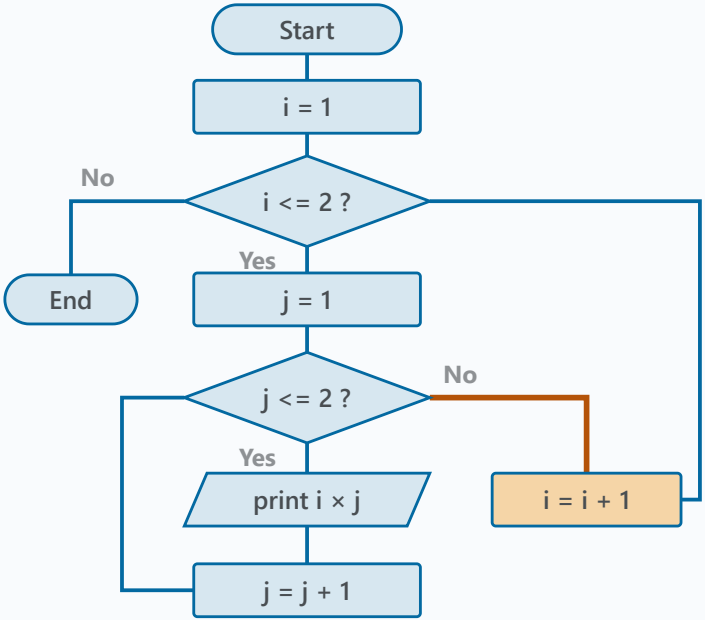
VARIABLES

`i = 1` `j = 3`

OUTPUT

```
1 2
```

Nested Loop Step by Step (i = 1)



Row done: **println()** ends the line (the cursor moves down). Then **i = 2**; back to the outer decision.

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
▶     end line (println), i = i + 1, back
  to outer decision
9 End
```

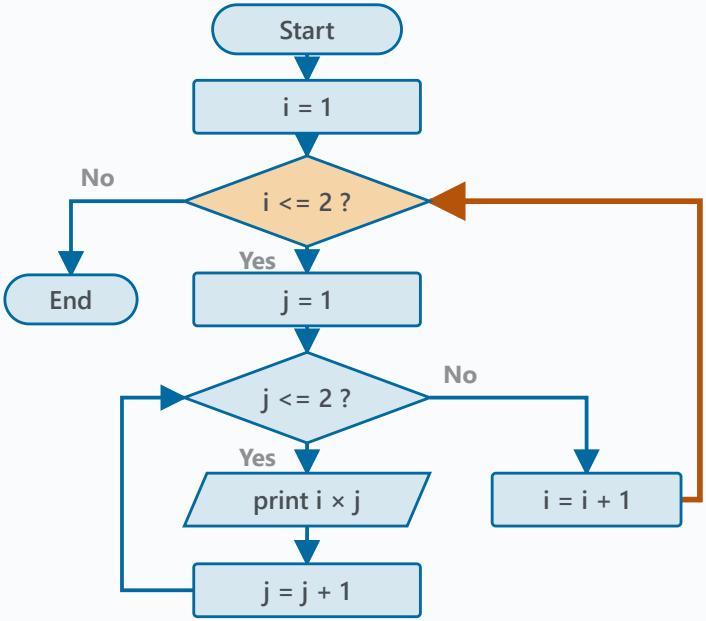
VARIABLES

`i = 2` `j = 3`

OUTPUT

```
1 2
```

Nested Loop Step by Step (cont.: i = 2)



Outer decision: **2 <= 2** → **true** → enter the inner loop.

```
1 Start
2   i = 1
▶   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

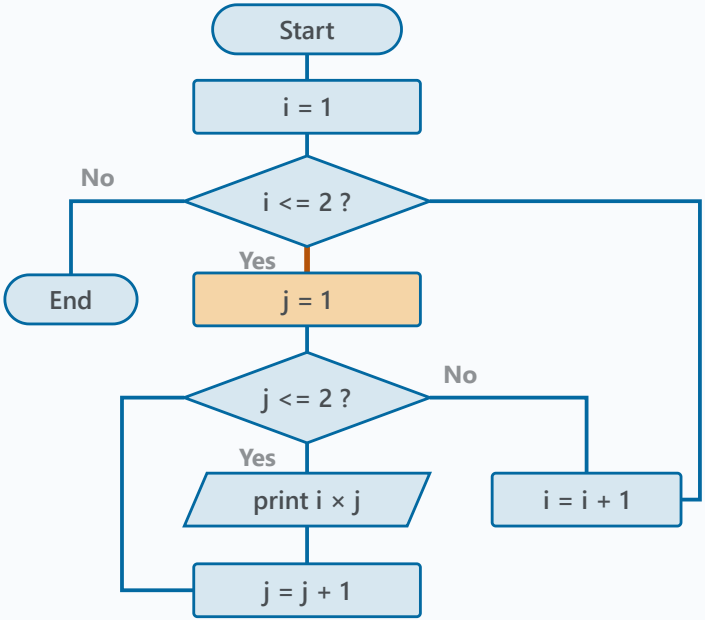
VARIABLES

`i = 2`

OUTPUT

```
1 2
```

Nested Loop Step by Step (cont.: i = 2)



j = 1: the inner counter is reset to 1 **on every** outer pass.

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
▶     j = 1
5     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

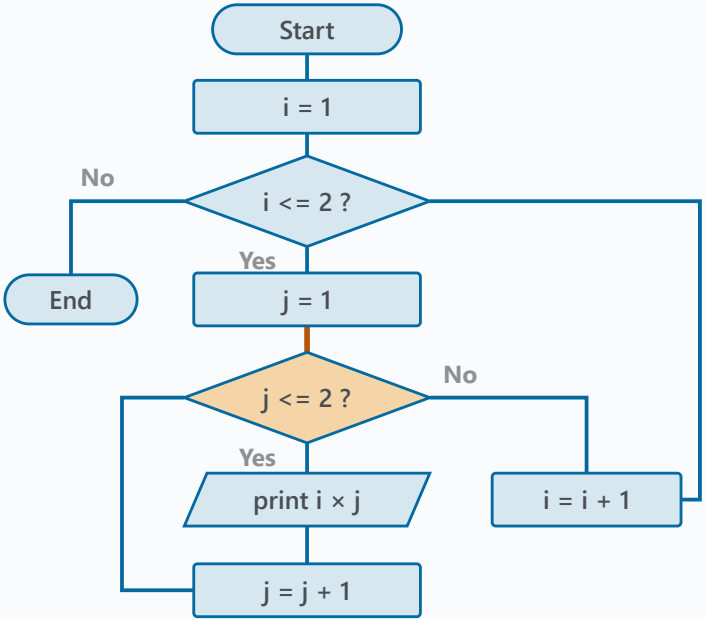
VARIABLES

`i = 2` `j = 1`

OUTPUT

```
1 2
```

Nested Loop Step by Step (cont.: i = 2)



Inner decision: **1 <= 2 → true**

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
▶     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

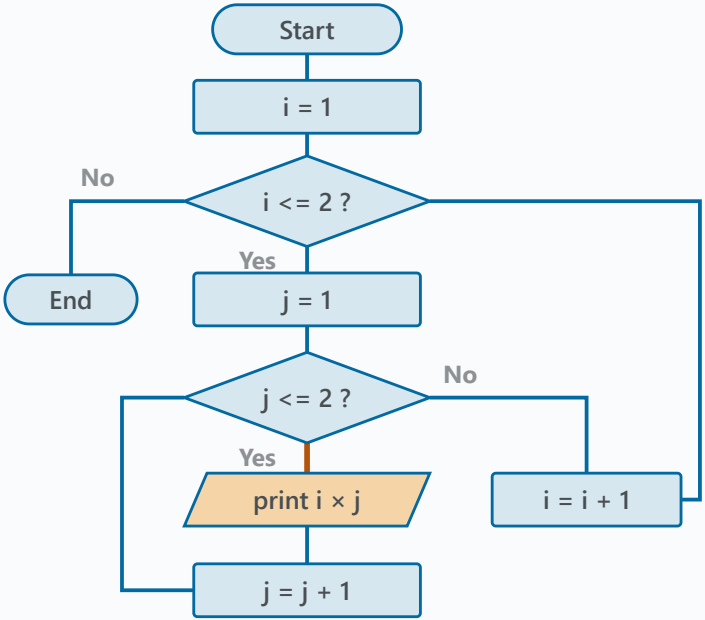
VARIABLES

`i = 2` `j = 1`

OUTPUT

```
1 2
```

Nested Loop Step by Step (cont.: i = 2)



2 × 1 = 2 is printed.

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
▶       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

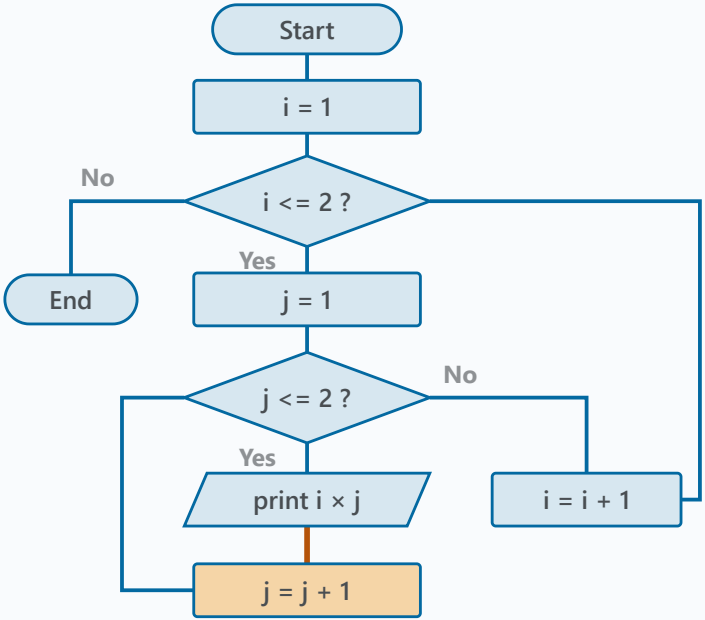
VARIABLES

`i = 2` `j = 1`

OUTPUT

```
1 2
2
```

Nested Loop Step by Step (cont.: i = 2)



j = 2; back to the inner decision.

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
6       print i × j
▶       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

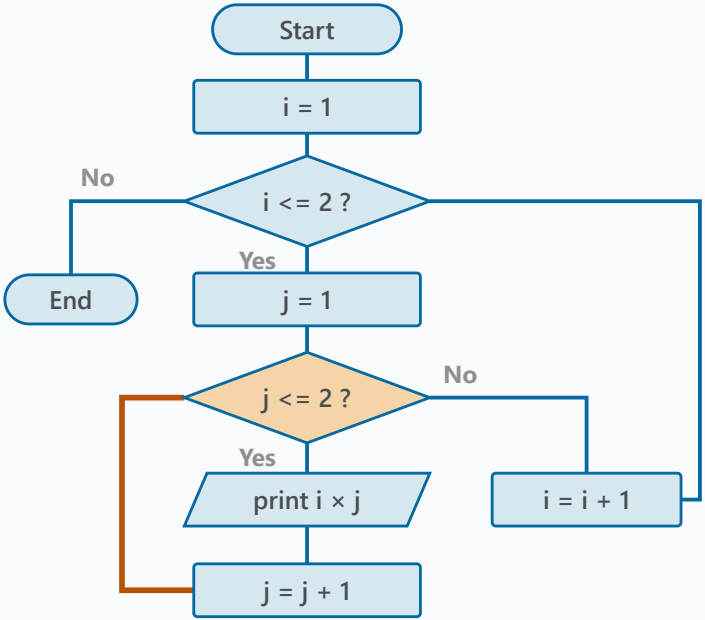
VARIABLES

`i = 2` `j = 2`

OUTPUT

```
1 2
2
```

Nested Loop Step by Step (cont.: i = 2)



Inner decision: **2 <= 2 → true**

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
▶     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

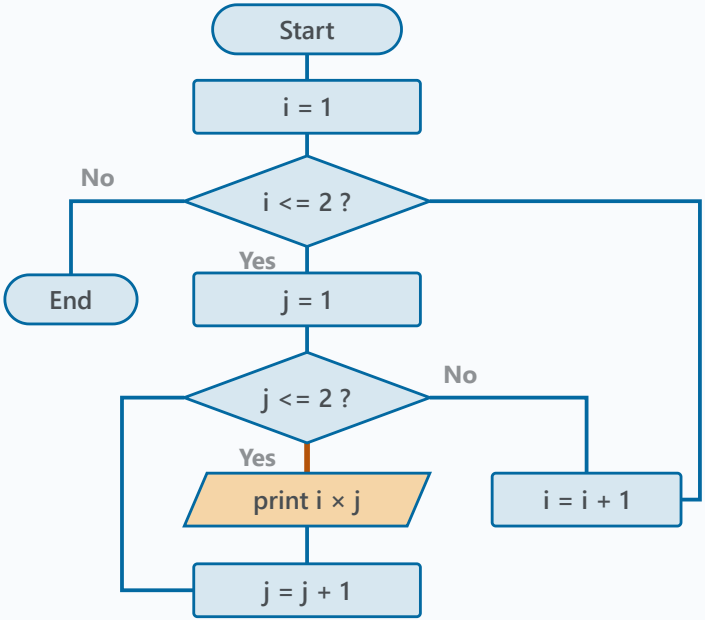
VARIABLES

`i = 2` `j = 2`

OUTPUT

```
1 2
2
```

Nested Loop Step by Step (cont.: i = 2)



2 × 2 = 4 is printed.

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
▶       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

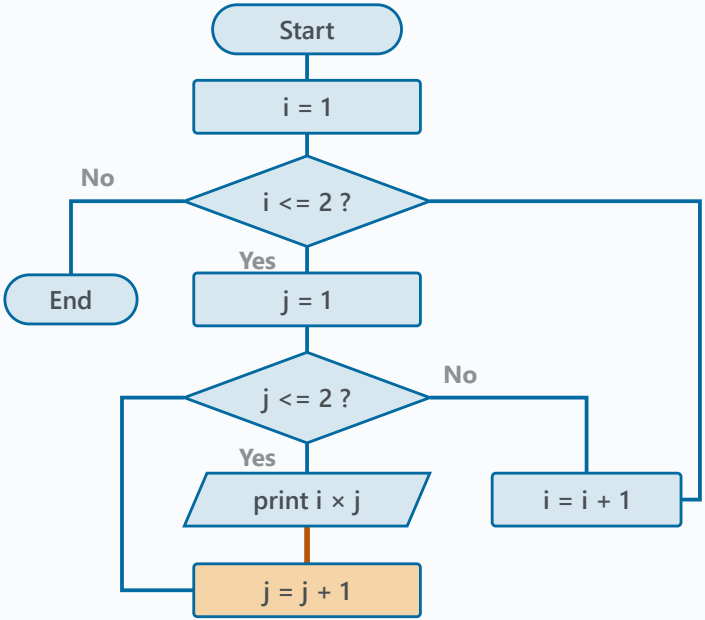
VARIABLES

`i = 2` `j = 2`

OUTPUT

```
1 2
2 4
```

Nested Loop Step by Step (cont.: i = 2)



j = 3; back to the inner decision.

