intel
Software

Intro to Natural Language Processing (NLP)

What is Natural Language Processing (NLP)?

By "natural language" we mean a language that is used for everyday communication by humans.

NLP is an Intersection of several fields

- Computer Science
- Artificial Intelligence
- Linguistics

It is basically teaching computers to process human language

Two main components:

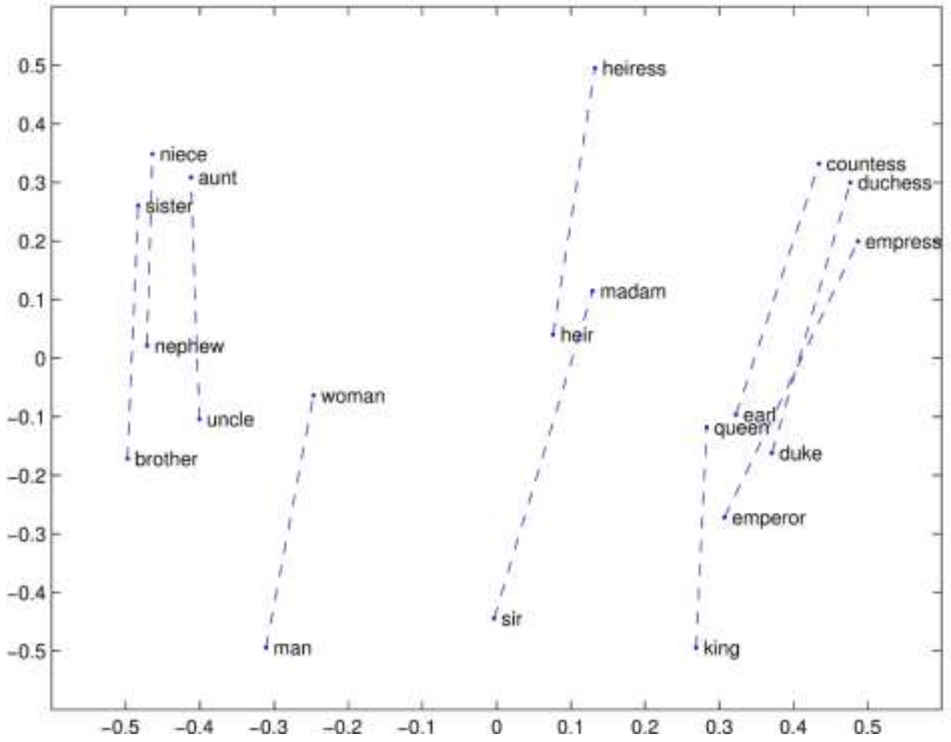
- Natural Language Understanding (NLU)
- Natural Language Generation (NLG)

NLP is AI Complete

- Requires all types of knowledge humans possess → It's hard!

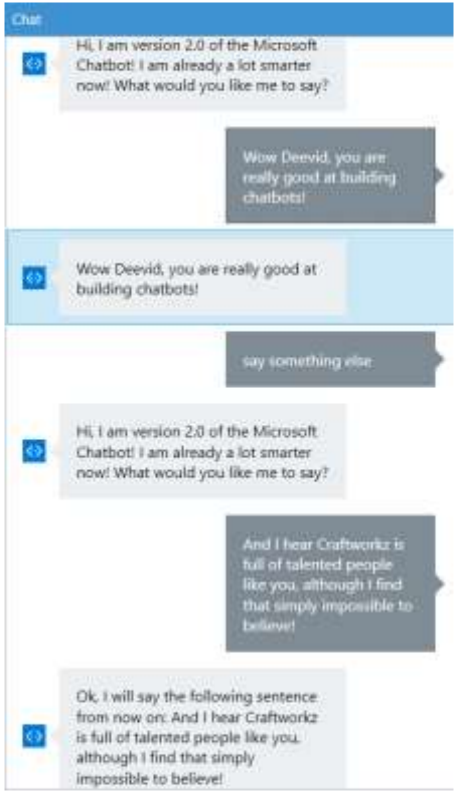
Natural Language Understanding (NLU)

- Deriving meaning from natural language
- Imagine a Concept (aka Semantic or Representation) space
  - In it, any idea/word/concept has unique computer representation
  - Usually via a vector space
  - NLU → Mapping language into this space



Natural Language Generation (NLG)

- Mapping from computer representation space to language space
- Opposite direction of NLU
  - Usually need NLU to perform NLG!
- NLG is really hard!


Chat

Hi, I am version 2.0 of the Microsoft Chatbot! I am already a lot smarter now! What would you like me to say?

Wow Deevid, you are really good at building chatbots!

Wow Deevid, you are really good at building chatbots!

say something else

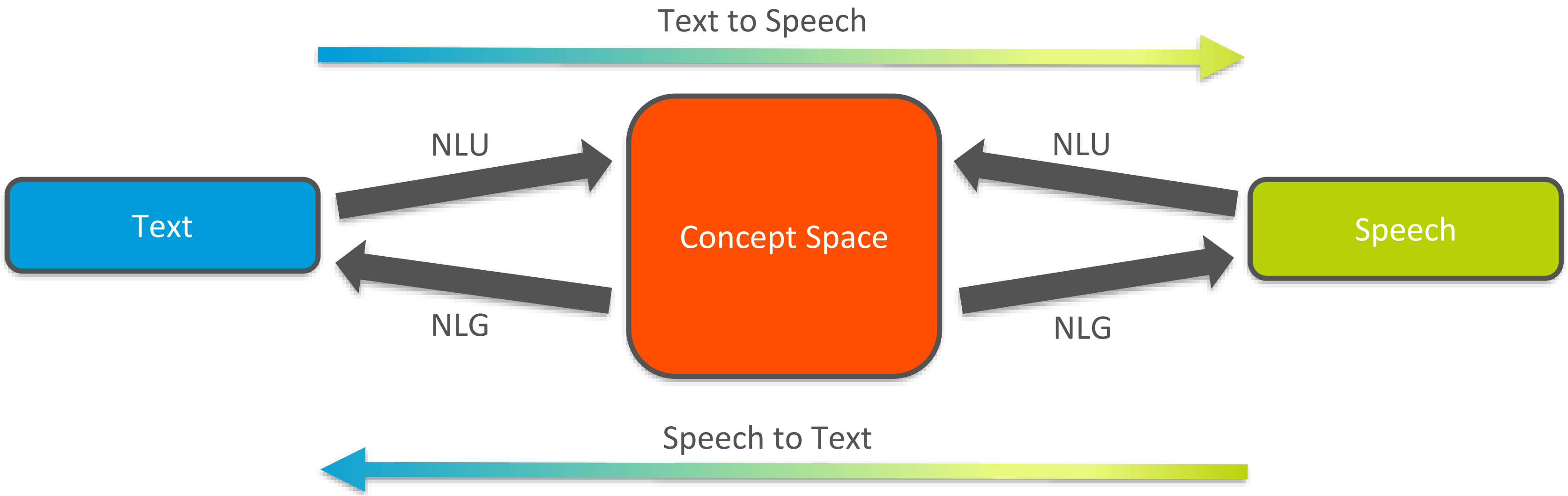
Hi, I am version 2.0 of the Microsoft Chatbot! I am already a lot smarter now! What would you like me to say?

