

ASSIGNMENT-04

V.Shiva
krishna
2303A52151
BATCH-41

Task 1: Sentiment Classification for Customer Reviews

ZERO SHOT PROMPT: Create a system that automatically generates ten customer reviews and analyzes each review to determine its sentiment. Every review must be classified into exactly one category: Positive, Negative, or Neutral. The solution should be clear, efficient, and easy to understand.

CODE:

```
Lab-4.py > ...
1  #Create a system that automatically generates ten customer reviews and analyzes each review
2  import random
3  import re
4  POSITIVE_WORDS = [
5      "great", "excellent", "amazing", "love", "loved", "good", "awesome", "perfect",
6      "fantastic", "happy", "satisfied", "recommend", "fast", "friendly", "helpful"
7  ]
8  NEGATIVE_WORDS = [
9      "bad", "terrible", "awful", "hate", "hated", "poor", "broken", "slow", "rude",
10     "refund", "disappointed", "worse", "worst", "problem", "issues"
11 ]
12
13 def tokenize(text: str):
14     return re.findall(r"[a-z']+", text.lower())
15
16 def sentiment_label(text: str) -> str:
17     tokens = tokenize(text)
18     pos = sum(1 for t in tokens if t in POSITIVE_WORDS)
19     neg = sum(1 for t in tokens if t in NEGATIVE_WORDS)
20
21     if pos > neg:
22         return "Positive"
23     if neg > pos:
24         return "Negative"
25     return "Neutral" # exactly one category
```

```
Lab-4.py > ...
25     return "Neutral" # exactly one category
26
27 def generate_reviews(n=10, seed=7):
28     random.seed(seed)
29
30     templates = [
31         "The delivery was {delivery} and the staff was {staff}.",
32         "Product quality is {quality}; I'm {feeling} with the purchase.",
33         "Support was {support}, but the app is {app}.",
34         "Packaging was {packaging} and the item arrived {arrival}.",
35         "Overall experience was {overall}.",
36     ]
37
38     delivery_opts = ["fast", "slow", "on time"]
39     staff_opts = ["friendly", "rude", "professional"]
40     quality_opts = ["excellent", "good", "poor", "average"]
41     feeling_opts = ["happy", "satisfied", "disappointed", "okay"]
42     support_opts = ["helpful", "great", "terrible", "okay"]
43     app_opts = ["amazing", "broken", "fine", "average"]
44     packaging_opts = ["perfect", "bad", "standard", "fine"]
45     arrival_opts = ["on time", "late", "early"]
46     overall_opts = ["excellent", "great", "neutral", "terrible", "average"]
47
48     reviews = []
49
```

```
Lab-4.py > ...
27     def generate_reviews(n=10, seed=7):
28         reviews = []
29         for _ in range(n):
30             t = random.choice(templates)
31             review = t.format(
32                 delivery=random.choice(delivery_opts),
33                 staff=random.choice(staff_opts),
34                 quality=random.choice(quality_opts),
35                 feeling=random.choice(feeling_opts),
36                 support=random.choice(support_opts),
37                 app=random.choice(app_opts),
38                 packaging=random.choice(packaging_opts),
39                 arrival=random.choice(arrival_opts),
40                 overall=random.choice(overall_opts),
41             )
42             reviews.append(review)
43         return reviews
44
45 if __name__ == "__main__":
46     reviews = generate_reviews(n=10, seed=7)
47
48     print("Generated Reviews + Sentiment")
49     print("-" * 35)
50     for i, r in enumerate[Any](reviews, 1):
51         print(f"{i}. {r}")
52         print(f"    Sentiment: {sentiment_label(r)}")
```

OUTPUT:

```
1. Support was helpful, but the app is fine.  
Sentiment: Positive  
2. The delivery was fast and the staff was rude.  
Sentiment: Neutral  
3. The delivery was fast and the staff was professional.  
Sentiment: Positive  
4. Support was great, but the app is amazing.  
Sentiment: Positive  
5. Overall experience was average.  
Sentiment: Neutral  
6. Packaging was standard and the item arrived early.  
Sentiment: Neutral  
Sentiment: Neutral  
6. Packaging was standard and the item arrived early.  
Sentiment: Neutral  
7. Support was helpful, but the app is average.  
Sentiment: Positive  
8. Packaging was fine and the item arrived early.  
Sentiment: Neutral  
9. The delivery was fast and the staff was rude.  
Sentiment: Neutral  
10. Support was helpful, but the app is average.  
Sentiment: Positive
```

PS C:\Users\saite\Downloads\AI ASSISTANT CODING> □

JUSTIFICATION:

This code generates 10 random customer reviews using templates and fills them with random options like delivery speed, staff behavior, product quality, etc. It then tokenizes each review into words using regular expressions and counts how many words match predefined positive or negative word lists. Based on the count, it classifies the review as Positive, Negative, or Neutral, ensuring only one label is assigned. The program prints each review along with its sentiment label. Overall, it demonstrates basic sentiment analysis using keyword matching and random review generation.

□ZERO SHOT PROMPT: Analyze each customer review and identify its sentiment.

Classify every review into exactly one category: Positive, Negative, or Neutral.

Example:

Customer Review: The product quality is excellent and I am very satisfied.

Sentiment: Positive

Now analyze ten customer reviews and assign the correct sentiment label. CODE:

```
Lab-4.py > ...
18  #Now analyze ten customer reviews and assign the correct sentiment label.
19  import re
20  from collections import Counter
21  # Simple, readable rule-based sentiment classifier (no external Libraries)
22  POSITIVE_WORDS = {
23      "excellent", "amazing", "awesome", "great", "good", "love", "loved", "like",
24      "satisfied", "happy", "perfect", "fantastic", "wonderful", "superb",
25      "recommend", "recommended", "fast", "quick", "helpful", "friendly", "best"
26  }
27  NEGATIVE_WORDS = {
28      "bad", "poor", "terrible", "awful", "worst", "hate", "hated", "dislike",
29      "unsatisfied", "unhappy", "broken", "damaged", "slow", "late", "delay",
30      "rude", "refund", "problem", "issues", "buggy", "defective", "disappointed"
31  }
32  NEGATIONS = {"not", "no", "never", "n't"}
33  def tokenize(text: str):
34      return re.findall("[a-z']+", text.lower())
35  def classify_sentiment(review: str) -> str:
36      words = tokenize(review)
37      score = 0
38      for i, w in enumerate[Any](words):
39          prev = words[i - 1] if i > 0 else ""
40          negated = (prev in NEGATIONS)
41          if w in POSITIVE_WORDS:
42              score += -1 if negated else 1
43          elif w in NEGATIVE_WORDS:
44              score += 1 if negated else -1
45      if score > 0:
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
```

