CalcuList: an educational language to practice with MapReduce computation schemes

Domenico Saccà Angelo Furfaro

DIMES, Università della Calabria, 87036 Rende, Italy
{sacca,a.furfaro}@unical.it

Json and MapReduce

- Json (JavaScript Object Notation) is language-independent data interchange format that is now widespread in many applications and support for Json has been added to the standard libraries of many programming languages

Json and MapReduce

- Json (JavaScript Object Notation) is language-independent data interchange format that is now widespread in many applications and support for Json has been added to the standard libraries of many programming languages
- a list of jsons can be thought of as a dataset of a document NoSQL datastore such as MongoDB and CouchDB

Json and MapReduce

- Json (JavaScript Object Notation) is language-independent data interchange format that is now widespread in many applications and support for Json has been added to the standard libraries of many programming languages
- a list of jsons can be thought of as a dataset of a document NoSQL datastore such as MongoDB and CouchDB
- MapReduce is a popular model in distributed computing for processing and generating large data sets spread over clusters of computers

Json and MapReduce

- Json (JavaScript Object Notation) is language-independent data interchange format that is now widespread in many applications and support for Json has been added to the standard libraries of many programming languages
- a list of jsons can be thought of as a dataset of a document NoSQL datastore such as MongoDB and CouchDB
- MapReduce is a popular model in distributed computing for processing and generating large data sets spread over clusters of computers
- MapReduce has been inspired by the functions map and reduce commonly used in functional programming.

Json and MapReduce in CalcuList

- CalcuList (*Calcu*lator with *List* manipulation) is an educational language for teaching functional programming extended with some imperative and side-effect features, which are enabled under explicit request by the programmer.

Json and MapReduce in CalcuList

- CalcuList (*Calcu*lator with *List* manipulation) is an educational language for teaching functional programming extended with some imperative and side-effect features, which are enabled under explicit request by the programmer.
- In addition to strings and lists, the language natively supports json objects and may be effectively used to implement generic MapReduce recursive procedures to manipulate json lists.

Json and MapReduce in CalcuList

- CalcuList (*Calcu*lator with *List* manipulation) is an educational language for teaching functional programming extended with some imperative and side-effect features, which are enabled under explicit request by the programmer.
- In addition to strings and lists, the language natively supports json objects and may be effectively used to implement generic MapReduce recursive procedures to manipulate json lists.
- CalcuList can be used as a tool for teaching advanced query algorithms for document datastores such as MongoDB and CouchDB.

Types, Variables and Queries

- The main interface to CalcuList, like other languages (e.g. Python, Scala), is a REPL environment where the user can define functions and issue valid expressions (queries in CalcuList), which in turn are parsed, evaluated and printed before the control is given back to the user.

Types, Variables and Queries

- The main interface to CalcuList, like other languages (e.g. Python, Scala), is a REPL environment where the user can define functions and issue valid expressions (queries in CalcuList), which in turn are parsed, evaluated and printed before the control is given back to the user.
- The basic types for CalcuList are six: (1) *double*, (2) *int* , (3) *char*, (4) *bool* (with values *true* or *false*), (5) *null* (that has a unique value named `null` as well) and (6) *type* (whose values are all the other types: *double*, *int*, etc.).

Types, Variables and Queries

- The main interface to CalcuList, like other languages (e.g. Python, Scala), is a REPL environment where the user can define functions and issue valid expressions (queries in CalcuList), which in turn are parsed, evaluated and printed before the control is given back to the user.
- The basic types for CalcuList are six: (1) *double*, (2) *int* , (3) *char*, (4) *bool* (with values *true* or *false*), (5) *null* (that has a unique value named `null` as well) and (6) *type* (whose values are all the other types: *double*, *int*, etc.).
- The language also supports three compound types: *string* (immutable sequences of characters), *list* (sequences of elements of any type), and *json*.

Types, Variables and Queries

- The main interface to CalcuList, like other languages (e.g. Python, Scala), is a REPL environment where the user can define functions and issue valid expressions (queries in CalcuList), which in turn are parsed, evaluated and printed before the control is given back to the user.
- The basic types for CalcuList are six: (1) *double*, (2) *int* , (3) *char*, (4) *bool* (with values *true* or *false*), (5) *null* (that has a unique value named `null` as well) and (6) *type* (whose values are all the other types: *double*, *int*, etc.).
- The language also supports three compound types: *string* (immutable sequences of characters), *list* (sequences of elements of any type), and *json*.
- As high order functions are supported, *function* is a type as well.

A simple session

A simple session

```
>> x=2+.1;
```

A simple session

```
>> x=2+.1;
>> x *= 2;
```

A simple session

```
>> x=2+.1;
>> x *= 2;
>> ^x;
```

A simple session

```
>> x=2+.1;
>> x *= 2;
>> ^x;
4.2
```

A simple session

```
>> x=2+.1;
>> x *= 2;
>> ^x;
4.2
>>z='A'+1=='B';
```

A simple session

```
>> x=2+.1;
>> x *= 2;
>> ^x;
4.2
>>z='A'+1=='B';
>> ^z;
```

A simple session

```
>> x=2+.1;
>> x *= 2;
>> ^x;
4.2
>>z='A'+1=='B';
>> ^z;
true
```

A simple session

```
>> x=2+.1;
>> x *= 2;
>> ^x;
4.2
>>z='A'+1=='B';
>> ^z;
true
>> y="Hello "+"Worl"+'d';
```

A simple session

```
>> x=2+.1;
>> x *= 2;
>> ^x;
4.2
>>z='A'+1=='B';
>> ^z;
true
>> y="Hello "+"Worl"+'d';
>>^y;
```

A simple session

```
>> x=2+.1;
>> x *= 2;
>> ^x;
4.2
>>z='A'+1=='B';
>> ^z;
true
>> y="Hello "+"Worl"+'d';
>>^y;
Hello World
```

A simple session

```
>> x=2+.1;
>> x *= 2;
>> ^x;
4.2
>>z='A'+1=='B';
>> ^z;
true
>> y="Hello "+"Worl"+'d';
>>^y;
Hello World
>> ^y[0:1]+'i '+y[5:];
```

A simple session

```
>> x=2+.1;
>> x *= 2;
>> ^x;
4.2
>>z='A'+1=='B';
>> ^z;
true
>> y="Hello "+"Worl"+'d';
>>^y;
Hello World
>> ^y[0:1]+'i '+y[5:];
Hi World
```

A simple session

```
>> x=2+.1;
>> x *= 2;
>> ^x;
4.2
>>z='A'+1=='B';
>> ^z;
true
>> y="Hello "+"Worl"+'d';
>>^y;
Hello World
>> ^y[0:1]+'i '+y[5:];
Hi World
>>^z@type;
```

A simple session

```
>> x=2+.1;
>> x *= 2;
>> ^x;
4.2
>>z='A'+1=='B';
>> ^z;
true
>> y="Hello "+"Worl"+'d';
>>^y;
Hello World
>> ^y[0:1]+'i '+y[5:];
Hi World
>>^z@type;
bool
```

Functions

- A function is defined as $\texttt{fname}(\texttt{par}_1, \ldots, \texttt{par}_n) :< \texttt{expr} >$.

Functions

- A function is defined as $\texttt{fname}(\texttt{par}_1, \ldots, \texttt{par}_n) :< \texttt{expr} >$.
- No type is to be assigned to parameters and to the return value so type checking is done at run time when all the types become available. Lazy evaluation is not supported.

