Curso de Python para Análisis de Datos

Clase 4

Indice:

Bibliotecas (library)

Que son?

En informática, una biblioteca o librería (del inglés library) es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca. Esto quiere decir que son un paquete de funciones ya previamente creadas para poder hacer mas sencilla la programacion en un ambito determinado. Existen infinidad de bibliotecas, algunas colecciones de ellas pueden importarse a python con fines especificos: por ejemplo, pueden ser para programar web, otras pueden ayudarnos a crear un programa de escritorio y otras pueden hacernos mas sencillo el trabajo con los datos.

Como se importan?

Para importar una libreria en Python se usa la siguiente sintaxis:

`import libreria as alias`

Para importar una funcion en particular de un modulo o libreria, utilizamos la sintaxis:

`from libreria import funcion`

Como ejemplo vamos a importar la biblioteca `math`, que nos provee distintas funciones y utilidades matematicas para Python. Es probable que se trates con más frecuencia con logaritmos y potencias, que con funciones hiperbólicas o trigonométricas. Afortunadamente, el módulo de `math` te proporciona muchas funciones que nos ayudan a calcular logaritmos.

In [ ]:
```
import math
```

Tambien podemos renombrar a la biblioteca para hacer mas facil y rapido la invocacion de la misma.

In [ ]:
```
import math as m
```

Ahora que ya importamos la libreria podemos invocar algunas de sus funciones:

In [ ]:
```
pi = m.pi
print(pi)
```

Potencias y Raices

In [ ]:
```
math.pow(2, 3)  # Potencia con flotante
math.sqrt(9)    # Raíz cuadrada (square root)
```

Redondeo: Ceil & Floor

In [ ]:
```
print(math.floor(3.99))  # Redondeo a la baja
print(math.ceil(3.01))   # Redondeo al alta
```

Truncamiento : Trunc

In [ ]:
```
# No importa los decimales, la funcion trunc devulve la parte entera del nume
ro
# Funciona de la misma forma que int()
math.trunc(123.45)
```

Funciones Trigonometricas:

Estas funciones relacionan los ángulos de un triángulo con sus lados. Tienen muchas aplicaciones, incluyendo el estudio de triángulos y el modelado de fenómenos periódicos como el sonido y las ondas de luz. Ten en cuenta que el ángulo que te proporciona es en radianes.

Puedes calcular sin(x), cos(x) y tan(x) directamente utilizando este módulo. Sin embargo, no hay una fórmula directa para calcular cosec(x), sec(x) y cot(x), pero su valor es igual al recíproco del valor devuelto por sin(x), cos(x) y tan(X), respectivamente.

¿Te suena el teorema de Pitágoras? Indica que el cuadrado de la hipotenusa (el lado opuesto al ángulo recto) es igual a la suma de los cuadrados de los otros dos lados (catetos). La hipotenusa es también el lado más grande de un triángulo rectángulo. El módulo math te proporciona la función hipot(a, b) para calcular la longitud de la hipotenusa.

In [ ]:
```
print(math.sin(math.pi/4))
print(math.cos(math.pi))
print(math.tan(math.pi/6))
print(math.hypot(12,5))
```

Funciones logaritmicas

Puedse utilizar `log(x, [base])` para calcular el registro de un número dado x a la base dada. Si omites el argumento de base opcional, el registro de x se calcula a la base `e`. Aquí, e es una constante matemática cuyo valor es 2.71828182 .... y puedes acceder a él usando `math.e`.

In [ ]:
```
print(math.exp(5))
print(math.e**5 )
```

Si deseas calcular los valores de un logaritmo base-2 o base-10, utilizando `log2(x)` y `log10(x)` se obtendrán resultados más precisos que `log(x, 2)` y `log(x, 10)`. Ten en cuenta que no existe ninguna función log3(x), por lo que tendrá que seguir utilizando log (x, 3) para calcular los valores de logaritmo base-3. Lo mismo ocurre con todas las demás bases.

In [ ]:
```
print(math.log(148.41315910257657))
print(math.log(148.41315910257657, 2))
print(math.log2(148.41315910257657))
```

In [ ]:
```
print(math.log10(148.41315910257657))
print(math.log(148.41315910257657, 10))
```

Es importante conocer las bibliotecas que estemos utilizando, la clave va a ser conocer la documentacion de las mismas para saber que funciones poseen y como se implementan.

Para buscar el listado de todas las funciones de las bibliotecas podemos acceder a la documentacion oficial de python (https://www.docs.python.org (https://www.docs.python.org))

Bibliotecas en Data Science

Python tiene implementadas muchas librerias para poder trabajar con datos. En la clase de hoy trabajaremos con dos de ellas: `Numpy` y `Pandas`.

