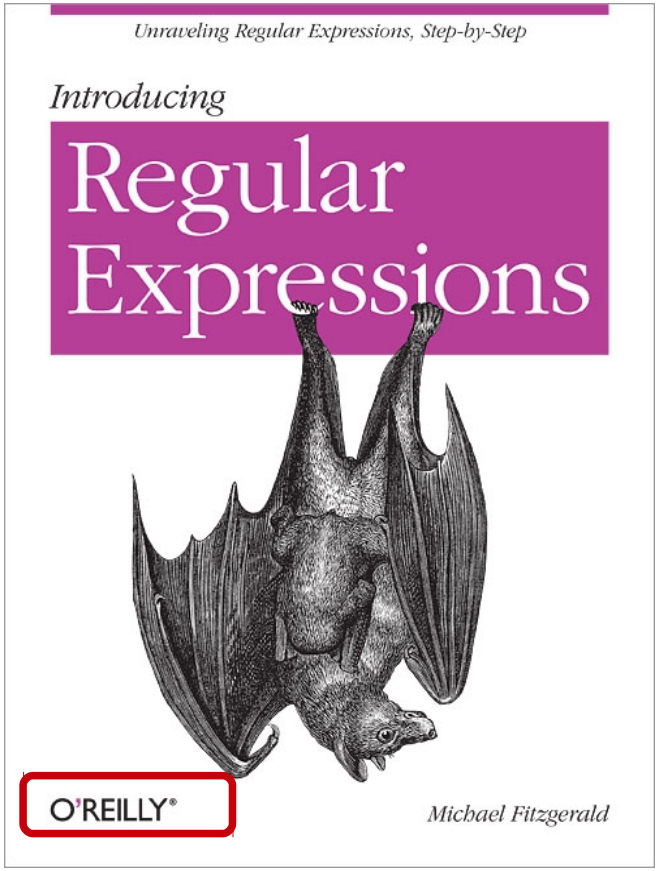
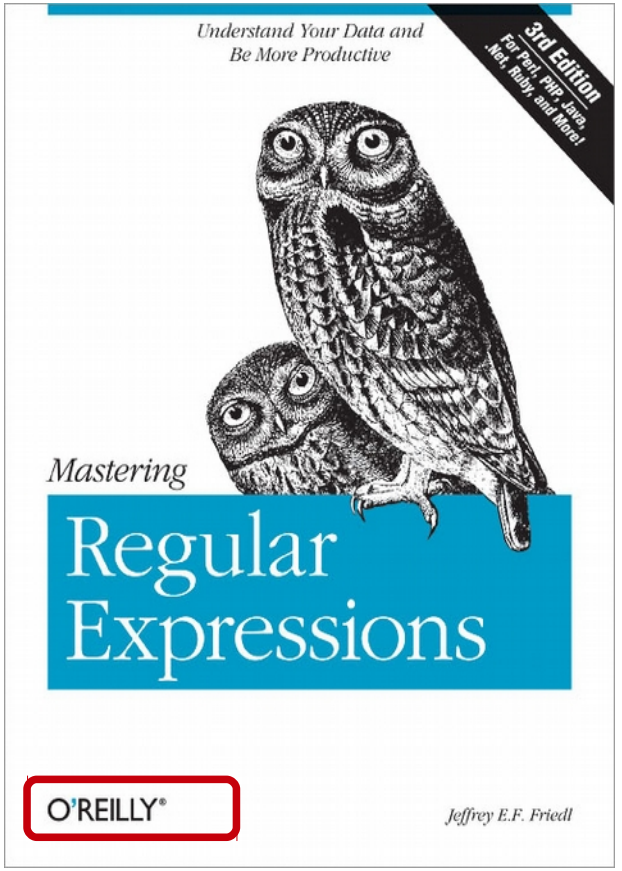
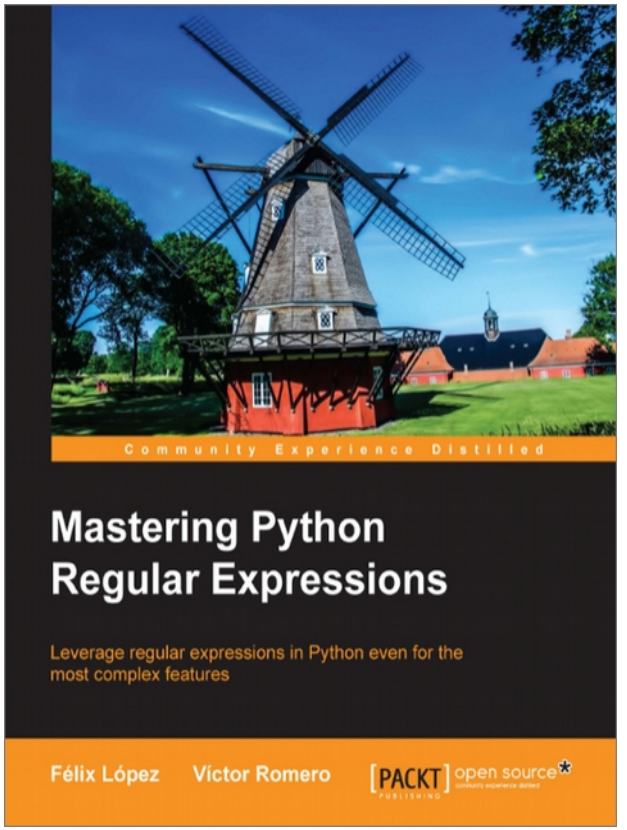
Regular Expressions

