Indexación y Selección de Datos

En este cuaderno vamos a estudiar distintas formas de navegar, recuperar y modificar la información contenida en los objetos Pandas Series y DataFrame. Este notebook se basa en esta documentación oficial.

Inicialización del entorno.

Lo primero será importar el paquete Pandas con alias `pd`. Posteriormente visualizamos la versión usada de Pandas ya que es un dato importante para consultar la documentación. En el momento de editar este notebook la versión de pandas es: 1.4.3

Además para facilitar los ejercicios también importamos el paquete Numpy con el alias `np`.

In [1]:
```
import pandas as pd
import numpy as np
np.set_printoptions(floatmode='fixed', precision=3)
print('Pandas versión: ', pd.__version__)
```

```
Pandas versión:  1.4.3
```

Indexación de objetos Series.

Considerando una serie como un diccionario.

Una forma de ver una objeto Serie es como una colección de pares (*clave*, *valor*), es decir, como un diccionario. El campo *clave* será un valor del objeto Index asociado a la serie, y el campo *valor*, el item de datos correspondiente. Utilizaremos el operador `[]` para acceder a la serie. Se recomienda repasar el tipo python dict.

Ejercicio 01: Dada la una serie con las cotizaciones para varias empresas, se desea conocer el valor de cotización de la empresa 'VVBA'.

La salida esperada debe ser algo parecido a lo siguiente:

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Cotización de la empresa 'VVBA': 2.892
```

In [2]:
```
ibex = pd.Series({'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
                  'CAJANOR':1.763})
```

```
print ('Cotizaciones:\n', ibex)
print ('\nCotización de la empresa \'VVBA\': ', end='')
#Pon tu código aquí.
#Sugerencia: utiliza el operador [] con la etiqueta de índice 'VVBA'.

#
```

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Cotización de la empresa 'VVBA':
```

Como una Serie puede ser utilizada como un diccionario Python, podemos utilizar las operaciones de diccionario con la misma.

Ejercicio 02: Dada la una serie con las cotizaciones para varias empresas, se desea saber si hay alguna cotización asociada a la empresa 'BANFINTER'.

La salida esperada debe ser algo parecido a lo siguiente:

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Hay dato de cotización para la empresa 'BANFINTER'?: False
```

In [3]:
```
ibex = pd.Series({'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763})
print ('Cotizaciones:\n', ibex)
print ('\nHay dato de cotización para la empresa \'BANFINTER\'?: ',
      end='')
#Pon tu código aquí.
#Sugerencia utiliza el operador 'in' con la etiqueta 'BANFINTER'

#
```

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Hay dato de cotización para la empresa 'BANFINTER'?:
```

Ejercicio 03: Dada la una serie con las cotizaciones para varias empresas, se desea obtener una lista con todas las empresas con dato de cotización (claves del diccionario).

La salida esperada debe ser algo parecido a lo siguiente:

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
```

```
CAJANOR    1.763
dtype: float64

Empresas: ['BANQIA', 'VVBA', 'SANTAN', 'CAJANOR']
```

In [4]:
```
ibex = pd.Series({'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763})
print ('Cotizaciones:\n', ibex)
print('\nEmpresas: ', end='')
#Pont tu código aquí.
#Sugerencia: crea una lista usando el resultado del método
#  keys() de un diccionario.

#
```

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Empresas:
```

Ejercicio 04: Dada una serie con las cotizaciones para varias empresas, se desea obtener una lista con todos los pares (tuplas) (*clave*,*valor*).

La salida esperada debe ser algo parecido a lo siguiente:

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Valores: [('BANQIA', 0.88), ('VVBA', 2.892), ('SANTAN', 2.076),
('CAJANOR', 1.763)]
```

In [5]:
```
ibex = pd.Series({'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763})
print ('Cotizaciones:\n', ibex)
print('\nValores: ', end='')
#Pont tu código aquí.
#Sugerencia: crea una lista a partir del resultado del
#método items() de un diccionario.

#
```

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Valores:
```

Ejercicio 05: Dada la una serie con las cotizaciones para varias empresas, se desea cambiar el valor de contización de la empresa 'CAJANOR' a 1.0.

La salida esperada debe ser algo parecido a lo siguiente:

```
Cotizaciones:
BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Serie actualizada:
BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.000
dtype: float64
```

In [6]:

```
ibex = pd.Series({'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763})
print ('Cotizaciones:')
print (ibex)

#Pon tu código aquí
#Sugerencia: Utiliza el operador [] para asignar el nuevo valor.

#

print ('\nSerie actualizada:')
print(ibex)
```

```
Cotizaciones:
BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Serie actualizada:
BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64
```

Ejercicio 06: Dada la una serie con las cotizaciones para varias empresas, se desea añadir la cotización de una nueva empresa 'BANFINTER' con el valor 4.6.

La salida esperada debe ser algo parecido a lo siguiente:

```
Cotizaciones:
BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Cotizaciones actualizas:
BANQIA       0.880
VVBA         2.892
SANTAN       2.076
CAJANOR      1.763
```

```
BANFINTER    4.600
dtype: float64
```

In [7]:
```
ibex = pd.Series({'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763})
print ('Cotizaciones:')
print (ibex)

#Pon tu código aquí.
#Sugerencia: Como una serie funciona con un diccionario,
#simplemente asigna a la nueva etiqueta 'BANFINTER' el valor 4.6

#

print('')
print('Cotizaciones actualizas:')
print(ibex)
```

```
Cotizaciones:
BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Cotizaciones actualizas:
BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64
```

Una posibilidad que no coincide directamente con un diccionario es poder seleccionar rangos usando valores de indice con el formato `'ETIQUETA-INICIO':'ETIQUETA-FIN'` con la particulariad de que en este caso la etiqueta que indica fin también entra en el rango.

