# Lab Assignment Report

**Name: Debal Ghosh**

**Roll number: 21CS4132**

**Problem Statement:**

(INPUT: 2 program segments with if-then-else but, (i) without loop and (ii) with loop)

- Write a program to store and display the CFG for given program segment.
- Write a program to identify the number of bounded region in a generated CFG.
- Write a Program to identify the maximal set of independent paths in a generated CFG.
- Write a program to create CFG and list out its properties from a given program file of any high level language (input file).

Additional exercise: Write an application with suitable UI to generate the CFG from a given program file. On editing of the program file the CFG should also modify accordingly. The UI should also show the basic characteristics (like, no. of nodes, no. of edges, predicate nodes, no. of bounded region, no. of paths. Etc.) of the CFG.

**Python Libraries Used:**

- matplotlib (for graph plotting)
- networkx (for graph plotting)
- tkinter (for GUI)
- pyglet (for GUI)

**For running code:**

Unzip the assignment.rar file and navigate inside the assignment folder.

If python is already installed run the following commands

- pip install matplotlib
- pip install networkx
- pip install pyglet

Open cmd and enter ‘python main.py’ command for GUI.

Paste code in myTxt.txt and run ‘python myCfg.py’ for command prompt based script.

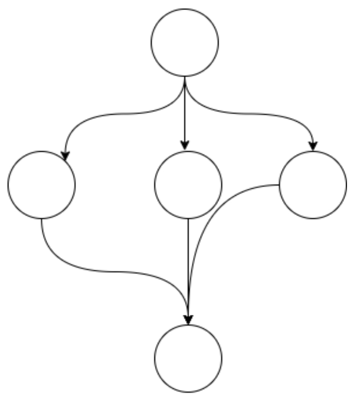
## Brief summary of Control flow graphs for if else and loops

A Control Flow Graph (CFG) is the graphical representation of control flow or computation during the execution of programs or applications. Control flow graphs are mostly used in static analysis as well as compiler applications, as they can accurately represent the flow inside of a program unit. The control flow graph was originally developed by Frances E. Allen.

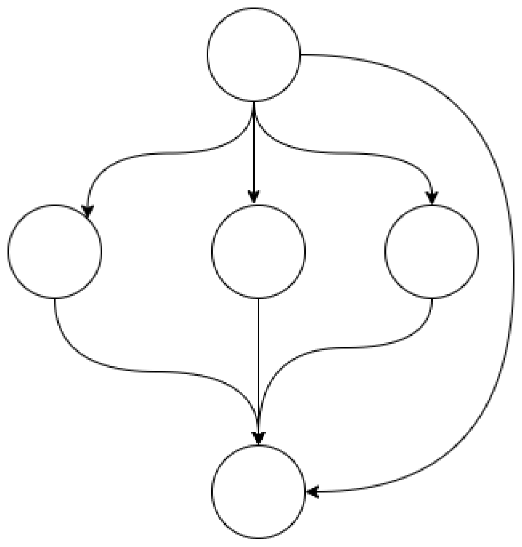
Characteristics of Control Flow Graph:

- Control flow graph is process oriented.
- Control flow graph shows all the paths that can be traversed during a program execution.
- Control flow graph is a directed graph.
- Edges in CFG portray control flow paths and the nodes in CFG portray basic blocks.

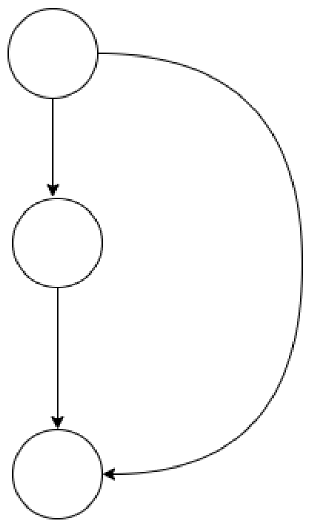
### Control flow graph for if else-if else statements



