COMP10001 Foundations of Computing
Semester 2, 2022

Tutorial Solutions: Week 9

— VERSION: 1479, DATE: SEPTEMBER 19, 2022 —

Discussion

1. 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 wrapping a pair of brackets around* `<expression> <for iteration statement> <optional if filter condition>`*. The iteration statement will be run and for each iteration, the result of the expression will be added to the collection - which could be a list, set or dictionary. 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.

2. What happens if we use curly brackets instead of square brackets around a "list" comprehension? What happens if we use parentheses?

   A: *If we use curly brackets* `{}` *around a comprehension syntax it will evaluate into a set. For example,* `{n**2 for n in range(1, 10)}`*, will give us a set containing* 1, 4, 9, . . . , 64, 81.
   We can also use the `key:value` *syntax for the expression part of a comprehension with curly brackets to get a dictionary comprehension. For example,* `{word:len(word) for word in ["apple", "banana"]}` *will evaluate into a dictionary with the words as keys and the length of each word as the values.*
   Perhaps unexpectedly, parentheses do ***not*** *give us a tuple comprehension, instead it creates an object called a generator, which is an efficient way of creating an iterator. We don't expect you to know how to use generators, but just beware if you thought you'd get a tuple!*

   Now try Exercise 1

3. 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. It would not be possible to use code for solving many big data problems without them. In fact, it would not be possible to write code without them!

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

   A: *In Python, to open a file we use the* `open()` *function, which takes two arguments: the file's filename as a string; and another string which represents the mode for access. Some common modes 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()` *method 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 of the file split as a separate element in the list. To write, use the* `file.write()` *method to write a string to output.*
   When finished with a file, be sure to close it with the `file.close()` *method to prevent buffer errors.*

5. 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, similar to that of a spreadsheet. That is, as rows of data with individual values 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. Python's* `csv` *library is very useful for processing csv files.*

   Now try Exercises 2 & 3

Exercises

1. 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:

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

2. 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()`

3. "travel.csv" is a csv file containing data on how people get to work in different cities in Australia. "process.py" is a python program which processes this data. What information does the "process.py" attempt to find and print?

`travel.csv`

```
City,Train,Tram,Bus,Ferry,Car,Total
Melbourne,242969,55169,31937,783,1282997,1613855
Sydney,368572,3210,138340,9007,1206350,1725482
Adelaide,13715,4137,33673,211,390360,442102
Brisbane,62069,229,58228,3761,663353,787650
Perth,56417,223,37899,373,594571,689489
```

Data Source: Census of Population and Housing, 2016, TableBuilder

`process.py`

```
1 import csv
2
3 fp = open("travel.csv")
4 city = ''
5 curr_max = 0.0
```

```
6  for row in csv.DictReader(fp):
7      ferry = int(row["Ferry"])
8      total = int(row["Total"])
9      if ferry / total > curr_max:
10         city = row["City"]
11         curr_max = ferry / total
12 print(city)
```

A: *"process.py" finds the city in which the highest proportion of its population uses the ferry to get to work. It will print* `Sydney`*.*
To customise this program, we could change line 7 to fetch a different column, giving it a new header string such as `"Train"` *or* `"Car"`*.*
As an exercise, try writing a program which prints the proportion of transport mode used in each city.

Problems

1. Let's head back to worksheet 8 and solve "For a While ..." again! **Using a list comprehension**, rewrite the function `allnum` that takes a list of strings, and returns a list of those that exclusively contain digits. `allnum(['3', '-4', '5', '3.1416', '0xfff', 'blerg!'])` should return `['3', '5']`.

   A:

   ```
   def allnum(strlist):
       return [curr_str for curr_str in strlist if curr_str.isdigit()]
   ```

2. Remember the "Gamertag Problem" from worksheet 8? Rewrite the `make_gamertag` function that takes a `name` string and returns a string with a hyphen after each letter, but this time, **using a list comprehension**. `make_gamertag('Alex')` should return `'A-l-e-x-'`.

   A:

   ```
   def make_gamertag(name):
       return "".join([letter + "-" for letter in name])
   ```

3. You've found a secret message:

   `secret_message.txt`

   ```
   pf wizveu,
   nv riv gcreezex r jligizjv grikp wfi ave'j szikyurp.
   zk svxzej rk 5gd fe jleurp.
   gcvrjv nvri r k-ivo tfjkldv - zk'j r uivjj lg grikp nzky r uzefjrli
   kyvdv.
   ufe'k cvk lj ufne!
   ```

   All that you know about the message is that it it encrypted by a basic shift cipher (also known as a Caesar cipher, where each letter is shifted by some constant number of places in the alphabet), any alphabetic character in the message is lowercase, and that it contains the string segment `'plan'`.

   Write a function to decrypt the message that takes an `infilename`, `outfilename` and `segment` (all strings), You can use a brute-force approach (try all possible values) to guess the number to shift by. You might find the functions, `ord(character)` and `chr(number)`, useful!

A:

```
ALPHABET_SIZE = 26

def shift_cipher_decoder(infilename, outfilename, segment):
    with open(infilename) as infile, open(outfilename, 'w') as outfile:
        encrypted = infile.read()
        for shift in range(ALPHABET_SIZE):
            decrypted = ""
            for letter in encrypted:
                if letter.isalpha():
                    letter_number = (ord(letter) + shift) % ALPHABET_SIZE
                    decrypted += chr(ord("a") + letter_number)
                else:
                    decrypted += letter
            if segment in decrypted:
                outfile.write(decrypted)
```