Session 9

Scraping 2 - Parsing

Snorre Ralund

Yesterday I gave you some powerful tricks. Tricks that will work when the data is already shipped in a neat format. However this is not the rule. Today we shall learn the art of parsing unstructured text and a more principled and advanced method of parsing HTML.

This will help you build ***custom datasets*** within just a few hours or days work, that would have taken ***months*** to curate and clean manually.

Agenda

Parsing and cleaning raw data

HTML

- Understanding the basics of HTML syntax.
- Traversing and Navigating HTML trees using BeautifulSoup. Examples include:
  - Extracting Text from HTML,
  - Extracting Tables,
  - Parsing of "unknown" structures.

Raw Text

- Learning the of Regular Expressions for extracting patterns in strings.
  - Very valuable when cleaning and validating data, and for information extraction from raw text.

Sidestep: Interactions and Automated Browsing

Sometimes scraping tasks demand interactions (e.g. login, scrolling, clicking), and a no XHR data can be found easily, so you need the browser to execute the scripts before you can get the data.

Here we use the `Selenium` package in combination with the `geckodriver` - download the latest release here (download the latest geckodriver here: https://github.com/mozilla/geckodriver/releases). It allows you to animate a browser. I won't go into detail here, but just wanted to mention it.

Installation (and maintainance of compatability) can be a little tough, but instead of trawling through crazy stackoverflow threads about your Issue, my experience tells me that downloaded the latest release, and installing the latest selenium version is always the cure.

In [ ]:

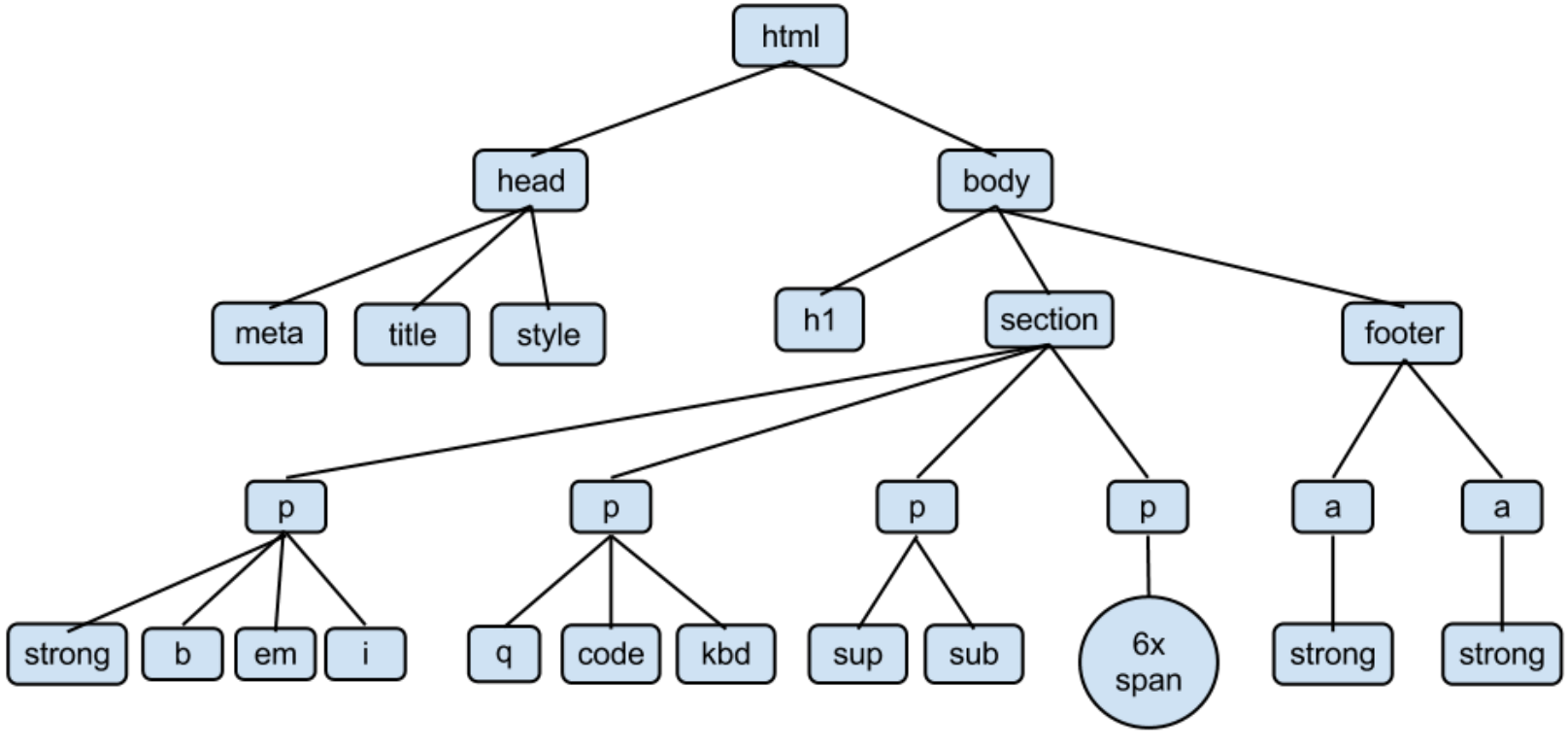
```
from selenium import webdriver
## download the latest geckodriver here: https://github.com/mozilla/geckodriver/releases
path2gecko = 'C:/Users/jbv933/Downloads/geckodriver-v0.21.0-win64/geckodriver.exe' # def
ine path to your geckodriver
browser = webdriver.Firefox(executable_path=path2gecko) # start the browser with a path
 to the geckodriver.
browser.get('https://www.facebook.com') # opens a webpage using the browser objects get
 method.
selector = '#email' # define selector to the email field
element = browser.find_element_by_css_selector(sel) # find this element using the .find_
element_by_css_selector method
email = 'sdfsjsd@hotmail.com' # define email to be send to the field
element.send_keys(email) # send the data
selector = '#pass' # define the css selector to the password field
element = browser.find_element_by_css_selector(sel) # find the element
# send password
login_sel = '#u_0_2' # define the selector to the login button
element = browser.find_element_by_css_selector(sel) # find the element
element.click() # click on the login button to send password and email.
```

The HTML Tree

HTML has a Tree structure.

Each node in the tree has:

```
- Children, siblings, parents - descendants.
- Ids and attributes
```



Important syntax and patterns

```
<p>The p tag indicates a paragraph <p/>
```

```
<b>The b tag makes the text bold, giving us a clue to its importance <b/>
```

output: **The b tag makes the text bold, giving us a clue to its importance**

```
<h1>h1</h1><h2>h2</h2><h2>h3</h3><b>Headers give similar clues</b>
```

output:

h1

h2

h3

Headers give similar clues

```
<a href="www.google.com">The a tag creates a hyperlink <a/>
```

output: The a tag creates a hyperlink (www.google.com)

Finally you have the terrible and confusing iframe:

```
<iframe src="https://www.google.com"></iframe>
```

How do we find our way around this tree?

- **extracting string patterns using .split and regular expresssions as we did yesterday.**
- **Specifying paths using css-selectors,xpath syntax.**
- **A more powerful and principled (+readable) way is to use the `python` module `BeautifulSoup` to parse and traverse the tree.**

Selectors

Define a unique path to an element in the HTML tree.

- **quick but has to be hardcoded and also more likely to break.**

Parsing HTML with BeautifulSoup

BeautifulSoup makes the html tree navigable. It allows you to:

```
* Search for elements by tag name and/or by attribute.
* Iterate through them, go up, sideways or down the tree.
* Furthermore it helps you with standard tasks such as extracting raw text from
 html,
which would be a very tedious task if you had to hardcode it using `.split` comm
ands and using your own regular expressions will be unstable.
```

In [15]:
```
# scraping newspaper articles example.
url = 'https://www.theguardian.com/us-news/2018/aug/10/omarosa-trump-book-the-apprentice
-memoir'
import requests
response = requests.get(url)
html = response.text
```

In [16]:
```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html,'lxml') # parse the raw html using BeautifoulSoup
```

In [25]:
```
# extract hyperlinks
links = soup.find_all('a') # find all a tags -connoting a hyperlink.
link_tag = links[-10]
link_tag['href']
[link['href'] for link in links if link.has_attr('href')][0:5] # unpack the hyperlink fr
om the a nodes.
```

Out[25]:
```
['#maincontent',
 'https://www.theguardian.com/preference/edition/int',
 'https://www.theguardian.com/preference/edition/uk',
 'https://www.theguardian.com/preference/edition/us',
 'https://www.theguardian.com/preference/edition/au']
```

