Class 10 - Text and Binary Data

[W200] MIDS Python Summer 2018

Class 10 Agenda

Data Encoding
Regular Expressions

Appendix: Outputting Text
Appendix: Handling Files

Schedule | The Second Half

Class 10 - Working with Text and Binary Data

Class 11 - NumPy

Class 12 - Data Analysis with Pandas

Class 13 - More Data Analysis with Pandas

Class 14 - Group work

Class 15 - Code Testing and Final Project Showcase

Schedule | Where we're going - projects/exams

Live Session 11 - Discuss Final Project
Live Session 12 - Proposal Finalized
Live Session 13 - Final Exam Distributed
Live Session 14 - Final Exam due, Projects Due, Final Project Showcase

https://docs.google.com/spreadsheets/d/11DxadnNwyFaJIPYLUJSPUlNGCtTenBCR4yaR1CbFBKg

Schedule | HW Update

You will have your regularly scheduled homework (Strings HW) this week

** look for the Homework in your upstream folder**

Grading | Reminder of Breakdown

1. Homework (30%)
2. Midterm (10%)
3. Project 1 (20%)
4. Final (10%)
5. Project 2 (20%)
6. Participation (10%)

Grading | Reminder of Breakdown

1. Homework (30%)
2. Midterm (10%)
3. Project 1 (20%)
4. Final (10%)
5. Project 2 (20%)
6. Participation (10%)

Class 10 Agenda

Handling encoded data
Finding outputting text
regular expressions
Handling files
Project 2 Intro

Project Showcase

Some questions that you may address

1. Explain your project at a high level
2. Share your screen, run your code, and show off what you've done!
3. Open up the code itself and discuss:
    a. What classes did you use to solve your problem?
    b. What were the major challenges of your implementation?

Class 10 Agenda

Data Encoding

Regular Expressions

Appendix: Outputting Text

Appendix: Handling Files

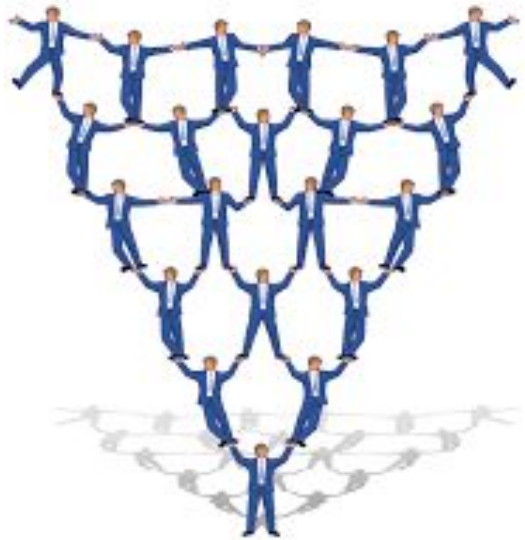
Pause

We have traveled up in levels of abstraction

- Fundamental types: ints, floats
- Container objects: lists , strings
- Classes

Now we will drill down

- Characters
- Bytes



http://kc-communications.com/case-inverted-pyramid/

The challenge is to translate

- Get these into python

* Perform meaningful operations on them

But how?

Encoding schemes

Everything in your computer stored as binary.

We know that at some level, text has to be translatable into this common format.

Encoding schemes

Everything in your computer stored as binary. We know that at some level, text has to be

- We think in terms of a base 10 system, how can we think in terms of base X

– 0000 is 0 zero in all places

– 0001 is 1 (1 x 2**0) one in the ones place

– 0010 is 2 (1 x 2**1) one in the twos place

– 0100 is 4 (1 x 2**2) one in the fours place

– 1000 is 8 (1 x 2**3) one in the eights place

– 1100 is 12 8 + 4

one in the eights place + one in the fours place

And so on…

In binary every number takes one of 2 values 0 or 1)

The places are multiples of 16 -> 1, 2, 4, 6, 8, 10

Encoding schemes

Computers also use the base 16 (hexadecimal) encoding scheme

e3=14 x 16 + 3 = 227

88=8 x 16 + 8 =131

b2=11 x 16 + 2 =178

Hexadecimal is the combination of

the base 6 “hexa” system
A-F = 10-15 (6 numbers)

And the familiar base 10 “decimal” system
0-9 = 0-9 (10 numbers)

Therefore
0-F = 0-15 (16 numbers)

Encoding schemes

- The choice of base 16 comes directly from the use of phrases binary strings; first 3 (Octal) and then 4 digit (Nybble, aka: hexadecimal)

3 digit binary phrase (Octal)

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

These 8 numbers represent one Octal digit that can take values 0-7

Similarly the 4 digit binary phrase (Nybble) has 16 values and represents one hexadecimal digit (0-F)

simple.wikipedia.org/wiki/Hexadecimal_numeral_system

Encoding schemes

Computers also use hexadecimal which is base 16,

– 000 is 0 zero in all places

– 005 is 5 (5 x 2**0) one in the 1s place

– 050 is 16 (5 x 16**1) one in the 16s place

– 500 is 1280 (5 x 16**2) one in the 256s place

	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)

In hexadecimal every number takes one of 16 values coded as 0-F

simple.wikipedia.org/wiki/Hexadecimal_numeral_system

Encoding schemes

ASCII uses 7 binary bits

2^7 = 128 characters

There are other encoding systems

What are some?

