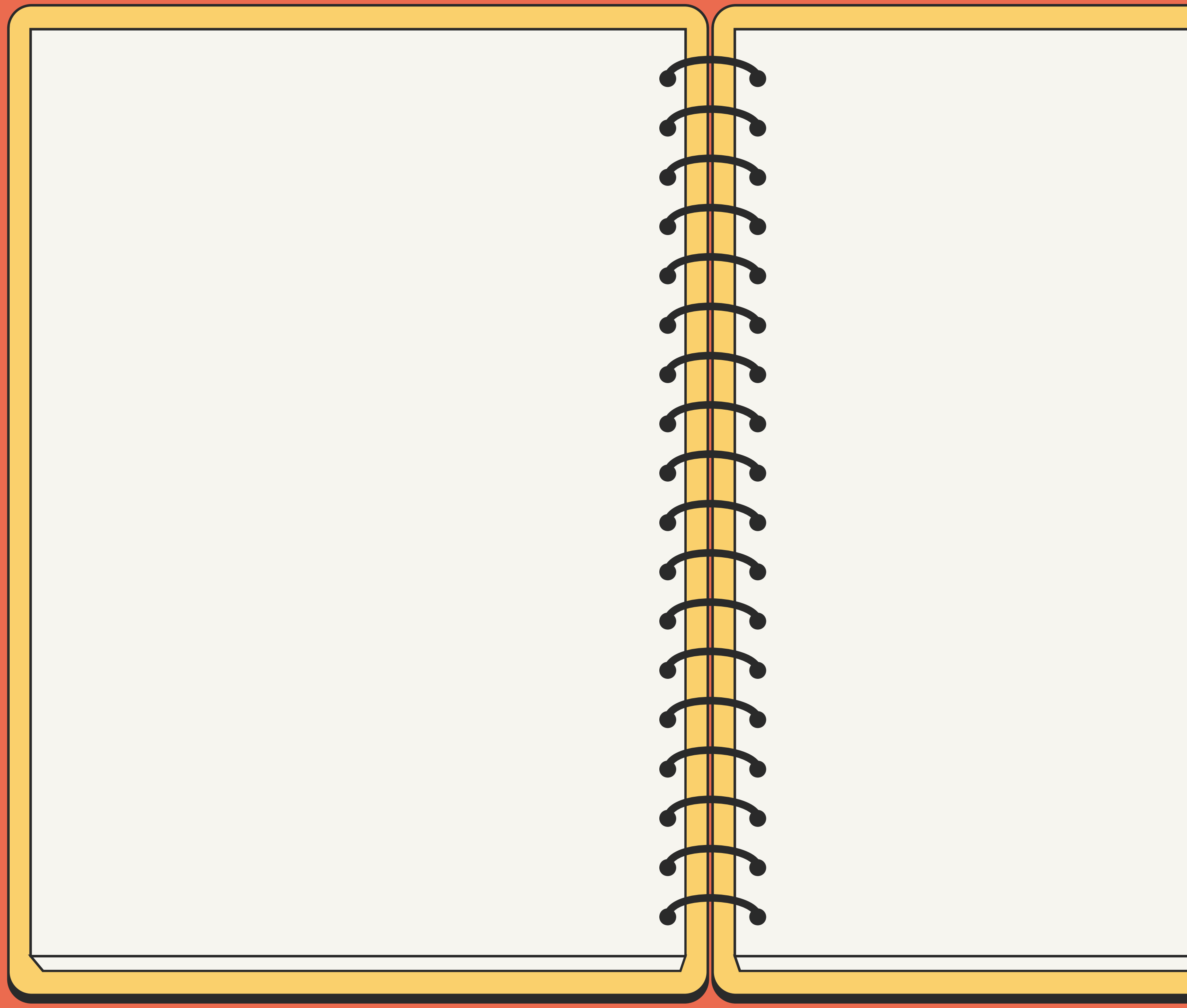
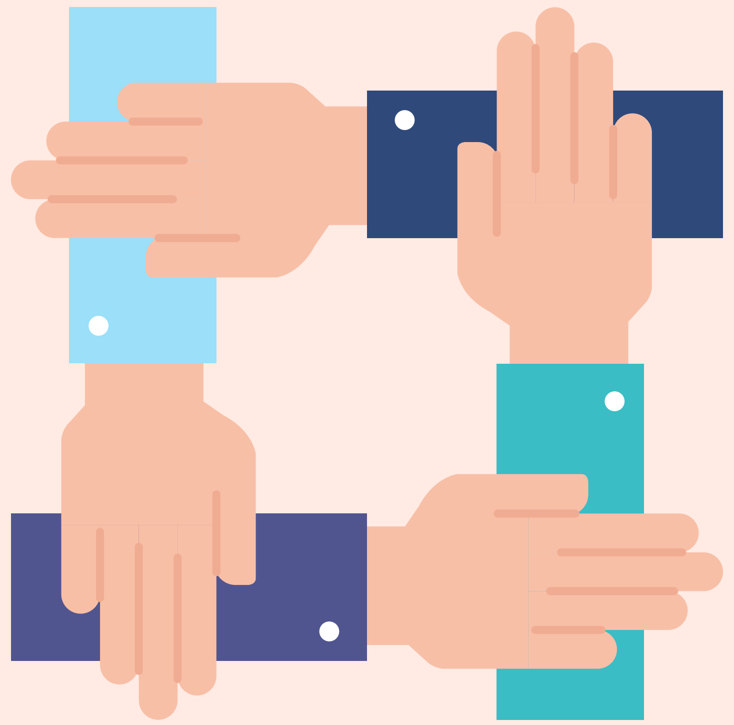
DATA MANIPULATION WITH PYTHON

By Matplotlib Group





Our Team

1. Tobias Mikha Sulistiyo
2. Daud Ibadurahman
3. Fitri Alfaqrina
4. Putri Reghina Hilmi P.
5. Muhammad Iqbal Rustan

Use Case : Data Manipulation with Python

Use Case Summary

Objective Statement

- Get new insight about the fundamental of data manipulation with Python
- Get the insight about object series
- Get the insight about loc and iloc
- Get the insight about data frame in Python

Challenges

- Python is a case sensitive programming laguage, so the writing of the syntax must be considered
- A lot of new things about data manipulation will be learned here, so it needs a detailed and easy understand explanation

Benefits

- As a basic material before manipulating actual data using Python
- As additional knowledge about object series, loc and iloc, and data frame in Python

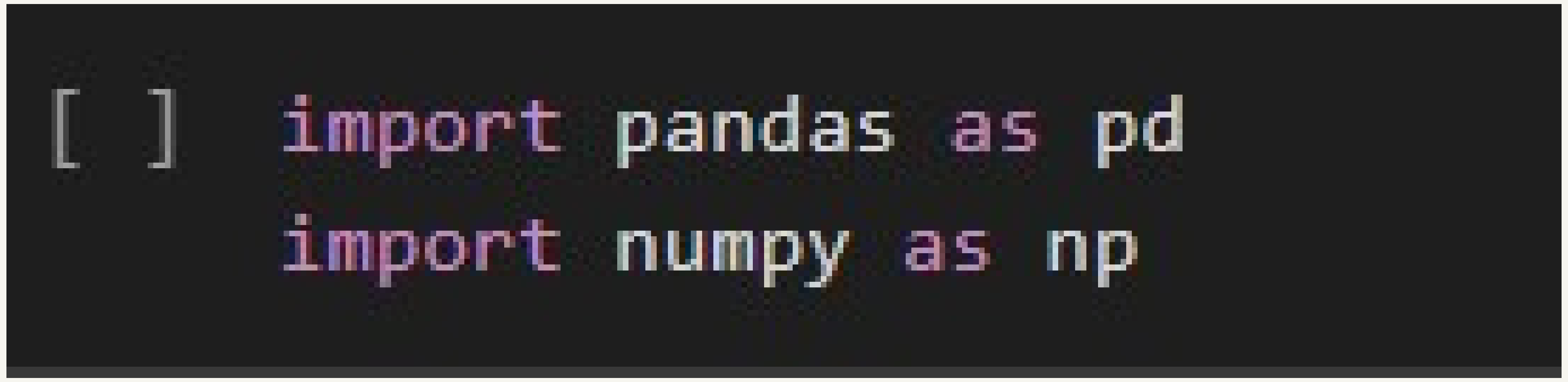
Expected Outcome

- Understand the basics knowledge that needed to manipulate data using python
- Knowing the difference between implisit and explisit index
- Knowing the difference between loc and iloc
- Knowing how to convert data into object series and data frame

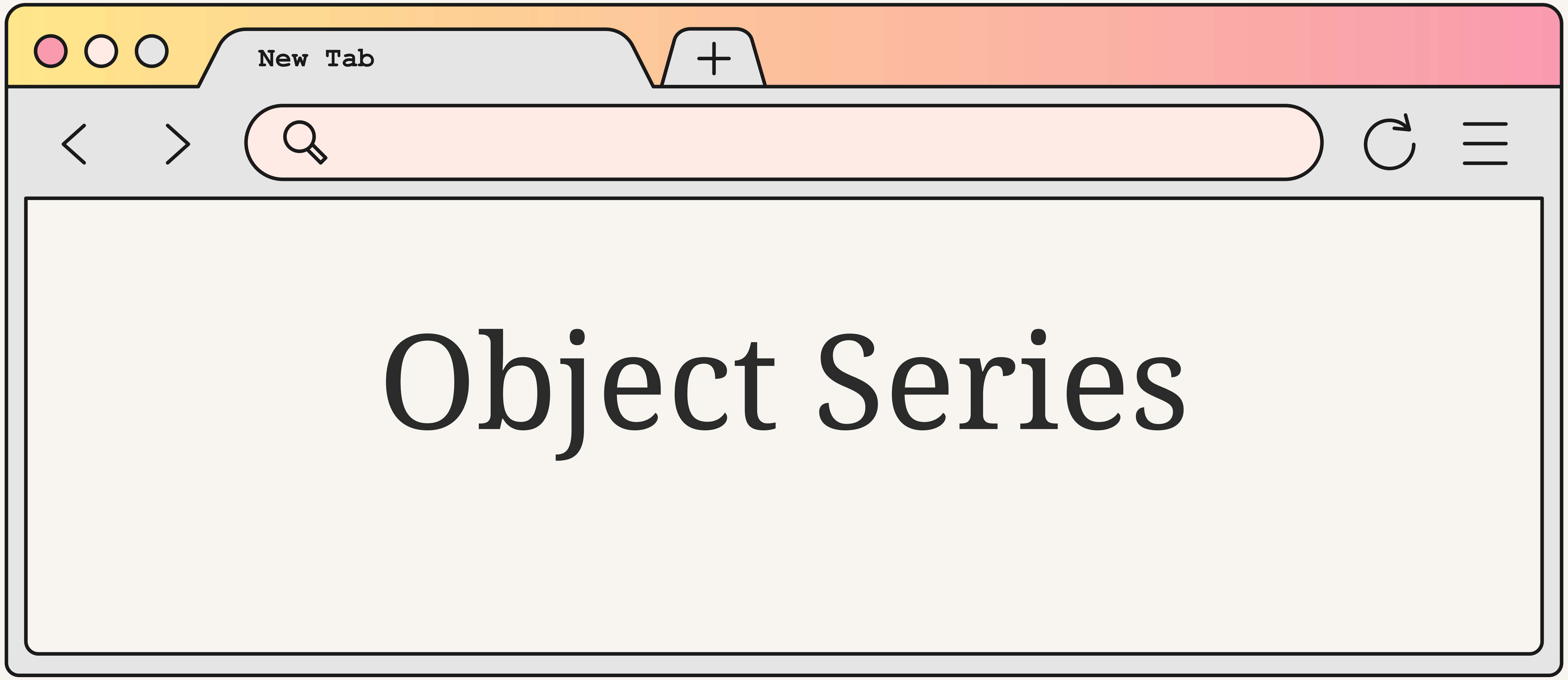
Data Manipulation

We will do manipulation data but it's not to change the data value. Data manipulation make this data easier to read by machine.

For the first stage, import the libraries required in data manipulation with Python. The librarys packages to be imported are Pandas and Numpy. Pandas have two objects (series and data frames).



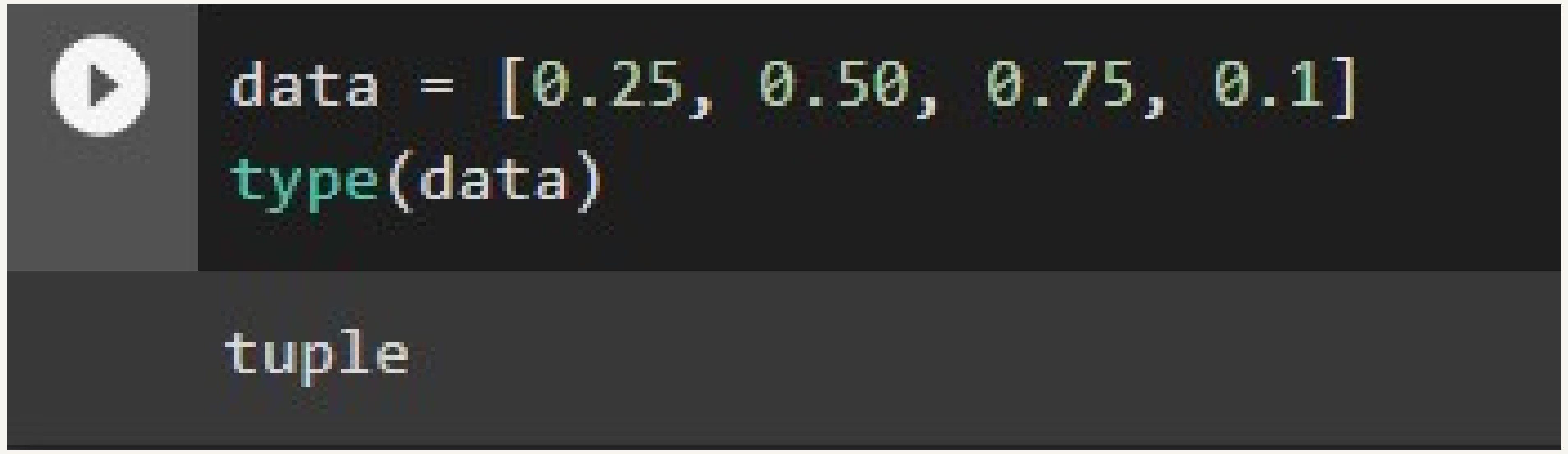
```
[ ]  import pandas as pd
     import numpy as np
```



Object Series

Object Series

- Has one dimension of data.
- Only has one column.
- Does not have a column name.
- Has an index.
- Can change the data (example: list or tuple type) into object series.

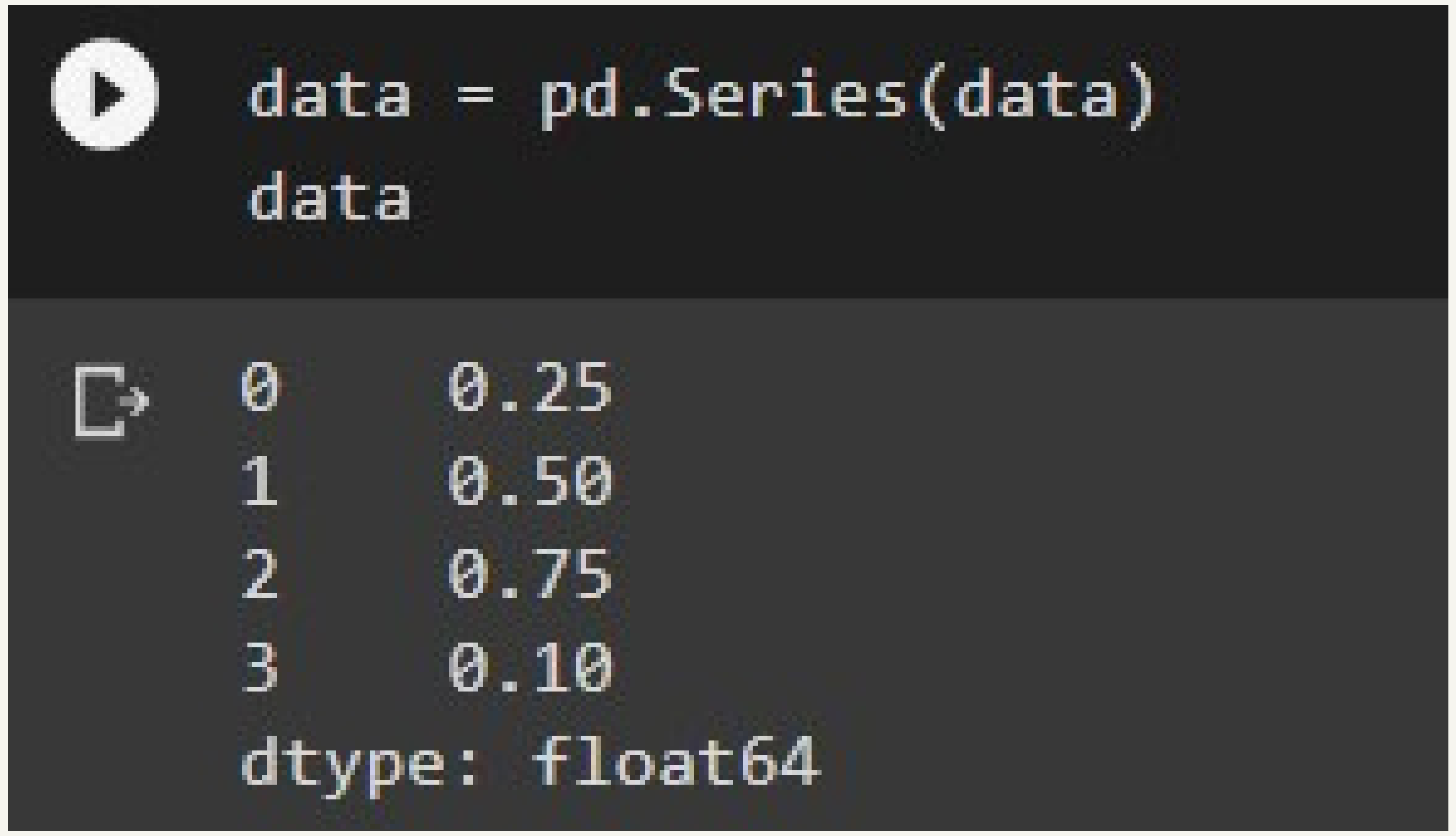


```
data = [0.25, 0.50, 0.75, 0.1]
type(data)
```

```
tuple
```

Enter the data to be processed into an object series.