And I hear Craftworkz is full of talented people like you, although I find that simply impossible to believe!

Ok, I will say the following sentence from now on: And I hear Craftworkz is full of talented people like you, although I find that simply impossible to believe!

NLP: Speech vs Text

- Natural Language can refer to Text or Speech
- Goal of both is the same: translate raw data (text or speech) into underlying concepts (NLU) then possibly into the other form (NLG)



History of NLP

- NLP has been through (at least) 3 major eras:
  - 1950s-1980s: Linguistics Methods and Handwritten Rules
  - 1980s-Now: Corpus/Statistical Methods
  - Now-???: Deep Learning
- Lucky you! You're right near the start of a paradigm shift!

1950s - 1980s: Linguistics/Rule Systems

- NLP systems focus on:
  - Linguistics: Grammar rules, sentence structure parsing, etc
  - Handwritten Rules: Huge sets of logical (if/else) statements
  - Ontologies: Manually created (domain-specific!) knowledge bases to augment rules above
- Problems:
  - Too complex to maintain
  - Can’t scale!
  - Can’t generalize!

1980s - Now: Corpus/Statistical Methods

- NLP starts using Machine Learning methods
- Use statistical learning over huge datasets of unstructured text
  - Corpus: Collection of text documents
  - e.g. Supervised Learning: Machine Translation
  - e.g. Unsupervised Learning: Deriving Word "Meanings" (vectors)

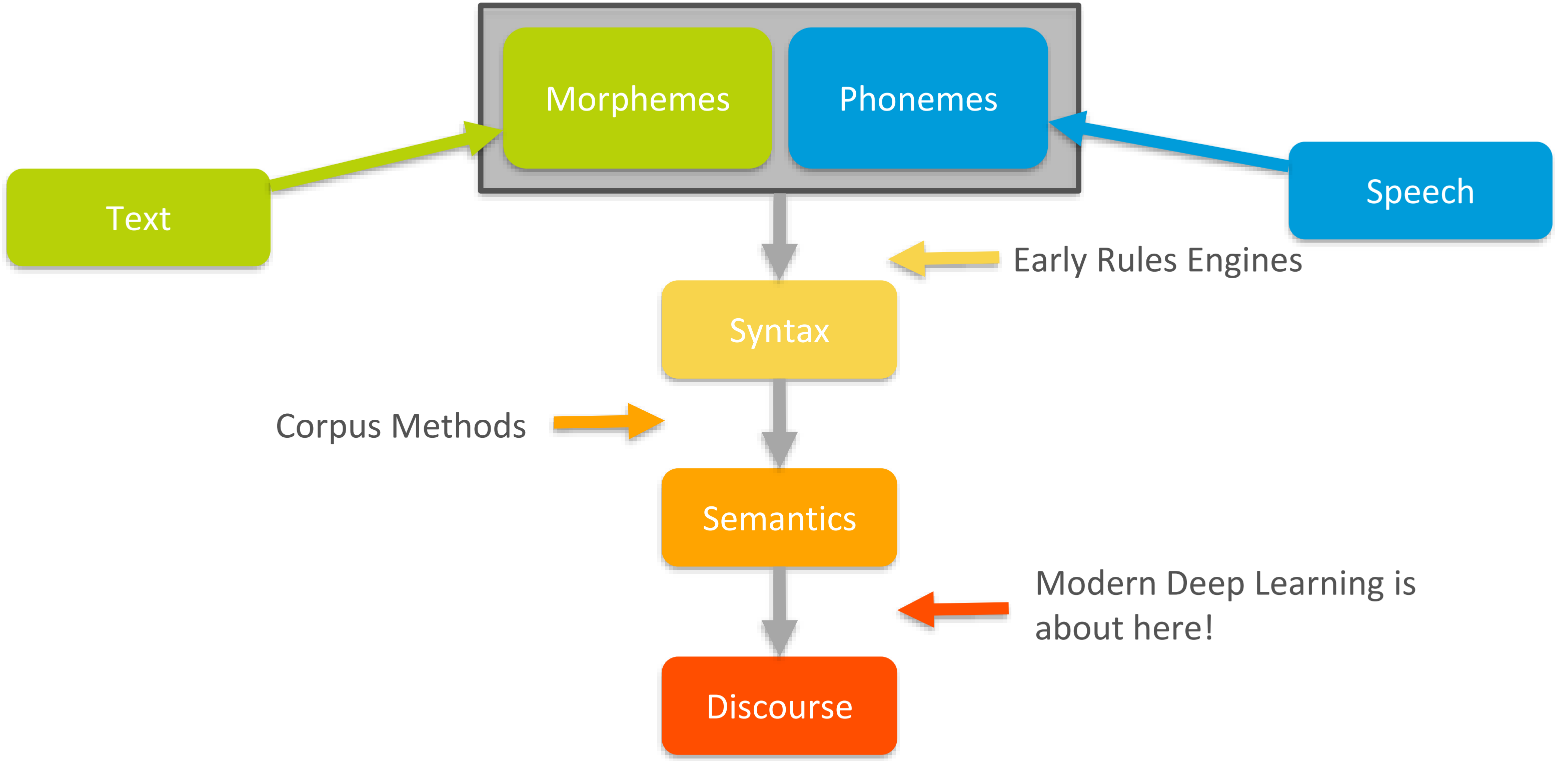
Now - ???: Deep Learning

- Deep Learning made its name with Images first
- 2012: Deep Learning has major NLP breakthroughs
  - Researchers use a neural network to win the Large Scale Visual Recognition Challenge (LSVRC)
  - This state of the art approach beat other ML approaches with half their error rate (26% vs 16%)
- Very useful for unified processing of Language + Images

NLP Definitions

- Phonemes: the smallest *sound* units in a language
- Morphemes: the smallest units of *meaning* in a language
- Syntax: how words and sentences are constructed from these two building blocks
- Semantics: the *meaning* of those words and sentences
- Discourse: semantics *in context*. Conversation, persuasive writing, etc.

Levels of NLP



NLU Applications

- ML on Text (Classification, Regression, Clustering)
- Document Recommendation
- Language Identification
- Natural Language Search
- Sentiment Analysis
- Text Summarization
- Extracting Word/Document Meaning (vectors)
- Relationship Extraction
- Topic Modeling
- …and more!