In [34]:
```
headline = soup.find('h1') # search for the first headline: h1 tag.
#headline.text.strip().replace('Trump','idiot')
#headline
name = headline['class'][0].strip() # use the class attribute name as column name.
value = headline.text.strip() # extract text using build in method.
print({name:value})
```

```
{'content__headline': 'Omarosa says Trump is a racist who uses N-word –\xa0and claims
there is tape to prove it'}
```

In [38]:
```
article_text = soup.find('div',{'class':'content__article-body from-content-api js-artic
le__body'})#.text # find the content.
article_text.text
#article_text.find_all('p')
```

Out[38]:
```
"\nDonald Trump is a “racist” who has used the “N-word” repeatedly, Omarosa Manigault
Newman, once the most prominent African American in the White House, claims in a sear
ing memoir.\nThe future US president was caught on mic uttering the taboo racial slur
“multiple times” during the making of his reality TV show The Apprentice and there is
a tape to prove it, according to Manigault Newman, citing three unnamed sources.\nTru
mp has been haunted from around the time of his election in 2016 by allegations that
outtakes from the reality TV show exist in which he is heard saying the N-word and us
ing other offensive language.\n\n\n\nSign up to receive the top US stories every morn
ing\n\n\nIn her book, Unhinged, a copy of which was obtained by the Guardian ahead of
its publication next week, the former Apprentice participant  insists that the report
s are true, although she does not say she heard him use the word herself.\nShe also c
laims that she personally witnessed Trump use racial epithets about the White House c
ounselor Kellyanne Conway’s husband George Conway, who is half Filipino. “Would you l
ook at this George Conway article?” she quotes the president as saying. “F**ing FLIP!
Disloyal! Fucking Goo-goo.”\nBoth flip and goo-goo are terms of racial abuse for Fili
pinos.\nCritics have previously questioned Manigault Newman’s credibility and are lik
ely to accuse her of seeking revenge against the administration after her abrupt dism
issal last December.\n\n\n\n\nAttack ads, lawsuits, insults and fights: Trump's charg
ed history with race\n\n\n\n\n\n\n \n\n\nRead more\n\n\n\n\n\n\nAt the time, she writ
es, she felt a “growing realization that Donald Trump was indeed a racist, a bigot an
d a misogynist. My certainty about the N-word tape and his frequent uses of that word
were the top of a high mountain of truly appalling things I’d experienced with him, d
uring the last two years in particular.”\nRecalling her sudden and unceremonious depa
rture, she writes: “It had finally sunk in that the person I’d thought I’d known so w
ell for so long was actually a racist. Using the N-word was not just the way he talks
but, more disturbing, it was how he thought of me and African Americans as a whol
e.”\nTrump hosted NBC’s The Apprentice from 2004-2015 before running for the presiden
cy and still likes to laud his high ratings.\nHis insurgent election campaign was roc
ked in October 2016 by the release of an Access Hollywood tape in which he bragged ab
out grabbing women “by the pussy”. The media firestorm prompted Bill Pruitt, a produc
er on the first two seasons of The Apprentice, to tweet that there were “far worse” t
apes of Trump behind the scenes of the show.\nFurther allegations emerged that Trump
```

