**Introduction**

In this lesson, you'll practice wrangling data with Spark. If you are familiar with both SQL and Python's pandas library, you'll notice quite a few similarities with the Spark SQL module and Spark Dataset API.

**Lesson Overview**

* Wrangling data with Spark
* Functional programming
* Read in and write out data
* Spark environment and Spark APIs
* RDD API

# Maps and Lambda Functions

For more about the theory and origins of lambda functions, take a look at this [**blog post**](http://palmstroem.blogspot.com/2012/05/lambda-calculus-for-absolute-dummies.html). Why are lambda functions called "lambda" functions?

According to legend, the inventor of Lambda Calculus, Alonzo Church, originally used the wedge symbol \wedge∧ as part of his notation. But the typsetter transcribing his manuscript used \lambda*λ* instead. You can read more about it in the blog post.

**FYI**

* The PySpark API allows you to write programs in Spark and ensures that your code uses functional programming practices. Underneath the hood, the Python code uses py4j to make calls to the Java Virtual Machine (JVM).

# Data Wrangling with Data Frames

##### Correction

You might notice in the screencast code that the SparkSession object wasn't instantiated explicitly. This is because Judit was using the same environment from a previous exercise. In general, you would have to instantiate an object using code like this:

spark = SparkSession \

.builder \

.appName("Wrangling Data") \

.getOrCreate()

### **Functions**

In the previous video, we've used a number of functions to manipulate our dataframe. Let's take a look at the different type of functions and their potential pitfalls.

#### **General functions**

We have used the following general functions that are quite similar to methods of pandas dataframes:

* select(): returns a new DataFrame with the selected columns
* filter(): filters rows using the given condition
* where(): is just an alias for filter()
* groupBy(): groups the DataFrame using the specified columns, so we can run aggregation on them
* sort(): returns a new DataFrame sorted by the specified column(s). By default the second parameter 'ascending' is True.
* dropDuplicates(): returns a new DataFrame with unique rows based on all or just a subset of columns
* withColumn(): returns a new DataFrame by adding a column or replacing the existing column that has the same name. The first parameter is the name of the new column, the second is an expression of how to compute it.

#### **Aggregate functions**

Spark SQL provides built-in methods for the most common aggregations such as count(), countDistinct(), avg(), max(), min(), etc. in the pyspark.sql.functions module. These methods are not the same as the built-in methods in the Python Standard Library, where we can find min() for example as well, hence you need to be careful not to use them interchangeably.

In many cases, there are multiple ways to express the same aggregations. For example, if we would like to compute one type of aggregate for one or more columns of the DataFrame we can just simply chain the aggregate method after a groupBy(). If we would like to use different functions on different columns, agg()comes in handy. For example agg({"salary": "avg", "age": "max"}) computes the average salary and maximum age.

#### **User defined functions (UDF)**

In Spark SQL we can define our own functions with the udf method from the pyspark.sql.functions module. The default type of the returned variable for UDFs is string. If we would like to return an other type we need to explicitly do so by using the different types from the pyspark.sql.types module.

#### **Window functions**

Window functions are a way of combining the values of ranges of rows in a DataFrame. When defining the window we can choose how to sort and group (with the partitionBy method) the rows and how wide of a window we'd like to use (described by rangeBetween or rowsBetween).

For further information see the [**Spark SQL, DataFrames and Datasets Guide**](https://spark.apache.org/docs/latest/sql-programming-guide.html) and the [**Spark Python API Docs**](https://spark.apache.org/docs/latest/api/python/index.html).

# Spark SQL

**Spark SQL resources**

Here are a few resources that you might find helpful when working with Spark SQL

* [**Spark SQL built-in functions**](https://spark.apache.org/docs/latest/api/sql/index.html)
* [**Spark SQL guide**](https://spark.apache.org/docs/latest/sql-getting-started.html)

**RDDS**

RDDs are a low-level abstraction of the data. In the first version of Spark, you worked directly with RDDs. You can think of RDDs as long lists distributed across various machines. You can still use RDDs as part of your Spark code although data frames and SQL are easier. This course won't go into the details of RDD syntax, but you can find some further explanation of the difference between RDDs and DataFrames in Databricks' [**A Tale of Three Apache Spark APIs: RDDs, DataFrames, and Datasets**](https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html) blog post.

Here is a link to the Spark documentation's [**RDD programming guide**](https://spark.apache.org/docs/latest/rdd-programming-guide.html).