Functions

- A function is defined as $\texttt{fname}(\texttt{par}_1, \ldots, \texttt{par}_n) :< \texttt{expr} >$.
- No type is to be assigned to parameters and to the return value so type checking is done at run time when all the types become available. Lazy evaluation is not supported.
- No side effects are allowed in the basic definitions of functions, i.e., functional operations do not modify current global variables and always create new data objects. Differently from Python, global variables are not accessible inside a function so that a possible change of state does not affect the function behavior.

A Session with Functions

A Session with Functions

```
>> fib(x):  x <=1?  x:  fib(x-1)+fib(x-2);
```

A Session with Functions

```
>> fib(x):  x <=1?  x:  fib(x-1)+fib(x-2);
>> ^fib(10);
```

A Session with Functions

```
>> fib(x):  x <=1?  x:  fib(x-1)+fib(x-2);
>> ^fib(10);
55;
```

A Session with Functions

```
>> fib(x):  x <=1?  x:  fib(x-1)+fib(x-2);
>> ^fib(10);
55;
>> !clops;
```

A Session with Functions

```
>> fib(x):  x <=1?  x:  fib(x-1)+fib(x-2);
>> ^fib(10);
55;
>> !clops;
38157;
```

A Session with Functions

```
>> fib(x):  x <=1?  x:  fib(x-1)+fib(x-2);
>> ^fib(10);
55;
>> !clops;
38157;
>> fb(x,f2,f1,k):  x==k?f1:  fb(x,f1,f1+f2,k+1);
```

A Session with Functions

```
>> fib(x):  x <=1?  x:  fib(x-1)+fib(x-2);
>> ^fib(10);
55;
>> !clops;
38157;
>> fb(x,f2,f1,k):  x==k?f1:  fb(x,f1,f1+f2,k+1);
>> fibe(x) :  x <= 1?  x:  fb(x,0,1,1);
```

A Session with Functions

```
>> fib(x):  x <=1?  x:  fib(x-1)+fib(x-2);
>> ^fib(10);
55;
>> !clops;
38157;
>> fb(x,f2,f1,k):  x==k?f1:  fb(x,f1,f1+f2,k+1);
>> fibe(x) :  x <= 1?  x:  fb(x,0,1,1);
>> ^fibe(10); 55;
```

A Session with Functions

```
>> fib(x):  x <=1?  x:  fib(x-1)+fib(x-2);
>> ^fib(10);
55;
>> !clops;
38157;
>> fb(x,f2,f1,k):  x==k?f1:  fb(x,f1,f1+f2,k+1);
>> fibe(x) :  x <= 1?  x:  fb(x,0,1,1);
>> ^fibe(10); 55;
```

A Session with Functions

```
>> fib(x):  x <=1?  x:  fib(x-1)+fib(x-2);
>> ^fib(10);
55;
>> !clops;
38157;
>> fb(x,f2,f1,k):  x==k?f1:  fb(x,f1,f1+f2,k+1);
>> fibe(x) :  x <= 1?  x:  fb(x,0,1,1);
>> ^fibe(10); 55;
>> !clops; 3360;
```

A Session with Functions

```
>> fib(x):  x <=1?  x:  fib(x-1)+fib(x-2);
>> ^fib(10);
55;
>> !clops;
38157;
>> fb(x,f2,f1,k):  x==k?f1:  fb(x,f1,f1+f2,k+1);
>> fibe(x) :  x <= 1?  x:  fb(x,0,1,1);
>> ^fibe(10); 55;
>> !clops; 3360;
>> ^fib(30); 832040;
```

A Session with Functions

```
>> fib(x):  x <=1?  x:  fib(x-1)+fib(x-2);
>> ^fib(10);
55;
>> !clops;
38157;
>> fb(x,f2,f1,k):  x==k?f1:  fb(x,f1,f1+f2,k+1);
>> fibe(x) :  x <= 1?  x:  fb(x,0,1,1);
>> ^fibe(10); 55;
>> !clops; 3360;
>> ^fib(30); 832040;
>> !clops; 580241737;
```

A Session with Functions

```
>> fib(x):  x <=1?  x:  fib(x-1)+fib(x-2);
>> ^fib(10);
55;
>> !clops;
38157;
>> fb(x,f2,f1,k):  x==k?f1:  fb(x,f1,f1+f2,k+1);
>> fibe(x) :  x <= 1?  x:  fb(x,0,1,1);
>> ^fibe(10); 55;
>> !clops; 3360;
>> ^fib(30); 832040;
>> !clops; 580241737;
>> ^fibe(30); 832040;
```

A Session with Functions

```
>> fib(x):  x <=1?  x:  fib(x-1)+fib(x-2);
>> ^fib(10);
55;
>> !clops;
38157;
>> fb(x,f2,f1,k):  x==k?f1:  fb(x,f1,f1+f2,k+1);
>> fibe(x) :  x <= 1?  x:  fb(x,0,1,1);
>> ^fibe(10); 55;
>> !clops; 3360;
>> ^fib(30); 832040;
>> !clops; 580241737;
>> ^fibe(30); 832040;
>> !clops; 9920;
```

A Session with Functions and Lists

A Session with Functions and Lists

```
>> member(x,L) : L!=[] && ( x == L[.] || member(x,L[>]) );
```

A Session with Functions and Lists

```
>> member(x,L) : L!=[] && ( x == L[.] || member(x,L[>]) );
>> sumL(L) : L==[]? 0: L[.]+sumL(L[>]);
```

A Session with Functions and Lists

```
>> member(x,L) : L!=[] && ( x == L[.] || member(x,L[>]) );
>> sumL(L) : L==[]? 0: L[.]+sumL(L[>]);
>> range(x1,x2) : x1 > x2?[] : [x1 | range(x1 + 1,x2)];
```

A Session with Functions and Lists

```
>> member(x,L) : L!=[] && ( x == L[.] || member(x,L[>]) );
>> sumL(L) : L==[]? 0: L[.]+sumL(L[>]);
>> range(x1,x2) : x1 > x2?[] : [x1 | range(x1 + 1,x2)];
>> r16= range(1,16);
```

A Session with Functions and Lists

```
>> member(x,L) : L!=[] && ( x == L[.] || member(x,L[>]) );
>> sumL(L) : L==[]? 0: L[.]+sumL(L[>]);
>>range(x1,x2) : x1 > x2?[] : [x1 | range(x1 + 1,x2)];
>> r16= range(1,16);
>>^sumL(r16);
```

A Session with Functions and Lists

```
>> member(x,L) : L!=[] && ( x == L[.] || member(x,L[>]) );
>> sumL(L) : L==[]? 0: L[.]+sumL(L[>]);
>>range(x1,x2) : x1 > x2?[] : [x1 | range(x1 + 1,x2)];
>> r16= range(1,16);
>>^sumL(r16);
136
```

Higher-Order Functions

- CalcuList supports higher-order functions since a function parameter can also be a function and a function may return a function

Higher-Order Functions

- CalcuList supports higher-order functions since a function parameter can also be a function and a function may return a function
- A function parameter `f` is written as `f/`*n*, where *n* is the arity of the function `f`. In addition, adding */n* after a function head `g(...)` prescribes that the function g must return a function with arity *n*.

