Lab Questions: Lab Session 9

Deadline: 25.10.2017 11:59pm SGT

Complete all assignments below. For those questions that are marked with an asterisk *, i.e. **Questions 8 and 9**, create the script/function files as requested. Once you are done with it, submit the file(s) via `iNTU`. Remember to put plenty of comments for all assignments, as this helps us to better understand your program (which might give you higher marks).

Important!!! Make sure your scripts work properly, as we give 0 marks otherwise. Please name the scripts according to the requirements, and upload each file separately and not in a Zip file or similar. The submission system closes at the deadline. Hence after that, you will get no marks for your solution.

1. Create three lists $L1$, $L2$ and $L3$, filled with random data (it does not matter what data you actually put in the lists). Open a file named `my_data.txt` using `w` mode and save in it the contents of $L1$ (you can open the file with a text editor to see the result). Then, open again the file `my_data.txt` using `w` mode and save in it the contents of $L2$ (you can open again the file with a text editor to see the result). Finally, open the file `my_data.txt` using `a` mode and save in it the contents of $L3$ (you can open again the file with a text editor to see the result). Note that a list can be converted to a string using the `str()` built-in function.

   Solution:

   my_data.py

```
L1 = [4, 'Hi !', True, 5.6]
L2 = ['My name is Thomas', 3.14, 'p', (1,2,3)]
L3 = [7, 'Hello', [4,5,6]]

my_file = open('my_data.txt','w')
my_file.write(str(L1))
my_file.close()

my_file = open('my_data.txt','w')
my_file.write(str(L2))
my_file.close()

my_file = open('my_data.txt','a')
my_file.write(str(L3))
my_file.close()
```

2. Download the file `transport.txt` from iNTU Learn and open it in a text file editor to check how it is organised (it contains the average daily public transport ridership in Singapore for each year from 1995 to 2016, for MRT, LRT, Bus and Taxi). Write a script `transport.py` that will load the data from that file and print in the console the average daily public transport ridership for the entire period 1995 to 2016 for all four transports: MRT, LRT, Bus and Taxi.

For example, the output of the script can look like this:

```
The average MRT ridership is 1638954.55
The average LRT ridership is 72181.82
The average Bus ridership is 3228545.45
The average Taxi ridership is 671727.27
```

Solution:

transport.py

```
# load the data from the file split into words
lines = [line.split() for line in open('transport.txt')]

# count the total number of average daily public transport ridership
MRT = LRT = Bus = Taxi = 0
for line in lines:
    if line[1] == 'MRT':
        MRT += int(line[2])
    elif line[1] == 'LRT':
        LRT += int(line[2])
    elif line[1] == 'Bus':
        Bus += int(line[2])
    elif line[1] == 'Taxi':
        Taxi += int(line[2])

years = (len(lines)-1)/4

# print the average for each transport
print('The average MRT ridership is %.2f' % (MRT/years))
print('The average LRT ridership is %.2f' % (LRT/years))
print('The average Bus ridership is %.2f' % (Bus/years))
print('The average Taxi ridership is %.2f' % (Taxi/years))
```

3. What will be the output of the following programs ?

(a)

```
try:
    print("a")
except:
    print("b")
else:
    print("c")
finally:
    print("d")
```

Solution:
The output will be

```
a
c
d
```

Indeed, no exception will be raised as there is no problem with the statement

in the `try` block. Thus, `print(b)` will not be executed, but `print(c)` will be executed (since the `else` block is executed when no exception is triggered) and `print(d)` will be executed (since the `finally` block is always executed regardless if an exception is triggered or not).

(b)

```
try:
    print("%d",5/0)
except:
    print("b")
else:
    print("c")
finally:
    print("d")
```

Solution:
The output will be
`b`
`d`

Indeed, an exception will be raised as there is a division by zero in the `try` block. Thus, `print(b)` will be executed, but `print(c)` will not be executed (since the `else` block is executed when no exception is triggered) and `print(d)` will be executed (since the `finally` block is always executed regardless if an exception is triggered or not).

4. Write a Python program `copy_no_space.py` that will ask the user to input a source file name and a target file name, and that will copy the content of the source file to the target file, while removing all space characters (space, tab, newline, return, etc.). An error check must be done using `try/catch` in case the source file input by the user does not exist (in which case you can print a message explaining that the source file does not exist).

