

## 1 Example: A Comprehensive PMP Program

The Python program below demonstrates a complete PMP application that:

- Normalizes text using robust, adaptive methods.
  - Tokenizes the normalized text.
  - Computes detailed metrics and renders a polished, visually appealing graph.

This implementation encapsulates internal states and process functions within modular components, which are composed via dependency injection to form a coherent, adaptive system.

```

1 import unicodedata
2 from typing import Tuple, List, Dict, Any, Optional
3 import matplotlib.pyplot as plt
4
5 """
6 Processual Modular Programming (PMP) Paradigm:
7 Robust Text Analysis from Real-World Sources
8
9 This script demonstrates PMP via a text processing pipeline designed for
10 unstructured, real-world text. The pipeline is engineered for robustness
11 and
12 performance, effectively processing chaotic text from diverse sources. It
13 shows:
14 - Well-defined interfaces for modular interoperability.
15 - Encapsulated states within modules for clarity and maintainability.
16 - Process functions as explicit state transformations.
17 - Dynamic adaptation mechanisms that refine processing based on input.
18 - Modular composition via dependency injection for flexibility and
19 scalability.
20 """
21
22 # Module 1: Advanced Text Normalization Module
23 class TextNormalizer:
24     """
25         PMP module for advanced text normalization. Incorporates Unicode
26             normalization,
27             whitespace handling, and adaptive rule application.
28
29         Attributes:
30             state (Dict[str, Any]): Internal state including 'rules',
31                 'notes', and
32                 'adaptation_strategy'.
33     """
34     def __init__(
35         self,
36         rules: Optional[List[str]] = None,
37         adaptation_strategy: Optional[Dict[str, Any]] = None,
38     ) -> None:
39         """
40             Initializes with normalization rules and an adaptation strategy.
41             Defaults to Unicode NFC normalization and whitespace stripping.
42         """

```

```

38     default_rules = ["NFC", "strip_whitespace"]
39     self.state: Dict[str, Any] = {
40         "rules": rules if rules is not None else default_rules,
41         "notes": [],
42         "adaptation_strategy": (
43             adaptation_strategy if adaptation_strategy is not None
44             else {"short_text_threshold": 15}
45         ),
46     }
47
48     def process(self, input_text: str) -> Tuple[str, Dict[str, Any]]:
49         """
50             Processes input text with normalization and dynamic state
51             adaptation.
52         """
53         normalized_text = self.normalize(input_text)
54         text_length = len(normalized_text)
55         short_text_threshold = self.state["adaptation_strategy"].get(
56             "short_text_threshold", 15
57         )
58         note = f"Short text detected ({text_length} chars), consider
59             content sparsity."
60         if text_length < short_text_threshold:
61             if note not in self.state["notes"]:
62                 self.state["notes"].append(note)
63             else:
64                 if note in self.state["notes"]:
65                     self.state["notes"].remove(note)
66         return normalized_text, self.state.copy()
67
68     def normalize(self, text: str) -> str:
69         """
70             Applies the configured normalization rules to the text.
71         """
72         processed_text = text
73         if "strip_whitespace" in self.state["rules"]:
74             processed_text = processed_text.strip()
75         if "NFC" in self.state["rules"]:
76             processed_text = unicodedata.normalize("NFC", processed_text)
77         return processed_text
78
79 # Module 2: Intelligent Text Tokenizer Module
80 class Tokenizer:
81     """
82         PMP module for text tokenization with configurable delimiters.
83
84         Attributes:
85             state (Dict[str, Any]): Internal state including 'delimiter',
86                 'token_count',
87                 and 'tokenization_strategy'.
88         """
89     def __init__(
90         self,
91         delimiter: str = " ",

