Class 23: Using regular expressions to analyze data

April 16, 2019

In this class, we will discuss a few more real-world scenarios of how we can use regular expressions to analyze data. We will work with the *E. coli* genome. As usual, we first download it:

In [1]:
```
from Bio import Entrez
Entrez.email = "wilke@austin.utexas.edu" # put your email here

# Download E. coli K12 genome:
download_handle = Entrez.efetch(db="nucleotide", id="CP009685", rettype="gb", r
etmode="text")
data = download_handle.read()
download_handle.close()

# Store data into file "Ecoli_K12.gb":
out_handle = open("Ecoli_K12.gb", "w")
out_handle.write(data)
out_handle.close()
```

Let's assume that we want to find *E. coli* genes that are enzymes. Enzymes can be identified because their name ends in "ase". The gene name is stored in the "product" feature qualifier of CDS features.

We will write code that loops over all CDS features in the genome, find the protein-coding sequences (CDSs), and analyze their product feature. To analyze the name of the product, we will use the following regular expression: `r"ase($|\s)"`. Remember that the vertical line `|` indicates logical or. So this regular expression searches for two alternative patterns. The first pattern, `r"ase$"` looks for strings that end in `ase`. The second pattern, `r"ase\s"` looks for strings that contain a word ending in `ase`. (Word ends are indicated by subsequent whitespace, which is matched by `\s`.)

Note that we will limit our search to the first 100 protein-coding sequences only, to make the code run more quickly.

In [2]:
```
import re
from Bio import SeqIO

# read in the E. coli genome from local storage
in_handle = open("Ecoli_K12.gb", "r")
record = SeqIO.read(in_handle, "genbank")
in_handle.close()

max_i = 100 # number of protein-coding sequences we will analyze
i = 0 # counter that will keep track of the number of CDSs found
enzyme_count = 0 # number of enzymes found
for feature in record.features:
    if feature.type == 'CDS':
        i += 1

        # we can only proceed if the CDS has a 'product' qualifier
        if "product" in feature.qualifiers:
            product = feature.qualifiers["product"][0]

            # the heart of the matter. does the product string end in 'ase'
            # or contain a word that ends in 'ase'?
            match = re.search(r"ase($|\s)", product)
            if match:
                # yes, we found something that looks like an enzyme
                print(product)
                enzyme_count += 1

    # stop after max_i CDSs have been processed
    if i >= max_i:
        break

print("\nTotal number of probable enzymes found:", enzyme_count)
```

```
cellulose synthase
cellulose synthase
endo-1,4-D-glucanase
cellulose synthase
ketodeoxygluconokinase
ketodeoxygluconokinase
c-di-GMP phosphodiesterase
trehalase
cytochrome C peroxidase
glutamate decarboxylase
transposase
arsenate reductase
glutathione reductase
ribosomal RNA large subunit methyltransferase J
oligopeptidase A
methyltransferase
peptide ABC transporter permease
nickel transporter permease NikC
nickel transporter permease NikB
ACP synthase
permease
zinc ABC transporter ATPase
16S rRNA methyltransferase
RNA polymerase factor sigma-32
branched-chain amino acid transporter permease subunit LivH
leucine/isoleucine/valine transporter permease subunit
glycerol-3-phosphate transporter permease
glycerophosphodiester phosphodiesterase
gamma-glutamyltranspeptidase
transposase
transposase

Total number of probable enzymes found: 31
```

Problems

Problem 1:

Find out if there are any products that contain the letters "ase" in the middle of a word. For example, the word "based" contains these letters but does not end in them.

Hint: Set `max_i=5000` to search the entire genome.

In [3]:
```
import re
from Bio import SeqIO

# read in the E. coli genome from local storage
in_handle = open("Ecoli_K12.gb", "r")
record = SeqIO.read(in_handle, "genbank")
in_handle.close()

max_i = 5000 # search the entire genome
i = 0 # counter that will keep track of the number of CDSs found
for feature in record.features:
    if feature.type == 'CDS':
        i += 1

        # we can only proceed if the CDS has a 'product' qualifier
        if "product" in feature.qualifiers:
            product = feature.qualifiers["product"][0]

            # The heart of the matter. Does the product have 'ase'
            # in the middle? The '.+' on either side assures that
            # 'ase' is neither at the beginning nor at the end.
            match = re.search(r"\S+ase\S+", product)
            if match:
                # yes, we found a match
                print(product)

    # stop after max_i CDSs have been processed
    if i >= max_i:
        break
```

```
polynucleotide phosphorylase/polyadenylase
bifunctional glutamine-synthetase adenylyltransferase/deadenyltransferase
flap endonuclease-like protein
hydrogenase-4 component G
hydrogenase-4 F-S subunit
bifunctional folylpolyglutamate synthase/ dihydrofolate synthase
nicotinamidase/pyrazinamidase
bifunctional beta-cystathionase/maltose regulon regulatory protein
ethanol-active dehydrogenase/acetaldehyde-active reductase
cob(I)alamin adenolsyltransferase/cobinamide ATP-dependent adenolsyltransferase
hydrogenase-1 operon protein HyaF
hydrogenase-1 operon protein HyaE
pyruvate formate lyase-activating enzyme 1
stationary phase/starvation inducible regulatory protein CspD
pyruvate formate lyase-activating protein
[citrate [pro-3S]-lyase] ligase
S-adenosylmethionine:tRNA ribosyltransferase-isomerase
bifunctional glycosyl transferase/transpeptidase
RNA polymerase-binding transcription factor
transposase, IS1 family protein
biotin--[acetyl-CoA-carboxylase] synthetase
bifunctional N-acetylglucosamine-1-phosphate uridyltransferase/glucosamine-1-ph
osphate acetyltransferase
bifunctional phosphopantothenoylcysteine decarboxylase/phosphopantothenate synt
hase
```

