# Apache PigLatin

Sinziana Gafitanu

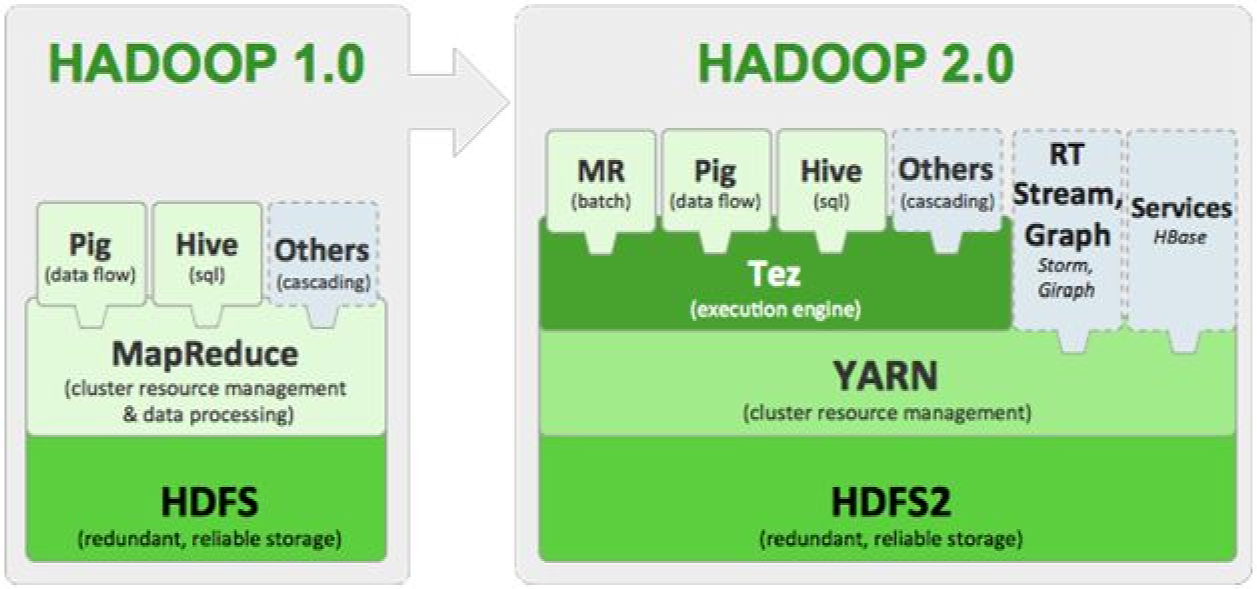
## Pig vs PigLatin

**PigLatin** - High Level language that compiles in a sequence of MapReduce programs.

**Pig** - The runtime environment where the PigLatin scripts are executed.

**Grunt Shell** - Shell for running Pig in interactive mode.

# Pig and Hadoop



# Running Pig

## II. Interactive Mode

a. Local

```
pig -x local
```

b. Remote

```
pig -x
```

## II. Script Mode

a. Local

```
pig -x local myscript.pig
```

b. Remote

```
pig -x myscript.pig
```

## Comments in scripts

---

```
/*...*/   - Block Comments
---       - Line Comments
```

## Operators

---

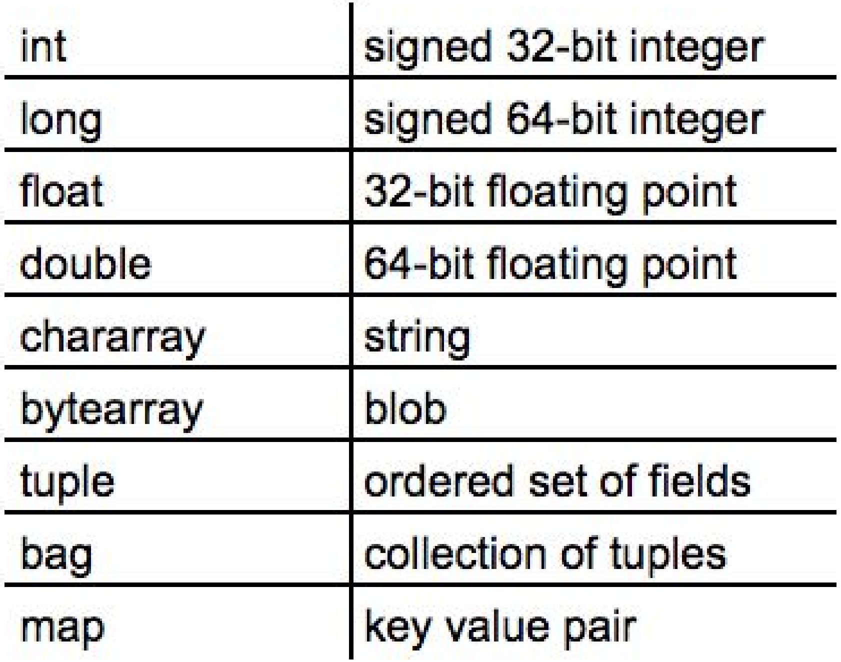
```
 +  -  /  *  %  ?

and or not

== != <  >  <= >=

is null  is not null
```