* Often incompatible *

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(null)	NUL	032	(space)	064	@	096	
001	☺	SOH	033	!	065	A	097	a
002	☻	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♦	EOT	036	$	068	D	100	d
005	♣	ENQ	037	%	069	E	101	e
006	♠	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	◘	BS	040	(	072	H	104	h
009	(tab)	HT	041	)	073	I	105	i
010	(line feed)	LF	042	*	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FF	044	,	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	♫	SO	046	.	078	N	110	n
015	☼	SI	047	/	079	O	111	o
016	►	DLE	048	0	080	P	112	p
017	◄	DC1	049	1	081	Q	113	q
018	↕	DC2	050	2	082	R	114	r
019	‼	DC3	051	3	083	S	115	s
020	¶	DC4	052	4	084	T	116	t
021	§	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↨	ETB	055	7	087	W	119	w
024	↑	CAN	056	8	088	X	120	x
025	↓	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[	123	{
028	(cursor right)	FS	060	<	092	\	124	|
029	(cursor left)	GS	061	=	093	]	125	}
030	(cursor up)	RS	062	>	094	^	126	~
031	(cursor down)	US	063	?	095	_	127	⌂

Encoding schemes

Unicode encodes 120k characters

Modern and ancient languages, math

UTF-8

Compatible with unicode

Python 3 has native support for unicode

Note: Windows 10 may create files using UTF-10. For example, if you try to create a file using the command line: echo “test” >> test.txt

If you run into encoding issues, this may be the cause.

Unicode

Python 3 has native support for unicode - All strings are Unicode strings!

```
>>> "\u0047\u0072\u0072\u0021" == 'Grr!'
```
True

```
>>> "\u0047\u0072\u0072\u0021" == 'GRR!'
```
False

```
print("Great!")
print("\u0047\u0072\u0065\u0061\u0074\u0021")
```

```
Great!
Great!
```

```
place = 'caf\u00e9'
print(place)
```

```
café
```

Unicode: value name pair

Every unicode value has a standard name

Unicodedata.name() to get name from a value

Value can be literal “B” or unicode value “/u0042”

Returns “LATIN CAPITAL LETTER B”

Encode and decode

You can often paste exotic characters

We can encode a text string in unicode using encode('utf-8'). Other options are available.

To decode unicode, use decode('utf-8')

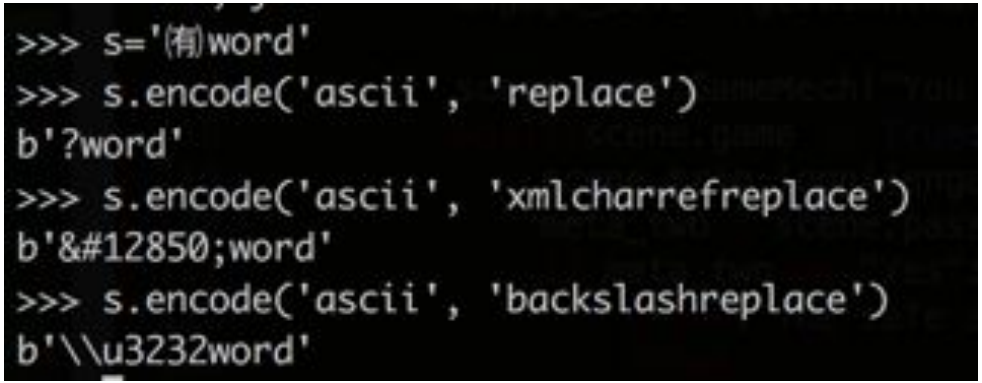
```
>>> s.encode('utf-8')
b'\xe3\x88\xb2'
>>> s.encode('unicode_escape')
b'\\u3232'
```

b denotes **bitwise encoding**
\x means hexidecimal

Encode and decode

Not all characters can be represented in each encoding scheme! You can specify how to handle this

1) Replace with blank (‘?’)
2) XML friendly
3) Unicode Escape (backslash)



Encoding/ decoding | methods/packages

Try the following commands:

- Unicodedata package
- encode (), decode()
- type (), len()

Class 10 Agenda

Data Encoding

Regular Expressions

Appendix: Outputting Text

Appendix: Handling Files

RegEx | Finding Text

1. re.compile() # compile a search string
2. re.search() # gets the first match
3. re.match() # extract match - if at beginning
4. re.split() # split on matches
5. re.sub() # substitute on matches
6. re.findall() # get all matches as list
7. .group() # used after matching to pull out groups

