Python for Data Science

LECTURE 3

Organization of Lecture 3

- Essential features
  - string functions
  - data structures
  - list comprehension
  - Counters
  - file and web functions
  - regular expressions
  - globbing
  - data pickling
- Process data
  - extract data
  - store data in structures
  - locate pieces matching in data
  - serialize & deserialize Python objects

Why do some of this low level stuff?

- High-level tools exist …
  - Anaconda's Python has 350 Python packages
- Why split strings, read files line by line?
  - NON-STANDARD data sources !!!
  - Many data out of compliance.
- Data scientists have to know how to be a useful programmer.

Basic Built-in String Functions

- Case conversion – important for normalization
  - `lower()  upper ()  capitalize ()`
  - NOTE: `s = s.upper()`
- Predicate tests – return **`True`** or **`False`** – validation of data
  - `islower()  isupper ()  isspace ()`
  - `isdigit()  isalpha ()`
- Strings as raw binary arrays
  - `bin =b"Hello"` and `bin[0]` is `72`
  - `bin.decode()` → `string` `s.encode ()` → `binary array`

Basic Built-in String Functions

- Stripping strings (getting rid of white spaces at front or end)
  - `lstrip()  rstrip()  strip()`
  - `print ("  Hello,  world!  \t\t\n".strip())  → 'Hello,  world!'`
- Separating tokens in strings
  - split(delim=‘’) if no delimiter, splits on white space
- Combining list of strings into one string, using object string as the “glue”
  - `s = "".join (["alpha", "bravo", "charlie", "delta"])`
    `→ alphabravocharliedelta`
  - `s = ", ".join (["alpha", "bravo", "charlie", "delta"])`
    `→ alpha, bravo, charlie, delta`
  - `" ".join ("This string\n\r  has  many  \t\tspaces".split())`
    `→ 'This string has many spaces'`

Basic Built-in String Functions

- Substring search
  - `obj.find(subst)` -- 1st occurrence of substring in object string or -1
  - `num = s.count(subst)` -- number of non-overlapping occurrences

main.py

```
dnaseq = "GTACACTTACGTACCATGTTACGTCA"
print (dnaseq.find("GTAC"))
print (dnaseq.count("GTAC"))
```

Console Shell

```
0
2
>
```

Choosing the Right Data Structure

- Commonly used Python data structures: lists, tuples, sets and dictionaries
- All these are collections
- Lists are implemented as arrays
- Tuples are immutable lists (can't change, must assign to another variable)
- Sets are *not* sequences
  - Do not have indices, only have one copy of an item.
  - Search is sublinear O(log N)
  - Useful for membership look-ups and eliminating duplicates

Sets and Membership sample code

```
myList = list (set(myList))  #remove dupes from myList

bigList = [str(i) for i in range (1000000)]
print("abc" in bigList)     #takes 0.2 sec
bigSet = set(bigList)
print("abc" in bigSet)      #takes 15-30 μsec – 10000X faster
```

Dictionaries

- Dictionaries map keys to values
  - Keys must be hashable data type (number, Boolean, string, tuple)
  - Sublinear O(log N) search time

```
myDict = {"M2": "Stringfellow", "M3": "Simpson", "M1": "Knox"}
```

- Can create from a list of (key, value) tuples

```
seq = ["Stringfellow", "Simpson", "Knox"]
print (dict(enumerate(seq)))
```

→ **{0: 'Stringfellow', 1: 'Simpson', 2: 'Knox'}**

Dictionaries cont.

