Procesado y Manejo de Datos Masivos

Resumen

En esta asignatura se aprenderán técnicas para procesar y manipular grandes volúmenes de datos utilizando programación y línea de comandos. Se abordará el manejo y análisis de formatos de datos comunes en bioinformática, como FASTA, GFF y GenBank, utilizando patrones que optimizan el uso de memoria y tiempo de ejecución, así como técnicas de programación paralela. Además, se realizarán operaciones avanzadas en bases de datos relacionales y no relacionales, y se explorará el acceso programático a bases de datos biomédicas online a través de sus APIs.

Sandra Mingo Ramírez

UAM - 2024/25

10 de octubre de 2024 17:56

Universidad Autónoma de Madrid
Bioinformática y Biología Computacional

Código en Github

Índice general

Capítulo I

Procesado y manejo de datos masivos

I.1. Formatos de ficheros bioinformáticos

Los formatos de fichero son en formato texto, y los principales son:

- FASTA: guarda secuencia de nucleótidos o aminoácidos.
- FASTQ: conjunto de secuencias de nucleótidos leídas junto con sus puntuaciones de calidad
- GFF/GTF: formato de anotación de características del genoma
- GenBank: formato enriquecido para secuencias de nucleótidos o aminoácidos.

I.1.1. FASTA

Guarda una o varias secuencias de nucleótidos o aminoácidos. Se pronuncia como "fast A", que significa "fast all", siendo all cualquier alfabeto. Los archivos fasta pueden tener las siguientes extensiones:

- **.fasta, .fa**: extensión FASTA genérica
- **.fna**: contiene secuencia de nucleótidos
- **.faa**: contiene secuencia de aminoácidos
- **.frn**: contiene secuencias de ARN no codificante

En la figura I.1 se puede ver la estructura de un fichero FASTA. Cada secuencia consiste en una línea descriptiva seguida de las líneas que contienen la secuencia. La línea descriptiva cuenta con un símbolo > seguido del identificativo de la secuencia. Tras un espacio, se pone la descripción de la secuencia. En algunos casos, las secuencias se encuentran en mayúscula, en minúscula o en una mezcla de ambas. Por ello, se

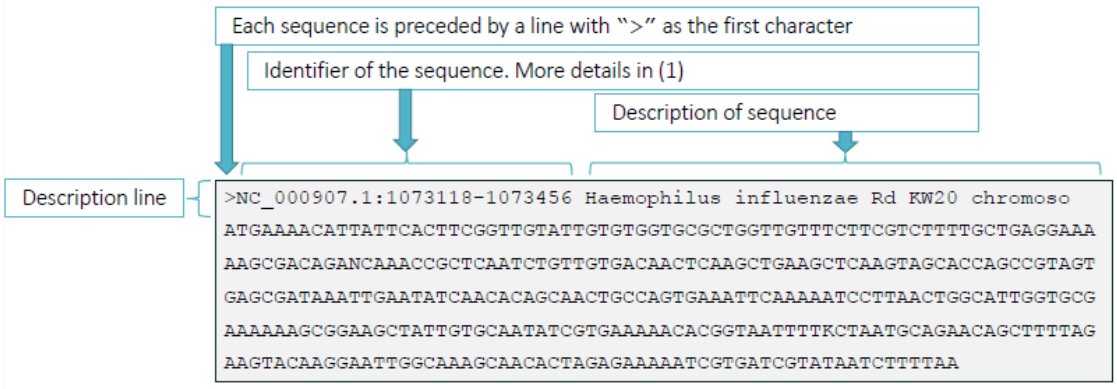


Figura I.1: *Anatomía de un fichero FASTA*

puede pasar toda la secuencia a mayúsculas o minúsculas con .upper() y .lower() respectivamente.

La secuencia se puede expandir por múltiples líneas, las cuales generalmente tienen unos 80 caracteres. No hay espacios en las líneas o líneas en blanco en una secuencia. Cada elemento de una secuencia se describe con un único carácter, y normalmente se escriben en mayúscula (aunque esto es una convención, no es estándar). No es recomendable utilizar guiones para representar gaps, ya que hay algunos programas que no los pueden manejar. Es mejor utilizar una N en el caso de secuencias de nucleótidos y X de aminoácidos (véase tabla I.1).

