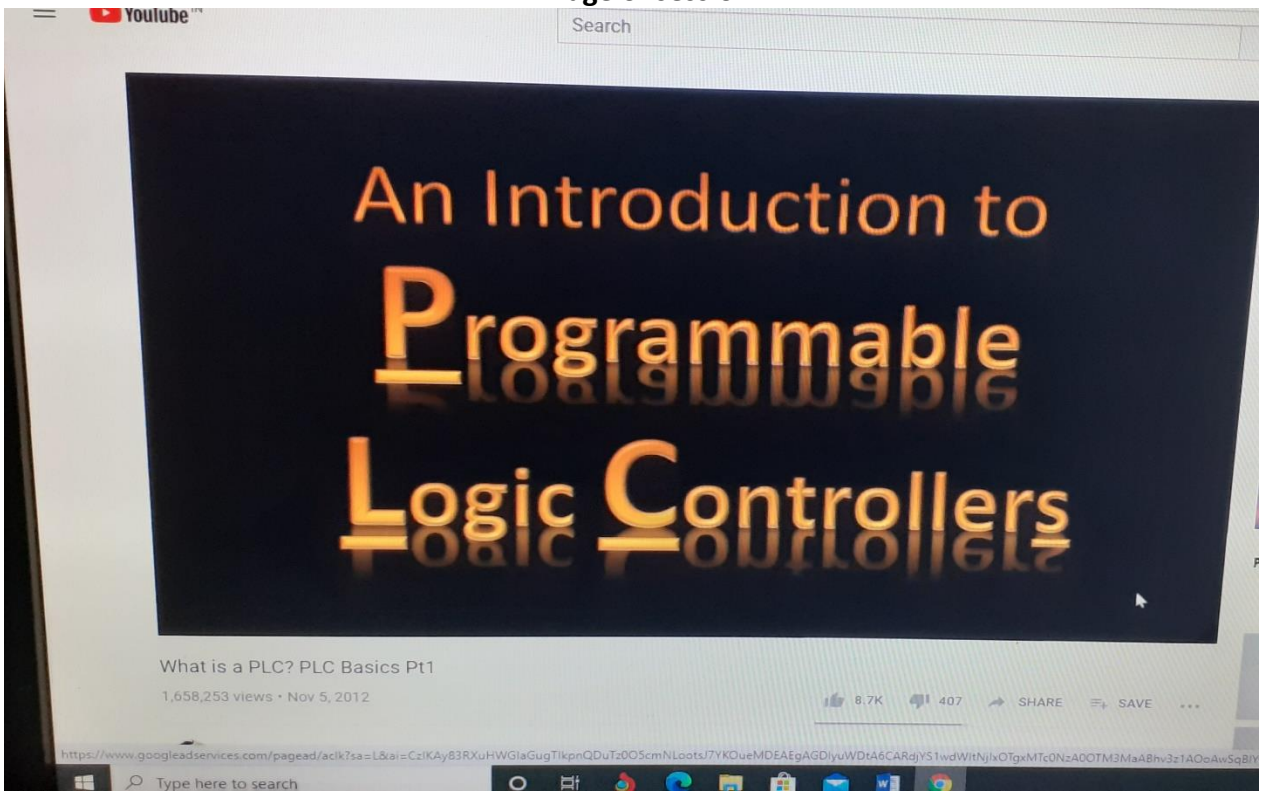
# DAILY ASSESSMENT

| | | | |
|---|---|---|---|
| **Date:** | **30-5-2020** | **Name:** | **Kavyashree m** |
| **Course:** | **Logic design** | **USN:** | **4al15ec036** |
| **Topic:** | **Applications of Programmable logic controllers** | **Semester & Section:** | **8th A** |
| **Github Repository:** | **kavya** | | |

