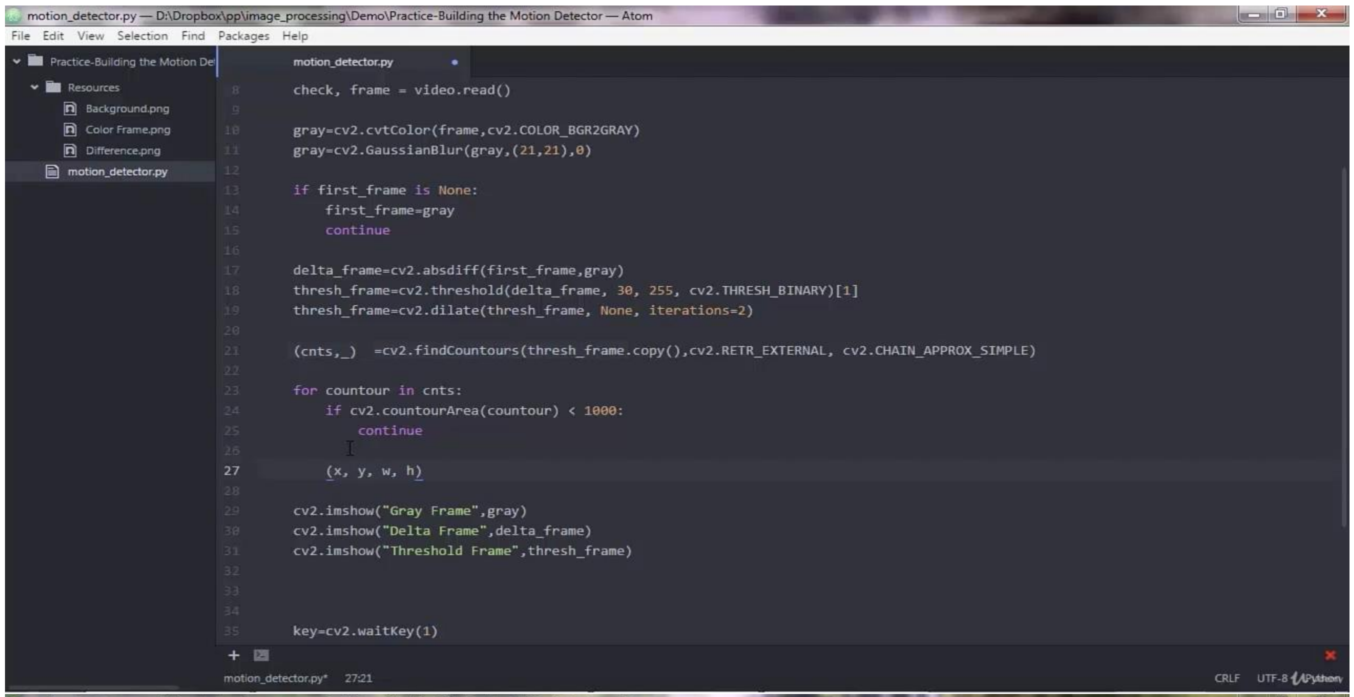
DAILY ASSESSMENT FORMAT

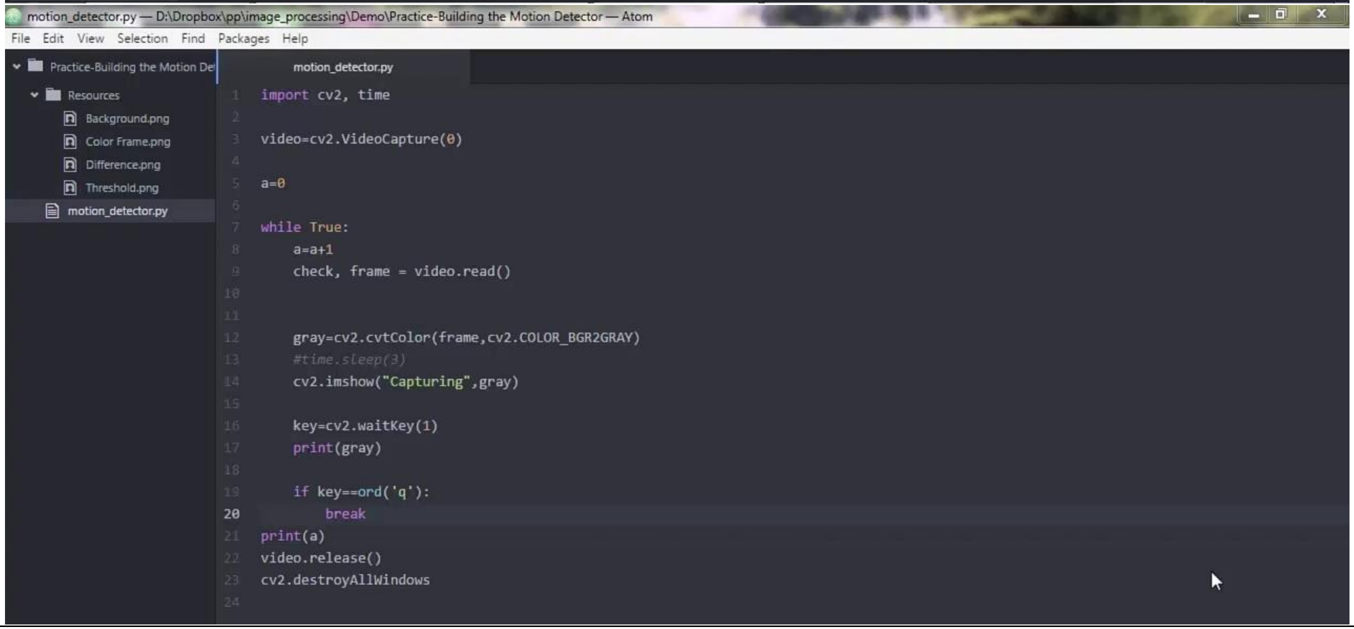
Date:	30th May 2020	**Name:**	Soundarya NA
Course:	UDEMY	**USN:**	4AL16EC077
Topic:	PYTHON: Application 6: Build a webcam motion detector	**Semester & Section:**	8th - B

FORENOON SESSION DETAILS

Image of session


motion_detector.py — D:\Dropbox\pp\image_processing\Demo\Practice-Building the Motion Detector — Atom
File Edit View Selection Find Packages Help
Practice-Building the Motion De
Resources
Background.png
Color Frame.png
Difference.png
motion_detector.py
motion_detector.py* 27:21
CRLF UTF-8 Python

```
    check, frame = video.read()

    gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    gray=cv2.GaussianBlur(gray,(21,21),0)

    if first_frame is None:
        first_frame=gray
        continue

    delta_frame=cv2.absdiff(first_frame,gray)
    thresh_frame=cv2.threshold(delta_frame, 30, 255, cv2.THRESH_BINARY)[1]
    thresh_frame=cv2.dilate(thresh_frame, None, iterations=2)

    (cnts,_)  =cv2.findCountours(thresh_frame.copy(),cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    for countour in cnts:
        if cv2.countourArea(countour) < 1000:
            continue

        (x, y, w, h)

    cv2.imshow("Gray Frame",gray)
    cv2.imshow("Delta Frame",delta_frame)
    cv2.imshow("Threshold Frame",thresh_frame)



    key=cv2.waitKey(1)
```


motion_detector.py — D:\Dropbox\pp\image_processing\Demo\Practice-Building the Motion Detector — Atom
File Edit View Selection Find Packages Help
Practice-Building the Motion De
Resources
Background.png
Color Frame.png
Difference.png
Threshold.png
motion_detector.py

```
import cv2, time

video=cv2.VideoCapture(0)

a=0

while True:
    a=a+1
    check, frame = video.read()


    gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    #time.sleep(3)
    cv2.imshow("Capturing",gray)

    key=cv2.waitKey(1)
    print(gray)

    if key==ord('q'):
        break
print(a)
video.release()
cv2.destroyAllWindows
```

Report:

This python program will allow you to detect motion and also store the time interval of the motion.

