Experiment No.1

Code:

```
class MorphologyAnalyzer:
    def __init__(self):
        # Predefined morphemes (for simplicity)
        self.free_morphemes = {"run", "jump", "play", "happy", "kind",
"work"}
        self.prefixes = {"un", "re", "dis", "pre", "mis"}
        self.suffixes = {"ing", "ed", "er", "est", "ly", "able", "ness"}
        self.compound_words = {"sunflower", "notebook", "basketball",
"toothbrush"}


    def analyze_word(self, word):
        morphemes = []


        # Check for compounding
        for compound in self.compound_words:
            if word == compound:
                parts = self.split_compound_word(word)
                return {"type": "Compound Word", "components": parts}


        # Check for prefixes
        for prefix in self.prefixes:
            if word.startswith(prefix):
                root = word[len(prefix):]
                if root in self.free_morphemes:
                    morphemes.append(("Prefix", prefix))
                    morphemes.append(("Root", root))
                    return {"type": "Affixation", "components": morphemes}


        # Check for suffixes
```

```
        for suffix in self.suffixes:
            if word.endswith(suffix):
                root = word[:-len(suffix)]
                if root in self.free_morphemes:
                    morphemes.append(("Root", root))
                    morphemes.append(("Suffix", suffix))
                    return {"type": "Affixation", "components": morphemes}

        # Check if word is a free morpheme
        if word in self.free_morphemes:
            return {"type": "Free Morpheme", "components": [word]}

        return {"type": "Unknown", "components": [word]}

    def split_compound_word(self, word):
        for free_morpheme in self.free_morphemes:
            if word.startswith(free_morpheme):
                second_part = word[len(free_morpheme):]
                return [free_morpheme, second_part]
        return [word]

    def explain_analysis(self, analysis):
        print("Word Type:", analysis["type"])
        print("Components:")
        for component in analysis["components"]:
            if isinstance(component, tuple):
                print(f"  - {component[0]}: {component[1]}")
            else:
                print(f"  - {component}")

# Example usage:
analyzer = MorphologyAnalyzer()
```

```
words = ["unhappy", "happiness", "running", "notebook", "work"]

for word in words:
    analysis = analyzer.analyze_word(word)
    analyzer.explain_analysis(analysis)
    print("-")
```

Output:

```
Word Type: Affixation
Components:
  - Prefix: un
  - Root: happy
-
Word Type: Unknown
Components:
  - happiness
-
Word Type: Unknown
Components:
  - running
-
Word Type: Compound Word
Components:
  - notebook
-
Word Type: Free Morpheme
Components:
  - work
```

Experiment No.2

Code:

```
class MorphologyTable:
    def __init__(self):
        self.table = {}

    def add_entry(self, word, morpheme_type):
        if word not in self.table:
            self.table[word] = morpheme_type
            print(f"Added: {word} ({morpheme_type})")
        else:
            print(f"{word} already exists in the table.")

    def delete_entry(self, word):
        if word in self.table:
            del self.table[word]
            print(f"Deleted: {word}")
        else:
            print(f"{word} not found in the table.")

    def display_table(self):
        if not self.table:
            print("Table is empty.")
        else:
            print("Morphology Table:")
            for word, morpheme_type in self.table.items():
                print(f"  {word}: {morpheme_type}")

# Example usage:
print("-----------------------------------")
morph_table = MorphologyTable()
```

```
morph_table.display_table()
print("-----------------------------------")
morph_table.add_entry("running", "Verb (Present Participle)")
morph_table.add_entry("unhappy", "Adjective (Negative Prefix)")
morph_table.display_table()
print("-----------------------------------")
morph_table.delete_entry("running")
morph_table.display_table()
print("-----------------------------------")
```

Output:

```
-----------------------------------
Table is empty.
-----------------------------------
Added: running (Verb (Present Participle))
Added: unhappy (Adjective (Negative Prefix))
Morphology Table:
  running: Verb (Present Participle)
  unhappy: Adjective (Negative Prefix)
-----------------------------------
Deleted: running
Morphology Table:
  unhappy: Adjective (Negative Prefix)
-----------------------------------
```