## Pig Data Type



| | |
|---|---|
| int | signed 32-bit integer |
| long | signed 64-bit integer |
| float | 32-bit floating point |
| double | 64-bit floating point |
| chararray | string |
| bytearray | blob |
| tuple | ordered set of fields |
| bag | collection of tuples |
| map | key value pair |

## Local Pig

---

```
docker run --rm -it -v /Users/sgafitan/apachePig:/data:rw
chalimartines/local-pig
```

## LOAD operator

---

Format:

```
data = LOAD ‘<data>’ [using<function()>][as(<schema>)];
```

Example:

```
code = LOAD '/data/AirportCodeLocationLookupClean.csv'
USING PigStorage();

delay = LOAD '/data/FlightDelaysWithAirportCodes.csv';

wthr = LOAD '/data/FlightWeatherWithAirportCode.csv';
```

## LOAD operator with schema

Format:

```
data = LOAD ‘<data>’ [USING<function()>][AS(<schema>)];
```

Example:

```
code = LOAD '/data/AirportCodeLocationLookupClean.csv'
USING PigStorage(',') AS(AIRPORT_ID:int,
AIRPORT:chararray, DISPLAY_AIRPORT_NAME:chararray,
LATITUDE:double, LONGITUDE:double);
```

## DUMP operator

---

Format:

```
DUMP <alias>;
```

Example:

```
DUMP code;
```

## DESCRIBE operator

---

Format:

```
DESCRIBE <alias>;
```

Example:

```
DESCRIBE code;
```

## FILTER operator

---

Format:

```
FILTER <alias> BY expression;
```

Example:

```
code_u = FILTER code BY LATITUDE > 50;
```

## ORDER BY operator

---

Format:

```
ORDER alias BY col [ASC|DESC]
```

Example:

```
code_o = ORDER code BY LATITUDE ASC;
```

# ORDER BY operator

---

Format:

```
ORDER alias BY col [ASC|DESC];
```

Example:

```
code_o = ORDER code BY LATITUDE ASC;
```

## GROUP BY operator

Format:

```
alias = GROUP alias { ALL | BY expression} [, alias ALL
| BY expression …] [USING 'collected' | 'merge']
[PARTITION BY partition] [PARALLEL n];
```

Example:

```
wthr_year = GROUP wthr BY $0;
```

http://pig.apache.org/docs/r0.12.0/basic.html#group

## FOR EACH operator

Format:

```
alias = FOREACH … GENERATE ...;
```

Example:

```
code_f = FOREACH dataset GENERATE
AIRPORT_ID,FLOOR(LATITUDE),FLOOR(LONGITUDE);
```