Requirement:

- Python3
- OpenCV(libraries)
- Pandas(libraries)

Code:

```
# Python program to implement

# Webcam Motion Detector

# importing OpenCV, time and Pandas library

import cv2, time, pandas

# importing datetime class from datetime library

from datetime import datetime

# Assigning our static_back to None

static_back = None

# List when any moving object appear

motion_list = [ None, None ]

# Time of movement

time = []

# Initializing DataFrame, one column is start

# time and other column is end time

df = pandas.DataFrame(columns = ["Start", "End"])

# Capturing video

video = cv2.VideoCapture(0)

# Infinite while loop to treat stack of image as video

while True:

        # Reading frame(image) from video

        check, frame = video.read()

        # Initializing motion = 0(no motion)
```

```
motion = 0
# Converting color image to gray_scale image
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# Converting gray scale image to GaussianBlur
# so that change can be find easily
gray = cv2.GaussianBlur(gray, (21, 21), 0)
# In first iteration we assign the value
# of static_back to our first frame
if static_back is None:
        static_back = gray
        continue
# Difference between static background
# and current frame(which is GaussianBlur)
diff_frame = cv2.absdiff(static_back, gray)
# If change in between static background and
# current frame is greater than 30 it will show white color(255)
thresh_frame = cv2.threshold(diff_frame, 30, 255, cv2.THRESH_BINARY)[1]
thresh_frame = cv2.dilate(thresh_frame, None, iterations = 2)
# Finding contour of moving object
cnts,_ = cv2.findContours(thresh_frame.copy(),
                                cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
for contour in cnts:
        if cv2.contourArea(contour) < 10000:
                continue
        motion = 1
        (x, y, w, h) = cv2.boundingRect(contour)
        # making green rectangle arround the moving object
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)
# Appending status of motion
motion_list.append(motion)
```

```
        motion_list = motion_list[-2:]
        # Appending Start time of motion
        if motion_list[-1] == 1 and motion_list[-2] == 0:
                time.append(datetime.now())
        # Appending End time of motion
        if motion_list[-1] == 0 and motion_list[-2] == 1:
                time.append(datetime.now())
        # Displaying image in gray_scale
        cv2.imshow("Gray Frame", gray)
        # Displaying the difference in currentframe to
        # the staticframe(very first_frame)
        cv2.imshow("Difference Frame", diff_frame)
        # Displaying the black and white image in which if
        # intensity difference greater than 30 it will appear white
        cv2.imshow("Threshold Frame", thresh_frame)
        # Displaying color frame with contour of motion of object
        cv2.imshow("Color Frame", frame)
        key = cv2.waitKey(1)
        # if q entered whole process will stop
        if key == ord('q'):
                # if something is movingthen it append the end time of movement
                if motion == 1:
                        time.append(datetime.now())
                break
# Appending time of motion in DataFrame
for i in range(0, len(time), 2):
        df = df.append({"Start":time[i], "End":time[i + 1]}, ignore_index = True)
# Creating a CSV file in which time of movements will be saved
df.to_csv("Time_of_movements.csv")
video.release()
```

```
# Destroying all the windows
cv2.destroyAllWindows()
```

Analysis of all windows:

After running the code there 4 new windows will appear on screen.

Gray Frame: In Gray frame the image is a bit blur and in grayscale we did so because, In gray pictures there is only one intensity value whereas in RGB (Red, Green and Blue) image there are three intensity values. So, it would be easy to calculate the intensity difference in grayscale.