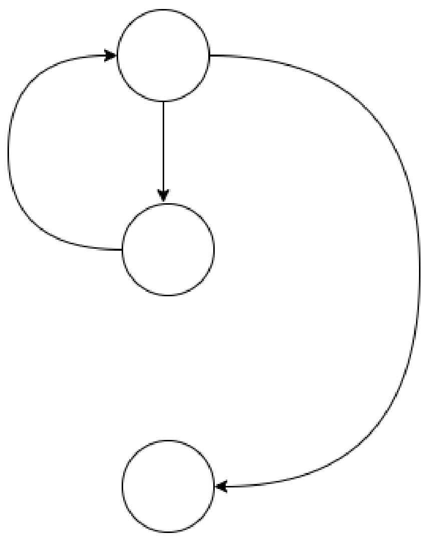
### Control flow graph for if else-if elseif statements



## Control flow graph for if statements



## Control flow graph for loops



## Algorithm for tokenizing code

```
function tokenize(characters):
    global txt //the actual code string
    myList=initializeList()
    pad characters list
    i=0
    while(i<length(characters)-padding):
        if(txt[i:i+3]=='for'):
            insertToken(myList,txt[i:i+3])
            i+=3
        elif(txt[i:i+5]=='while'):
            insertToken(myList,txt[i:i+5])
            i+=5
        elif(txt[i:i+2]=='if'):
            insertToken(myList,txt[i:i+2])
            i+=2
        elif(txt[i:i+2]=='do'):
            insertToken(myList,txt[i:i+2])
            i+=2
        elif(txt[i:i+4]=='else'):
            insertToken(myList,txt[i:i+4])
            i+=4
        elif(txt[i:i+5]=='break'):
            insertToken(myList,txt[i:i+5])
            i+=5
        elif(txt[i]!=' ' and txt[i]!='\n'):
            insertToken(myList,txt[i])
            i+=1
        else:
            i+=1
    return(myList)
```

Black spaces and newline characters are excluded and keywords like ‘if’, ‘else’, ‘for’ etc are tokenized

**Algorithm for generating Control Flow Graph**

(Via a depth first search approach)

Assumptions made:

- All if,else-if,else,for,while,do-while blocks will be followed by parenthesized code blocks
- No go-to statements exist

```
def seekBlockClosure(location):
    parenthesisCount=0
    while(True):
        if(location.char=='{'):
            parenthesisCount+=1
        elif(location.char=='}'):
            parenthesisCount-=1
        if(parenthesisCount==0):
            break
        location=location.right
    return(location)
def generateCfg(myGraph,body,exit,myList,listEnd):
    global maxIndex
    while(myList!=listEnd):
        if(myList.char=='if'):
            while(myList.char!='{'):
                myList=myList.right
            myNode=seekBlockClosure(myList)
            if(myNode.right.char!='else'):
                maxIndex+=2
                copyEdges(myGraph,body,maxIndex)
                myGraph[body]=None
                insertEdge(myGraph,body,maxIndex-1)
                insertEdge(myGraph,maxIndex-1,maxIndex)
                insertEdge(myGraph,body,maxIndex)
                generateCfg(myGraph,maxIndex-1,exit,myList,myNode)
                myList=myNode.right
                continue
            else:
                maxIndex+=1
                copyEdges(myGraph,body,maxIndex)
                myGraph[body]=None
                destinationNode=maxIndex

                maxIndex+=1
                insertEdge(myGraph,maxIndex,destinationNode)
                insertEdge(myGraph,body,maxIndex)
                generateCfg(myGraph,maxIndex,exit,myList,myNode)

                while(myNode.right.char=='else' and
myNode.right.right.char=='if'):
                    maxIndex+=1
                    insertEdge(myGraph,maxIndex,destinationNode)
                    insertEdge(myGraph,body,maxIndex)
                    while(myNode.char!='{'):
                        myNode=myNode.right
                    newNode=seekBlockClosure(myNode)
                    generateCfg(myGraph,maxIndex,exit,myNode,newNode)
                    myNode=newNode
                if(myNode.right.char!='else'):
                    insertEdge(myGraph,body,destinationNode)
                else:
                    maxIndex+=1
                    insertEdge(myGraph,maxIndex,destinationNode)
                    insertEdge(myGraph,body,maxIndex)
                    while(myNode.char!='{'):
                        myNode=myNode.right
                    newNode=seekBlockClosure(myNode)
                    generateCfg(myGraph,maxIndex,exit,myNode,newNode)
                    myNode=newNode
                myList=myNode.right
                continue
        elif(myList.char=='while' or myList.char=='for'):
            while(myList.char!='{'):
                myList=myList.right
            myNode=seekBlockClosure(myList)
            maxIndex+=2
            copyEdges(myGraph,body,maxIndex)
            myGraph[body]=None
            insertEdge(myGraph,body,maxIndex-1)
            insertEdge(myGraph,maxIndex-1,body)
            insertEdge(myGraph,body,maxIndex)
            generateCfg(myGraph,maxIndex-1,maxIndex,myList,myNode)
            myList=myNode.right
            continue
        elif(myList.char=='do'):
            while(myList.char!='{'):
                myList=myList.right
            myNode=seekBlockClosure(myList)
            maxIndex+=2
            copyEdges(myGraph,body,maxIndex)
            myGraph[body]=None
            insertEdge(myGraph,body,maxIndex-1)
            insertEdge(myGraph,maxIndex-1,maxIndex)
            insertEdge(myGraph,maxIndex-1,body)
            generateCfg(myGraph,body,maxIndex,myList,myNode)
            while(myNode.char!='while'):
                myNode=myNode.right
            while(myNode.char!=')'):
                myNode=myNode.right
            myList=myNode.right
            continue
        elif(myList.char=='break'):
            insertEdge(myGraph,body,exit)
        myList=myList.right
```

