Programming Languages – Functional Languages

Mahmoud Abdelsalam

Functional Programming concepts

- Pure functions & side effects
- Referential transparency
- First class functions & higher order functions
- Immutability
- Recursion & tail-recursion
- Lambda functions
- Strict and lazy evaluation
- Pattern matching

Functional Programming concepts – Lambda Functions

- Lambda function is a function literal (anonymous function) – Recall Previous lecture.

Functional Programming concepts – Lambda Functions

- Lambda function is a function literal (anonymous function) – Recall Previous lecture.

  val add = (a: Int, b: Int) => a + b

Functional Programming concepts – Lambda Functions

- Lambda function is a function literal (anonymous function) – Recall Previous lecture.

```
val add = (a: Int, b: Int) => a + b
val add: (Int, Int) => Int = (a, b) => a + b
```

Functional Programming concepts – Lambda Functions

- Lambda function is a function literal (anonymous function) – Recall Previous lecture.

  val add = (a: Int, b: Int) => a + b

  val add: (Int, Int) => Int = (a, b) => a + b

  Even Better:

  val add: (Int, Int) => Int = _ + _

  //Each underscore stands for a new unnamed parameter

Functional Programming concepts

- Pure functions & side effects
- Referential transparency
- First class functions & higher order functions
- Immutability
- Recursion & tail-recursion
- Lambda functions
- Strict and lazy evaluation
- Pattern matching

Functional Programming concepts – Strict and Lazy Evaluation

- Strict evaluation:
  - Used by most traditional programming languages

Functional Programming concepts – Strict and Lazy Evaluation

- Strict evaluation:
  - Used by most traditional programming languages
  - Evaluates an expression as soon as it is bound to a variable

Functional Programming concepts – Strict and Lazy Evaluation

- Strict evaluation:
  - Used by most traditional programming languages
  - Evaluates an expression as soon as it is bound to a variable, for example:

```
val x = 1 + 2 // x = 3 at this line
```

Functional Programming concepts – Strict and Lazy Evaluation

- Strict evaluation:
  - Used by most traditional programming languages
  - Evaluates an expression as soon as it is bound to a variable, for example:

```
val x = 1 + 2 // x = 3 at this line
val x = add(1, 2) // x = 3
```

Functional Programming concepts – Strict and Lazy Evaluation

- Strict evaluation:
  - Used by most traditional programming languages
  - Evaluates an expression as soon as it is bound to a variable, for example:

```
val x = 1 + 2 // x = 3 at this line
val x = add(1, 2) // x = 3
val date = new java.util.Date //Data is determined at this line
```

Functional Programming concepts – Strict and Lazy Evaluation

- Lazy evaluation:
  - Delays the evaluation of an expression until its value is needed

Functional Programming concepts – Strict and Lazy Evaluation

- Lazy evaluation:
  - Delays the evaluation of an expression until its value is needed, for example:

```
        val x = 1 + 2 // x is NOT 3 at this line
println(x) // x = 3 at this line
```

Functional Programming concepts – Strict and Lazy Evaluation

- Lazy evaluation:
  - Delays the evaluation of an expression until its value is needed, for example:

```
lazy val x = 1 + 2 // x is NOT 3 at this line
println(x) // x = 3 at this line
```

Functional Programming concepts – Strict and Lazy Evaluation

- Lazy evaluation:
  - Delays the evaluation of an expression until its value is needed, for example:

```
lazy val x = 1 + 2 // x is NOT 3 at this line
println(x) // x = 3 at this line

lazy val x = add(1, 2) // x is NOT 3 at this line
println(x) // x = 3 at this line
```

Functional Programming concepts – Strict and Lazy Evaluation

```
val date = new java.util.Date //Data is NOT determined at this line
println(date) //data is set at this point
```

Note: might have a delay between the time it is defined and the time it is needed (used)

Functional Programming concepts – Strict and Lazy Evaluation

- Example:

```
val date = new java.util.Date
lazy val lazy_date = new java.util.Date
Thread.sleep(5000) //Sleep for 5 seconds
println(date)
println(lazy_date) //Print time with 5
seconds after "date"
```

Functional Programming concepts – Strict and Lazy Evaluation

- Another Example:

```
lazy val lazy_add: (Int, Int) => Int =
                    {println("Lazy"); _ + _}
val add: (Int, Int) => Int =
                    {println("Strict"); _ + _}
lazy val l = lazy_add(1, 2)
val s = add(1, 2)
println(l)
```

Output:

```
Strict
Lazy
3
```

Functional Programming concepts

- Pure functions & side effects
- Referential transparency
- First class functions & higher order functions
- Immutability
- Recursion & tail-recursion
- Lambda functions
- Strict and lazy evaluation
- Pattern matching