- Can create dictionary from sequence of keys and a sequence of values using zip

```
kseq = [“M2”, “M3”, “M1”]
vseq = [“Stringfellow”, “Simpson”, “Knox”]
dict (zip (kseq, vseq))
→ {"M2": "Stringfellow", "M3": "Simpson", "M1": "Knox"}
```

- `enumerate(seq)` and `zip(seq1, seq1)` are list generators
- Can use them in for loops

List Comprehension

- Transforms a collection into a list - USED to apply SAME operation to all or a range of the list elements
- How it works
  1. Iterates over collection and visits each item (like a for each loop)
  2. Optional Boolean expression is evaluated for each item (if none default is T)
  3. If Bool expr is T, operation performed on current item and appended to result
  4. If F, current item is ignored

```
#copy list
myList = ['a', 'c','d', 'f']
result = [x for x in myList]
→ ['a', 'c','d', 'f']
```

```
#copy non-negative values
myList = [2, 3, -5, 6, -7]
result = [2*x for x in myList if x>0]
→ [4, 6, 12]
```

Counter

- Aids in tallying items in collections
- `most_common(n)` gets list of n most frequent items and their frequencies (no n? Returns a list of all items)

```
from collections import Counter
phrase = "a man a plan a canal panama"
cntr = Counter (phrase.split())
freqlist = cntr.most_common()
print (freqlist)          → [('a',3), ('canal',1), ('panama', 1), ('plan', 1), ('man', 1)]

cntrDict = dict(freqlist)
print (cntrDict)          → {'a': 3, 'canal': 1, 'panama': 1, 'plan': 1, 'man': 1}
print (cntrDict['a'])     → 3
```

Working with Files

- Open files in modes, “r”, “w” or “a” and “rb”, “wb” or “ab”

```
f=open(name, mode= “r”)
f.close()
f.read(), f.read(n)      #n is number of bytes
f.readline(),  f.readlines()
f.write(line), f.writelines(lines) #add \n on your own
```

- Can also use `with` statement

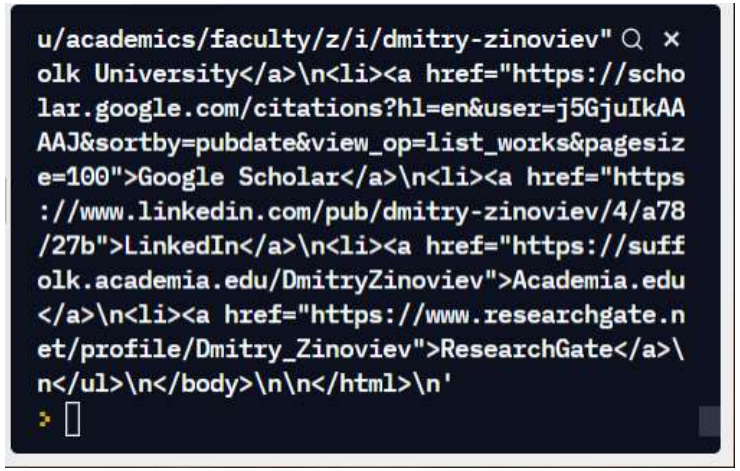
```
with open(name, mode = “r”) as f:
  read()
  close()
```

Reaching the Web

- Indexed web has 4.05 billion web pages (www.worldwidewebsize.com)
- Module `urlib.request` contains functions for downloading data from the web
- Provides automated iterative or recursive downloads
- Open the URL and obtain open URL handle `urllib.request.urlopen(URL)`
- Possible for open to fail, so do exception handling
- If website, requires authentication, use module `SSL` and `OpenSSL`

Reading web doc

```
import urllib.request
try:
  with urllib.request.urlopen ("http://networksciencelab.com") as doc:
    html = doc.read()  # if successful, connection auto closed
    print (html)
except:
   print ("Could not open %s" %doc, file=sys.err)
   #handle this someway
```



Parsing and Unparsing

- Module `urllib.parse` has tools for parsing and unparsing(building) URLs
- Function `urlparse()` splits a URL into a 6-tuple

```
import urllib.parse
URL = "http://networksciencelab.com/index.html;param?foo=bar#content"
print(urllib.parse.urlparse(URL))
```