http://pig.apache.org/docs/r0.12.0/basic.html#group

## JOIN operator

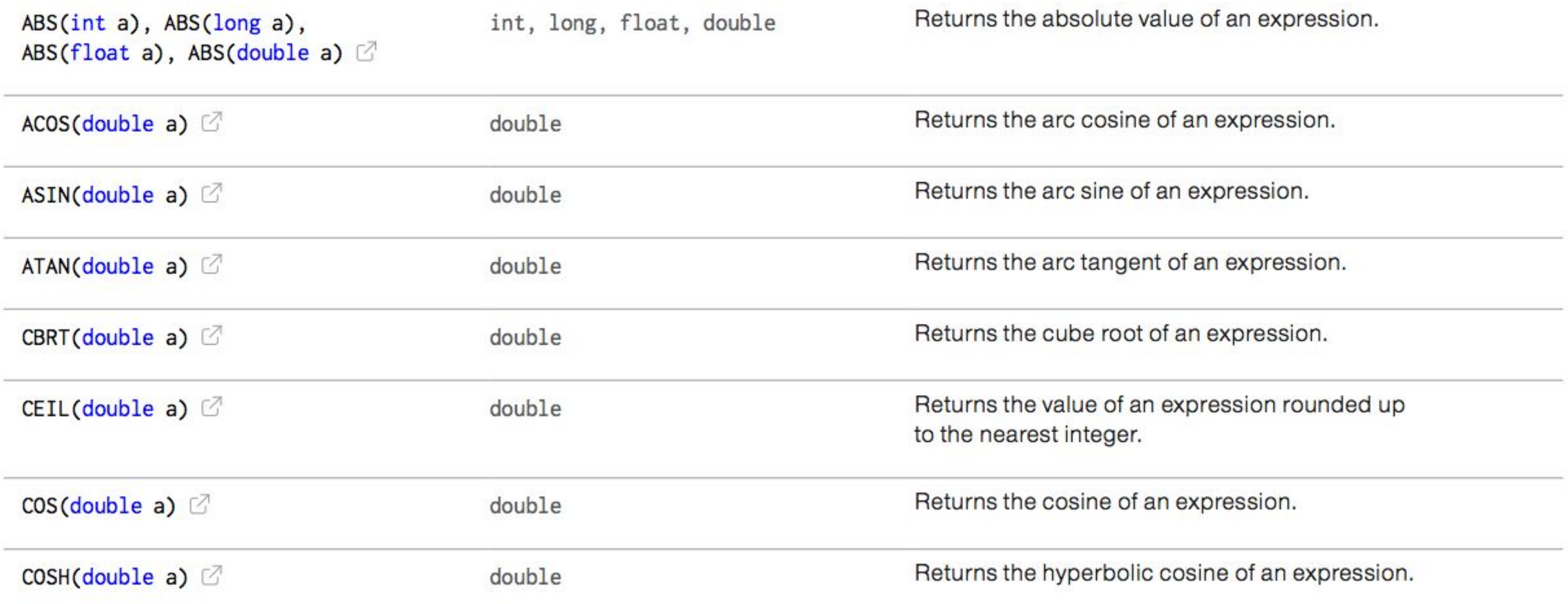
Format:

```
alias = JOIN smaller_alias BY expression
[LEFT|RIGHT|OUTER], larger_alias BY expression
```

