# Finger Exercises 5

*Conditions*

≈ 30 min + stretch

Comp 102 — Programming Fundamentals

**What this is.** A short workout for what we just covered in class. Every program here is new — none are from the slides — so you have to *trace* them, not remember them. The skill being drilled is simple to state and easy to get wrong: *which lines run, and which are skipped?*

**How to do it.** On **paper**, **no computer**, in one sitting. Put your finger on each line as Python would reach it. If an item takes you more than a minute, skip it, finish the rest, then come back. Parts A–H are for *everyone*. Part I is a **stretch**: skip it freely, or attack it if A–H felt smooth. The Console Quests at the end are for tonight, *at* a computer. The answer key is on the last page — do not peek until you are done.

**Predict first, then check.** Write down what you think *before* you look anything up or type anything in — even for the Console Quests. A guess that turns out wrong teaches you far more than an answer you only watched the computer produce.

## Part A • Do You Remember?

RECALL ≈ 3 min

**A1.** The three keywords that build a decision are _____ , _____ and _____ . The one that *never* takes a condition is _____ .

**A2.** Python decides which lines belong to a body by their _____ . A body begins on the line after one that ends with a _____ .

**A3.** In one `if`/`elif`/`else` chain, _____ of the bodies run. With three *separate* `if`s in a row, anywhere from ____ to ____ of the bodies may run.

**A4.** A body that must exist but should do nothing yet contains the single keyword _____ .

**A5.** Circle **True** or **False**:

(a) An `if` with no `else` may end up running none of its body lines. True / False

(b) `elif` can appear on its own, without an `if` above it. True / False

(c) A non-indented line after an `if` block runs whatever the condition was. True / False

(d) In a chain, Python keeps testing the remaining `elif` conditions even after one has come out `True`. True / False

(e) `if 3 <= n <= 9:` means the same as `if 3 <= n and n <= 9:`. True / False

(f) The condition of an `if` must be a comparison; a bare variable like `if flag:` is not allowed. True / False

## Part B • Which Lines Print?

TRACE ≈ 7 min

Write the output, one line per box. Write *(nothing)* if a program prints nothing.

**B1.** A strict comparison on the boundary.

```
1 speed = 60
2 if speed > 60:
3     print("Fine")
4     print("Slow down")
5 print("Drive safe")
```

Output:

_____

_____

_____

**B2.** Strings in a condition.

```
1 colour = "Red"
2 if colour == "red":
3     print("stop")
4 else:
5     print("go")
6 print("end")
```

Output:

_____

_____

**B3.** No comparison at all.

```
1 count = 0
2 if count:
3     print("some")
4 else:
5     print("none")
```

Output:

_____

**B4.** One chain, three different values of `h`. Write what is printed for each.

```
1 if h < 12:
2     print("morning")
3 elif h < 17:
4     print("afternoon")
5 elif h < 21:
6     print("evening")
7 else:
8     print("night")
```

`h = 12` → _____

`h = 21` → _____

`h = 5` → _____

**B5.** An `if` inside an `if`. Watch the indentation of line 5.

```
1 n = 24
2 if n % 2 == 0:
3     if n % 3 == 0:
4         print("six")
5     print("two")
6 else:
7     print("odd")
8 print("bye")
```

Output:

_____

_____

_____

**B6.** Two `if`s that change the variable they test.

```
1 x = 4
2 if x > 3:
3     x = x * 2
4 if x > 6:
5     x = x - 1
6 print(x)
```

Prints: _____

If line 4 said `elif` instead, it would print: _____

**B7.** Three separate `if`s that all assign to the same name.

```
1 marks = 72
2 if marks >= 40:
3     grade = "pass"
4 if marks >= 70:
5     grade = "merit"
6 if marks >= 85:
7     grade = "distinction"
8 print(grade)
```

How many bodies run? _____

Prints: _____

With `elif` on lines 4 and 6 it would print: _____

**B8.** Two `else`s. Decide which `if` each one belongs to before you trace.

```
1 a = 5
2 b = 12
3 if a > 3:
4     if b > 15:
5         print("P")
6     else:
7         print("Q")
8 else:
9     print("R")
10 print("S")
```

Output:

_____

_____

The `else` on line 6 belongs to the `if` on line _____

## Part C • Same Program, Four Inputs

TRACE ≈ 6 min

An airline weighs each bag. Trace the program for each input and write **everything** it prints (there may be one or two lines).

```
1 kg = int(input("Bag weight (kg)? "))
2 if kg > 30:
3     print("Refused")
4 elif kg > 20:
5     fee = (kg - 20) * 500
6     print("Fee:", fee)
7 else:
8     print("Free")
9 print("Next!")
```

| User types | First line printed | Second line printed (if any) |
|---|---|---|
| 12 | _____ | _____ |
| 25 | _____ | _____ |
| 30 | _____ | _____ |
| 45 | _____ | _____ |

**C1.** An `input()` *inside* a branch. For each run, write how many questions the user is asked and everything printed.

```
code = input("Code? ")
if code == "A1":
    pin = input("PIN? ")
    if pin == "9090":
        print("Open")
    else:
        print("Wrong PIN")
else:
    print("Unknown code")
print("Bye")
```

| User types | Questions asked | Printed |
|---|---|---|
| B2 | _____ | _____ |
| A1, then 1234 | _____ | _____ |
| A1, then 9090 | _____ | _____ |

## Part D • Runs, Skips, or Crashes? CLASSIFY ≈ 3 min

Let `n = 6` and `word = "hi"`. Each line below is the header of an `if`. Tick whether its body **runs**, is **skipped**, or Python reports an **error**.

| Header | Runs | Skipped | Error |
|---|---|---|---|
| `if n % 3 == 0:` | ☐ | ☐ | ☐ |
| `if n = 6:` | ☐ | ☐ | ☐ |
| `if word:` | ☐ | ☐ | ☐ |
| `if len(word) > 2:` | ☐ | ☐ | ☐ |
| `if n > 5 and word == "Hi":` | ☐ | ☐ | ☐ |
| `if not n < 10:` | ☐ | ☐ | ☐ |
| `if 5 < n < 7:` | ☐ | ☐ | ☐ |
| `if word + 1:` | ☐ | ☐ | ☐ |
| `if not word:` | ☐ | ☐ | ☐ |
| `if word > 1:` | ☐ | ☐ | ☐ |

## Part E • Put the Indentation Back

FIX ≈ 2 min

Someone deleted every indent from this program. The *intended* behaviour: if `age` is 18 or more, print `adult` and `can vote`; otherwise print `minor`; print `goodbye` in every case. Tick each line that must be indented.

| Line | Code | Indent? |
|---|---|---|
| 1 | `age = 16` | ☐ |
| 2 | `if age >= 18:` | ☐ |
| 3 | `print("adult")` | ☐ |
| 4 | `print("can vote")` | ☐ |
| 5 | `else:` | ☐ |
| 6 | `print("minor")` | ☐ |
| 7 | `print("goodbye")` | ☐ |

With `age = 16` the fixed program prints:

_____

_____

## Part F • Find and Fix

FIX ≈ 6 min

**F1.** Delivery cost by distance: over 50 km should cost 400, over 10 km should cost 200, otherwise 100. For `km = 80` this prints the wrong number. What does it print, and what is the fix?

```
1 km = 80
2 if km > 10:
3     cost = 200
4 elif km > 50:
5     cost = 400
6 else:
7     cost = 100
8 print(cost)
```

It prints: _____

Fix: _____

_____

**F2.** This crashes with `NameError: name 'label' is not defined`. Why, and what one addition fixes it?

```
1 temp = 22
2 if temp > 25:
3     label = "warm"
4 print(label)
```

Add: _____

_____

**F3.** Python refuses line 3. Circle the mistake and write the corrected line.

```
1 if n % 2 == 0:
2     print("even")
3 else n % 2 == 1:
4     print("odd")
```

Line 3: _____

**F4.** Also refused. The programmer indented something one level too far. Which line, and where should it go?

```
1 score = 70
2 if score >= 50:
3     print("pass")
4     else:
5         print("fail")
```

Line ____ must line up with _____ ;

line ____ moves left with it.

