Question 1)
a)

The performance of mobile search engines can be improved in a number of ways by utilising Natural Language Processing (NLP):

1. Improved Query Understanding: By identifying the meaning and intent underlying the words, NLP approaches can aid mobile search engines in comprehending users' search queries. This entails accurately reading the user's search intent, finding pertinent terms, and analyzing the query's syntactic and semantic structure.

2. Personalization based on user choices : NLP algorithms can make recommendations for pertinent content and customize search results to each user's preferences by examining user interactions, search history, and contextual data.

3. Natural language Generation: NLP techniques can be used to create natural language responses or summaries for search results, which is known as natural language generation (NLG). This may enable mobile search engines to offer educational summaries or snippets that help users rapidly comprehend the content of search results without having to click on each one individually.

4. Enhanced relevancy Ranking: NLP can help to raise the ranking of search results' relevance. NLP algorithms can identify the relevance of search results to the user's query by examining the context. To deliver more precise and customized search results, this may incorporate techniques like semantic matching, word embeddings, and document similarity measurements.

5. Sentiment Analysis: NLP can be used to examine the sentiment contained in user-generated content connected to search queries or search results, such as reviews or social media posts. Mobile search engines can better grasp user preferences, opinions, and sentiment towards particular topics or entities with the aid of this sentiment analysis, resulting in more relevant and personalized search results.

6. Entity Recognition and Linking: Search queries and content can contain named entities that NLP algorithms can recognise and extract. Search engines can deliver more accurate and contextually relevant search results, particularly when handling ambiguous questions, by recognising entities such as people, places, organizations, and particular products.

Mobile search engines may greatly enhance the accuracy, relevance, and personalisation of search results by utilizing these NLP approaches, providing users with a better overall search experience.

Natural Language Processing (NLP) can be used to improve the performance of mobile search engines in a number of ways, including the following:

1. Techniques for comprehending queries include:
   a. Part-of-Speech Tagging, which can be used to grasp the grammatical structure of a query and give words the proper weights based on their type.

   b. Named Entity Recognition (NER): NER approaches can recognise and extract named entities, such as individuals, places, businesses, and particular goods mentioned in the query, which helps users better understand and apply it to real-world data.

2. Word Embeddings: Words can be represented as dense vector representations which capture semantic links and similarity. This makes it possible to comprehend query intent and context better.

3. A word's function in a phrase, such as its subject, object, or predicate, is identified using a technique called semantic role labeling (SRL). This aids in query understanding and relevance ranking by illuminating the connections between entities and activities in a query.

4. Techniques for Personalization:
   a. Collaborative Filtering: By examining user preferences and behavior, collaborative filtering can provide personalized search results based on the habits and interests of like users.

   b. User Profiling: Based on search history, demographics, and interests, user profiles can be created using user profiling techniques. This makes it possible for personalized search results and recommendations that are in line with user

5. Natural Language Generation (NLG):
   a. Text summarizing: NLP methods like extractive or abstractive summarizing are able to provide succinct summaries of search results, giving consumers a rapid summary without requiring them to read the entire document.

   b. Question Answering: By using question-answering models, search engines are able to produce direct responses to user inquiries, enhancing the search experience and offering quick replies.

6. Mobile search engines can utilize NLP to better comprehend user queries, improve relevance ranking, enable personalisation, and produce informative summaries. As a result, they can provide users with more precise and customized search results.

QUESTION 2)

A)

The robot described is a mobile robot, specifically designed for smart room-cleaning tasks. It is similar in functionality to a Roomba, which is a popular brand of floor-cleaning robot. The robot is equipped with features and capabilities that allow it to navigate through the house autonomously, detect obstacles in its path, collect dust and dirt, sweep the floors, and remove dirt efficiently. Its purpose is to provide automated and convenient cleaning solutions for various types of floor surfaces within a home environment.

B)

A combination of active and passive sensors can be used in the smart room-cleaning robot.

- Active sensors: Examples of active sensors in the context of the cleaning robot are infrared (IR) sensors and ultrasonic sensors. By producing infrared beams and observing their reflection, IR sensors can be utilized to locate things or obstacles or a stair or a height difference . The robot can sense distances and avoid collisions thanks to ultrasonic sensors, which send out sound waves and track the return of those waves.

- Passive sensors: A camera or vision sensor can be used as a passive sensor in the context of the cleaning robot. It takes in visual data and processes it to identify items, different kinds of flooring, and dust or debris on the floors.

- The cleaning robot can improve its perception abilities by combining active and passive sensors. While passive sensors provide visual information and enable the detection of dirt or dust particles on the floor, active sensors aid in the rapid detection of obstacles and navigation. This enables the robot to go across the house quickly, avoid obstacles, and gather dust while delivering

C)

Actuators are required for the robot to be able to move around the room and use the mobility effector. These might feature tracks or wheels with motors that let the robot move across various floor surfaces and around obstacles.

- Cleaning Effector: The robot would need a cleaning mechanism to gather dust from the flooring. This could take the shape of brushes or rollers that sweep or stir the floor surface, displacing and catching the dust and dirt particles.

- Dust Collection Effector: The robot needs an effector to collect and store the dust and dirt after they have been loosened. This could be a trash can or a gathering area where the gathered rubbish is kept while the cleaning is being done.

- Mopping Effector: If the robot is intended to clean floors or perform wet mopping, it will need an additional effector. This could be a mopping pad or a spray device to wet the floor surface with water or a cleaning agent, followed by a device to mop or wipe the wet surface.

- Identifying and avoiding obstacles Effector: The robot needs an effector to recognise obstacles in its path and maneuver past them, such as sensors or actuators. To change course and prevent crashes, this can entail motorized steering or maneuvering systems.

These effectors allow the robot to move around the space efficiently, gather debris from the floors, and perhaps even undertake mopping or wet cleaning duties. The robot can effectively clean the space while adapting to various floor surfaces and dodging obstacles thanks to the combination of its movement, cleaning, dust collecting, and obstacle avoidance effectors.

D)

- Environmental Knowledge: The robot should be aware of the room's layout, including where furniture, walls, and other impediments are located (Mapping). This data can be kept in a spatial representation, such a 2D map or a 3D environment model.

- Recognition and classification of things: The robot should be able to identify and classify various objects that are frequently seen in a room. With this information, the robot can accurately identify items for navigation and cleaning chores.

- Cleaning Techniques and Strategies: The robot should be knowledgeable about various cleaning methods and strategies. This entails comprehending the best cleaning path, spotting places that need additional focus, and modifying the cleaning strategy dependent on the sort of floor or surface.

- Detection of dust and dirt: The robot should be able to recognise the traits and look of dust and dirt. The robot can successfully detect and identify places that need cleaning, as well as choose the best cleaning procedure, thanks to this expertise.