OUTPUT:

```
te/Downloads/AI ASSISTENT CODING/AS4.py/task2.py
01. Review: The product quality is excellent and I am very satisfied.
    Sentiment: Positive

02. Review: Delivery was late and the item arrived damaged.
    Sentiment: Negative

03. Review: It works as expected, nothing special.
    Sentiment: Neutral

04. Review: Amazing performance and fast shipping. Love it!
    Sentiment: Positive

05. Review: Customer support was rude and not helpful.
    Sentiment: Negative

06. Review: Good value for money, I would recommend it.
    Sentiment: Positive

07. Review: The app is buggy and I hate the new update.
    Sentiment: Negative

08. Review: Packaging was okay, but the product is fine.
    Sentiment: Neutral

09. Review: Not bad at all, I am happy with the purchase.
    Sentiment: Positive

10. Review: Worst experience: defective item and refund took forever.
    Sentiment: Negative
```

JUSTIFICATION:

This Python program performs basic rule-based sentiment analysis without using any external libraries. It defines lists of positive and negative words, and also handles negation words like "not" and "never" to reverse sentiment when needed. Each review is tokenized into words, and a sentiment score is calculated by adding +1 for positive words and -1 for negative words, reversing the score if the word is preceded by a negation. Based on the final score, the review is classified as Positive, Negative, or Neutral. Finally, the program prints each review with its sentiment and shows a summary count of each sentiment category.

Task 2: Email Priority Classification

ONE SHOT PROMPT: You are an AI system that classifies emails into exactly one priority level: High Priority, Medium Priority, or Low Priority. Example: Input Email: "The production server is down and needs immediate attention." Output Priority: High Priority.

CODE:

```
131 import re
132 # Keyword sets for priority classification
133 HIGH_PRIORITY_WORDS = [
134     "urgent", "immediate", "critical", "emergency", "down", "broken", "failed",
135     "crash", "outage", "security", "breach", "hack", "attack", "deadline",
136     "asap", "as soon as possible", "important", "escalate", "priority"
137 ]
138 MEDIUM_PRIORITY_WORDS = [
139     "request", "needed", "required", "update", "review", "meeting", "schedule",
140     "follow up", "reminder", "pending", "issue", "problem", "concern"
141 ]
142 LOW_PRIORITY_WORDS = [
143     "newsletter", "update", "information", "general", "when convenient",
144     "no rush", "optional", "news", "announcement"
145 ]
146 def tokenize(text: str):
147     """Convert text to lowercase tokens."""
148     return re.findall(r"[a-z']+", text.lower())
149 def classify_priority(email: str) -> str:
150     """
151     Classify email into exactly one priority level: High, Medium, or Low Priority.
152     Args:
153         email: The email text to classify
154     Returns:
155         "High Priority", "Medium Priority", or "Low Priority"
156     """
157     words = tokenize(email)
158     text_lower = email.lower()
```

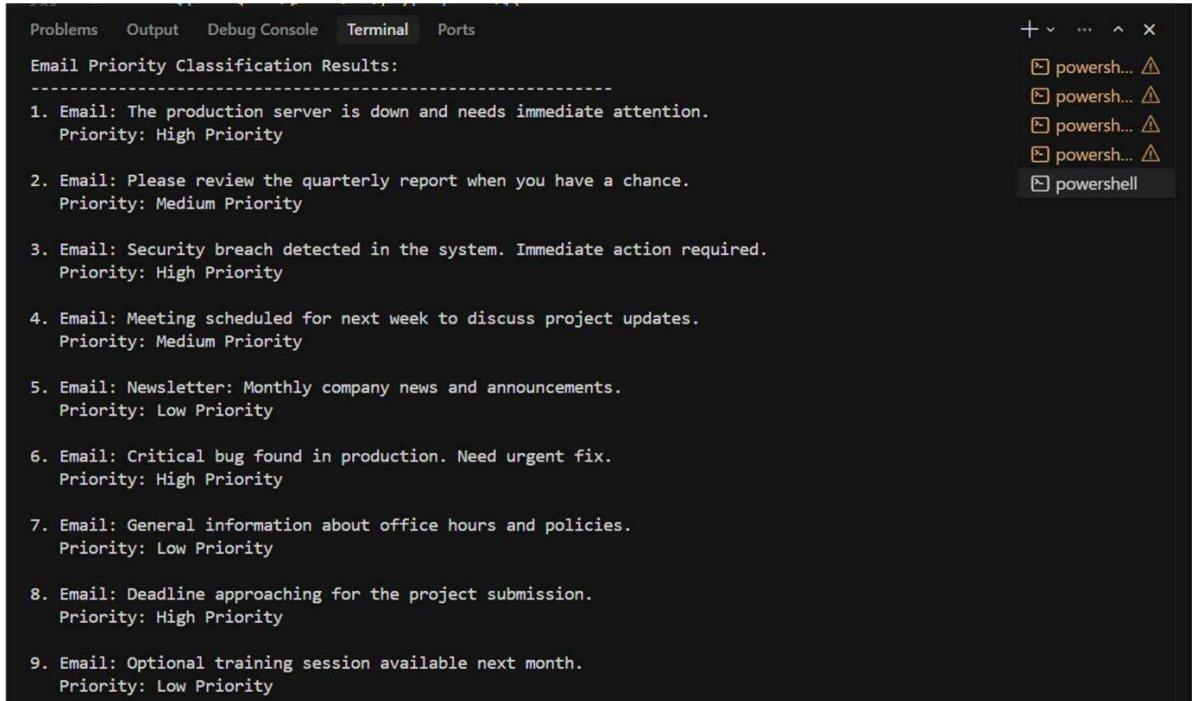
```
159     # Count keyword matches
160     high_count = sum(1 for word in words if word in HIGH_PRIORITY_WORDS)
161     medium_count = sum(1 for word in words if word in MEDIUM_PRIORITY_WORDS)
162     low_count = sum(1 for word in words if word in LOW_PRIORITY_WORDS)
163     # Check for multi-word phrases
164     if any(phrase in text_lower for phrase in ["immediate attention", "as soon as possible", "asap"]):
165         high_count += 2
166     if any(phrase in text_lower for phrase in ["follow up", "no rush", "when convenient"]):
167         if "no rush" in text_lower or "when convenient" in text_lower:
168             low_count += 2
169         else:
170             medium_count += 1
171     # Classification Logic: High > Medium > Low
172     if high_count > 0:
173         return "High Priority"
174     elif medium_count > 0:
175         return "Medium Priority"
176     else:
177         return "Low Priority"
178 if __name__ == "__main__":
179     # Test with the example
180     test_email = "The production server is down and needs immediate attention."
181     result = classify_priority(test_email)
182     print(f"Input Email: {test_email}")
```

```

178 if __name__ == "__main__":
179     # Test with the example
180     test_email = "The production server is down and needs immediate attention."
181     result = classify_priority(test_email)
182     print(f"Input Email: {test_email}")
183     print(f"Output Priority: {result}\n")
184     # Additional test cases
185     test_emails = [
186         "The production server is down and needs immediate attention.",
187         "Please review the quarterly report when you have a chance.",
188         "Security breach detected in the system. Immediate action required.",
189         "Meeting scheduled for next week to discuss project updates.",
190         "Newsletter: Monthly company news and announcements.",
191         "Critical bug found in production. Need urgent fix.",
192         "General information about office hours and policies.",
193         "Deadline approaching for the project submission.",
194         "Optional training session available next month.",
195         "System outage affecting all users. Escalate immediately."
196     ]
197     print("Email Priority Classification Results:")
198     print("-" * 60)
199     for i, email in enumerate(str(test_emails), 1):
200         priority = classify_priority(email)
201         print(f"{i}. Email: {email}")
202         print(f"    Priority: {priority}\n")