Ejercicio 07: Dada la una serie con las cotizaciones para varias empresas, se desea seleccionar el rango que va desde la etiqueta 'BANQIA' a 'SANTAN'.

La salida esperada debe ser algo parecido a lo siguiente:

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Cotizaciones en el rango 'BANQIA':'SANTAN':
BANQIA    0.880
VVBA      2.892
SANTAN    2.076
dtype: float64
```

In [8]:
```
ibex = pd.Series({'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763})
print ('Cotizaciones:\n', ibex)

print("\nCotizaciones en el rango 'BANQIA':'SANTAN':")
#Pon tu código aquí:
#Sugerencia: indexa usando el rango 'BANQIA':'SANTAN'
```

```
#
```

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Cotizaciones en el rango 'BANQIA':'SANTAN':
```

También podemos indicar directamente una lista los valores de indice que queremos seleccionar.

Ejercicio 08: Dada la una serie con las cotizaciones para varias empresas, se desea seleccionar los valore de contización para las empresas `['BANQIA', 'CAJANOR']`.

La salida esperada debe ser algo parecido a lo siguiente:

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Cotizaciones para 'BANQIA' y 'CAJANOR':
BANQIA     0.880
CAJANOR    1.763
dtype: float64
```

In [9]:
```
ibex = pd.Series({'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
             'CAJANOR':1.763})
print ('Cotizaciones:\n', ibex)

print("\nCotizaciones para 'BANQIA' y 'CAJANOR':")
#Pon tu código aquí:
#Sugerencia: indexa con el operador [] usando
#     la lista de etiquetas ['BANQIA', 'CAJANOR']

#
```

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Cotizaciones para 'BANQIA' y 'CAJANOR':
```

Considerando la Serie como un numpy.ndarray de una dimensión.

Un objeto Series también puede ser manipulado como si fuera un numpy.ndarray una dimensión, esto es con format (*x*,). En este caso el operador `[]` se utiliza con valores enteros de índice el ndarray.

Ejercio 09: Dada la una serie con las cotizaciones para varias empresas, se desea obtener la segunda cotización.

La salida esperada debe ser algo parecido a lo siguiente:

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

El segundo valor de cotización es: 2.892
```

In [10]:
```
ibex = pd.Series({'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763})
print ('Cotizaciones:\n', ibex)

print('\nEl segundo valor de cotización es: ', end='')
#Pon tu código aquí.
#Sugerencia: utiliza la series como si fuera un array. No
#olvides que los arrays se indexan con base 0.

#
```

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

El segundo valor de cotización es:
```

También podemos usar máscaras para seleccionar valores que cumplan un predicado lógico.

Ejercicio 10: Dada la una serie con las cotizaciones para varias empresas, se desea obtener las cotizaciones de la empresas cuya cotización esté en el intervalo $[2.0, 3.0]$.

La salida esperada debe ser algo parecido a lo siguiente:

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Empesas con cotización en el intervalo [2.0, 3.0]:
VVBA      2.892
SANTAN    2.076
dtype: float64
```

In [11]:
```
ibex = pd.Series({'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763})
print ('Cotizaciones:\n', ibex)

print('\nEmpesas con cotización en el intervalo [2.0, 3.0]:')
#Pon tu código aquí.
#Sugerencia: crea un máscara lógica que evalue el predicado lógico
#de pertenencia al intervalo.

#
```

```
Cotizaciones:
 BANQIA     0.880
```

```
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Empesas con cotización en el intervalo [2.0, 3.0]:
```

Ejercicio 11: Dada la una serie con las cotizaciones para varias empresas, se desea obtener las cotizaciones de la empresas, que están en posiciones impares.

La salida esperada debe ser algo parecido a lo siguiente:

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Cotizaciones en posición impar:
VVBA       2.892
CAJANOR    1.763
dtype: float64
```

In [12]:
```
ibex = pd.Series({'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763})
print ('Cotizaciones:\n', ibex)

print('\nCotizaciones en posición impar:')
#Pon tu código aquí.
#Sugerencia: indexa con un rango para posiciones impares tal y como
#harías con un ndarray.

#
```

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Cotizaciones en posición impar:
```

Indexación del objeto DataFrame.

Considerando un DataFrame como un diccionario.

Un objeto DataFrame puede ser utilizado como si fuera un diccionario donde las claves son los nombres de las columnas y los valores son objetos Series asociado a cada columna.

Ejercicio 12: Dada un DataFrame `df` queremos obtener la serie correspondiente a la columna 'c'.

La salida esperada debe ser algo parecido a lo siguiente:

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
```

```
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644

Serie para columna 'c':
0    0.028320
1    0.124283
2    0.670624
3    0.647190
4    0.615385
5    0.383678
6    0.997210
7    0.980835
8    0.685542
9    0.650459
Name: c, dtype: float64
```

In [13]:

```
gen = np.random.default_rng(0)
df = pd.DataFrame(dict(zip(list('abcde'),
                           [gen.random(10) for i in range(5)])))
print('DataFrame:\n', df)
print("\nSerie para columna 'c':")
#Pon tu código aquí
#Sugerencia: utiliza el operador [] con la etiqueta de la
#    columna requerida.

#
```

