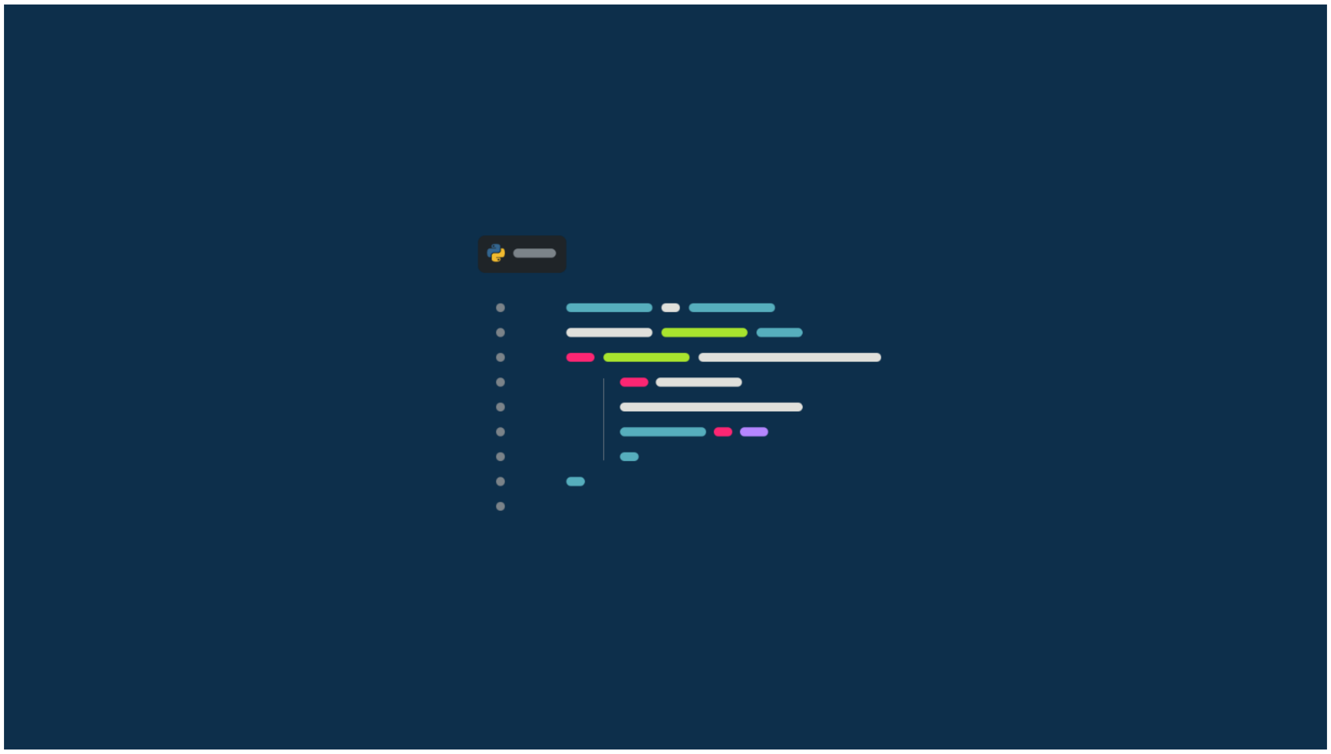
LBYCPA1

Programming Logic and Design Laboratory



Laboratory Module 9

Files and Exceptions

By

John Carlo Theo S. Dela Cruz || LBYCPA1 – EQ1

INTRODUCTION

The ninth module of this course will mainly focus on Files and Exceptions, particularly importing, writing external files from python code and exception handling. Defining the concepts of each function or feature will be the first step with every module; In python file objects or data can be drawn from an external file or can write a data and export it into an external file. This feature can be handy in terms of handling whenever a programmer is handling a huge amount of data to be used in their code and using file storage can maximize the space. Just like how the module demonstrated how can the program or language interact with the said external file. Exceptions is defined as a statement will show an error even if it is syntactically correct, or vice-versa. It is handled using a try statement and inside will be the critical operation which can raise an exception, while the code that handles the exceptions is written in the except clause. The rule of thumb of every laboratory report is to provide opportunity for learning and provide a strong foundation in programming, molded by experience. Through this module, the students will reflect on how data analysis and organization will be a big role in programming. Despite the hindrances we face in our reality today, we utilize and appreciate the utilities that is provided by the world wide web to provide us students various information about programming. Through the process of programming alongside with the method of planning; algorithms, pseudocodes, and flowcharts that provides process on how the program executes. This laboratory report will be helpful for aspiring web developers who are studying Python right now.

What do you think are the main objectives for this module? (Enumerate as many as you can.)

(a) Objectives

1. To perform reading from and writing to a file
2. To understand the different types of exceptions
3. To handle exceptions for developing robust programs
4. To utilize every function and syntax provided in the module.

What are the materials used for this module?

