code004

June 5, 2024

[ ]:
```
#Creando arrays
import numpy as np
```

0.1 Recordando que:

Para no usar la sintaxis completa de numpy.method() Abreviamos **numpy** como **np**. Entonces usamos: > *np.method()*

[ ]:
```
#Range de python
range(0,10)
#volviendolo a lista con el metodo list()
lista=list(range(0,10))
print(f'Lista = {lista}\nTipo = {type(lista)}\n')

#Hay que tener en cuenta que es diferente a esto
print(f'Lista = {list[range(0,10)]}\ntipo = {type(list[range(0,10)])}\n')
#Esto es debido a que list es un metodo y lleva (), a pesar de que una lista se␣
 ↪declara entre corchetes
```

```
Lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Tipo = <class 'list'>

Lista = list[range(0, 10)]
tipo = <class 'types.GenericAlias'>
```

[ ]:
```
#Range en numpy
print(f'np.arange(0,10) = {np.arange(0,10)}')
```

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

0.1.1 El formato de *numpy.arange()* al igual que *range()* tiene varios parametros

range(x,y,z):	numpy.arange(x,y,z)

- x: inicio del rango [es inclusivo]
- y: fin del rango [es exclusivo]

- z: paso entre cada elemento

[ ]:
```
#Range en numpy
print(f'np.arange(0,11,2) = {np.arange(0,11,2)}')
```

```
np.arange(0,11,2) = [ 0  2  4  6  8 10]
```

[ ]:
```
#Vector de ceros
print(f'Vector = {np.zeros(5)}')

#Matriz de ceros
print(f'Matriz = \n{np.zeros((5,5))}')
```

```
Vector = [0. 0. 0. 0. 0.]
Matriz =
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

[ ]:
```
#Vector de unos
print(f'Vector = {np.ones(5)}')

#Matriz de unos
print(f'Vector = \n{np.ones((5,5))}')
```

```
Vector = [1. 1. 1. 1. 1.]
Vector =
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

0.1.2 Hay que poner especial enfásis en como se declaran los métodos

Vector->''numpy.ones(5)"

Matriz->''numpy.zeros((5,5,)")

Como te puedes dar cuenta tanto 'numpy.ones()' como 'numpy.zeros()' llevan dobles ( ) para distinguir que tipo de arreglo se está creando

[ ]:
```
#Metodo de distribución con linspace

#Creando un serie de datos mediante linspace
print(f'Datos = \n{np.linspace(0,10,10)}\n')
print(f'Datos = \n{np.linspace(0,10,20)}')
```

```
Datos =
[ 0.          1.11111111  2.22222222  3.33333333  4.44444444  5.55555556
  6.66666667  7.77777778  8.88888889 10.        ]

Datos =
[ 0.          0.52631579  1.05263158  1.57894737  2.10526316  2.63157895
  3.15789474  3.68421053  4.21052632  4.73684211  5.26315789  5.78947368
  6.31578947  6.84210526  7.36842105  7.89473684  8.42105263  8.94736842
  9.47368421 10.        ]
```

0.1.3 linspace(inicio,fin,número_de_elementos)

x = numpy.linspace(0, 10, 100)

Nos genera un secuencia de elementos que van desde **0 (cero)** a **10 (diez)** con **100** elementos dentro del arreglo

De esta manera puedo generar una serie de **n** datos con un rango establecido y una *distrubución normalizada*

[ ]:
```
#Función Matriz diagonal de unos
print(f'Matriz = \n{np.eye(5)}')
```

```
Matriz =
[[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.]]
```

0.1.4 *numpy.eye(x)*

Genera una estructura matricial con la diagonal principal en 1 y el parametro **x** nos da la dimensión de la matriz cuadrada

[ ]:
```
#Valores aleatorios

#Me genera valores aleatorios entre 0 y 1
print(f'numpy.random.rand() = {np.random.rand()}\n')

#Generando un vector de valores aleatorios
print(f'numpy.random.rand(5) = \n{np.random.rand(5)}\n')

#Generando una matriz de valores aleatorios
print(f'numpy.random.rand(3,4) = \n{np.random.rand(3,4)}\n')
```

```
numpy.random.rand() = 0.7984750553988339

numpy.random.rand(5) =
[0.77003583 0.26826937 0.49346322 0.09572636 0.5751962 ]
```

```
numpy.random.rand(3,4) =
[[0.47046769 0.97909431 0.59043578 0.76388229]
 [0.45832617 0.90161256 0.44050942 0.62489968]
 [0.4892355  0.1383628  0.92642458 0.50815927]]
```

0.1.5 *numpy.random.rand(x)*

Nos entrega un vector de números aleatorios en un rango de **0** a **1**, con **x** número de datos.

0.1.6 *numpy.random.rand(x,y)*

Nos entrega una matriz de números aleatorios en un rango de **0** a **1**, con **x** número de filas y **y** número de columnas.

[ ]:
```
#Obteniendo un número aleatorio entre un rango a - b
print(f'numpy.random.randint(1,15) = {np.random.randint(1,15)}')

#Obteniendo una matriz de números aleatorios en un rango a - b
print(f'numpy.random.randint(1,15, size=(3,3)) = \n{np.random.randint(1,15,␣
 ↪size=(3,3))}')
```

```
numpy.random.randint(1,15) = 3
numpy.random.randint(1,15, size=(3,3)) =
[[10 14  7]
 [ 6  5 13]
 [12 10 13]]
```