BINF2111 – Introduction to Bioinformatics Computing

Regular expressions – the red ‘pill’ of python

Richard Allen White III, PhD
RAW Lab
Lecture 21 – Tuesday Nov 16th, 2021

Learning Objectives

- Regular expressions

- Uses

- Quantifiers

- Grep vs. Python

- The return of the CSV to TSV converter problem

- Quiz 21

How useful are regular expressions?

Regular expressions allow us to search for patterns in strings.

Regular expressions are extremely useful to search through DNA, RNA, and protein sequences.

- protein domains
- DNA transcription factor binding motifs
- restriction enzyme cut sites
- degenerate PCR primer sites
- runs of mononucleotides
- read mapping locations
- geographical sample coordinates
- taxonomic names
- gene names
- gene accession numbers
- BLAST searches

How useful are regular expressions?

Regular expressions allow us to search for patterns in strings.

Regular expressions are extremely useful to search through DNA, RNA, and protein sequences.

- protein domains
- DNA transcription factor binding motifs
- restriction enzyme cut sites
- degenerate PCR primer sites
- runs of mononucleotides
- read mapping locations
- geographical sample coordinates
- taxonomic names
- gene names/gene accession numbers/BLAST searches

How useful are regular expressions?

Searched for patterns before … e.g., startswith(“ATTA”) or matching characters with ==.

However, these relate to a **fixed** set of characters. Regular expressions allow more flexible patterns.

For example:

What's the length of the poly-A tail? Could be any size.

How many copies of a microsatellite are there?

Looking for a sequencing library tag, but that tag might have mutated a single base pair.

Regular expressions

Regular expressions use the backslash character ('\') to indicate special forms or to allow special characters to be used without invoking their special meaning.

For example, we've used regular expressions

\n means start a new line

\t means insert a tab

We can force python to output backslashes and ignore special characters (r = raw) this way:

print (r"\t"n") #will output \t\n

Regular expressions

First off, we have to import re if we want to use regular expressions (python3)

```
#!/usr/bin/python3


import re
```

Regular expressions

```
1 #!/usr/bin/python3
2
3 import re
4
5 dna = "\\g"
6 print(dna)
7 if re.search("\\\g", dna):   #note that you need 3 \
8     print("restriction site found!") #to find pattern without raw string
9 if re.search(r'\\g', dna):
10     print("restriction site found!")
```

Regular expressions

```
1  #!/usr/bin/python3
2
3  import re
4
5  dna = "ATCAGATACTACGTCATACCGCGCGCG"
6  print(dna)
7  if re.search("CGCGCGCG", dna):
8      print("Restriction site found!")
9  else:
10     print("No Restriction site found!")
```

Regular expressions

```
1 #!/usr/bin/python3
2
3 import re
4
5 dna = "ATCAGATACTACGTCATACCGCGCGCG"
6 print(dna)
7 if re.search("X", dna):
8     print("Restriction site found!")
9 else:
10    print("No Restriction site found!")
```

Regular expressions

```
1 #!/usr/bin/python3
2
3 import re
4
5 dna = "TTTTTTTTTT"
6 dna1 = "TTATATAGTCCC"
7
8 print(dna)
9 print(dna1)
10
11 if re.search("GGACC", dna) or re.search(r"GTCCC",dna1):
12     print("Restriction site found!")
13 else:
14     print("No Restriction site found!")
```

Regular expressions

```
1 #!/usr/bin/python3
2
3 import re
4
5 dna = "TTTTTTTTTT"
6 dna1 = "TTATATAGTCCC"
7
8 print(dna) and print(dna1)
9
10 if re.search("GGACC", dna) and re.search(r"GTCCC",dna1):
11     print("Restriction site found!")
12 else:
13     print("No Restriction site found!")
```

Regular expressions (Quantifiers)

```
1 #!/usr/bin/python3
2
3 import re
4
5 dna = "TTTTTTTTTT"
6 dna1 = "TTATATGGACC"
7
8 print(dna) and print(dna1)
9
10 if re.search("GGACC", dna) or re.search(r"GG(A|T)CC",dna1):
11     print("Restriction site found!")
12 else:
13     print("No Restriction site found!")
```

The (X|Y) allows you to search for patterns X or Y when separated by a pipe | character.

Regular expressions (Quantifiers)

```
#!/usr/bin/python3

import re

dna = "ATGATCGCGGGCCATTCAC"

print(dna)

if re.search("GG[ACTG]CC", dna):
    print("Restriction site 1 found!")
else:
    print("Restriction site 1 not found")

if re.search("GG.CC", dna):
    print("Restriction site 2 found!")
else:
    print("Restriction site 2 not found")

if re.search("GG(^YYY)CC", dna):
    print("Restriction site 3 found!")
else:
    print("Restriction site 3 not found")

if re.search("GG[^YYY]CC", dna):
    print("Restriction site 4 found!")
else:
    print("Restriction site 4 not found")

if re.search("^ATG", dna):
    print("Start codon found!")
else:
    print("Start codon not found")
```

```
#!/usr/bin/python3

import re

dna = "GGATGATCGCGGGCCATTCAC"

print(dna)

if re.search("GG[ACTG]CC", dna):
    print("Restriction site 1 found!")
else:
    print("Restriction site 1 not found")

if re.search("GG.CC", dna):
    print("Restriction site 2 found!")
else:
    print("Restriction site 2 not found")

if re.search("GG(^YYY)CC", dna):
    print("Restriction site 3 found!")
else:
    print("Restriction site 3 not found")

if re.search("GG[^YYY]CC", dna):
    print("Restriction site 4 found!")
else:
    print("Restriction site 4 not found")

if re.search("^ATG", dna):
    print("Start codon found!")
else:
    print("Start codon not found")
```

re.search((selective))
re.search([not selective])