**FORENOON SESSION DETAILS**

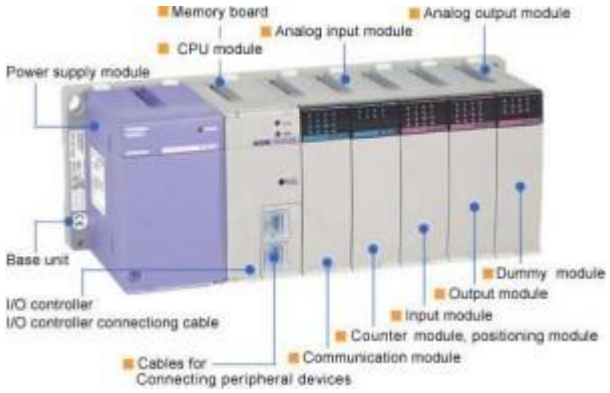
**Image of session**




**Programmable logic controllers**

Programmable Logic Controller (PLC) is a special computer device used in industrial control systems. Due to its robust construction, exceptional functional features like sequential control, counters and timers, ease of programming, reliable controlling capabilities and ease of hardware usage – this PLC is used as more than a special-purpose digital computer in industries as well as in other control-system areas. Most of

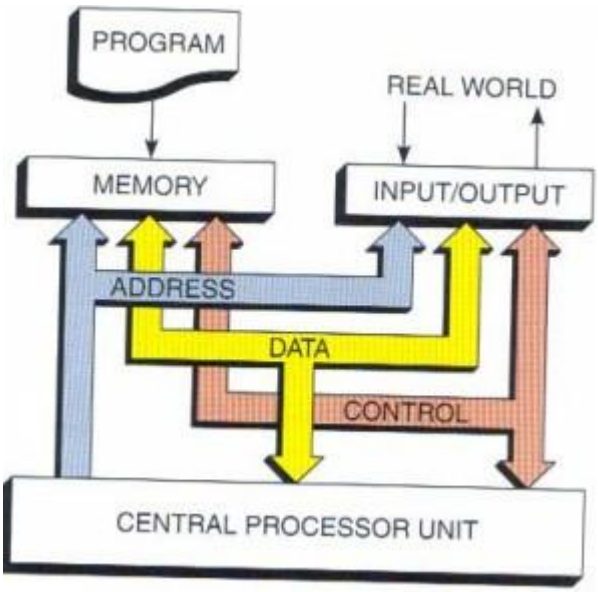
the industries abbreviate these devices as “PC” but it is also used for personal computers; due to this, many manufacturers named these devices as PLCs.




**Principle of Programmable Logic Controller:**

Programmable Logic Controllers are used for continuously monitoring the input values from sensors and produces the outputs for the operation of actuators based on the program. Every PLC system comprises these three modules:

- CPU module
- Power supply module
- One or more I/O module



**CPU Module:**

A CPU module consists of central processor and its memory. The processor is responsible for performing all the necessary computations and processing of data by accepting the inputs and producing the appropriate outputs.

**Power Supply Module:**

This module supplies the required power to the whole system by converting the available AC power to DC power required for the CPU and I/O modules. The 5V DC output drives the computer circuitry**.**

**I/O Modules:**

The input and out modules of the programmable logic controller are used to connect the sensors and actuators to the system to sense the various parameters such as temperature, pressure and flow, etc. These I/O modules are of two types: digital or analog.

**Communication Interface Modules:**

These are intelligent I/O modules which transfers the information between a CPU and communication network. These communication modules are used for communicating with other PLC's and computers, which are placed at remote place or far-off locate.

The program in the CPU of programmable logic controller consists of operating system and user programs. The purpose of the operating system with CPU is to deal with the tasks and operations of the PLC such as starting and stopping operations, storage area and communication management, etc. A user program is used by the user for finishing and controlling the tasks in automation.