Antes de comenzar, vamos a hablar un poco de estas dos librerias o modulos.

Numpy es una librería optimizada para realizar cálculos numéricos con vectores y matrices. A diferencia de otros lenguajes de programación, Python no posee en su estructura central la figura de matrices. Eso quiere decir que para poder trabajar con esta estructura de datos deberiamos trabajar con listas de listas. NumPy introduce el concepto de arrays o matrices.

Por otro lado, **Pandas** es una libreria que es una extensión de NumPy. Basicamente al utilizar `pandas`, utilizo `numpy` por debajo.

Esta orientada a la manipulación y análisis de datos debido a que ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales.

La estructura principal de `pandas` es el `DataFrame` que es muy similar a una tabla. Así también, contiene otra estrucutra denominada `Serie`.

Al ser de código abierto, `pandas` y `numpy` poseen una documentación muy amplia que es **SIEMPRE RECOMENDABLE** consultar.

- Documentacion NumPy (https://devdocs.io/numpy/)
- Documentacion Pandas (https://pandas.pydata.org/pandas-docs/stable/)

Modulo 1: NumPy

Como convencion, cuando se importa `numpy` se le asigna el alias `np`. Pero esto no es obligatorio, solo me facilita muchas veces la escritura del codigo.

In [ ]:
```
#Importar numpy
import numpy as np
```

In [ ]:
```
np.pi == m.pi
```

In [ ]:
```
np.pi
```

De ahora en mas para utilizar una función de Numpy solo tengo que usar `np` y luego llamar función. Por ejemplo, si quiero llamar la función `.mean()`, debo escribir: `np.mean()`.

Comenzemos a ver que funciones se pueden utilizar en NumPy.

- `.array()`

Esta función me permite crear un array. Es posible crear arrays a partir de listas.

Cual es la diferencia entre un array y una lista?

Los array y las listas se usan en Python para almacenar datos, pero no sirven exactamente para los mismos propósitos. Ambos se pueden usar para almacenar cualquier tipo de datos (números reales, cadenas, etc.), y ambos se pueden indexar e iterar, pero las similitudes entre los dos no van mucho más allá. La principal diferencia entre una lista y un array son las funciones que puede realizarles. Por ejemplo, puede dividir un array por 3, y cada número en el array se dividirá por 3 y el resultado se imprimirá si lo solicita. Si intenta dividir una lista por 3, Python le dirá que no se puede hacer y se generará un error.

In [ ]:
```
#creo un array
arr = np.array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
type(arr)
```

In [ ]:
```
arr
```

In [ ]:
```
#crea una lista llamada mi_lista que contenga 10 numeros

mi_lista = [45,123,89,3,45,2,99,56,34,10]
```

In [ ]:
```
type(mi_lista)
```

In [ ]:
```
#Ahora transforma tu lista en un array usando np.array y asignalo a la variab
le mi_array

mi_array = np.array(mi_lista)
#Imprimi mi_array
print(mi_array)
```

Podemos usar `type` para obtener el tipo de estructura de datos que estamos trabajando.

In [ ]:
```
#Aplica type sobre mi_array para mostrar que tipo de estructura de datos es
type(mi_array)
```

In [ ]:
```
mi_lista + mi_lista
```

Los arrays y las listas se comportan diferente frente operaciones matematicas con números.

In [ ]:
```
#Corre el siguiente codigo
print(mi_lista + 2)
```

No es posible sumar un numero a cada uno de los elementos de una lista. Sin embargo, esto funciona distinto en Numpy.

In [ ]:
```
#Suma 2 a cada elemento de mi_array
mi_lista2 = [i + 2 for i in mi_lista]

print(mi_lista2)
```

El comportamiento tambien es distinto para la operacion de multiplicacion.

In [ ]:
```
#Corre el siguiente codigo y fijate que pasa
print(mi_lista * 3)
```

In [ ]:
```
#Ahora multiplica por dos cada elemento de mi_array

mi_lista3 =  [i * 2 for i in mi_lista]

print(mi_lista3)
```

In [ ]:
```
mi_array
```

In [ ]:
```
mi_array * 2
```

Los arrays y las listas se comportan diferente frente a operaciones con otros arrays/listas

In [ ]:
```
#Corre el siguiente codigo
lista1 = [1, 2, 3, 4, 5]
lista2 = [5, 4, 3, 2, 1]

arr1 = np.array(lista1)
arr2 = np.array(lista2)
```

In [ ]:
```
#Concatena las dos listas
lista1 + lista2
```

In [ ]:
```
arr1 + arr2
```

In [ ]:
```
#Suma los elementos de los dos arrays

#lista3 = []
#for i in lista1:
#  for j in lista2:
#    lista3.append(i + j)
#print(lista3)
lista1 = [1,2,3,4]
lista2 = [9,2,3,4]

array1 = np.array(lista1)
array2 = np.array(lista2)

print(array1 + array2)
```