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644

Serie para columna 'c':
```

Podemos preguntar si hay alguna columna con una etiqueta dada. Esto es útil para impedir que se produzcan excepciones del tipo 'KeyError'.

Ejercicio 13: Dado un DataFrame `df`, comprueba hay una columna con etiqueta 'c' y otra con la etiqueta 'f' y qué ocurre cuando intentamos acceder a una columna con etiqueta inexistente.

La salida esperada debe ser algo parecido a lo siguiente:

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
```

```
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644

Hay una columna con etiqueta 'c'? True

Hay una columna con etiqueta 'f'? False

Serie asociada a la etiqueta 'f':
df['f'] ha generado una excepción KeyError
```

In [14]:

```
gen = np.random.default_rng(0)
df = pd.DataFrame(dict(zip(list('abcde'),
                           [gen.random(10) for i in range(5)])))
print('DataFrame:\n', df)
print("\nHay una columna con etiqueta 'c'? ", end="")
#Pon tu código aquí
#Sugerencia: utiliza el operador in.

#

print("\nHay una columna con etiqueta 'f'? ", end="")
#Pon tu código aquí
#Sugerencia: utiliza el operador in.

#

print("\nSerie asociada a la etiqueta 'f':")
try:
#Pon tu código aquí
#Sugerencia: utiliza el operador [] para obtener la serie
#   con nombre 'f'
    print()
#
except KeyError:
    print("df['f'] ha generado una excepción KeyError")
```

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644

Hay una columna con etiqueta 'c'?
Hay una columna con etiqueta 'f'?
Serie asociada a la etiqueta 'f':
```

Ejercicio 14: Dada un DataFrame `df`, queremos obtener una lista con todas las etiquetas de columna. Utiliza el constructor de lista `list(obj)` para ello.

La salida esperada debe ser algo parecido a lo siguiente:

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644

Etiquetas de columna: ['a', 'b', 'c', 'd', 'e']
```

In [15]:
```
gen = np.random.default_rng(0)
df = pd.DataFrame(dict(zip(list('abcde'),
                           [gen.random(10) for i in range(5)])))
print('DataFrame:\n', df)
print("\nEtiquetas de columna: ", end="")
#Pon tu código aquí
#Sugerencia: utiliza el resultado del método keys()
#    para crear la lista.

#
```

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644

Etiquetas de columna:
```

Ejercicio 15: Dada un DataFrame `df`, queremos obtener una lista con todos los pares (*etiqueta*, Series) que forman la tabla.

La salida esperada debe ser algo parecido a lo siguiente:

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644
```

```
Lista de elmentos: [('a', 0    0.636962
1    0.269787
2    0.040974
3    0.016528
4    0.813270
5    0.912756
6    0.606636
7    0.729497
8    0.543625
9    0.935072
Name: a, dtype: float64), ('b', 0    0.815854
1    0.002739
2    0.857404
3    0.033586
4    0.729655
5    0.175656
6    0.863179
7    0.541461
8    0.299712
9    0.422687
Name: b, dtype: float64), ('c', 0    0.028320
1    0.124283
2    0.670624
3    0.647190
4    0.615385
5    0.383678
6    0.997210
7    0.980835
8    0.685542
9    0.650459
Name: c, dtype: float64), ('d', 0    0.688447
1    0.388921
2    0.135097
3    0.721488
4    0.525354
5    0.310242
6    0.485835
7    0.889488
8    0.934044
9    0.357795
Name: d, dtype: float64), ('e', 0    0.571530
1    0.321869
2    0.594300
3    0.337911
4    0.391619
5    0.890274
6    0.227158
7    0.623187
8    0.084015
9    0.832644
Name: e, dtype: float64)]
```

In [16]:

```python
gen = np.random.default_rng(0)
df = pd.DataFrame(dict(zip(list('abcde'),
                           [gen.random(10) for i in range(5)])))
print('DataFrame:\n', df)
print("\nLista de elmentos: ", end="")
#Pon tu código aquí
```

```
#Sugerencia: utiliza el resultado del método items()
#   para crear la lista.

#
```

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644

Lista de elmentos:
```

Considerando el DataFrame como un numpy.ndarray de dos dimensiones.

Una forma alternativa de ver un objeto DataFrame es como una numpy.ndarray de dos dimensiones. La forma recomendada para acceder a esta vista del objeto sería usando el método to_numpy().

Ejercicio 16: Dada un DataFrame `df`, queremos obtener una vista en formanto numpy.ndarray.

La salida esperada debe ser algo parecido a lo siguiente:

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644

Vista como ndarray:
 [[0.637 0.816 0.028 0.688 0.572]
 [0.270 0.003 0.124 0.389 0.322]
 [0.041 0.857 0.671 0.135 0.594]
 [0.017 0.034 0.647 0.721 0.338]
 [0.813 0.730 0.615 0.525 0.392]
 [0.913 0.176 0.384 0.310 0.890]
 [0.607 0.863 0.997 0.486 0.227]
 [0.729 0.541 0.981 0.889 0.623]
 [0.544 0.300 0.686 0.934 0.084]
 [0.935 0.423 0.650 0.358 0.833]]
```

In [17]:
```
gen = np.random.default_rng(0)
df = pd.DataFrame(dict(zip(list('abcde'),
                           [gen.random(10) for i in range(5)])))
print('DataFrame:\n', df)
vista = 0
#Pon tu código aquí
```

```
#Sugerencia: utiliza el método to_numpy().

#
print('\nVista como ndarray:\n', vista)
```

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644

Vista como ndarray:
 0
```

Hay que tener en cuenta que el objeto devuelto puede ser una copia o una vista, dependiendo de los datos almacenados en el DataFrame por lo que, no está gerantizado que al modificar el ndarray se modifique el DataFrame correspondiente (ver *setting-with-copy warning*). Lo único que podemos hacer es forzar que siempre sea una copia utilizando el parámetro `copy`.

Ejercicio 17: Dado dos objetos DataFrame `df1` y `df1`, comprobar si cambian al asignar el elmento $[0, 0]$ el valor -1 en sus correspondientes vistas como ndarray. En uno cambiará y en el otro no ¿puedes adivinar en cuál?

La salida esperada debe ser algo parecido a lo siguiente:

