Finger Exercises 17

Dictionaries

≈ 60 min + stretch

Comp 102 — Programming Fundamentals

What this is. A workout for what we just covered in class. Most of the code here is new, so you have to *work it out*, not remember it. For every lookup, ask: *is this a key? Is it there? What comes back?*

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–N are for *everyone*; Part O 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 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.

Writing a dictionary. When you write a whole dictionary, write its pairs in the order they were added, the way Thonny prints them: `{'a': 1, 'b': 2}`.

Part A • Do You Remember?

RECALL ≈ 4 min

A1. A dictionary stores pairs. The label you look something up by is called the _____ , and the thing you get back is called the _____.

A2. Dictionary keys must be _____ and _____ (no repeats). Dictionary values can be _____.

A3. The three dictionary methods we used for looping:

1. _____ 2. _____ 3. _____

A4. Circle **True** or **False**:

(a) If `'zain'` is not in `d`, then `d['zain']` quietly gives `None`. True / False

(b) `'A+' in grades` is True when some student's grade is `'A+'`. True / False

(c) A dictionary numbers its entries, so `d[0]` always gives you the first pair. True / False

(d) Two different keys are allowed to store the same value. True / False

(e) `d = {}` makes an empty dictionary. True / False

(f) If `'ali'` is already a key, `d['ali'] = 'B'` adds a second `'ali'` pair. True / False

(g) `del d['zain']` raises a `KeyError` when `'zain'` is not a key. True / False

(h) Like a list, a dictionary can be changed after it is made. True / False

Part B • Value or Error?

CLASSIFY ≈ 6 min

Use these three dictionaries. For each line, write the value it gives. If it raises an error, write the error's name instead.

```
grades = {'ali': 'A+', 'bilal': 'B', 'zain': 'A'}
marks  = {'ali': [85, 90, 78], 'sana': [92, 88]}
mixed  = {1: 'one', (2, 3): 'pair', 'x': 10}
```

Line	Value or error	Line	Value or error
`grades['ali']`	_____	`marks['sana']`	_____
`grades['Ali']`	_____	`marks['sana'][1]`	_____
`grades['A+']`	_____	`marks['ali'][3]`	_____
`'A+' in grades`	_____	`marks[0]`	_____
`'zain' in grades`	_____	`mixed[1]`	_____
`'A+' in grades.values()`	_____	`mixed[(2, 3)]`	_____
`grades['bilal'] + '+'`	_____	`mixed['x'] + 5`	_____
`'B' in grades['bilal']`	_____	`mixed[2]`	_____

Part C • The Dictionary After Each Line

TRACE ≈ 5 min

```
1 stock = {}
2 stock['pens'] = 10
3 stock['ink'] = 4
4 stock['pens'] = 7
5 stock['clips'] = stock['ink'] * 2
6 del stock['ink']
7 stock['pens'] += 1
```

A shop keeps how many of each thing it has. Write the whole dictionary `stock` after each line runs.

Line	`stock` after the line
1	_____
2	_____
3	_____
4	_____
5	_____
6	_____
7	_____

C1. A line 8 is added: `stock['ink'] += 1`. What happens, and why?

_____

C2. Instead, line 8 is `stock['Pens'] = 1`. How many keys does `stock` have now? _____

Part D • What Does It Print?

TRACE ≈ 7 min

Write the output exactly, one value per line, in order.

D1.

```
d = {'a': 1, 'b': 2}
d['c'] = 3
d['a'] = 9
del d['b']
print(d['a'])
print('b' in d)
```

Output:

_____

_____

D2.

```
d = {'x': 10, 'y': 20}
for k in d:
    print(k)
```

Output:

_____

D3. One line changed. What is different?

```
d = {'x': 10, 'y': 20}
for k in d:
    print(d[k])
```

Output:

_____

D4.

```
prices = {'pen': 20, 'book': 150, 'bag': 900}
total = 0
for item in ['pen', 'pen', 'bag']:
    total += prices[item]
print(total)
```

Output:

_____

D5.

```
def count_matches(d):
    count = 0
    for k, v in d.items():
        if k == v:
            count += 1
    return count
```

What does each call return?

```
count_matches(
  {'a': 'a', 1: 1, 2: '2'})
```

_____

```
count_matches({})
```

_____

Part E • Crash or No Crash?