FASTA for genome data

A adenosine	C cytidine	G guanine
T thymidine	N A/G/C/T (any)	U uridine
K G/T (keto)	S G/C (strong)	Y T/C (pyrimidine)
M A/C (amino)	W A/T (weak)	R G/A (purine)
B G/T/C	D G/A/T	H A/C/T
V G/C/A	- gap	

FASTA for protein data

A alanine	B aspartate/asparagine	C cystidine
D aspartate	E glutamate	F phenylalanine
G glycine	H histidine	I isoleucine
K lysine	L leucine	M methionine
N asparagine	P proline	Q glutamine
R arginine	S serine	T threonine
U selenocysteine	V valine	W tryptophan
Y tyrosine	Z glutamate/glutamine	X any
* translation stop	- gap	

Tabla I.1: *Caracteres en los ficheros FASTA para datos genómicos y de proteínas.*

I.1.1.1. Parsear ficheros FASTA en Python

En informática, parsear significa leer un fichero. Se pueden leer ficheros FASTA en Python de la siguiente forma:

```
def readFasta(file):
    """ Reads all sequences of a FASTA file
        returns a dictionary """
    d = {}
    identifier = ''

    with open(file, 'r') as f:
        for linea in f:
            if linea[0] == ">": #alternativa: linea.startswith(">")
                if identifier != '':
                    d[identifier] = ''.join(d[identifier])
                descriptors = linea[1:].split()
                identifier = descriptors[0]
                #Alternativa: identifier = linea[1:linea.find(' ')]
                d[identifier] = []

            else:
                d[identifier].append(linea.strip('\n'))
        d[identifier] = ''.join(d[identifier])
    return d

readFasta("phix174/phix.fa")
```

I.1.2. FASTQ

Los ficheros FASTQ guardan secuencias de nucleótidos junto con las calidades.

Cada secuencia de un archivo FASTQ consta de 4 líneas. La primeras es la línea de descripción precedida de . Tiene un formato libre sin límite de longitud. A veces se coloca un identificador justo después de la . La segunda línea es la línea de secuencia, e incluye la secuencia propiamente dicha. Sigue normas y convenciones similares a las del fichero FASTA: normalmente en mayúsculas, sin espacios permitidos, etc. Esta línea podría estar envuelta en múltiples líneas como en un archivo FASTA. La siguiente es la línea que señala el final de la secuencia. Está marcada con el signo +. Puede ir seguida de la misma descripción dada en la primera línea. Sin embargo, es más sensato dejarla sólo con el '+' para reducir el tamaño de los ficheros. La última línea es la línea de puntuaciones de calidad: una secuencia codificada en caracteres con la calidad de las lecturas de la secuencia. Contiene un carácter por base. Los caracteres permitidos son como máximo ASCII 33-126 inclusive, que son todos imprimibles. Esta línea puede ser envuelta como en un archivo FASTA y su longitud total debe ser igual a la secuencia. Véase un ejemplo de un archivo FASTQ en la figura I.2.

Hay varias cuestiones que deben tenerse en cuenta al analizar un archivo FASTQ: Tanto el carácter como + pueden aparecer en la línea con los índices de calidad. Pueden aparecer en la primera posición, por lo que hay que tener cuidado al utilizar

