S-expressions

Ken Q Pu, Faculty of Science, Ontario Tech University

S-expressions: a formal syntax from Lambda Calculus

Lambda Calculus inspired language design:

1. Data representation format
    a. S-expression
    b. Extended Data Notation (EDN)
2. Programming languages based on s-expression and EDN

Ken Q Pu, Faculty of Science, Ontario Tech University

S-expression

Atoms:

- Name, aka symbol

  E.g. first-name, last-name, age, x, y, z,

  Even function names are symbols:

  +, -, *, /, ...

- Constant: number, string, character

  E.g. 3.1415, 42, "Hello world", \a

Lists:

- Empty list: (), nil
- Parenthesis list: (...)

  E.g.

  (+ 2 3)

  (+ 1 2 (* 3 4) "hello world")

  (((1) (1 2)))

From (extended) LC to S-expressions: a first look

Extended LC:

- Multi-arity in function declaration
- Symbol binding of names to expressions

We will be using Clojure, which has an extended version of S-expression with better readability.

Lambda Calculus	**S-expression**
λ	fn
λx.y.z <*expr*>	(fn (x y z) <*expr*>)
<*name*> = <*expr*>	(def <*name*> <*expr*>)
<*func*> <*arg1*> <*arg2*> ...	(<*func*> <*arg1*> <*arg2*> ...)

Ken Q Pu, Faculty of Science, Ontario Tech University

S-expressions: data representation vs functional programs

S-expressions can be describe:

1. Data
2. Program

Any well-formed S-expression is a data value.

However, only some well-formed S-expressions are valid programs in Lisp.

Data s-expressions

They are not valid programs.

```
((first-name "Ken")
 (last-name "Pu")
 (courses "CSCI 3055U" "CSCI 1000U"))

(123 "blah" () () (((123))))
```

Program s-expressions

```
(print "Hello world")

(def f +)
(f 1 2 3)
```

These s-expressions are also data representations.

Ken Q Pu, Faculty of Science, Ontario Tech University

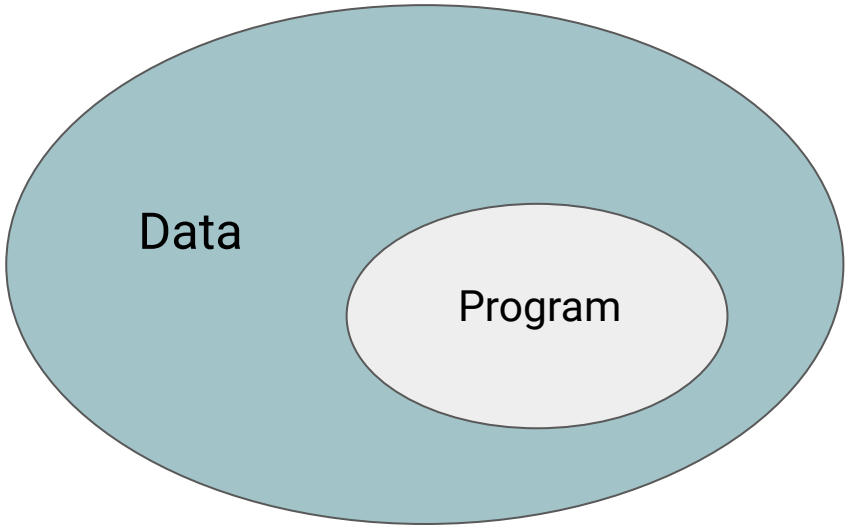
S-expressions: data representation vs functional programs

S-expressions can be describe:

1. Data
2. Program

Any well-formed S-expression is a data value.

However, only some well-formed S-expressions are valid programs in Lisp.



Ken Q Pu, Faculty of Science, Ontario Tech University

Extended Data Notation

A first look at Clojure's EDN

Ken Q Pu, Faculty of Science, Ontario Tech University

EDN: Extensible Data Notation

EDN is used to represent:

1. Data structures
2. Clojure programs

Why?

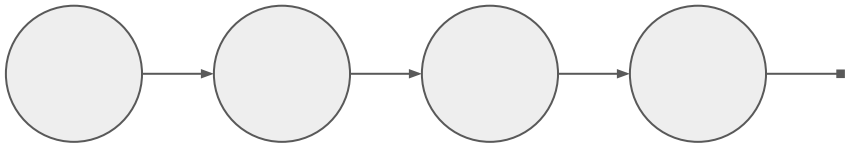
- EDN can describe data structures efficiently. It goes beyond lists.
- EDN uses more than parenthesis to improve readability.