Regular expressions (or patterns) is an advanced topic (there are entire books devoted to this topic!), but an important one in bioinformatics (and a handy aid in routine text searching). Thus, we will, at least introduce it. Note thought that we'll barely scratch its surface....







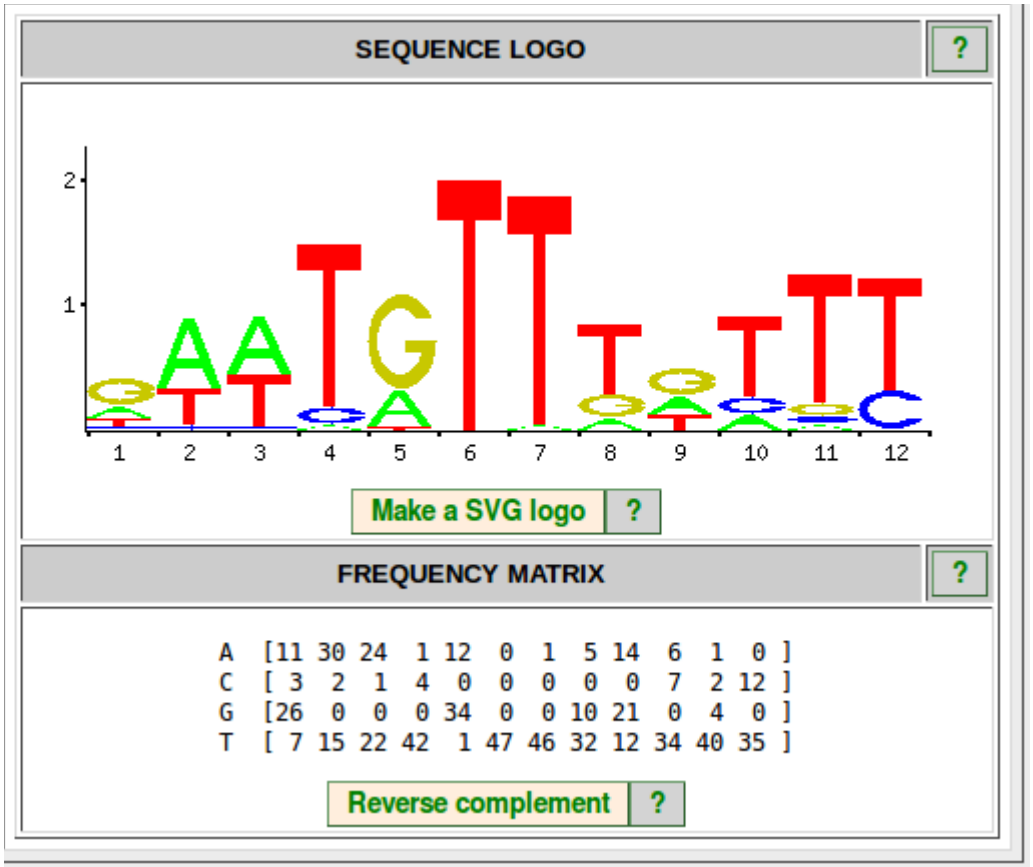
Regular Expressions: finding patterns in text strings

It is easy to find exact substrings in a text, but how do you find non-exact matches?
For example it is easy to find the word “superman” in a text (e.g. using *find* and/or *in*)
But how would you find all the words in a text with the prefix “super-” and ending in “r” (i.e you want to find superpower, superior, and superuser, but not superheroe, supernatural,...)