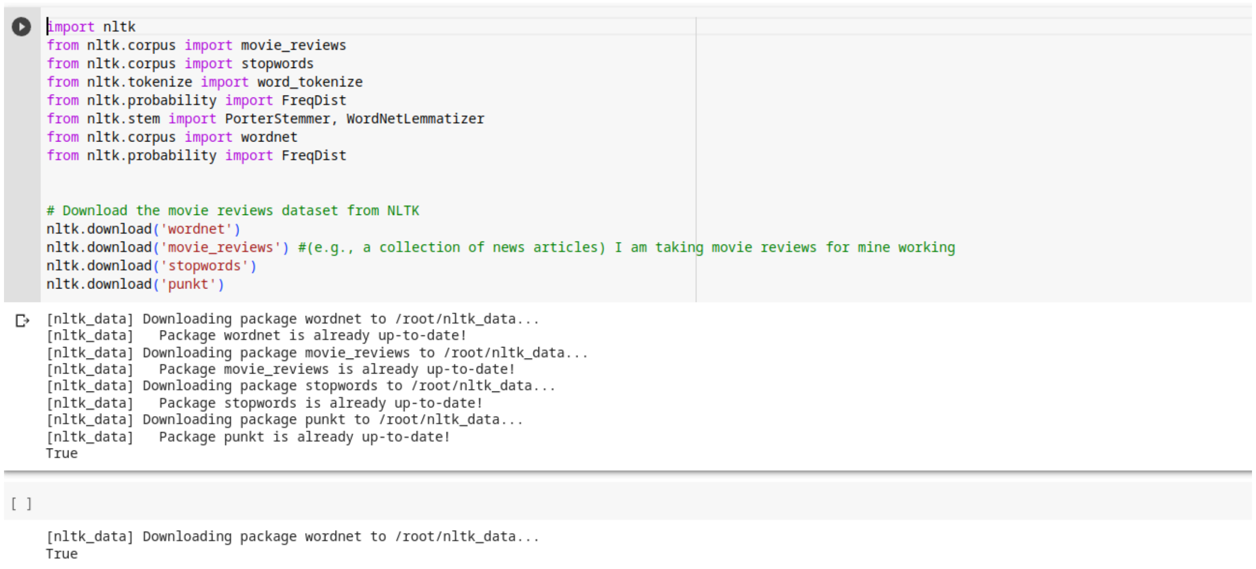
- Task Scheduling and Planning: The cleaning robot should be able to prioritize its jobs based on their urgency or priority and schedule its activities accordingly. Thinking critically about the environment's current status, cleaning goals, and resource constraints is required.

- Learning and Adaptation: The robot needs to be able to draw lessons from its encounters and exchanges. In order to continuously enhance performance, it can update its knowledge base by getting user feedback, examining cleaning outcomes, and changing its behavior.

To analyze and infer information from the knowledge base, the reasoning system may use strategies like rule-based reasoning, symbolic reasoning, or probabilistic reasoning. On the basis of customer feedback and historical data, machine learning algorithms can also be used to understand trends, preferences, and optimize cleaning techniques.

The intelligent behavior of the smart room-cleaning robot and its ability to adapt to various cleaning situations are made possible by a knowledge base and reasoning system that takes into account these factors.

Q2) A) # these are well commented codes with answers



```
import nltk
from nltk.corpus import movie_reviews
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.probability import FreqDist


# Download the movie reviews dataset from NLTK
nltk.download('wordnet')
nltk.download('movie_reviews') #(e.g., a collection of news articles) I am taking movie reviews for mine working
nltk.download('stopwords')
nltk.download('punkt')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
[ ]
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

Stemming: Stemming is the process of removing affixes from words to obtain their base form, known as the stem. This involves heuristics and rule-based approaches to remove common suffixes and prefixes. The resulting stems may not always be actual words, but they represent the core meaning of the word. Stemming helps to reduce word variations and consolidate related words.

Lemmatization: Lemmatization, on the other hand, aims to determine the base form of a word, known as the lemma, using linguistic knowledge and morphological analysis. The lemma is a valid word that represents the canonical form of the word. Unlike stemming, lemmatization considers the context and part of speech (POS) of the word, ensuring that the resulting lemma is a valid word in the language. Lemmatization provides more accurate base forms, facilitating better understanding and analysis of text data.

```python
# Get user input
user_input = input("enter text to be stemmed or lemmatized")

# Tokenize the user input
tokens = nltk.word_tokenize(user_input)

# Initialize stemming and lemmatization tools
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# Perform stemming
stemmed_tokens = [stemmer.stem(token) for token in tokens]

# Perform lemmatization
lemmatized_tokens = [lemmatizer.lemmatize(token, wordnet.VERB) for token in tokens]

# Print the stemmed tokens
print("Stemmed Tokens:", stemmed_tokens)

# Print the lemmatized tokens
print("Lemmatized Tokens:", lemmatized_tokens)
```

```
enter text to be stemmed or lemmatizedThe quick brown fox jumps over the lazy dog
Stemmed Tokens: ['the', 'quick', 'brown', 'fox', 'jump', 'over', 'the', 'lazi', 'dog']
Lemmatized Tokens: ['The', 'quick', 'brown', 'fox', 'jump', 'over', 'the', 'lazy', 'dog']
```

Most Frequent word

```
[ ]

    # Get user input
    user_input = input("Enter the text: ")

    # Tokenize the user input
    tokens = nltk.word_tokenize(user_input)

    # Create a frequency distribution of the tokens
    freq_dist = FreqDist(tokens)

    # Get the most common word and its frequency
    most_common_word, frequency = freq_dist.most_common(1)[0]

    # Print the most frequent word and its frequency
    print("Most Frequent Word:", most_common_word)
    print("Frequency:", frequency)
```

```
    Enter the text: hello hello my name is krishna shukla
    Most Frequent Word: hello
    Frequency: 2
```

Stopwords are common words that are often considered insignificant or irrelevant in the analysis of text data. These words, such as "the," "and," "a," and "in," do not carry much semantic meaning and occur frequently across different documents or texts. Stopwords are often removed from the text during preprocessing to improve the efficiency and accuracy of natural language processing tasks, such as text classification, information retrieval, and sentiment analysis. Removing stopwords helps to reduce noise and focus on more meaningful words that carry the essence of the text.

```
[ ] user_input = input("Enter the text to tokenize and remove stop words: ")

    # Tokenize the user input
    tokens = nltk.word_tokenize(user_input)

    # Remove stop words
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [token for token in tokens if token.lower() not in stop_words]

    # Print the filtered tokens
    print("Filtered Tokens:", filtered_tokens)
```

```
    Enter the text to tokenize and remove stop words: and you are the one for whom i am living
    Filtered Tokens: ['one', 'living']
```

Stemming: Stemming is the process of removing affixes from words to obtain their base form, known as the stem. This involves heuristics and rule-based approaches to remove common suffixes and prefixes. The resulting stems may not always be actual words, but they represent the core meaning of the word. Stemming helps to reduce word variations and consolidate related words.

Lemmatization: Lemmatization, on the other hand, aims to determine the base form of a word, known as the lemma, using linguistic knowledge and morphological analysis. The lemma is a valid word that represents the canonical form of the word. Unlike stemming, lemmatization considers the context and part of speech (POS) of the word, ensuring that the resulting lemma is a valid word in the language. Lemmatization provides more accurate base forms, facilitating better understanding and analysis of text data.

