ECOR 1042
Data Management

Reading Data From Text Files
Iterative, Incremental Software Development

Carleton University

Recap Learning Outcomes Previous Lecture

- Know the meaning of these words and phrases
  - Dictionary/map (type `dict`)
  - Key, value associated with a key, key/value pair, entry
- Be able to evaluate expressions consisting of `dict` objects and some of the operations supported by that type
- Understand the key differences between lists, tuples, sets and dictionaries

References

- *Practical Programming*, 3rd ed.
  - Chapter 10, *Reading and Writing Files*
    - *What Kinds of Files are There?* (pp. 173 - 174)
    - *Opening a File* (pp. 175 - 178)
    - *Techniques for Reading Files* (pp. 179 - 183)
    - *Processing Whitespace-Delimited Data* (pp. 192 - 195)

Lecture Objectives

- Reading data from text files
- Using a case study, introduce iterative, incremental software development

Learning Outcomes (Vocabulary)

- Know the meaning of these words and phrases
  - Iterative development
  - Incremental development
  - Text file
  - `open` and `close` (built-in functions)
  - `split` and `strip` methods (type `str`)

Learning Outcomes

- Be able to apply iterative, incremental development when working on a small-scale software project
- Be able to write code that processes data read from a text file

Case Study: Project Goal

- Build a program that displays all the distinct words in a text file, converted to lowercase and sorted in ascending order

- Duplicate words should not be displayed

Case Study: Development Process

- Use *iterative*, *incremental* development
- Iterative development. Repeatedly cycle through these steps:
  - Identify the requirements for the next iteration
  - Design and implement the code that achieves those requirements
  - Test and debug the code
  - Reflect on progress so far
- Rationale: it is easier to find bugs if we construct a program in a step-wise manner

Case Study: Development Process

- Incremental development
  - Each iteration adds one or more features to the code developed in previous iterations
- Rationale: it is easier to determine if we are taking the wrong approach (incomplete or incorrect requirements, weak design) if we "grow" a program through a series of prototypes

Case Study: Prep Work

- Prepare some text files that we will use to demonstrate the application
- whyEnglishIsSoHard.txt
- walrus_and_carpenter.txt contains the poem *The Walrus and the Carpente*r by Lewis Carroll, from his book *Through the Looking-Glass, and What Alice Found There*
- For initial testing, use a text editor to prepare a smaller, simpler text file

File sample.txt

- Contains

```
First line of text.
Second line of text?
Third line of text!
```

- 3 lines (to check if the program can read multiple lines)
- Different words (to check if the program sorts the words)
- Some duplicate words (to check if the program discards duplicate words)
- Has punctuation marks (to check if the program treats `text.`, `text?` and `text!` as the same word)

Proposed Solution

- Read a text file, one line at a time
- Split the current line into words
- Convert each word to lowercase
- Discard the word if it has already been found; otherwise, save it in a collection of distinct words
- Sort the distinct words into ascending order
- Display the words

Proposed Design

- The solution is small enough that it can be implemented as a single Python module
- Most of the text-processing steps will be handled by one function
  - If the function becomes too complex, split it into simpler functions
  - Use Python's container types as much as possible
- We will need a short script to call the function

Iteration 1: Function Specification

- Use the Function Design Recipe that you learned in ECOR 1041 to define the header and docstring of the function that processes text files
  - The function will take the name of a file (a string) and return a sorted list of words (list of strings)
- Minimal implementation of the function body: just return an empty list

Iteration 1

```
def build_word_list(filename: str) -> list[str]:
    """Return a list of all the distinct words in the
    specified file, sorted in ascending order.

    >>> word_list = build_word_list('sample.txt')
    >>> word_list
    ['first', 'line', 'of', 'second', 'text',
    'third']
    """
    return []
```

Iteration 2: Read and Print a Text File

- Edit `build_word_list` to demonstrate that we can read and print the text from the specified text file, one line at a time
- Before we can read information from a file, we need to *open* it

```
infile = open(filename, "r")
```

- Function `open` opens the file in mode `"r"` (read-from-file) and returns an object that "knows" how to access the file

