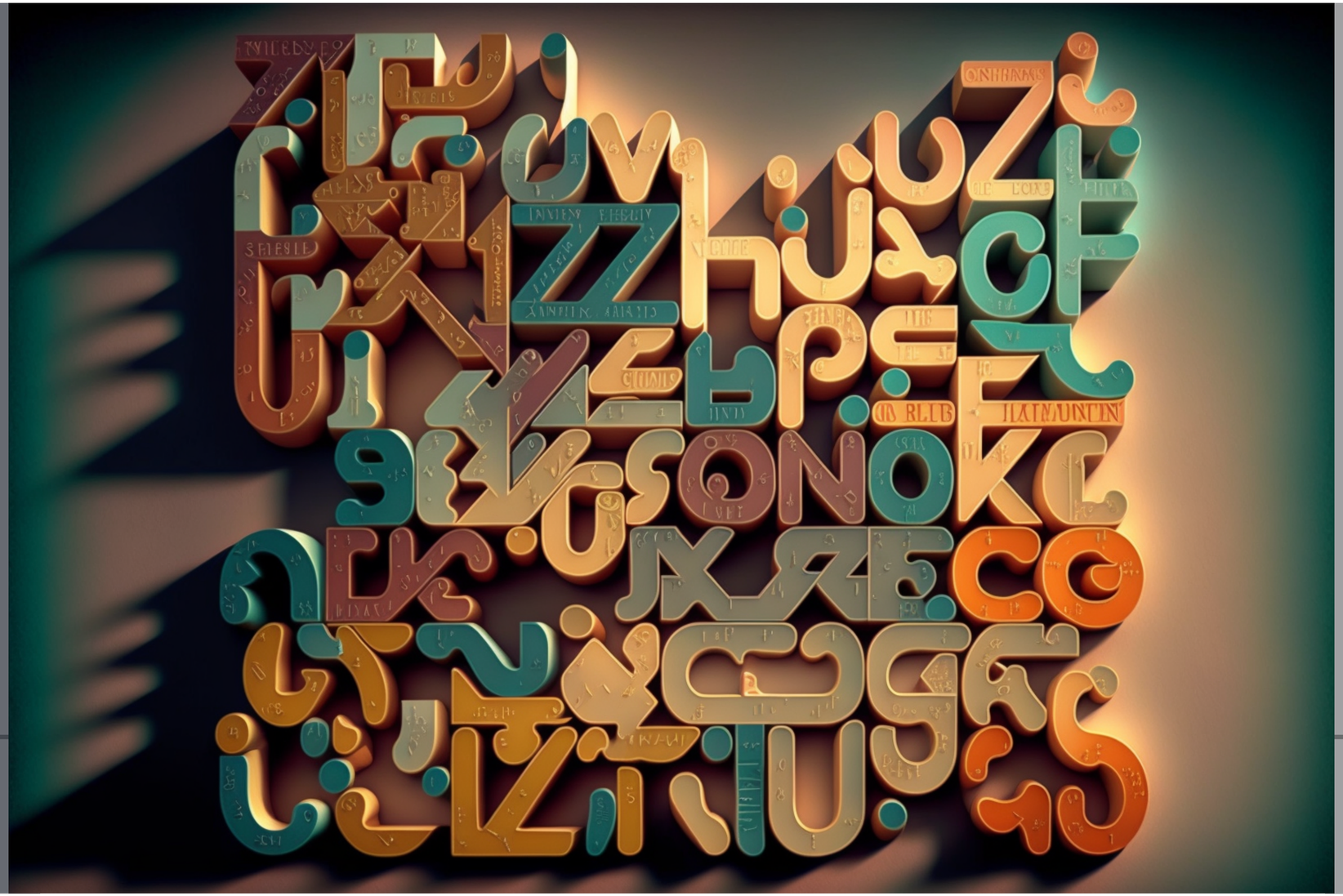


Mid Journey V4 prompt: word puzzle --ar 3:2 --q 2

LECTURE 4-2: PYTHON BASICS III - STRING PROCESSING

Dr. James Chen, Animal Data Scientist, School of Animal Sciences

2023 APSC-5984 SS: Agriculture Data Science

Overview

- APSC-5984 Lab 4: File system
  - 3. Interacting with files
    - 3.1 Create a file (mode `w`)
    - 3.1.1 Write to a file
    - 3.1.2 End of line (EOL)
    - 3.1.3 Close the file and `with` statement
    - 3.2 Append to a file (mode `a`)
    - 3.3 Read a file (mode `r`)
  - 4. String processing
    - 4.1 String slicing
    - 4.2 String split
    - 4.3 String replace
    - 4.4 Regular expression (RE)
      - 4.4.1 Find all numbers
      - 4.4.2 Find numbers that match a string pattern
      - 4.4.3 Greedy vs. non-greedy matching
      - 4.4.4 Exclusions
      - 4.4.5 Another example

3. Interacting with files

Now that you know how to navigate the file system and basic knowledge of paths, we can start to interact with files in Python!

3.1 Create a file (mode `w`)

