1. Section A

a.
Sentiment Analysis: Employing sentiment analysis as an NLP technique can enhance the performance of mobile search engines. By examining the sentiment of a user's search query, search engines can deliver more relevant and tailored results that resonate with the user's mood or emotional state. For example, a user feeling anxious might be interested in finding resources on relaxation techniques, meditation, or stress relief.

Personalization: NLP empowers mobile search engines to personalize search results based on individual preferences, search history, and behavior. By analyzing usage patterns and interests, search engines can offer customized search results, recommendations, and suggestions that align with the unique needs of each user. NLP models enable an understanding of user preferences, categorization of interests, and adaptation of search results accordingly.

Voice Search and Conversational Interfaces: With the rise of voice assistants and mobile devices supporting voice input, NLP is crucial in enabling voice search and conversational interfaces. NLP models can convert spoken language into text, understand spoken queries, and provide accurate responses. Voice-based NLP also allows for more natural and interactive conversations with the search engine, enhancing the overall user experience.

Named Entity Recognition (NER): NER can identify and categorise named entities in search queries, such as people, organisations, locations, and products. This information can be used to improve search result relevance and enable more precise filtering and categorisation of search results.

User Profiling and Personalization: NLP can create user profiles based on search history, behaviour, and preferences. By analysing past search queries and interactions, search engines can personalise search results and recommendations to align with individual user interests and needs.

Part-of-Speech Tagging: Part-of-speech tagging assigns grammatical tags to each word in a sentence. This technique helps search engines understand the syntactic structure of search queries and disambiguate word meanings. It enables more accuracy.

b.
i. The robot is a smart room-cleaning robot designed specifically for cleaning tasks within a home environment.

ii. To solve this problem, a combination of passive and active sensors can be used. Here are some examples:

Passive sensors, such as cameras, enable the robot to capture visual information about the environment, detect obstacles, and identify areas that require cleaning. On the other hand, active sensors like ultrasonic and infrared sensors allow the robot to measure distances, detect obstacles in real-time, and assess the presence of dust or dirt on the floors. By utilizing a combination of passive and active sensors, the robot can effectively gather information, make informed decisions, and carry out cleaning tasks efficiently.

iii. Movement effectors, such as wheels or tracks, are crucial for the robot's mobility, enabling it to navigate rooms, avoid obstacles, and access various areas for cleaning purposes. Collecting dust effectors, such as suction mechanisms or brushes, play a vital role in efficiently gathering dust and dirt from floors and surfaces. Sweeping effectors, such as brushes or brooms, effectively loosen debris, facilitating its collection by the robot. Additionally, mopping effectors, such as mopping pads or cleaning mechanisms, aid in dampening and thoroughly cleaning the floors. By utilizing these diverse effectors, the robot can navigate, collect dust, sweep, and mop to achieve comprehensive cleaning outcomes.

iv. Rule-based reasoning: A set of predefined rules can guide the robot's behavior and decision-making process. For example, if the robot encounters an obstacle, it can be programmed to change its direction or find an alternative path.
Environmental mapping: The robot can create a map of the environment, incorporating information from the sensors. This map can help in navigation, obstacle avoidance, and efficient cleaning planning.
Computer vision: The robot can employ computer vision techniques to recognize objects, detect obstacles, and identify areas that require cleaning. It can use image processing algorithms to analyze the visual data captured by the cameras and make informed decisions accordingly.

2. Section B

a.i.

```
import nltk
nltk.download('punkt')

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
[ ] from nltk.tokenize import word_tokenize

    text = input()
    tokens = word_tokenize(text)
    print(tokens)

    This is an example text corpus. It contains several sentences and words that we will process.
    ['This', 'is', 'an', 'example', 'text', 'corpus', '.', 'It', 'contains', 'several', 'sentences', 'and', 'words', 'that', 'we', 'will', 'process', '.']
```

ii.

```
[ ] nltk.download('stopwords')

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

```
[ ] from nltk.corpus import stopwords

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

    ['example', 'text', 'corpus', '.', 'contains', 'several', 'sentences', 'words', 'process', '.']
```

iii.

```
[ ] nltk.download('wordnet')
    nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
True
```

```
[ ] from nltk.stem import WordNetLemmatizer

    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in filtered_tokens]
    print(lemmatized_tokens)
```

```
['example', 'text', 'corpus', '.', 'contains', 'several', 'sentence', 'word', 'process', '.']
```

iv.