```
# Get user input
user_input = input("enter text to be stemmed or lemmatized")

# Tokenize the user input
tokens = nltk.word_tokenize(user_input)

# Initialize stemming and lemmatization tools
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# Perform stemming
stemmed_tokens = [stemmer.stem(token) for token in tokens]

# Perform lemmatization
lemmatized_tokens = [lemmatizer.lemmatize(token, wordnet.VERB) for token in tokens]

# Print the stemmed tokens
print("Stemmed Tokens:", stemmed_tokens)

# Print the lemmatized tokens
print("Lemmatized Tokens:", lemmatized_tokens)
```

```
enter text to be stemmed or lemmatizedThe quick brown fox jumps over the lazy dog
Stemmed Tokens: ['the', 'quick', 'brown', 'fox', 'jump', 'over', 'the', 'lazi', 'dog']
Lemmatized Tokens: ['The', 'quick', 'brown', 'fox', 'jump', 'over', 'the', 'lazy', 'dog']
```

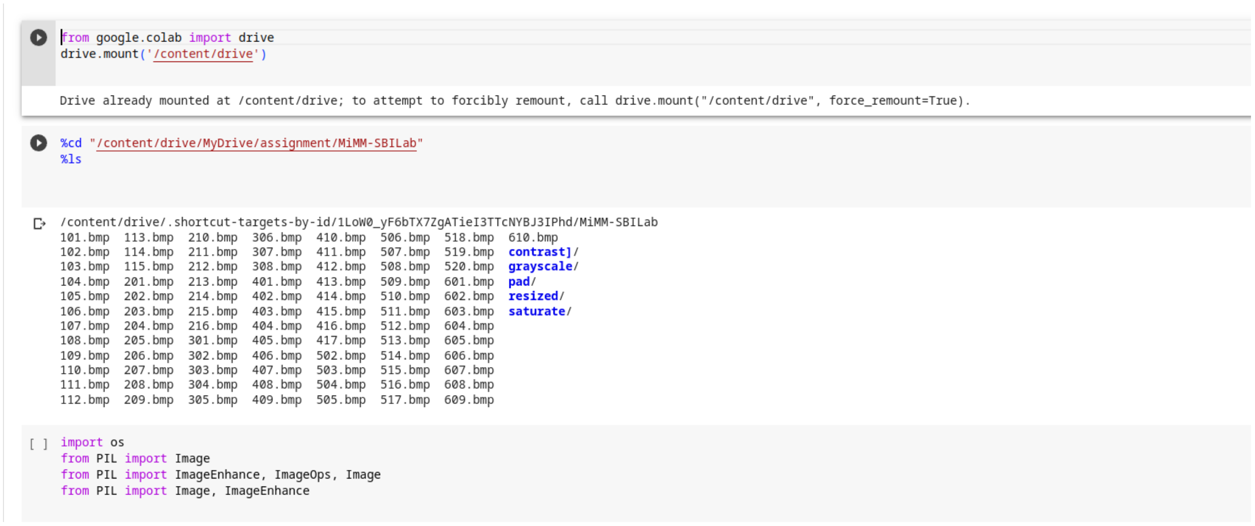
Most Frequent word

[ ]

```
# Get user input
user_input = input("Enter the text: ")

# Tokenize the user input
tokens = nltk.word_tokenize(user_input)

# Create a frequency distribution of the tokens
freq_dist = FreqDist(tokens)

# Get the most common word and its frequency
most_common_word, frequency = freq_dist.most_common(1)[0]

# Print the most frequent word and its frequency
print("Most Frequent Word:", most_common_word)
print("Frequency:", frequency)
```

```
Enter the text: hello hello my name is krishna shukla
Most Frequent Word: hello
Frequency: 2
```

Q2)b)# Well commented codes with oytput



```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
%cd "/content/drive/MyDrive/assignment/MiMM-SBILab"
%ls
```

```
/content/drive/.shortcut-targets-by-id/1LoW0_yF6bTX7ZgATieI3TTcNYBJ3IPhd/MiMM-SBILab
101.bmp  113.bmp  210.bmp  306.bmp  410.bmp  506.bmp  518.bmp  610.bmp
102.bmp  114.bmp  211.bmp  307.bmp  411.bmp  507.bmp  519.bmp  contrast]/
103.bmp  115.bmp  212.bmp  308.bmp  412.bmp  508.bmp  520.bmp  grayscale/
104.bmp  201.bmp  213.bmp  401.bmp  413.bmp  509.bmp  601.bmp  pad/
105.bmp  202.bmp  214.bmp  402.bmp  414.bmp  510.bmp  602.bmp  resized/
106.bmp  203.bmp  215.bmp  403.bmp  415.bmp  511.bmp  603.bmp  saturate/
107.bmp  204.bmp  216.bmp  404.bmp  416.bmp  512.bmp  604.bmp
108.bmp  205.bmp  301.bmp  405.bmp  417.bmp  513.bmp  605.bmp
109.bmp  206.bmp  302.bmp  406.bmp  502.bmp  514.bmp  606.bmp
110.bmp  207.bmp  303.bmp  407.bmp  503.bmp  515.bmp  607.bmp
111.bmp  208.bmp  304.bmp  408.bmp  504.bmp  516.bmp  608.bmp
112.bmp  209.bmp  305.bmp  409.bmp  505.bmp  517.bmp  609.bmp
```

```
import os
from PIL import Image
from PIL import ImageEnhance, ImageOps, Image
from PIL import Image, ImageEnhance
```

Saturation enhancement, on the other hand, focuses on intensifying the colors in an image. By increasing the saturation, the vibrancy and richness of the colors are enhanced. This adjustment can make the image appear more vivid and visually appealing, as the colors become more saturated and vibrant without affecting the overall brightness or contrast.

```
output_directory =  '/content/drive/MyDrive/assignment/MiMM-SBILab/saturate' # Replace with the desired output directory path


if not os.path.exists(output_directory):
    os.makedirs(output_directory)


for filename in os.listdir(dataset_directory):
    if filename.endswith('.bmp'):
        image_path = os.path.join(dataset_directory, filename)
        img = Image.open(image_path)


        # Enhance the saturation of the image
        saturation_enhancer = ImageEnhance.Color(img)
        saturation_factor = 1.5  # Adjust the factor for desired saturation enhancement
        saturated_img = saturation_enhancer.enhance(saturation_factor)


        # Save the saturated image
        output_path = os.path.join(output_directory, filename)
        saturated_img.save(output_path)

        img.close()


print("Saturation enhancement complete!")
```

```
Saturation enhancement complete!
```

Contrast enhancement is a technique used to increase the visual difference between the light and dark areas of an image. It involves adjusting the range of pixel intensities, making the dark areas darker and the light areas lighter. The goal is to improve the overall contrast, making the image appear sharper and more defined.

```
 # Replace with the desired output directory path
output_directory = '/content/drive/MyDrive/assignment/MiMM-SBILab/contrast]'
if not os.path.exists(output_directory):
    os.makedirs(output_directory)


for filename in os.listdir(dataset_directory):


    if filename.endswith('.bmp'):
        image_path = os.path.join(dataset_directory, filename)
        img = Image.open(image_path)


        # Enhance the contrast of the image
        contrast_enhancer = ImageEnhance.Contrast(img)
        contrast_factor = 1.5  # Adjust the factor for desired contrast enhancement
        contrast_img = contrast_enhancer.enhance(contrast_factor)


        # Save the contrast-enhanced image
        output_path = os.path.join(output_directory, filename)
        contrast_img.save(output_path)

        img.close()


print("Contrast enhancement complete!")
```

```
Contrast enhancement complete!
```

Grayscale refers to the process of converting an image from its original color representation to shades of gray. In grayscale images, each pixel is represented by a single intensity value, ranging from black (0) to white (255), with various shades of gray in between. This conversion removes the color information from the image, resulting in a grayscale representation that highlights the luminance or brightness values of the original image.