To create a file, we can use the `open()` method. The `open()` method takes two arguments: the path to the file and the mode. The path just works as the way we learned in the previous section, it can be either absolute or relative path. The mode is a string that specifies how you want to interact with the file. The most common modes are:

- `r`: read-only mode
- `w`: write-only mode
- `a`: append-only mode


```
f = open('file.txt', 'w')
f.close()
```

Write Contents to a File

Python

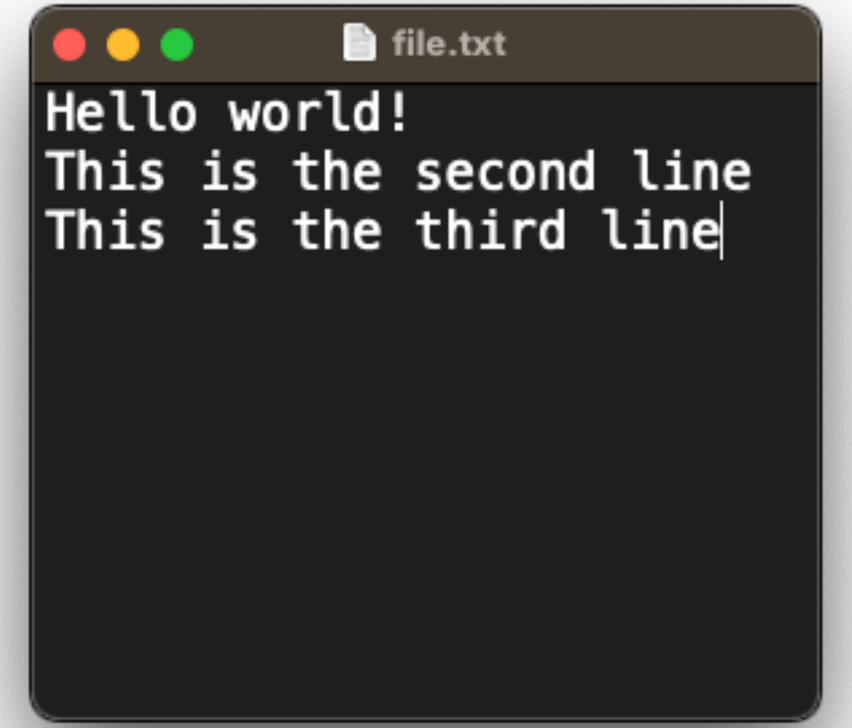
```
f = open('file.txt', 'w')
f.write('Hello world!\n')
f.write('This is the second line\n')
f.write('This is the third line\n')
f.close()
```

‘cat’ to print the file contents

Shell

```
cat file.txt
# Hello world!
# This is the second line
# This is the third line
```

Text editor



End of Line (EOL)

3.1.2 End of line (EOL)

It is noteworthy that the `write()` method will not automatically add an end of line (EOL) character (`\n`) at the end of the line. Therefore, we need to add it manually.

