Save your solutions in a docx file with name in this format: cnit381_2_hw4_YOURNAME.docx and submit this file to Stout's Canvas.

Provide codes (in text) for each Challenge with result-evidences (can be screenshots).

If your solution is not correct, then try to understand the error messages, check the slides again, post questions on Canvas, rewrite the solution and test it again. Repeat this step until you get the correct solution.

Save the solutions in files for future reference or recap.

I'd recommend you use my Virtual Lab and CSR1000v virtual router to complete this homework.

Import CSR1000v (CSR1000v.ova) to VirtualBox like you did with the Virtual Lab machine.

Make note of the IP address of this router and use it in the rest of this homework in place of the default of 192.168.56.101.

Hands-On – NETCONF

Part 1: Launch the VMs and Verify Connectivity

Step 1: Verify connectivity between the VMs.

a. In the CSR1kv VM, press Enter to get a command prompt and then use **show ip interface brief** to verify that the IPv4 address is 192.168.56.101. If the address is different, make a note of it.

b. Open a terminal in Virtual Lab Machine.

c. Ping the CSR1kv to verify connectivity.

```
~$ ping -c 5 192.168.56.101
PING 192.168.56.101 (192.168.56.101) 56(84) bytes of data.
64 bytes from 192.168.56.101: icmp_seq=1 ttl=254 time=1.37 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=254 time=1.15 ms
64 bytes from 192.168.56.101: icmp_seq=3 ttl=254 time=0.981 ms
64 bytes from 192.168.56.101: icmp_seq=4 ttl=254 time=1.01 ms
64 bytes from 192.168.56.101: icmp_seq=5 ttl=254 time=1.14 ms

--- 192.168.56.101 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 0.981/1.130/1.365/0.135 ms
devasc@labvm:~$
```

Step 2: Verify SSH connectivity to the CSR1kv VM.

a. In the terminal for the Virtual Lab Machine, SSH to the CSR1kv VM with the following command:

```
~$ ssh  cisco@192.168.56.101
```

Note: The first time you SSH to the CSR1kv, your Virtual Lab Machine warns you about the authenticity of the CSR1kv. Because you trust the CSR1kv, answer yes to the prompt. If you have key mismatch problem, check out NETCONF slides at slide 12.

The authenticity of host '192.168.56.101 (192.168.56.101)' can't be established.

RSA key fingerprint is SHA256:HYv9K5Biw7PFiXeoCDO/LTqs3EfZKBuJdiPo34VXDUY.

Are you sure you want to continue connecting (yes/no/[fingerprint])? **yes**

Warning: Permanently added '192.168.56.101' (RSA) to the list of known hosts.

b. Enter **cisco123!** as the password and you should now be at the privileged EXEC command prompt for the CSR1kv.

Password:

CSR1kv#

c. Leave the SSH session open for the next Part.

Part 2: Use a NETCONF Session to Gather Information

In this Part, you will verify that NETCONF is running, enable NETCONF if it is not, and verify that NETCONF is ready for an SSH connection. You will then connect to the NETCONF process, start a NETCONF session, gather interface information, and close the session.

Step 1: Check if NETCONF is running on the CSR1kv.

a. NETCONF may already be running, or if a later IOS version enables it by default. From your SSH session with the CSR1kv, use the **show platform software yang-management process** command to see if the NETCONF SSH daemon (**ncsshd**) is running.

CSR1kv# **show platform software yang-management process**

```
confd          : Running
nesd           : Running
syncfd         : Running
ncsshd         : Running
dmiauthd       : Running
nginx          : Running
ndbmand        : Running
pubd           : Running
```

CSR1kv#

b. If NETCONF is not running, as shown in the output above, enter the global configuration command **netconf-yang**.

CSR1kv# **config t**

CSR1kv (config)# **netconf-yang**

c. Type **exit** to close the SSH session.

Step 2: Access the NETCONF process through an SSH terminal.