Higher-Order Functions

- CalcuList supports higher-order functions since a function parameter can also be a function and a function may return a function
- A function parameter `f` is written as `f/`*n*, where *n* is the arity of the function `f`. In addition, adding */*n after a function head `g(...)` prescribes that the function g must return a function with arity *n*.
- As an example, consider the higher-order function `twice(f/1)/1 : lambda x : f(f(x));`

Higher-Order Functions

- CalcuList supports higher-order functions since a function parameter can also be a function and a function may return a function
- A function parameter `f` is written as `f/`*n*, where *n* is the arity of the function `f`. In addition, adding */*n after a function head `g(...)` prescribes that the function g must return a function with arity *n*.
- As an example, consider the higher-order function `twice(f/1)/1 : lambda x : f(f(x));`
- the query `^twice(lambda x:  x+3)(7);`

Higher-Order Functions

- CalcuList supports higher-order functions since a function parameter can also be a function and a function may return a function
- A function parameter `f` is written as `f/`*n*, where *n* is the arity of the function `f`. In addition, adding */*n after a function head `g(...)` prescribes that the function g must return a function with arity *n*.
- As an example, consider the higher-order function `twice(f/1)/1 : lambda x : f(f(x));`
- the query `^twice(lambda x:  x+3)(7);`

  returns the value 13

A Session with Higher-Order Functions

A Session with Higher-Order Functions

```
>> map(L, f/1, m/1) : L == []? [] : f(L[.])?
                        [m(L[.]) | map(L[>], f, m)] : map(L[>], f, m);
```

A Session with Higher-Order Functions

```
>> map(L, f/1, m/1) : L == []? [] : f(L[.])?
                      [m(L[.]) | map(L[>], f, m)] : map(L[>], f, m);
>> d2or3(x) :  x%2==0 || x%3==0;
```

A Session with Higher-Order Functions

```
>> map(L,f/1,m/1) : L == []? [] : f(L[.])?
                    [m(L[.])|map(L[>],f,m)] : map(L[>],f,m);
>> d2or3(x) :  x%2==0 || x%3==0;
>> ^map(r16,d2or3,lambda x: x*x*x);
```

A Session with Higher-Order Functions

```
>> map(L,f/1,m/1) : L == []? [] : f(L[.])?
                    [m(L[.])|map(L[>],f,m)] : map(L[>],f,m);
>> d2or3(x) :  x%2==0 || x%3==0;
>> ^map(r16,d2or3,lambda x: x*x*x);
[ 8, 27, 64, 216, 512, 729, 1000, 1728, 2744,
3375, 4096 ]
```

A Session with Higher-Order Functions

```
>> map(L, f/1, m/1) : L == []? [] : f(L[.])?
                    [m(L[.]) | map(L[>], f, m)] : map(L[>], f, m);
>> d2or3(x) :  x%2==0 || x%3==0;
>> ^map(r16,d2or3,lambda x: x*x*x);
[ 8, 27, 64, 216, 512, 729, 1000, 1728, 2744,
3375, 4096 ]
>> reduce(L, f/2, init) : L == []?init :
                    f(L[.], reduce(L[>], f, init));
```

A Session with Higher-Order Functions

```
>> map(L, f/1, m/1) : L == []? [] : f(L[.])?
                [m(L[.]) | map(L[>], f, m)] : map(L[>], f, m);
>> d2or3(x) :  x%2==0 || x%3==0;
>> ^map(r16,d2or3,lambda x: x*x*x);
[ 8, 27, 64, 216, 512, 729, 1000, 1728, 2744,
3375, 4096 ]
>> reduce(L, f/2, init) : L == []?init :
                f(L[.], reduce(L[>], f, init));
>> ^reduce(map(r16, d2or3, lambda x : x * x * x),
                lambda x, y : x + y, 0);
```

A Session with Higher-Order Functions

```
>> map(L, f/1, m/1) : L == []? [] : f(L[.])?
                [m(L[.]) | map(L[>], f, m) ] : map(L[>], f, m);
>> d2or3(x) :  x%2==0 || x%3==0;
>> ^map(r16,d2or3,lambda x: x*x*x);
[ 8, 27, 64, 216, 512, 729, 1000, 1728, 2744,
3375, 4096 ]
>> reduce(L, f/2, init) : L == []?init :
                f(L[.], reduce(L[>], f, init));
>> ^reduce(map(r16, d2or3, lambda x : x * x * x),
                lambda x, y : x + y, 0);
14499
```

A Session with Higher-Order Functions

```
>> map(L, f/1, m/1) : L == []? [] : f(L[.])?
                [m(L[.]) | map(L[>], f, m) ] : map(L[>], f, m);
>> d2or3(x) :  x%2==0 || x%3==0;
>> ^map(r16,d2or3,lambda x: x*x*x);
[ 8, 27, 64, 216, 512, 729, 1000, 1728, 2744,
3375, 4096 ]
>> reduce(L, f/2, init) : L == []?init :
                f(L[.], reduce(L[>], f, init));
>> ^reduce(map(r16, d2or3, lambda x : x * x * x),
                lambda x, y : x + y, 0);
14499
>> d2and3(x) :  x%2==0 && x%3==0;
```

A Session with Higher-Order Functions

```
>> map(L, f/1, m/1) : L == []? [] : f(L[.])?
                [m(L[.]) | map(L[>], f, m)] : map(L[>], f, m);
>> d2or3(x) :  x%2==0 || x%3==0;
>> ^map(r16,d2or3,lambda x: x*x*x);
[ 8, 27, 64, 216, 512, 729, 1000, 1728, 2744,
3375, 4096 ]
>> reduce(L, f/2, init) : L == []?init :
                f(L[.], reduce(L[>], f, init));
>> ^reduce(map(r16, d2or3, lambda x : x * x * x),
                lambda x, y : x + y, 0);
14499
>> d2and3(x) :  x%2==0 && x%3==0;
>> ^reduce(map(r16, d2and3, lambda x : x * x),
                lambda x, y : x * y, 1);
```

A Session with Higher-Order Functions

```
>> map(L, f/1, m/1) : L == []? [] : f(L[.])?
                [m(L[.]) | map(L[>], f, m)] : map(L[>], f, m);
>> d2or3(x) :  x%2==0 || x%3==0;
>> ^map(r16,d2or3,lambda x: x*x*x);
[ 8, 27, 64, 216, 512, 729, 1000, 1728, 2744,
3375, 4096 ]
>> reduce(L, f/2, init) : L == []?init :
                f(L[.], reduce(L[>], f, init));
>> ^reduce(map(r16, d2or3, lambda x : x * x * x),
                lambda x, y : x + y, 0);
14499
>> d2and3(x) :  x%2==0 && x%3==0;
>> ^reduce(map(r16, d2and3, lambda x : x * x),
                lambda x, y : x * y, 1);
5184
```

ReduceCount with Imperative Features

```
>> rc1*(_,_) :  null ;
```

ReduceCount with Imperative Features

```
>> rc1*(_,_) :  null ;
>> rc2*(L,M,C): M==[]?rc1(L[ >],[[L [.],1]| C ]):
    L[.]==M [.][0]?{!M[0][1]+=1!} rc1(L[>],C):
                    rc2(L,M[>],C);
```

ReduceCount with Imperative Features

```
>> rc1*(_,_) :  null ;
>> rc2*(L,M,C): M==[]?rc1(L[ >],[[L [.],1]| C ]):
    L[.]==M [.][0]?{!M[0][1]+=1!} rc1(L[>],C):
                        rc2(L,M[>],C);
>> rc1*(L,M) : L ==[]?  M: rc2(L,M,M);
```

ReduceCount with Imperative Features

```
>> rc1*(_,_) :  null ;
>> rc2*(L,M,C): M==[]?rc1(L[ >],[[L [.],1]| C ]):
   L[.]==M [.][0]?{!M[0][1]+=1!} rc1(L[>],C):
                    rc2(L,M[>],C);
>> rc1*(L,M) : L ==[]?  M: rc2(L,M,M);
>> reduceCount*(L) : rc1(L ,[]);
```

ReduceCount with Imperative Features

```
>> rc1*(_,_) :  null ;
>> rc2*(L,M,C): M==[]?rc1(L[ >],[[L [.],1]| C ]):
   L[.]==M [.][0]?{!M[0][1]+=1!} rc1(L[>],C):
                    rc2(L,M[>],C);
>> rc1*(L,M) : L ==[]?  M: rc2(L,M,M);
>> reduceCount*(L) : rc1(L ,[]);
>> ^reduceCount("Hello_World"@list);
```

ReduceCount with Imperative Features

```
>> rc1*(_,_) :  null ;
>> rc2*(L,M,C): M==[]?rc1(L[ >],[[L [.],1]| C ]):
   L[.]==M [.][0]?{!M[0][1]+=1!} rc1(L[>],C):
                    rc2(L,M[>],C);
>> rc1*(L,M) : L ==[]?  M: rc2(L,M,M);
>> reduceCount*(L) : rc1(L ,[]);
>> ^reduceCount("Hello_World"@list);
[ [ 'H', 1 ], [ 'e', 1 ], [ 'l', 3 ], [ 'o', 2 ],
[ '_', 1 ], [ 'W', 1 ], [ 'r', 1 ], [ 'd', 1 ] ]
```

ReduceCount with Imperative Features

```
>> rc1*(_,_) :  null ;
>> rc2*(L,M,C): M==[]?rc1(L[ >],[[L [.],1]| C ]):
   L[.]==M [.][0]?{!M[0][1]+=1!} rc1(L[>],C):
                    rc2(L,M[>],C);
>> rc1*(L,M) : L ==[]?  M: rc2(L,M,M);
>> reduceCount*(L) : rc1(L ,[]);
>> ^reduceCount("Hello_World"@list);
[ [ 'H', 1 ], [ 'e', 1 ], [ 'l', 3 ], [ 'o', 2 ],
[ '_', 1 ], [ 'W', 1 ], [ 'r', 1 ], [ 'd', 1 ] ]
>> ^reduceCount(map(r16, lambda x:  true,
        lambda x:  x%2==0?  "even":  "odd"));
```

ReduceCount with Imperative Features

```
>> rc1*(_,_) :  null ;
>> rc2*(L,M,C): M==[]?rc1(L[ >],[[L [.],1]| C ]):
   L[.]==M [.][0]?{!M[0][1]+=1!} rc1(L[>],C):
                    rc2(L,M[>],C);
>> rc1*(L,M) : L ==[]?  M: rc2(L,M,M);
>> reduceCount*(L) : rc1(L ,[]);
>> ^reduceCount("Hello_World"@list);
[ [ 'H', 1 ], [ 'e', 1 ], [ 'l', 3 ], [ 'o', 2 ],
[ '_', 1 ], [ 'W', 1 ], [ 'r', 1 ], [ 'd', 1 ] ]
>> ^reduceCount(map(r16, lambda x:  true,
        lambda x:  x%2==0?  "even":  "odd"));
[ [ "even", 8 ], [ "odd", 8 ] ]
```

Definition of Json Objects

- Json objects (referred to simply as *json*) are represented as (possibly empty) sequences of fields separated by comma and enclosed into curly braces.
- A field is a pair (key, value) separated by a colon: *key* is a string and *value* can be of any type.
- Two fields of a json object must not have the same key.

Definition of Json Objects

- Json objects (referred to simply as *json*) are represented as (possibly empty) sequences of fields separated by comma and enclosed into curly braces.
- A field is a pair (key, value) separated by a colon: *key* is a string and *value* can be of any type.
- Two fields of a json object must not have the same key.

```
>> emps = [ { "name":  "e1", "age":  30 },
{"name":"e2","age":32,"proj":["p1","p2"],"b":10},
{"name":"e3", "age":28, "proj":["p1","p3"] } ];
```

Definition of Json Objects

- Json objects (referred to simply as *json*) are represented as (possibly empty) sequences of fields separated by comma and enclosed into curly braces.
- A field is a pair (key, value) separated by a colon: *key* is a string and *value* can be of any type.
- Two fields of a json object must not have the same key.

```
>> emps = [ { "name":  "e1", "age":  30 },
{"name":"e2","age":32,"proj":["p1","p2"],"b":10},
{"name":"e3", "age":28, "proj":["p1","p3"] } ];
>> ^emps[2]["proj"];
```

Definition of Json Objects

- Json objects (referred to simply as *json*) are represented as (possibly empty) sequences of fields separated by comma and enclosed into curly braces.
- A field is a pair (key, value) separated by a colon: *key* is a string and *value* can be of any type.
- Two fields of a json object must not have the same key.

```
>> emps = [ { "name":  "e1", "age":  30 },
{"name":"e2","age":32,"proj":["p1","p2"],"b":10},
{"name":"e3", "age":28, "proj":["p1","p3"] } ];
>> ^emps[2]["proj"];
[ "p1", "p3" ]
```

Definition of Json Objects

- Json objects (referred to simply as *json*) are represented as (possibly empty) sequences of fields separated by comma and enclosed into curly braces.
- A field is a pair (key, value) separated by a colon: *key* is a string and *value* can be of any type.
- Two fields of a json object must not have the same key.

```
>> emps = [ { "name":  "e1", "age":  30 },
{"name":"e2","age":32,"proj":["p1","p2"],"b":10},
{"name":"e3", "age":28, "proj":["p1","p3"] } ];
>> ^emps[2]["proj"];
[ "p1", "p3" ]
>> ^emps[0]["proj"];
```

Definition of Json Objects

- Json objects (referred to simply as *json*) are represented as (possibly empty) sequences of fields separated by comma and enclosed into curly braces.
- A field is a pair (key, value) separated by a colon: *key* is a string and *value* can be of any type.
- Two fields of a json object must not have the same key.

```
>> emps = [ { "name":  "e1", "age":  30 },
{"name":"e2","age":32,"proj":["p1","p2"],"b":10},
{"name":"e3", "age":28, "proj":["p1","p3"] } ];
>> ^emps[2]["proj"];
[ "p1", "p3" ]
>> ^emps[0]["proj"];
null
```

MapReducing a List of Jsons

MapReducing a List of Jsons

```
>> jsFilter(LJ,filtC/3,K,V): LJ==[]?  []:
          filtC(LJ[.],K,V)?
          [LJ[.]|jsFilter(LJ[>],filtC,K,V)]:
          jsFilter(LJ[>],filtC,K,V);
```

MapReducing a List of Jsons

```
>> jsFilter(LJ,filtC/3,K,V): LJ==[]?  []:
          filtC(LJ[.],K,V)?
          [LJ[.]|jsFilter(LJ[>],filtC,K,V)]:
          jsFilter(LJ[>],filtC,K,V);
>> selVeqK(J,K,V): J[K]!=null && J[K]==V;
```

