Worksheet 29: Python Inverted Index

Name: ____________________ Date: ____________________

Instructions

- Answer in the blanks.
- For “write code” questions, write valid Python code (no functions needed).
- For “what does it print” questions, write the **exact** output (line by line).
- If a question says “order may vary”, any correct order is acceptable.

Part A — Inverted Dictionary (Scoreboard Warm-up)

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

We start with a dictionary like this:

- original dictionary: `player -> score`

An **inverted dictionary** groups players by score:

- inverted dictionary: `score -> ______________________________`

Fill in:

- In the inverted dictionary, the **key** is a __________.
- In the inverted dictionary, the **value** is a __________ of players. (Choose from: string, list, set, or dict.)

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

What does this print?

```
scoreboard = {"Amy": 3, "Ben": 2, "Chloe": 3, "Dan": 1}

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)
```

Output (key order may vary):

```

```

3) Invert a scoreboard (fill in the blanks)

Complete the code so it builds the inverted dictionary.

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

score_to_players = {}

for player, score in scoreboard.items():
    if score ____________ score_to_players:
        score_to_players[score] = ____________
    score_to_players[score].____________(player)

print(score_to_players)
```

Expected output (key order may vary):

```
{1: ['Eve', 'Frank'], 2: ['Gina']}
```

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

What does it print?

```
score_to_players = {5: ["Amy", "Ben"], 1: ["Chloe"], 3: ["Dan"]}

query_score = 2

if query_score in score_to_players:
    print(score_to_players[query_score])
else:
    print([])
```

Output:

```

```

Part B — Reminder: `.split()` by whitespace

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

What does it print?

```
text = "cat eats fish"
words = text.split()
print(words)
print(len(words))
```

Output:

```

```

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

What does it print?

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

print(len(docs[0].split()))
print(len(docs[1].split()))
```

Output:

7) Fix the bug: split the string into words

This code tries to loop over **words**, but it actually loops over **letters**.

Fix it by changing **one line**.

Buggy code:

```
text = "cat eats fish"
words = text  # BUG: this is a string, not a list of words

for w in words:
    print(w)
```

Write the fixed line:

```
words = ____________________________
```

Part C — Build an Inverted Index (word --> [doc_ids])

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

What does it print?

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

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)
```

Output (key order may vary):

```

```

9) Trace it (fill in the table)

We run this code:

```
docs = {
    0: "cat eats",
    1: "eats fish"
}

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] = []
        word_to_docs[w].append(doc_id)
```

Fill in the table after each word is processed:

Step	`(doc_id, w)` processed	`word_to_docs`
0 (start)	—	`{}`
1	(0, "cat")	________________
2	(0, "eats")	________________
3	(1, "eats")	________________
4	(1, "fish")	________________

10) Fill in the blanks: build `word_to_docs`

Complete the code.

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

word_to_docs = {}

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

print(word_to_docs)
```

Expected output (key order may vary):

```
{'red': [0], 'fish': [0, 1], 'blue': [1]}
```

11) Fix the bug (inverted index)

This code has a bug. It crashes when it sees a new word.

Fix it by adding **one missing line**.

Buggy code:

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

word_to_docs = {}

for doc_id, text in docs.items():
    for w in text.split():
        if w not in word_to_docs:
            # missing line here
        word_to_docs[w].append(doc_id)

print(word_to_docs)
```

Write the missing line:

```

```

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

Sometimes a word repeats **inside the same document**:

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

If we use the basic code, `"fish"` would get `[0, 0, 0, 1]`.

We want `"fish"` to be `[0, 1]` (each doc_id only once per word).

Fill in the blanks so a doc_id is added **only if it is not already in the list**.

```
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 ____________ not in word_to_docs[w]:
            word_to_docs[w].append(____________)

print(word_to_docs)
```

Expected output (key order may vary):

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

Part D## Part D — Search One Word

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

What does it print?

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

query = "cat"

if query in word_to_docs:
    print(word_to_docs[query])
else:
    print([])
```

Output:

```
```

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

What does it print?

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

query = "dog"

if query in word_to_docs:
    print(word_to_docs[query])
else:
    print([])
```

Output:

```

```

Part E — Mini Challenges

15) Count unique words (fill in the blanks)

Use a set to count how many **different** words appear in all documents.

Fill in the blanks.

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

all_words = ____________

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

print(____________(all_words))
```

Expected output:

```
6
```

---

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

You are given an inverted index `word_to_docs`. We want a list of **all words** that appear in `doc_id = 2`.

Hint: check whether `2` is in each word's doc list.

Fill in the blanks.

```
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.____________():
    if doc_id ____________ doc_list:
        words_in_doc.____________(w)

print(words_in_doc)
```

Expected output (order may vary):

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

---

17) Time complexity (fill in the blanks)

Fill in the blanks using `n` or `1`:

- If we scan a whole dictionary to find something, it is usually **O(____)**.
- If we do a dictionary lookup like `d[key]`, it is usually **O(____)** (average).