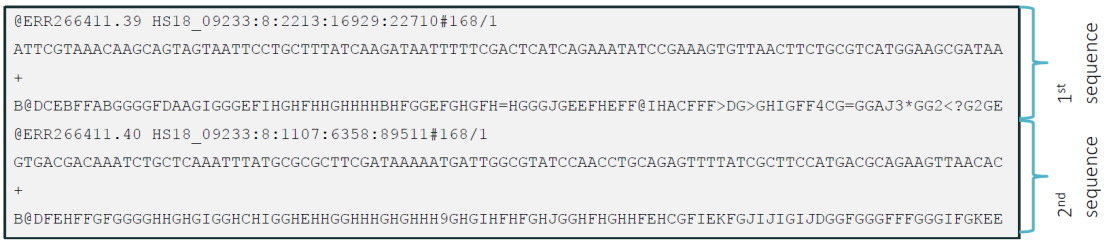


Figura I.2: *Anatomía de un fichero FASTQ*

grep o herramientas similares: por ejemplo `$> grep -v "[ +]" file.fastq` (Este comando para eliminar las líneas + y no es una buena idea). En principio, tanto la secuencia como las puntuaciones de calidad pueden tener múltiples saltos de línea. El programa de análisis debería eliminar los saltos de línea. Para reducir los problemas de análisis sintáctico, la mayoría de las herramientas producen ahora archivos fastq en los que las secuencias y las puntuaciones de calidad se dan en una sola línea, posiblemente muy larga. Por lo tanto, cada secuencia tiene exactamente 4 líneas, lo cual es conveniente.

La codificación más utilizada para las puntuaciones de calidad es el formato Sanger o codificación Phred+33. Cada carácter codifica la calidad de la lectura, Q, que se define como:

$$Q = -10 \cdot log_{10}p$$

siendo p la probabilidad de error de la lectura. Esta puntuación de calidad se utiliza porque permite una interpretación fácil de la probabilidad (Q = 10, p = 1/10; Q = 20, p = 1/100; Q = 30, p = 1000) y porque permite una codificación en un único carácter ASCII. No obstante, no se puede guardar Q directamente como un carácter porque no todos son imprimibles. Por ello, se emplea la codificación Phred+33, que codifica el valor Q como:

$$Q = ord(chr) - 33$$

donde ord() convierte el carácter a su representación ASCII. Por ejemplo, el símbolo tiene el código ASCII de 64, por lo que su Q sería de 31. Una p de 0 significa que es una lectura segura, mientras que una p de 1 indica que la lectura no es nada segura. Para calcular la p, se despeja la fórmula para que quede:

$$p = 10^{-\frac{Q}{10}}$$

Ejemplos: Para el carácter " ", el código ASCII es 126, por lo que su Q es $126 - 33 = 93$. En este caso, la p sería de $0,5 \cdot 10^{-10}$. Así, el carácter "!", cuyo ASCII es 33, tiene una Q de 0 y una p de 1.

En este caso, como las letras en mayúscula y minúscula tienen un código ASCII diferente, no se puede pasar todo a mayúsculas o minúsculas como en FASTA.

I.1.3. GFF: generic feature format

Los ficheros GFF tienen un formato de texto sin formato para representar características genómicas. Cada característica está representada por nueve columnas separadas por tabuladores:

1. seqid: id de la secuencia donde se localiza la característica. No se permiten espacios
2. source: nombre del programa o algoritmo que ha generado esta característica.
3. type: tipo de característica (region, gene, CDS, etc.)
4. start: posición donde inicia la característica, su primera base
5. end: posición donde finaliza la característica, su última base
6. score: un valor flotante que indica la confianza en la característica. Si no se conoce, se pone un punto.
7. strand: indica el sentido de la cadena, siendo + la cadena positiva y - la cadena negativa.
8. phase: puede ser 0, 1 o 2 para CDS y un punto para todo lo demás.
9. attributes: Una lista de atributos de la característica en formato tag=value separados por punto y coma (;). Se permiten espacios en blanco en estas columnas, así como tabuladores y punto y coma si se escapan. Los atributos son:
   - **ID**: El id de la característica. Debe ser único dentro del archivo gfffile. Tenga en cuenta que para las características no contiguas el id puede aparecer en varias líneas. Se considera una característica única
   - **Name**: Nombre de la característica. No es necesario que sea único.
   - **Parent**: Id del elemento padre. Indica la relación «parte de». Un elemento puede tener varios padres. En este caso, los identificadores se separan por comas.
   - **Note**: nota de texto libre
   - ...

