Contents

[1. Introduction 2](#_Toc508823498)

[1.1 Problem That I am Solving 2](#_Toc508823499)

[1.2 Background Information: 2](#_Toc508823500)

[2. Requirements 3](#_Toc508823501)

[3. Description of my Python program 3](#_Toc508823502)

[4. Screenshots of the program output 4](#_Toc508823503)

[Main HTML Form: 4](#_Toc508823504)

[Results Download Page: 5](#_Toc508823505)

[Click on the Download Robot File option: 5](#_Toc508823506)

[Click on Go to Main Page: 5](#_Toc508823507)

[5. Conclusion 5](#_Toc508823508)

[6. Python program 7](#_Toc508823509)

# Introduction

1.1 Problem That I am Solving*:*

*As part of this project I developed a web based tool which will generate a .ROBOT script for testing various Negative triggers. The script generated by this tool is readily executable without any additional changes or efforts.*

## 1.2 Background Information:

In my current job as part of functional test team, my primary responsibilities includes testing of various new L2 and L3 routing functionality.

Typical tasks involved our day to day job are:

1. Designing test network topology.
2. Write FTP (Functional Test Plan).
3. Build the testbed.
4. Automate all the TCs (test cases) in FTP.
5. Perform Testing using scripts developed

Each FTP contains good amount (approximately 30%) of Negative TCs.

In our company we use ROBOT frame work for our automation.

**Typical Steps involved in a Negative TC are:**

1. Load Initial Configuration on all the devices in the testbed.
2. Perform Checks in Steady state – Check that all primary protocols are UP and control and Forwarding Plane entries exists as expected.
3. Perform Negative Trigger on DUT (Device Under Test)
4. Check again all primary protocols are UP and control and Forwarding Plane entries exists as expected (after negative trigger)
5. Perform core functionality check, to ensure DUT is recovered after Negative trigger.

Typically testers will write KW (Keywords – Equivalent to Functions/Subroutines in other languages) for functional TCs (which covers steps# a, b, d and e mentioned above). For Negative Triggers, the only additional step is #c. This tool reuses the KWs already developed by the tester for any kind of Checks and only adds script snippet related to Negative triggers and any common checks.

**Here are the typical Negative Triggers we use**:

1. Flap Protocols – Protocols like OSPF, ISIS, BGP, MPLS, etc.
2. Restart Daemons – Daemons like RPD, L2ALD, Chassisd, etc.
3. Links Flap – Core links, access links, etc.
4. High Availability Tests – GR, NSR, ISSU, etc.

***Advantages of this tool****:*

1. Saves time. Though the time saved per user is just few hours, it will be huge if we consider the total time saved for the whole team.
2. Include Common Checks like CPU Hogs, memory leaks, errors in log files, etc – Typically testers focus on the checks related to their functionality and tend to ignore common check (due to lck of knowledge or lack of time) and this tool covers such checks by adding KW related to common checks automatically.
3. This approach can be extended for other kind of tasks.

# Requirements

Here is the list of all the Python modules that need to be installed to run my program:

import re

from flask import Flask, render\_template, request, url\_for, redirect, send\_file

