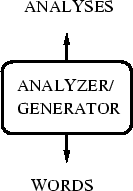
**1. Word Analysis**

[**Aim**](https://nlp-iiith.vlabs.ac.in/exp/word-analysis/index.html)

A word can be simple or complex. For example, the word 'cat' is simple because one cannot further decompose the word into smaller part. On the other hand, the word 'cats' is complex, because the word is made up of two parts: root 'cat' and plural suffix '-s'



[**Theory**](https://nlp-iiith.vlabs.ac.in/exp/word-analysis/theory.html)

Analysis of a word into root and affix(es) is called as Morphological analysis of a word. It is mandatory to identify root of a word for any natural language processing task. A root word can have various forms. For example, the word 'play' in English has the following forms: 'play', 'plays', 'played' and 'playing'. Hindi shows more number of forms for the word 'खेल' (khela) which is equivalent to 'play'. The forms of 'खेल'(khela) are the following:

खेल(khela), खेला(khelaa), खेली(khelii), खेलूंगा(kheluungaa), खेलूंगी(kheluungii), खेलेगा(khelegaa), खेलेगी(khelegii), खेलते(khelate), खेलती(khelatii), खेलने(khelane), खेलकर(khelakar)

For Telugu root ఆడడం (Adadam), the forms are the following::

Adutaanu, AdutunnAnu, Adenu, Ademu, AdevA, AdutAru, Adutunnaru, AdadAniki, Adesariki, AdanA, Adinxi, Adutunxi, AdinxA, AdeserA, Adestunnaru, ...

Thus we understand that the morphological richness of one language might vary from one language to another. Indian languages are generally morphologically rich languages and therefore morphological analysis of words becomes a very significant task for Indian languages.

**Types of Morphology**

Morphology is of two types,

1. Inflectional morphology

Deals with word forms of a root, where there is no change in lexical category. For example, 'played' is an inflection of the root word 'play'. Here, both 'played' and 'play' are verbs.

1. Derivational morphology

Deals with word forms of a root, where there is a change in the lexical category. For example, the word form 'happiness' is a derivation of the word 'happy'. Here, 'happiness' is a derived noun form of the adjective 'happy'.

**Morphological Features**:

All words will have their lexical category attested during morphological analysis. A noun and pronoun can take suffixes of the following features: gender, number, person, case For example, morphological analysis of a few words is given below:

| **Language** | **input:word** | **output:analysis** |
| --- | --- | --- |
| Hindi | लडके (ladake) | rt=लड़का(ladakaa), cat=n, gen=m, num=sg, case=obl |
| Hindi | लडके (ladake) | rt=लड़का(ladakaa), cat=n, gen=m, num=pl, case=dir |
| Hindi | लड़कों (ladakoM) | rt=लड़का(ladakaa), cat=n, gen=m, num=pl, case=obl |
| English | Boy | rt=boy, cat=n, gen=m, num=sg |
| English | Boys | rt=boy, cat=n, gen=m, num=pl |

A verb can take suffixes of the following features: tense, aspect, modality, gender, number, person

| **Language** | **input:word** | **output:analysis** |
| --- | --- | --- |
| Hindi | हँसी(hansii) | rt=हँस(hans), cat=v, gen=fem, num=sg/pl, per=1/2/3 tense=past, aspect=pft |
| English | Toys | rt=toy, cat=n, num=pl, per=3 |

'rt' stands for root. 'cat' stands for lexical category. Thev value of lexicat category can be noun, verb, adjective, pronoun, adverb, preposition. 'gen' stands for gender. The value of gender can be masculine or feminine. 'num' stands for number. The value of number can be singular (sg) or plural (pl). 'per' stands for person. The value of person can be 1, 2 or 3

The value of tense can be present, past or future. This feature is applicable for verbs. The value of aspect can be perfect (pft), continuous (cont) or habitual (hab). This feature is not applicable for verbs.

