Lecture 9

Regular Expressions

Prof. Steven Ludtke
N410.07, sludtke@bcm.edu

e-coli

- Find possible coding proteins from an e-coli plasmid
- Shine-Dalgarno consensus sequence (AGGAGG)
- Start (within 3-10 residues):
  - 83% ATG (3542/4284)
  - 14% GTG (612)
  - 3% TTG (103)
- Stop: TGA, TAA, TAG

Example

- Write a program to extract potential protein coding regions from the e-coli genome

With Strings

```
seq=open("ecoli.k12.txt","r").read()

def myfind(str,substr):
     r=str.find(substr)
     if r<0 : return 9999999999
     return r

curloc=0
while True:
     sdloc=seq[curloc:].find("AGGAGG")
     if sdloc<0 : break

     start=curloc+sdloc+6
     subseq=seq[start:start+12]
     atg=myfind(subseq,"ATG")
     gtg=myfind(subseq,"GTG")
     ttg=myfind(subseq,"TTG")

     if min(atg,gtg,ttg)=="" :
          curloc=start
          continue
     start+=min(atg,gtg,ttg)

     srch=start
     while True:
          subseq=seq[srch:srch+3]
          print(subseq,end="")
          if subseq in ("TGA","TAA","TAG"): break
          srch+=3

     print("\n")
     curloc=srch
```

Regular Expressions

- Language describing "patterns"
- Reasonably standardized across most programming languages
- Often available in applications, eg - search dialogs
- Very useful in bioinformatics, tight integration with PERL one of the reasons popular in that community
- Python is largely PERL compatible with a few extensions
- *import re*

Regular Expressions

- '.' - any character
- [abcd] - match any character in the list, may use '-' or '^'
- '\s' - any whitespace character [\t\n\r\f\v]
- '|' - or, match either of 2 expressions
- (...) - used to group parts of an expression
- (?P<name>...) - a 'named' group (see groupdict)
- '*' - 0 or more repetitions of the preceding element
- '+' - 1 or more repetitions of the preceding element
- '?' - 0 or 1 repetitions of the preceding element
- '*?','+?','??' - non greedy version of *, + and ?
- {m,n} - match m-n copies of previous expression
- '^' - start of the string
- '$' - end of the string
- ..... there are more

Regular Expressions

re functions:

- re.search(pattern,string) - search the entire string for pattern
- re.match(pattern,string) - check the beginning of the string only
- re.split(pattern,string) - much like string.split()
- re.findall(pattern,string) - list of all non-overlapping instances
- re.finditer(pattern,string) - Match object for each match
- re.sub(pattern,repl,string) - replace matches with repl

Regular Expressions

Match objects:

- group(n) - returns the matching part of the string in group n
- groups() - returns a tuple with all subgroups
- groupdict() - returns a dictionary of results based on <> names
- start(),end() - index of start or end of match

e-coli

- Find possible coding proteins from an e-coli plasmid
- Shine-Dalgarno consensus sequence (AGGAGG)
- Start (within 3-10 residues):
  - 83% ATG (3542/4284)
  - 14% GTG (612)
  - 3% TTG (103)
- Stop: TGA, TAA, TAG

With Strings

```
seq=open("ecoli.k12.txt","r").read()

def myfind(str,substr):
    r=str.find(substr)
    if r<0 : return 9999999999
    return r
curloc=0
n=0
while True:
    sdloc=seq[curloc:].find("AGGAGG")
    if sdloc<0 : break
    start=curloc+sdloc+6
    subseq=seq[start:start+12]
    atg=myfind(subseq,"ATG")
    gtg=myfind(subseq,"GTG")
    ttg=myfind(subseq,"TTG")

    if min(atg,gtg,ttg)==9999999999 :
        curloc=start
        continue
    start+=min(atg,gtg,ttg)

    srch=start
    while True:
        subseq=seq[srch:srch+3]
        print(subseq,end="")
        if subseq in ("TGA","TAA","TAG"): break
        srch+=3

    print("\n")
    curloc=srch
    n+=1

print(n," total")
```

Equivalent with Regex

```
import re
seq=open("ecoli.k12.txt","r").read()

pat="(AGGAGG)(.{3,12})(ATG|TTG|GTG)(([CATG]..)+?)(TGA|TAA|TAG)"

matches=re.findall(pat,seq)

for match in matches: print(match[2],match[3],match[-1])
```