TRACE ≈ 3 min

```
def safe_lookup(d, key):
    try:
        return d[key]
    except KeyError:
        return None

grades = {'ali': 'A+', 'bilal': 'B'}
```

Write the value returned, or the exception that crashes the program.

E1. `safe_lookup(grades, 'bilal')` _____

E2. `safe_lookup(grades, 'fatima')` _____

E3. `safe_lookup(grades, 'B')` _____

E4. `safe_lookup([10, 20], 5)` _____

Part F • Keys, Values, or Pairs?

CLASSIFY ≈ 2 min

For `d = {'da': 3, 'mah': 3, 'ro': 1}`, write **K** if you get the keys, **V** if you get the values, **P** if you get key–value pairs.

F1. `d.keys()` _____ **F4.** `d.values()` _____

F2. `d.items()` _____ **F5.** `max(d.values())` _____

F3. `for k in d:` _____ **F6.** `for k, v in d.items():` _____

Part G • The Counting Pattern

TRACE ≈ 6 min

```
1  def generate_word_dict(song):
2      words_list = song.lower().split()
3      word_dict = {}
4      for w in words_list:
5          if w in word_dict:
6              word_dict[w] += 1
7          else:
8              word_dict[w] = 1
9      return word_dict
10
11 def find_frequent_word(word_dict):
12     highest = max(word_dict.values())
13     words = []
14     for k, v in word_dict.items():
15         if v == highest:
16             words.append(k)
17     return (words, highest)
```

These are the lecture's two functions. Call `generate_word_dict('Ho ho HEY ho hey')`. Fill in `word_dict` after each turn of the loop.

`w`	`word_dict` after this turn

G1. What does `find_frequent_word` return for the dictionary you built? _____

G2. And for `{'a': 2, 'b': 5, 'c': 5, 'd': 1}`? _____

G3. The same pattern counts letters. Write the dictionary it builds when the loop runs `for ch in 'banana':` instead of over words.

_____

G4. `generate_word_dict('')` returns _____.
Passing that to `find_frequent_word` crashes on line 12. Which error? (Hint: what is the biggest of *no* numbers?) _____

Part H • Can It Be a Key?

CLASSIFY ≈ 2 min

Tick one column for each. Remember the rule: a key has to be **immutable**.

Could this be used as a dictionary key?	Yes	No		Yes	No
`'ali'`	☐	☐	`-1`	☐	☐
`7`	☐	☐	`('ali', 'khan')`	☐	☐
`[1, 2]`	☐	☐	`['ali']`	☐	☐
`(1, 2)`	☐	☐	`{}`	☐	☐
`3.5`	☐	☐	`''`	☐	☐

Which of the ten could be used as a *value*? _____

Part I • Alias or Copy? TRACE ≈ 5 min

I1.

```
a = {'x': 1}
b = a
c = a.copy()
b['y'] = 2
c['z'] = 3
print(a)
print(c)
```

Output:

_____

_____

I2.

```
def add_bonus(d):
    for k in d:
        d[k] += 5

marks = {'ali': 80, 'sana': 90}
add_bonus(marks)
print(marks['ali'])
```

Output:

_____

I3.

```
d1 = {'a': 1}
d2 = d1
d2 = {'a': 5}
print(d1['a'])
```

Output:

_____

Part J • Dictionaries Inside Dictionaries TRANSLATE ≈ 5 min

```
grades = {
  'ali':  {'quiz': [5, 4, 4], 'final': 'B'},
  'zaid': {'quiz': [6, 7, 8], 'final': 'A'}
}
```

J1. Write the value, or the error's name.

