

# Introduction to Functional Programming

Jean-Louis Giordano (@jellismymind)

*<2015-04-08 Wed>*

## Contents

<b>1</b>	<b>What is FP?</b>	<b>2</b>
<b>2</b>	<b>What is a value?</b>	<b>2</b>
<b>3</b>	<b>What is an expression?</b>	<b>3</b>
<b>4</b>	<b>What is evaluation?</b>	<b>3</b>
<b>5</b>	<b>What is a variable?</b>	<b>3</b>
<b>6</b>	<b>What are side-effects?</b>	<b>4</b>
<b>7</b>	<b>Statement</b>	<b>4</b>
7.1	In Python: . . . . .	4
7.2	In Clojure and Haskell . . . . .	5
7.3	Why is that? . . . . .	5
<b>8</b>	<b>What is a function?</b>	<b>5</b>
<b>9</b>	<b>What is the arity of a function?</b>	<b>6</b>
<b>10</b>	<b>Routine vs Function</b>	<b>6</b>
<b>11</b>	<b>What is a pure function?</b>	<b>6</b>
<b>12</b>	<b>Immutable vs Mutable</b>	<b>7</b>
<b>13</b>	<b>Referencial Transparency</b>	<b>7</b>
<b>14</b>	<b>What is Application?</b>	<b>7</b>

<b>15 Higher order function</b>	<b>8</b>
<b>16 Partial Application</b>	<b>8</b>
<b>17 Partial Application (bis)</b>	<b>8</b>
<b>18 How to do things?</b>	<b>9</b>
18.1 map . . . . .	9
18.2 filter . . . . .	9
18.3 reduce / fold . . . . .	9
18.4 list comprehension . . . . .	9
18.5 recursion . . . . .	9
18.6 pattern matching . . . . .	9
18.7 function composition . . . . .	9
<b>19 What are Types?</b>	<b>9</b>
<b>20 What is polymorphism?</b>	<b>9</b>
20.1 how to achieve polymorphism in FP vs OOP? . . . . .	9
20.2 subtyping . . . . .	9
20.3 ad hoc polymorphism (function overloading) . . . . .	9
20.4 parametric polymorphism . . . . .	9
<b>21 Macros vs Monads?</b>	<b>9</b>
<b>22 Does it scale?</b>	<b>9</b>



## 1 What is FP?

## 2 What is a value?

A value is the final result of a computation.

The value of  $1 + 1$  is 2.

### 3 What is an expression?

A symbol or combination of symbols that represents a value or a relationship between values

1 + 1 is an expression, it reduces to the value 2.

2 is also an expression as well as a value.

### 4 What is evaluation?

Evaluation is the reduction of an expression to its value.

Evaluate: e- (ex-, out) + value "To extract the value"

Example of evaluation:

1 + 1 + 1

1 + 2

3

+-----+

| Expression |

+-----+

|

V

+-----+

| Evaluation |

+-----+

|

V

+-----+

| Value |

+-----+

### 5 What is a variable?

A variable is a reference to a value.

In:

a = 2

a is a variable, referencing the value 2

## 6 What are side-effects?

A side effect is a step in the evaluation of an expression that has effects outside of the expression itself.

Examples:

```
print "hello"
```

```
a = 1; a += 1
```

```
+-----+
| Expression |
+-----+
      |
      V
+-----+
| Evaluation |~~~> Side Effect
+-----+
      |
      V
+-----+
|   Value   |
+-----+
```

## 7 Statement

A statement is an expression that evaluates to nothing.

### 7.1 In Python:

Expression:

```
a = 1 + 1
a = lambda x: x ** 2
```

Statement:

```
if True:
    a = "hello"
else:
    a = "goodbye"
```

## 7.2 In Clojure and Haskell

Only expressions

```
(def a
  (if true "hello" "goodbye"))

a = if True
    then "hello"
    else "goodbye"
```

## 7.3 Why is that?

Statements require side effects, intrinsically imperative.

```
+-----+
| Expression |
+-----+
    |
    V
+-----+
| Evaluation |~~> Side Effect
+-----+
    |
    V
    X
```

## 8 What is a function?

An abstraction for an expression, where one or several values in the expression are replaced by variables.

Let's abstract the following expression:

```
1 + 1
```

```
def inc (x):
  return x + 1
```

```
inc = lambda x: x + 1
```

```
(+ 1 1)
(def inc (fn [x] (+ x 1)))
(defn inc [x] (+ x 1))
```

```
1 + 1
inc = \x -> x + 1
inc x = 1 + x
```

Question: Is a function a value?

## 9 What is the arity of a function?

The number of arguments a function takes is its arity.

```
# arity 0
def zero ():
    return 0

# arity 1
def inc (x):
    return x + 1

# arity 2
def add (x, y):
    return x + y
```

## 10 Routine vs Function

A routine is an abstraction that do not return a value.

```
def a (x):
    print x

def b (x):
    return x
```

a is a routine, b is a function.

A procedure can either be a routine or a function.

## 11 What is a pure function?

A pure function is a side-effect free function that always maps a given input to the same output.

Which of the following is a pure function?

```
def inc (x):
    return x + 1

def one (x):
    print x
    return x

def rand (x):
    # non deterministic!
    return x * random.random()
```

## 12 Immutable vs Mutable

Immutable means that cannot change. Think "read only".

Persistent Datastructures are immutable, and can't be updated in-place.

## 13 Referencial Transparency

An expression that is deterministic and without side-effects is referentially transparent.

It means it can be replaced by its value without changing the behaviour of the program.

## 14 What is Application?

Calling a function with some arguments is applying that function to the value of those arguments.

Abstraction and Application are the core concepts of functional programming.

```
f(arg1, arg2)
1 + 2
```

```
(f arg1 arg2)
(+ 1 2)
```

```
f arg1 arg2
1 + 2
(+) 1 2
```

## 15 Higher order function

Functions can return functions, and take functions as argument.

```
def apply (f, x, y):  
    return f(x, y)  
  
apply(add, 1, 2)  
  
def incrementer (n):  
    return lambda m: m + n  
  
add2 = incrementer(2)  
add2(4)
```

## 16 Partial Application

Take a function of arity  $n$ , and  $m < n$  arguments, and return a function of arity  $n - m$ .

Example:

```
(+ 1 2 3)  
((partial +) 1 2 3)  
((partial + 1) 2 3)  
((partial + 1 2) 3)  
((partial + 1 2 3))  
  
1 + 1  
(+) 1 1  
((+) 1) 1  
-- Currying
```

## 17 Partial Application (bis)

```
(defn part [f & args]  
  (fn [& rest]  
    (apply f (concat args rest))))  
  
((part + 1 2) 3 4)
```



## 18 How to do things?

### 18.1 map

### 18.2 filter

### 18.3 reduce / fold

### 18.4 list comprehension

### 18.5 recursion

### 18.6 pattern matching

### 18.7 function composition

## 19 What are Types?

Types are sets of values.

1 belongs to several types: it's an Integer, a Number, a Value, the value 1.

One of the elements of the set of all Values.

One of the elements of the set of all Integers.

The only element in the set of all values that are 1.

1 has the type Value, Integer, Being 1

## 20 What is polymorphism?

### 20.1 how to achieve polymorphism in FP vs OOP?

### 20.2 subtyping

### 20.3 ad hoc polymorphism (function overloading)

### 20.4 parametric polymorphism

## 21 Macros vs Monads?

## 22 Does it scale?

- Is it fast?