MapReducing a List of Jsons

```
>> jsFilter(LJ,filtC/3,K,V): LJ==[]?  []:
          filtC(LJ[.],K,V)?
          [LJ[.]|jsFilter(LJ[>],filtC,K,V)]:
          jsFilter(LJ[>],filtC,K,V);
>> selVeqK(J,K,V): J[K]!=null && J[K]==V;
>> ^jsFilter(emps,selVeqK,"age",28);
```

MapReducing a List of Jsons

```
>> jsFilter(LJ,filtC/3,K,V): LJ==[]?  []:
          filtC(LJ[.],K,V)?
          [LJ[.]|jsFilter(LJ[>],filtC,K,V)]:
          jsFilter(LJ[>],filtC,K,V);
>> selVeqK(J,K,V): J[K]!=null && J[K]==V;
>> ^jsFilter(emps,selVeqK,"age",28);
[ "name":"e3", "age":28, "proj":["p1","p3"]  ]
```

MapReducing a List of Jsons

```
>> jsFilter(LJ,filtC/3,K,V): LJ==[]?  []:
          filtC(LJ[.],K,V)?
          [LJ[.]|jsFilter(LJ[>],filtC,K,V)]:
          jsFilter(LJ[>],filtC,K,V);
>> selVeqK(J,K,V): J[K]!=null && J[K]==V;
>> ^jsFilter(emps,selVeqK,"age",28);
[ "name":"e3", "age":28, "proj":["p1","p3"]  ]
>> selVinK(J,K,V) : J[K]!=null && member(V,J[K]);
```

MapReducing a List of Jsons

```
>> jsFilter(LJ,filtC/3,K,V): LJ==[]?  []:
          filtC(LJ[.],K,V)?
          [LJ[.]|jsFilter(LJ[>],filtC,K,V)]:
          jsFilter(LJ[>],filtC,K,V);
>> selVeqK(J,K,V): J[K]!=null && J[K]==V;
>> ^jsFilter(emps,selVeqK,"age",28);
[ "name":"e3", "age":28, "proj":["p1","p3"]  ]
>> selVinK(J,K,V) : J[K]!=null && member(V,J[K]);
>> ^jsFilter(emps,selVinK,"proj","p1");
```

MapReducing a List of Jsons

```
>> jsFilter(LJ,filtC/3,K,V): LJ==[]?  []:
          filtC(LJ[.],K,V)?
          [LJ[.]|jsFilter(LJ[>],filtC,K,V)]:
          jsFilter(LJ[>],filtC,K,V);
>> selVeqK(J,K,V): J[K]!=null && J[K]==V;
>> ^jsFilter(emps,selVeqK,"age",28);
[ "name":"e3", "age":28, "proj":["p1","p3"]  ]
>> selVinK(J,K,V) : J[K]!=null && member(V,J[K]);
>> ^jsFilter(emps,selVinK,"proj","p1");
[ {"n":"e2","a":32,"p":["p1","p2"],"b":10},
  {"name" : "e3","age" : 28,"proj" : ["p1","p3"]} ]
```

ReduceCount on a List of Jsons

```
>> mapProj(E): E==[]?  []:  E[.]["proj"] != null?
        E[.]["proj"][:]+mapProj(E[>]):
        mapProj(E[>]);
```

ReduceCount on a List of Jsons

```
>> mapProj(E): E==[]?  []:  E[.]["proj"] != null?
        E[.]["proj"][:]+mapProj(E[>]):
        mapProj(E[>]);
>> ^reduceCount(mapProj(emps));
```

ReduceCount on a List of Jsons

```
>> mapProj(E): E==[]?  []:  E[.]["proj"] != null?
        E[.]["proj"][:]+mapProj(E[>]):
        mapProj(E[>]);
>> ^reduceCount(mapProj(emps));
[ [ "p1", 2 ], [ "p2", 1 ], [ "p3", 1 ] ]
```

Updating Json Lists

- Add a 100$ bonus to employees working for project `"p1"`.

Updating Json Lists

- Add a 100$ bonus to employees working for project "p1".

```
>> aBonus*(E,v):E==[]?true:E[.]["bonus"]==null?
       {!  E[0]["bonus"]=v !} aBonus(E[>],v):
       {!  E[0]["bonus"]+=v !} aBonus(E[>],v);
```

Updating Json Lists

- Add a 100$ bonus to employees working for project "p1".

```
>> aBonus*(E,v):E==[]?true:E[.]["bonus"]==null?
       {!  E[0]["bonus"]=v !} aBonus(E[>],v):
       {!  E[0]["bonus"]+=v !} aBonus(E[>],v);
>> ^aBonus(jsFilter(emps,selVinK,"proj","p1"),100);
```

Updating Json Lists

- Add a 100$ bonus to employees working for project "p1".

```
>> aBonus*(E,v):E==[]?true:E[.]["bonus"]==null?
      {!  E[0]["bonus"]=v !} aBonus(E[>],v):
      {!  E[0]["bonus"]+=v !} aBonus(E[>],v);
>> ^aBonus(jsFilter(emps,selVinK,"proj","p1"),100);
true
```

Updating Json Lists

- Add a 100$ bonus to employees working for project "p1".

```
>> aBonus*(E,v):E==[]?true:E[.]["bonus"]==null?
       {!  E[0]["bonus"]=v !} aBonus(E[>],v):
       {!  E[0]["bonus"]+=v !} aBonus(E[>],v);
>> ^aBonus(jsFilter(emps,selVinK,"proj","p1"),100);
true
>> ^emps;
```

Updating Json Lists

- Add a 100$ bonus to employees working for project "p1".

```
>> aBonus*(E,v):E==[]?true:E[.]["bonus"]==null?
       {!  E[0]["bonus"]=v !} aBonus(E[>],v):
       {!  E[0]["bonus"]+=v !} aBonus(E[>],v);
>> ^aBonus(jsFilter(emps,selVinK,"proj","p1"),100);
true
>> ^emps;
[ { "name":  "e1", "age":  30 },
  { "name":  "e2", "age":  32,
    "proj":  ["p1","p2"], "bonus":  110 },
  { "name":  "e3", "age":  28,
    "proj":  ["p1","p3"], "bonus":  100 } ]
```

The Dataset on Nobel Prizes

- Using the json dataset "Laureate" describing nobel prizes published in a GitHup web page
- Each laureate json includes a number of properties for the laureate (code, name, gender, date of birth, date of death, born country, born city, died country, died city) and the list of prizes (in general 1 prize except for 5 laureates who were winners of two prizes and 1 laureate with 3 prizes)
- Each prize is described by the year (from 1901 to 2017), the category (Phisics, Chemistry, Medicine, Literature, Peace, Economy), the share (from 1 to 4), the motivation and the list of affiliations for the laureates, typically one, which were determinant in letting her/him achieve the prize.