**F5.** Line 4 is indented, yet Python refuses it. What is the error, and what are the **two** legal places for that line?

```
1 n = 5
2 if n > 0:
3     print("positive")
4       print("done")
```

Error: _____

Either indent it like line ____ , or _____

**F6.** The programmer wanted `same` *and* `match!` to print only when the numbers are equal. With `a = 3`, `b = 4` it prints `match!`. Why, and what is the one-keystroke fix?

```
1 a = 3
2 b = 4
3 if a == b:
4     print("same")
5 print("match!")
```

Why: _____

Fix: _____

**F7.** For `n = 30` this should print `FizzBuzz` (30 is a multiple of both 3 and 5). It prints something else, and one branch can *never* run. Which branch, what prints, and what is the fix?

```
1 n = 30
2 if n % 3 == 0:
3     print("Fizz")
4 elif n % 5 == 0:
5     print("Buzz")
6 elif n % 3 == 0 and n % 5 == 0:
7     print("FizzBuzz")
8 else:
9     print(n)
```

It prints: _____

Dead branch: line _____

Fix: _____

## Part G • Write the Header

TRANSLATE ≈ 3 min

Write only the `if` line (with its colon) for each situation. Assume `hour` is a whole number from 0 to 23, `day` is a string such as `"Mon"`, and `balance` is a number.

**G1.** "the shop is open: from 9 o'clock up to, but not including, 21"

`if` _____:

**G2.** "it is the weekend" (the day is `"Sat"` or `"Sun"`)

`if` _____:

**G3.** "the balance is zero or below"

`if` _____:

**G4.** "it is *not* the lunch break, which runs from 13 to 14 inclusive"

```
if _____:
```

## Part H • One Sentence Each

EXPLAIN ≈ 3 min

**H1.** If you can always write several `if`s, what does `elif` buy you?

**H2.** A friend writes `if answer == "yes" or "y":` and says it accepts both spellings. Why does it accept *everything*?

**H3.** A two-branch program prints `positive` when `x > 0` and `negative` otherwise. For `x = 0` it says `negative`. Is that right, and how would you fix the design?

**H4.** `if h >= 120 and a >= 8:` and a nested pair of `if`s accept exactly the same people. When is the nested version worth its extra lines?

## Part I • Write the Code

STRETCH ≈ 10 min, optional

Everything so far ran *forwards*: here is code, what does it do? These run *backwards*: here is the behaviour — **you** write the program. Each is a few lines. Mind your colons and your indentation. On paper, as always.

**I1.** Ask the user for a whole number. Print `multiple of 7` or `not a multiple of 7`.

**I2.** `marks` already holds a number. Print `A` for 85 and above, `B` for 70 to 84, `C` for 50 to 69, and `F` below 50. Use one chain.

**I3.** A museum ticket: free if `age` is under 5 *or* 60 and over; otherwise half price if `student` is `"yes"`; otherwise full price. Print `free`, `half` or `full`.

**I4.** Two numbers `p` and `q` are already bound. Without using `min()`, print the smaller one; if they are equal, print `same`.

**I5.** For a number `n`: print `both` if it is a multiple of 4 *and* 6, `four` if only of 4, `six` if only of 6, otherwise the number itself.

### Console Quests

tonight · 5 min · at a computer

*When unsure, just try it in the console.* These three are deliberately **not** answered anywhere on this sheet — the console is the answer key. Predict first, then type.

1. Type `"a" == "b" or "Sun"` and look at what comes back — it is not `True` or `False`. Then `"a" == "a" or "Sun"`. Now explain H2 to yourself.
2. Type each header on one line with a body after the colon, e.g. `if "no": print("ran")`. Try `"no"`, `0.0`, `-1`, `None`, `" "` and `""` as the condition. Which ones print?
3. Set `x = "5"` and try `if x > 3: print("big")`. Then `if int(x) > 3: print("big")`. Then `if x == 5: print("five")` — no error this time, but does it print?

**Stuck on more than two items?** Re-read the Lecture 5 slides on `elif` ("first True wins") and on indentation before the next class — then redo Parts B and C cold. Loops, next week, are built on exactly this skill.

## Answer Key

Check your work *after* attempting everything. If an answer surprises you, that item is worth re-reading in the slides. Console Quests have no key — the console is the key.

**A1.** `if`, `elif`, `else`; it is `else` that takes no condition — it is the "otherwise" branch, so `else x < 5:` is a `SyntaxError`.

**A2.** **indentation** (4 spaces, consistently); a **colon** `:`. Forget the colon and you get `SyntaxError`; forget the indent and you get `IndentationError`.

**A3.** **Exactly one** body runs in a chain that ends in `else` (at most one if there is no `else`). Three separate `if`s are tested independently, so anywhere from **0** to **3** bodies may run.