```
output_directory = '/content/drive/MyDrive/assignment/MiMM-SBILab/grayscale'

if not os.path.exists(output_directory):
    os.makedirs(output_directory)


for filename in os.listdir(dataset_directory):
    if filename.endswith('.bmp'):
        image_path = os.path.join(dataset_directory, filename)
        img = Image.open(image_path)



        # Convert the image to grayscale
        grayscale_img = img.convert('L')



        # Save the grayscale image
        output_path = os.path.join(output_directory, filename)
        grayscale_img.save(output_path)

        img.close()



print("Grayscale conversion complete!")
```

```
Grayscale conversion complete!
```

Image padding is the process of adding extra pixels around the edges of an image to increase its size or create a border. It helps maintain the image's aspect ratio and can be used for various purposes, such as preserving proportions when resizing or creating a visual boundary.

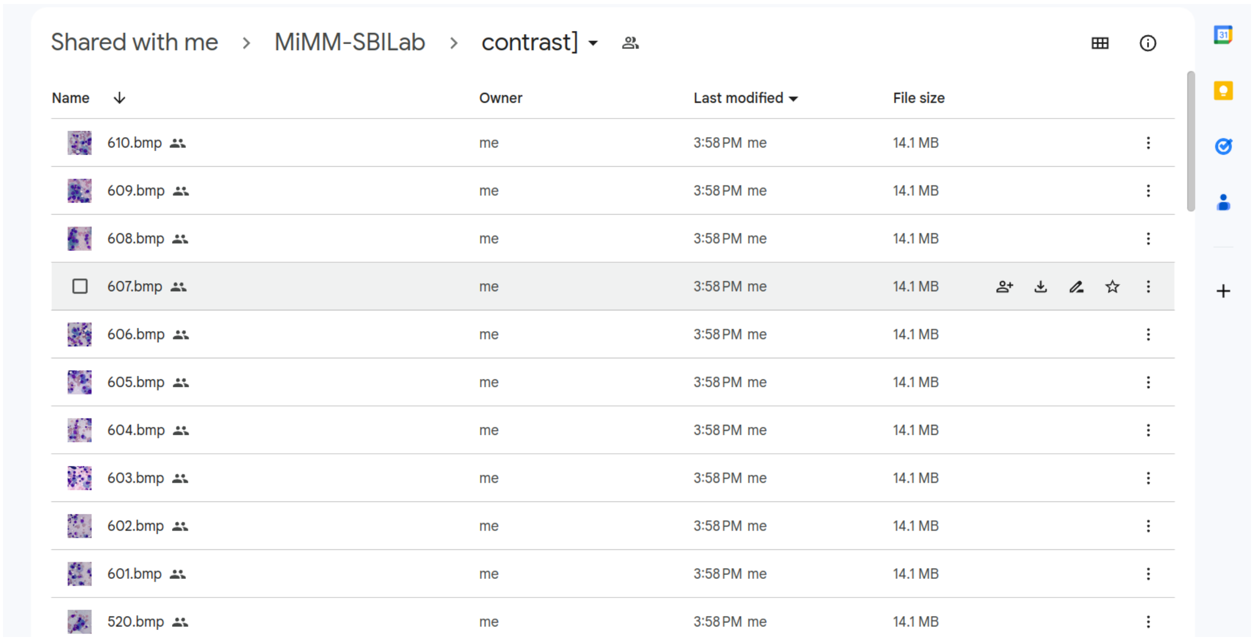
```
[ ]

dataset_directory = '/content/drive/MyDrive/assignment/MiMM-SBILab'


output_directory = '/content/drive/MyDrive/assignment/MiMM-SBILab/pad'
if not os.path.exists(output_directory):
    os.makedirs(output_directory)

for filename in os.listdir(dataset_directory):
    if filename.endswith('.bmp'):
        image_path = os.path.join(dataset_directory, filename)
        img = Image.open(image_path)

        # Pad the image
        padded_img = ImageOps.pad(img, (img.width + 100, img.height + 100), color='white')


        output_path = os.path.join(output_directory, filename)
        padded_img.save(output_path)

        img.close()

print("Image padding complete!")
```

```
Image padding complete!
```

```
directory = '/content/drive/MyDrive/assignment/MiMM-SBILab'
new_directory = '/content/drive/MyDrive/assignment/MiMM-SBILab/resized'

# Create a new directory for resized images if it doesn't exist
if not os.path.exists(new_directory):
    os.makedirs(new_directory)

# Iterate over each file in the directory
for filename in os.listdir(directory):
    if filename.endswith('.bmp'):
        # Open the image file
        image_path = os.path.join(directory, filename)
        img = Image.open(image_path)

        # Resize the image (you can change the desired dimensions)
        new_width, new_height = 800, 600
        resized_img = img.resize((new_width, new_height))

        # Save the resized image to the new directory
        new_path = os.path.join(new_directory, filename)
        resized_img.save(new_path)