Loading Laureates

Loading Laureates

```
>> nobel_laureates =<< ("./CL_laureates.dat");
```

Loading Laureates

```
>> nobel_laureates =<< ("./CL_laureates.dat");
- data are correct
```

Loading Laureates

```
>> nobel_laureates =<< ("./CL_laureates.dat");
- data are correct
>> laureates=nobel_laureates["laureates"];
```

Loading Laureates

```
>> nobel_laureates =<< ("./CL_laureates.dat");
- data are correct
>> laureates=nobel_laureates["laureates"];
>> ^laureates[5] %*;
```

Loading Laureates

```
>> nobel_laureates =<< ("./CL_laureates.dat");
- data are correct
>> laureates=nobel_laureates["laureates"];
>> ^laureates[5] %*;
{"id":"6", "firstname":"Marie",
"surname":"Curie", "born": "1867-11-07", "died":
"1934-07-04", "bornCountry": "Poland",
"bornCountryCode": "PL", "bornCity": "Warsaw",
"diedCountry": "France", "diedCountryCode":
"FR", "diedCity": "Sallanches",
"gender": "female",
```

Loading Laureates – continued


```
"prizes": [
    { "year": "1903", "category":"physics", "share":4,
      "motivation":"..researches on radiation", "affiliations": []
    },
    { "year":"1911", "category":"chemistry", "share":1,
      "motivation":"..discovery of..radium and polonium",
      "affiliations": [ {"name":"Sorbonne University",
        "city":"Paris", "country":"France" } ]
   }
  ]
}
```

Loading Laureates – continued

```
"prizes": [
    { "year": "1903", "category":"physics", "share":4,
      "motivation":"..researches on radiation", "affiliations": []
    },
    { "year":"1911", "category":"chemistry", "share":1,
      "motivation":"..discovery of..radium and polonium",
      "affiliations": [ {"name":"Sorbonne University",
        "city":"Paris", "country":"France" } ]
   }
  ]
}
>> ^_len(laureates);
```

Loading Laureates – continued

```
"prizes": [
    { "year": "1903", "category":"physics", "share":4,
      "motivation":"..researches on radiation", "affiliations": []
    },
    { "year":"1911", "category":"chemistry", "share":1,
      "motivation":"..discovery of..radium and polonium",
      "affiliations": [ {"name":"Sorbonne University",
        "city":"Paris", "country":"France" } ]
   }
  ]
}
>> ^_len(laureates);
916
```

Restructuring Laureates

- The collection `Laureates` in restructured as: (1) reduce the number of fields by removing the ones that are not relevant for the queries and (2) flatten the structure so that data on laureates with multiple prizes are replicated for each prize.
- To give an idea of the restructuring, the original json about Marie Curie is restructured into two jsons as follows: .

Restructuring Laureates

- The collection `Laureates` in restructured as: (1) reduce the number of fields by removing the ones that are not relevant for the queries and (2) flatten the structure so that data on laureates with multiple prizes are replicated for each prize.
- To give an idea of the restructuring, the original json about Marie Curie is restructured into two jsons as follows: .

```
{ "id": "6", "bornCountryCode": "PL", "gender":
"female", "year": "1903", "category":
"physics", "share": "4" }
```

Restructuring Laureates

- The collection `Laureates` in restructured as: (1) reduce the number of fields by removing the ones that are not relevant for the queries and (2) flatten the structure so that data on laureates with multiple prizes are replicated for each prize.
- To give an idea of the restructuring, the original json about Marie Curie is restructured into two jsons as follows: .

```
{ "id": "6", "bornCountryCode": "PL", "gender":
"female", "year": "1903", "category":
"physics", "share": "4" }
{ "id": "6", "bornCountryCode": "PL", "gender":
"female", "year": "1911", "category":
"chemistry", "share": "1" }
```

1: Counting EU laureates and with Science Prizes

1: Counting EU laureates and with Science Prizes

```
>> EU_Codes = [ "AT", ..., "UK" ];
```

1: Counting EU laureates and with Science Prizes

```
>> EU_Codes = [ "AT", ..., "UK" ];
>> Sciences= ["physics","chemistry","medicine"];
```

1: Counting EU laureates and with Science Prizes

```
>> EU_Codes = [ "AT", ..., "UK" ];
>> Sciences= ["physics","chemistry","medicine"];
>> filterCount(L,f/2,Q): L==[]?  0:
                f(L[.],Q)? 1+filterCount(L[>],f,Q):
                filterCount(L[>],f,Q);
```

1: Counting EU laureates and with Science Prizes

```
>> EU_Codes = [ "AT", ..., "UK" ];
>> Sciences= ["physics","chemistry","medicine"];
>> filterCount(L,f/2,Q): L==[]?  0:
            f(L[.],Q)? 1+filterCount(L[>],f,Q):
            filterCount(L[>],f,Q);
>> lFilter1(L1,k,vL) : vL!=[] &&
     ( L1[k]==vL[.]|| lFilter1 (L1,k,vL[>]));
```

1: Counting EU laureates and with Science Prizes

```
>> EU_Codes = [ "AT", ..., "UK" ];
>> Sciences= ["physics","chemistry","medicine"];
>> filterCount(L,f/2,Q): L==[]?  0:
            f(L[.],Q)? 1+filterCount(L[>],f,Q):
            filterCount(L[>],f,Q);
>> lFilter1(L1,k,vL) : vL!=[] &&
     ( L1[k]==vL[.]|| lFilter1 (L1,k,vL[>]));
>> ^filterCount(elaureates,laureateFilter,
            [ ["bornCC", EU_Codes] ]);
```

1: Counting EU laureates and with Science Prizes

```
>> EU_Codes = [ "AT", ..., "UK" ];
>> Sciences= ["physics","chemistry","medicine"];
>> filterCount(L,f/2,Q): L==[]?  0:
          f(L[.],Q)? 1+filterCount(L[>],f,Q):
          filterCount(L[>],f,Q);
>> lFilter1(L1,k,vL) : vL!=[] &&
     ( L1[k]==vL[.]|| lFilter1 (L1,k,vL[>]));
>> ^filterCount(elaureates,laureateFilter,
          [ ["bornCC", EU_Codes] ]);
445
```

1: Counting EU laureates and with Science Prizes

```
>> EU_Codes = [ "AT", ..., "UK" ];
>> Sciences= ["physics","chemistry","medicine"];
>> filterCount(L,f/2,Q): L==[]?  0:
          f(L[.],Q)? 1+filterCount(L[>],f,Q):
          filterCount(L[>],f,Q);
>> lFilter1(L1,k,vL) : vL!=[] &&
     ( L1[k]==vL[.]|| lFilter1 (L1,k,vL[>]));
>> ^filterCount(elaureates,laureateFilter,
          [ ["bornCC", EU_Codes] ]);
445
>> ^filterCount(elaureates,laureateFilter,
  [ ["bornCC",EU_Codes],["category",Sciences] ]);
```

1: Counting EU laureates and with Science Prizes

```
>> EU_Codes = [ "AT", ..., "UK" ];
>> Sciences= ["physics","chemistry","medicine"];
>> filterCount(L,f/2,Q): L==[]?  0:
          f(L[.],Q)? 1+filterCount(L[>],f,Q):
          filterCount(L[>],f,Q);
>> lFilter1(L1,k,vL) : vL!=[] &&
     ( L1[k]==vL[.]|| lFilter1 (L1,k,vL[>]));
>> ^filterCount(elaureates,laureateFilter,
          [ ["bornCC", EU_Codes] ]);
445
>> ^filterCount(elaureates,laureateFilter,
  [ ["bornCC",EU_Codes],["category",Sciences] ]);
302
```

2: Filtering laureates with Range Conditions

2: Filtering laureates with Range Conditions

```
>> rangeFilter(L1,R) : R==[] ||
   R[.][1] <= L1[R[.][0]]&&L1[R[.][0]] <= R[.][2]
   && rangeFilter(L1,R[>]);
```

