Extensible Data Notation (EDN) is a format for serializing data.

The notation is used internally by Clojure to represent programs. It is also used as a data transfer format like JSON. Though it is more commonly used in Clojure, there are implementations of EDN for many other languages.

The main benefit of EDN over JSON and YAML is that it is extensible. We will see how it is extended later on.

```
; Comments start with a semicolon.
; Anything after the semicolon is ignored.

;;;;;;;;;;;;;;;;;;
;;; Basic Types ;;;
;;;;;;;;;;;;;;;;;;

nil         ; also known in other languages as null

; Booleans
true
false

; Strings are enclosed in double quotes
"hungarian breakfast"
"farmer's cheesy omelette"

; Characters are preceeded by backslashes
\g \r \a \c \e

; Keywords start with a colon. They behave like enums. Kind of
; like symbols in Ruby.
:eggs
:cheese
:olives

; Symbols are used to represent identifiers. They start with #.
; You can namespace symbols by using /. Whatever preceeds / is
; the namespace of the name.
#spoon
#kitchen/spoon ; not the same as #spoon
#kitchen/fork
#github/fork   ; you can't eat with this

; Integers and floats
42
3.14159

; Lists are sequences of values
(:bun :beef-patty 9 "yum!")

; Vectors allow random access
[:gelato 1 2 -2]

; Maps are associative data structures that associate the key with its value
{:eggs        2
```

```
 :lemon-juice 3.5
 :butter      1}

; You're not restricted to using keywords as keys
{[1 2 3 4] "tell the people what she wore",
 [5 6 7 8] "the more you see the more you hate"}

; You may use commas for readability. They are treated as whitespace.

; Sets are collections that contain unique elements.
#{:a :b 88 "huat"}

;;;;;;;;;;;;;;;;;;;;;
;;; Tagged Elements ;;;
;;;;;;;;;;;;;;;;;;;;;

; EDN can be extended by tagging elements with # symbols.

#MyYelpClone/MenuItem {:name "eggs-benedict" :rating 10}

; Let me explain this with a clojure example. Suppose I want to transform that
; piece of EDN into a MenuItem record.

(defrecord MenuItem [name rating])

; To transform EDN to clojure values, I will need to use the built in EDN
; reader, edn/read-string

(edn/read-string "{:eggs 2 :butter 1 :flour 5}")
; -> {:eggs 2 :butter 1 :flour 5}

; To transform tagged elements, define the reader function and pass a map
; that maps tags to reader functions to edn/read-string like so

(edn/read-string {:readers {'MyYelpClone/MenuItem map->menu-item}}
                 "#MyYelpClone/MenuItem {:name \"eggs-benedict\" :rating 10}")
; -> #user.MenuItem{:name "eggs-benedict", :rating 10}
```

References

- EDN spec
- Implementations
- Tagged Elements