Programming Languages – Functional Languages

Mahmoud Abdelsalam

Functional Programming

- Recall – In functional programming, you can write programs ONLY using pure functions and immutable values.
- This is, however, not how it works in many cases.

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

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

Pure Functions & Side Effects

- Pure functions, just **maps** input to output.
  - No I/O operations
  - No input modifications
  - No console reads/prints
  - No database calls
  - No global variable usage/modifications

Pure Functions & Side Effects

- Pure functions, just **maps** input to output.
  - No I/O operations
  - No input modifications
  - No console reads/prints
  - No database calls
  - No global variable usage/modifications

Simply: It can’t affect the outside world in anyway since this will be a **side effect**!

Referential Transparency

Referential transparency can be used to check if a function is pure!

HOW?

Referential Transparency

Referential transparency can be used to check if a function is pure!
HOW?

If you can replace an expression with its value without changing the program behavior

Referential Transparency - Examples

```
def add1(i: Int) : Int = {
    i + 1
}


def add1(i: Int): Int = {
    println(i)
    i  + 1
}
```

Referential Transparency - Examples

```
def add1(i: Int) : Int = {
    i + 1
}
```

Replace add1(x) with the value of x + 1.
Does it change the program behavior?

```
def add1(i: Int): Int = {
    println(i)
    i  + 1
}
```

Referential Transparency - Examples

```
def add1(i: Int) : Int = {
    i + 1
}
```

✓

Replace add1(x) with the value of x + 1.
Does it change the program behavior?

```
def add1(i: Int): Int = {
    println(i)
    i  + 1
}
```

Referential Transparency - Examples

```
def add1(i: Int) : Int = {
    i + 1
}
```

✓

Replace add1(x) with the value of x + 1.
Does it change the program behavior?

```
def add1(i: Int): Int = {
    println(i)
    i  + 1
}
```

✗

Referential Transparency - Examples

Math.sqrt(4.0)

Referential Transparency - Examples

Math.sqrt(4.0)

Can you replace *Math.sqrt(4.0)* with 2.0 without changing the program behavior?

Referential Transparency - Examples

Math.sqrt(4.0)

✓

Can you replace *Math.sqrt(4.0)* with 2.0
without changing the program behavior?

Referential Transparency - Examples

```
Math.sqrt(4.0)
```

Can you replace *Math.sqrt(4.0)* with 2.0 without changing the program behavior?

✓

```
var i: Int = 1
def addi(j: Int): Int = i + j
```

Referential Transparency - Examples

```
Math.sqrt(4.0)
```

Can you replace *Math.sqrt(4.0)* with 2.0 without changing the program behavior?

✓

```
var i: Int = 1
def addi(j: Int): Int = i + j
```

Can you replace addi(10) with it’s corresponding value everywhere in the program without changing it’s behavior?

Referential Transparency - Examples

```
Math.sqrt(4.0)
```

Can you replace *Math.sqrt(4.0)* with 2.0 without changing the program behavior?

✓

```
var i: Int = 1
def addi(j: Int): Int = i + j
```

✗

Can you replace addi(10) with it's corresponding value everywhere in the program without changing it's behavior?

Recall – Why Pure functions?

- Safe programming – no side effects
- Easy to test
- Composable
- Results can be cached
- Can be lazy (discussed later)

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

First Class Functions & Higher Order Functions

- First class functions:
  - NOT ONLY declared and called, but CAN also be used in every segment of the language as any other data type.
  - Everything a variable can do, a function can do.

First Class Functions & Higher Order Functions

- First class functions:
  - NOT ONLY declared and called, but CAN also be used in every segment of the language as any other data type.
  - Everything a variable can do, a function can do.
    1) Created in literal form without ever having been assigned an identifier

First Class Functions & Higher Order Functions

- First class functions:
  - NOT ONLY declared and called, but CAN also be used in every segment of the language as any other data type.
  - Everything a variable can do, a function can do.
    1) Created in literal form without ever having been assigned an identifier
    2) Can be stored in a container such as a variable

First Class Functions & Higher Order Functions

- First class functions:
  - NOT ONLY declared and called, but CAN also be used in every segment of the language as any other data type.
  - Everything a variable can do, a function can do.
    1) Created in literal form without ever having been assigned an identifier
    2) Can be stored in a container such as a variable
    3) Can be used as a parameter to another function

First Class Functions & Higher Order Functions

- First class functions:
  - NOT ONLY declared and called, but CAN also be used in every segment of the language as any other data type.
  - Everything a variable can do, a function can do.
    1) Created in literal form without ever having been assigned an identifier
    2) Can be stored in a container such as a variable
    3) Can be used as a parameter to another function
    4) Can be used as the return value from another function

First Class Functions & Higher Order Functions

- First class functions:
  - NOT ONLY declared and called, but CAN also be used in every segment of the language as any other data type.
  - Everything a variable can do, a function can do.
    1) Created in literal form without ever having been assigned an identifier
    2) Can be stored in a container such as a variable
    3) Can be used as a parameter to another function
    4) Can be used as the return value from another function
  - All functions are first class functions by default in Scala.
- Higher order functions either:
  - Use other functions as parameters
  - Use functions as return values

First Class Functions & Higher Order Functions - Example

Examples from: https://docs.scala-lang.org/tour/higher-order-functions.html

1) Created in literal form without ever having been assigned an identifier

First Class Functions & Higher Order Functions - Example

Examples from: https://docs.scala-lang.org/tour/higher-order-functions.html

1) Created in literal form without ever having been assigned an identifier

*(x: Int) => x * 2*

Known as literal/anonymous function

First Class Functions & Higher Order Functions - Example

2) Can be stored in a container such as a variable

First Class Functions & Higher Order Functions - Example

2) Can be stored in a container such as a variable

```
val double = (x: Int) => x * 2
```

First Class Functions & Higher Order Functions - Example

3) Can be used as a parameter to another function

First Class Functions & Higher Order Functions - Example

3) Can be used as a parameter to another function

```
val salaries = List(10, 20 50)
val double = (x: Int) => x * 2
val newSalaries = salaries.map(double)
```

First Class Functions & Higher Order Functions - Example

3) Can be used as a parameter to another function

```
val salaries = List(10, 20 50)
val double = (x: Int) => x * 2
val newSalaries = salaries.map(double)
```

Can be rewritten as:

```
val salaries = List(10, 20, 50)
val newSalaries = salaries.map(x => x * 2)
```

First Class Functions & Higher Order Functions - Example

3) Can be used as a parameter to another function

```
val salaries = List(10, 20 50)
val double = (x: Int) => x * 2
val newSalaries = salaries.map(double)
```

Can be rewritten as:

```
val salaries = List(10, 20, 50)
val newSalaries = salaries.map(x => x * 2)
```

Anonymous function

First Class Functions & Higher Order Functions - Example

4) Can be used as the return value from another function

First Class Functions & Higher Order Functions - Example

4) Can be used as the return value from another function

```
def urlBuilder (use_ssl: Boolean, domainName: String): (String,
String) => String = {
    val ssl = if (use_ssl) "https://" else "http://"
    (endpoint: String, query: String) => ssl + domainName + "/"
                                         + endpoint + "?" + query
  }
```

First Class Functions & Higher Order Functions - Example

4) Can be used as the return value from another function

```
val domainName = "www.example.com"
def getURL = urlBuilder(true, domainName)
val endpoint = "users"
val query = "id=1"
val url = getURL(endpoint, query)
println("url: " + url) //url: https://www.example.com/users?id=1
```

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