PROGRAMACIÓN EN PYTHON 3

Clara Higuera

Laboratorio Integrado de Biofísica y Bioinformática

Nov-2015

Ficheros de texto

- Los ficheros son secuencias de datos almacenados en disco (real o virtual)
- Primero, abrir el fichero para obtener un manejador (*handle*)
- Se especifica la ruta hasta el fichero y el modo de operación (opcional), ambos como cadenas

```
>>> fh = open('file.txt', 'r')
```

- Rutas:Absolutas: ruta completa desde raiz
  - 'C:\Users\Alice\file.txt'
- Relativas: Desde el directorio actual

Ficheros de texto

- Modos
  - 'r': Sólo lectura (modo por defecto)
  - 'w': Escritura (¡OJO! El fichero se crea de nuevo)
  - 'a': Adición
- En modo 'w' o 'a', el fichero se crea. En modo 'w' se sobreescriben los contenidos
- Un '+' tras la letra indica tanto lectura como escritura

Ficheros de texto

- Lectura de ficheros
  - Un fichero de texto se puede leer de una vez

```
>>> fh = open('fichero.txt')
>>> contenido = fh.read() # Leemos todo el fichero
```

    - Devuelve una cadena con todo el contenido del fichero.

  - También se puede leer linea a linea

```
>>> l1 = fh.readline() # Leemos una línea
>>> l2 = fh.readline() # Hasta que se agotan
```

  - O almacenar todas las líneas en una lista, donde cada elemento de la lista contiene una línea

```
>>> lista = fh.readlines()
```

Ficheros de texto

- Lectura
  - Al terminar de acceder al fichero debemos cerrar el recurso

```
>>> fh.close()
```

  - A menudo, necesitaremos leer el fichero línea a línea y eliminar los saltos de línea

  - También podemos usar un bucle para ir accediendo a cada línea

```
for linea in fh:
    print linea
```

Ficheros de texto

- Escritura
  - Para escribir un fichero, primero lo abrimos en mode 'w' o 'a'

```
>>> fh = open('nuevo.txt','w')
```

  - Cuidado con borrar el contenido de un fichero ya existente
  - El método para escribir es **write**

```
>>> fh.write("Contenido\n") # No olvidar '\n'
```

    - El método write recibe como parámetro una cadena.

```
>>> fh.write(10)  # ERROR! Hay que convertir con str
```

Ficheros de texto

- Lectura
  - Ejercicio 1
    - Descargar del campus fichero fasta_file.fasta

```
>gi|182723|gb|J00140.1|HUMFOLMES Human dihydrofolate reductase gene
TGCAGGGGGGGGGGGGGGGCGGAGGTCCTCCCGCTGCTGTCATGGTTGGTTCGCTAAACTGCATCGTCG
CTGTGTCCCAGAACATGGGCATCGGCAAGAACGGGGACCTGCCCTGGCCAACCGCTCAGGAATGAATTCAG
ATATTTCCAGAGAATGACCACAACCTCTTCAGTAGAAGGTAAACAGAATCTGGTGATTATGGGTAAGAAG
ACCTGGTTCTCCATTCCTGAGAAGAATCGACCTTTAAAGGGTAGAATTAATTTAGTTCTCAGCAGAGAAC
TCAAGGAACCTCCACAAGGAGCTCATTTTCTTTCCAGAAGTCTAGATGATGCCTTAAAACTTACTGAACA
ACCAGAATTAGCAAATAAAGTAGACATGGTCTGGATAGTTGGTGGCAGTTCTGTTTATAAGGAAGCCATG
AATCACCCAGGCCATCTTAAACTATTTGTGACAAGGATCATGCAAGACTTTGAAAGTGACACGTTTTTTC
CAGAAATTGATTTGGAGAAATATAAACTTCTGCCAGAATACCCAGGTGTTCTCTCTGATGTCCAGGAGGA
GAAAGGCATTAAGTACAAATTTGAAGTATATGAGAAGAATGATTAATATGAAGGTGTTTTCTAGTTTAAG
TTGTTCCCCCTCCCTCTGAAAAAAGTATGTATTTTTACATTAGAAAAGGTTTTTTGTTGACTTTAGATCT
ATAATTATTTCTAAGCAACTTGTTTTTATTCCCCACTACTCTTCTCTCTATCAGATACCATTTATGAGAC
ATTCTTGCTATAACTAAGTGCTTCTCCAAGACCCCAACTGAGTCCCCAGCACCTGCTACAGTGAGCTGCC
ATTCCACACCCATCACATGTGGCACTCTTGCCAGTCCTTGACATTGTCGGGCTTTTCACATGTTGGTAAT
ATTTATTAAAGATGAAGATCCACATACCCTTCAACTGAGCAGTTTCACTAGTGGAAATACCAAAAGCTT
```