'case' can be direct or oblique. This feature is applicable for nouns. A case is an oblique case when a postposition occurs after noun. If no postposition can occur after noun, then the case is a direct case. This is applicable for hindi but not english as it doesn't have any postpositions. Some of the postpsitions in hindi are: का(kaa), की(kii), के(ke), को(ko), में(meM)

[**Objective**](https://nlp-iiith.vlabs.ac.in/exp/word-analysis/objective.html)

The objective of the experiment is to learn about morphological features of a word by analysing it.

[**Procedure**](https://nlp-iiith.vlabs.ac.in/exp/word-analysis/procedure.html)

**STEP 1**: Select the language.

**OUTPUT**: Drop down for selecting words will appear.

**STEP 2**: Select the word.

**OUTPUT**: Drop down for selecting features will appear.

**STEP 3**: Select the features.

**STEP 4**: Click "Check" button to check your answer.

**OUTPUT**: Right features are marked by tick and wrong features are marked by cross.

[**Assignment**](https://nlp-iiith.vlabs.ac.in/exp/word-analysis/assignment.html)

1. Choose a typical masculine noun, ending in 'A', from your language. Write down its various forms along with various features and their values associated with them.

Source Code:

import nltk

spanish\_lemmatizer = nltk.stem.WordNetLemmatizer()

# Singular forms

dia = "día"

dia\_lemma = spanish\_lemmatizer.lemmatize(dia, "n")

dia\_plural = "días"

dia\_plural\_lemma = spanish\_lemmatizer.lemmatize(dia\_plural, "n")

# Gender

gender = "masculine"

# Number

number = "singular"

number\_plural = "plural"

# Part of speech

pos = "noun"

# Syntactic features

features = {

"Gender": gender,

"Number": number

}

features\_plural = {

"Gender": gender,

"Number": number\_plural

}

# Print results

print(f"Singular form: {dia}, lemma: {dia\_lemma}, features: {features}")

print(f"Plural form: {dia\_plural}, lemma: {dia\_plural\_lemma}, features: {features\_plural}")

**2.** English has a suffix -en whose use is illustrated in the following lists:

| **List A** | **List B** |
| --- | --- |
| Red | Redden |
| Mad | Madden |
| Soft | Soften |
| Wide | Widen |
| Sharp | Sharpen |

In regard to these data, answer the following questions:

A. What part of speech does the suffix -en attach to? That is, what is the part of speech of the words in list A?

Source Code:

import nltk

list\_a = ['Red', 'Mad', 'Soft', 'Wide', 'Sharp']

tagged\_list\_a = nltk.pos\_tag(list\_a)

print(tagged\_list\_a)

output:

(Here, 'JJ' stands for adjective, confirming that the words in List A are indeed adjectives.)

B. When the suffix -en is attached to a word, what part of speech is the resulting word? Give some specific morphological properties of one of the words in list B, in order to justify your answer.

Source Code:

import nltk

from nltk.stem import WordNetLemmatizer

wnl = WordNetLemmatizer()

word = 'madden'

lemma = wnl.lemmatize(word, 'v')

print(f"Word: {word}")

print(f"Lemma: {lemma}")

print(f"Suffix: {word[len(lemma):]}")

**3.** Take one verb from your mother tongue, gloss it (i.e., give the Engish meaning) and conjugate it in all tenses and aspects and persons.