from werkzeug.datastructures import CombinedMultiDict, MultiDict

# Description of my Python program

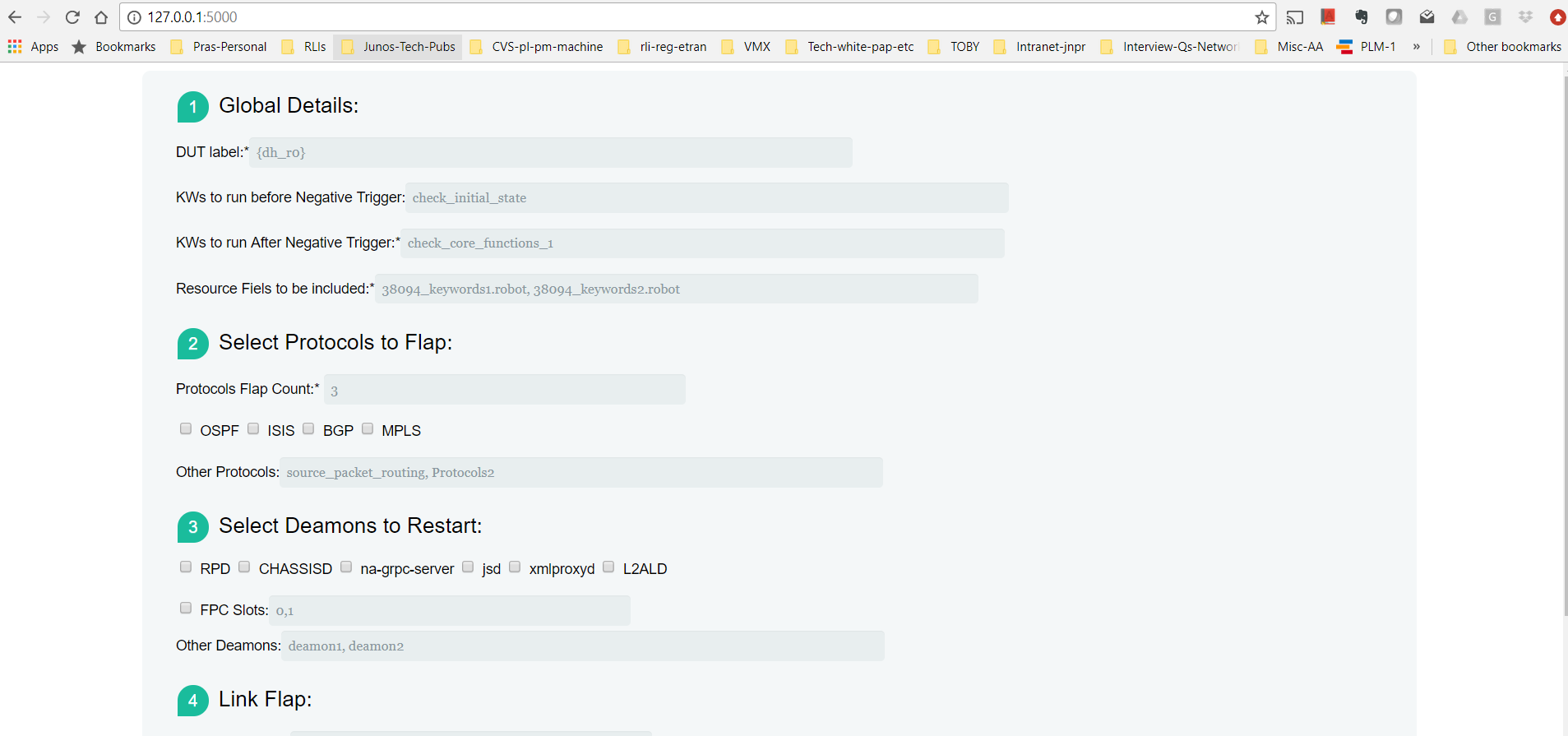
The program starts a webserver (using FLASK) and when users access the root HTML page, it will show a HTML form. Users will enter required fields and clicks on the SUBMIT button. Next the program takes to a different page where the user has option to download the automatically generated .ROBOT file or to go back to the main page to re-submit the form.

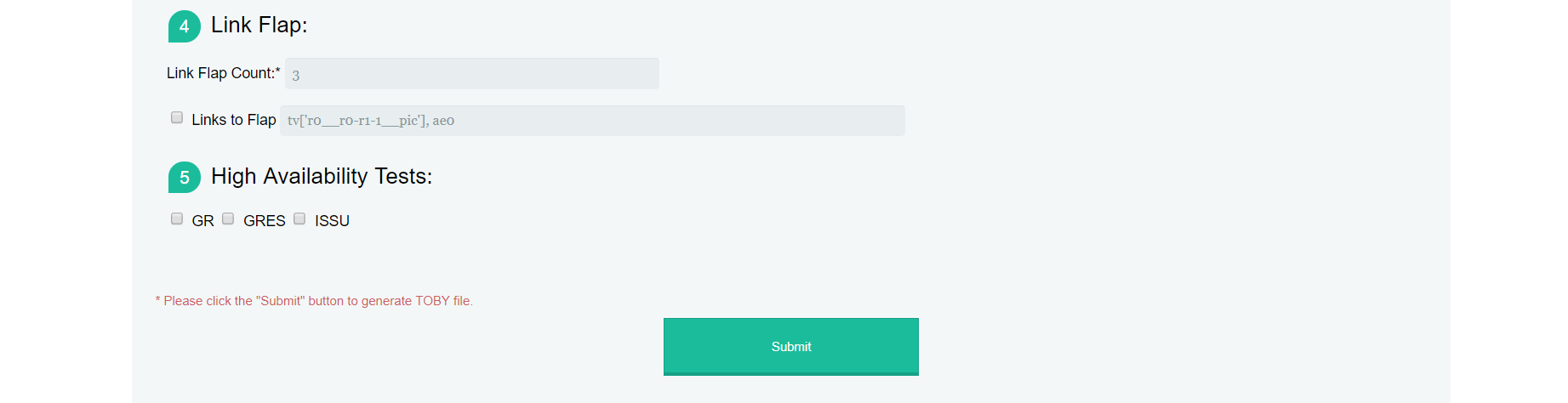
Steps involved in my python program execution:

1. Start Web server and users can access the main HTML form from <http://127.0.0.1:5000/>.
2. Users will enter the appropriate fields in the form and click on Submit.
3. Python program receives the data from HTML page and saves it into a python dictionary.
4. Python program reads each key in the dictionary and generates code for corresponding TC and writes it into a .ROBOT file.
5. After reading the whole Dictionary and generating and writing all the corresponding TCs, the file is closed and a 2nd HTML page is displayed to the user.
6. User has option to download the automatically generated .ROBOT file or to go back to the main page to re-submit the form.

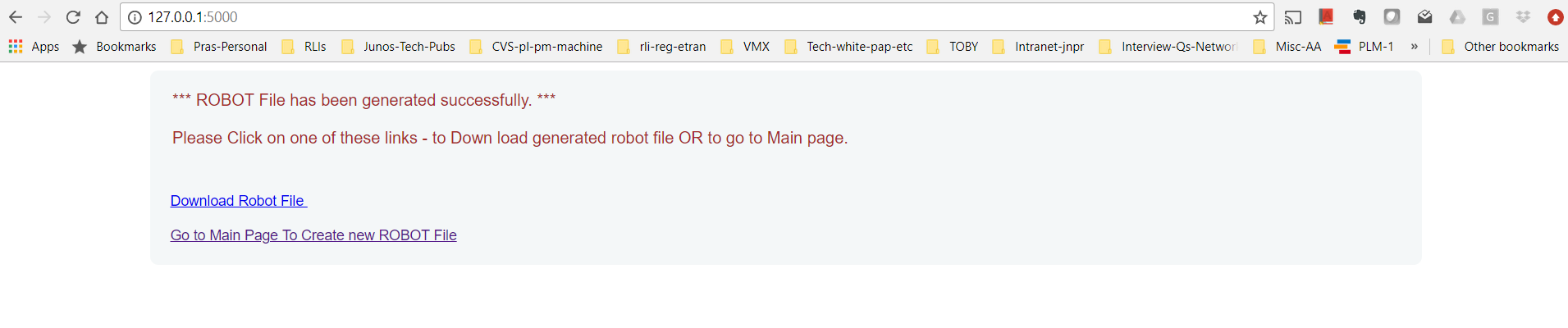
