In [1]:
```
import json
import math
```

In [2]:
```
# Load the document representatives
with open("./input/documents.json", "r") as f:
    documents = json.load(f)
```

In [4]:
```
# ---- Cosine Similarity ----
def cosine_similarity(doc1, doc2):
    # union of all words
    words = set(doc1.keys()).union(set(doc2.keys()))

    # create vectors
    v1 = [doc1.get(w, 0) for w in words]
    v2 = [doc2.get(w, 0) for w in words]

    # dot product
    dot = sum(a*b for a, b in zip(v1, v2))

    # magnitudes
    mag1 = math.sqrt(sum(a*a for a in v1))
    mag2 = math.sqrt(sum(b*b for b in v2))

    if mag1 == 0 or mag2 == 0:
        return 0.0

    return dot / (mag1 * mag2)
```

In [5]:
```
# ---- Single-pass clustering ----
def single_pass_clustering(documents, threshold=0.3):
    clusters = []   # list of (cluster_rep, [docs])

    for doc_name, doc_rep in documents.items():
        if not clusters:
            clusters.append((doc_rep, [doc_name]))
            continue

        placed = False
        for i, (rep, docs) in enumerate(clusters):
            sim = cosine_similarity(doc_rep, rep)
            if sim >= threshold:
                docs.append(doc_name)
                # update cluster representative (average frequencies)
                for w, f in doc_rep.items():
                    rep[w] = rep.get(w, 0) + f
                clusters[i] = (rep, docs)
                placed = True
                break

        if not placed:
            clusters.append((doc_rep, [doc_name]))

    return clusters
```

In [22]:
```
# ---- Run clustering ----
clusters = single_pass_clustering(documents, threshold=0.75)
```

In [23]:
```python
# Print result
for idx, (rep, docs) in enumerate(clusters, 1):
    print(f"\nCluster {idx}: {docs}")
```

```
Cluster 1: ['doc1.txt', 'doc2.txt']

Cluster 2: ['doc3.txt']

Cluster 3: ['doc4.txt']
```

In [ ]: