W200
Python Fundamentals for Data Science

Working with Text & Binary Data

A rapid review of encoding, files, regex, parsing, and NumPy

Week 9

Updated Oct 21, 2019

Today's Agenda

- Schedule
- Guidance for Project 1
- Check the extra notebooks for extra examples!

---

1. Number Representations
2. Text Representations: Text Files, Binary Files, and Encodings (ASCII, UTF-8, UTF-16, Unicode)
3. Python Strings
4. Text Encoding (Codecs)
5. Regular Expressions (RegEx)
6. Text Output
7. Files
8. Introduction to Numpy

Schedule: Where We Are

					Topic (study prior to class meeting)		In-Class Presentation	Homework		Exam		Project	
Week #	Mon	Tue	Wed	Thu	Unit	Description		Start	Due	Start	Due	Start	Due
1	Jan 7	Jan 8	Jan 9	Jan 10	Unit 1	Introduction, Command Line, & Source Control		HW unit 1					
2	Jan 14	Jan 15	Jan 16	Jan 17	Unit 2	Starting Out with Python		HW unit 2	HW unit 1				
3	x	Jan 22	Jan 23	Jan 24	Unit 3	Sequence Types and Dictionaries		HW unit 3	HW unit 2				
4	Jan 28	Jan 29	Jan 30	Jan 31	Unit 4	More About Control and Algorithms		HW unit 4	HW unit 3				
5	Feb 4	Feb 5	Feb 6	Feb 7	Unit 5	Functions		HW unit 5	HW unit 4				
6	Feb 11	Feb 12	Feb 13	Feb 14	Unit 6	Complexity		HW unit 6	HW unit 5			Project 1	
7	x	Feb 19	Feb 20	Feb 21	Unit 7	Classes		HW unit 7	HW unit 6				
8	Feb 25	Feb 26	Feb 27	Feb 28	Unit 8	Object-Oriented Programming			HW unit 7	Exam 1			Project 1 Proposal
9	Mar 4	Mar 5	Mar 6	Mar 7	Unit 9	Text and Binary Data					Exam 1		
10	Mar 11	Mar 12	Mar 13	Mar 14	Unit 10	NumPy - Vectors	Project 1 Presentation	HW unit 9					Project 1 Code
11	Mar 18	Mar 19	Mar 20	Mar 21	Unit 11	Pandas - Dataframes		HW unit 10	HW unit 9			Project 2	
SPRING BREAK													
12	Apr 1	Apr 2	Apr 3	Apr 4	Unit 12	MatPlotLib - Data Visualization		HW units 11,12,13	HW unit 10				Project 2 Proposal
13	Apr 8	Apr 9	Apr 10	Apr 11	Unit 13	Advanced Pandas - Aggregation & Groups			HW units 11,12,13	Exam 2			
14	Apr 15	Apr 16	Apr 17	Apr 18	Unit 14	Testing	Project 2 Presentation				Exam 2		Project 2 Report

https://docs.google.com/spreadsheets/d/1GN3rVDfJqpJWmxPgKPupHQWAX2fYCTHv-7jJOud73FI/edit#gid=0

Schedule: Where We're Going

The 2nd half of the course …

- Unit 9 | Working With Text and Binary Data
- Unit 10 | NumPy
- Unit 11 | Data Analysis With Pandas
- Unit 12 | More Analysis With Pandas; Data Visualization
- Unit 13 | Testing
- Team Project
- Exam 2

Guidance for Project 1

- Explain your project at a high level
- Share your screen, run your code, a couple of slides if you want, … show off what you’ve done!
- What were the major challenges of your implementation?
- Please practice a brief (about 5 min, tops) presentation that communicates your project to others. Very important skill.

- If necessary to answer questions, open the code and share to discuss …
- What classes did you use to solve your problem?

Up and Down Levels of Abstraction

- We’ve traversed the levels of abstraction … up and down …
  - Fundamental types: ints, floats … [aka primitives]
  - Container objects: lists, strings
  - Classes
- Now … drill down to characters and bytes

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

2) Number Representations

1. **Binary** (base 2; 0 or 1)
2. **Octal** (base 8; 0,1,2,3,4,5,6,7)
3. **Decimal** (base 10; 0,1,2,3,4,5,6,7,8,9)
4. **Hexadecimal** (base 16; 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)

This leads to various expressions

- “byte” [8-bits]
- “nibble” [4-bits]
- “multibyte” [8, 16, 32, 64, 128]

Setting file permissions: 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?

Berkeley
SCHOOL OF INFORMATION

2.1) Number Representations - Binary / Decimal

- **Binary** (base 2) to **Decimal** (base 10) examples
  - 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 place and 1 in the four’s place

```
int("226")
int("11100010", 2) # binary (base 2)
int("342", 8) # 3 binary digits per 1 octal (base 8) digit
int("E2", 16) # 4 binary digits per 1 hexadecimal (base 16) digit
```

2.2) Number Representations - Octal

- The choice of base 16 comes directly from the use of phrases binary strings; first 3 (Octal) and then 4 digit (nibble, 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 ("nibble") has 16 possible values and represents one hexadecimal digit (0-F)

2.3) Number Representations - Hex

- Hexadecimal (base 16) to Decimal (base 10)

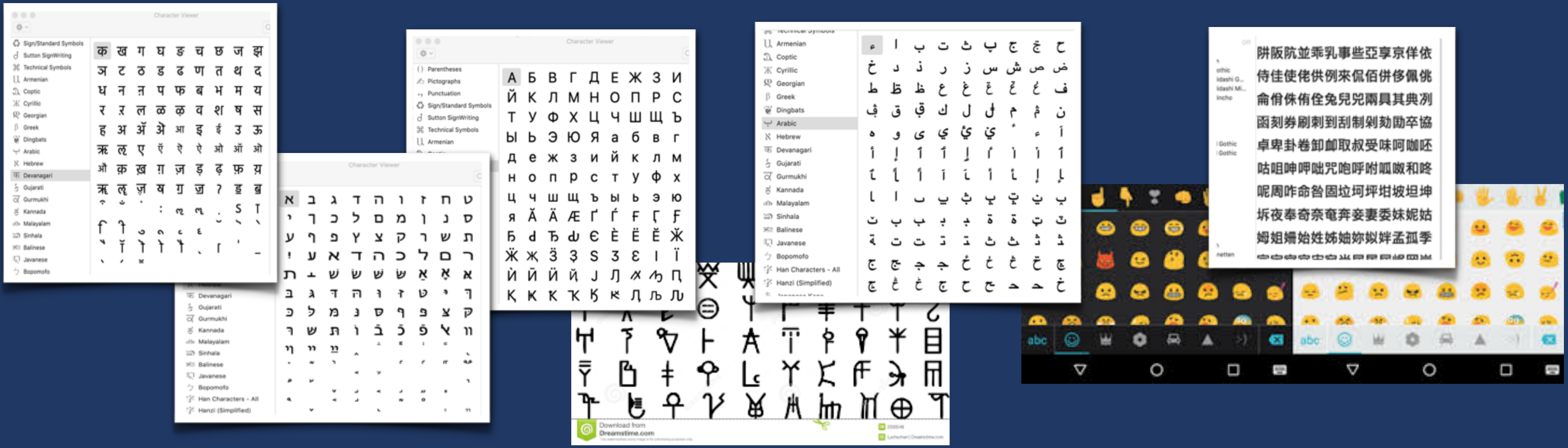
  - 000 is 0 zero in all places
  - 005 is 5 (5×16^0) one in the 1's place
  - 050 is 16 (5×16^1) one in the 16's place
  - 500 is 1280 (5×16^2) one in the 256's place

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

3 Text Representations

- Translate icons and typed text to code & back
- http://unicode.org and check out the codesheets.



3.1) Text Representations

- Ultimately all data are stored as binary (0, 1).
- “Internal reflection” of the data is how the data are stored in computer memory and secondary storage (usually as UTF-8; Windows 10 uses UTF-16 and has difficulties with UTF-8)
- “External reflection” of the data is what’s shown on the screen (often UTF-8, but could be win-1285, MacRoman, koi-8, etc.) … Keep in mind: It’s all just data! We can call “X” the same thing in a variety of “dialects”
- Most end-users think about what they type … but most professional/industrial standards require UTF-8 (https://www.w3schools.com/charsets/ref_html_utf8.asp). There are hundreds of encoding schemes…
- When data don’t appear on screen correctly it is often an encoding mismatch between the data stream and the output device’s encoding settings. In word processing programs and in CJK and other language groups, byte-shifting is critical to storage and retrieval. https://en.wikipedia.org/wiki/Character_encoding

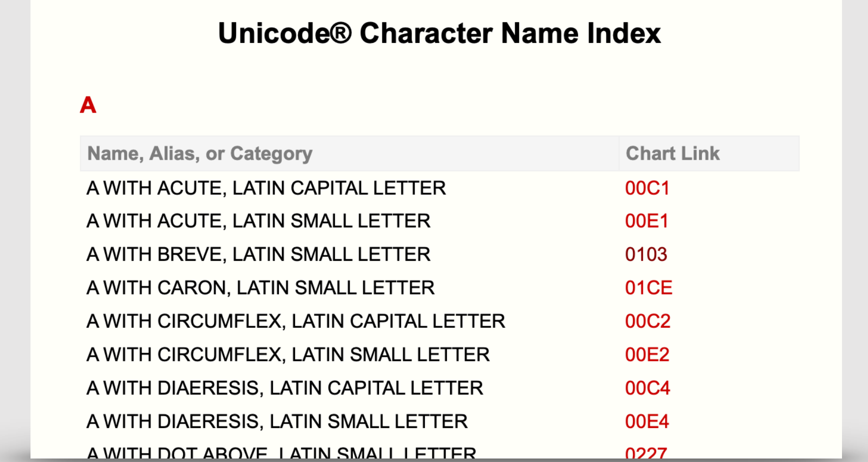
3.2) Text Representations | ASCII

- ASCII uses 7 binary bits
  - $2^7 = 128$ characters
- ASCII with 8 binary bits
  - $2^8 = 256$ characters - aka “Extended ASCII character set”

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

3.3) Text Representations | Unicode

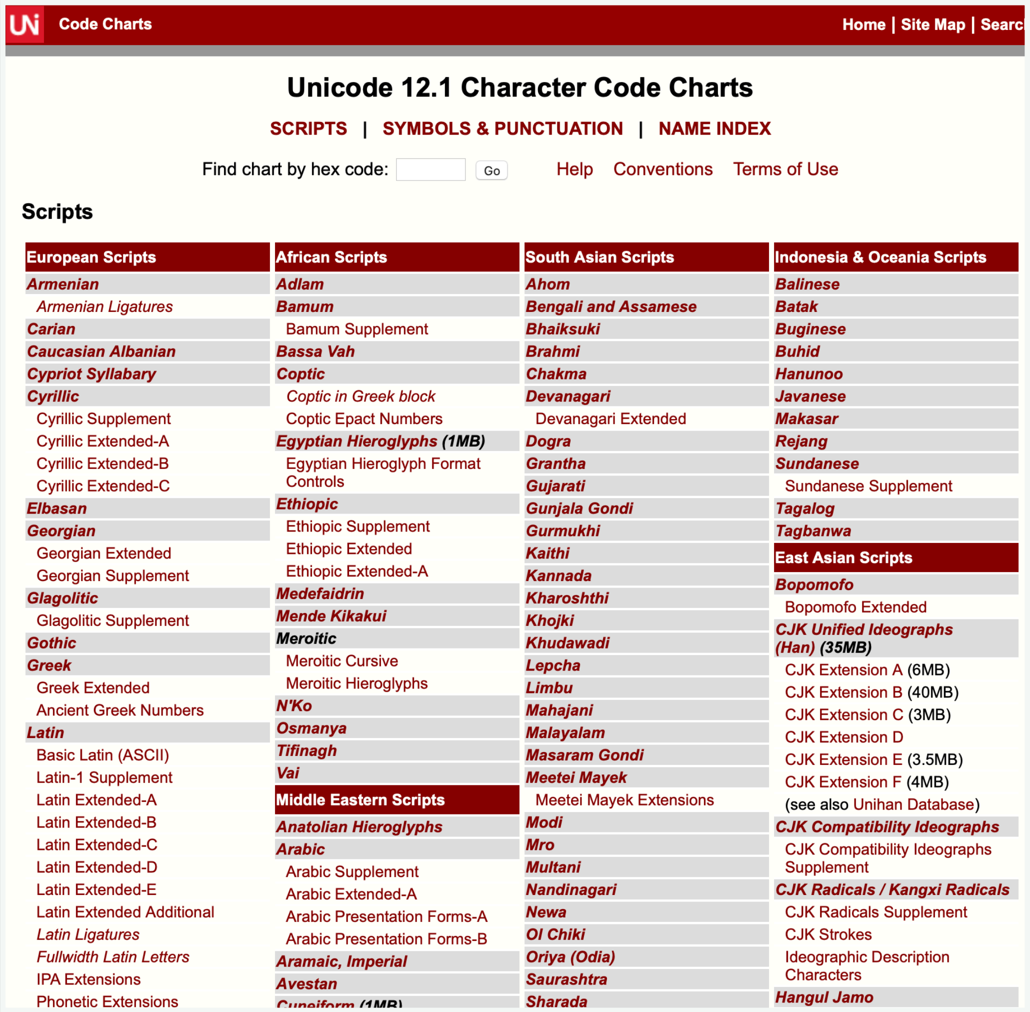
- Unicode encodes 120,000 characters
  - Modern and ancient languages, math
  - Python 3 has native support for Unicode
- http://unicode.org/charts/charindex.html
- http://www.unicode.org/charts/