In [ ]:
```
#Multiplica los elementos de cada array
array1*array2
```

In [ ]:
```
#Resta los dos arrays
array1-array2
```

In [ ]:
```
#Dividi los dos arrays
array1/array2
```

In [ ]:
```
#Eleva el array1 al array2
array1**array2
```

In [ ]:
```
array_largo = np.array([1,2,3,4,5])
array_largo
```

In [ ]:
```
array_largo.shape
```

In [ ]:
```
array_largo * array1
# Para operar arrays tienen que tener la misma forma (misma cantidad de dato
s)
```

In [ ]:
```
array_largo * 5
```

In [ ]:
```
array1.dtype
```

Para acceder a los elementos de los arrays de una dimension se utilizan los indices como las listas.

In [ ]:
```
#Corre el siguiente codigo
print(lista1[0], arr1[0])
```

In [ ]:
```
#Obtene el 3er elemento de arr1
print(lista1[2], arr1[2])
```

In [ ]:
```
#Obtene el 5to elemento de arr2
print (lista2[4], arr2[4])
```

In [ ]:
```
#Obtene el ultimo elemento de arr1
arr1[-1]
```

Asi tambien obtenemos una porción del array de la misma manera que obtenemos una parte de una lista. A su vez podemos reasignar nuevos valores a una porción del array.

In [ ]:
```
array1 = np.array( [1,2,3,4,5])
```

In [ ]:
```
#Corre el siguiente codigo
array1[2:5] = [22, 23, 24]
```

In [ ]:
```
#Imprimi el arr1
print(array1)
```

Arreglos multidimensionales

Las listas son cadenas unidimensionales de elementos. Los arreglos pueden ser multidimensionales. Para comprender mejor que son, miremos el siguiente ejemplo:

In [ ]:
```
#Corre el siguiente codigo
lista = [[0, 1, 3], [3, 4, 5]]
arr2d = np.array(lista)
arr2d
```

In [ ]:
```
#Veamos cuantas dimensiones tiene el array
print(arr2d.ndim)
```

In [ ]:
```
#Veamos cuantos elementos tiene el array
print(arr2d.size)
```

In [ ]:
```
#Arma ahora un array de 3 dimensiones.
arr3 =  [ [2,4,6,5],[1,3,5,90],[8,10,12,-8] ]
arr3d = np.array(arr3)
arr3d
```

In [ ]:
```
#Chequea usando ndim que efectivamente tenga tres dimensiones
print(arr3d.ndim)
```

In [ ]:
```
arr3d.shape
```

¿Como accedemos entonces a un array de mas de una dimension?

In [ ]:
```
#definamos un nuevo array a partir de una matriz
lista2 = [[0, 1, 3], [3, 4, 5], [9, 10, 4], [2, 5, 6]]
nuevo_array = np.array(lista2)
```

In [ ]:
```
#Dimensiones de nuevo_array
print(nuevo_array.ndim)
```

In [ ]:
```
#Mostra nuevo_array
print(nuevo_array)
```

Si para un array de 1 dimension, usabamos un indice, para un array de n dimensiones tenemos que usar n indices. En el caso de 2 dimensiones, usaremos 2 indices. El primero indica la fila y el segundo la columna.

In [ ]:
```
#Corre el siguiente codigo
print(nuevo_array[0, 1])
```

In [ ]:
```
#Accede al tercer elemento de la segunda fila de nuevo_array
print(nuevo_array[1,2])
```

In [ ]:
```
#Accede al cuarto elemento de la tercer columna de nuevo_array
print(nuevo_array[3,2])
```

In [ ]:
```
#Accedo a toda la segunda columna
nuevo_array[:, 1]
```

In [ ]:
```
#Accede a un subarray que vaya desde el segundo elemento de la
#primer columna hasta el cuarto elemento de la tercer
#columna de nuevo_array
print(nuevo_array[: , 2])
```

In [ ]:
```
#Accede a un subarray que vaya desde el primer elemento de la
#segunda fila hasta el segundo elemento de la cuarta
#fila de nuevo_array
nuevo_array[1:, 0:2]
```

In [ ]:
```
nuevo_array
```

Creación de arreglos

- **np.zeros(shape, dtype, order)**: Crea un array con todos ceros en sus posiciones