        # Close the image file
        img.close()

print("Image resizing complete!")
```

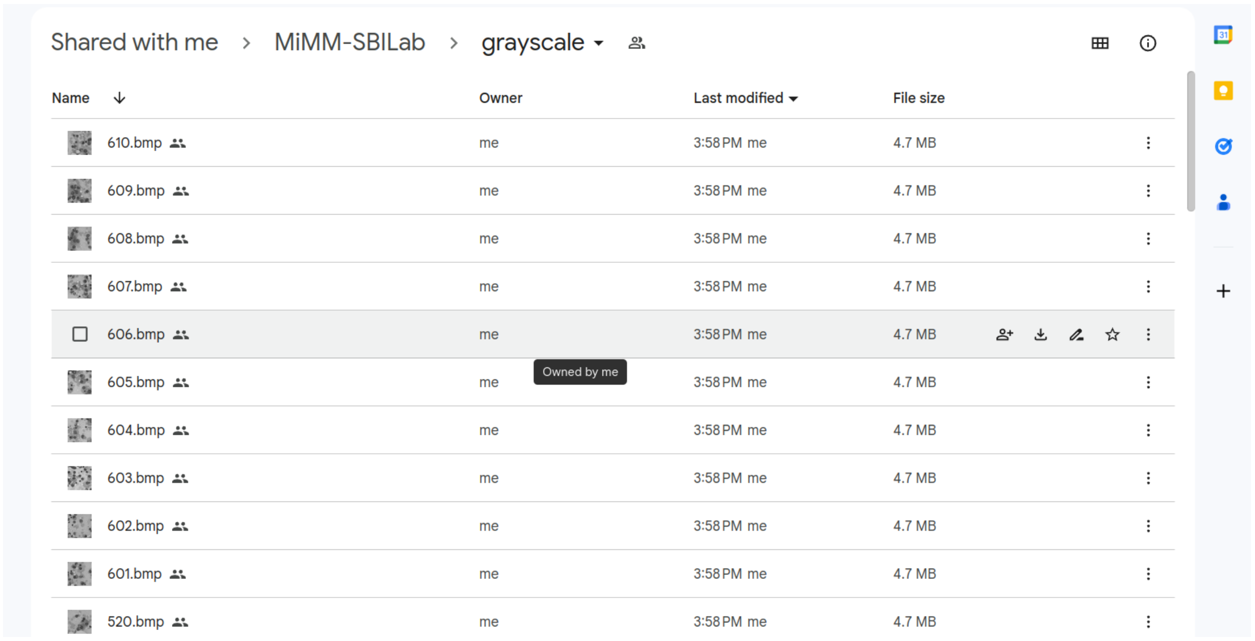
```
Image resizing complete!
```

Image padding is the process of adding extra pixels around the edges of an image to increase its size or create a border. It helps maintain the image's aspect ratio and can be used for various purposes, such as preserving proportions when resizing or creating a visual boundary.



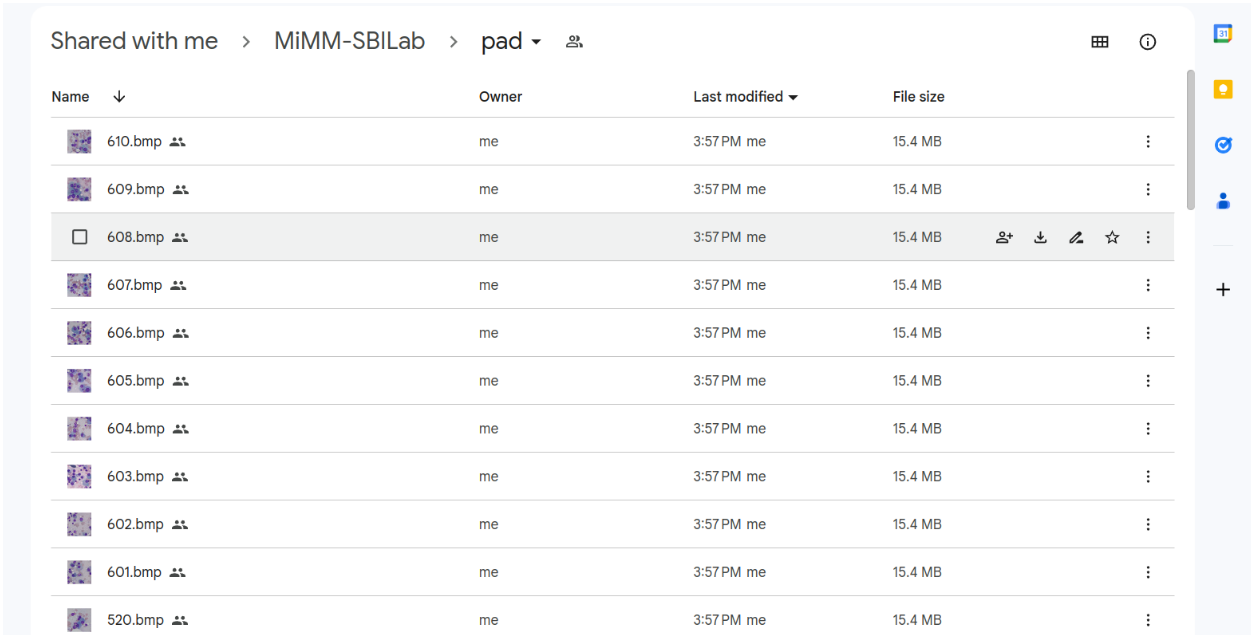
Shared with me › MiMM-SBILab › contrast]

Name	Owner	Last modified	File size
610.bmp	me	3:58 PM me	14.1 MB
609.bmp	me	3:58 PM me	14.1 MB
608.bmp	me	3:58 PM me	14.1 MB
607.bmp	me	3:58 PM me	14.1 MB
606.bmp	me	3:58 PM me	14.1 MB
605.bmp	me	3:58 PM me	14.1 MB
604.bmp	me	3:58 PM me	14.1 MB
603.bmp	me	3:58 PM me	14.1 MB
602.bmp	me	3:58 PM me	14.1 MB
601.bmp	me	3:58 PM me	14.1 MB
520.bmp	me	3:58 PM me	14.1 MB



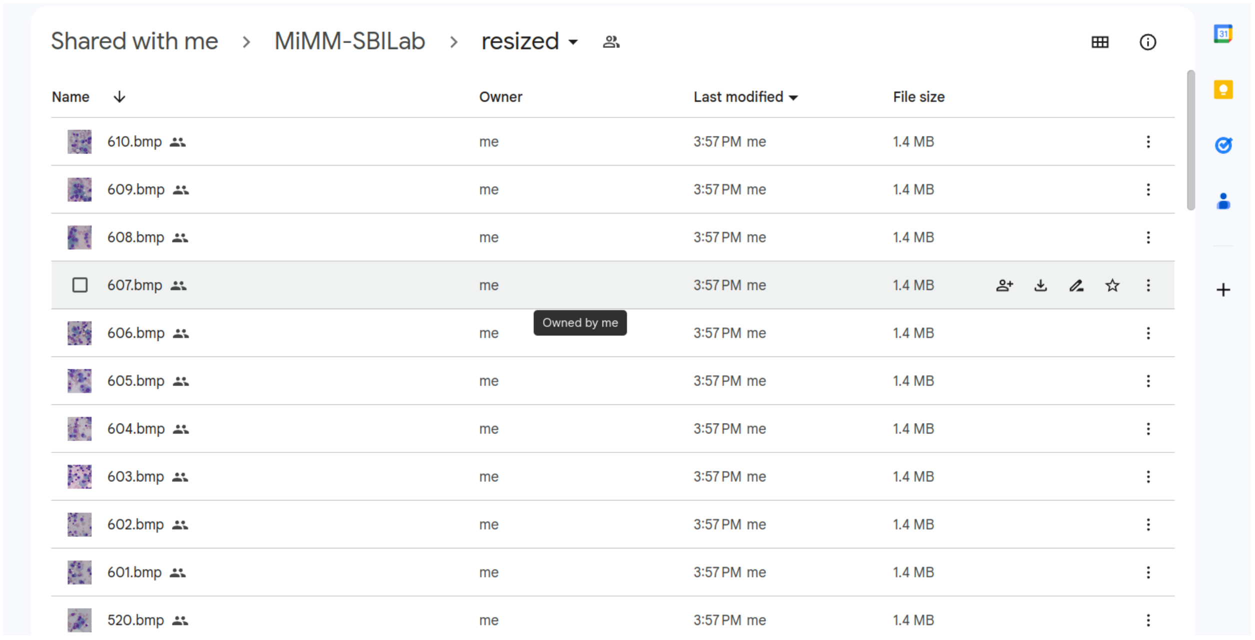
Shared with me › MiMM-SBILab › grayscale

Name	Owner	Last modified	File size
610.bmp	me	3:58 PM me	4.7 MB
609.bmp	me	3:58 PM me	4.7 MB
608.bmp	me	3:58 PM me	4.7 MB
607.bmp	me	3:58 PM me	4.7 MB
606.bmp	me	3:58 PM me	4.7 MB
605.bmp	me	3:58 PM me	4.7 MB
604.bmp	me	3:58 PM me	4.7 MB
603.bmp	me	3:58 PM me	4.7 MB
602.bmp	me	3:58 PM me	4.7 MB
601.bmp	me	3:58 PM me	4.7 MB
520.bmp	me	3:58 PM me	4.7 MB

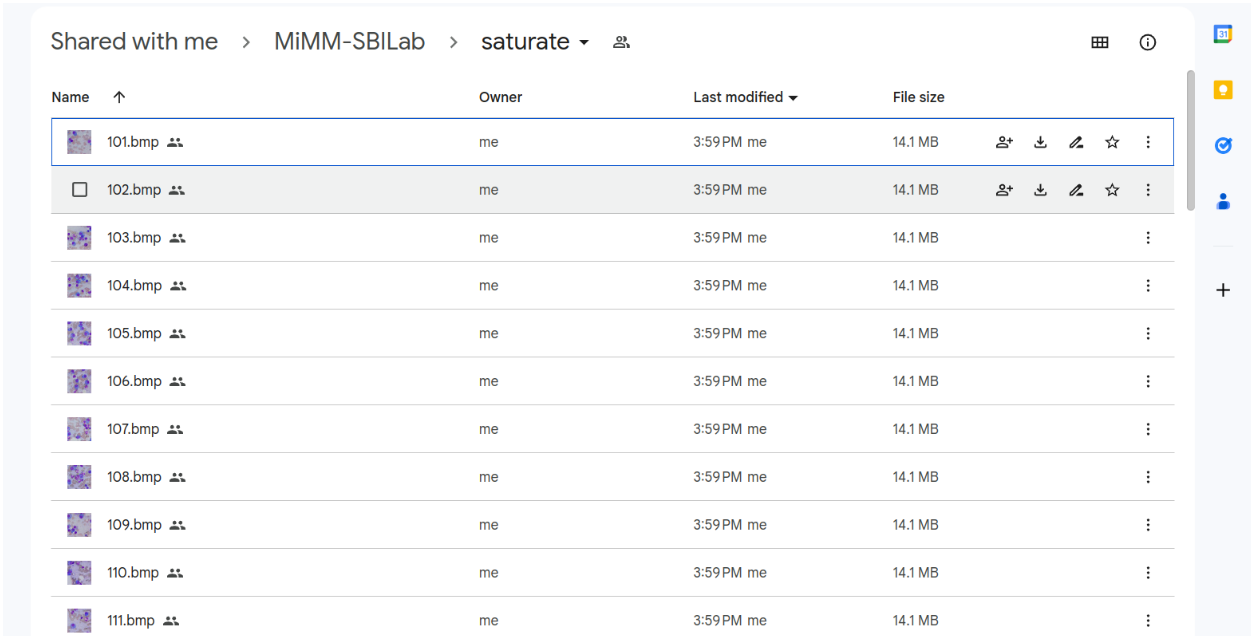


Shared with me › MiMM-SBILab › pad

Name	Owner	Last modified	File size
610.bmp	me	3:57 PM me	15.4 MB
609.bmp	me	3:57 PM me	15.4 MB
608.bmp	me	3:57 PM me	15.4 MB
