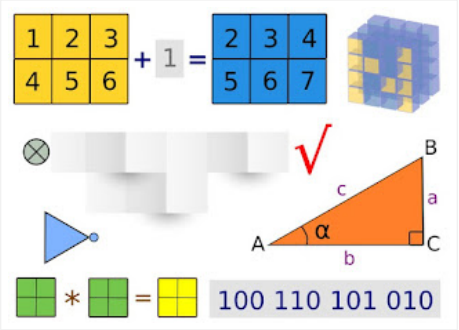
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

miércoles, 25 de diciembre de 2019

Cálculo con arrays Numpy



Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebida la biblioteca es una de sus principales virtudes si lo comparamos con los cálculos que pueden realizarse en Python con otras estructuras de datos como listas, tuplas y diccionarios.

Numpy permite realizar la misma operación aritmética con cada elemento de un array por separado y operar con arrays de igual o distintas dimensiones.

También cuenta con funciones matemáticas que abarcan distintas necesidades para operar con todos los elementos de un array o por ejes, sin olvidar el cálculo a nivel binario, estadístico y con números reales.

Operaciones aritméticas con un mismo valor

Con Numpy se puede operar un mismo valor con todos los elementos del array, elemento a elemento, utilizando los símbolos matemáticos: **+** (sumar), **-** (restar), ***** (multiplicar), **/** (dividir), **//** (dividir, cociente como entero), ****** (potenciación).

```
a = np.array([(1, 2, 3), (4, 5, 6)])
a = a + 1
print(a)

# [[2 3 4]
#  [5 6 7]]

a = np.array([(1, 2, 3), (4, 5, 6)])
a = a - 1
print(a)

[[0 1 2]
 [3 4 5]]

a = np.array([(1, 2, 3), (4, 5, 6)])
a = a * 5
print(a)

# [[ 5 10 15]
#  [20 25 30]]

a = np.array([(1, 2, 3), (4, 5, 6)])
a = a / 2
print(a)

# [[0.5 1.  1.5]
#  [2.  2.5 3. ]]
```

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

El módulo random
El módulo random de la librería estándar de Python incluye un conjunto de funciones

```
a = np.array([(1, 2, 3), (4, 5, 6)])
a = a // 2
print(a)

# [[0 1 1]
#  [2 2 3]]

a = np.array([(1, 2, 3), (4, 5, 6)])
a = a ** 2
print(a)

[[ 1  4  9]
 [16 25 36]]
```

Operaciones con arrays con las mismas dimensiones

Los mismos símbolos matemáticos (**+**, **-**, *, /, //, **) también se pueden utilizar para realizar operaciones aritméticas con dos arrays con las mismas dimensiones, elemento a elemento.

```
a = np.array([(1, 2, 3), (4, 5, 6)], dtype=np.int8)
b = np.ones((2, 3))
c = a + b
print(a)

# [[1 2 3]
#  [4 5 6]]

print(b)

# [[1. 1. 1.]
#  [1. 1. 1.]]

print(c)

# [[2. 3. 4.]
#  [5. 6. 7.]]

c = a - b
print(c)

# [[0. 1. 2.]
#  [3. 4. 5.]]
```

Operaciones con arrays de diferentes dimensiones

Es una característica avanzada de Numpy para realizar operaciones aritméticas con arrays de distintas dimensiones, elemento a elemento, utilizando los símbolos matemáticos (**+**, **-**, *, /, //, **). Estas operaciones son posibles cuando el array resultante tiene la misma dimensión que al menos uno de los arrays operados.

