Clojure

Ken Q Pu, Faculty of Science, Ontario Tech University

Quick remark

The line comments are supported by the semicolon ; character.

Forms

Ken Q Pu, Faculty of Science, Ontario Tech University

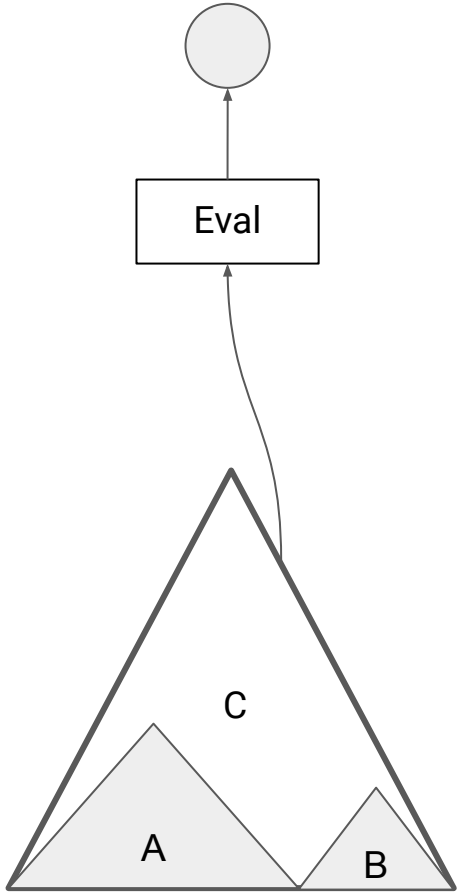
Forms

Form is a fragment of *valid* Clojure code.

- Literal representation of scalars are forms
- Function applications are forms
- Data structure constructions are forms
- Programming constructs as forms

It's important to remember that

- Formes **always** evaluate to data value
- Whitespaces and commas are **ignored** outside of string literals. This is the *tokenization rule* of Clojure.



Scalars

Ken Q Pu, Faculty of Science, Ontario Tech University

Scalar forms

Numbers	Strings	Characters	Keywords
42	"Hello world"	\H	:first-name
4.2		\a	:red
-42, -4.2		\"	:*red*
42M			
4.2M			
5/9			

