

Módulo 2
Clase 2-2

Obtención y Preparación de Datos

Objetivos



- Creación de arreglos numpy
- Funciones preconstruidas de creación
- Métodos útiles de arreglos
- Indexación y selección
- Referencia y copia de un arreglo
- Operaciones entre arreglos
- Constantes y funciones matemáticas

Librería NumPy

- Es una Librería de Álgebra Lineal para Python.
- La mayoría de las librerías de Ciencia de Datos utiliza esta librería como base, de ahí su importancia.
- Es una librería increíblemente rápida, utiliza C.
- La distribución Anaconda la trae incluida
- Numpy es una de las más usadas librerías de ciencia de datos en Python.
- La interfaz de vectores y matrices es la mejor y más importante característica de Numpy.
- Pandas, TensorFlow y otras librerías usan Numpy internamente para ejecutar distintas operaciones

Arreglos Numpy

Arreglo Numpy

Vectores

Son arreglos en 1-D

Matrices

Son arreglos en 2-D

- Los arreglos pueden almacenar datos de forma muy compacta y son más eficientes para almacenar una gran cantidad de datos. Son excelentes para operaciones numéricas.

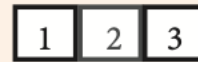
Arreglos Numpy

El módulo Numpy introduce en escena un nuevo tipo de objeto, ndarray (n dimensional array), caracterizado por:

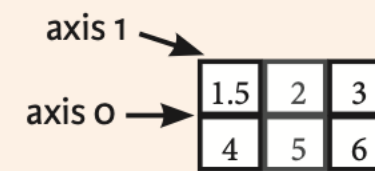
- Almacenamiento eficiente de colecciones de datos del mismo tipo
- Conjunto de métodos que permiten operar de forma vectorizada sobre sus datos

NumPy Arrays

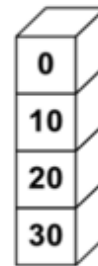
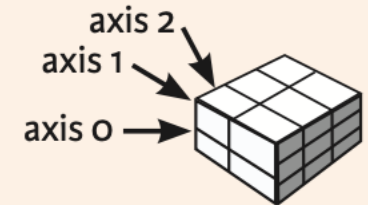
1D array