```
[ ] from collections import Counter

    filtered_tokens_alpha = [token for token in lemmatized_tokens if token.isalnum()]
    word_count = Counter(filtered_tokens_alpha)
    print(word_count)
```

```
Counter({'example': 1, 'text': 1, 'corpus': 1, 'contains': 1, 'several': 1, 'sentence': 1, 'word': 1, 'process': 1})
```

v.

```
[ ] most_common_words = word_count.most_common(10)
    for word, frequency in most_common_words:
        print(f"{word}: {frequency} ({frequency/len(tokens):.2%} frequency)")
```

```
example: 1 (5.56% frequency)
text: 1 (5.56% frequency)
corpus: 1 (5.56% frequency)
contains: 1 (5.56% frequency)
several: 1 (5.56% frequency)
sentence: 1 (5.56% frequency)
word: 1 (5.56% frequency)
process: 1 (5.56% frequency)
```

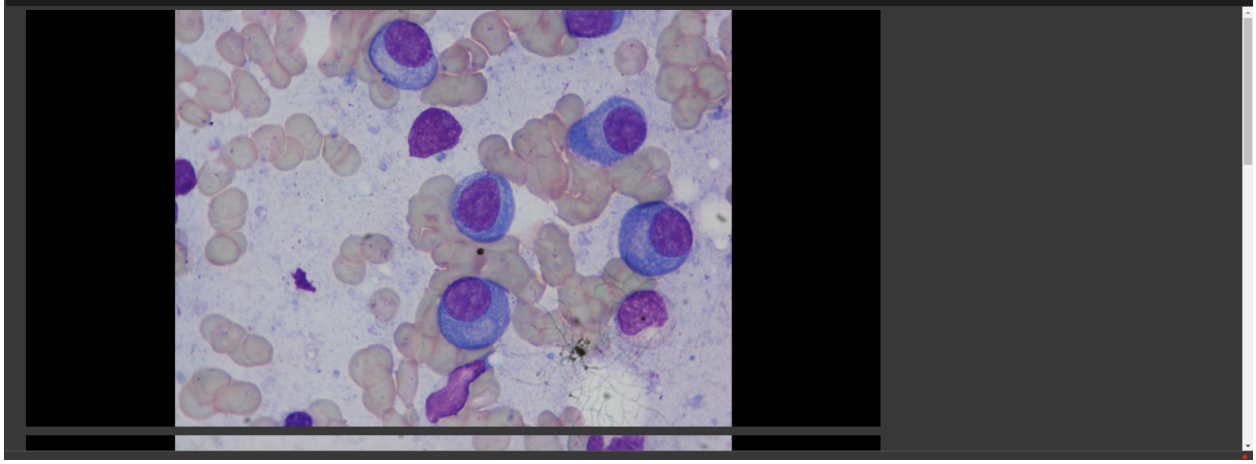
b.

```
b)

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

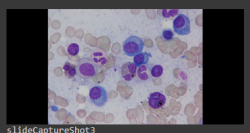
```
Mounted at /content/drive
```

```
[ ] fil = os.listdir('/content/drive/MyDrive/CV')
    image_files = [f for f in fil if f.endswith('.jpg')]
    from PIL import Image,ImageOps,ImageEnhance
    random_images = random.sample(image_files, 3)
    for image_name in random_images:
        image_path = os.path.join('/content/drive/MyDrive/CV/', image_name)
        image = Image.open(image_path)
        image.show()
        print()
```

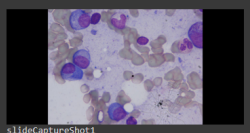


i.

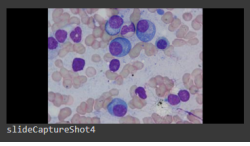
```
random_images = random.sample(image_files, 3)
for image_name in random_images:
  image_path = os.path.join('/content/drive/MyDrive/CV/', image_name)
  image = Image.open(image_path)
  width, height = image.size
  new_width = 350
  new_height = int((height / width) * new_width)
  resized_image = image.resize((new_width,new_height))
  print()
  resized_image.show()
  print(str(image_name)[:17])
```



slideCaptureShot3



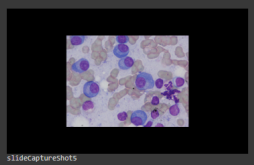
slideCaptureShot1



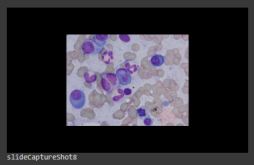
slideCaptureShot4

ii.