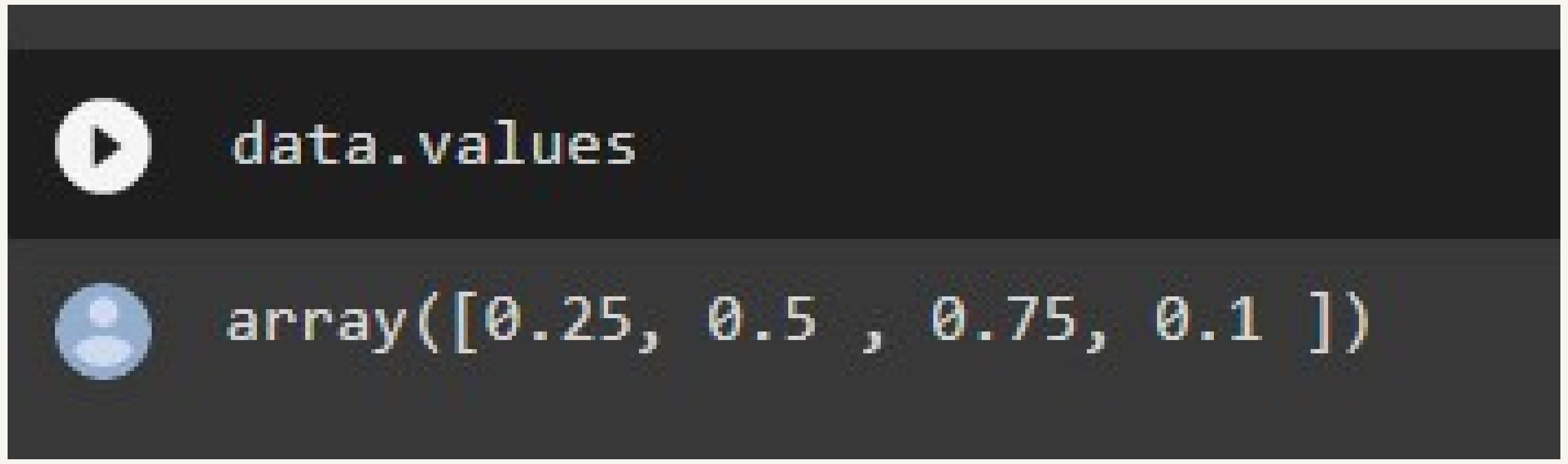
Convert data into object series with using the Pandas library (pd). The data will be has an index only has one column.



```
data = pd.Series(data)
data
```

```
0    0.25
1    0.50
2    0.75
3    0.10
dtype: float64
```

You can also convert the object series into a data array.



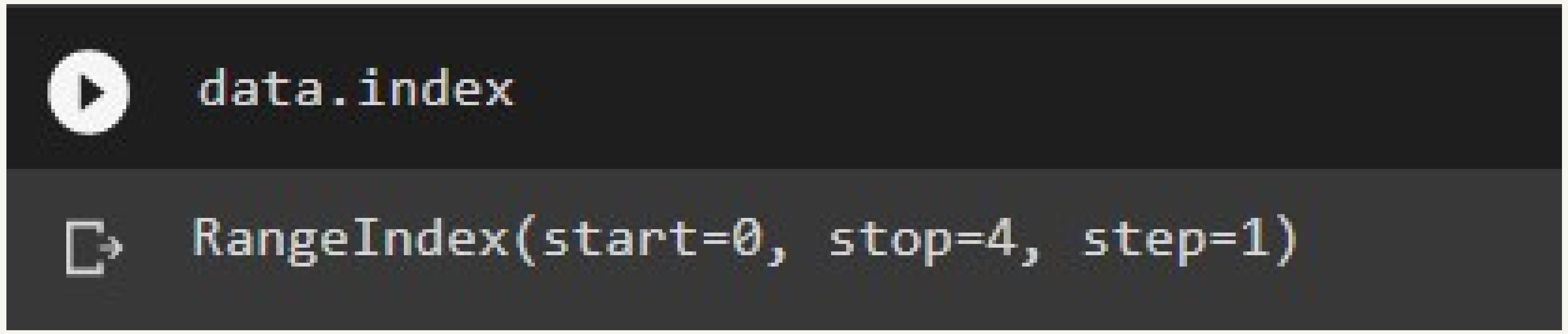
```
data.values
```

```
array([0.25, 0.5 , 0.75, 0.1 ])
```

In some cases of data processing, the data is required to be converted into an data array before the calculation operation (ex: matrix) is performed on the data.

Showing an index

The index is in the form of a range, where the starting point is inclusive in the range and the stop point is exclusive of the range.

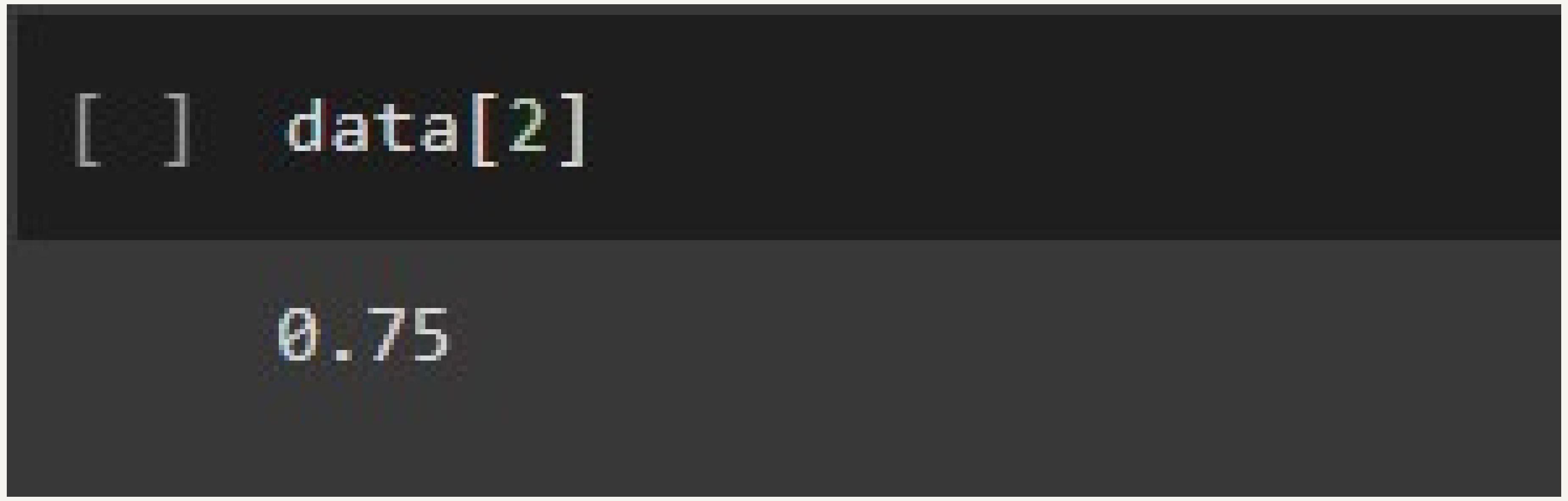


```
data.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

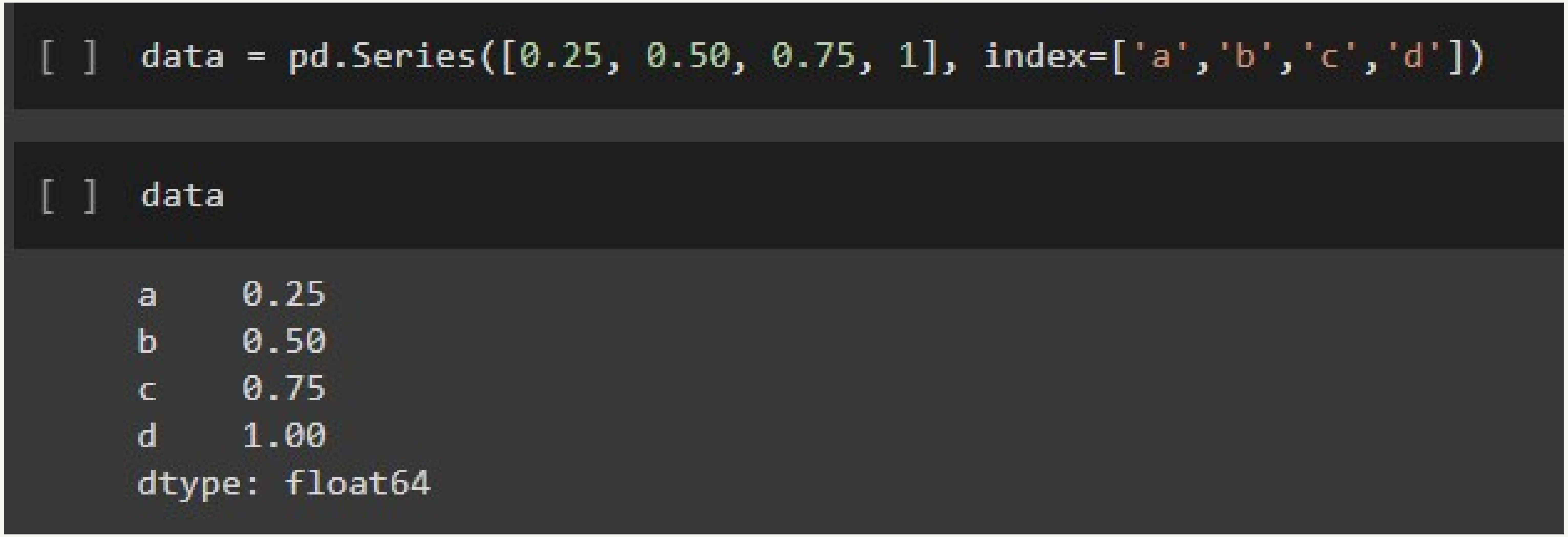
Call data with indexing

It can be seen that the index starts from 0 and stops at 4 with the difference between each index which is 1. **Implicit index is Python's default index (0, 1, 2, 3, etc)**. Based on the previous data series, it can be seen that the index 2 is 0.75



```
[ ]  data[2]

     0.75
```

We can define an index. This is commonly called an explicit index or a defined index. When defining an index, the number of indexes should be equal to the amount of data.

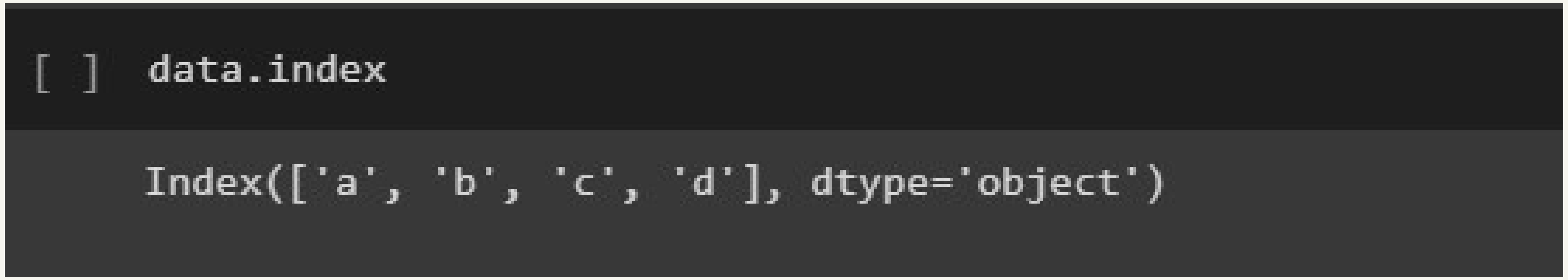


```
[ ]  data = pd.Series([0.25, 0.50, 0.75, 1], index=['a','b','c','d'])
```

```
[ ]  data
```

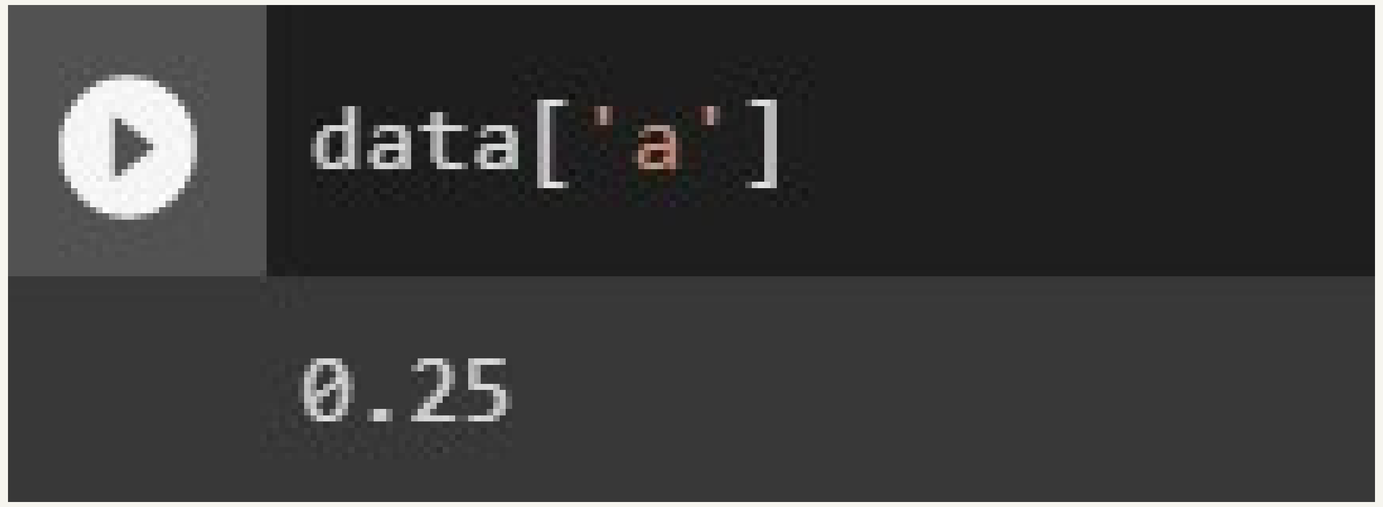
```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```

We can check the explicit index that we have defined.

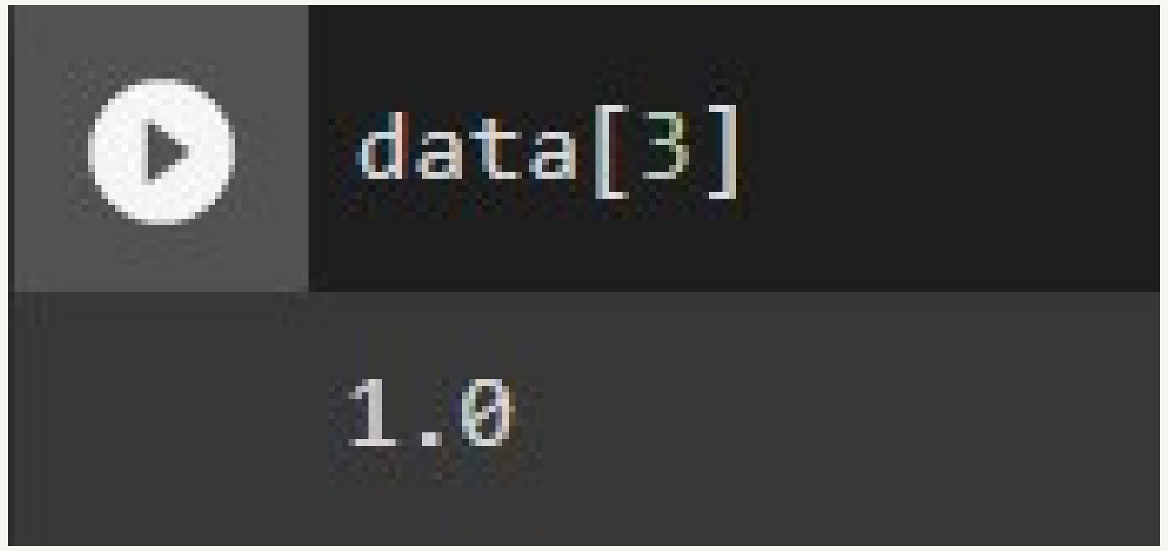


```
[ ]  data.index

     Index(['a', 'b', 'c', 'd'], dtype='object')
```

Calling data using its implicit index is an understanding of data selection.



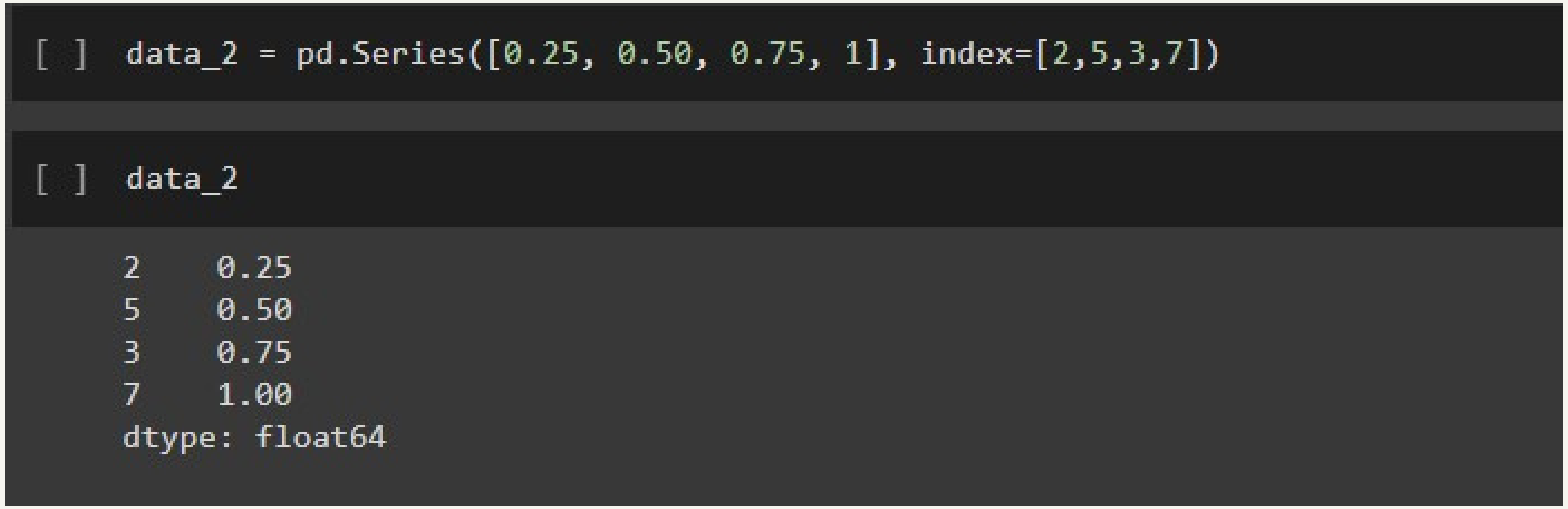
```
data['a']

0.25
```



```
data[3]

1.0
```

Even if we've made the index explicit, we can still call the implicit index. We try to call the the implicit index 3.

