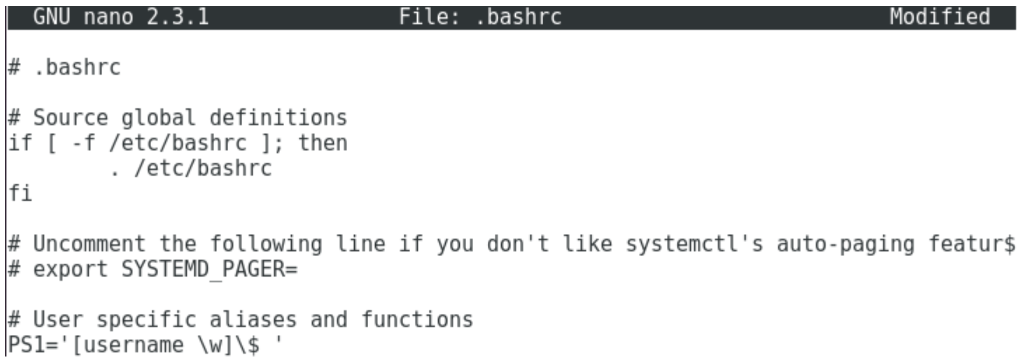
R·I·T Information Sciences and Technology Department

NSSA-220 Task Automation Using Interpretive Languages
Lab 3: Python Data Processing

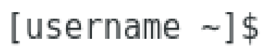
Activity 0 – Modify your shell prompt

Before you begin working on the lab, you need to modify the prompt in your shell (terminal) to reflect your username, rather than the generic prompt of "student@localhost". To modify your shell prompt, open a Terminal and type "nano .bashrc". We're now going to edit a file that executes every time you start up a terminal in your current session.



```
  GNU nano 2.3.1               File: .bashrc                            Modified

# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-paging featur$
# export SYSTEMD_PAGER=

# User specific aliases and functions
PS1='[username \w]\$ '
```

Now add a line to the end of the file similar to the one above, except that you should use your RIT username in place of the word "username". Save the file and exit nano. Then type exit to leave the Terminal. Open another Terminal and your prompt should now look like the following:



```
[username ~]$
```

We will be looking for the correct username in your screen shots when the lab is graded, so make sure you follow these steps before working on any lab.

Activity 1 – Python data processing

Perform the following tasks. For each task, include screenshot(s) that clearly indicate the output that proves that the task was accomplished correctly and submit your script that generated the output. Make sure that *your* username is in the screenshot prompt. If you're unable to perform the task as specified, you may receive partial credit by providing the output you were able to get, as well as the script that generated the output, and explain where you had difficulty. Any text you write should be written in red font.

Task 1 (50 points): Write a Python script called Lab3_Task1.py that takes a single command line argument to represent the input file name that the script will process. Your script will be processing a file called iris_full.txt, which is a famous classification algorithm benchmark data set used in the field of data mining/machine learning.

The Iris data set contains 150 records of iris flowers with five fields:

1. **Sepal length (in cm) of the iris**
2. **Sepal width (in cm) of the iris**
3. **Petal length (in cm) of the iris**

4. **Petal width (in cm) of the iris**
5. **Type of iris (either Iris Setosa, Iris Virginica, or Iris Versicolor)**

Your script will compute and output a report to the console that includes the following values:

1. **Minimum value of each of the first four fields**
2. **Maximum value of each of the first four fields**
3. **Average value of each of the first four fields**
4. **The number of occurrences of each type of iris**

The output should be in the following format:

Sepal Length: min = <min_value>, max = <max_value>, average = <average_value>

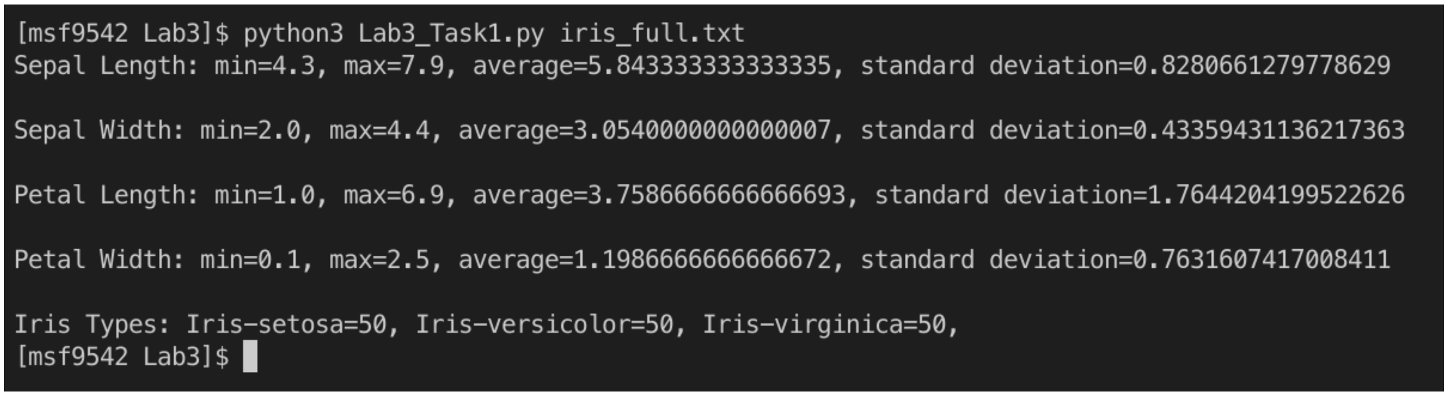
Sepal Width: min = <min_value>, max = <max_value>, average = <average_value>

Petal Length: min = <min_value>, max = <max_value>, average = <average_value>

Petal Width: min = <min_value>, max = <max_value>, average = <average_value>

Iris Types: Iris Setosa = <num_iris_setosa>, Iris Versicolor = <num_iris_versicolor>, Iris Virginica = <num_iris_virginica>

The correct answers for all of these outputs are shown in the comments section of iris_full.txt



```
[msf9542 Lab3]$ python3 Lab3_Task1.py iris_full.txt
Sepal Length: min=4.3, max=7.9, average=5.843333333333335, standard deviation=0.8280661279778629

Sepal Width: min=2.0, max=4.4, average=3.0540000000000007, standard deviation=0.43359431136217363

Petal Length: min=1.0, max=6.9, average=3.7586666666666693, standard deviation=1.7644204199522626

Petal Width: min=0.1, max=2.5, average=1.1986666666666672, standard deviation=0.7631607417008411

Iris Types: Iris-setosa=50, Iris-versicolor=50, Iris-virginica=50,
[msf9542 Lab3]$ 
```

Your script should contain the following functions:

1. **read_data: this function takes two arguments – a file name to read and a List L to store all the records from the file in. Your script must read the file *as is*. Do not modify the file prior to reading it.**
2. **process_numeric_field: the function takes two arguments – the List of records L and the field number to process (1, 2, 3, or 4). The function returns three values – min, max, and average of the field.**
3. **count_iris_types: the function takes one argument – the List of records L. The function returns three values – the number of Iris Setosa, Iris Versicolor, and Iris Virginica in the data set.**

<u>Extra credit (10 points)</u>: process_numeric_field also computes the standard deviation of a field. The correct answers for each field are also shown in iris_full.txt (refer to SD). If you compute the standard deviation of a field, output it along with the rest of the statistics for a field by including ", standard deviation = <std_value>" in the output.

For this task, submit the Lab3_Task1.py script as part of your Lab3.zip file.

<u>Task 2 (50 points)</u>: Write a Python script called Lab3_Task2.py that takes two command line arguments, one for each of the files that the script will process.

Your company's security department recently discovered a network breach and they suspect that the CentOS VM image you have been using was compromised. Specifically, they're worried that the executables in /usr/bin could have been modified with versions of those same executables that now contain malicious code.

