Finger Exercises 6

Reading a Problem Statement

as long as it takes

Comp 102 — Programming Fundamentals

What this is. In class we solved five contest problems. The code was short. The hard part was the *reading*. Most people who get stuck on a problem like these can already write an `if`. They get stuck one step earlier: they do not know what the statement is asking. This sheet slows that step right down. It gives you five questions to ask of every statement. Then it makes you ask them, one at a time, of the same five problems we did in class. You already know the answers. That is the point. This time, watch *how* you get there.

How to do it. On **paper**, **no computer**. Take as long as you need. Do not skip a box that asks you to *copy a sentence* — copying it *is* the exercise. Each problem has its own short Parts A–E. The answer key is at the back — do not peek until you are done.

Predict first, then check. Write down what you think *before* you look anything up. A wrong guess teaches you more than an answer you only read.

The Five Questions

ask them in this order, every time

First, read the story *once*, quickly. Do not try to solve it yet. Just find out what it is about. Then start here:

❶ **IN — What comes in?** How many lines? Is each line a number or a word? What range? Write one `input()` per line.

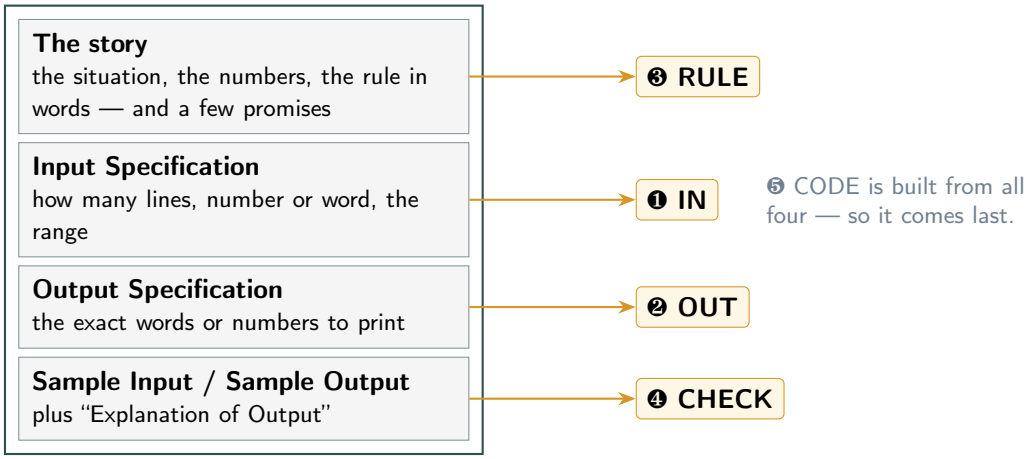
❷ **OUT — What goes out?** Which *exact* words or numbers? How many lines? Copy the spelling. Nothing extra.

❸ **RULE — What is the rule?** Find the one sentence that holds the formula or the condition. Copy it. Write it as maths.

❹ **CHECK — Does it match the sample?** Push Sample Input 1 through your rule *by hand*. Compare with Sample Output 1. Then do Sample 2. If they do not match, your rule is wrong — go back to ❸.

❺ **CODE — Now write code.** Not before. **IN** becomes `int(input())` lines. **RULE** becomes arithmetic or an `if`. **OUT** becomes `print`.

every CCC statement has these four sections



Problem 1 • CCC '22 J1 — Cupcake Party

no `if` needed — find out why

Have the statement from class in front of you. Every answer below comes from it.

Part A • Read

RECALL three numbers, one promise

A1. The story has three numbers: 8, 3 and 28. Say what each one counts.

8 = _____ 3 = _____

28 = _____

A2. What is the program asked to find? Copy the words.

_____

A3. The story makes a promise about the total number of cupcakes. Copy it.

_____

Part B • In and Out

CLASSIFY what comes in, what goes out

B1. How many lines of input? What is on each line? Do they count *boxes* or *cupcakes*?

Lines: _____ Line 1: _____ Line 2: _____

Boxes or cupcakes? _____

B2. Write the two Python lines that read them.

