UC Berkeley
SCHOOL OF INFORMATION

Week 09

Working with text & binary data

Programming for Data Science

Everything bits & bytes; preparing, too, for data cleaning and analysis

March 05, 2020

Agenda

UC Berkeley
SCHOOL OF INFORMATION

- Number representations
- Text Representations: ASCII, Unicode, Python Strings
- Text Encoding (codex)
- Regular Expressions (regex)
- Text Output
- Files
- Intro to NumPy

Schedule

UC Berkeley
SCHOOL OF INFORMATION

9	3-Mar	4-Mar	5-Mar	7-Mar	Unit 9	Text and Binary Data					Exam 1		
10	10-Mar	11-Mar	12-Mar	14-Mar	Unit 10	NumPy - Vectors	Project 1 Presentation	HW unit 9					Project 1 Code
11	17-Mar	18-Mar	19-Mar	21-Mar	Unit 11	Pandas - Dataframes		HW unit 10	HW unit 9			Project 2	
	24-Mar	25-Mar	26-Mar	28-Mar		Spring Break - no classes!							
12	31-Mar	1-Apr	2-Apr	4-Apr	Unit 12	MatPlotLib - Data Visualization		HW on units 11-13	HW unit 10				Project 2 Proposal
13	7-Apr	8-Apr	9-Apr	11-Apr	Unit 13	Advanced Pandas - Aggregation & Groups			HW units 11-13	Exam 2			
14	14-Apr	15-Apr	16-Apr	18-Apr	Unit 14	Testing	Project 2 Presentation				Exam 2		Project 2 Report

Calendar

- 10: NumPy
- 11: Data Analysis
- 12: More Analysis
- 13: Testing
- Team Project Presentation & Exam 2

For Project 1:

UC Berkeley
SCHOOL OF INFORMATION

- Present your project at a high level: share your screen, a couple of slides, if you'd like
- Practice your presentation: what classes did you use to solve your coding problem? Major challenges?
- Solutions?
- Don't just read thru the code.
- Practice(!) and present in no more than 5 minutes: communicate your project to others …. Classmates are encouraged to ask questions.

Drilling down to bits & bytes

UC Berkeley
SCHOOL OF INFORMATION

- Base 2: Binary (0s and 1s)
- Base 8: Octal (0,1,2,3,4,5,6,7)
- Base 10: Decimal (0,1,2,3,4,5,6,7,8,9)
- Base 16: Hexadecimal (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)

8-bits = byte
4-bits = nibble
≥ 8 = multibyte

Converting the text "hope" into binary

Characters:	h	o	p	e
ASCII Values:	104	111	112	101
Binary Values:	01101000	01101111	01110000	01100101
Bits:	8	8	8	8

ComputerHope.com

Number representations

UC Berkeley
SCHOOL OF INFORMATION

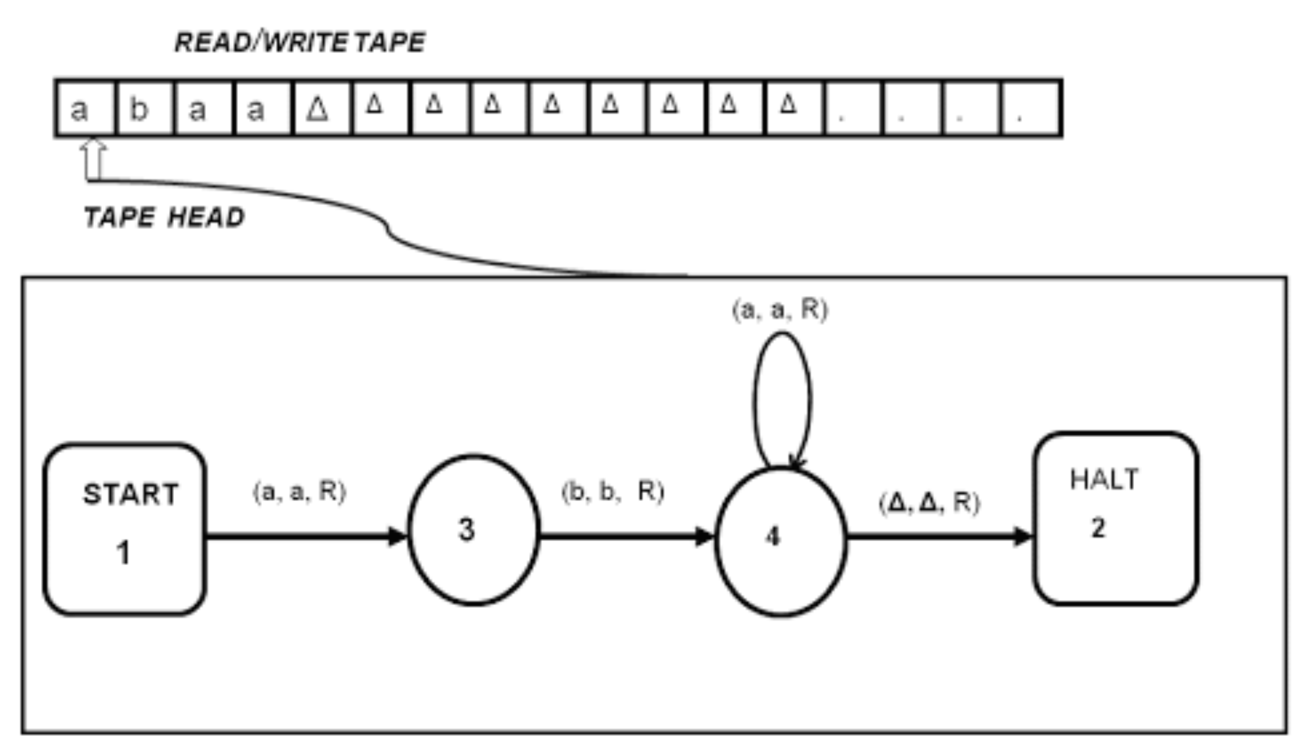
- 0000 = 0 [zero in all places: 0 0 0 0]
- 0001 = 1 [1 x 2^0; 1 in the "one's" place]
- 0010 = 2 [1 x 2^1; 1 in the "two's" place]
- 0100 = 4 [1 x 2^2; 1 in the "four's" place]
- 1000 = 8 [1 x 2^3; 1 in the eight's place]