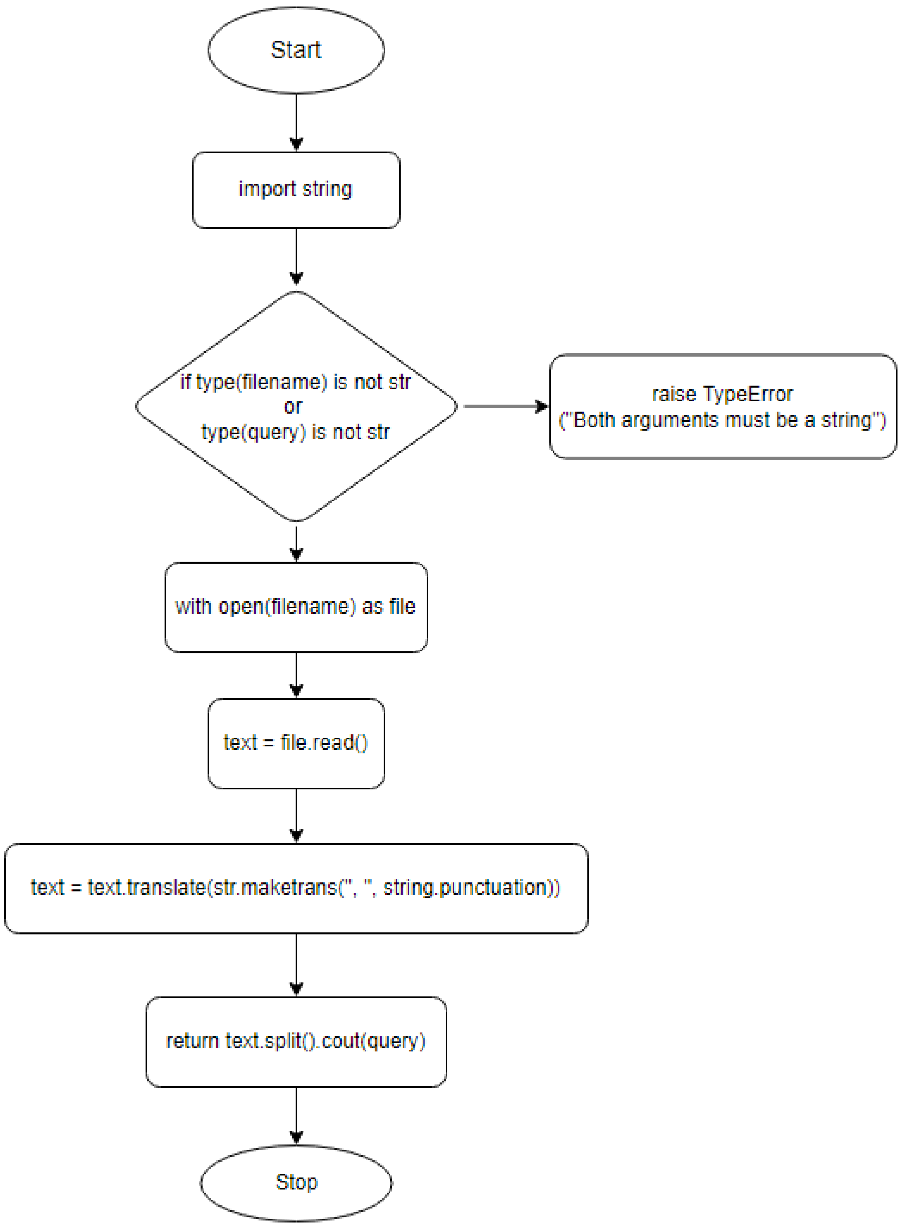
(b) Materials and Tools

1. Instructor's lecture notes
2. Jupyter Notebook
3. Flowchart Software (Diagrams.net)
4. Snipping Tool
5. Pycharm.edu
6. Google Browser

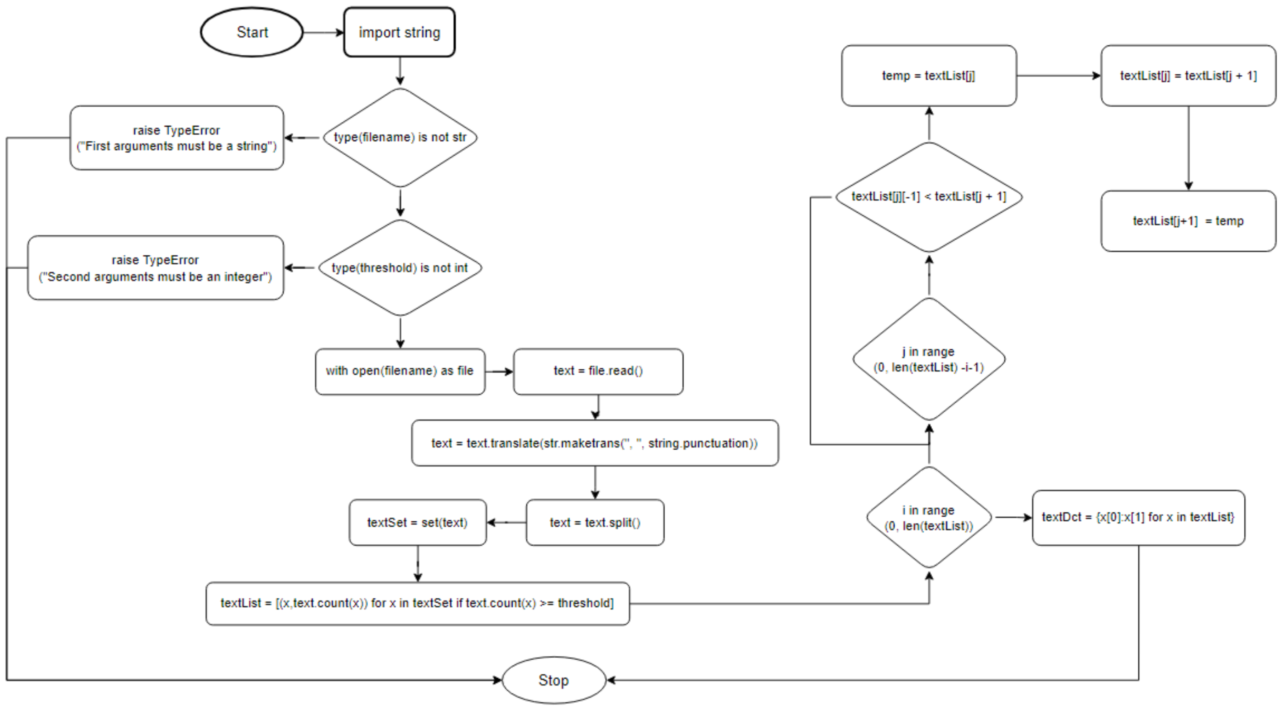
PROCEDURES (*Individual*) / EXPERIMENTAL PLAN

