Finger Exercises 20

Codes & Codebreakers

≈ 70 min + stretch

Comp 102 — Programming Fundamentals

What this is. A workout for Lecture 20. You will lock and unlock words by hand, run the lecture's functions in your head, fix broken ones, and open a locked sentence with no key, the way Al-Kindi did.

How to do it. On **paper**, **no computer**, in one sitting. If an item takes you more than a minute, skip it and come back. Parts A–P are for *everyone*; Part Q is a **stretch**. The Console Quests are for tonight, *at* a computer. The answer key is on the last pages — do not peek until you are done.

Predict first, then check. Write your answer *before* you type anything. A wrong guess teaches you more than an answer you only watched the computer give.

Your tool. The bottom of every page has the **letter ruler**: each letter and its place in `ABC`. Use it for every shift. Everywhere on this sheet, `ABC = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`.

Part A • Do You Remember?

RECALL ≈ 4 min

A1. In `ABC`, the letter `A` is at place _____ and `Z` is at place _____.

A2. `ABC.index(ch)` gives the _____ of `ch`. `ABC[i]` gives the _____ at place `i`.

A3. `shift(msg, k)` locks with key `k`. The call `shift(secret, _____)` unlocks it.

A4. Caesar's shift has _____ keys. The mixed alphabet has _____ keys (write it with a `!`).

A5. Al-Kindi's idea: in a _____ secret, the most common letter usually stands for _____. The next one usually stands for _____.

A6. Circle **True** or **False**:

(a) `shift` moves spaces and commas too. True / False

(b) `shift` changes small letters such as `a`. True / False

(c) Key 26 locks exactly like key 0. True / False

(d) `n % 26` is always 0 to 25, even when `n` is negative. True / False

(e) Mixing the alphabet hides the letters but not their counts. True / False

(f) `count_letters` reads each letter once, so it is $O(n)$. True / False

(g) With many shifts, one plain letter always becomes the same secret letter. True / False

(h) `crack_caesar` works on any secret, even a five-letter one. True / False

Part B • Letters Are Places

TRACE ≈ 4 min

Write the value. If the line stops with an error, write the error's name.

Expression	Value or error	Expression	Value or error
`ABC.index('A')`	_____	`ABC.index('a')`	_____
`ABC.index('Z')`	_____	`'a' in ABC`	_____
`ABC[25]`	_____	`ABC.index(' ')`	_____
`ABC[26]`	_____	`ABC[ABC.index('M') + 1]`	_____
`ABC[-1]`	_____	`ABC[ABC.index('Y') + 2]`	_____
`len(ABC)`	_____	`ABC[(ABC.index('Y') + 2) % 26]`	_____

Part C • Round the Wheel

TRACE ≈ 3 min

C1. Write the value of each line.

`28 % 26` _____ `26 % 26` _____ `52 % 26` _____ `53 % 26` _____ `0 % 26` _____

`-1 % 26` _____ `-3 % 26` _____ `-26 % 26` _____ `-30 % 26` _____ `100 % 26` _____

C2. Circle every key that locks *exactly* like key 3: `29` `-3` `23` `-23` `55` `81`

C3. Unlocking key 3 with `shift(secret, -3)` is the same as *locking* with key _____.

Part D • Lock and Unlock by Hand

TRACE ≈ 6 min

Write what each call returns, one letter in each empty box. A space stays a gap.

D1. `shift('DOG', 3)`

D	O	G

D2. `shift('XYZ', 3)`

X	Y	Z

D3. `shift('HI, SANA!', 1)`

H	I	,	S	A	N	A	!