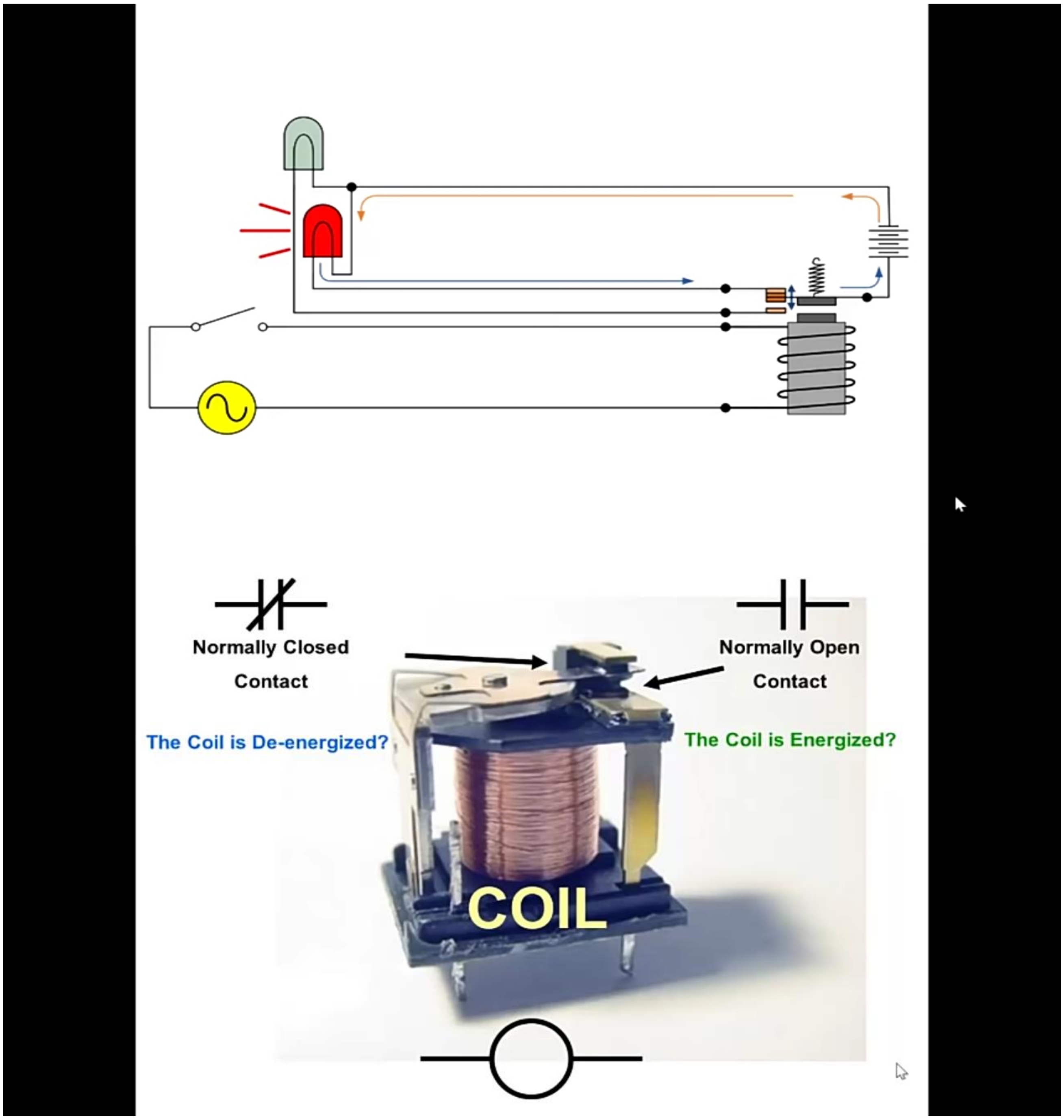
Difference Frame: Difference frame shows the difference of intensities of first frame to the current frame.

Threshold Frame: If the intensity difference for a particular pixel is more than 30(in my case) then that pixel will be white and if the difference is less than 30 that pixel will be black.

Color Frame: In this frame you can see the color images in color frame along with green contour around the moving objects.

Date:	30th May 2020	**Name:**	Soundarya NA
Course:	Logic Design	**USN:**	4AL16EC077
Topic:	Logic Design Day 3	**Semester & Section:**	8th - B

Image:



Report:

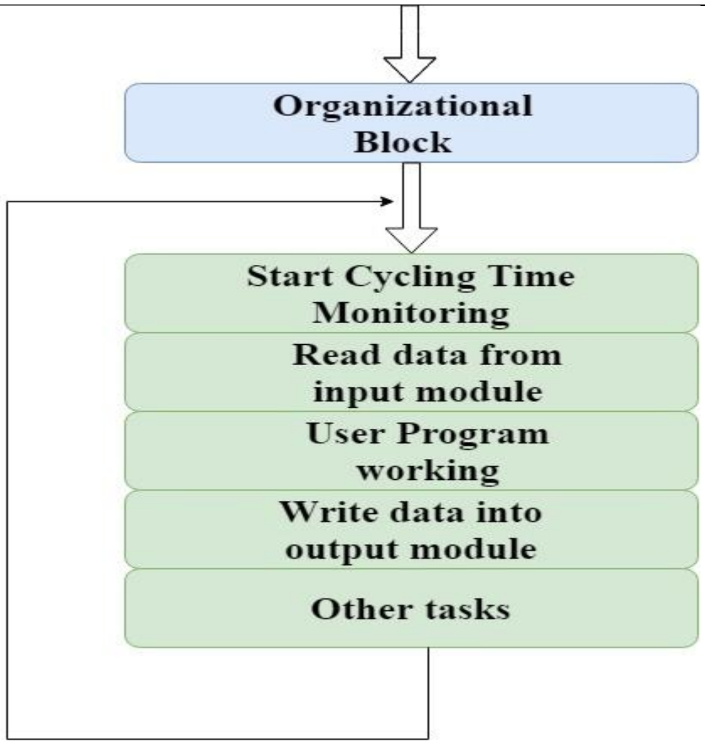
PLC stands for “Programmable Logic Controller”. A PLC is a computer specially designed to operate reliably under harsh industrial environments – such as extreme temperatures, wet, dry, and/or dusty conditions. It is used to automate industrial processes such as a manufacturing plant’s assembly line, an ore processing plant, or a wastewater treatment plant.

PLCs share many features of the personal computer you have at home. They both have a power supply, a CPU (Central Processing Unit), inputs and outputs (I/O), memory, and operating software (although it’s a different operating software). The biggest differences are that a PLC can perform discrete and continuous functions that a PC cannot do, and a PLC is much better suited to rough industrial environments. A PLC can be thought of as a ‘ruggedized’ digital computer that manages the electromechanical processes of an industrial environment.

PLCs play a crucial role in the field of automation, using forming part of a larger SCADA system. A PLC can be programmed according to the operational requirement of the process. In the manufacturing industry, there will be a need for reprogramming due to the change in the nature of production. To overcome this difficulty, PLC-based control systems were introduced. We’ll first discuss PLC basics before looking at various applications of PLCs.

How does a PLC works?

The working of a PLC can be easily understood as a cyclic scanning method known as the scan cycle.


Organizational Block
Start Cycling Time Monitoring
Read data from input module
User Program working
Write data into output module
Other tasks

Block Diagram of How PLC works

A PLC Scan Process includes the following steps

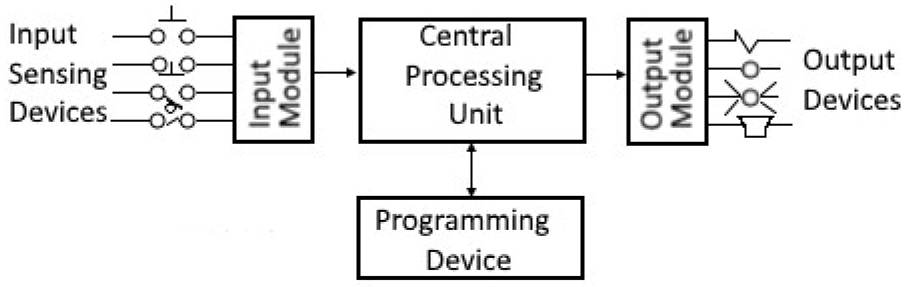
- The operating system starts cycling and monitoring of time
- The CPU starts reading the data from the input module and checks the status of all the inputs
- The CPU starts executing the user or application program written in relay-ladder logic or any other PLC-programming language
- Next, the CPU performs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

Physical Structure of PLC:

The structure of a PLC is almost similar to a computer’s architecture.