```
DataFrame df1
          a         b         c
0  0.636962  0.912756  0.815854
1  0.269787  0.606636  0.002739
2  0.040974  0.729497  0.857404
3  0.016528  0.543625  0.033586
4  0.813270  0.935072  0.729655
DataFrame df2
    A  B
0  1  a
1  2  b
2  3  c

DataFrame df1 después
          a         b         c
0 -1.000000  0.912756  0.815854
1  0.269787  0.606636  0.002739
2  0.040974  0.729497  0.857404
3  0.016528  0.543625  0.033586
4  0.813270  0.935072  0.729655
DataFrame df2 depués
    A  B
0  1  a
1  2  b
2  3  c

¿Quién ha cambiado? ¿Por qué?
```

In [19]:
```
gen = np.random.default_rng(0)
df1 = pd.DataFrame(dict(zip(list('abc'),
                            [gen.random(5) for i in range(5)])))
df2 = pd.DataFrame(dict(zip(list('AB'), [[1,2,3], ['a','b','c']])))

print("DataFrame df1\n", df1)
print("DataFrame df2\n", df2)

a1 = np.zeros(df1.shape)
a2 = np.zeros(df2.shape)
#Asignar a a1 la versión ndarray de df1 y a2 la correspondiente
#de df2.
#Pon tu código aquí.
#Sugerencia: usa el método to_numpy()

#endif

#Modificamos las veriones.
a1[0,0] = -1
a2[0,0] = -1

print("\nDataFrame df1 después\n", df1)
print("\nDataFrame df2 depués\n", df2)
print("¿Quién ha cambiado? ¿Por qué?")
```

```
DataFrame df1
           a         b         c
0  0.636962  0.912756  0.815854
1  0.269787  0.606636  0.002739
2  0.040974  0.729497  0.857404
3  0.016528  0.543625  0.033586
4  0.813270  0.935072  0.729655
DataFrame df2
    A  B
0  1  a
1  2  b
2  3  c

DataFrame df1 después
           a         b         c
0  0.636962  0.912756  0.815854
1  0.269787  0.606636  0.002739
2  0.040974  0.729497  0.857404
3  0.016528  0.543625  0.033586
4  0.813270  0.935072  0.729655

DataFrame df2 depués
    A  B
0  1  a
1  2  b
2  3  c
¿Quién ha cambiado? ¿Por qué?
```

Accediendo a los datos con el operador atributo `.`

Además de acceder a los datos usando el operador `[]`, pandas también ofrece la posibilidad de acceder a los mismos usando el operador `.`, es decir, como si fueran un atributo cuyo nombre es la etiqueta de clase correspondiente.

Ejercicio 18: Dada la una serie con las cotizaciones para varias empresas, se desea obtener usando el operador `.` la cotización de la empresa 'BANQIA'.

La salida esperada debe ser algo parecido a lo siguiente:

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Cotización de "BANQUIA": 0.88
```

In [ ]:
```
ibex = pd.Series({'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763})
print ('Cotizaciones:\n', ibex)
print('\nCotización de "BANQUIA": ', end='')
#Pon tu código aquí.
#Sugerencia: usa el operador atributo . en objeto serie como
# si la etiqueta 'BANQUIA' fuera un atributo.

#
```

Ejercicio 19: Dada un DataFrame `df` queremos obtener la serie correspondiente a la columna 'c'. Se debe utilizar el operador atributo `'.'`.

La salida esperada debe ser algo parecido a lo siguiente:

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644

Serie para columna 'c':
0    0.028320
1    0.124283
2    0.670624
3    0.647190
4    0.615385
5    0.383678
6    0.997210
7    0.980835
8    0.685542
9    0.650459
Name: c, dtype: float64
```

In [ ]:
```
gen = np.random.default_rng(0)
df = pd.DataFrame(dict(zip(list('abcde'),
                           [gen.random(10) for i in range(5)])))
print('DataFrame:\n', df)
print("\nSerie para columna 'c':")
#Pon tu código aquí
```

```
#Sugerencia: utiliza el operador . con la etiqueta de la
#   columna requerida.

#
```

Accediendo a los datos con los atributos `.loc` e `.iloc`.

Ya se ha mostrado cómo utilizar los operadores `[]` y `.` para acceder a los valores de un objeto Series o DataFrame. Estos métodos son intuitivos y fácil de utilizar en especial si estás ya familiarizado con usar un diccionario de Python o un ndarray de Numpy. Por ello estos métodos están indicados para trabajar con un objeto Series o DataFrame de forma interactiva en la terminal.

El precio a pagar por la facilidad de uso de estas alternativas es doble:

- Dependiendo del contexto, el objeto devuelto puede ser una referencia o una copia independiente (ver asignación encadenada).
- Se sufre una perdida de rendimiento en el acceso.

Sin embargo cuando se desea desarrollar programas buscamos conseguir gran desempecho en el acceso a los datos, en especial cuando los datasets son muy grandes. Para ello Pandas ofrece dos alternativas para superar los inconvenientes de los métodos de acceso vistos hasta ahora:

- El atributo `.loc[]` para acceder usando etiquetas de índice.
- El atributo `.iloc[]` para acceder usando posiciones de índice.
- Usar un objeto *callable* para indexar a aplicar con `.loc`, `.iloc` o `[]`.

Usando `.loc`

Usaremos el método `.loc[]` para acceder a los datos usando etiquetas de índice.

Ejercicio 20: Dada la una serie con las cotizaciones para varias empresas, se desea obtener usando el atributo `.loc` la cotización de la empresa 'BANQIA'.

La salida esperada debe ser algo parecido a lo siguiente:

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Cotización de "BANQUIA": 0.88
```

In [ ]:
```
ibex = pd.Series({'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763})
print ('Cotizaciones:\n', ibex)
print('\nCotización de "BANQUIA": ', end='')
#Pon tu código aquí.
#Sugerencia: usa el atributo .loc[] y la etiqueta 'BANQUIA'.

#
```

El atributo `.loc[]` en un Dataframe necesita dos argumentos, el primero para indicar la fila y el segundo

la columna.