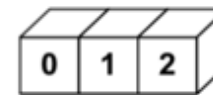
2D array



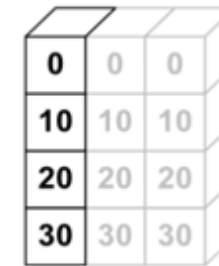
3D array



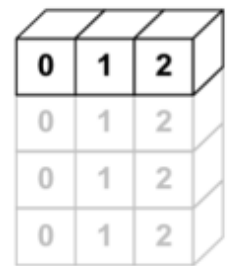
+





=



+



Arreglos Numpy

SciPy.org Docs NumPy v1.13 Manual NumPy Reference Routines index next previous

Mathematical functions

Trigonometric functions

<code>sin</code> (<code>x</code> , <code>/</code> , <code>out</code> , <code>where</code> , <code>casting</code> , <code>order</code> , ...)	Trigonometric sine, element-wise.
<code>cos</code> (<code>x</code> , <code>/</code> , <code>out</code> , <code>where</code> , <code>casting</code> , <code>order</code> , ...)	Cosine element-wise.
<code>tan</code> (<code>x</code> , <code>/</code> , <code>out</code> , <code>where</code> , <code>casting</code> , <code>order</code> , ...)	Compute tangent element-wise.
<code>arcsin</code> (<code>x</code> , <code>/</code> , <code>out</code> , <code>where</code> , <code>casting</code> , <code>order</code> , ...)	Inverse sine, element-wise.
<code>arccos</code> (<code>x</code> , <code>/</code> , <code>out</code> , <code>where</code> , <code>casting</code> , <code>order</code> , ...)	Trigonometric inverse cosine, element-wise.
<code>arctan</code> (<code>x</code> , <code>/</code> , <code>out</code> , <code>where</code> , <code>casting</code> , <code>order</code> , ...)	Trigonometric inverse tangent, element-wise.
<code>hypot</code> (<code>x1</code> , <code>x2</code> , <code>/</code> , <code>out</code> , <code>where</code> , <code>casting</code> , ...)	Given the "legs" of a right triangle, return its hypotenuse.
<code>arctan2</code> (<code>x1</code> , <code>x2</code> , <code>/</code> , <code>out</code> , <code>where</code> , <code>casting</code> , ...)	Element-wise arc tangent of <code>x1/x2</code> choosing the quadrant correctly.
<code>degrees</code> (<code>x</code> , <code>/</code> , <code>out</code> , <code>where</code> , <code>casting</code> , <code>order</code> , ...)	Convert angles from radians to degrees.
<code>radians</code> (<code>x</code> , <code>/</code> , <code>out</code> , <code>where</code> , <code>casting</code> , <code>order</code> , ...)	Convert angles from degrees to radians.
<code>unwrap</code> (<code>p</code> , <code>discont</code> , <code>axis</code>)	Unwrap by changing deltas between values to 2π complement.
<code>deg2rad</code> (<code>x</code> , <code>/</code> , <code>out</code> , <code>where</code> , <code>casting</code> , <code>order</code> , ...)	Convert angles from degrees to radians.
<code>rad2deg</code> (<code>x</code> , <code>/</code> , <code>out</code> , <code>where</code> , <code>casting</code> , <code>order</code> , ...)	Convert angles from radians to degrees.

Hyperbolic functions

<code>sinh</code> (<code>x</code> , <code>/</code> , <code>out</code> , <code>where</code> , <code>casting</code> , <code>order</code> , ...)	Hyperbolic sine, element-wise.
<code>cosh</code> (<code>x</code> , <code>/</code> , <code>out</code> , <code>where</code> , <code>casting</code> , <code>order</code> , ...)	Hyperbolic cosine, element-wise.
<code>tanh</code> (<code>x</code> , <code>/</code> , <code>out</code> , <code>where</code> , <code>casting</code> , <code>order</code> , ...)	Compute hyperbolic tangent element-wise.

Table Of Contents

- Mathematical functions
 - Trigonometric functions
 - Hyperbolic functions
 - Rounding
 - Sums, products, differences
 - Exponents and logarithms
 - Other special functions
 - Floating point routines
 - Arithmetic operations
 - Handling complex numbers
 - Miscellaneous

Previous topic
[numpy.not_equal](#)

Next topic
[numpy.sin](#)

<https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.math.html>

Creación de Arreglos Numpy

Para utilizar la librería numpy, debemos realizar la siguiente importación.

```
import numpy as np
```

Podemos crear un arreglo numpy a partir de una lista.

```
mi_lista = [1, 2, 3]  
mi_lista  
[1, 2, 3]
```

```
mi_np_array = np.array(mi_lista)  
mi_np_array  
array([1, 2, 3])
```

Nótese el tipo de datos del arreglo

```
print(type(mi_np_array))  
print(mi_np_array.dtype)  
  
<class 'numpy.ndarray'>  
int32
```

Creación de Arreglos Numpy

Creación de Arreglos Numpy

Ahora crearemos una matriz Numpy

```
# creacion de una matriz  
mi_matriz = [[1,2,3], [4,5,6], [7,8,9]]  
mi_matriz
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
mi_matriz_np = np.array(mi_matriz)  
mi_matriz_np
```

```
array([[1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9]])
```

Funciones Preconstruidas de Creación

A continuación crearemos un arreglo numpy con un rango de valores enteros, similar a la función range()

```
# rango de valores de 0 a 10  
arr1 = np.arange(10)  
print(arr1)
```

```
# rango de valores entre 10 y 20  
arr1 = np.arange(10,20)  
print(arr1)
```

```
# rango de valores entre 10 y 20 con incremento 2  
arr1 = np.arange(10,20,2)  
print(arr1)
```