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
6       print i × j
▶       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
9 End
```

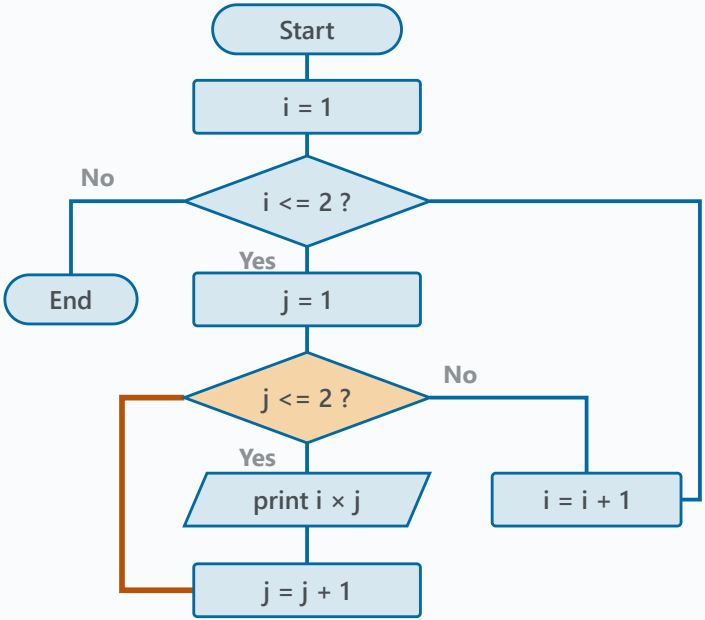
VARIABLES

`i = 2` `j = 3`

OUTPUT

```
1 2
2 4
```

Nested Loop Step by Step (cont.: i = 2)



Inner decision: **3 <= 2** → **false** → No, the inner loop ends.

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
▶     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back to outer decision
9 End
```

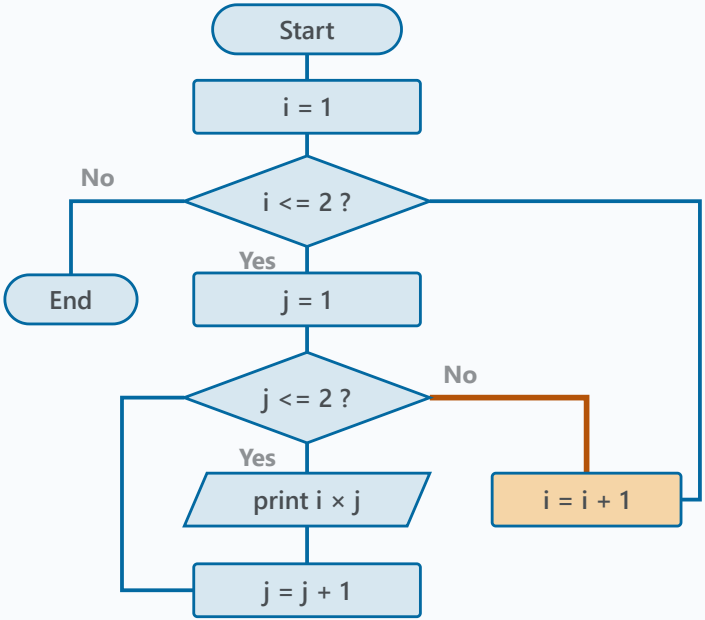
VARIABLES

`i = 2` `j = 3`

OUTPUT

```
1 2
2 4
```

Nested Loop Step by Step (cont.: i = 2)



Row done: **println()** ends the line. Then **i = 3**; back to the outer decision.

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
▶     end line (println), i = i + 1, back
  to outer decision
9 End
```

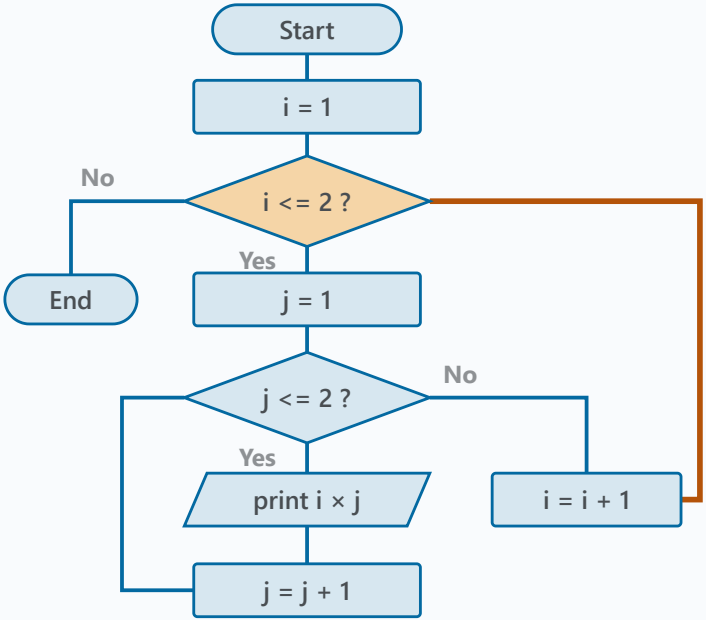
VARIABLES

`i = 3` `j = 3`

OUTPUT

```
1 2
2 4
```

Nested Loop Step by Step (cont.: i = 2)



Outer decision: **3 <= 2** → **false** → No, go to End.

```
1 Start
2   i = 1
▶   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back to outer decision
9 End
```

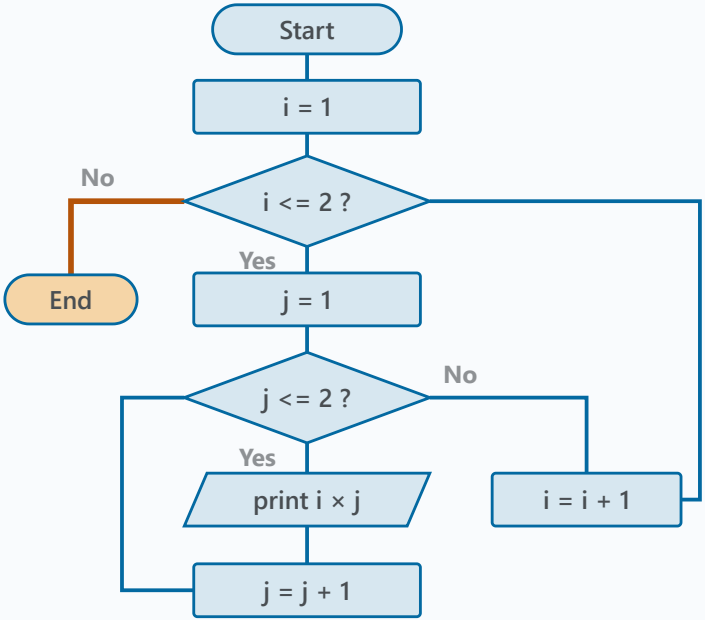
VARIABLES

`i = 3`

OUTPUT

```
1 2
2 4
```

Nested Loop Step by Step (cont.: i = 2)



End. The inner body ran 2 × 2 = **4** times; the output is a small multiplication table.

Step 12 / 12

```
1 Start
2   i = 1
3   i <= 2 ?  (No → End)
4     j = 1
5     j <= 2 ?  (No → i = i + 1)
6       print i × j
7       j = j + 1, back to inner decision
8     end line (println), i = i + 1, back
  to outer decision
▶ End
```

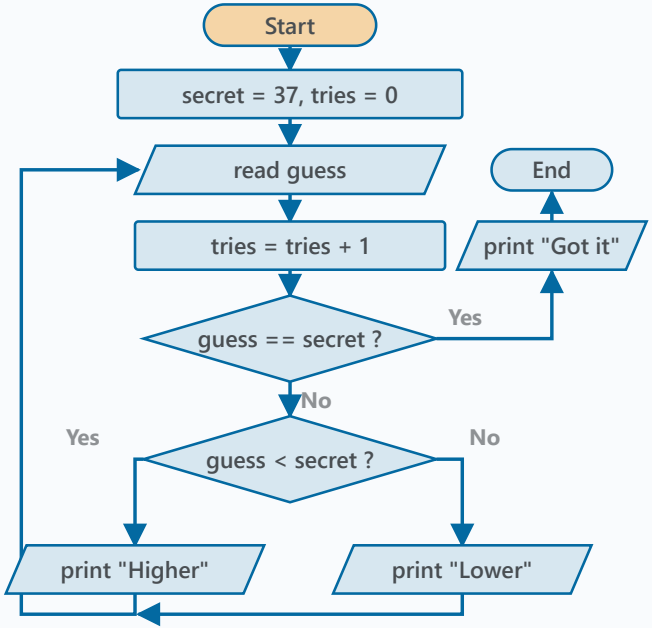
VARIABLES

`i = 3` `j = 3`

OUTPUT