`grades['zaid']['final']`	_____	`sum(grades['ali']['quiz'])`	_____
`grades['ali']['quiz'][1]`	_____	`grades['ali']['mid']`	_____
`grades['zaid']['quiz'][-1]`	_____	`grades['quiz']`	_____

J2. Now the other way. Write one line of code for each.

(a) Ali's final grade. _____

(b) Zaid's first quiz mark. _____

(c) Change Ali's final grade to 'A'. _____

(d) Add a fourth quiz mark, 9, for Zaid. _____

Part K • List or Dictionary?

CLASSIFY ≈ 5 min

K1. Tick the better choice for each job.

Job	List	Dictionary
Find a friend's phone number from the friend's name.	☐	☐
Keep the runs of each over of a cricket match, over 1 first.	☐	☐
Count how many times each word comes in a song.	☐	☐
Keep one student's five quiz marks, quiz 1 to quiz 5.	☐	☐
Find a city's population from the city's name.	☐	☐
People in a line at the bank: the first to come is served first.	☐	☐

K2. `L = ['ali', 'A+']` and `d = {'ali': 'A+'}`. Write the value, or the error.

`'A+' in L` _____ `'A+' in d` _____ `L[0]` _____

`'ali' in L` _____ `'ali' in d` _____ `d[0]` _____

K3. Two parallel lists become one dictionary.

```
1 names  = ['ali', 'bilal', 'ali']
2 grades = ['A', 'B', 'C']
3 d = {}
4 for i in range(len(names)):
5     d[names[i]] = grades[i]
6 print(d)
```

Output:

_____

How many keys? _____

Part L • Fill in the Blanks

FILL IN ≈ 6 min

Each function is almost done. Fill in every blank.

L1. Count each letter of a word: `count_letters('noon')` returns `{'n': 2, 'o': 2}`.

```
1 def count_letters(word):
2     counts = ____________
3     for ch in word:
4         if ____________________:
5             counts[ch] += 1
6         else:
7             ____________________
8     return counts
```

L2. The cost of a shopping list: `total_cost({'pen': 20, 'bag': 900}, ['bag', 'pen'])` returns 920.

```
1 def total_cost(prices, cart):
2     total = 0
3     for item in ____________:
4         total += ____________________
5     return total
```

L3. The student with the top mark (there are no ties): `best_student({'ali': 70, 'sana': 95})` returns `'sana'`.

```
def best_student(marks):
    top = max(____________________)
    for name, m in ____________________:
        if ____________________:
            return name
```

L4. A phone book that never crashes: `lookup(book, 'zara')` returns `'not found'` when Zara is not in it.

```
def lookup(book, name):
    try:
        return ____________________
    except ____________________:
        return 'not found'
```

Part M • Circle the Bug, Then Fix It

FIX ≈ 8 min

Each snippet has exactly one bug of a kind we saw in class. Circle it, then write the fix.

M1. This should count how many times each word appears, but it crashes on the very first word.

```
def count_words(words):
    d = {}
    for w in words:
        d[w] += 1
    return d
```

Fix:

_____

_____

M2. This should print `yes` when somebody scored an `'A'`. It always prints `no`.

```
grades = {'ali': 'A',
          'bilal': 'B'}
if 'A' in grades:
    print('yes')
else:
    print('no')
```

Fix:

_____

_____

M3. The user types `Ali`, and the program crashes with `KeyError: 'Ali'`.

```
1 grades = {'ali': 'A+', 'bilal': 'B'}
2 name = input('Name: ')
3 print(grades[name])
```

Fix line 2 or 3:

_____

M4. `find_in_L([{1: 2}, {2: 4}], 2)` returns `False`, but `2` is a key of the second dictionary.

```
1 def find_in_L(Ld, k):
2     for d in Ld:
3         if k in d:
4             return True
5         else:
6             return False
```

Fix:

_____

_____

M5. For `{'rah': 2, 'da': 3, 'ro': 1}` this returns `([], 'ro')` in place of `(['da'], 3)`.

```
1 def find_frequent_word(word_dict):
2     highest = max(word_dict)
3     words = []
4     for k, v in word_dict.items():
5         if v == highest:
6             words.append(k)
7     return (words, highest)
```

Fix line 2:

_____

Where did 'ro' come from?