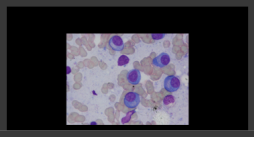
```
random_images = random.sample(image_files, 3)
for image_name in random_images:
  image_path = os.path.join('/content/drive/MyDrive/CV/', image_name)
  image = Image.open(image_path)
  width, height = image.size
  new_width = 350
  new_height = int((height / width) * new_width)
  resized_image = image.resize((new_width,new_height))
  padding = 50  # Adjust the padding size as needed
  padded_image = ImageOps.expand(resized_image, border=padding)
  print()
  padded_image.show()
  print(str(image_name)[:17])
```



slideCaptureShot5

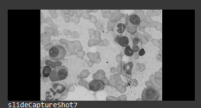


slideCaptureShot6

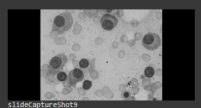


iii.

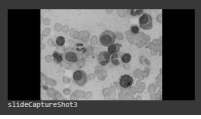
```
[ ] random_images = random.sample(image_files, 3)
    for image_name in random_images:
      image_path = os.path.join('/content/drive/MyDrive/CV/', image_name)
      image = Image.open(image_path)
      width, height = image.size
      new_width = 350
      new_height = int((height / width) * new_width)
      resized_image = image.resize((new_width,new_height))
      grayscale_image = resized_image.convert('L')
      print()
      grayscale_image.show()
      print(str(image_name)[:17])
```



slideCaptureShot7



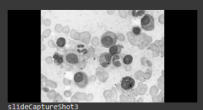
slideCaptureShot9



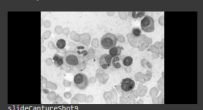
slideCaptureShot3

iv.

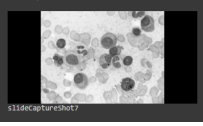
```
[ ] random_images = random.sample(image_files, 3)
    for image_name in random_images:
      image_path = os.path.join('/content/drive/MyDrive/CV/', image_name)
      image = Image.open(image_path)
      width, height = image.size
      new_width = 350
      new_height = int((height / width) * new_width)
      resized_image = image.resize((new_width,new_height))
      contrast_image = ImageOps.autocontrast(grayscale_image)
      print()
      contrast_image.show()
      print(str(image_name)[:17])
```



slideCaptureShot3



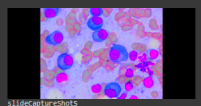
slideCaptureShot9



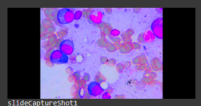
slideCaptureShot7

v.

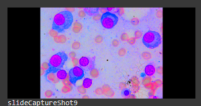
```
[ ] random_images = random.sample(image_files, 3)
    for image_name in random_images:
      image_path = os.path.join('/content/drive/MyDrive/CV/', image_name)
      image = Image.open(image_path)
      width, height = image.size
      new_width = 350
      new_height = int((height / width) * new_width)
      resized_image = image.resize((new_width,new_height))
      saturation_factor = 5  # increase the saturation (greater than 1) or decrease it (less than 1)
      enhancer = ImageEnhance.Color(resized_image)
      saturated_image = enhancer.enhance(saturation_factor)
      print()
      saturated_image.show()
      print(str(image_name)[:17])
```



slideCaptureShot5

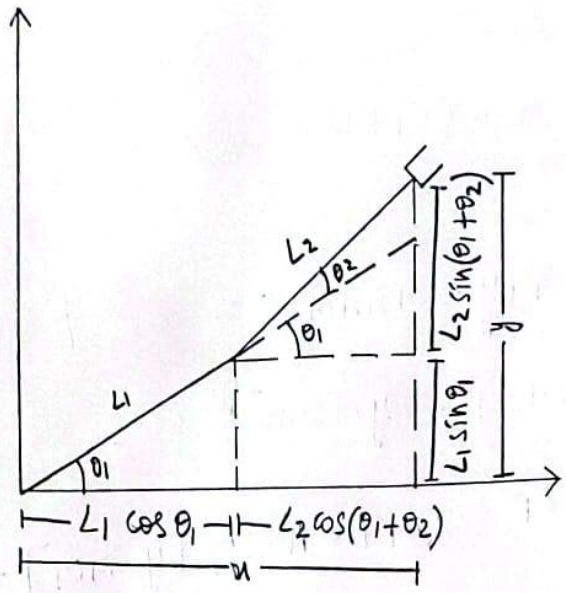


slideCaptureShot1



slideCaptureShot9

C.