Iteration 2

- *Practical Programming* explains several techniques for reading files
- The "for line in file" technique is a good choice when we want to do the same thing with every line in a file
- This `for` loop reads the file, one line at a time:
  - `line` is assigned a string containing the line

```
for line in infile:
    process line
```

Iteration 2

- Edit `build_word_list`

```
def build_word_list(filename: str) -> list[str]:
    infile = open(filename, "r")

    for line in infile:
        print(line)

    infile.close()
    return []
```

We usually close a file after we have finished reading it

Iteration 2

`>>> build_word_list('sample.txt')` outputs

```
First line of text.

Second line of text?

Third line of text!
```

- Output is double-spaced because each line read from the file has a newline character at the end, and `print` also outputs a newline

Iteration 2: Reflection

- Should we change `build_word_list` so that takes the object returned by `open` instead of the name of the file?

- Should we change `build_word_list` so that prints the sorted list, instead of returning it?

Iteration 3: Split Lines into Words

- Edit `build_word_list` to call method `split` to split each line into a list of words (strings)
- By default, `split` removes leading and trailing whitespace (spaces, tabs, newlines), but does not remove punctuation marks

```
>>> line = '  Hello,    world!   '
```

`>>> line.split()` returns the list

```
['Hello,', 'world!']
```

Iteration 3

```
def build_word_list(filename: str) -> list[str]:
    infile = open(filename, "r")

    for line in infile:
        word_list = line.split()
        print(word_list)

    infile.close()
    return []
```

Iteration 3

`>>> build_word_list('sample.txt')` outputs one list for each line read from the file

```
['First', 'line', 'of', 'text.']
['Second', 'line', 'of', 'text?']
['Third', 'line', 'of', 'text!']
```

Reflection: move forward or go back and try a different approach?

Iteration 4: "Clean up" the Words

- Edit `build_word_list` to loop over the list of words
- In the loop, call method `strip` on each word to remove leading and trailing punctuation
- Example:

```
>>> word = 'Hello,'
```

`>>> word.strip(string.punctuation)` returns the string `'Hello'`

Iteration 4

- `punctuation` is a variable in module `string` containing all the punctuation characters:
- `'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'`

Iteration 4

- In the loop, call method `lower` to convert the word to lowercase
- Example

```
>>> word = 'Hello'
```

`>>> word.lower()` returns the string `'hello'`

Iteration 4

```
def build_word_list(filename: str) -> list[str]:
    infile = open(filename, "r")

    for line in infile:
        word_list = line.split()
        for word in word_list:
            word = word.strip(string.punctuation)
            word = word.lower()
            print(word)
    infile.close()
    return []
```

Iteration 4

`>>> build_word_list('sample.txt')` outputs all the words in the file, one per line

```
first
line
of
text
. . .
third
line
of
text
```

Iteration 4

- Reflection: `strip` returns a string and `lower` is called on that string, so we can put the two calls in a single statement

```
for word in word_list:
    word = word.strip(string.punctuation).lower()
```

Iteration 5: Store Distinct Words

- Edit `build_word_list` to discard duplicate words
- An easy way to do this is to modify the inner `for` loop to add the lowercase words to an initially-empty set
- Before returning from the function, print the set

Iteration 5

```
def build_word_list(filename: str) -> list[str]:
    infile = open(filename, "r")
    word_set = set()
    for line in infile:
        word_list = line.split()
        for word in word_list:
            word = word.strip(string.punctuation).lower()
            if word != ''
                word_set.add(word)

    infile.close()
    print(word_set)
    return []
```

Iteration 5

`>>> build_word_list('sample.txt')` outputs

```
{'third', 'second', 'text', 'first', 'line', 'of'}
```

Iteration 6: Return the List of Words

- Edit `build_word_list` to create the list of unique words from the set
- Remove the `print` calls and return the list

Iteration 6

```
def build_word_list(filename: str) -> list[str]:
    infile = open(filename, "r")
    word_set = set()
    for line in infile:
        word_list = line.split()
        for word in word_list:
            word = word.strip(string.punctuation).lower()
            if word != ''
                word_set.add(word)

    infile.close()
    word_list = list(word_set)
    return word_list
```