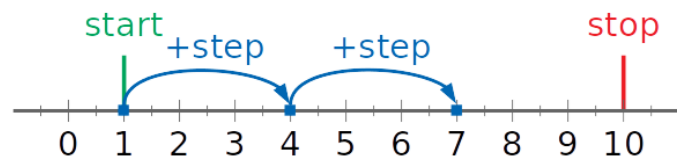
```
[0 1 2 3 4 5 6 7 8 9]  
[10 11 12 13 14 15 16 17 18 19]  
[10 12 14 16 18]
```

Arreglo con Rango de Valores

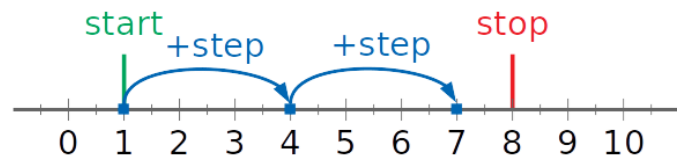


Note que no incluye el límite superior del intervalo

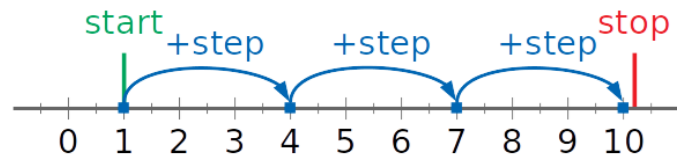
```
>>> np.arange(1, 10, 3)  
array([1, 4, 7])
```



```
>>> np.arange(1, 8, 3)  
array([1, 4, 7])
```



```
>>> np.arange(1, 10.1, 3)  
array([1., 4., 7., 10.])
```



Arreglo con Rango de Valores

Podemos rápidamente crear un vector o matriz numpy relleno con valores 0.

```
# arreglo de ceros
arr = np.zeros(3)
print(arr)

# matriz de ceros
arr = np.zeros((3,4))
print(arr)
```

```
[0. 0. 0.]
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

Lo mismo se puede hacer para crear un vector o matriz con valores 1

```
# arreglo de unos
arr = np.ones(3)
print(arr)

arr = np.ones((3,4))
print(arr)
```

```
[1. 1. 1.]
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

Matriz de Ceros y Unos



Recuerda que las tuplas (m,n) son inmutables

Podemos crear un arreglo con elementos distribuidos dentro de un intervalo

```
# vector con distribución de puntos  
arr = np.linspace(0,1,5)  
print(arr)
```

```
arr = np.linspace(0,10,20)  
print(arr)
```

```
[0.  0.25 0.5  0.75 1.  ]  
[ 0.  0.52631579  1.05263158  1.57894737  2.1052631  
6  2.63157895  
3.15789474  3.68421053  4.21052632  4.73684211  5.2631578  
9  5.78947368  
6.31578947  6.84210526  7.36842105  7.89473684  8.4210526  
3  8.94736842  
9.47368421 10.  ]
```

Vector con Distribución de Puntos

Podemos crear una matriz identidad, es decir, con valores 1 en la diagonal

```
# matriz identidad  
arr = np.eye(5)  
print(arr)
```

```
[[1. 0. 0. 0. 0.]  
 [0. 1. 0. 0. 0.]  
 [0. 0. 1. 0. 0.]  
 [0. 0. 0. 1. 0.]  
 [0. 0. 0. 0. 1.]]
```

Matriz Identidad

Arreglo con Valores Aleatorios

Podemos crear un vector o matriz con valores aleatorios

```
In [39]: np.random.rand(4)
```

```
Out[39]: array([ 0.12955727,  0.55090457,  0.82620168,  0.81458055])
```

```
In [40]: np.random.rand(3,3)
```

```
Out[40]: array([[ 0.35493984,  0.53348351,  0.38648226],  
               [ 0.77357578,  0.17288758,  0.16500478],  
               [ 0.24249277,  0.20543965,  0.47486082]])
```