D4. `shift('Pakistan', 2)`

P	a	k	i	s	t	a	n

D5. `shift('ZOO', 0)`

Z	O	O

D6. `shift('ZOO', 26)`

Z	O	O

D7. `shift('ABC', -1)`

A	B	C

D8. `shift('MANGO', 29)`

M	A	N	G	O

D9. `shift('GRJ', -3)`

G	R	J

D10. `shift('2026', 4)`

2	0	2	6

D11. `shift('', 5)` returns _____

`shift(shift('PYTHON', 13), 13)` returns _____

Why? _____

Part E • Inside `shift`, Pass by Pass

TRACE ≈ 4 min

```
1 def shift(msg, k):
2     out = ''
3     for ch in msg:
4         if ch in ABC:
5             i = ABC.index(ch)
6             out += ABC[(i + k) % 26]
7         else:
8             out += ch
9     return out
```

Fill in one row for each pass of the loop in `shift('Y Za', 3)`. Write — where a line does not run.

ch	in ABC?	i	(i+k)%26	out after
'Y'	_____	___	_____	_____
' '	_____	___	_____	_____
'Z'	_____	___	_____	_____
'a'	_____	___	_____	_____

The call returns _____.

Part F • Bug Hunt

FIX ≈ 8 min

Each version of `shift` below has **one** changed line; every other line is the same as in Part E. Write the line number and the fix.

F1. Line 6 now reads `out += ABC[i + k % 26]`
`shift('WAX', 3)` crashes with `IndexError`. "But it has `% 26`!" Why does it still crash?
_____ Fix: _____

F2. Line 6 now reads `out = ABC[(i + k) % 26]`
`shift('CAT', 1)` returns _____. Fix: _____

F3. Line 9, `return out`, is now indented one level more, so it is inside the loop.
`shift('CAT', 1)` returns _____. `shift(' CAT', 1)` returns _____.

F4. Lines 7 and 8 (`else:` and `out += ch`) were deleted.
`shift('HI MOM', 3)` returns _____. Which calls still work? _____

F5. Line 6 now reads `out += ABC[(i + k) % 25]`
Tick the calls that still give the **right** answer:
☐ `shift('CAT', 1)` ☐ `shift('XYZ', 1)` ☐ `shift('BED', 2)`
☐ `shift('Y', 0)` ☐ `shift('X', 1)`

F6. The lecture's first try had line 6 as `out += ABC[i + k]` (no `%` at all).
Tick the calls that **crash**:
☐ `shift('HELLO', 3)` ☐ `shift('X Y', 2)` ☐ `shift('DWWDFN', -3)`
☐ `shift('ABC', -3)` ☐ `shift('AB', 30)`

Part G • Try Every Key

TRACE ≈ 4 min

A spy found the secret `'FKHHU'`. The loop `for k in range(26): print(k, shift('FKHHU', -k))` prints:

0	FKHHU	6	ZEBBO	12	TYVVI	18	NSPPC	24	HMJJW
1	EJGGT	7	YDAAN	13	SXUUH	19	MROOB	25	GLIIV
2	DIFFS	8	_____	14	RWTTG	20	_____		
3	_____	9	WBYYL	15	QVSSF	21	KPMMZ		
4	BGDDQ	10	VAXXK	16	PURRE	22	_____		
5	AFCCP	11	UZWWJ	17	OTQQD	23	INKKX		

G1. Fill in the four empty rows.

G2. Two rows are English words. Which keys? _____
So what was the message? _____

G3. How many lines does the loop print? _____ Which row is just the secret, unchanged? _____

G4. Bilal forgot the minus: `shift('FKHHU', k)`.
Does `CHEER` still show up? _____ At which `k`? _____

Part H • Mix the Alphabet

TRACE ≈ 7 min

Recall `substitute(msg, key)`: a letter at place `i` in `ABC` becomes `key[i]`. `mystery(secret, key)` goes the other way: a letter at place `i` in `key` becomes `ABC[i]`. Other characters are copied in both.

letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
K	Z	E	B	R	A	S	C	D	F	G	H	I	J	K	L	M	N	O	P	Q	T	U	V	W	X	Y

letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
REV	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

H1. `substitute('BAD', K)` _____ `substitute('FACE IT!', K)` _____

H2. `mystery('QDA', K)` _____ `mystery('ZEBRAS', K)` _____

H3. `substitute('WIZARD', REV)` _____ `mystery('DRAZIW', REV)` _____
What is special about the key `REV`? _____

H4. `substitute('HELLO', ABC)` returns _____. Why? _____

H5. Write the key that makes `substitute(msg, key)` do the same as `shift(msg, 3)`:
`key = '`_____`'`
The same key in Python, with two slices: `ABC[____] + ABC[____]`