Unicode® Character Name Index

A

Name, Alias, or Category	Chart Link
A WITH ACUTE, LATIN CAPITAL LETTER	00C1
A WITH ACUTE, LATIN SMALL LETTER	00E1
A WITH BREVE, LATIN SMALL LETTER	0103
A WITH CARON, LATIN SMALL LETTER	01CE
A WITH CIRCUMFLEX, LATIN CAPITAL LETTER	00C2
A WITH CIRCUMFLEX, LATIN SMALL LETTER	00E2
A WITH DIAERESIS, LATIN CAPITAL LETTER	00C4
A WITH DIAERESIS, LATIN SMALL LETTER	00E4
A WITH DOT ABOVE, LATIN SMALL LETTER	0227



Code Charts — Home | Site Map | Search

Unicode 12.1 Character Code Charts

SCRIPTS | SYMBOLS & PUNCTUATION | NAME INDEX

Find chart by hex code: Go — Help Conventions Terms of Use

Scripts

European Scripts
Armenian
Armenian Ligatures
Carian
Caucasian Albanian
Cypriot Syllabary
Cyrillic
Cyrillic Supplement
Cyrillic Extended-A
Cyrillic Extended-B
Cyrillic Extended-C
Elbasan
Georgian
Georgian Extended
Georgian Supplement
Glagolitic
Glagolitic Supplement
Gothic
Greek
Greek Extended
Ancient Greek Numbers
Latin
Basic Latin (ASCII)
Latin-1 Supplement
Latin Extended-A
Latin Extended-B
Latin Extended-C
Latin Extended-D
Latin Extended-E
Latin Extended Additional
Latin Ligatures
Fullwidth Latin Letters
IPA Extensions
Phonetic Extensions

African Scripts
Adlam
Bamum
Bamum Supplement
Bassa Vah
Coptic
Coptic in Greek block
Coptic Epact Numbers
Egyptian Hieroglyphs (1MB)
Egyptian Hieroglyph Format Controls
Ethiopic
Ethiopic Supplement
Ethiopic Extended
Ethiopic Extended-A
Medefaidrin
Mende Kikakui
Meroitic
Meroitic Cursive
Meroitic Hieroglyphs
N'Ko
Osmanya
Tifinagh
Vai

Middle Eastern Scripts
Anatolian Hieroglyphs
Arabic
Arabic Supplement
Arabic Extended-A
Arabic Presentation Forms-A
Arabic Presentation Forms-B
Aramaic, Imperial
Avestan
Cuneiform (1MB)

South Asian Scripts
Ahom
Bengali and Assamese
Bhaiksuki
Brahmi
Chakma
Devanagari
Devanagari Extended
Dogra
Grantha
Gujarati
Gunjala Gondi
Gurmukhi
Kaithi
Kannada
Kharoshthi
Khojki
Khudawadi
Lepcha
Limbu
Mahajani
Malayalam
Masaram Gondi
Meetei Mayek
Meetei Mayek Extensions
Modi
Mro
Multani
Nandinagari
Newa
Ol Chiki
Oriya (Odia)
Saurashtra
Sharada

Indonesia & Oceania Scripts
Balinese
Batak
Buginese
Buhid
Hanunoo
Javanese
Makasar
Rejang
Sundanese
Sundanese Supplement
Tagalog
Tagbanwa

