Universidad Católica

San Pablo

Numpy

Python para Ciencia de Datos

Graciela Meza Lovon, Yessenia Yari Ramos, Alvaro Mamani Aliaga

Contenido

- Creación de arrays
- Segmentos
- Creación de arrays especiales
- Operaciones con Array
- Indexación Booleana

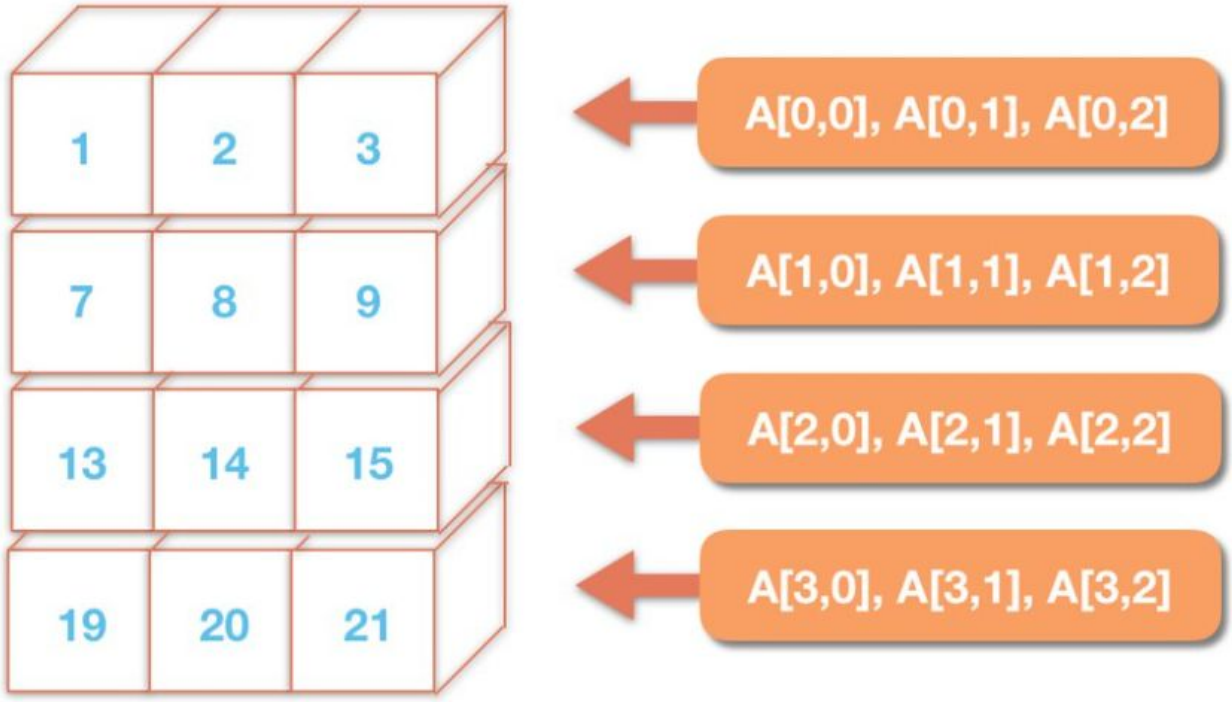
¿Qué es Numpy?

