CONQUER BLOCKS

PYTHON

TRABAJANDO CON ARRAYS (PARTE 2)

CLASE ANTERIOR

1) Convertir Listas en Arrays

2) Multidimensionalidad en los Arrays

3) Crear Arrays sin usar Listas

4) Crear Arrays unidad

5) Reasignar el contenido de los Arrays

5) Ordenar el contenido de los Arrays

CREACION DE UN ARRAY

```
import numpy as np

a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
print(a)
```

✓ 0.0s

```
[[2 2 2]
 [2 2 2]
 [2 2 2]]
```

```
import numpy as np

b = np.arange(1,10)
print(b)
b = b.reshape((3,3))
print(b)
```

✓ 0.0s

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

CREACION DE UN ARRAY

```
import numpy as np

a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
print(a)
```

✓ 0.0s

```
[[2 2 2]
 [2 2 2]
 [2 2 2]]
```

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
print(b)
```

✓ 0.0s

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

LLENAR ARRAYS DE VALORES

fill()

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a.fill(8)
print(a)
```

✓ 0.0s

```
[[8 8 8]
 [8 8 8]
 [8 8 8]]
```

operador de asignación

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a += 6
print(a)
```

✓ 0.0s

```
[[8 8 8]
 [8 8 8]
 [8 8 8]]
```

LLENAR ARRAYS DE VALORES

fill()

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a.fill(8)
print(a)
```

✓ 0.0s

```
[[8 8 8]
 [8 8 8]
 [8 8 8]]
```

operador de asignación

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a -= 6
print(a)
```

✓ 0.0s

```
[[-4 -4 -4]
 [-4 -4 -4]
 [-4 -4 -4]]
```

LLENAR ARRAYS DE VALORES

fill()

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a.fill(8)
print(a)
```

✓ 0.0s

```
[[8 8 8]
 [8 8 8]
 [8 8 8]]
```

operador de asignación

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a *= 6
print(a)
```

✓ 0.0s

```
[[12 12 12]
 [12 12 12]
 [12 12 12]]
```

LLENAR ARRAYS DE VALORES

fill()

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a.fill(8)
print(a)
```

✓ 0.0s

```
[[8 8 8]
 [8 8 8]
 [8 8 8]]
```

operador de asignación

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a /= 6
print(a)
```

⊗ 0.2s

```
---------------------------------------------------------------------------
UFuncTypeError                            Traceback (most recent call last)
Cell In[10], line 4
      2 a = np.zeros((3,3), dtype = np.int64)
      3 a[:] = 2
----> 4 a /= 6
      5 print(a)

UFuncTypeError: Cannot cast ufunc 'divide' output from dtype('float64') to dtype('int64') with casting rule 'same_kind'
```

LLENAR ARRAYS DE VALORES

fill()

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a.fill(8)
print(a)
```

✓ 0.0s

```
[[8 8 8]
 [8 8 8]
 [8 8 8]]
```

operador de asignación

```
import numpy as np
a = np.zeros((3,3))
a[:] = 2
a /= 6
print(a)
```

✓ 0.0s

```
[[0.33333333 0.33333333 0.33333333]
 [0.33333333 0.33333333 0.33333333]
 [0.33333333 0.33333333 0.33333333]]
```

SUMAR ELEMENTOS DE UN ARRAY

sum()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
suma_array = b.sum()
print(b)
print(suma_array)
```

✓ 0.0s

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
45
```

suma de todos los elementos del array

sum(0)

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
suma_array = b.sum(axis = 0)
print(b)
print(suma_array)
```

✓ 0.0s

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[12 15 18]
```

suma de las columnas del array

sum(1)

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
suma_array = b.sum(axis = 1)
print(b)
print(suma_array)
```

✓ 0.0s

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[ 6 15 24]
```

suma de las filas del array

SUMAR ELEMENTOS DE UN ARRAY

sum()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
suma_array = b.sum()
print(b)
print(suma_array)
```

✓ 0.0s

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
45
```

suma de todos los elementos del array

sum(0)

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
suma_array = b.sum(0)
print(b)
print(suma_array)
```

✓ 0.0s

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[12 15 18]
```

suma de las columnas del array

sum(1)

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
suma_array = b.sum(1)
print(b)
print(suma_array)
```

✓ 0.0s

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[ 6 15 24]
```

suma de las filas del array

MULTIPLICAR ELEMENTOS DE UN ARRAY

prod()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
prod_array = b.prod()
print(b)
print(prod_array)
```

✓ 0.0s

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
362880
```

multiplica todos los elementos del array

prod(0)

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
prod_array = b.prod(axis = 0)
print(b)
print(prod_array)
```

✓ 0.0s

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[ 28  80 162]
```

multiplica las columnas del array

prod(1)