Source Code:

import nltk

from nltk.stem import WordNetLemmatizer

# Initialize the WordNetLemmatizer

wnl = WordNetLemmatizer()

# Define the verb

verb = "run"

# Define the tenses and aspects

tenses\_aspects = ["infinitive", "present", "past", "present\_continuous", "past\_continuous", "present\_perfect", "past\_perfect", "present\_perfect\_continuous", "past\_perfect\_continuous"]

# Define the persons

persons = ["first\_singular", "second\_singular", "third\_singular", "first\_plural", "second\_plural", "third\_plural"]

# Loop through all combinations and print the result

for tense\_aspect in tenses\_aspects:

for person in persons:

if tense\_aspect == "infinitive":

if person == "third\_singular":

result = wnl.lemmatize(verb, "v")

else:

result = verb

elif tense\_aspect == "present":

if person == "third\_singular":

result = verb + "s"

else:

result = verb

elif tense\_aspect == "past":

if verb[-1] == "e":

result = verb + "d"

elif verb[-2:] == "y":

result = verb[:-1] + "ied"

else:

result = verb + "ed"

elif tense\_aspect == "present\_continuous":

result = "am/is/are " + verb + "ing"

elif tense\_aspect == "past\_continuous":

result = "was/were " + verb + "ing"

elif tense\_aspect == "present\_perfect":

result = "have/has " + verb + "ed"

elif tense\_aspect == "past\_perfect":

result = "had " + verb + "ed"

elif tense\_aspect == "present\_perfect\_continuous":

result = "have/has been " + verb + "ing"

elif tense\_aspect == "past\_perfect\_continuous":

result = "had been " + verb + "ing"

# Add the person to the result

if person == "first\_singular":

result += " (I)"

elif person == "second\_singular":

result += " (you)"

elif person == "third\_singular":

result += " (he/she/it)"

elif person == "first\_plural":

result += " (we)"

elif person == "second\_plural":

result += " (you all)"

elif person == "third\_plural":

result += " (they)"

# Print the result

print(tense\_aspect.capitalize() + " " + person.capitalize() + ": " + result)

**4.** Refer to the following data and answer the question below:

List 1: taller, shorter, higher, lower, smarter <br>

List 2: mower, teacher, sailor, caller, operator <br>

List 3: never, cover, finger, river <br>

Are the words ending with 'er'/'or' have some common features?

Source Code:

import nltk

# Define the three lists of words

list1 = ['taller', 'shorter', 'higher', 'lower', 'smarter']

list2 = ['mower', 'teacher', 'sailor', 'caller', 'operator']

list3 = ['never', 'cover', 'finger', 'river']

# Use the NLTK POS tagger to get the POS tags of the words

pos\_tags\_list1 = nltk.pos\_tag(list1)

pos\_tags\_list2 = nltk.pos\_tag(list2)

pos\_tags\_list3 = nltk.pos\_tag(list3)

# Filter out the words that end with 'er' or 'or'

er\_words = [word for word, pos in pos\_tags\_list1 if word[-2:] == 'er']

or\_words = [word for word, pos in pos\_tags\_list2 if word[-2:] == 'or']

# Check if the filtered words have a common POS tag

er\_pos\_tags = set(pos for word, pos in pos\_tags\_list1 if word[-2:] == 'er')

or\_pos\_tags = set(pos for word, pos in pos\_tags\_list2 if word[-2:] == 'or')

if len(er\_pos\_tags) == 1 and len(or\_pos\_tags) == 1 and er\_pos\_tags == or\_pos\_tags:

print("The words ending with 'er'/'or' have a common feature of indicating the agent or doer of an action or the one who performs a specific role or function.")

else:

print("The words ending with 'er'/'or' do not have a common feature.")

5. Identify root and suffix in the following words:

kissed

stronger

goodness

teacher

achievement

Source Code:

import nltk

from nltk.stem import SnowballStemmer

# Create a Snowball stemmer for English

stemmer = SnowballStemmer('english')

# Define the list of words

words = ['kissed', 'stronger', 'goodness', 'teacher', 'achievement']

# Iterate over the words and identify their root and suffix

for word in words:

root = stemmer.stem(word)

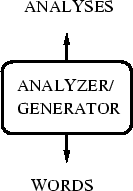
suffix = word[len(root):]

print(f"{word}: root='{root}', suffix='{suffix}'")

**2. Word Generation**

[**Aim**](https://nlp-iiith.vlabs.ac.in/exp/word-generation/index.html)