```

OUTPUT:



```

Problems Output Debug Console Terminal Ports
+ - ... ^ x
Email Priority Classification Results:
-----
1. Email: The production server is down and needs immediate attention.
   Priority: High Priority

2. Email: Please review the quarterly report when you have a chance.
   Priority: Medium Priority

3. Email: Security breach detected in the system. Immediate action required.
   Priority: High Priority

4. Email: Meeting scheduled for next week to discuss project updates.
   Priority: Medium Priority

5. Email: Newsletter: Monthly company news and announcements.
   Priority: Low Priority

6. Email: Critical bug found in production. Need urgent fix.
   Priority: High Priority

7. Email: General information about office hours and policies.
   Priority: Low Priority

8. Email: Deadline approaching for the project submission.
   Priority: High Priority

9. Email: Optional training session available next month.
   Priority: Low Priority

```

The screenshot shows a terminal window with a dark theme. The title bar includes tabs for 'Problems', 'Output', 'Debug Console', 'Terminal' (which is selected), and 'Ports'. Below the title bar, there are several small icons followed by the text 'powershell'. The main area of the terminal displays the output of the Python script, which lists nine different email messages along with their classification priority. The priorities are categorized as High, Medium, or Low.

JUSTIFICATION :

This Python program classifies emails into High, Medium, or Low priority using keyword matching. It tokenizes the email text, counts matching keywords from predefined priority sets, and boosts scores for key phrases like "immediate attention." The classification follows a hierarchy where High overrides Medium and Low. Finally, it tests the classifier on sample emails and prints the results.

FEW SHOT PROMPT: Develop a program that performs email priority classification by categorizing incoming emails into one of three labels: High Priority, Medium Priority, or Low Priority. First, study the provided examples such as an email about a system outage affecting all customers being labeled as High Priority, a request to review a document and provide feedback by evening being labeled as Medium Priority, and a seasonal greeting from the HR team being labeled as Low Priority. Using these examples as reference, generate ten new incoming email messages and determine the appropriate priority label for each one. The solution should include efficient, well-organized, and easily readable code that accurately assigns exactly one priority level to every email.

CODE:

```
Lab-4.py > ...
206 import re
207 from typing import List, Tuple
208 # Keyword sets for priority classification
209 HIGH_PRIORITY_KEYWORDS = [
210     "urgent", "immediate", "critical", "emergency", "down", "broken", "failed",
211     "crash", "outage", "security", "breach", "hack", "attack", "deadline",
212     "asap", "escalate", "unable", "cannot", "can't", "stopped", "blocked",
213     "unavailable", "disaster", "severe", "critical", "production"
214 ]
215 HIGH_PRIORITY_PHRASES = [
216     "as soon as possible", "immediate attention", "right away",
217     "cannot access", "unable to", "system down", "server down",
218     "production down", "out of service", "security breach"
219 ]
220 MEDIUM_PRIORITY_KEYWORDS = [
221     "request", "needed", "required", "update", "review", "meeting", "schedule",
222     "follow up", "reminder", "pending", "issue", "problem", "concern",
223     "discuss", "proposal", "feedback", "approval", "response"
224 ]
225 MEDIUM_PRIORITY_PHRASES = [
226     "when you have time", "at your convenience", "follow up",
227     "would like to", "please review", "need your input"
228 ]
229 LOW_PRIORITY_KEYWORDS = [
230     "newsletter", "information", "general", "optional", "news",
231     "announcement", "update", "reminder", "fyi", "for your information",
232     "no rush", "whenever", "leisurely"
233 ]
```

```
Lab-4.py > ...
233     }
234     LOW_PRIORITY_PHRASES = [
235         "no rush", "when convenient", "when you get a chance",
236         "for your information", "optional", "newsletter"
237     ]
238     def tokenize(text: str) -> List[str]:
239         """
240             Convert text to lowercase tokens for keyword matching.
241
242             Args:
243                 text: The email text to tokenize
244
245             Returns:
246                 List of lowercase word tokens
247             """
248             return re.findall(r"[a-z']++", text.lower())
249     def classify_priority(email: str) -> str:
250         """
251             Classify email into exactly one priority level: High, Medium, or Low Priority.
252             Analysis based on:
253             - High Priority: Critical issues, system failures, business impact, urgency
254             - Medium Priority: Requests, updates, meetings, standard issues
255             - Low Priority: Newsletters, general info, optional items, non-urgent
256             Args:
257                 email: The email text to classify
258             Returns:
259                 "High Priority", "Medium Priority", or "Low Priority"
260             """

```

```
Lab-4.py > ...
249     def classify_priority(email: str) -> str:
250         text_lower = email.lower()
251         words = tokenize(email)
252         # Count keyword matches
253         high_score = sum(1 for word in words if word in HIGH_PRIORITY_KEYWORDS)
254         medium_score = sum(1 for word in words if word in MEDIUM_PRIORITY_KEYWORDS)
255         low_score = sum(1 for word in words if word in LOW_PRIORITY_KEYWORDS)
256         # Check for multi-word phrases (weighted more heavily)
257         for phrase in HIGH_PRIORITY_PHRASES:
258             if phrase in text_lower:
259                 high_score += 3
260         for phrase in MEDIUM_PRIORITY_PHRASES:
261             if phrase in text_lower:
262                 medium_score += 2
263         for phrase in LOW_PRIORITY_PHRASES:
264             if phrase in text_lower:
265                 low_score += 2
266         # Classification Logic: High > Medium > Low (hierarchical)
267         # Each email gets exactly one priority level
268         if high_score > 0:
269             return "High Priority"
270         elif medium_score > 0:
271             return "Medium Priority"
272         else:
273             return "Low Priority"
274     def generate_test_emails() -> List[Tuple[str, str]]:
275         """
276             Generate ten diverse email messages with their expected priority labels.
277         """