_____

_____

B3. For Sample 1, write *exactly* what the program prints — every character.

_____

Would `print("Left over:", 3)` be accepted? _____ Why?

_____

Part C • Rule and Check

TRANSLATE sentences into maths, then test it

C1. R regular boxes hold _____ cupcakes. S small boxes hold _____ cupcakes.

Total = _____

C2. How many cupcakes are eaten? Is that number an input, or is it in the story?

_____

C3. Left over = _____

C4. Push these through your formula. The first two rows are the samples. The third is a test of your own.

R	S	**Total**	**Left over**
2	5	_____	_____
2	4	_____	_____
8	0	_____	_____

Do the first two rows match the sample outputs? _____

Part D • Do We Need an `if`?

EXPLAIN the question everyone asks

An `if` is for when the program must do *one thing* for some inputs and *a different thing* for others.

D1. In C4, did you do the same steps for every row? _____ So how many `if`s does this program need? _____

D2. Which sentence of the story makes sure total -28 is never negative?

_____

D3. Now pretend that sentence is gone, and the input is `1` and `1`. What does the formula give? _____ Does that make sense? _____

Now there *are* two cases. Write the `if` line that separates them:

```
if _____:
```

Why `>=` and not `>`?

_____

Part E • Code

TRANSLATE the rule, in four lines

E1. Fill in the blanks.

```
regular = int(input())
small = _____
total = _____
print(_____)
```

E2. A student wrote `print(total % 28)` instead. It prints `3` for Sample 1 and `0` for Sample 2 — both correct. Use your row $R = 8$, $S = 0$ to show it is still wrong. Which word in the story says why?

_____

_____

Problem 2 • CCC '19 J1 — Winning Score

three outcomes — the first `if`

Part A • Read

RECALL shots, points, outcomes

A1. There are three ways to score. Name each one and say how many points it is worth.

_____ = _____ points _____ = _____ points

_____ = _____ point

A2. How many different answers can the program give? Write them exactly as the Output Specification spells them, and say when each is used.

_____ when _____ _____ when _____

_____ when _____

Part B • In

CLASSIFY six lines, fixed order

B1. Six lines come in. Say what each one counts.

1: _____ 2: _____ 3: _____

4: _____ 5: _____ 6: _____

B2. Sample 1 starts with `10`. Is that 10 *points* or 10 *shots*? How many points is it worth?

_____

Part C • Rule and Check

TRANSLATE two totals, then compare

C1. Call the Apples' three counts `a3`, `a2`, `a1`. Write their points as a formula.

Apples' points = _____

C2. Fill in the table for both samples.

	Apples' points	Bananas' points	Output
Sample 1	_____	_____	_____
Sample 2	_____	_____	_____

C3. Sample 2 is a tie. Why do you think the setter put a tie in the samples?

_____

Part D • The `if`

EXPLAIN three answers means three branches

D1. Cupcake Party printed the *result of a formula*. This program prints a *letter*. Where does the letter come from — a formula, or a decision?

_____

D2. Three answers, so three cases. Write each case in words, then as Python.

`A`: _____ → `if` _____`:`

`B`: _____ → `elif` _____`:`

`T`: _____ → `else:`

D3. A student wrote only two branches: `if a_total > b_total:` print `A`, `else:` print `B`. Which sample catches this, and what does it print there?

_____

D4. Another student wrote `if a_total >= b_total:` for the first branch. What goes wrong, and on which sample?

Part E • Code

TRANSLATE six reads, two totals, one decision

E1. Fill in the blanks. The six reads are done, the way we wrote them in class.

```
a3, a2, a1 = int(input()), int(input()), int(input())
b3, b2, b1 = int(input()), int(input()), int(input())
a_total = _____________________________
b_total = _____________________________
if _____________________:
    print("A")
elif _____________________:
    print(________)
else:
    print(________)
```

E2. A student compared the 3-point counts instead of the totals: `if a3 > b3:`. Sample 1 prints _____. Which of the five questions did they skip?