H6. Two keys typed by mistake. What does each line give?
`BAD = 'QQERTYUIOPASDFGHJKLZXCVBNM'` (a second `Q` where `W` should be)
`SHORT = 'QWERTYUIOPASDFGHJKLZXCVBN'` (25 letters: no `M`)
`substitute('AB', BAD)` _____ `substitute('FOOD', SHORT)` _____
`mystery('QQ', BAD)` _____ `substitute('ZOO', SHORT)` _____
What makes a good key? _____

H7. Each line below takes the place of `out += key[ABC.index(ch)]` in `substitute`. For each one, write what the function now does (**lock**, **unlock** or **nothing**) and what it returns for `('BAD', K)`.

	does	*returns*
(a) `out += key[ABC.index(ch)]`	_____	_____
(b) `out += ABC[key.index(ch)]`	_____	_____
(c) `out += ABC[ABC.index(ch)]`	_____	_____
(d) `out += key[key.index(ch)]`	_____	_____

Part I • How Many Keys?

COUNT ≈ 4 min

I1. An alphabet of only three letters, `A`, `B`, `C`. Write every mixed key: `ABC`, _____, _____, _____, _____, _____. How many? _____ Which one locks nothing? _____

I2. With 4 letters: _____ keys. With 5 letters: _____ keys. With 26 letters: _____.

I3. A very fast computer tries 10^9 keys a second. Use $26! \approx 4 \times 10^{26}$ and 1 year $\approx 3 \times 10^7$ s.
Seconds: _____ Years: _____ Compare with the age of the universe (13.8 billion years): _____

I4. With many shifts, a key word of 5 letters has $26^5 = 11{,}881{,}376$ choices. At 10 million a second, trying them all takes about _____. What does that tell you? _____

Part J • Count the Letters

TRACE ≈ 5 min

```
1 def count_letters(text):
2     counts = {}
3     for ch in text:
4         if ch in ABC:
5             if ch in counts:
6                 counts[ch] += 1
7             else:
8                 counts[ch] = 1
9     return counts
```

J1. Write `counts` after each pass of `count_letters('A BAD BAG')`. Keep the keys in the order they were added. Write "same" if nothing changed.

`'A'` _____

`' '` _____

`'B'` _____

`'A'` _____

`'D'` _____

. . . and after the whole text:

_____

J2. What does each call return?

`count_letters('Hello, WORLD')` _____

`count_letters('123 !?')` _____ `count_letters('A BAD bag')` _____

J3. `c = count_letters('BANANA')`. Write the value or the error.

`c` _____ `c['A']` _____ `c['Z']` _____

`'Z' in c` _____ `len(c)` _____ `sum(c.values())` _____

J4. Three broken copies of `count_letters`. What does each one return for `'SEE'`? Write the dictionary or the error.

(a) Lines 5–8 are replaced by `counts[ch] += 1` _____

(b) Lines 5–8 are replaced by `counts[ch] = 1` _____

(c) Line 2 (`counts = {}`) is moved inside the loop, after line 3 _____

Part K • Most Common First

TRACE ≈ 4 min

```
1 def by_count(counts):
2     return sorted(counts,
3         key=lambda ch: counts[ch],
4         reverse=True)
```

`d = {'X': 2, 'K': 5, 'Q': 1}`

When two letters have the same count, `sorted` keeps them in the order they had in the dictionary.

Write what each line returns.

`by_count(d)` _____

`sorted(d)` _____

`sorted(d, key=lambda ch: d[ch])` _____

`by_count(count_letters('A BAD BAG'))` _____

`by_count({'T': 2, 'O': 2, 'E': 2})` _____

`by_count(d)[-1]` _____

`by_count({})` and `by_count({})[0]` _____

Part L • Crack It by Counting

TRACE ≈ 5 min

```
1 def crack_caesar(secret):
2     top = by_count(count_letters(secret))[0]
3     k = (ABC.index(top) - ABC.index('E')) % 26
4     return shift(secret, -k)
```

L1. Each row is a long secret, locked with a shift. Its most common letter is given. Find the key, then unlock the word.