Example:

```
joined = JOIN code_f by (AIRPORT_ID), code BY
(AIRPORT_ID);
```

# Mathematical Functions

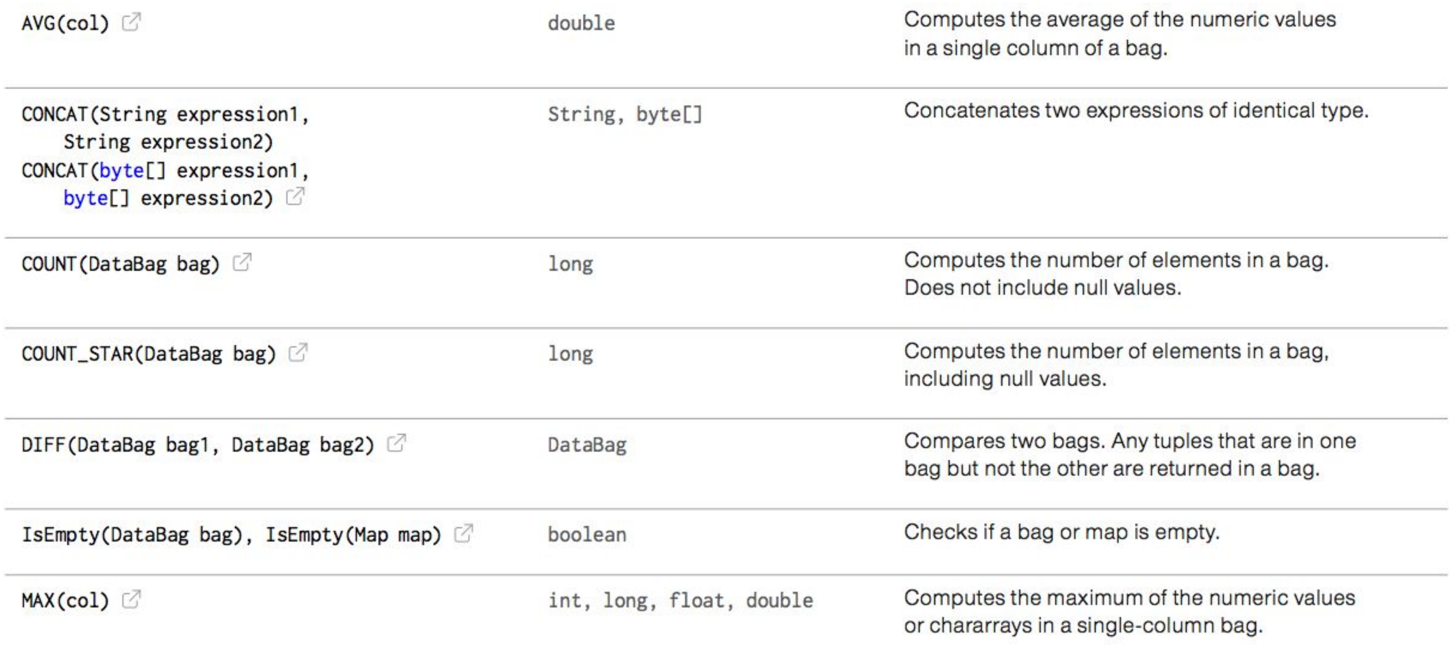


| | | |
|---|---|---|
| `ABS(int a), ABS(long a), ABS(float a), ABS(double a)` | `int, long, float, double` | Returns the absolute value of an expression. |
| `ACOS(double a)` | `double` | Returns the arc cosine of an expression. |
| `ASIN(double a)` | `double` | Returns the arc sine of an expression. |
| `ATAN(double a)` | `double` | Returns the arc tangent of an expression. |
| `CBRT(double a)` | `double` | Returns the cube root of an expression. |
| `CEIL(double a)` | `double` | Returns the value of an expression rounded up to the nearest integer. |
| `COS(double a)` | `double` | Returns the cosine of an expression. |
| `COSH(double a)` | `double` | Returns the hyperbolic cosine of an expression. |

… and more;

Source: http://blog.mortardata.com/post/77915006384/introducing-the-pig-cheat-sheet