Solution:

copy_no_space.py

```
# ask the user for a source and target filename
source = input('What is your source file name ? ')
target = input('What is your target file name ? ')

try:
    # extract the data from the source file, and split it into words
    lines = [line.split() for line in open(source)]
except FileNotFoundError:
    print('This source file does not exist !')
else:
    # put back the data in target file, without the spaces
    my_target_file = open(target,'w')
    for line in lines:
        for word in line:
            my_target_file.write(word)
    my_target_file.close()
```

5. Write a Python program `count_occurences.py` that will ask the user to input a source file name and a character, and that will count how many times the character is present in that file. An error check must be done using `try/catch` in case the source file input by the user does not exist (in which case you can print a message explaining that the source file does not exist).

Solution:

count_occurences.py

```
# ask the user for a source and target filename
source = input('What is your source file name ? ')
target_char = input('What character would you like to check ? ')

try:
    # open the source file
    my_source_file = open(source,'r')
except FileNotFoundError:
    print('This source file does not exist !')
else:
    # extract the data from the source file
    my_string = my_source_file.read()
    my_source_file.close()

    # count and print how many times the character appears
    count = 0
    for my_char in my_string:
        if my_char == target_char:
            count+=1
    print('There are %d characters %s in %s' % (count,target_char,
        source))
```

6. Debug the following programs (you can use any technique that you want for that), it should execute without errors and it should have the functionality described. You can download the Python files on iNTU Learn.

   (a) We would like to generate 100 random integers in [0,10] and print them in the console.

debug1.py

```
import random

L = [random.randint(0,10) for i in range(100)]

for i in L
    print("the number is %d",L[i])
```

Solution:
1. The colon is missing in the for loop;
2. `i` iterates though all the element of the list `L`, so in order to range through all of them we simply use `i` and not `L[i]`;
3. the syntax in the print function to provide the value to the place-holder `%d` is wrong. Overall, the script corrected looks like this.

```
import random

L = [random.randint(0,10) for i in range(100)]

for i in L:
    print("the number is %d" % (i))
```

(b) We would like to ask the user to input positive integers, and to continue asking him as long as he didn't 10 of them (in which case we quit). We would also like to give him the opportunity to stop the program by simply entering the special number -10. You can assume that the user only inserts integers.

debug2.py

```
count = 0
while count < 10:
    my_int = -1
    while my_int<0:
        my_int = int(input('Insert a positive integer: '))
        if my_int==-10:
            break

    count+=1
    print('You have entered %d positive integers' % (count))
```

Solution:
One can observe that with this program when the user enters -10 it doesn't quit. This is due to the fact that the `break` is not enough to exit the two `while` loops. Indeed, a `break` only terminates the current inner loop he belongs to, but it does not quit from potential outer loops. Thus, one way to correct this is to use another `break` outside the inner `while` loop.

```
count = 0
while count < 10:
    my_int = -1
    while my_int<0:
        my_int = int(input('Insert a positive integer: '))
        if my_int==-10:
            print('exiting ...')
            break

    if my_int==-10:
        print('exiting ...')
        break
    count+=1
    print('You have entered %d positive integers' % (count))
```

(c) We would like to generate a list of 100 random integers in [0,10] and ask the user which element he would like to see. The program will then print this element in the console. You can assume that the user only inserts integers.

debug3.py

```
import random
```

```
L = [random.randint(0,10) for i in range(100)]

my_element = input('which element would you like to print ? ')
print('The element is %d' % (L[my_element]))
```

Solution:
One can observe that the program is reaching an error when the user enters an integer. The reason is because the `input` function outputs a string and not an integer. Thus, one can't directly use `my_element` as an index to access elements in L. Instead, it should first be type-casted into an integer. Another bug with this program is that the user might enter an index bigger than the number of elements in the list, in which case the program will return an error. Thus, an error check must be done before trying to access the index provided by the user.

```
import random

L = [random.randint(0,10) for i in range(100)]

my_element = int(input('which element would you like to print ? ')
    )
if 0 < my_element < 100:
    print('The element is %d' % (L[my_element]))
else:
    print('The element is out of the list range')
```

7. Download the file `temp_pressure.txt` from iNTU Learn. It contains temperature and air pressure measurements: one measurement per line, the first element of each line representing the temperature in degrees Celsius, and the second element representing the air pressure in millibar. Unfortunately, the researcher that generated this file is not very reliable, and the file contains some errors. Write a script `temp_pressure.py` that will read the data from the file, compute and print the overall temperature and air pressure average. Use appropriate `try/catch` block to handle the file errors: if either the temperature or the air pressure measurement contains an error, print in the console the file line where this error is located. Moreover the entire measurement of the erroneous lines should be discarded and not used for the average computations).