In [ ]:
```
np.zeros?
```

In [ ]:
```
zeros = np.zeros(shape = (2,8))
```

In [ ]:
```
#Corre el siguiente codigo
print(zeros)
```

In [ ]:
```
zeros.ndim
```

In [ ]:
```
zeros.shape
```

In [ ]:
```
zeros.size
```

Docstrings y comentarios

In [ ]:
```
def mi_funcion():
  '''
  Esta es la documentaciòn de mi función con el tilde al revés
  '''
  # esto es un comentario a parte
  return True
```

In [ ]:
```
mi_funcion?
```

- **np.ones(shape, dtype, order)**: Crea un array con todos unos en sus posiciones

In [ ]:
```
np.ones?
```

In [ ]:
```
#Corre el siguiente codigo
print(np.ones(10))
```

Seed()

Nos permite iniciar una secuencia pseudo random. Se comporta como random pero siempre siguiendo el mismo patron por semilla. Se lo invoca de la siguiente manera: `np.random.seed(int)`.

In [ ]:
```
np.random.seed(seed=None)
```

- **np.random.rand(d0, d1, ..., dn)**: Genera un array con valores al azar con una distribución N~(0, 1) y segun las dimensiones pasada como argumento.

In [ ]:
```
#Genera un array con valores al azar
print(np.random.randn(10,2))
```

In [ ]:
```
r = np.random.randn(1000)
```

In [ ]:
```
r.ndim
```

In [ ]:
```
r.shape
```

In [ ]:
```
r.mean()
```

In [ ]:
```
r.std()
```

- **np.random.randint(low, high, size)**: Devuelve un array con enteros al azar. Cuando se especifica un solo entero como argumento este entero es entendido como el mayor valor que ese numero random puede tomar. Si se pasan dos, uno es el menor valor y el otro es el mayor valor. Con size se le puede decir que dimensiones tiene el array. Si no le paso ningun valor, devolvera solo un numero entero.

In [ ]:
```
#Genera un array con enteros al azar
np.random.randint(5, 10, 4)
```

In [ ]:
```
np.random.randint?
```

In [ ]:
```
for i in range(5,10,3):
  print(i)
```

- **np.arange([start,]stop, [step,])**: Devuelve numero simetricamente distribuidos segun los valores datos. El primer valor es el comienzo, el segundo el final y el tercero representa cada cuanto queremos esos valores.

Range + array

In [ ]:
```
#Corre esta linea de codigo y comprende como funciona np.arange
np.arange(2, 16, 2)
```

Funciones estadisticas

NumPy nos facilita varias funciones que nos permitiran obtener algunos parametros estadisticos de nuestros arrays. Veamos cuales son estas funciones.

In [ ]:
```
#Volvamos a ver como era nuestro array nuevo_array
print(nuevo_array)
```

In [ ]:
```
#Media
np.mean(nuevo_array)
```

In [ ]:
```
#Media
nuevo_array.mean()
```

In [ ]:
```
#usa la funcion mean para obtener el promedio de los valores en nuevo_array
np.mean(nuevo_array)
```

- `np.var()`: Devuelve la dispersión de los valores alrededor de la media

Voy a calcular la varianza con la fórmula

var = 1/n\sum_{i=0}_{n} (x_i - X)^2

- Elemento de la lista
- Elemento de la lista

In [ ]:
```
#usa la funcion var para obtener la varianza de los valores en nuevo_array
np.var(nuevo_array)
```

In [ ]:
```
nuevo_array.std()
```

- `np.sum()`: devuelve la suma de todos los valores en el array.
- `np.min()`: devuelve el menor valor en el array
- `np.max()`: devuelve el maximo valor en el array

In [ ]:
```
nuevo_array
```

In [ ]:
```
#Muestra la suma de los valores en nuevo_array
np.sum(nuevo_array)
```

In [ ]:
```
nuevo_array.sum(axis=0)
```

In [ ]:
```
nuevo_array.sum(axis=1)
```

In [ ]:
```
np.min(nuevo_array)
```

In [ ]:
```
#Muestra el minimo y maximo valor en nuevo array
np.max(nuevo_array)
```

In [ ]:
```
nuevo_array.min(axis=0)
```

In [ ]:
```
nuevo_array.max(axis=1)
```

Ejercicios

1. Crear un arreglo de ceros de longitud 12
2. Crear un arreglo de longitud 10 con ceros en todas sus posiciones y un 10 en la posición número 5
3. Crear un arreglo que tenga los números del 10 al 49
4. Crear una arreglo 2d de shape (3, 3) que tenga los números del 0 al 8
5. Crear un arreglo de números aleatorios de longitud 100 y obtener su media y varianza
6. Calcular la media de un arreglo usando np.sum
7. Calcular la varianza de un arreglo usando np.sum y np.mean
8. Crear un array de números aleatorios usando np.random.randn.

1. Varianza: $S^2 = \frac{\sum (x_i - \bar{x})^2}{n}$