A more biologically-motivated example:
By now it should be quite easy for you to identify EcoRI sites (“GAATTC”), but what about Foxd3 sites?

[AG]-[AT]-[AT]-T-[AG]-T-T-[TG]-{C}-T-T-T

You might be thinking about loops...forget it!



Regular Expressions: definition

A regular expression is just a string in which certain characters (the so-called *wildcards* or *metacharacters*) have a special meaning and can match more than one particular character in the text being searched. This, in turn, enables a single RE to match many different specific strings.
Regular expression are widely used in bioinformatics and many programming languages implement them (but be aware that the metacharacters and syntax vary between languages !!!).

Example of wildcard character (in a terminal):

```
$ ls
3L8Z.pdb
Unit4_CountMethod_v3.py
aa_frequencies.csv
Unit4_CountMethod_v4.py
aa_frequencies_v2.csv
Unit4_CountMethod.zip
$ ls *.py
Unit4_CountMethod_v3.py
Unit4_CountMethod_v4.py
```

Wildcard	meaning
\w	Any letter or number
\d	Any digit (0-9)
.	Any chracter[1]
\s	Any white space[2]
\t	tab
\n	end-of-line

[1] except end-of-line [2] includes tab and end-of-line

Open file Proteins.fasta with Jedit and search for (use “find” and make sure that you check “Regular expressions” option):

- The motif L-X-X-L-A-P
- Swissprot Accession numbers.

Swissprot Accession is a 6-character code with the format:

1	2	3	4	5	6
[A-N,R-Z]	[0-9]	[A-Z]	[A-Z, 0-9]	[A-Z, 0-9]	[0-9]
[O,P,Q]	[0-9]	[A-Z, 0-9]	[A-Z, 0-9]	[A-Z, 0-9]	[0-9]

http://web.expasy.org/docs/userman.html#AC_line

Regular Expressions: more elements in RE

Special char.	meaning
[AbC1]	Any of the characters within the brackets (A,b,C or 1)
[^AbC1]	Any character except those within the brackets (A,b,C or1)
\	Do not use the “normal” meaning of the following character

Character	Meaning
+	one or more matches of the previous character
*	zero or more matches of the previous character
?	zero or one matches of the previous character
{3}	Exactly 3 matches of the previous character
{3,8}	Between 3 and 8 matches of the previous character
{3,}	At least 3 matches of the previous character

Open file Proteins.fasta with Jedit and search for
(use “find” and make sure that you check “Regular expressions” option):

- Swissprot Accession numbers

Swissprot Accession is a 6-character code with the format:

1	2	3	4	5	6
[A-N,R-Z]	[0-9]	[A-Z]	[A-Z, 0-9]	[A-Z, 0-9]	[0-9]
[O,P,Q]	[0-9]	[A-Z, 0-9]	[A-Z, 0-9]	[A-Z, 0-9]	[0-9]

http://web.expasy.org/docs/userman.html#AC_line

Regular Expressions in Python: the *re* module (I)

1 Import the regular expression module

```
Import re
```

This tells python that the string is a *raw string* and should be taken as such ("*Don't process special sequences in this string*")

2 Define the regular expression you'd like to use

```
MyRE = r'AnyRegularExpression'
```

Variable that contains an object representation of the regular expression (RegexObject)

3 Compile the regular expression

```
MyRegex=re.compile(MyRE)
```

Function of the *re* module that transforms a string into a pattern-type object.