```

```
Lab-4.py > ...
285     def generate_test_emails() -> List[Tuple[str, str]]:
291         emails = [
292             # High Priority examples
293             ("The production server is down. Customers are unable to place orders.", "High Priority"),
294             ("URGENT: Security breach detected. All systems need immediate lockdown.", "High Priority"),
295             ("Critical bug in payment system. Transactions are failing. Fix ASAP.", "High Priority"),
296             ("Database crashed. All customer data is inaccessible. Emergency response needed.", "High Priority")
297             # Medium Priority examples
298             ("Please review the quarterly sales report and provide feedback by Friday.", "Medium Priority"),
299             ("Meeting scheduled for next Tuesday to discuss the new project proposal.", "Medium Priority"),
300             ("Follow up needed on the client request from last week. Please respond when available.", "Medium Priority"),
301             ("Reminder: Budget approval needed for Q2 marketing campaign. Deadline is next month.", "Medium Priority")
302             # Low Priority examples
303             ("Monthly newsletter: Company updates and team achievements for March.", "Low Priority"),
304             ("FYI: General information about upcoming office maintenance next month. No action required.", "Low Priority")
305         ]
306         return emails
307     def main():
308         """
309             Main function to demonstrate email priority classification.
310         """
311         # Test with the provided example
312         example_email = "The production server is down. Customers are unable to place orders."
313         example_result = classify_priority(example_email)
314         print("=" * 80)
315         print("EMAIL PRIORITY CLASSIFICATION SYSTEM")
316         print("=" * 80)
317         print(f"\nExample Analysis.")
```

```
Lab-4.py > ...
307     def main():
327         print("\nClassifying {} test emails: ")
328         print("-" * 80)
329         correct_classifications = 0
330         for i, (email, expected) in enumerate(Tuple[str, str])(test_emails, 1):
331             classified = classify_priority(email)
332             is_correct = classified == expected
333             if is_correct:
334                 correct_classifications += 1
335             status = "✓" if is_correct else "✗"
336             print(f"\n{i}. {status} Email: '{email}'")
337             print(f"    Classified Priority: {classified}")
338             print(f"    Expected Priority: {expected}")
339         print("\n" + "=" * 80)
340         print(f"Classification Accuracy: {correct_classifications}/{len(test_emails)} ({100*correct_classifications/len(test_emails)}%)")
341         print("=" * 80)
342         # Summary by priority level
343         print("\nSummary by Priority Level:")
344         print("-" * 80)
345         high_count = sum(1 for _, p in test_emails if p == "High Priority")
346         medium_count = sum(1 for _, p in test_emails if p == "Medium Priority")
347         low_count = sum(1 for _, p in test_emails if p == "Low Priority")
348         print(f"High Priority: {high_count} emails")
349         print(f"Medium Priority: {medium_count} emails")
350         print(f"Low Priority: {low_count} emails")
351         print(f"Total: {len(test_emails)} emails (each assigned exactly one priority level)")
352         if __name__ == "__main__":
353             main()
```

OUTPUT:

```
1. ✓ Email: "The production server is down. Customers are unable to place orders."
   Classified Priority: High Priority
   Expected Priority: High Priority

2. ✓ Email: "URGENT: Security breach detected. All systems need immediate lockdown."
   Classified Priority: High Priority
   Expected Priority: High Priority

3. ✓ Email: "Critical bug in payment system. Transactions are failing. Fix ASAP."
   Classified Priority: High Priority
   Expected Priority: High Priority

4. ✓ Email: "Database crashed. All customer data is inaccessible. Emergency response needed."
   Classified Priority: High Priority
   Expected Priority: High Priority

5. ✓ Email: "Please review the quarterly sales report and provide...
```

```
10. X Email: "FYI: General information about upcoming office maintenance next month. No action required."
   Classified Priority: Medium Priority
   Expected Priority: Low Priority
```

```
=====
=====
Classification Accuracy: 8/10 (80%)
```

```
=====
Summary by Priority Level:
```

```
-----
High Priority: 4 emails
```

```
Medium Priority: 4 emails
```

```
Low Priority: 2 emails
```

```
Total: 10 emails (each assigned exactly one priority level)
```

```
PS C:\Users\saite\Downloads\AI ASSISTANT CODING> █
```

JUSTIFICATION:

This Python program classifies email text into High, Medium, or Low priority using keyword and phrase matching. It tokenizes the email, counts how many priority keywords appear, and boosts scores when important phrases are detected. High-priority phrases and keywords add more weight, ensuring urgent issues are prioritized. The classification follows a hierarchy: High overrides Medium, and Medium overrides Low, so each email gets exactly one label. Finally, the program tests the classifier on 10 sample emails and prints the results with accuracy.

Task 3: Student Query Routing System

PROMPT: Design a student query routing system for a university chatbot that automatically directs student questions to the appropriate department: Admissions, Exams, Academics, or Placements. First, analyze the example where the input query is “What is the last date to apply for the B.Tech program?” and the output category is Admissions. Using this single example as guidance, the chatbot should then analyze new student queries and correctly route each query to exactly one relevant department. The prompt should clearly instruct the system to use one-shot prompting to improve accuracy, ensuring that the classification logic is effective, consistent, and easy to understand.