```
1 2
2 4
```

Number Guessing Game Step by Step (secret = 37)



Start. The secret number (normally random) is 37; the player guesses 50, 25, 37.

Step 1 / 17

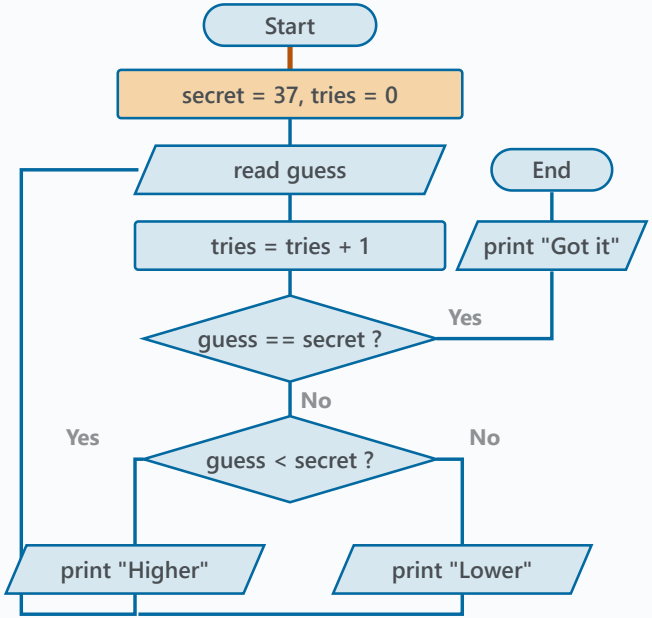
```
▶ Start
2   secret = 37, tries = 0
3   read guess
4   tries = tries + 1
5   guess == secret ?
6     Yes → print "Got it (try " tries ")",
  End
7   guess < secret ?
8     Yes → print "Higher"
9     No → print "Lower"
10  back to line 3 (read)
11 End
```

VARIABLES

—

OUTPUT

Number Guessing Game Step by Step (secret = 37)



The secret is set and the try counter is 0.

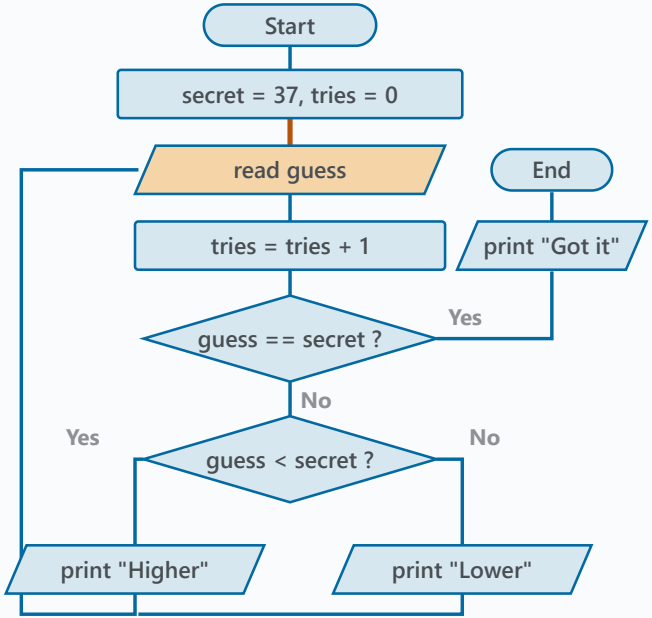
```
1  Start
▶    secret = 37, tries = 0
3    read guess
4    tries = tries + 1
5    guess == secret ?
6      Yes → print "Got it (try " tries ")",
   End
7    guess < secret ?
8      Yes → print "Higher"
9      No → print "Lower"
10   back to line 3 (read)
11 End
```

VARIABLES

`secret = 37` `tries = 0`

OUTPUT

Number Guessing Game Step by Step (secret = 37)



The player enters **50**.

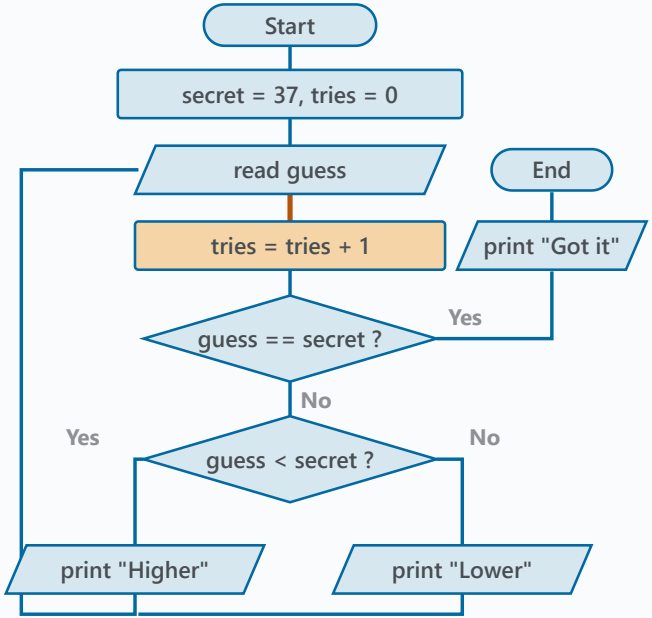
```
1  Start
2    secret = 37, tries = 0
▶    read guess
4    tries = tries + 1
5    guess == secret ?
6      Yes → print "Got it (try " tries ")",
   End
7    guess < secret ?
8      Yes → print "Higher"
9      No → print "Lower"
10   back to line 3 (read)
11 End
```

VARIABLES

`secret = 37` `tries = 0` `guess = 50`

OUTPUT

Number Guessing Game Step by Step (secret = 37)



tries = 1.

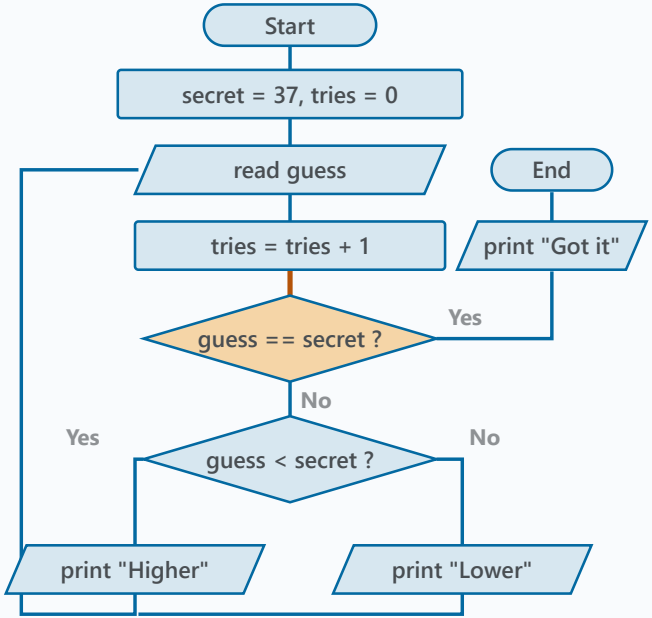
```
1  Start
2    secret = 37, tries = 0
3    read guess
▶    tries = tries + 1
5    guess == secret ?
6      Yes → print "Got it (try " tries ")",
   End
7    guess < secret ?
8      Yes → print "Higher"
9      No → print "Lower"
10   back to line 3 (read)
11 End
```

VARIABLES

`secret = 37` `tries = 1` `guess = 50`

OUTPUT

Number Guessing Game Step by Step (secret = 37)



Decision: **50 == 37** → **false** → No, give a hint.

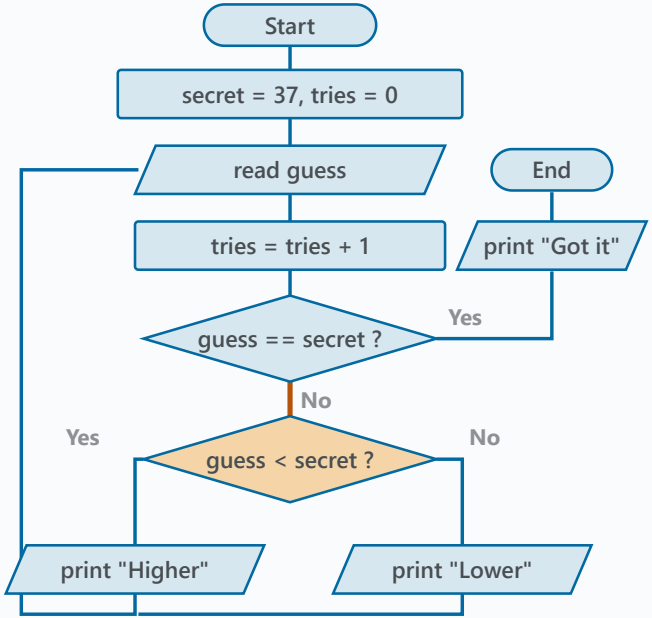
```
1  Start
2    secret = 37, tries = 0
3    read guess
4    tries = tries + 1
▶    guess == secret ?
6      Yes → print "Got it (try " tries ")",
   End
7    guess < secret ?
8      Yes → print "Higher"
9      No → print "Lower"
10   back to line 3 (read)
11 End
```

VARIABLES

`secret = 37` `tries = 1` `guess = 50`

OUTPUT

Number Guessing Game Step by Step (secret = 37)



Decision: **50 < 37 → false**.

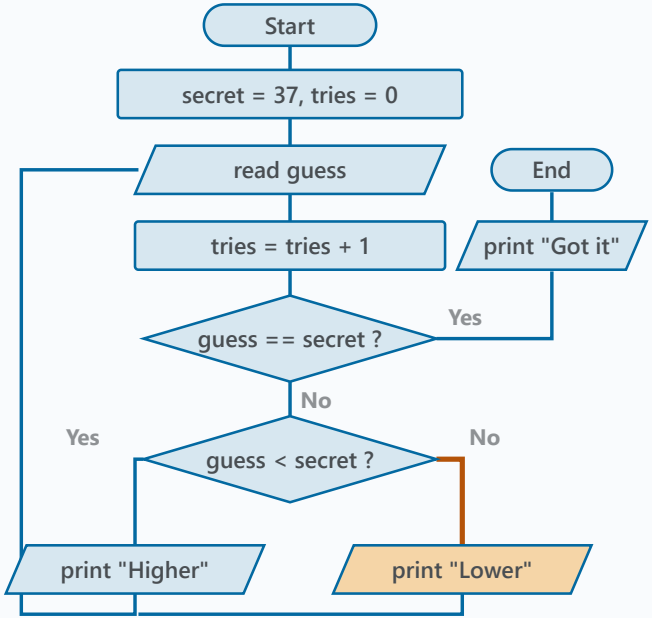
```
1  Start
2    secret = 37, tries = 0
3    read guess
4    tries = tries + 1
5    guess == secret ?
6      Yes → print "Got it (try " tries ")",
   End
▶    guess < secret ?
8      Yes → print "Higher"
9      No → print "Lower"
10   back to line 3 (read)
11 End
```

VARIABLES

`secret = 37` `tries = 1` `guess = 50`

OUTPUT

Number Guessing Game Step by Step (secret = 37)



"Lower" is printed; the arrow returns to the **read** box.

```
1  Start
2    secret = 37, tries = 0
3    read guess
4    tries = tries + 1
5    guess == secret ?
6      Yes → print "Got it (try " tries ")",
   End
7    guess < secret ?
8      Yes → print "Higher"
▶      No → print "Lower"
10   back to line 3 (read)
11 End
```

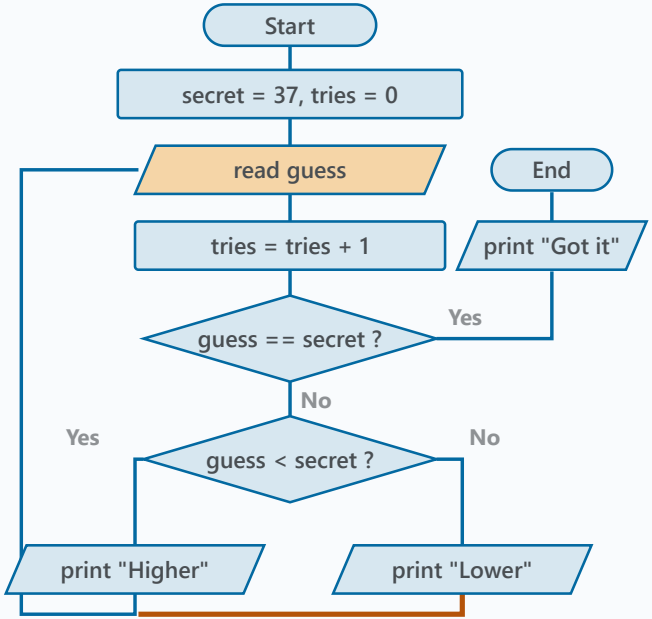
VARIABLES

`secret = 37` `tries = 1` `guess = 50`

OUTPUT

```
Lower
```

Number Guessing Game Step by Step (secret = 37)



The player enters **25**.

```
1  Start
2    secret = 37, tries = 0
▶    read guess
4    tries = tries + 1
5    guess == secret ?
6      Yes → print "Got it (try " tries ")",
   End
7    guess < secret ?
8      Yes → print "Higher"
9      No → print "Lower"
10   back to line 3 (read)
11 End
```

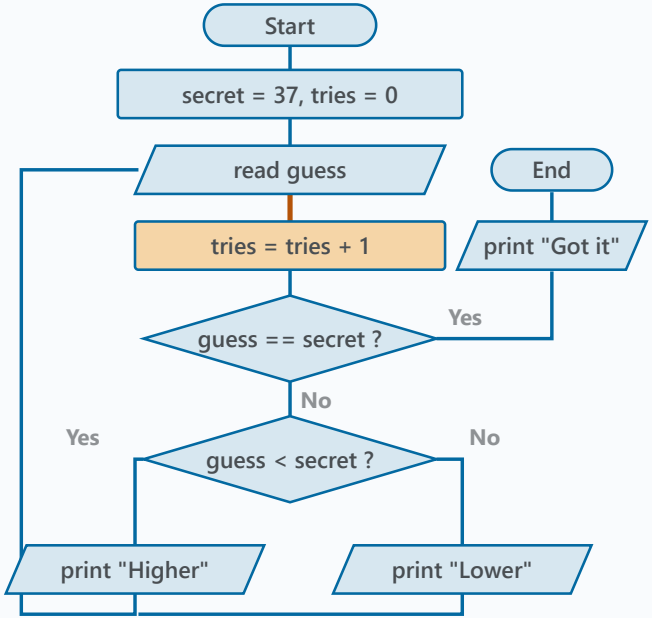
VARIABLES

`secret = 37` `tries = 1` `guess = 25`

OUTPUT

```
Lower
```

Number Guessing Game Step by Step (secret = 37)



tries = 2.

```
1  Start
2    secret = 37, tries = 0
3    read guess
4    tries = tries + 1
5    guess == secret ?
6      Yes → print "Got it (try " tries ")",
   End
7    guess < secret ?
8      Yes → print "Higher"
9      No → print "Lower"
10   back to line 3 (read)
11 End
```

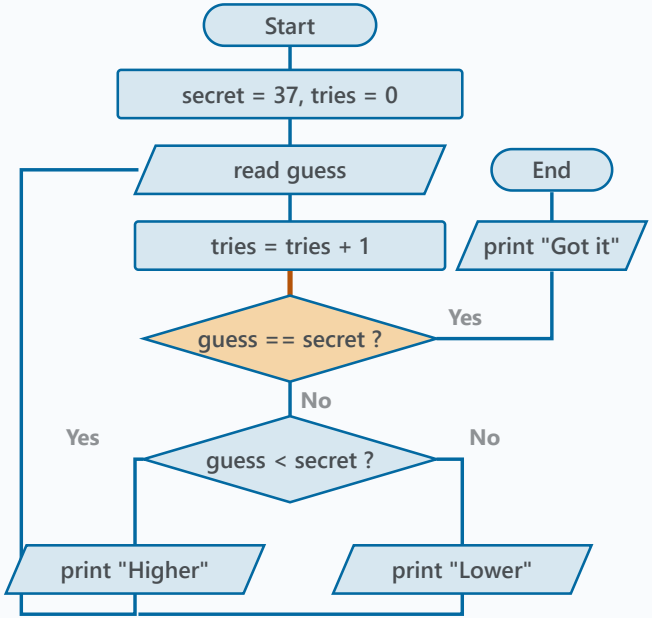
VARIABLES

`secret = 37` `tries = 2` `guess = 25`

OUTPUT

```
Lower
```

Number Guessing Game Step by Step (secret = 37)



Decision: **25 == 37** → **false** → No, give a hint.

```
1  Start
2    secret = 37, tries = 0
3    read guess
4    tries = tries + 1
▶    guess == secret ?
6      Yes → print "Got it (try " tries ")",
   End
7    guess < secret ?
8      Yes → print "Higher"
9      No → print "Lower"
10   back to line 3 (read)
11 End
```

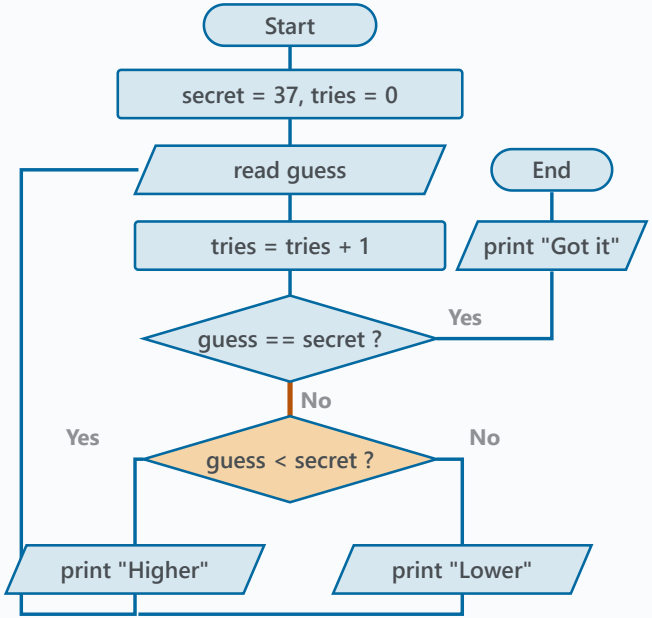
VARIABLES

`secret = 37` `tries = 2` `guess = 25`

OUTPUT

```
Lower
```

Number Guessing Game Step by Step (secret = 37)



Decision: **25 < 37 → true**.