Problem 2:

Find products whose description starts with the letters "RNA". Again search the entire genome.

```
In [4]: max_i = 5000 # search the entire genome
        i = 0 # counter that will keep track of the number of CDSs found
        for feature in record.features:
            if feature.type == 'CDS':
                i += 1

                # we can only proceed if the CDS has a 'product' qualifier
                if "product" in feature.qualifiers:
                    product = feature.qualifiers["product"][0]

                    # Search for strings with "RNA" at the beginning
                    match = re.search(r"^RNA", product)
                    if match:
                        # yes, we found a match
                        print(product)

            # stop after max_i CDSs have been processed
            if i >= max_i:
                break
```

```
RNA polymerase factor sigma-32
RNA ligase
RNA 3'-terminal-phosphate cyclase
RNA polymerase factor sigma-54
RNA-binding protein YhbY
RNA polymerase sigma factor RpoD
RNA pyrophosphohydrolase
RNA polymerase sigma factor RpoS
RNA polymerase sigma factor RpoE
RNA methyltransferase RsmF
RNA polymerase-binding transcription factor
RNA methyltransferase
RNA 2'-phosphotransferase
RNA polymerase sigma factor FecI
RNA-binding protein Hfq
```

Problem 3:

Transcriptional regulators can belong to different families. These families are generally listed in the `product` field, e.g. "LysR family transcriptional regulator" or "AraC family transcriptional regulator". Write a program that extracts the family name for each transcriptional regulator and then counts how many regulators for each family are found.

In [5]:
```
max_i = 5000 # do the entire genome
i = 0
family_dict = {}
for feature in record.features:
    if feature.type == 'CDS':
        i += 1
        if "product" in feature.qualifiers:
            product = feature.qualifiers["product"][0]
            match = re.search(r"(.* family) transcriptional regulator$", produc
t)
            if match:
                family = match.group(1)
#                print("found transcriptional regulator:", family)
                if family in family_dict:
                    family_dict[family] += 1
                else:
                    family_dict[family] = 1
    if i >= max_i:
        break

print("family \t\tcount")  # \t creates a tab stop to make a nicely formatted t
able
for key in family_dict:
    print(key, "\t", family_dict[key])
```

```
family          count
NrdR family      1
HxlR family      1
LytTR family     1
LacI family      1
LuxR family      7
TetR family      3
XylR family      1
AbrB family      1
GntR family      8
IclR family      3
ArsR family      1
Fis family       2
LysR family      20
MerR family      1
TorR family      1
AraC family      11
XRE family       4
Crp/Fnr family   2
CysB family      1
```

If this was easy

Problem 4:

Write a function that takes a string holding a full name as input and that prints the first name as output. The function should be able to handle the following cases:

- first last
- first initial last
- initial first last
- last, first
- last, first initial
- last, initial first

In all cases, the output should be "first". Assume that initials are given as one letter and a period.

Hint: First separate the last name from first + initial, and then extract the first name from first + initial.

In [6]:

```
def extract_first_name(name):
    # first extract the first name + initial
    match = re.search(r"\S+,\s(.+)", name) # is the name given in the form "las
t, ..."?
    if match:
        first_and_initial = match.group(1)
    else: # no, name is given in the form "... last"
        match = re.search(r"(.+)\s\S+", name)
        if match:
            first_and_initial = match.group(1)
        else:
            print("Error: name doesn't match the expected pattern.")
            return

    match = re.search(r"(\S+)\s\S\.", first_and_initial) # is the name given as
first + initial?
    if match:
        print("First name:", match.group(1))
        return

    match = re.search(r"\S\.\s(\S+)", first_and_initial) # is the name given as
initial + first?
    if match:
        print("First name:", match.group(1))
        return

    # no initial given
    print("First name:", first_and_initial)

extract_first_name("John Smith")
extract_first_name("Miller, Jack")
extract_first_name("Susie R. Benner")
extract_first_name("Smith, April B.")
extract_first_name("Miller, R. Ben")
extract_first_name("A. Jane Doe")
extract_first_name("abcde") # not a valid name, creates an error
```

```
First name: John
First name: Jack
First name: Susie
First name: April
First name: Ben
First name: Jane
Error: name doesn't match the expected pattern.
```