String Functions and Regular Expressions

Anastasis Oulas
Evangelos Pafilis
Jacques Lagnel

Strings - Revision

- Declaration and value assignment eg.

courseName **=** **'**Introduction to Python**'**

- Concatenation

field = 'computational' **+** ' ' **+** 'Biology'

- Equality check

stringA **==** stringB , stringA **!=** stringC

- Containment check

stringA **in** stringB, stringA **not in** stringB

Relevance to Bioinformatics

- In Bioinformatics many of the tasks have to do with sequences
- Sequences can be represented as Strings
- Elements on sequences are also Strings
- Pick your own choice: codons, transcription factor binding sites, tata-box, restriction enzyme cutting sites, primer sequences, intron/exon boundary sequences
- Data/Result file handling is String manipulation

Strings - Revision

- Declaration and value assignment eg.

  seqA = 'ACGTC'
- Concatenation

  seqB = seqA **+** 'AAAA'
- Equality check

  seqA **==** seqB , seqA **!=** seqB
- Containment check

  seqA **in** seqB, seqB **not in** seqA

Example

```
seq = 'ACGTCATAATTAGCTGACGAG'
site = 'AATT' #EcoRI cutting site
print('seq contains the site: ', site in seq)
```

Example

Sometimes you want the position in the sequence:

```
seq = 'ACGTCATAATTAGCTGACGAG'
site = 'AATT' #EcoRI cutting site
startingPosition = seq.find(site)
print(startingPosition)
```

find() returns an integer

Example

Sometimes you want the position in the sequence:

```
0 1 2 3 4 5 6 7.............20
A C G T C A T A A T T A ....G
```

startingPosition

Example

```
seqA = 'ACGTCAUUUUUUUU'
seqB = 'ACGT'
if seqA.startswith(seqB):
   print('Seq A starts with seq B')
```

startswith() returns a Boolean (True/False)

Example

```
seqA = 'ACGTCAUUUUUUUU'
seqB = 'ACGT'
print ('SeqB starts with seqA (t/f):')
print (seqB.startswith(seqA))
```

startswith() returns a Boolean (True/False)

Example: substring

General view:

substring = mainString[start position:end position]

The character at the 'end position' is **NEVER** included

Example: substring

Sometimes you want extract pa part of the string:

```
seq = 'ACGTCATAAT'
```

Example: substring

Sometimes you want extract pa part of the string:

```
index: 0123456789
seq = 'ACGTCATAAT'

substr=seq[3:6]
```

Example: substring

Sometimes you want extract pa part of the string:

```
index: 0123456789
seq = 'ACGTCATAAT'

substr=seq[3:6]
print (substr)
```

Gives: TCA

Example: split string

Sometimes you want build a list of words from a string

string1='hello my world'

list1=['hello', 'my', 'world']

space as Separator

```
list1=string1.split(' ')
```

Example: strings join

Sometimes you want the reverse eg you have the list:

list1=['hello', 'my', 'world']

And you want to join the words the a space.
This can be done using **join()**

```
listA=['hello', 'my', 'world']
space = ' '
stringA = space.join(listA)
print( stringA )
```

=> Prints **hello my world**

String functions

Searching

- *str1*.**startswith**(*str2*[, *startpos*, [*endpos*]])
  - Returns true if *str1* starts with *str2*
- *str1*.**endswith**(*str2*[, *startpos*, [*endos*]])
  - Returns true if *str1* ends with *str2*
- *str1*.**find**(*str2*[, *startpos*[, *endpos*]])
  - Returns the lowest index of *str1* at which *str2* is found, or −1 if it is not found
- *str1*.**index**(*str2*[, *startpos*[, *endpos*]])
  - Returns the lowest index of *str1* at which *str2* is found, or ValueError if it is not found

String functions - Table

Replacing and changing case

- *str1*.**lower**()
  - Returns a copy of the string with all of its characters **converted to lowercase**
- *str1*.**upper**()
  - Returns a copy of the string with all of its characters **converted to uppercase**
- *str1*.**replace**(*oldstr*, *newstr*[, *count*])
  - Returns a copy of *str1* with **all occurrences** of the substring ***oldstr*** **replaced** by the string ***newstr***; if *count* is specified, only the first *count* occurrences are replaced

String functions

- *str1.**join**(list1)*
- Returns a string containing the elements of *list1* separated by the *str1 string*

Testing

- *str1*.**islower**()
  - Returns true if *str1* contains at least one “cased” character and all of its cased characters are lowercase
- *str1*.**isupper**()
  - Returns true if *str1* contains at least one “cased” character and all of its cased characters are uppercase

Regular Expressions

- However the requirements of Bioinformatics / Computational Biology exceed what can be achieved with the available String functions
- This has given rise to wide usage of Regular Expressions
- What is a Regular Expression and why is it so useful?