_____

M6. `keys_with_value(grades, 'A')` should return `['ali', 'zain']`, but returns `['A', 'A']`.

```
1 def keys_with_value(d, target):
2     result = []
3     for k, v in d.items():
4         if v == target:
5             result.append(v)
6     return result
```

Fix:

_____

M7. The program saves a backup before the change, yet the backup shows `'ali': 'F'` too.

```
1 grades = {'ali': 'A+', 'bilal': 'B'}
2 backup = grades
3 grades['ali'] = 'F'
4 print(backup)
```

Fix:

_____

Part N • One Sentence Each

EXPLAIN ≈ 3 min

N1. Why is one dictionary better than two parallel lists (one of names, one of grades) for looking up a student's grade?

_____

N2. After `d2 = d1` a change to `d2` also changes `d1`, but after `d2 = d1.copy()` it does not. Why?

_____

N3. Why does `d[w] = 1` work for a new word, when `d[w] += 1` crashes?

_____

Part O • Write the Code

STRETCH ≈ 10 min, optional

Write each function in full. Run the sample call on paper before you call it done.

O1. `invert(d)`: `d` maps names to grades. Return a new dictionary that maps each grade to the list of names that got it.
`invert({'ali': 'A', 'bilal': 'B', 'zain': 'A'})`
returns `{'A': ['ali', 'zain'], 'B': ['bilal']}`.

O2. `averages(marks)`: `marks` maps names to lists of marks. Return a dictionary of each student's average. Leave out a student with no marks.
`averages({'Ali': [80, 90, 70], 'Fatima': [90, 100], 'Deadpool': []})`
returns `{'Ali': 80.0, 'Fatima': 95.0}`.

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.

1. Type `d = {'a': 1, 'a': 2}`, then `d`. The same key was written twice. Which pair is kept?
2. Type `d = {}` and then `d[[1, 2]] = 5`. Read the last line of the red error. What word does Python use for a type that cannot be a key?
3. Type `{'a': 1, 'b': 2} == {'b': 2, 'a': 1}`. True or False? Then copy `generate_word_dict` from Part G and call it on a line of your favourite song.

Stuck on more than two items? Re-read the Lecture 17 slides "List vs Dictionary", "Dictionary Operations", "Membership Testing", "Aliasing", "Looping Over a Dictionary" and "Step 1: `generate_word_dict`" before the next class — then redo Parts B, C and G cold. For every lookup, ask: *is this a key, and is it there?* If it is not, the program crashes with a `KeyError` — unless a `try` is waiting for it.

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. the **key**; the **value**. A dictionary maps keys to values.

A2. **immutable** (str, int, float, tuple) and **unique**; values can be **anything** — any type at all, and two keys may hold the same value.

A3. `.keys()`, `.values()`, `.items()`.

A4. (a) **False** — it raises a `KeyError` and stops the program. You only get `None` if you wrap the lookup in `try/except` yourself, the way `safe_lookup` did. (b) **False** — `in` checks *keys* only, never values. (c) **False** — a dictionary has no positions. `d[0]` looks for a *key* called `0`, and raises `KeyError` if there is no such key. (d) **True** — keys must be unique, values need not be. (e) **True**. (f) **False** — the old value is replaced; there is still one `'ali'`. (g) **True** — you cannot delete what is not there. (h) **True** — dictionaries are mutable: add, change, delete.