Valor aleatorio entero

```
In [43]: np.random.randint(1,100)
```

```
Out[43]: 58
```

```
In [50]: np.random.randint(1,10,10)
```

```
Out[50]: array([1, 9, 6, 8, 2, 9, 9, 9, 1, 3])
```

Vector con valores aleatorios distribuidos de forma normal

```
In [47]: np.random.randn(4)
```

```
Out[47]: array([ 0.13753209, -0.69642849,  1.24177319, -0.61000545])
```


Métodos Útiles de Arreglos

Podemos cambiar la dimensionalidad de un vector o matriz

```
In [54]: arr = np.arange(0,25)
```

```
In [56]: arr
```

```
Out[56]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24])
```

```
In [58]: mtr = arr.reshape(5,5)
```

```
In [59]: mtr
```

```
Out[59]: array([[ 0,  1,  2,  3,  4],
 [ 5,  6,  7,  8,  9],
 [10, 11, 12, 13, 14],
 [15, 16, 17, 18, 19],
 [20, 21, 22, 23, 24]])
```

Consultamos la dimensionalidad de una matriz

Redimensionamiento de un Arreglo

¿Y si no cuadran los elementos con la dimensionalidad?

```
In [61]: mtr = arr.reshape(5,8)
```

```
-----  
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-61-142cdbdb375e> in <module>()  
----> 1 mtr = arr.reshape(5,8)  
  
ValueError: cannot reshape array of size 25 into shape (5,8)
```

Redimensionando un Arreglo



Las dimensionalidades deben ser consistentes

Máximo y Mínimo

```
In [64]: arr = np.random.randint(1,10,10)
```

```
In [66]: arr
```

```
Out[66]: array([5, 6, 5, 2, 1, 6, 9, 4, 4, 5])
```

```
In [68]: arr.max()
```

← Valor máximo

```
Out[68]: 9
```

```
In [70]: arr.argmax()
```

← Índice del Valor máximo

```
Out[70]: 6
```

```
In [72]: arr.min()
```

```
Out[72]: 1
```

```
In [73]: arr.argmin()
```

```
Out[73]: 4
```


Indexación y Selección

Así referenciamos un elemento del arreglo

```
In [3]: import numpy as np
```

```
In [5]: arr = np.arange(0,11)
```

```
In [7]: arr
```

```
Out[7]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
In [9]: arr[3]
```

```
Out[9]: 3
```

Índice 3

Posición 1

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

Índice 0

Índices y Posiciones de un Arreglo

Podemos seleccionar un subset de elementos del arreglo

```
In [10]: arr[3:5]
```

```
Out[10]: array([3, 4])
```

```
In [12]: arr[0:5]
```

```
Out[12]: array([0, 1, 2, 3, 4])
```

Podemos seleccionar desde y hasta un índice particular

```
In [16]: arr[5:]
```

```
Out[16]: array([ 5,  6,  7,  8,  9, 10])
```

```
In [17]: arr[:5]
```

```
Out[17]: array([0, 1, 2, 3, 4])
```

Selección de Rango

```
In [69]: arr2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

```
In [71]: arr2
```

```
Out[71]: array([[1, 2, 3],  
               [4, 5, 6],  
               [7, 8, 9]])
```

```
In [75]: arr2[1][1]
```

```
Out[75]: 5
```

```
In [77]: arr2[1,1]
```

```
Out[77]: 5
```

```
In [78]: arr2[1]
```

```
Out[78]: array([4, 5, 6])
```

Seleccionando una sección con la notación “slice”

```
In [80]: arr2[1:3,1:3]
```

```
Out[80]: array([[5, 6],  
               [8, 9]])
```

Columna 0
↓
array([[1, 2, 3],
 [4, 5, 6],
 [7, 8, 9]])
Fila 2 →

Selección en una Matriz

Podemos asignar valores a un elemento particular o bien a un rango de elementos del arreglo