RegEx | Special Characters and Specifiers

1. . # any character 1 place
2. * # any number of char
3. ? # any character optional
4. [0-9] , /d # any digit
5. [a-z] # any letter lowercase letter
6. /w # any alpha-numeric char
7. r’ ’ # the raw string literal

Regular Expressions | Special Characters

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

RegEx | Specifiers

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

RegEx | Basic Examples

```
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']
```

RegEx | Phone number example

Compact version

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

expanded version

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

RegEx | Matching Groups

```
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')
```

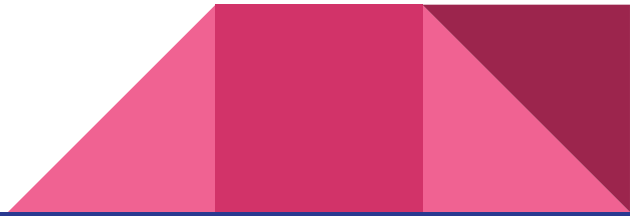
```
phone_number_pattern = re.compile(r'(?P<areacode>\d{3})-(?P<number>\d{3}-\d{4})')
m = phone_number_pattern.search(large_source)

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

```
650
555-3948
```

Class 10 Agenda

Project Showcase!
Data Encoding
Regular Expressions
Primer: Project 2 Intro

Appendix: Outputting Text
Appendix: Handling Files



Class 10 Agenda

Data Encoding
Regular Expressions
Primer: Project 2 Intro

Appendix: Outputting Text
Appendix: Handling Files

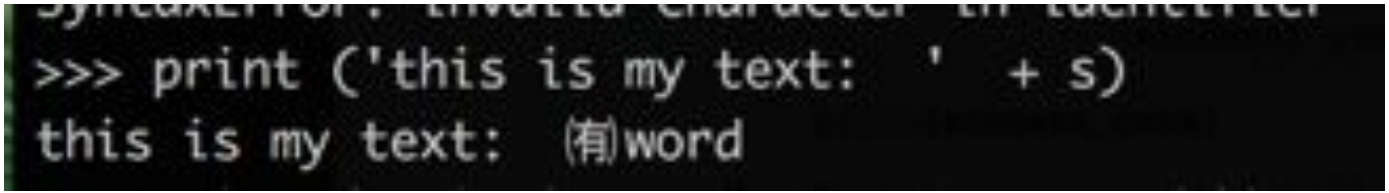
Text output | The Basics

Consider:

>>> s='㈲word'

Simple concatenation

print (‘this is my text: ’ + s)



Text output | The Basics

The old s(%) style

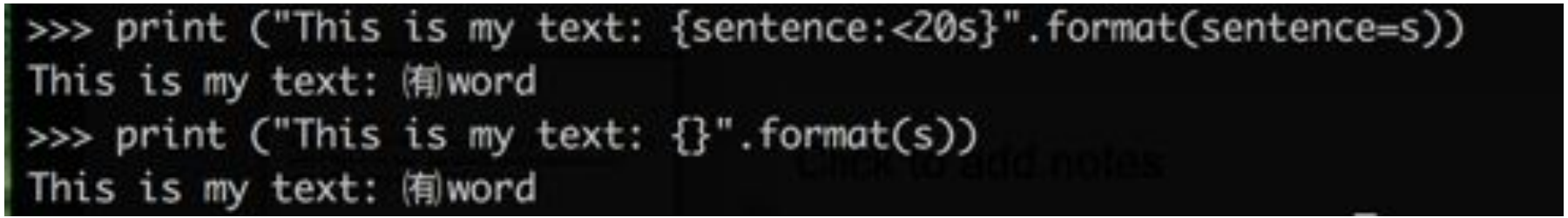
>>> print (‘this is my text: %10s ’ % (s))

```
>>> print ('this is my text: %s '  %  (s))
this is my text: ㈲word
>>> print ('this is my text: %10s '  %  (s))
this is my text:       ㈲word
```

Text output |

The new {} style

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



Class 10 Agenda

Project Showcase!
Data Encoding
Regular Expressions
Primer: Project 2 Intro

Appendix: Outputting Text
Appendix: Handling Files

The basic pattern | Python Functions

1. .open(file, mode),
   a. **open modes (‘wt’, ‘rt’, ‘at’, ’rb’, ‘wb’)**
2. **Action on file:**
   a. write(), read(), readlines(), readline()
3. .close()

Loading files | Python Functions

1. .open(file, mode), .close()
2. open modes (‘wt’, ‘rd’, ‘at’, ‘rb’, ’wb’)
3. with() # you don’t need to close this one
4. read()
5. readlines() # reads all lines as a list
6. readline() #reads one line in at a time

File Formats

JSON, PKL CSV, XML

packages: json, pkl, csv, xml.etree

json.dumps(), json.loads()

json.dump(), json.load()

csv.writer()

csv.reader()

writerows()