CMPT 120 LECTURE 4-3

Prof. Nick Vincent

Simon Fraser University

2023-09-27

4-3 HOUSEKEEPING

- Assignment questions
    - Some great questions in Discussion / Discord / email. Thank you!
- Non-graded coding practice opportunities?
    - Great discussion in Discord
- Summary script examples

ASSIGNMENT QUESTIONS

- Clarification to Coding Exercise 3: user will give a non-negative number
- initializing variables in `if` statements

ONLINE PLATFORMS FOR CODING PRACTICE

These are very popular for interview prep down the line.

Some platforms to try out:

- https://www.codewars.com
  - Important: sort answers by “Best Practice” instead of “Clever”
- https://www.hackerrank.com/
- https://www.codecademy.com/

I’ll try to curate some example exercises, but please feel free to share.

SOME CAVEATS

Online coding problems / puzzles / challenges don't accurately reflect real-world programming.

This is why many people are mad about CS interview practices (though there are advantages).

That's where do *projects* is important.

WHY YOU MIGHT WANT TO TRY CODING CHALLENGES

- Getting instant feedback (seeing other people's solutions) can be really helpful.
- Give it an honest try (say, 30 mins) and try to reflect on how it compares to doing class assignments, live coding, interactive textbook, etc.
- Reflecting on learning methods that work for you is a great skill to practice for uni and life

4-3 AGENDA

- Reading from files
  - New methods for reading, splitting strings
- List indexing
  - How we grab a specific item from a list?
- We'll weave these together as we work towards a recommendation example

RECOMMENDING WITH DATA

"BIG" DATA

- It's time to load data from a file!
- Instead of populating our list right in the code, we'll *read* from a file
- Can think of "databases" as fancy ways to handle files

CONCEPTS WE'LL USE

- breaking up a long strings into list (`split()`)
- open a file
- read a file
- accessing a specific element of a list ("indexing" using `my_list[1]`)
- Comparing numbers

INTRODUCTION TO LIST INDEXING IN PYTHON

- Python lists are a versatile data structure.
- They are ordered collections, which means items have a defined order that will not change.
- This allows us to access items in a list by their position, or "index".

BASIC INDEXING

In Python, indexing starts from `0`, which means the first item is accessed using an index of `0`, the second with an index of `1`, and so on.

```python
my_list = ["apple", "banana", "cherry", "date"]
first_fruit = my_list[0]
print(first_fruit)  # Outputs: apple
```

```
apple
```

NEGATIVE INDEXING

Python also supports negative indexing. This means the last item has an index of -1, the second to last -2, and so on.

```python
last_fruit = my_list[-1]
print(last_fruit)  # Outputs: date
```

```
date
```

INDEX OUT OF RANGE

If you try to access an index that does not exist in the list, you’ll get an error.

```
# Uncommenting the next line will cause an error
# nonexistent_fruit = my_list[10]
```

INDEXING SUMMARY TABLE

Action	List Example	String Example
Accessing by index	`my_list[2]`	`my_string[2]`
Accessing from the end (negative index)	`my_list[-1]`	`my_string[-1]`
Slicing (start to end)	`my_list[1:3]`	`my_string[1:3]`
Slicing (start to end, with step)	`my_list[0:5:2]`	`my_string[0:5:2]`
Slicing (everything until end)	`my_list[2:]`	`my_string[2:]`
Slicing (from start until position)	`my_list[:3]`	`my_string[:3]`
Slicing (using negative indices)	`my_list[-3:-1]`	`my_string[-3:-1]`
Accessing every nth element/character	`my_list[::2]`	`my_string[::2]`

OPENING AND READING FILES IN PYTHON

- Files are a fundamental means of data storage.
- Python provides built-in functions to work with files
- we can read from and write to them.

THE `open` FUNCTION

To work with a file, we first need to open it using the `open` function.

```python
# Syntax: open(filename, mode)
file = open("example.csv", "r")
```

- filename: Name of the file you want to open. Have to watch out for path issues
- mode: Mode in which you want to open the file ("r" for read, "w" for write, etc.)