```
In [24]: arr[0:5]=500
```

```
In [26]: arr
```

```
Out[26]: array([500, 500, 500, 500, 500,  5,  6,  7,  8,  9, 10])
```

```
In [28]: arr[3]=300
```

```
In [29]: arr
```

```
Out[29]: array([500, 500, 500, 300, 500,  5,  6,  7,  8,  9, 10])
```

Asignar Valores

Selección Condicional

```
In [83]: arr = np.arange(0,11)
```

```
In [85]: arr
```

```
Out[85]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
In [87]: arr_bool = arr > 5
```

```
In [89]: arr_bool
```

```
Out[89]: array([False, False, False, False, False, False,  True,
                True,  True,  True,  True], dtype=bool)
```

```
In [90]: arr[arr_bool]
```

```
Out[90]: array([ 6,  7,  8,  9, 10])
```



Sólo retorna los elementos que tienen valor True

Modo resumido

```
In [93]: arr[arr>5]
```

```
Out[93]: array([ 6,  7,  8,  9, 10])
```

Ejercicio

Dado el siguiente arreglo 2D, selección la submatriz que se indica.

```
# selección en una matriz numpy  
arr = np.arange(0,160,2).reshape(10,8)  
arr
```

```
array([[ 0,  2,  4,  6,  8, 10, 12, 14],  
       [16, 18, 20, 22, 24, 26, 28, 30],  
       [32, 34, 36, 38, 40, 42, 44, 46],  
       [48, 50, 52, 54, 56, 58, 60, 62],  
       [64, 66, 68, 70, 72, 74, 76, 78],  
       [80, 82, 84, 86, 88, 90, 92, 94],  
       [96, 98, 100, 102, 104, 106, 108, 110],  
       [112, 114, 116, 118, 120, 122, 124, 126],  
       [128, 130, 132, 134, 136, 138, 140, 142],  
       [144, 146, 148, 150, 152, 154, 156, 158]])
```

Referencia y Copia de un Arreglo

Extraigamos un slice y modifiquemos sus valores

```
# el problema de la referencia  
arr = np.arange(100,200,10)  
print(arr)
```

```
[100 110 120 130 140 150 160 170 180 190]
```

```
# tomamos un subset a partir del arreglo definido  
subset = arr[:5]  
print(subset)
```

```
[100 110 120 130 140]
```

```
# modificamos los elementos del subset  
subset[:] = 0  
print(subset)
```

```
[0 0 0 0 0]
```

```
# se modifica el arreglo original!!!!  
print(arr)
```

```
[ 0  0  0  0  0  0 150 160 170 180 190]
```

Referencias a un Arreglo Numpy



Al hacer un slice, se extrae una referencia al arreglo original y no una copia

Hay que expresamente utilizar una función para copiar un arreglo

```
# copia de un arreglo  
arr = np.arange(100,200,10)  
print(arr)
```

[100 110 120 130 140 150 160 170 180 190] ← La instrucción mágica

```
# copiamos el objeto  
subset = arr[:5].copy()  
print(subset)
```

[100 110 120 130 140]

```
# modificamos el subset  
subset[:] = 0  
print(subset)
```

[0 0 0 0 0]

```
# el arreglo original no es alterado  
print(arr)
```

[100 110 120 130 140 150 160 170 180 190]

Copiando un Arreglo

Operaciones entre Arreglos

Los arreglos numpy admiten operaciones aritméticas, las cuales son realizadas índice a índice.