Problem 3 • CCC '18 J1 — Telemarketer or not? three properties, one and

Part A • Read RECALL what makes a telemarketer number

A1. A telemarketer number has three properties. Copy them, one per line.

1. _____
2. _____
3. _____

A2. Must a number have *all three*, or is *one* enough? Copy the words that tell you.

_____

A3. "The first of these four digits" — the first digit of the whole phone number, or of the last four?

_____

A4. Which word is printed for a telemarketer number, and which for any other? Copy the spelling.

telemarketer → _____ any other → _____

Part B • In CLASSIFY four digits, four lines

B1. How many lines? What is on each? Name the lines to match the wording of the properties.

Lines: _____ Names: _____

B2. We read them with `int(input())`. So is `first` the number 8 or the text `"8"`? Which comparison is right: `first == 8` or `first == "8"`?

_____

Part C • Rule and Check TRANSLATE three properties, three conditions

C1. Write each property as a Python condition.

1. _____
2. _____
3. _____

C2. Which word joins the three: `and` or `or`? Why?

_____

C3. Tick the properties each number has, then write the output.

Digits	1st is 8/9	2nd = 3rd	last is 8/9	Output
9 6 6 8 (Sample 1)	☐	☐	☐	_____
5 6 6 8 (Sample 2)	☐	☐	☐	_____
8 3 3 8	☐	☐	☐	_____
8 1 2 3	☐	☐	☐	_____

C4. For Sample 2, property 1 fails. Do you still need to look at properties 2 and 3 to know the answer? _____ Why?

_____

Part D • The `if`

EXPLAIN two answers, one condition

D1. How many different outputs? _____ So how many branches? _____ Which keywords? _____

D2. A student wrote property 1 as `first == 8 or 9`. It looks right. Why is it *always* true? Which sample catches it?

_____

D3. In class we also saw a version with three nested `if`s and `print("answer")` written three times. Why three times?

_____

Part E • Code

TRANSLATE the one-line version

E1. Fill in the blanks.

```
if (first == 8 or _________) and (second ____ third) and (fourth == 8 or _________):
    print(___________)
else:
    print(___________)
```

E2. A student joined the three brackets with `or` instead of `and`. Use your table in C3: which numbers get the wrong output?

_____

Problem 4 • CCC '20 J1 — Dog Treats

a formula, then a boundary

Part A • Read

RECALL the formula is given to you

A1. The story gives the formula. Copy it, and say what S, M and L count.

Formula: _____

$S =$ _____ $M =$ _____ $L =$ _____

A2. Copy the sentence that decides happy or sad. Underline the two words that tell you 10 itself counts as happy.

_____

Part B • In and Out

CLASSIFY three counts in, one word out

B1. Three lines come in. In what order? Copy the phrase that fixes it.

_____

B2. "Each line contains a non-negative integer less than 10." Smallest value on a line? _____ Largest? _____ Can the score still reach 10? Give an input that does.

_____

B3. Copy the two possible outputs exactly. Would `Happy` be accepted? _____

_____ _____

Part C • Rule and Check

TRANSLATE score, then compare

C1. Write the score in Python, using `s`, `m`, `l`.

```
score = _____
```

C2. Fill in for both samples.

	S, M, L	Score	Output
Sample 1	3, 1, 0	_____	_____
Sample 2	3, 2, 1	_____	_____

C3. Sample 2 scores *exactly* 10. Why 10, and not 12 or 15?

_____

Part D • The `if`

EXPLAIN Cupcake's shape, with one difference

D1. Like Cupcake Party, this is multiply, multiply, multiply, add. What is the one difference that makes an `if` necessary here?

_____

D2. Write the condition. `if` _____:

A student wrote `if score > 10:`. What does Sample 2 print? _____

D3. Another student wrote the `if` with `print("happy")` but no `else`. What does Sample 1 print?