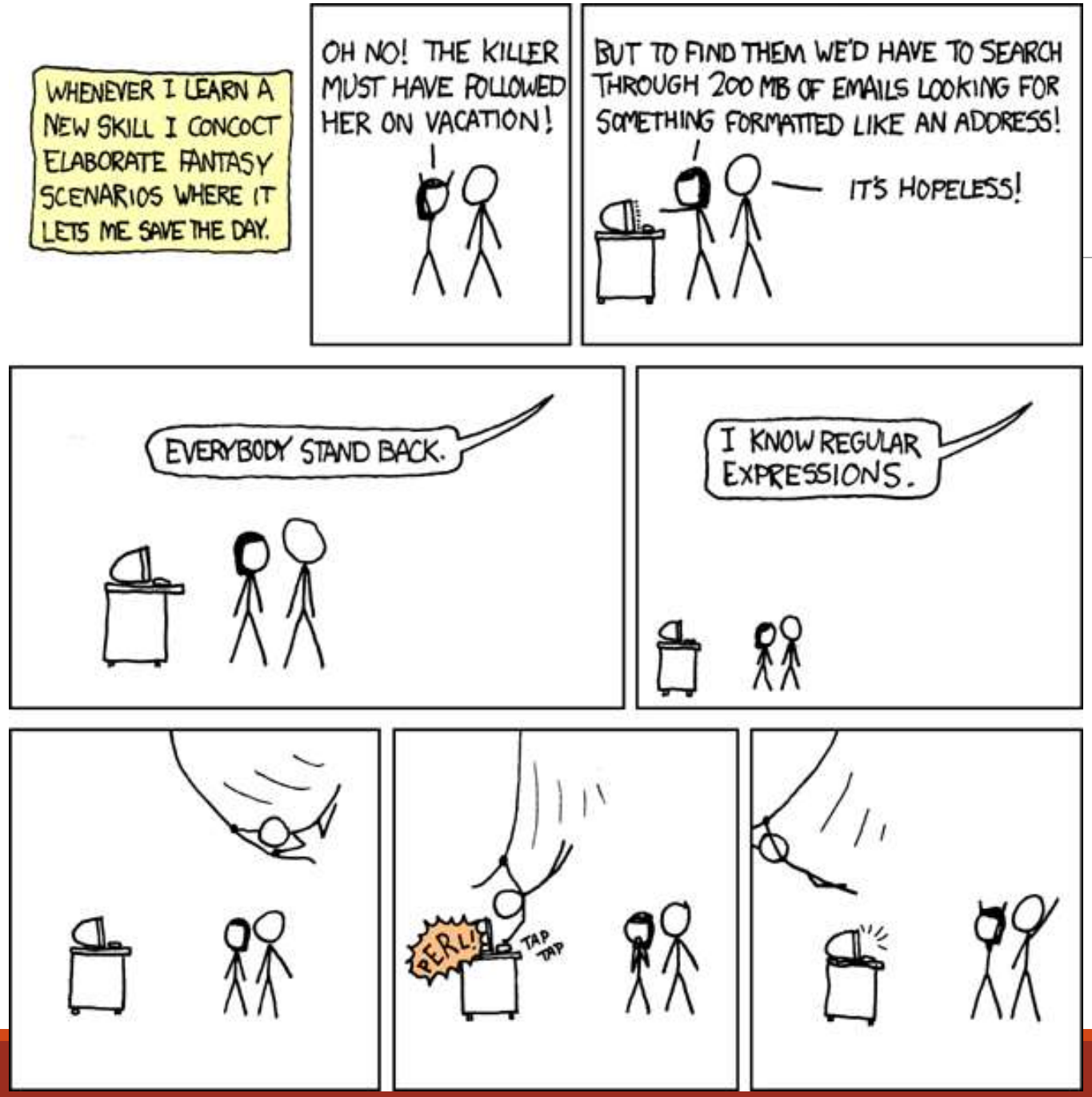
```
6.5493">D.Zinoviev, V.Duong, a
ParseResult(scheme='http', netloc='networksciencelab.com', path='/index.html', params='param', query='foo=bar', fragment='content')
>
```

Pattern Matching with Regular Expressions (REs)

- Mechanism for searching, splitting, and replacing strings based on pattern
- Import module **`re`** - provides pattern description language and functions
- More efficient to compile a RE, if you use it more than once

```
compiledPattern = re.compile(pattern, flags=0)
```

