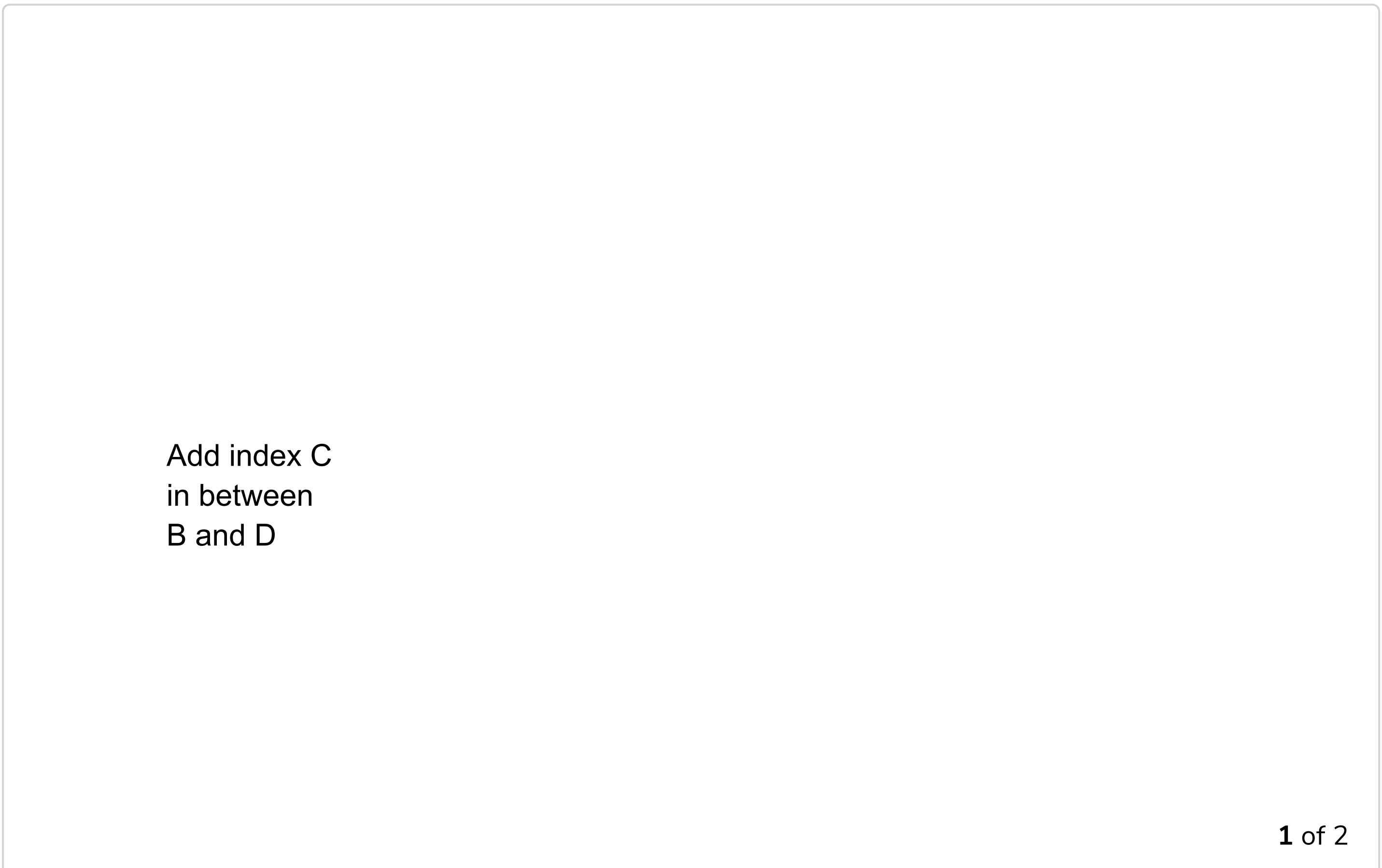
Reindex Objects

In this lesson, reindexing methods for pandas objects is explained.

We'll cover the following

Re-indexing

This method allows for adding new indexes and columns in `Series` and `DataFrames` without disturbing the initial setting of the objects. The following illustration might make it clear.



The value of the index `C` in the last slide of the illustration is automatically set to `NaN` because no value was defined to it.

> **Note**: Re-indexing rules are the same for both `Series` and `DataFrame` objects.

The function used for this purpose is `reindex()`. It is called by a `Series` or a `DataFrame` object, and a list of **indexes** is passed as a parameter.