^ just like in bash or awk, perl or sed is at the beginning of the string

Regular expressions (Quantifiers)

```
1 #!/usr/bin/python3
2
3 import re
4
5 dna = "ATGATCGCGGGCCATTCAC"
6
7 print(dna)
8
9 if re.search("GG?C", dna):
10     print("Restriction site 1 found!")
11 else:
12     print("Restriction site 1 not found")
13
14 if re.search("^ATG", dna):
15     print("Start codon found!")
16 else:
17     print("Start codon not found")
```

The question mark allows to match 0 or 1 times

Regular expressions (Quantifiers)

```
1 #!/usr/bin/python3
2
3 import re
4
5 dna = "ATGATCGCGGGCCATTCAC"
6
7 print(dna)
8
9 if re.search("GG*C", dna):
10     print("Restriction site 1 found!")
11 else:
12     print("Restriction site 1 not found")
13
14 if re.search("^ATG", dna):
15     print("Start codon found!")
16 else:
17     print("Start codon not found")
```

The * allows to match 0 or more times.

Regular expressions (Quantifiers)

```
1 #!/usr/bin/python3
2
3 import re
4
5 dna = "ATGATCGCGGGCCATTCACTAG"
6
7 print(dna)
8
9 if re.search("TAG$", dna):
10     print("Stop codon found!")
11 else:
12     print("Stop codon not found")
13
14 if re.search("^ATG", dna):
15     print("Start codon found!")
16 else:
17     print("Start codon not found")
```

Just like awk, sed, perl, bash in regex ^ beginning and $ at the end

Special Regular expressions

\d

Matches any decimal digit; this is equivalent to the class [0-9].

\D

Matches any non-digit character; this is equivalent to the class [^0-9].

\s

Matches any whitespace character; this is equivalent to the class [\t\n\r\f\v].

\S

Matches any non-whitespace character; this is equivalent to the class [^ \t\n\r\f\v].

\w

Matches any alphanumeric character; this is equivalent to the class [a-zA-Z0-9_].

\W

Matches any non-alphanumeric character; this is equivalent to the class [^a-zA-Z0-9_].

Special Regular expressions

\d

Matches any decimal digit; this is equivalent to the class [0-9].

If you don’t use raw strings, you will have to denote this as :

```
if re.search('\\d', location2):
```

```
if re.search(r'\d', location2):
```

Use raw strings when doing regular expressions..

Special Regular expressions

We can search for special regular expressions

```
#!/usr/bin/env python
import re

line = "here    is      a       line"

if re.search(r'\t', line):
        print("tab found")
if re.search(r'\s', line):
        print("any space found")
```

```
tab found
any space found
```

Special Regular expressions

More elaborate regex matches can extract patterns that you are interested in.

```
#!/usr/bin/python3

import re

s = "Jose’s ID: Comp 42"



fullid = re.search(r'ID: (.+)', s)
print(fullid.group())
print(fullid.group(1))

idnum = re.search(r'ID: (...)(\d+)', s)
print(idnum.group(2))
```

Regrex Multiple Matches

It is possible to match many regular expressions and put in *list* using the function **findall**.

```
#!/usr/bin/python3

import re

s = "ACGTAGAGACATTAAAATACAAGTAAAA"

temp =  re.findall(r'AAAA', s)
print(temp)

temp2 = re.findall(r'\w', s)
print(temp2)
```

Findall from a file

It is possible to use findall as you read in a file. Recall that read reads in all the text of a file into a single string.

```
#!/usr/bin/env python

import re

Open file
f = open('test.txt', 'r')

matched = re.findall(r'some pattern', f.read())
```

Finditer and matches

Findall only produces a list of your matches. If you want to actually manipulate the matches, you need to use **finditer**. This is often used with a for loop.

This allows more flexibility than findall.

```
#!/usr/bin/env python

import re
s = "ACGTAGAGACATTAAAATACAAGTAAAA"

runs = re.finditer(r"[A]{4}", s)
for match in runs:
      print match
        run_start = match.start()
        run_end = match.end()
        print("A-4mer from " + str(run_start) + " to " + str(run_end))
```

Finditer and matches

```
#!/usr/bin/env python

import re
s = "ACGTAGAGACATTAAAATACAAGTAAAA"

runs = re.finditer(r"[A]{4}", s)
for match in runs:
      print match
        run_start = match.start()
        run_end = match.end()
        print("A-4mer from " + str(run_start) + " to " + str(run_end))
```

Finditer stores the matches into an object at a memory address(just like file headers). You can call back the object using methods (e.g., file.read()).