had used the N-word in the recordings. Then, following the New York property tycoon’s
shocking victory over Hillary Clinton, the actor and comedian Tom Arnold claimed to h
ave the video of Trump using racist language.\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n \n
\n\n \n\n \n\n\n Facebook \n \n\n\n Twitter \n \n\n\n Pinterest \n\n\n\n\n\n The
White House staffer Kellyanne Conway and former aides Hope Hicks and Omarosa Manigaul
t Newman at a press briefing last year. Photograph: Alex Wong/Getty Images\n\n\n“I ha
ve the outtakes to The Apprentice where he says every bad thing ever, every offensiv
e, racist thing ever,” Arnold told the Seattle-based radio station KIRO. “It was him
sitting in that chair saying the N-word, saying the C-word, calling his son a retard,
just being so mean to his own children.”\nBut Metro-Goldwyn-Mayer, which owns the rig
hts to the reality TV show, and its British creator, Mark Burnett, have resisted pres
sure to release the footage because of “various contractual and legal requirement
s”.\nOnce close to Trump, Manigault Newman was among his most high-profile supporters
during the election campaign and drew a top salary of $179,700 as director of communi
cations for the White House office of public liaison. She held her April 2017 wedding
at Trump’s luxury hotel, close to the White House.\nHers is the second memoir from a
former Trump administration member, following that of the ex-press secretary Sean Spi
cer, but it was always expected to be less adulatory. This week the Daily Beast repor
ted that she had secretly recorded conversations with the president and “leveraged” t
his while seeking a book deal. On Sunday she is due to appear on NBC’s flagship polit
ical show Meet the Press.\nSome commentators have struck a note of scepticism about h
er book. Brian Stelter, senior media correspondent at CNN, wrote in an email newslett
er: “Is former ‘Apprentice’ star Omarosa Manigault-Newman a reliable source of info a
bout the Trump White House? Buckle up for debates about that in the coming week. Beca
use she’s about to betray Trump in a new tell-all book.”\nFor its part, the White Hou
se has previously dismissed criticisms from her. In February the deputy press secreta
ry, Raj Shah, said: “Omarosa was fired three times on The Apprentice and this was the
fourth time we let her go. She had limited contact with the president while here. She
has no contact now.”\nIn the book, she recalls how in late 2016 Trump’s team held a c
onference call and scrambled for how to respond to the tape but it never came out. Th
en a source from The Apprentice contacted her and claimed to be in possession of it.
Trump was in office and Manigault Newman continued to investigate.\nShe continues: “B
y that point, three sources in three separate conversations had described the content
s of this tape. They all told me that President Trump hadn’t just dropped a single N-
word bomb. He’d said it multiple times throughout the show’s taping during off-camera
outtakes, particularly during the first season of The Apprentice.”\nRecalling that sh
e appeared on the first season, Manigault Newman reflects: “I would look like the big
gest imbecile alive for supporting a man who used that word.” She says she confided i

n the former White House communications director Hope Hicks, who said, “I need to hea r it for myself,” and continued to ask her frequently about what progress she was mak ing.\nShe believes that Hicks told the White House chief of staff, John Kelly, that O marosa was close to getting her hands on the tape, and this gave him cause to termina te her job, though he found a different pretext.\nFour months after her departure, sh e spoke by phone to one of her Apprentice sources. “I was told exactly what Donald Tr ump said – yes, the N-word and others in a classic Trump-goes-nuclear rant – and when he’s said them. During production he was miked, and there is definitely an audio trac k.”\nManigault Newman also recalls her interactions with Trump during the filming of The Celebrity Apprentice in late 2007 – a time when the little known Democrat Barack Obama was in the ascendent. “During boardroom outtakes, Donald talked about Obama oft en. He hated him. He never explained why, but now I believe it was because Obama was black.”\n\n\n\n\nFormer Apprentice star on leaving Trump's White House: 'I've seen th ings that made me uncomfortable'\n\n\n\n\n\n\n\n \n\n\nRead more\n\n\n\n\n\n\n\nIn Januar y, Trump was widely condemned for reportedly dismissing Haiti, El Salvador and Africa n nations as “shitholes”. Manigault Newman describes a similar experience that appear s to support that account. When she told Trump that she was going to Haiti, she write s, he demanded: “Why did you choose that shitty country as your first foreign tri p?”\nHe added: “You should have waited until the confirmations were done and gone to Scotland and played golf at [his course] Turnberry.”\nIn another damning passage, she describes his “broken outlook” and how “the bricks in his racist wall kept getting hi gher”, wondering if he did “want to start a race war”. She adds: “The only other expl anation was that his mental state was so deteriorated that the filter between the wor st impulses of his mind and his mouth were completely gone.”\nThe book comes days aft er Trump faced renewed allegations of racism over his persistent descriptions of the congresswoman Maxine Waters as having a “low IQ” and CNN journalist Don Lemon as “the dumbest man on television”, as well as criticism of the basketball star LeBron James. This weekend marks the first anniversary of the white supremacist march in Charlottes ville, Virginia, that erupted in deadly violence; the president claimed there were “v ery fine people on both sides”.\nElsewhere in Unhinged, published by Gallery Books, a n imprint of Simon & Schuster, Manigault Newman takes aim at Trump’s sexism. Recallin g more outtakes from The Apprentice, she says he asked personal questions about femal e contestants such as “What do you think she’s like in bed?” and “Do you think she’s sexy?” He allegedly asked male contestants: “Who you think would be better in bed bet ween the two of them?”\nAsked about the allegations in Manigault Newman’s book, White House press secretary Sarah Sanders said: “Instead of telling the truth about all the good President Trump and his administration are doing to make America safe and prospe rous, this book is riddled with lies and false accusations. \n“It’s sad that a disgru

Say we are interested in how articles cite sources to back up their story i.e. their hyperlink behaviour within the article, and we want to see if the media has changed their behaviour.