Re-indexing in `Series`

Let's take the same example from the series part of the course and try adding new indexes to it.

```
#importing pandas in our program
import pandas as pd

# Defining a series object
srs1 = pd.Series([11.9, 36.0, 16.6, 21.8, 34.2], index = ['China', 'India', 'USA', 'Brazil', 'Paki

# Set Series name
srs1.name = "Growth Rate"
```

```
srs1.name = "Growth Rate"

# Set index name
srs1.index.name = "Country"

srs2 = srs1.reindex(['China', 'India', 'Malaysia', 'USA', 'Brazil', 'Pakistan', 'England'])
print("The series with new indexes is:\n",srs2)

srs3 = srs1.reindex(['China', 'India', 'Malaysia', 'USA', 'Brazil', 'Pakistan', 'England'], fill_va
print("\nThe series with new indexes is:\n",srs3)
```

It can be seen in the output that the new **indexes** are added with `NaN` as their values. The new **indexes** can be placed anywhere around the original indexes. For example, `Malaysia` was added in-between two original indexes `India` and `USA`, whereas `England` was added at the end.

On **Line 16**, another parameter was passed in the `reindex()` function. The `fill_value` parameter assigns a default value to the new **indexes** instead of `NaN`. In this case, `fill_value` is assigned `0`. So, both the new **indexes** now have a default value of `0`.

Re-indexing in `DataFrame`

A `DataFrame` can be re-indexed in two ways. One through the indexes and the other through the columns. The following example makes this clear.

```
import numpy as np
import pandas as pd

# Define a 2-D array
arr2d = np.arange(16).reshape(4,4)

# Give 2-D array to Dataframe and assign index and column names.
df = pd.DataFrame(arr2d, index=['Row1', 'Row2', 'Row4', 'Row5'], columns=['Column1','Column2','Col
print("The original DataFrame\n", df)

df2 = df.reindex(['Row1', 'Row2', 'Row3', 'Row4', 'Row5'])
print("\nNew DataFrame with reindexed indexes:\n", df2)

df2 = df2.reindex(columns=['Column0', 'Column1', 'Column2', 'Column3', 'Column4'])
print("\nNew DataFrame with reindexed columns:\n", df2)
```

On **Line 11**, the *indexes* are changed, just like we changed for the `series`. The new **index** name is added in-between `Row2` and `Row4`, and similarly, by default, `NaN` values are assigned to the whole row.

One **Line 14**, `columns` keyword should be specifically used to reindex the columns of `DataFrame`. The rules are the same as for the indexes. `NaN` values were assigned to the whole column by default.