Programmable Logic Controllers continuously monitors the input values from various input sensing devices (e.g. accelerometer, weight scale, hardwired signals, etc.) and produces corresponding output depending on the nature of production and industry. A typical block diagram of PLC consists of five parts namely:

- Rack or chassis
- Power Supply Module
- Central Processing Unit (CPU)
- Input & Output Module
- Communication Interface Module



PLC Block Diagram

PLC applications:

PLCs have a variety of applications and uses, including:

- Process Automation Plants (e.g. mining, oil &gas)
- Glass Industry
- Paper Industry
- Cement Manufacturing
- In boilers – Thermal Power Plants

PLC Programming:

When using a PLC, it's important to design and implement concepts depending on your particular use case. To do this we first need to know more about the specifics of PLC programming. A PLC program consists of a set of instructions either in textual or graphical form, which represents the logic that governs the process the PLC is controlling. There are two main classifications of PLC programming languages, which are further divided into many sub-classified types.

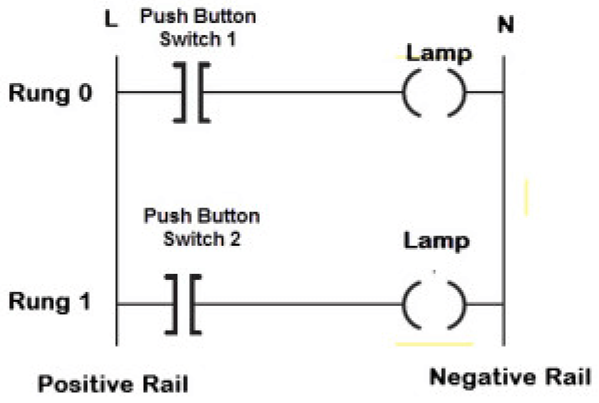
- Textual Language
- Instruction list
- Structured text
- Graphical Form

- Ladder Diagrams (LD) (i.e. Ladder Logic)
- Function Block Diagram (FBD)
- Sequential Function Chart (SFC)

Due to the simple and convenient features, graphical representation is much preferred to textual languages.

Ladder Logic:

Ladder logic is the simplest form of PLC programming. It is also known as “relay logic”. The relay contacts used in relay-controlled systems are represented using ladder logic. The below figure shows the simple example of a ladder diagram.

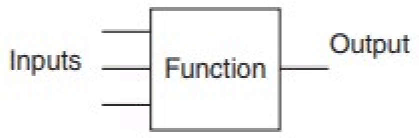


PLC Ladder Logic

In the above-mentioned example, two pushbuttons are used to control the same lamp load. When any one of the switches is closed, the lamp will glow. The two horizontal lines are called rungs and two vertical lines are called rails. Every rung form the electrical connectivity between Positive rail (P) and Negative rail (N). This allows the current to flow between input and output devices.

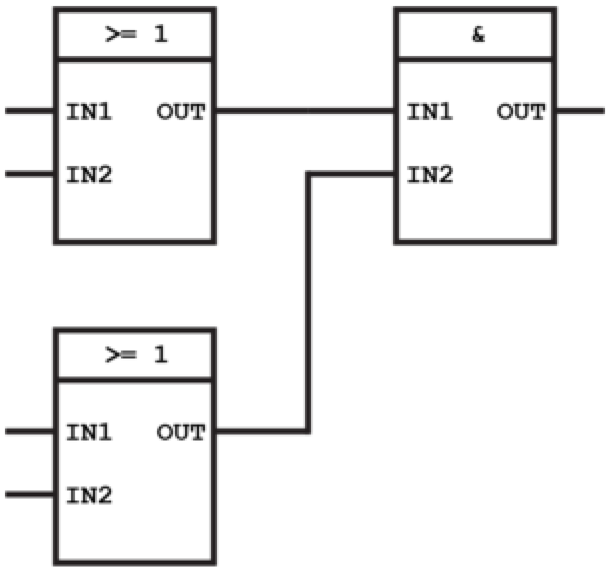
Functional Block Diagrams:

Functional Block Diagram (FBD) is a simple and graphical method to program multiple functions in PLC. PLC Open has described using FBD in the standard IEC 61131-3. A function block is a program instruction unit that, when executed, yields one or more output values. It is represented by a block as shown below. It is represented as a rectangular block with inputs entering on left and output lines leaving at the right. It gives a relation between the state of input and output