In every experimental plan of every Laboratory Report we always place our initial plans before performing executing our codes, in programming we always use Algorithms and flowcharts, as a visual representation of how we can show the separate steps of each process in a sequential manner. The overview.

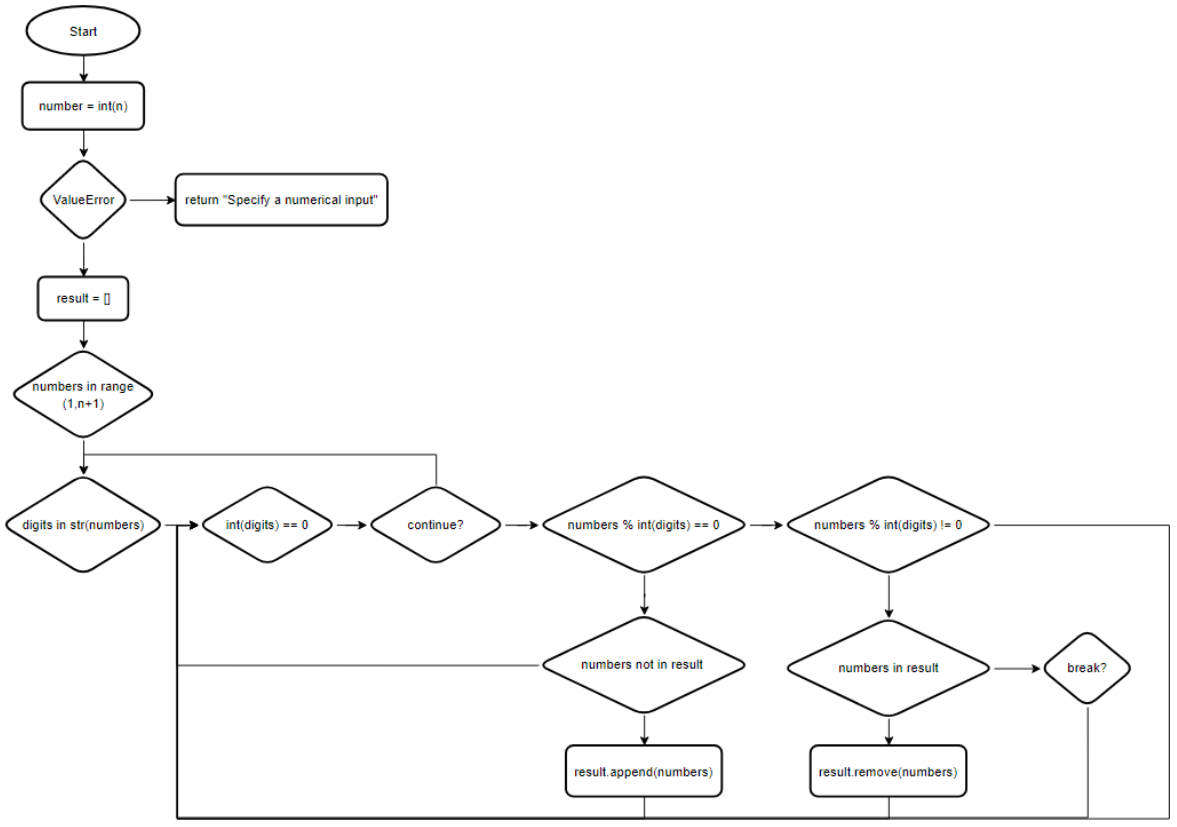
1. In Familiarization Exercise 1, its required output should be a word counter, in which the data or the string will be from the file named: alice.txt in which the program will count the numbers of a specific word, additionally raising a type error that verifies the query's data type.


Start
import string
if type(filename) is not str
or
type(query) is not str
raise TypeError
("Both arguments must be a string")
with open(filename) as file
text = file.read()
text = text.translate(str.maketrans('', '', string.punctuation))
return text.split().cout(query)
Stop

