```
import spacy
from nltk import Tree
import nltk
from spacy import displacy
import sys
# Load the English language model from spaCy
nlp = spacy.load("en_core_web_sm")

def install_nltk():
    try:
        import nltk
    except ImportError:
        print("Installing NLTK...")
        import subprocess
        subprocess.check_call([sys.executable, "-m", "pip", "install",
"nltk"])
        import nltk
    nltk.download('punkt')

def build_constituency_tree(doc):
    # A helper function to recursively build a tree structure
    def build_tree(token):
        if not list(token.children):
            return Tree(f"{token.text}/{token.pos_}", [])
        return Tree(f"{token.text}/{token.pos_}", [build_tree(child) for child
in token.children])
    # Find the root token and build the tree from it
    root = [token for token in doc if token.head == token][0]
    tree = build_tree(root)
    return tree

def identify_parts_of_speech(text):
    # Process the text using spaCy
    doc = nlp(text)

    # Tokens and Lexemes Output

    print("\nTokens and Lexemes:")
    print("-" * 50)
    print(f"{'Token':<15}{'Lexeme':<15}{'Is Alpha':<10}{'Is Stop Word':<15}")
    print("-" * 50)
    for token in doc:
        print(f"{token.text:<15}{token.lemma_:<15}{str(token.is_alpha):<10}{st
r(token.is_stop):<15}")

    # POS Tagging Output
    print("\nParts of Speech (POS) Tagging:")
    print("-" * 50)
```

```
    print(f"{'Word':<15}{'POS':<15}{'Explanation':<30}")
    print("-" * 50)
    for token in doc:
        print(f"{token.text:<15}{token.pos_:<15}{spacy.explain(token.pos_):<30
}")

    # Dependency Parsing Output
    print("\nDependency Parsing (Syntactic Role):")
    print("-" * 50)
    print(f"{'Word':<15}{'Dependency':<20}{'Head Word':<15}")
    print("-" * 50)
    for token in doc:
        print(f"{token.text:<15}{token.dep_:<20}{token.head.text:<15}")

    # Build and display the constituency tree
    constituency_tree = build_constituency_tree(doc)
    print("\nConstituency Parse Tree:")
    constituency_tree.pretty_print()

    # Identifiers (Named Entities)
    print("\nNamed Entities:")
    print("-" * 50)
    print(f"{'Entity':<15}{'Label':<15}{'Explanation':<30}")
    print("-" * 50)
    for ent in doc.ents:
        print(f"{ent.text:<15}{ent.label_:<15}{spacy.explain(ent.label_):<30}"
)

if __name__ == "__main__":
    # Install nltk if it's not already installed
    install_nltk()

    while True:


        input_text = input("Enter an English word, phrase, or sentence (or
type 'quit' to exit): ")
        if input_text.lower() == 'quit':
            print("Exiting the program. Goodbye!")
            break
        identify_parts_of_speech(input_text)

# Load the English language model from spaCy
nlp = spacy.load("en_core_web_sm")

def install_nltk():
    try:
        import nltk
```

```
    except ImportError:
        print("Installing NLTK...")