Ejercicio 21: Dado el DataFrame 'ibex', obtén la 'Cotización' de la empresa 'SANTAN' (utiliza el método .loc).

La salida debería ser algo parecido a lo siguiente:

```
DataFrame:
          Cotización  Capitalización Categoría
BANQIA        0.880            2769         A
VVBA          2.892           19777        AA
SANTAN        2.076           35521         A
CAJANOR       1.763           10779         B

Contización de 'SANTAN': 2.076
```

In [ ]:
```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
            'CAJANOR':'B'}
ibex = pd.DataFrame({'Cotización':cot_dict, 'Capitalización':cap_dict,
                     'Categoría':cat_dict})
s = 0
print('DataFrame:\n', ibex)
print("\nContización de 'SANTAN': ", end='')
#Pon tu código aquí
#Sugerencia: utiliza el operador .loc[] con las etiquetas de fila y de
#   columna requeridas.

#
```

Observa que con el método `.loc` accedemos al DataFrame indicando fila y columna, es decir `df.loc[<fila>,<columna>]`, mientras que si usamos encadenamiento el orden es "columna, fila", esto es `df[<columna>][<fila>]`.

Con el método `.loc` también podemos indicar rangos (*slices*) para los ejes usando la notación `inicio:fin`, indicando con `:` todos los valores de ese eje (fila o columna).

Ejercicio 22: Dada un DataFrame `df` queremos obtener todos los valores correspondientes a la columna 'c' (utilizando el atributo `.loc[]`).

La salida esperada debe ser algo parecido a lo siguiente:

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644
```

```
Serie para columna 'c':
0    0.028320
1    0.124283
2    0.670624
3    0.647190
4    0.615385
5    0.383678
6    0.997210
7    0.980835
8    0.685542
9    0.650459
Name: c, dtype: float64
```

In [ ]:
```
gen = np.random.default_rng(0)
df = pd.DataFrame(dict(zip(list('abcde'),
                           [gen.random(10) for i in range(5)])))
print('DataFrame:\n', df)
print("\nValores de columna 'c':")
#Pon tu código aquí
#Sugerencia: utiliza el operador .loc con la etiqueta de la
#    columna requerida. Utiliza ':' para indicar todas las filas.

#
```

Ejercicio 22b: Dada un DataFrame `df` queremos obtener todos los valores correspondientes a la fila 6 (utilizando el atributo `.loc[]`).

La salida esperada debe ser algo parecido a lo siguiente:

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644

Serie para columna 'c':
0    0.028320
1    0.124283
2    0.670624
3    0.647190
4    0.615385
5    0.383678
6    0.997210
7    0.980835
8    0.685542
9    0.650459
Name: c, dtype: float64
```

In [ ]:
```
gen = np.random.default_rng(0)
df = pd.DataFrame(dict(zip(list('abcde'),
                           [gen.random(10) for i in range(5)])))
```

```
print('DataFrame:\n', df)
print("\nValores de fila 6: ")
#Pon tu código aquí
#Sugerencia: utiliza el operador .loc con la etiqueta de la
#    fila requerida. Utiliza ':' para indicar todas las columnas.

#
```

Ejercicio 22c: Dada un DataFrame `df` queremos obtener todos los valores correspondientes a las filas 2 a 5 de las columnas 'b' a 'c' (utilizando el atributo `.loc[]`).

La salida esperada debe ser algo parecido a lo siguiente:

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644

Sub DataFrame:
          b         c
2  0.857404  0.670624
3  0.033586  0.647190
4  0.729655  0.615385
5  0.175656  0.383678
```

In [ ]:
```
gen = np.random.default_rng(0)
df = pd.DataFrame(dict(zip(list('abcde'),
                           [gen.random(10) for i in range(5)])))
print('DataFrame:\n', df)
print("\nSub DataFrame:")
#Pon tu código aquí
#Sugerencia: utiliza el operador .loc con los rangos de etiqueta de
#   fila y columna indicados. Utiliza 'incio:fin' para indicar los rangos.

#
```

También podemos seleccionar individualmente un conjunto de filas/columnas mediante:

- Un array de etiquetas.
- Un array máscara de tipo lógico (True|False).

Ejercicio 22d: Dada un DataFrame `df` queremos obtener las filas impares de las columnas 'a' y 'd'. (utilizando el atributo `.loc[]`).

La salida esperada debe ser algo parecido a lo siguiente:

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
1  0.269787  0.002739  0.124283  0.388921  0.321869
```

```
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644

Sub DataFrame:
          a         d
1  0.269787  0.388921
3  0.016528  0.721488
5  0.912756  0.310242
7  0.729497  0.889488
9  0.935072  0.357795
```

In [ ]:
```
gen = np.random.default_rng(0)
df = pd.DataFrame(dict(zip(list('abcde'),
                           [gen.random(10) for i in range(5)])))
print('DataFrame:\n', df)
print("\nSub DataFrame:")
#Pon tu código aquí
#Sugerencia: utiliza el operador .loc con la lista de etiquetas de columna
#   indicadas. Utiliza ':' para indicar todas las filas.

#
```

Por último vamos a comparar modificar un dataframe usando encadenamiento `[][]` con utilizar el atributo `.loc[]`.