If the implicit index and explicit index are the same type, then the data call will rely solely on its explicit index.

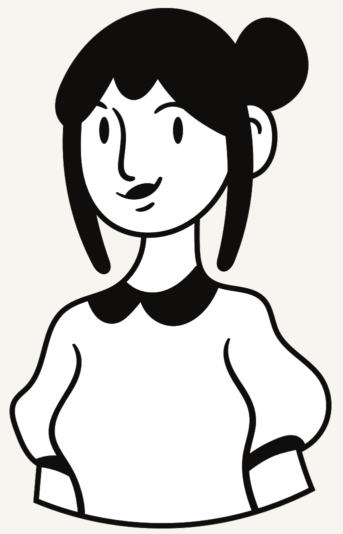


```
[ ]  data_2 = pd.Series([0.25, 0.50, 0.75, 1], index=[2,5,3,7])
```

```
[ ]  data_2
```

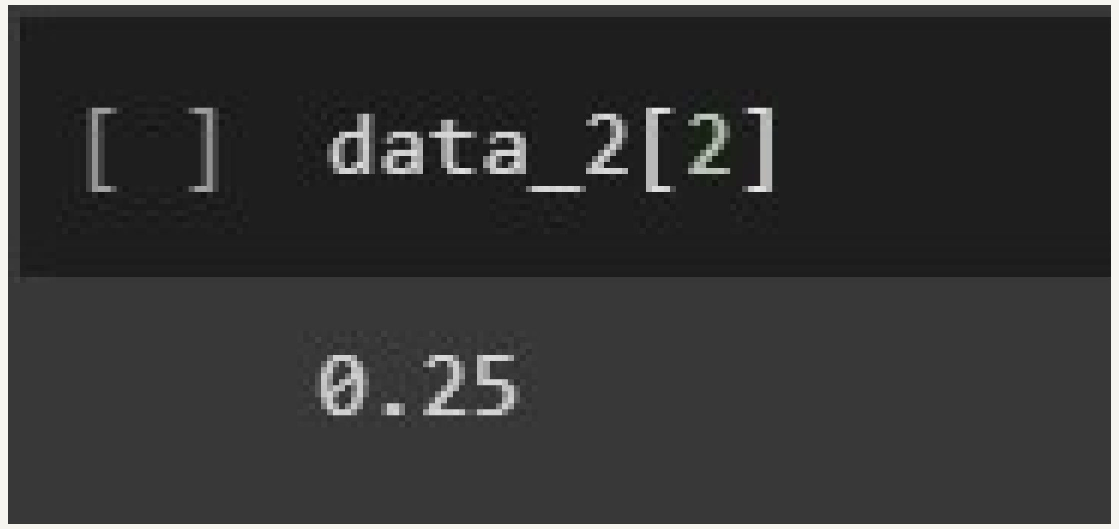
```
2    0.25
5    0.50
3    0.75
7    1.00
dtype: float64
```

We try to index with random numbers



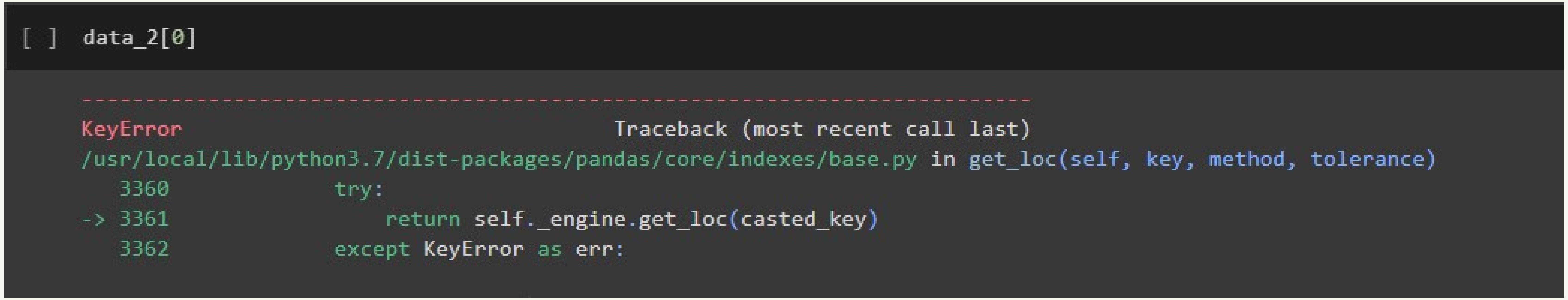
It appears that the indexing order will follow according to the indexing input entered.

We try to call the data_2 by entering an index 2. It can be seen that what is displayed is an explicit index of the data_2.



```
[ ]  data_2[2]

     0.25
```

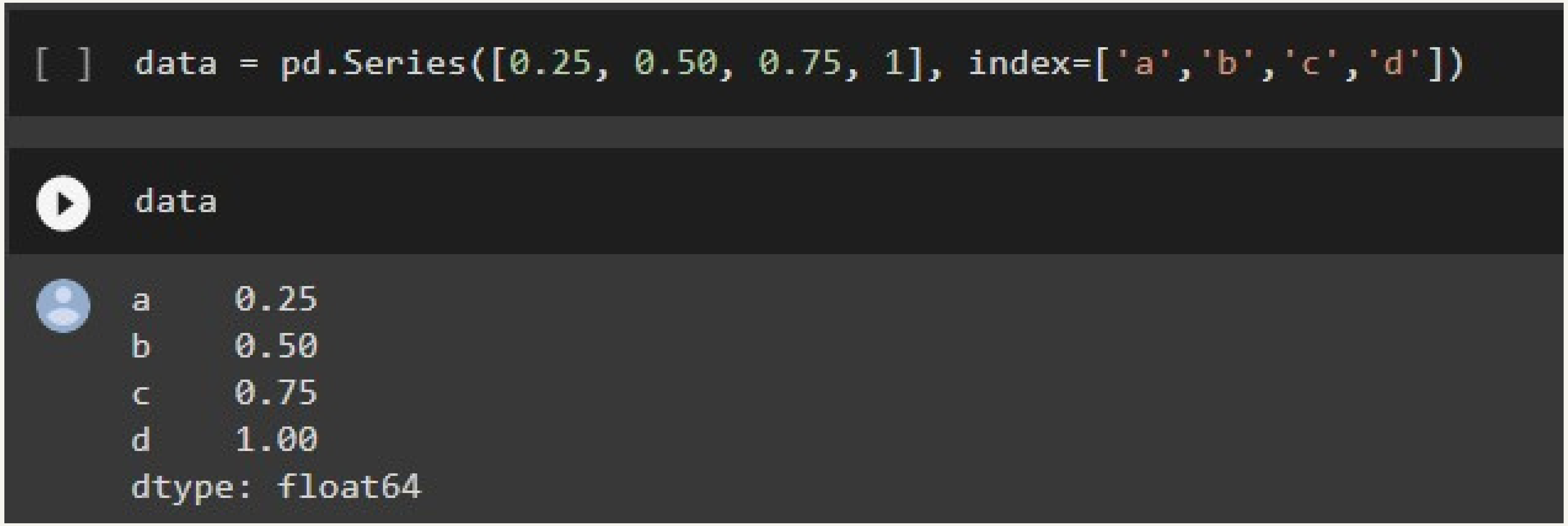
Now we try to calling its index 0 on the data_2. Because in the explicit index there is no index 0, an error will appear.



```
[ ]  data_2[0]

     ---------------------------------------------------------------------------
     KeyError                                  Traceback (most recent call last)
     /usr/local/lib/python3.7/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
        3360             try:
     -> 3361                 return self._engine.get_loc(casted_key)
        3362             except KeyError as err:
```

Data Slicing

Slicing is a technique of selecting data from a data set.
Now we enter the data by doing defined on the index.

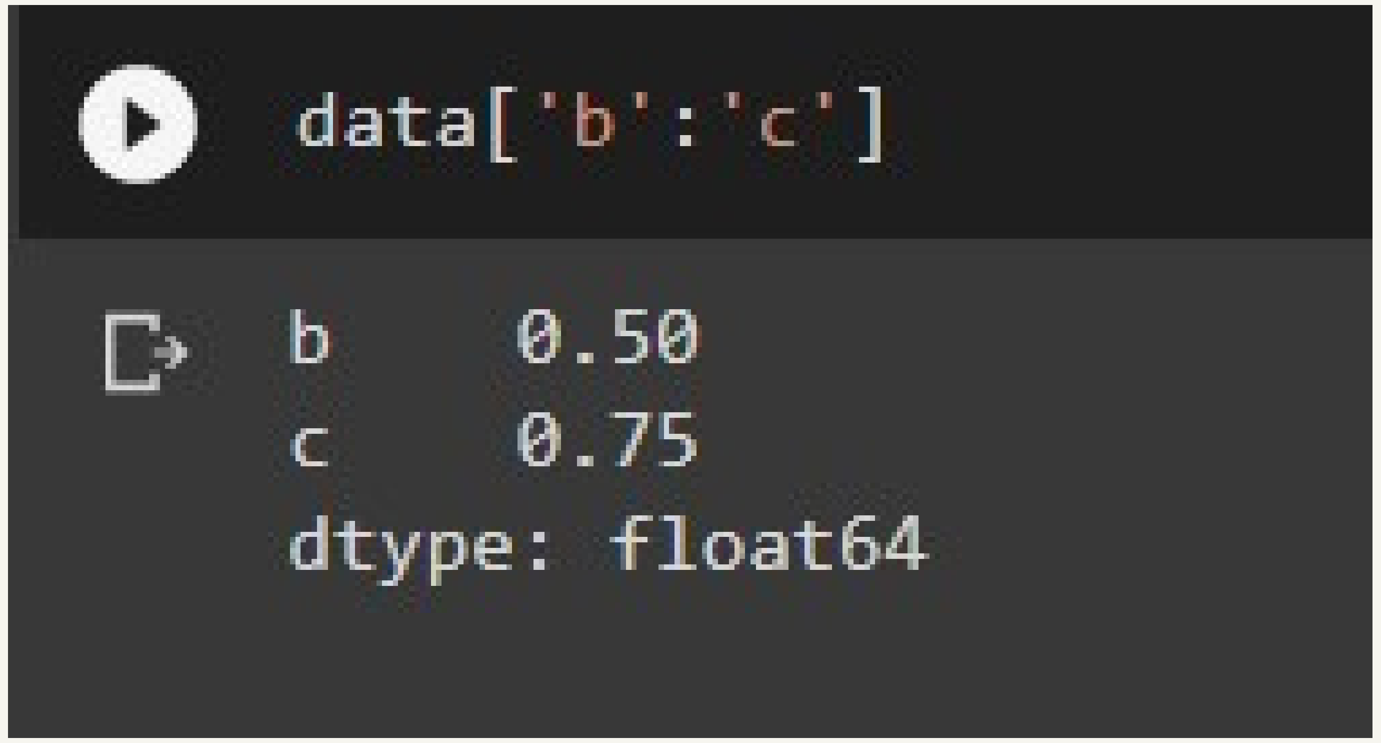


```
[ ]  data = pd.Series([0.25, 0.50, 0.75, 1], index=['a','b','c','d'])
```

```
data
```

```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```

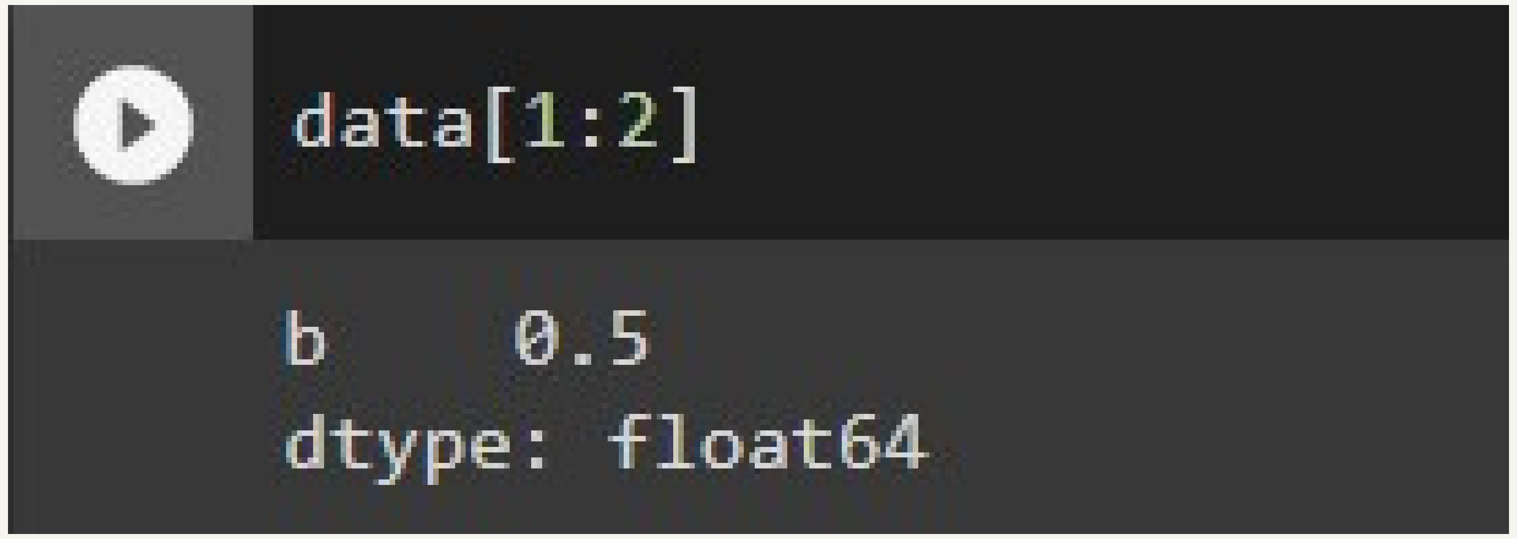
Call data with index b to index c using explicit index.



```
data['b':'c']
```

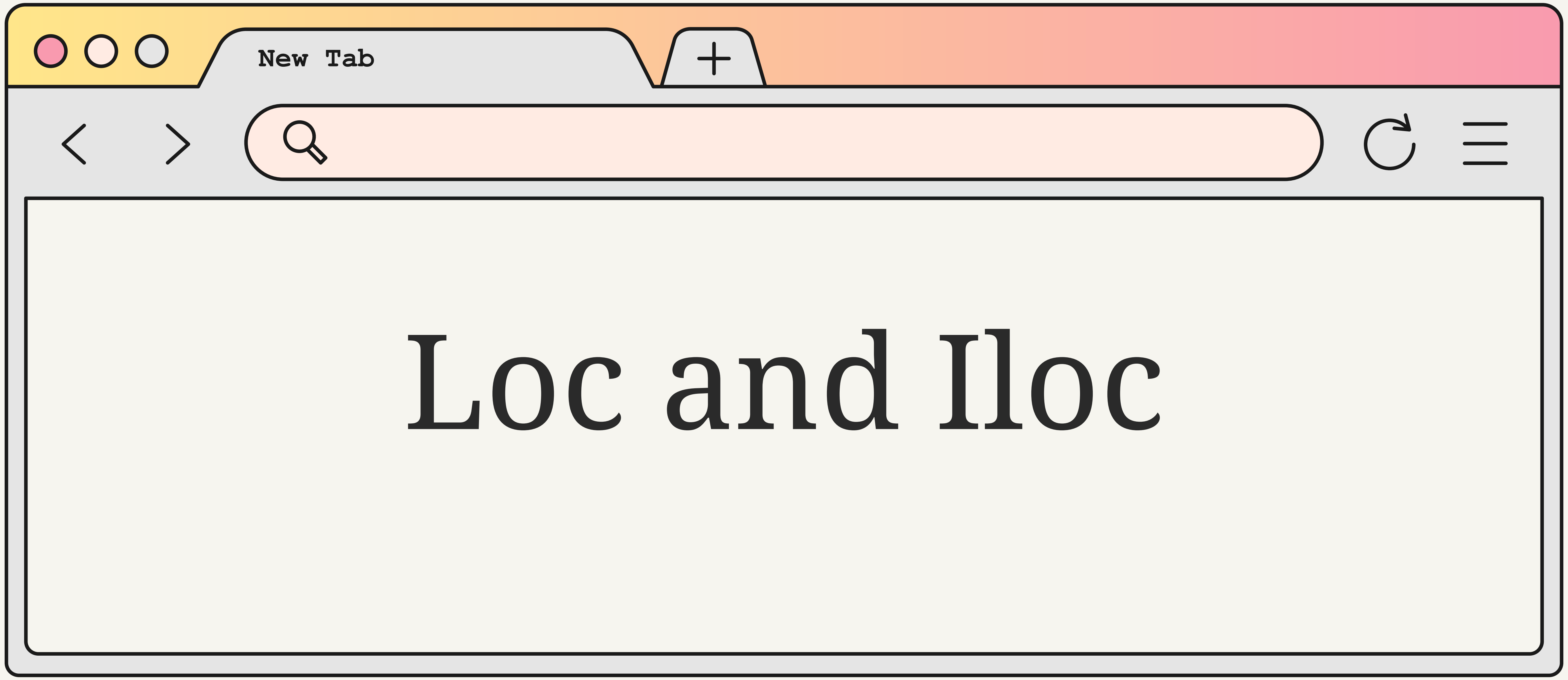
```
b    0.50
c    0.75
dtype: float64
```

If we slicing on the implicit index, it will be inclusive (start index: stop-1 index).



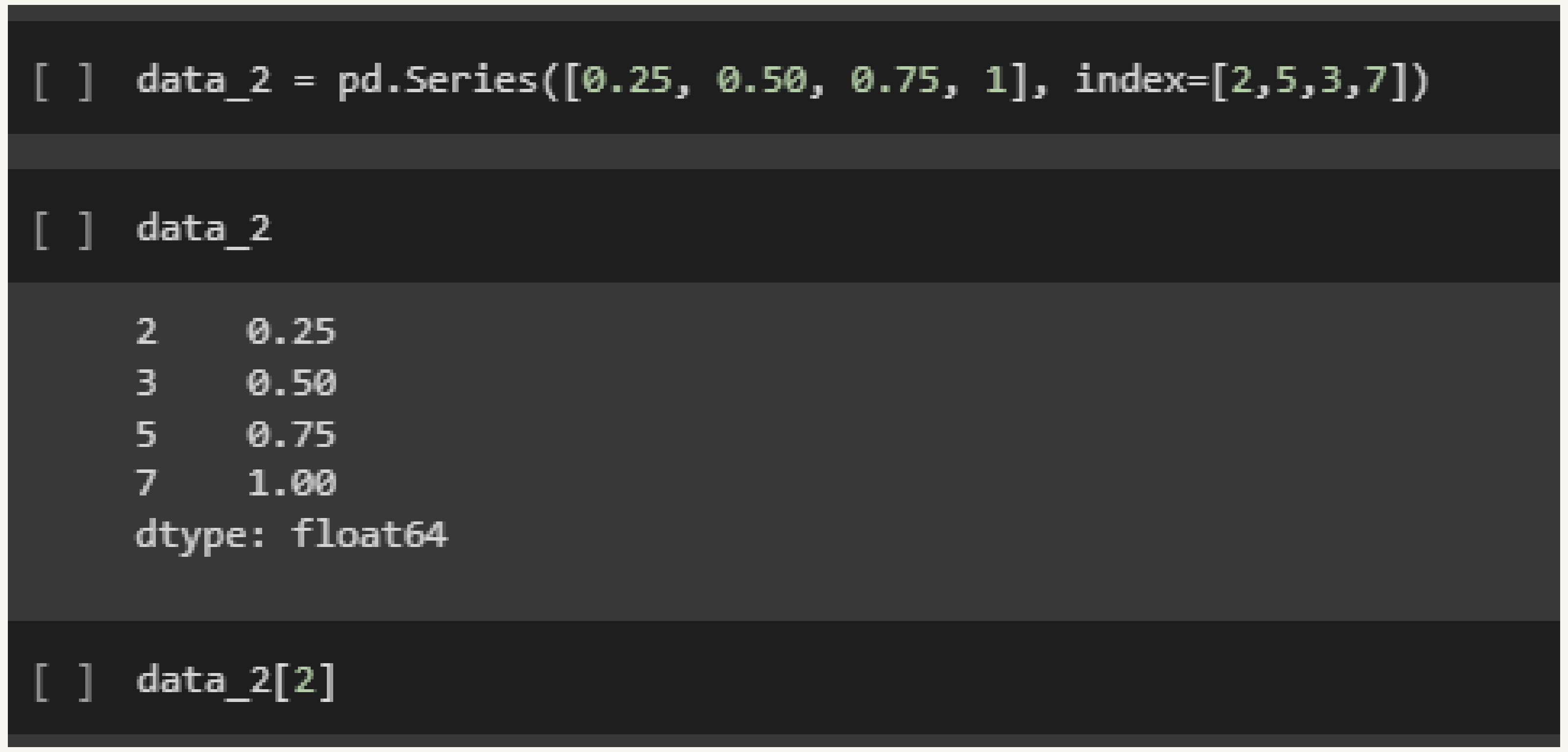
```
data[1:2]
```