        import subprocess
        subprocess.check_call([sys.executable, "-m", "pip", "install",
"nltk"])
        import nltk
    nltk.download('punkt')

def build_constituency_tree(doc):
    # A helper function to recursively build a tree structure
    def build_tree(token):
        if not list(token.children):
            return Tree(f"{token.text}/{token.pos_}", [])
        return Tree(f"{token.text}/{token.pos_}", [build_tree(child) for child
in token.children])
    # Find the root token and build the tree from it
    root = [token for token in doc if token.head == token][0]
    tree = build_tree(root)
    return tree

def identify_parts_of_speech(text):
    # Process the text using spaCy
    doc = nlp(text)
    # Tokens and Lexemes Output
    print("\nTokens and Lexemes:")
    print("-" * 50)
    print(f"{'Token':<15}{'Lexeme':<15}{'Is Alpha':<10}{'Is Stop Word':<15}")
    print("-" * 50)
    for token in doc:
        print(f"{token.text:<15}{token.lemma_:<15}{str(token.is_alpha):<10}{st
r(token.is_stop):<15}")

    # POS Tagging Output
    print("\nParts of Speech (POS) Tagging:")
    print("-" * 50)
    print(f"{'Word':<15}{'POS':<15}{'Explanation':<30}")
    print("-" * 50)
    for token in doc:
        print(f"{token.text:<15}{token.pos_:<15}{spacy.explain(token.pos_):<30
}")

    # Dependency Parsing Output
    print("\nDependency Parsing (Syntactic Role):")
    print("-" * 50)
    print(f"{'Word':<15}{'Dependency':<20}{'Head Word':<15}")
    print("-" * 50)
    for token in doc:
        print(f"{token.text:<15}{token.dep_:<20}{token.head.text:<15}")
```

```
    # Build and display the constituency tree
    constituency_tree = build_constituency_tree(doc)
    print("\nConstituency Parse Tree:")
    constituency_tree.pretty_print()

    # Identifiers (Named Entities)
    print("\nNamed Entities:")
    print("-" * 50)
    print(f"{'Entity':<15}{'Label':<15}{'Explanation':<30}")
    print("-" * 50)
    for ent in doc.ents:
        print(f"{ent.text:<15}{ent.label_:<15}{spacy.explain(ent.label_):<30}"
)

if __name__ == "__main__":
    # Install nltk if it's not already installed
    install_nltk()

    while True:
        # Take user input
        input_text = input("Enter an English word, phrase, or sentence (or
type 'quit' to exit): ")
        if input_text.lower() == 'quit':
            print("Exiting the program. Goodbye!")
            break
        identify_parts_of_speech(input_text)

# Load the English language model from spaCy

nlp = spacy.load("en_core_web_sm")

def install_nltk():
    try:
        import nltk
    except ImportError:
        print("Installing NLTK...")
        import subprocess
        subprocess.check_call([sys.executable, "-m", "pip", "install",
"nltk"])
        import nltk
    nltk.download('punkt')

def build_constituency_tree(doc):
    # A helper function to recursively build a tree structure
    def build_tree(token):
        if not list(token.children):
            return Tree(f"{token.text}/{token.pos_}", [])
```

```
        return Tree(f"{token.text}/{token.pos_}", [build_tree(child) for child
in token.children])

    # Find the root token and build the tree from it
    root = [token for token in doc if token.head == token][0]
    tree = build_tree(root)
    return tree

def identify_parts_of_speech(text):
    # Process the text using spaCy
    doc = nlp(text)

    # Tokens and Lexemes Output
    print("\nTokens and Lexemes:")
    print("-" * 50)
    print(f"{'Token':<15}{'Lexeme':<15}{'Is Alpha':<10}{'Is Stop Word':<15}")
    print("-" * 50)
    for token in doc:
        print(f"{token.text:<15}{token.lemma_:<15}{str(token.is_alpha):<10}{st
r(token.is_stop):<15}")
    # POS Tagging Output
    print("\nParts of Speech (POS) Tagging:")
    print("-" * 50)
    print(f"{'Word':<15}{'POS':<15}{'Explanation':<30}")
    print("-" * 50)

    for token in doc:
        print(f"{token.text:<15}{token.pos_:<15}{spacy.explain(token.pos_):<30
}")

    # Dependency Parsing Output
    print("\nDependency Parsing (Syntactic Role):")
    print("-" * 50)
    print(f"{'Word':<15}{'Dependency':<20}{'Head Word':<15}")
    print("-" * 50)
    for token in doc:
        print(f"{token.text:<15}{token.dep_:<20}{token.head.text:<15}")

    # Build and display the constituency tree
    constituency_tree = build_constituency_tree(doc)
    print("\nConstituency Parse Tree:")
    constituency_tree.pretty_print()

    # Identifiers (Named Entities)
    print("\nNamed Entities:")
    print("-" * 50)
    print(f"{'Entity':<15}{'Label':<15}{'Explanation':<30}")
    print("-" * 50)
```

```
    for ent in doc.ents:
        print(f"{ent.text:<15}{ent.label_:<15}{spacy.explain(ent.label_):<30}"
)

if __name__ == "__main__":
    # Install nltk if it's not already installed
    install_nltk()

    while True:
        # Take user input
        input_text = input("Enter an English word, phrase, or sentence (or
type 'quit' to exit): ")
        if input_text.lower() == 'quit':
            print("Exiting the program. Goodbye!")
            break
        identify_parts_of_speech(input_text)
```