Regular expressions

This is a formalism, a set of rules on how to search for patterns of letters in text

It can be used in find and replace operations, in string splitting, etc

Regular expressions were developed by Stephen Kleene in 1951. Kleene was a mathematician, and he came up with a systematic approach to searching text that is still in wide use. This is a domain specific language (in modern terms) that is used within many other systems, including R, Python and SQL.

The MS office tools have a more limited approach to text searches, regex is more powerful, but a bit harder to learn.

Many tools for using regex are built into r strings, but the re package is also available

In [1]:
```
import re
import pandas as pd
```

Search

The first tool in re we will look at is search

This searches a string for a pattern

It returns a match object

In [2]:
```
a="Hey, that cat just stole my hamburger!"

res1=re.search("cat",a)

print(res1)

if(res1!=None):
    print(res1.span())
    print(res1.start())
    print(res1.end())
    print(res1.group())
```

```
<re.Match object; span=(10, 13), match='cat'>
(10, 13)
10
13
cat
```

In [3]:
```
res2=re.search("dog",a)

print(res2)
```

```
if(res2==None):
    print("Ah, no, a dog is too loyal to steal your burger!")
```

```
None
Ah, no, a dog is too loyal to steal your burger!
```

The re.search gives us info on whether or not the pattern was found and if so where it was found

In [4]:
```
# findall returns all matches
# the pattern "sp.. " means the letters sp followed by any two letters,  indicated
# any letter

b="spam, spam, spam, spam, beautiful spam"

re.findall("sp..", b)
```

Out[4]:
```
['spam', 'spam', 'spam', 'spam', 'spam']
```

We can make replacements with the re.sub() function

we send in a search target ("sp") and a replacement ("cr")

In [5]:
```
re.sub("sp","dre",b)
```

Out[5]:
```
'dream, dream, dream, dream, beautiful dream'
```

Splitting a string

this refers to splitting a string into pieces based on some delimiter, like a space, or comma or colon

In [6]:
```
re.split(",",b)
```

Out[6]:
```
['spam', ' spam', ' spam', ' spam', ' beautiful spam']
```

In [7]:
```
re.split(" ",a)
```

Out[7]:
```
['Hey,', 'that', 'cat', 'just', 'stole', 'my', 'hamburger!']
```

Using regex with pandas

In [8]:
```
spam_df=pd.DataFrame( {"Menu" : ["spam and eggs","spam, spam, spam and eggs","spam,
```

In [9]:
```
spam_df
```

Out[9]:

	Menu	prices
0	spam and eggs	2.00
1	spam, spam, spam and eggs	2.50
2	spam, spam, spam	1.75
3	spam,eggs,spam, bacon and spam	3.25

In [10]:
```
spam_df.head()
```

Out[10]:

	Menu	prices
0	spam and eggs	2.00
1	spam, spam, spam and eggs	2.50
2	spam, spam, spam	1.75
3	spam,eggs,spam, bacon and spam	3.25

We can now use the str.find operation on the column spam_df.Menu

The return value is a list.

For each entry in the df, a negative one means the value was not found, otherwise the value is the first location the pattern was found at

In [11]:
```
temp=spam_df.Menu.str.find("eggs")
temp
```

Out[11]:
```
0     9
1    21
2    -1
3     5
Name: Menu, dtype: int64
```

In [12]:
```
temp=spam_df.Menu.str.find("spam")
temp
```

Out[12]:
```
0    0
1    0
2    0
3    0
Name: Menu, dtype: int64
```

In [13]:
```
# the replace function should work normally

spam_df.Menu.str.replace("spam","waffles")
```

Out[13]:
```
0                          waffles and eggs
1        waffles, waffles, waffles and eggs
2                 waffles, waffles, waffles
3    waffles,eggs,waffles, bacon and waffles
Name: Menu, dtype: object
```

In [14]:
```
#here is the string split

temp=spam_df.Menu.str.split(",")
temp
```

Out[14]:
```
0                       [spam and eggs]
1         [spam,  spam,  spam and eggs]
2                  [spam,  spam,  spam]
3    [spam, eggs, spam,  bacon and spam]
Name: Menu, dtype: object
```

In [15]:
```
temp[1]
```

Out[15]:
```
['spam', ' spam', ' spam and eggs']
```

In [16]:
```
temp[1][2]
```

Out[16]:
```
' spam and eggs'
```

the str.split on a pandas column returns a list of lists of strings, so each item returned at locations 0-3 is itself a list of variable length, the contents of which are strings. Takes a bit of thought

we can specify n=1, meaning only the first item in the split

In [17]:
```
temp=spam_df.Menu.str.split(",",n=1)
temp
```

Out[17]:
```
0                     [spam and eggs]
1         [spam,  spam, spam and eggs]
2                  [spam,  spam, spam]
3    [spam, eggs,spam, bacon and spam]
Name: Menu, dtype: object
```

In [18]:
```
temp=spam_df.Menu.str.split(",",n=2)
temp
```

Out[18]:
```
0                       [spam and eggs]
1         [spam,  spam,  spam and eggs]
2                  [spam,  spam,  spam]
3    [spam, eggs, spam, bacon and spam]
Name: Menu, dtype: object
```

In [19]:
```
# Regex options

. a wild card, meaning any symbol
* a wild card, meaning any number of symbols
```

```
(a|b)   the Symbol a or the symbol b

a{3}- the value a three times in a row
a{2,3} - the value a two or three times in a row
a+     - one or more a

^a   an a at the start of the string
a$   an a at the end of the string

[0-9]    numbers in the range 0 to 9
[a-z]    letters in the range a to z
[A-Z]    letters in the range A to Z
```

```
  Cell In[19], line 3
    . a wild card, meaning any symbol
    ^
SyntaxError: invalid syntax
```

In [20]:
```
test_1="I got your number written on the back of my hand 1-800-121-3456"

res_1=re.search("[0-9]{3}",test_1)

print(res_1)
```

```
<re.Match object; span=(51, 54), match='800'>
```

In [ ]: