Regular expressions: Beginnings

6/27/2025, 7/29/2025, HD Sheets

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*, *counting instances of words in a document* 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

Regex is a *formalism* for specifying what pattern of letters and symbols you are looking for.

You also need a software tool to carry out the actual search or string manipulation. Tools to work with regex are available in many languages and software packages so once you have learned regex, you will be able to use it in many different situations when processing text.

The basic package for working with Regex in Pyton is re (regular expressions).

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

Normal Characters and Escape Characters

We have the ordinary letters, uppper and lower case, plus the digits, and these are written as ordinary text

Punctuation in regex has to be written as *Escape characters*:

```
$()*+.?{\^[
```

all have to be written as & ( ) * + . etc

There are some other common escape sequences as well

\n - newline \t- tab

and we'll see more

Search

The first tool in re we will look at is search

This searches a string for a pattern

It returns a match object that tells us if the targeted text was found, and if so the start and end position

In [3]:
```
a="Hey, that cat just stole my hamburger!"


# we want to locate the word cat in this sentence
# the regex search pattern is just the word cat,  which is the simplest poss

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

# res1 has the results of the search

print(res1)

if(res1!=None):

    # span is the start and stop points in the sentence
    print(res1.span())
    print(res1.start())
    print(res1.end())

    # group shows us what was found,  this seems obvious here,  we'll see wh
    print(res1.group())

#knowing the word is between locations 10 and 13, we can extract it

a[10:13]
```

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

Out[3]: 'cat'

In [4]:
```
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!
```

In [5]:
```
# Here is a slight nuance in how we have formed the search

# instead of searching for "cat", we search for "c.t", where "." stands for
# so the search pattern c.t will search for a c followed by any letter follo

res1=re.search("c.t",a)

#since we searched for c.t, we might get cat or cot, since both are english

print(res1)
```

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

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

Wildcards

The period . in a regex target string detects any single symbol

The question mark detects any number of any symbol

In [6]:
```
# findall returns all matches
# the pattern "sp.. " means the letters sp followed by any two letters,  inc
# any letter

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

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

Out[6]:
```
['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 [7]:
```
re.sub("sp","dre",b)
```

Out[7]:
```
'dream, dream, dream, dream, beautiful dream'
```

Question/Action

Write a regex target pattern that will find all the word dream in the sentence below

In [28]:
```
s="Dan dreamt of a dozen dreams"

#use find to locate dream
re.findall("dre..?", s)
```

Out[28]: ['dream', 'dream']

Question/Action

For the same sentence, use a search pattern with a . in it to find dreamt

In [29]:
```
s="Dan dreamt of a dozen dreams"
re.search("dre...", s)
```

Out[29]: <re.Match object; span=(4, 10), match='dreamt'>

Question/Action

For the same sentence, use a search pattern with a ? in it to find Dan

In [37]:
```
s="Dan dreamt of a dozen dreams"
re.search("Da?n", s)
```

Out[37]: <re.Match object; span=(0, 3), match='Dan'>

Splitting a string

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

the splitting value "," is the first input, the string b is the second input

In [38]:
```
re.split(",",b)
```

Out[38]: ['spam', ' spam', ' spam', ' spam', ' beautiful spam']

In [39]:
```
re.split(" ",a)
```

Out[39]: ['Hey,', 'that', 'cat', 'just', 'stole', 'my', 'hamburger!']

Using Search functions in Pandas

Sadly, Pandas does not have full regex capabilities

In [40]:
```
spam_df=pd.DataFrame( {"Menu" : ["spam and eggs","spam, spam, spam and eggs"
```

In [41]:
```
spam_df
```

Out[41]:

	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 [42]:
```
spam_df.head()
```

Out[42]:

	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 [43]:
```
temp=spam_df.Menu.str.find("eggs")
temp
```

Out[43]:
```
0     9
1    21
2    -1
3     5
Name: Menu, dtype: int64
```

Annoyingly, Pandas doesn't implement all of Regex in the str.find operation

It won't use the regex wildcards . or ?