Variable that contains a list of all matches of MyRegex in *AnyString*

4 Use the RegexObject to, for example, find occurences of regex

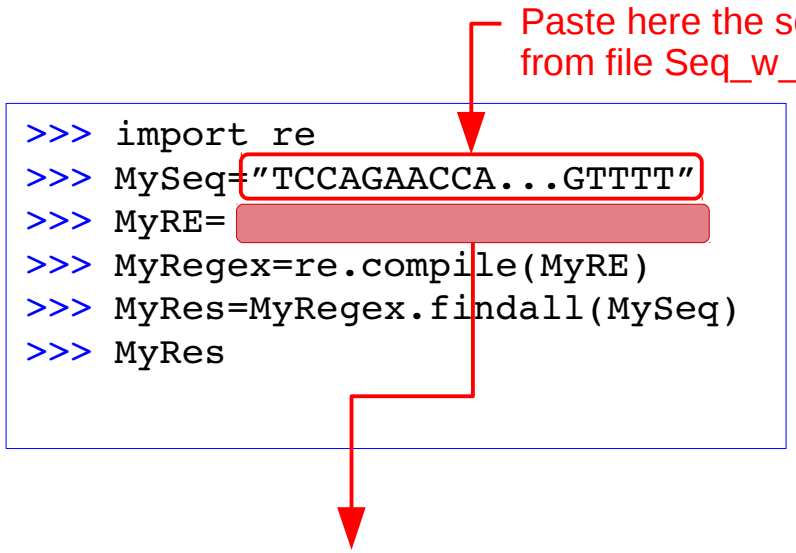
```
Result=MyRegex.findall (AnyString)
```

String where we want to search for the regex.

Method associated to pattern-objects that allows to search for that pattern in the string provided as argument. Returns a list with all matches of MyRegex in *AnyString*

Regular Expressions in Python: the *re* module (I). Example

Back to our original problem, How do we find Foxd3 sites in a sequence?
From the course's Moodle page, open the document Seq_w_Foxd3.fasta and assign its content to a variable using a pyhton interactive session. The find all occurrences of the Foxd3 binding sites whose pattern is:
[AG]-[AT]-[AT]-T-[AG]-T-T-[TG]-{C}-T-T-T (Please note that this is the PROSITE syntax!!!)




```
>>> import re
>>> MySeq="TCCAGAACCA...GTTTT"
>>> MyRE=r'[AG][AT][AT]T[AG]TT[TG][^C]TTT'
>>> MyRegex=re.compile(MyRE)
>>> MyRes=MyRegex.findall(MySeq)
>>> MyRes
['GTTTGTTTGTTT', 'GAATGTTTGTTT', 'GTTTGTTGTTTT']
>>> MyRE=r'[AG][AT]{2}T[AG]T{2}[TG][^C]T{3}'
>>> MyRegex=re.compile(MyRE)
>>> MyRes=MyRegex.findall(MySeq)
>>> MyRes
['GTTTGTTTGTTT', 'GAATGTTTGTTT', 'GTTTGTTGTTTT']
```

However, many times we'll want the location of the hit. How do we get it?

Regular Expressions in Python: the *re* module(II)

1. Import the regular expression module

```
Import re
```

2. Define the regular expression you'd like to use

```
MyRE = r 'AnyRegularExpression'
```

3. Compile the regular expression

```
MyRegex=re.compile(MyRE)
```

4. Use the RegexObject in, for example, searches

Variable that contains an object representation of the search result (MatchObject)

String to be searched. OPTIONAL: start and end pos

```
Result=MyRegex.search (AnyString,start, end)
```

Method associated to pattern-objects that allows to search for that pattern in the string provided as argument.
Returns an **object of the type MatchObject that contains** only the first hit in the string after start and before end

5. Access info to the match

`Result.group()`	Returns the match.
`Result.start()`	Start position of the hit
`Result.end()`	end position of the hit

Regular Expressions in Python: the *re* module (II). Example

Let's now find all Foxd3 sites in Seq_w_Foxd3.fasta using search()

```
>>> import re
>>> MySeq="TCCAGAACCA...GTTTT"
>>> MyRE=r'[AG][AT][AT]T[AG]TT[TG]
[^C]TTT'
>>> MyRegex=re.compile(MyRE)
>>> MyRes2=MyRegex.search(MySeq)
>>> MyRes2
>>> MyRes2.group()
>>> MyRes2.start()
>>> MyRes2.end()
>>> MyRes2=MyRegex.search(MySeq,608)
>>> MyRes2.start()
>>> MyRes2.end()
```

```
>>> import re
>>> MySeq="TCCAGAACCA...GTTTT"
>>> MyRE=r'[AG][AT][AT]T[AG]TT[TG][^C]TTT'
>>> MyRegex=re.compile(MyRE)
>>> MyRes2=MyRegex.search(MySeq)
<_sre.SRE_Match object at 0x7f565f3139f0>
>>> MyRes2.group()
'GTTTGTTTGTTT'
>>> MyRes2.start()
596
>>> MyRes2.end()
608
>>> MyRes2=MyRegex.search(MySeq,608)
'GAATGTTTGTTT'
>>> MyRes2.start()
1024
>>> MyRes2.end()
1036
```