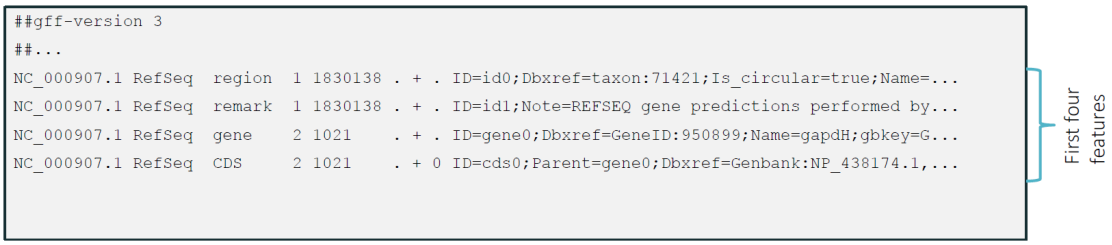
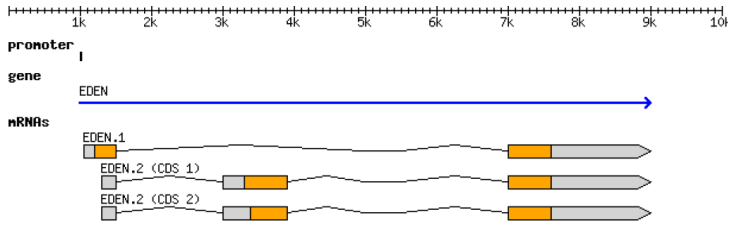


Figura I.3: *Primeras columnas de un fichero GFF.*

Hay varias cuestiones que deben tenerse en cuenta al analizar un archivo GFF: Las columnas sólo están separadas por tabuladores, pero se permiten espacios en blanco en algunos campos. Por tanto, hay que tener cuidado con grep y herramientas shell similares. Algunos caracteres se escapan en algunos campos, es decir, no queremos que los caracteres se interpreten como los caracteres que son. Esto ocurre con los puntos y coma y los espacios. Estos caracteres se escriben de otra forma; por ejemplo, un valor

de a en punto y coma debe codificarse con %3B. Para decodificar estos caracteres puede utilizar urllib. Además, una fila con triple almohadilla (###) indica que se han resuelto todas las características multilínea hasta ese punto. Hay al menos una de estas líneas al final del fichero. Las líneas que empiezan por # son comentarios. La región definida por los rasgos CDS incluye los codones de inicio y fin.



Example adapted from: https://github.com/The-Sequence-Ontology/Specifications/blob/master/gff3.md

```
1  ctg123 . gene            1000  9000  .  +  .  ID=gene00001;Name=EDEN
2  ctg123 . TF_binding_site 1000  1012  .  +  .  ID=tfbs00001;Parent=gene00001
3  ctg123 . mRNA            1050  9000  .  +  .  ID=mRNA00001;Parent=gene00001;Name=EDEN.1
4  ctg123 . mRNA            1300  9000  .  +  .  ID=mRNA00002;Parent=gene00001;Name=EDEN.2
5  ctg123 . exon            1300  1500  .  +  .  ID=exon00001;Parent=mRNA00002
6  ctg123 . exon            1050  1500  .  +  .  ID=exon00002;Parent=mRNA00001
7  ctg123 . exon            3000  3902  .  +  .  ID=exon00003;Parent=mRNA00002
8  ctg123 . exon            7000  9000  .  +  .  ID=exon00005;Parent=mRNA00001,mRNA00002
9  ctg123 . CDS             1201  1500  .  +  0  ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
10 ctg123 . CDS             7000  7600  .  +  0  ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
11 ctg123 . CDS             3301  3902  .  +  0  ID=cds00002;Parent=mRNA00002;Name=edenprotein.2
12 ctg123 . CDS             7000  7600  .  +  1  ID=cds00002;Parent=mRNA00002;Name=edenprotein.2
13 ctg123 . CDS             3391  3902  .  +  0  ID=cds00003;Parent=mRNA00002;Name=edenprotein.3
14 ctg123 . CDS             7000  7600  .  +  1  ID=cds00003;Parent=mRNA00002;Name=edenprotein.3
```