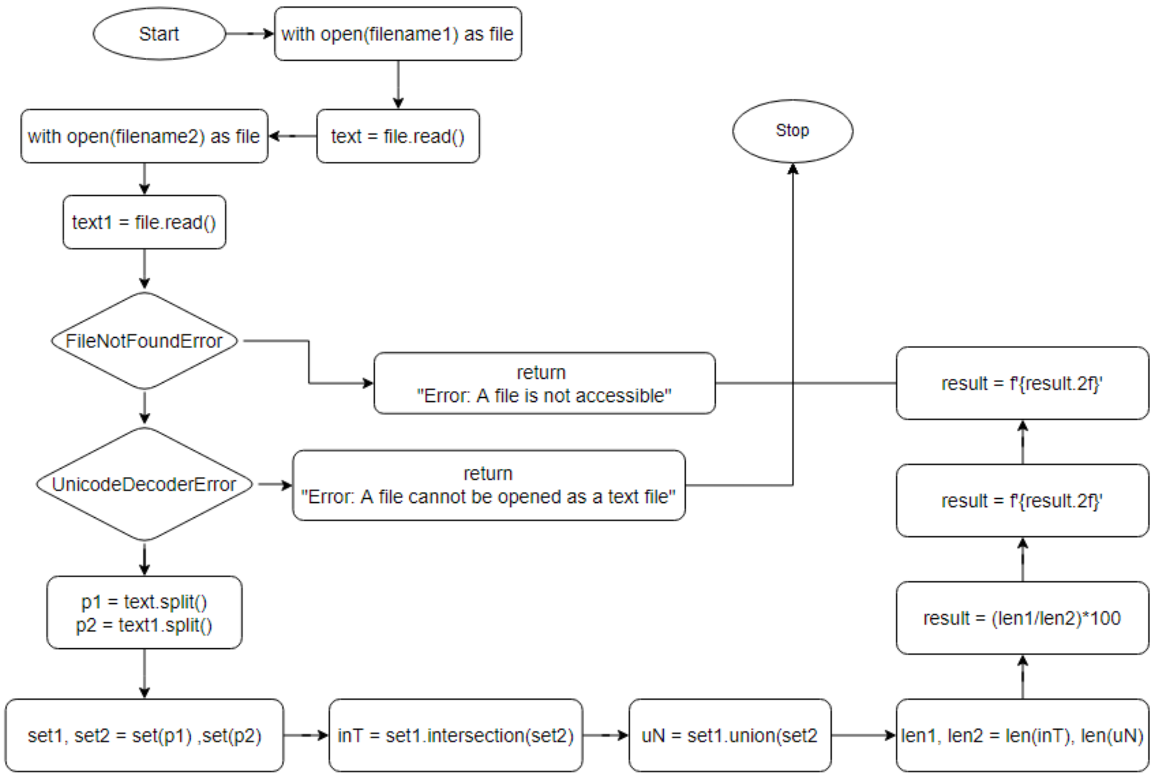
2. In Familiarization Exercise 2, the procedure in exercise 1 is somehow similar because we’ll be getting the string and verifying the data type by raising a type error. In this case we’ll get each words using set and counting those set of words, lastly returning them into a dictionary.


Start
import string
type(filename) is not str
raise TypeError ("First arguments must be a string")
type(threshold) is not int
raise TypeError ("Second arguments must be an integer")
with open(filename) as file
text = file.read()
text = text.translate(str.maketrans('', '', string.punctuation))
text = text.split()
textSet = set(text)
textList = [(x,text.count(x)) for x in textSet if text.count(x) >= threshold]
i in range (0, len(textList))
j in range (0, len(textList) -i-1)
textList[j][-1] < textList[j + 1]
temp = textList[j]
textList[j] = textList[j + 1]
textList[j+1] = temp
textDct = {x[0]:x[1] for x in textList}
Stop

3. In Familiarization Exercise 3, first is raising a value error that verifies the user's input should be in integer form, or else the program would return an error message. The function's output would be a list of numbers from 1 up to a given number to check if the value is divisible by the input and after the condition is satisfied, it will now check if that value will have no remainder.



4. In Familiarization Exercise 4, Intersection over Union will be computed as the intersection between two sets over their union to determine the similarity between two programs that works like "Plagiarism Checker". To check their similarity, first we'll use the try-except clause to open the files at the same time will return an error message if the file is not accessible or cannot be opened. Then else will be the result if the files are readable then the data from each respective files will be a string, then using the creating a variable the transforms the intersection and union between 2 sets, getting the length of each set and dividing it and multiplying by 100 to transform it into a floating point, lastly rounding up the value into 2 decimal places.

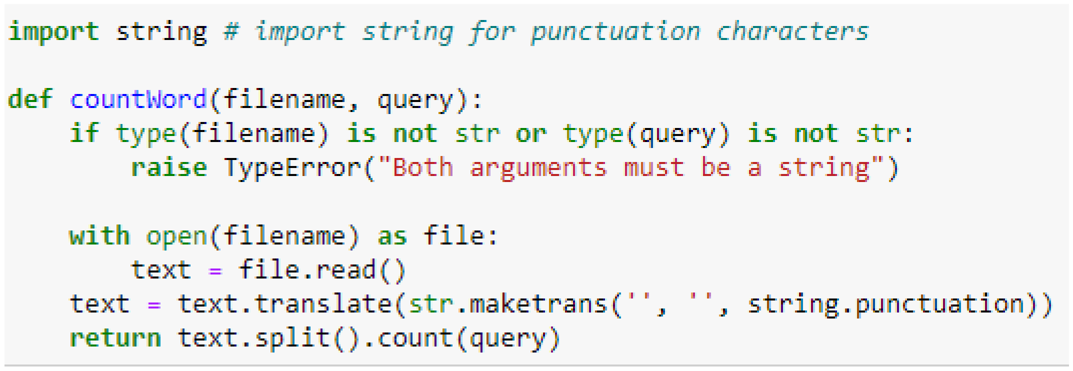

Start
with open(filename1) as file
text = file.read()
with open(filename2) as file
text1 = file.read()
FileNotFoundError
return "Error: A file is not accessible"
UnicodeDecoderError
return "Error: A file cannot be opened as a text file"
Stop
p1 = text.split()
p2 = text1.split()
set1, set2 = set(p1) ,set(p2)
inT = set1.intersection(set2)
uN = set1.union(set2
len1, len2 = len(inT), len(uN)
result = (len1/len2)*100
result = f'{result.2f}'
result = f'{result.2f}'