In this step, you will re-establish an SSH session with the CSR1kv. But this time, you will specify the NETCONF port 830 and send **netconf** as a subsystem command.

Note: For more information on these options, explore the manual pages for SSH (**man ssh**).

a. Enter the following command in another terminal window in Virtual Lab Machine. You can use the up arrow to recall the latest SSH command and just add the **-p** and **-s** options as shown. Then, enter **cisco123!** as the password.

devasc@labvm:~$ **ssh cisco@192.168.56.101 -p 830 -s netconf**

cisco@192.168.56.101's password:

b. The CSR1kv will respond with a **hello** message that includes over 400 lines of output listing all of its NETCONF capabilities. The end of NETCONF messages are identified with **]]>]]>**.

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
<capability>urn:ietf:params:netconf:base:1.1</capability>
<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
(output omitted)
    </capability>
</capabilities>
<session-id>20</session-id></hello>]]>]]>
```

Step 3: Start a NETCONF session by sending a hello message from the client.

To start a NETCONF session, the client needs to send its own hello message. The hello message should include the NETCONF base capabilities version the client wants to use.

a. Copy and paste the following XML code into the SSH session. Notice that the end of the client hello message is identified with a **]]>]]>**.

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <capabilities>
   <capability>urn:ietf:params:netconf:base:1.0</capability>
 </capabilities>
</hello>
]]>]]>
```

b. Switch to the CSR1kv VM and use the **show netconf-yang sessions** command to verify that a NETCONF session has been started. If the CSR1kv VM screen is dark, press **Enter** to wake it up.

CSR1kv> **en**

CSRk1v# **show netconf-yang sessions**

```
R: Global-lock on running datastore
C: Global-lock on candidate datastore
S: Global-lock on startup datastore

Number of sessions : 1

session-id  transport    username          source-host         global-lock
--------------------------------------------------------------------------------
20          netconf-ssh  cisco             192.168.56.1        None
```

CSR1kv#

Step 4: Send RPC messages to an IOS XE device.

During an SSH session, a NETCONF client can use Remote Procedure Call (RPC) messages to send NETCONF operations to the IOS XE device. The table lists some of the more common NETCONF operations.

Operation	Description
<get>	Retrieve running configuration and device state information
<get-config>	Retrieve all or part of a specified configuration data store
<edit-config>	Loads all or part of a configuration to the specified configuration data store
<copy-config>	Replace an entire configuration data store with another
<delete-config>	Delete a configuration data store
<commit>	Copy candidate data store to running data store
<lock> / <unlock>	Lock or unlock the entire configuration data store system
<close-session>	Graceful termination of NETCONF session
<kill-session>	Forced termination of NETCONF session

a. Copy and paste the following RPC **get** message XML code into the terminal SSH session to retrieve information about the interfaces on R1.

```
<rpc message-id="103" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <get>
  <filter>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"/>
   </filter>
 </get>
</rpc>
]]>]]>
```

b. Recall that XML does not require indention or white space. Therefore, the CSR1kv will return a long string of XML data.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="103"><data><interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"><interface><name>GigabitEthernet1</name><description>VBox</description><type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">ianaift:ethernetCsmacd</type><enabled>true</enabled><ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"></ipv4><ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"></ipv6></interface></interfaces></data></rpc-reply>]]>]]>
```

c. Copy the XML that was returned, but do not include the final “]]>]]>” characters. These characters are not part of the XML that is returned by the router.

d. Search the internet for “prettify XML”. Find a suitable site and use its tool to transform your XML into a more readable format, such as the following:

```
<?xml version="1.0"?>
```

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="103">
 <data>
  <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
   <interface>
    <name>GigabitEthernet1</name>
    <description>VBox</description>
    <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">ianaift:ethernetCsmacd</type>
    <enabled>true</enabled>
    <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"/>
    <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"/>
   </interface>
  </interfaces>
 </data>
</rpc-reply>
```