_____

Part E • Code

TRANSLATE three reads, one decision

E1. Fill in the blanks.

```
s = int(input())
m = int(input())
l = int(input())
if ____________________________:
    print(__________)
else:
    print(__________)
```

Problem 5 • CCC '17 J1 — Quadrant Selection

four answers, two facts each

Part A • Read

RECALL the diagram is part of the statement

A1. The story explains quadrants 1 and 2 in words. Where do you find out about quadrants 3 and 4? _____

A2. Fill in the sign table.

Quadrant	x is	y is
1	_____	_____
2	_____	_____
3	_____	_____
4	_____	_____

A3. Copy the promise about the coordinates. What does it let you skip?

_____

Part B • In and Out

CLASSIFY two numbers in, one digit out

B1. Two lines: which is x and which is y? Can they be negative? Copy the range.

_____

B2. Copy the four possible outputs. Would `Quadrant 1` be accepted? _____

_____

Part C • Rule and Check

TRANSLATE each quadrant is two facts at once

C1. "$x > 0$" on its own — which quadrants does it match? _____ So how many facts must you check to pin down one quadrant? _____

C2. Write the four conditions in Python, from your sign table.

1: _____ 2: _____

3: _____ 4: _____

C3. Check the samples, then add one point of your own in a quadrant the samples miss.

x	y	Output
12	5	_____
9	−13	_____
_____	_____	_____

Part D • The `ifs`

EXPLAIN four cases, and a trap with `else`

D1. Four outputs, so four cases. In class we used four separate `ifs`. Can two of them be true for the same point? _____ Why not?

_____

D2. A student kept the first three `ifs` and replaced the fourth with `else:`. What does Sample 1 print? _____ Why?

_____

D3. How would you change the program so that `else` *is* safe for the fourth case? Which promise makes it safe?

_____

Part E • Code

TRANSLATE four conditions

E1. Fill in the four conditions.

```
x = int(input())
y = int(input())
if _____________________:
    print("1")
if _____________________:
    print("2")
if _____________________:
    print("3")
if _____________________:
    print("4")
```

Answer Key

Check your work *after* attempting everything. If an answer surprises you, go back to the statement and find the sentence that settles it.

Problem 1 — CCC '22 J1 — Cupcake Party

A1. 8 = cupcakes in a regular box. 3 = cupcakes in a small box. 28 = *students*, not cupcakes.

A2. "how many cupcakes will be left over". Not the total. Not the boxes.

A3. "a total of at least 28 cupcakes". Keep it in mind — Part D is about it.

B1. Two. Line 1 is R, the regular boxes; line 2 is S, the small boxes. Boxes — the cupcakes are for the program to work out.

B2. `regular = int(input())` / `small = int(input())`. `int` because we will multiply. Nothing inside the brackets: a prompt would be printed, and the judge reads everything printed.

B3. `3` — one number, nothing else. `"Left over:"` is rejected: the Output Specification says "Output the number". Anything not asked for is wrong.

C1. $8 \times R$ and $3 \times S$. Total $= 8R + 3S$. Multiply, not add: two boxes of 8 is $8 + 8$, which is 8×2.

C2. 28 — one per student. It is in the story, so it is typed straight into the program, never read with `input()`.

C3. Total -28, that is $8R + 3S - 28$. Total *minus* 28, not 28− total.

C4. 2, 5: 31 and 3. 2, 4: 28 and 0. 8, 0: 64 and 36. Both samples match, so the rule is right. Row 3 is used in E2.

D1. Yes — multiply, multiply, add, subtract, every time. Zero `if`s. One recipe for all inputs means there is nothing to decide.

D2. The promise from A3: "at least 28 cupcakes". It removes the "not enough" case, so there is no second case to handle.

D3. $8 + 3 - 28 = -17$; no. `if total >= 28:`. Exactly 28 is still enough — everyone gets one and 0 are left — and Sample 2 is that case. An `if` appears the moment the statement has two different outcomes. This statement has one.

E1. `small = int(input())` / `total = regular*8 + small*3` / `print(total - 28)`.

E2. 64 cupcakes: `64 % 28` is 8, but 36 are left. The remainder keeps handing cupcakes round; the story says each student gets *one*. Two samples agreeing is not proof — that is why C4 has a row of your own.