Ficheros de texto

- Ejercicio 1
  - Abrir el fichero para lectura
  - Almacenar en una variable la primera linea que contiene información del gen e imprimirla.
  - Leer el resto de líneas y crear una variable que contenga toda la secuencia de nucleotidos reconstruida sin saltos de línea.
    - Nota: usar la función rstrip() y la concatenación de cadenas
  - Escribir en un nuevo fichero la linea de información del gen en la primera linea y la secuencia completa de DNA reconstruída sin saltos de linea.

Ficheros de texto

- Ejercicio 2
  - Abrir el fichero dna1.txt.
  - Este fichero contiene 5 líneas de cabecero y el resto con DNA. Ignorar las 5 primeras.
  - Leer las líneas del fichero que contienen DNA en una lista y reconstruir la cadena no teniendo en cuenta los números que aparecen al principio
    - Nota: hacer uso de la función split() de cadenas
  - Contar cuántas veces aparece cada una de las bases en la cadena de DNA.
  - Escribir en un fichero el numero de a's de c's, de g's y t's

Expresiones regulares

- Una expresión regular, ER, es una cadena que describe un patrón.
- Su utilidad es:
  - Buscar una cadena en otra
  - Extraer las partes deseadas de una cadena
  - Tratar ficheros con formato, como Fasta o GB
  - Analizar secuencias de importancia biológica
    - Por ejemplo, motivos de unión de factores de transcripción, sitios de enzimas de restricción, etc.
- Se basan en:
  - Repetición
  - Concatenación
  - Alternancia

Expresiones regulares

- La ER más sencilla es simplemente una **cadena de caracteres** que describe ese patrón
- **‘Hola’** es la ER que describe al patrón **Hola**
- Lo interesante es que una única expresión regular puede describir a muchos patrones
- Ejemplo:
  - El punto en una ER describe a cualquier carácter: **‘.ython’** describe al patrón **‘python’**, a **’sython’** , o cualquier palabra de 6 letras acabada en ‘ython’

Expresiones regulares

- Para poder manejar expresiones regulares:

```
>>> import re
```

- Para hacer una búsqueda, usamos la función **search.** Busca en la cadena especificada la primera aparición del patrón si es que se encuentra.

```
>>> re.search(exp_regular, cadena)
```

- Devuelve un objeto (MatchObject) con el resultado

```
>>> busqueda = re.search("GCG", "GCAGCATGAGCG")
```

- A este objeto están asociadas varias funciones:

```
>>> busqueda.group()
'GCG'
>>> busqueda.span()
(9, 12)
```

```
>>> busqueda.start()
9
>>> busqueda.end()
12
```

Distingue entre mayúsculas y minúsculas

Expresiones regulares

- El objeto resultado también puede usarse como condición:

```
busqueda= re.search('hola','hola buenos dias')
if busqueda:
    print "encontrado el patron en la cadena”
elif:
    print “no encontrado”
```

- Otras funciones de el modulo re
  - **re.sub**(patron, repl, cadena)
    Devuelve la cadena obtenida reemplazando las apariciones (no solapadas y por la izquierda) del patrón en la cadena con repl.

```
>>> re.sub('tardes','noches','Hola, Buenas tardes')
'Hola, Buenas noches'
```

Expresiones regulares

- Caracteres especiales. Hay una serie de caracteres que tienen significado especial dentro de una ER, por ejemplo el punto.
- Los caracteres especiales son:

  { } [ ] ( ) ^ $. | * + ? \

- Como en las cadenas de caracteres \ sirve para mostrar caracteres especiales, si se quiere usar \ para dar significado a algo en ER debe ponerse doble \\

Expresiones regulares

- Caracteres especiales
  - . (punto) = Cualquier carácter menos '\n'

Algunos sirven para indicar **repeticiones:**

* = Cero o más repeticiones del carácter anterior

- + = Una o más repeticiones del carácter anterior
- ? = Zero o una vez el carácter anterior
- {n} = Exactamente n repeticiones del carácter anterior
- {m,n} = Entre m y n repeticiones del carácter anterior

Expresiones regulares

- Caracteres especiales

( ) = Grupo. Ej: (CT)* = cero o más repeticiones de CT

\ = Escape. El siguiente carácter se interpreta literalmente

  - P. ej. Para representar ?, es necesario escribir \?