NLU Application: Document Classification

- Classify "documents" - discrete collections of text - into categories
  - Example: classify emails as spam vs. not spam
  - Example: classify movie reviews as positive vs. negative
  - Example: classify legal documents as relevant vs. not relevant to a topic

NLU Application: Document Recommendation

- Choosing the most relevant document based on some information:
  - Example: show most relevant webpages based on query to search engine
  - Example: recommend news articles based on past articles liked
  - Example: recommend restaurants based on Yelp reviews

explain

NLU Application: Topic Modeling

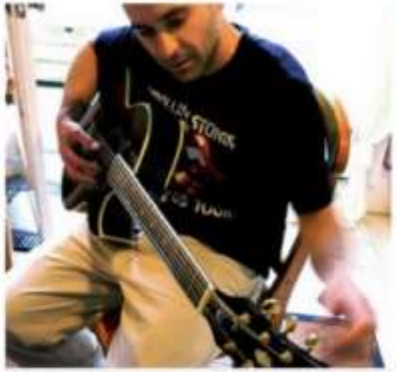
- Breaking a set of documents into topics at the word level
  - Example: see how prevalence of certain topics covered in a magazine changes over time
  - Example: find documents belonging to a certain topic

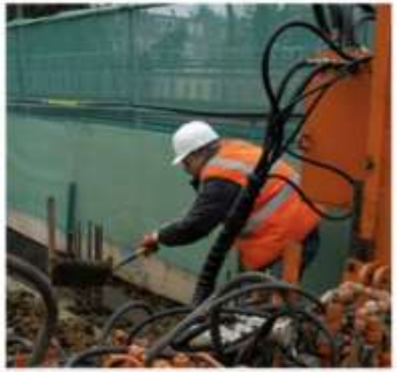
NLG Applications

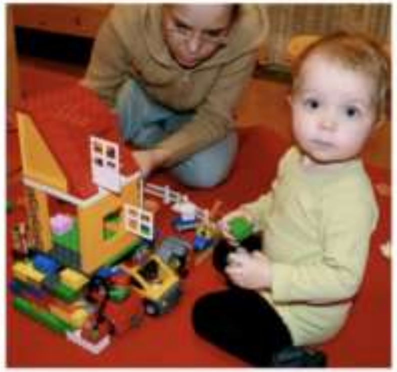
- Image Captioning
- (Better) Text Summarization
- Machine Translation
- Question Answering/Chatbots
- …so much more
- Notice NLU is almost a prerequisite for NLG

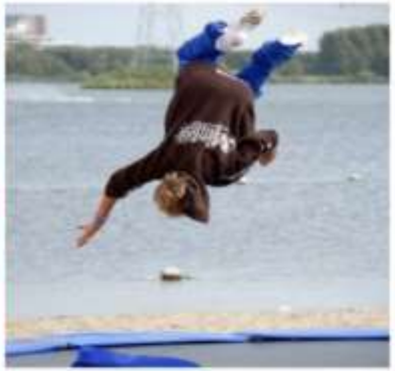
NLG Application: Image Captioning

- Automatically generate captions for images


man in black shirt is playing guitar.


construction worker in orange safety vest is working on road.


two young girls are playing with lego toy.

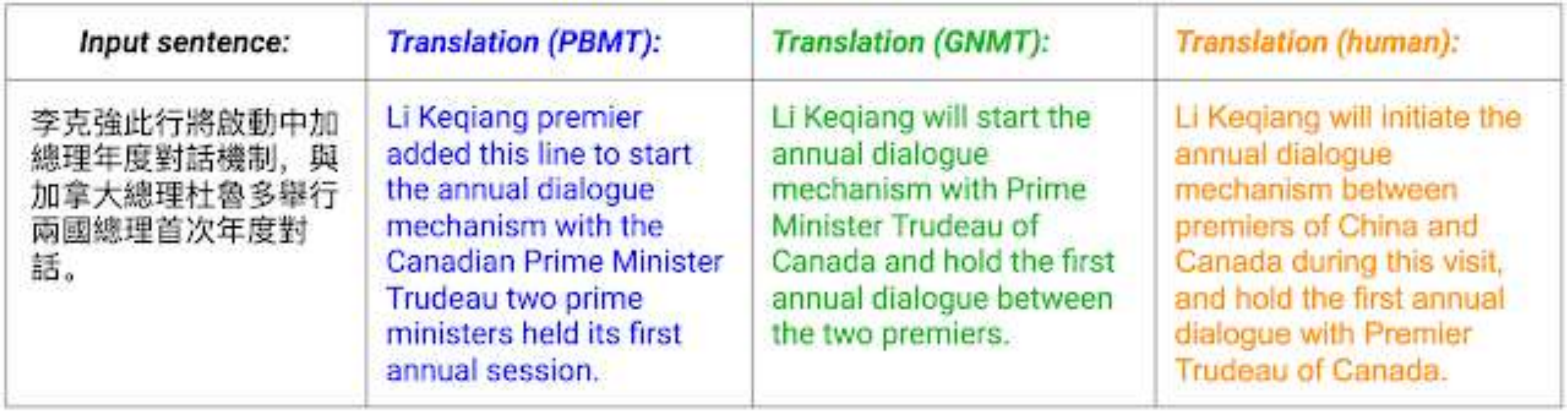

boy is doing backflip on wakeboard.

Captions automatically generated.

