Introduction to Functional Programming

Jean-Louis Giordano (@jellismymind)

<2015-04-08 Wed>

Contents

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?

Definition:

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
```

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
```

7 What is a function?

An abstraction for an expression, where one or several values are 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
```

Question: Is a function a value?

8 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.

9 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):
  return x * random.random()
```

10 What is application?

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

Abstraction and Application are the core concepts of functional programming.

11 immutable vs mutable

12 referencial transparency

13 function vs method

14 partial function

15 Statement vs Expression

Expressions return something, Statements return nothing.

During evaluation: Expressions reduce to a value, Statements reduce to and action.

15.1 In Python:

Expression:

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

Statement:

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

15.2 In Clojure and Haskell

Only expressions

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

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

15.3 Why is that?

Statements require side effects, intrinsically imperative.

16 What is polymorphism?

- how to achieve polymorphism in FP vs OOP?
- subtyping
- ad hoc polymorphism (method overloading)
- parametric polymorphism

17 Macros vs Monads?

18 Does it scale?

- Is it fast?