## Algorithm for evaluating cyclomatic complexity:

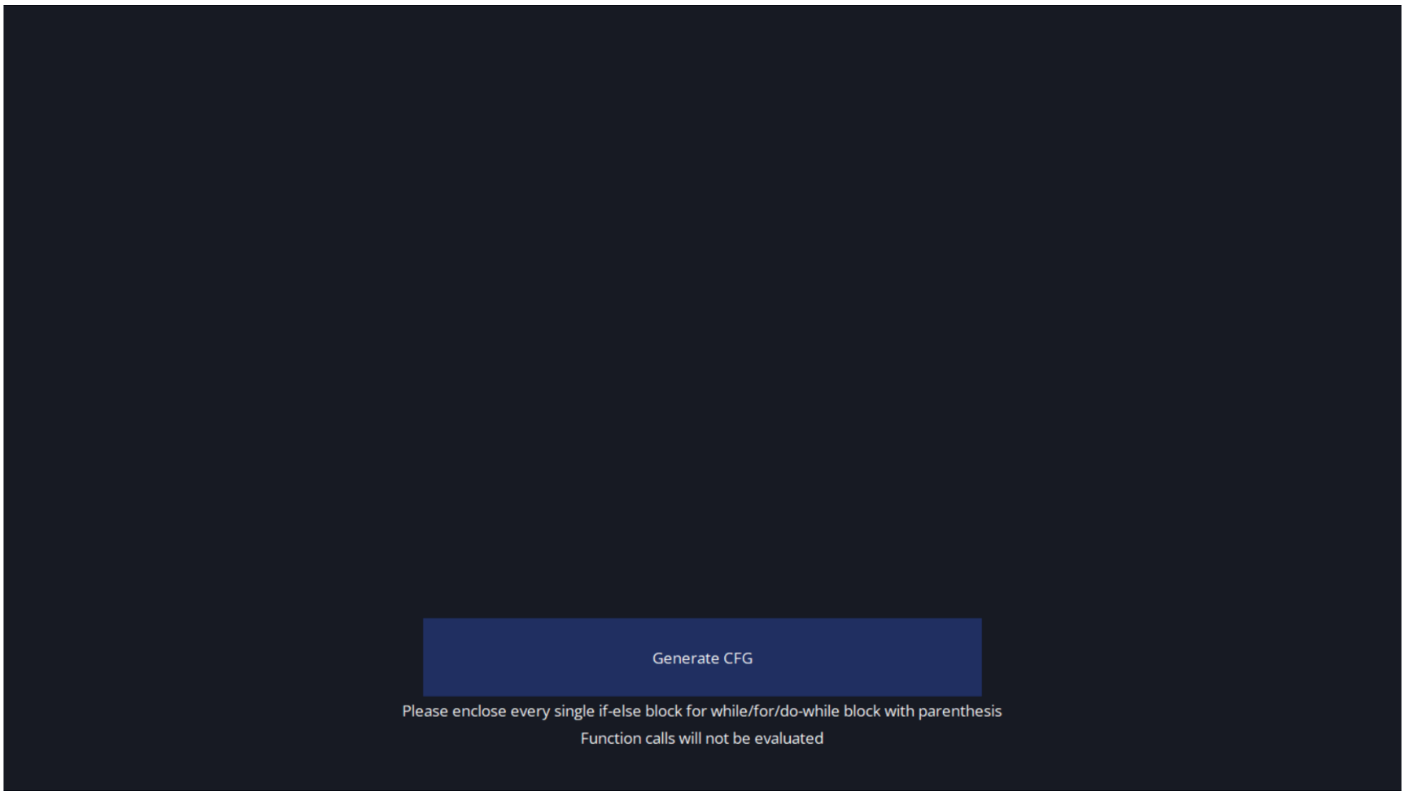
Cyclomatic complexity=E-n+2(p)