Ejercicio 23: Dado el DataFrame 'ibex', cambiar la 'Cotización' de la empresa 'SANTAN' al valor 1.0. Utiliza el método .loc, y su capitalización al valor 40000 usando encadenamiento `[columna][fila]`.

La salida debería ser algo parecido a lo siguiente:

```
DataFrame:
          Cotización  Capitalización Categoría
BANQIA         0.880            2769         A
VVBA           2.892           19777        AA
SANTAN         2.076           35521         A
CAJANOR        1.763           10779         B

DataFrame modificado:
          Cotización  Capitalización Categoría
BANQIA         0.880            2769         A
VVBA           2.892           19777        AA
SANTAN         1.000           40000         A
CAJANOR        1.763           10779         B
:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  ibex['Capitalización']['SANTAN'] = 40000
```

In [ ]:
```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
```

```
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
            'CAJANOR':'B'}
ibex = pd.DataFrame({'Cotización':cot_dict, 'Capitalización':cap_dict,
                     'Categoría':cat_dict})
s = 0
print('DataFrame:\n', ibex)

#Cambiar la cotización de SANTAN a 1.0
#Pon tu código aquí
#Sugerencia: utiliza el atributo .loc[]

#

#Cambiar la capitalización de SANTAN a 40000
#Pon tu código aquí
#Sugerencia: utiliza encadenamiento [][]

#

print('\nDataFrame modificado:\n', ibex)
```

Usando `.iloc`

Hay ocasiones en las cuales puede ser más cómodo procesar un dataset usando posiciones de índice en vez de etiquetas. Supongamos un objeto Series `s` con un índice de etiquetas `['a', 'b', 'c', 'd']`. Ya sabemos que podemos obtener el valor asociado a la etiqueta `'b'` de la forma `s['b']` o también con `s.loc['b']`. La etiqueta `'b'` tiene la posición de indice `1`. No podemos hacer `s[1]` ni `s.loc[1]` ya que `1` no es una etiqueta válida del índice (generará una excepción `KeyError`).

Si queremos acceder usando posiciones de índice, usaremos el método .iloc.

Ejercicio 24: Dada la una serie con las cotizaciones para varias empresas, se desea obtener usando el atributo `.iloc[]` la cotización de la empresa 'VVBA'. Observa que la etiqueta 'VVBA' ocupa la posición de índice `1`.

La salida esperada debe ser algo parecido a lo siguiente:

```
Cotizaciones:
 BANQIA     0.880
VVBA       2.892
SANTAN     2.076
CAJANOR    1.763
dtype: float64

Cotización de "VVBA": 2.892
```

In [ ]:
```
ibex = pd.Series({'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763})
print ('Cotizaciones:\n', ibex)
print('\nCotización de "VVBA": ', end='')
#Pon tu código aquí.
#Sugerencia: usa el atributo .iloc[] con la posición de índice asociada
#   a la etiqueta requerida.

#
```

Ejercicio 25: Dado el DataFrame `ibex`, obtén la 'Capitalización' de la empresa 'SANTAN' utilizando el atributo `.iloc[]`. Observa que 'Capitalización' tiene la posición `1` en el índice de columnas y 'SANTAN' tiene la posición `2` en el índice de columnas.

La salida debería ser algo parecido a lo siguiente:

```
DataFrame:
          Cotización  Capitalización Categoría
BANQIA         0.880            2769         A
VVBA           2.892           19777        AA
SANTAN         2.076           35521         A
CAJANOR        1.763           10779         B

Contización de 'SANTAN': 2.076
```

In [ ]:
```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
            'CAJANOR':'B'}
ibex = pd.DataFrame({'Cotización':cot_dict, 'Capitalización':cap_dict,
                     'Categoría':cat_dict})
s = 0
print('DataFrame:\n', ibex)
print("\nCapitalización de 'SANTAN': ", end='')
#Pon tu código aquí
#Sugerencia: utiliza el operador .iloc[] con las posiciones de índice de
#   las etiquetas de fila y columna requeridas.

#
```

Con el método `.iloc` también podemos indicar rangos (*slices*) para los ejes usando la notación `posición-inicio:posición-fin`, indicando con `:` todos los valores de ese eje (fila o columna). Observa que de forma distinta con rangos de etiquetas usados con `.loc`, en el rango `posicion-inicio:posicion-fin`, la `posición-fin` no se incluye en el rango.

Ejercicio 26: Dado un DataFrame `df` queremos obtener la serie correspondiente a la columna `'c'`, utilizando el atributo `.iloc[]`. Observa que la etiqueta `'c'` tiene la posición de índice `2`.

La salida esperada debe ser algo parecido a lo siguiente:

```
DataFrame:
           a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644

Serie para columna 'c':
0    0.028320
```

```
1    0.124283
2    0.670624
3    0.647190
4    0.615385
5    0.383678
6    0.997210
7    0.980835
8    0.685542
9    0.650459
Name: c, dtype: float64
```

In [ ]:
```
gen = np.random.default_rng(0)
df = pd.DataFrame(dict(zip(list('abcde'),
                           [gen.random(10) for i in range(5)])))
print('DataFrame:\n', df)
print("\nSerie para columna 'c':")
#Pon tu código aquí
#Sugerencia: utiliza el atributo .iloc[] con la posición de índice de
#   columna correspondiente a la etiqueta requerida. Recuerda que queremos
#   los valores de la columna.

#
```

Ejercicio 26b: Dado un DataFrame `df` queremos obtener la serie correspondiente a la fila `'C'`, utilizando el atributo `.iloc[]`. Observa que la etiqueta `'C'` tiene la posición de índice `2`.