```
b    0.5
dtype: float64
```



Loc and Iloc

Before exploring what loc and iloc are, let's have a look at the following Python code

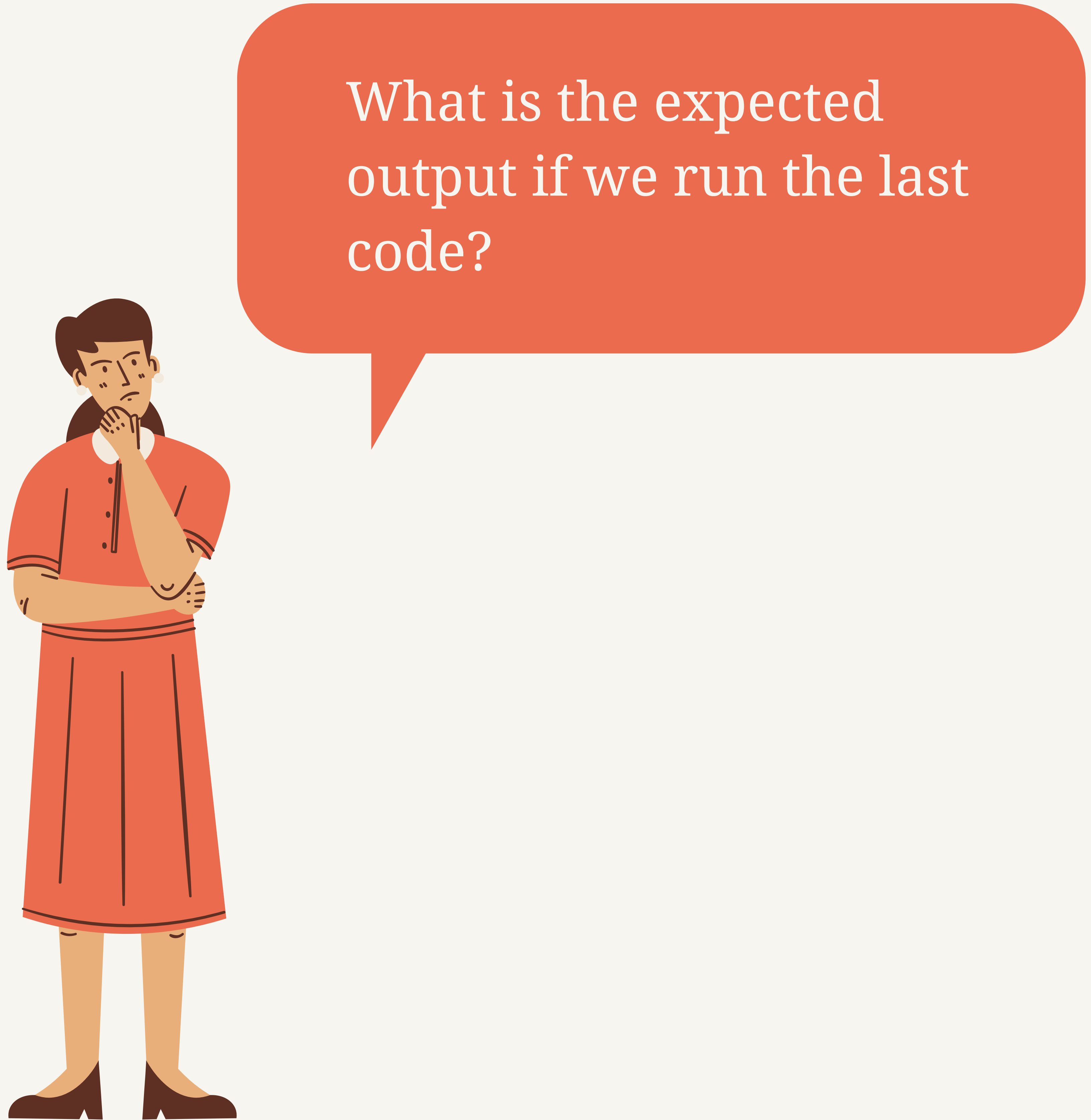


```
[ ]  data_2 = pd.Series([0.25, 0.50, 0.75, 1], index=[2,5,3,7])
```

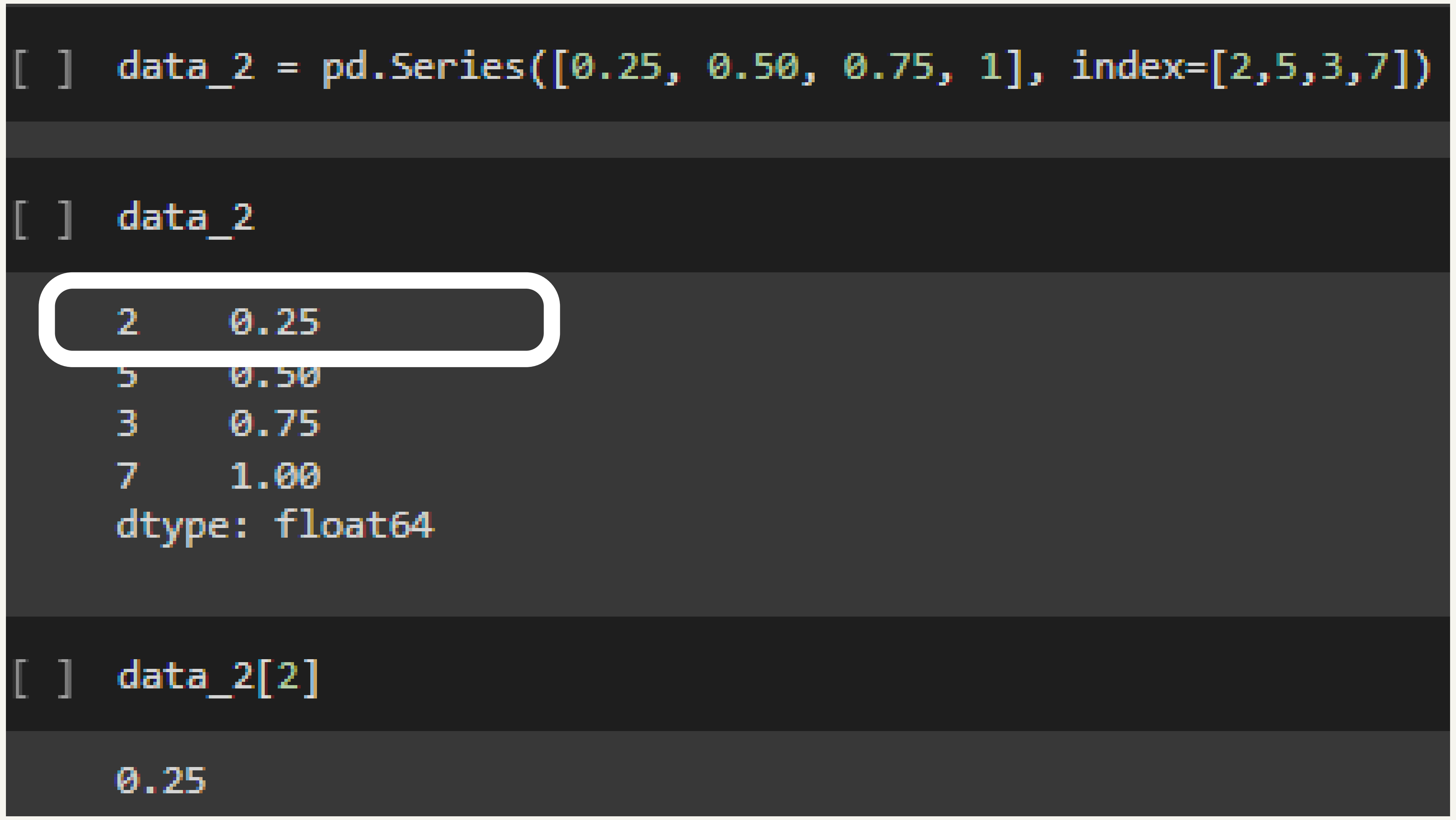
```
[ ]  data_2
```

```
     2    0.25
     3    0.50
     5    0.75
     7    1.00
     dtype: float64
```

```
[ ]  data_2[2]
```



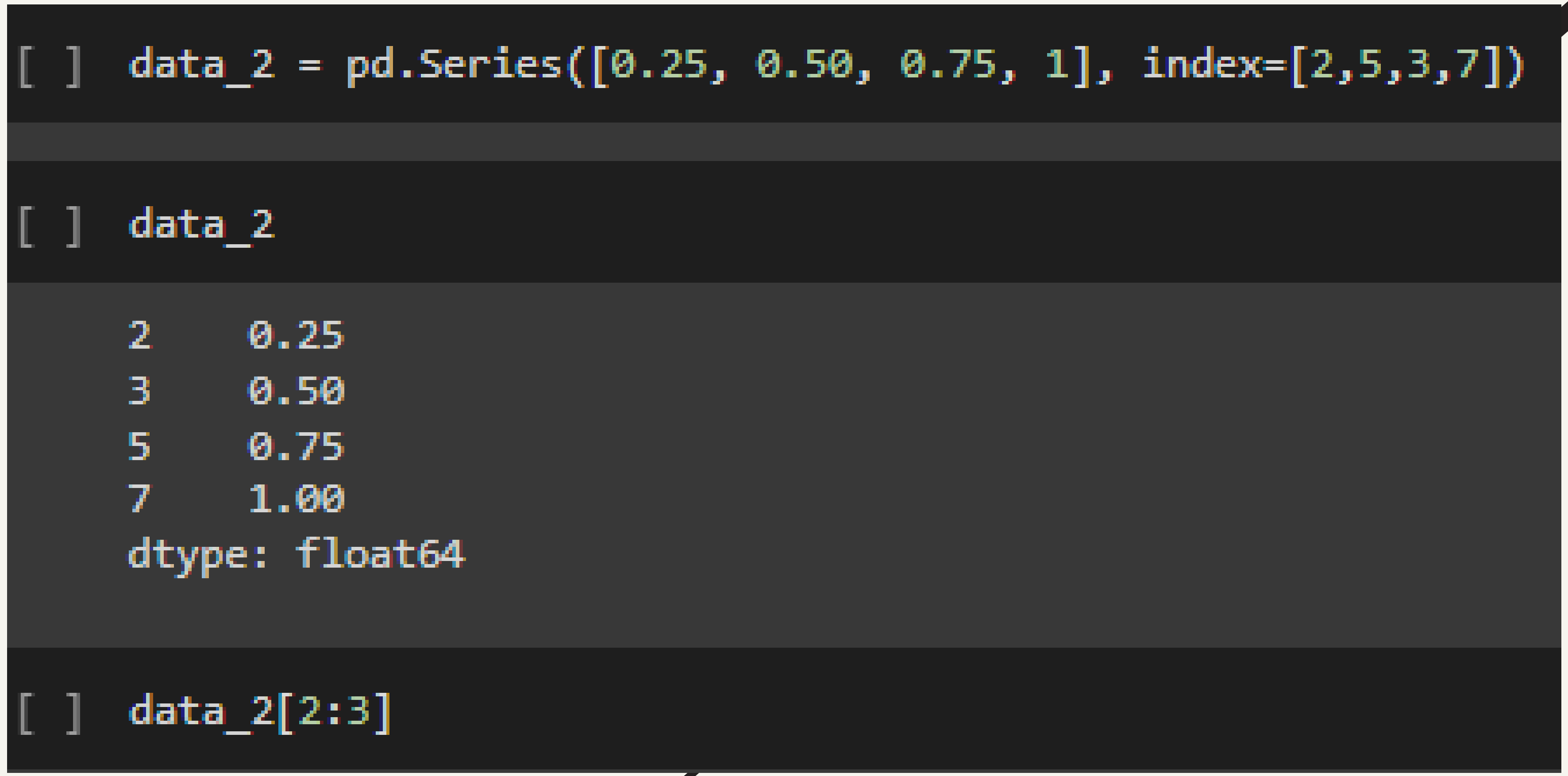
As we have seen before, when the implicit index and the explicit index are the same, then the data return will only rely on the explicit index



```
[ ]  data_2 = pd.Series([0.25, 0.50, 0.75, 1], index=[2,5,3,7])

[ ]  data_2

     2     0.25
     5     0.50
     3     0.75
     7     1.00
     dtype: float64

[ ]  data_2[2]

     0.25
```

So, the output from the previous code is
is 0.25 according to the explicit index 2 on data_2

But what will happen if we call the element on data_2 using index slicing like this



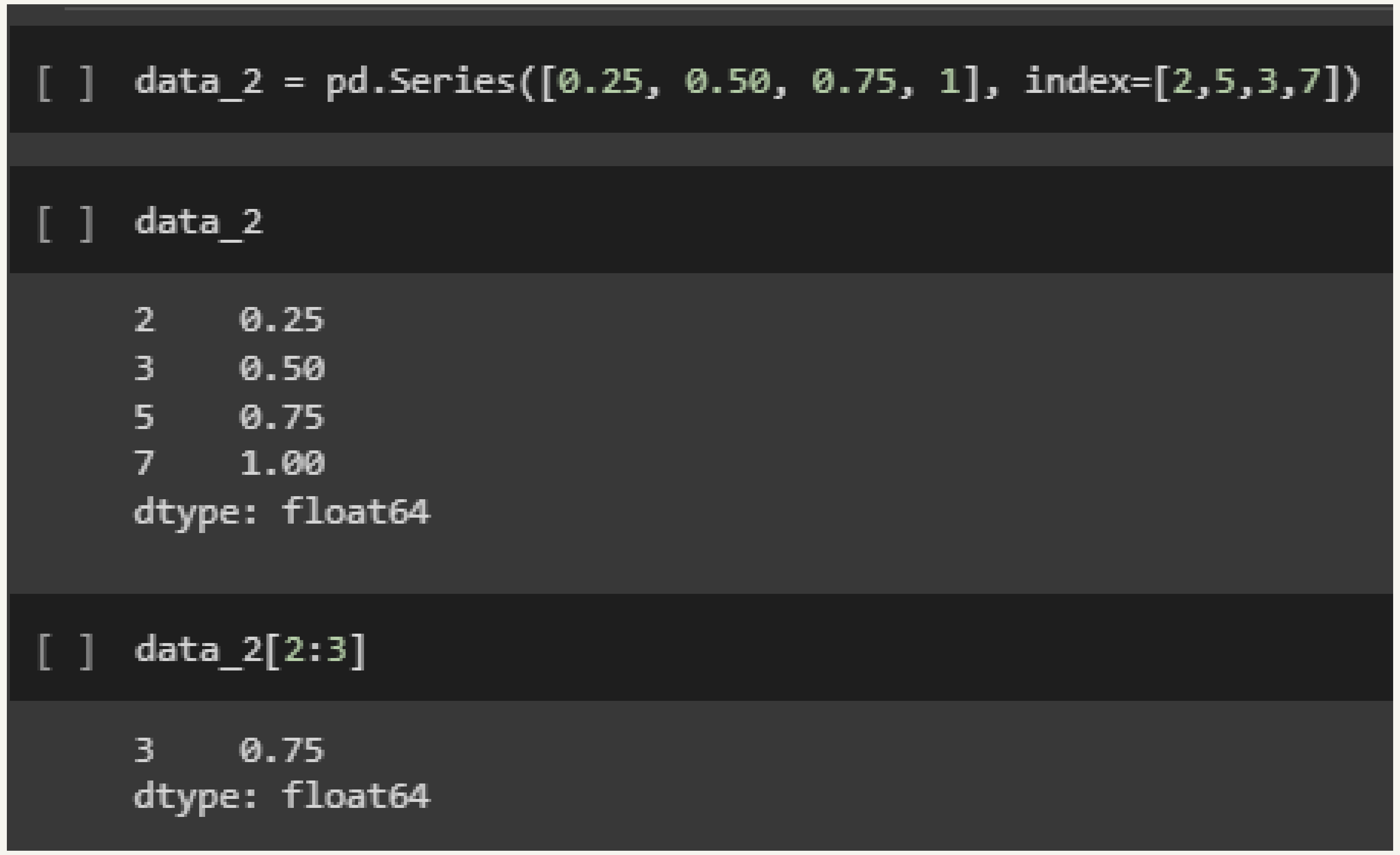
```
[ ]  data_2 = pd.Series([0.25, 0.50, 0.75, 1], index=[2,5,3,7])
```

```
[ ]  data_2
```

```
     2    0.25
     3    0.50
     5    0.75
     7    1.00
     dtype: float64
```

```
[ ]  data_2[2:3]
```

What is the expected output if we run the last code?



