Instituto Politécnico Formosa

TALLER DE LENGUAJES DE PROGRAMACIÓN III - PY PARA CIENCIA DE DATOS

Numpy

NumPy es una biblioteca fundamental para el lenguaje Python, diseñada para el cálculo numérico y el manejo eficiente de arreglos y matrices multidimensionales. Su núcleo está escrito en C, lo que le permite ofrecer un rendimiento óptimo en comparación con las estructuras de datos nativas de Python. Además, proporciona una amplia gama de funciones matemáticas de alto nivel, facilitando la manipulación y análisis de datos en ciencia, estadística e inteligencia artificial.

Instalación:

1. Montar un entorno virtual de python (si aun no lo tienes creado, si ya lo tienes, sáltate este paso)

En Windows

- py -m venv env

En Linux y Mac

- python3 -m venv env

2. Activar el entorno virtual

En Windows

- .\env\Script\activate\

En Linux y Mac

- source env/bin/activate

3. Instalar numpy

En ambos SO

- pip install numpy

Utilización de Numpy

Creación de arreglos: Numpy permite crear arreglos a partir de listas, tuplas y otros objetos iterables, así como generar arreglos con valores específicos (ceros, unos, secuencias).

Importamos numpy y le asignamos el alias "np" por convención.

```
import numpy as np
```
✓ 18.5s Python

Creamos las listas y le asignamos un valor

```
lista = [1,2,3,4,5,6]
lista2 = [7,8,9,10,11,12]
```
[38] ✓ 0.0s Python

Ahora podemos convertir las listas en Arreglos con el método array().

```
arr = np.array(lista)
arr2 = np.array(lista2)
print(type(arr))
print(type(arr2))
```
✓ 0.0s Python

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

La variable arr y arr2 se convirtieron en objetos de la clase numpy, ahora del tipo "ndarray", esto significa que ahora es un array de N dimensiones.

Operaciones aritméticas.

Numpy soporta una amplia gama de operaciones matemáticas elemento por elemento (suma, resta, multiplicación, división) y funciones matemáticas aplicadas a arreglos (seno, coseno, logaritmo).

Sumas de arreglos: Para la suma de arreglos podemos utilizar el operador “+” o utilizar la función “add()” de numpy.

Utilizamos la función add() para sumar los arreglos arr y arr2.

```
#Suma de arrays con la funcion "add()" numpy
sumaArrays = np.add(arr,  arr2)
print(sumaArrays)
```

[42] ✓ 0.0s Python

```
[ 8 10 12 14 16 18]
```

También podemos hacerlo con el operador “+”.

```
#Suma de arrays con el operador "+" numpy
sumaArrays = arr +  arr2
print(sumaArrays)
```

[42] ✓ 0.0s Python

```
[ 8 10 12 14 16 18]
```

Sumando los arreglos de las dos formas, obtenemos el mismo resultado.

Resta de arreglos: para la resta de arreglos podemos utilizar el operador ‘ - ’ o también la función de la clase numpy ‘subtract()’.

Utilizamos la función subtract() y pasamos como argumentos arr y arr2.

```
#Resta con numpy
restaArr = np.subtract(arr,arr2)
restaArr
```

[35] ✓ 0.0s Python

```
array([-6, -6, -6, -6, -6, -6])
```

También podemos utilizar el operador "-".

```
#Resta con numpy
restaArr = arr - arr2
restaArr
```

[46] ✓ 0.0s Python

```
array([-6, -6, -6, -6, -6, -6])
```

Multiplicación de arreglos: Con numpy podemos multiplicar arreglos de forma muy sencilla, simplemente usando la función multiply o el operador "*".

El operador multiply recibe como argumento arr y arr2.

```
mult  = np.multiply(arr, arr2)
mult
```

[48] ✓ 0.0s Python

```
array([ 7, 16, 27, 40, 55, 72])
```

Utilizando el operador "*".

```
mult  = arr * arr2
mult
```

[49] ✓ 0.1s Python

```
array([ 7, 16, 27, 40, 55, 72])
```

División de arreglos con numpy: La función "divide()" y el operador "/" simplifican la división de arreglos con numpy.

```
#Division de arreglos con numpy

div = np.divide(arr, arr2)
div
```

[50] ✓ 0.1s Python

```
array([0.14285714, 0.25      , 0.33333333, 0.4       , 0.45454545,
       0.5       ])
```

Creación de Arreglos:

Los arreglos, también llamados matrices, son estructuras de datos fundamentales para almacenar y manipular datos numéricos en una amplia gama de aplicaciones científicas y de ingeniería. En Python, la función "array()" se usa comúnmente para crear arreglos; sin embargo, existen otras funciones disponibles para este propósito.