$x = L_1\cos\theta_1 + L_2\cos(\theta_1+\theta_2)$ — ①

$y = L_1\sin\theta_1 + L_2\sin(\theta_1+\theta_2)$ — ②

$\rightarrow x - L_2\cos(\theta_1+\theta_2) = L_1\cos\theta_1$ — ③

$y - L_2\sin(\theta_1+\theta_2) = L_1\sin\theta_1$ — ④

By doing ③² + ④²,

$\rightarrow [x - L_2\cos(\theta_1+\theta_2)]^2 + [y - L_2\sin(\theta_1+\theta_2)]^2 = L_1^2$

$[\sin^2\theta + \cos^2\theta = 1]$

$\rightarrow x^2 - 2L_2\cos(\theta_1+\theta_2) + L_2^2 + y^2 - 2L_2\sin(\theta_1+\theta_2) = L_1^2$

$[\sin^2\theta + \cos^2\theta = 1]$

$\rightarrow x^2 + y^2 + L_2^2 - L_1^2 - 2L_2\cos(\theta_1+\theta_2) - 2L_2\sin(\theta_1+\theta_2) = 0$

$\rightarrow x - L_1\cos\theta_1 = L_2\cos(\theta_1+\theta_2)$ — ③

$y - L_1\sin\theta_1 = L_2\sin(\theta_1+\theta_2)$ — ④

③^2 + ④,

$\rightarrow (x - L_1\cos\theta_1)^2 + (y - L_1\sin\theta_1)^2 = L_2^2$ [$\sin^2\theta + \cos^2\theta = 1$]

$\rightarrow x^2 - 2xL_1\cos\theta_1 + L_1^2 - 2yL_1\sin\theta + y^2 = L_2^2$

$\rightarrow x^2 + y^2 + L_1^2 - 2L_1[x\cos\theta_1 + y\sin\theta_1] = L_2^2$

$\rightarrow x^2 + y^2 + L_1^2 - 2L_1[L_1 + L_2\cos\theta_2] = L_2^2$

$\rightarrow \cos\theta_2 = \dfrac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1L_2}$

$\rightarrow \theta_2 = \cos^{-1}\left(\dfrac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1L_2}\right)$

$\theta_1 = \tan^{-1}\left(\dfrac{y}{x}\right) = \tan^{-1}\left(\dfrac{L_2\sin\theta_2}{L_1 + L_2\cos\theta_2}\right)$

Hence, $\theta_1 = \tan^{-1}\left(\dfrac{y}{x}\right) - \tan^{-1}\left(\dfrac{L_2\sin\theta_2}{L_1 + L_2\cos\theta_1}\right)$

$\theta_2 = \cos^{-1}\left(\dfrac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1L_2}\right)$

c)

```
[3]  import math

     # Define the end-effector position and link lengths
     x = float(input('Enter the end effector value of x : '))
     y = float(input('Enter the end effector value of y : '))
     l1 = float(input('Enter the lenght L1 : '))
     l2 = float(input('Enter the lenght L2 : '))

     # Calculate theta2
     cos_theta2 = (x**2 + y**2 - l1**2 - l2**2) / (2 * l1 * l2)
     theta2 = math.acos(cos_theta2)

     # Calculate theta1
     k1 = l1 + l2 * math.cos(theta2)
     k2 = l2 * math.sin(theta2)
     theta1 = math.atan2(y, x) - math.atan2(k2, k1)

     # Print the joint angles (θ1, θ2) in radians
     print("Joint angles (θ1, θ2) in radians:", theta1, theta2)
```

```
Enter the end effector value of x : 2.5
Enter the end effector value of y : 1.5
Enter the lenght L1 : 3
Enter the lenght L2 : 2
Joint angles (θ1, θ2) in radians: -0.1487983708856151 1.955193101290535
```