- Flags are `re.I` (ignore character case) and `re.M` (multiline mode)



http://xkcd.com/208/

Regular Expessions Quick Guide

.	Any char except \n
a	character a itself
ab	string ab
x|y	x or y
\y	escape characters
[a-d]	a, b, c or d character
[^a-d]	any charcter other than a,b,c, or d
\d	one digit
\D	one non-digit
\s	one whitespace
\S	one non-whitespace
\w	alphanumeric character
\W	non-alphanumeric character
\\	back slash character

x*	zero or more x's
x+	one or more x's
x{2}	exactly two x's
x{2,5}	between 2-5 x's
\n	
\r	
\t	
^	start of string
\b	word boundary
\B	non-word boundar
$	end of the string
(x)	capturing group
(?:x)	non-capturing group

Raw Strings

- Always write REs as raw strings
- `\\n` → `r"\n"`
- `r"w[-\w\.]*@\w[-\w]*(\.\w[-\w\*)+"` is an email address
- `R"[-+]?((\d*\.?\d+)|(\d\.))([eE][-+]?\d+)?"` is floating point number
- File name matching use glob…

re.split

- **`re.split (pattern, string, maxsplit=0, flags=0)`** splits a string into at most maxsplit substrings by the pattern and returns the list of substrings) THIS IS A TOKENIZER!

```
import re
print (re.split ("\W", "Hello, world!"))
print (re.split ("\W+", "Hello, world!"))
```

```
['Hello', '', 'world', '']
['Hello', 'world', '']
>
```

re.match

- `re.match (pattern, string, flags=0)` checks if beginning of string matches RE

```
mo=re.match(r"\d+", "067 starts with a number")
print (mo)
print(mo.group())

nmo=re.match(r"\d+", "Does starts with a number")
print (nmo)
```

```
<re.Match object; span=(0, 3), match='067'>
067
None
>
```

re.search

- `re.search (pattern, string, flags=0)` checks if any part of string matches RE. Flag re.I is case-insenstive

```
mo=re.search(r"[a-z]+", "0010010 Has at least one 010
letter 0010010", re.I)

print (mo)
```

```
<re.Match object; span=(8, 11), match='Has'>
<re.Match object; span=(9, 11), match='as'>
```

```
mo=re.search(r"[a-z]+", "0010010 Has at least one 010
letter 0010010")

print (mo)
```

re.findall

- `re.findall (pattern, string, flags=0)` finds all substrings that match the RE, returns a list of substrings

```
mo=re.findall(r"[a-z]+", "0010010 Has at least one 010
letter 0010010", re.I)

print (mo)
```

```
['Has', 'at', 'least', 'one', 'letter']
>
```

re.sub

- **`re.sub (pattern, repl, string, flags=0)`** replaces all non-overlapping matching parts of a string with repl. Can restrict the number of replacements with option parameter count.

```
mo=re.sub(r"[a-z]+", “[…]”, “0010010 has at least one 010
letter 0010010”)

print (mo)
```

```
0010010 […] […] […] […] 010 […] 0010010
>
```

Globbing Filenames and Other Strings

- Process of matching specific file names and wildcards (using simplified Res)

```
import glob
glob.glob("home/Desktop/*.txt")     # return list of all
text files
```

- e.g. `['policy.txt', 'big.data.txt']`

Pickling and Unpickling data

- Module pickle implements serialization – saves Python data structures into a file and reads them back. Must use Python program. ONLY unpickle data you trust!

- https://www.geeksforgeeks.org/understanding-python-pickling-example/

```
 358-363, July 2010\n<li><a href="http://arxiv.org/abs/1006.5493">D.Zinoviev, V.Duong, a
Omkar => {'key': 'Omkar', 'name': 'Omkar Pathak', 'age': 21, 'pay': 40000}
Jagdish => {'key': 'Jagdish', 'name': 'Jagdish Pathak', 'age': 50, 'pay': 50000}
>
```