Features in EDN:

- Vectors
- Dictionaries
- Keywords
- Comments

S-expressions are computationally inefficient

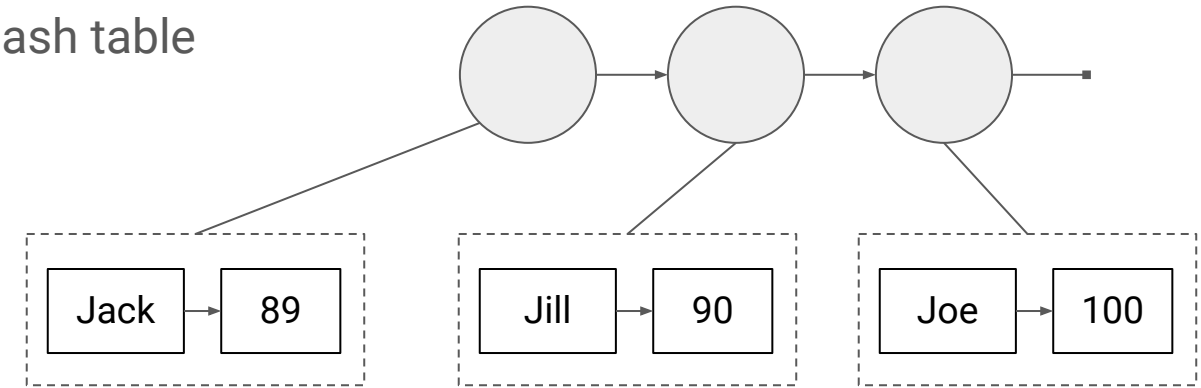
S-expressions can only lists. So classical Lisp implementations use *linked lists* as the underlying data structure to express s-expressions.



S-expressions are computationally inefficient

So we have to *simulate* other data structures, such as a hash table using nested lists:

```
(("Jack" 89)
 ("Jill" 90)
 ("Joe" 100))
```



How do we retrieve by key?

What's the grade of "Jill"?

Sequential scan: **O(n)**

But if we use hash table, we have **O(1)**. If we use balanced tree, we have **O(log n)**.

Going beyond lists

There is **no reason** to limit the data model to lists.

- Vectors: dynamic arrays that support random access by index
- Hash-map: dynamic key-value pairs that support fast key-based lookup

Vectors in EDN:

[...]

Hashmap in EDN:

{ key1 val1 key2 val2 ... }

Going beyond basic scalars

- Consider the difference between java.lang.String and Java enum?
- Python uses strings to represent symbolic constants like:
  if status == "busy": ...
  This is inefficient because string comparison is costly.
- Clojure introduces **keywords**, which are constants with efficient comparison: O(1)

```
:busy

:age

:csci-3055u
```

Examples

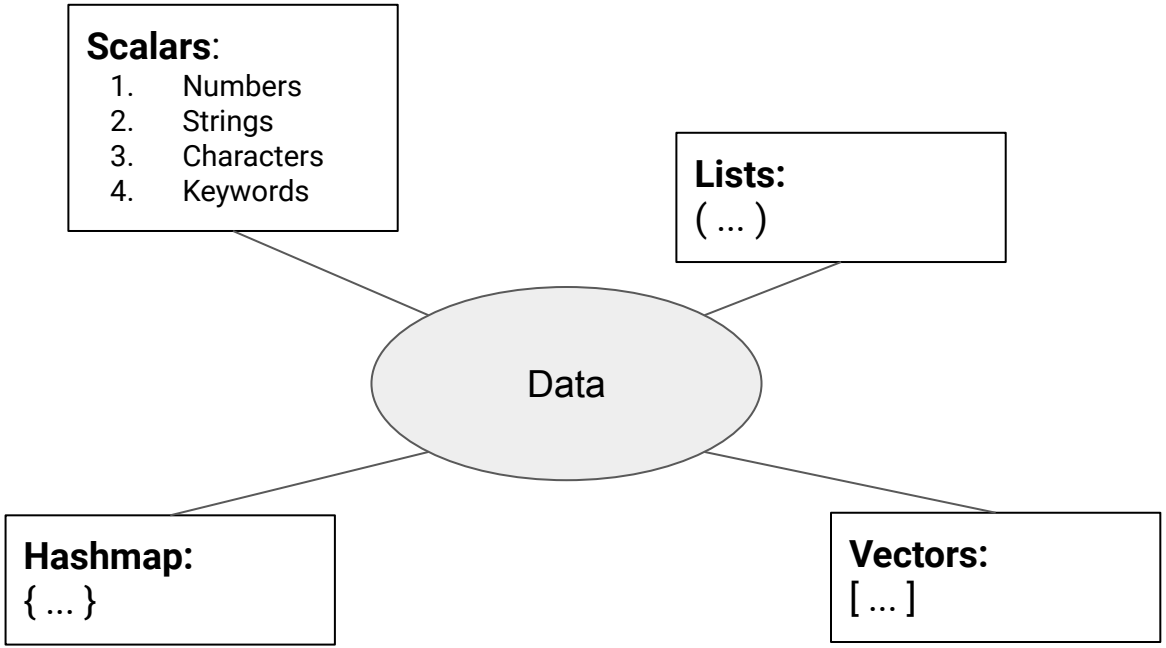
A class of students

```
(("Jack" ("CS" 89)
         ("Math" 76))
 ("Jill" ("CS" 90)
         ("Math" 80))
 ("Joe"  ("CS" 78)
         ("Math" 95)))
```