The Introduction together with individual Procedures and plan comprises the Experimental Plan and Conducting Experiment/ Activity criteria in the Final Laboratory Report Rubric:

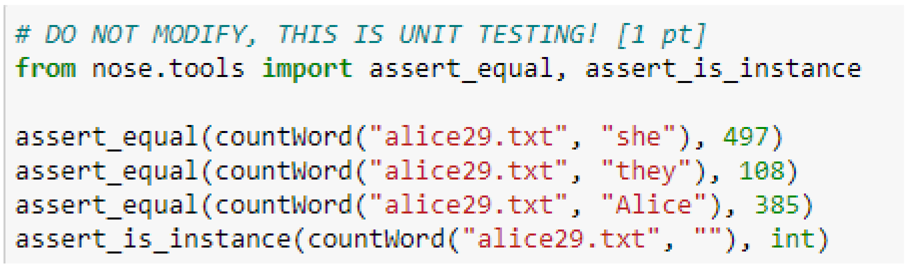
CRITERIA	**EXEMPLARY (90-100)**	**SATISFACTOR Y (80-89)**	**DEVELOPING (70-79)**	**BEGINNING (below 70)**	**WEIGHT**
Experimental Plan (Flowchart/ Algorithm) ***(SO-PI: B1)***	Experimental plan has supporting details and diagram/algorithm that is stated and well explained	Experimental plan has supporting details and diagram/algorithm that is stated but not explained	Experimental plan is vague or brief. It has supporting details and doesn't have diagram/algorith m	No experimental plan presented	30%
Conducting Experiment/ Activity ***(SO-PI: B1)*** ***(SO-PI: K1)***	Objective and Materials used (modern engineering tools, software, and instruments/equipm ent) are identified. Steps are easy to follow for conducting and experiment/activity.	Objective is not stated. Materials used (modern engineering tools, software, and instruments/equip ment) are identified. Steps are easy to follow for conducting and experiment/activit y	Does not provide enough information to conduct an experiment/activ ity	No Objective, Materials used (modern engineering tools, software, and instruments/equi pment), and steps for conducting experiment/activ ity provided.	20%

RESULTS AND DISCUSSION/COMPUTATIONS (*Include the program output screenshots, and discussions per problem solution*)

1. Familiarization Exercise 1 Result:



```python
import string # import string for punctuation characters

def countWord(filename, query):
    if type(filename) is not str or type(query) is not str:
        raise TypeError("Both arguments must be a string")

    with open(filename) as file:
        text = file.read()
    text = text.translate(str.maketrans('', '', string.punctuation))
    return text.split().count(query)
```

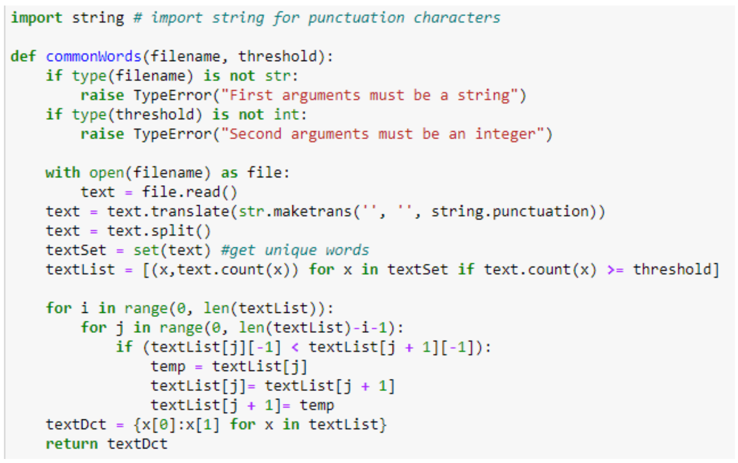


```python
# DO NOT MODIFY, THIS IS UNIT TESTING! [1 pt]
from nose.tools import assert_equal, assert_is_instance

assert_equal(countWord("alice29.txt", "she"), 497)
assert_equal(countWord("alice29.txt", "they"), 108)
assert_equal(countWord("alice29.txt", "Alice"), 385)
assert_is_instance(countWord("alice29.txt", ""), int)
```

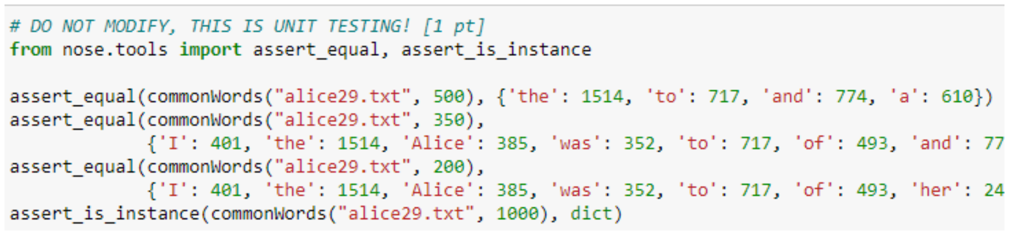
Explanation:

The program's required output would be the counted words according to the query input, First the declaration of the raise type error, and will verify that the required input should be a string (filename) or int (query). After creating an error exception, we now proceed on opening and reading the file that will be used by using translate and make trans methods to remove the strings. Lastly, return the variable text and embed it with split the strings and counting the query.

