

```
"""
```

```
🌺 APRIL FOOLS CORPORATE EMOJI TRANSLATOR 🌺  
Because every standup deserves a little chaos.
```

```
Usage:
```

```
    translator = CorporateEmojiTranslator()  
    print(translator.encode("We need to deploy to  
production before the deadline."))  
    print(translator.decode("We need to 🙌🔥🚀 to  
😓👉📅 before the 🔥🕒🕒."))
```

```
Tip: Paste encoded output into Slack/Teams and  
watch your coworkers slowly lose their minds.
```

```
"""
```

```
import re  
import random
```

```
LEXICON = {  
    # Corporate buzzwords  
    "meeting": ["📅🕒🕒", "👉👉👉", "📞🕒"],  
    "synergy": ["👉👉👉", "🌈👉👉", "👉👉👉"],  
    "agile": ["👉👉👉", "👉👉👉", "👉👉👉"],  
    "scrum": ["👉👉👉", "👉👉👉"],  
    "stakeholder": ["👉👉👉", "👉👉👉", "👉👉👉"],  
    "bandwidth": ["👉👉👉", "👉👉👉"],  
    "deliverable": ["👉👉👉", "👉👉👉"],  
    "roadmap": ["👉👉👉", "👉👉👉"],  
    "pivot": ["👉👉👉", "👉👉👉"],  
    "disruptive": ["👉👉👉", "👉👉👉"],  
    "innovative": ["👉👉👉", "👉👉👉"],
```

```

"leverage": ["👤📊😄", "💪📈😄"],
"ecosystem": ["🌱💰💡", "🔗🏢🔗"],
"paradigm": ["🧠🎯?", "📐😐😐"],
"holistic": ["🌈👤💼", "👉📊🎯"],
"scalable": ["📈👉🙌", "📈👉😄],
"streamline": ["🏊➡️😐", "✂️📋🚶"],
"proactive": ["🚶🌐✅", "⚡📋😐],
"circle back": ["🔄📞😐", "🔄☕😐],
"touch base": ["👉📞😐", "👉😐📅17],
"deep dive": ["🔍🕒😐", "🏊📊💀],
"low hanging fruit":["🍏😐✅", "👉👤📋],

```

Engineering terms

```

"deploy": ["🙌👤💥", "🚀😐🔥", "🏠💰💰"],
🔥],
"production": ["😐🔥❄️", "💀🌐⚠️", "🏭👤],
🔥],
"deadline": ["🔥👤🕒", "💀📅17⚡", "🕒😐],
🔥],
"engineer": ["👤☕🔧", "👤📝📺", "👤🔧],
☕],
"bug": ["🐛😐❌", "👤💀🔍", "🐛👤],
🔥],
"feature": ["👉📋😐", "🌟💬👤", "📺📺],
😐],
"refactor": ["👉😐📝", "🔧😐🔧", "✂️📋],
😐],
"legacy code": ["🏠👤⚠️", "😐💻😐", "🐛🔧],
💀],
"technical debt": ["🇩🇪😐📈", "🏠😐🕒", "👉📺],
🌌],
"sprint": ["🚶🌐📋", "⚡📅17😐],
"ticket": ["📺😐📋", "🐛📋🔄],
"review": ["👀😐👉", "🔍😐📋],
"merge": ["👉💥🙌", "🔍😐👉],
"rollback": ["🔍👤🔥", "🔍😐💀],
"outage": ["💀🌐📺", "🔥❄️👤", "⚠️👤],
📞],
"monitoring": ["👀😐📊", "🚶😐📈],
"pipeline": ["🔧🏭💰", "⚙️➡️🙌],

```

```

"kubernetes": ["🐳🔧", "🧠🔧"],
"microservice": ["🔧🔧", "🔧🔧🔧"],
"serverless": ["🐳🔧", "📧🔧"],

# AI company shoutouts
"anthropic": ["🧠🔧", "🧠🔧", "🧠🔧"],
"openai": ["🔧🔧", "🔧🔧", "🔧🔧"],
"google": ["🔍🔧", "🌐🔧"],
"microsoft": ["🔍🔧", "🔍🔧"],
"amazon": ["📦🔧", "📦🔧"],
"claude": ["🧠🔧", "🧠🔧"],
"gpt": ["🔍🔧", "🔍🔧"],
"ai": ["🔍🔧", "🔍🔧"],

# Meeting room classics
"let's take this offline": ["📺🔧", "🔧🔧"],
"per my last email": ["📧🔧", "👉📧"],
"going forward": ["📅🔧", "🔧🔧"],
"as per": ["📅🔧"],
"action items": ["✅🔧", "📝🔧"],
"bandwidth": ["📶🔧", "🔧🔧"],

}

# Build reverse map for decoding
REVERSE_MAP = {}
for word, variants in LEXICON.items():
    for v in variants:
        REVERSE_MAP[v] = word

class CorporateEmojiTranslator:
    """
    Translates corporate speak into pure emoji