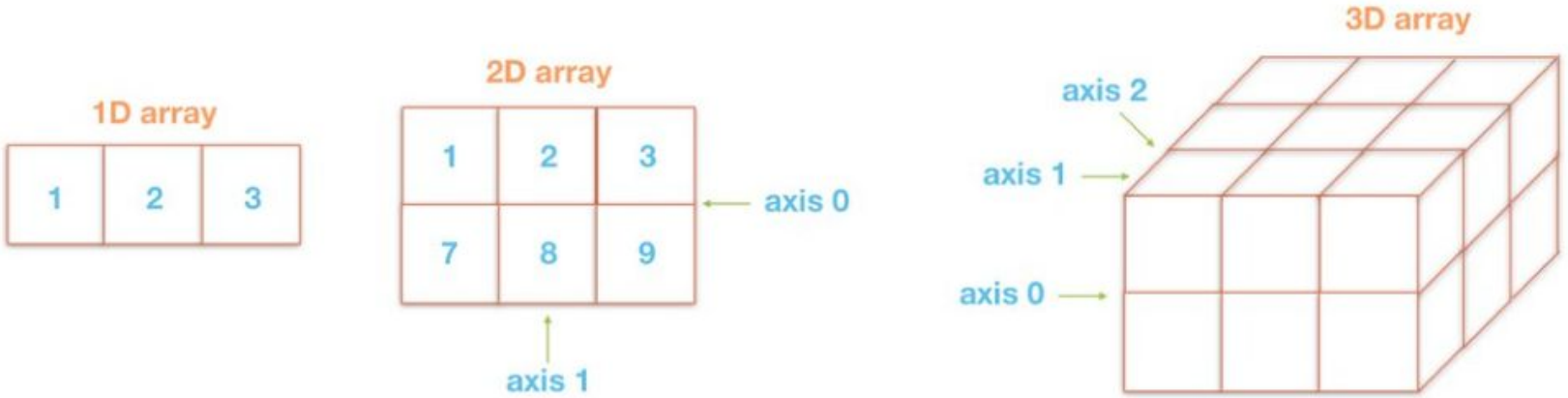
- NumPy es un paquete de Python que significa “Numerical Python”, es la librería principal para la informática científica, proporciona potentes estructuras de datos. Estas estructuras de datos garantizan cálculos eficientes con matrices.
- NumPy es una biblioteca para el lenguaje de programación Python que da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas.

¿Qué es Numpy?

- Proporciona una estructura de datos de matriz que tiene algunos beneficios sobre las listas regulares de Python:
  - ser más compacto,
  - acceder más rápido a leer y escribir artículos,
  - ser más conveniente y más eficiente.

A =


1
2
3
7
8
9
13
14
15
19
20
21
A[0,0], A[0,1], A[0,2]
A[1,0], A[1,1], A[1,2]
A[2,0], A[2,1], A[2,2]
A[3,0], A[3,1], A[3,2]


1D array
1
2
3
2D array
1
2
3
7
8
9
axis 0
axis 1
3D array
axis 2
axis 1
axis 0

Creación

- Importar la librería

```
import numpy
```

Creación

- Un array se puede crear explícitamente o a partir de una lista de la forma siguiente:

```
import numpy
x = numpy.array([2.0, 4.6, 9.3, 1.2])
print(type(x))
```

```
Salida:
<class 'numpy.ndarray'>
```

Creación

- Un array se puede crear explícitamente o a partir de una lista de la forma siguiente:

```
import numpy as np
x = np.array([2.0, 4.6, 9.3, 1.2])
print(type(x))
```

```
Salida:
<class 'numpy.ndarray'>
```

Acceso

- Mediante el operador corchete [] y un índice

```
import numpy as np
x = np.array([2.0, 4.6, 9.3, 1.2])
print(x[0])
```

```
Salida:
2.0
```

Longitud de un array de Numpy

- Mediante la función len()

```
import numpy as np
x = np.array([2.0, 4.6, 9.3, 1.2])
print(len(x))
```

Salida:
4

Segmentos

- Usando el operador corchete []

```
import numpy as np

numeros = np.array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])

print(numeros)
print(numeros[3:8])
print(numeros[:4])
print(numeros[5:])
print(numeros[:])
```

Segmentos

- Usando el operador corchete []

```
import numpy as np

numeros = np.array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])

print(numeros)
print(numeros[3:8])
print(numeros[:4])
print(numeros[5:])
print(numeros[:])
```

```
Salida:
[10 11 12 13 14 15 16 17 18 19]
[13 14 15 16 17]
[10 11 12 13]
[15 16 17 18 19]
[10 11 12 13 14 15 16 17 18 19]
```

Creación de arrays especiales

- La librería NumPy contiene una serie de funciones orientadas a generar arrays especiales como:
  - Componentes son todas nulas,
  - Componentes todas unos,
  - Array vacío a rellenar en un futuro.

Creación de arrays especiales

- La librería NumPy contiene una serie de funciones orientadas a generar arrays especiales como:
  - Componentes son todas nulas,
  - Componentes todas unos,
  - Array vacío a rellenar en un futuro.

```
ceros = np.zeros(5)
unos = np.ones(3)
vacio = np.empty(4)

print(ceros)
print(unos)
print(vacio)
```

```
Salida:
[0. 0. 0. 0. 0.]
[1. 1. 1.]
[2.  4.6 9.3 1.2]
```

Creación de un array 2D

- Un array se puede crear explícitamente o a partir de una lista de la forma siguiente:

```
import numpy as np
x = np.array([[2.0, 4.6],[9.3, 1.2]])
print(type(x))
print(x)
```

```
Salida:
<class 'numpy.ndarray'>
[[2.  4.6]
 [9.3 1.2]]
```

