

Master of Engineering in Internetworking

INWK 6312

Programming for Internetworking Applications

Lab 4

Data Structure Selection, and Files

Table of Contents

[Chapter 1 - Prerequisites 3](#_Toc493629215)

[Chapter 2 - Introduction 3](#_Toc493629216)

[Chapter 3 - Objectives of the lab 3](#_Toc493629217)

[Chapter 4 - DataStructure Selection 3](#_Toc493629218)

[Key Topics 3](#_Toc493629219)

[Task 1 4](#_Toc493629220)

[Task 2: 4](#_Toc493629221)

[Task 3 5](#_Toc493629222)

[Task 4 5](#_Toc493629223)

[Task 5 5](#_Toc493629224)

[Chapter 5 - Files 6](#_Toc493629225)

[Introduction to Files 6](#_Toc493629226)

[Usefull OS functions 6](#_Toc493629227)

[Task 6 7](#_Toc493629228)

[Task 7 8](#_Toc493629229)

[Task 8 8](#_Toc493629230)

# Prerequisites

Ensure that you have completed the all previous tasks for the data structures.

# Introduction

This lab is to introduce the learner to the concepts of data structure selection and file access.

# Objectives of the lab

At the end of this lab you will have learnt the following:

* Effectively use data structures
* Write functions that can be rescued
* Apply principles learned in the previous modules to real world problems
* Access the filesystem

# DataStructure Selection

## Key Topics

**deterministic:** Pertaining to a program that does the same thing each time it runs, given the same inputs.

**pseudorandom:** Pertaining to a sequence of numbers that appear to be random, but are generated by a deterministic program.

**default value:** The value given to an optional parameter if no argument is provided.

**override:** To replace a default value with an argument.

**benchmarking:** The process of choosing between data structures by implementing alternatives and testing them on a sample of the possible inputs.

Go to Project Gutenberg ([*http://gutenberg.org*](http://gutenberg.org/)) and download your favorite out-of-copyright book in plain text format. Use wget to get it.

## Task 1

Write a program that reads a file, breaks each line into words, strips whitespace and punctuation from the words, and converts them to lowercase.

Hint: The*string*module provides strings named*whitespace*, which contains space, tab, newline, etc., *punctuation*which contains the punctuation characters and *printable* which contains printable characters.

>>> import string

>>> print string.punctuation

!"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~

Also, you might consider using the string methods*strip* and*replace*

## Task 2:

Modify your program from the previous exercise to read the book you downloaded, , and process the rest of the words as before.

Then modify the program to count the total number of words in the book, and the number of times each word is used.

Print the number of different words used in the book. Compare different books by different authors, written in different eras. Which author uses the most extensive vocabulary?

Bonus: write a function that takes in any number of books and produce the book with the most number of words

## Task 3

*Modify the program from the previous exercise to print the 20 most frequently-used words in the book.*

## Task 4

Modify the previous program to read a word list and then print all the words in the book that are not in the word list. How many of them are typos? How many of them are common words that should be in the word list, and how many of them are obscure?

Python provides a data structure called set that provides many common set operations. Read the documentation and write a program that uses set subtraction to find words in the book that are not in the word list.

## Task 5

The “rank” of a word is its position in a list of words sorted by frequency: the most common word has rank 1, the second most common has rank 2, etc.

Zipf’s law describes a relationship between the ranks and frequencies of words in natural languages ([*http://en.wikipedia.org/wiki/Zipf's\_law*](http://en.wikipedia.org/wiki/Zipf's_law)). Specifically, it predicts that the frequency,f, of the word with rankris:

|  |
| --- |
| f = c r−s |

wheresandcare parameters that depend on the language and the text. If you take the logarithm of both sides of this equation, you get:

|  |
| --- |
| logf = logc − s logr |

So if you plot logfversus logr, you should get a straight line with slope −sand intercept logc.

Write a program that reads a text from a file, counts word frequencies, and prints one line for each word, in descending order of frequency, with logfand logr. Use the graphing program of your choice to plot the results and check whether they form a straight line. Can you estimate the value ofs?