2: Filtering laureates with Range Conditions

```
>> rangeFilter(L1,R) : R==[] ||
   R[.][1] <= L1[R[.][0]]&&L1[R[.][0]] <= R[.][2]
   && rangeFilter(L1,R[>]);
>> laureateFilterR(L1,Q): laureateFilter(L1,Q[0])
   && rangeFilter(L1,Q[1]);
```

2: Filtering laureates with Range Conditions

```
>> rangeFilter(L1,R) : R==[] ||
   R[.][1] <= L1[R[.][0]]&&L1[R[.][0]] <= R[.][2]
   && rangeFilter(L1,R[>]);
>> laureateFilterR(L1,Q): laureateFilter(L1,Q[0])
   && rangeFilter(L1,Q[1]);
>> ^filterCount(elaureates, laureateFilterR, [
   [["bornCC",EU_Codes],["category",Sciences]],
   [["year","2000","2017"]] ]);
```

2: Filtering laureates with Range Conditions

```
>> rangeFilter(L1,R) : R==[] ||
   R[.][1] <= L1[R[.][0]]&&L1[R[.][0]] <= R[.][2]
   && rangeFilter(L1,R[>]);
>> laureateFilterR(L1,Q): laureateFilter(L1,Q[0])
   && rangeFilter(L1,Q[1]);
>> ^filterCount(elaureates, laureateFilterR, [
   [["bornCC",EU_Codes],["category",Sciences]],
   [["year","2000","2017"]] ]);
46
```

2: Filtering laureates with Range Conditions

```
>> rangeFilter(L1,R) : R==[] ||
   R[.][1] <= L1[R[.][0]]&&L1[R[.][0]] <= R[.][2]
   && rangeFilter(L1,R[>]);
>> laureateFilterR(L1,Q): laureateFilter(L1,Q[0])
   && rangeFilter(L1,Q[1]);
>> ^filterCount(elaureates, laureateFilterR, [
   [["bornCC",EU_Codes],["category",Sciences]],
   [["year","2000","2017"]] ]);
46
>> ^filterCount(elaureates, laureateFilterR, [
   [ ["bornCC",EU_Codes],["category",Sciences] ],
   [ ["year","2000","2017"],["age",0,60] ] ]);
```

2: Filtering laureates with Range Conditions

```
>> rangeFilter(L1,R) : R==[] ||
   R[.][1] <= L1[R[.][0]]&&L1[R[.][0]] <= R[.][2]
   && rangeFilter(L1,R[>]);
>> laureateFilterR(L1,Q): laureateFilter(L1,Q[0])
   && rangeFilter(L1,Q[1]);
>> ^filterCount(elaureates, laureateFilterR, [
   [["bornCC",EU_Codes],["category",Sciences]],
   [["year","2000","2017"]] ]);
46
>> ^filterCount(elaureates, laureateFilterR, [
   [ ["bornCC",EU_Codes],["category",Sciences] ],
   [ ["year","2000","2017"],["age",0,60] ] ]);
9
```

3: MapReduce to Aggregate by Gender

3: MapReduce to Aggregate by Gender

```
>> laureateMap(L, f/2, Q, m/1) : L == []?[] : f(L[.], Q)?
                [m(L[.]) | laureateMap(L[>], f, Q, m)] :
                 laureateMap(L[>], f, Q, m);
```

3: MapReduce to Aggregate by Gender

```
>> laureateMap(L, f/2, Q, m/1) : L == []?[] : f(L[.], Q)?
                [m(L[.]) | laureateMap(L[>], f, Q, m)] :
                 laureateMap(L[>], f, Q, m);
>> mapG(L1):  L1["gender"];
```

3: MapReduce to Aggregate by Gender

```
>> laureateMap(L,f/2,Q,m/1):L==[]?[]:f(L[.],Q)?
        [m(L[.])|laureateMap(L[>],f,Q,m)]:
         laureateMap(L[>],f,Q,m);
>> mapG(L1):  L1["gender"];
>> ^reduceCount(laureateMap(elaureates,lFilterR,
    [[["bornCC",EU_Codes],["category",Sciences]],
    [["year","2000","2017"]]],mapG));
```

3: MapReduce to Aggregate by Gender

```
>> laureateMap(L, f/2, Q, m/1) : L == []?[] : f(L[.], Q)?
            [m(L[.]) | laureateMap(L[>], f, Q, m)] :
             laureateMap(L[>], f, Q, m);
>> mapG(L1):  L1["gender"];
>> ^reduceCount(laureateMap(elaureates, lFilterR,
     [[["bornCC", EU_Codes], ["category", Sciences]],
     [["year","2000","2017"]]], mapG));
[ ["male", 44], ["female", 2] ]
```

3: MapReduce to Aggregate by Gender

```
>> laureateMap(L,f/2,Q,m/1) : L == []?[] : f(L[.],Q)?
            [m(L[.]) | laureateMap(L[>],f,Q,m)] :
             laureateMap(L[>],f,Q,m);
>> mapG(L1):  L1["gender"];
>> ^reduceCount(laureateMap(elaureates,lFilterR,
     [[["bornCC",EU_Codes],["category",Sciences]],
     [["year","2000","2017"]]],mapG));
[ ["male", 44], ["female", 2] ]
>> ^reduceCount(laureateMap(elaureates,lFilterR,
     [[["bornCC",EU_Codes],["category",Sciences]],
     []],mapG));
```

3: MapReduce to Aggregate by Gender

```
>> laureateMap(L,f/2,Q,m/1) : L == []?[] : f(L[.],Q)?
          [m(L[.]) | laureateMap(L[>],f,Q,m)] :
           laureateMap(L[>],f,Q,m);
>> mapG(L1):  L1["gender"];
>> ^reduceCount(laureateMap(elaureates,lFilterR,
     [[["bornCC",EU_Codes],["category",Sciences]],
     [["year","2000","2017"]]],mapG));
[ ["male", 44], ["female", 2] ]
>> ^reduceCount(laureateMap(elaureates,lFilterR,
     [[["bornCC",EU_Codes],["category",Sciences]],
     []],mapG));
[ ["male", 293], ["female", 9] ]
```

3: MapReduce to Aggregate by Gender

```
>> laureateMap(L,f/2,Q,m/1):L==[]?[]:f(L[.],Q)?
          [m(L[.])|laureateMap(L[>],f,Q,m)]:
           laureateMap(L[>],f,Q,m);
>> mapG(L1):  L1["gender"];
>> ^reduceCount(laureateMap(elaureates,lFilterR,
     [["bornCC",EU_Codes],["category",Sciences]],
     [["year","2000","2017"]]],mapG));
[ ["male", 44], ["female", 2] ]
>> ^reduceCount(laureateMap(elaureates,lFilterR,
     [["bornCC",EU_Codes],["category",Sciences]],
     []],mapG));
[ ["male", 293], ["female", 9] ]
>> ^reduceCount(laureateMap(elaureates,lFilterR,
     [["bornCC",EU_Codes]],[]],mapG));
```