```
[ ] data_2 = pd.Series([0.25, 0.50, 0.75, 1], index=[2,5,3,7])
```

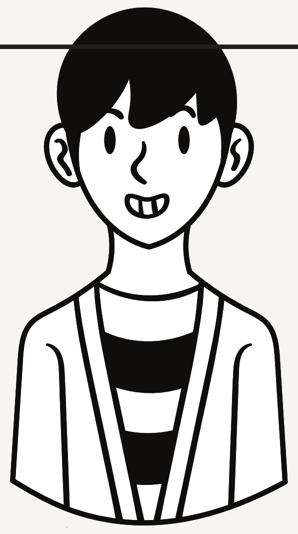
```
[ ] data_2
```

```
2    0.25
3    0.50
5    0.75
7    1.00
dtype: float64
```

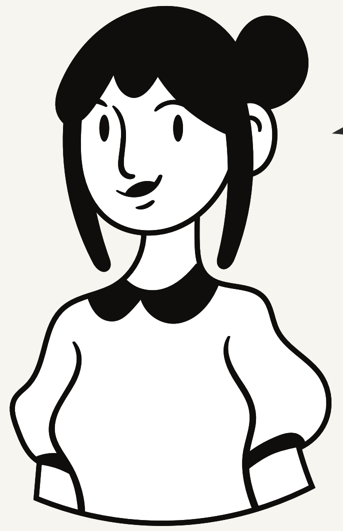
```
[ ] data_2[2:3]
```

```
3    0.75
dtype: float64
```

When the explicit and implicit indexes are the same, there will be inconsistencies as in the case above.
To overcome this inconsistency, we use the **loc and iloc** rules.



So, what are loc and iloc ?

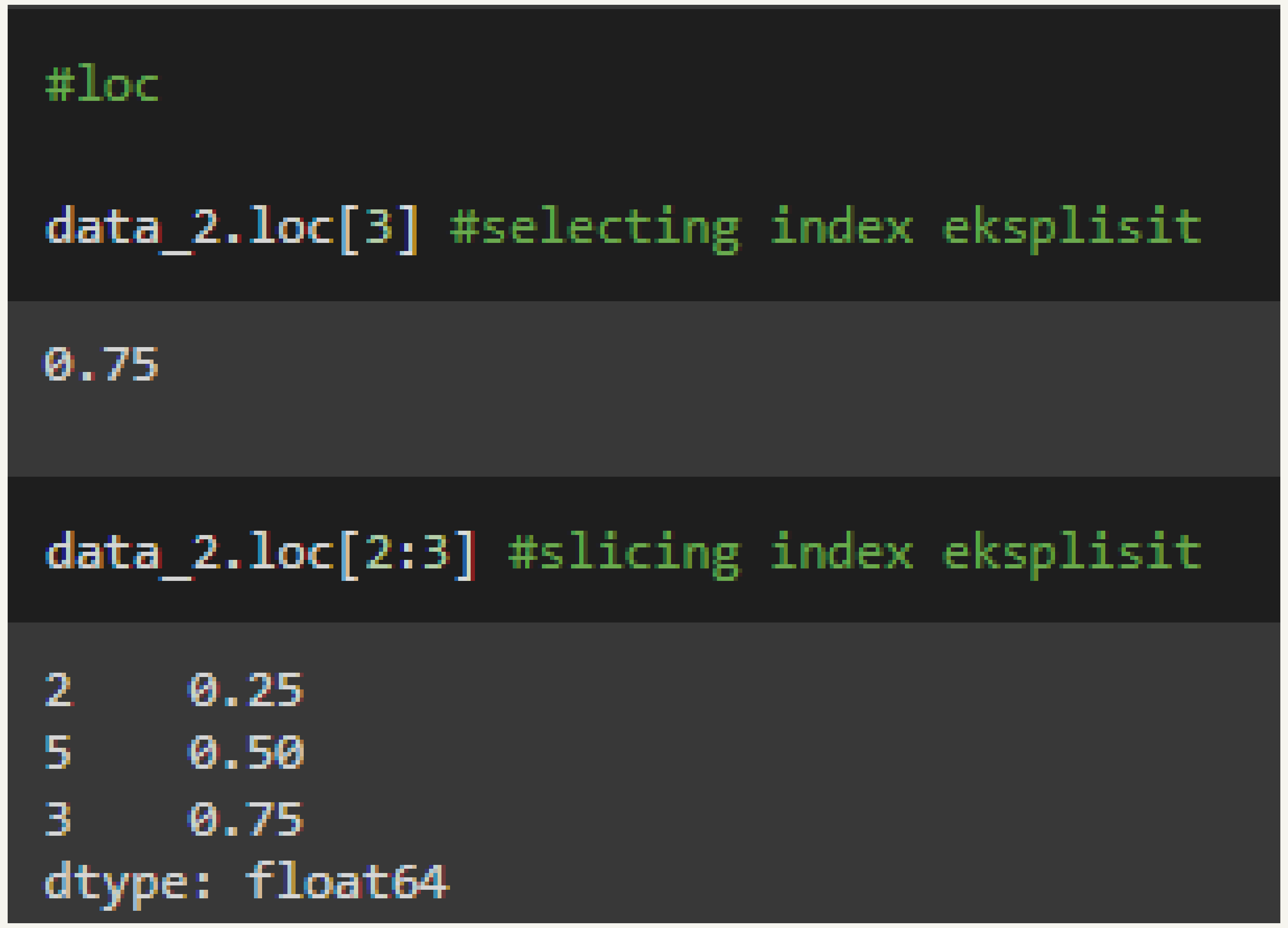


loc ( ) and iloc ( ) are methods in the Pandas library.

loc() is label based data selecting method which means that we have to pass the name of the row or column which we want to select. In other words, loc ( ) is used to call an explicit index.

iloc() is a indexed based selecting method which means that we have to pass integer index in the method to select specific row/column. In the other words, iloc ( ) is used to call an implisit index.

Loc



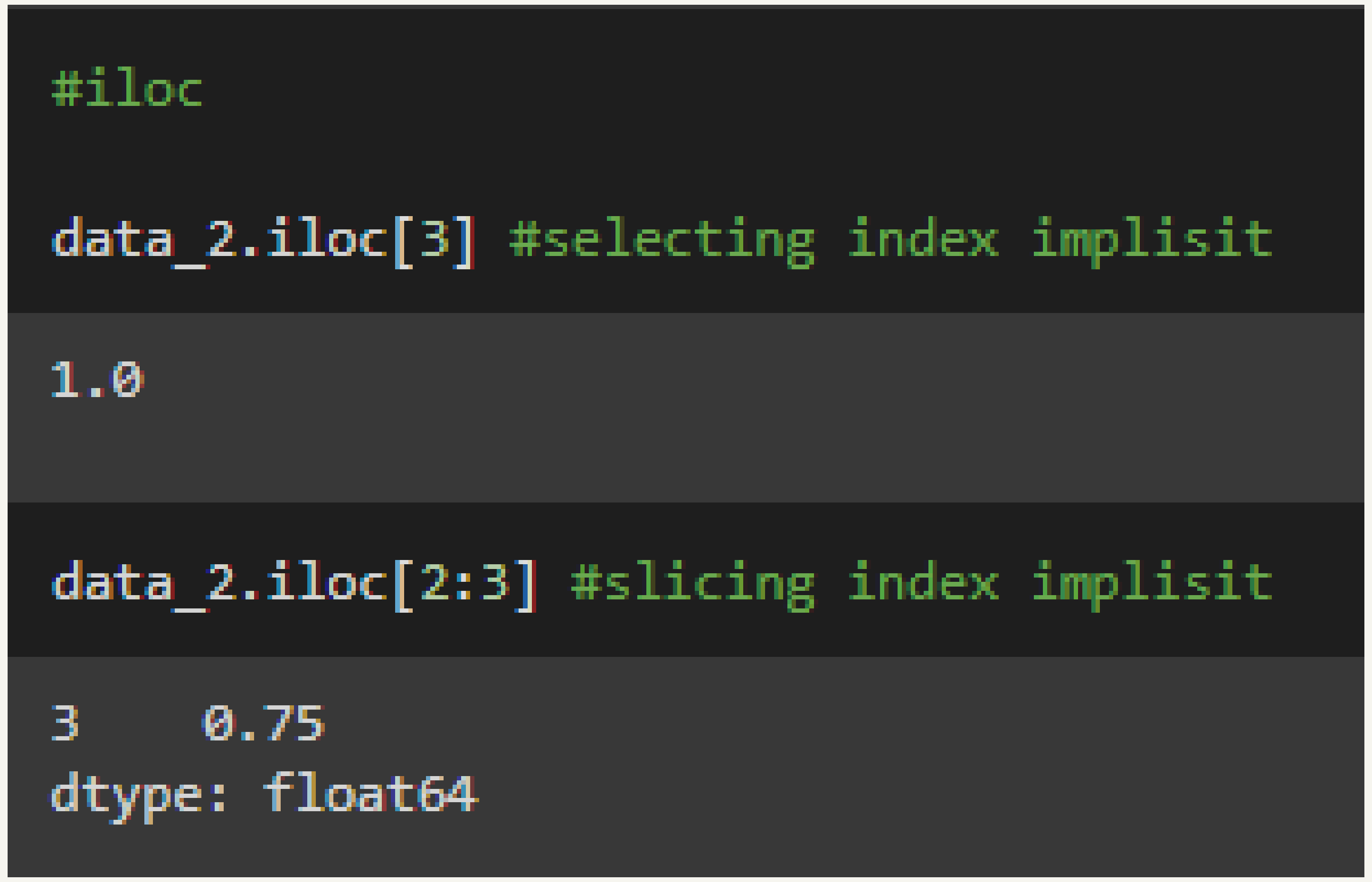
```
#loc

data_2.loc[3] #selecting index eksplisit
```

```
0.75
```

```
data_2.loc[2:3] #slicing index eksplisit
```

```
2    0.25
5    0.50
3    0.75
dtype: float64
```

Iloc

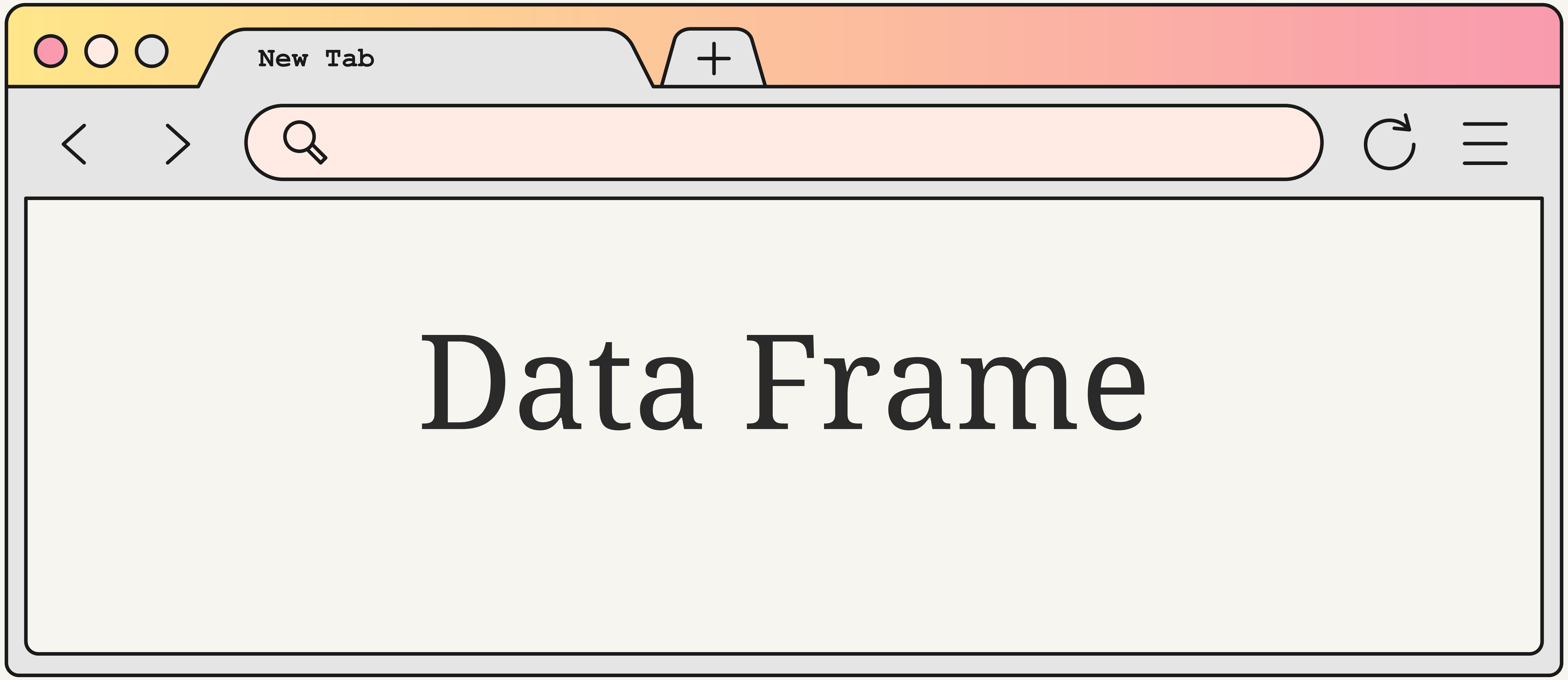


```
#iloc

data_2.iloc[3] #selecting index implisit
```

```
1.0
```

```
data_2.iloc[2:3] #slicing index implisit
```

```
3    0.75
dtype: float64
```



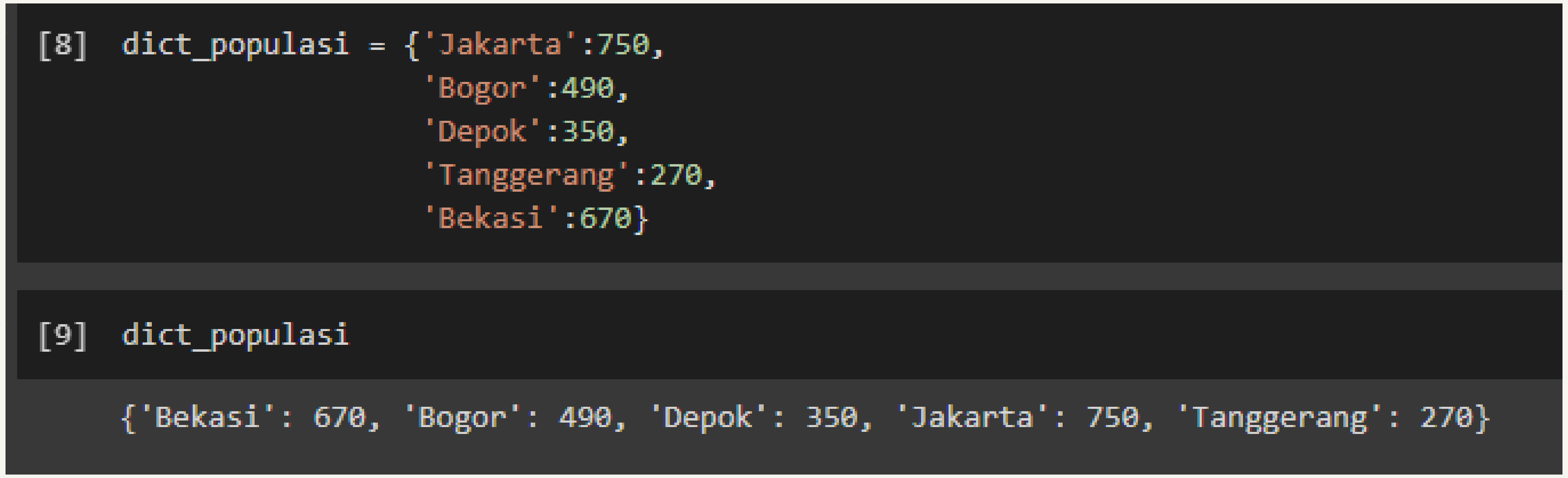
Data Frame

Data Frame

A Pandas DataFrame is a 2 dimensional data structure,
like a 2 dimensional array, or a table with rows and columns.
Data frame is a part of pandas library.

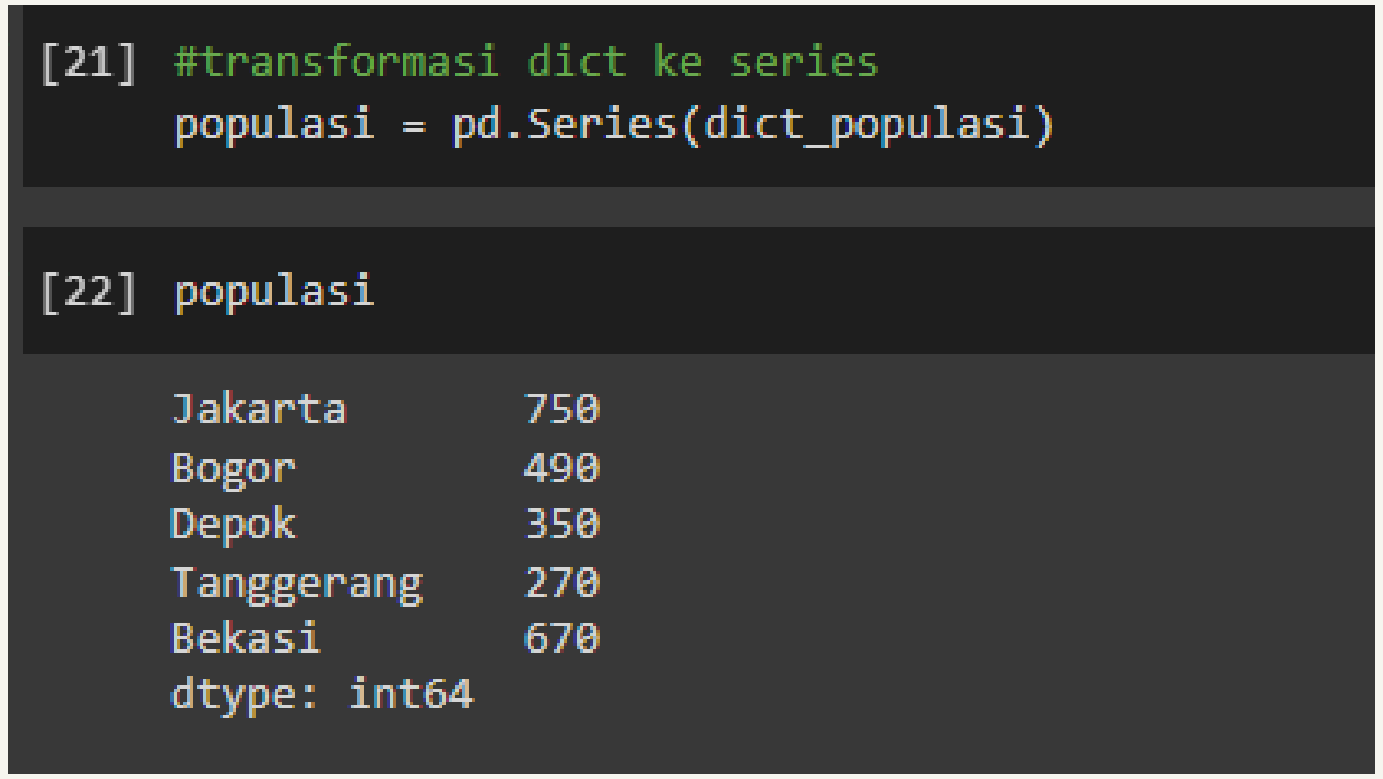
example 1

we can create a simple data frame from a dictionary