Source: https://cs.stanford.edu/people/karpathy/cvpr2015.pdf

NLG Application: Machine Translation

- Automatically translate text between language

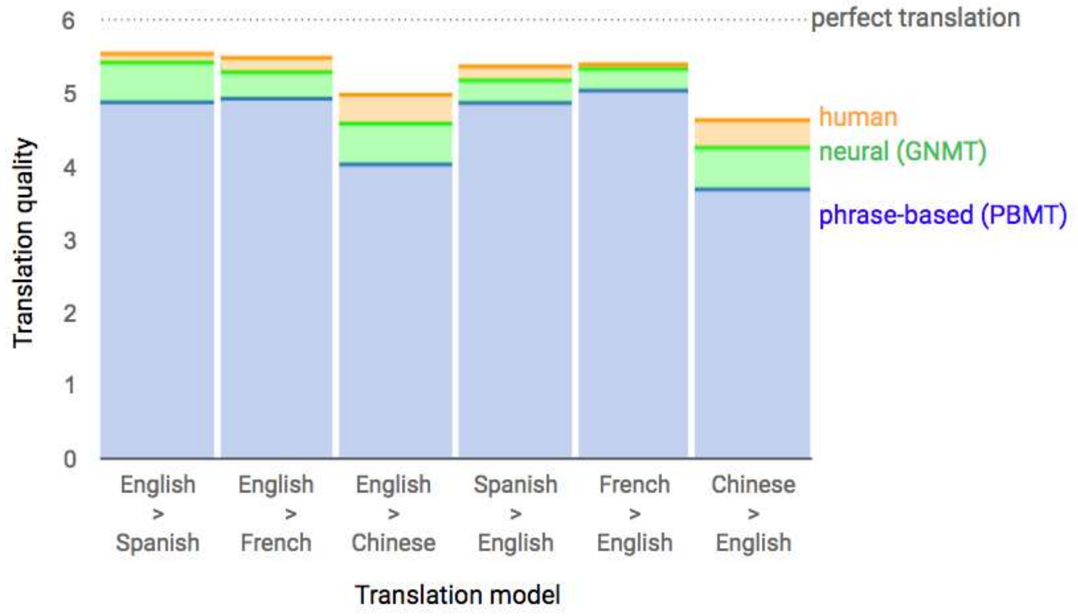


Input sentence:	*Translation (PBMT):*	*Translation (GNMT):*	*Translation (human):*
李克強此行將啟動中加總理年度對話機制，與加拿大總理杜魯多舉行兩國總理首次年度對話。	Li Keqiang premier added this line to start the annual dialogue mechanism with the Canadian Prime Minister Trudeau two prime ministers held its first annual session.	Li Keqiang will start the annual dialogue mechanism with Prime Minister Trudeau of Canada and hold the first annual dialogue between the two premiers.	Li Keqiang will initiate the annual dialogue mechanism between premiers of China and Canada during this visit, and hold the first annual dialogue with Premier Trudeau of Canada.

Example from Google®'s machine translation system (2016)

Source: https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html

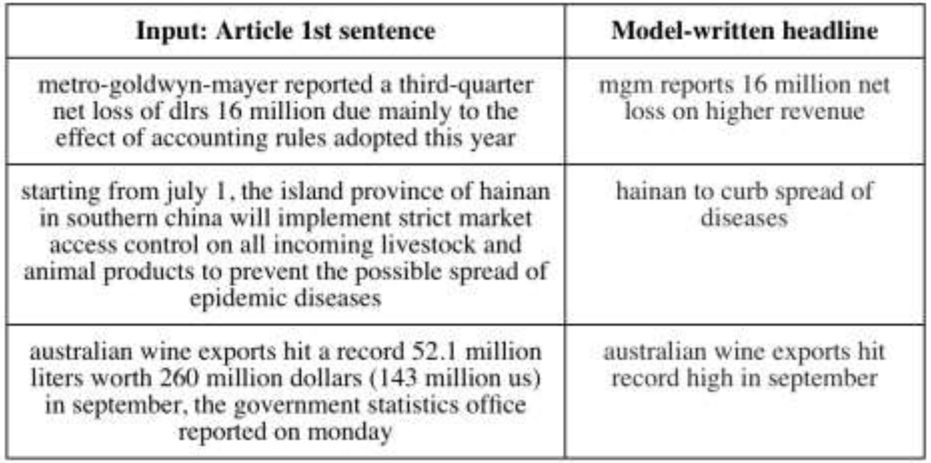
NLG Application: Machine Translation



Source: https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html

NLG Application: Text Summarization

- Automatically generate text summaries of documents
  - Example: generate headlines of news articles



Input: Article 1st sentence	Model-written headline
metro-goldwyn-mayer reported a third-quarter net loss of dlrs 16 million due mainly to the effect of accounting rules adopted this year	mgm reports 16 million net loss on higher revenue
starting from july 1, the island province of hainan in southern china will implement strict market access control on all incoming livestock and animal products to prevent the possible spread of epidemic diseases	hainan to curb spread of diseases
australian wine exports hit a record 52.1 million liters worth 260 million dollars (143 million us) in september, the government statistics office reported on monday	australian wine exports hit record high in september

Source: https://ai.googleblog.com/2016/08/text-summarization-with-tensorflow.html

Where We're Headed

- Today:
  - Regular Expression (regex) Basics
- Upcoming:
  - NLP Preprocessing Tasks (Week 2)
  - Deriving features from text (Week 3)
  - Machine Learning with Text (Week 4)
  - Topic Modeling on Text (Weeks 5-7)
  - Advanced Topics for NLG including Deep Learning (Week 8)

Regular Expressions

What are Regular Expressions (“RegEx”)?

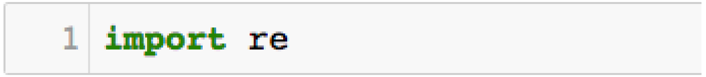
- A regex describes a pattern to look for in text
- Examples:
  - Does a string contain the word ”cat”?
  - Does a string contain 3 letters and then 2 numbers followed by a space?
  - Does a string contain one or more letters (and only letters)?
- Typical Usage:
  - Find strings that contain the regex pattern
  - Retrieve the part of a string matching the pattern

Use Cases for Regex

- Parsing documents with expected component structure
  - Use regex to grab the pieces you want
  - e.g. HTML, find all headers aka within <h> tags
  - e.g. remove known boilerplate sections from emails
- Validating User Input
  - e.g. does email match xxx@xxx.com?
- NLP Preprocessing
  - What pattern(s) represent individual words in text?
  - e.g. space + letters + space → grab the letters

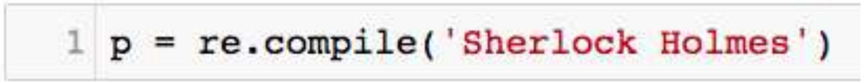
Regex in Python

- Import



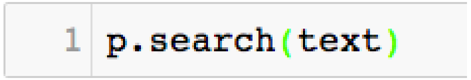
```python
import re
```

- Compile pattern

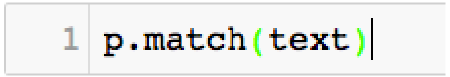


```python
p = re.compile('Sherlock Holmes')
```

- Pattern methods



```python
p.search(text)
```



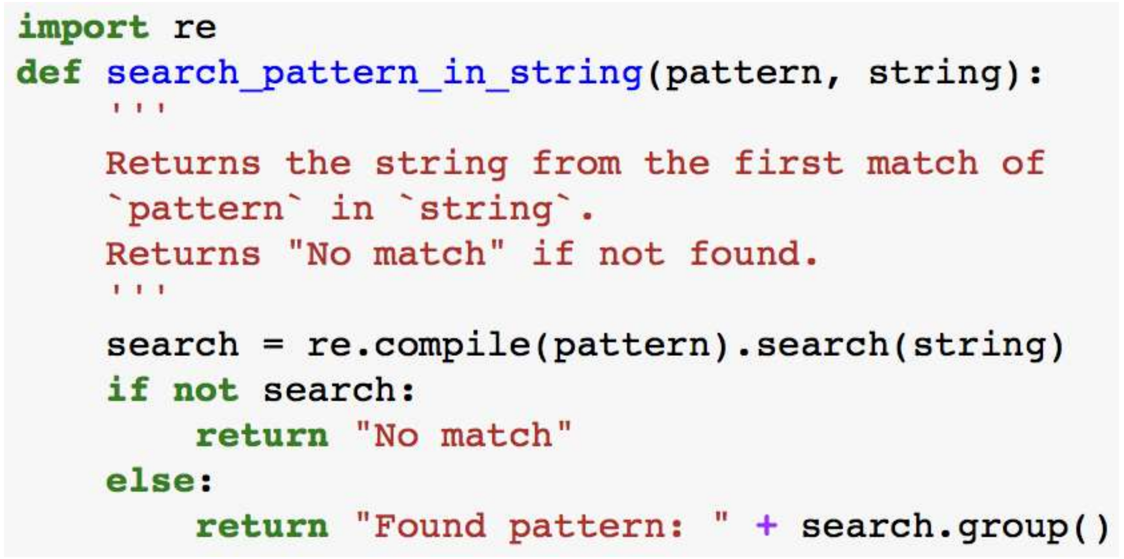
```python
p.match(text)
```

Metacharacters

- . Matches any single character
- ^ Beginning of string
- $ End of string
- * matches 0 or more characters
- + matches 1 or more characters
- ? Optional character