```

```

89         tokenization_strategy: Optional[Dict[str, Any]] = None,
90     ) -> None:
91         """
92             Initializes the Tokenizer with a delimiter and optional strategy.
93         """
94         self.state: Dict[str, Any] = {
95             "delimiter": delimiter,
96             "token_count": 0,
97             "tokenization_strategy": (
98                 tokenization_strategy if tokenization_strategy is not None
99                 else {}
100            ),
101        }
102
103    def process(self, text: str) -> Tuple[List[str], Dict[str, Any]]:
104        """
105            Tokenizes input text and updates token count.
106        """
107        tokens = self.tokenize(text)
108        self.state["token_count"] = len(tokens)
109        return tokens, self.state.copy()
110
111    def tokenize(self, text: str) -> List[str]:
112        """
113            Splits text into tokens using the configured delimiter.
114        """
115        return text.split(self.state["delimiter"])
116
117 # Module 3: Comprehensive Metrics Collector and Visualizer Module
118 class MetricsCollector:
119     """
120         PMP module for computing text metrics and visualizing the token
121         length distribution.
122
123     Attributes:
124         metrics (Dict[str, Any]): Contains computed metrics such as
125             average, maximum, and
126             minimum token lengths, token count, vocabulary size, and the
127             token list.
128     """
129     def __init__(self) -> None:
130         self.metrics: Dict[str, Any] = {}
131
132     def process(self, tokens: List[str]) -> Dict[str, Any]:
133         """
134             Computes various metrics based on tokens and updates internal
135             state.
136
137             if not tokens:
138                 self.metrics = {
139                     "average_token_length": 0,
140                     "max_token_length": 0,
141                     "min_token_length": 0,
142                     "token_count": 0,

```

```

139         "vocabulary_size": 0,
140         "tokens": tokens,
141     }
142 else:
143     token_lengths = [len(token) for token in tokens]
144     vocabulary = set(tokens)
145     self.metrics = {
146         "average_token_length": sum(token_lengths) /
147             len(token_lengths),
148         "max_token_length": max(token_lengths),
149         "min_token_length": min(token_lengths),
150         "token_count": len(tokens),
151         "vocabulary_size": len(vocabulary),
152         "tokens": tokens,
153     }
154 return self.metrics.copy()
155
156 def visualize(self) -> None:
157     """
158     Generates and displays a bar chart of token length distribution.
159     """
160     tokens = self.metrics.get("tokens", [])
161     lengths = [len(token) for token in tokens] if tokens else []
162     if not lengths:
163         print("No tokens available for visualization.")
164         return
165
166     plt.figure(figsize=(12, 7))
167     bars = plt.bar(
168         range(len(lengths)),
169         lengths,
170         color="mediumseagreen",
171         edgecolor="darkgreen",
172         alpha=0.85,
173     )
174     for bar in bars:
175         yval = bar.get_height()
176         plt.text(
177             bar.get_x() + bar.get_width() / 2,
178             yval + 0.5,
179             round(yval, 1),
180             ha="center",
181             va="bottom",
182             fontsize=8,
183             color="dimgray",
184         )
185     plt.xlabel("Token Index", fontsize=14, color="dimgray")
186     plt.ylabel("Token Length (Characters)", fontsize=14,
187                 color="dimgray")
188     plt.title(
189         "Distribution of Token Lengths in Processed Text",
190         fontsize=18,
191         fontweight="bold",
192         color="midnightblue",