Problem 2 — CCC '19 J1 — Winning Score

A1. 3-point shot = 3. 2-point field goal = 2. 1-point free throw = 1.

A2. Three. `A` when the Apples have more points; `B` when the Bananas have more; `T` otherwise — a tie. Three different answers is the first sign that this program must *choose*.

B1. 1–3: the Apples' 3-point shots, 2-point field goals, free throws. 4–6: the same three for the Bananas. The order is fixed by the Input Specification; the program can only tell the lines apart by their order.

B2. 10 shots — the input counts shots, never points. Ten 3-point shots are worth 30 points. Turning shots into points is the program's job.

C1. `a3*3 + a2*2 + a1` (the `*1` on free throws is optional). Each count is multiplied by what that kind of shot is *worth*.

C2. Sample 1: $30 + 6 + 7 = 43$ against $24 + 18 + 6 = 48$ — `B`. Sample 2: $21 + 6 + 0 = 27$ against $18 + 8 + 1 = 27$ — `T`.

C3. To catch programs with only two branches ("A wins, otherwise B"). The tie is the case people forget. When a sample looks like a special case, it is there on purpose.

D1. A decision: compare the two totals and pick the letter. Arithmetic cannot produce a letter. Choosing is exactly the job of an `if`.

D2. `A`: Apples have more — `if a_total > b_total:`. `B`: Bananas have more — `elif b_total > a_total:`. `T`: neither — `else:`. The `else` needs no condition: if neither total is bigger, they are equal.

D3. Sample 2: 27 is not greater than 27, so it prints `B` instead of `T`. Two branches can only give two answers; the statement asks for three.

D4. A tie now prints `A` — Sample 2 again. "More points" is strictly more: `>`, not `>=`. Read the comparison word.

E1. `a_total = a3*3 + a2*2 + a1`; `b_total = b3*3 + b2*2 + b1`; `if a_total > b_total: ... elif b_total > a_total:` print `"B"` ... `else:` print `"T"`. (In class the middle branch tested `==` and printed `T`; both orders are right.)

E2. `A` — 10 is more than 8 — but the answer is `B`. They skipped **RULE**: the story says *points* decide, and points come from all three kinds of shot. Caught by Sample 1.

Problem 3 — CCC '18 J1 — Telemarketer or not?

A1. 1. The first of the four digits is an 8 or 9. 2. The last digit is an 8 or 9. 3. The second and third digits are the same.

A2. All three: "the last four digits *have* three properties". The story's own examples 8229, 8338 and 9008 each pass all three.

A3. Of the last four. The program never sees the other three digits; the input is only the last four.

A4. Telemarketer → `ignore`. Any other → `answer`. Easy to swap: the number that *matches* is the one you *ignore*.

B1. Four, one digit each: `first`, `second`, `third`, `fourth`. Naming them after the statement makes the properties easy to write.

B2. The number, so `first == 8`. (Reading with plain `input()` would also work, but then *every* comparison must use `"8"`. Mixing the two is never true.)

C1. 1. `first == 8 or first == 9`. 2. `fourth == 8 or fourth == 9`. 3. `second == third`. Each side of an `or` is a complete comparison.

C2. `and` — all three must hold (A2). With `or`, one property would be enough, and 8123 would be ignored.

C3. 9668: all three, `ignore`. 5668: property 1 fails (5); the other two hold — `answer`, because one failure is enough. 8338: all three, `ignore`. 8123: only property 1 — `answer`.

C4. No. With `and`, one failed property settles it. (Python stops there too.)

D1. Two outputs, two branches: `if` / `else`. No `elif` — there is no third answer.

D2. Python reads it as `(first == 8) or (9)`, and a bare 9 counts as true. So it never rejects any digit. Sample 2 catches it: 5668 prints `ignore` instead of `answer`. Write `first == 8 or first == 9`.

D3. There are three different places a number can fail — one per property — and each needs its own `else`. The single `and` condition says the same thing in one line, with one `else`. Same answers, less to get wrong.