B. Left column: `'A+'`; **KeyError** (keys are case-sensitive: 'Ali' is not 'ali'); **KeyError** ('A+' is a *value*, not a key); **False** (`in` checks keys); **True**; **True** (now we asked the values); `'B+'` (the value is a string, so `+` joins); **True** (`grades['bilal']` is the string 'B', and `in` on a *string* looks inside it).
Right column: `[92, 88]`; `88`; **IndexError** — the key 'ali' is fine, but its list has only indices 0 to 2; **KeyError** (no key `0`; a dictionary has no positions); `'one'`; `'pair'` (a tuple is immutable, so it can be a key); `15`; **KeyError** (`2` alone is not a key; `(2, 3)` is).

C. 1: `{}`. 2: `{'pens': 10}`. 3: `{'pens': 10, 'ink': 4}`. 4: `{'pens': 7, 'ink': 4}` (replaced, not added). 5: `{'pens': 7, 'ink': 4, 'clips': 8}`. 6: `{'pens': 7, 'clips': 8}`. 7: `{'pens': 8, 'clips': 8}` — `+= 1` works here because 'pens' is already a key.

C1. **KeyError: 'ink'**. Line 6 deleted 'ink', so there is no old value to read and add 1 to. (The same bug as M1.)

C2. **3**. 'Pens' and 'pens' are different strings, so this *adds* a new key.

D1. `9` then `False`. `d['c'] = 3` added a new entry; `d['a'] = 9` did *not* add a second `'a'` — it replaced the old value, because keys are unique. `del` removed `'b'` completely.

D2. `x` then `y`. Looping over a dictionary gives you the **keys**, not the values — that is the default.

D3. `10` then `20`. `k` is still a key — we just used it to look the value up with `d[k]`. `for k, v in d.items():` would give you both at once.

D4. `940`: $20 + 20 + 900$. The loop walks the *list*, and each item is used as a key — the same shape as `find_grades`. The 'book' is never looked up.

D5. `2`: 'a' equals 'a' and `1` equals `1`, but the number `2` is not the string '2'. `0` for the empty dictionary: the loop never runs.

E1. `'B'` — the key is there, so the `try` block simply returns.

E2. `None` — the `KeyError` is caught and the function returns `None` instead of crashing.

E3. `None` — 'B' is a value, not a key, so the lookup fails and is caught.

E4. **Crash: `IndexError`**. A list was passed in, and a list raises `IndexError`, not `KeyError`. The `except` names only `KeyError`, so this error flies straight past.

F. **K, P, K, V, V, P.** Item 3 is the one people miss: a plain `for` loop over a dictionary walks the *keys*. Item 5 looks at the values and picks the largest one (`3`) — that is the line `find_frequent_word` used.

G. `.lower()` first, so the words are 'ho', 'ho', 'hey', 'ho', 'hey'. After each turn: `{'ho': 1}`; `{'ho': 2}`; `{'ho': 2, 'hey': 1}`; `{'ho': 3, 'hey': 1}`; `{'ho': 3, 'hey': 2}`. A new word takes the `else` branch once; after that it always takes the `if`.

G1. `(['ho'], 3)`. One winner, but still inside a list.

G2. `(['b', 'c'], 5)` — a tie, so both keys are collected. That is why the function returns a list.

G3. `{'b': 1, 'a': 3, 'n': 2}`. Looping over a string gives one character at a time.

G4. `{}`, an empty dictionary (the loop never runs). Then `max` of nothing raises **ValueError**.

H. **Yes, Yes, No, Yes, Yes.** Watch rows 3 and 4: they look almost the same, but `[1, 2]` is a *list* (mutable, so it cannot be a key) while `(1, 2)` is a *tuple* (immutable, so it can). Strings, ints and floats are all immutable and all fine.
Right column: **Yes, Yes, No, No, Yes.** A negative number and a tuple of strings are fine; `['ali']` is a list and `{}` is a dictionary, and both are mutable. The empty string is still a string. As a *value*:

all ten — values can be anything.

I1. `{'x': 1, 'y': 2}` then `{'x': 1, 'z': 3}`. `b` is another name for `a`'s dictionary, so line 4 changes `a` too. `c` is a separate dictionary.