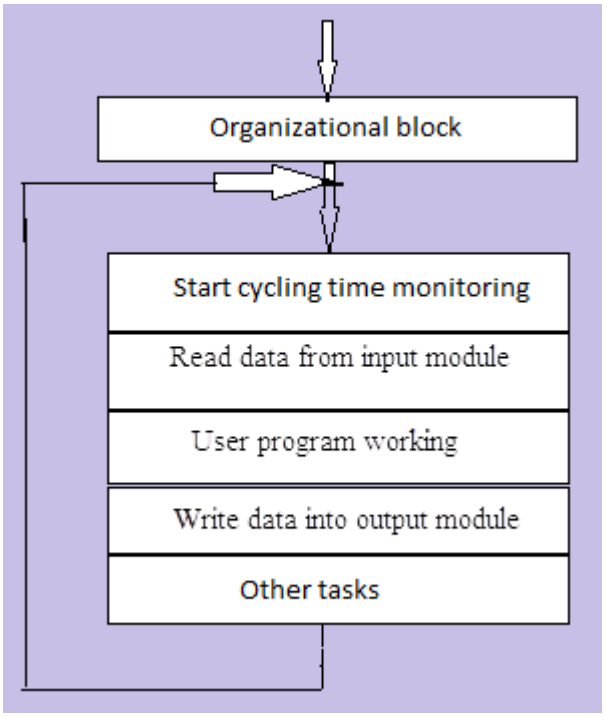
**Programmable Logic Controller**

Programmable Logic Controller (PLC) is a special computer device used in industrial control systems. Due to its robust construction, exceptional functional features like sequential control, counters and timers, ease of programming, reliable controlling capabilities and ease of hardware usage – this PLC is used as more than a special-purpose digital computer in industries as well as in other control-system areas. Most of

the industries abbreviate these devices as “PC” but it is also used for personal computers; due to this, many manufacturers named these devices as PLCs.

The programmable logic controller is used not only for industrial purpose but also in civil applications such as washing machine, elevators working and traffic signals control. Different types of PLCs from a vast number of manufacturers are available in today’s market. Therefore, in the following paragraphs, let us study about programmable logic controller’s basics, principles and applications.

The Principle of operation of the PLC can be understood with the cyclic scanning also called as scan cycle, which is given in the below figure.



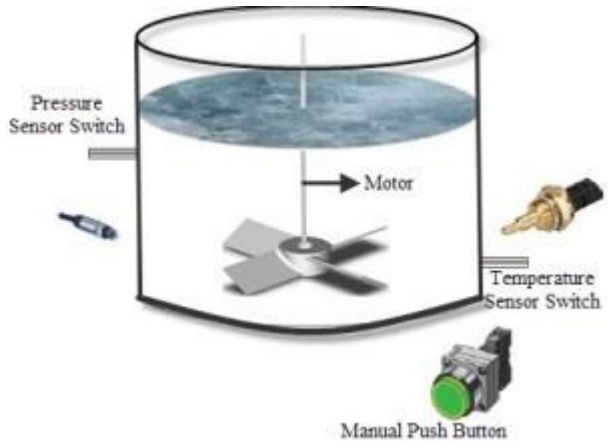

**PLC Scan Cycle**

A typical PLC scans cycle includes of the following steps:

- The operating system starts cycling and monitoring of time.
- The CPU starts reading the data from the input module and checks the status of all the inputs.
- The CPU starts executing the user or application program written in relay-ladder logic or any other PLC-programming language.

- Next, the CPU performs all the internal diagnosis and communication tasks.
- According to the program results, it writes the data into the output module so that all outputs are updated.
- This process continues as long as the PLC is in run mode.

To get an idea about the PLC operation,consider the simple mixer process control as shown in the figure below.




**Simple Mixer Process Control**

In the above figure, the set up has a mixer motor to stir liquid automatically in the container whenever the temperature and pressure reaches the preset values. In addition, a separate manual push button station is used for the operation of the motor.The process is monitored with a pressure-sensor switch and temperature-sensor switch. These switches close their respective contacts when conditions reach their preset values.

**Applications of Programmable Logic Controller (PLC)**

**1. Application of PLC in Glass Industry**

From the year 1980 the Programmable-logic controllers are in use in the glass industry, and they are assembled bit by bit. PLCs are used mainly in every procedure and workshop for controlling the material ratio, processing of flat glasses, etc.