La salida esperada debe ser algo parecido a lo siguiente:

```
DataFrame:
          a         b         c         d         e
A  0.636962  0.815854  0.028320  0.688447  0.571530
B  0.269787  0.002739  0.124283  0.388921  0.321869
C  0.040974  0.857404  0.670624  0.135097  0.594300
D  0.016528  0.033586  0.647190  0.721488  0.337911
E  0.813270  0.729655  0.615385  0.525354  0.391619
F  0.912756  0.175656  0.383678  0.310242  0.890274
G  0.606636  0.863179  0.997210  0.485835  0.227158
H  0.729497  0.541461  0.980835  0.889488  0.623187
I  0.543625  0.299712  0.685542  0.934044  0.084015
J  0.935072  0.422687  0.650459  0.357795  0.832644

Serie para fila 'C':
a    0.040974
b    0.857404
c    0.670624
d    0.135097
e    0.594300
Name: C, dtype: float64
```

In [ ]:
```
gen = np.random.default_rng(0)
df = pd.DataFrame(dict(zip(list('abcde'),
                           [gen.random(10) for i in range(5)])), index=list('ABCDEFG
print('DataFrame:\n', df)
print("\nSerie para fila 'C':")
#Pon tu código aquí
#Sugerencia: utiliza el atributo .iloc[] con la posición de índice de
#   fila correspondiente a la etiqueta requerida. Recuerda que queremos
#   los valores de toda la fila.

#
```

Ejercicio 26c: Dado un DataFrame `df` queremos obtener el subconjunto de datos correspondientes a las filas `'B'` a `'E'`, columnas `'b'` a `'d'` utilizando el atributo `.iloc[]`. Observa que en los rangos de posiciones, la 'posición-fin' no se incluye en el rango.

La salida esperada debe ser algo parecido a lo siguiente:

```
DataFrame:
          a         b         c         d         e
A  0.636962  0.815854  0.028320  0.688447  0.571530
B  0.269787  0.002739  0.124283  0.388921  0.321869
C  0.040974  0.857404  0.670624  0.135097  0.594300
D  0.016528  0.033586  0.647190  0.721488  0.337911
E  0.813270  0.729655  0.615385  0.525354  0.391619
F  0.912756  0.175656  0.383678  0.310242  0.890274
G  0.606636  0.863179  0.997210  0.485835  0.227158
H  0.729497  0.541461  0.980835  0.889488  0.623187
I  0.543625  0.299712  0.685542  0.934044  0.084015
J  0.935072  0.422687  0.650459  0.357795  0.832644

Subconjunto de datos:
          b         c         d
B  0.002739  0.124283  0.388921
C  0.857404  0.670624  0.135097
D  0.033586  0.647190  0.721488
E  0.729655  0.615385  0.525354
```

In [ ]:
```
gen = np.random.default_rng(0)
df = pd.DataFrame(dict(zip(list('abcde'),
                           [gen.random(10) for i in range(5)])), index=list('ABCDEFG
print('DataFrame:\n', df)
print("\nSubconjunto de datos:")
#Pon tu código aquí
#Sugerencia: utiliza el atributo .iloc[] usando los rangos de fila y columna
#  requeridos. Recuerda que en los rangos de posiciones, la 'posición-fin'
#  no se incluye en el rango.

#
```

También podemos seleccionar individualmente un conjunto de filas/columnas mediante:

- Un array de posiciones de indices.
- Un array máscara de tipo lógico (True|False).

Ejercicio 26d: Dado un DataFrame `df` queremos obtener el subconjunto de datos correspondientes a las filas `'B'` y `'E'`, columnas `'a'` y `'d'` utilizando el atributo `.iloc[]`.

La salida esperada debe ser algo parecido a lo siguiente:

```
DataFrame:
          a         b         c         d         e
A  0.636962  0.815854  0.028320  0.688447  0.571530
B  0.269787  0.002739  0.124283  0.388921  0.321869
C  0.040974  0.857404  0.670624  0.135097  0.594300
D  0.016528  0.033586  0.647190  0.721488  0.337911
E  0.813270  0.729655  0.615385  0.525354  0.391619
F  0.912756  0.175656  0.383678  0.310242  0.890274
```

```
G  0.606636  0.863179  0.997210  0.485835  0.227158
H  0.729497  0.541461  0.980835  0.889488  0.623187
I  0.543625  0.299712  0.685542  0.934044  0.084015
J  0.935072  0.422687  0.650459  0.357795  0.832644

Subconjunto de datos:
          a         d
B  0.269787  0.388921
E  0.813270  0.525354
```

In [ ]:
```
gen = np.random.default_rng(0)
df = pd.DataFrame(dict(zip(list('abcde'),
                           [gen.random(10) for i in range(5)])),
                  index=list('ABCDEFGHIJ'))
print('DataFrame:\n', df)
print("\nSubconjunto de datos:")
#Pon tu código aquí
#Sugerencia: utiliza el atributo .iloc[] usando listas de posiciones de fila
#  y columna requeridas.

#
```

Usando un objeto *callable* para indexar.