# Screenshots of the program output

## Main HTML Form:

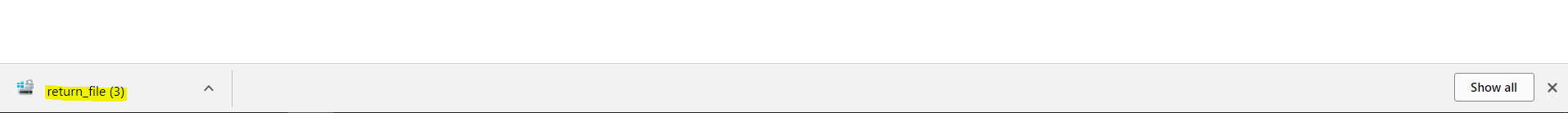




## Results Download Page:



## Click on the Download Robot File option:

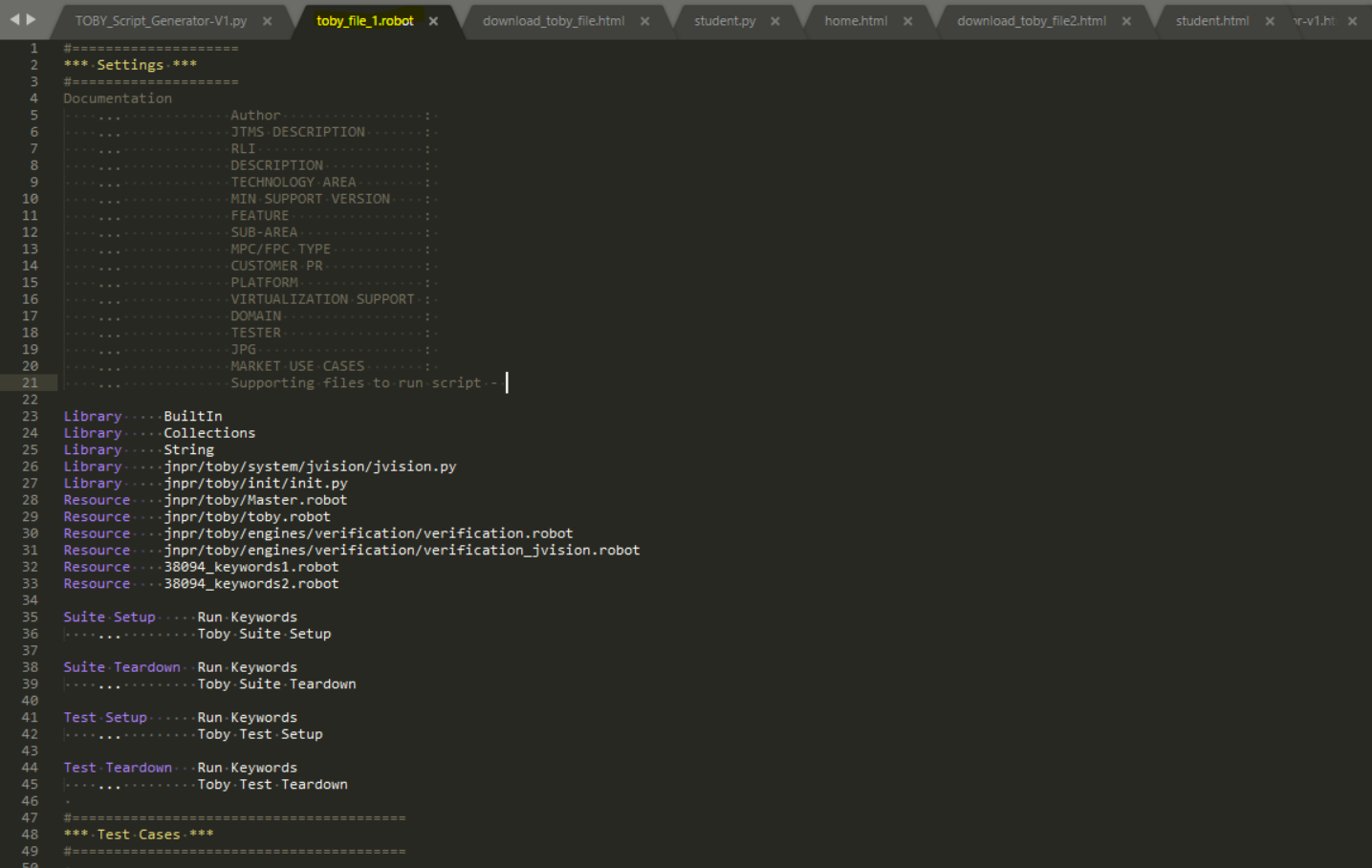


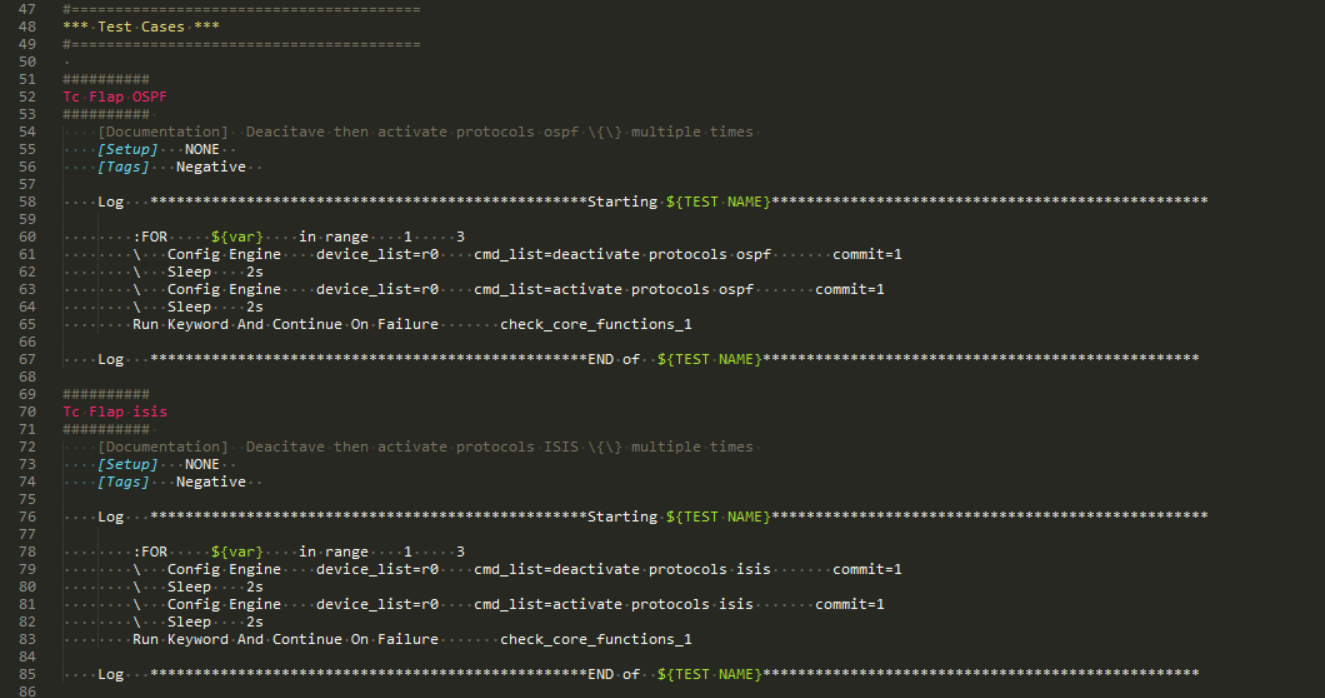
## Click on Go to Main Page:

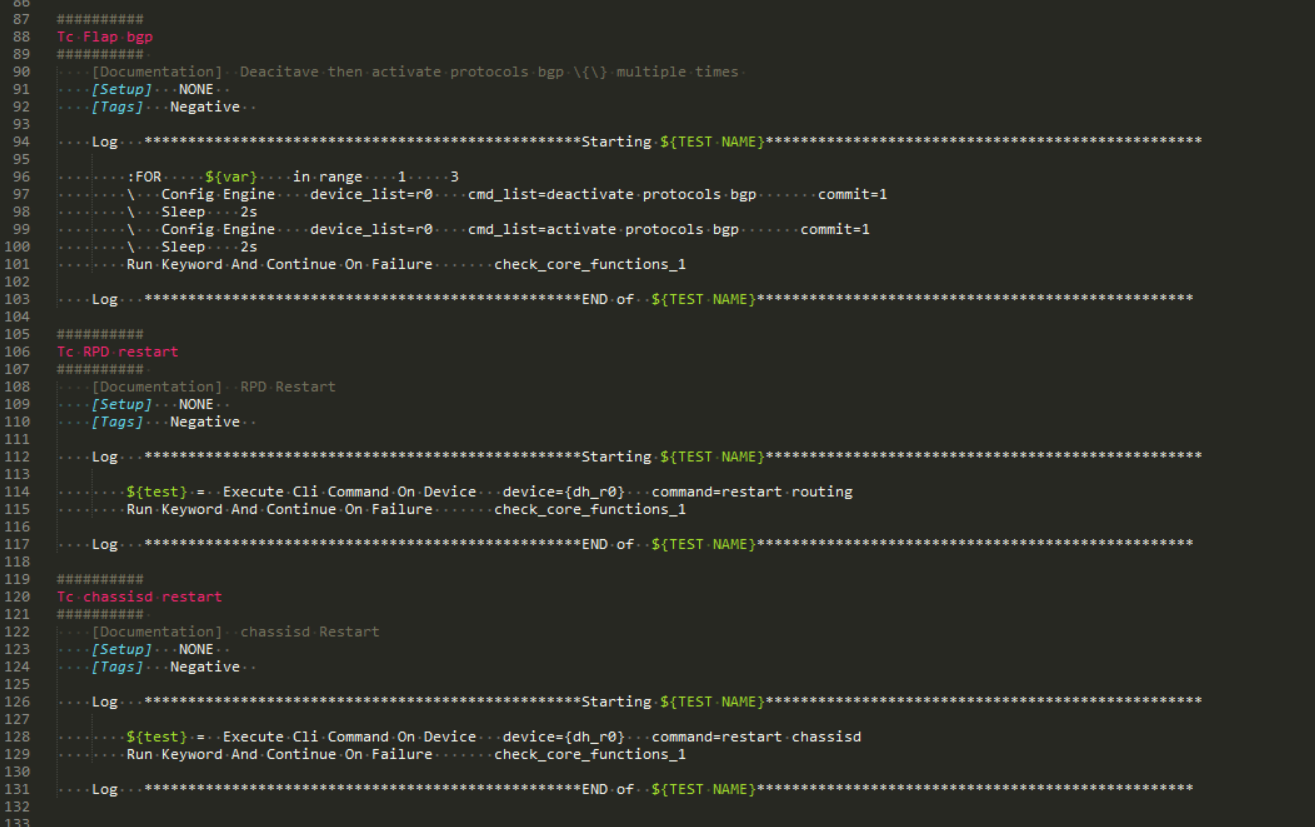
Control goes to the the Main HTML Page, shown above.