With the development of PLC and increasing demand in the real world, the control mode of the programmable-logic controller with an intelligent device is applied in the glass

industry. In making of a float glass, PLC itself cannot finish some controlling tasks because of the complexity of the control system and processing of huge data. For the production of glass, we make use of bus technology to construct the control mode of a PLC with a distributed-control system. This control system deals with analog controlling and data recording; the PLC is also used for digital quality control and position control. This type of control mode is a big advantage for PLC and DCS for improving reliability and flexibility of the control system.

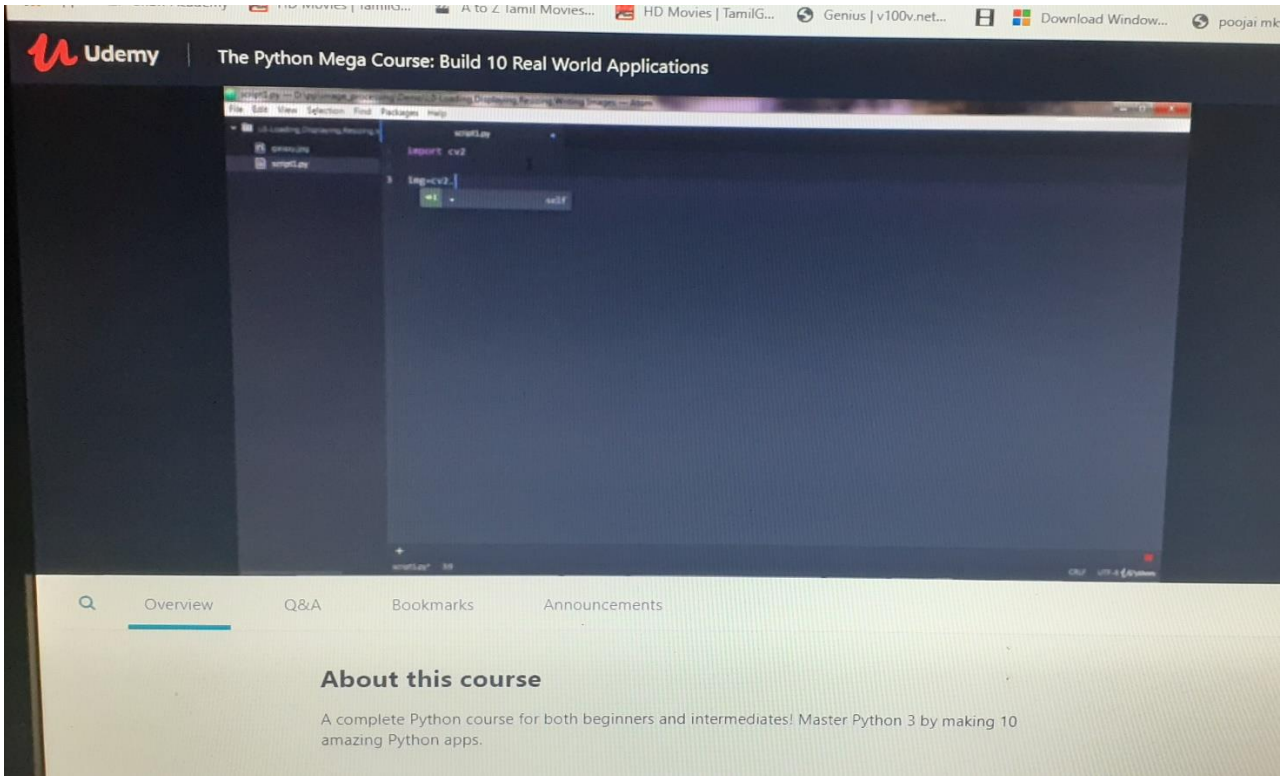
**2. Applications of PLC in Cement Industry**

Along with the best-quality raw materials, the accurate data regarding process variables, especially during mixing processes within the kiln, ensures that the output provided should be of the best possible quality. Nowadays a DCS with bus technology is used in the production and management industry. By using this existing DCS control system, the PLC is in user mode of SCADA. This mode comprises PLC and configuration software. This SCADA mode comprises the PLC and host computer. The host computer consists of slave and master station. The PLC is used for controlling the ball milling, shaft kiln and Kiln of coal.