- 1100 = 12 [1 x 2^3 + 1 x 2^2; 1 in the eight's and four's places]

Number representations: octal

- 000 = 0
- 001 = 1
- 010 = 2
- 011 = 3
- 100 = 4
- 101 = 5
- 011 = 6
- 111 = 7

- Setting file permissions in octal: read, write, execute
  - read/write $1 + 2 = 2^0 + 2^1 = 3$
  - read./execute $1 + 4 = 2^0 + 2^2 = 5$
  - read/write/execute $1 + 2 + 4 = 2^0 + 2^1 + 2^2 = 7$
  - So what is 755?



A Turing Machine for aba*

Why? because a Turing machine can put only one symbol at a time in the tape-head position; hence need one symbol.

Number reps: hexadecimal

UC Berkeley
SCHOOL OF INFORMATION

- 000 = 0
- 005 = 5 (5 x 16^0)
- 050 = 16 (5 x 16^1)
- 500 = 1280 (5 x 16^2)
- The alphabet is 0 1 2 3 4 5 6 7 8 9 A B C D E F

	Location					
	6	5	4	3	2	1
Value	1048576 (16^5)	65536 (16^4)	4096 (16^3)	256 (16^2)	16(16^1)	1 (16^0)

UC Berkeley
SCHOOL OF INFORMATION

Text Representations: Unicode

- Every grapheme has a unique identifying number and unique identifying name. Can be represented in hex; some historical characters require the GID (Glyph IDentifier).
- Grapheme forms are linguistically- and culturally-sensitive

हिंदी ihⁿdī

/u093F/u0939/u0902/u0926/u0940

i h n d ī

/u0939/u0907/u0928/u0926/u0940

h i n d i

Notice the differences between code points and position of grapheme.

مرحبا

marhabaan

م مم مامم

M mm mm

Cultural realia expressed in code.

ﷺ

辵 辶

walk; and form in combination with other characters.

different code points for the same concept/character

明娃

hao

Each grapheme has its own code - so the byte string varies based on the linguistic context! This is a challenge in compression and text processing 'cause tonal languages (Thai, Vietnamese, etc.), Arabic/Hebrew, and CJK languages often use bytes instead of https://en.wikipedia.org/wiki/ASCII tokens (words).

Why does this matter to you? What if the file stream were "ascii-us" or "JIS" or "IIS" or "Big5" or … and so on. There's a combo of non-printable characters (control chars) and bytes that make processing a challenge. Used in automatic language and file-format detection.