```
1  Start
2    secret = 37, tries = 0
3    read guess
4    tries = tries + 1
5    guess == secret ?
6      Yes → print "Got it (try " tries ")",
   End
▶    guess < secret ?
8      Yes → print "Higher"
9      No → print "Lower"
10   back to line 3 (read)
11 End
```

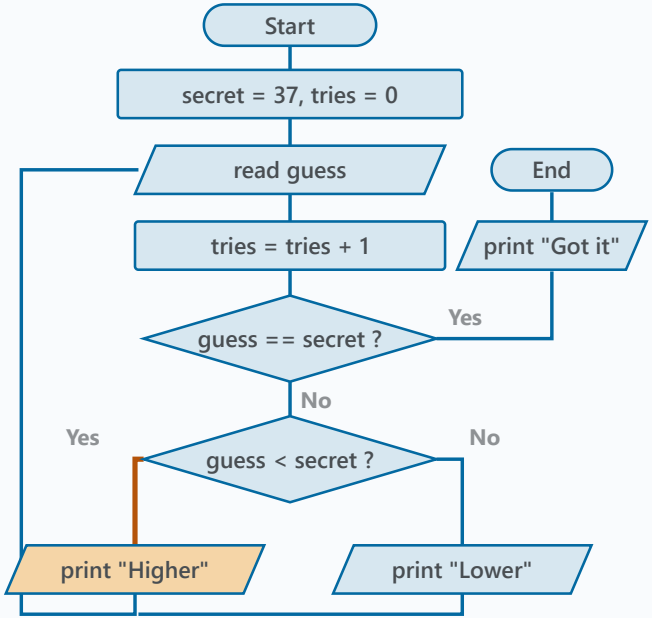
VARIABLES

`secret = 37` `tries = 2` `guess = 25`

OUTPUT

```
Lower
```

Number Guessing Game Step by Step (secret = 37)



"Higher" is printed; the arrow returns to the **read** box.

```
1  Start
2    secret = 37, tries = 0
3    read guess
4    tries = tries + 1
5    guess == secret ?
6      Yes → print "Got it (try " tries ")",
   End
7    guess < secret ?
▶      Yes → print "Higher"
9      No → print "Lower"
10   back to line 3 (read)
11 End
```

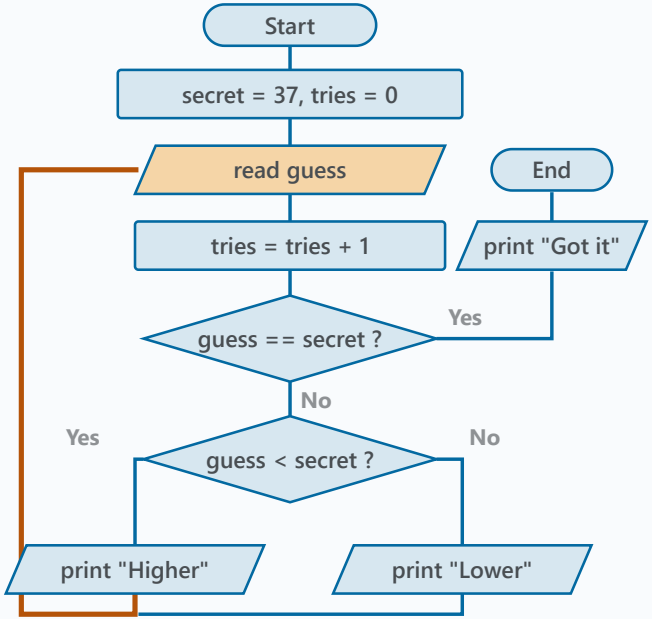
VARIABLES

`secret = 37` `tries = 2` `guess = 25`

OUTPUT

```
Lower
Higher
```

Number Guessing Game Step by Step (secret = 37)



The player enters **37**.

```
1  Start
2    secret = 37, tries = 0
▶    read guess
4    tries = tries + 1
5    guess == secret ?
6      Yes → print "Got it (try " tries ")",
   End
7    guess < secret ?
8      Yes → print "Higher"
9      No → print "Lower"
10   back to line 3 (read)
11 End
```

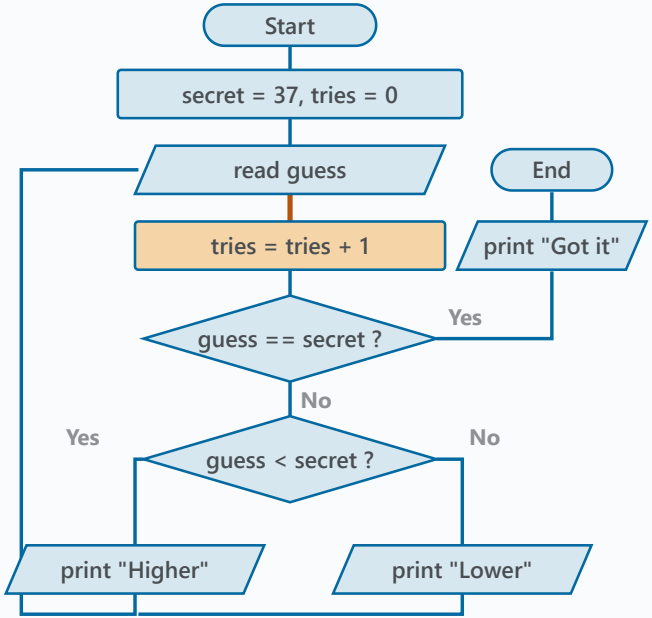
VARIABLES

`secret = 37` `tries = 2` `guess = 37`

OUTPUT

```
Lower
Higher
```

Number Guessing Game Step by Step (secret = 37)



tries = 3.

```
1  Start
2    secret = 37, tries = 0
3    read guess
▶    tries = tries + 1
5    guess == secret ?
6      Yes → print "Got it (try " tries ")",
   End
7    guess < secret ?
8      Yes → print "Higher"
9      No → print "Lower"
10   back to line 3 (read)
11 End
```

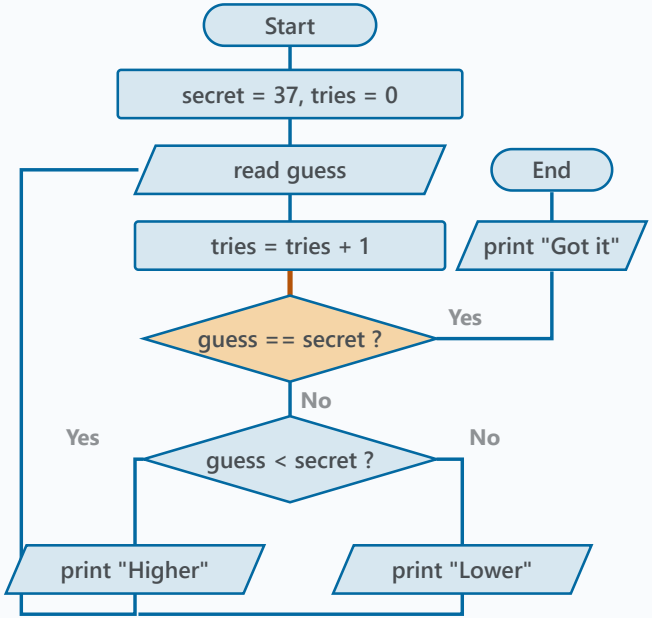
VARIABLES

`secret = 37` `tries = 3` `guess = 37`

OUTPUT

```
Lower
Higher
```

Number Guessing Game Step by Step (secret = 37)



Decision: **37 == 37** → **true** → Yes.

```
1  Start
2    secret = 37, tries = 0
3    read guess
4    tries = tries + 1
▶    guess == secret ?
6      Yes → print "Got it (try " tries ")",
   End
7    guess < secret ?
8      Yes → print "Higher"
9      No → print "Lower"
10   back to line 3 (read)
11 End
```

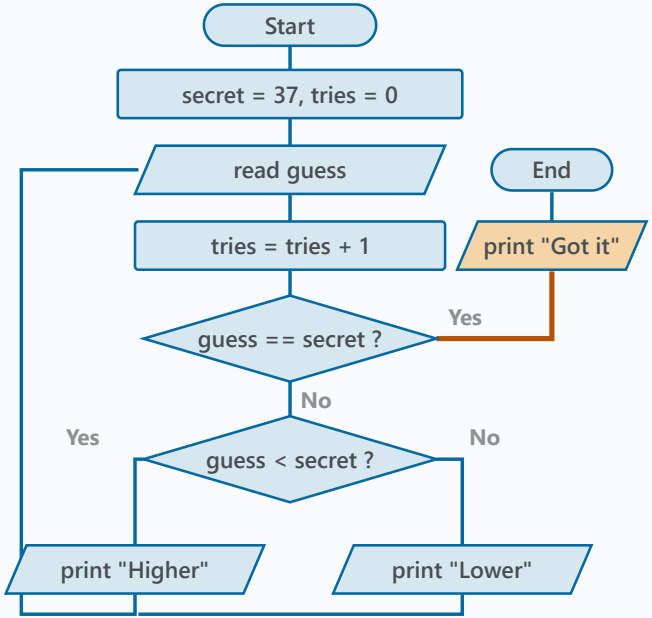
VARIABLES

`secret = 37` `tries = 3` `guess = 37`

OUTPUT

```
Lower
Higher
```

Number Guessing Game Step by Step (secret = 37)



"Correct!" is printed with the try count, **3**.

```
1  Start
2    secret = 37, tries = 0
3    read guess
4    tries = tries + 1
5    guess == secret ?
▶      Yes → print "Got it (try " tries ")",
   End
7    guess < secret ?
8      Yes → print "Higher"
9      No → print "Lower"
10   back to line 3 (read)
11 End
```

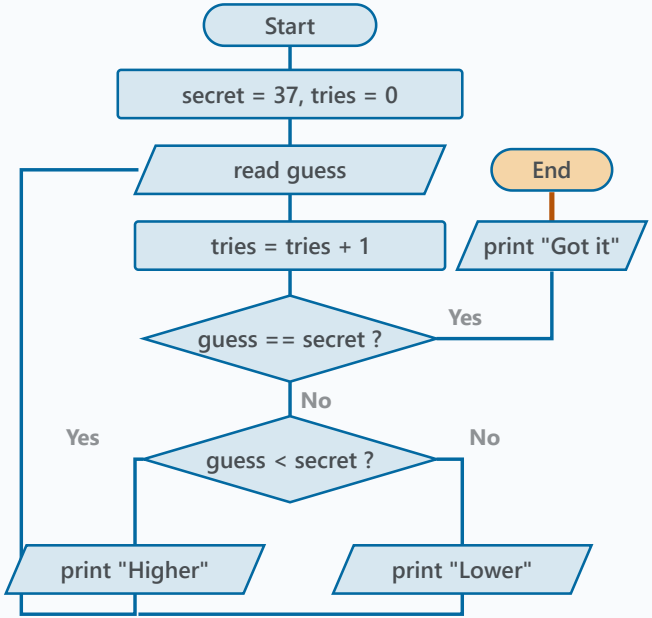
VARIABLES

`secret = 37` `tries = 3` `guess = 37`

OUTPUT

```
Lower
Higher
Got it (try 3)
```

Number Guessing Game Step by Step (secret = 37)



End. The loop ended when its condition (guess == secret) became true; the number of passes is not known in advance.


```
1  Start
2    secret = 37, tries = 0
3    read guess
4    tries = tries + 1
5    guess == secret ?
6      Yes → print "Got it (try " tries ")",
   End
7    guess < secret ?
8      Yes → print "Higher"
9      No → print "Lower"
10   back to line 3 (read)
   ▶ End
```

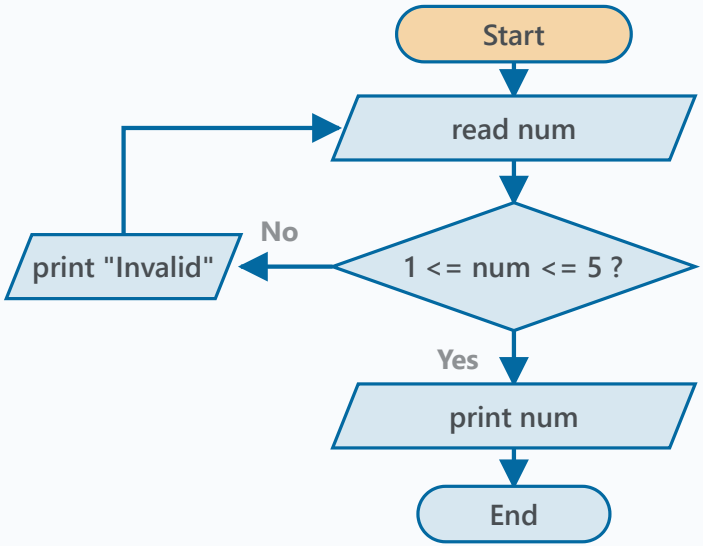
VARIABLES

`secret = 37` `tries = 3` `guess = 37`

OUTPUT

```
Lower
Higher
Got it (try 3)
```

do-while Flowchart Step by Step (inputs: 7, 0, 3)




```
▶ Start
2   read num
3   1 <= num <= 5 ?
4     No → print "Invalid", back to read
5     Yes → print num
6 End
```

VARIABLES

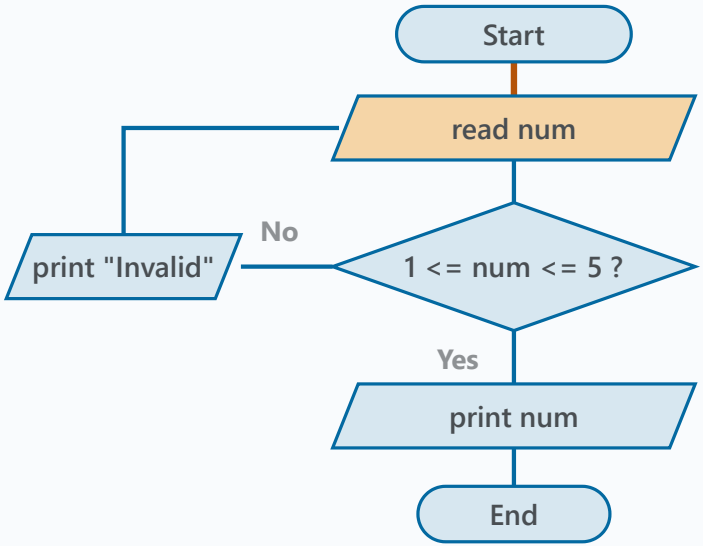
—

OUTPUT

Start. The inputs arrive in order: 7, 0, 3. The **body** (read) runs first and the condition is checked **at the end**: a post-test loop.

Step 1 / 11

do-while Flowchart Step by Step (inputs: 7, 0, 3)




```
1 Start
▶   read num
3   1 <= num <= 5 ?
4     No → print "Invalid", back to read
5     Yes → print num
6 End
```

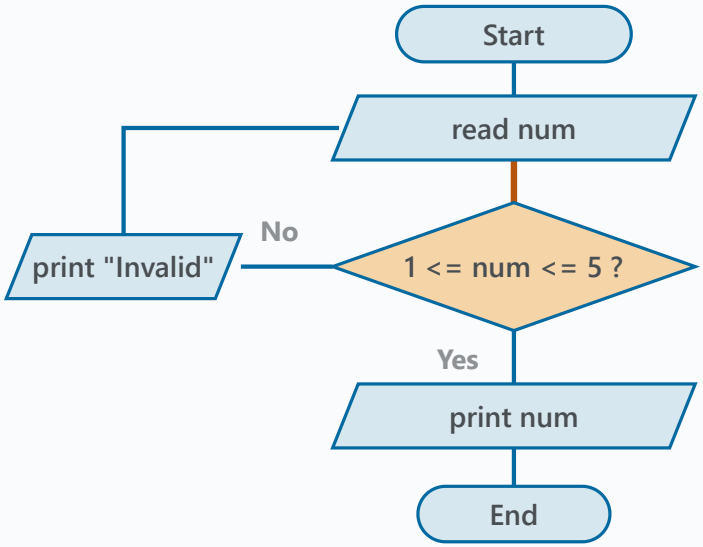
VARIABLES

`num = 7`

OUTPUT

Body first: read **num = 7**. The condition has not been checked yet.

do-while Flowchart Step by Step (inputs: 7, 0, 3)




```
1 Start
2   read num
▶   1 <= num <= 5 ?
4     No → print "Invalid", back to read
5     Yes → print num
6 End
```

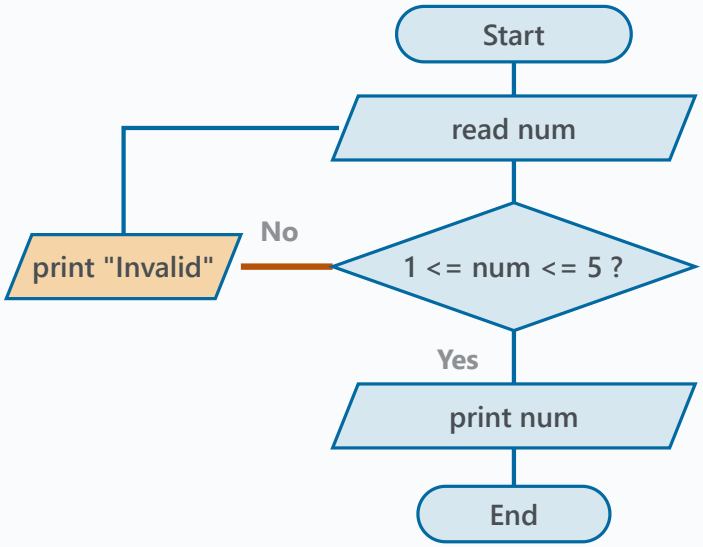
VARIABLES

`num = 7`

OUTPUT

Decision: 1 ≤ 7 ≤ 5 ? Since 7 > 5, **false** → No.

do-while Flowchart Step by Step (inputs: 7, 0, 3)



"Invalid" is printed.

```
1 Start
2   read num
3   1 <= num <= 5 ?
      No → print "Invalid", back to read
5     Yes → print num
6 End
```

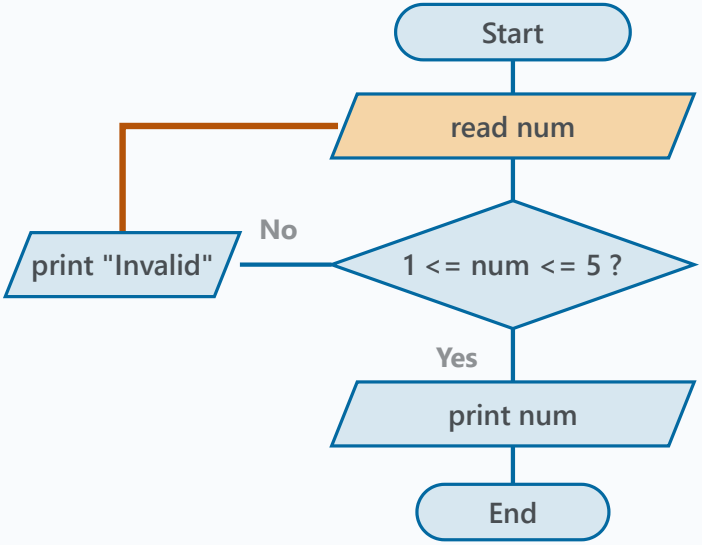
VARIABLES

`num = 7`

OUTPUT

```
Invalid
```

do-while Flowchart Step by Step (inputs: 7, 0, 3)



The arrow returns to the **read** box: read **num = 0**.

```
1 Start
▶   read num
3   1 <= num <= 5 ?
4     No → print "Invalid", back to read
5     Yes → print num
6 End
```

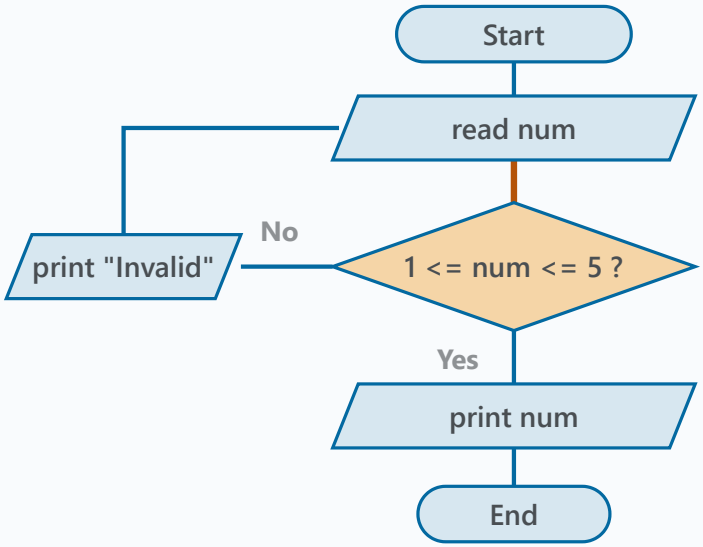
VARIABLES

`num = 0`

OUTPUT

```
Invalid
```

do-while Flowchart Step by Step (inputs: 7, 0, 3)




```
1 Start
2   read num
▶   1 <= num <= 5 ?
4     No → print "Invalid", back to read
5     Yes → print num
6 End
```

VARIABLES

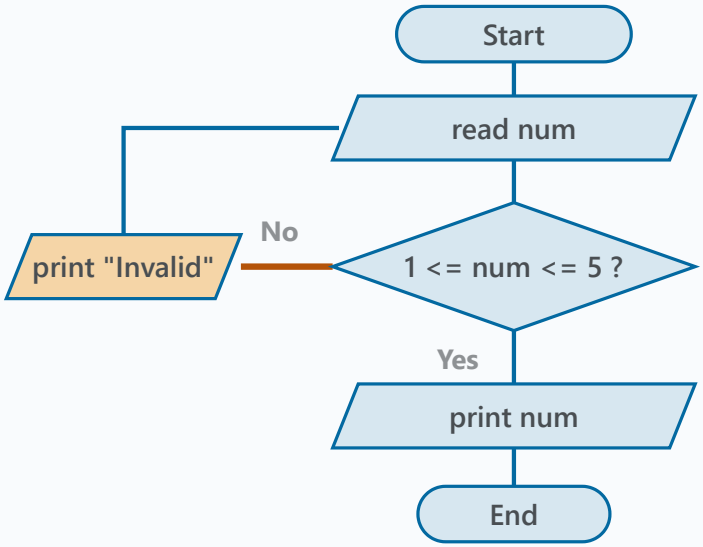
`num = 0`

OUTPUT

```
Invalid
```

Decision: 1 ≤ 0 ≤ 5 ? Since 0 < 1, **false** → No.

do-while Flowchart Step by Step (inputs: 7, 0, 3)



"Invalid" is printed again.

```
1 Start
2   read num
3   1 <= num <= 5 ?
      No → print "Invalid", back to read
5     Yes → print num
6 End
```

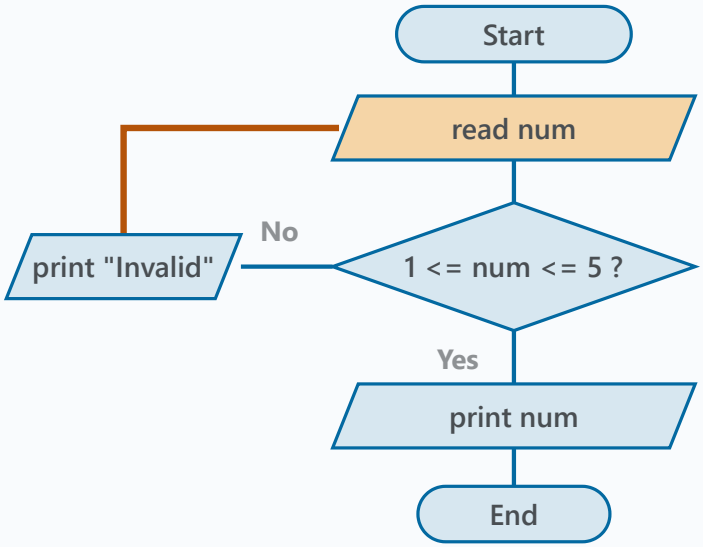
VARIABLES

`num = 0`

OUTPUT

```
Invalid
Invalid
```

do-while Flowchart Step by Step (inputs: 7, 0, 3)



Read again: **num = 3**.

```
1 Start
▶   read num
3   1 <= num <= 5 ?
4     No → print "Invalid", back to read
5     Yes → print num
6 End
```

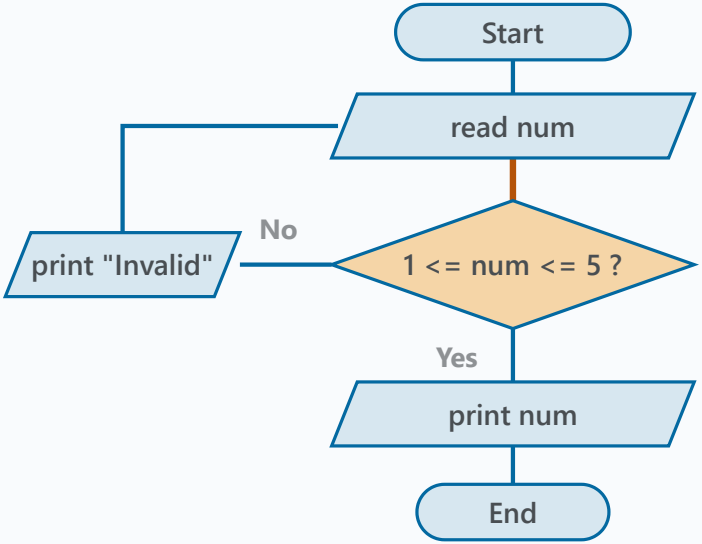
VARIABLES

`num = 3`

OUTPUT

```
Invalid
Invalid
```

do-while Flowchart Step by Step (inputs: 7, 0, 3)



Decision: **1 ≤ 3 ≤ 5** → **true** → Yes.

```
1 Start
2   read num
▶   1 <= num <= 5 ?
4     No → print "Invalid", back to read
5     Yes → print num
6 End
```

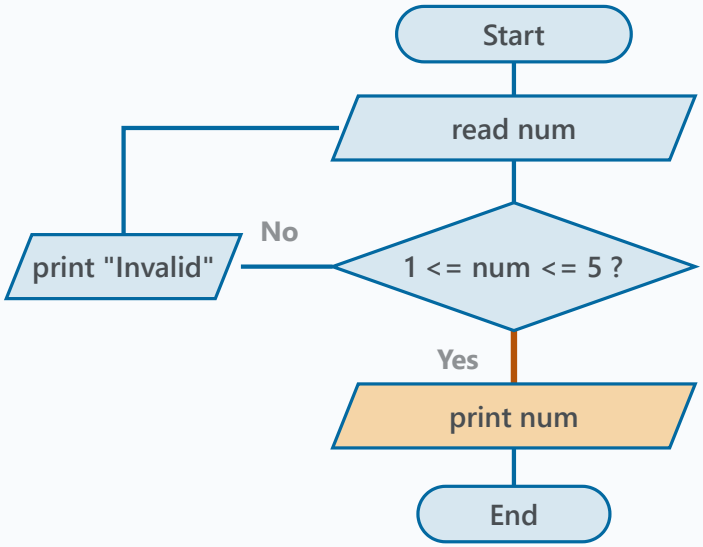
VARIABLES

`num = 3`

OUTPUT

```
Invalid
Invalid
```

do-while Flowchart Step by Step (inputs: 7, 0, 3)




```
1 Start
2   read num
3   1 <= num <= 5 ?
4     No → print "Invalid", back to read
      Yes → print num
6 End
```

VARIABLES

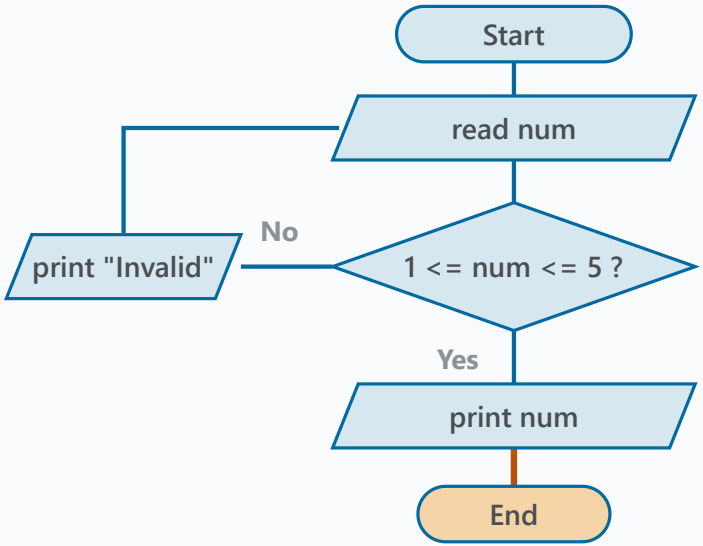
`num = 3`

OUTPUT

```
Invalid
Invalid
Valid: 3
```

The valid number is printed: **3**.

do-while Flowchart Step by Step (inputs: 7, 0, 3)



End. The body ran 3 times. Because the condition is checked **after** the body, the body runs at least once. In Java: `do { … } while (num < 1 || num > 5);`