3: MapReduce to Aggregate by Gender

```
>> laureateMap(L,f/2,Q,m/1) : L == []?[] : f(L[.],Q)?
          [m(L[.]) | laureateMap(L[>],f,Q,m)] :
           laureateMap(L[>],f,Q,m);
>> mapG(L1):  L1["gender"];
>> ^reduceCount(laureateMap(elaureates,lFilterR,
     [[["bornCC",EU_Codes],["category",Sciences]],
     [["year","2000","2017"]]],mapG));
[ ["male", 44], ["female", 2] ]
>> ^reduceCount(laureateMap(elaureates,lFilterR,
     [[["bornCC",EU_Codes],["category",Sciences]],
     []],mapG));
[ ["male", 293], ["female", 9] ]
>> ^reduceCount(laureateMap(elaureates,lFilterR,
     [[["bornCC",EU_Codes]],[]],mapG));
[ ["male", 425], ["female", 20] ]
```

4: MapReduce to Compute Average Age

4: MapReduce to Compute Average Age

```
>> redAvg_1(L): <CS> L==[]?  [0,0]:
     {! CS = red_1(L[>]) !}[CS[0] + 1, CS[1] + L[.]];
```

4: MapReduce to Compute Average Age

```
>> redAvg_1(L): <CS> L==[]?  [0,0]:
     {! CS = red_1(L[>]) !}[CS[0] + 1, CS[1] + L[.]];
>> reduceAvg(L) : <CS>{! CS = redAvg_1(L) !} CS[1]/CS[0];
```

4: MapReduce to Compute Average Age

```
>> redAvg_1(L): <CS> L==[]?  [0,0]:
     {!CS = red_1(L[>])!}[CS[0]+1, CS[1]+L[.]];
>> reduceAvg(L) : <CS>{!CS = redAvg_1(L)!}CS[1]/CS[0];
>> mapA(L1):  L1["age"];
```

4: MapReduce to Compute Average Age

```
>> redAvg_1(L): <CS> L==[]?  [0,0]:
    {!CS = red_1(L[>])!}[CS[0]+1, CS[1]+L[.]];
>> reduceAvg(L) : <CS>{!CS = redAvg_1(L)!}CS[1]/CS[0];
>> mapA(L1):  L1["age"];
>> ^reduceAvg(laureateMap(elaureates, lFilterR,
      [[["bornCC", EU_Codes], ["category", Sciences]],
      [["year", "2000", "2017"]]], mapA));
```

4: MapReduce to Compute Average Age

```
>> redAvg_1(L): <CS> L==[]?  [0,0]:
    {!CS = red_1(L[>])!}[CS[0]+1, CS[1]+L[.]];
>> reduceAvg(L) : <CS>{!CS = redAvg_1(L)!}CS[1]/CS[0];
>> mapA(L1):  L1["age"];
>> ^reduceAvg(laureateMap(elaureates, lFilterR,
      [[["bornCC",EU_Codes],["category",Sciences]],
      [["year","2000","2017"]]],mapA));
69.54347826086956
```

4: MapReduce to Compute Average Age

```
>> redAvg_1(L): <CS> L==[]?  [0,0]:
    {!CS = red_1(L[>])!}[CS[0]+1, CS[1]+L[.]];
>> reduceAvg(L) : <CS>{!CS = redAvg_1(L)!}CS[1]/CS[0];
>> mapA(L1):  L1["age"];
>> ^reduceAvg(laureateMap(elaureates, lFilterR,
      [[["bornCC", EU_Codes], ["category", Sciences]],
      [["year", "2000", "2017"]]], mapA));
69.54347826086956
>> ^reduceAvg(laureateMap(elaureates, lFilterR,
      [[["bornCC", EU_Codes], ["category", Sciences]],
      []], mapA));
```

4: MapReduce to Compute Average Age

```
>> redAvg_1(L): <CS> L==[]?  [0,0]:
    {!CS = red_1(L[>])!}[CS[0]+1, CS[1]+L[.]];
>> reduceAvg(L) : <CS>{!CS = redAvg_1(L)!}CS[1]/CS[0];
>> mapA(L1):  L1["age"];
>> ^reduceAvg(laureateMap(elaureates, lFilterR,
      [[["bornCC", EU_Codes], ["category", Sciences]],
      [["year", "2000", "2017"]]], mapA));
69.54347826086956
>> ^reduceAvg(laureateMap(elaureates, lFilterR,
      [[["bornCC", EU_Codes], ["category", Sciences]],
      []], mapA));
56.63245033112583
```

4: MapReduce to Compute Average Age

```
>> redAvg_1(L): <CS> L==[]?  [0,0]:
    {!CS = red_1(L[>])!}[CS[0]+1, CS[1]+L[.]];
>> reduceAvg(L) : <CS>{!CS = redAvg_1(L)!}CS[1]/CS[0];
>> mapA(L1):  L1["age"];
>> ^reduceAvg(laureateMap(elaureates, lFilterR,
      [[["bornCC", EU_Codes], ["category", Sciences]],
      [["year", "2000", "2017"]]], mapA));
69.54347826086956
>> ^reduceAvg(laureateMap(elaureates, lFilterR,
      [[["bornCC", EU_Codes], ["category", Sciences]],
      []], mapA));
56.63245033112583
>> ^reduceAvg(laureateMap(elaureates, lFilterR,
      [[["bornCC", EU_Codes]], []], mapA));
```

4: MapReduce to Compute Average Age

```
>> redAvg_1(L): <CS> L==[]?  [0,0]:
    {!CS = red_1(L[>])!}[CS[0]+1, CS[1]+L[.]];
>> reduceAvg(L) : <CS>{!CS = redAvg_1(L)!}CS[1]/CS[0];
>> mapA(L1):  L1["age"];
>> ^reduceAvg(laureateMap(elaureates, lFilterR,
      [[["bornCC", EU_Codes], ["category", Sciences]],
      [["year", "2000", "2017"]]], mapA));
69.54347826086956
>> ^reduceAvg(laureateMap(elaureates, lFilterR,
      [[["bornCC", EU_Codes], ["category", Sciences]],
      []], mapA));
56.63245033112583
>> ^reduceAvg(laureateMap(elaureates, lFilterR,
      [[["bornCC", EU_Codes]], []], mapA));
59.6561797752809
```

Conclusion: Wake-Up, it's almost over!

- We have presented CalcuList, a new educational functional programming language extended with imperative programming features, which are enabled under explicit request by the programmer.

Conclusion: Wake-Up, it's almost over!

- We have presented CalcuList, a new educational functional programming language extended with imperative programming features, which are enabled under explicit request by the programmer.
- CalcuList expressions and functions are first compiled and then executed each time a query is issued. The object code produced by a compilation is a program that will be eventually executed by the CalcuList Virtual Machine (CLVM). There is also an assembler component to run CLVM programs using an assembler language.

Conclusion: Wake-Up, it's almost over!

- We have presented CalcuList, a new educational functional programming language extended with imperative programming features, which are enabled under explicit request by the programmer.
- CalcuList expressions and functions are first compiled and then executed each time a query is issued. The object code produced by a compilation is a program that will be eventually executed by the CalcuList Virtual Machine (CLVM). There is also an assembler component to run CLVM programs using an assembler language.
- The CalcuList programming environment has been implemented as a small-sized Java project in Eclipse 4.4.1 with 6 packages and 20 classes all together. The size of the Jar File is rather small: 134 kb.

Future Work

- Provide an interface between CalcuList and documentary datastores, such as MongoDB and CouchDB, so that the language may be used as a powerful query system for teaching advanced query algorithms for such document datastores.

Future Work

- Provide an interface between CalcuList and documentary datastores, such as MongoDB and CouchDB, so that the language may be used as a powerful query system for teaching advanced query algorithms for such document datastores.
- Reinforce static type checking to avoid too many protests from functional programming purists.