**AFTERNOON SESSION DETAILS**

| | | | |
|---|---|---|---|
| **Date:** | **30-5-2020** | **Name:** | **Kavyashree m** |
| **Course:** | **Python programming** | **USN:** | **4al15ec036** |
| **Topic:** | **Python for Image and Video processing with Open CV** | **Semester & Section:** | **8th A** |
| **Github Repository:** | **kavya** | | |

**Image of session**




# Python for Image and Video processing with Open CV

Processing a video means, performing operations on the video frame by frame. Frames are nothing but just the particular instance of the video in a single point of time. We may

have multiple frames even in a single second. Frames can be treated as similar to an image.

So, whatever operations we can perform on images can be performed on frames as well. Let us see some of the operations with examples.

**Adaptive Threshold**

By using this technique we can apply thresholding on small regions of the frame. So the collective value will be different for the whole frame.

```
# importing the necessary libraries
import cv2
import numpy as np

# Creating a VideoCapture object to read the video
cap = cv2.VideoCapture('sample.mp4')



# Loop untill the end of the video
while (cap.isOpened()):

   # Capture frame-by-frame
   ret, frame = cap.read()
   frame = cv2.resize(frame, (540, 380), fx = 0, fy = 0,
                      interpolation = cv2.INTER_CUBIC)

   # Display the resulting frame
   cv2.imshow('Frame', frame)
```

```
    # conversion of BGR to grayscale is necessary to apply this operation
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # adaptive thresholding to use different threshold
    # values on different regions of the frame.
    Thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
                                   cv2.THRESH_BINARY_INV, 11, 2)

    cv2.imshow('Thresh', Thresh)
    # define q as the exit button
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

# release the video capture object
cap.release()
# Closes all the windows currently opened.
cv2.destroyAllWindows()
```

**Smoothing**

Smoothing a video means removing the sharpness of the video and providing a blurriness to the video. There are various methods for smoothing such as cv2.Gaussianblur(), cv2.medianBlur(), cv2.bilateralFilter(). For our purpose, we are going to use cv2.Gaussianblur().

filter_none

brightness_4

```
# importing the necessary libraries
import cv2
```

```
import numpy as np

# Creating a VideoCapture object to read the video
cap = cv2.VideoCapture('sample.mp4')



# Loop untill the end of the video
while (cap.isOpened()):
   # Capture frame-by-frame
   ret, frame = cap.read()
   frame = cv2.resize(frame, (540, 380), fx = 0, fy = 0,
                      interpolation = cv2.INTER_CUBIC)

   # Display the resulting frame
   cv2.imshow('Frame', frame)

   # using cv2.Gaussianblur() method to blur the video

   # (5, 5) is the kernel size for blurring.
   gaussianblur = cv2.GaussianBlur(frame, (5, 5), 0)
   cv2.imshow('gblur', gaussianblur)

   # define q as the exit button
   if cv2.waitKey(25) & 0xFF == ord('q'):
      break

# release the video capture object
```

```
cap.release()

# Closes all the windows currently opened.
cv2.destroyAllWindows()
```

**Edge Detection**

Edge detection is a useful technique to detect he edges of surfaces and objects in the video. Edge detection involves the following steps:

- Noise reduction
- Gradient calculation
- Non-maximum suppression
- Double threshold
- Edge tracking by hysteresis

```
filter_none
brightness_4
# importing the necessary libraries
import cv2
import numpy as np

# Creating a VideoCapture object to read the video
cap = cv2.VideoCapture('sample.mp4')


# Loop untill the end of the video
while (cap.isOpened()):
   # Capture frame-by-frame
   ret, frame = cap.read()
```

```
    frame = cv2.resize(frame, (540, 380), fx = 0, fy = 0,
                       interpolation = cv2.INTER_CUBIC)

    # Display the resulting frame
    cv2.imshow('Frame', frame)

    # using cv2.Canny() for edge detection.
    edge_detect = cv2.Canny(frame, 100, 200)
    cv2.imshow('Edge detect', edge_detect)

    # define q as the exit button
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

# release the video capture object
cap.release()
# Closes all the windows currently opened.
cv2.destroyAllWindows()
```