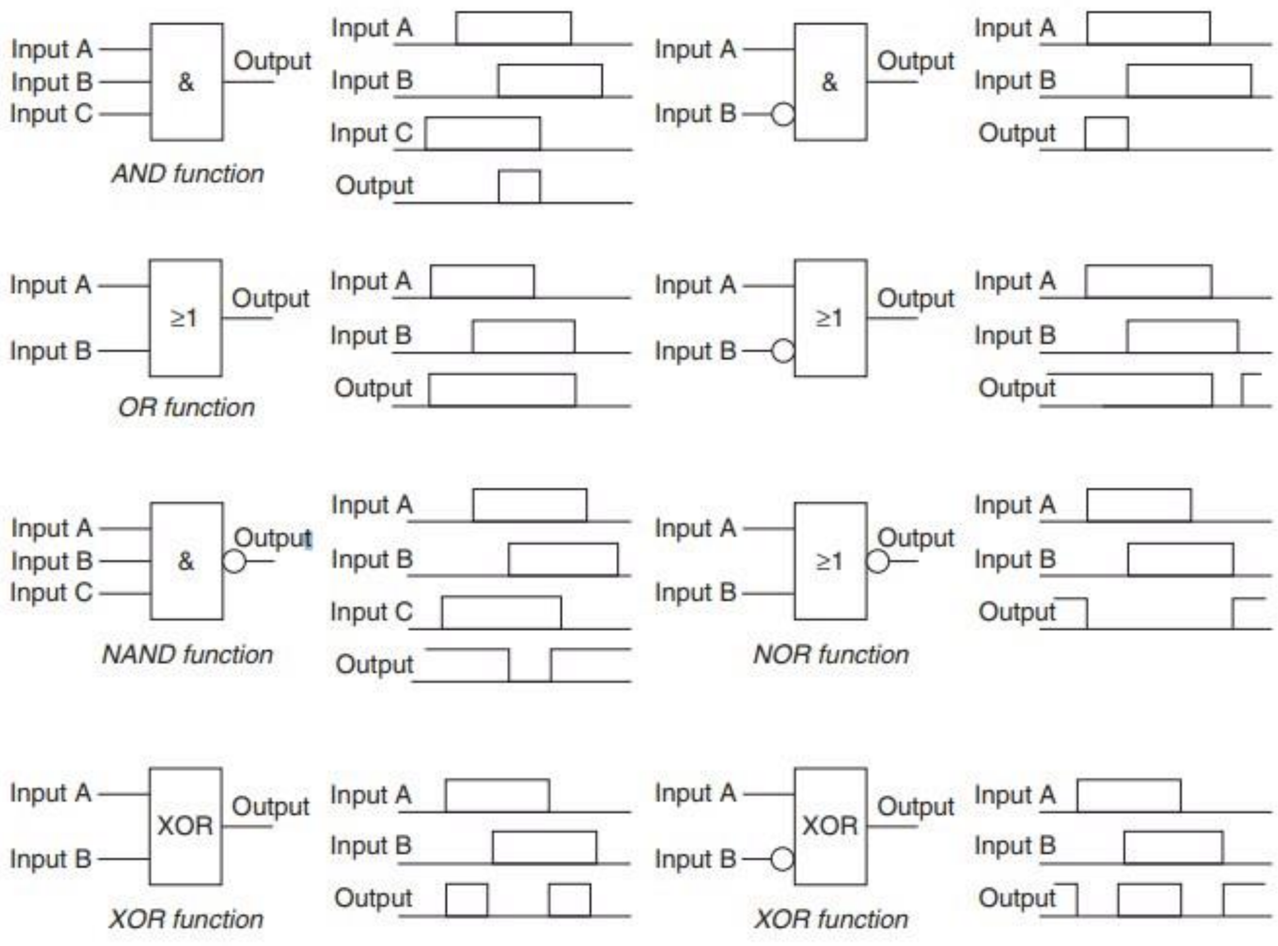
Function Block

The advantage of using FBD is that any number of inputs and outputs can be used on the functional block. When using multiple input and output, you can connect the output of one function block to the input of another. Whereby building a Function Block Diagram.