En el ejemplo siguiente el array **a** tiene 3 elementos por fila que coincide con el número de elementos del array **b**. Numpy para realizar el cálculo suma cada fila de **a** con los elementos de **b**, elemento a elemento:

```
a = np.ones((3, 3))
b = np.array([1, 2, 3])
c = a + b

#    a          b          c
# 1, 1, 1     1, 2, 3     2, 3, 4
# 1, 1, 1  +  1, 2, 3  =  2, 3, 4
# 1, 1, 1     1, 2, 3     2, 3, 4
```

En el ejemplo que sigue el array **a** tiene 3 elementos por columna que coincide con los elementos del array **b**. Numpy para realizar el cálculo suma cada columna de **a** con los elementos de **b**, elemento a elemento:

```
a = np.ones((3, 3))
b = np.array([[5], [10], [15]])

# 1, 1, 1     5,  5,  5    6,  6,  6
# 1, 1, 1  + 10, 10, 10  = 11, 11, 11
# 1, 1, 1    15, 15, 15    16, 16, 16
```

que permiten obtener de distintos modos números a...

Archivo

diciembre 2019 (2)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

Funciones matemáticas (para operar con arrays)

add()

Sumar dos arrays, elemento a elemento.

```
a = np.array([(1, 2, 3), (4, 5, 6)], dtype=np.int8)
b = np.ones((2, 3))
c = np.add(a, b)
print(c)

# [[2. 3. 4.]
#  [5. 6. 7.]]
```

subtract()

Restar dos arrays, elemento a elemento.

```
a = np.array([(1, 2, 3), (4, 5, 6)], dtype=np.int8)
b = np.ones((2, 3))
c = np.subtract(a, b)
print(c)

# [[0. 1. 2.]
#  [3. 4. 5.]]
```

multiply()

Multiplicar dos arrays, elemento a elemento.

```
a = np.array([(1, 2, 3), (4, 5, 6)], dtype=np.int8)
b = np.array([(2, 2, 2), (3, 3, 3)], dtype=np.int8)
c = np.multiply(a, b)
print(c)

# [[ 2  4  6]
#  [12 15 18]]
```

dot()

Multiplicar un array por un valor, elemento a elemento.

```
a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.dot(a, 4)
print(b)

# [[ 4  8 12]
#  [16 20 24]]
```

Producto de dos arrays

En la teoría de matrices dos arrays se pueden multiplicar si el número de columnas de la primera coincide con el número de filas de la segunda.

El producto de dos arrays se realiza utilizando el operador **@** (a partir de Python 3.5) o con la función **np.dot()** de Numpy.

```
a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.ones((3, 2))
c = a @ b
print(c)

# [[ 6.  6.]
#  [15. 15.]]

c = a.dot(b)
print(c)

# [[ 6.  6.]
#  [15. 15.]]
```

divide()

Dividir dos arrays, elemento a elemento.

```
a = np.array([(1, 2, 3), (4, 5, 6)], dtype=np.int8)
b = np.array([(1, 1, 1), (2, 2, 2)], dtype=np.int8)
c = np.divide(a, b)
print(c)

# [[1.  2.  3. ]
#  [2.  2.5 3. ]]
```

mod()

Dividir dos matrices y obtener el resto de la división, elemento a elemento.

```
a = np.array([(1, 2, 3), (4, 5, 6)], dtype=np.int8)
b = np.array([(1, 1, 1), (2, 2, 2)], dtype=np.int8)
c = np.mod(a, b)
print(c)

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

divmod()

Dividir dos arrays y obtener el cociente y el resto de la división, elemento a elemento.

```
a = np.array([(1, 2, 3), (4, 5, 6)], dtype=np.int8)
b = np.array([(1, 1, 1), (2, 2, 2)], dtype=np.int8)
c = np.divmod(a, b)
print(c)

# Se obtiene una tupla con 2 arrays para cocientes y restos:
#
# (array([[1, 2, 3],
#         [2, 2, 3]], dtype=int8),
#  array([[0, 0, 0],
#         [0, 1, 0]], dtype=int8))
```

negative()

Cambiar el signo a los valores de un array.