most common	key `k`	locked word	unlocked
H	_____	WKLV	_____
E	_____	CODE	_____
I	_____	WYR	_____
X	_____	MBFX	_____
B	_____	TXQBO	_____

L2. A short secret: `crack_caesar('DWWDFN DW GDZQ')`. It was locked with key 3.
Most common letter: _____ `k` = _____ Returns: _____
Why does it go wrong? _____

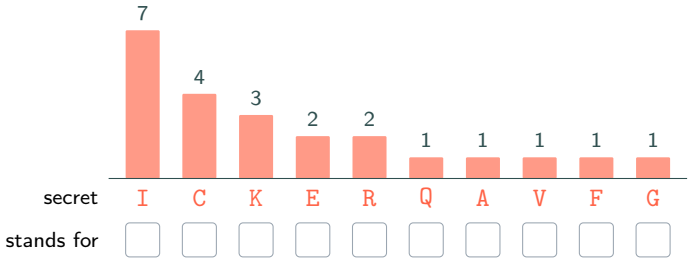
L3. What does `crack_caesar('')` do? _____

Part M • Al-Kindi by Hand

DECODE ≈ 6 min

A short sentence was locked with a mixed key. You do **not** have the key.

CEI RIQKIC AR VFGIK CEI CKII



M1. The tallest bar is `I`. It most likely stands for ____. The next, `C`, most likely stands for ____.

M2. The word `CEI` comes twice. With your two letters it reads `T?E`. The word is _____, so `E` stands for ____.

M3. `CKII` is now `T?EE`. So `K` stands for ____.

M4. Write the four letters you know under the sentence below, then guess the rest of the words:

C	E	I		R	I	Q	K	I	C		A	R		V	F	G	I	K		C	E	I		C	K	I	I

Part N • Many Shifts

TRACE ≈ 6 min

```
1  def many_shifts(msg, word):
2      out = ''
3      n = 0
4      for ch in msg:
5          if ch in ABC:
6              k = ABC.index(word[n % len(word)])
7              out += shift(ch, k)
8              n += 1
9          else:
10             out += ch
11     return out
```

Letter number `n` of the message is shifted by letter number `n` of the key word (`A` = 0, `B` = 1, ...). When the word runs out, it starts again.

Line 8 adds 1 only for a *letter*.

N1. Trace `many_shifts('NO NO', 'CAB')`, one row a pass.

ch	n at the start	word[n % 3]	k	new character	out after
'N'	0	____	____	____	_____
'O'	___	____	____	____	_____
' '	___	___	___	____	_____
'N'	___	____	____	____	_____
'O'	___	____	____	____	_____

N2. Write what each call returns.

`many_shifts('EEEE', 'AB')` _____ `many_shifts('HELLO', 'A')` _____

`many_shifts('HELLO', 'D')` _____ `many_shifts('AB', 'BA')` _____

`many_shifts('HI', 'LEMON')` _____

Part O • Finish the Function

FILL IN ≈ 4 min

Fill in every blank. You may call the lecture's functions.

O1.

```
1  def unshift(secret, k):
2      return shift(secret, ____)
```

```
1  def key_from_top(top):
2      return (ABC.index(____)
3              - ABC.index(____)) % ____
```

O2. A key is good when it has 26 characters and all 26 capital letters appear in it (Part H6). Finish it with `count_letters`.

```
1  def is_good_key(key):
2      return len(key) == 26 and len(_________________) == ____
```

O3. The key that does Caesar's shift (Part H5), for any `k` from 0 to 25:

```
1  def caesar_key(k):
2      return ABC[______] + ABC[______]
```

Part P • One Sentence Each

EXPLAIN ≈ 3 min

P1. Why does `% 26` stop the `IndexError` in `shift`?

P2. The mixed alphabet has 26! keys, Caesar only 26. Al-Kindi broke the mixed alphabet anyway. What is the lesson?

P3. Why does counting fail against many shifts?

Part Q • Write the Code

STRETCH ≈ 10 min, optional

Q1. Write `flip(msg)`: `A` becomes `Z`, `B` becomes `Y`, . . . , `Z` becomes `A`. It does the same as `substitute(msg, REV)` from Part H, but with **no key**: work out the new place from `i`. `flip('WIZARD')` is `'DRAZIW'`. Other characters stay.