**Bitwise Operations**

Bitwise operations are useful to mask different frames of a video together. Bitwise operations are just like we have studied in the classroom such as AND, OR, NOT, XOR.

filter_none

brightness_4

```
# importing the necessary libraries
import cv2
```

```
import numpy as np

# Creating a VideoCapture object to read the video
cap = cv2.VideoCapture('sample.mp4')



# Loop untill the end of the video
while (cap.isOpened()):
    # Capture frame-by-frame
    ret, frame = cap.read()
    frame = cv2.resize(frame, (540, 380), fx = 0, fy = 0,
                       interpolation = cv2.INTER_CUBIC)

    # Display the resulting frame
    cv2.imshow('Frame', frame)

    # conversion of BGR to grayscale is necessary to apply this operation
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    _, mask = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

    # apply NOT operation on image and mask generated by thresholding
    BIT = cv2.bitwise_not(frame, frame, mask = mask)
    cv2.imshow('BIT', BIT)

    # define q as the exit button
    if cv2.waitKey(25) & 0xFF == ord('q'):
```

```
    break

# release the video capture object
cap.release()

# Closes all the windows currently opened.
cv2.destroyAllWindows()
```

## Batch image resize

### Introduction

The following functions are supported:

- resize_crop crop the image with a centered rectangle of the specified size.
- resize_cover resize the image to fill the specified area, crop as needed (same behavior as background-size: cover).
- resize_contain resize the image so that it can fit in the specified area, keeping the ratio and without crop (same behavior as background-size: contain).
- resize_height resize the image to the specified height adjusting width to keep the ratio the same.
- resize_width resize the image to the specified width adjusting height to keep the ratio the same.
- resize_thumbnail resize image while keeping the ratio trying its best to match the specified size.

### Installation

Install python-resize-image using pip:

```
pip install python-resize-image
```

**Usage**

python-resize-image takes as first argument a PIL.Image and then size argument which can be a single integer or tuple of two integers.

In the following example, we open an image, crop it and save as new file:

```
from PIL import Image

from resizeimage import resizeimage

with open('test-image.jpeg', 'r+b') as f:
    with Image.open(f) as image:
        cover = resizeimage.resize_cover(image, [200, 100])
        cover.save('test-image-cover.jpeg', image.format)
```

Before resizing, python-image-resize will check whether the operation can be done. A resize is considered valid if it doesn't require to increase one of the dimension. To avoid the test add validate=False as argument:

```
cover = resizeimage.resize_cover(image, [200, 100], validate=False)
```

You can also create a two step process validation then processing using validate function attached to resized function which allows to test the viability of the resize without doing it just after validation. validate is available using the dot . operator on every resize function e.g. resize_cover.validate.

**API Reference**

resize_crop(image, size, validate=True)

Crop the image with a centered rectangle of the specified size.

Crop an image with a 200x200 cented square:

```
from PIL import Image
from resizeimage import resizeimage

fd_img = open('test-image.jpeg', 'r')
img = Image.open(fd_img)
img = resizeimage.resize_crop(img, [200, 200])
img.save('test-image-crop.jpeg', img.format)
fd_img.close()
```

resize_cover(image, size, validate=True, resample=Image.LANCZOS)

Resize the image to fill the specified area, crop as needed. It's the same behavior as css background-size: cover property.

**Face detection**

In today's blog post you are going to learn how to perform face recognition in both images and video streams using:

- OpenCV
- Python
- Deep learning

**Install your face recognition libraries**

In order to perform face recognition with Python and OpenCV we need to install two additional libraries:

- dlib
- face_recognition