- | = Separa alternativas, es decir (A|B)C = AC o BC
- [ ] = Representa conjuntos de caracteres
- \w = un carácter alfanumerico
- \W = un carácter no alfanumérico
- \d = un carácter numérico
- \D = un carácter no numérico
- \s = cualquier espacio (lo mismo que \t\n)
- \S = un no espacio

Expresiones regulares

- Conjuntos de caracteres
  - Permiten a un conjunto de caracteres, en vez de a uno solo, aparecer en un punto dado de una ER.
  - Se representan mediante corchetes [ ], donde lo que aparece dentro de los corchetes son los caracteres que se quiere hacer coincidir.
- Ej:
  - [bcr]at identificaría “bat”, “cat” o “rat”
  - [yY][eE][sS] identificaría yes, sin preocuparse de mayúsculas y minúsculas.

```
>>> busqueda = re.search("GC[AG]", "GCAGCATGAGCG")
>>> busqueda.group()
'GCA'
```

Expresiones regulares

- Conjuntos de caracteres
  - Rangos. Dentro de los conjuntos de caracteres se puede usar rangos, especificados con –
    - [a-z] identifica cualquier carácter alfabético en minúsculas
    - [a-zA-Z0-9] identifica cualquier carácter alfabético o numérico
    - Un circunflejo al principio niega, i.e. [^0-9] es cualquier cosa menos un número

- Alternancia
  - Se expresan mediante el pipe: |
  - ‘python|perl’ identifica o python o perl
  - ‘p(ython|erl)’ identifica lo mismo

Expresiones regulares

- Grupos
  - () identifica la ER que esté dentro de los paréntesis.
  - Indica el comienzo y fin de un grupo identificado
  - Se puede acceder a los grupos que se numeran por su paréntesis izquierdo
  - El grupo 0 es todo el patrón

```
import re

contactInfo = 'Doe, John: 555-1212'
match=re.search('\w+, \w+: \w+-\w+', contactInfo)

if match:
    print "FOUND"

print "Grupo:",match.group(0),"\n"

match2 = re.search('(\w+), (\w+): (\w+-\w+)', contactInfo)
#match2= re.search('(\w+), (\w+): (\S+)', contactInfo)

grupo1= match2.group(1)
grupo2= match2.group(2)
grupo3= match2.group(3)

print "Grupo1:",grupo1,"\n"
print "Grupo2:",grupo2,"\n"
print "Grupo3:",grupo3,"\n"
```

Expresiones regulares

- Anclajes
  - ^ busca una subcadena al principio de una cadena
  - ‘^http’ 
  - $ busca una subcadena al final de una cadena ‘org$’ 
  - ‘^http$’ identifica solo http 

Expresiones regulares

- **Ejercicio 3**
  - Para el siguiente código:

    busqueda1= re.search(‘(.*)(cat)(.*)', ‘the cat in the hat’)

  - Determinar qué se identifica en cada grupo.

  - Hacer lo mismo para este código:

    busqueda2= re.search('(.*)(at)(.*)', 'the cat in the hat')

Expresiones regulares

Ejercicio 4

- En PERL los nombres de variable se forman igual que en Python pero van precedidos del símbolo $.
- Crear un programa que pida al usuario una palabra y diga si es una variable legal de PERL

Expresiones regulares

Ejercicio 5

En Python los números reales (float) pueden aparecer con los siguientes formatos:

- 1.23
- 1.
- 3.14e-10
- 4E21
- 4.0e+45

Crear un programa que pida al usuario un número y diga si es un número real

Expresiones regulares

- Multiples resultados: **findall.** Encuentra todos los patrones no solapantes. Devuelve una lista

```
>>> mo = re.findall("A[GC]A", "AGACAAGACATAGCACA")
>>> mo
['AGA', 'AGA', 'ACA']
```

- Podemos iterar sobre los resultados con **finditer**

```
>>> iter = re.finditer("A[GC]A", "AGACAAGACATAGCACA")
>>> for i in iter:
...     i.group()
...     i.span()
...
'AGA'
(0, 3)
'AGA'
(5, 8)
'ACA'
(14, 17)
```

Expresiones regulares

Ejercicio 6. Subir al campus

- Bajar el genoma de *E. coli* del Campus Virtual y encontrar hasta tres sitios de restricción de las siguientes enzimas y mostrarlos por pantalla.
  - EcoRII: CCAGG or CCTGG
  - DsaI: CCGCGG, CCGTGG, CCACGG, o CCATGG
  - CjuI: "CA", seguido de C o T, seguido de 5 bases cualquiera y de una G o A