Why Clojure?

Simple syntax

Macros

REPL

Interoperability

“Clojure is a dialect of Lisp, and shares with Lisp the code-as-data philosophy and a powerful macro system. Clojure is predominantly a functional programming language, and features a rich set of immutable, persistent data structures. When mutable state is needed, Clojure offers a software transactional memory system and reactive Agent system that ensure clean, correct, multi-threaded designs.”

- Rich Hickey

For more Clojure projects and platforms, check out:

Lumo:

Clojure hosted on Node.js

https://http://lumo-cljs.org/

ClojureScript:

Clojure-to-JavaScript compiler

https://clojurescript.org/

Clojure CLR:

Clojure hosted on Common Language Runtime (.NET)

https://github.com/clojure/clojure-clr

https://clojure.org
https://github.com/clojure

Clojure

Simplify your development

A simple, yet powerful modern LISP hosted on top of the JVM

Developed by Rich Hickey

Syntax

S-expressions

Clojure uses s-expressions as syntax.

S-expressions allow the user to see the scope of your functions as you write it, which provides clarity when reading code.

Polish/prefix notation

Polish/prefix notation is when operators come before operands. This notation, after being learned, provides a consistent and less ambiguous interface for the developer than traditional mathematical notation – operators always come before operands.

Infix notation converted to Polish/ prefix notation:

2 + (5 - 3) → (+ 2 (- 5 3))

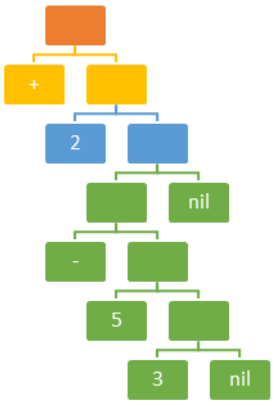

Figure 1: Tree data structure representing (+ 2 (- 5 3) as an s-expression

Java interoperability

Clojure is a hosted language on top of the JVM. This means that Clojure can use any function that was declared in Clojure, and/ as well as any Java function. This means it can work on top of already-existing Java code, and use core and internal Java libraries. This is possible, because Clojure is compiled to the same bytecode that Java is compiled to, that is, Java bytecode.

Example

Instance method 1:

```
(let [d (java.util.Date.)]
    (. D getTime))
```

Instance method 2 (most common):

```
(let [d (java.util.Date.)]
    (.getTime d))
```

Static method 1:

```
(. Math floor 5.677)
```

Static method 2 (ClassName/ methodName):

```
(Math/floor 5.677)
```

The REPL

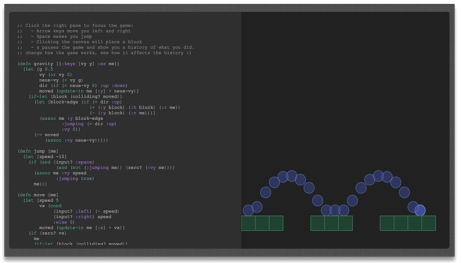

Figure 2: screenshot of a game written

The REPL is an environment that provides the user with a programming sandbox where code can be input, evaluated, printed out to the user, and repeated. This is not only quicker than coding, compiling, and debugging, but it also allows the developer to view and edit code/data as their program runs. An example of this workflow in progress is a screenshot of a developer editing a simple game while it runs, viewing movement trails as they code (see Figure 2 above).

For more information about Clojure:

https://clojure.org

https://clojuredocs.org/