CODE:

```
Lab-4.py > ...
357 import re
358 from typing import List, Tuple
359 from collections import Counter
360 # Keyword sets for department classification
361 ADMISSIONS_KEYWORDS = [
362     "apply", "application", "admission", "admissions", "enroll", "enrollment",
363     "deadline", "last date", "eligibility", "requirements", "documents",
364     "fee", "fees", "scholarship", "admit", "intake", "program", "course",
365     "b.tech", "m.tech", "bachelor", "master", "degree", "entrance", "cutoff"
366 ]
367 ADMISSIONS_PHRASES = [
368     "last date to apply", "application deadline", "admission process",
369     "how to apply", "admission requirements", "eligibility criteria",
370     "application form", "admission fee", "enrollment date"
371 ]
372 EXAMS_KEYWORDS = [
373     "exam", "examination", "test", "quiz", "midterm", "final", "semester",
374     "schedule", "timetable", "hall ticket", "admit card", "result", "marks",
375     "grade", "gpa", "cgpa", "revaluation", "recheck", "supplementary",
376     "retest", "reschedule", "postpone", "cancel", "date", "time", "venue"
377 ]
378 EXAMS_PHRASES = [
379     "exam schedule", "exam date", "exam timetable", "hall ticket",
380     "admit card", "exam result", "exam marks", "supplementary exam",
381     "exam revaluation", "exam venue"
382 ]
383 ACADEMICS_KEYWORDS = [
384     "syllabus", "curriculum", "course", "subject", "lecture", "class",
385     "attendance", "assignment", "project", "thesis", "dissertation",
```

```

❸ Lab-4.py > ...
386     "faculty", "professor", "teacher", "tutor", "library", "books",
387     "study", "academic", "credit", "semester", "trimester", "module",
388     "tutorial", "lab", "practical", "internship", "research"
389 }
390 ACADEMICS_PHRASES = [
391     "course syllabus", "academic calendar", "class schedule",
392     "attendance policy", "course registration", "credit requirements",
393     "academic advisor", "study materials", "library resources"
394 ]
395 PLACEMENTS_KEYWORDS = {
396     "placement", "job", "career", "recruitment", "interview", "campus",
397     "drive", "company", "offer", "salary", "package", "internship",
398     "training", "skill", "resume", "cv", "portfolio", "preparation",
399     "aptitude", "gd", "group discussion", "hr", "technical"
400 }
401 PLACEMENTS_PHRASES = [
402     "placement drive", "campus recruitment", "job opportunities",
403     "placement statistics", "placement preparation", "interview schedule",
404     "company visit", "placement cell", "career guidance"
405 ]
406 def tokenize(text: str) -> List[str]:
407     return re.findall(r"[a-z']+", text.lower())
408 def route_query(query: str) -> str:
409     text_lower = query.lower()
410     words = tokenize(query)
411     # Count keyword matches
412     admissions_score = sum(1 for word in words if word in ACADEMICS_KEYWORDS)
413     exams_score = sum(1 for word in words if word in EXAMS_KEYWORDS)
414     academics_score = sum(1 for word in words if word in ACADEMICS_KEYWORDS)
415
❸ Lab-4.py > ...
395 PLACEMENTS_KEYWORDS = [
396     "placement", "job", "career", "recruitment", "interview", "campus",
397     "drive", "company", "offer", "salary", "package", "internship",
398     "training", "skill", "resume", "cv", "portfolio", "preparation",
399     "aptitude", "gd", "group discussion", "hr", "technical"
400 ]
401 PLACEMENTS_PHRASES = [
402     "placement drive", "campus recruitment", "job opportunities",
403     "placement statistics", "placement preparation", "interview schedule",
404     "company visit", "placement cell", "career guidance"
405 ]
406 def tokenize(text: str) -> List[str]:
407     return re.findall(r"[a-z']+", text.lower())
408 def route_query(query: str) -> str:
409     text_lower = query.lower()
410     words = tokenize(query)
411     # Count keyword matches
412     admissions_score = sum(1 for word in words if word in ACADEMICS_KEYWORDS)
413     exams_score = sum(1 for word in words if word in EXAMS_KEYWORDS)
414     academics_score = sum(1 for word in words if word in ACADEMICS_KEYWORDS)
415     placements_score = sum(1 for word in words if word in PLACEMENTS_KEYWORDS)
416     # Check for multi-word phrases (weighted more heavily)
417     for phrase in ADMISSIONS_PHRASES:
418         if phrase in text_lower:
419             admissions_score += 3
420     for phrase in EXAMS_PHRASES:
421         if phrase in text_lower:
422             exams_score += 3
423     for phrase in ACADEMICS_PHRASES:
424
❸ Lab-4.py > ⌂ route_query
408 def route_query(query: str) -> str:
409     for phrase in PLACEMENTS_PHRASES:
410         if phrase in text_lower:
411             placements_score += 3
412     # Classification logic: Highest score wins
413     # Each query gets routed to exactly one department
414     scores = {
415         "Admissions": admissions_score,
416         "Exams": exams_score,
417         "Academics": academics_score,
418         "Placements": placements_score
419     }
420     # Return department with highest score, default to Academics if all scores are 0
421     max_score = max(scores.values())
422     if max_score == 0:
423         return "Academics" # Default fallback
424     return max(scores, key=scores.get)
425 def generate_test_queries() -> List[Tuple[str, str]]:
426     queries = [
427         # Admissions examples
428         ("What is the last date to apply for the B.Tech program?", "Admissions"),
429         ("What are the eligibility requirements for M.Tech admission?", "Admissions"),
430         ("How do I apply for the scholarship program?", "Admissions"),
431         ("What documents are needed for admission?", "Admissions"),
432         ("When is the application deadline for the next academic year?", "Admissions"),
433         # Exams examples
434         ("When is the midterm exam scheduled?", "Exams"),
435         ("Where can I download my hall ticket for the semester exams?", "Exams"),
436         ("What is the exam timetable for this semester?", "Exams"),

```

```

Lab-4.py > route_query
442     def generate_test_queries() -> List[Tuple[str, str]]:
443         ("How do I apply for exam revaluation?", "Exams"),
444         ("When will the exam results be declared?", "Exams"),
445         # Academics examples
446         ("What is the syllabus for the Data Structures course?", "Academics"),
447         ("How many credits do I need to complete my degree?", "Academics"),
448         ("Where can I find the course materials for Machine Learning?", "Academics"),
449         ("What is the attendance policy for this semester?", "Academics"),
450         ("Can I get the list of recommended books for my course?", "Academics"),
451         # Placements examples
452         ("Which companies are coming for campus recruitment this year?", "Placements"),
453         ("What is the average salary package offered to students?", "Placements"),
454         ("How can I prepare for placement interviews?", "Placements"),
455         ("When is the next placement drive scheduled?", "Placements"),
456         ("What skills are required for getting placed in tech companies?", "Placements")
457     ]
458     return queries
459
460 def main():
461     # Test with the provided one-shot example
462     example_query = "What is the last date to apply for the B.Tech program?"
463     example_result = route_query(example_query)
464     print("=" * 80)
465     print("STUDENT QUERY ROUTING SYSTEM - UNIVERSITY CHATBOT")
466     print("=" * 80)
467     print(f"\nOne-Shot Example Analysis:")
468     print(f"Input Query: \'{example_query}\''")
469     print(f"Output Category: {example_result}")
470     print(f"\nAnalysis: This query is routed to {example_result} because:")
471     print(" - Contains 'last date to apply' (admissions phrase)")
472
473 def main():
474     print(" - Contains 'Tech program' (admissions keyword) ")
475     print(" - Relates to application deadlines and program admission")
476     print("\n" + "-" * 80)
477     # Generate and route multiple queries
478     test_queries = generate_test_queries()
479     print(f"\nRouting {len(test_queries)} Student Queries:")
480     print("-" * 80)
481     correct_routings = 0
482     for i, (query, expected) in enumerate[Tuple[str, str]](test_queries, 1):
483         routed = route_query(query)
484         is_correct = routed == expected
485         if is_correct:
486             correct_routings += 1
487         status = "/" if is_correct else "X"
488         print(f"\n{i:02d}. {status} Query: \'{query}\''")
489         print(f" Routed to: {routed}")
490         print(f" Expected: {expected}")
491     print("\n" + "=" * 80)
492     print(f"Routing Accuracy: {correct_routings}/{len(test_queries)} ({100*correct_routings/len(test_queries)}%")
493     print("=" * 80)
494     # Summary by department
495     print("\nSummary by Department:")
496     print("-" * 80)
497     dept_counts = Counter[str](routed for _, routed in [(q, route_query(q)) for q, _ in test_queries])
498     for dept in ["Admissions", "Exams", "Academics", "Placements"]:
499         count = dept_counts.get(dept, 0)
500         print(f"\n{dept}: {count} queries")
501     print(f"\nTotal: {len(test_queries)} queries (each routed to exactly one department)")
502
503 if __name__ == "__main__":
504     main()