```
1 Start
2   read num
3   1 <= num <= 5 ?
4     No → print "Invalid", back to read
5     Yes → print num
▶ End
```

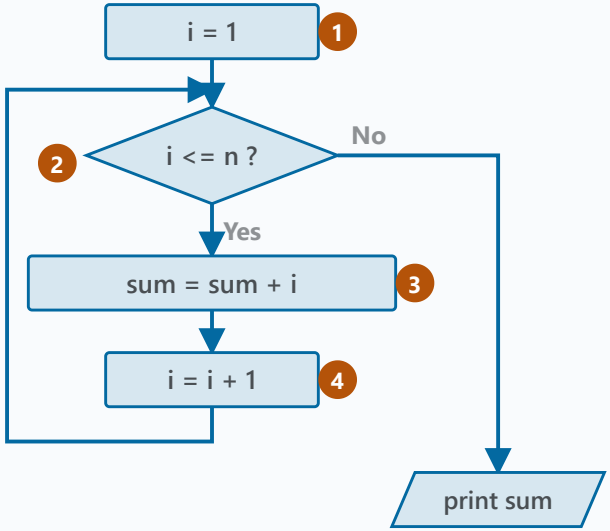
VARIABLES

`num = 3`

OUTPUT

```
Invalid
Invalid
Valid: 3
```

Flowchart of a `for` Loop

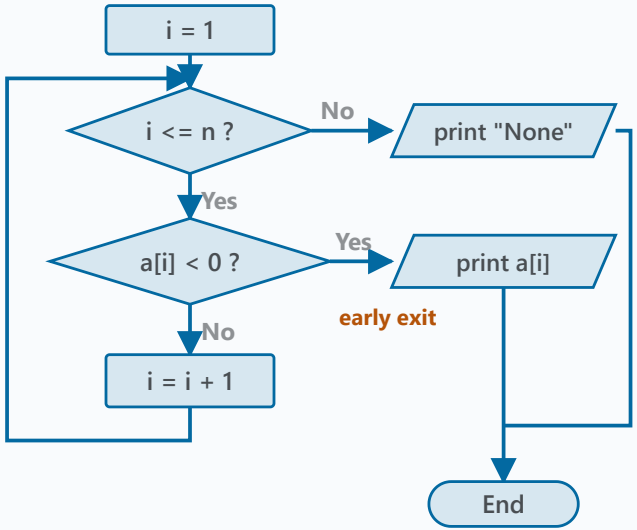



```
for (i = 1; i <= n; i++) {
    sum = sum + i;
}
```

1. **Initialization** (i = 1): runs **once**, on entering the loop.
2. **Condition** (i <= n): checked **before** every pass; for is a pre-test loop.
3. **Body**: runs if the condition is true.
4. **Update** (i++): runs **after** the body; the arrow returns to the condition.

The chart is the same as a while chart; for just gathers parts 1, 2 and 4 into one line. Some texts show 1, 2 and 4 in a single **preparation** symbol (a hexagon, “i = 1 → n”).

Early Exit and Flags

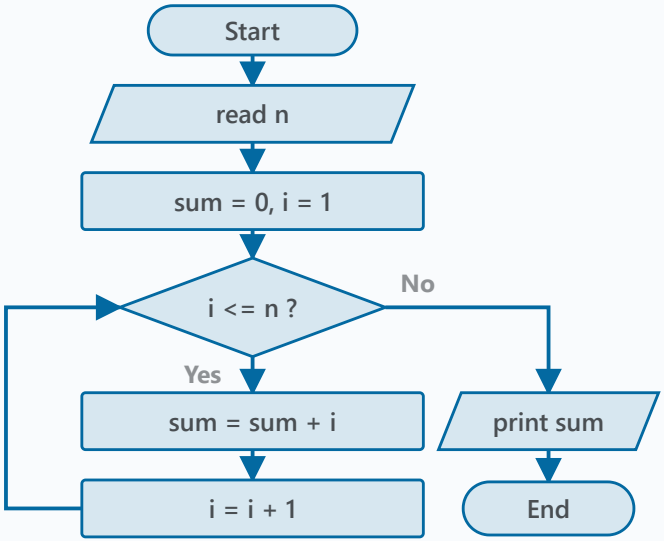


Find the first negative number: a = [4, 7, −2, 5, −8]

- Once the target is found, the remaining passes are wasted. A second arrow from inside the body to outside the loop is an **early exit**; in Java, `break`.
- In the example, −2 is found at i = 3 and printed; 5 and −8 are never examined. If there is no negative number, the loop exits via its condition and prints “None”.
- The loop now has two exits. If you want a single exit, use a **flag**: start with found = false, set it to true on a match, and make the condition **i <= n AND not found ?**. After the loop, check the flag to print the result.
- **Skip** (`continue`) does not leave the loop: it skips the rest of the pass and goes to the update and the condition.

TRANSLATING TO CODE

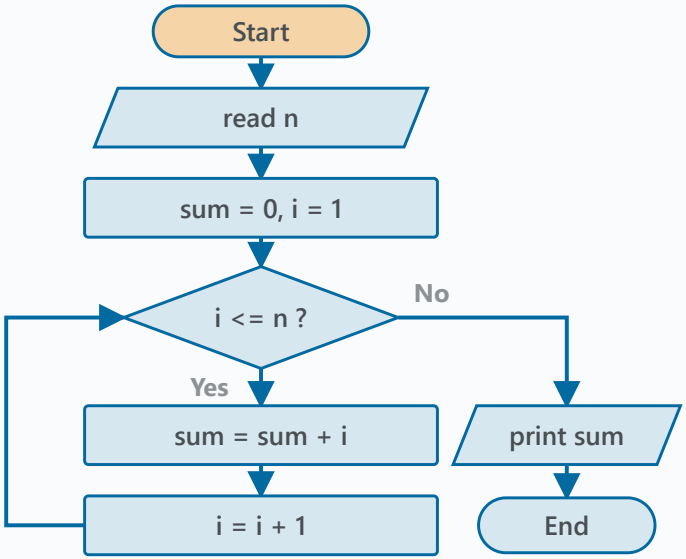
From Flowchart to Java Code




```java
public static void main(String[] args) {
    int n = 3;
    int sum = 0, i = 1;
    while (i <= n) {
        sum = sum + i;
        i = i + 1;
    }
    System.out.println(sum);
}
```

- **Start / End** → the start and end of the method.
- **Input** → a variable (here a fixed value).
- **Process** → an assignment statement.
- **Decision + back arrow** → the `while` condition and body.
- **Output** → `println`.

From Flowchart to Java Code Step by Step (n = 3)



The **Start** symbol is the start of the method: `main` opens.

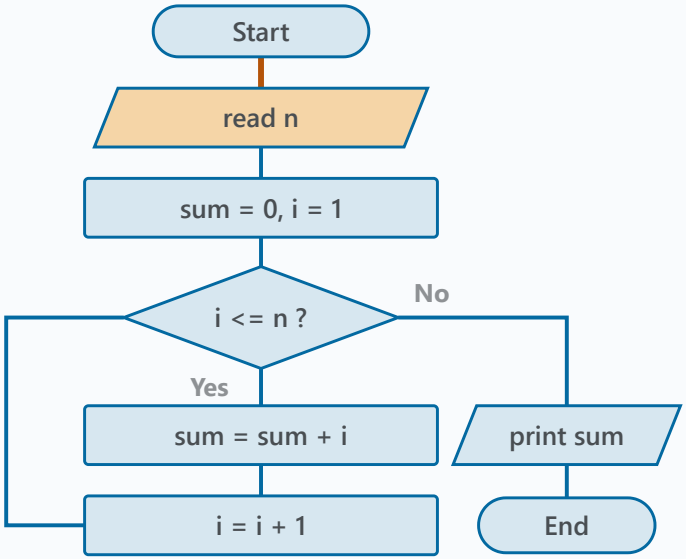
```
▶ public static void main(String[] args) {
2     int n = 3;
3     int sum = 0, i = 1;
4     while (i <= n) {
5         sum = sum + i;
6         i = i + 1;
7     }
8     System.out.println(sum);
9 }
```

VARIABLES

—

OUTPUT

From Flowchart to Java Code Step by Step (n = 3)



Input symbol → `int n = 3;` (the input is fixed).

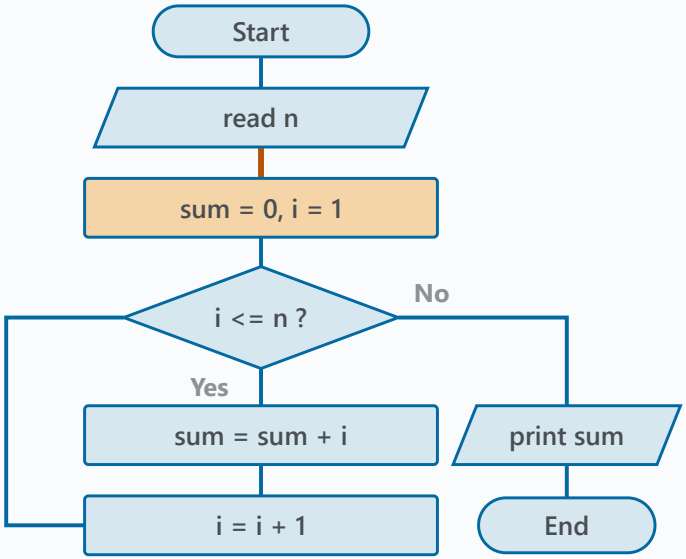
```
1 public static void main(String[] args) {
▶     int n = 3;
3     int sum = 0, i = 1;
4     while (i <= n) {
5         sum = sum + i;
6         i = i + 1;
7     }
8     System.out.println(sum);
9 }
```

VARIABLES

`n = 3`

OUTPUT

From Flowchart to Java Code Step by Step (n = 3)



Process symbol → the initial values of two variables.

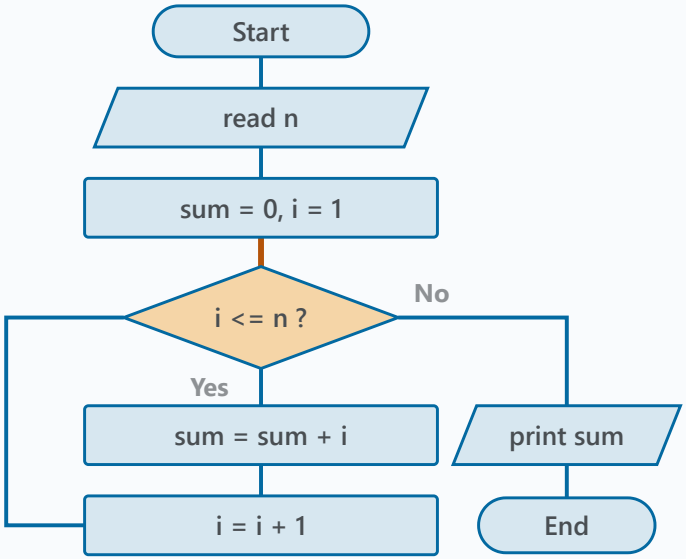
```
1 public static void main(String[] args) {
2     int n = 3;
▶     int sum = 0, i = 1;
4     while (i <= n) {
5         sum = sum + i;
6         i = i + 1;
7     }
8     System.out.println(sum);
9 }
```

VARIABLES

`n = 3` `sum = 0` `i = 1`

OUTPUT

From Flowchart to Java Code Step by Step (n = 3)



Decision symbol → `while (i <= n)` : 1 <= 3 is true.

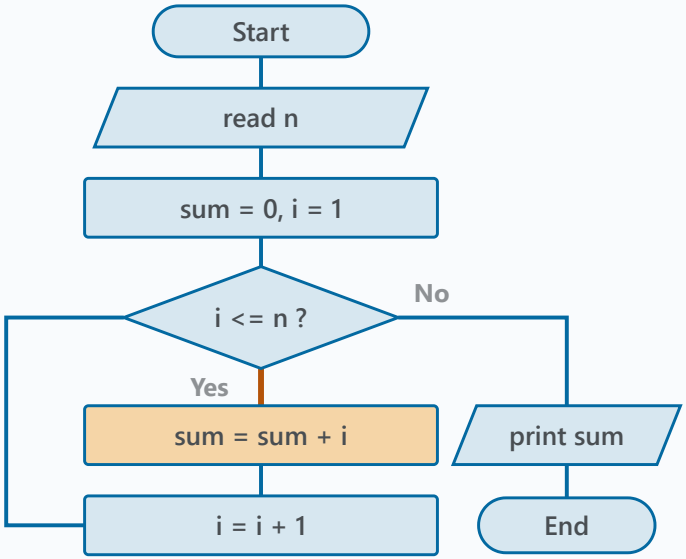
```
1 public static void main(String[] args) {
2     int n = 3;
3     int sum = 0, i = 1;
▶     while (i <= n) {
5         sum = sum + i;
6         i = i + 1;
7     }
8     System.out.println(sum);
9 }
```

VARIABLES

`n = 3` `sum = 0` `i = 1`

OUTPUT

From Flowchart to Java Code Step by Step (n = 3)



The **Yes** path is the loop body: `sum = 0 + 1` = 1.

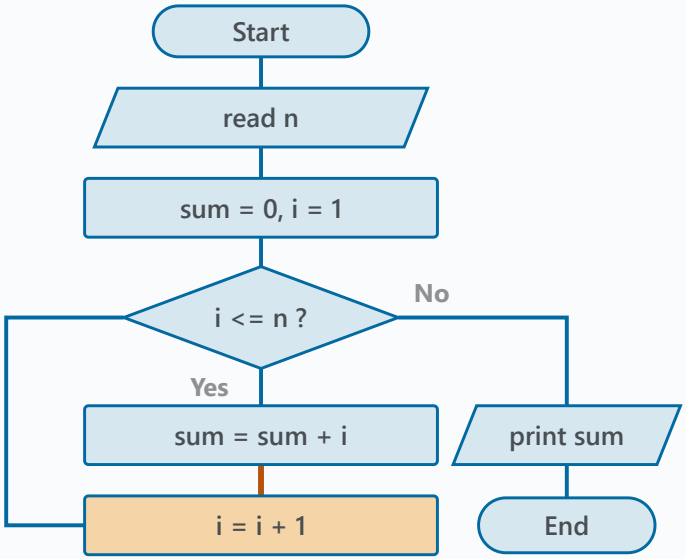
```
1 public static void main(String[] args) {
2     int n = 3;
3     int sum = 0, i = 1;
4     while (i <= n) {
▶         sum = sum + i;
6         i = i + 1;
7     }
8     System.out.println(sum);
9 }
```

VARIABLES

`n = 3` `sum = 1` `i = 1`

OUTPUT

From Flowchart to Java Code Step by Step (n = 3)