A word can be simple or complex. For example, the word 'cat' is simple because one cannot further decompose the word into smaller part. On the other hand, the word 'cats' is complex, because the word is made up of two parts: root 'cat' and plural suffix '-s'



[**Theory**](https://nlp-iiith.vlabs.ac.in/exp/word-generation/theory.html)

Given the root and suffix information, a word can be generated. For example,

| **Language** | **input:analysis** | **output:word** |
| --- | --- | --- |
| Hindi | rt=लड़का(ladakaa), cat=n, gen=m, num=sg, case=obl | लड़के(ladake) |
| Hindi | rt=लड़का(ladakaa), cat=n, gen=m, num=pl, case=dir | लड़के(ladake) |
| English | rt=boy, cat=n, num=pl | boys |
| English | rt=play, cat=v, num=sg, per=3, tense=pr | plays |

* Morphological analysis and generation: Inverse processes.
* Analysis may involve non-determinism, since more than one analysis is possible.
* Generation is a deterministic process. In case a language allows spelling variation, then till that extent, generation would also involve non-determinism.

[**Objective**](https://nlp-iiith.vlabs.ac.in/exp/word-generation/objective.html)

The objective of the experiment is to generate word forms from root and suffix information.

[**Procedure**](https://nlp-iiith.vlabs.ac.in/exp/word-generation/procedure.html)

STEP 1: Select the language.

OUTPUT: Drop downs for selecting root and other features will appear.

STEP 2: Select the root and other features.

STEP 3: After selecting all the features, select the word corresponding above features selected.

STEP 4: Click the check button to see whether right word is selected or not

OUTPUT: Output tells whether the word selected is right or wrong

[**Assignment**](https://nlp-iiith.vlabs.ac.in/exp/word-generation/assignment.html)

1. Generate words for the following features:

English:  
root: boy category: noun number: singular  
root: child category: noun number: plural  
root: play category: verb gender: male number: singular person:first tense: simple-present  
root: play category: verb gender: male number: singular person: third tense: simple-present

Source Code:

For the word with root "boy", category "noun", and number "singular":

Source line:

import nltk

# Generate word

word = nltk.corpus.wordnet.morphy('boy', nltk.corpus.wordnet.NOUN)

print(word)

output:

For the word with root "child", category "noun", and number "plural":

Source line:

import nltk

# Generate word

word = nltk.corpus.wordnet.morphy('child', nltk.corpus.wordnet.NOUN)

word\_plural = nltk.PluralizePluralize.pluralize(word)

print(word\_plural)

output:

For the word with root "play", category "verb", gender "male", number "singular", person "first", and tense "simple-present":

Source line:

import nltk

# Generate word

word = nltk.corpus.wordnet.morphy('play', nltk.corpus.wordnet.VERB)

word\_inflected = nltk.conjugate.conjugate(word, tense='present', number='singular', person='1st', gender='male')

print(word\_inflected)

output:

For the word with root "play", category "verb", gender "male", number "singular", person "third", and tense "simple-present":

import nltk

# Generate word

word = nltk.corpus.wordnet.morphy('play', nltk.corpus.wordnet.VERB)

word\_inflected = nltk.conjugate.conjugate(word, tense='present', number='singular', person='3rd', gender='male')

print(word\_inflected)

output:

**3. Morphology**

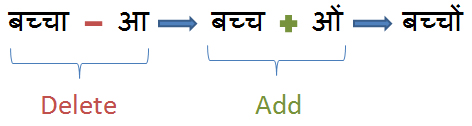
[**Aim**](https://nlp-iiith.vlabs.ac.in/exp/morphology/index.html)

Morphology is the study of the way words are built up from smaller meaning bearing units i.e., morphemes. A morpheme is the smallest meaningful linguistic unit. For eg:

* बच्चों(bachchoM) consists of two morphemes, बच्चा(bachchaa) has the information of the root word noun "बच्चा"(bachchaa) and ओं(oM) has the information of plural and oblique case.
* played has two morphemes play and -ed having information verb "play" and "past tense", so given word is past tense form of verb "play".