In [44]:
```
temp=spam_df.Menu.str.find("sp.m")
temp
```

Out[44]:
```
0   -1
1   -1
2   -1
3   -1
Name: Menu, dtype: int64
```

In [45]:
```
# the replace function should work normally

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

Out[45]:
```
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
```

Work-around for applying Regex based searches to a Pandas data frame

The idea is to

a.) extract the pandas column to a list of strings b.) apply the re match function to the list, this will return True and False values which may then be used to index the data frame

In [46]:
```
menu_list=spam_df["Menu"]

# just using the regex form e.gs that does not work in pandas

match_values=menu_list.str.contains("e.gs")
match_values
```

Out[46]:
```
0     True
1     True
2    False
3     True
Name: Menu, dtype: bool
```

In [47]:
```
# We can now use this list of which lines contain e.gs to filter the data fr

spam_df[match_values]
```

Out[47]:

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

Question/Action

Use this "contains" tactic to find all the items on the list whose name contains a comma

Show your code below

In [55]:
```
temp = menu_list.str.contains(",")
spam_df[temp]
```

Out[55]:

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

In [56]:
```
# here is the string split operation used on a Pandas column

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

Out[56]:
```
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 [57]:
```
temp[1]
```

Out[57]:
```
['spam', ' spam', ' spam and eggs']
```

In [58]:
```
# the output here is a list of lists,  which looks a bit confusing
temp[1][2]
```

Out[58]:
```
' 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

Question/Action

write code that accesses the split apart word eggs on the last line

In [59]:
```
temp[3][1]
```

Out[59]:
```
'eggs'
```

Controlling the number of splits

We can specify how many splits we want

n=1 gives use one split, so two strings are created, one before the delimiter (,) and one after

In [60]:
```
temp=spam_df.Menu.str.split(",",n=1)
temp
```

Out[60]:
```
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 [61]:
```
# notice that n=1

temp[2]
```

Out[61]:
```
['spam', ' spam, spam']
```

In [62]:
```
temp=spam_df.Menu.str.split(",",n=2)
temp
```

Out[62]:
```
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
```

Regex options

We can create more complex search structures using more advanced regex options

Wild cards

. a wild card, meaning any symbol

- ? a wild card, meaning any number of symbols

The OR option

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

Repeated symbols

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

Symbols at the beginning or end of a line

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

Ranges of symbols

[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

Regex for a digit

\d - one digit \d+ -one or more digits

In [63]:
```
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 [64]:
```
#alternate version

# this is reporting a warning here, kind of oddly,   not all implementations

test_1="I got your number written on the back of my hand 1-800-121-3456"

res_1=re.search("\d{3}",test_1)

print(res_1)
```

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

Question/Action

Alter the code above to find the 4 digit code at the end of the phone number

In [68]:
```
test_2="I got your number written on the back of my hand 1-800-121-3456"

res_2=re.search("\d{4}$",test_2)
res_3 = re.search("[0-9]{4}$",test_2)

print(res_2)
print(res_3)
```

```
<re.Match object; span=(59, 63), match='3456'>
<re.Match object; span=(59, 63), match='3456'>
```

In [71]:
```
#Example- extracting parts of a phone number from a data frame


# taken from https://saturncloud.io/blog/how-to-apply-regex-to-a-pandas-data

# set up the example data base
df = pd.DataFrame({'phone': ['(123) 456-7890', '(456) 789-0123', '(789) 012-
```

```
#extract the phone column to manipulate and process
phone_column = df['phone']
# The r in front a string is indicating is is a regex target

# the re string function "extract" is used here to remove the targetted text

area_code = phone_column.str.extract(r'\((\d{3})\)')
last_four_digits = phone_column.str.extract(r'-(\d{4})')


# with the extraction completed put the data back into the dataframe

df['area_code'] = area_code
df['last_four_digits'] = last_four_digits

print(df)
```

```
           phone area_code last_four_digits
0  (123) 456-7890       123             7890
1  (456) 789-0123       456             0123
2  (789) 012-3456       789             3456
```

Question/Action

Figure out how to extract the middle three digits of the phone number using regex, write the code for it below

In [74]:
```
# extract the middle 3 digits
mid_three = phone_column.str.extract(r'\) (\d{3})-')

# put data back into the dataframe
df["mid_three"] = mid_three

print(df)
```

```
           phone area_code last_four_digits mid_three
0  (123) 456-7890       123             7890       456
1  (456) 789-0123       456             0123       789
2  (789) 012-3456       789             3456       012
```

In [ ]:
```

```