Step 5: Close the NETCONF session.

a. To close the NETCONF session, the client needs to send the following RPC message:

```
<rpc message-id="9999999" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <close-session />
</rpc>
```

b. After a few seconds, you will be returned to the terminal prompt. Return to the CSR1kv prompt and show the open netconf sessions. You will see that the session has been closed.

```
CSR1kv# show netconf-yang sessions
There are no active sessions

CSR1kv#
```

NETCONF-Challenges

Challenge #1

Create a Python script that connects to NETCONF of CSR1000v Router using *ncclient*. The script should return the capabilities of the router.

Challenge #2

Change the solution from Challenge #1 so that it returns the XML code of running configurations (in ‘running’ datastore). The return XML should be prettified by using *xml.dom.minidom* library.

Challenge #3

Change the solution from Challenge #2 so that it returns only Cisco-IOS-EX configurations.

Challenge #4

Change the solution from Challenge #3 so that it changes hostname of CSR1000v to R1.

Challenge #5

Create a Python script that connects to NETCONF of CSR1000v Router using ncclient. The script should create loopback 0 with IP address of 1.1.1.1, 255.255.255.0, and gives the interface a description.

Challenge #6

Create a Python script that connects to NETCONF of CSR1000v Router using ncclient. The script should show detail information about interface loopback 0. Note: running datastore doesn't store detail information. Try to understand the structure of the result, it will be helpful for the very next challenge.

Challenge #7

Change the solution from Challenge #6 so that it will print out information of Loopback 0:

1. IPv4 Address.
2. Interface Type.
3. MAC Address.
4. Packet-Input
5. Packet-Output
6. Admin-Status.
7. Oper-Status.

Can you print-out this information with Paramiko/Netmiko?

- If yes, how hard is it?
- If no, why?

Challenge #8

Create an XML file in your working folder with the contents below and name it *config_templ_ietf_interface.xml*.

```
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <{int_name}>
    <name>{int_num}</name>
    <description>{int_desc}</description>
    <ip>
     <address>
      <primary>
       <address>{ip_address}</address>
       <mask>{subnet_mask}</mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
```

Create an Python file in your working folder with the contents below and name it *routers.py*.

```
router = {"host":"192.168.56.101",
          "port":"830",
          "username":"cisco",
          "password":"cisco123!",
          "hostkey_verify":"False"
        }
```

Create a Python script that connects to NETCONF of CSR1000v Router using ncclient. The script should create loopback 100 with IP address of 100.1.1.1, 255.255.255.0, and gives the interface a description. You must:

- Use routers.py for manager.connect() method to set up NETCONF connection.
- Use *config_templ_ietf_interface.xml'* content as the netconf_payload to create and configure loopback 100.

Hands-On – RESTCONF

Part 1: Configure an IOS XE Device for RESTCONF Access

In this Part, you will configure the CSR1kv VM to accept RESTCONF messages. You will also start the HTTPS service.

Note: The services discussed in this Part may already be running on your VM. However, make sure you know the commands to view the running services and to enable them.

Step 1: Verify that the RESTCONF daemons are running.

RESTCONF should already be running because it was part of the default configuration provided by NetAcad. From the terminal, you can use the **show platform software yang-management process** command to see if all the daemons associated with the RESTCONF service are running. The NETCONF daemon may also be running, but it won't be used in this lab. If one or more of the required daemons are not running, proceed to Step 2.

```
CSR1kv# show platform software yang-management process
confd            : Running
nesd             : Running
syncfd           : Running
ncsshd           : Running
dmiauthd         : Running
nginx            : Running
ndbmand          : Running
pubd             : Running

CSR1kv#
```

Note: The purpose and function of all the daemons is beyond the scope of this course.

Step 2: Enable and verify the RESTCONF service.

a. Enter the global configuration command **restconf** to enable the RESTCONF service on the CSR1kv.

```
CSR1kv#configure terminal
CSR1kv(config)# restconf
```

b. Verify that the required RESTCONF daemons are now running. Recall that **ncsshd** is the NETCONF service, which may be running on your device. We do not need it for this lab. However, you do need **nginx**, which is the HTTPS server. This will allow you to make REST API calls to the RESTCONF service.

```
CSR1kv(config)# exit
CSR1kv# show platform software yang-management process
confd            : Running
nesd             : Running
syncfd           : Running
ncsshd           : Not Running
dmiauthd         : Running
nginx            : Not Running
ndbmand          : Running
pubd             : Running
```

Step 3: Enable and verify the HTTPS service.

a. Enter the following global configuration commands to enable the HTTPS server and specify that server authentication should use the local database.

```
CSR1kv# configure terminal
CSR1kv(config)# ip http secure-server
CSR1kv(config)# ip http authentication local
```

b. Verify that the HTTPS server (nginx) is now running.

```
CSR1kv(config)# exit
CSR1kv# show platform software yang-management process
confd          : Running
nesd           : Running
syncfd         : Running
ncsshd         : Not Running
dmiauthd       : Running
nginx          : Running
ndbmand        : Running
pubd           : Running
```

Part 2: Open and Configure Postman

In this Part, you will open Postman, disable SSL certificates, and explore the user interface.

Step 1: Open Postman.

a. In the **Virtual Lab Machine**, open the Postman application.

b. If this is the first time you have opened Postman, it may ask you to create an account or sign in. At the bottom of the window, you can also click the “Skip” message to skip signing in. Signing in is not required to use this application.

Step 2: Disable SSL certification verification.

By default, Postman has SSL certification verification turned on. You will not be using SSL certificates with the CSR1kv; therefore, you need to turn off this feature.

a. Click **File > Settings**.

b. Under the **General** tab, set the **SSL certificate verification** to **OFF**.

c. Close the **Settings** dialog box.

Part 3: Use Postman to Send GET Requests

In this Part, you will use Postman to send a GET request to the CSR1kv to verify that you can connect to the RESTCONF service.

Step 1: Explore the Postman user interface.

a. In the center, you will see the **Launchpad**. You can explore this area if you wish.

b. Click the plus sign (+) next to the **Launchpad** tab to open a **GET Untitled Request**. This interface is where you will do all of your work in this lab.

Step 2: Enter the URL for the CSR1kv.

a. The request type is already set to GET. Leave the request type set to GET.

b. In the “Enter request URL” field, type in the URL that will be used to access the RESTCONF service that is running on the CSR1kv:

   https://192.168.56.101/restconf/

Step 3: Enter authentication credentials.

Under the URL field, there are tabs listed for **Params**, **Authorization**, **Headers**, **Body**, **Pre-request Script**, **Test**, and **Settings**. In this lab, you will use **Authorization**, **Headers**, and **Body**.

a. Click the **Authorization** tab.

b. Under Type, click the down arrow next to “Inherit auth from parent” and choose **Basic Auth**.

c. For **Username** and **Password**, enter the local authentication credentials for the CSR1kv:

   Username: **cisco**

   Password: **cisco123!**

d. Click **Headers**. Then click the **7 hidden**. You can verify that the Authorization key has a Basic value that will be used to authenticate the request when it is sent to the CSR1kv.

Step 4: Set JSON as the data type to send to and receive from the CSR1kv.

You can send and receive data from the CSR1kv in XML or JSON format. For this lab, you will use JSON.

a. In the **Headers** area, click in the first blank **Key** field and type **Content-Type** for the type of key. In the **Value** field, type **application/yang-data+json**. This tells Postman to send JSON data to the CSR1kv.

b. Below your **Content-Type** key, add another key/value pair. The **Key** field is **Accept** and the **Value** field is **application/yang-data+json**.