Iteration 6

`>>>build_word_list('sample.txt')` returns this list

```
['third', 'of', 'text', 'line', 'second', 'first']
```

Iteration 7: Return a Sorted List

- Edit `build_word_list` to return a sorted list of words
- Built-in function `sorted` returns a new list containing all the items from its argument
- Replace: `word_list = list(word_set)`

  with: `word_list = sorted(word_set)`

Iteration 7

```
def build_word_list(filename: str) -> list[str]:
    infile = open(filename, "r")
    word_set = set()
    for line in infile:
        word_list = line.split()
        for word in word_list:
            word = word.strip(string.punctuation).lower()
            if word != ''
                word_set.add(word)

    infile.close()
    word_list = sorted(word_set)
    return word_list
```

Iteration 7

- `build_word_list('sample.txt')` returns this list

```
['first', 'line', 'of', 'second', 'text', 'third']
```

Iteration 7

- Any other ideas on how to sort the words?
  - Sort method from the list collection

Iteration 7

- It looks like `build_word_list` is finished, so add a short script that calls the function

We will discuss `if __name__ == '__main__'` when we look at modules

```
if __name__ == '__main__':
    filename = 'sample.txt'
    word_list = build_word_list(filename)
    print('File', filename, 'contains',
          len(word_list), 'distinct words')
    print('The words are:', word_list)
```

Iteration 7

The program outputs

```
File sample.txt contains 6 distinct words
The words are: ['first', 'line', 'of', 'second',
'text', 'third']
```

Iteration 7

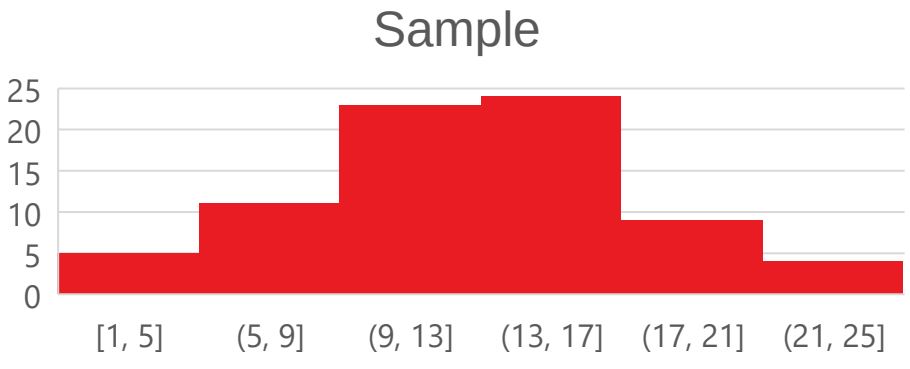
- We are almost done!
- We should try the program with different text files
  - whyEnglishIsSoHard.txt and walrus_and_carpenter.txt are posted on Brightspace
- Reflection: maybe modify the program to read the name of the text file from the keyboard?

More practice!

Carleton University

Histogram for a text file

- We can store the *histogram* for a text file as a collection of counters (in upcoming lectures, we will see how to plot it)
- Each counter keeps track of the number of occurrences of one word
- This can be implemented as a dictionary in which the keys are words, and the value associated with each key is the count of occurrences of that word



Exercise 1

- Design and implement a function named `build_histogram` that takes the name of a text file and returns a dictionary containing the *histogram* the words in the file
  - You should be able to reuse much of the code from `build_word_list`

Exercise 2

- Design and implement a function named `most_frequent_word` that takes a histogram and returns a tuple containing the most frequently-occurring word and its frequency

Exercise 3

- Write a script that displays the histogram of a text file, the word that occurs most frequently, and the number of times that word occurs in the file

Recap Learning Outcomes

- Be able to apply iterative, incremental development when working on a small-scale software project
- Be able to write code that processes data read from a text file
- Know the meaning of these words and phrases
  - Iterative development
  - Incremental development
  - Text file
  - `open` and `close` (built-in functions)
  - `split` and `strip` methods (type `str`)

ECOR 1042
Data Management

Reading Data From Text Files
Iterative, Incremental Software Development