607.bmp	me	3:57 PM me	15.4 MB
606.bmp	me	3:57 PM me	15.4 MB
605.bmp	me	3:57 PM me	15.4 MB
604.bmp	me	3:57 PM me	15.4 MB
603.bmp	me	3:57 PM me	15.4 MB
602.bmp	me	3:57 PM me	15.4 MB
601.bmp	me	3:57 PM me	15.4 MB
520.bmp	me	3:57 PM me	15.4 MB


Shared with me > MiMM-SBILab > resized

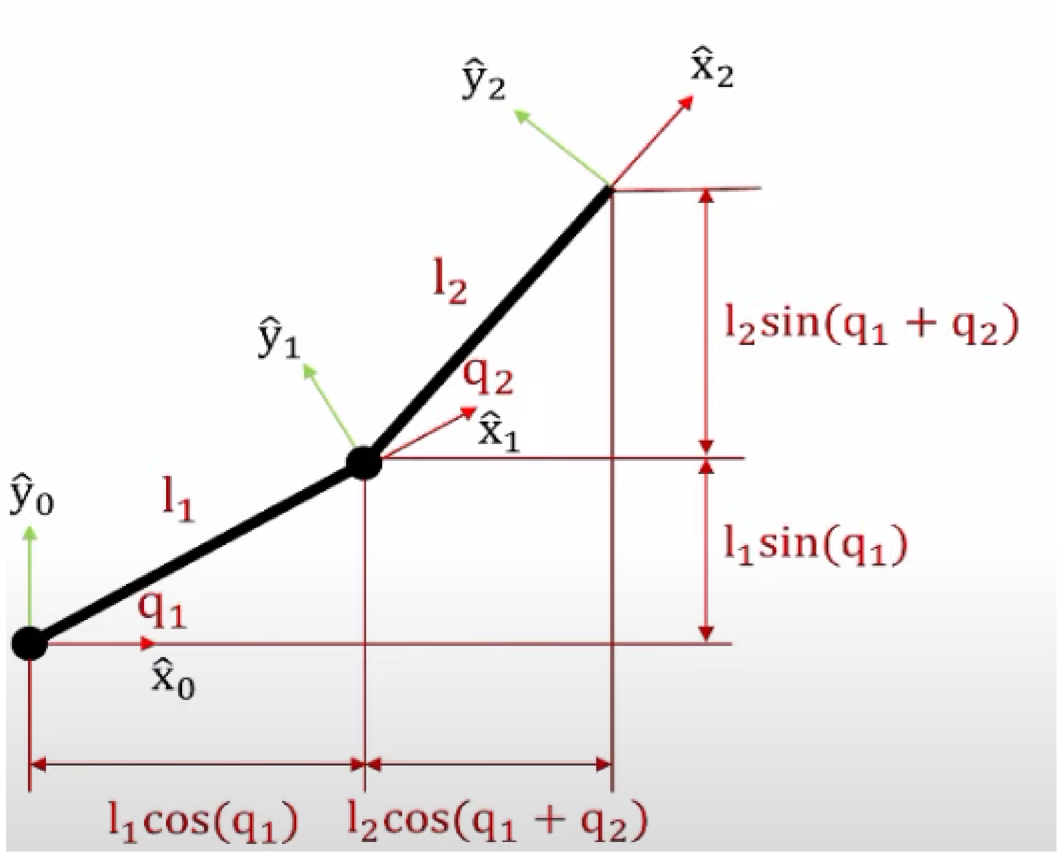
Name	Owner	Last modified	File size
610.bmp	me	3:57 PM me	1.4 MB
609.bmp	me	3:57 PM me	1.4 MB
608.bmp	me	3:57 PM me	1.4 MB
607.bmp	me	3:57 PM me	1.4 MB
606.bmp	me	3:57 PM me	1.4 MB
605.bmp	me	3:57 PM me	1.4 MB
604.bmp	me	3:57 PM me	1.4 MB
603.bmp	me	3:57 PM me	1.4 MB
602.bmp	me	3:57 PM me	1.4 MB
601.bmp	me	3:57 PM me	1.4 MB
520.bmp	me	3:57 PM me	1.4 MB

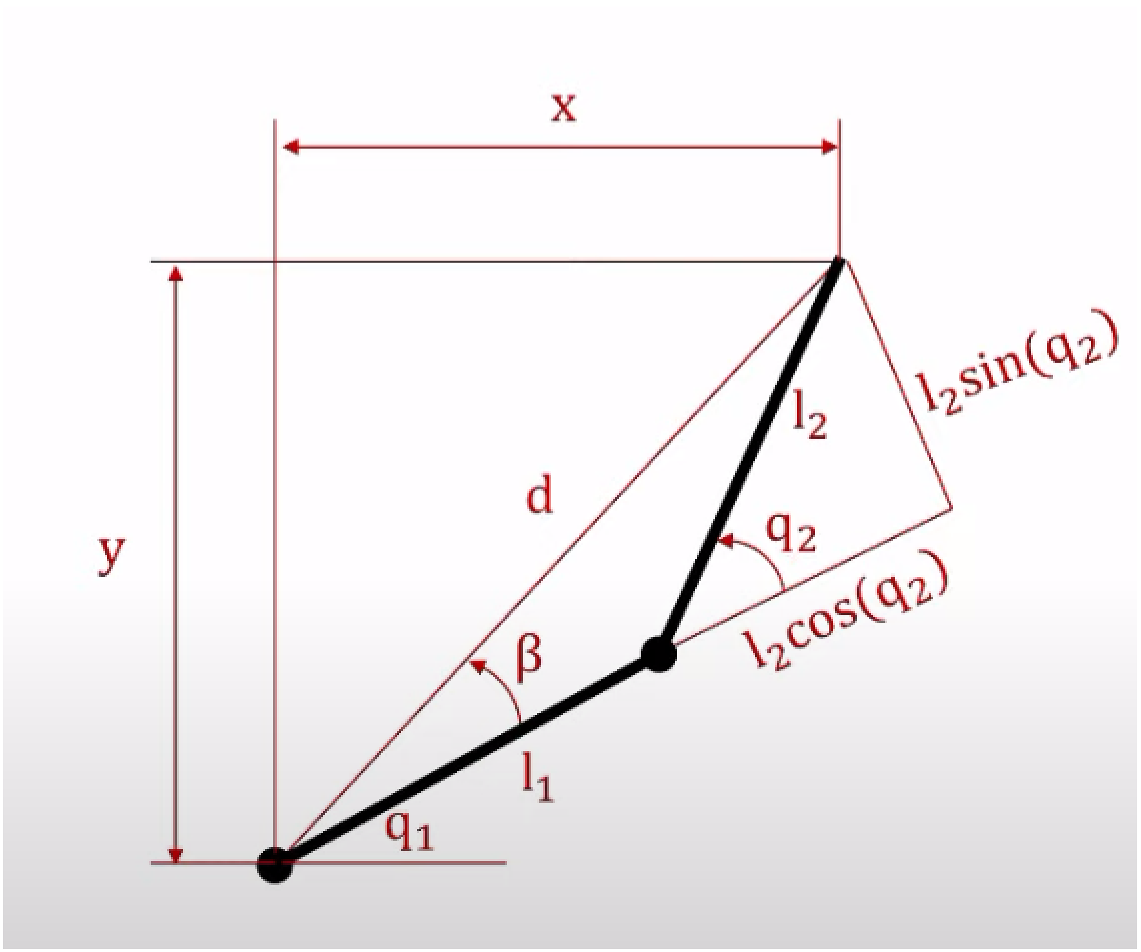

Shared with me > MiMM-SBILab > saturate

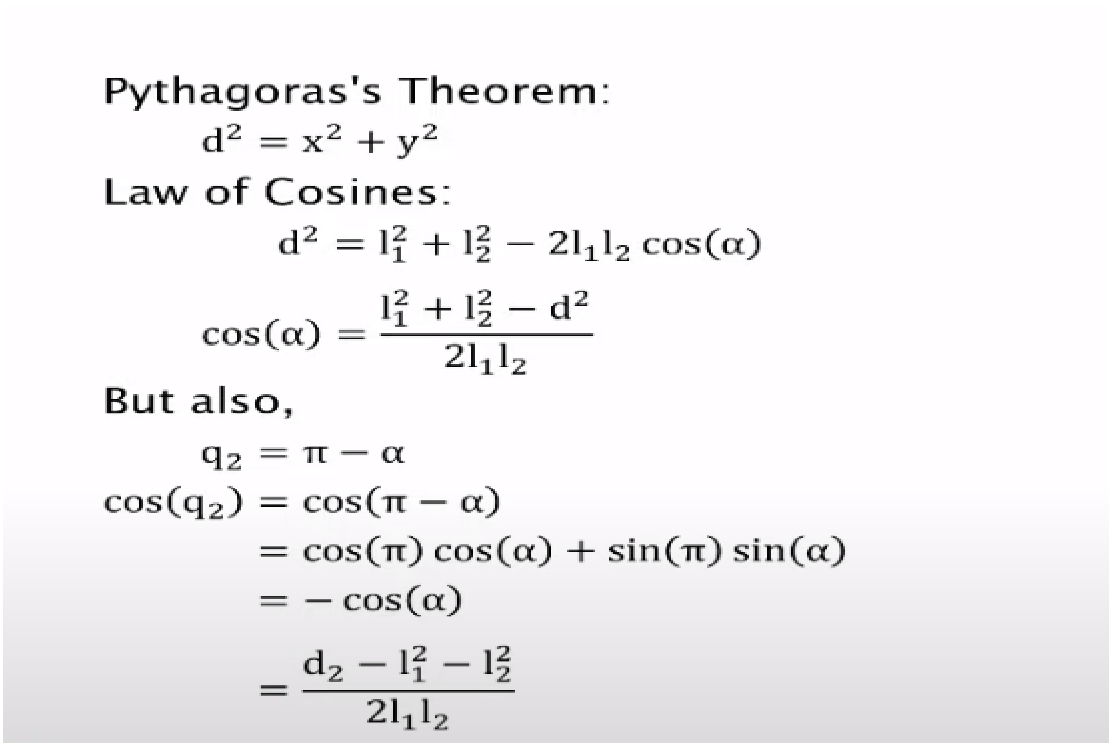
Name	Owner	Last modified	File size
101.bmp	me	3:59 PM me	14.1 MB
102.bmp	me	3:59 PM me	14.1 MB
103.bmp	me	3:59 PM me	14.1 MB
104.bmp	me	3:59 PM me	14.1 MB
105.bmp	me	3:59 PM me	14.1 MB
106.bmp	me	3:59 PM me	14.1 MB
107.bmp	me	3:59 PM me	14.1 MB
108.bmp	me	3:59 PM me	14.1 MB
109.bmp	me	3:59 PM me	14.1 MB
110.bmp	me	3:59 PM me	14.1 MB
111.bmp	me	3:59 PM me	14.1 MB

Inverse Kinematics is the process of determining the joint angles or control inputs required to achieve a desired end effector position and Orientation

Q3)c)







Pythagoras's Theorem:

$$d^2 = x^2 + y^2$$

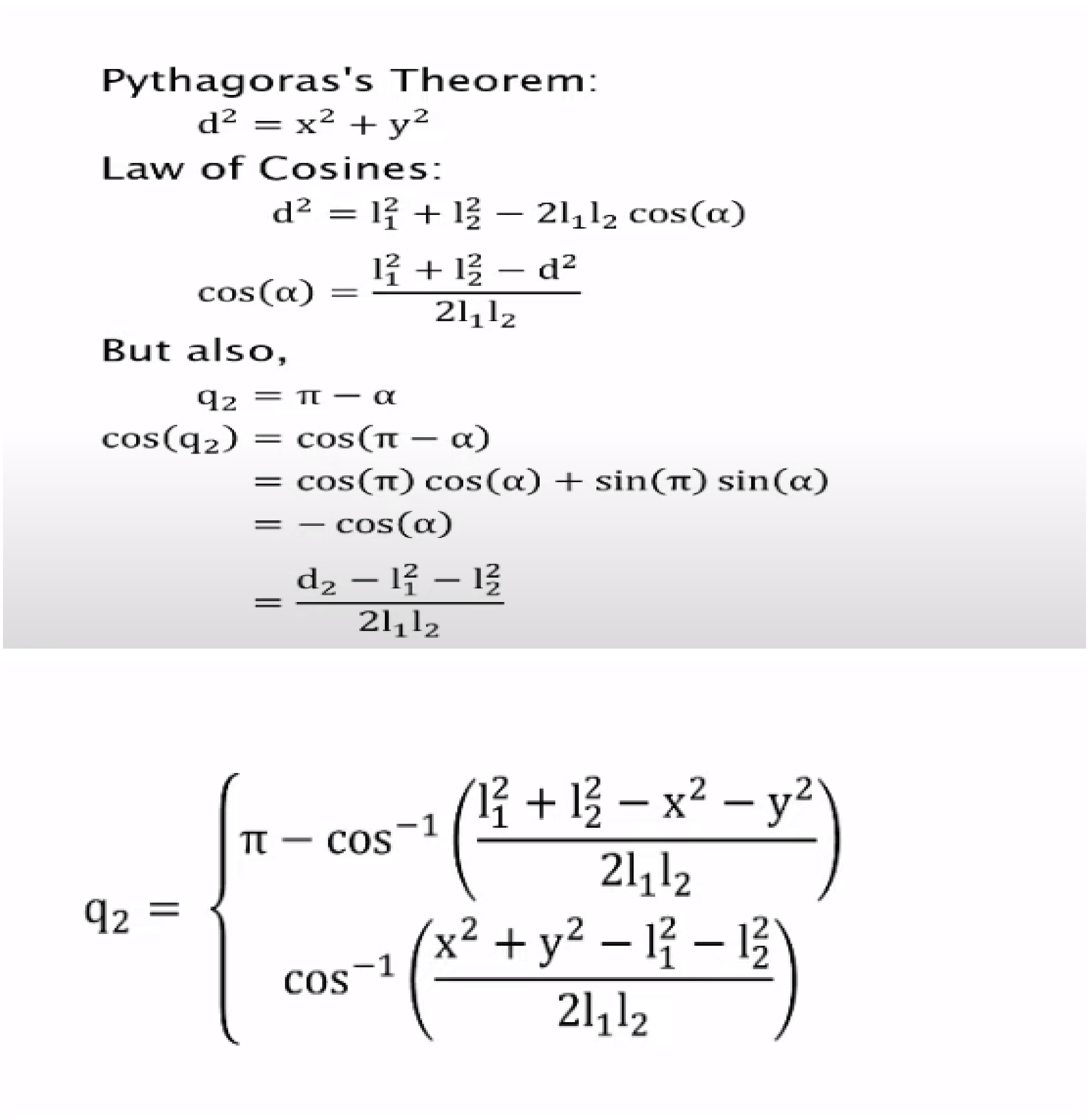
Law of Cosines:

$$d^2 = l_1^2 + l_2^2 - 2l_1l_2\cos(\alpha)$$

$$\cos(\alpha) = \frac{l_1^2 + l_2^2 - d^2}{2l_1l_2}$$

