Aplicar funciones a elementos de `Series` o `DataFrame`s

Cualquier procesado, manipulación o análisis de conjuntos de datos implica el aplicar transformaciones, funciones o calcular indicadores elemento a elemento, fila a fila o columna a columna.

```
# Preliminares
import pandas as pd
import numpy as np
```

Para ilustrar, generamos un `DataFrame` usando un generador de números aleatorios.

```
# Usamos el generador por defecto del módulo random de numpy
rng = np.random.default_rng()
#
df = pd.DataFrame(
    rng.standard_normal(size=10).reshape(5, 2),
    columns=list('xy')
)
df
```

	x	y
0	-0.131601	-0.186435
1	-1.908522	0.039505
2	0.324792	1.964660
3	0.093982	0.760966
4	1.246065	-0.346505

Funciones universales `ufuncs` de `numpy`

Puesto que un `DataFrame` también es un array de `numpy`, podemos aplicarle cualquier función universal de `numpy`. Las funciones universales de `numpy` (ufuncs) se pueden consultar en la referencia Available ufuncs

```
# Calculamos la exponencial de x e y, elemento a elemento
np.exp(df)
```

	x	y
0	0.876691	0.829913
1	0.148299	1.040296
2	1.383743	7.132488
3	1.098540	2.140343
4	3.476635	0.707155

```python
# Elevamos x a la potencia y, elemento a elemento:
np.power(np.abs(df['x']), df['y'])
```

```
0          NaN
1          NaN
2     0.109767
3     0.165395
4     0.926605
dtype: float64
```

También se pueden aplicar métodos de `pandas` directamente a todo un `DataFrame`

```
# Para obtener la suma, columna por columna
df.sum()
```

```
x   -0.375284
y    2.232192
dtype: float64
```

```
# Para obtener la media, columna por columna
df.mean()
```

```
x   -0.075057
y    0.446438
dtype: float64
```

```python
# Para obtener la desviación típica, columna por columna
df.std()
```

```
x    1.150935
y    0.948526
dtype: float64
```

También podríamos calcular estos indicadores, fila por fila, usando el parámetro `axis=1`.

```
df.sum(axis=1)
```

```
0   -0.318036
1   -1.869017
2    2.289452
3    0.854949
4    0.899560
dtype: float64
```

Podéis encontrar y explorar la gran cantidad de métodos disponibles para un `DataFrame` en la documentación de `pandas`.

Aplicar funciones a elementos de `Series` o `DataFrame`s (II)

En el vídeo anterior vimos cómo aplicar funciones universales de `numpy` o los métodos de `pandas` a `DataFrame`s. Pero puede ser que queramos aplicar nuestras propias funciones.

Al igual que en el vídeo anterior, para ilustrar, generamos un `DataFrame` usando un generador de números aleatorios.

```
# Usamos el generador por defecto del módulo random de numpy
rng = np.random.default_rng()
#
df = pd.DataFrame(
    rng.standard_normal(size=10).reshape(5, 2),
    columns=list('xy')
)
df
```

	x	y
0	-0.131601	-0.186435
1	-1.908522	0.039505
2	0.324792	1.964660
3	0.093982	0.760966
4	1.246065	-0.346505

Empezamos por crear una función

```
def mi_funcion(x):
    return 'es positivo' if x >= 0 else 'es negativo'
# hemos usado la construcción if else
```

Para aplicar nuestra función a cada elemento del `DataFrame` usamos `applymap`

```
df.applymap(mi_funcion)
```

	x	y
0	es negativo	es negativo
1	es negativo	es positivo
2	es positivo	es positivo
3	es positivo	es positivo
4	es positivo	es negativo

Si quisieramos aplicar nuestra función a cada elemento de una `Serie` usaríamos `apply`:

```
df['x'].apply(mi_funcion)
```

```
0    es negativo
1    es negativo
2    es positivo
3    es positivo
4    es positivo
Name: x, dtype: object
```

Las funciones anónimas en Python

Es muy útil tener la posibilidad de definir sobre la marcha una función sin darle un nombre y usarla como argumento de un método. Se hace con `lambda`.

```
# Para obtener el doble de cada elemento
df.applymap(lambda x: x * 2)
```

```
x   -0.375284
y    2.232192
dtype: float64
```

El método `apply` para un `DataFrame`

El método `apply` si se aplica a un `DataFrame` debe tener como argumento a una función que se aplica a toda una columna o toda una fila, no a un elemento individual.

```
# Ejemplo con una función que se aplica a un vector: calculamos el número de valores positivos por columna
df.apply(lambda x: (x >= 0).sum())
```

```
x    3
y    3
dtype: int64
```

El método `apply` admite también el argumento `axis=1`, para que se aplique la función fila por fila.

```
df.apply(lambda x: (x >= 0).sum(), axis=1)
```

```
0    0
1    1
2    2
3    2
4    1
dtype: int64
```

> *Es muy recomendable evitar `apply` si es posible, y usar las funciones de `numpy` o los métodos de `pandas` que son muy optimizados y llevan a cabo la iteración en C. `apply` lleva a cabo la iteración en Python.*

En el caso anterior, podríamos haber usado directamente el método `sum` en `pandas`.

```
(df >= 0).sum()
```

```
x    3
y    3
dtype: int64
```

```
# Fila por fila:
(df >= 0).sum(axis=1)
```