Acceso de un array 2D

- Mediante el operador corchete [] y un índice

```
import numpy as np
x = np.array([[2.0, 4.6],[9.3, 1.2]])
print(x[0][0])
```

```
Salida:
2.0
```

Longitud y dimensiones de un array 2D de Numpy

- Mediante la función len()
- Mediante el atributo shape

```
import numpy as np
x = np.array([[2.0, 4.6],[9.3, 1.2],[1.0, 0.6]])
print(len(x))
print(x.shape)
```

```
Salida:
3
(3,2)
```

Segmentos de un array 2D

- Usando el operador corchete []

```
import numpy as np
x = np.array([[2.0, 4.6],[9.3, 1.2],[1.0, 0.6]])
print("x =\n",x)
print("Segmento de x =\n",x[1:,:])
```

Segmentos de un array 2D

- Usando el operador corchete []

```
import numpy as np
x = np.array([[2.0, 4.6],[9.3, 1.2],[1.0, 0.6],])
print("x =\n",x)
print("Segmento de x =\n",x[1:,:])
```

```
Salida:
x =
 [[2.  4.6]
 [9.3 1.2]
 [1.  0.6]]
Segmento de x =
 [[9.3 1.2]
 [1.  0.6]]
```

Creación de arrays especiales de un array 2D

- La librería NumPy contiene una serie de funciones orientadas a generar arrays especiales como:
  - Componentes son todas nulas,
  - Componentes todas unos,
  - Array vacío a rellenar en un futuro.

```
ceros = np.zeros((5,4))
unos = np.ones((3,2))
vacio = np.empty((2,3))

print(ceros)
print(unos)
print(vacio)
```

Creación de arrays especiales de un array 2D

- La librería NumPy contiene una serie de funciones orientadas a generar arrays especiales como:
  - Componentes son todas nulas,
  - Componentes todas unos,
  - Array vacío a rellenar en un futuro.

```
ceros = np.zeros((5,4))
unos = np.ones((3,2))
vacio = np.empty((2,3))

print(ceros)
print(unos)
print(vacio)
```

```
Salida:
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
[[1. 1.]
 [1. 1.]
 [1. 1.]]
[[2.  4.6 9.3]
 [1.2 1.  0.6]]
```

Creación de arrays especiales de un array 2D

- La librería NumPy contiene una serie de funciones orientadas a generar arrays especiales como:
  - Componentes aleatorias.

```
alea = np.random.rand(2,3)
print(alea)
```

```
Salida:
[[0.70441314 0.95135037 0.99186517]
 [0.09036095 0.95426791 0.25205858]]
```

Operaciones con Arrays (Element-wise)

- Operaciones matemáticas
  - Suma
  - Resta
  - División
  - Exponenciación
  - Raíz cuadrada

Operaciones con Arrays (Element-wise)

- Operaciones matemáticas
  - Suma

```
a = np.array([1, 2, 3, 4])
b = np.array([1, 3, 1, 2])

print("Suma = \n",a+b)
```

```
Salida:
Suma =
 [2 5 4 6]
```

Operaciones con Arrays (Element-wise)

- Operaciones matemáticas
  - Resta

```
a = np.array([1, 2, 3, 4])
b = np.array([1, 3, 1, 2])

print("Resta = \n",a-b)
```

```
Salida:
Resta =
 [ 0 -1  2  2]
```

Operaciones con Arrays (Element-wise)

- Operaciones matemáticas
  - Multiplicación

```
a = np.array([1, 2, 3, 4])
b = np.array([1, 3, 1, 2])

print("Multiplicación = \n",a*b)
```

```
Salida:
Multiplicación =
 [1 6 3 8]
```

Operaciones con Arrays (Element-wise)

- Operaciones matemáticas
  - División

```
a = np.array([1, 2, 3, 4])
b = np.array([1, 3, 1, 2])

print("División = \n",a/b)
```

```
Salida:
División =
 [1.  0.66666667 3.         2.         ]
```

Operaciones con Arrays (Element-wise)

- Operaciones matemáticas
  - Exponenciación

```
a = np.array([1, 2, 3, 4])
b = np.array([1, 3, 1, 2])

print("Exponenciación = \n",a**b)
```

```
Salida:
Exponenciación =
 [ 1  8  3 16]
```