But also,

$$\begin{aligned} q_2 &= \pi - \alpha \\ \cos(q_2) &= \cos(\pi - \alpha) \\ &= \cos(\pi)\cos(\alpha) + \sin(\pi)\sin(\alpha) \\ &= -\cos(\alpha) \\ &= \frac{d_2 - l_1^2 - l_2^2}{2l_1l_2} \end{aligned}$$



Pythagoras's Theorem:

$$d^2 = x^2 + y^2$$

Law of Cosines:

$$d^2 = l_1^2 + l_2^2 - 2l_1l_2\cos(\alpha)$$

$$\cos(\alpha) = \frac{l_1^2 + l_2^2 - d^2}{2l_1l_2}$$

But also,

$$q_2 = \pi - \alpha$$

$$\begin{aligned}\cos(q_2) &= \cos(\pi - \alpha)\\ &= \cos(\pi)\cos(\alpha) + \sin(\pi)\sin(\alpha)\\ &= -\cos(\alpha)\\ &= \frac{d_2 - l_1^2 - l_2^2}{2l_1l_2}\end{aligned}$$

$$q_2 = \begin{cases}\pi - \cos^{-1}\left(\dfrac{l_1^2 + l_2^2 - x^2 - y^2}{2l_1l_2}\right)\\ \cos^{-1}\left(\dfrac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1l_2}\right)\end{cases}$$

Using tangent rule:

$$\tan(q_1 + \beta) = \frac{y}{x}$$

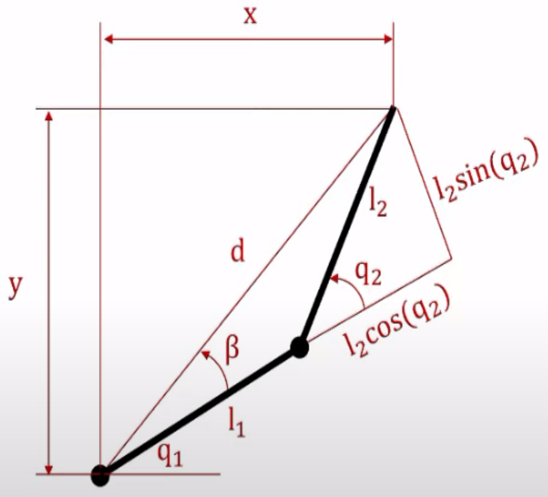
$$q_1 = \tan^{-1}\left(\frac{y}{x}\right) - \beta$$

Using Law of Cosines:

$$\beta = \tan^{-1}\left(\frac{l_2 \sin(q_2)}{l_1 + l_2 \cos(q_2)}\right)$$

Hence:

$$q_1 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{l_2 \sin(q_2)}{l_1 + l_2 \cos(q_2)}\right)$$



Using tangent rule:

$$\tan(q_1 + \beta) = \frac{y}{x}$$

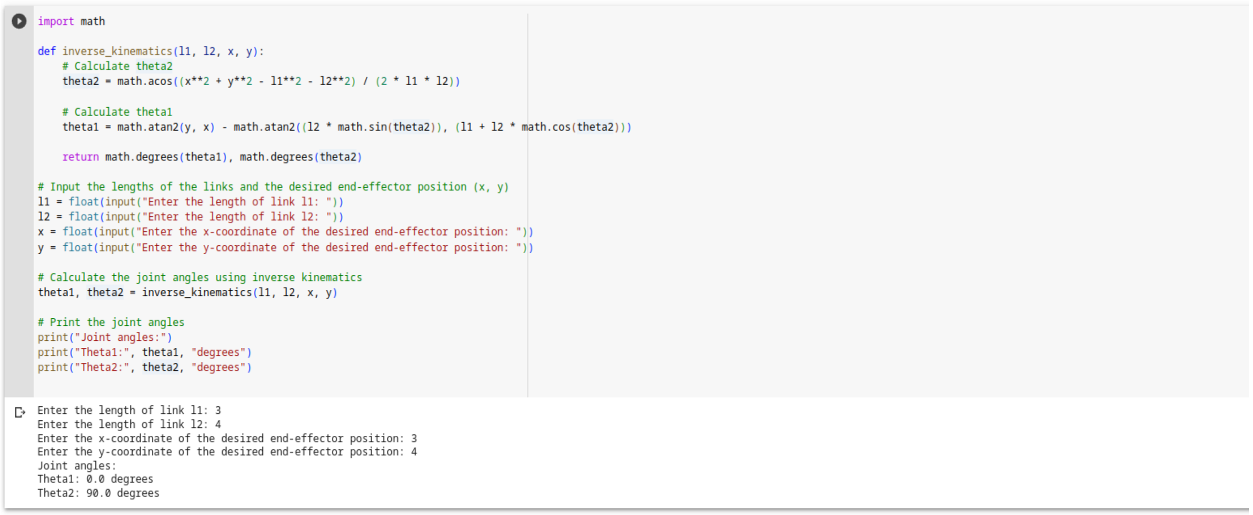
$$q_1 = \tan^{-1}\left(\frac{y}{x}\right) - \beta$$

Using Law of Cosines:

$$\beta = \tan^{-1}\left(\frac{l_2 \sin(q_2)}{l_1 + l_2 \cos(q_2)}\right)$$

Hence:

$$q_1 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{l_2 \sin(q_2)}{l_1 + l_2 \cos(q_2)}\right)$$