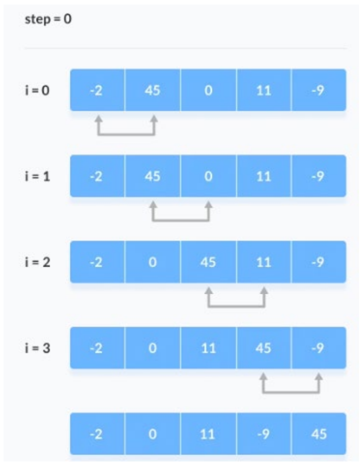
2. Familiarization Exercise 2 Result:



```python
import string # import string for punctuation characters

def commonWords(filename, threshold):
    if type(filename) is not str:
        raise TypeError("First arguments must be a string")
    if type(threshold) is not int:
        raise TypeError("Second arguments must be an integer")

    with open(filename) as file:
        text = file.read()
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = text.split()
    textSet = set(text) #get unique words
    textList = [(x,text.count(x)) for x in textSet if text.count(x) >= threshold]

    for i in range(0, len(textList)):
        for j in range(0, len(textList)-i-1):
            if (textList[j][-1] < textList[j + 1][-1]):
                temp = textList[j]
                textList[j]= textList[j + 1]
                textList[j + 1]= temp
    textDct = {x[0]:x[1] for x in textList}
    return textDct
```



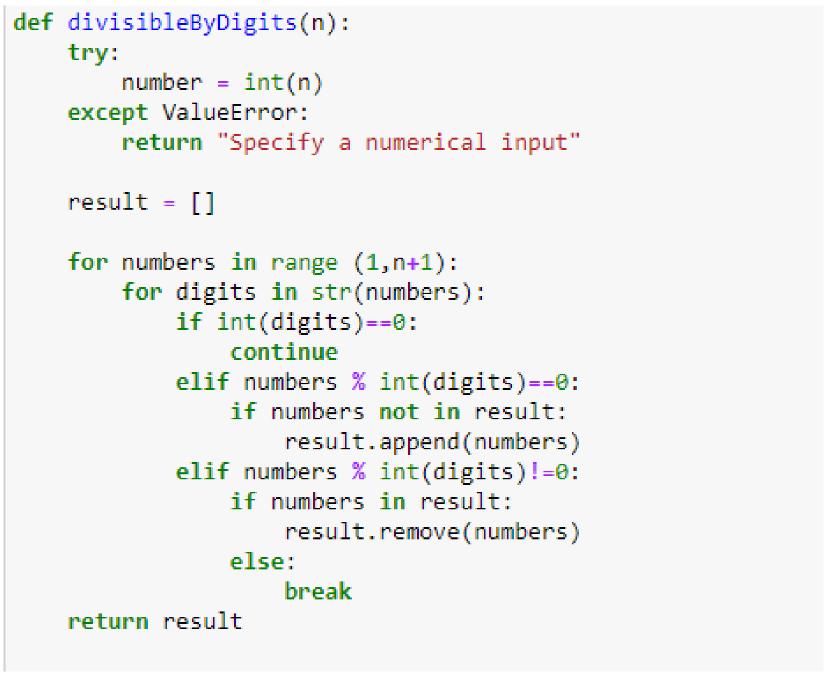
```python
# DO NOT MODIFY, THIS IS UNIT TESTING! [1 pt]
from nose.tools import assert_equal, assert_is_instance

assert_equal(commonWords("alice29.txt", 500), {'the': 1514, 'to': 717, 'and': 774, 'a': 610})
assert_equal(commonWords("alice29.txt", 350),
             {'I': 401, 'the': 1514, 'Alice': 385, 'was': 352, 'to': 717, 'of': 493, 'and': 77
assert_equal(commonWords("alice29.txt", 200),
             {'I': 401, 'the': 1514, 'Alice': 385, 'was': 352, 'to': 717, 'of': 493, 'her': 24
assert_is_instance(commonWords("alice29.txt", 1000), dict)
```



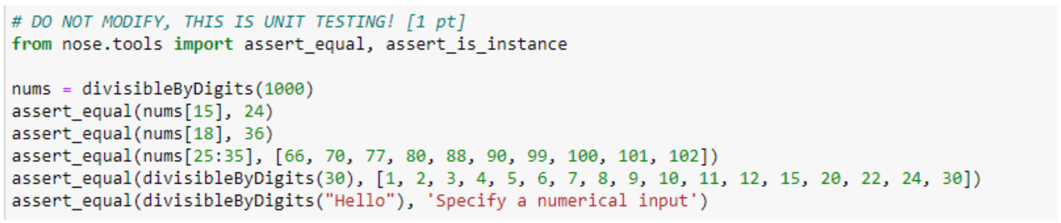
Explanation:

The program's required output would be a dictionary of words with the respective key values; the key would be the word, and the value would be the number of words in the text file. We'll follow the same method with the 1st exercise, and after splitting the text variable, we get the unique words in the file by using a set, because the set does not accept repeating variables. I created a list comprehension in which in every word in the text Set, it will append the variable (the word, value (count)) if the text count will be greater than or equal to the threshold, and I followed a method called Bubble sorting in which it will compare 2 adjacent elements and swaps them until they are not in the right order. After sorting, I turned the list into a dictionary since that is the required output and returned the variable.