We know how to search for links. But the cool part is that we can search from anywhere in the HTML tree. This means that once we have located the article content node - as above - we can search from there. This results in hyperlinks used within the article text.

In [321]:

```
# find the article_content node
article_content = soup.find('div',{'class':'content__article-body from-content-api js-ar
ticle__body'})
# find citations within the article content.
citations = article_content.find_all('a')
```

In [325]:
```
citation_links = [] # define container to the hyperlinks
for citation in citations: # iterate through each citation node
    if citation.has_attr('data-link-name'): # check if it has the right attribute
        if citation['data-link-name'] =='in body link': # and if the value of that attri
bute is correct
            #print(citation['href'])
            citation_links.append(citation['href']) #  add link to the container
```

Scraping without hardcoding every element to collect.

Hands-on example: Continuing the Cryptomarket scrape.

When scraping a large and diverse website or even crawling many different, it can be useful to design more generic parsing schemes, were you haven't seen all elements you want to keep before hand. In the following example I demonstrate a simple example of this.

Imagine we wanted data on the Cryptomarkets:

- **Go to the front page of a Cryptomarket page (https://coinmarketcap.com). Looking in the >Network Monitor< we find a XHR file (helping their search function) containing links to Cryptocoin. Now we have the link to each page we want to visit.**

Visit this example: http://coinmarketcap.com/currencies/ethereum/ (http://coinmarketcap.com/currencies/ethereum/)

Yesterday we saw how to find the XHR file with the underlying data behind the Chart. Now we want all the metadata displayed.

Using the inspector we find that the information we want is located by a node tagged with `ul` - *"unordered list"* tag - with the defining class attribute *'list-unstyled details-panel-item--links'*. And the information is located in the `li` tag *"list item"*.

In [216]:
```
url = 'https://coinmarketcap.com/currencies/ethereum/' # define the example url
response = requests.get(url) #
soup = BeautifulSoup(response.text,'lxml') # parse the HTML
```

In [221]:
```
list_node = soup.find('ul',{'class':'list-unstyled details-panel-item--links'}) # search
 for the ul node
```

Now starting from this list_node, we can search for each list item - li node, using the find_all method.

In [326]:
```
list_items = list_node.find_all('li') # search for all list elements children of the lis
t_node
```

From this we extract the information. Without having to hardcode all extractions we exploit that each html node has a attribute ('title'). We can therefore just loop through each node, extracting the title attribute and the text. -- *This furthermore allows us to scrape content we did not know was there.*

We use the title as the key in the dictionary. The value we are interested in two things, either the *hyperlink*, or the the *text* on display.

In [249]:
```
d = {} # defining our container
for list_item in list_items:
    key = list_item.span['title'] # attributes of a node can be fetched with dictionary-like syntax.
    if list_item.a!=None: # check if the node has a hyperlink.
        value = list_item.a['href'] # list_item.a ==list_item.find('a') returns the first node found.
    else:
        value = list_item.text.strip()
    d[key] = value
d
```

Out[249]:
```
{'Announcement': 'https://bitcointalk.org/index.php?topic=428589.0',
 'Chat': 'https://gitter.im/orgs/ethereum/rooms',
 'Explorer': 'https://etherchain.org/',
 'Message Board': 'https://forum.ethereum.org/',
 'Rank': 'Rank 2',
 'Source Code': 'https://github.com/ethereum',
 'Tags': 'Coin\nMineable',
 'Website': 'https://www.ethereum.org/'}
```

In [280]:
```
d['Source Code'],d['Source Code'].split('/')[-1]
```

Out[280]:
```
'ethereum'
```

ERC example

Imagine we wanted to analyze whether the European funding behaviour was biased towards certain countries and gender. We might decide to scrape who has received funding from the ERC.