With "\n"

```
f = open('file.txt', 'w')
f.write('Hello world!\n')
f.write('This is the second line\n')
f.write('This is the third line\n')
f.close()
```

```
cat file.txt
# Hello world!
# This is the second line
# This is the third line
```

Without "\n"

```
f = open('file.txt', 'w')
f.write('Hello world!')
f.write('This is the second line')
f.write('This is the third line')
f.close()
```

```
cat file.txt
# Hello world!This ia the second lineThis ia the third line
```

Always Close the File

3.1.3 Close the file and `with` statement

We always want to use `close()` to release the file resource after we are done with the file. Let's see what happens if we do not use `close()`:

Without close()

```
f1 = open('file.txt', 'w')
f1.write("Hello World! from f1\n")
```

```
cat file.txt
```

It is still an empty file. Nothing was written.

Use with statement

```
with open('file.txt', 'w') as f1:
    f1.write("Hello World! from f1\n")
```

In Python, we can use `with` statement to automatically close the file

```
cat file.txt
# Hello World! from f1
```

Always Close the File - Another Example

Here is another case without using `close()` properly where you have more than two file streams opened at the same time:

f1.close() first

```
# create the first file stream
f1 = open('file.txt', 'w')
f1.write("--- F1 ---\n")
# create the second file stream
f2 = open('file.txt', 'w')
f2.write("--- F2F2F2 ---\n")
# close the first file stream
f1.close()
```

f2.close() later

```
# close the second file stream
f2.close()
```

Successfully saved the content from f1

```
cat file.txt
# --- F1 ---
```

The file content was overwritten by f2

```
cat file.txt
# --- F2F2F2 ---
```

Append a File

3.2 Append to a file (mode a)

The a mode is similar to the w mode, except that it will append the content to the end of the file instead of overwriting the existing content.

Mode w

```
f = open('file.txt', 'w')
f.write('Hello world!\n')
f.write('This is the second line\n')
f.write('This is the third line\n')
f.close()
```

```
cat file.txt
# Hello world!
# This is the second line
# This is the third line
```

Mode a

```
f = open('file.txt', 'a')
f.write('This is the fourth line\n')
f.write('This is the fifth line\n')
f.close()
```

```
cat file.txt
# Hello world!
# This is the second line
# This is the third line
# This is the fourth line
# This is the fifth line
```

Read a File - Read()

3.3 Read a file (mode `r`)

The `r` mode is used to read the content of a file. There are two ways to read the content of a file:

- `read()` : read the entire content of the file.
- `readlines()`: store each line of the file as an element in a list.

Read a File - Read()

read()

One single line

```
f = open('file.txt', 'r')
content = f.read()
print(content)
f.close()
# output: 'Hello world!\nThis ia the second line\nThis ia the third
line\n'
```

readlines()

Multiple lines separated by EOL

```
f = open('file.txt', 'r')
lines = f.readlines()
print(lines)
f.close()
# output: ['Hello world!\n', 'This ia the second line\n', 'This ia the
third line\n']
```

```
len(lines)
# output: 3
```

4. String processing

String processing is an essential skill to extract meaningful information from a text-based file. In this section, we will go through common string processing methods.

```
file_list = [
            "2022/09/08_trialA_trt1.txt",
            "2022/09/15_trialA_trt2.txt",
            "2022/09/28_trialB_trt1.txt",
            "2022/09/30_trialB_trt2.txt",
            "2022/10/01_trialC_trt1.txt",
            "2022/10/08_trialC_trt2.txt"]
```

indexing

```
file_list[0]
# output: '2022/09/08_trialA_trt1.txt'
```

Negative index + slicing

```
file_list[0][-3:]
# output: 'txt'
```

A for loop

```
extension_list = []
for file in file_list:
    extension_list.append(file[-3:])
print(extension_list)
# output: ['txt', 'txt', 'txt', 'txt', 'txt', 'txt']
```

String Split

4.2 String split

Now, we want to break down each string to smaller pieces of information such as trials and treatments. We noticed that they were separated by the underscore _. Hence, we can use the `split()` method that takes the separator (i.e., _) as the argument to split the string:

```
filename = file_list[0] # '2022/09/08_trialA_trt1.txt'
filename.split('_')
# output: ['2022/09/08', 'trialA', 'trt1.txt']
```