Solution:

temp_pressure.py

```
# load the data from the file split into words
lines = [line.split() for line in open('temp_pressure.txt')]

# try to read the temperature and pressure for each mounth.
# ignore the line if the data is wrong
temperature = pressure = error_counter = 0
for (i,line) in enumerate(lines):
    try:
        temp_read = float(line[0])
        pressure_read = float(line[1])
    except IndexError:
```

```
        print('data error: value missing at line %d' % (i))
        error_counter +=1
    except ValueError:
        print('data error: not a float at line %d' % (i))
        error_counter +=1
    except Exception as e:
        print('data error at line %d: %s' % (i,e))
        error_counter +=1
    else:
        temperature += temp_read
        pressure += pressure_read

months = len(lines)-error_counter

# print the average temperature and pressure
print('The average temperature is %.2f' % (temperature/months))
print('The average pressure is %.2f' % (pressure/months))
```

8. * Download from iNTU Learn the file `pokemons.txt` that contains a list of Pokemons, their index (from 1 to 718 included), and their type. The file is sorted according to the Pokemons names. Write a Python script <`YourMatricNo`>`_Lab9_catching_pokemons.py` that will read the content of this file (the potential non-existence of the file must be handled with an appropriate try/catch block) and that will create another file `pokemons_new.txt` that contains the same data, but this time sorted according to the pokemons indexes.

   Solution:

   catching_pokemons.py

```
# load the data from the file split into words
try:
    lines = [line.split() for line in open('pokemons.txt')]
except FileNotFoundError:
    print('Sorry, the file pokemons.txt does not exist\n')
else:
    # open and create the new pokemon file
    my_new_file = open('pokemons_new.txt','w')

    # for all possible index values up to 999
    for i in range(1,719):
        # find in which line of the original file the index i appears
        for line in lines:
            # when we found the index in the original file
            # we save the data in the new file
            if (int(line[1])==i):
                my_new_file.write('%d %s %s\n' % (i,line[0],line[2]))

    my_new_file.close()
```

9. * A student was requested to produce a function named `sum_and_product` that takes as input a matrix of unknown size (implemented as a Numpy array), and that outputs two values: the sum of all the elements of the matrix, and the product of all the elements in the matrix. The student came up with the following code (available on iNTU Learn):

sum_and_prod.py

```
import numpy as np

def sum_and_prod(my_mat):
    (rows,columns) = my_mat.shape
    for r in rows
        my_sum = 0
        my_product = 0
        for c in columns
            my_sum += my_mat[r,c]
            my_product *= my_mat[r,c]

    return (my_sum,my_product)
```

His code contains syntax as well as logical errors. Debug this program, in order to ensure that it executes properly and that it actually does what it is supposed to do. Copy the corrected code in a file <YourMatricNo>_Lab9_sum_and_prod.py.

Solution:
First, when interpreting this function, Python interpreter will directly tell us that there is some syntax errors. Indeed, the colons are missing for both the `for` loops. Moreover, `rows` and `columns` are integer values, which are not iterable. Thus, you can't write `for r in rows`, you have to use instead the `range` function: `for r in range(rows)`.

Once these issues corrected, we can test the function with some random matrix as input and we can see that it doesn't work properly: the sum and product computed are wrong. This is due to two reasons. First, the initialization of the running sum and running product variables are placed inside the first `for` loop, thus these values are reset everytime we are moving to a new row in this loop. This initialisation should actually be located before this `for` loop. Finally, the initialisation of a running product should be done with 1, and not a 0. Overall, the corrected code looks like this:

sum_and_prod_sol.py

```
import numpy as np

def sum_and_prod(my_mat):
    (rows,columns) = my_mat.shape
    my_sum = 0
    my_product = 1
    for r in range(rows):
        for c in range(columns):
            my_sum += my_mat[r,c]
            my_product *= my_mat[r,c]

    return (my_sum,my_product)
```