Un objeto *callable* es un objeto que puede actuar como función que recibe un parámetro, el objeto Series o Dataframe que vamos a indexar, y devuelve algo que puede ser utilizado para indexar. Si se devuelve un array de valores lógicos (máscaras) o un array de etiquetas utilizaremos el atributo `.loc[]`. Si se devuelve un array de posiciones utilizaremos el atributo `.iloc[]`.

Ejemplos de objeto *callable* pueden ser entre otros:

- Una función definida (tiene nombre).
- Una expresión lambda (función sin nombre).
- Un objeto que tenga definido el método `__call__`. Este tipo de objetos (llamados *functores*) pueden ser utilizados como si fueran funciones, es decir, si el variable `obj` referencia un objeto functor podremos escribir: `obj(df)`. Observa que `obj` parece una función.

Ejercicio 27: Dado un DataFrame `df` queremos obtener el subconjunto dónde los valores de la columna 'c' sean ≤ 0.5. Utilizar una función para indicar este criterio de selección. Observa que como el resultado de la función criterio será un array de valores lógicos deberemos utilizar el atributo `.loc[]`.

```
DataFrame:
          a         b         c         d         e
0  0.636962  0.815854  0.028320  0.688447  0.571530
1  0.269787  0.002739  0.124283  0.388921  0.321869
2  0.040974  0.857404  0.670624  0.135097  0.594300
3  0.016528  0.033586  0.647190  0.721488  0.337911
4  0.813270  0.729655  0.615385  0.525354  0.391619
5  0.912756  0.175656  0.383678  0.310242  0.890274
6  0.606636  0.863179  0.997210  0.485835  0.227158
7  0.729497  0.541461  0.980835  0.889488  0.623187
8  0.543625  0.299712  0.685542  0.934044  0.084015
9  0.935072  0.422687  0.650459  0.357795  0.832644

Sub conjunto 'c'<=0.5:
          a         b         c         d         e
```

```
   0  0.636962  0.815854  0.028320  0.688447  0.571530
   1  0.269787  0.002739  0.124283  0.388921  0.321869
   5  0.912756  0.175656  0.383678  0.310242  0.890274
```

In [ ]:
```
gen = np.random.default_rng(0)
df = pd.DataFrame(dict(zip(list('abcde'),
                           [gen.random(10) for i in range(5)])))

def criterio( df ):
    '''Seleccionar filas de df cuando el valor de la columna 'c'
    es <= 0.5'''
    return df['c']<=0.5

print('DataFrame:\n', df)
print("\nSub conjunto donde 'c'<=0.5:")
#Pon tu código aquí
#Sugerencia: utiliza el atributo .loc[] y la función que
#  define el criterio.
#Sugerencia: puedes hacer lo mismo usando una expresión lambda?

#
```

Accediendo rápido a un elemento de datos.

Los métodos vistos hasta aquí para acceder a un objeto Series o DataFrame permiten seleccionar un elemento de datos (*un valor escalar*) pero también un rango (*slice*) de datos. Esta flexibilidad implica una sobrecarga previa ya que Pandas debe detectar la forma elegida para acceder en cada momento.

Cuando lo que nos intersa es acceder/modificar únicamente un elemento de datos (una posición de una serie o una celda de un dataframe), se recomienda usar los atributos `.at[]` o `.iat[]` ya que el acceso será más eficiente.

Similarmente a `.loc` e `.iloc`, usaremos `.at[]` si accedemos usando etiquetas de índice y usaremos `.iat[]` si accedemos usando posiciones de índice.

Ejercicio 28: Dado el DataFrame 'ibex', obtén la 'Capitalización' de la empresa 'SANTAN' (utiliza el método `.at[]`).

La salida debería ser algo parecido a lo siguiente:

```
DataFrame:
          Cotización  Capitalización Categoría
BANQIA         0.880            2769         A
VVBA           2.892           19777        AA
SANTAN         2.076           35521         A
CAJANOR        1.763           10779         B

Capitalización de 'SANTAN': 35521
```

In [ ]:
```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
            'CAJANOR':'B'}
ibex = pd.DataFrame({'Cotización':cot_dict, 'Capitalización':cap_dict,
                     'Categoría':cat_dict})
s = 0
```

```
print('DataFrame:\n', ibex)
print("\nCapitalización de 'SANTAN': ", end='')
#Pon tu código aquí
#Sugerencia: utiliza el atributo .at[] con las etiquetas
#   de fila y columna indicadas.

#
```

Ejercicio 28: Dado el DataFrame 'ibex', obtén la 'Categoría' de la empresa 'CAJANOR' (utiliza el método `.iat[]`).

La salida debería ser algo parecido a lo siguiente:

```
DataFrame:
          Cotización  Capitalización Categoría
BANQIA         0.880            2769         A
VVBA           2.892           19777        AA
SANTAN         2.076           35521         A
CAJANOR        1.763           10779         B

Categoría de 'CAJANOR': B
```

In [ ]:
```
cot_dict = {'BANQIA':0.88, 'VVBA':2.892, 'SANTAN':2.076,
            'CAJANOR':1.763}
cap_dict = {'BANQIA':2769, 'VVBA':19777, 'SANTAN':35521,
            'CAJANOR':10779}
cat_dict = {'BANQIA':'A', 'VVBA':'AA', 'SANTAN':'A',
            'CAJANOR':'B'}
ibex = pd.DataFrame({'Cotización':cot_dict, 'Capitalización':cap_dict,
                     'Categoría':cat_dict})
s = 0
print('DataFrame:\n', ibex)
print("\nCategoría de 'CAJANOR': ", end='')
#Pon tu código aquí
#Sugerencia: utiliza el atributo .iat[] con las posiciones de
#  indice de fila y columna indicados.

#
```