1.5 Collections – lists, dictionaries, tuples, sets

Name: _____ Class: _____ Date: _____ **Total: 24 marks**

KEY WORDS list • dictionary • tuple • set • aliasing

Objective

Choose the right collection for the job – list, dictionary, tuple or set – and manipulate it correctly, including the aliasing and mutate-while-iterating traps.

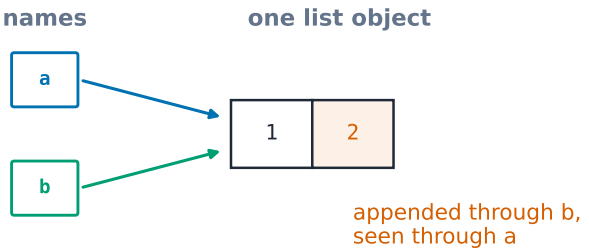
1 Key knowledge and worked examples

■ Four collections, four jobs

- **list** : ordered, changeable, allows duplicates – `marks = [72, 85, 61]`, indexed from 0, sliced `marks[1:3]`.
- **dictionary** : key → value lookup – `price = {"tea": 2.8}`; read and write by key; `.get(key, default)` never raises.
- **tuple** : fixed grouping – `(53.1, -1.5)`; cannot be changed after creation.
- **set** : uniqueness, no order – `{"a", "b"}`; `in` is fast.

Choosing is the design decision: order + duplicates → list; lookup by name → dictionary; fixed grouping → tuple; uniqueness → set.

```
a = [1]
b = a
b.append(2)
a  →  [1, 2]
```



■ Worked example: tally by category

```
orders = ["tea", "flat white", "tea", "tea", "mocha"]
tally = {}
for item in orders:
    tally[item] = tally.get(item, 0) + 1
print(tally)        # {'tea': 3, 'flat white': 1, 'mocha': 1}
```

`.get(item, 0)` supplies 0 for an unseen key, so the same line handles the first and the later sightings. A dictionary keyed by the thing you are counting is the standard tally shape.

■ Two traps: aliasing and mutation under iteration

Aliasing – two names, one list:

```
a = [1, 2]
b = a              # NOT a copy
b.append(3)
print(a)           # [1, 2, 3] - a changed too
```

Copy when independence matters: `b = a[:]`.

Mutating a list while looping over it skips elements – `for m in marks[:]` iterates a snapshot, so removal inside is safe.

2 Practice

2.1 For each requirement, name the best collection: (a) a runner's split times in order; (b) product code → stock level; (c) the three RGB parts of a colour; (d) the distinct tags on a photo. [4]

2.2 Given `marks = [72, 85, 61, 90]`, write one line each for: the length, the maximum, the last item, and the middle two as a slice. [4]

2.3 After `a = [1, 2]; b = a[:]; b.append(3)`, give the value of `a` and of `b`, and say why they differ from the aliasing case. [3]

3 Exam-style questions

3.1 A stock system starts as `stock = {"tea": 12, "mocha": 5}`. Two sales of tea and one delivery of 10 mochas occur.

Write the code that applies the transactions (guarding the tea sale so stock never goes below zero), prints the final stock, and reports which drinks are low (below 6) as a list. Give the exact final output. [7]

3.2 A survey collects duplicates: `votes = ["a", "b", "a", "c", "b", "a"]`.

Write code that (i) counts the votes with a dictionary; (ii) names the winner (the key with the highest count, breaking ties by first-seen order); (iii) gives the number of DISTINCT options seen. Give all three results for the list above. [6]

Solutions

2.1

- (a) list – order matters.
- (b) dictionary – lookup by code.
- (c) tuple – fixed grouping of three parts.
- (d) set – distinct values only.

2.2

- `len(marks)` → 4.
- `max(marks)` → 90.
- `marks[-1]` → 90.
- `marks[1:3]` → `[85, 61]`.

2.3

- `a` is `[1, 2]`.
- `b` is `[1, 2, 3]`.
- `a[:]` made a copy, so the two names refer to two lists; only `b` changed.

3.1

```
stock = {"tea": 12, "mocha": 5}
stock["tea"] = max(0, stock["tea"] - 2)
stock["mocha"] = stock["mocha"] + 10
print(stock)
low = [drink for drink, n in stock.items() if n < 6]
print("Low:", low)
```

- Sales decrease with the max(0, ...) guard.
- Delivery increases mocha.
- Final stock printed: `{'tea': 10, 'mocha': 15}`.
- Low-drink comprehension over `.items()` with the `< 6` condition.
- Output `Low: []` – nothing is below 6.

3.2

```
counts = {}
for v in votes:
    counts[v] = counts.get(v, 0) + 1
winner = max(counts, key=counts.get)
distinct = len(set(votes))
```

- Tally loop with `.get`.
- `winner`: `max` over the dictionary keys by count – with ties, `max` keeps the first key encountered with that count, and dictionaries preserve insertion (first-seen) order.
- `winner` → `'a'` (3 votes).
- `counts` → `{'a': 3, 'b': 2, 'c': 1}`; `distinct` → 3.