File I/O and Exception Handling

CSE/IT 107L

NMT Department of Computer Science and Engineering

"The danger that computers will become like humans is not as big as the danger that humans will become like computers." ("Die Gefahr, dass der Computer so wird wie der Mensch ist nicht so groß, wie die Gefahr, dass der Mensch so wird wie der Computer.")

— Konrad Zuse

"First, solve the problem. Then, write the code."

— John Johnson

"I don't need to waste my time with a computer just because I am a computer scientist."

— Edsger W. Dijkstra

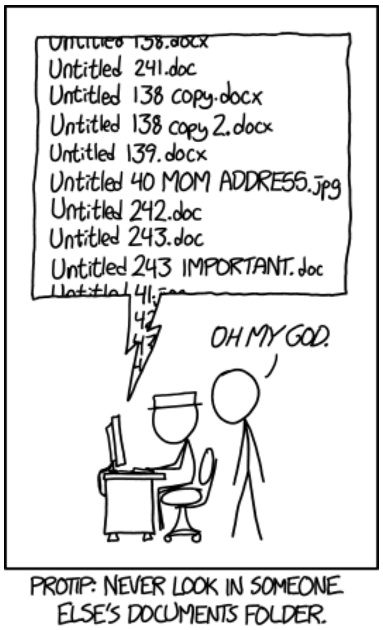

Figure 1: Documents `http://xkcd.com/1459`

Introduction

In this lab you will continue to practice opening and using files within your programs. You will also learn how error handling works in Python. You will learn how to gracefully handle exceptions within your programs using the `try` and `exccept` statements.

Contents

1 Exceptions

An *exception* is an error message in Python. When a certain operation encounters an error, it can *raise* an exception that is then passed on to the user. If your script is being executed as a program, exceptions cause it to print the exception and exit.

```
>>> 43 / 0
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ZeroDivisionError: division by zero
```

However, exceptions can be handled using the `try` and `except` statements. The `except` block is only executed if an exception is caught in the `try` block. Additionally, when an error is caught in the `try` block we stop executing commands in the `try` block and immediately jump to the `except` block.

The following example throws a division by zero error and prints "division by zero":

```
1 prime = 7
2 
3 try:
4     result = prime / 0
5     result = 7 * 42
6 except ZeroDivisionError as err:
7     print(err)
```

Since the error is thrown on line 5, line 6 is never executed. Also, we are only catching `ZeroDivisionError` exceptions, any other exceptions will remain uncaught.

If you are experimenting with code and want to know the name of an exception that is thrown, take a look at the error message:

```
>>> float('obviously not a float)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ValueError: could not convert string to float: 'obviously not a float'
```

The part highlighted in red here is the name of the error message, `ValueErrror`.

Hence, if you are getting user input and want to check whether it is the correct type, use a try-except block around the conversion:

```
try:
    x = int(input('Enter an integer number: '))
except ValueError:
    print('You did not enter an integer!')
else:
    print('You entered {}'.format(x))
```

Notice that you can use an `else` block to execute code if no exception was thrown.

Any `except` block that does not list built-in exceptions will catch all exceptions not listed in previous `except` blocks. For example, the following code will throw an error if the user enters anything but an integer:

```
try:
    x = int(input("Enter a number: "))
except:
    print("Unknown error.")
    raise
else:
    print("You entered: " + str(x))
```

The `raise` keyword causes a detailed trace and prints out additional information if an exception is encountered. The `else` block is optional and will be executed if no exceptions are thrown in the `try` block. There are many more built-in exceptions such as `IOError` that can be found here:

`https://docs.python.org/3/library/exceptions.html`

1.1 File I/O Exceptions

When working with files it is important to check for exceptions. These are only some of the exceptions that can be raised when doing file I/O.

- `FileNotFoundError` is raised when we try to open a file that. It should be handled so we can exit the program safely or recover rather than observe unexpected results.

```
>>> open("not_a_file.txt", "r")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'not_a_file.txt'
```

- `IsADirectoryError` is raised when trying open an directory as a file.

```
>>> open("directory", "w")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IsADirectoryError: [Errno 21] Is a directory: 'directory'
```

The `finally` block is where clean-up actions are performed and is always executed after leaving the `try` block. The following is a good example of using `with` and exception handling together to open and read a file.

```
filename = input('Enter a filename >>> ')

try:
    with open(filename, 'r') as f:
        sum = 0
        for line in f:
            sum += float(line)
except FileNotFoundError:
    print('File "{}" not found'.format(filename))
except IsADirectoryError:
    print('File "{}" is a directory'.format(filename))
except ValueError:
    print('File "{}" contains a line that'.format(filename)
        + 'cannot be converted to a float')
else:
    print('Sum of all numbers in file is {}'.format(sum))
finally:
    print('Goodbye.')
```

2 Exercises

Using `with`

If you manually close the file by calling `.close()`, the file may not be closed in exceptional circumstances. **Always use the `with` statement when opening files** in Python. See prelab9 for more detail.