Text Representations

https://en.wikipedia.org/wiki/ASCII

- ASCII (7-bit, 27 = 128 chars; also 8-bit ASCII "extended char set"
- Unicode (120,000+ characters; Python has native Unicode support). Import `unicodedata.name()` to return name from a value: e.g., Literal value "`B`" or Unicode value "`/u0042`" returns "`LATIN CAPITAL LETTER B`" (and can go the other way, too)
- Text can return boolean when compared:
  - "\u0047\u0072\u0072\u0021" == 'Grr!' is True
  - "\u0047\u0072\u0072\u0021" == 'GRR!' is False

http://unicode.org/charts/charindex.html

https://www.sciencebuddies.org/science-fair-projects/references/table-of-8-bit-ascii-character-codes

Text Encodings

UC Berkeley
SCHOOL OF INFORMATION

- UTF-8: the multibyte implementation of Unicode, a descriptive standard. Use encode/decode

```
>>> s.encode('utf-8')
b'\xe3\x88\xb2'
>>> s.encode('unicode_escape')
b'\\u3232'
```

- b prefix denotes bitwise encoding
- \x prefix denotes hex.
- We're not always sure what the data will be like when integrating heterogeneous data sources … Win10 might create files in UTF-10!
- Check out the documentation: https://docs.python.org/3/howto/unicode.html

https://docs.python.org/2.4/lib/standard-encodings.html

RegEx: regular expressions

UC Berkeley
SCHOOL OF INFORMATION

There's a debate about time improvement and binding var name to data. No clear preference in practice.

- `re.compile()` — compile a search string
- `re.search()` — get the first match
- `re.match()` — extract match (at beginning)
- `re.split()` — split based on matches
- `re.sub()`
- `re.findall()` — get all matches as a list
- `.group()` — used after matching to pull out groups

- **matchObject** = re.search(*pattern*, *input_str*, *flags*=0)
- https://regexone.com/references/python

RegEx: Special characters

UC Berkeley
SCHOOL OF INFORMATION

```
.        any character 1 place
*        any number of characters
?        any optional character
[0-9]    any digit
[a-z]    any lower-case letter
r’ ‘     raw string literal
```

https://pythonforbiologists.com/regular-expressions

RegEx: Specifiers & Example

UC Berkeley
SCHOOL OF INFORMATION

Pattern	Matches
\d	a single digit
\D	a single non-digit
\w	an alphanumeric character
\W	a non-alphanumeric character
\s	a whitespace character
\S	a non-whitespace character
\b	a word boundary (between a \w and a \W, in either order)
\B	a non-word boundary

Pattern	Matches
abc	literal abc
(expr)	expr
expr1 | expr2	expr1 or expr2
.	any character except \n
^	start of source string
$	end of source string
prev ?	zero or one prev
prev *	zero or more prev, as many as possible
prev *?	zero or more prev, as few as possible
prev +	one or more prev, as many as possible
prev +?	one or more prev, as few as possible
prev { m }	m consecutive prev
prev { m, n }	m to n consecutive prev, as many as possible
prev { m, n }?	m to n consecutive prev, as few as possible
[abc]	a or b or c (same as a|b|c)
[^ abc]	not (a or b or c)
prev (?= next)	prev if followed by next
prev (?! next)	prev if not followed by next
(?<= prev) next	next if preceded by prev
(?<! prev) next	next if not preceded by prev

```
import re
# Lets create a pattern and extract some information with it
regex = re.compile(r"(\w+) World")
result = regex.search("Hello World is the easiest")
if result:
    # This will print:
    #   0 11
    # for the start and end of the match
    print(result.start(), result.end())

# This will print:
#   Hello
#   Bonjour
# for each of the captured groups that matched
for result in regex.findall("Hello World, Bonjour World"):
    print(result)

# This will substitute "World" with "Earth" and print:
#   Hello Earth
print(regex.sub(r"\1 Earth", "Hello World"))
```

https://www.machinelearningplus.com/python/python-regex-tutorial-examples/

RegEx: Examples

Usually find options locate *first* or *last* occurrence but we may want *every* occurrence. Some find options *return the index* of the desired substring; this requires more coding to profit. Patterns are possible, too, and this is useful for known/expected structuring of the source data.

```
middle_pattern = re.compile("that is")
m = middle_pattern.search("that is")

if m:
    print(m.group())
```

```
that is
```

```
n_pattern = re.compile("n") #Lets find all of the n's
m = n_pattern.findall(source)
print("Found", len(m), "matches")
print(m)
```

```
Found 2 matches
['n', 'n']
```

```
phone_number_pattern = re.compile(r'\d{3}-\d{3}-\d{4}')
```

```
(r'[0123456789]{3}-[0123456789]{3}-[0123456789]{4}')
```