# Conclusion

Using my program a .ROBOT file shown below is generated:







# Python program

import os, re

from flask import Flask, render\_template, request, url\_for, redirect, json, send\_file

from werkzeug.datastructures import CombinedMultiDict, MultiDict

app = Flask(\_\_name\_\_)

@app.route('/',methods=['GET', 'POST'])

def form1():

if request.method == "POST":

create\_toby\_file(request.values)

#return render\_template('download\_toby\_file.html')

return render\_template('download\_toby\_file2.html')

else:

return render\_template('toby\_script\_generator-v1.html')

@app.route('/return\_file')

def return\_file():

return send\_file('toby\_file\_1.robot')

@app.route('/results',methods=['GET', 'POST'])

def form1results():

if request.method == "POST":

return render\_template('form1result.html')

else:

return render\_template('form1result.html')

def create\_toby\_file(data\_from\_form):

print ("inside create\_toby\_file")

form\_dict = data\_from\_form.to\_dict()

print(form\_dict)

patterns = [',']

dut\_handle = form\_dict.get('dut\_handle', 'dh\_none')

#

kw0 = form\_dict.get('kw0')

if (kw0):

if re.search(',', kw0):

kw0 = kw0.split(',')

kw0 = [x.strip() for x in kw0]

else:

kw0 = kw0.strip()

#

kw1 = form\_dict.get('kw1')

if (kw1):

if re.search(',', kw1):

kw1 = kw1.split(',')

kw1 = [x.strip() for x in kw1]

else:

kw1 = kw1.strip()

#

fr = open("C:\\Users\\pgudipati\\Me-Pras-Cloud\\Technical-Docs\\Python-UCSC\\Project\\toby\_file\_1.robot", 'w')

fr.write('''#====================\n''')

fr.write('''\*\*\* Settings \*\*\*\n''')

fr.write('''#====================\n''')

fr.write('''Documentation\n''')

fr.write(''' ... Author : \n''')

fr.write(''' ... JTMS DESCRIPTION : \n''')

fr.write(''' ... RLI : \n''')

fr.write(''' ... DESCRIPTION : \n''')

fr.write(''' ... TECHNOLOGY AREA : \n''')

fr.write(''' ... MIN SUPPORT VERSION : \n''')

fr.write(''' ... FEATURE : \n''')

fr.write(''' ... SUB-AREA : \n''')

fr.write(''' ... MPC/FPC TYPE : \n''')

fr.write(''' ... CUSTOMER PR : \n''')

fr.write(''' ... PLATFORM : \n''')

fr.write(''' ... VIRTUALIZATION SUPPORT : \n''')

fr.write(''' ... DOMAIN : \n''')