Exercise 2.1 (word_count.py).
Write a program that takes in a filename and string as input. Then print how many times that string appears inside the chosen file. If the file does not exist, continue asking for a filename until one is given that exists. Use your source code file as test input.

Make sure to test files with that contain the same word multiple times.

```
$ python3 word_count.py
Please enter a filename: word_count.py
Please enter a string to search for: print
The string 'print' appears 102 times in the file 'word_count.py'
```

Exercise 2.2 (design, simplediff.py).
Before attempting to code this problem, create a file `design` that contains some analysis of how you think the problem will be solved. Examples include but are not limited to: a flowchart of events in the program, pseudocode, or a step-by-step process written in plain English. If you choose to scan your design, please make sure that it is legible.

Write a "diff" program that prints out the differences, line by line, of two files. Your program should ask the user for the names of two files, then print the differences between them. Follow the format output as shown below. Make sure to use proper error handling techniques for file I/O.

Assume all files have the same number of lines. The following output shows the output of the files `file1.txt` and `file2.txt`.

```
$ cat file1.txt
John goes to work.
Keith and Kyle went to the Ensiferum concert.
Alice ate an apple pie.
Joe cut down a tree.
The dog jumped over the wall.
$ cat file2.txt
John goes to work.
Coral went to a Kesha concert.
Alice ate an apple pie.
Joe planted a tree.
The dog jumped over the wall.
```

This is the result of running the script `simplediff.py` on the two files:

```
$ python simplediff.py
Enter file name 1 >>> file1.txt
Enter file name 2 >>> file2.txt

2c0
< Keith and Kyle went to the Ensiferum concert.
---
> Coral went to a Kesha concert.
4c4
< Joe cut down a tree.
---
> Joe planted a tree.
```

The `2c0` tag refers to where the difference occurred and can be read as line 2 character 0 (where lines are 1 indexed and characters are zero indexed). Compare the output of your script to that of the `diff` program by typing `diff file1.txt file2.txt` in your shell.

Exercise 2.3 (readscores.py).
Download the file `actsat.txt` provided on Canvas. It contains the following columns of whitespace-separated data:

Column 1	2-letter state/territory code (includes DC)
Column 2	% of graduates in that state taking the ACT
Column 3	Average composite ACT score
Column 4	% of graduates in that state taking the SAT
Column 5	Average SAT Math score
Column 6	Average SAT Reading score
Column 7	Average SAT Writing score

You must open this file and generate a list of dictionaries containing each row of data. Please use these keys for the dictionaries:

- `"state"`
- `"act_percent_taking"`
- `"act_average_score"`
- `"sat_percent_taking"`
- `"sat_average_math"`
- `"sat_average_reading"`
- `"sat_average_writing"`

For example, your code should process this two line file to form the following list of dictionaries.

```
AK         27  21.2     48       517          519                  491
AL       81       20.3    9    556             563       554
```

```
[{"state": "AK",
   "act_percent_taking":  27
   "act_average_score":   21.2
   "sat_percent_taking":  48
   "sat_average_math":    517
   "sat_average_reading": 519
   "sat_average_writing": 491
 },
 {"state": "AK",
   "act_percent_taking":  81
   "act_average_score":   20.3
   "sat_percent_taking":  9
   "sat_average_math":    556
   "sat_average_reading": 563
   "sat_average_writing": 554
}]
```

Exercise 2.4 (README).
For the following labs you will be required to create aand submit a README file.

A README file is a simple text file that contains basic information regarding the purpose and use of the software program, utility, or game. README files often contain instructions or additional help regarding the software it accompanies.

For this lab you must have the following in your README file::

- **Purpose**: Describes what each program does and what problem it solves. You can keep this breif.
- **Conclusion**:
  - What you learned during the lab? What new aspect of programming did you learn from the lab? Be analytical about what you learned.
  - Did pair programming help in solving the problems and completing the prelab? Did you have problems with your budy?
  - Did you work with your budy on the lab? What sections did you discuss? Did you and your budy carry out a review session with each others' code?
  - Did you encounter any problems? How did you fix those problems?
  - What improvements could you make?

The conclusion does not have to be lengthy, but it should be thorough. You will include the README file with your submissions.

Submitting

You should submit your code as a `.zip` that contains all the exercise files for this lab. The submitted file should be named

```
cse107_firstname_lastname_lab9.zip
```

Upload your .zip file to Canvas.

List of Files to Submit

Exercises start on page 4.