Why a regular expression

- `'AATT' #EcoRI cutting site`
  - `'AATT' in sequence`

- DsaI possible cutting sites: CC - **G or A - T or C -** GG
- `'CC`**`GT`**`GG' in sequence`
- `'CC`**`GC`**`GG' in sequence`

Why a regular expression

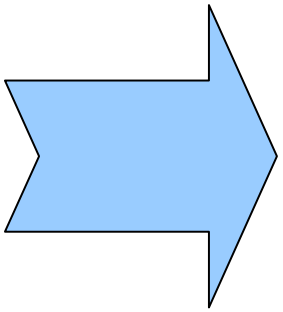
- `'AATT' #EcoRI cutting site`
  - `'AATT' in sequence`

- DsaI possible cutting sites: CC - **G or A - T or C -** GG
- `'CC`**`GT`**`GG' in sequence`
- `'CC`**`GC`**`GG' in sequence`
- `'CC`**`AT`**`GG' in sequence`
- `'CC`**`AC`**`GG' in sequence`

Why a regular expression

- `'AATT' #EcoRI cutting site`
  - `'AATT' in sequence`

- DsaI possible cutting sites: CC - **G or A - T or C -** GG
- `'CCGTGG' in sequence`
- `'CCGCGG' in sequence`
- `'CCATGG' in sequence`
- `'CCACGG' in sequence`



Combinatorial Explosion

What is a regular expression

- Regular Expressions provide the tool to manage this "combinatorial explosion"
- A regular expression for Dsal's site would be:
  - 'CC**[GA][TC]**GG'

    [ ] → a set of possible characters at a single position

    [GA]: this position will contain either G or A
    (ie possible characters)

    [TC]: this position will contain either T or C

Regular expressions: Another example

- Find the pattern **enzym** followed by **any character (.) any number of times incl zero (*)**
  - Eg Reg Expr: enzym**.***
    - **enzyme**
    - **enzymes**
    - **enzymatic**
    - **enzym**

Regular Expression Syntax

- . Any character
- [ ] A character set
- [ACTG] One DNA base character
- [A-Za-z_] One underscore or letter
- [0-9] a digit

Regular Expression Syntax

- \n a newline character
- \d Any digit
- \D Any nondigit
- \s Any whitespace character
  - space ' ' , tab \t, new line: \n\r
  - ie. shorthand for [\t\n\r]
- \S Any non-whitespace character

  ie. all characters excluding [\t\n\r]

Regular Expression Syntax

- ***** Zero or more repetitions of the preceding regular expression
- **?** Zero or one repetitions of the preceding regular expression
- **+** One or more repetitions of the preceding regular expression
- **{*n*}** Exactly ***n*** repetitions of the preceding regular expression
- **{*m*,*n*}** Between ***m*** and ***n*** (inclusive) repetitions of the preceding regular expression

Regular Expressions

- **( )** : captures a group of characters

eg. (TA) : matches TA in ACGA**TA**GACC

- Can be combined with the repetition quantifiers
  eg. (TA)***{3}*** **:** matches **TATATA** in ACGA**TATATA**CC

The re Module

- `import re`
- By writing the above statement in a python script the **re** (regular expression) **module** is imported and ready to use.
- You are now able to use the methods of the regular expression library in your algorithm

Example code

```
import re
seq = 'ACCGTGGCAAATTTCCACGGACGAG'
regEx = 'CC[GA][TC]GG'
aList = re.findall(regEx,seq)
for i in range(0,len(aList)):
    print('Found', aList[i])
```

- finds any DsaI cutting sites in the given sequence
- The result is : Found CCGTGG
  Found CCACGG

Example code

```
import re
text = 'this is a test paragraph'
regEx = 'A\stest'
aList = re.findall(regEx,text)
if len(aList) == 0:
    print('Not Found')
```

- Checks whether the sentence contains “A text”

The result is : Not Found

Example code

```
import re
seq = 'ACGATATACC'
regEx = '(TA){2}'
aList = re.findall(regEx,seq)
if len(aList) > 0:
      print('Found TATA')
else:
      print ('Not Found')
```

The result is : ?

Example code

```
import re
seq = 'ACGATATACC'
regEx = '(TA){3}'
aList = re.findall(regEx,seq)
if len(aList) > 0:
      print('Found TATA')
else:
      print ('Not Found')
```

The result is : ?

Substitution example: re.sub()

Regular expressions can be used to perform substitutions eg replace all T's or C's with a "-" in a sequence

```
seq = 'AAACGCTGTCAATACAATCTTCTTTCGGATTTGAATTTTGCAAAGCTGCC'
regEx = '[TC]'
replacement = '-'
new_seq = re.sub(regEx , replacement , seq )
print (new_seq )
```

The result is :

AAA-G--G--AA-A-AA--------GGA---GAA----G-AAAG--G--

findall() function of the re module

re.sub(regEx , replacement , targetString)

Returns a string with all the matches of the *regEx* in the *targetString* substituted with the *replacement* string

re.findall(*pattern*, *target*[, *flags*])

Returns a list of all nonoverlapping matches in *target* as a list of strings or, if the pattern included groups, a list of lists of strings

([, *flags*]: it is optional and exceeds the scope of this tutorial, however if required we would be happy to explain you more)

More functions are available at http://docs.python.org/library/re.html

File I/O – reading from a file

```
F = open('C:\Documents and
  Settings\Administrator\Desktop\User\Pyt
  hon course\Seq.txt', 'r')
```

F is the file handler allows you to have a direct link to the contents of the file - Seq.txt

```
lines = F.readlines() # command reads all
  the lines of the file into a list
  called lines
```

```
F.close()
```

File I/O – writing to a file

```
F = open('C:\Documents and
  Settings\Administrator\Desktop\User\Pyt
  hon course\Out.txt', 'w')
```

F is the file handler allows you to have a direct link to the contents of the file - Seq.txt

```
F.write('Hello') # command writes the
  word “Hello” in the file Out.txt
```

```
F.close()
```