```python
import math

def inverse_kinematics(l1, l2, x, y):
    # Calculate theta2
    theta2 = math.acos((x**2 + y**2 - l1**2 - l2**2) / (2 * l1 * l2))

    # Calculate theta1
    theta1 = math.atan2(y, x) - math.atan2((l2 * math.sin(theta2)), (l1 + l2 * math.cos(theta2)))

    return math.degrees(theta1), math.degrees(theta2)

# Input the lengths of the links and the desired end-effector position (x, y)
l1 = float(input("Enter the length of link l1: "))
l2 = float(input("Enter the length of link l2: "))
x = float(input("Enter the x-coordinate of the desired end-effector position: "))
y = float(input("Enter the y-coordinate of the desired end-effector position: "))

# Calculate the joint angles using inverse kinematics
theta1, theta2 = inverse_kinematics(l1, l2, x, y)

# Print the joint angles
print("Joint angles:")
print("Theta1:", theta1, "degrees")
print("Theta2:", theta2, "degrees")
```

```
Enter the length of link l1: 3
Enter the length of link l2: 4
Enter the x-coordinate of the desired end-effector position: 3
Enter the y-coordinate of the desired end-effector position: 4
Joint angles:
Theta1: 0.0 degrees
Theta2: 90.0 degrees
```