Note: You can change application/yang-data+json to application/yang-data+xml to send and receive XML data instead of JSON data, if necessary.

Step 5: Send the API request to the CSR1kv.

Postman now has all the information it needs to send the GET request. Click **Send**. Below **Temporary Headers**, you should see the following JSON response from the CSR1kv. If not, verify that you completed the previous steps in this part of the lab and correctly configured RESTCONF and HTTPS service in Part 2.

```
{
   "ietf-restconf:restconf": {
      "data": {},
      "operations": {},
      "yang-library-version": "2016-06-21"
   }
}
```

This JSON response verifies that Postman can now send other REST API requests to the CSR1kv.

Step 6: Use a GET request to gather the information for all interfaces on the CSR1kv.

a. Now that you have a successful GET request, you can use it as a template for additional requests. At the top of Postman, next to the **Launchpad** tab, right-click the **GET** tab that you just used and choose **Duplicate Tab**.

b. Use the **ietf-interfaces** YANG model to gather interface information. For the URL, add **data/ietf-interfaces:interfaces**:

https://192.168.56.101/restconf/data/ietf-interfaces:interfaces

c. Click **Send**. You should see a JSON response from the CSR1kv that is similar to the output shown below. Your output may be different depending on your particular router.


```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "GigabitEthernet1",
        "description": "VBox",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
        "ietf-ip:ipv4": {},
        "ietf-ip:ipv6": {}
      }
    ]
  }
}
```


Step 7: Use a GET request to gather information for a specific interface on the CSR1kv.

In this lab, only the GigabitEthernet1 interface is configured. To specify just this interface, extend the URL to only request information for this interface.

a. Duplicate your last GET request.

b. Add the **interface=** parameter to specify an interface and type in the name of the interface.

https://192.168.56.101/restconf/data/ietf-interfaces:interfaces/interface=GigabitEthernet1

Note: If you request interface information from a different Cisco device with names that use forward slashes, such as GigabitEthernet0/0/1, use the HTML code **%2F** for the forward slashes in the interface name. So, **0/0/1** becomes **0%2F0%2F1**.

c. Click **Send**. You should see a JSON response from the CSR1kv that is similar to output below. Your output may be different depending on your particular router. In the default CSR1kv setup, you will not see IP addressing information.


```
{
  "ietf-interfaces:interface": {
    "name": "GigabitEthernet1",
    "description": "VBox",
    "type": "iana-if-type:ethernetCsmacd",
    "enabled": true,
    "ietf-ip:ipv4": {},
    "ietf-ip:ipv6": {}
  }
}
```


d. This interface receives addressing from a Virtual Box template. Therefore, the IPv4 address is not shown under **show running-config**. Instead, you will see the **ip address dhcp** command. You can see this also in the show ip interface brief output.

```
CSR1kv# show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.101  YES DHCP   up                    up
```

CSR1kv#

e. In the next Part you will need to use the JSON response from a manually configurated interface. Open a command terminal with the CSR1kv and manually configure the GigabitEthernet1 interface with the same IPv4 address currently assigned to it by Virtual Box.

```
CSR1kv# conf t
CSR1kv(config)# interface g1
CSR1kv(config-if)# ip address 192.168.56.101 255.255.255.0
CSR1kv(config-if)# end
CSR1kv# show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.101  YES manual up                    up
CSR1kv#
```

f. Return to Postman and send your GET request again. You should now see IPv4 addressing information in the JSON response, as shown below. In the next Part, you will copy this JSON format to create a new interface.


```
{
  "ietf-interfaces:interface": {
    "name": "GigabitEthernet1",
    "description": "VBox",
    "type": "iana-if-type:ethernetCsmacd",
    "enabled": true,
    "ietf-ip:ipv4": {
      "address": [
        {
          "ip": "192.168.56.101",
          "netmask": "255.255.255.0"
        }
      ]
    },
    "ietf-ip:ipv6": {}
  }
}
```


Part 4: Use Postman to Send a PUT Request

In this Part, you will configure Postman to send a PUT request to the CSR1kv to create a new loopback interface.

