Forms

- Clojure code is composed of nested expressions, or *forms.*

- The simplest of forms evaluate to themselves.

Self-evaluating forms, or literals

```
=> 42
=> “Hello World!”
=> nil
```

Function Calls

- A list is denoted by a pair of parentheses.

- To call a function, write the function name at the beginning of a list followed by its arguments.

- The arguments of a function can be any Clojure form.

```
(<fn-name> <arg1> <arg2>)
```

Function Calls

```
=> (+ 1 2)

=> (+ 1
      (* 5 7))

=> (str “Hello” “ “ “World!”)

=> (println “Hello World!”)

=> (println (str “Hello” “ “ “World!”))
```

Prefix notation

- Eliminates precedence rules
- Supports an arbitrary number of operands easily
- Makes the syntax very consistent

Naming values with def

- To assign a name to the result of a form, use def.
- def is a *special form* — it is an important language primitive that does not follow the same evaluation rules as function calls.
- Clojure has only a few special forms — see https://clojure.org/reference/special_forms
- Names defined using def can be used in all subsequent expressions.

Naming values with def

```
=> (def pi 3.14159)
=> (def radius 10)
=> (* pi (* radius radius))
=> (def area (* pi (* radius radius)))
=> (println pi)
```