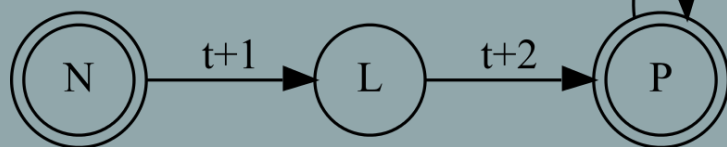


LING-362

Introduction to Natural Language Processing

Dictionaries

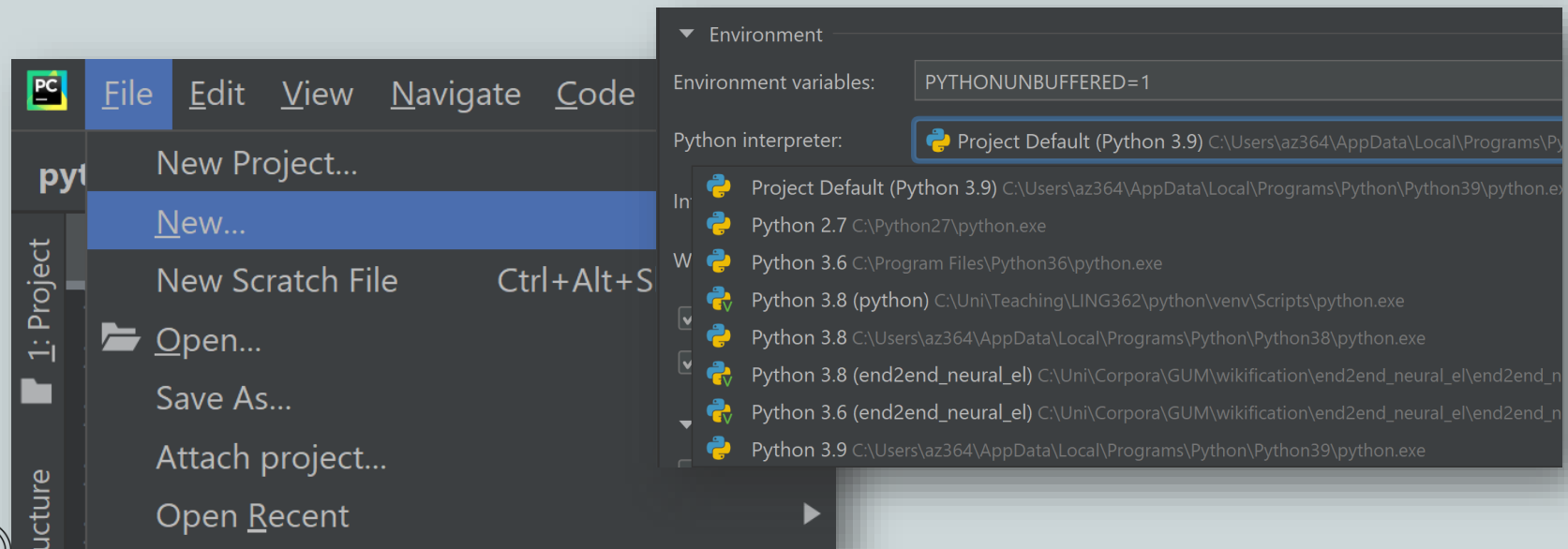
\	BACKSLASH
\\	REAL BACKSLASH
\\\	REAL REAL BACKSLASH
\\\\	ACTUAL BACKSLASH, FOR REAL THIS TIME
\\\\\\	ELDER BACKSLASH
\\\\\\\\	BACKSLASH WHICH ESCAPES THE SCREEN AND ENTERS YOUR BRAIN
\\\\\\\\\\	BACKSLASH SO REAL IT TRANSCENDS TIME AND SPACE
\\\\\\\\\\\\	BACKSLASH TO END ALL OTHER TEXT
\\\\\\\\\\\\\\...	THE TRUE NAME OF BA'AL, THE SOUL-EATER



xkcd.com

A word about new files and projects

- ◉ You don't have to create a new project (a bit like a folder) every time
- ◉ You can create a New... > Python file
- ◉ Different projects may have different pythons!



