Text manipulation with Python

•••

UW Libraries Python Interest Group • April, 2024

Who’s John Borwick?

Director, IT Services & Digital Strategies

Programming in Python on and off since ~2004

- API work (e.g. with Microsoft Graph API, EDMS)
- Bulk data conversions
- Django (Python web framework)
- Extract/Transform/Load (ETL) work

Thank you!

Thanks to Jeff Sherwood, who contributed to and reviewed these slides!

Thanks to everyone who took the optional survey!

Using Python 3.11

```
$ python
Python 3.11.8 (main, Feb 10 2024,
12:35:50) [Clang 15.0.0
(clang-1500.1.0.2.5)] on darwin
Type "help", "copyright", "credits" or
"license" for more information.
>>>
```

Agenda

- Strings
- Getting to where you can use strings
- String methods
- Regular expressions
- Review

Strings

Python 3 strings: Unicode-aware

- Strings are *not* bytes anymore
- Strings have encodings, e.g.:
  - `utf-8`
  - `latin-1`
- FYI Python has a “codecs” list

Creating strings

```
>>> s1 = "Test"
>>> s2 = 'Test'
>>> s3 = """Test"""
>>> s1 == s2 == s3
True
```

Formatting strings

```
>>> name = "John"
>>> s1 = "Hello, " + name
>>> s2 = "Hello, {}".format(name)
>>> s3 = f"Hello, {name}"
>>> s1 == s2 == s3
True
```

Strings are arrays

```
>>> s1 = "hello”
>>> s1[0]
‘h’
>>> s1[0:2]
‘he’
>>> s1[-1]
‘o’
>>> s1[-1::-1]
‘olleh’
```

Removing the first N characters

```
>>> s1 = "1234Hello"
>>> s1[4:]
'Hello'
>>> s2 = "Hello1234"
>>> s2[:-4]
'Hello'
```

Getting to where you can use strings

Opening a file and printing the first character of each line

```
>>> with open('test.txt',
...           encoding='utf-8') as file_h:
...     for line in file_h:
...         print(line[0])
...
```

Opening a CSV file and printing the first column

```
>>> import csv
>>> with open('test.csv',
...           encoding='utf-8') as file_h:
...     csv_reader = csv.reader(file_h)
...     for row in csv_reader:
...         col1 = row[0]
```

Many, many Python libraries will help you get strings

- `openpyxl` (one of several Exel options)
- `json`
- `requests`
- `xml.etree.ElementTree`

String methods

String methods

```
>>> s1 = "hello”
>>> dir(s1)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__getstate__', '__gt__', '__hash__',
'__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
'__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index',
'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier',
'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper',
'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition',
'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith',
'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

Learning about string methods

```
>>> s1 = "hello”
>>> help(s1.isdigit)
Help on built-in function isdigit:

isdigit() method of builtins.str instance
    Return True if the string is a digit string,
False otherwise.

    A string is a digit string if all characters
in the string are digits and there
    is at least one character in the string.
```

Cleaning up strings with `strip`

```
>>> s1 = "   hello   \n"
>>> s1.strip()
'hello'
>>> s1.rstrip()
'   hello'
>>> s1.lstrip()
'hello   \n'
```

Fixing/setting capitalization

```
>>> s1 = "HeLlO"
>>> s1.upper()
'HELLO'
>>> s1.lower()
'hello'
>>> s1.capitalize()
'Hello'
```

Replacing simple matches