Q2. Write `unlock_many(secret, word)`, which undoes `many_shifts`.
For example, `unlock_many('PO OQ', 'CAB')` is `'NO NO'`. Change as few lines of `many_shifts` as you can.

Console Quests

tonight · 5 min · at a computer

When unsure, just try it in the console. These are deliberately **not** answered anywhere on this sheet — the console is the answer key. Predict first, then type. Start from `lec20.py`.

1. Type `shift('Hello', 3)`. Then try `shift('Hello'.upper(), 3)`. (`.upper()` is the twin of `.lower()` from Lecture 17.)
2. Run the try-every-key loop on `'FKHHU'`. Then lock a whole sentence of your own with key 7 and run the loop on it. How many rows look like English now?
3. Paste a long English paragraph into a string, in capitals. Is `by_count(count_letters(text))[:5]` close to E, T, A, O, I? Then try `crack_caesar(shift(text, 11))`.

Stuck on more than two items? Re-read the Lecture 20 slides "Letters Are Places", "% Goes Round the Wheel", "Mixing in Python", "Counting in Python" and "One Count, One Key" before the next class — then redo Parts D, F and M cold.

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. **0** and **25**. There are 26 letters, so the places are 0 to 25.

A2. the **place** (a number); the **letter**. The two undo each other: `ABC[ABC.index('Q')]` is `'Q'`.

A3. `-k`: move every letter back by the same amount.

A4. **26** (and key 0 changes nothing, so 25 useful ones); **26!**, about 4×10^{26}.

A5. **long**; **E**; **T** (English order: E, T, A, O, I, N, . . .).

A6. (a) **False** — only letters in `ABC` move; the `else` branch copies everything else. (b) **False** — `'a'` is not in `ABC`, so it is copied as it is (see D4). (c) **True** — $(i + 26) \bmod 26 = i$: once round the wheel. (d) **True** — `-1 % 26` is 25. (e) **True** — that is the whole lecture. (f) **True** — one pass over the text. (g) **False** — each letter gets the shift of its own key-word letter: the five Es of `SEE THE TREE` became five different letters. (h) **False** — a short secret may not have E as its most common letter (see L2).

B. Left: `0`; `25`; `'Z'`; **IndexError** (places stop at 25); `'Z'` (a negative place counts from the end); `26`. Right: **ValueError** (`index` cannot find a small `a`); `False`; **ValueError** (a space is not in `ABC`: that is why `shift` asks `if ch in ABC` first); `'N'`; **IndexError** ($24 + 2 = 26$); `'A'` ($26 \bmod 26 = 0$: round the wheel).

C1. Top: **2**, **0**, **0**, **1**, **0**. Bottom: **25**, **23**, **0**, **22**, **22** ($100 = 3 \times 26 + 22$). With a negative number, Python still gives 0 to 25: $-30 = -2 \times 26 + 22$.

C2. **29**, **-23**, **55**, **81**: each one `% 26` is 3. Key `-3` is `-3 % 26` = 23, the *unlock* for key 3; key 23 is the same as `-3`.

C3. **23**. Going back 3 places or forward 23 places ends on the same letter, because $-3 \bmod 26 = 23$.

D. D1 `'GRJ'`. D2 `'ABC'` (round the wheel). D3 `'IJ, TBOB!'` (the comma, space and `!` stay). D4 `'Rakistan'`: only the capital `P` is in `ABC`; small letters are copied as they are. D5 and D6 `'ZOO'`: key 0 and key 26 change nothing. D7 `'ZAB'` ($-1 \bmod 26 = 25$). D8 `'PDQJR'`: key 29 is key 3. D9 `'DOG'`: it undoes D1. D10 `'2026'`: digits are not letters. D11 `''`, the empty string: the loop never runs; and `'PYTHON'`: $13 + 13 = 26$, once round the wheel (key 13 is its own unlock).

E. `'Y'`: True, 24, 1, `'B'`. `' '`: False, —, —, `'B '`. `'Z'`: True, 25, 2, `'B C'`. `'a'`: False, —, —, `'B Ca'`. It returns `'B Ca'`. A space and a small letter both go to line 8.