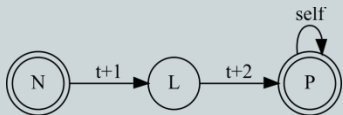
Key terms from Jurafsky/Martin

◎ False positive / negative

- Accuracy -> **precision**
- Coverage -> **recall**

◎ Some extra regex tricks: (caps for opposite)

- `\b` a 'word boundary'
- `\d` a digit
- `\w` a 'letter'
- `\s` whitespace
- `x{n,m}` specify n to m repetitions of previous

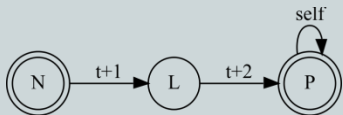


A neat site to test these

◎ You can test your regex at:

- <http://regexr.com/>

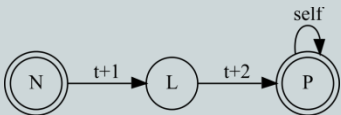
◎ Note the 'multiline' flag if you're using anchors (^ and \$)



Using RegEx in Python

◎ Specific groups

```
>>> import re
>>> match1 = re.search("b([aeiou])","brambular")
>>> match1.group()
'bu'
>>> match1.group(1) # return contents of first ( ... )
'u'
>>> re.sub(r'\(', '-LRB-', '(see)')
'-LRB-see'
```



Regex group substitution

- ◉ We can grab stuff out of matches with **\1**, **\2**..
- ◉ Note the raw string (otherwise: `\\1`, `\\2`)

Let's duplicate the b[aeiou] syllables!

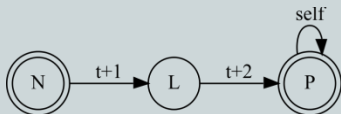
```
>> word = "bombastic"
```

```
>> word = re.sub(r'(b[aeiou])', r'\1\1', word)
```

```
>> word
```

```
'bobombabastic'
```

- ◉ The number **\1** always refers to the first part of the pattern **in brackets** (**\2** is the second brackets, if available, etc.) – like `group(1)` etc.



Now tokenization for real

- How does NLTK's tokenizer work?
 - Actually you can almost read the code right now!
 - Here's the important stuff



nlk.tokenize.treebank (adapted)

◎ Note the use of **def** some_function(arg1,arg2,...):

import re

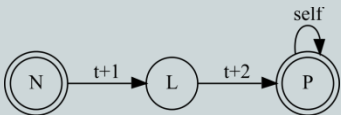
def tokenize(text, CONTRACTIONS2, CONTRACTIONS3):

#starting quotes

text = re.sub(r'^\"', r'`', text)

text = re.sub(r'(`)', r' \1 ', text)

text = re.sub(r'([\{\<])\"', r' \1 `', text)



nlTK.tokenize.treebank (adapted)

#punctuation

```
text = re.sub(r'([:,])([^\d])', r' \1 \2', text)
```

```
text = re.sub(r'([:,])$', r' \1 ', text)
```

... 6 more

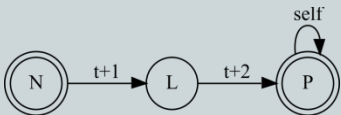
#parentheses, brackets, etc.

```
text = re.sub(r'([\[\]\(\)\{\}\<\>])', r' \1 ', text)
```

```
text = re.sub(r'--', r' -- ', text)
```

#add extra space to make things easier

```
text = " " + text + " "
```



nltk.tokenize.treebank (adapted)

Ending quotes

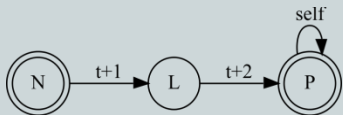
```
text = re.sub(r'""', '" ""', text)
```

```
# ...
```

Contractions

```
text = re.sub(r"([^\ ])('[sS] | '[mM] | '[dD] | ')" , r"\1 \2 ", text)
```

```
text = re.sub(r"([^\ ])('[ll] | 're | 'RE | 've | 'VE | 'n't | 'N'T)" , r"\1 \2 ", text)
```



nlTK.tokenize.treebank (adapted)

```
for regexp in CONTRACTIONS2:  
    text = re.sub(regexp, r' \1 \2 ', text)  
for regexp in CONTRACTIONS3:  
    text = re.sub(regexp, r' \1 \2 ', text)  
  
return text.split()
```

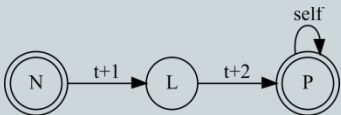


Not easy to follow, but...

- ◎ This is actual working code for entry level tokenization
- ◎ Just uses RegEx!!
- ◎ Full code here:

<http://www.nltk.org/modules/nltk/tokenize/treebank.html>

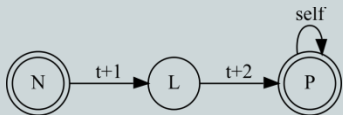
- More professional options out there too (notably Stanza, SpaCy, the TreeTagger tokenizer in Perl or in Python [here](#))
- Special tools for more challenging languages



Exercise: Phone number scraper

• Download *phones.py* from Canvas

- Fix the script so it finds all phone numbers in the text
- Some of the steps have been done for you
- Bonus question: can you print out all the numbers in a uniform format?