Where

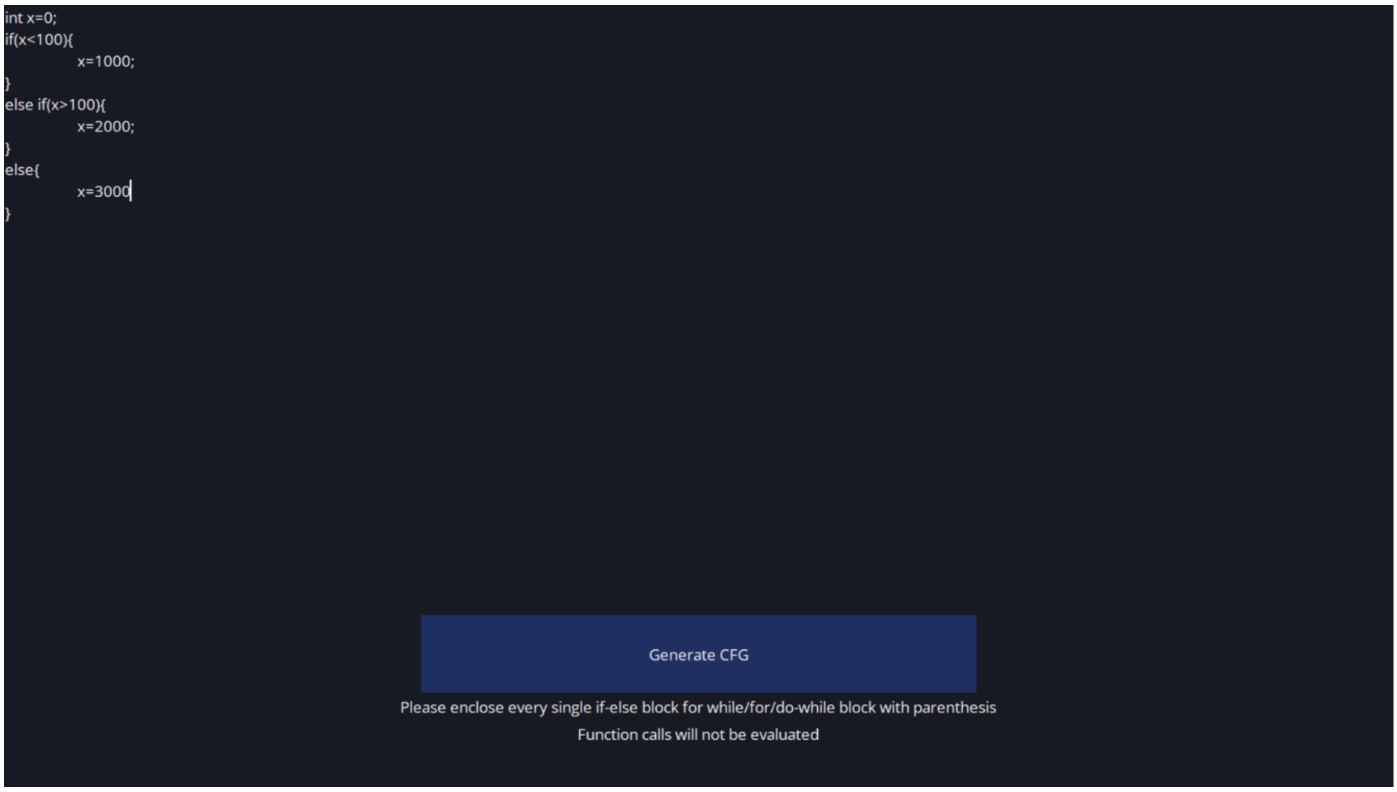
- E=number of edges
- n=number of nodes
- p=number of end points