String Split - Extract Date Information

yyyy/mm/dd

```
filename = file_list[0] # '2022/09/08_trialA_trt1.txt'
elements = filename.split('_')
print(elements)
# output: ['2022/09/08', 'trialA', 'trt1.txt']
date = elements[0]
print(date)
# output: '2022/09/08'
```

yyyy, mm, dd

```
yyyymmdd = date.split("/")
print(yyyymmdd)
# output: ['2022', '09', '08']

year = yyyymmdd[0]
month = yyyymmdd[1]
day = yyyymmdd[2]
```

```
print("File name = " + filename)
print("Year = " + year)
print("Month = " + month)
print("Day = " + day)
# output:
# File name = 2022/09/08_trialA_trt1.txt
# Year = 2022
# Month = 09
# Day = 08
```

String Replace - Extract Treatment Info

4.3 String replace

From the split result, we can see that the treatment information is not in the format we wnat. Eech substring `trt1.txt` still tails with the file extension name `.txt`.

```
filename = file_list[0] # '2022/09/08_trialA_trt1.txt'
elements = filename.split('_')
print(elements)
# output: ['2022/09/08', 'trialA', 'trt1.txt']
trt = elements[2]
print(trt)
# output: 'trt1.txt'
```

Slicing

```
new_trt = trt[:-4]
print(new_trt)
# output: 'trt1'
```

replace

```
new_trt = trt.replace('.txt', '')
print(new_trt)
# output: 'trt1'
```

4.4 Regular expression (RE)

Regular expression (RE) is a more flexible pattern recognition method than simple string slicing or replacement. We will introduce the `re` module that provides a set of methods to implement regular expression.

As always, we need to import the `re` module first:

```
import re
```

The `re` module provides a set of methods to implement regular expression. In this lab, we will only use `re.findall()` method, which returns a list of all the non-overlapping matches in the string. The syntax of the method is:

```
re.findall(pattern, string)
```

Regular Expression - Pattern Table

Pattern	Description
`.`	Matches any character except newline.
`^`	Matches the start of the string.
`$`	Matches the end of the string or just before the newline at the end of the string.
`*`	Matches 0 or more repetitions (greedy) of the preceding RE.
`+`	Matches 1 or more repetitions (greedy) of the preceding RE.
`?`	Matches 0 or 1 (greedy) of the preceding RE.
`*?`	Matches 0 or more repetitions (non-greedy) of the preceding RE.
`+?`	Matches 1 or more repetitions (non-greedy) of the preceding RE.
`??`	Matches 0 or 1 (non-greedy) of the preceding RE.
`\w`	Matches any alphanumeric character including the underscore.
`\W`	Matches any non-alphanumeric character.
`\d`	Matches any numeric digit.
`\D`	Matches any non-numeric digit.
`\s`	Matches any whitespace character.
`\S`	Matches any non-whitespace character.
`[]`	Matches any character in the brackets.
`{m}`	Matches exactly m copies of the previous RE.
`{m,n}`	Matches from m to n (inclusive) copies of the previous RE.

Regular Expression - Find all Numbers

4.4.1 Find all numbers

Example sentence from the USDA web page

https://www.aphis.usda.gov/aphis/ourfocus/animalhealth/animal-disease-information/cattle-disease-information/cattle-surveillance

\d	Matches any numeric digit.
+	Matches 1 or more repetitions (greedy) of the preceding RE.

```
From 2006 to 2015, the BSE Ongoing Surveillance Program tested
approximately 40,000 samples per year.
```

```
string = "From 2006 to 2015, the BSE Ongoing Surveillance Program tested
approximately 40,000 samples per year."
re.findall(r"\d+", string)
# output: ['2006', '2015', '40', '000']
```

Regular Expression - Find all Numbers


```
From 2006 to 2015, the BSE Ongoing Surveillance Program tested
approximately 40,000 samples per year.
```

All numbers

```
string = "From 2006 to 2015, the BSE Ongoing Surveillance Program tested
approximately 40,000 samples per year."
re.findall(r"\d+", string)
# output: ['2006', '2015', '40', '000']
```

All 4-digit numbers

```
re.findall(r"\d{4}", string)
# output: ['2006', '2015']
```

\d	Matches any numeric digit.
+	Matches 1 or more repetitions (greedy) of the preceding RE.
{m}	Matches exactly m copies of the previous RE.
{m,n}	Matches from m to n (inclusive) copies of the previous RE.