Operaciones con Arrays (Element-wise)

- Operaciones matemáticas
  - Raíz cuadrada

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

print("Raíz = \n", np.sqrt(a))
```

```
Salida:
Raíz =
[1. 1.41421356 1.73205081 2.   ]
```

Operaciones con Arrays 2D (Element - wise)

- Operaciones matemáticas vistas anteriormente también son válidas para arrays de 2 dimensiones
- Dadas dos matrices a y b, calcule la suma, resta, multiplicación y división de ambas.

$$a = \begin{bmatrix} 1 & 0 \\ -1 & 2 \\ 1 & 3 \end{bmatrix} \quad b = \begin{bmatrix} -3 & 3 \\ 2 & 0 \\ 7 & 16 \end{bmatrix}$$

Operaciones con Arrays 2D (Element - wise)

- Operaciones matemáticas vistas anteriormente también son válidas para arrays de 2 dimensiones
- También se puede utilizar las funciones:
  - np.add(a,b))
  - np.subtract(a,b))
  - np.divide(a,b))
  - np.multiply(a,b))
  - np.power(a,b))

Operaciones con Arrays 2D (Multiplicación)

- Multiplicación de matrices: operador @
- Cómo funciona la multiplicación de matrices?

Operaciones con Arrays 2D (Multiplicación)

- Multiplicación de matrices: operador @
- ¿Cómo funciona la multiplicación de matrices?

$$\begin{pmatrix} 1 & 2 \\ -3 & 0 \end{pmatrix} \cdot \begin{pmatrix} 3 & 5 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 1 \cdot 3 + 2 \cdot 4 & \end{pmatrix} = \begin{pmatrix} 11 & \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ -3 & 0 \end{pmatrix} \cdot \begin{pmatrix} 3 & 5 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 11 & 1 \cdot 5 + 2 \cdot 1 \end{pmatrix} = \begin{pmatrix} 11 & 7 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ -3 & 0 \end{pmatrix} \cdot \begin{pmatrix} 3 & 5 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 11 & 7 \\ -3 \cdot 3 + 0 \cdot 4 & \end{pmatrix} = \begin{pmatrix} 11 & 7 \\ -9 & \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ -3 & 0 \end{pmatrix} \cdot \begin{pmatrix} 3 & 5 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 11 & 7 \\ -9 & -3 \cdot 5 + 0 \cdot 1 \end{pmatrix} = \begin{pmatrix} 11 & 7 \\ -9 & -15 \end{pmatrix}$$

Operaciones con Arrays 2D (Multiplicación)

- Implemente la multiplicación de matrices

Transpuesta de una matriz

- Transpuesta de una matriz:
  - np.transpose(A)
  - A.T

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

Transpuesta de una matriz

- Transpuesta de una matriz:
  - np.transpose(A)
  - A.T

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

- Calcule la transpuesta de la matriz A

Cambio de las dimensiones de un array

- Mediante reshape()

```
a = np.array([1, 2, 3, 4, 5, 8])
b = a.reshape(2, 3)
c = a.reshape(3, 2)

print(b)
print(c)
```

```
Salida:

[[1 2 3]
 [4 5 8]]
[[1 2]
 [3 4]
 [5 8]]
```

Indexación Booleana

- Indexación Booleana permite seleccionar elementos basado en condiciones.
- Ejemplo

```
a = np.array([1, 2, 3, 4, 5, 8])
mask = np.array([True, True, False, False, False, True])
c = a[mask]
print(c)
```

Indexación Booleana

- Indexación Booleana permite seleccionar elementos basado en condiciones.
- Ejemplo 1:

```
a = np.array([1, 2, 3, 4, 5, 8])
mask = np.array([True, True, False, False, False, True])
c = a[mask]
print(c)
```

Salida:

```
[1 2 8]
```

Indexación Booleana

- Indexación Booleana permite seleccionar elementos basado en condiciones.
- Ejemplo 2:

```
a = np.array([1, 2, 3, 4, 5, 8])
mask = a > 4
c = a[mask]
print(c)
```

Salida:

```
[5 8]
```

Indexación con listas

- Indexación con lista permite seleccionar elementos basado en una lista.
- Ejemplo 4:

```
a = np.array([1, 2, 3, 4, 5, 8])
mask = [1,5,2]
c = a[mask]
print(c)
```

Salida:

```
[2 8 3]
```

¡Gracias!