```
a = np.array([(1.5, -2, 3.1), (-4.8, 5.4, -6.6)])
b = np.negative(a)
print(b)

# [[-1.5  2.  -3.1]
#  [ 4.8 -5.4  6.6]]
```

rint()

Redondear todos los valores del array al entero más próximo.

```
a = np.array([(1.5, 2, 3.1), (4.8, 5.4, 6.6)])
print(a)

# [[1.5 2.  3.1]
#  [4.8 5.4 6.6]]

b = np.rint(a)
print(b)

# [[2. 2. 3.]
#  [5. 5. 7.]]
```

sqrt()

Calcular la raíz cuadrada, elemento a elemento.

```
a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.sqrt(a)
print(b)

# [[1.  1.41421356 1.73205081]
#  [2.  2.23606798 2.44948974]]
```

exp()

Calcular la exponencial, elemento a elemento

```
a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.exp(a)
print(b)

# [[  2.71828183   7.3890561   20.08553692]
#  [ 54.59815003 148.4131591  403.42879349]]
```

power()

Elevar los elementos a la potencia de un número.

```
# Elevar los elementos a la potencia de un número, elemento
# a elemento.

a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.power(a, 5)
print(b)

# [[   1   32  243]
#  [1024 3125 7776]]

# Elevar los elementos de un array a los valores de
# otro array, elemento a elemento.

a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.array([(2, 2, 2), (3, 3, 3)])
c = np.power(a, b)
print(c)

# [[  1   4   9]
#  [ 64 125 216]]
```

sin()

Aplicar la función de seno a un array, elemento a elemento.

```
a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.sin(a)
print(b)

# [[ 0.84147098  0.90929743  0.14112001]
#  [-0.7568025  -0.95892427 -0.2794155 ]]
```

cos()

Aplicar la función de coseno a un array, elemento a elemento.

```
a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.cos(a)
print(b)

# [[ 0.54030231 -0.41614684 -0.9899925 ]
#  [-0.65364362  0.28366219  0.96017029]]
```

tan()

Aplicar la función de tangente, elemento a elemento.

```
a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.tan(a)
print(b)
```

```
# [[ 1.55740772 -2.18503986 -0.14254654]
#  [ 1.15782128 -3.38051501 -0.29100619]]
```

log()

Calcular el logaritmo de los elementos del array, elemento a elemento.

```
a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.log(a)
print(b)

# [[0.         0.69314718 1.09861229]
#  [1.38629436 1.60943791 1.79175947]]
```

Funciones matemáticas (para operar por ejes)

add.reduce()

Obtener array con la suma de los elementos por ejes.

```
# Obtener array con la suma de los elementos de las filas (eje 0).

a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.add.reduce(a, axis=0)  # axis se puede omitir en este caso
print(b)

# [5 7 9]

# Obtener de un array 1D la suma de todos sus elementos (eje 0).

a = np.array([1, 2, 3, 4, 5, 6])
b = np.add.reduce(a)   # Es equivalente a np.sum(a)
print(b)

# 21

# Obtener array con la suma de los elementos de las columnas (eje 1).

a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.add.reduce(a, axis=1)
print(b)

# [ 6 15]

# Obtener la suma de todos los elementos (eje 0 y eje1).

a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.add.reduce(a, None)  # Es equivalente a np.sum(a)
print(b)

# 21
```

multiply.reduce()

Obtener array con la multiplicación de los elementos de las filas (eje 0).

```
a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.multiply.reduce(a, axis=0)
print(b)

# [ 4 10 18]
```

add.accumulate()

Obtener array con acumulados de sumas de filas (eje 0).

```
a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.add.accumulate(a, axis=0)
print(b)

# [[1 2 3]
#  [5 7 9]]
```

multiply.accumulate()

Obtener array con acumulados de multiplicaciones de columnas (eje 1).

```
a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.multiply.accumulate(a, axis=1)
print(b)

# [[  1   2   6]
#  [  4  20 120]]
```

add.reduceat()

Obtener array con acumulados de sumas parciales.

```
a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Suma desde el elemento 0 al 5 y desde el 6 al final.

b = np.add.reduceat(a, [0, 5])
print(b)

# [15 40]
```

multiply.reduceat()

Obtener array con acumulados de multiplicaciones parciales eje 1.

```
a = np.arange(20).reshape(5, 4)
b = np.multiply.reduceat(a, [0,2], axis=1)
print(a)

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

print(b)