```
1 public static void main(String[] args) {
2     int n = 3;
3     int sum = 0, i = 1;
4     while (i <= n) {
5         sum = sum + i;
▶         i = i + 1;
7     }
8     System.out.println(sum);
9 }
```

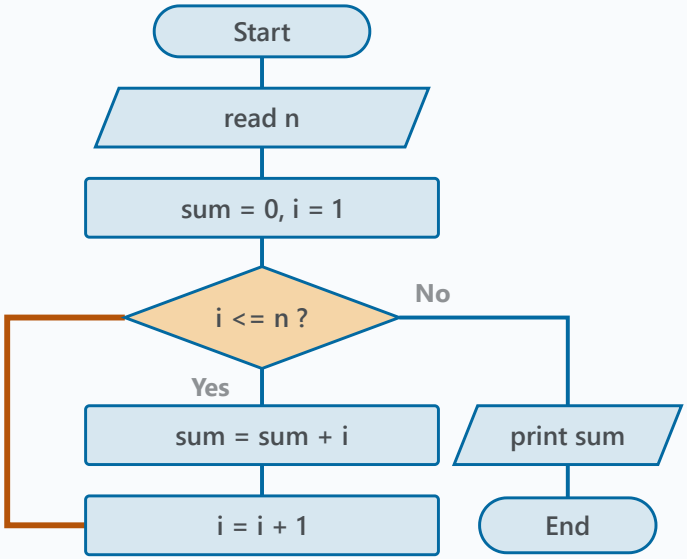
VARIABLES

`n = 3` `sum = 1` `i = 2`

OUTPUT

`i = i + 1` → i = 2. The next arrow goes **back** to the decision.

From Flowchart to Java Code Step by Step (n = 3)



The back arrow is the jump from the end of the loop block to the condition: 2 <= 3 is true.

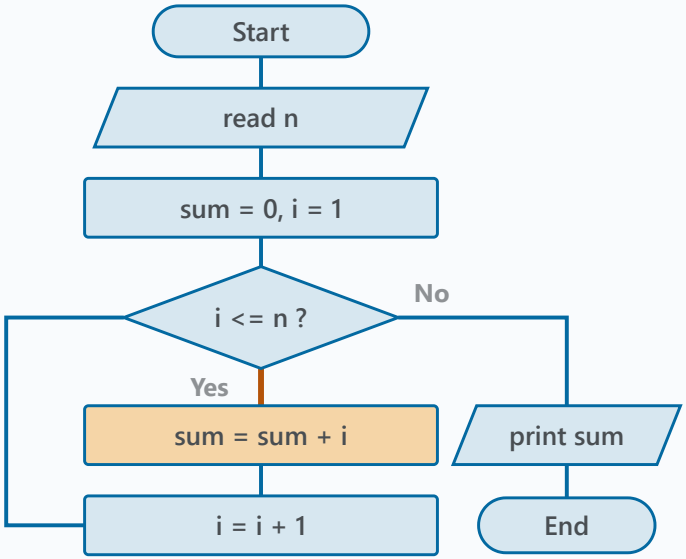
```
1 public static void main(String[] args) {
2     int n = 3;
3     int sum = 0, i = 1;
▶     while (i <= n) {
5         sum = sum + i;
6         i = i + 1;
7     }
8     System.out.println(sum);
9 }
```

VARIABLES

`n = 3` `sum = 1` `i = 2`

OUTPUT

From Flowchart to Java Code Step by Step (n = 3)



`sum = 1 + 2` = 3.

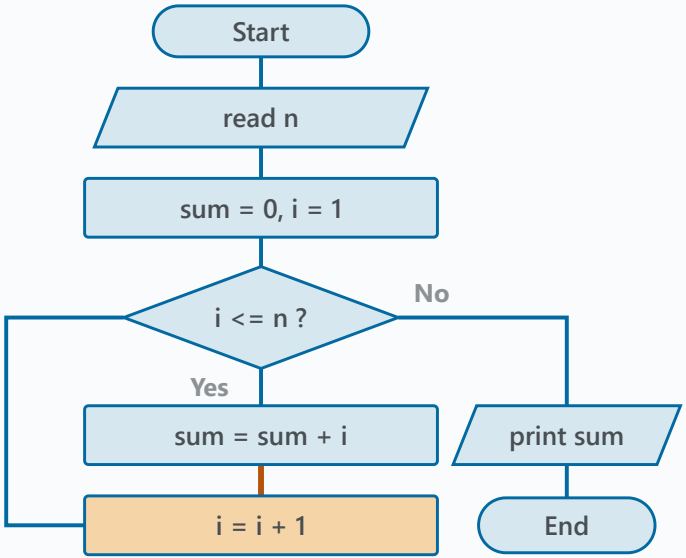
```
1 public static void main(String[] args) {
2     int n = 3;
3     int sum = 0, i = 1;
4     while (i <= n) {
▶         sum = sum + i;
6         i = i + 1;
7     }
8     System.out.println(sum);
9 }
```

VARIABLES

`n = 3` `sum = 3` `i = 2`

OUTPUT

From Flowchart to Java Code Step by Step (n = 3)



`i = 3` .

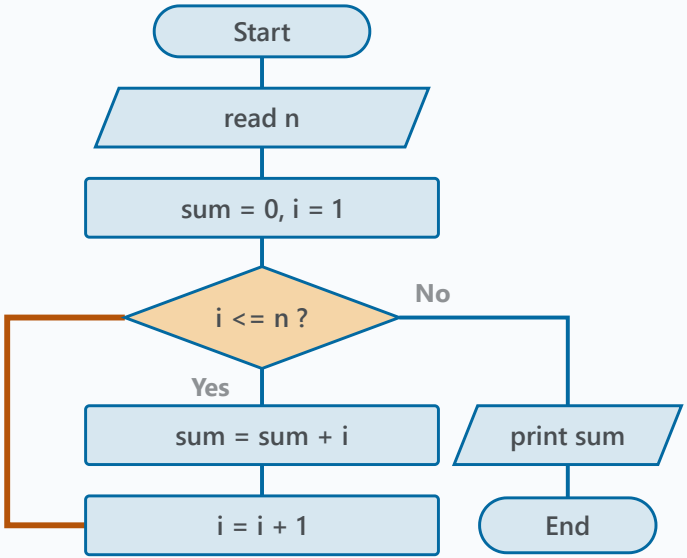
```
1 public static void main(String[] args) {
2     int n = 3;
3     int sum = 0, i = 1;
4     while (i <= n) {
5         sum = sum + i;
▶         i = i + 1;
7     }
8     System.out.println(sum);
9 }
```

VARIABLES

`n = 3` `sum = 3` `i = 3`

OUTPUT

From Flowchart to Java Code Step by Step (n = 3)



3 <= 3 is true.

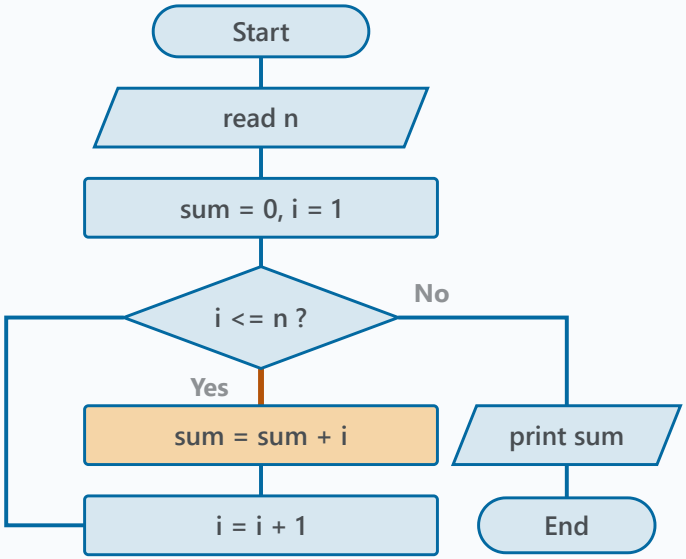
```
1 public static void main(String[] args) {
2     int n = 3;
3     int sum = 0, i = 1;
▶     while (i <= n) {
5         sum = sum + i;
6         i = i + 1;
7     }
8     System.out.println(sum);
9 }
```

VARIABLES

`n = 3` `sum = 3` `i = 3`

OUTPUT

From Flowchart to Java Code Step by Step (n = 3)



`sum = 3 + 3` = 6.

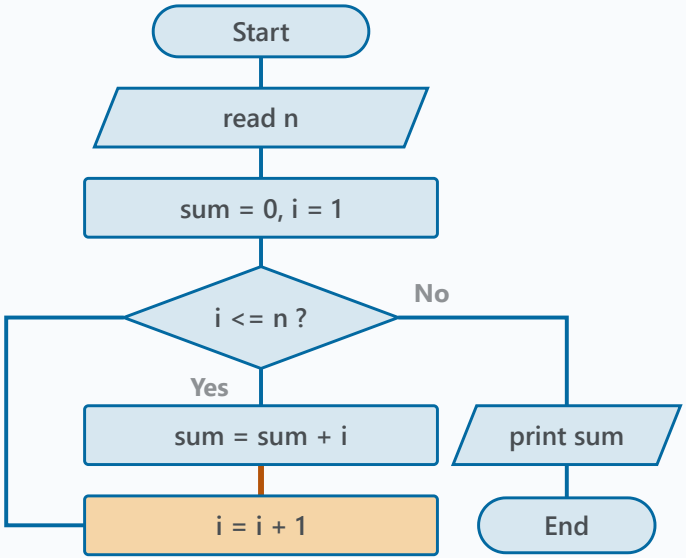
```
1 public static void main(String[] args) {
2     int n = 3;
3     int sum = 0, i = 1;
4     while (i <= n) {
▶         sum = sum + i;
6         i = i + 1;
7     }
8     System.out.println(sum);
9 }
```

VARIABLES

`n = 3` `sum = 6` `i = 3`

OUTPUT

From Flowchart to Java Code Step by Step (n = 3)



`i = 4` .

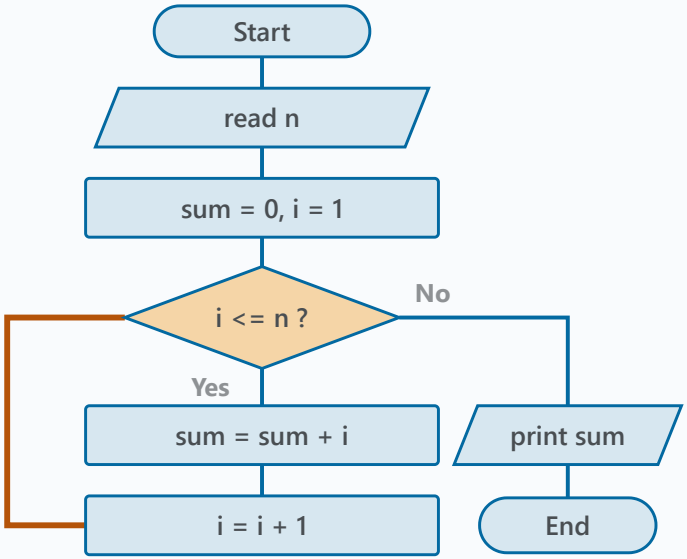
```java
public static void main(String[] args) {
    int n = 3;
    int sum = 0, i = 1;
    while (i <= n) {
        sum = sum + i;
        i = i + 1;
    }
    System.out.println(sum);
}
```

VARIABLES

`n = 3` `sum = 6` `i = 4`

OUTPUT

From Flowchart to Java Code Step by Step (n = 3)



4 <= 3 is **false** → the No path leaves the loop block.

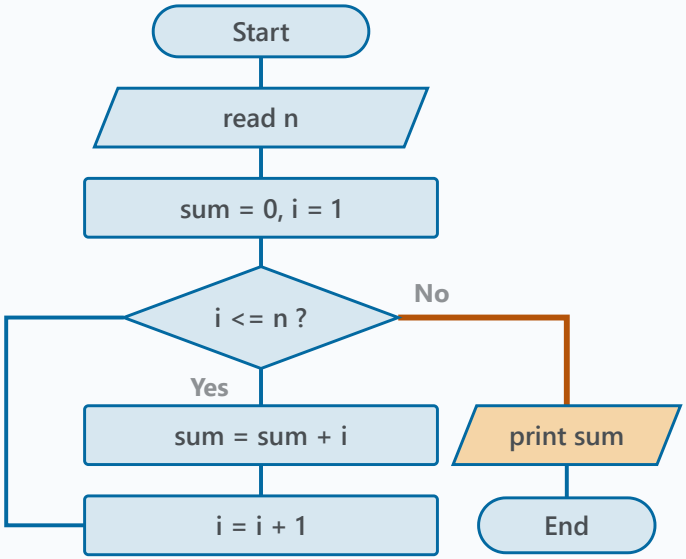
```
1 public static void main(String[] args) {
2     int n = 3;
3     int sum = 0, i = 1;
▶     while (i <= n) {
5         sum = sum + i;
6         i = i + 1;
7     }
8     System.out.println(sum);
9 }
```

VARIABLES

`n = 3` `sum = 6` `i = 4`

OUTPUT

From Flowchart to Java Code Step by Step (n = 3)



Output symbol → `println(sum)` : **6** is printed.

```
1 public static void main(String[] args) {
2     int n = 3;
3     int sum = 0, i = 1;
4     while (i <= n) {
5         sum = sum + i;
6         i = i + 1;
7     }
▶     System.out.println(sum);
9 }
```

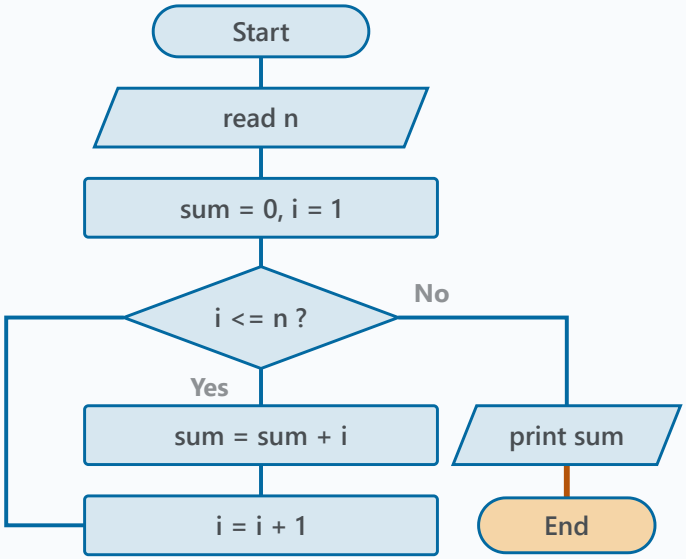
VARIABLES

`n = 3` `sum = 6` `i = 4`

OUTPUT

```
6
```

From Flowchart to Java Code Step by Step (n = 3)



End → the method closes. Every symbol has a code counterpart.