CLOSING THE FILE

If we use `file = open("example.csv", "r")`, we also have to run `file.close()` after.

`with open(...)`

Alternatively, we can using the `with` pattern

```python
with open("example.csv", "r") as file:
    content = file.read()
    print(content)
```

```
Name,Drink Preference,Basketball Team Preference
John Doe,Coffee,Lakers
Jane Smith,Tea,Raptors
Emily Johnson,Coffee,Raptors
Michael Brown,Tea,Lakers
Oliver White,Coffee,Lakers
Sophia Martinez,Tea,Raptors
Ethan Garcia,Coffee,Lakers
Mia Hernandez,Tea,Raptors
Lucas Robinson,Coffee,Lakers
Isabella Wilson,Tea,Raptors
```

`with open(...)`

With this method, you don't need to explicitly close the file. It'll be closed automatically when the block of code is exited.

THE `.readline()` METHOD IN PYTHON

When working with files, `.readline()` provides a way to read individual lines.

BASIC USAGE

Each call to `.readline()` reads the next line from the file.

```python
file = open("example.csv", "r")
first_line = file.readline()
print(first_line)
file.close()
```

```
Name,Drink Preference,Basketball Team Preference
```

WHY USE .READLINE()

- Efficient for reading large files without loading them into memory entirely.
- Reading the first few lines of a file without processing the entire content.
- Note: If called after reading the entire file, it returns an empty string (““).
- Can also use it to “throw away” the first line (e.g., a header)

READING LINE BY LINE

```python
with open("example.csv", "r") as file:
    for line in file:
      print(line)
```

```
Name,Drink Preference,Basketball Team Preference

John Doe,Coffee,Lakers

Jane Smith,Tea,Raptors

Emily Johnson,Coffee,Raptors

Michael Brown,Tea,Lakers

Oliver White,Coffee,Lakers

Sophia Martinez,Tea,Raptors

Ethan Garcia,Coffee,Lakers
```

READING LINE BY LINE (SKIP THE FIRST)

```python
with open("example.csv", "r") as file:
    _ = file.readline()
    for line in file:
      print(line)
```

```
John Doe,Coffee,Lakers

Jane Smith,Tea,Raptors

Emily Johnson,Coffee,Raptors

Michael Brown,Tea,Lakers

Oliver White,Coffee,Lakers

Sophia Martinez,Tea,Raptors

Ethan Garcia,Coffee,Lakers

Mia Hernandez,Tea,Raptors
```

STRING SPLITTING & READING CSV FILES IN PYTHON

The `split()` method in Python is powerful for parsing strings, especially when reading structured data like CSV files.

BASIC SPLITTING

Split a string into a list based on a delimiter (default is whitespace).

```python
text = "apple,banana,cherry"
fruits = text.split(",")
print(fruits)  # Outputs: ['apple', 'banana', 'cherry']
```

```
['apple', 'banana', 'cherry']
```

OPEN, GO LINE BY LINE, AND PRINT

```python
with open("example.csv", "r") as file:
    for line in file:
        columns = line.strip().split(",")
        print(columns)
```

```
['Name', 'Drink Preference', 'Basketball Team Preference']
['John Doe', 'Coffee', 'Lakers']
['Jane Smith', 'Tea', 'Raptors']
['Emily Johnson', 'Coffee', 'Raptors']
['Michael Brown', 'Tea', 'Lakers']
['Oliver White', 'Coffee', 'Lakers']
['Sophia Martinez', 'Tea', 'Raptors']
['Ethan Garcia', 'Coffee', 'Lakers']
['Mia Hernandez', 'Tea', 'Raptors']
['Lucas Robinson', 'Coffee', 'Lakers']
['Isabella Wilson', 'Tea', 'Raptors']
```

NOTES ABOUT READING FILES WITH SPLIT

- Note 1: if data contains commas (,), it might lead to incorrect splitting.
- Note 2: in industry, you'll probably use libraries to do this kind of thing. But have the learn the "raw" way for now!