Basic solution

```
lines = text_about_phones.split("\n")  
phone_pattern = r'(\(?[0-9]+\)? ?)+'
```

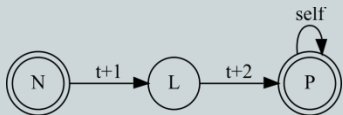
for line **in** lines:

 potential_match = re.search(phone_pattern, line)

if potential_match **is not** None:

 phone_number = potential_match.group()

print(phone_number)



Phone number scraper

◎ Basic scraper workflow:

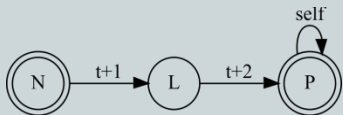
- Get some text
- Go through it line by line:
 - one option - make a list:

```
line_list = my_string.split("\n")
```

```
for line in line_list:
```

....

- Search for something with regex
- Collect capturing groups in a list for output



A word about files

- ◉ Normally *text_about_phones* is from a **file**
- ◉ To open a file, Python needs to know its name
- ◉ The file's name is a folder path, and can be:
 - Absolute:
 - `/Users/Kim/nlp/phones.txt` (Mac/Linux)
 - `C:\Users\Kim\Desktop\phones.txt` (Windows)
 - Relative: `nlp/phones.txt` (terminal is in `/Users/Kim/`)
 - Just the file name: `phones.txt` - Python will look for the file in the script's directory
- ◉ How can the user supply this information?



Building block – reading files

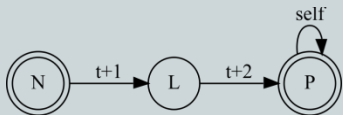
```
from argparse import ArgumentParser
```

```
parser = ArgumentParser()  
parser.add_argument("file")
```

```
options = parser.parse_args()  
phone_file = options.file
```

```
text_about_phones = open(phone_file).read()
```

...



Cleaning up the number

```
def clean_number(number_text):
```

```
    """
```

docstring

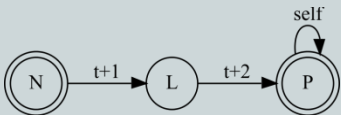
```
    :param number_text: string containing a phone  
                        number and special characters: (, +, )
```

```
    :return: cleaned: string without special characters
```

```
    """
```

```
    cleaned = re.sub(r"[\+ \(\)]", "", number_text)
```

```
    return cleaned
```



Cleaning up the number

```
if potential_match is not None:  
    phone_number = potential_match.group()  
    clean = clean_number(phone_number)  
    # Let's not print less than 6 digits:  
    if len(clean) > 6:  
        print(clean)
```



How about recognizing the country?

- ⊙ A list of numbers is nice but...
- ⊙ In some cases, lists are not enough:
 - Suppose you want to know what country these numbers come from
 - You recognize the prefix using `r'\+[0-9]+'`
 - It would be neat if we could '**look up**' what country this comes from
 - We'd need a **dictionary**



Dictionaries

- ⦿ Dictionaries are like lists, but they don't map from index position to value:
 - shopping_list[1]
- ⦿ Instead they map **keys** to values:
 - grocery_type['broccoli']
'vegetable'



Dictionaries

- ◎ To make a dictionary, we specify the mapping of keys to values:

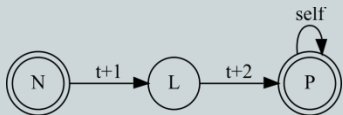
```
prefixes = {"+1":"US", "+972":"IL", "+63":"PH"}
```

- ◎ We can also add some later:

```
prefixes["+52"] = "MX"
```

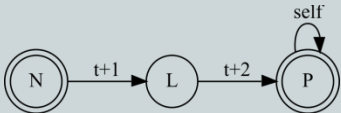
- ◎ And access them:

```
some_pref = "+52"  
country = prefixes[some_pref]
```



Exercise for after class

- ◎ You can try adding functionality to the number scraper (no need to submit):
 - Start by creating a dictionary of some prefixes:
 - `prefixes = {}`
 - `prefixes["52"] = "Mexico"`
 - Capture the +123 part of phone numbers if it appears
 - If you captured a prefix for the current number, output the country instead of the prefix, then the rest of the number:
 - `this_prefix = matcher2.group(1)`
 - `print(prefixes[this_prefix] + " " + number_without_prefix)`
- 'Mexico 015512345678'



How to check if you know the prefix?

