COMP10001

Week 9

What is a “list comprehension”? How do we write one and how do they simplify our code?

What is a “list comprehension”? How do we write one and how do they simplify our code?

- A cheeky shortcut for simple iteration tasks involving a collection (i.e. list, set, dictionary)

What is a “list comprehension”? How do we write one and how do they simplify our code?

- A cheeky shortcut for simple iteration tasks involving a collection (i.e. list, set, dictionary)
- [<expression> <for iteration statement> <optional if filter condition>]

What is a “list comprehension”? How do we write one and how do they simplify our code?

- A cheeky shortcut for simple iteration tasks involving a collection (i.e. list, set, dictionary)
- [<expression> <for iteration statement> <optional if filter condition>]
- For each iteration, the outcome of <expression> is added to the collection
    - If the <if filter condition> is included, this only happens when the condition is True

What is a “list comprehension”? How do we write one and how do they simplify our code?

- A cheeky shortcut for simple iteration tasks involving a collection (i.e. list, set, dictionary)
- [<expression> <for iteration statement> <optional if filter condition>]
- For each iteration, the outcome of <expression> is added to the collection
  - If the <if filter condition> is included, this only happens when the condition is True
- Avoid overcomplicating these! They’re a neat way to wrap up simple loops into one line!

What happens if we use curly brackets instead of square brackets around a list comprehension? How about parentheses?

What happens if we use curly brackets instead of square brackets around a list comprehension? How about parentheses?

- [i**2 for i in range(10)] -> list!
- {i**2 for i in range(10)} -> set!

What happens if we use curly brackets instead of square brackets around a list comprehension? How about parentheses?

- [i**2 for i in range(10)] -> list!
- {i**2 for i in range(10)} -> set!
- (i**2 for i in range(10)) -> NOT a tuple!
  - This is a generator, which we can iterate over. You don’t need to know about generators, but you do need to know that this expression doesn’t give us a tuple!

Exercise!

Evaluate the following list comprehensions. Also, write some equivalent Python code that doesn't use a comprehension.

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

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

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

(d)
```
[(row, col) for row in range(3, 5) for col in range(2)]
```

Why do we use files? Could we use computers without them?

Why do we use files? Could we use computers without them?

- Files store data permanently - they persist after a program is terminated

Why do we use files? Could we use computers without them?

- Files store data permanently - they persist after a program is terminated
- That’s different to internal data storage like lists and dictionaries - they are in temporary memory and are erased when the program finishes

Why do we use files? Could we use computers without them?

- Files store data permanently - they persist after a program is terminated
- That’s different to internal data storage like lists and dictionaries - they are in temporary memory and are erased when the program finishes
- Files also allow us to organise/structure our data …

Why do we use files? Could we use computers without them?

- Files store data permanently - they persist after a program is terminated
- That’s different to internal data storage like lists and dictionaries - they are in temporary memory and are erased when the program finishes
- Files also allow us to organise/structure our data …
- … and share it!

What are the steps to reading and writing files?

What are the steps to reading and writing files?

- In Python, to **open** a file we use the open() function, which has two arguments:
  - open(<filename>, <mode>)

What are the steps to reading and writing files?

- In Python, to **open** a file we use the open() function, which has two arguments:
  - open(<filename>, <mode>)
  - <mode> defaults to “r”, for reading
  - <mode> could also be set to <w> for writing
    - The contents of a file, if it already exists, are overwritten
  - … Or <a> for appending to a file that already exists

What are the steps to reading and writing files?

- In Python, to **open** a file we use the open() function, which has two arguments:
  - open(<filename>, <mode>)
  - <mode> defaults to “r”, for reading
  - <mode> could also be set to <w> for writing
    - The contents of a file, if it already exists, are overwritten
  - … Or <a> for appending to a file that already exists
- To **read** a file, we can use:

What are the steps to reading and writing files?