Experiment No.3

Code:

```
import nltk
from nltk import pos_tag
from nltk.tokenize import word_tokenize

class POSTagger:
    def __init__(self):
        nltk.download('averaged_perceptron_tagger')
        nltk.download('punkt')

    def tag_sentence(self, sentence):
        words = word_tokenize(sentence)
        tagged_words = pos_tag(words)
        return tagged_words

    def display_tags(self, tagged_words):
        print("POS Tagging Result:")
        for word, tag in tagged_words:
            print(f"  {word}: {tag}")

# Example usage:
pos_tagger = POSTagger()
sentence = "The quick brown fox jumps over the lazy dog."
tagged_words = pos_tagger.tag_sentence(sentence)
pos_tagger.display_tags(tagged_words)
```

Output:

```
POS Tagging Result:
  The: DT
  quick: JJ
  brown: NN
  fox: NN
  jumps: VBZ
  over: IN
  the: DT
  lazy: JJ
  dog: NN
```

Experiment No.4

Code:

```
import nltk
from nltk import pos_tag, word_tokenize
from nltk.corpus import wordnet

class SemanticAnalyzer:
    def __init__(self):
        nltk.download('averaged_perceptron_tagger')
        nltk.download('punkt')
        nltk.download('wordnet')

    def get_synonyms(self, word):
        synonyms = set()
        for syn in wordnet.synsets(word):
            for lemma in syn.lemmas():
                synonyms.add(lemma.name())
        return list(synonyms)

    def get_antonyms(self, word):
        antonyms = set()
        for syn in wordnet.synsets(word):
            for lemma in syn.lemmas():
                if lemma.antonyms():
                    antonyms.add(lemma.antonyms()[0].name())
        return list(antonyms)

    def analyze_sentence(self, sentence):
        words = word_tokenize(sentence)
        tagged_words = pos_tag(words)
        result = {}
```

```
        for word, tag in tagged_words:
            synonyms = self.get_synonyms(word)
            antonyms = self.get_antonyms(word)
            result[word] = {"POS": tag, "Synonyms": synonyms, "Antonyms":
antonyms}

        return result

    def display_analysis(self, analysis):
        print("Semantic Analysis Result:")
        for word, details in analysis.items():
            print(f"  {word} ({details['POS']}):")
            print(f"    Synonyms: {', '.join(details['Synonyms']) if
details['Synonyms'] else 'None'}")
            print(f"    Antonyms: {', '.join(details['Antonyms']) if
details['Antonyms'] else 'None'}")

# Example usage:
semantic_analyzer = SemanticAnalyzer()
sentence = "The quick brown fox jumps over the lazy dog."
analysis = semantic_analyzer.analyze_sentence(sentence)
semantic_analyzer.display_analysis(analysis)
```

Output:

```
Semantic Analysis Result:
  The (DT):
    Synonyms: None
    Antonyms: None
  quick (JJ):
    Synonyms: warm, spry, flying, prompt, straightaway, quick, fast,
promptly, quickly, immediate, agile, speedy, nimble, ready
```

Antonyms: None

brown (NN):

Synonyms: brown, Robert_Brown, Brown_University, browned, chocolate-brown, brownish, dark-brown, Brown, embrown, brownness, John_Brown

Antonyms: None

fox (NN):

Synonyms: confound, fox, play_a_joke_on, Fox, trick, pull_a_fast_one_on, befuddle, dodger, slyboots, George_Fox, fuddle, confuse, Charles_James_Fox, play_tricks, fob, play_a_trick_on, flim-flam, bedevil, throw, discombobulate

Antonyms: None

jumps (VBZ):

Synonyms: jump_out, climb_up, start, pass_over, jumpstart, bound, stick_out, jump_off, skip, jumping, leap_out, skip_over, jump-start, chute, stand_out, jump, startle, parachuting, leap, rise, spring, saltation, parachute, alternate, derail

Antonyms: None

over (IN):

Synonyms: ended, o'er, terminated, complete, over, all_over, concluded

Antonyms: None

the (DT):

Synonyms: None

Antonyms: None

lazy (JJ):

Synonyms: lazy, faineant, indolent, slothful, work-shy, otiose

Antonyms: None

dog (NN):

Synonyms: andiron, cad, blackguard, frank, go_after, chase_after, firedog, chase, click, domestic dog, wienerwurst, give_chase, hot_dog, detent, bounder, tail, frankfurter, trail, tag, frump, hound, weenie, dog-iron, heel, dog, hotdog, pawl, Canis_familiaris, wiener, track

Antonyms: None

. (.):

Synonyms: None

Antonyms: None