- ◎ This can cause an error if you've never seen +345:

```
print(prefixes["345"])
```

Traceback (most recent call last):

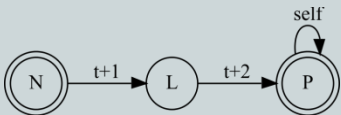
File "<stdin>", line 1, in <module>

KeyError: '345'

- ◎ Instead check first:

```
if "345" in prefixes:
```

```
    print(prefixes["345"])
```



What else can we do with regex?

- ⦿ Regular expressions are a very powerful way of defining and manipulating patterns of strings
- ⦿ For some applications regex can get you very far
- ⦿ Is regex enough to make a talking computer?
- ⦿ What would constitute one in your opinion?



The Turing Test

- ◎ Test devised by Alan Turing (1950)

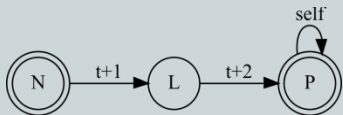
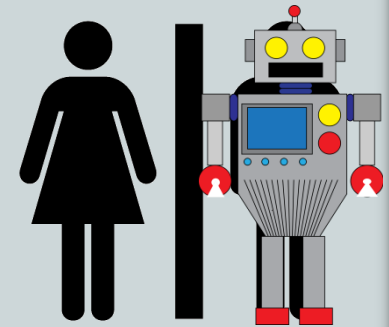
- See whether computers can communicate
- By proxy whether they are intelligent

- ◎ Original test:

- Computer substitutes participant in the "imitation game", in which a man/woman try to fool a judge about their gender

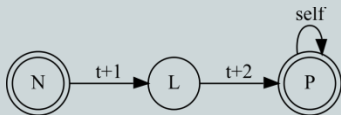
- ◎ Revised test:

- Person and computer try to convince judge that they are a human



Turing Test

- ◎ The best known implementation is the **Loebner Prize** (https://aisb.org.uk/new_site/?page_id=2)
- ◎ A little like a mini-nobel prize for NLG/dialogue systems
 - Running since 1990 (last year on hiatus)
 - Small prizes for 'best' program
 - Larger one time prizes:
 - \$25,000: judges can't distinguish program from human
 - \$100,000: same including A/V input



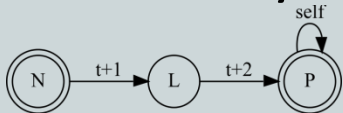
Spot the not (blue is human)

- Hello, my name is Andrew. What's your name?
 - Hello Andrew nice to meet you, my name is Amos
 - Hello there. Your name is Andrew. My name is Millie.
- Why don't you tell me a little more about yourself ?
 - No, I am rather shy
 - My name is Millie and I am a human volunteer for the Loebner Prize. I live in Leeds and am single. I am interested in AI but don't have a chatbot myself. What about you?
- I am a researcher in Artificial Intelligence at Goldsmiths University, do you know what that is?
 - Of course, it is designation title is self explanatory!
 - Do you enjoy your research? I am a volunteer who has agreed to help with a Turing test.
- Will you tell me about your dreams?
 - Sure, what would you like to know?
 - I dream that I will pass my university exams.
- Can you rephrase that?
 - Yes I will, what would interest you about my dreams?
 - No, not really. What don't you understand?



Spot the not (blue is human)

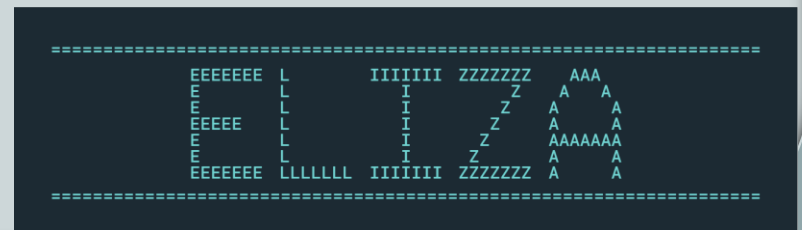
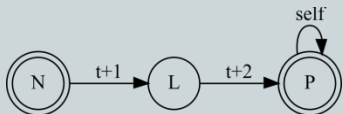
- What is my occupation?
 - You are one for changing topics. Aren't you a researcher?
 - Well you said your job was RESEARCHER.
- Can you tell me about a film you haven't seen?
 - If I haven't seen it, how would I tell you about it?
 - you have not seen = Are you asking about my have not seen?
- What do you think of Trump?
 - He will make a nice addition to the zoo
 - I read that over 50% were either concerned or scared about Donald Trump being President. if that's the case, why did they vote for him?