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
prod_array = b.prod(axis = 1)
print(b)
print(prod_array)
```

✓ 0.0s

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[  6 120 504]
```

multiplica las filas del array

MULTIPLICAR ELEMENTOS DE UN ARRAY

prod()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
prod_array = b.prod()
print(b)
print(prod_array)
```

✓ 0.0s

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
362880
```

multiplica todos los elementos del array

prod(0)

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
prod_array = b.prod(0)
print(b)
print(prod_array)
```

✓ 0.0s

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[ 28  80 162]
```

multiplica las columnas del array

prod(1)

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
prod_array = b.prod(1)
print(b)
print(prod_array)
```

✓ 0.0s

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[  6 120 504]
```

multiplica las filas del array

MÁXIMO MÍNIMO Y VALOR MEDIO DE LOS ELEMENTOS DE UN ARRAY

mean()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
mean_array = b.mean()
print(b)
print(mean_array)
```

✓ 0.0s

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
5.0
```

valor medio de todos los elementos del array

max()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
mean_array = b.max()
print(b)
print(mean_array)
```

✓ 0.0s

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

máximo de todos los elementos del array

min()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
mean_array = b.min()
print(b)
print(mean_array)
```

✓ 0.0s

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

mínimo de todos los elementos del array

OBTENER INDICES DE MÁXIMO Y MÍNIMO DE LOS ELEMENTOS DE UN ARRAY

argmin()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
min_array_id = b.argmin()
print(b)
print(min_array_id)
```

✓ 0.0s

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
0
```

argmax()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
max_array_id = b.argmax()
print(b)
print(max_array_id)
```

✓ 0.0s

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

APLANAR UN ARRAY

Pasar de lineas y columnas a tener todo en una linea

reshape() + size

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
array_plano = b.reshape(b.size)
print(b)
print(array_plano)
```

✓ 0.0s

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

flatten()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
array_plano = b.flatten()
print(b)
print(array_plano)
```

✓ 0.0s

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

Crea una copy() del array

ravel()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
array_plano = b.ravel()
print(b)
print(array_plano)
```

✓ 0.0s

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

Crea una view() del array

TRANSPOSICIÓN DE UN ARRAY

Intercambiar filas y columnas:

swapaxes()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
array_trasnp = np.swapaxes(b, 0, 1)
print(b)
print(array_trasnp)
```

✓ 0.0s

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

transpose()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
print(b)
array_trasnp = b.transpose(1, 0)
print(array_trasnp)
```

✓ 0.0s

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

OPERACIONES CON MATRICES

```python
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
b = np.arange(1,10).reshape((3,3))
print(a)
print(b)
print('-----------')
resultado_suma = a + b
print(resultado_suma)
```

✓ 0.0s

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

suma de matrices

```python
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
b = np.arange(1,10).reshape((3,3))
print(a)
print(b)
print('-----------')
resultado_rests = a - b
print(resultado_resta)
```

✓ 0.0s

```
[[2 2 2]
 [2 2 2]
 [2 2 2]]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
-----------
[[ 1  0 -1]
 [-2 -3 -4]
 [-5 -6 -7]]
```

resta de matrices

```python
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
b = np.arange(1,10).reshape((3,3))
print(a)
print(b)
print('-----------')
resultado = (a + b - 2*a)/4
print(resultado)
```

✓ 0.0s

```
[[2 2 2]
 [2 2 2]
 [2 2 2]]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
-----------
[[-0.25  0.    0.25]
 [ 0.5   0.75  1.  ]
 [ 1.25  1.5   1.75]]
```

combinación de operaciones

MULTIPLICACIÓN MATRICIAL

Importante para IA y ML

MULTIPLICACIÓN MATRICIAL


```python
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
b = np.arange(1,10).reshape((3,3))
print(a)
print(b)
print('-----------')
# multiplicacion de matrices
matrix_multi = np.matmul(a, b)
print(matrix_multi)
print(a*b)
```

✓ 0.0s

```
[[2 2 2]
 [2 2 2]
 [2 2 2]]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
-----------
[[24 30 36]
 [24 30 36]
 [24 30 36]]
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
```

MULTIPLICACIÓN MATRICIAL


```python
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
b = np.arange(1,10).reshape((3,3))

# multiplicacion de matrices
matrix_multi = np.matmul(a, b)
print(matrix_multi)
```

✓ 0.0s

```
[[24 30 36]
 [24 30 36]
 [24 30 36]]
```

```python
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
b = np.arange(1,10).reshape((3,3))

# multiplicacion de matrices
matrix_multi = a.dot(b)
print(matrix_multi)
```

✓ 0.0s

```
[[24 30 36]
 [24 30 36]
 [24 30 36]]
```

```python
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
b = np.arange(1,10).reshape((3,3))

# multiplicacion de matrices
matrix_multi = a @ b
print(matrix_multi)
```

✓ 0.0s

```
[[24 30 36]
 [24 30 36]
 [24 30 36]]
```

NUMPY DOCS

https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html

CONQUER BLOCKS