Introduction to Functional Programming

Jacek Wasilewski

Overview

- Functional Programming
- Imperative vs Functional Programming
- λ-calculus

Functional Programming

- **Functional programming** is a programming paradigm, where function calls are the primary programming constructs.
- In the restricted sense, functional programming means programming without **mutable variables**, **assignments**, **loops**, **imperative control** structures.
- In the wider sense, functional programming means **focusing on functions**.

Mutable Variables & Assignments

Imperative Programming

```
<var1> = 5;
<var2> = <expression>;
<var1> = <var1> + <var2>
<var2> = 10;
...
```

The same name may be associated with different values.

Functional Programming

```
<function1>(
  <function2>(
    <function3>(<value>)
    ...
  )
)
```

A name is only ever associated with one value.

Loops

Imperative Programming

```
int sum = 0;
for (int i = 1; i <= n; i++) {
  sum = sum + i;
}
```

New values may be associated with the same name through command repetition.

Functional Programming

```
def sum(i: Int, n: Int): Int = {
  if (i > n)
    return 0
  else
    return i + sum(i + 1, n)
}
sum(1, 1000)
```

New values are associated with new names through recursive function call nesting.

Execution Order

Imperative Programming

```
// X=1, Y=10
T = X;  // T=1, X=1, Y=10
X = Y;  // T=1, X=10, Y=10
Y = T;  // T=1, X=10, Y=1
```

```
// X=1, Y=10
X = Y;  // X=10, Y=10
T = X;  // T=10, X=10, Y=10
Y = T;  // T=10, X=10, Y=10
```

Fixed execution orders.

Functional Programming

```
def f(x: Int, y: Int, z: Int): Int = ...
def a(p: Int): Int = ...
def b(q: Int): Int = ...
def c(r: Int): Int = ...
```

```
f(a(d), b(d), c(d))
```

No fixed execution orders.

λ-calculus

- λ-calculus provides a theoretical framework for describing functions and their evaluation.
- It is simple and it is based on:
  - function abstraction, to generalise expressions through names,
  - function application, to evaluate generalised expressions by giving names values.
- λ-calculus treats functions “anonymously”, without naming them.
- Sample λ-expression:

$$(x, y) \to x \times x + y \times y$$