Funciones Principales para la Creación de Arreglos

1.`array`: Esta es la función principal para crear un arreglo a partir de una lista, tupla u otra secuencia de datos. Esta función la utilizamos anteriormente en el ejemplo de más arriba.

```
arr = np.array(lista)
arr2 = np.array(lista2)
print(type(arr))
print(type(arr2))
```

✓ 0.0s Python

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

2.`arange`: Genera un arreglo con valores espaciados uniformemente dentro de un rango especificado. Recibe como parámetro el inicio, límite (no incluido) y el espacio entre elementos (pasos).

```
#funcion arange()
print(np.arange(0 , 10 , 2))
```

[6] ✓ 0.0s Python

```
[0 2 4 6 8]
```

3. `linspace`: Similar a `arange`, pero especifica el número de elementos en lugar del espaciado. Recibe como parámetro el inicio, límite(incluido) y la cantidad de elementos entre estos dos.

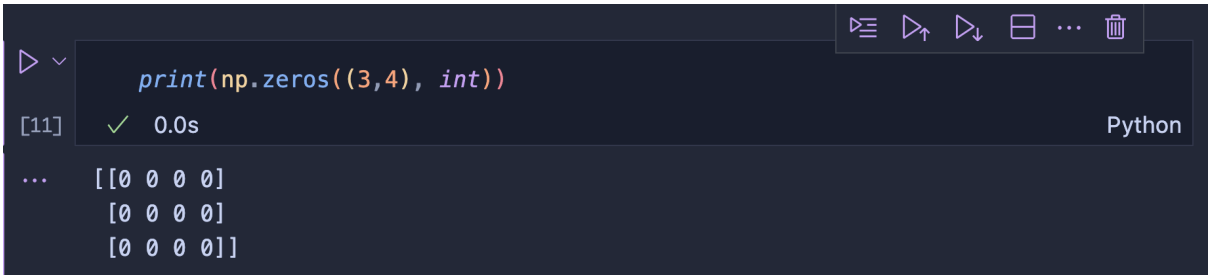
```
#funcion linspace()

print(np.linspace(1, 10, 5))
```

[8] ✓ 0.0s Python

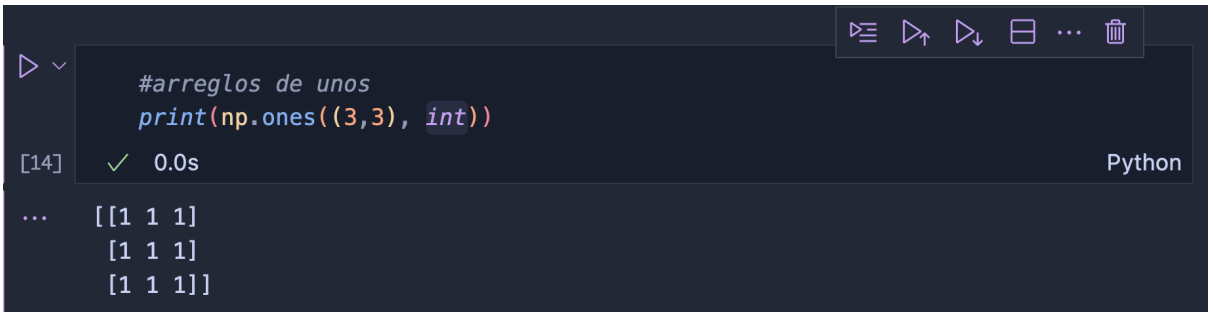
```
[ 1.    3.25  5.5   7.75 10.  ]
```

+ Código + Markdown

4. `zeros`: Crea un arreglo lleno de ceros con la forma deseada. Recibe como parámetros shape (forma del arreglo / dimensiones) y dtype (tipo de valores, por defecto 'float').



5. `ones`: Crea un arreglo lleno de unos con la forma deseada.Recibe como parámetros shape (forma del arreglo / dimensiones) y dtype (tipo de valores, por defecto 'float').



6.`empty`: Crea un arreglo sin inicializar los valores, lo que puede ser más rápido pero produce resultados impredecibles.

7. `eye`: Genera una matriz identidad con la diagonal principal llena de unos y el resto de ceros.

Parámetros:

N: Número de filas de la matriz.

M (opcional): Número de columnas de la matriz. Si no se especifica, M será igual a N, lo que crea una matriz cuadrada.

k (opcional): Desplazamiento de la diagonal.

Si k = 0, la diagonal principal tendrá los 1s.

Si k > 0, la diagonal será desplazada hacia arriba.

Si k < 0, la diagonal será desplazada hacia abajo.

dtype (opcional): Tipo de datos del arreglo resultante. El valor por defecto es float.

```
print(np.eye(4))
```

[20] ✓ 0.0s Python

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

8. `diag`: Crea una matriz diagonal a partir de una lista de valores o extrae la diagonal de una matriz existente.

Parámetros:
v: Si es un vector, se convertirá en una matriz diagonal. Si es una matriz, se extraerán los elementos de la diagonal.
k (opcional): Es el índice de la diagonal a extraer o colocar.
Si k = 0, se trabaja con la diagonal principal.
Si k > 0, se trabaja con una diagonal por encima de la principal.
Si k < 0, se trabaja con una diagonal por debajo de la principal.

```
vector = [1,2,3]
print(np.diag(vector, k=0))
```

[21] ✓ 0.1s Python

```
[[1 0 0]
 [0 2 0]
 [0 0 3]]
```