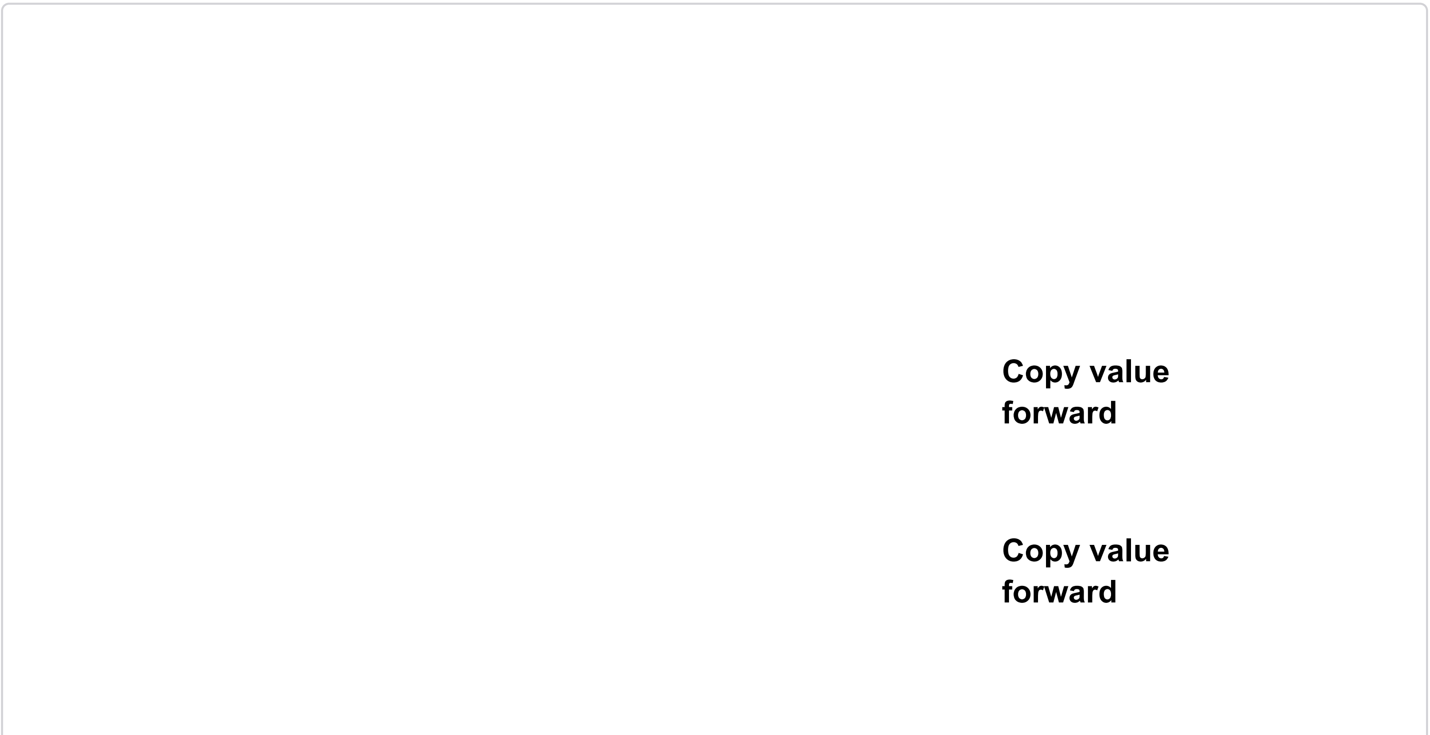
Forward filling

This is a way of assigning values to the default `NaN` that occurs due to re-indexing.

For a `Series`, it assigns the value that was before the `NaN` to `NaN` and keeps doing it until another value other than `NaN` appears. Then, it takes this new value and assigns it to the other `NaN` that might come after it. This process continues until the end of the `Series`.

For a `DataFrame`, the same process works in two ways. The process can propagate either through the rows or columns. Axis needs to be defined to decide which way the `NaN` values will be filled. The axis for the row is `0`, and it'll be filled top to bottom. The axis for the column is `1`, and it'll be filled left to right.

Let's understand this with an example:



Forward filling in Series

1 of 4

Values Forwarded

Forward filling in Series

Forward filling in DataFrame

Forward filling along the column axis

Forward filling in DataFrame

4 of 4

The function used for this is `ffill()`. It is called with the `Series` or `DataFrame` object. For `DataFrame`, the axis is passed as a parameter to this function.

Forward filling in `Series`

Let’s look at the code to forward fill a `Series` object.

```
import pandas as pd

srs1 = pd.Series([11.9, 36.0, 16.6, 21.8, 34.2], index = ['China', 'India', 'USA', 'Brazil', 'Paki
srs1.name = "Growth Rate"
srs1.index.name = "Country"

srs2 = srs1.reindex(['China', 'India', 'Malaysia', 'USA', 'Brazil', 'Pakistan', 'England'])
print("The series NaN Values:\n",srs2)

print("\nThe series with new Values:\n",srs2.ffill())
```

It can be clearly seen in the output that the `NaN` values are replaced with the values above them.

> **Note**: If a `NaN` value is at the top of the `Series` then it is not filled because there is no value above it.

Forward filling in `DataFrame`

Let’s look at the code to forward fill a `DataFrame` object. As the `DataFrame` is two dimensional, values can be filled either from the rows or columns. The `axis` parameter is used with the `ffill()` method to specifically indicate in which way the values will get filled.

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(16).reshape(4,4), index=['Row1', 'Row2', 'Row4', 'Row5'], columns=['Co

df2 = df.reindex(['Row1', 'Row2', 'Row3', 'Row4', 'Row5'])
df2 = df2.reindex(columns=['Column1', 'Column2', 'Column3', 'Column4', 'Column5'])
print("DataFrame with NaN Values:\n", df2)

# Fill values row wise
print("\nDataFrame with new values around Axis 0:\n", df2.ffill(axis = 0))

# Fill values column wise
print("\nDataFrame with new values around Axis 1:\n", df2.ffill(axis = 1))
```

It can be seen in the output that in both cases values were changed. For rows, the values above the `NaN` are copied, and for columns, the values to the left are copied.

In the next lesson, more features of pandas are explored.