```

```

chaos and back again.
    Perfect for April Fools, Slack bombs, and
    passive-aggressive standups.
    """

    def __init__(self, deterministic: bool = False,
seed: int = 42):
        """
        Args:
            deterministic: If True, always picks
the first variant (good for tests).
            seed: Random seed for reproducible
chaos. Because chaos needs reproducibility.
        """
        self.deterministic = deterministic
        if deterministic:
            random.seed(seed)

    def _pick_variant(self, variants: list) -> str:
        if self.deterministic:
            return variants[0]
        return random.choice(variants)

    def encode(self, text: str) -> str:
        """Turn boring corporate text into glorious
emoji soup."""
        result = text

        # Sort by length descending so multi-word
phrases match before single words
        sorted_keys = sorted(LEXICON.keys(),
key=len, reverse=True)

        for phrase in sorted_keys:
            variants = LEXICON[phrase]
            replacement =
self._pick_variant(variants)
            # Case-insensitive replacement,
preserve surrounding punctuation
            pattern = re.compile(re.escape(phrase),

```

```

re.IGNORECASE)
        result = pattern.sub(replacement,
result)

        return result

    def decode(self, text: str) -> str:
        """Translate emoji soup back into soul-
crushing corporate speak."""
        result = text
        # Sort by length descending so longer emoji
sequences match first
        sorted_keys = sorted(REVERSE_MAP.keys(),
key=len, reverse=True)
        for emoji_seq in sorted_keys:
            original = REVERSE_MAP[emoji_seq]
            result = result.replace(emoji_seq,
original)
        return result

    def prank_slack_message(self, original: str) ->
str:
        """
        Format a full prank-ready Slack message.
        Includes the encoded text plus a helpful
'explanation'.
        """
        encoded = self.encode(original)
        return (
            f"{encoded}\n\n"
            f"_(Sent from my new AI-powered
communication 🧠⚡? assistant. "
            f"Please 🙌📧😓 if you need the
original.)_"
        )

    def list_vocabulary(self) -> None:
        """Print the full chaos dictionary. For
orientation purposes."""
        print("📖 CORPORATE EMOJI DICTIONARY\n" +

```

```

"*40)
    for word, variants in
sorted(LEXICON.items()):
    print(f" {word:<25} → {' /
'.join(variants)}")

# — Demo


---




---



if __name__ == "__main__":
    t = CorporateEmojiTranslator()

    examples = [
        "We need to deploy to production before the
deadline.",
        "Let's circle back after the meeting to
touch base on the roadmap.",
        "The engineer found a bug in the legacy
code during the sprint.",
        "Our agile process needs more synergy and
stakeholder alignment.",
        "Going forward, let's leverage our
ecosystem for scalable deliverables.",
        "The AI model caused an outage in
production. Per my last email, monitoring is key.",
        "Anthropic and OpenAI are both disruptive
and innovative.",
    ]

    print("🥰🥰 APRIL FOOLS CORPORATE EMOJI
TRANSLATOR 🥰🥰\n")
    print("— ENCODED
_____")

    for msg in examples:
        encoded = t.encode(msg)
        print(f" IN: {msg}")
        print(f" OUT: {encoded}\n")

```

```

print("\n— DECODED ROUND-TRIP
——")

test = "We need to 🙋🏻🔧💥 to 😊👉🎉 before the
👉🏻🕒."
print(f"  IN:  {test}")
print(f"  OUT: {t.decode(test)}")

print("\n— PRANK-READY SLACK MESSAGE
——")

print(t.prank_slack_message(
    "Action items from today's meeting:
refactor the legacy code "
    "and deploy the new feature to production
by end of sprint."
))

print("\n— FULL VOCABULARY
——")

t.list_vocabulary()

```