- In Python, to **open** a file we use the open() function, which has two arguments:
  - open(<filename>, <mode>)
  - <mode> defaults to “r”, for reading
  - <mode> could also be set to <w> for writing
    - The contents of a file, if it already exists, are overwritten
  - … Or <a> for appending to a file that already exists
- To **read** a file, we can use:
  - file.read() for a whole file
  - file.readline() to read one line of a file, as a string
  - file.readlines() to read an entire file, returning a list with each row of the file a string in the list

What are the steps to reading and writing files?

- In Python, to **open** a file we use the open() function, which has two arguments:
  - open(<filename>, <mode>)
  - <mode> defaults to “r”, for reading
  - <mode> could also be set to <w> for writing
    - The contents of a file, if it already exists, are overwritten
  - … Or <a> for appending to a file that already exists
- To **read** a file, we can use:
  - file.read() for a whole file
  - file.readline() to read one line of a file, as a string
  - file.readlines() to read an entire file, returning a list with each row of the file a string in the list
- To **write** a file, we use file.write() to write a string to the output

What are the steps to reading and writing files?

- In Python, to **open** a file we use the open() function, which has two arguments:
  - open(<filename>, <mode>)
  - <mode> defaults to “r”, for reading
  - <mode> could also be set to <w> for writing
    - The contents of a file, if it already exists, are overwritten
  - … Or <a> for appending to a file that already exists
- To **read** a file, we can use:
  - file.read() for a whole file
  - file.readline() to read one line of a file, as a string
  - file.readlines() to read an entire file, returning a list with each row of the file a string in the list
- To **write** a file, we use file.write() to write a string to the output
- When we’re done, we **close** the file, with file.close()

What is a “csv” file and why is it useful for storing and manipulating data?

What is a “csv” file and why is it useful for storing and manipulating data?

- A **comma separated values** file is a text file stored in a specific format, like a spreadsheet

What is a “csv” file and why is it useful for storing and manipulating data?

- A **comma separated values** file is a text file stored in a specific format, like a spreadsheet
- Rows of data with individual values, separated by a comma (,) and rows separated by a newline character (\n)

What is a “csv” file and why is it useful for storing and manipulating data?

- A **comma separated values** file is a text file stored in a specific format, like a spreadsheet
- Rows of data with individual values, separated by a comma (,) and rows separated by a newline character (\n)
- When we’re using data, it’s often natural to store it in rows and columns
    - So csvs are very powerful!

Fill in the blanks!

```
outfile = ("out.txt", "w")
with open("in.txt", ) as infile:
    line_no = 1
    for line in :
        outfile. f"line: {line_no}, length: {len(line)}\n")
        line_no += 1
outfile.write("The End")
```

Fill in the blanks!

```
outfile = ____("out.txt", "w")
with open("in.txt", ____) as infile:
    line_no = 1
    for line in ____:
        outfile.____(f"line: {line_no}, length: {len(line)}\n")
        line_no += 1
outfile.write("The End")
____
```

open
'r'
infile.readlines()
write
outfile.close()

Given the following csv file and Python script, what is the code attempting 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
```

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)
```

Using a list comprehension, (re)write 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']`

Using a list comprehension, (re)write 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']`

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

Using a list comprehension, (re)write the make_gamertag function that takes a name string and returns a string with a hyphen after each letter

`make_gamertag('Alex')` should return `'A-l-e-x-'`.

Using a list comprehension, (re)write the make_gamertag function that takes a name string and returns a string with a hyphen after each letter

`make_gamertag('Alex')` should return `'A-l-e-x-'`.

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

You've found a secret message:

`secret_message.txt`

```
erkbvl ur kbvd tlmexr:
gxoxk zhggt zbox rhn ni
gxoxk zhggt exm rhn whpg
gxoxk zhggt kng tkhngw tgw wxlxkm rhn
gxoxk zhggt ftdx rhn vkr
gxoxk zhggt ltr zhhwurx
gxoxk zhggt mxee t ebx tgw ankm rhn
```

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 `'desert'`.

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!