```
arr1 = np.arange(1,20,2)
arr2 = np.arange(11,40,3)

print('Arreglo 1:',arr1,'\n')
print('Arreglo 2:',arr2,'\n')

# suma de arreglos
print('Adición:', arr1 + arr2, '\n')

# resta
print('Sustracción:',arr2 - arr1,'\n')

# mult
print('Multiplicación:',arr1*arr2,'\n')

# división
print('División:',arr1/arr2,'\n')
```

Arreglo 1: [1 3 5 7 9 11 13 15 17 19]

Arreglo 2: [11 14 17 20 23 26 29 32 35 38]

Adición: [12 17 22 27 32 37 42 47 52 57]

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

Multiplicación: [11 42 85 140 207 286 377 480 595 722]

División: [0.09090909 0.21428571 0.29411765 0.35 0.39130435 0.423
0.44827586 0.46875 0.48571429 0.5]

Operaciones entre Arreglos

De la misma manera, los arreglos numpy pueden ser operados con escalares, en donde cada elemento del arreglo será operado independientemente.

```
# operaciones con escalares
print('Adición Escalar:', arr1 + 10, '\n')

print('Sustracción Escalar:', arr1 - 10, '\n')

print('Multiplicación Escalar:', arr1 * 2, '\n')

print('División Escalar:', arr1 / 2, '\n')

print('Potencia:', arr1**3, '\n')
```

Adición Escalar: [11 13 15 17 19 21 23 25 27 29]

Sustracción Escalar: [-9 -7 -5 -3 -1 1 3 5 7 9]

Multiplicación Escalar: [2 6 10 14 18 22 26 30 34 38]

División Escalar: [0.5 1.5 2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5]

Potencia: [1 27 125 343 729 1331 2197 3375 4913 6859]

Operaciones con Escalares

Constantes y Funciones Matemáticas

Además de arrays, NumPy contiene también constantes y funciones matemáticas de uso cotidiano.

```
np.e
```

```
2.718281828459045
```

```
np.pi
```

```
3.141592653589793
```

```
np.log(2)
```

```
0.6931471805599453
```

```
lista = np.log([10., np.e, np.e**2, 0])  
print(lista)
```

```
[2.30258509 1.          2.         -inf]
```

Constantes y Funciones Matemáticas

```
arr = np.arange(10,20,2)
print('Arreglo:',arr,'\n')

# maximo
print('Valor máximo:',np.max(arr),'\n')

# minimo
print('Valor mínimo:',np.min(arr),'\n')

# sin
print('Función seno:',np.sin(arr),'\n')

# cos
print('Función coseno:',np.cos(arr),'\n')
```

Arreglo: [10 12 14 16 18]

Valor máximo: 18

Valor mínimo: 10

Función seno: [-0.54402111 -0.53657292 0.99060736 -0.28790332 -0.75098725]

Función coseno: [-0.83907153 0.84385396 0.13673722 -0.95765948 0.66031671]

Aplicando Funciones a un Arreglo


```
# sqrt
print(np.sqrt(arr), '\n')
```

```
# exp
print(np.exp(arr), '\n')
```

```
# Log
print(np.log(arr), '\n')
```

```
[0.          1.          1.41421356  1.73205081  2.          2.23606798
 2.44948974  2.64575131  2.82842712  3.          ]
```

```
[1.00000000e+00  2.71828183e+00  7.38905610e+00  2.00855369e+01
 5.45981500e+01  1.48413159e+02  4.03428793e+02  1.09663316e+03
 2.98095799e+03  8.10308393e+03]
```

```
[      -inf  0.          0.69314718  1.09861229  1.38629436  1.60943791
 1.79175947  1.94591015  2.07944154  2.19722458]
```

```
C:\Users\jsepulveda\Anaconda3\envs\dataanalysis\lib\site-packages\ipyke
rnel_launcher.py:8: RuntimeWarning: divide by zero encountered in log
```

Nótese que en este caso entrega un valor infinito negativo y levanta un warning al entregar el arreglo.

Aplicando Funciones a un Arreglo

The background of the slide features a grayscale photograph of a mountain range. The mountains are covered in dense, dark evergreen forests. A bright, white, diagonal line, possibly a snowfield or a path, runs across the upper right portion of the image. Overlaid on the lower-left side of the image is a solid teal-colored rectangular banner with rounded corners.

Dudas y consultas

Gracias