Solution 29: Python Inverted Index

1) Fill in the blanks: What is an “inverted dictionary”?

- inverted dictionary: `score -> a list of players`

Fill in:

- Key is a **score**.
- Value is a **list** of players.

2) Invert a scoreboard (what does it print?)

One possible output (key order may vary):

```
{3: ['Amy', 'Chloe'], 2: ['Ben'], 1: ['Dan']}
```

3) Invert a scoreboard (fill in the blanks)

```
scoreboard = {"Eve": 1, "Frank": 1, "Gina": 2}

score_to_players = {}

for player, score in scoreboard.items():
    if score not in score_to_players:
        score_to_players[score] = []
    score_to_players[score].append(player)

print(score_to_players)
```

4) Quick lookup by score (what does it print?)

```
[]
```

5) `.split()` (what does it print?)

```
['cat', 'eats', 'fish']
3
```

6) Count words in a document (what does it print?)

`"fish swims fast"` has 3 words, `"cat runs"` has 2 words.

```
3
2
```

7) Fix the bug: split the string into words

Fixed line:

```
words = text.split()
```

8) Build the inverted index (what does it print?)

One possible output (key order may vary):

```
{'cat': [0, 2], 'eats': [0], 'fish': [0, 1], 'swims': [1], 'runs': [2], 'fast': [2]}
```

9) Trace it (fill in the table)

Step	(doc_id, w) processed	word*to*docs
0 (start)	—	`{}`
1	(0, "cat")	`{'cat': [0]}`
2	(0, "eats")	`{'cat': [0], 'eats': [0]}`
3	(1, "eats")	`{'cat': [0], 'eats': [0, 1]}`
4	(1, "fish")	`{'cat': [0], 'eats': [0, 1], 'fish': [1]}`

(Any equivalent dictionary with the same keys and lists is fine.)

10) Fill in the blanks: build `word_to_docs`

```
docs = {
    0: "red fish",
    1: "blue fish"
}

word_to_docs = {}

for doc_id, text in docs.items():
    words = text.split()
    for w in words:
        if w not in word_to_docs:
            word_to_docs[w] = []
        word_to_docs[w].append(doc_id)

print(word_to_docs)
```

11) Fix the bug (inverted index)

Missing line:

```
word_to_docs[w] = []
```

12) Challenge: avoid duplicate doc_ids for repeated words (fill in the blanks)

Fill-in:

- First blank: `doc_id`
- Second blank: `doc_id`

Completed code:

```
docs = {
    0: "fish fish fish",
    1: "fish swims"
}

word_to_docs = {}

for doc_id, text in docs.items():
    for w in text.split():
        if w not in word_to_docs:
            word_to_docs[w] = []
        if doc_id not in word_to_docs[w]:
            word_to_docs[w].append(doc_id)

print(word_to_docs)
```

One possible output (key order may vary):

```
{'fish': [0, 1], 'swims': [1]}
```

13)## 13) Search one word (what does it print?)

```
[0, 2]
```

14) Search a missing word (what does it print?)

```
[]
```

15) Count unique words (fill in the blanks)

Fill-in:

- `all_words = set()`
- `all_words.add(w)`
- `print(len(all_words))`

Completed code:

```
docs = {
    0: "cat eats fish",
    1: "fish swims",
    2: "cat runs fast"
}

all_words = set()

for doc_id, text in docs.items():
    for w in text.split():
        all_words.add(w)

print(len(all_words))
```

Output:

```
6
```

16) Which words appear in a doc? (fill in the blanks)

Fill-in:

- `word_to_docs.items()`
- `in`
- `append`

Completed code:

```
word_to_docs = {
    "cat": [0, 2],
    "eats": [0],
    "fish": [0, 1],
    "swims": [1],
    "runs": [2],
    "fast": [2]
}

doc_id = 2
words_in_doc = []

for w, doc_list in word_to_docs.items():
    if doc_id in doc_list:
        words_in_doc.append(w)

print(words_in_doc)
```

One possible output (order may vary):

```
['cat', 'runs', 'fast']
```

---

17) Time complexity (fill in the blanks)

- Scanning a whole dictionary: **O(n)**
- Dictionary lookup: **O(1)** (average)