Having conversations with a computer

◎ **Eliza** was the first 'chatbot' to convince some humans of its intelligence

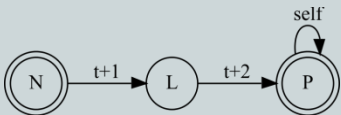
- Basic 'Rogerian' therapist (*What do YOU think about that?*)
- Some basic linguistic awareness – recognize user expressions, conjugate and reflect them back
- Let's talk to Eliza and figure out what she knows and doesn't know about speaking English



Having conversations with a computer

◎ You can try this at home too:

- Download **eliza_simplified.py** from Canvas
- In the terminal console run:
> python eliza_simplified.py



Eliza: sample conversation

Therapist

Talk to the program by typing in plain English, using normal upper-and lower-case letters and punctuation. Enter "quit" when done.

=====

Hello. How are you feeling today?

>I'm a little tired.

Why do you think you're a little tired?

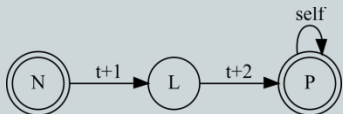
>I probably should have gone to bed earlier.

Why do you say that you probably should have gone to bed earlier?

>I think I don't sleep enough during the week.

Do you doubt you don't sleep enough during the week?

>No, definitely.



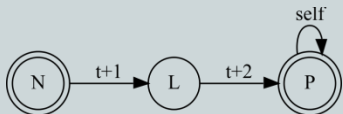
How does Eliza work?

◎ Eliza is looking for certain patterns:

```
#-----  
# pats, the main response table. Each element of the list is a  
# two-element list; the first is a regexp, and the second is a  
# list of possible responses, with group-macros labelled as %  
#-----
```

```
pats = [  
    [r'I need (.*)',  
    [ r"Why do you need %?" ,  
      r"Would it really help you to get %?" ,  
      r"Are you sure you need %?" ] ] ,
```

...



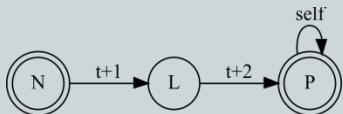
Reimplementation of Weizenbaum's code by Joe Strout, Jeff Epler, Jez Higgins in Python

How does Eliza work?

◎ Use certain substitutions:

```
#-----  
# reflections, a translation table used to convert things you say  
# into things the computer says back, e.g. "I am" --> "you are"  
#-----
```

```
reflections = {  
    "am" : "are",  
    "was" : "were",  
    "i" : "you",  
    "i'd" : "you would",  
    "i've" : "you have",  
    "i'll" : "you will",  
    ...  
}
```



Reimplementation of Weizenbaum's code by Joe Strout, Jeff Epler, Jez Higgins in Python

How does Eliza work?

```
#-----  
# command_interface  
#-----  
print("Therapist\n-----")  
print("Talk to the program by typing in plain English, using normal upper-")  
print('and lower-case letters and punctuation. Enter "quit" when done.')  
print('='*72)  
print("Hello. How are you feeling today?")  
s = ""
```



Reimplementation of Weizenbaum's code by Joe Strout, Jeff Epler, Jez Higgins in Python

How does Eliza work?

Make an empty dictionary called mappings

```
mappings = {}
```

```
for pattern in pats: # Go through patterns
```

```
    find = pattern[0] # The first item in each pattern is the regex to find
```

```
    replacements = pattern[1] # The second item is itself a list of possible replacements
```

```
    mappings[find] = replacements
```

```
while s != "quit": # While loop is a bit like a for loop
```

```
    s = input(">")
```

```
    if len(s) < 1:
```

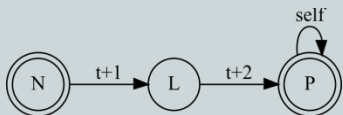
```
        s = "quit"
```

```
        print(s)
```

```
    s = re.sub(r"[\.!]\$", "", s) # Remove trailing punctuation
```

```
    response = respond(s, mappings, reflections)
```

```
    print(response)
```

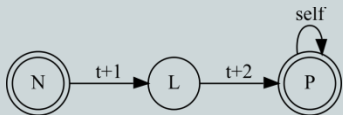


Reimplementation of Weizenbaum's code by Joe Strout, Jeff Epler, Jez Higgins in Python

How can we improve on Eliza?

◎ Try to make Eliza produce a blatant error:

- What kind of error is it?
 - Morphology
 - Syntax
 - Semantics
 - Pragmatics
 - Memory?
- How can we solve it?
 - More patterns
 - More reflections
 - Completely new mechanisms



Homework

- ◉ Please do the next assignment to practice RegEx in Canvas (pig_latin.py)
 - ◉ Due on **Monday**
- Instructions are in the script file+Canvas assignment