TASK

Create a Python script that: - Reads a CSV file named example.csv. - Skips the header line. - Prints each record in a readable format

Example: “John Doe | Coffee | Lakers”

ANSWER

```python
with open("example.csv", "r") as file:
    # Skip header
    header = file.readline()

    # Process and print each record
    for line in file:
        columns = line.strip().split(",")
        nice_output = ""
        for column in columns:
          nice_output += column + " | "
        print(nice_output)
```

```
John Doe | Coffee | Lakers |
Jane Smith | Tea | Raptors |
Emily Johnson | Coffee | Raptors |
Michael Brown | Tea | Lakers |
Oliver White | Coffee | Lakers |
Sophia Martinez | Tea | Raptors |
Ethan Garcia | Coffee | Lakers |
Mia Hernandez | Tea | Raptors |
Lucas Robinson | Coffee | Lakers |
Isabella Wilson | Tea | Raptors |
```

ERROR HANDLING: A CRUCIAL SKILL

When working with files, errors are inevitable.

You might try to open a nonexistent file or read past the end of a file.

COMMON FILE ERRORS

- FileNotFoundError: The file you're trying to open doesn't exist.
- PermissionError: You don't have permission to access the file.
- IOError: A general error related to file I/O operations.

HANDLING FILE ERRORS IN PYTHON

Using try and except blocks, you can gracefully handle these errors.

```python
try:
    with open("nonexistent_file.csv", "r") as file:
        content = file.read()
        print(content)
except FileNotFoundError:
    print("The file does not exist. Please check the filename.")
except PermissionError:
    print("You don't have the permission to read the file.")
except IOError:
    print("An error occurred while accessing the file.")
```

```
The file does not exist. Please check the filename.
```

PYTHON COMPARISON OPERATORS

Operator	Name	Description	Example	Result
`!=`	Not equal to	Checks if two values are not equal	`5 != 3`	True
`>`	Greater than	Checks if left value is greater	`5 > 3`	True
`>=`	Greater or equal	Checks if left value is greater/equal	`5 >= 5`	True
`<`	Less than	Checks if left value is smaller	`3 < 5`	True
`<=`	Less or equal	Checks if left value is smaller/equal	`3 <= 5`	True

COMMON MISTAKE: X == 5 OR 6

❌ Incorrect: writing `x == 5 or 6`

This checks if x is equal to 5 OR if 6 is a truthy value. Since 6 (or any non-zero number) is always truthy, this statement will always evaluate to True.

✅ Correct: writing `x == 5 or x == 6`

This checks if x is equal to 5 OR if x is equal to 6.

PYTHON OPERATOR PRECEDENCE

Precedence	Operators	Description
1	`()`	Parentheses (grouping)
2	`**`	Exponentiation
3	`+x`, `-x`, `~x`	Unary plus, Unary minus, Bitwise NOT
4	`*`, `/`, `//`, `%`	Multiplication, Division, Floor division, Modulus
5	`+`, `-`	Addition, Subtraction
6	`<<`, `>>`	Bitwise shift left, Bitwise shift right
7	`&`	Bitwise AND
8	`^`	Bitwise XOR
9	`|`	Bitwise OR
10	`==`, `!=`, `<`, `<=`, `>`, `>=`	Comparisons, Equality and Inequality
11	`not`	Logical NOT

Precedence	Operators	Description
12	`and`	Logical AND
13	`or`	Logical OR
14	`=`, `+=`, `-=`, `*=`, …	Assignment operators

PYTHON OPERATOR PRECEDENCE (SIMPLER)

LIVE CODE WITH REMAINING TIME

Let’s try to count up and make a recommendation about both coffee vs. tea and Lakers vs. Raptors