RegEx: Groups

UC Berkeley
SCHOOL OF INFORMATION

```
phone_number_pattern = re.compile(r'(\d{3})-(\d{3}-\d{4})')
m = phone_number_pattern.search(large_source)

if m:
    print(m.group())
    print(m.groups())
```

```
650-555-3948
('650', '555-3948')
```

```
phone_number_pattern = re.compile(r'(?P<areacode>\d{3})-(?P<number>\d{3}-\d{4})')
m = phone_number_pattern.search(large_source)

if m:
    print(m.group("areacode"))
    print(m.group("number"))
```

```
650
555-3948
```

Example: biology

UC Berkeley
SCHOOL OF INFORMATION

```
dna = “ATCGCGAATTCAC”
if re.search(r”GAATTC”, dna):
    print(“restriction site found.”)

if re.search(r”GGACC”, dna)
   or re.search(r”GGTCC”, dna):
   print(restriction site found)

if re.search(r”GG(A|T)CC”, dna):
    print(“Yup, another restriction site.”)

if re.search(r”GG[ATGC]GC”, dna):
     print(“Restrictions…”)
```

? optional, e.g., GAT?C matches GATC or GAC

GGG(AAA)?TTT means the group of 3 s is optional. matches GGGAAATTT or GGGTTT

- + means char/group must be present and can be repeated
- * following a group/char means group/char is optional but can be repeated 0+ times

| is the “pipe” or “either|or”

https://pythonforbiologists.com/regular-expressions

Example: biology

Position commands:

`{start,stop}`. GA{2,4}T matches G then 2 to 4 As, then T.

^ matches the start of a string; $ matches the end of a string.

^AAA matches AAATTT but not GGGAAATTT

GGG$ matches AAAGGG but not AAAGGGCCC

```
^AUG[AUGC]{30,1000}A{5,10}$
```

This complex pattern will identify full-length eukaryotic messenger RNA sequences. Reading the pattern from left to right, it will match:

- an AUG start codon at the beginning of the sequence
- followed by between 30 and 1000 bases, which can be A, U, G or C
- followed by a poly-A tail of between 5 and 10 bases at the end of the sequence

Text output using .format()

Let's say `s = 'Cat'`

```
print("Some text: {sentence: <20s}".format(sentence = s))

print("Some text: {sentence}".format(sentence = s))

print("Some text: {}".format(s))
```

```
>>> print("This is my text: {sentence: <20s}".format(sentence = s))
This is my text: Cat
>>> print("This is my text: {sentence}".format(sentence = s))
This is my text: Cat
>>> print("This is my text: {}".format(s))
This is my text: Cat
```

Files!

UC Berkeley
SCHOOL OF INFORMATION

- All languages have (seemingly generic) file objects for reading/writing; and have variations depending on file type and architecture (e.g., FileObjects and various stream options).
- `open(file, mode)` ('wt','rd','at','rb', 'wb');
  - be sure to `close()`!
- `with()` command; don't need to close
- `write()` `read()` `readlines()` `readline()`

```
try:
    with open("words.txt", "r", encoding = 'utf-8', newline = None) as fd:
        lines = fd.readlines()
        lines = [w.replace("\n","") for w in lines]
        sorted_lines = lines.sort()
```

https://docs.python.org/3/tutorial/inputoutput.html

https://docs.python.org/3/c-api/file.html

Breakout Rooms

- To practice reading files (.json), string parsing, and a stream

NumPy: quick intro

UC Berkeley
SCHOOL OF INFORMATION

- work with n-dimensional arrays of numeric data
- Works with Pandas to provide a more user-friendly experience. Use a “dataset” and include non-numeric variables.
- Using NumPy deepens knowledge of using python and other libraries.

NumPy

- https://timothyhelton.github.io/pandas_best_practices.html
- https://docs.scipy.org/doc/numpy/user/quickstart.html
- http://www.numpy.org

NumPy basic functions

UC Berkeley
SCHOOL OF INFORMATION

- `import numpy as np`
- `np.array()`
- `np.arange()`, `np.linspace()`
- Measures of Central Tendency: `np.min()`, `np.max()`, `np.std()`, `np.var()`
- `np.argmax()`, `np.argmin()`
- `np.shape()`, `np.reshape()`
- `np.zeros()` (more helpful than you'd think)
- `np.random.seed()`, `np.random.random_integers()`
- `np.vstack`, `np.hstack()`
- dealing with n-dimensions; `axis = 0.` or `axis = 1`

UC Berkeley
SCHOOL OF INFORMATION

Summary

- Numeric and Text Representations
- Encoding: ASCII, Unicode, Python Strings
- Text encoding/decoding
- Regular Expressions and examples
- Formatting text with .format()
- File I/O basics
- Breakout room processing tweets
- A look at NumPy basic functions