COMP10001 Foundations of Computing
Semester 1, 2019

Tutorial Questions: Week 9

— VERSION: 1474, DATE: MAY 1, 2019 —

Discussion

1. Why do we use "files"? Could we use computers without them?

   A: *Files allow us to store data on a computer permanently: they will persist on storage media after a program is terminated, as opposed to internal data storage such as lists and dictionaries which exist in the computer's temporary memory and are erased when the program finishes.*
   Files are also useful for storing large amounts of data in a structured way and sharing it with others.

2. What are the steps to reading and writing files?

   A: *In Python, to open a file we use the `open()` function, which takes the file's filename as a string and another string which represents the "mode" for access: some common ones are `'r'` for reading; `'w'` for writing (erasing all file contents if the file exists initially); and `'a'` for appending to an already-existing file. `open()` returns a "file object" which is used to access the file.*
   Some common ways to read a file are using the `file.read()` to read a whole file, returning a string; `file.readline()` to read one line of the file, returning a string; and `file.readlines()` to read an entire file, returning a list with each row split as a separate element in the list list. To write, use the `file.write()` to write a string to output.
   When finished with a file, be sure to close it with `file.close()` to prevent buffer errors.

3. What is a "csv" file and why is it useful for storing and manipulating data?

   A: *A csv (comma separated values) file is a text file stored in a particular format, that being as rows of a spreadsheet with individual data cells stored separated by a comma (`,`)and rows separated by a new line character `\n`. csv data is useful for storing statistics or measurement data because the structure of the file allows Python to read and process the data in its spreadsheet-like format.*

   Now try Exercises 1

4. What is a "list comprehension"? How do we write one and how do they make our code simpler?

   A: *A list comprehension is a shortcut notation used to accomplish simple iteration tasks involving a collection (usually list, but also possibly a set or dictionary) in one line of code. List comprehensions are formed by a pair of brackets wrapping ¡expression¿ ¡for iteration statement¿ ¡optional if filter condition¿. The for iteration statement will be run and for each iteration, the result of ¡expression¿ will be added to the collection. If a filter condition is included, the object will only be added if that condition evaluates to `True`.*
   List comprehensions are useful to shorten repetitive, simple loops into readable single lines of code. They are especially useful for initialising lists. Avoid cramming too much into list comprehensions or nesting them inside each other, we want them to remain readable.

5. What is a "`defaultdict`"? How do we initialise and use it?

   A: *A `defaultdict` is a data type which has the same behaviour as a dictionary with the added functionality of initialising new keys to a default value when trying to use them. To use a defaultdict, you must first import it by including the line `from collections import defaultdict` at the top of your program. Then, to create a defaultdict, you call `defaultdict(type)` where `type` is the data type which will be stored as a value for keys in the dictionary. When requesting or updating values which haven't been previously set, instead of resulting a `KeyError`, Python will initialise the value to the "zero value" of that data type (zero for numbers, an empty collection for strings, lists, dictionaries ect.)*
   Defaultdicts are especially useful for counting, as you can increment for each time you see something without worrying about initialising it if it's the first time you've seen that value.

   Now try Exercises 2 & 3

Exercises

1. Fill in the blanks in the program below which reads from `in.txt` and writes to `out.txt`.

```
outfile = [    1    ]("out.txt", "w")
with open("in.txt", [    2    ]) as infile:
    line_no = 1
    for line in [    3    ]:
        outfile.[    4    ](f"line: {line_no}, length: {len(line)}\n")
        line_no += 1
```

```
outfile.write("The End")
```
```
5
```

A: *(1)* `open`
(2) `'r'`
(3) `infile.readlines()`
(4) `write`
(5) `outfile.close()`

2. Evaluate the following list comprehensions. For each one, also write some python code to generate the same list without using a comprehension.

   (a) `[(name, 0) for name in {"evelyn", "alex", "sam"}]`

   A:

   ```
   [('sam', 0), ('evelyn', 0), ('alex', 0)]
   ```

   ```
   my_list = []
   for name in {"evelyn", "alex", "sam"}:
       my_list.append((name, 0))
   ```

   (b) `[i**2 for i in range(5) if i % 2 == 1]`

   A:

   ```
   [1, 9]
   ```

   ```
   my_list = []
   for i in range(5):
       if i % 2 == 1:
           b.append(i**2)
   ```

   (c) `"".join([letter.upper() for letter in "python"])`

   A:

   ```
   'PYTHON'
   ```

   ```
   my_list = []
   for letter in "python":
       my_list.append(letter.upper())
   my_str = "".join(my_list)
   ```

3. Rewrite the following with a default dictionary

```
my_dict = {}
for i in range(20):
    if i % 3 in my_dict:
        my_dict[i % 3].append(i)
    else:
        my_dict[i % 3] = [i]
```

A:

```
from collections import defaultdict

my_dict = defaultdict(list)
for i in range(20):
    my_dict[i % 3].append(i)
```

Problems

1. Write a function which takes a filename as a string and opens the file, printing statistics for every line in the file: the line number, amount of words, amount of characters. Remember to close the file when you're done!

   A:

   ```
   def statistics(filename):
       f = open(filename, 'r')
       line_no = 1
       print("Line number: num words, num characters")
       for line in f.readlines():
           print(f"{line_no}: {len(line.split())}, {len(line)}")
           line_no += 1
       f.close()
   ```

2. Write a function which takes a string as input and returns a sorted list of the words which occur only once in the string. Try using a defaultdict and list comprehensions in your solution.

   A:

   ```
   from collections import defaultdict

   def once_words(text):
       freqs = defaultdict(int)
       for word in text.split():
           freqs[word] += 1
       return sorted([word for word in freqs if freqs[word] == 1])
   ```