I2. `85`. Inside the function, `d` is another name for `marks` — the same dictionary, not a copy. The function returns nothing, but the caller's dictionary has changed.

I3. `1`. Line 3 does not change a dictionary: it makes a *new* one and points `d2` at it. Now the two names are no longer linked. Compare `d2['a'] = 5`, which would change `d1` too.

J1. `'A'`; `13`; `4`; **KeyError** (Ali's inner dictionary has no 'mid'); `8`; **KeyError** — 'quiz' is a key of the *inner* dictionaries, not of `grades`. Read each line left to right: every `[...]` goes one level deeper.

J2. (a) `grades['ali']['final']`. (b) `grades['zaid']['quiz'][0]`. (c) `grades['ali']['final'] = 'A'`. (d) `grades['zaid']['quiz'].append(9)` — the value is a list, so list methods work on it.

K1. **Dictionary, List, Dictionary, List, Dictionary, List.** Look up by a *name* → dictionary. Order or position matters → list.

K2. `True`; `False`; `'ali'`; `True`; `True`; **KeyError**. On a list, `in` checks the *values*; on a dictionary, the *keys*.

K3. `{'ali': 'C', 'bilal': 'B'}`, so **2** keys. The second 'ali' does not make a new pair: it replaces 'A' with 'C'. Keys are unique.

L1. `{}`; `ch in counts`; `counts[ch] = 1`. The counting pattern, with letters in place of words.

L2. `cart`; `prices[item]`. Walk the list, look each item up in the dictionary.

L3. `marks.values()`; `marks.items()`; `m == top`. The first half of `find_frequent_word`, stopping at the first match.

L4. `book[name]`; `KeyError`. This is `safe_lookup` with a different answer for a missing key.

M1. `d[w] += 1` means "read `d[w]`, add 1, store it back" — but on a brand-new word there is nothing to read, so it raises `KeyError`. Use the counting pattern: `if w in d:` then `d[w] += 1`, `else:` then `d[w] = 1`.

M2. `in` searches the *keys*, and the keys here are names, not grades. Write `if 'A' in grades.values():` — or loop with `for k, v in grades.items():` and test `v == 'A'`, exactly as `keys_with_value` did.

M3. Keys are case-sensitive: 'Ali' is not 'ali'. Make the typed name lower case first: `name = input('Name: ').lower()` (the same `.lower()` that `generate_word_dict` uses).

M4. The `else: return False` ends the function after the *first* dictionary, so the second is never looked at. Delete lines 5–6 and put `return False` *after* the loop, level with `for`: say "no" only when every dictionary has said no.

M5. `highest = max(word_dict.values())`. A plain dictionary means its *keys*, so `max` picked the biggest key in A-to-Z order, 'ro'. No count equals 'ro', so the list stays empty.

M6. Line 5 should be `result.append(k)`. Every `v` kept is equal to `target`, so appending `v` only repeats 'A'.

M7. Line 2 makes an alias, not a backup. Write `backup = grades.copy()`.

N1. Two lists can drift out of step if someone updates one and forgets the other, and finding a name means scanning the whole list with `.index()`; a dictionary keeps the pair together and looks the grade up directly by name.

N2. `d2 = d1` makes an **alias** — two names for the *same* dictionary — while `.copy()` builds a *new* dictionary, so the two are now separate objects. Same rule you learned for lists.

N3. `d[w] = 1` only *stores* a value, while `+= 1` must first *read* the old value, and a new word has none.

O1. `result = {}` / `for k, v in d.items():` / `if v in result:` / `result[v].append(k)` / `else:` / `result[v] = [k]` / `return result`. The counting pattern again, with a list where the count was: start it with one name, then append.

O2. `result = {}` / `for name, L in marks.items():` / `if len(L) > 0:` / `result[name] = sum(L) / len(L)` / `return result`. Without the `if`, Deadpool's empty list would raise `ZeroDivisionError`. This is the class grades example from the start of the lecture, done with a dictionary.