# Evaluation Functions

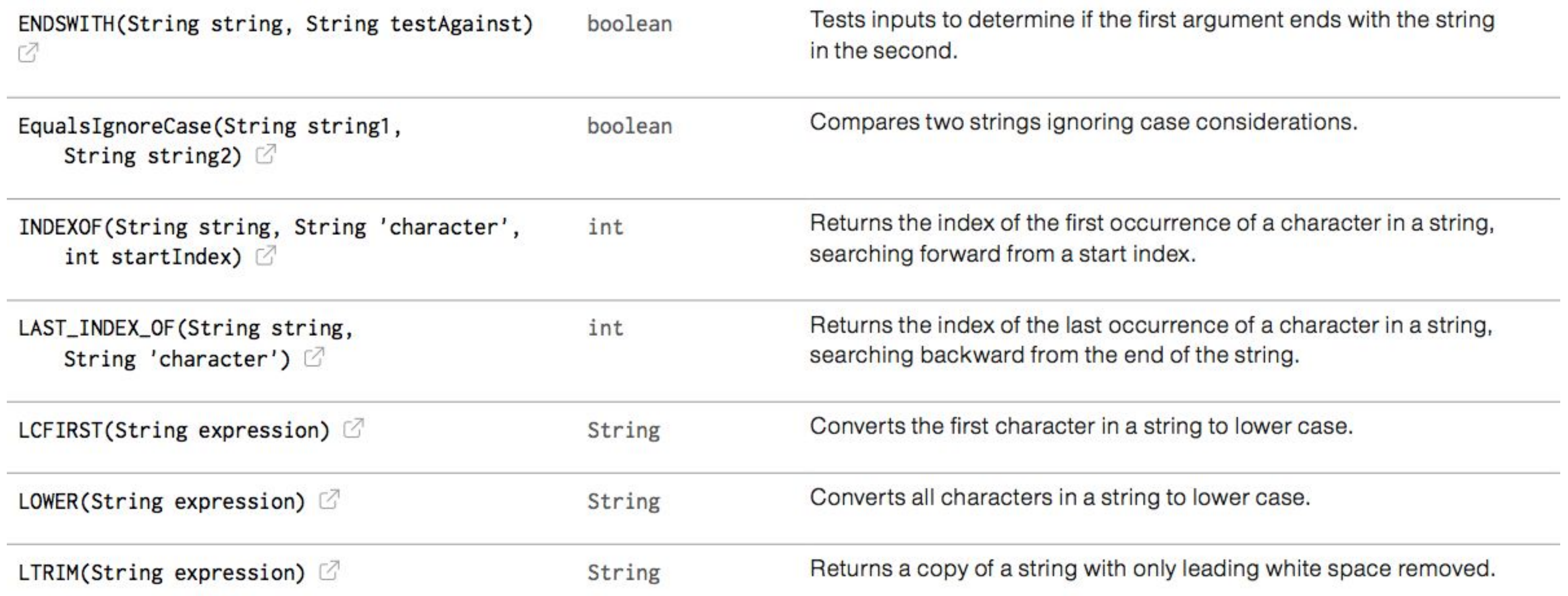


| | | |
|---|---|---|
| `AVG(col)` | `double` | Computes the average of the numeric values in a single column of a bag. |
| `CONCAT(String expression1, String expression2)` `CONCAT(byte[] expression1, byte[] expression2)` | `String, byte[]` | Concatenates two expressions of identical type. |
| `COUNT(DataBag bag)` | `long` | Computes the number of elements in a bag. Does not include null values. |
| `COUNT_STAR(DataBag bag)` | `long` | Computes the number of elements in a bag, including null values. |
| `DIFF(DataBag bag1, DataBag bag2)` | `DataBag` | Compares two bags. Any tuples that are in one bag but not the other are returned in a bag. |
| `IsEmpty(DataBag bag), IsEmpty(Map map)` | `boolean` | Checks if a bag or map is empty. |
| `MAX(col)` | `int, long, float, double` | Computes the maximum of the numeric values or chararrays in a single-column bag. |

… and more;

Source: http://blog.mortardata.com/post/77915006384/introducing-the-pig-cheat-sheet