Figura I.4: *Ejemplo gráfico de una región genómica con el fichero GFF.*

Se puede trabajar con un fichero GFF desde la terminal de Linux. Por ejemplo, se pueden buscar todas las regiones (`grep region fichero.gff`) u obtener las terceras columnas (`cut -f3 fichero.gff`).

```python
def readGFF(file):
    """ Counts region types of a GFF file
        returns a dictionary """
    d = {}

    with open(file, 'r') as f:
        for linea in f:
            if linea[0] != "#":
                campos = linea.split("\t")
                region_type = campos[2]
                if region_type not in d: #o d.keys()
                    d[region_type] = 1
                else:
                    d[region_type] += 1
    return d

readGFF('data/haemophilus_influenzae/GCF_000027305.1_ASM2730v1_genomic.gff')
```

I.1.4. GenBank record

GenBank es una base de datos de secuencias de libre acceso mantenida por el National Center for Biotechnology Information (NCBI). Esta base de datos tiene un formato de archivo plano para representar un registro en la base de datos. Este formato codifica la secuencia, las características y las referencias de proteínas y genes.

```
LOCUS       NC_000907              1656 bp    DNA     linear   CON 02-AUG-2016
DEFINITION  Haemophilus influenzae Rd KW20 chromosome, complete genome.
ACCESSION   NC_000907 REGION: complement(36376..38031)
VERSION     NC_000907.1
DBLINK      BioProject: PRJNA57771
            Assembly: GCF_000027305.1
KEYWORDS    RefSeq.
SOURCE      Haemophilus influenzae Rd KW20
  ORGANISM  Haemophilus influenzae Rd KW20
            Bacteria; Proteobacteria; Gammaproteobacteria; Pasteurellales;
            Pasteurellaceae; Haemophilus.
```

Figura I.5: *Principio de un documento GenBank.*

Las partes de un documento GenBank (figura I.5) son:

- LOCUS: Contiene un identificador (NC_000907 en el ejemplo siguiente), la longitud de la secuencia (1656 pb), el tipo de molécula (ADN) y la fecha de la última modificación del registro.
- DEFINITION: Breve descripción de la secuencia.
- ORGANISM: Nombre científico del organismo .
- REFERENCE: Publicación científica relacionada con la secuencia. La última suele referirse al autor que envió la secuencia (esta referencia se marca con «direct submission» en lugar del título del artículo).
- FEATURES: Información sobre genes y productos génicos de la secuencia. Tiene varios subcampos:
  - Source: Campo obligatorio que describe la longitud y el organismo de la secuencia.
  - gene: región de un gen
  - CDS: región codificante de una proteína. Incluye translation, tabla, protein_id, producetname, notas, etc.

I.2. Librerías de Python

I.2.1. NumPy

NumPy (Numerical Python) es una librería de Python que permite realizar operaciones con arrays y matrices. También incluye herramientas de álgebra lineal, estadística o generación aleatoria de números. Se combina frecuentemente con otras librerías como SciPy para computaciones numéricas y pandas para análisis de datos.

I.2.1.1. Clase ndarray

Creación de arrays El elemento principal de NumPy es el objeto **ndarray** que representa arrays multidimensionales. Es capaz de realizar operaciones vectoriales eficientes (en velocidad y memoria). Al contrario que las listas de Python, todos los elementos de un ndarray son del mismo tipo. Los tipos posibles son int, float, boolean, string, etc. El atributo shape te da el tamaño del array y dtypet el tipo de los elementos.

```
import numpy as np

data = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
data.shape
#Output: (3, 3)
data.dtype
#Output: dtype('int64')
```

Se puede crear un array a partir de cualquier otra secuencia (listas u otros ndarrays), utilizando la función numpy.array. Es posible especificar en la construcción el tipo del array (dtype) y la dimensión. Las dimensiones son importantes para cuando haya que fusionar arrays, ya que deberán coincidir. Además, la dimensión debe coincidir en todo el array, es decir, debe ser homogéneo.

```
x = np.array([1, 2, 3], ndmin = 2)
# array([[1, 2, 3]])
y = np.array([1, 2], dtype = 'int32')

error = np.array([1, 2, 3], [4, 5])
```