```
[8]  dict_populasi = {'Jakarta':750,
                      'Bogor':490,
                      'Depok':350,
                      'Tanggerang':270,
                      'Bekasi':670}

[9]  dict_populasi

     {'Bekasi': 670, 'Bogor': 490, 'Depok': 350, 'Jakarta': 750, 'Tanggerang': 270}
```

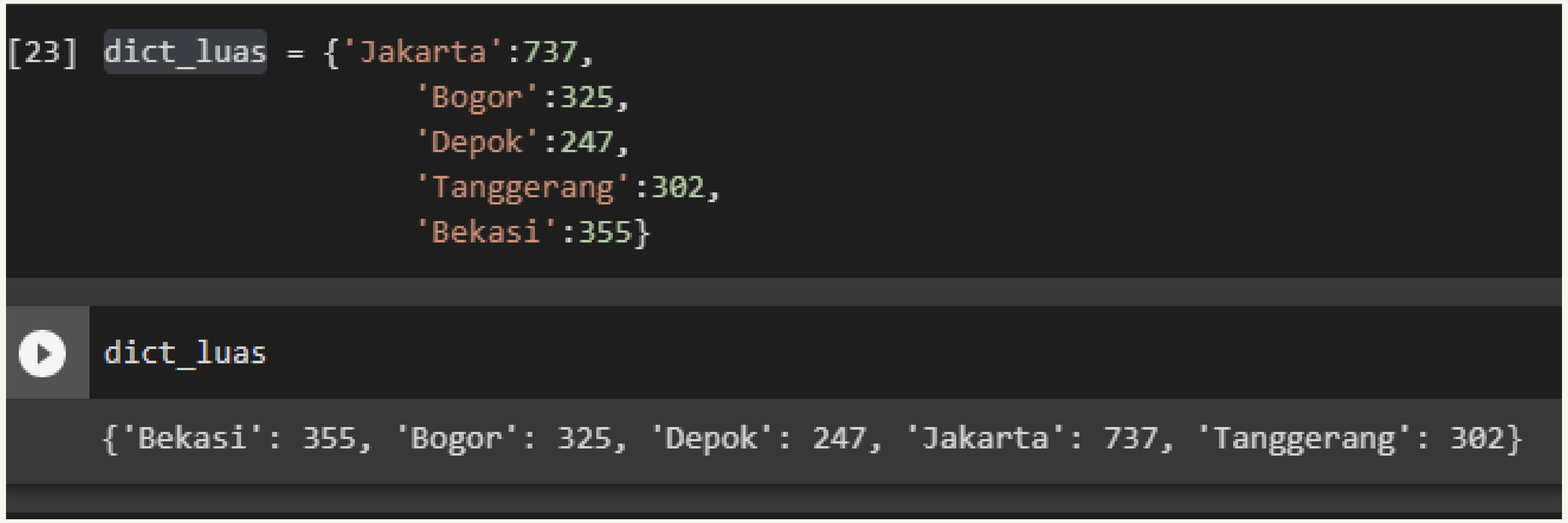
next we change the dictionary into a series object



```
[21] #transformasi dict ke series
     populasi = pd.Series(dict_populasi)

[22] populasi

     Jakarta       750
     Bogor         490
     Depok         350
     Tanggerang    270
     Bekasi        670
     dtype: int64
```

example 2

create a simple data frame from a dictionary

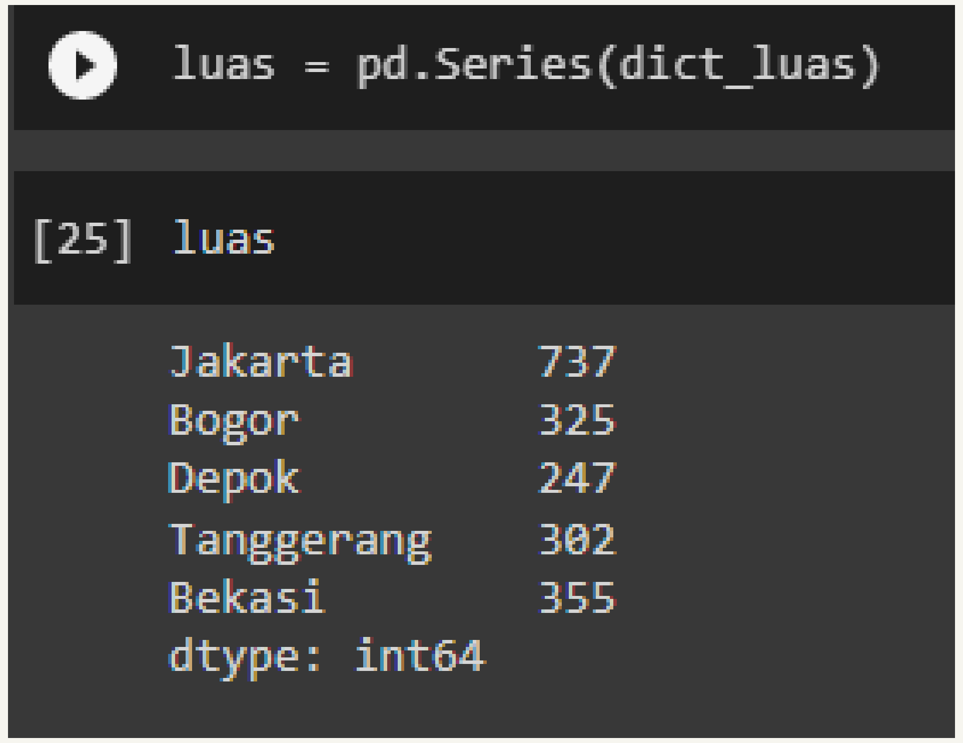


```
[23] dict_luas = {'Jakarta':737,
                  'Bogor':325,
                  'Depok':247,
                  'Tanggerang':302,
                  'Bekasi':355}
```

```
dict_luas
```

```
{'Bekasi': 355, 'Bogor': 325, 'Depok': 247, 'Jakarta': 737, 'Tanggerang': 302}
```

change the dictionary into a series object

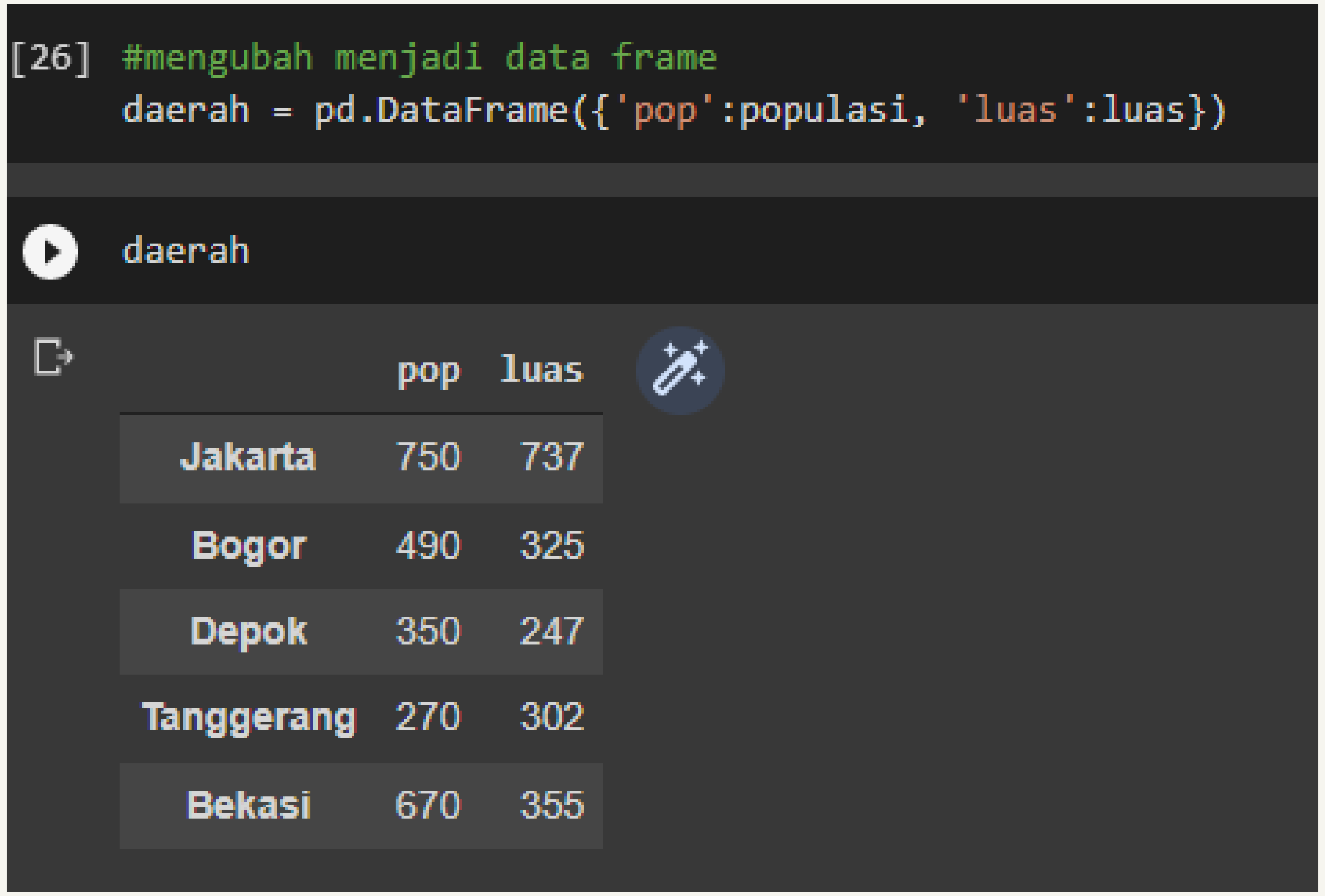


```
luas = pd.Series(dict_luas)
```

```
[25] luas
```

```
Jakarta       737
Bogor         325
Depok         247
Tanggerang    302
Bekasi        355
dtype: int64
```

Finally, we convert the two data series above into one dataframe

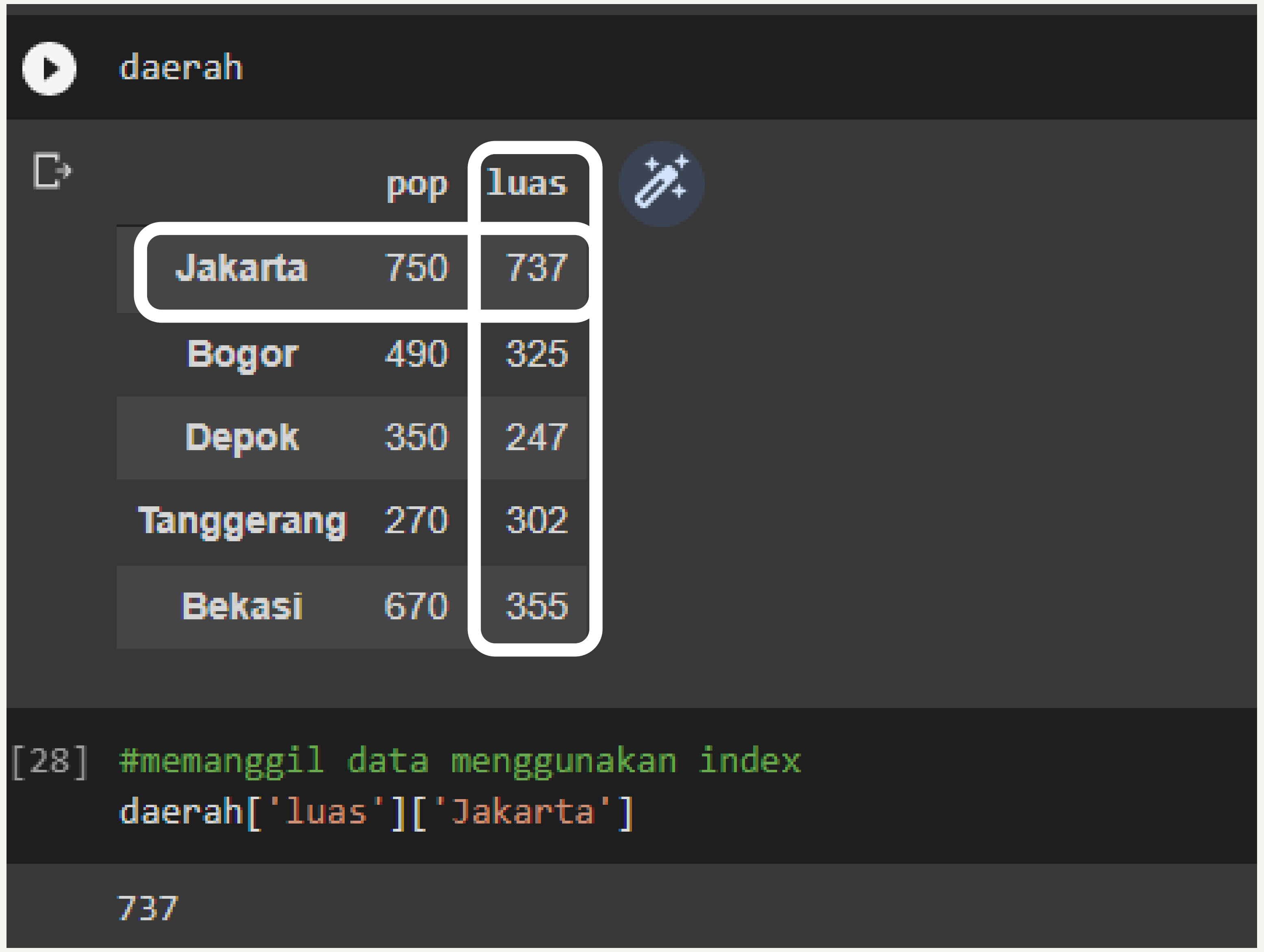


```
[26] #mengubah menjadi data frame
     daerah = pd.DataFrame({'pop':populasi, 'luas':luas})
```

```
daerah
```

```
            pop  luas
Jakarta     750   737
Bogor       490   325
Depok       350   247
Tanggerang  270   302
Bekasi      670   355
```

we can call data by using explicit index as below

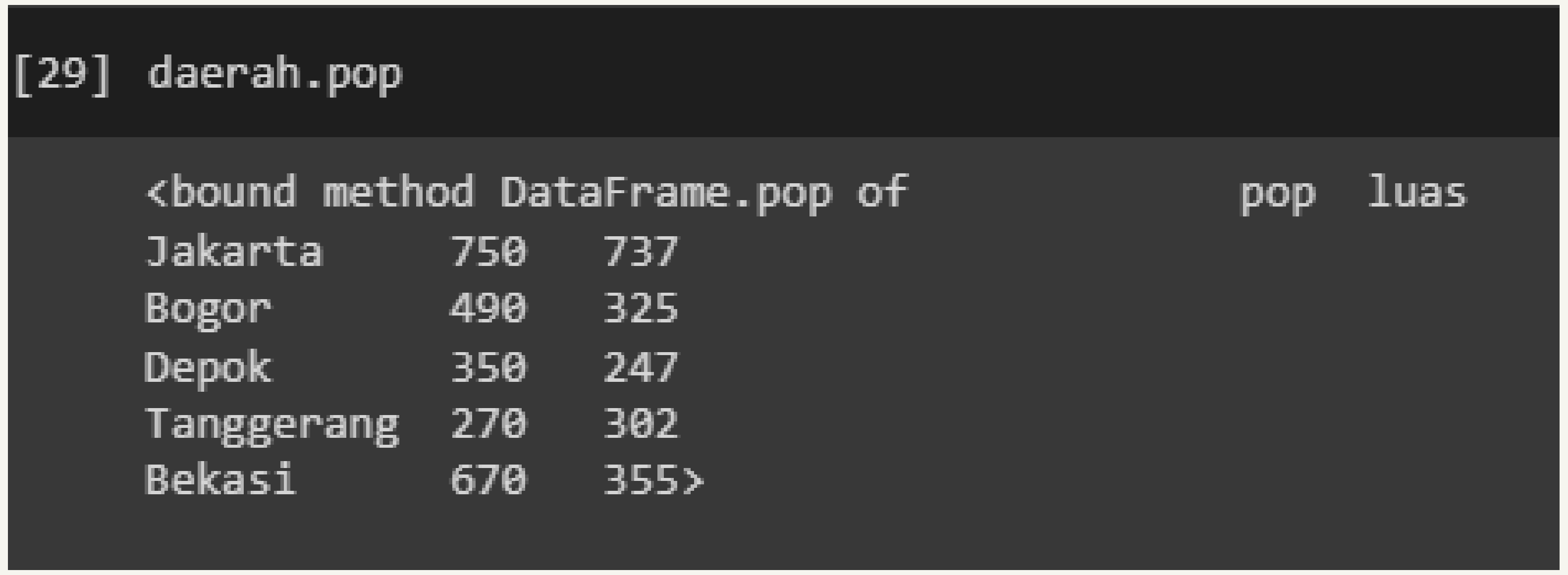