Metacharacters: Examples

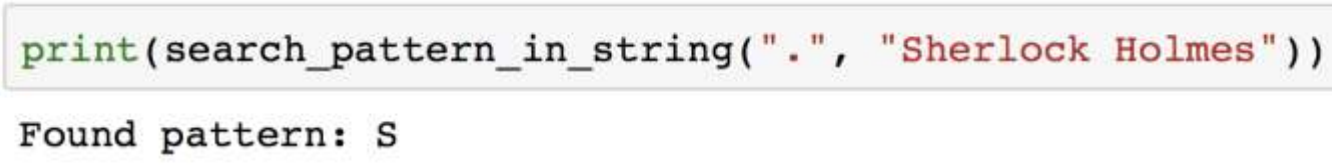
- Helper function:



```python
import re
def search_pattern_in_string(pattern, string):
    '''
    Returns the string from the first match of
    `pattern` in `string`.
    Returns "No match" if not found.
    '''
    search = re.compile(pattern).search(string)
    if not search:
        return "No match"
    else:
        return "Found pattern: " + search.group()
```

Metacharacters: Examples

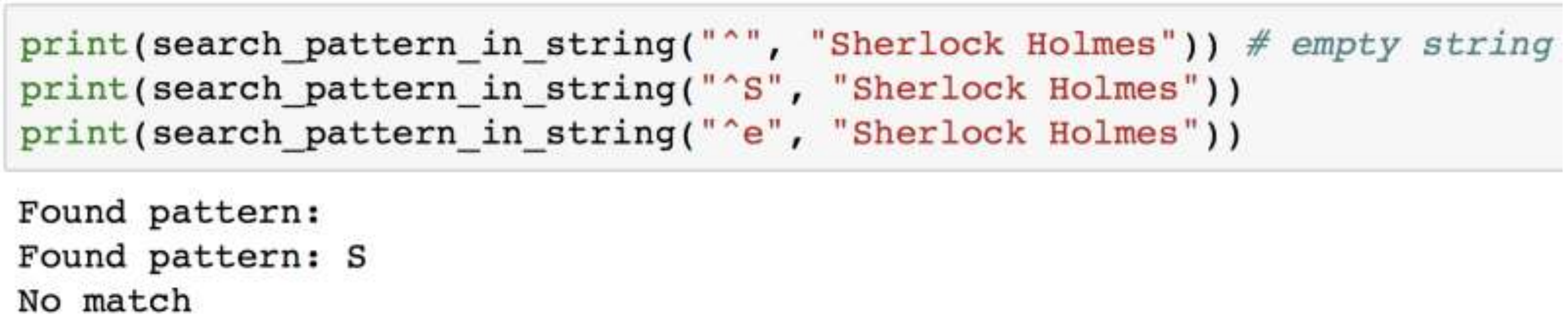
- . Matches any single character



```
print(search_pattern_in_string(".", "Sherlock Holmes"))
```

```
Found pattern: S
```

- ^ Beginning of string

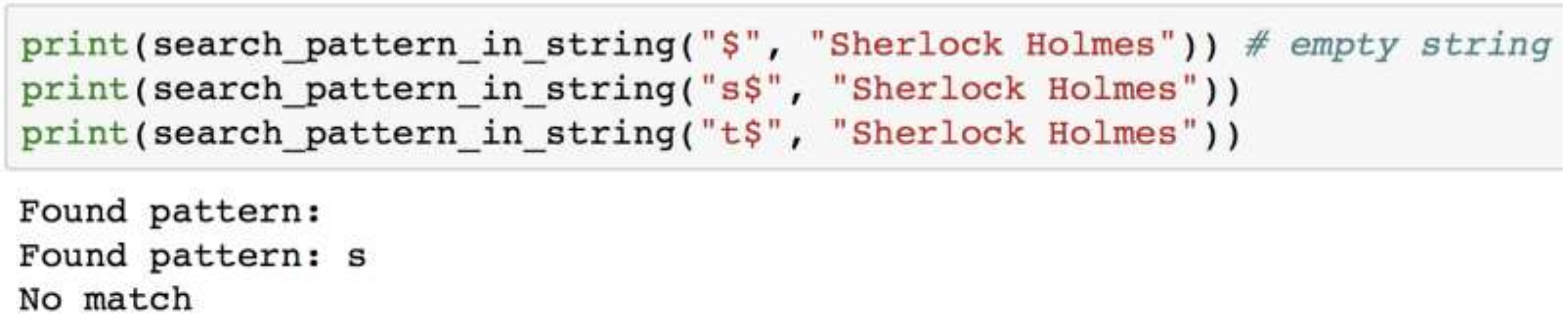


```
print(search_pattern_in_string("^", "Sherlock Holmes")) # empty string
print(search_pattern_in_string("^S", "Sherlock Holmes"))
print(search_pattern_in_string("^e", "Sherlock Holmes"))
```

```
Found pattern: 
Found pattern: S
No match
```

Metacharacters: Examples

- $ End of string

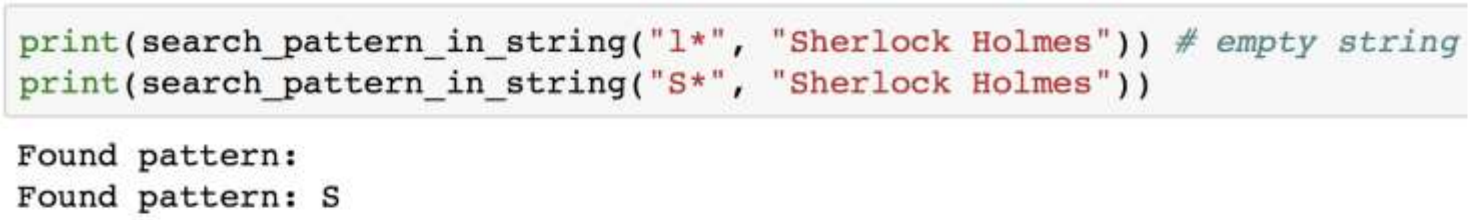


```
print(search_pattern_in_string("$", "Sherlock Holmes")) # empty string
print(search_pattern_in_string("s$", "Sherlock Holmes"))
print(search_pattern_in_string("t$", "Sherlock Holmes"))
```

```
Found pattern: 
Found pattern: s
No match
```