Function Applications

Ken Q Pu, Faculty of Science, Ontario Tech University

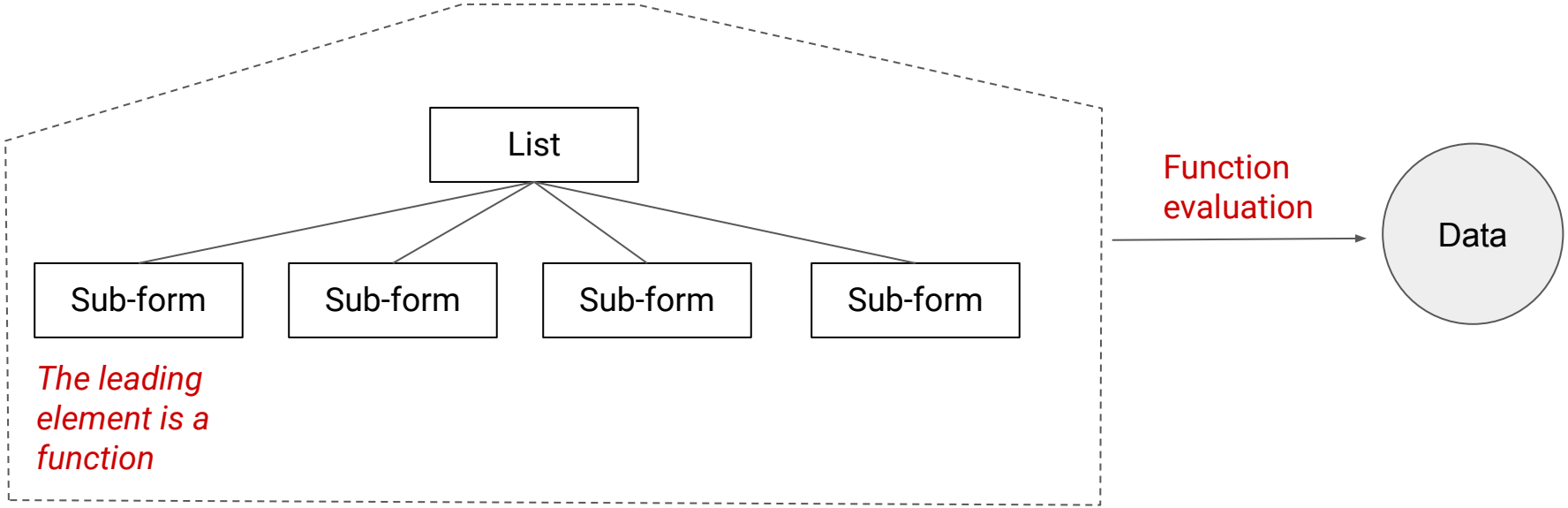
Function applications

Function applications are **lists**.

But not all lists are function applications.

```
( <function>  <argument 1>  <argument 2> ... )
```

Function applications



Examples

```
(+ 1 2 3)
⇒ 6
```

```
(str "Hello" "World")
⇒ "HelloWorld"
```

Remember the tokenization rule?

> *Whitespaces and commas are equivalent, and are both separators of tokens.*

The following are all equivalent

- `(+ 1,2,3)`
- `(  + ,,,, 1,,,, 2,,3)`
-
```
(+ 1
   2
   3)
```

Data Structure Literals

Ken Q Pu, Faculty of Science, Ontario Tech University

Data Structure Construction

There are several data structures that are *natively* supported by Clojure:

1. Vectors
2. Hashmaps
3. Sets
4. Lists
5. Functions

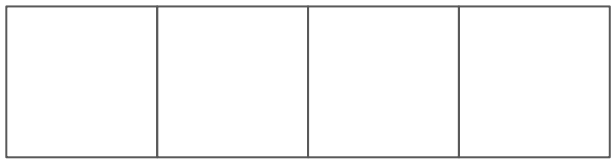
We can specify instances of these data structures using specialized syntax.

Vector

[<*elements*...>]

; a vector of heterogeneous elements with
; strings and numbers

```
["hello" 123 "world" 456]
```



Ken Q Pu, Faculty of Science, Ontario Tech University

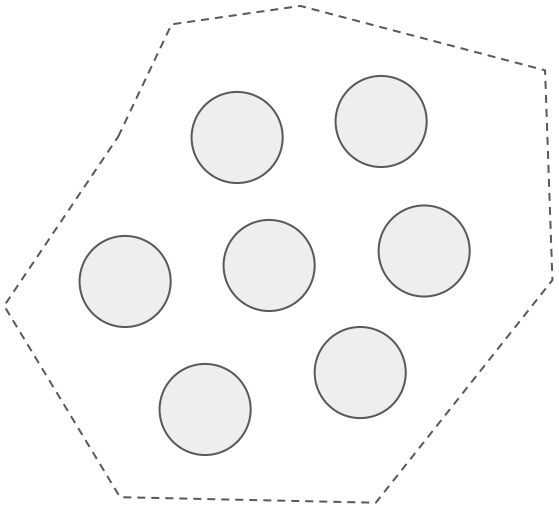
Hashmap

keys	values

A hashmap of keys and values

```
{ key1   val1
  key2   val2
  ... }
```

Sets

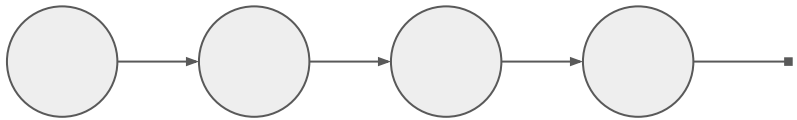


Elements must be unique in the set literal

```
#{ <elements ...> }
```

Ken Q Pu, Faculty of Science, Ontario Tech University

Lists



Lists are specified as

(*<elements...>*)

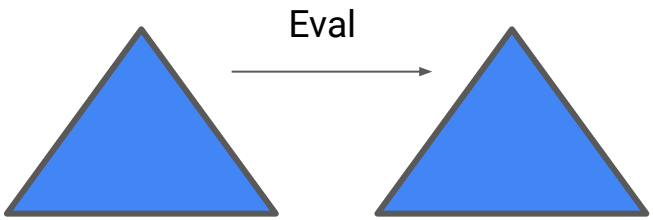
However, lists in source code are evaluated by the ***Eval***.