```
daerah
```

```
            pop  luas
Jakarta     750   737
Bogor       490   325
Depok       350   247
Tanggerang  270   302
Bekasi      670   355
```

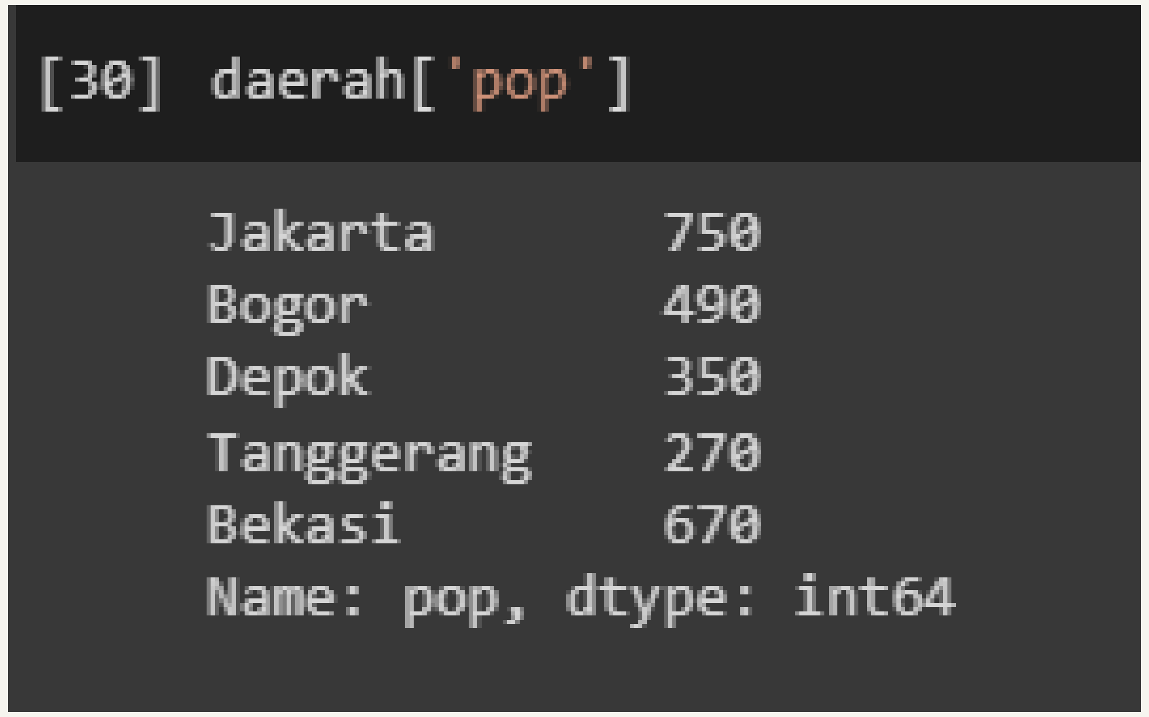
```
[28] #memanggil data menggunakan index
     daerah['luas']['Jakarta']

737
```

warning, avoid using the daerah.pop syntax because it will display the entire data frame



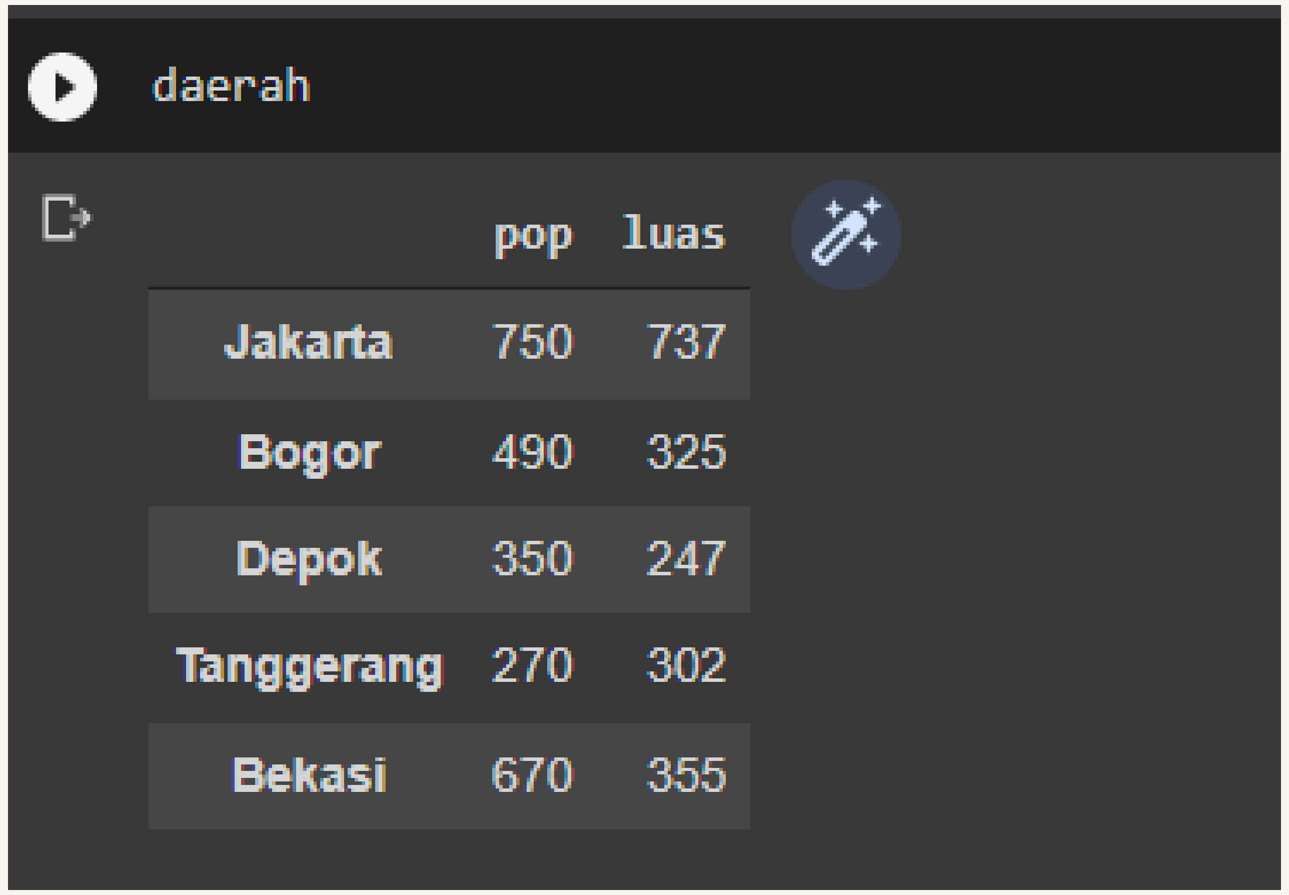
```
[29] daerah.pop

     <bound method DataFrame.pop of             pop  luas
     Jakarta     750   737
     Bogor       490   325
     Depok       350   247
     Tanggerang  270   302
     Bekasi      670   355>
```

to display the population column data, use the syntax as below



```
[30] daerah['pop']

     Jakarta       750
     Bogor         490
     Depok         350
     Tanggerang    270
     Bekasi        670
     Name: pop, dtype: int64
```

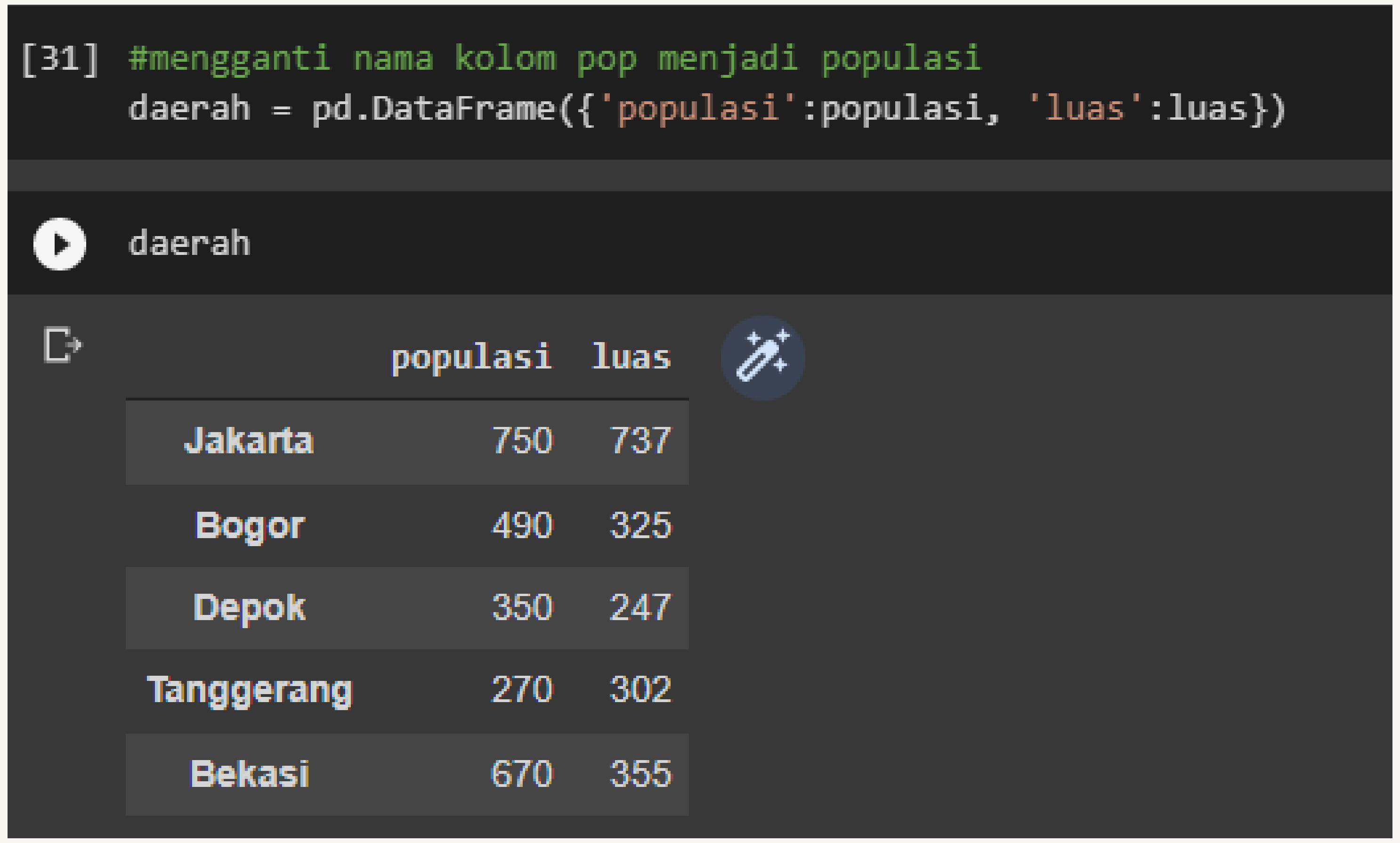
rename the columns contained in the data frame



```
daerah
```

	pop	luas
Jakarta	750	737
Bogor	490	325
Depok	350	247
Tanggerang	270	302
Bekasi	670	355

before



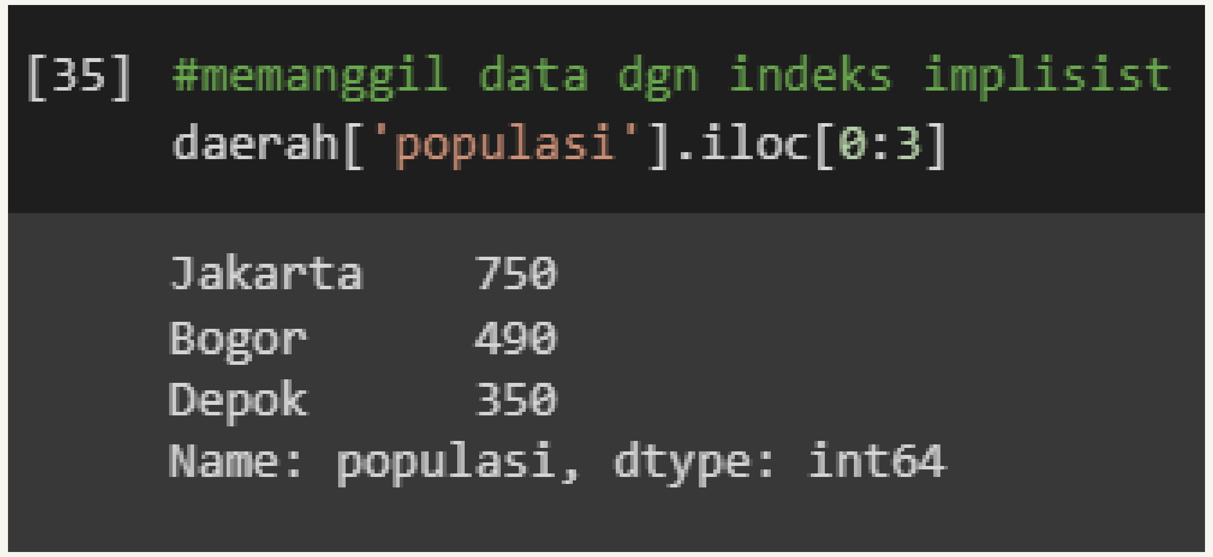
```
[31] #mengganti nama kolom pop menjadi populasi
     daerah = pd.DataFrame({'populasi':populasi, 'luas':luas})
```

```
daerah
```

	populasi	luas
Jakarta	750	737
Bogor	490	325
Depok	350	247
Tanggerang	270	302
Bekasi	670	355

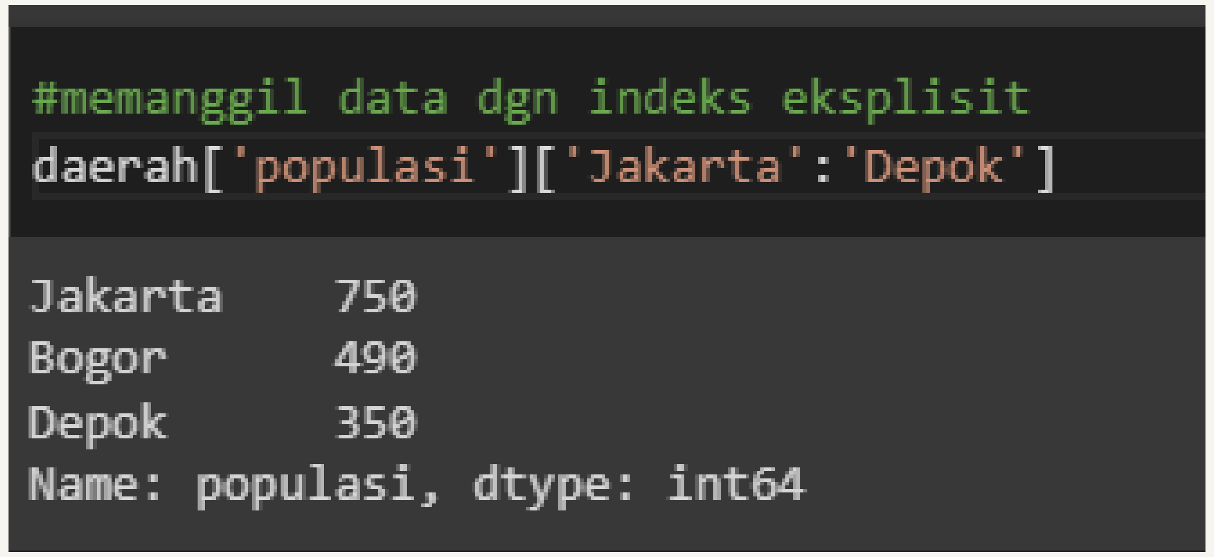
after

display data using implicit index and explicit index



```
[35] #memanggil data dgn indeks implisist
     daerah['populasi'].iloc[0:3]

     Jakarta    750
     Bogor      490
     Depok      350
     Name: populasi, dtype: int64
```

index implisit

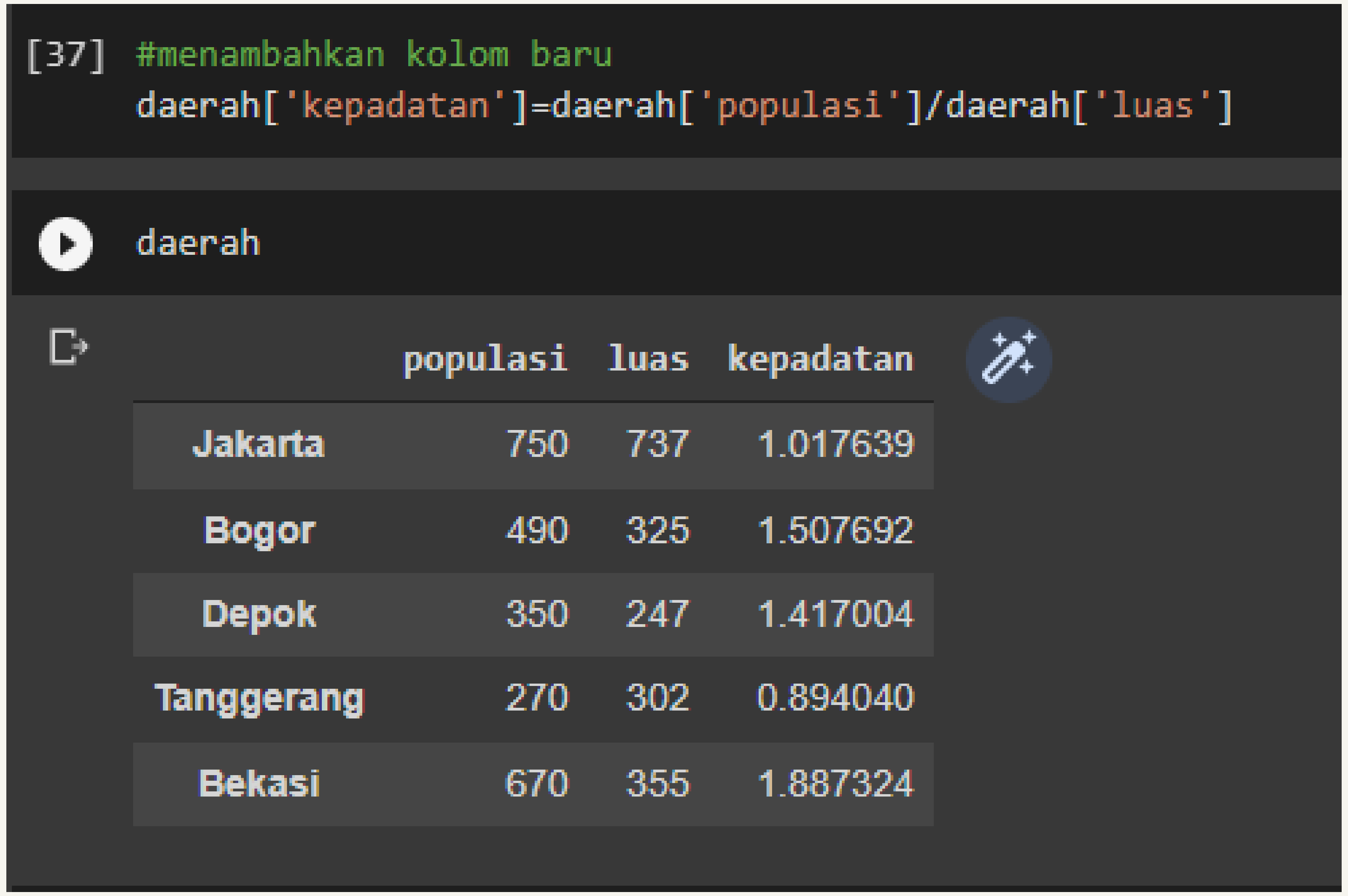