East Asian Scripts
Bopomofo
Bopomofo Extended
CJK Unified Ideographs (Han) (35MB)
CJK Extension A (6MB)
CJK Extension B (40MB)
CJK Extension C (3MB)
CJK Extension D
CJK Extension E (3.5MB)
CJK Extension F (4MB)
(see also Unihan Database)
CJK Compatibility Ideographs
CJK Compatibility Ideographs Supplement
CJK Radicals / Kangxi Radicals
CJK Radicals Supplement
CJK Strokes
Ideographic Description Characters
Hangul Jamo

3.4) Text Representations | Unicode

- Every Unicode value has a standard name
  - Use `unicodedata.name()` to get name from a value
  - E.g., Value can be literal “B” or Unicode value “\u0042”
  - E.g., Returns “`LATIN CAPITAL LETTER B`”
- Benefit? Every character can be identified uniquely!
- You can often paste exotic characters

3.5) Text Representations | Python Strings

- All Python 3 strings are encoded as Unicode

- "\u0047\u0072\u0072\u0021" == 'Grr!’ is True
- "\u0047\u0072\u0072\u0021" == 'GRR!’ is False

- *Note: Some characters, particularly historical ones and ligatures, may be software or O/S dependent. Accessing these characters requires knowing and being able to read the “GID”: graph identification number.*

3.6) Text Encoding | UTF-8

- UTF-8
  - Compatible with Unicode
  - To encode a text string in Unicode, use `encode('utf-8')`
  - To decode Unicode, use `decode('utf-8')`
  - `b` prefix denotes bitwise encoding
  - `\x` prefix denotes "hexadecimal"

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

- Note: Windows 10 may create files using UTF-16
  - For example, when you create a file using `echo "test" >> test.txt`
  - If you run into encoding issues, this may be the cause.

3.7) Text Encoding | Unicode-8

- `unicodedata` package
- encode (), decode()
- type (), len()

BTW, in almost all situations data sent over the Net have to be checked for illegal characters, control characters, insertion attacks, etc.

E.g., Data sent over the net might have ‘ in it: O’Reilly … but we must make sure the ‘ won’t conflict with an apostrophe in the code …

E.g., reading and writing data should be encoded/ decoded so they can be stored/retrieved without accidental reading issues.

4) RegEx | Functions to Find Patterns

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

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

.	# any character 1 place
*	# any number of char
?	# any character optional
[0-9]	# any digit
[a-z]	# any letter lowercase letter
r’ ’	# the raw string literal

RegEx | Special Characters

RegEx | Specifiers

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

not expected to know all this instantly!

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 | Example

```
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/

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

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

```
that is
```

```python
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']
```

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

or

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

Berkeley
SCHOOL OF INFORMATION

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
```

Berkeley SCHOOL OF INFORMATION

5) Text Output

Consider: `s='㈲word'`

Simple concatenation: `print(‘this is my text:  ’  + s)`

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

Old 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
```

5.1) Text Output | 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))
```

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

6) Files

`open(file, mode)` open modes (‘wt’, ‘rd’, ‘at’, ‘rb’, ’wb’)

`with()` # you don’t need to close this one

write()

read()

readlines() # reads all lines as a list

readline() #reads one line in at a time

close()

See the extra notebook about files, too.

7) Breakout Activity

Files

String parsing

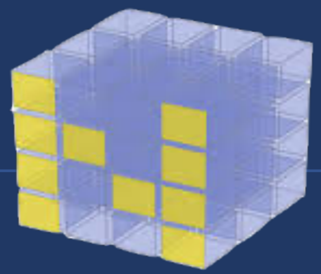
8) NumPy

- **NumPy**: work with n-dimensional arrays of numeric data of many types.
  - Pandas is built on top of NumPy and provides a more user friendly experience. There, we work with a “dataset” and include non-numeric variables.
  - Understanding NumPy is critical to understanding more advanced packages.
  - A basic understanding of NumPy will deepen your understanding of Pandas.
  - NumPy offers vectorized operations

https://timothyhelton.github.io/pandas_best_practices.html

https://docs.scipy.org/doc/numpy/user/quickstart.html

http://www.numpy.org



8.1) Introduction to NumPy | Python Functions

- np.array()
- np.arange(), np.linspace()
- np.min(), np.max(), np.std(), np.var()
- np.argmax(), np.argmin()
- np.shape(), np.reshape()
- np.zeros()
- np.random.seed(), np.random.random_integers()
- np.vstack(), np.hstack()
- Dealing with n-dimensions: “axis = “ (0 or 1)