3. Familiarization Exercise 3 Result:



```
def divisibleByDigits(n):
    try:
        number = int(n)
    except ValueError:
        return "Specify a numerical input"

    result = []

    for numbers in range (1,n+1):
        for digits in str(numbers):
            if int(digits)==0:
                continue
            elif numbers % int(digits)==0:
                if numbers not in result:
                    result.append(numbers)
            elif numbers % int(digits)!=0:
                if numbers in result:
                    result.remove(numbers)
                else:
                    break
    return result
```

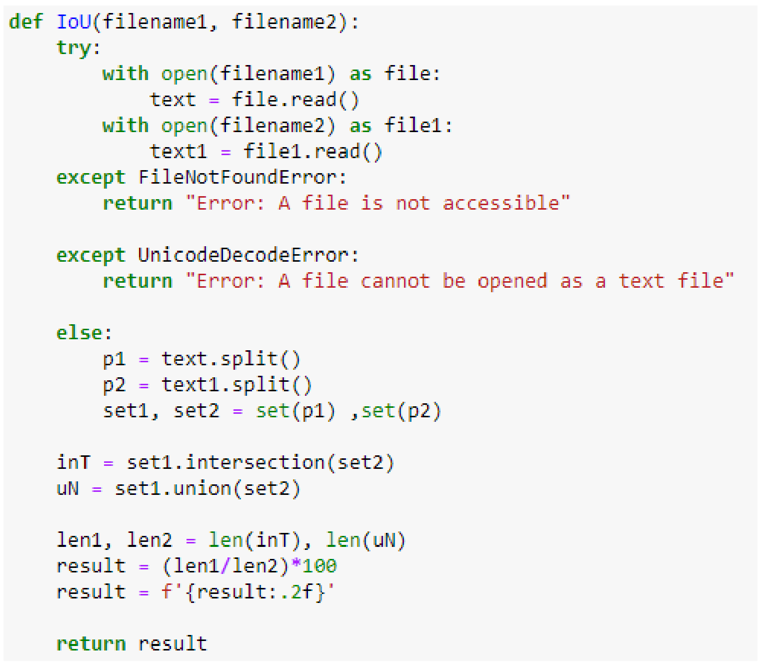


```
# DO NOT MODIFY, THIS IS UNIT TESTING! [1 pt]
from nose.tools import assert_equal, assert_is_instance

nums = divisibleByDigits(1000)
assert_equal(nums[15], 24)
assert_equal(nums[18], 36)
assert_equal(nums[25:35], [66, 70, 77, 80, 88, 90, 99, 100, 101, 102])
assert_equal(divisibleByDigits(30), [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 20, 22, 24, 30])
assert_equal(divisibleByDigits("Hello"), 'Specify a numerical input')
```

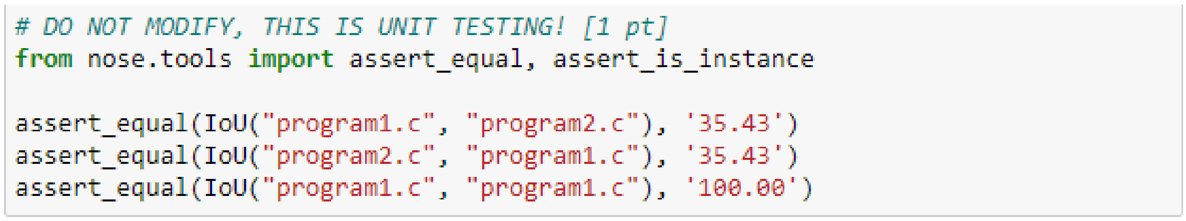
Explanation:

This program's required output would be Digit which will be divisible by the inputted number, first step is verifying the input using try-except clauses to verify the inputted number if it is an integer form or else it will return a value error. Declare an empty list and create 2 loops that verify if the digit is divisible and if it is divisible, it will check if it has no remainder, or else it will be removed from the list.

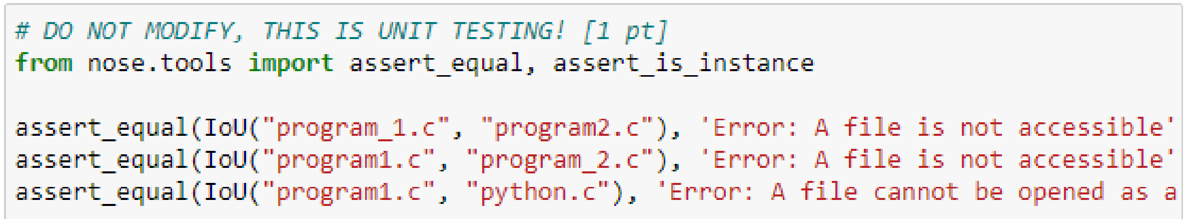
4. Familiarization Exercise 4 Result:



```python
def IoU(filename1, filename2):
    try:
        with open(filename1) as file:
            text = file.read()
        with open(filename2) as file1:
            text1 = file1.read()
    except FileNotFoundError:
        return "Error: A file is not accessible"

    except UnicodeDecodeError:
        return "Error: A file cannot be opened as a text file"

    else:
        p1 = text.split()
        p2 = text1.split()
        set1, set2 = set(p1) ,set(p2)

    inT = set1.intersection(set2)
    uN = set1.union(set2)

    len1, len2 = len(inT), len(uN)
    result = (len1/len2)*100
    result = f'{result:.2f}'

    return result
```



```python
# DO NOT MODIFY, THIS IS UNIT TESTING! [1 pt]
from nose.tools import assert_equal, assert_is_instance

assert_equal(IoU("program1.c", "program2.c"), '35.43')
assert_equal(IoU("program2.c", "program1.c"), '35.43')
assert_equal(IoU("program1.c", "program1.c"), '100.00')
```



```python
# DO NOT MODIFY, THIS IS UNIT TESTING! [1 pt]
from nose.tools import assert_equal, assert_is_instance

assert_equal(IoU("program_1.c", "program2.c"), 'Error: A file is not accessible'
assert_equal(IoU("program1.c", "program_2.c"), 'Error: A file is not accessible'
assert_equal(IoU("program1.c", "python.c"), 'Error: A file cannot be opened as a
```