F1. `%` is done *before* `+`, like `*`. So it is `i + (k % 26)` $= 23 + 3 = 26$ for `X`: no place 26. (`W` and `A` worked.) Fix: `ABC[(i + k) % 26]`.

F2. `'U'`: each letter *replaces* `out`, so only the last one is left. Fix: `+=`.

F3. `'D'`: the function returns after the first pass. `'  '` (one space): the first character is a space, copied, then returned. Fix: move `return out` back under `for`.

F4. `'KLPRP'`: the space is lost. It still works on any message with *only capital letters*, such as `'DOG'`. Fix: put the `else` back.

F5. `'CAT'` (DBU) ✓; `'XYZ'` gives `'YAB'`, not `'YZA'` ✗; `'BED'` (DGF) ✓; `'Y'` gives `'Y'` ✓; `'X'` gives `'Y'` ✓. Only a sum of 25 or more goes wrong (here `Y`+1 and `Z`+1), so short tests with early letters never see it. `Z` can never come out. Fix: `% 26`.

F6. Crash: `'X Y'` (`Y`: $24 + 2 = 26$) and `'AB'`, 30 ($0 + 30$). No crash: `'HELLO'` (KHOOR), `'DWWDFN'`, −3 (ATTACK) — and `'ABC'`, −3 gives the *right* answer `'XYZ'`: `ABC[-3]` is `'X'`, because a negative place counts from the end. Unlocking hides the bug; only locking past `Z` shows it.

G1. 3 `CHEER`; 8 `XCZZM`; 20 `LQNNA`; 22 `JOLLY`.

G2. Keys **3** (CHEER) and **22** (JOLLY). We **cannot tell**: with only five letters, two keys both give words. A longer secret leaves only one row that makes sense.

G3. **26** (`range(26)` is 0 to 25). Row **0**: `shift(secret, 0)` changes nothing.

G4. **Yes**, at **k = 23**: forward 23 is the same as back 3. (JOLLY shows up at k = 4.) Trying all 26 keys finds every row in either direction.

H1. `'EZR'`; `'SZBA FQ!'`.

H2. `'THE'` (Q is at place 19 of `K`, which is T in `ABC`). `'ABCDEF'`: the first six letters of the key.

H3. `'DRAZIW'`; `'WIZARD'`. With `REV`, A ↔ Z, B ↔ Y, ...: locking twice gives the message back, so `substitute` with `REV` is also its own unlock.

H4. `'HELLO'`: each letter becomes the letter at its own place. The key `ABC` is not mixed at all.

H5. `'DEFGHIJKLMNOPQRSTUVWXYZABC'`, which is `ABC[3:] + ABC[:3]`. Caesar's shift is one of the 26! mixed keys.

H6. `'QQ'`; `'AA'` — A and B both lock to Q, so the unlock cannot know which one it was (`index` finds the first Q). `'YGGR'`; **IndexError**: Z is place 25 and `SHORT` has no place 25. A good key has all **26** letters, each **once**.

H7. (a) lock, `'EZR'` (H1). (b) unlock, the line of `mystery`: `'CEH'` (B is at place 2 of `K`, and `ABC[2]` is C). (c) and (d) nothing, `'BAD'`: finding a letter's place in a string and then reading the *same* string at that place gives the letter back. To lock or unlock, the two strings must differ.

I1. `ABC, ACB, BAC, BCA, CAB, CBA`: **6** = 3!. `ABC` changes nothing.

I2. 4! = **24**; 5! = **120**; **26!**. Each new letter multiplies the count.

I3. $4 \times 10^{26} \div 10^9 = 4 \times 10^{17}$ s; $\div 3 \times 10^7 \approx 1.3 \times 10^{10}$, about **13 billion years**: about the age of the universe, even 100 times faster than the lecture's computer.

I4. About **1.2 seconds**. A short key word can be found by brute force; many shifts need a **long** key word.

J1. `{'A': 1}`; same; `{'A': 1, 'B': 1}`; `{'A': 2, 'B': 1}`; `{'A': 2, 'B': 1, 'D': 1}`. At the end: `{'A': 3, 'B': 2, 'D': 1, 'G': 1}`.