Metacharacters: Examples

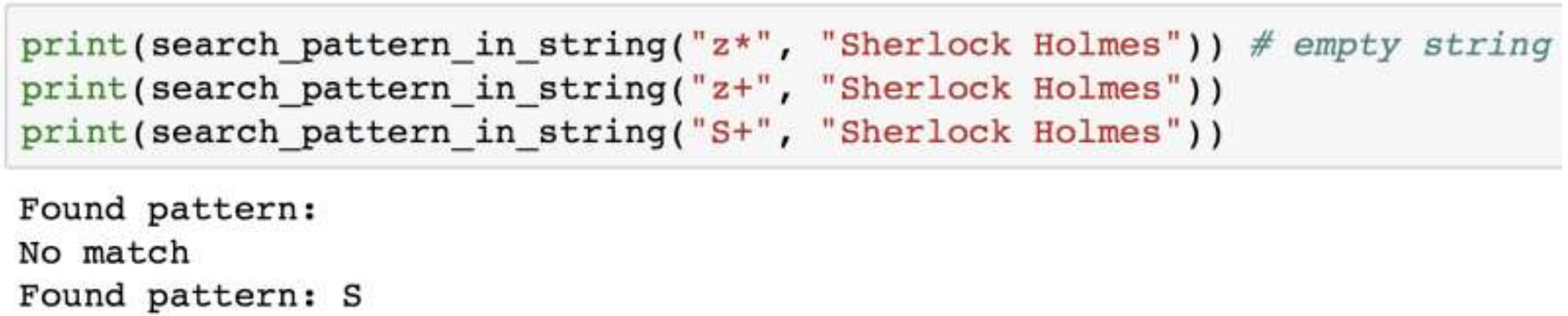
- *   matches 0 or more characters



```
print(search_pattern_in_string("l*", "Sherlock Holmes")) # empty string
print(search_pattern_in_string("S*", "Sherlock Holmes"))
```

```
Found pattern:
Found pattern: S
```

- +   matches 1 or more characters

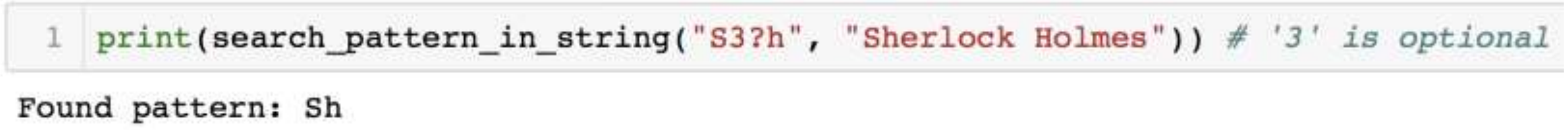


```
print(search_pattern_in_string("z*", "Sherlock Holmes")) # empty string
print(search_pattern_in_string("z+", "Sherlock Holmes"))
print(search_pattern_in_string("S+", "Sherlock Holmes"))
```

```
Found pattern:
No match
Found pattern: S
```

Metacharacters: Examples

- ? Optional character



```
print(search_pattern_in_string("S3?h", "Sherlock Holmes")) # '3' is optional
```

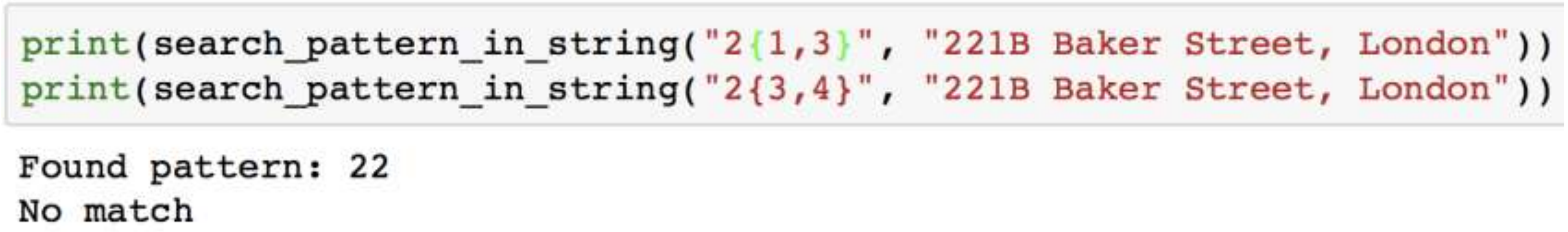
```
Found pattern: Sh
```

More Metacharacters

- { m,n}   specify number of times character is matched between m and n times
- [ ]   list characters to be matched
- \   escape character
- | or
- ( ) capture group inside parenthesis

Metacharacters: Examples

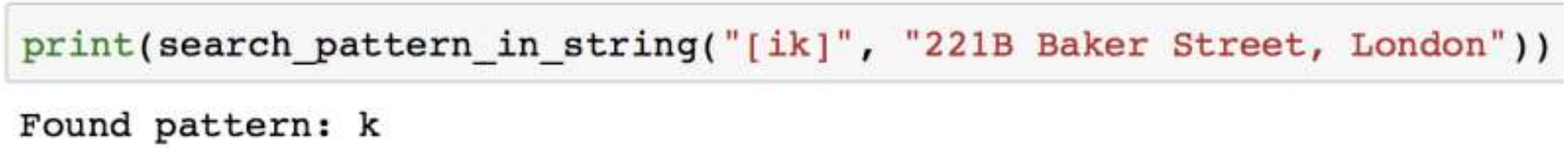
- { m,n}   specify number of times character is matched between m and n times



```
print(search_pattern_in_string("2{1,3}", "221B Baker Street, London"))
print(search_pattern_in_string("2{3,4}", "221B Baker Street, London"))
```

```
Found pattern: 22
No match
```

- [ ]   list characters to be matched

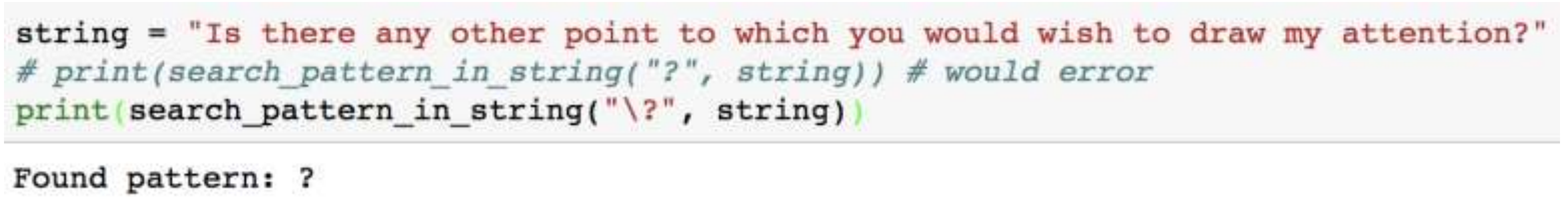


```
print(search_pattern_in_string("[ik]", "221B Baker Street, London"))
```

```
Found pattern: k
```