```
>>> s1="cats are cats"
>>> s1.replace("cats", "dogs")
'dogs are dogs'
>>> s1.replace("cats", "dogs", 1)
'dogs are cats'
```

FYI there are tools for replacing specific characters

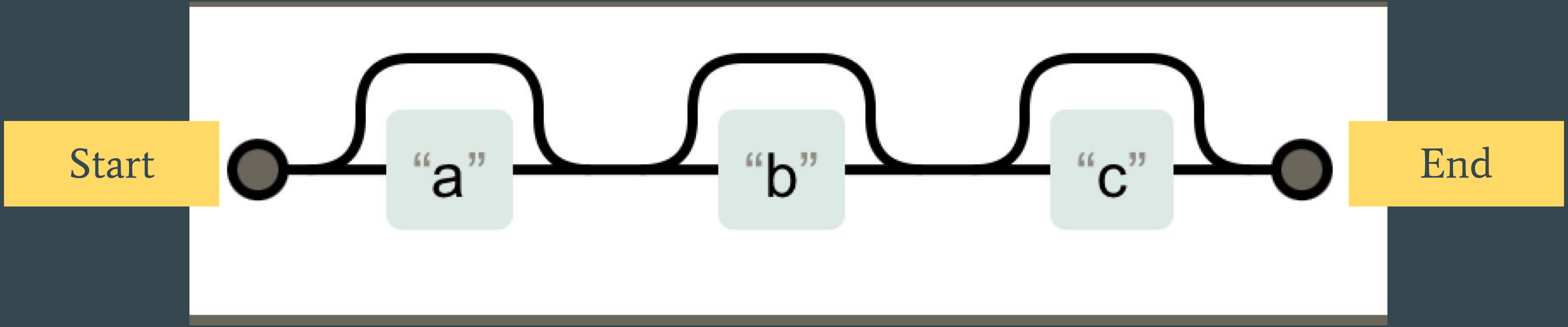
- `maketrans`
- `codecs`

Regular expressions

What are regular expressions?

They search through text.

Term comes from computer science.



https://regexper.com/#a%3Fb%3Fc%3F

Using regular expressions: search vs. match

```
>>> import re
>>> re.search("h", "This")
<re.Match object; span=(1, 2), match='h'>
>>> re.match("h", "This")
```

Compiling a regular expression

```
>>> H_RE = re.compile("h")
>>> if H_RE.search("This"):
...     print("There's an h!")
...
There's an h!
```

Getting fancy: $ by default means “end of string”

```
>>> ENDS_YES_RE = re.compile(r"yes$")
>>> if ENDS_YES_RE.search("  yes"):
...   print("Ends with yes!")
...
Ends with yes!
>>>
```

Selected regular expression symbols

```
^  : beginning of line
$  : end of line
.  : Virtually any character
*  : 0 or more of the previous thing
+  : 1 or more of the previous thing
[] : set of characters e.g. “[a-c]”
() : group of stuff e.g. “(abc)*”
```

Selected regular expression symbols

```
\w : Word character ([A-Za-z0-9_])
\s : Space character (e.g. tab)
```

Aside: “Raw” strings in Python