fr.write(''' ... TESTER : \n''')

fr.write(''' ... JPG : \n''')

fr.write(''' ... MARKET USE CASES : \n''')

fr.write(''' ... Supporting files to run script - \n''')

fr.write('''\n''')

fr.write('''Library BuiltIn\n''')

fr.write('''Library Collections\n''')

fr.write('''Library String\n''')

fr.write('''Library jnpr/toby/system/jvision/jvision.py\n''')

fr.write('''Library jnpr/toby/init/init.py\n''')

fr.write('''Resource jnpr/toby/Master.robot\n''')

fr.write('''Resource jnpr/toby/toby.robot\n''')

fr.write('''Resource jnpr/toby/engines/verification/verification.robot\n''')

fr.write('''Resource jnpr/toby/engines/verification/verification\_jvision.robot\n''')

#

resource\_files = form\_dict.get('resource\_files')

if resource\_files:

if re.search(',', resource\_files):

resource\_files = resource\_files.split(',')

resource\_files = [x.strip() for x in resource\_files]

else:

resource\_files = resource\_files.strip()

#

if isinstance(resource\_files, list):

for i in resource\_files:

fr.write('''Resource '''+i+'''\n''')

else:

fr.write('''Resource '''+resource\_files+'''\n''')

fr.write('''\n''')

fr.write('''Suite Setup Run Keywords\n''')

fr.write(''' ... Toby Suite Setup\n''')

fr.write('''\n''')

fr.write('''Suite Teardown Run Keywords\n''')

fr.write(''' ... Toby Suite Teardown\n''')

fr.write('''\n''')

fr.write('''Test Setup Run Keywords\n''')

fr.write(''' ... Toby Test Setup\n''')

fr.write('''\n''')

fr.write('''Test Teardown Run Keywords\n''')

fr.write(''' ... Toby Test Teardown\n''')

fr.write(''' \n''')

fr.write('''#========================================\n''')

fr.write('''\*\*\* Test Cases \*\*\*\n''')

fr.write('''#========================================\n''')

fr.write(''' \n''')

# TCs for Protocols Flapping:

# OSPF

proto\_flap\_count = form\_dict.get('prot\_flap\_count')

ospf\_flag = form\_dict.get('ospf', 'No')

if (ospf\_flag == "yes"):

fr.write('''##########\n''')

fr.write('''Tc Flap OSPF\n''')

fr.write('''########## \n''')

fr.write(''' [Documentation] Deacitave then activate protocols ospf \{\} multiple times \n''')

fr.write(''' [Setup] NONE \n''')

fr.write(''' [Tags] Negative \n\n''')

fr.write(''' Log \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Starting ${TEST NAME}\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n''')

fr.write(''' :FOR ${var} in range 1 ''' + str(proto\_flap\_count) + '''\n''')

fr.write(''' \\ Config Engine device\_list=r0 cmd\_list=deactivate protocols ospf commit=1\n''')

fr.write(''' \\ Sleep 2s\n''')

fr.write(''' \\ Config Engine device\_list=r0 cmd\_list=activate protocols ospf commit=1\n''')

fr.write(''' \\ Sleep 2s\n''')

fr.write(''' Run Keyword And Continue On Failure ''' + kw1 + '''\n\n''')

fr.write(''' Log \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END of ${TEST NAME}\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n''')

# isis

isis\_flag = form\_dict.get('isis', 'No')

if (isis\_flag == "yes"):

fr.write('''##########\n''')

fr.write('''Tc Flap isis\n''')

fr.write('''########## \n''')

fr.write(''' [Documentation] Deacitave then activate protocols ISIS \{\} multiple times \n''')

fr.write(''' [Setup] NONE \n''')

fr.write(''' [Tags] Negative \n\n''')

fr.write(''' Log \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Starting ${TEST NAME}\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n''')

fr.write(''' :FOR ${var} in range 1 ''' + str(proto\_flap\_count) + '''\n''')

fr.write(''' \\ Config Engine device\_list=r0 cmd\_list=deactivate protocols isis commit=1\n''')

fr.write(''' \\ Sleep 2s\n''')

fr.write(''' \\ Config Engine device\_list=r0 cmd\_list=activate protocols isis commit=1\n''')

fr.write(''' \\ Sleep 2s\n''')

fr.write(''' Run Keyword And Continue On Failure ''' + kw1 + '''\n\n''')

fr.write(''' Log \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END of ${TEST NAME}\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n''')