**Output:**

1. Display Control Flow Graph and identify Cyclomatic complexity, independent paths.

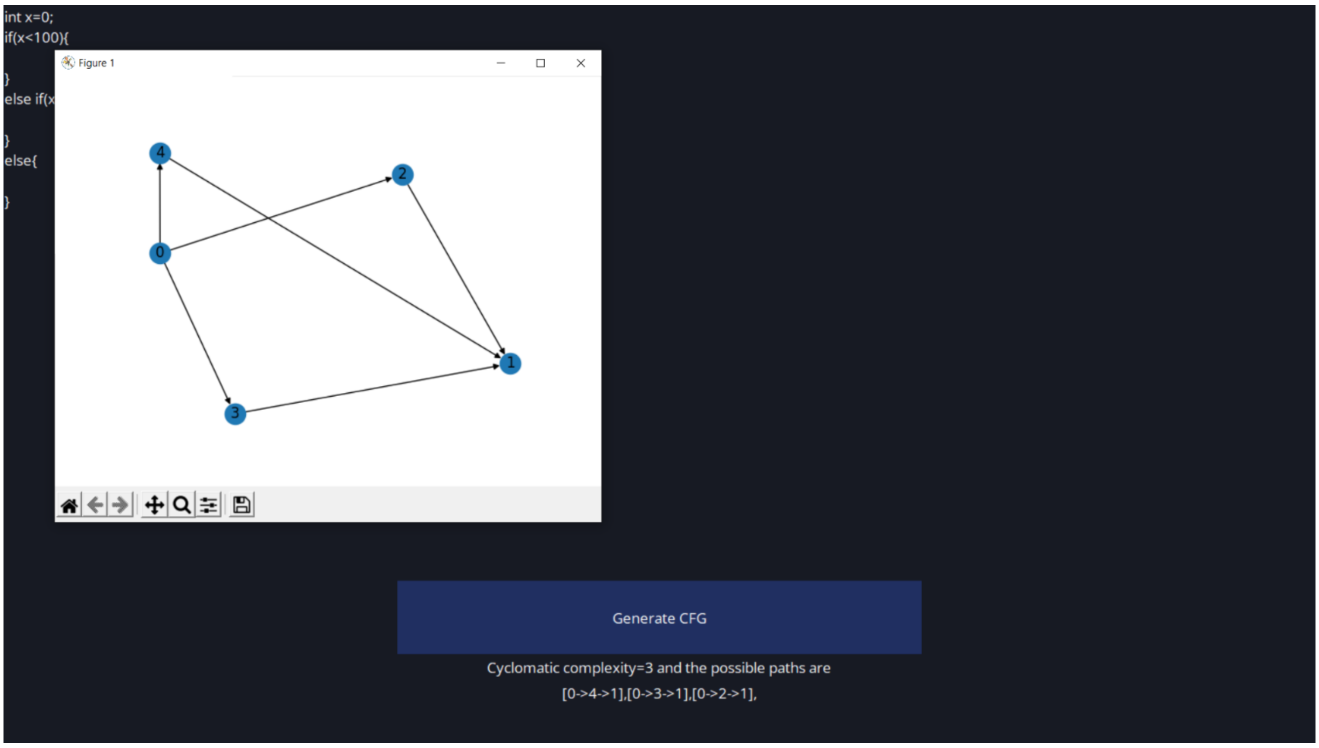


Please enclose every single if-else block for while/for/do-while block with parenthesis
Function calls will not be evaluated