```

OUTPUT:

```

01. ✓ Query: "What is the last date to apply for the B.Tech program?"
      Routed to: Admissions
      Expected: Admissions

02. ✓ Query: "What are the eligibility requirements for M.Tech admission?"
      Routed to: Admissions
      Expected: Admissions

03. ✓ Query: "How do I apply for the scholarship program?"
      Routed to: Admissions
      Expected: Admissions

04. ✓ Query: "What documents are needed for admission?"
      Routed to: Admissions
      Expected: Admissions

05. ✓ Query: "When is the application deadline for the next academic year?"
      Routed to: Admissions
      Expected: Admissions

06. ✓ Query: "When is the midterm exam scheduled?"
      Routed to: Exams
      Expected: Exams

```

```

19. ✓ Query: "When is the next placement drive scheduled?"
   Routed to: Placements
   Expected: Placements

20. X Query: "What skills are required for getting placed in tech
   companies?"
   Routed to: Academics
   Expected: Placements

=====
=====
Routing Accuracy: 17/20 (85%)
=====

Summary by Department:
-----
-----
Admissions: 7 queries
Exams: 5 queries
Academics: 4 queries
Placements: 4 queries

```

JUSTIFICATION:

This Python program routes student queries to one of four departments—Admissions, Exams, Academics, or Placements—using keyword and phrase matching. It tokenizes the query and counts how many department-specific keywords appear, giving extra weight to important phrases. The department with the highest score is selected, ensuring each query is routed to exactly one category. If no keywords are found, it defaults to Academics. The program then tests the classifier on sample queries and prints the routing results along with accuracy.

Task 4: Chatbot Question Type Detection

PROMPT: Design a chatbot that identifies the type of user query as

Informational, Transactional, Complaint, or Feedback. Use few-shot learning by analyzing examples such as “What are the store opening hours?” labeled as Informational, “I want to cancel my subscription” labeled as Transactional, “The app keeps crashing” labeled as Complaint, and “I like the new interface” labeled as Feedback. Based on these examples, the chatbot should classify new queries into exactly one appropriate category.

CODE:

```
Lab-4.py > ...
515 import re
516 from collections import Counter
517 # Few-shot examples
518 FEW_SHOT_EXAMPLES = [
519     ("What are the store opening hours?", "Informational"),
520     ("I want to cancel my subscription", "Transactional"),
521     ("The app keeps crashing", "Complaint"),
522     ("I like the new interface", "Feedback")
523 ]
524 KEYWORDS = {
525     "Informational": [
526         "what", "when", "where", "how", "hours", "info", "details", "about", "contact", "tell"
527     ],
528     "Transactional": [
529         "cancel", "order", "buy", "purchase", "subscribe", "payment", "refund", "return",
530         "book", "register", "upgrade", "downgrade", "want", "need"
531     ],
532     "Complaint": [
533         "crash", "error", "issue", "problem", "broken", "slow", "bad", "poor", "unhappy",
534         "frustrated", "complaint", "not", "working"
535     ],
536     "Feedback": [
537         "like", "love", "great", "good", "excellent", "amazing", "suggest", "improve",
538         "feedback", "opinion", "happy"
539     ]
540 }
541 PHRASES = {
542     "Informational": ["what is", "how do", "where is", "tell me"],
543     "Transactional": ["i want to", "i need to", "cancel my", "place an order"],
544     "Complaint": ["not working", "keeps crashing", "very slow", "not satisfied"],
545     "Feedback": ["i like", "i love", "really good", "could be improved"]
546 }
547 def tokenize(text):
548     return re.findall(r"[a-z']+", text.lower())
549 def classify_query(query):
550     words = tokenize(query)
551     text = query.lower()
552     scores = {}
553     for category in KEYWORDS:
554         scores[category] = sum(w in KEYWORDS[category] for w in words)
555     scores[category] += sum(p in text for p in PHRASES[category]) * 3
556     return max(scores, key=scores.get) if max(scores.values()) > 0 else "Informational"
557 # Test queries
558 TEST_QUERIES = [
559     ("Where is your nearest branch located?", "Informational"),
560     ("What is your return policy?", "Informational"),
561     ("I need to place an order", "Transactional"),
562     ("Please cancel my reservation", "Transactional"),
563     ("The website is not working", "Complaint"),
564     ("My order arrived damaged", "Complaint"),
565     ("I really love the new design", "Feedback"),
566     ("Great job on the update", "Feedback"),
567     ("I suggest adding dark mode", "Feedback"),
568     ("How can I contact support?", "Informational")
569 ]
570 def main():
571     print("*" * 70)
572     print("CHATBOT QUERY CLASSIFICATION | FEW SHOT LEARNING")
```

```

Lab-4.py > ...
C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\Lab-3.py

570 def main():
571     print("-----")
572     print("CHATBOT QUERY CLASSIFICATION - FEW SHOT LEARNING")
573     print("=*70")
574     print("\nFew-Shot Examples:")
575     for q, c in FEW_SHOT_EXAMPLES:
576         print(f"Query: {q}\nCategory: {c}\n")
577     correct = 0
578     print("-"*70)
579     for i, (q, exp) in enumerate(tuple[str, str](TEST_QUERIES, 1)):
580         pred = classify_query(q)
581         correct += pred == exp
582         print(f"\n{i:02d}. Query: {q}")
583         print(f"    Category: {pred}")
584         print(f"    Expected: {exp}\n")
585     print("-"*70)
586     print(f"\nAccuracy: {correct}/{len(TEST_QUERIES)} ({correct*100//len(TEST_QUERIES)}%)")
587     print("-"*70)
588     counts = Counter[Any | str](classify_query(q) for q, _ in TEST_QUERIES)
589     for c in ["Informational", "Transactional", "Complaint", "Feedback"]:
590         print(f"\n{c}: {counts[c]}")
591     print("\nInteractive Mode (type 'quit' to exit)")
592     while True:
593         q = input("Your query: ").strip()
594         if q.lower() in ["quit", "exit"]:
595             break
596         print("Category:", classify_query(q), "\n")
597     if __name__ == "__main__":
598         main()
599