```
>>> s1 = "A\b"
>>> s1
'A\x08'
>>> s2 = r"A\b"
>>> s2
'A\\b'
```

Learning more about regular expressions

https://regexcrossword.com

https://regex101.com

https://ihateregex.io

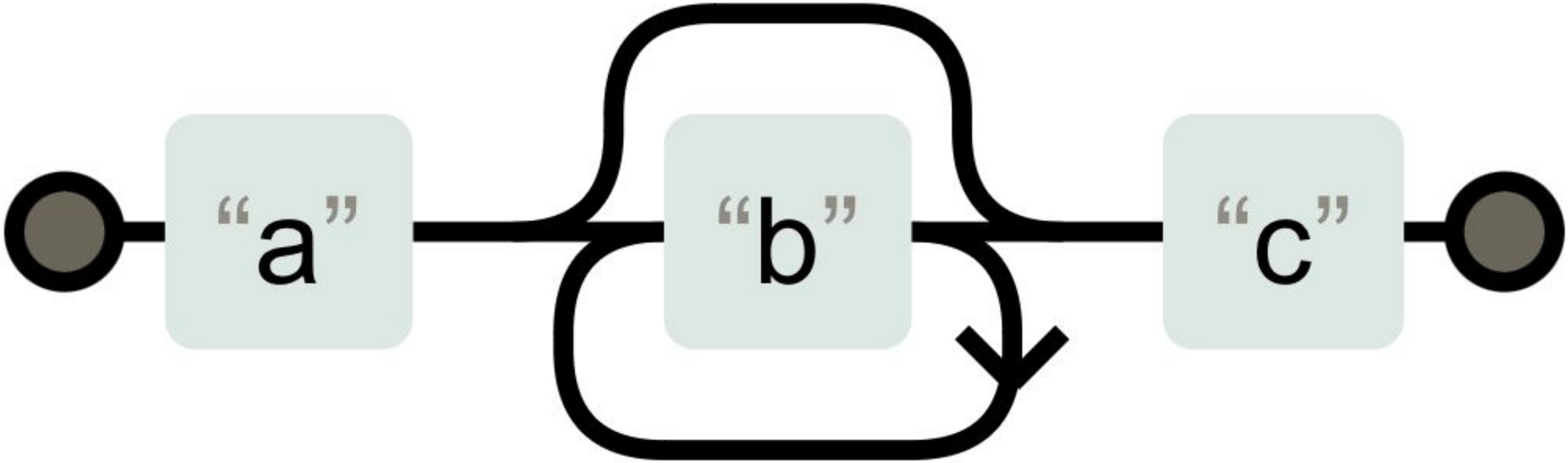
https://regexone.com

https://regexper.com

n.b. many programmers dread regular expressions

Visualizing regular expressions

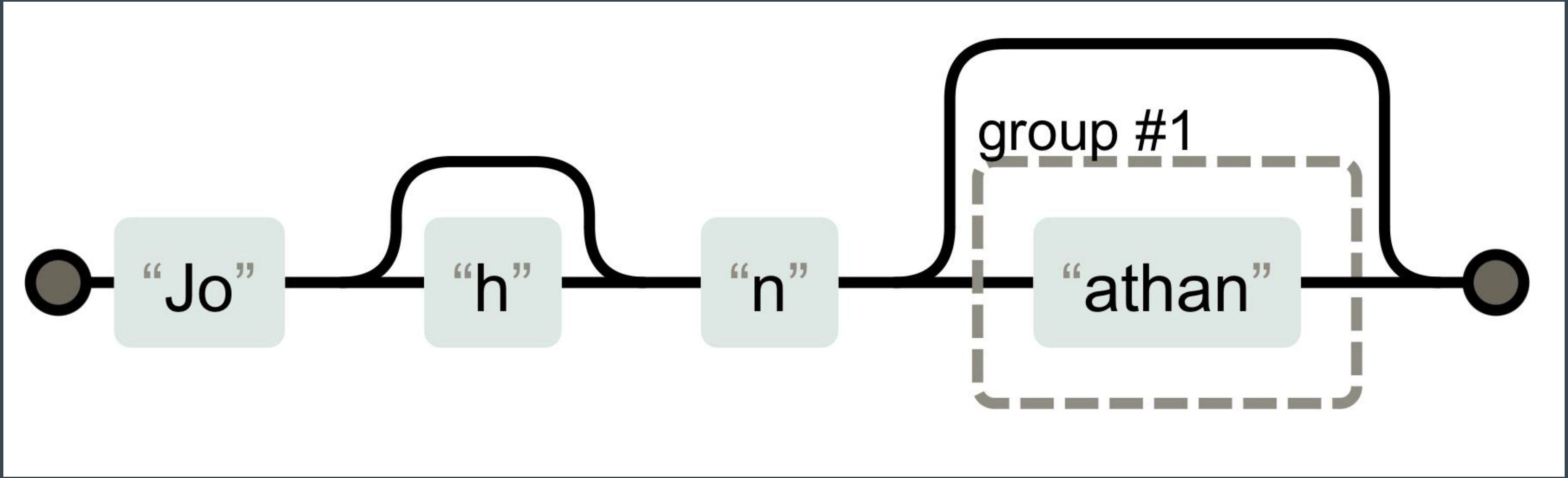
ab*c



https://regexper.com/#ab*c

Named John?

```
Joh?n(athan)?
```



Named John?

```
>>> JOHN_RE = re.compile("Joh?n(athan)?")
>>> if JOHN_RE.search("John"):
...     print("Contains John")
...
Contains John
```

Regular expression flags: `re.VERBOSE`

```
>>> JOHN_RE = re.compile(r"""
  Jo              # chars ‘Jo’
  h?              # maybe an ‘h’
  n               # always an ‘n’
  (athan)?        # maybe ‘athan’
""",
  re.VERBOSE)
```

Iterating over matches

```
>>> for match in JOHN_RE.finditer("I 
talked with John and Jon"):
...      print(match.group())
...
John
Jon
>>>
```

Replacing text

```
>>> JOHN_RE.sub(
  "[censored]",
  "I talked with John and Jon")
'I talked with [censored] and [censored]'
>>>
```

Replacing text

```
>>> JOHN_RE.sub(
  lambda match: match.group().upper(),
  "I talked with John and Jon")
'I talked with JOHN and JON'
```

Review

Agenda review

- Strings
- String methods
- Regular expressions