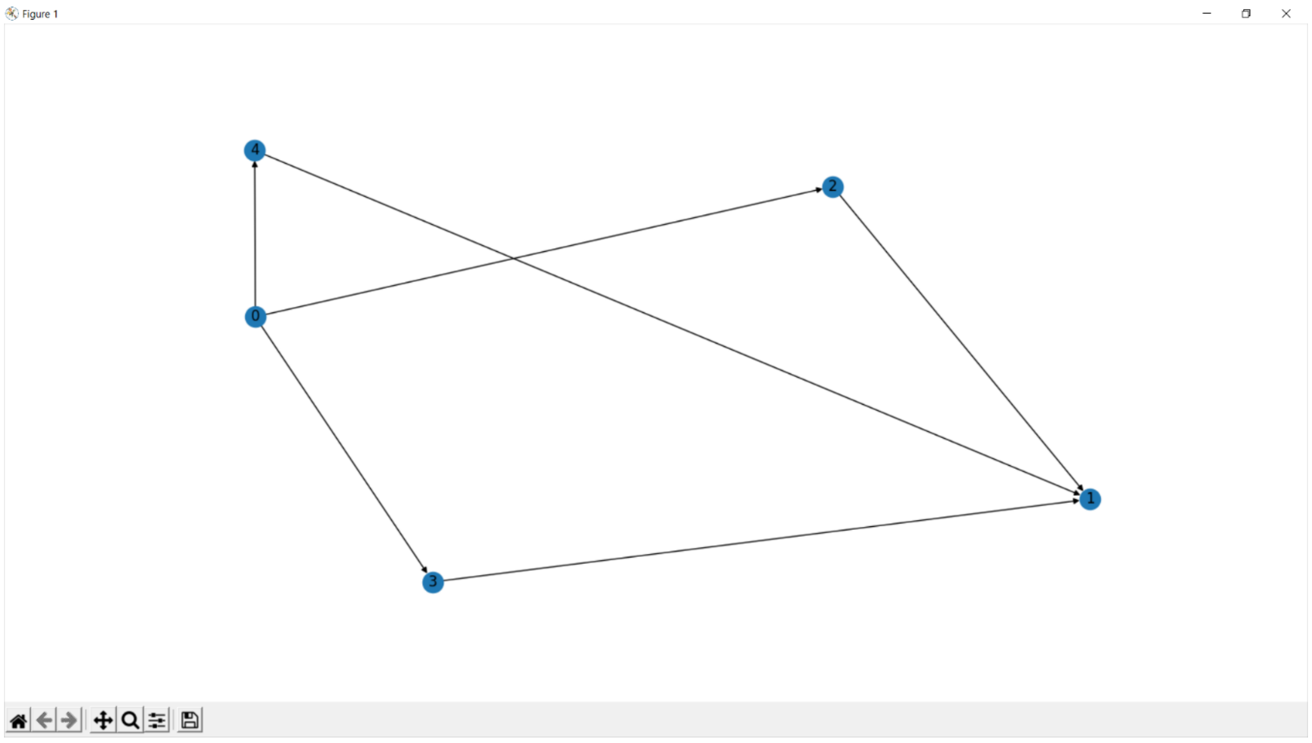
```
int x=0;
if(x<100){
		x=1000;
}
else if(x>100){
		x=2000;
}
else{
		x=3000
}
```

Please enclose every single if-else block for while/for/do-while block with parenthesis
Function calls will not be evaluated