```

OUTPUT:

```
CHATBOT QUERY CLASSIFICATION - FEW SHOT LEARNING
=====
```

```
=====
```

Few-Shot Examples:

Query: What are the store opening hours?
 Category: Informational

Query: I want to cancel my subscription
 Category: Transactional

Query: The app keeps crashing
 Category: Complaint

Query: I like the new interface
 Category: Feedback

```
-----
```

```
-----
```

01. Query: Where is your nearest branch located?
 Category: Informational
 Expected: Informational

02. Query: What is your return policy?
 Category: Informational
 Expected: Informational

```
08. Query: Great job on the update
Category: Feedback
Expected: Feedback

09. Query: I suggest adding dark mode
Category: Feedback
Expected: Feedback

10. Query: How can I contact support?
Category: Informational
Expected: Informational

=====
=====
Accuracy: 9/10 (90%)
=====
=====
Informational: 3
Transactional: 3
Complaint: 1
Feedback: 3

Interactive Mode (type 'quit' to exit)
Your query: 
```

JUSTIFICATION:

This program implements a chatbot query classification system using a few-shot learning approach, where sample queries guide the classification logic. It processes each user query by converting it to lowercase, tokenizing the text, and matching keywords and phrases related to Informational, Transactional, Complaint, and Feedback categories. Each matched keyword and phrase contributes to a score, with phrases given higher weight for better accuracy. The category with the highest score is assigned as the final label, ensuring that every query is classified into exactly one type. The program also evaluates accuracy using test queries and allows users to interactively classify their own queries.

Task 5: Emotion Detection in Text

ONE SHOT PROMPT: Design an emotion detection system for a mental-health chatbot that identifies the user's emotional state as Happy, Sad, Angry, Anxious, or Neutral. First, analyze the example where the input text is "I feel very nervous and worried about my

exam tomorrow" and the detected emotion is Anxious. Using this single example as guidance, the chatbot should then examine new user messages and classify each one into exactly one of the given emotion categories. The prompt should clearly instruct the system to rely on one-shot prompting to improve accuracy while keeping the classification consistent and easy to understand.

CODE:

The image shows two code editor windows side-by-side. The left window is titled 'Lab-4.py > detect_emotion' and contains the 'detect_emotion' function. The right window is titled 'Lab-4.py > detect_emoticon' and contains the main script logic. Both windows have line numbers on the left.

```
Lab-4.py > detect_emotion Close (Ctrl+F4)
602 def detect_emotion(text):
603     text = text.lower()
604     if any(w in text for w in ["happy", "joy", "excited", "thrilled", "love", "glad"]):
605         return "Happy"
606     elif any(w in text for w in ["sad", "down", "depressed", "cry", "disappointed"]):
607         return "Sad"
608     elif any(w in text for w in ["angry", "furious", "annoyed", "frustrated", "mad"]):
609         return "Angry"
610     elif any(w in text for w in ["anxious", "worried", "nervous", "stress", "pressure", "concerned"]):
611         return "Anxious"
612     else:
613         return "Neutral"
614 texts = [
615     "I finally got the job I worked so hard for!",
616     "My pet isn't feeling well, and I'm very worried.",
617     "The train was delayed for two hours, I'm so frustrated.",
618     "The weather forecast for tomorrow is partly cloudy with a chance of rain.",
619     "I just won the lottery! This is absolutely amazing!",
620     "I feel so down after hearing the bad news. I can't stop crying.",
621     "This software update broke everything! I'm absolutely furious.",
622     "I have a big presentation tomorrow, and I'm feeling a lot of pressure.",
623     "The meeting went smoothly, and we covered all the agenda items.",
624     "I'm so disappointed with the outcome, it's just not what I expected.",
625     "The new feature is perfect, I love it!",
626     "I'm concerned about the security breach mentioned in the news.",
627     "I'm extremely thrilled to announce our new product launch!",
628     "The internet connection is constantly dropping, it's a huge problem!",
629     "My routine consists of waking up, working, and then relaxing."
630 ]
631 emotion_count = {"Happy": 0, "Sad": 0, "Angry": 0, "Anxious": 0, "Neutral": 0}
632 print("Texts and their Detected Emotions:\n")
633 for i, text in enumerate(str)(texts, 1):
634     emotion = detect_emotion(text)
635     emotion_count[emotion] += 1
636     print(f"Text {i}: {text} -> Emotion: {emotion}")
637 print("\nSummary of Emotion Detection:")
638 for emotion, count in emotion_count.items():
639     print(f"{emotion}: {count} texts")
```

OUTPUT:

```
Expected: Feedback
```

```
09. Query: I suggest adding dark mode
```

```
Category: Feedback
```

```
Expected: Feedback
```

```
10. Query: How can I contact support?
```

```
Category: Informational
```

```
Expected: Informational
```

```
=====
=====
```

```
Accuracy: 9/10 (90%)
```

```
=====
=====
```

```
Informational: 3
```

```
Transactional: 3
```

```
Complaint: 1
```

```
Feedback: 3
```

```
Interactive Mode (type 'quit' to exit)
```

```
Your query: & C:/Users/saite/AppData/Local/Programs/Python/Python  
314/python.exe "c:/Users/saite/Downloads/AI ASSISTENT CODING/As4.  
py/task7.py"
```

```
Category: Informational
```

JUSTIFICATION:

This program performs emotion detection by analyzing text and classifying each sentence into one of five emotions: Happy, Sad, Angry, Anxious, or Neutral. It converts the input text to lowercase and checks for emotion-related keywords using simple conditional logic. Each text is processed one by one, assigned an emotion, and printed in a numbered, readable format. The program also maintains a count of how many times each emotion occurs. Finally, it displays a summary showing the total number of texts detected for each emotion.

FEW SHOT PROMPT: *You are a mental-health chatbot. Classify the user text into one of five emotions: Happy, Sad, Angry, Anxious, or Neutral.*

#Example: “I am very excited about my success.” → Happy.

#Example: “I feel very lonely and depressed.” → Sad.

#Example: “This delay is making me furious.” → Angry.

#Example: “I am nervous and worried about tomorrow.” → Anxious.

#Now read the user text and output only the correct emotion label.