```

```

191     )
192     plt.xticks(fontsize=11, color="gray")
193     plt.yticks(fontsize=11, color="gray")
194     plt.grid(axis="y", linestyle="--", alpha=0.7)
195     plt.tight_layout()
196     ax = plt.gca()
197     ax.spines["top"].set_visible(False)
198     ax.spines["right"].set_visible(False)
199     ax.set_facecolor("whitesmoke")
200     plt.show()
201
202 # PMP Pipeline: Composition via Dependency Injection.
203 def PMP_pipeline(input_text: str) -> Dict[str, Any]:
204     """
205     Orchestrates normalization, tokenization, and metrics analysis
206     in the PMP pipeline.
207
208     Returns a dictionary with:
209     - 'normalized_text'
210     - 'tokens'
211     - 'normalizer_state'
212     - 'tokenizer_state'
213     - 'metrics'
214     """
215     normalizer = TextNormalizer()
216     tokenizer = Tokenizer()
217     metrics_collector = MetricsCollector()
218     normalized_text, normalizer_state = normalizer.process(input_text)
219     tokens, tokenizer_state = tokenizer.process(normalized_text)
220     metrics = metrics_collector.process(tokens)
221     metrics_collector.visualize()
222     return {
223         "normalized_text": normalized_text,
224         "tokens": tokens,
225         "normalizer_state": normalizer_state,
226         "tokenizer_state": tokenizer_state,
227         "metrics": metrics,
228     }
229
230 if __name__ == "__main__":
231     """
232     Main execution demonstrating the PMP pipeline with various text
233     sources.
234     """
235     print("PMP Pipeline Demonstration: Real-World Text Analysis\n" + "=" * 60)
236     # Example 1: Social Media Text
237     raw_text_1 = (
238         "OMG! Just saw the most amazing sunset #nofilter #blessed. "
239         "Gonna grab a coffee now! What's everyone else up to?"
240     )
241     result_1 = PMP_pipeline(raw_text_1)
242     print("\nExample 1: Social Media Text Processing")
243     print("-" * 60)

```

```

243 print("Normalized Text:", f"'{result_1['normalized_text']}''")
244 print("Tokenized Output:", result_1["tokens"])
245 print("Normalizer State:", result_1["normalizer_state"])
246 print("Tokenizer State:", result_1["tokenizer_state"])
247 print("Computed Metrics:",
248     {k: v for k, v in result_1["metrics"].items() if k != "tokens"})
249
250 # Example 2: News Headline
251 raw_text_2 = ("Breaking News: Global Markets React to Unexpected "
252                 "Economic Data Release.")
253 result_2 = PMP_pipeline(raw_text_2)
254 print("\nExample 2: News Headline Processing")
255 print("-" * 60)
256 print("Normalized Text:", f"'{result_2['normalized_text']}''")
257 print("Tokenized Output:", result_2["tokens"])
258 print("Normalizer State:", result_2["normalizer_state"])
259 print("Tokenizer State:", result_2["tokenizer_state"])
260 print("Computed Metrics:",
261     {k: v for k, v in result_2["metrics"].items() if k != "tokens"})
262
263 # Example 3: Online Review Text
264 raw_text_3 = ("The new restaurant was okay, I guess. Service was
265                 kinda slow,
266                     "but the food was decent")
267 result_3 = PMP_pipeline(raw_text_3)
268 print("\nExample 3: Online Review Text Processing")
269 print("-" * 60)
270 print("Normalized Text:", f"'{result_3['normalized_text']}''")
271 print("Tokenized Output:", result_3["tokens"])
272 print("Normalizer State:", result_3["normalizer_state"])
273 print("Tokenizer State:", result_3["tokenizer_state"])
274 print("Computed Metrics:",
275     {k: v for k, v in result_3["metrics"].items() if k != "tokens"})
276
277 # Example 4: Telegraphic Text
278 raw_text_4 = "Urgent meeting tmrw 9am sharp!"
279 result_4 = PMP_pipeline(raw_text_4)
280 print("\nExample 4: Telegraphic Text & Dynamic Adaptation")
281 print("-" * 60)
282 print("Normalized Text:", f"'{result_4['normalized_text']}''")
283 print("Tokenized Output:", result_4["tokens"])
284 print("Normalizer State:", result_4["normalizer_state"])
285 print("Tokenizer State:", result_4["tokenizer_state"])
286 print("Computed Metrics:",
287     {k: v for k, v in result_4["metrics"].items() if k != "tokens"})
288
289 print("\n" + "=" * 60 + "\nEnd of PMP Demonstration: Real-World Text
Analysis")

```

Listing 1: A Comprehensive Text Processing Pipeline in PMP

This comprehensive example illustrates the PMP approach by:

- Encapsulating internal states and process functions in well-documented, self-contained modules.

- Demonstrating dynamic adaptation via runtime state updates.
- Implementing dependency injection through modular composition.
- Delivering polished, graphical feedback of token metrics.