# String Functions

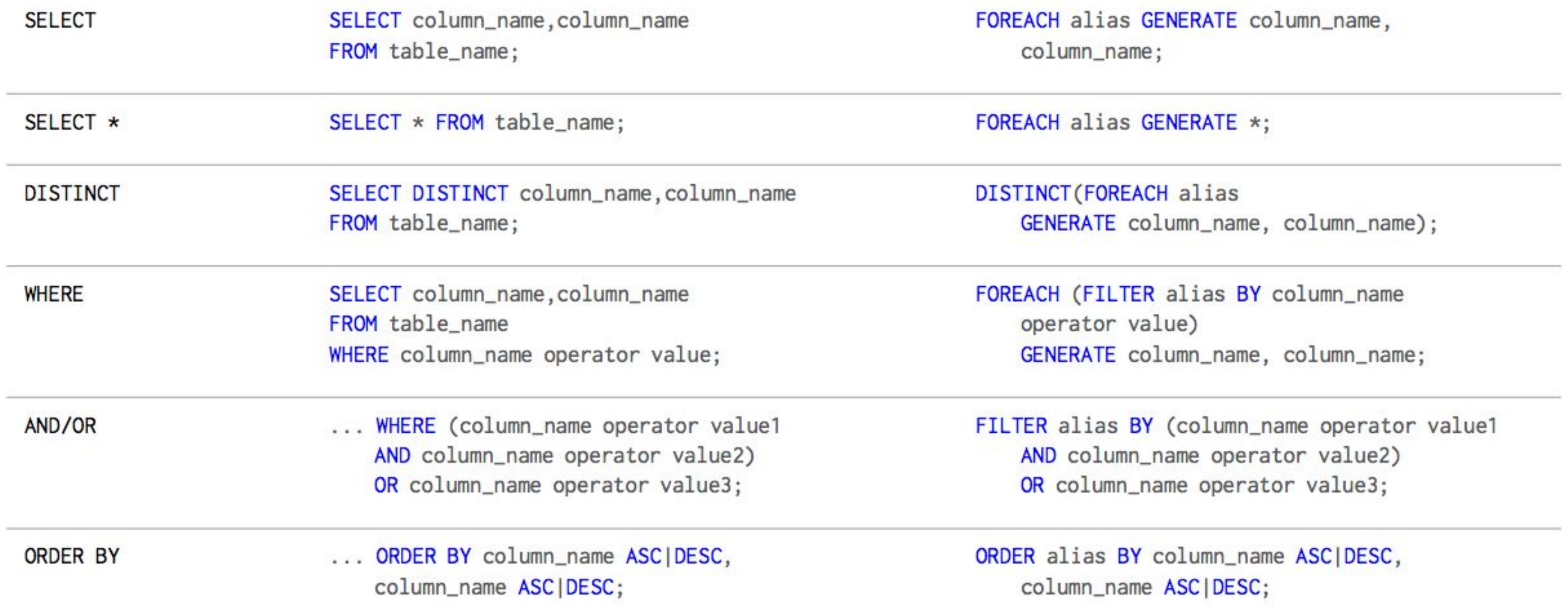


| | | |
|---|---|---|
| `ENDSWITH(String string, String testAgainst)` | `boolean` | Tests inputs to determine if the first argument ends with the string in the second. |
| `EqualsIgnoreCase(String string1, String string2)` | `boolean` | Compares two strings ignoring case considerations. |
| `INDEXOF(String string, String 'character', int startIndex)` | `int` | Returns the index of the first occurrence of a character in a string, searching forward from a start index. |
| `LAST_INDEX_OF(String string, String 'character')` | `int` | Returns the index of the last occurrence of a character in a string, searching backward from the end of the string. |
| `LCFIRST(String expression)` | `String` | Converts the first character in a string to lower case. |
| `LOWER(String expression)` | `String` | Converts all characters in a string to lower case. |
| `LTRIM(String expression)` | `String` | Returns a copy of a string with only leading white space removed. |

… and more;

Source: http://blog.mortardata.com/post/77915006384/introducing-the-pig-cheat-sheet

# SQL -> Pig



| | | |
|---|---|---|
| SELECT | `SELECT column_name,column_name`<br>`FROM table_name;` | `FOREACH alias GENERATE column_name,`<br>`    column_name;` |
| SELECT * | `SELECT * FROM table_name;` | `FOREACH alias GENERATE *;` |
| DISTINCT | `SELECT DISTINCT column_name,column_name`<br>`FROM table_name;` | `DISTINCT(FOREACH alias`<br>`    GENERATE column_name, column_name);` |
| WHERE | `SELECT column_name,column_name`<br>`FROM table_name`<br>`WHERE column_name operator value;` | `FOREACH (FILTER alias BY column_name`<br>`    operator value)`<br>`    GENERATE column_name, column_name;` |
| AND/OR | `... WHERE (column_name operator value1`<br>`    AND column_name operator value2)`<br>`    OR column_name operator value3;` | `FILTER alias BY (column_name operator value1`<br>`    AND column_name operator value2)`<br>`    OR column_name operator value3;` |
| ORDER BY | `... ORDER BY column_name ASC\|DESC,`<br>`    column_name ASC\|DESC;` | `ORDER alias BY column_name ASC\|DESC,`<br>`    column_name ASC\|DESC;` |

… and more;

Source: http://blog.mortardata.com/post/77915006384/introducing-the-pig-cheat-sheet

## User Defined Functions

Much of Pig’s power comes from the fact that it can be extended using other languages. User-defined functions (UDFs) can be written in Java, Python, Jython, Ruby, and JavaScript.

Java Example:

```
REGISTER ../udfs/java/my_jar.jar;

DEFINE MyUDF the.full.package.path.MyUDF();
```

Source: http://help.mortardata.com/technologies/pig/writing_java_udfs