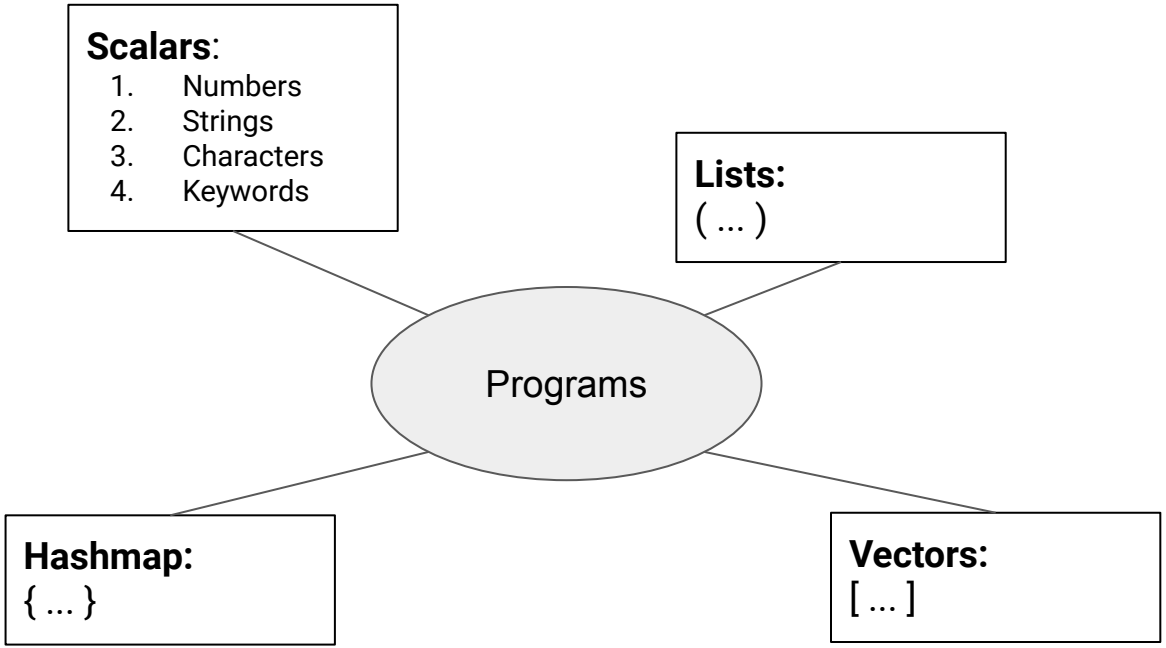
A class of students in EDN

```
[ {:name "Jack"
   :grades {:cs 89 :math 76}}
  {:name "Jill"
   :grades {:cs 90 :math 80}}
  {:name "Joe"
   :grades {:cs 78 :math 95}}
]
```

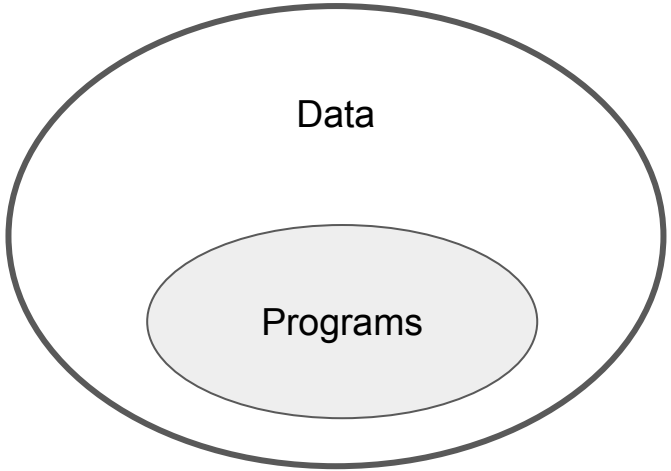
Working with EDN



Working with EDN



Working with EDN



Homoiconicity