- **First we figure find navigate the grant listings.**
- **Next we figure out how to page these results.**
- **And finally we want to grab the information.**

In [2]:
```
import requests
url = 'https://erc.europa.eu/projects-figures/erc-funded-projects/results?items_per_page
=100&f[0]=funding_scheme%3AConsolidator%20Grant%20(CoG)'
response = requests.get(url)
```

In [166]:
```
from bs4 import BeautifulSoup
import bs4
soup = BeautifulSoup(response.text,'lxml')
```

You can search for nodes through their Tag, attribute, or

In [22]:
```
result_list = soup.find('div',{'class':'view-content'}) # find elements using tag name a
nd class attribute
```

In [206]:
```
datapoint = {}
for child2 in child.children:
    if type(child2)==bs4.NavigableString:
        continue
    spans = child2.findAll('span')

    if len(spans)==2:
        key,value = spans
    else:
        key,value = [i for i in list(child2.find('span').children) if not type(i)==bs4.N
avigableString]
    datapoint[key.text] = value.text
    # Finding unknown titles using the strong tag as a clue.
    strong = child2.findAll('strong')
    if len(strong)>0:
        for node in strong:
            key = node.text
            value = str(node.next.next)
            print(key,node.next,value)
    #break
```

```
Max ERC Funding Max ERC Funding
1 999 172 €

Duration Duration
Start date: 2014-04-01, End date: 2019-03-31
```

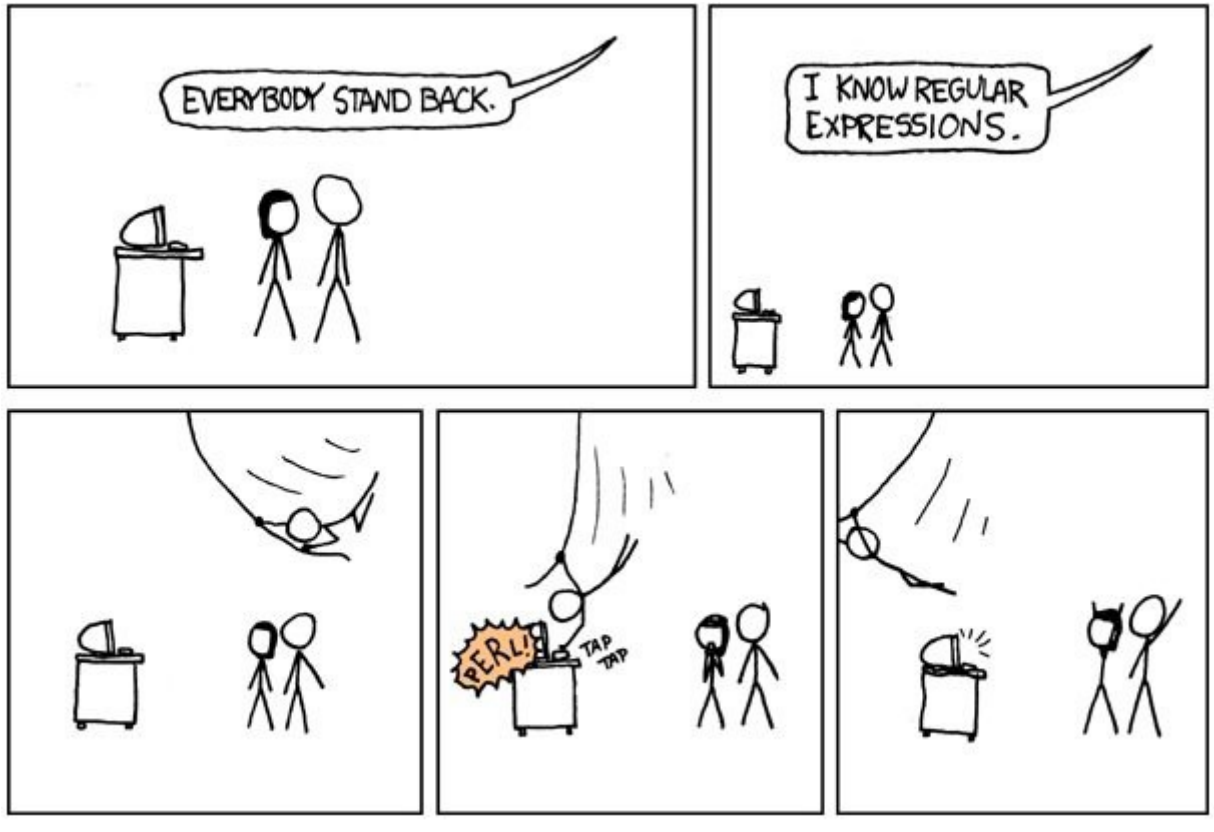
Extracting patterns from Raw Text

of course you already now your basic string operations:

```
string.split
string.strip
string.replace
```

And sometimes this will be enough, but sometimes it is not.