Otras funciones para crear e inicializar arrays/matrices son:

- Matrices de ceros: numpy.zeros y numpy.zeros_like (el primero crea un array de las dimensiones dadas, el segundo crea un array como el que se le pasa, es decir, con sus dimensiones y el tipo).
- Matrices de unos: numpy.ones y numpy.ones_like
- Matriz identidad: numpy.eye y numpy.identity
- La función numpy.arange se comporta de forma similar a la función estándar de python range, pero devuelve un ndarray.
- La función numpy.linspace devuelve números espaciados uniformemente en un intervalo especificado. Se proporciona el primer y último valor y el número total de valores. La función numpy.logspace hace algo similar pero de forma logarítmica.

```
np.zeros((3,2))
#array([[0., 0.], [0., 0.], [0., 0.]])

x = np.array([1, 2, 3])
np.ones_like(x)
#array([1, 1, 1])
```

```
y = np.arange(10)
#[0 1 2 3 4 5 6 7 8 9]

np.linspace(-np.pi,np.pi,10)
#array([-3.14159265, -2.44346095, -1.74532925, -1.04719755, -0.34906585,
    0.34906585, 1.04719755, 1.74532925, 2.44346095, 3.14159265])
```

Operaciones básicas con arrays Las operaciones aritméticas pueden realizarse con ndarrays utilizando la misma sintaxis que la empleada para los escalares y evitando el uso de bucles explícitos.

```
data = np.array([[1, 2, 3], [4, 5, 6]])
data + 1
#array([[2, 3, 4], [5, 6, 7]])
data * 10
array([[10, 20, 30], [40, 50, 60]])
```

Arrays del mismo tamaño se pueden sumar y multiplicar celda a celda (si son de distinto tamaño no funciona):

```
data + data
#array([[ 2, 4, 6], [ 8, 10, 12]])
data * data
#array([[ 1, 4, 9], [16, 25, 36]])
```

Todas las operaciones booleanas (>, <, ==, etc) se aplican elemento a elemento y se devuelve un array de booleanos:

```
data = np.array([[1, 2, 3], [4, 5, 6], [4, 5, 6]])
#array[[1 2 3], [4 5 6], [4 5 6]]
data > 4
#array([[False, False, False], [False, True, True], [False, True, True]])
```

Ejercicios Construye una matriz de 5x5 elementos en la que los elementos de la diagonal sean 10 y los elementos fuera de la diagonal sean 5.

```
np.identity(5)*5+5
(np.identity(5)+1)*5

def diag2(n, f):
   ''' f: factor, n: dimensiones del array'''
   return (np.identity(n) + 1) * f
diag2(5,5)
```

Crea un array que contenga los días de la semana. ¿Cuál es su dtype?

```
weekdays_matrix = np.array(['Monday', 'Tuesday', 'Wednesday',
    'Thursday', 'Friday', 'Saturday', 'Sunday'])
print(weekdays_matrix.dtype)
#<U9
#El 9 es la longitud de palabra más grande del array
```

I.2.1.2. Indexación y slicing

Para arrays unidimensionales, el funcionamiento es como en el caso de las listas:

```
x = np.arange(10)
#array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
x[4]
#4
x[2:7]
#array([2, 3, 4, 5, 6])
x[-3:]
#array([7, 8, 9])
```

Si se asigna un valor a un slice, el valor se asigna a todas las posiciones en el slice.

```
x[4:7] = -1
#array([ 0, 1, 2, 3, -1, -1, -1, 7, 8, 9])
x = np.arange(10)
#array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
x[0::2] = [19,17,15,13,11]
#array([19, 1, 17, 3, 15, 5, 13, 7, 11, 9])
```