So to specify lists, we have to rely on **quoted forms**.

Quoted form is a form that **Eval** does not perform function evaluation.

Quoted Form

Quoted form is a form that **Eval** does not perform function evaluation.



A quoted form

There are several ways Clojure allows one to quote a form:

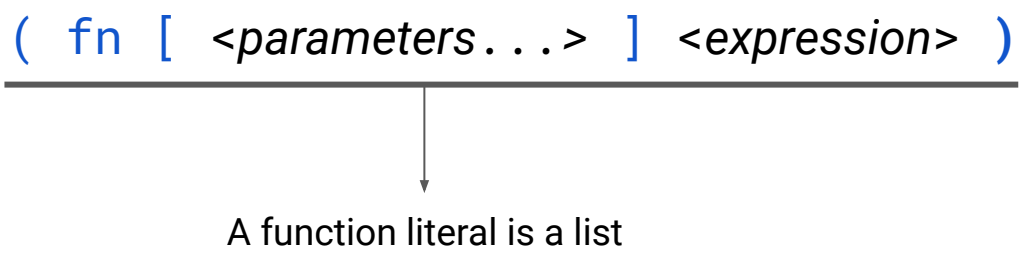
1. Use **quote** keyword.
2. Use single quote
3. Use backtick quote

```
(quote <form>)

'<form>

`<form>
```

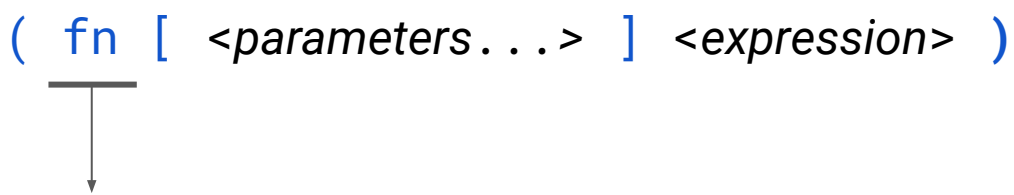
List literals

```
; a list of three numbers

'(1 2 3)
⇒ (1 2 3)

; quote is applied to sub-forms too

'(1 2 (3 4))
⇒ (1 2 (3 4))
```

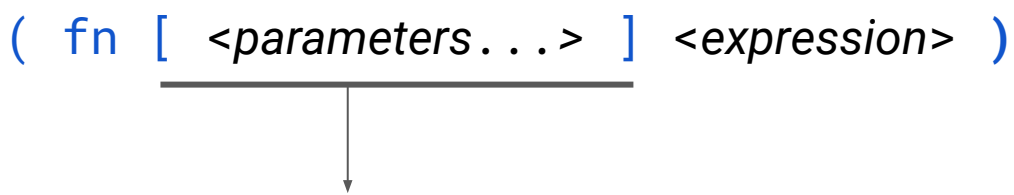
Function literals



Function literals

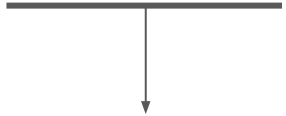


Function literals



Function literals

```
( fn [ <parameters...> ] <expression> )
```



The last element is the body of the function. The **Eval** will evaluate all function invocations using the body expression.

Example

```
(fn [r] (* Math/pi r r))
```

Area of a circle of a given radius.

```
((fn [r] (* Math/pi r r)) 10)
```


Beta-reduction

```
(* Math/PI 10 10)
```



```
314.1592653589793
```