You have been provided with a file called md5_new.txt that contains the MD5 message digests for nearly all of the executables in /usr/bin. MD5 message digests are 128-bit hashes that have been computed for the executables shown in md5_new.txt. Each line in the file contains the name of an executable in /usr/bin and its MD5 message digest that was computed before the network breach. These message digests are important because they should be unique to those specific executables and if the message digest is now different, that executable was affected by the breach. MD5 message digests for any file may be computed using a system call to md5sum.

In addition, you've been provided with a file called md5_original.txt, which contains the MD5 message digests for the executables in /usr/bin prior to the suspected network breach.

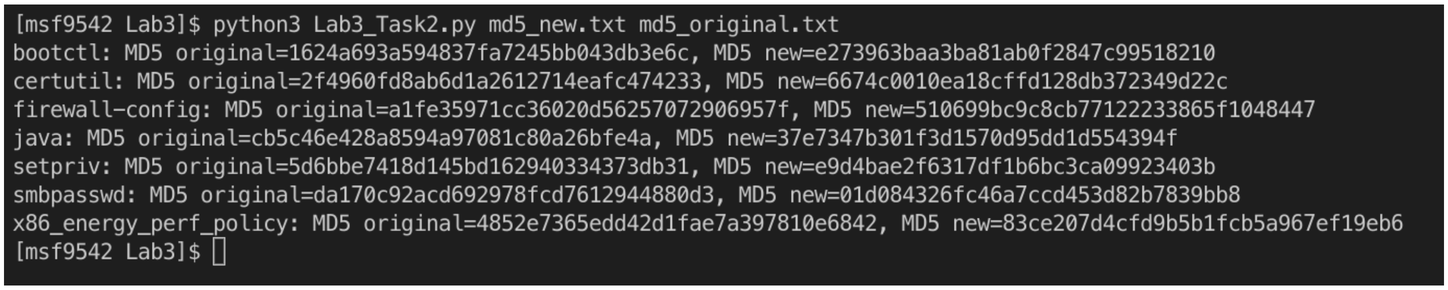
Your script will output the names of executables that may have been compromised to the console using the following format:

<affected_executable_name_1>: MD5 original = <md5_from_file>, MD5 new = <md5_from_your_script>

<affected_executable_name_2>: MD5 original = <md5_from_file>, MD5 new = <md5_from_your_script>

…

<affected_executable_name_n>: MD5 original = <md5_from_file>, MD5 new = <md5_from_your_script>



```
[msf9542 Lab3]$ python3 Lab3_Task2.py md5_new.txt md5_original.txt
bootctl: MD5 original=1624a693a594837fa7245bb043db3e6c, MD5 new=e273963baa3ba81ab0f2847c99518210
certutil: MD5 original=2f4960fd8ab6d1a2612714eafc474233, MD5 new=6674c0010ea18cffd128db372349d22c
firewall-config: MD5 original=a1fe35971cc36020d56257072906957f, MD5 new=510699bc9c8cb77122233865f1048447
java: MD5 original=cb5c46e428a8594a97081c80a26bfe4a, MD5 new=37e7347b301f3d1570d95dd1d554394f
setpriv: MD5 original=5d6bbe7418d145bd162940334373db31, MD5 new=e9d4bae2f6317df1b6bc3ca09923403b
smbpasswd: MD5 original=da170c92acd692978fcd7612944880d3, MD5 new=01d084326fc46a7ccd453d82b7839bb8
x86_energy_perf_policy: MD5 original=4852e7365edd42d1fae7a397810e6842, MD5 new=83ce207d4cfd9b5b1fcb5a967ef19eb6
[msf9542 Lab3]$ 
```

The only requirements for your code are that the script accepts the file names as command line arguments and it compares the message digests between md5_new.txt and md5_original.txt.

For this task, submit the Lab3_Task2.py script as part of your Lab3.zip file.