```
1 public static void main(String[] args) {
2     int n = 3;
3     int sum = 0, i = 1;
4     while (i <= n) {
5         sum = sum + i;
6         i = i + 1;
7     }
8     System.out.println(sum);
▶ }
```

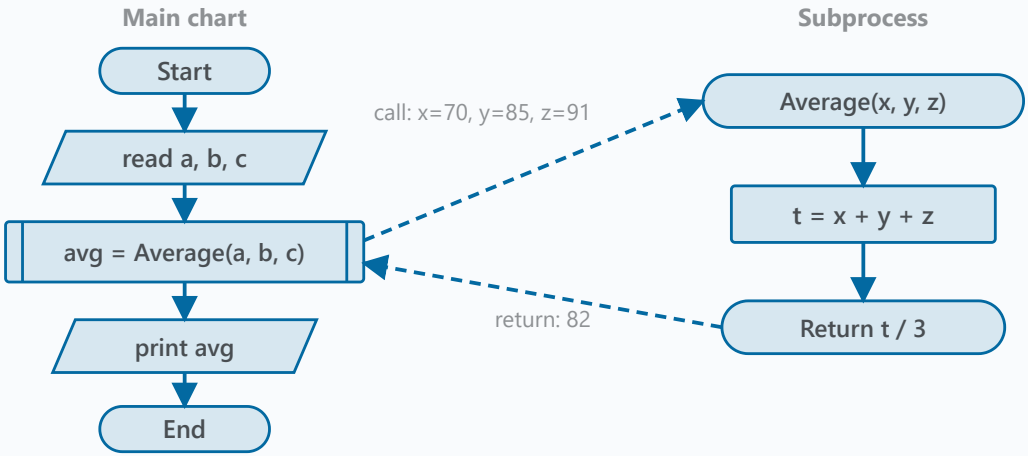
VARIABLES

`n = 3` `sum = 6` `i = 4`

OUTPUT

```
6
```

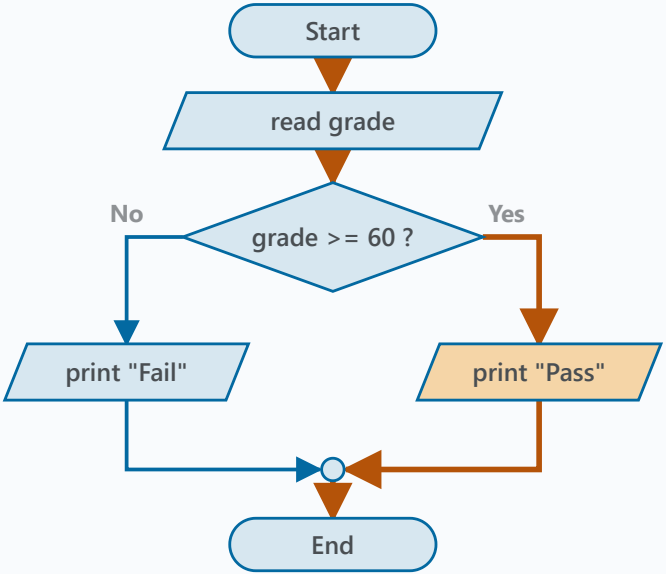
Modular Charts with Subprocesses



- The **predefined process** symbol calls a separately drawn chart; when the subprocess ends, flow resumes in the main chart **at the same point**.
- The subchart starts with its **name and parameters** instead of Start, and ends with **Return** instead of End. For a = 70, b = 85, c = 91: t = 246, returned value 82.
- One subprocess can be called from many places but is drawn once. In Java each subprocess becomes a **method**: `static int average(int x, int y, int z) { return (x + y + z) / 3; }`

VERIFICATION

Testing a Flowchart

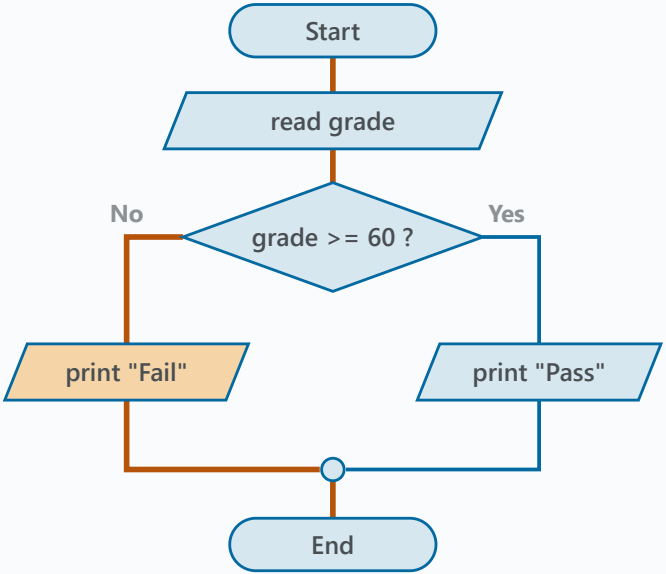


Normal input grade = 72: the Yes path runs.

Step 1 / 3

- Follow the arrows step by step with a normal input.
- Choose test values that exercise **every exit** of each decision.
- Test boundary values (like 60) separately.
- Try cases where the loop runs zero times, once and many times.
- Look for paths that never return or never reach End.

Testing a Flowchart

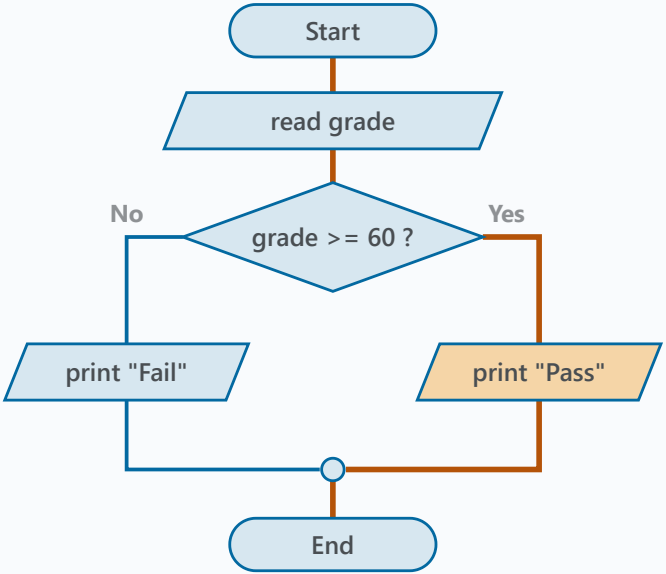


The other branch with grade = 45: the No path runs. Now every box has run at least once.

- Follow the arrows step by step with a normal input.
- Choose test values that exercise **every exit** of each decision.
- Test boundary values (like 60) separately.
- Try cases where the loop runs zero times, once and many times.
- Look for paths that never return or never reach End.

VERIFICATION

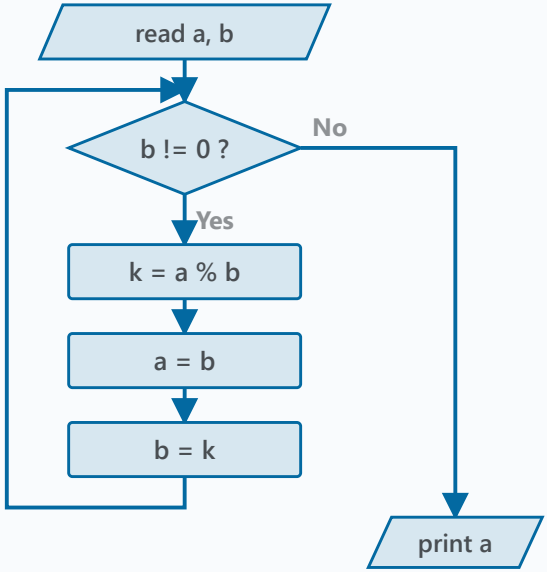
Testing a Flowchart



Boundary value grade = 60: 60 >= 60 is true → "Pass". Had `>` been written, this test would catch it.

- Follow the arrows step by step with a normal input.
- Choose test values that exercise **every exit** of each decision.
- Test boundary values (like 60) separately.
- Try cases where the loop runs zero times, once and many times.
- Look for paths that never return or never reach End.

Hand Tracing with a Trace Table: Euclid (48, 18)



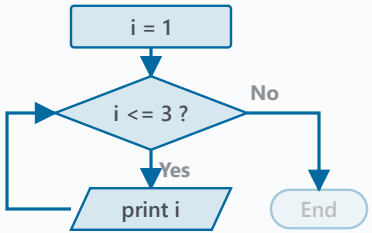
Step	a	b	k	b != 0 ?
start	48	18	—	18 != 0 true
pass 1	18	12	12	12 != 0 true
pass 2	12	6	6	6 != 0 true
pass 3	6	0	0	0 != 0 false

- Each variable is a column and each pass a row; the decision's result is recorded too. Result: **6** is printed, GCD(48, 18) = 6.
- The table shows why the order in the chart matters: if **a = b** and **b = k** were swapped, a would take b's new value and the result would be wrong.
- The loop ran 3 passes and the decision was asked 4 times: as always, the decision runs once more than the body.

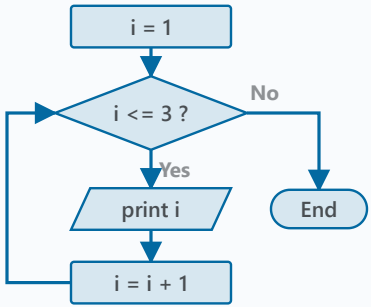
VERIFICATION

Common Mistakes

Mistake	Effect	Fix
Unlabeled decision path	Unclear which path is which	Write Yes/No
Counter never updated	Infinite loop	Add a counter step
Unconnected symbol	Flow breaks	Connect with an arrow
Many steps in one symbol	Harder to read	Split into steps



✗ i stays 1: 1 1 1 1 … End is never reached



✓ i = i + 1 added: 1 2 3, then End

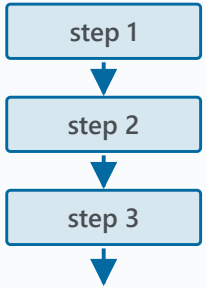
Common Mistakes (cont.)

Mistake	Symptom	Fix
The accumulator is reset **inside** the loop	For 4, 7, 2 it prints 2 instead of 13: only the last value survives	Move the sum = 0 box before the loop
The back arrow skips the **read** box	The same x is processed over and over; the sentinel never arrives	Route the arrow back above the read box
Assignment in a decision box: **x = 0 ?**	Unclear whether it is a question or an assignment	Write a comparison: x == 0 ?
Sentinel checked **after** processing	−1 is added to the sum and the count is one too high	Ask the decision before adding and counting
Broad condition first in a decision chain	The lower branches never run	Order from narrow to broad
Only one arrow leaves a decision	Flow breaks in the other case	Draw both exits, label Yes/No
No zero check before dividing	Division by zero if no data is entered	Add a count > 0 ? decision

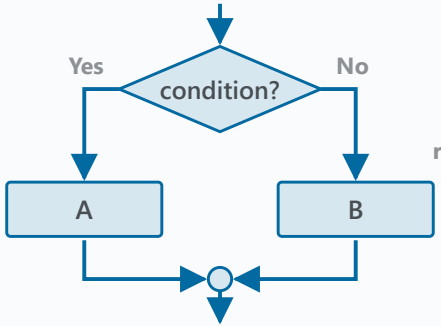
SUMMARY

Chapter Summary

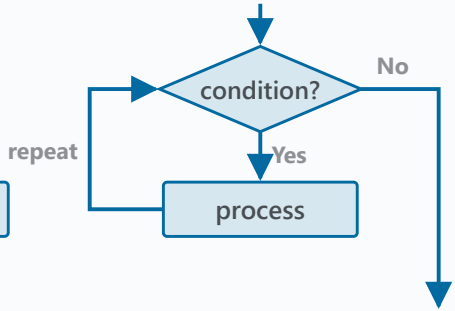
- A flowchart shows an algorithm's order of execution with standard symbols.
- Sequential flow is linear, conditional flow branches, and repetition has a path going back.
- Decision paths must be labeled, and every path must reach a next step.
- Verify a chart by hand-tracing it with different test inputs.



Sequential: linear



Conditional: branches



Repetition: loops back

SUMMARY

Review Questions

1. What do the process and decision symbols represent?
2. Why must decision exits be labeled?
3. What causes an infinite loop in a flowchart?
4. What is the difference between a counter and an accumulator?
5. Which inputs should you choose to test a flowchart?

Exercises

1. Draw a flowchart that prints whether a number is odd or even (hint: num % 2 == 0 ?).
2. Draw a flowchart with nested decisions that converts an exam score from 0–100 to a letter grade AA / BA / … / FF; trace it for 85, 60 and 42.
3. Draw a chart for the **product** of the numbers from 1 to n; what should the accumulator start at?
4. Draw a chart that reads numbers until the user enters 0, then prints their sum and how many were entered (counter + accumulator).
5. Add a 5-try limit to the guessing game: if the number is not found in 5 tries, print "You lose".
6. Find the bug: i = 1, i <= 10 ? Yes → print i, back to the decision. How do you fix it?

▶ **Hints**

References

- ISO 5807:1985, Information processing: Documentation symbols and conventions for data, program and system flowcharts.
- Deitel and Deitel, Java: How to Program, Early Objects, 11th ed., Pearson, 2018.
- Sedgewick and Wayne, Computer Science: An Interdisciplinary Approach, Addison-Wesley, 2016: **1.3 Conditionals and Loops**.
- Oracle, The Java Tutorials: **Control Flow Statements**.
- Downey and Mayfield, Think Java, 2nd ed., O'Reilly, 2019 (**free online edition**).

The End

Chapter 4.2: Flowcharts

Previous: Chapter 4.1: Algorithms · Next: Chapter 4.3: Introduction to Programming