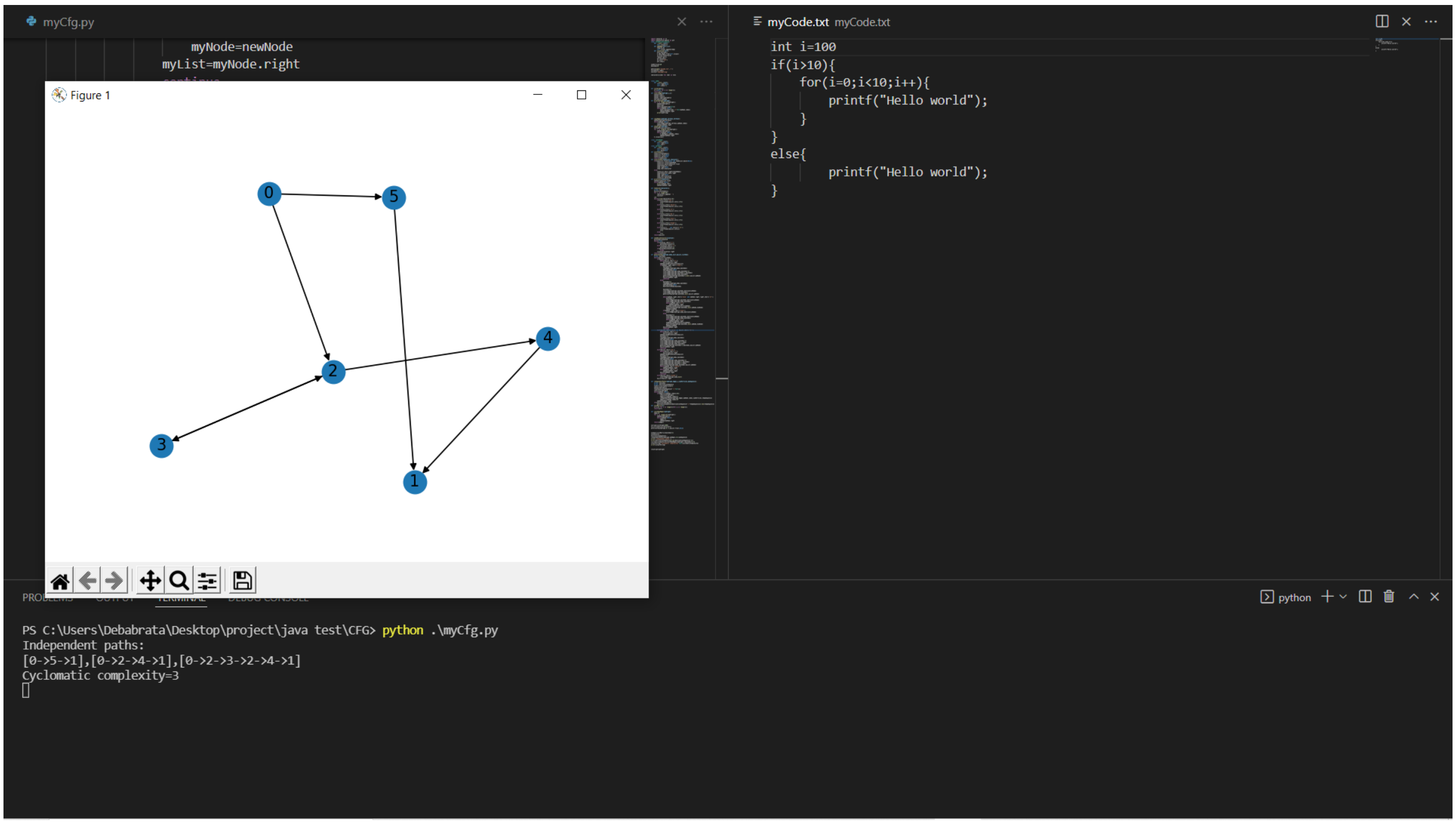
```
int x=0;
if(x<100){

}
else if(x

}
else{

}
```

Cyclomatic complexity=3 and the possible paths are
[0->4->1],[0->3->1],[0->2->1],



Write a program to create CFG and list out its properties from a given program file of any high level language (input file).

In the given code, the input file is myCode.txt



```
        myNode=newNode
    myList=myNode.right
```

```
int i=100
if(i>10){
    for(i=0;i<10;i++){
        printf("Hello world");
    }
}
else{
        printf("Hello world");
}
```

```
PS C:\Users\Debabrata\Desktop\project\java test\CFG> python .\myCfg.py
Independent paths:
[0->5->1],[0->2->4->1],[0->2->3->2->4->1]
Cyclomatic complexity=3
```

## Code for evaluating set of independent paths:

```
def independentPaths(myGraph,edges,i,numVertices,mySequence):
    global maxIndex
    global destinationSequence
    numVertices=numVertices+1
    myNode=myGraph[i]
    tempSequence=mySequence+'->'+str(i)
    undiscoveredFlag=0
    while(myNode!=None):
        if(edges[i][myNode.index]==0):
            undiscoveredFlag=1
            edges[i][myNode.index]=1
            independentPaths(myGraph,edges,myNode.index,numVertices,tempSequen
ce)
            edges[i][myNode.index]=0
        myNode=myNode.right
    if(undiscoveredFlag==0):
        destinationSequence=destinationSequence+'['+tempSequence[2:len(tempSeq
uence)]+'],'
```

**Citations:**

- **GeeksForGeeks website for documentation and diagrams**
- **Prof. CHOUHAN KUMAR RATH for code review**
- **Prof. Anirban Sarkar**