# bgp

bgp\_flag = form\_dict.get('bgp', 'No')

if (bgp\_flag == "yes"):

fr.write('''##########\n''')

fr.write('''Tc Flap bgp\n''')

fr.write('''########## \n''')

fr.write(''' [Documentation] Deacitave then activate protocols bgp \{\} multiple times \n''')

fr.write(''' [Setup] NONE \n''')

fr.write(''' [Tags] Negative \n\n''')

fr.write(''' Log \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Starting ${TEST NAME}\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n''')

fr.write(''' :FOR ${var} in range 1 ''' + str(proto\_flap\_count) + '''\n''')

fr.write(''' \\ Config Engine device\_list=r0 cmd\_list=deactivate protocols bgp commit=1\n''')

fr.write(''' \\ Sleep 2s\n''')

fr.write(''' \\ Config Engine device\_list=r0 cmd\_list=activate protocols bgp commit=1\n''')

fr.write(''' \\ Sleep 2s\n''')

fr.write(''' Run Keyword And Continue On Failure ''' + kw1 + '''\n\n''')

fr.write(''' Log \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END of ${TEST NAME}\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n''')

# mpls

mpls\_flag = form\_dict.get('mpls', 'No')

if (mpls\_flag == "yes"):

fr.write('''##########\n''')

fr.write('''Tc Flap mpls\n''')

fr.write('''########## \n''')

fr.write(''' [Documentation] Deacitave then activate protocols mpls \{\} multiple times \n''')

fr.write(''' [Setup] NONE \n''')

fr.write(''' [Tags] Negative \n\n''')

fr.write(''' Log \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Starting ${TEST NAME}\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n''')

fr.write(''' :FOR ${var} in range 1 ''' + str(proto\_flap\_count) + '''\n''')

fr.write(''' \\ Config Engine device\_list=r0 cmd\_list=deactivate protocols mpls commit=1\n''')

fr.write(''' \\ Sleep 2s\n''')

fr.write(''' \\ Config Engine device\_list=r0 cmd\_list=activate protocols mpls commit=1\n''')

fr.write(''' \\ Sleep 2s\n''')

fr.write(''' Run Keyword And Continue On Failure ''' + kw1 + '''\n\n''')

fr.write(''' Log \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END of ${TEST NAME}\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n''')

# TCs for Deamons Restart:

# RPD restart

rpd\_flag = form\_dict.get('rpd', 'No')

if (rpd\_flag == "yes"):

fr.write('''##########\n''')

fr.write('''Tc RPD restart\n''')

fr.write('''########## \n''')

fr.write(''' [Documentation] RPD Restart\n''')

fr.write(''' [Setup] NONE \n''')

fr.write(''' [Tags] Negative \n\n''')

fr.write(''' Log \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Starting ${TEST NAME}\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n''')

fr.write(''' ${test} = Execute Cli Command On Device device=''' + dut\_handle + ''' command=restart routing\n''')

fr.write(''' Run Keyword And Continue On Failure ''' + kw1 + '''\n\n''')

fr.write(''' Log \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END of ${TEST NAME}\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n''')

#chassisd

chassisd\_flag = form\_dict.get('chassisd', 'No')

if (chassisd\_flag == "yes"):

fr.write('''##########\n''')

fr.write('''Tc chassisd restart\n''')

fr.write('''########## \n''')

fr.write(''' [Documentation] chassisd Restart\n''')

fr.write(''' [Setup] NONE \n''')

fr.write(''' [Tags] Negative \n\n''')

fr.write(''' Log \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Starting ${TEST NAME}\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n''')

fr.write(''' ${test} = Execute Cli Command On Device device=''' + dut\_handle + ''' command=restart chassisd\n''')

fr.write(''' Run Keyword And Continue On Failure ''' + kw1 + '''\n\n''')

fr.write(''' Log \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END of ${TEST NAME}\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n''')

# TCs for Links Flap:

# End of File Writing =======================================

fr.close()

return True

if \_\_name\_\_ == "\_\_main\_\_":

#host = os.getenv('IP', '127.0.0.1')

#port = int(os.getenv('PORT', 5000))

#print (host, port)

app.run()