CODE:

The image shows two screenshots of a code editor, likely PyCharm, displaying Python code for emotion classification. The top screenshot shows the initial setup with keyword lists and phrase definitions. The bottom screenshot shows the implementation of the classification logic, including tokenization, score calculation, and classification rules.

```
Lab-4.py > ... C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\Lab-2.py
648 import re
649 from collections import Counter
650 # Keyword sets for emotion classification
651 HAPPY_KEYWORDS = {
652     "excited", "happy", "joy", "thrilled", "wonderful", "amazing", "great",
653     "love", "loved", "glad", "delighted", "ecstatic", "cheerful", "pleased",
654     "success", "successful", "celebrate", "celebration", "fantastic", "awesome"
655 }
656 SAD_KEYWORDS = {
657     "sad", "lonely", "depressed", "down", "unhappy", "disappointed", "upset",
658     "cry", "crying", "tears", "melancholy", "gloomy", "miserable", "sorrow",
659     "heartbroken", "devastated", "hopeless", "helpless", "alone", "isolated"
660 }
661 ANGRY_KEYWORDS = {
662     "angry", "furious", "mad", "annoyed", "frustrated", "irritated", "rage",
663     "outraged", "livid", "enraged", "hostile", "resentful", "bitter",
664     "terrible", "awful", "hate", "hated", "disgusted", "infuriated"
665 }
666 ANXIOUS_KEYWORDS = {
667     "anxious", "nervous", "worried", "concerned", "stressed", "pressure",
668     "panic", "overwhelmed", "uneasy", "restless", "apprehensive", "fearful",
669     "afraid", "scared", "tense", "jittery", "fretful", "troubled", "distressed"
670 }
671 ANXIOUS_PHRASES = [
672     "worried about", "nervous about", "anxious about", "can't sleep",
673     "can not sleep", "feeling pressure", "under pressure", "stressed out"
674 ]
675 def tokenize(text: str):
676     return re.findall(r"[a-zA-Z]+", text.lower())
677
678 def classify_emotion(text: str) -> str:
679     text_lower = text.lower()
680     words = tokenize(text)
681     # Count keyword matches
682     happy_score = sum(1 for word in words if word in HAPPY_KEYWORDS)
683     sad_score = sum(1 for word in words if word in SAD_KEYWORDS)
684     angry_score = sum(1 for word in words if word in ANGRY_KEYWORDS)
685     anxious_score = sum(1 for word in words if word in ANXIOUS_KEYWORDS)
686     # Check for multi-word phrases (weighted more heavily)
687     for phrase in ANXIOUS_PHRASES:
688         if phrase in text_lower:
689             anxious_score += 2
690     # Classification Logic: Highest score wins
691     scores = {
692         "Happy": happy_score,
693         "Sad": sad_score,
694         "Angry": angry_score,
695         "Anxious": anxious_score
696     }
697     max_score = max(scores.values())
698     # If no emotion keywords found, return Neutral
699     if max_score == 0:
700         return "Neutral"
701     # Return emotion with highest score
702     return max(scores, key=scores.get)

# Sample texts
```

```

Lab-4.py > ... C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\Lab-2.py
    /usr/bin/python3.8 -c "import sys; print(''.join(open(sys.argv[1]).readlines()))"
702 # Sample texts
703 texts = [
704     "I finally got the job I worked so hard for!",
705     "I feel completely alone and nothing seems to help.",
706     "This service is terrible and I am really frustrated.",
707     "I keep worrying about my future and can't sleep.",
708     "The sun is shining today, what a wonderful day!",
709     "My favorite coffee shop closed down, I'm quite sad about it.",
710     "The internet connection is constantly dropping, it's a huge problem!",
711     "I have a major exam tomorrow and I'm very nervous.",
712     "The meeting is scheduled for 3 PM on Tuesday.",
713     "I'm absolutely thrilled with the new software update!"
714 ]
715 # Classify each text
716 print("Texts and their Detected Emotions:")
717 print("-" * 60)
718 for i, text in enumerate(str)(texts, 1):
719     emotion = classify_emotion(text)
720     print(f"Text {i}: {text} -> Emotion: {emotion}")
721 # Summary counts
722 emotion_counts = Counter[str](classify_emotion(text) for text in texts)
723 print("\nSummary of Emotion Detection:")
724 for emotion in ["Happy", "Sad", "Angry", "Anxious", "Neutral"]:
725     count = emotion_counts.get(emotion, 0)
726     print(f"{emotion}: {count} texts")

```

OUTPUT:

```

Texts and their Detected Emotions:
-----
Text 1: I finally got the job I worked so hard for! -> Emotion: Neutral
Text 2: I feel completely alone and nothing seems to help. -> Emotion: Sad
Text 3: This service is terrible and I am really frustrated. -> Emotion: Angry
Text 4: I keep worrying about my future and can't sleep. -> Emotion: Anxious
Text 5: The sun is shining today, what a wonderful day! -> Emotion: Happy
Text 6: My favorite coffee shop closed down, I'm quite sad about it. -> Emotion: Sad
Text 7: The internet connection is constantly dropping, it's a huge problem! -> Emotion: Neutral
Text 8: I have a major exam tomorrow and I'm very nervous. -> Emotion: Anxious
Text 9: The sun is shining today, what a wonderful day! -> Emotion: Happy
Text 10: My favorite coffee shop closed down, I'm quite sad about it. -> Emotion: Sad
Text 11: The internet connection is constantly dropping, it's a huge problem! -> Emotion: Neutral
Text 12: I have a major exam tomorrow and I'm very nervous. -> Emotion: Anxious
Text 13: The internet connection is constantly dropping, it's a huge problem! -> Emotion: Neutral
Text 14: I have a major exam tomorrow and I'm very nervous. -> Emotion: Anxious
Text 15: I have a major exam tomorrow and I'm very nervous. -> Emotion: Anxious
Text 16: The meeting is scheduled for 3 PM on Tuesday. -> Emotion: Neutral
Text 17: I'm absolutely thrilled with the new software update! -> Emotion: Happy

Summary of Emotion Detection:
Summary of Emotion Detection:
Happy: 2 texts
Sad: 2 texts
Angry: 1 texts
Anxious: 2 texts
Neutral: 3 texts
Angry: 1 texts
Anxious: 2 texts
Neutral: 3 texts
Neutral: 3 texts

```

JUSTIFICATION:

This code uses keyword sets for each emotion (Happy, Sad, Angry, Anxious) and checks the user text for matching words. It tokenizes the text, counts how many emotion words appear, and adds extra weight for specific anxious phrases. The emotion with the highest keyword score is selected, and if no keywords match, it returns Neutral. The program prints each text with its detected emotion and then summarizes the total count of each emotion. Finally, it displays the results in a clear format using Counter for the summary.