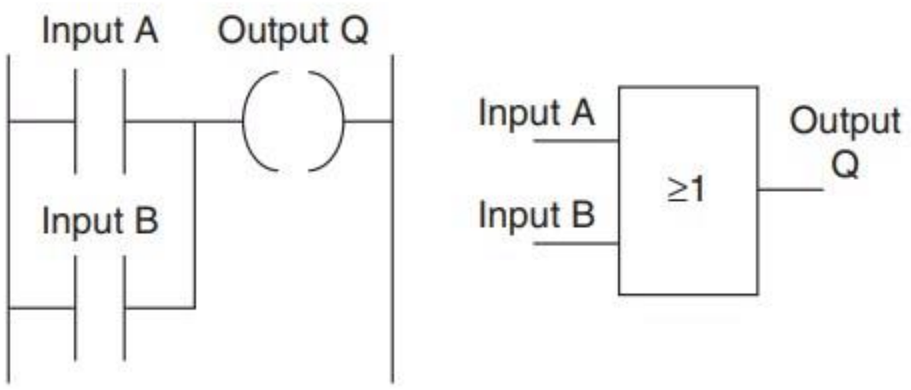
Example Functional Block Diagram

The figure below shows various function blocks used in FBD programming.

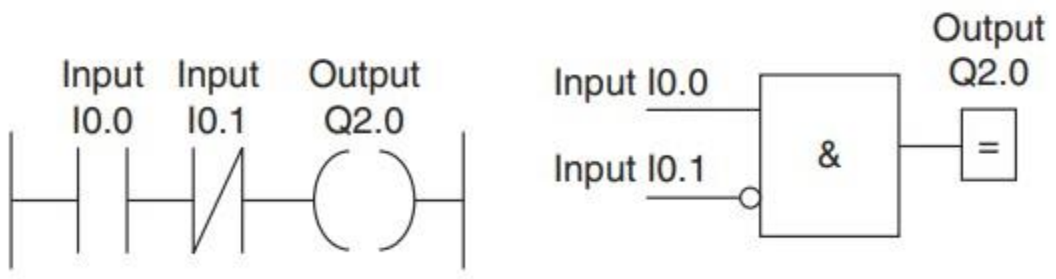


Functional Block Programming

The figure below shows a ladder diagram and its function block equivalent in Siemens notation.



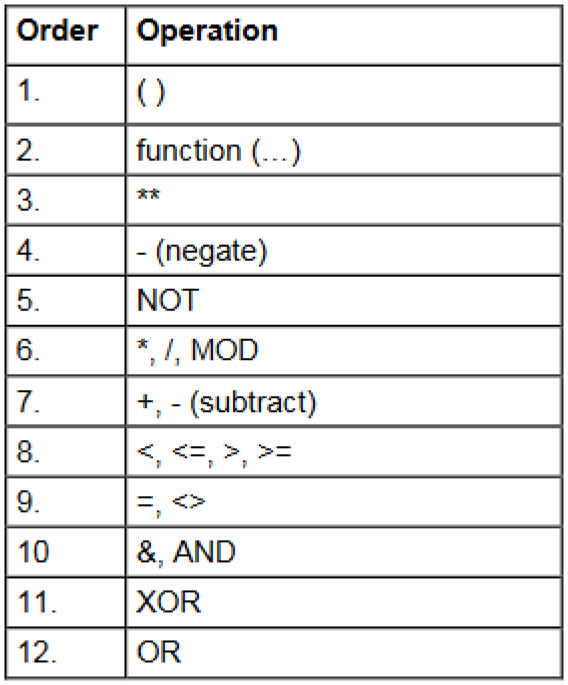
Ladder to functional block



Ladder to functional block diagram

Structured Text Programming:

Structured text is a textual programming language that utilizes statements to determine what to execute. It follows more conventional programming protocols but it is not case sensitive. A series of statements (logic) is constituted of expressing assignments and relationships using several operators. The structures text operators are listed below in the image.



Order	Operation
1.	( )
2.	function (…)
3.	**
4.	- (negate)
5.	NOT
6.	*, /, MOD
7.	+, - (subtract)
8.	<, <=, >, >=
9.	=, <>
10	&, AND
11.	XOR
12.	OR

Structured Text Programming