J2. `{'H': 1, 'W': 1, 'O': 1, 'R': 1, 'L': 1, 'D': 1}`: the small letters of `ello` are skipped. `{}`: no capital letters at all. `{'A': 2, 'B': 1, 'D': 1}`: `bag` is skipped.

J3. `{'B': 1, 'A': 3, 'N': 2}`; `3`; **KeyError** (a letter that never appeared is not a key: there is no 0 in the dictionary). `False`; `3` (three different letters); `6` (all the letters).

J4. (a) **KeyError**: the first `'S'` is not a key yet, so there is nothing to add 1 to. (b) `{'S': 1, 'E': 1}`: every count is 1. (c) `{'E': 1}`: the dictionary is emptied on every pass, so only the last letter is left. The `if`/`else` of lines 5–8 is the counting pattern of Lecture 17: new key gets 1, old key gets +1.

K. `['K', 'X', 'Q']`; `['K', 'Q', 'X']` (no `key=`: the *letters* in A–Z order); `['Q', 'X', 'K']` (no `reverse`: least common first); `['A', 'B', 'D', 'G']` (D and G tie at 1 and keep their order); `['T', 'O', 'E']` (all tie); `'Q'`, the least common; `[]` and **IndexError**: an empty list has no place 0.

L1. H: **3**, `THIS`. E: **0**, `CODE` (not locked at all). I: **4**, `SUN`. X: **19**, `TIME`. B: $(1-4) \bmod 26 =$ **23** (B comes *before* E, so the `% 26` goes round), `WATER`.

L2. `D` (4 times), so $k = (3-4) \bmod 26 = 25$, and it returns `'EXXEGO EX HEAR'` — nonsense. The message, `ATTACK AT DAWN`, is too short: its most common letter is A, not E.

L3. **IndexError**: `count_letters` gives `{}`, `by_count` gives `[]`, and `[0]` has nothing to give.

M1. `I` → **E**; `C` → **T**. Note the trap: the secret letter `E` is *not* the plain E.

M2. **THE**, so the secret `E` stands for **H**.

M3. `TREE`, so `K` → **R**.

M4. `THE SECRET IS UNDER THE TREE`. After C, E, I, K the middle reads `?E?RET ??  ???ER`: SECRET gives R → S and Q → C; then `AR` is IS and `VFGIK` is UNDER. The counts gave three letters, the words gave the rest. No key was tried.

N1. N: 0, C, 2, `P`, `'P'`. O: 1, A, 0, `O`, `'PO'`. Space: n stays **2**, copied, `'PO '`. N: 2, B, 1, `O`, `'PO O'`. O: 3, C (3 mod 3 = 0), 2, `Q`, `'PO OQ'`. The two NOs lock to different letters.

N2. `'EFEF'`: one letter, two secret letters. `'HELLO'`: A means shift 0. `'KHOOR'`: a one-letter key word is just Caesar's shift (D = 3). `'BB'`: two *different* letters lock to the *same* one — counts get mixed up both ways. `'SM'`: a key word longer than the message is fine (H + 11, I + 4).

O1. `-k`. Then `top`, `'E'`, `26`: `key_from_top('J')` is 5.

O2. `len(count_letters(key)) == 26`. 26 different capitals in 26 characters means each letter once: `BAD` gives 25, and `SHORT` fails the first test.

O3. `ABC[k:] + ABC[:k]`. For `k = 0`: `ABC + ''`, which is `ABC`, the key that changes nothing.

P1. It turns any number into 0 to 25, the places that `ABC` has, so a letter past `Z` goes round to the start.

P2. Many keys do not make a lock safe: if the lock leaves a pattern (the letter counts), the codebreaker uses the pattern and never tries the keys. Clever beats brute force.

P3. One plain letter becomes several secret letters, so the counts come out flat: there is no tall bar to match with E.

Q1. Like `shift`, with line 6 as `out += ABC[25 - i]`. No `% 26` is needed: for `i` from 0 to 25, `25 - i` is also 0 to 25.

Q2. Copy `many_shifts` and change one line: `out += shift(ch, -k)`. Line 8 must stay inside the `if`, or a space would move the key word along and the letters would no longer line up.