Note: If you created a Loopback interface in another lab, either remove it now or create a new one by using a different number.

Step 1: Duplicate and modify the last GET request.

a. Duplicate the last GET request.

b. For the **Type** of request, click the down arrow next to **GET** and choose **PUT**.

c. For the **interface=** parameter, change it to **=Loopback1** to specify a new interface.

https://192.168.56.101/restconf/data/ietf-interfaces:interfaces/interface=Loopback1

Step 2: Configure the body of the request specifying the information for the new loopback.

a. To send a PUT request, you need to provide the information for the body of the request. Next to the **Headers** tab, click **Body**. Then click the **Raw** radio button. The field is currently empty. If you click **Send** now, you will get error code **400 Bad Request** because Loopback1 does not exist yet and you did not provide enough information to create the interface.

b. Fill in the **Body** section with the required JSON data to create a new Loopback1 interface. You can copy the Body section of the previous GET request and modify it. Or you can copy the following into the Body section of your PUT request. Notice that the type of interface must be set to **softwareLoopback**.

```
{
 "ietf-interfaces:interface": {
  "name": "Loopback1",
  "description": "My first RESTCONF loopback",
  "type": "iana-if-type:softwareLoopback",
  "enabled": true,
  "ietf-ip:ipv4": {
   "address": [
    {
     "ip": "10.1.1.1",
     "netmask": "255.255.255.0"
    }
   ]
  },
  "ietf-ip:ipv6": {}
 }
}
```

c. Click **Send** to send the PUT request to the CSR1kv. Below the Body section, you should see the HTTP response code **Status: 201 Created**. This indicates that the resource was created successfully.

d. You can verify that the interface was created. Return to your SSH session with the CSR1kv and enter **show ip interface brief**. You can also run the Postman tab that contains the request to get information about the interfaces on the CSR1kv that was created in the previous Part of this lab.

```
CSR1kv# show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.101  YES manual up                    up
Loopback1              10.1.1.1        YES other  up                    up
CSR1kv#
```

RESTCONF-Challenges

Challenge #1

Create a Python script that connects to RESTCONF API of CSR1000v Router using request. The script has *get_configured_interfaces()* function that retrieves the list of interfaces (in JSON) on the device.

Test this function and provide the output.

Sample output:

```
[{'name': 'GigabitEthernet1',
  'type': 'iana-if-type:ethernetCsmacd',
  'enabled': True,
  'ietf-ip:ipv4': {},
  'ietf-ip:ipv6': {}},
 {'name': 'Loopback0',
  'type': 'iana-if-type:softwareLoopback',
  'enabled': True,
  'ietf-ip:ipv4': {'address': [{'ip': '10.1.1.1',
     'netmask': '255.255.255.0'}]},
  'ietf-ip:ipv6': {}}]
```

Challenge #2

Add shutdown_int() function to challenge#1. This function takes a string input as an interface name then use RESTCONF API to shut down this interface of CSR1kv.

Note: don't shut down GigabitEthernet1.

Test this function and provide the result.

Challenge #3

Add get_cpu() function to challenge#2. This function takes nothing and return the CPU usage of last five seconds of CSR1kv.

Test this function and provide the output.

Challenge #4

Add get_mem() function to challenge#3. This function takes nothing and return:

1. Total memory in MB.
2. Used memory in %.
3. Free memory in %

Test this function and provide the output.

Challenge #5

Add get_int_state() function to challenge#4. This function takes nothing and return interface-state of loopback1:

1. phys-address
2. speed
3. discontinuity-time
4. in-octets
5. in-unicast-pkts
6. in-broadcast-pkts
7. in-multicast-pkts
8. out-octets
9. out-unicast-pkts

Test this function and provide the outputs.

END.