Metacharacters: Examples

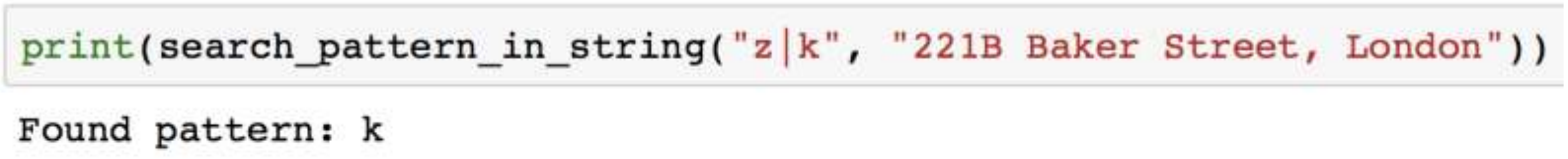
- \ escape character



```
string = "Is there any other point to which you would wish to draw my attention?"
# print(search_pattern_in_string("?", string)) # would error
print(search_pattern_in_string("\?", string))
```

```
Found pattern: ?
```

- | or



```
print(search_pattern_in_string("z|k", "221B Baker Street, London"))
```

```
Found pattern: k
```

Character Classes

- \s - matches any whitespace
- \w - matches any alpha character. Equivalent to [A-Za-z]
- \d - matches any numeric character. Equivalent to [0-9]

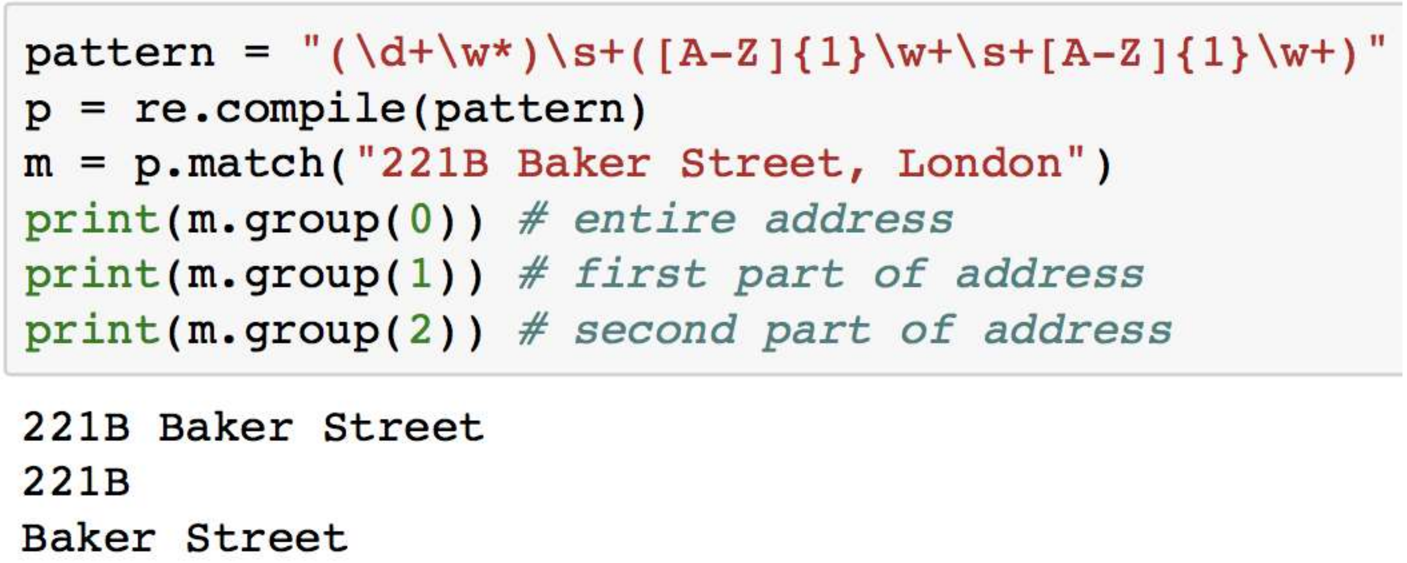
You may negate these by capitalizing. For example, \D matches anything not a digit

Handling Groups

- ()

Parenthesis control which parts of the string are returned after a match

Character Classes and Groups Example

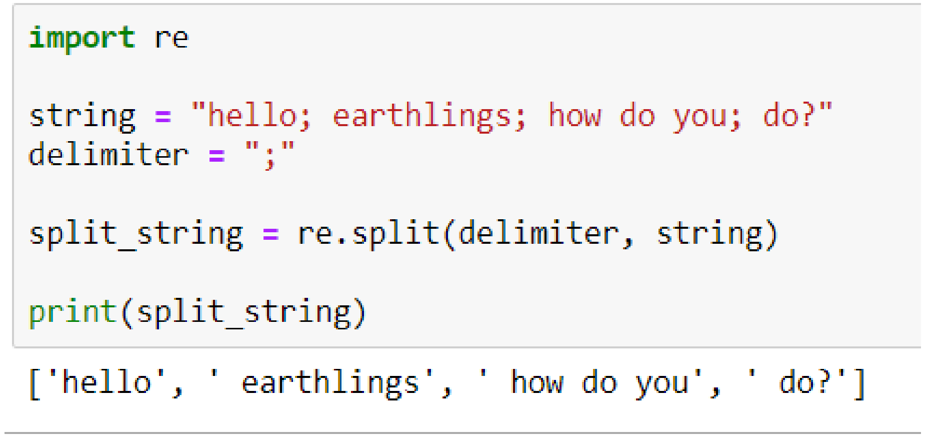


```
pattern = "(\d+\w*)\s+([A-Z]{1}\w+\s+[A-Z]{1}\w+)"
p = re.compile(pattern)
m = p.match("221B Baker Street, London")
print(m.group(0)) # entire address
print(m.group(1)) # first part of address
print(m.group(2)) # second part of address
```

```
221B Baker Street
221B
Baker Street
```

Splitting by Regex

- re.split()
- Can be used to split a string on any REGEX match
- Frequently done when you don't know the type of whitespace or to handle dashes and underscores



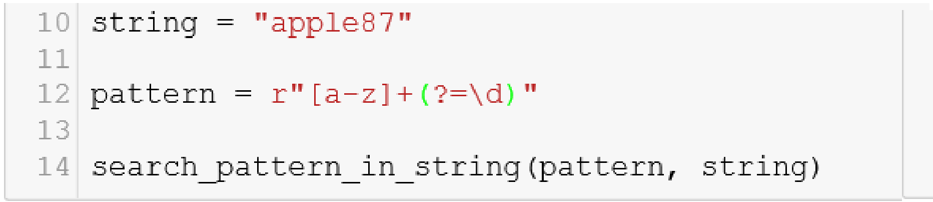
```
import re

string = "hello; earthlings; how do you; do?"
delimiter = ";"

split_string = re.split(delimiter, string)

print(split_string)
```

```
['hello', ' earthlings', ' how do you', ' do?']
```