Regular Expressions



Regex can be a little terrifying:

```
pattern = '(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+
(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|
(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:
[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()
<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[
\t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:
(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\]
(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-
\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\
[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000-
\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:
[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)*\<(?:(?:\r\n)?[
\t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=
[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.
(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[
\t])*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:
(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\]
(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-
\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\
[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*)*:(?:(?:\r\n)?[ \t])*)?(?:[^()
```

```
<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()
<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[
\t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:
(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:
(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()
<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()
<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:
(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[
\t])*))*\>(?:(?:\r\n)?[ \t])*)|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:
(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:
(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)*:(?:(?:\r\n)?[ \t])*(?:(?:(?:
[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()
<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[
\t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:
(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:
(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()
<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()
<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:
(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[
\t])*))*|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=
[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:
(?:\r\n)?[ \t])*)*\<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,;:\\".\[\] \000-
\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\
```

```
[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()
<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()
<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*(?:,@(?:
(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[
\t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:
(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\]
(?:(?:\r\n)?[ \t])*))*)*:(?:(?:\r\n)?[ \t])*)?(?:[^()<>@,;:\\".\[\]
\000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:
[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:
(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[
\t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\]
\000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\
[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()
<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()
<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*\>(?:
(?:\r\n)?[ \t])*)(?:,\s*(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:
(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:
(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()
<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()
<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[
\t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:
(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\]
(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-
```

```
\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\
[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000-
\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:
[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)*\<(?:(?:\r\n)?[
\t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=
[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.
(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[
\t])*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:
(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\]
(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-
\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\
[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*)*:(?:(?:\r\n)?[ \t])*)?(?:[^()
<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()
<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[
\t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:
(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:
(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()
<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()
<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:
(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[
\t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[
\t])*))*\>(?:(?:\r\n)?[ \t])*))*)?;\s*)'
```

However creating regular expressions is also fun, fun as in Suduko, and it is extremely valuable when working with any kind of text: e.g. Automating otherwise tedious manual tasks when cleaning a data set, searching, extracting and substituting specific patterns.

Examples could be:

- **Extract currency and amount from raw text: $ 20, 10.000 dollars 10,000 £**
- **email addresses: here you want to design a pattern (as above), that captures only the uses of @ within an email.**
- **urls. Here you are trying to define all the different ways of writing urls (https, http, no http).**
- **Dates. Again many variations: 17th of June 2017, 06/17/17 or 17. June 17**
- **addresses,**
- **phone numbers: 8888888 or 88 88 88 88 or +45 88 88 88 88,**
- **emojiies in text. Capturing all the different ways of expressing smiley faces with one regular expression.**

Regular Expression syntax

Ressources: Best way to learn is to practice, and with interactive examples. Two good ressources are here:

- **Community and interactive playground here (http://regexr.com/)**
- **Interactive tutorial here (https://regexone.com/)**
- **or you can use your notebook.**

Lookup all special characters here (https://www.regular-expressions.info/refquick.html)

- + = 1 or more times -- e.g. "a+" will match: "a", and "aaa"
- * = 0 or more times -- e.g. "ba*" will match: "b", and "ba", and "baaa"
- {3} = exactly three times --- e.g. "ba{3}" will match "baaa", but not "baa"
- ? = once or none
- \ = escape character, used to find characters that has special meaning with regex: e.g. + *
- [] = allows you to define a set of characters
- ^ = applied within a set, it becomes the inverse of the set defined. Applied outside a set it entails the beginning of a string. $ entails the end of a string.
- . = any characters except line break
- | = or statement. -- e.g. a|b means find characters a or b.
- \d = digits
- \D = any-non-digits.
- \s = whitespace-separator

Sequences

Regular expressions (2): define - inspect - refine

You are trying to balance getting (and learning/exploring) all the different variations of e.g. an emojii. while also making sure not to include ordinary use of :.

This means iterating through many steps, some expressions being too broad others being to narrow, and others not matching all that you need.