Regular Expressions in Python: the *re* module (II). Recursive example

The method search() is very convenient when we need the start and end position of each regex match, however it does not return all the matches, only the first one within the provided indices.
We can write a short code to work around this limitation. For example the following code makes use of a ***recursive function*** to identify all sites in a sequence using search()

```
>>> import re
>>> MySeq=”TCCAGAACCA...GTTTT”
>>> MyRE=r'[AG][AT][AT]T[AG]TT[TG][^C]TTT'
>>> MyRegex=re.compile(MyRE)
>>> def SearchAll(Regex,Seq,pos):
...    Res=Regex.search(Seq,pos)
...    if Res==None:
...       return()
...    else:
...       print(Res.group(),"\t",Res.start(),"\t",Res.end())
...       SearchAll(Regex,Seq,Res.start()+1)
>>> SearchAll(MyRegex,MySeq,0)
GTTTGTTTGTTT      596      608
GAATGTTTGTTT      1024     1036
GTTTGTTGTTTT      1364     1376
```

As we mentioned before recursivity frequently used. Make sure you understant how does the above code work.

Regular Expressions in Python: the *re* module (II). Recursive example

The method search() is very convenient when we need the start and end position of each regex match, however it does not return all the matches, only the first one within the provided indices.
We can write a short code to work around this limitation. For example the following code makes use of a ***recursive function*** to identify all sites in a sequence using search()

```
>>> import re
>>> MySeq=”TCCAGAACCA...GTTTT”
>>> MyRE=r'[AG][AT][AT]T[AG]TT[TG][^C]TTT'
>>> MyRegex=re.compile(MyRE)
>>> def SearchAll(Regex,Seq,pos):
...    Res=Regex.search(Seq,pos)
...    if Res==None:
...       return()
...    else:
...       print(Res.group(),"\t",Res.start(),"\t",Res.end())
...       SearchAll(Regex,Seq,Res.start()+1)
>>> SearchAll(MyRegex,MySeq,0)
GTTTGTTTGTTT      596      608
GAATGTTTGTTT      1024     1036
GTTTGTTGTTTT      1364     1376
```

As we mentioned before recursivity frequently used. Make sure you understant how does the above code work.

Regular Expressions in Python: groupings

Many times, we would like to find a match for the regular expression and then extract some elements from the match. This is done by using brackets around those elements of the regular expression that we want to extract. The set of elements within the brackets is called a ***group***.

Example: To match a protein sequence in fasta format and extract the identifier *and* the sequence. Open the file vRasFam.fasta and, working in a python interactive shell, asign this text to a variable (for example *Sequences)*. NOTE: use three single quotation marks to allow for multiple lines. Then do the following:

Paste here the sequence from file vRasFam.fasta

```
>>> import re
>>> Sequences=''>sp|P01115|RAS....CVLS''
>>> MyRE2=r'>(.+)\n([^BJUXZ]+)\n'
>>> MyReg2=re.compile(MyRE2)
>>> MyRes3=MyReg2.search(Sequences)
>>> MyRes3.groups()
('sp|P01115|RASH_MSVHA Transforming....CKCVLS')
```

Note the use of three quotation marks to allow text spand across several lines

Note the use of group**S**() to get all groups.

We get a tupla (a tupla is quite similar to a list) with all the groups

Example2: This is a tip for you to solve the Programming Problem for this week:

```
>>> import re
>>> Line='LUIS\tPESO\t915854440\n'
>>> print(Line)
>>> MyRE3=r'([A-Z]+)\t([A-Z]+)\t([0-9]+)\n'
>>> MyReg3=re.compile(MyRE3)
>>> MyRes4=MyReg3.search(Line)
>>> MyRes4.groups()
('LUIS', 'PESO', '915854440')
>>> MyRes4.groups()[2]
'915854440'
```