Grep vs. python - refresher

How do you count the number of sequences in a fasta file with grep?

Grep vs. python - refresher

How do you count the number of sequences in a fasta file with grep?

```
grep '>' one.fasta | wc –l
Or:
grep -c '>' one.fasta
```

Grep vs. python - refresher

How do you count the number of sequences in a fasta file with python?

```
#!/usr/bin/python3

import sys

count = 0
with open(sys.argv[1]) as reader:
    for line in reader:
        if line.startswith('>'):
            count += 1
print(count)
```

Grep vs. python - refresher

How do you count the number of sequences in a fasta file with grep? One liner!

```
#!/usr/bin/python3

import sys

print( len( [ x for x in open(sys.argv[1]) if x.
startswith('>') ] ) )
```

Grep vs. python - refresher

How do I drop all empty lines or ‘all white space’ in grep and python?

- Delete all the empty lines in the empty lines file with
  → grep: grep -v -e '^$' file
  → awk: awk '!/^$/' file

- Delete all the ‘all white space’ with grep
  → grep: grep -v -e '^[[:space:]]*$' file
  → awk: awk 'NF > 0' file

Grep vs. python - refresher

How do I drop all empty lines or ‘all white space’ in grep and python?

```
import pandas as pd

df = pd.read_csv('data.csv')
new_df = df.dropna()  or df.dropna(inplace = True)
print(new_df.to_string()) or print(df.to_string())
```

Give that a go with the empty lines file!

Converting CSV to TSV

With this file (file.csv):
Rat,steven,bear,Xi
Olf,thor,flower,Ton
I WOULD KNOW THREE – Hint, hint!

Command 1: sed ‘s/,/\t/g’ file.csv >file.tsv
Command 2: cat file.csv | tr -s ',', '\t' >file.tsv
Command 3: awk ‘{gsub(",", "\t"); print}’ file.csv > file.tsv
Command 4: perl -pi -e 's/,/\t/g' file.csv > file.tsv
Command 5: cat file.csv | awk -F ',' '{$1=$1}1' >file.tsv

Converting CSV to TSV

- Write a bash script that converts this into a csv file (**specific files** and all files)?

```
1 #!/bin/bash/
2
3 input=$1
4
5 sed 's/,/\t/g' $input
6
7 function print_to_terminal(){
8     echo "Your comma-separated file has been converted to tab-delim file" >$(tty)
9     echo -n "Wise choice!" >$(tty)
10 }
11
12 output=$(print_to_terminal)
13
```

Converting CSV to TSV

- Write a bash script that converts this into a csv file (specific files and **all files**)?

```
#!/bin/bash/

for i in *csv;
do
    sed 's/,/\t/g' "$i" >$(basename "$i" .csv).tsv
done

function print_to_terminal(){
    echo "Your comma-separated file has been converted to tab-delim file" >$(tty)
    echo -n "Wise choice!" >$(tty)
}

output=$(print_to_terminal)
```

Grep vs. python - refresher

How do convert tsv to csv using python pandas?

```
1 #!/usr/bin/python3
2
3 import pandas as pd
4
5 file_path = '/home/docwhite/Desktop/BINF2111/data/name_game.tsv'
6 csv_table=pd.read_table(file_path,sep='\t')
7 csv_table.to_csv('name_game_pandas.csv',index=False)
```

Grep vs. python - refresher

How do convert tsv to csv using python pandas (larger files)?

```
1 #!/usr/bin/python3
2
3 import pandas as pd
4
5 file_path = '/home/docwhite/Desktop/BINF2111/data/name_game.tsv'
6
7 dfs = pd.read_csv(file_path, sep='\t', chunksize=50)
8 for df in dfs:
9     df.to_csv('file.csv', sep=',', mode='a')
```

Grep vs. python - refresher

How do convert csv to tsv using python no pandas (larger files)?

```
1 #!/usr/bin/python3
2
3 import csv
4
5 with open('/home/docwhite/Desktop/BINF2111/data/name_game.csv','r') as csvin, open('NAD.tsv', 'w') as tsvout:
6     csvin = csv.reader(csvin)
7     tsvout = csv.writer(tsvout, delimiter='\t')
8
9     for row in csvin:
10        tsvout.writerow(row)
```

Grep vs. python - refresher

How do convert tsv to csv using python no pandas (larger files)?

```
1 #!/usr/bin/python3
2
3 import io
4 import re
5
6 def convert(file_name):
7     # Open csv file
8     csv = io.open(file_name + ".csv", mode="w", encoding="utf-8")
9
10    # Convert file_name.tsv to file_name.csv
11    with io.open(file_name + ".tsv", mode="r", encoding="utf-8") as tsv:
12        for line in tsv:
13            csv.write(re.sub('\t',',',re.sub('(^|[\t])([^\t]*\,[^\t\n]*)', r'\1"\2"', line)))
14
15    csv.close()
16
17 if __name__ == '__main__':
18    print('Start converting ...')
19    convert('table')
20    print('Converted!')
```

Quiz 21

- On canvas now