Ten en cuenta que si obtienes una referencia a un slice, los elementos no se copian y sólo se mantiene la referencia a los elementos del slice. Esto significa que si después utilizas el slice para modificar sus valores, éstos se modifican en el array original. Si queremos una copia del slice y no una referencia a los valores en los arrays originales, se puede utilizar el método copy().

```
x = np.ones((5,3), dtype=int)
#array([[1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1]])
y = x[:5:2,1:] #aquí se seleccionan primero filas y luego columnas
#array([[1, 1], [1, 1], [1, 1]])
y[:] = 25
print(x)
#array([[ 1, 25, 25], [ 1, 1, 1], [ 1, 25, 25], [ 1, 1, 1], [ 1, 25,
    25]])

x = np.ones((5,3), dtype=int)
z = x[:3].copy()
#array([0, 1, 2])
y[2] = 25
print(x)
#[0 1 2 3 4 5 6 7 8 9]
print(y)
```

```
#[ 0  1 25]
```

En arrays multidimensionales, se utilizan comas para separar los slices para cada dimensión.

```
data = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14,
    15, 16]])
#array([[ 1, 2, 3,  4],
# [ 5, 6,  7,  8],
#[ 9, 10, 11, 12],
#[13, 14, 15, 16]])
data[1,1:3]
#array([6, 7])
data[1:3]
#array([[ 5, 6, 7,  8],
# [ 9, 10, 11, 12]])
data[:, 1:3]
#array([[ 2, 3],
# [ 6, 7],
# [10, 11],
# [14, 15]])
data[1:4:2, 1:4:2]
#array([[ 6, 8],
# [14, 16]])
```

Indexación booleana Es posible utilizar un array de valores booleanos (Verdadero/Falso) para seleccionar elementos del ndarray.

```
data = np.random.randn(10, 3) #generación de matriz 10x3 con números de
    distribución normal
data[data>0] #números positivos
ix = data[:,0] > 0 #índices booleanos para las filas en las que el
    primer elemento es positivo
#array([ True, True, False, False, True, False, False, False, False,
    True])
selected = data[ix, :] #seleccionar las filas que empiezan por un número
    positivo
```

La indexación booleana siempre crea una copia, a diferencia del slicing. Es posible combinar varias operaciones booleanas utilizando & (and), | (or) y $\sim$ (negación). Además, se puede utilizar para asignar valores a los elementos de un array que cumplan con una condición.

```
data[(data[:, 0] > 0) & (data[:, 2] < 0), :] #selecciona las filas en
    las que el primer elemento es positivo y el último negativo
data[data < 0] = 0 #asigna el valor de 0 a todos los elementos negativos
```

Indexación fancy También es posible indexar utilizando matrices de enteros de forma similar al slicing. Estos enteros indican los índices de columna o fila a seleccionar. Este tipo de indexación es útil, por ejemplo, para seleccionar filas y columnas en un orden distinto al original.

```
nrows = 5
ncols = 3
p = 1
data = np.zeros((nrows, ncols))
for i in range(ncols):
    data[:, i] = np.arange(nrows)*p
    p*=10
#array([[ 0.,  0.,  0.],
#       [ 1., 10., 100.],
#       [ 2., 20., 200.],
#       [ 3., 30., 300.],
#       [ 4., 40., 400.]])
data[[3, 1]] #selección de las filas 3 y 1 en ese orden
#array([[ 3., 30., 300.],
#       [ 1., 10., 100.]])
data[:, [2, 0]]
#array([[ 0.,  0.],
#       [100.,  1.],
#       [200.,  2.],
#       [300.,  3.],
#       [400.,  4.]])
```

Ejercicios En el siguiente código, ¿qué diferencia supondría `y = 100 * y` con `y[:] = 100*y`?

```
x = np.array([1, 2, 3, 4, 5])
y = x[:3]
y = 100*y
print(y) # [100 200 300]
print(x) # [1 2 3 4 5]
y = x[:3]
y[:] = 100*y
print(y) # [100 200 300]
print(x) # [100 200 300 4  5]
```

De ambas formas, el resultado para y es igual; la diferencia se encuentra en el array original x. En la primera variante, x no se ve modificado, mientras que en la segunda variante, los valores de x sí se han visto modificados.

I.2.2. Matplotlib