Explanation:

The last program's required output would be like a "Plagiarism Checker" similar with Turnitin, and this function would first try to open the files and read them, except it will return Errors, and if the file is accessible then first, the strings would be split and turn into set, because we will check the unique data inside the file, and create a variable that will get the value of the intersection and union of the two files. After setting a variable, we now proceed on dividing the variables intersection over union and multiply it by 100 to set it into a floating point. Lastly rounding it off in to 2 decimal places.

The Results and Discussions constitutes the data criteria in the Lab Report Evaluation Rubric:

Codes/Data/ Program	Data is well utilized in the program. Program codes are easy to read. Program output has no error. Questions are answered completely and correctly	Data is somewhat utilized in the program. Program code are easy to read. Program output has an output but logically incorrect. Some questions are answered completely and correctly	Data is not utilized in the program. It has a missing significant code/syntax in the program.	No program presented	30%

CONCLUSION:

This 9th Module of LBYCPA1 mainly focuses on Files and Exceptions. This module taught me the basic things of how to read, write with an external file with the Python language. This module made me realize how important is file handling in many languages, because it can store data in the form of a text and readable between humans and computers. As well as exception handling, and this function would allow us programmers separate error-handling from normal code, and us students would recognize the raise error provided by our professor when we are accomplishing our previous modules. This module helped me to improvise and utilize the function of files and exceptions, as well as using my knowledge from the previous modules and I can say that I am in an improving state in coding. One challenge I faced is distractions and fear of completing the preliminary report, but I quickly resolved it by practicing and experimenting the functions in my own pace. I tried to understand each context in the module and recreate them and find exercises online with the similar use of file handling and exception handling. Practice makes perfect and everyone must start somewhere.

Grammar, logical presentation, and format ***(SO-PI: G1)***	The report was grammatically correct, logically presented and used the required format.	The report had minimal grammatical errors and somewhat presented logically. The required format was used.	The report had a lot of grammatical errors and not logically presented; the required format was barely used.	The report had a lot of grammatical errors, was not logically presented and the required format was not used.	20%

REFERENCES (*Enumerate references in APA format*)

1. W3school (n.d.). Retrieved from: https://www.w3schools.com/python/python_file_handling.asp
2. Programiz (n.d.). Retrieved from: https://www.programiz.com/dsa/bubble-sort

APPENDIX (*Attach all the source codes here per problem category*)

1. Familiarization Exercise 1:

```
import string # import string for punctuation characters

def countWord(filename, query):
    if type(filename) is not str or type(query) is not str:
        raise TypeError("Both arguments must be a string")

    with open(filename) as file:
        text = file.read()
    text = text.translate(str.maketrans('', '', string.punctuation))
    return text.split().count(query)
```

2. Familiarization Exercise 2:

```
import string # import string for punctuation characters

def commonWords(filename, threshold):
    if type(filename) is not str:
        raise TypeError("First arguments must be a string")
    if type(threshold) is not int:
        raise TypeError("Second arguments must be an integer")

    with open(filename) as file:
        text = file.read()
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = text.split()
    textSet = set(text) #get unique words
    textList = [(x,text.count(x)) for x in textSet if text.count(x) >=
threshold]

    for i in range(0, len(textList)):
        for j in range(0, len(textList)-i-1):
            if (textList[j][-1] < textList[j + 1][-1]):
                temp = textList[j]
                textList[j]= textList[j + 1]
                textList[j + 1]= temp
    textDct = {x[0]:x[1] for x in textList}
    return textDct
```

3. Familiarization Exercise 3:

```
def divisibleByDigits(n):
    try:
        number = int(n)
    except ValueError:
        return "Specify a numerical input"

    result = []

    for numbers in range (1,n+1):
        for digits in str(numbers):
            if int(digits)==0:
                continue
            elif numbers % int(digits)==0:
                if numbers not in result:
                    result.append(numbers)
            elif numbers % int(digits)!=0:
                if numbers in result:
                    result.remove(numbers)
                else:
                    break

    return result
```

4. Familiarization Exercise 4:

```
def IoU(filename1, filename2):
    try:
        with open(filename1) as file:
            text = file.read()
        with open(filename2) as file1:
            text1 = file1.read()
    except FileNotFoundError:
        return "Error: A file is not accessible"

    except UnicodeDecodeError:
        return "Error: A file cannot be opened as a text file"

    else:
        p1 = text.split()
        p2 = text1.split()
        set1, set2 = set(p1) ,set(p2)

    inT = set1.intersection(set2)
    uN = set1.union(set2)

    len1, len2 = len(inT), len(uN)
    result = (len1/len2)*100
    result = f'{result:.2f}'
    return result
```