Lookaheads/Lookbehinds

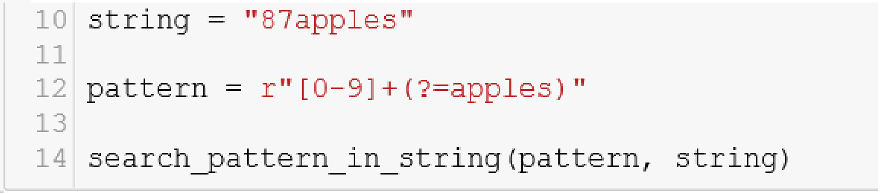
examples

- Allow you to keep the cursor in place, but still check to see if certain conditions are met before or after that character
- Look ahead: (?=\d) checks to see if the character is followed by a number
- Look Behind: (?<=[aeiou]) checks to see if the character is preceded by a vowel



```python
string = "apple87"

pattern = r"[a-z]+(?=\d)"

search_pattern_in_string(pattern, string)
```

```
'found pattern: apple'
```

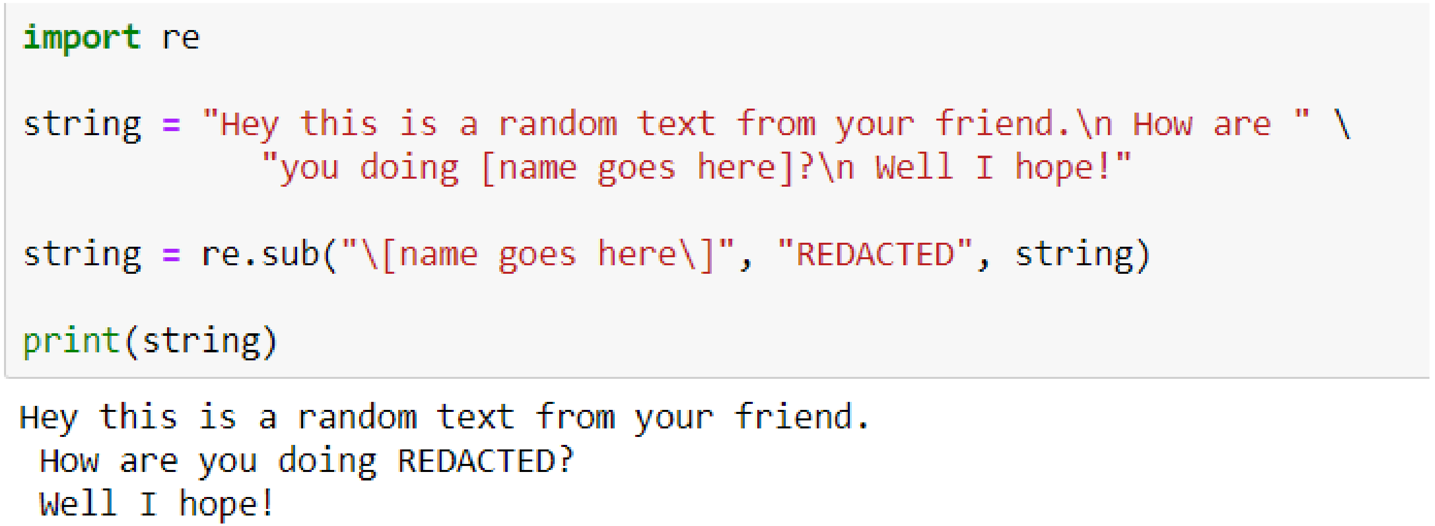


```python
string = "87apples"

pattern = r"[0-9]+(?=apples)"

search_pattern_in_string(pattern, string)
```

```
'found pattern: 87'
```

Search and Replace

- re.sub() allows for quick and easy search and replace

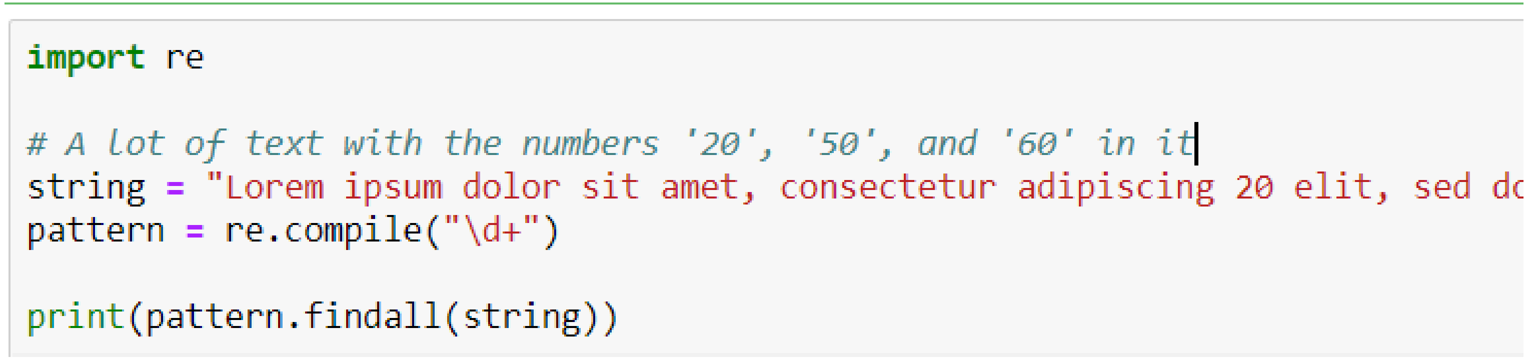


```
import re

string = "Hey this is a random text from your friend.\n How are " \
          "you doing [name goes here]?\n Well I hope!"

string = re.sub("\[name goes here\]", "REDACTED", string)

print(string)
```

```
Hey this is a random text from your friend.
 How are you doing REDACTED?
 Well I hope!
```

Find all

- Frequently we will want to find all occurrences of a certain bit of text in a corpus
- pattern.findall(text) is very useful for doing this



```
import re

# A lot of text with the numbers '20', '50', and '60' in it
string = "Lorem ipsum dolor sit amet, consectetur adipiscing 20 elit, sed dc
pattern = re.compile("\d+")

print(pattern.findall(string))
```

Regex Flags

- re.I - ignore case
- re.L - locale dependent
- re.M - Multiline
- re.S - dot matches all
- re.U unicode dependent
- re.X verbose

Greedy vs Non-Greedy

- + and * operators are greedy, they try to match as much as they can.

  <div>this is some text</div>

  <.+> will not match <div> it will match the entire line.

  To make it non greedy (lazy) you can add a ? After the operator

  <.+?> will match just <div>

intel®

Software