- Universal functions with a `Series`
- Universal functions with a `DataFrame`
- Universal functions with a `DataFrame` and `Series`

We know now how to use universal functions with NumPy arrays
- A universal function executes on all elements very efficiently
- Much faster and cleaner than using an explicit loop

You can also use NumPy universal functions in Pandas
- On a Series (or DataFrame column)
- On a DataFrame
- On a mixed operation with a DataFrame and Series

You can use universal functions with a Series object
- E.g. a specific column in a DataFrame

Ex1

You can use universal functions with a DataFrame object
- Applies to all data in all rows and columns

Ex2

You can use universal functions with a DataFrame and a Series object

Ex3

- One-to-one merges
- Many-to-one merges
- Many-to-many merges
- Merging on an explicit common column name
- Merging on explicit different column names
- Merging on indexes

Pandas enables you to merge two DataFrames together very efficiently
- pd.merge(dataframe1, dataframe2)

The way Pandas performs the merge operation depends on the relationship of rows in the two DataFrames
- one-to-one
- many-to-one
- many-to-many

We show all these scenarios on the following slides

In this example:
- merge() detects that df1 and df2 both have a name column, and implicitly joins on these columns (it's a one-to-one join)

Ex4

In this example:
- merge() detects that df1 and df2 both have an office column, and implicitly joins on these columns (it's a many-to-one join)

Ex5

In this example:
- merge() detects that df1 and df2 both have a region column, and implicitly joins on these columns (it's a many-to-many join)

Ex6

In this example:
- We tell merge() to merge on the common column called name
- Why would you do this?

Ex7

In this example:
- The column names we want to join on are different
- So we must specify the name of the join column in both datasets

Ex8

In this example:
- In the 1st dataset, we join on the index
- In the 2nd dataset, we join on the navn column

Ex9a

Ex9b

In this example, both datasets are joined by index

Ex9c

- Inner joins
- Outer joins
- Left joins
- Right joins

As you've seen, the merge() and join() functions allow you to join data from two datasets
- E.g. based on a common column name in the two datasets

But consider this:
- What if one dataset has some rows that aren't matched in the other dataset - what should happen then?

There are 4 possibilities
- Inner join - Only return rows that match in both datasets
- Outer join - Return all rows, with blanks for the missing bits
- Left join - Return all the rows from the left-hand-side dataset
- Right join - Return all the rows from the right-hand-side dataset

In an inner join (this is the default):
- It only returns rows that are matched in both datasets

Ex10

In an outer join:
- It returns all rows, with blanks for the missing bits

Ex11

In a left join:
- It returns all the rows from the left-hand-side dataset

Ex12

In a right join:
- It returns all the rows from the right-hand-side dataset

Ex13