Words can be analysed morphologically if we know all variants of a given root word. We can use an 'Add-Delete' table for this analysis.



[**Theory**](https://nlp-iiith.vlabs.ac.in/exp/morphology/theory.html)

Morph Analyser

Definition

Morphemes are considered as smallest meaningful units of language. These morphemes can either be a root word(play) or affix(-ed). Combination of these morphemes is called morphological process. So, word "played" is made out of 2 morphemes "play" and "-ed". Thus finding all parts of a word(morphemes) and thus describing properties of a word is called "Morphological Analysis". For example, "played" has information verb "play" and "past tense", so given word is past tense form of verb "play".

Analysis of a word :

बच्चों (bachchoM) = बच्चा(bachchaa)(root) + ओं(oM)(suffix) (ओं=3 plural oblique) A linguistic paradigm is the complete set of variants of a given lexeme. These variants can be classified according to shared inflectional categories (eg: number, case etc) and arranged into tables.

Paradigm for बच्चा

| Case/num | Singular | Plural |
| --- | --- | --- |
| Direct | बच्चा(bachchaa) | बच्चे(bachche) |
| Oblique | बच्चे(bachche) | बच्चों (bachchoM) |

Algorithm to get बच्चों(bachchoM) from बच्चा(bachchaa)

1. Take Root बच्च(bachch)आ(aa)
2. Delete आ(aa)
3. output बच्च(bachch)
4. Add ओं(oM) to output
5. Return बच्चों (bachchoM)

Therefore आ is deleted and ओं is added to get बच्चों

Add-Delete table for बच्चा

| Delete | Add | Number | Case | Variants |
| --- | --- | --- | --- | --- |
| आ(aa) | आ(aa) | Sing | dr | बच्चा(bachchaa) |
| आ(aa) | ए(e) | Plu | dr | बच्चे(bachche) |
| आ(aa) | ए(e) | Sing | ob | बच्चे(bachche) |
| आ(aa) | ओं(oM) | Plu | ob | बच्चों(bachchoM) |

Paradigm Class

Words in the same paradigm class behave similarly, for Example लड़क is in the same paradigm class as बच्च, so लड़का would behave similarly as बच्चा as they share the same paradigm class.

[**Objective**](https://nlp-iiith.vlabs.ac.in/exp/morphology/objective.html)

The Objective of the experiment is understanding the morphology of a word by the use of Add-Delete table.

[**Procedure**](https://nlp-iiith.vlabs.ac.in/exp/morphology/procedure.html)

STEP 1: Select a word root.

STEP 2: Fill the add-delete table and submit.

STEP 3: If wrong, see the correct answer or repeat STEP1.

[**Assignment**](https://nlp-iiith.vlabs.ac.in/exp/morphology/assignment.html)

**Q1. Select words from this which belong to the same paradigm: मनुष्य(manuShya), पक्षी(pakshii), शिशु(shishu), गुरु(guruu), नर(nar)**

**Source Code:**

import nltk

# Define the list of words

words = ['मनुष्य', 'पक्षी', 'शिशु', 'गुरु', 'नर']

# Define the morphological patterns for the paradigm

paradigm\_pattern = r'(.\*?)$'

# Create an empty dictionary to store the paradigm groups

paradigm\_groups = {}

# Iterate through each word

for word in words:

# Extract the morphological pattern of the word

morphological\_pattern = nltk.regexp\_tokenize(word, paradigm\_pattern)[0]

# If the pattern is not in the paradigm groups dictionary, add it with the current word as the value

if morphological\_pattern not in paradigm\_groups:

paradigm\_groups[morphological\_pattern] = [word]

else:

# If the pattern is already in the paradigm groups dictionary, append the current word to the existing value

paradigm\_groups[morphological\_pattern].append(word)