```
#memanggil data dgn indeks eksplisit
daerah['populasi']['Jakarta':'Depok']

Jakarta    750
Bogor      490
Depok      350
Name: populasi, dtype: int64
```

index explisit

add a new column to the data frame

in this case we will add a new column that is the density obtained by dividing the population and area



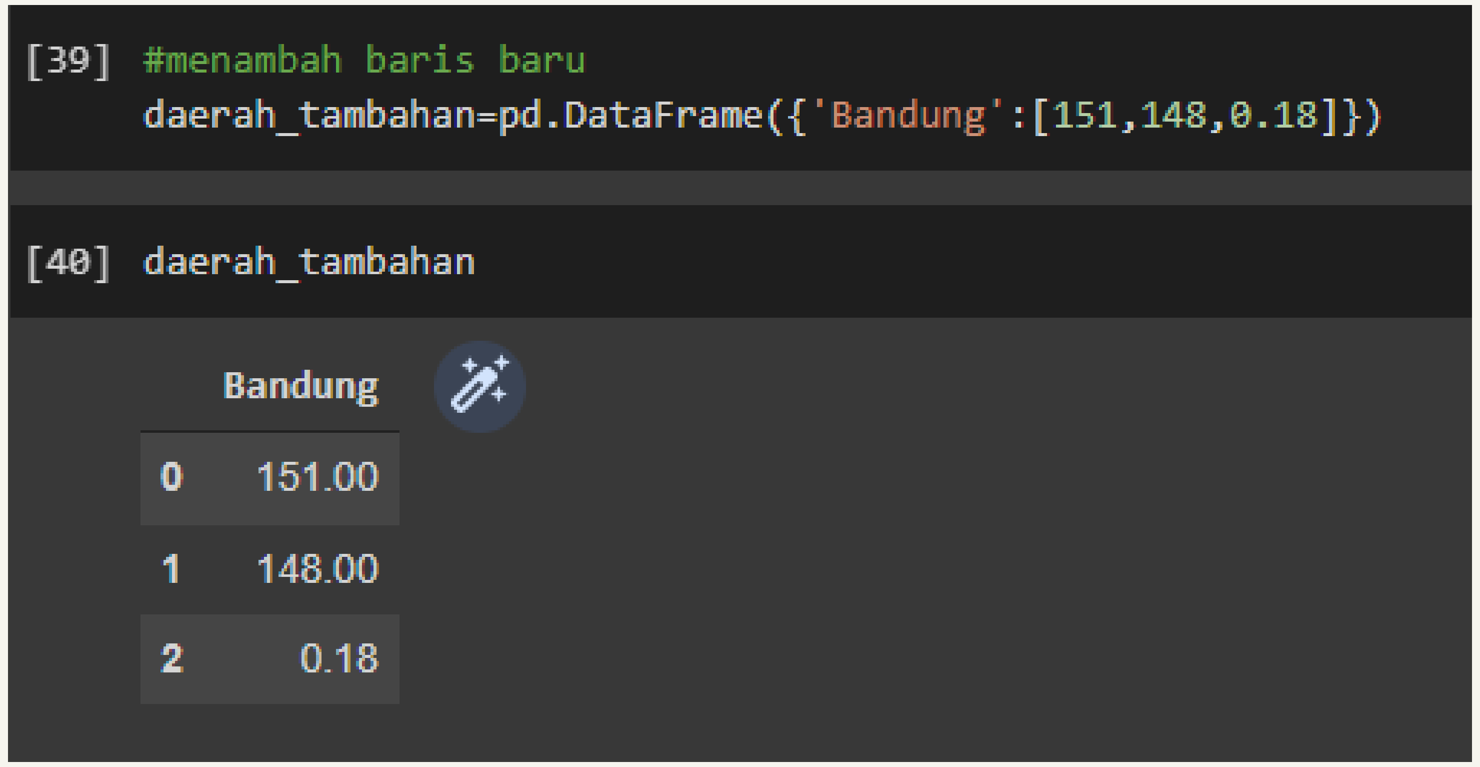
```
[37] #menambahkan kolom baru
     daerah['kepadatan']=daerah['populasi']/daerah['luas']
```

```
daerah
```

```
            populasi  luas  kepadatan
Jakarta          750   737   1.017639
Bogor            490   325   1.507692
Depok            350   247   1.417004
Tanggerang       270   302   0.894040
Bekasi           670   355   1.887324
```

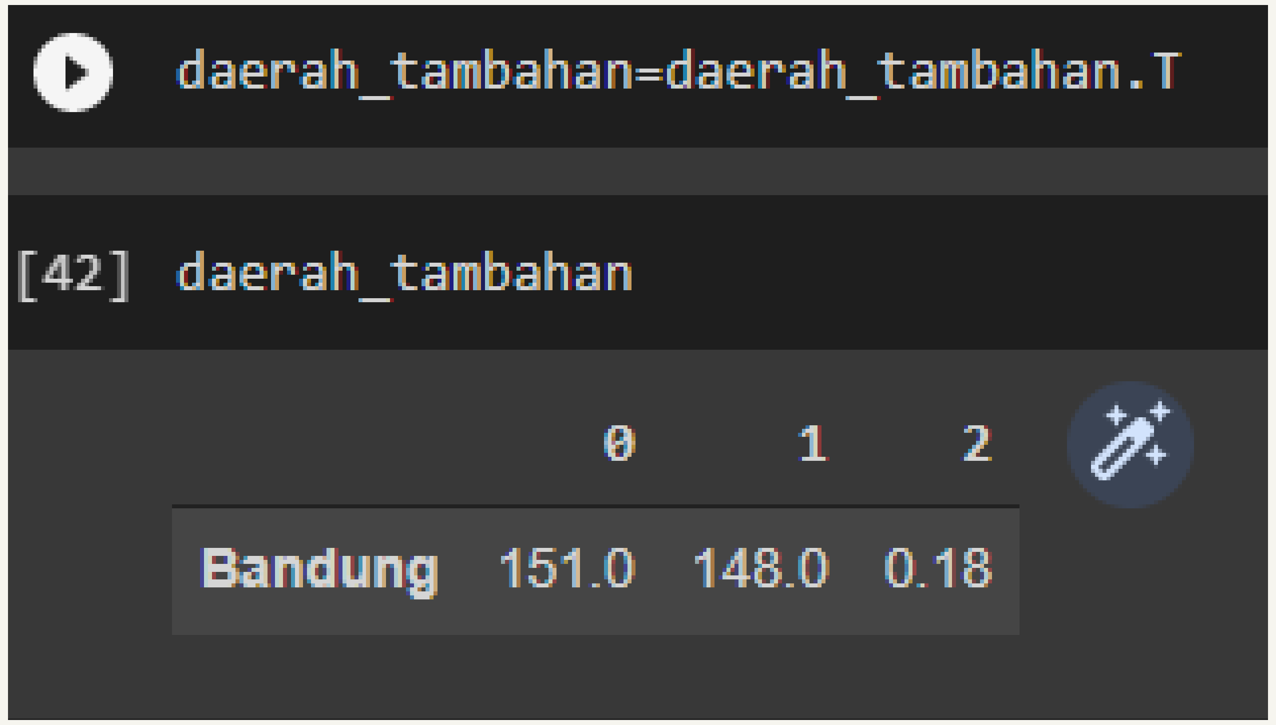
add a new row to the data frame

in this case we input the data into the syntax



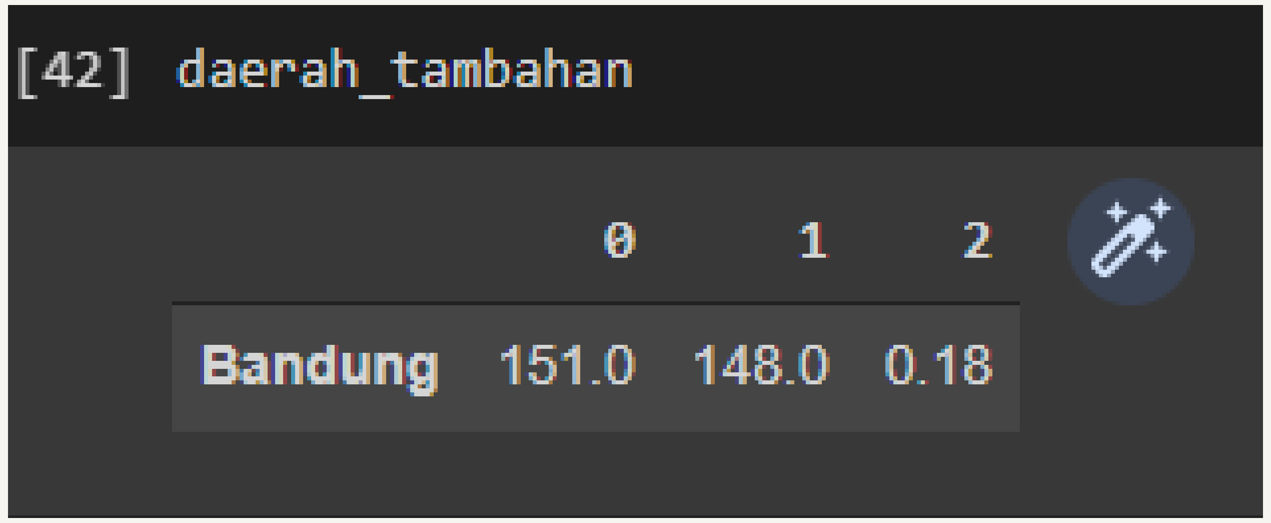
```
[39] #menambah baris baru
     daerah_tambahan=pd.DataFrame({'Bandung':[151,148,0.18]})

[40] daerah_tambahan

   Bandung
0   151.00
1   148.00
2     0.18
```

because the data is still in column form, we have to convert it to row form using transpose



```
daerah_tambahan=daerah_tambahan.T

[42] daerah_tambahan

             0      1     2
Bandung  151.0  148.0  0.18
```

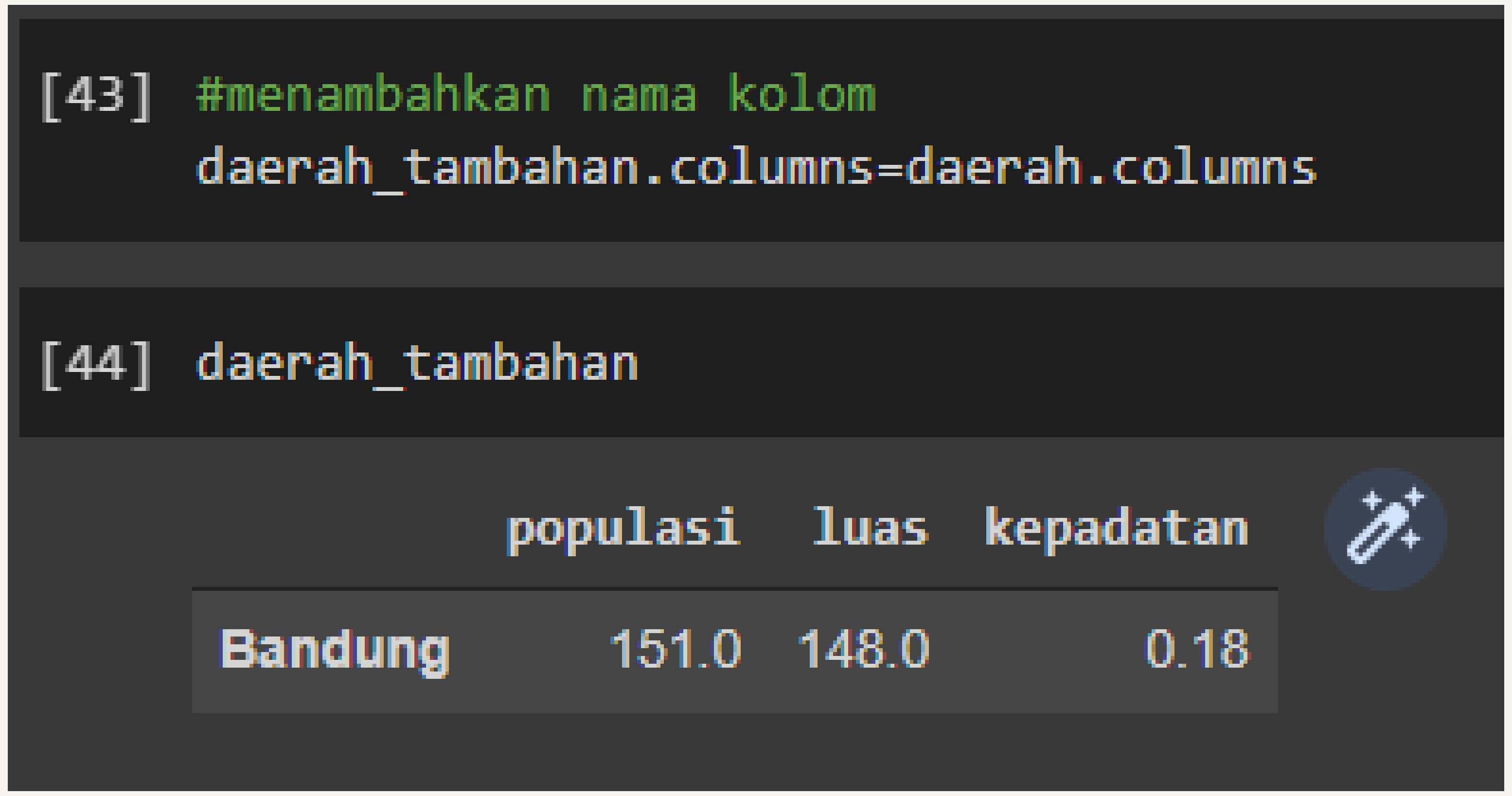
Next we add the column names to the transposed rows



```
[42] daerah_tambahan
```

```
                0      1     2
Bandung     151.0  148.0  0.18
```

before



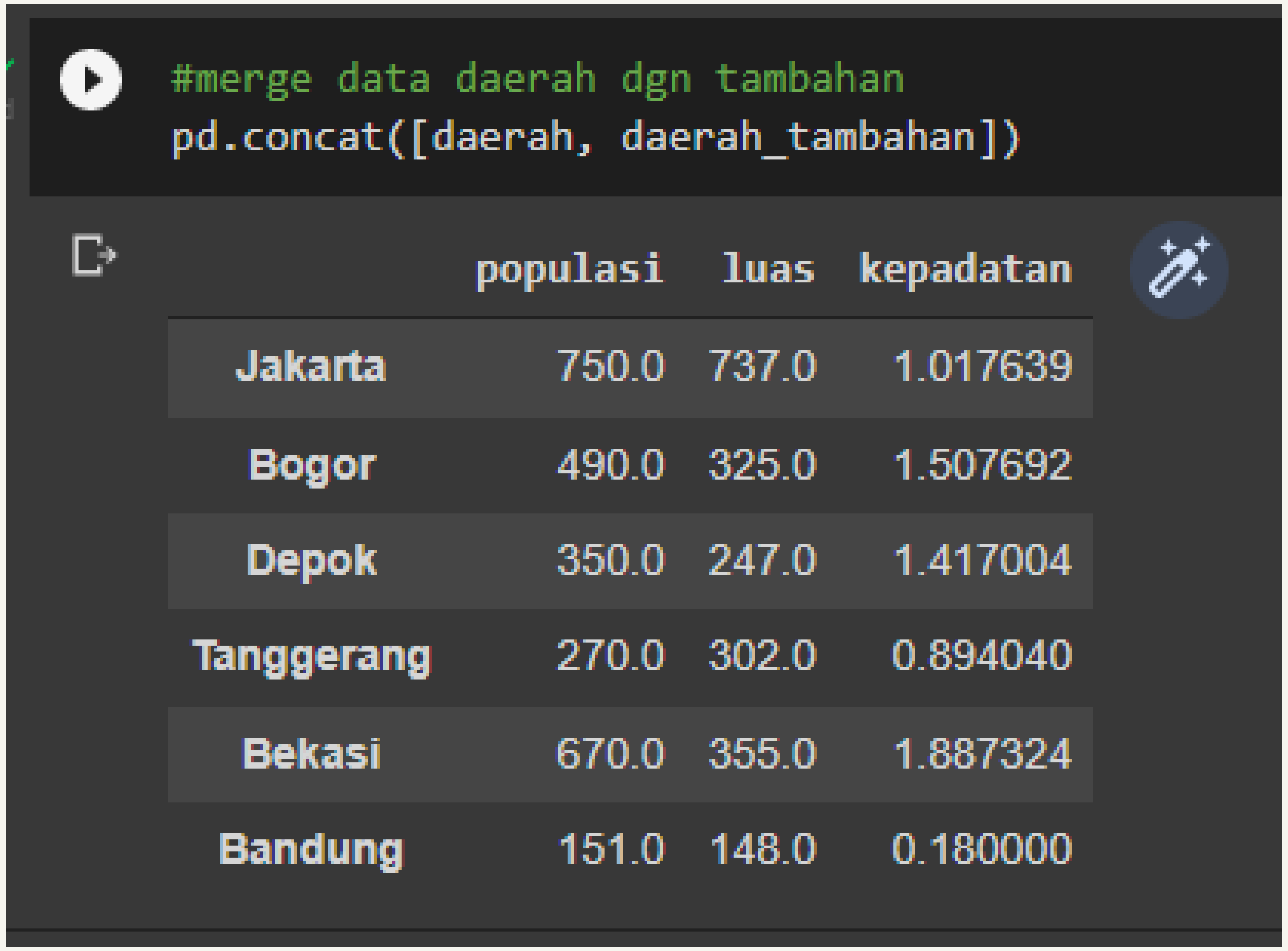
```
[43] #menambahkan nama kolom
     daerah_tambahan.columns=daerah.columns
```

```
[44] daerah_tambahan
```

```
          populasi  luas   kepadatan
Bandung      151.0  148.0       0.18
```

after

the last step is to combine the row with the data frame



```
#merge data daerah dgn tambahan
pd.concat([daerah, daerah_tambahan])
```

	populasi	luas	kepadatan
Jakarta	750.0	737.0	1.017639
Bogor	490.0	325.0	1.507692
Depok	350.0	247.0	1.417004
Tanggerang	270.0	302.0	0.894040
Bekasi	670.0	355.0	1.887324
Bandung	151.0	148.0	0.180000

THANK YOU