# [[  0   6]
#  [ 20  42]
#  [ 72 110]
#  [156 210]
#  [272 342]]
```

add.outer()

Obtener array 2D a partir de sumar dos arrays 2D creados con datos repetidos de 2 vectores.

En el primer array se repiten los datos de **a** por columnas y en el segundo los de **b** por filas.

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = np.add.outer(a, b)
print(a)

# [1 2 3]

print(b)

# [4 5 6]

print(c)

# [[5 6 7]
#  [6 7 8]
#  [7 8 9]]

#  1  1  1     4  5  6     5  6  7
#  2  2  2  +  4  5  6  =  6  7  8
#  3  3  3     4  5  6     7  8  9
```

Obtener array 4D a partir de sumar arrays 2D creados con datos repetidos de 2 arrays 2D.

En cada suma en el primer array se repiten los datos de una fila del array **a** por columnas y en el segundo los datos de una fila del array **b** por filas.

```
a = np.repeat([[1,2,3]], 3, axis=0)
b = np.arange(9).reshape(3, 3)
c = np.add.outer(a, b)
print(c.ndim)

# 4

print(a)

# [[1 2 3]
#  [1 2 3]
#  [1 2 3]]

print(b)

# [[0 1 2]
#  [3 4 5]
#  [6 7 8]]

print(c)

# [[[[ 1  2  3]
#    [ 4  5  6]
#    [ 7  8  9]]
#
#   [[ 2  3  4]
#    [ 5  6  7]
#    [ 8  9 10]]
#
#   [[ 3  4  5]
#    [ 6  7  8]
#    [ 9 10 11]]]
#
#  [[[ 1  2  3]
#    [ 4  5  6]
#    [ 7  8  9]]
#
#   [[ 2  3  4]
#    [ 5  6  7]
#    [ 8  9 10]]
#
#   [[ 3  4  5]
#    [ 6  7  8]
#    [ 9 10 11]]]
#
#  [[[ 1  2  3]
#    [ 4  5  6]
#    [ 7  8  9]]
#
#   [[ 2  3  4]
#    [ 5  6  7]
#    [ 8  9 10]]
#
#   [[ 3  4  5]
#    [ 6  7  8]
#    [ 9 10 11]]]]
```

multiply.outer()

Obtener array 2D a partir de multiplicar dos arrays 2D creados con datos repetidos de 2 vectores.

En el primer array se repiten los datos de **a** por columnas y en el segundo los de **b** por filas.

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = np.multiply.outer(a, b)
print(a)

# [1 2 3]

print(b)

# [4 5 6]
```

```
print(c)

# [[ 4  5  6]
#  [ 8 10 12]
#  [12 15 18]]

#  1  1  1     4  5  6     4  5  6
#  2  2  2  x  4  5  6  =  8 10 12
#  3  3  3     4  5  6    12 15 18
```

add.at()

Modificar el array sumando un valor solo a los elementos indicados.

```
a = np.array([1, 2, 3])
np.add.at(a, [0, 2], 5)
print(a)

# [6, 2, 8]
```

multiply.at()

Modificar el array multiplicando por un valor solo elementos que cumplen una condición.

```
a = np.array([10, 67, 35, 55])
np.multiply.at(a, (a*2 < 128), 2)
print(a)

# [ 20  67  70 110]
```

power.at()

Modificar el array elevando a la potencia de un número solo los elementos indicados.

```
a = np.array([1, 2, 3])
np.power.at(a, [1, 2], 3)
print(a)

# [ 1  8 27]
```

Operaciones binarias

bitwise_and()

Calcular el producto lógico a nivel binario.

```
# Calcular el producto lógico a nivel binario de dos números.
a = np.bitwise_and(6, 3)
print(a)

# 2

# Operación en binario (AND):
# 6 -> 110
# 3 -> 011
#     -----
#      010 -> 2

# Calcular el producto lógico de los elementos de un
# array con un número, elemento a elemento.

a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.bitwise_and(a, 3)
print(b)

# [[1 2 3]
#  [0 1 2]]

# Calcular producto lógico a nivel binario de dos
# arrays, elemento a elemento.

a = np.array([(1, 2, 3), (4, 5, 6)])
```

```
b = np.array([(1, 0, 1), (0, 1, 7)])
c = np.bitwise_and(a, b)
print(c)

# [[1 0 1]
#  [0 1 6]]

# Operación en binario (AND):
# 001 010 011 100 101 110
# 001 000 001 000 001 111
# --- --- --- --- --- ---
# 001 000 001 000 001 110
#   1   0   1   0   1   6
```

bitwise_or()

Calcular suma lógica a nivel binario.