Regular Expression - String Pattern

4.4.2 Find numbers that match a string pattern

Here is another example sentence

https://www.aphis.usda.gov/animal_health/animal_diseases/brucellosis/downloads/fpa-val-rpt.pdf

```
In fact, these samples had an average reading of 254 mP at the Texas
laboratory compared to the average negative control reading of 88 mP. The
FP also clearly identified the four sero-negative samples (samples 1, 7,
17, and 19) as negative.
```

All numbers

```
re.findall(r"\d+", string)
# output: ['254', '88', '1', '7', '17', '19']
```

numbers + "mP"

```
re.findall(r"\d+ mP", string)
# output: ['254 mP', '88 mP']
```

Regular Expression - Greedy vs Non-Greedy Matching

Let's take a look of this paragraph

```
The FP for B. abortus relies on an O-polysaccharide from B. abortus that
is covalently linked with a fluorescein isothiocyanate tracer molecule.
Infected cattle were those from which B. abortus had been cultured from
milk or tissues. Serum samples were also assayed from B. abortus strain 19
vaccinated cattle at various times post vaccination. The FP test was shown
to accurately classify uninfected cattle as negative, infected cattle as
positive, and strain 19 vaccinated cattle as negative.
```

Say we want to extract every sentence that contains the word "B. abortus". We can design a RE pattern like this:

Escape character

.	Matches any character except newline.
*	Matches 0 or more repetitions (greedy) of the preceding RE.

```
re.findall(r"B\. abortus.*\.", string)
# output: B. abortus relies on an ... cattle as negative.
```

Regular Expression - Greedy vs Non-Greedy Matching


```
The FP for B. abortus relies on an O-polysaccharide from B. abortus that
is covalently linked with a fluorescein isothiocyanate tracer molecule.
Infected cattle were those from which B. abortus had been cultured from
milk or tissues. Serum samples were also assayed from B. abortus strain 19
vaccinated cattle at various times post vaccination. The FP test was shown
to accurately classify uninfected cattle as negative, infected cattle as
positive, and strain 19 vaccinated cattle as negative.
```

`.`	Matches any character except newline.
`*`	Matches 0 or more repetitions (greedy) of the preceding RE.
`*?`	Matches 0 or more repetitions (non-greedy) of the preceding RE.

Greedy

Return an entire paragraph

```
re.findall(r"B\. abortus.*\.", string)
# output: B. abortus relies on an ... cattle as negative.
```

Non-greedy

Return three sentences in a list

```
re.findall(r".*?B. abortus.*?\.", string)
# output [
# 'The FP for B. abortus relies on an O-polysaccharide from B.',
# ' abortus that is covalently linked with a fluorescein isothiocyanate
tracer molecule. Infected cattle were those from which B. abortus had been
cultured from milk or tissues.',
# ' Serum samples were also assayed from B. abortus strain 19 vaccinated
cattle at various times post vaccination.']
```

Not a period!

Regular Expression - Exception


```
The FP for B. abortus relies on an O-polysaccharide from B. abortus that
is covalently linked with a fluorescein isothiocyanate tracer molecule.
Infected cattle were those from which B. abortus had been cultured from
```

```
re.findall(r".*?B. abortus.*?\.", string)
# output [
# 'The FP for B. abortus relies on an O-polysaccharide from B.',
# ' abortus that is covalently linked with a fluorescein isothiocyanate
tracer molecule. Infected cattle were those from which B. abortus had been
cultured from milk or tissues.',
```

```
re.findall(r".*?B. abortus.*?[^B]\.", string)
# output: [
# 'The FP for B. abortus relies on an O-polysaccharide from B. abortus
that is covalently linked with a fluorescein isothiocyanate tracer
molecule.',
# ' Infected cattle were those from which B. abortus had been cultured
from milk or tissues.',
```

Regular Expression - Exercise

4.4.5 Another example

```
filelines = [
    "File name: 2022/09/08_trialA_trt1.txt",
    "treatment A: (1, 3) and (2, 4)",
    "treatment B: (5, 1, 8)",
    "treatment C: (2, 4), (1, 9, 8)"]
```

The data `filelines` was intentionally made to be a little bit messy. We want to extract the treatment information in each pair of parenthesis (e.g., `(1, 3)` and `(1, 9, 8)`):