To make the plots, you might have to install matplotlib

To plot using matplotlib:

import matplotlib.pyplot as pyplot

pyplot.clf()

pyplot.xscale(scale)

pyplot.yscale(scale)

pyplot.title(“title string)

pyplot.xlabel(“label string”)

pyplot.ylabel(“label, string”)

pyplot.plot(x-value, y-value)

pyplot.show()

# Files

## Introduction to Files

**persistent:** Pertaining to a program that runs indefinitely and keeps at least some of its data in permanent storage.

**format operator:** An operator, %, that takes a format string and a tuple and generates a string that includes the elements of the tuple formatted as specified by the format string.

**format string:** A string, used with the format operator, that contains format sequences.

**format sequence:** A sequence of characters in a format string, like %d, that specifies how a value should be formatted.

**text file:** A sequence of characters stored in permanent storage like a hard drive.

**directory:** A named collection of files, also called a folder.

**path:** A string that identifies a file.

**relative path:** A path that starts from the current directory.

**absolute path:** A path that starts from the topmost directory in the file system.

**catch:** To prevent an exception from terminating a program using the try and except statements.

**database:** A file whose contents are organized like a dictionary with keys that correspond to values.

**item:** Another name for a key-value pair.

**key:** An object that appears in a dictionary as the first part of a key-value pair.

## Usefull OS functions

os.**listdir**(*path*)

Return a list containing the names of the entries in the directory given by *path*. The list is in arbitrary order. It does not include the special entries '.' and '..' even if they are present in the directory.

os.path.**join**(*path*, *\*paths*)

Join one or more path components intelligently. The return value is the concatenation of *path* and any members of *\*paths* with exactly one directory separator (os.sep) following each non-empty part except the last, meaning that the result will only end in a separator if the last part is empty. If a component is an absolute path, all previous components are thrown away and joining continues from the absolute path component.

os.path.**isfile**(*path*)

Return True if *path* is an existing regular file. This follows symbolic links, so both [islink()](https://docs.python.org/2.7/library/os.path.html?" \l "os.path.islink" \o "os.path.islink) and [isfile()](https://docs.python.org/2.7/library/os.path.html?" \l "os.path.isfile" \o "os.path.isfile) can be true for the same path.

os.path.**abspath**(*path*)

Return a normalized absolutized version of the pathname *path*. On most platforms, this is equivalent to calling the function [normpath()](https://docs.python.org/2.7/library/os.path.html?" \l "os.path.normpath" \o "os.path.normpath)as follows: normpath(join(os.getcwd(), path)).

os.path.**exists**(*path*)

Return True if *path* refers to an existing path. Returns False for broken symbolic links. On some platforms, this function may return False if permission is not granted to execute [os.stat()](https://docs.python.org/2.7/library/os.html" \l "os.stat" \o "os.stat) on the requested file, even if the *path* physically exists.

os.path.**isdir**(*path*)

Return True if *path* is an existing directory. This follows symbolic links, so both [islink()](https://docs.python.org/2.7/library/os.path.html?" \l "os.path.islink" \o "os.path.islink) and [isdir()](https://docs.python.org/2.7/library/os.path.html?" \l "os.path.isdir" \o "os.path.isdir) can be true for the same path.

## Task 6

The*os*module provides a function called*walk*that is similar to one below but more versatile. Read the documentation and use it to print the names of the files in a given directory and its subdirectories.

def walk(dirname):

for name in os.listdir(dirname):

path = os.path.join(dirname, name)

if os.path.isfile(path):

print path

else:

walk(path)

## Task 7

*Write a function called sed  that takes as arguments a pattern string, a replacement string, and two filenames; it should read the first file and write the contents into the second file (creating it if necessary). If the pattern string appears anywhere in the file, it should be replaced with the replacement string.*

*If an error occurs while opening, reading, writing, or closing files, your program should catch the exception, print an error message, and exit.*

## 

## Task 8

*In a large collection of MP3 files, there may be more than one copy of the same song, stored in different directories or with different file names. The goal of this exercise is to search for duplicates.*

1. *Write a program that searches a directory and all of its subdirectories, recursively, and returns a list of complete paths for all files with a given suffix (like .mp3). Hint: os.path provides several useful functions for manipulating file and path names.*
2. *To recognize duplicates, you can use md5sum to compute a “checksum” for each files. If two files have the same checksum, they probably have the same contents.*
3. *To double-check, you can use the Unix command diff.*