UAX
UNIVERSIDAD ALFONSO X EL SABIO

Expresiones Regulares

GRADO EN BIOMEDICINA

Bioinformática

idelhgar@uax.es

2023

Modulo re

- El módulo re contiene los métodos necesarios para analizar y buscar expresiones regulares en Python.
- **Import re**
- **Re.search(pattern,string)**
- La función usa dos argumentos, el patrón y la cadena dónde hay que buscarlo
- ***Para indicar a Python que ignore los caracteres especiales como “\t” o “\n” se puede usar r fuera de las comillas: print(r”\t\n”)**

Buscar un patrón en una string

- El resultado que devuelve re.search() es booleano, true/false

```
import re

dna = "ATCGCGAATTCAC"
if re.search(r"GAATTC", dna):
    print("restriction enzyme site found!")
```

- **ALTERNATION ($X|X_1$)**

```
dna = "ATCGCGAATTCAC"
if re.search(r"GGATC", dna) or if re.search(r"GGTCC", dna):
    print("restriction enzyme site found!")
```

```
dna = "ATCGCGAATTCAC"
if re.search(r"GG(A|T)TC", dna):
    print("restriction enzyme site found!")
```

Buscar un patrón en una string

UAX
UNIVERSIDAD ALFONSO X EL SABIO

- **CHARACTER GROUPS [X_i,X,X,X....X_n]**

Restriction enzyme BisI cuts in a wider range of motifs, the pattern is GCNGC, where N represent any base

```
dna = "ATCGCGAATTCAC"
if re.search(r"GC(A|T|G|C)GC", dna):
    print("restriction enzyme site found!")

dna = "ATCGCGAATTCAC"
if re.search(r"GC[ATGC]GC", dna):
    print("restriction enzyme site found!")
```

This is different to trying to match any character, for that we can use . -> GC . GC
It going to match to any character beyond DNA base. GCFGC, GC$GC,...

Buscar un patrón en una string

UAX
UNIVERSIDAD ALFONSO X EL SABIO

- **NEGATION [^XYZ]**

To specify the characters that we don't want to match ^ inside a group (only inside [])

```
#negation
dna = "ATCGCGAATTCAC"
if re.search(r"GC[^ATGC]GC", dna):
    print("restriction enzyme site found!")
```

Buscar un patrón en una string

UAX
UNIVERSIDAD ALFONSO X EL SABIO

- **QUANTIFIERS**

Let us describe variation in the number of times a section of a pattern is repeated

? - It can match either **zero or one time**
GAT?C -> GATC or GAC
GGG(AAA)?TTT -> GGGAAATTT or GGGTTT

+ - the character or group must to be present but can be repeated any number of times
GAT+C -> GATC or GATTC, GATTTC, GATTTTTTC, but not GAC

{ } - The number inside curly brackets will match exactly the number of repeats
GAT{4}C -> GATTTTC
GAT{,4}C -> up to 4 T
GAT{4,}C -> 4 or more T

Buscar un patrón en una string

- **POSITIONS**

^symbol for start and $ symbol for end

^AAA -> AAAT but not GAAAT

GGG$ -> AAAGGG but not GGGCCC

Buscar un patrón en una string

- **COMBINING**

Generating very flexible pattern. For example, here´s a complex pattern to identify full-length eukaryotic messenger RNA sequences:

^ATG[ATGC]{30,1000}A{A5,10}$

Reading from left to right:

- ATG start codon at the beginning of the sequence
- Followed by between 30 and 1000 bases (A,T,G,C)
- Followed by a poly-A tail between 5-10 bases at the end of the transcript.

Extraer el patrón

UNIVERSIDAD ALFONSO X EL SABIO

what happens if we want to know what part of the string was matched?

If re.search finds a match the value that actually returned is a match object (which is considered True in a IF statement). Match object is an object that represent a regular expression search. And this object has a number of useful methods for getting data out of it.

group method : extract the portion of input string that matched the pattern

```
#Extracting the part of the string that matched
dna = "ATGACGTACGTACGACTG"
#store the matches in a variable called m
m = re.search(r"GA[ATGC]{3}AC",dna)
m_pattern = m.group()
print(m_pattern)
```

group_method.py

Extraer el patrón

UAX
UNIVERSIDAD ALFONSO X EL SABIO

What if we want to extract more than one bit of the pattern?

GA([ATGC]{3})AC([ATGC]{2})AC

bit1 bit2

group(1) or group(2) to extract the bits

```
#extracting more than one bit of the pattern
dna = "ATGACGTACGTACGACTG"
m = re.search(r"GA([ATGC]{3})AC([ATGC]{2})AC", dna)
print("entire match: " + m.group())
print("first bit: " + m.group(1))
print("second bit: " + m.group(2))
```

extracting_bits.py

Obtener la posición del match

UAX
UNIVERSIDAD ALFONSO X EL SABIO

start() and end() methods gets the positions of the start and end of the pattern on the sequence.

```
#start end methods
dna = "ATGACGTACGTACGACTG"
m = re.search(r"GA([ATGC]{3})AC([ATGC]{2})AC", dna)
print("input string: " + dna)
print("entire match: " + m.group())

#start
print("start: " + str(m.start()))

#end
print("end: " + str(m.end()))
```

start_end.py

Dividir una string usando un regex

UAX
UNIVERSIDAD ALFONSO X EL SABIO

The re module has a split() function that split a string using a regular expression pattern.

re.split(regular expression, string)

```
dna = "ACTNGCATRGCTACGTYACGATSCGAWTCG"
runs = re.split(r"[^ATGC]", dna)
print(runs)
```

The output shows how the function works – the return value is a list of strings not a re object:

```
['ACT', 'GCAT', 'GCTACGT', 'ACGAT', 'CGA', 'TCG']
```

split.py

Encontrar múltiples matches

UAX
UNIVERSIDAD ALFONSO X EL SABIO

What happens if we want to find every place where a pattern occurs in a string?, there are two functions in the re module.

re.findall(regular expression, string) : return a list of all matches of a pattern in a string

```
#finall function
dna = "CTGCATTATATCGTACGAAATTATACGCGCG"
result = re.findall(r"[AT]{6,}", dna)
print(result)
```

the return value is a list of strings:

['ATTATAT', 'AAATTATA']

findall.py

Encontrar múltiples matches

UAX
UNIVERSIDAD ALFONSO X EL SABIO

What happens if we want to find every place where a pattern occurs in a string?, there are two functions in the re module.

re.finditer(regular expression, string) : return a sequence of match objects

```
#finditer function to detect wrong bases
dna = "CGCTCNTAGATGCGCRATGACTGCAYTGC"

matches = re.finditer(r"[^ATGC]", dna)
for m in matches:
    base = m.group()
    pos  = m.start()
    print(base + " found at position " + str(pos))
```

```
N found at position 5
R found at position 15
Y found at position 25
```

finditer.py

Recap

it's important to recognize that for any given pattern, there are probably multiple ways to describe it using a regular expression

```
GG(A|T)CC
```

to describe the AvaII restriction enzyme recognition site, but the same pattern could also be written as:

```
GG[AT]CC
(GGACC|GGTCC)
(GGA|GGT)CC
G{2}[AT]C{2}
```

As with other situations where there are multiple different ways to write the same thing, it's best to be guided by what is clearest to read.