```
# Calcular suma lógica a nivel binario de dos
# arrays, elemento a elemento.

a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.array([(1, 0, 1), (0, 1, 7)])
c = np.bitwise_or(a, b)
print(c)

# [[1 2 3]
#  [4 5 7]]

# Operación en binario (OR):
# 001 010 011 100 101 110
# 001 000 001 000 001 111
# --- --- --- --- --- ---
# 001 010 011 100 101 111
#   1   2   3   4   5   7
```

bitwise_xor()

Calcular OR exclusivo (XOR).

```
# Calcular OR exclusivo (XOR) de dos arrays,
# elemento a elemento.

a = np.array([(1, 2, 3), (4, 5, 6)])
b = np.array([(1, 0, 1), (0, 1, 7)])
c = np.bitwise_xor(a, b)
print(c)

# [[0 2 2]
#  [4 4 1]]

# Operación en binario (XOR):
# 001 010 011 100 101 110
# 001 000 001 000 001 111
# --- --- --- --- --- ---
# 000 010 010 100 100 001
#   0   2   2   4   4   1
```

invert()

Calcular inverso (NOT).

```
# Calcular inverso (NOT) de un número

valor = 25
a = np.invert(np.array(valor, dtype=np.uint8))
print(a)

# 230

#  25 -> 00011001
# 230 -> 11100110

# Calcular inversos (NOT) de un array de booleanos.

a = np.array([True, False, True])
```

```
b = np.invert(a, dtype=bool)
print(b)

# [False  True False]
```

binary_repr()

Expresar en binario el resultado de una operación binaria.

```
a = np.bitwise_and(6, 3)
b = np.binary_repr(a, width=8)
print(a)

# 2

print(b)

# 00000010
```

Cálculo Estadístico

max()

Obtener el valor máximo.

```
# Obtener valor máximo del array.

a = np.array([1, 2, 3, 4, 5])
maximo = a.max()
print(maximo)

# 5

# Obtener valor máximo del array.

a = np.array([[1, 2, 3], [4, 5, 6]])
maximo = a.max()
print(maximo)

# 6

# Obtener array con valores máximos por filas (eje 0).

a = np.array([[1, 2, 3], [4, 5, 6]])
maximos = a.max(axis=0)
print(maximos)

# [4, 5, 6]
```

argmax()

Obtener índices de los elementos con valores máximos.

```
# Obtener el índice del elemento con el valor máximo.

a = np.array([[1, 2, 3], [4, 5, 6]])
indice = a.argmax()
print(indice)

# 5

# Obtener array con índices de elementos con valor máximo
# por filas (eje 0).

a = np.array([[1, 2, 3], [3, 4, 1]])
indices = a.argmax(axis=0)
print(indices)

# [1 1 0]
```

min()

Obtener el valor mínimo.

```
# Obtener el valor mínimo del array.

a = np.array([1, 2, 3, 4, 5])
minimo = a.min()
print(minimo)

# 1

# Obtener el valor mínimo del array.

a = np.array([[1, 2, 3], [4, 5, 6]])
minimo = a.min()
print(minimo)

# 1

# Obtener array con valores mínimos por columnas (eje 1).

a = np.array([[1, 2, 3], [4, 5, 6]])
minimos = a.min(axis=1)
print(minimos)

# [1 4]
```

ptp()

Obtener la diferencia entre el valor máximo y el mínimo.

```
# Obtener la diferencia entre el valor máximo y el mínimo.

a = np.array([[1, 2, 3], [4, 5, 6]])
diferencia = a.ptp()
print(diferencia)