**A4.** `pass`. An `if` line followed by nothing indented is a `SyntaxError`, so `pass` holds the place.

**A5.** (a) **True** — if the condition is `False`, the whole body is skipped. (b) **False** — `elif` only makes sense as a continuation of an `if`. (c) **True** — indentation is what ties a line to the `if`; un-indented means "after the whole thing". (d) **False** — first `True` wins, the rest are never looked at. (e) **True** — a chained comparison, and it reads like maths. (f) **False** — any expression works; Python uses its truthiness (non-zero, non-empty = `True`).

**B1.** Just `Drive safe`. `60 > 60` is `False` (strictly greater), so *both* indented lines are skipped; the last line is outside the `if` and always runs.

**B2.** `go` then `end`. `"Red"` and `"red"` are different strings, so the `else` branch runs — exactly one of the two branches always does.

**B3.** `none`. The condition is just `count`, and 0 is falsy, so the `else` runs. With `count = 3` it would print `some`.

**B4.** `h = 12`: `afternoon` — `12 < 12` is `False`, `12 < 17` is `True`, stop. `h = 21`: `night` — `21 < 21` fails too, so all three tests fail and `else` runs. `h = 5`: `morning` — the very first test passes and nothing else is checked. The boundaries (12, 21) fall to the *next* branch because `<` is strict.

**B5.** `six`, `two`, `bye`. 24 is even, so we go inside; 24 is also a multiple of 3, so `six` prints. `print("two")` is indented one level — inside the outer `if` but *outside* the inner one — so it prints for any even `n`. `bye` is outside everything. With `n = 8` you would get `two`, `bye`.

**B6.** `7`. The first `if` doubles `x` to 8; the second `if` is tested *afterwards*, sees `8 > 6`, and subtracts 1. With `elif` the second test is skipped once the first has run, so `x` stays `8`. Separate `if`s are independent steps; `elif` makes them alternatives.

**B7.** **Two** bodies run (`72 >= 40` and `72 >= 70`); each one *overwrites* `grade`, so the last winner stands and it prints `merit`. With `elif` the chain stops at the first `True`, so it prints `pass` — which is wrong for 72. Lowest-first only works with separate `if`s; an `elif` chain needs the *strictest* test first.

**B8.** `Q` then `S`. `5 > 3` takes us inside; `12 > 15` is `False`, so the *inner* `else` prints `Q`. The `else` on line 6 is indented four spaces, so it pairs with the `if` on line 4 at the same depth; the one on line 8 pairs with line 3. `R` could only print if `a > 3` were `False`. `S` is outside everything.

**C.** `12`: `Free` then `Next!`. `25`: `Fee: 2500` then `Next!` ($5 \times 500$). `30`: `Fee: 5000` then `Next!` — the boundary case: `30 > 30` is `False`, so the bag is *not* refused; it falls to the `elif`, where `30 > 20` is `True`. `45`: `Refused` then `Next!`. `Next!` appears every time because it is not indented. Note the comma in `print("Fee:", fee)` inserts the space.

**C1.** `B2`: **one** question; prints `Unknown code` then `Bye` — the outer test fails, so line 3 never runs and no PIN is ever asked for. `A1`, `1234`: **two** questions; `Wrong PIN` then `Bye`. `A1`, `9090`: **two** questions; `Open` then `Bye`. A skipped body skips *everything* in it, including an `input()` — so the program itself behaves differently, not just its output.

**D.** **Runs:** `n % 3 == 0` (6 is a multiple of 3); `word` (a non-empty string is truthy); `5 < n < 7` (6 sits strictly between).
**Skipped:** `len(word) > 2` (2 is not greater than 2); `n > 5 and word == "Hi"` (`"hi"` is not `"Hi"`, so the `and` fails); `not n < 10` (`6 < 10` is `True`, and `not` flips it); `not word` (`"hi"` is truthy, so `not` makes it `False` — this header only runs for an *empty* string).
**Error:** `n = 6` (single `=` in a condition is a `SyntaxError` — the program never even starts); `word + 1` (`str + int` is a `TypeError` the moment that line runs); `word > 1` (Python refuses to *order* a string against a number — also a `TypeError`; cast first if the string holds digits).

**E.** Indent lines **3, 4 and 6** — and nothing else. Line 5 (`else:`) must line up with its `if`; line 7 must stay un-indented so that `goodbye` prints in every case. With `age = 16` the output is `minor` then `goodbye`.

Had line 7 been indented under `else`, an adult would never be told goodbye.