# Print the paradigm groups

for pattern, group in paradigm\_groups.items():

print(f'Words in paradigm with pattern "{pattern}": {", ".join(group)}')

**output:**

**Q2. Construct the paradigm table for the above words.**

Source Code:

import nltk

from nltk import word\_tokenize

# Define the words

words = ["मनुष्य", "पक्षी", "शिशु", "गुरु", "नर"]

# Define the morphological categories

categories = {

"Person": ["1s", "2s", "3s", "1p", "2p", "3p"],

"Gender": ["Masc", "Fem", "Neut"],

"Number": ["Sg", "Pl"]

}

# Define the inflection rules for each word

inflections = {

"मनुष्य": {

"Person": ["मैं", "तुम", "वह", "हम", "तुम्हारे", "वे"],

"Gender": ["Masc", "Fem", "Neut"],

"Number": ["Sg", "Pl"]

},

"पक्षी": {

"Person": ["मैं", "तुम", "वह", "हम", "तुम्हारे", "वे"],

"Gender": ["Masc", "Fem", "Neut"],

"Number": ["Sg", "Pl"]

},

"शिशु": {

"Person": ["मैं", "तुम", "वह", "हम", "तुम्हारे", "वे"],

"Gender": ["Masc", "Fem", "Neut"],

"Number": ["Sg"]

},

"गुरु": {

"Person": ["मैं", "तुम", "वह", "हम", "तुम्हारे", "वे"],

"Gender": ["Masc"],

"Number": ["Sg"]

},

"नर": {

"Person": ["मैं", "तुम", "वह", "हम", "तुम्हारे", "वे"],

"Gender": ["Masc"],

"Number": ["Sg"]

}

}

# Construct the paradigm table for each word

from tabulate import tabulate

for word in words:

table = []

for person in categories["Person"]:

for gender in categories["Gender"]:

for number in categories["Number"]:

inflected\_word = inflections[word]["Person"][categories["Person"].index(person)]

inflected\_word += " " + inflections[word]["Gender"][categories["Gender"].index(gender)]

inflected\_word += " " + inflections[word]["Number"][categories["Number"].index(number)]

table.append((person, gender, number, inflected\_word))

print(f"Paradigm table for {word}:")

print(tabulate(table, headers=["Person", "Gender", "Number", "Inflected Word"], tablefmt="pipe"))

**output:**

**Q3. Observe the following words from Bengali. Identify all the morphemes and their corresponding meanings.**

* **kori '(I)do'**
* **maari '(I) hit'**
* **korchille '(You) were doing'**
* **maar '(You) hit'**

Source Code

**import nltk**

**from nltk import word\_tokenize**

**# Define the words**

**words = ["kori", "maari", "korchille", "maar"]**

**# Define the morpheme boundaries and their meanings**

**morphemes = {**

**"ko": "1s Subject Pronoun",**

**"ri": "Do Verb Stem",**

**"ma": "1s Subject Pronoun",**

**"ar": "Hit Verb Stem",**

**"chille": "2s Past Tense Auxiliary Verb",**

**}**

**# Define a function to split a word into its morphemes**

**def split\_word(word):**

**morpheme\_list = []**

**while len(word) > 0:**

**for morpheme, meaning in morphemes.items():**

**if word.endswith(morpheme):**

**morpheme\_list.append((morpheme, meaning))**

**word = word[:len(word)-len(morpheme)]**

**break**

**else:**

**# No morpheme found**

**return None**

**morpheme\_list.reverse()**

**return morpheme\_list**

**# Identify the morphemes and their meanings for each word**

**for word in words:**

**morpheme\_list = split\_word(word)**

**if morpheme\_list is None:**

**print(f"Could not identify morphemes for {word}")**

**else:**

**print(f"Morphemes for {word}:")**

**for morpheme, meaning in morpheme\_list:**

**print(f"{morpheme}: {meaning}")**