# 5

# Obtener las diferencias entre el valor máximo y el
# mínimo por filas (eje 0).

a = np.array([[1, 2, 3], [4, 5, 7]])
diferencias = a.ptp(axis=0)
print(diferencias)

# [3 3 4]
```

clip()

Limitar los valores mínimos y máximos.

```
# Limitar los valores mínimos y máximos de un array
# a 3 y 6, respectivamente.

a = np.array([(1, 3, 5, 7, 3, 1)])
a = a.clip(3, 6)
print(a)

# [[3 3 5 6 3 3]]
```

round()

Obtener array redondeando los valores un número de decimales.

```
# Obtener array redondeando los valores a 1 la parte decimal.

a = np.array([1.51, 3.694, 5.44, 7.211, 3.9, 1.555])
b = a.round(1)
print(b)

# [1.5 3.7 5.4 7.2 3.9 1.6]
```

trace()

Obtener la suma de los valores de la diagonal del array.

```
a = np.array([(2, 4), (5, 6)])
total = a.trace()
print(total)

# 8
```

sum()

Obtener la suma de todos los valores del array.

```
a = np.array([(1, 2, 3), (4, 5, 6)])
total = a.sum()
print(total)

# 21
```

cumsum()

Obtener array con las sumas acumuladas de los elementos.

```
a = np.array([(10, 20, 30), (10, 10, 10)])
b = a.cumsum()
print(b)

# [10 30 60 70 80 90]
```

mean()

Obtener la media aritmética.

```
# Obtener la media aritmética de los elementos del array.

a = np.array([(1, 2, 3), (4, 5, 6)])
media = a.mean()
print(media)

# 3.5

# Obtener las medias aritméticas por filas (eje 0).

a = np.array([(1, 2, 3), (4, 5, 6)])
medias = a.mean(axis=0)
print(medias)

# [2.5 3.5 4.5]
```

var()

Calcular la varianza de los elementos del array.

```
a = np.array([(1, 2, 3), (4, 5, 6)])
varianza = a.var()
print(varianza)

# 2.9166666666666665
```

std()

Calcular la desviación típica.

```
a = np.array([(1, 2, 3), (4, 5, 6)])
desviacion = a.std()
print(desviacion)

# 1.707825127659933
```

prod()

Calcular el producto de todos los valores del array.

```
a = np.array([(1, 2, 3), (4, 5, 6)])
producto = a.prod()
print(producto)

# 720
```

cumprod()

Obtener array con las multiplicaciones acumuladas de los valores del array.

```
a = np.array([(1, 2, 3), (4, 5, 6)])
total = a.cumprod()
print(total)

# [  1   2   6  24 120 720]
```

median()

Calcular la mediana.

```
# Calcular la mediana con todos los valores del array.

a = np.array([(1, 2, 3), (4, 5, 6)])
mediana = np.median(a)
print(mediana)

# 3.5

# Calcular las medianas por columnas (eje 1)

a = np.array([(1, 2, 3), (4, 5, 6)])
medianas = np.median(a, axis=1)
print(medianas)

# [2. 5.]
```

Funciones para números reales (float)

floor()

Obtener un array con los enteros inmediatamente anteriores (suelo).

```
a = np.array([-1.1, 0.5, 1.9, 2.1, 2.9, 3])
b = np.floor(a)
print(b)

# [-2.  0.  1.  2.  2.  3.]
```

ceil()

Obtener un array con los enteros inmediatamente posteriores (techo).

```
a = np.array([-1.1, 0.5, 1.9, 2.1, 2.9, 3])
b = np.ceil(a)
print(b)

# [-1.  1.  2.  3.  3.  3.]
```

trunc()

Obtener un array con valores truncados (parte entera).

```
a = np.array([-1.1, 0.5, 1.9, 2.1, 2.9, 3])
b = np.trunc(a)
print(b)

# [-1.  0.  1.  2.  2.  3.]
```

Consultar más funciones para calcular con Numpy

Publicado por Pherkad en 9:36

Etiquetas: Numpy

Entrada más reciente Inicio Entrada antigua