**F1.** It prints `200`: `80 > 10` is already `True`, so the chain stops there and the `km > 50` test is never reached. Fix: put the strictest test first — `if km > 50: ... elif km > 10: ....` In a chain, order *is* logic.

**F2.** `22 > 25` is `False`, so the body never runs and `label` is never bound — then line 4 asks for it. Add an `else:` branch with `label = "cool"` (or bind `label` to a default *before* the `if`). Every path must give the name a value before it is used.

**F3.** `else:` — `else` never takes a condition. (`elif n % 2 == 1:` would also be accepted, but here the plain `else` is right: whatever is not even is odd.)

**F4.** Line 4, `else:`, must line up with its `if` on line 2 (no indent), and line 5 moves left with it to a single indent. An `else` belongs to the `if` at the *same* indentation level — indented as it is, Python finds an `else` with no `if` and reports a `SyntaxError`.

**F5.** `IndentationError: unexpected indent`. Every line of one body must start in the *same* column; six spaces after a four-space line means nothing to Python. Either indent it exactly like line 3 (then it is inside the `if`) or not at all (then it runs after the `if`, every time). Which one is right depends on what you meant — but "a bit more" indentation is never an option.

**F6.** Line 5 is not indented, so it is *after* the `if`, not inside it, and runs no matter what. Indent it by four spaces to make it part of the body. Indentation is the only thing that tells Python which lines a condition guards.

**F7.** It prints `Fizz`: `30 % 3 == 0` is `True` first, and the chain stops. The branch on line 6 is **dead** — any `n` that passes its `and` has already passed line 2. Fix: move the "both" test to the *top* (`if n % 3 == 0 and n % 5 == 0:` first, then the single tests). The most *specific* condition goes first, because a general one placed earlier swallows it.

**G1.** `if 9 <= hour < 21:` (or `hour >= 9 and hour < 21`). "Up to but not including" is the strict `<`.

**G2.** `if day == "Sat" or day == "Sun":` — each side of the `or` needs its own complete comparison. `day == "Sat" or "Sun"` is *always* true; see H2.

**G3.** `if balance <= 0:` — one operator says both "below" and "exactly zero".

**G4.** `if not (13 <= hour <= 14):` or, without `not`, `if hour < 13 or hour > 14:`. The brackets after `not` matter — without them `not` would apply to `13` alone.

**H1.** A guarantee that *at most one* branch runs and that testing stops at the first `True` — separate `if`s are all tested and several can run, overwriting each other (as in B6).

**H2.** Because the `or` joins `answer == "yes"` with the bare string `"y"`, and a non-empty string is always truthy — so the whole condition is `True` whatever `answer` holds. It must be `answer == "yes" or answer == "y"`.

**H3.** No — zero is neither. Two branches can only split the world in two; add a middle branch, `elif x == 0: print("zero")`, so all three cases get their own answer.

**H4.** When each failure needs its *own* message or action ("too short" versus "too young"): the flat `and` only knows that the test failed, not which half failed. If one message covers both, use `and`.

**I1.** `n = int(input("Number? "))` / `if n % 7 == 0:` / `print("multiple of 7")` / `else:` / `print("not a multiple of 7")`. Two outcomes, so `if`/`else` — never two `if`s for a yes/no question.

**I2.** `if marks >= 85: print("A")` / `elif marks >= 70: print("B")` / `elif marks >= 50: print("C")` / `else: print("F")` (each print indented under its header). Highest band first, so every `elif` only needs a lower bound — the upper bound is already guaranteed by the test above having failed. Check the edges: 85 gives A, 84 gives B, 70 gives B, 69 gives C, 50 gives C, 49 gives F.

**I3.** `if age < 5 or age >= 60: print("free")` / `elif student == "yes": print("half")` / `else: print("full")`. The `or` needs `age` on *both* sides; and because it is a chain, a 70-year-old student is `free`, not `half` — first True wins.

**I4.** `if p < q: print(p)` / `elif q < p: print(q)` / `else: print("same")`. Three outcomes, three branches; the `else` catches equality without needing `p == q` spelled out. (Two `<=` tests would print a number for equal inputs — read the spec.)

**I5.** `if n % 4 == 0 and n % 6 == 0: print("both")` / `elif n % 4 == 0: print("four")` / `elif n % 6 == 0: print("six")` / `else: print(n)`. The `and` test *must* come first — put it later and 24 prints `four` and stops (see F7). Test yourself with 24, 8, 18 and 7.