I developed a small module for this that you can use. Just run the following piece of code to download, save and import the module.

```
# download module
url = 'https://raw.githubusercontent.com/snorreralund/explore_regex/master/explore_rege
x.py'
response = requests.get(url)
# write script to your folder to create a locate module
with open('explore_regex.py','w') as f:
    f.write(response.text)
# import local module
import explore_regex as e_re
```

Lets do an example

first we get a dataset to play with. We download the following link using pandas and dump it to your local machine using the `pd.to_csv()` method.

In [39]:

```
import pandas as pd
import re
path2data = 'https://raw.githubusercontent.com/snorreralund/scraping_seminar/master/dani
sh_review_sample.csv'
df = pd.read_csv(path2data) # get the data
df.to_csv('danish_review_sample.csv',index=False) # save the data

digit_re = re.compile('[0-9]+') # compiled regular expression for matching digits
df['hasNumber'] = df.reviewBody.apply(lambda x: len(digit_re.findall(x))>0) # check if i
t has a number
```

In [41]:
```
sample_string = '\n'.join(df[df.hasNumber].sample(2000).reviewBody)
#sample_string[0:2000]
```

In [42]:
```
import explore_regex as e_re
%matplotlib inline
```

In [57]:
```
# money example
#explore_money = e_re.ExploreRegex(sample_string)
pattern = '[0-9]+(?:[.,][0-9]+)? ?kr'
#pattern = 'kr'
#explore_money.explore_pattern(pattern)

first = 'kr'
second = '[0-9]+kr'
third = '[0-9]+(?:[,.][0-9]+)?kr'
fourth = '[0-9]+(?:[,.][0-9]+)?\s{0,2}kr'
final = '[0-9]+(?:[,.][0-9]+)?\s{0,5}kr(?:oner)?'
patterns = [first,second,third,fourth,final]
for pattern in patterns:
    explore_money.explore_difference(pattern,patterns[0])
explore_money.explore_pattern(second)
explore_money.patterns
```

```
------ Pattern: kr       Matched 1071 patterns -----
Found 0 overlaps between the expressions:
        pattern1: kr     and
        pattern2: kr
        1071 included in pattern1 and not in the pattern2
        1071 was included in pattern2 and not in pattern1
------ Pattern: [0-9]+kr         Matched 83 patterns -----
Found 166 overlaps between the expressions:
        pattern1: [0-9]+kr       and
        pattern2: kr
        0 included in pattern1 and not in the pattern2
        988 was included in pattern2 and not in pattern1
------ Pattern: [0-9]+(?:[,.][0-9]+)?kr  Matched 83 patterns -----
Found 166 overlaps between the expressions:
        pattern1: [0-9]+(?:[,.][0-9]+)?kr        and
        pattern2: kr
        0 included in pattern1 and not in the pattern2
        988 was included in pattern2 and not in pattern1
------ Pattern: [0-9]+(?:[,.][0-9]+)?\s{0,2}kr   Matched 339 patterns -----
Found 678 overlaps between the expressions:
```

In [6]:
```
explore_money.report('hard')
```

```
------ Pattern: kr       Matched 1071 patterns -----
------ Pattern: [0-9]+kr         Matched 83 patterns -----
------ Pattern: [0-9]+(?:[,.][0-9]+)?kr  Matched 83 patterns -----
------ Pattern: [0-9]+(?:[,.][0-9]+)?\s{0,2}kr   Matched 343 patterns -----
------ Pattern: [0-9]+(?:[,.][0-9]+)?\s{0,5}kr(?:oner)?  Matched 343 patterns -----
```

In [7]:
```
print(explore_money.patterns[3],explore_money.patterns[4])
explore_money.explore_difference(explore_money.patterns[3],explore_money.patterns[4])
```

```
[0-9]+(?:[,.][0-9]+)?\s{0,2}kr [0-9]+(?:[,.][0-9]+)?\s{0,5}kr(?:oner)?
Found 686 overlaps between the expressions:
        pattern1: [0-9]+(?:[,.][0-9]+)?\s{0,2}kr         and
        pattern2: [0-9]+(?:[,.][0-9]+)?\s{0,5}kr(?:oner)?
        0 included in pattern1 and not in the pattern2
        0 was included in pattern2 and not in pattern1
```

Out[7]:
```
([], [])
```