E1. `first == 9`; `==`; `fourth == 9`; `"ignore"`; `"answer"`.

E2. 5668 and 8123 — each has at least one property, so `or` says `ignore`; both should be `answer`. Sample 2 catches it.

Problem 4 — CCC '20 J1 — Dog Treats

A1. $1 \times S + 2 \times M + 3 \times L$. S = small treats, M = medium, L = large. Bigger treat, bigger weight.

A2. "If Barley's happiness score is 10 or greater then he is happy. Otherwise, he is sad." The words are "or greater": 10 is included.

B1. Small, then medium, then large: "The first line contains the number of small treats, S, the second ... M, and the third ... L".

B2. 0 and 9. Yes: the limit is per *line*, not on the score. Sample 2 itself (3, 2, 1) scores exactly 10, and 9, 9, 9 scores 54.

B3. `happy` and `sad`, lower case. `Happy` is rejected — the judge compares letter by letter.

C1. `score = 1*s + 2*m + 3*l` (or `s + 2*m + 3*l`). Straight from the formula.

C2. Sample 1: $3 + 2 + 0 = 5$, `sad`. Sample 2: $3 + 4 + 3 = 10$, `happy`.

C3. 10 is the boundary. A program with `> 10` instead of `>= 10` says `sad` here, and the setter wants you to find that out before submitting. When a sample sits on the edge, the edge is where the mistakes are.

D1. Cupcake Party printed the number the formula gave. Here the formula's number is never printed — it is *compared* to 10, and a word is chosen. Two possible words means a decision.

D2. `if score >= 10:`. With `>`, Sample 2 (score 10) prints `sad`. "10 or greater" includes 10 — see A2.

D3. Nothing at all. A score of 5 skips the `if` body, and there is nothing else to run. Missing output is wrong output. Two outcomes need two branches.

E1. `if s + 2*m + 3*l >= 10:` print `"happy"`; `else:` print `"sad"`. (Or compute `score` first, then `if score >= 10:`.)

Problem 5 — CCC '17 J1 — Quadrant Selection

A1. From the diagram. A diagram is part of the statement; read it like a sentence.

A2. 1: $x > 0, y > 0$. 2: $x < 0, y > 0$. 3: $x < 0, y < 0$. 4: $x > 0, y < 0$. The numbering goes anticlockwise.

A3. "neither of the two coordinates will be 0". A point on an axis is in no quadrant, and the promise says it never comes — so there is no fifth case, and `>` and `<` are enough.

B1. Line 1 is x, line 2 is y. Yes: $-1000 \le x \le 1000$, and the same for y. `int(input())` reads `-13` without any help.

B2. `1`, `2`, `3` or `4` — a bare digit. `Quadrant 1` is rejected: extra words.

C1. Quadrants 1 *and* 4 — half the plane. Two facts: the sign of x and the sign of y. That is why every condition below needs an `and`.

C2. 1: `x > 0 and y > 0`. 2: `x < 0 and y > 0`. 3: `x < 0 and y < 0`. 4: `x > 0 and y < 0`.

C3. $(12, 5)$: `1`. $(9, -13)$: `4`. Your own: anything in quadrant 2 or 3, e.g. $(-3, -7)$ gives `3`. The samples test only two of the four branches; the other two are your job.

D1. No. x cannot be both > 0 and < 0, and the same for y. Exactly one of the four fires, so separate `if`s are safe here.

D2. `1` and then `4` — two lines. The `else` belongs only to the third `if`, and $(12, 5)$ is not in quadrant 3, so that `else` runs too. An `else` pairs with the `if` directly above it, nothing more.

D3. Make it one chain: `if` / `elif` / `elif` / `else`. Then `else` means "none of the first three", which is quadrant 4 — *because* the promise rules out points on the axes. Without that promise, `else` would also catch $(0, 5)$.

E1. The four conditions from C2, in order: `x > 0 and y > 0`, `x < 0 and y > 0`, `x < 0 and y < 0`, `x > 0 and y < 0`.