Lecture 4 Functional Programming

Distributed Systems

Semester 1 2022/2023

Dr. Ayman Khallel Ibrahim

Content

- What is functional programming
- Functional programming languages
- Features and concepts
- Examples
- Advantages
- Limitations
- Summary

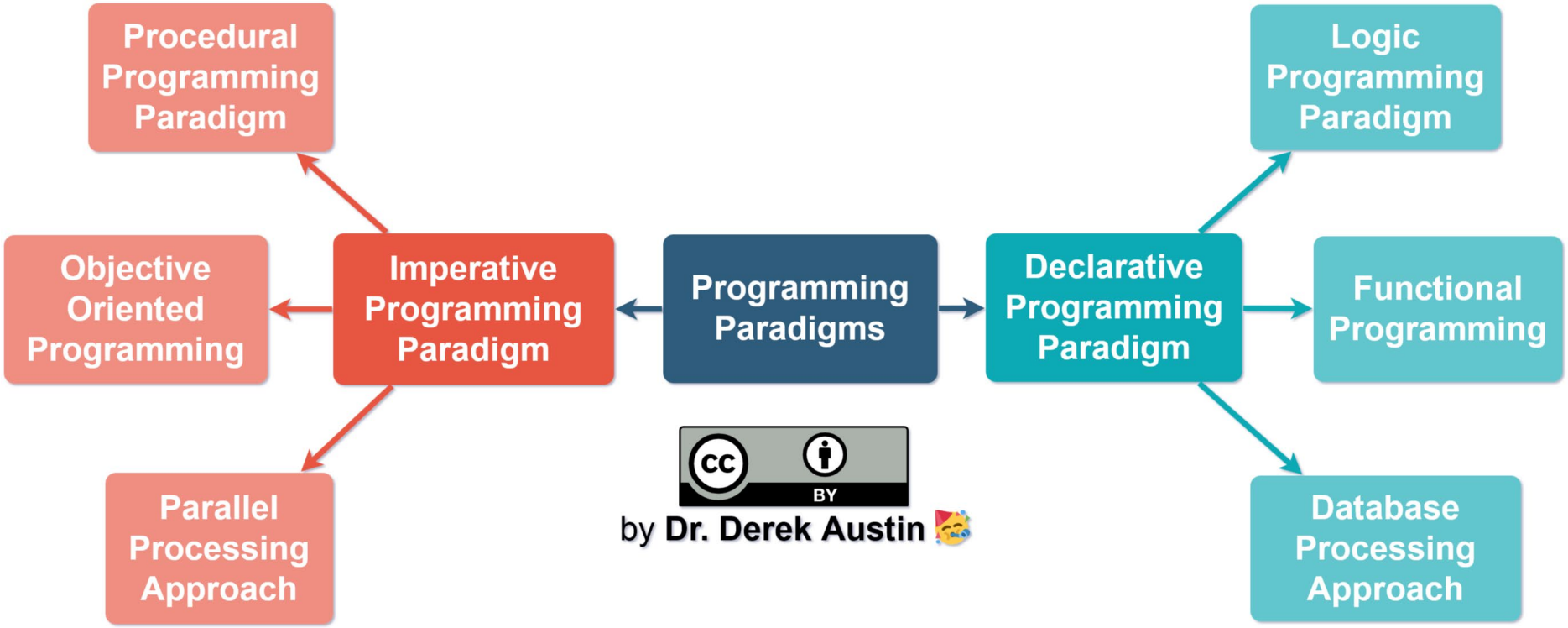
Throwback to Chapter 2

- We learn that
  - Locking is applied to data that both shared between threads and might change.
- Thus, we need a programming paradigm that handle this situation
  - Functional programming offers the concept for concurrency and parallelism
  - It has no mutable state to avoid problem with shared mutable state

What is functional programming?

- It is a **programming paradigm** where programs are constructed by applying and composing functions
- The design of the functional languages is based on mathematical functions
  - a function, in the mathematical sense, is a set of operations that perform some computation on the parameter(s) passed, and return a value
  - a function, in a programming language, is a set of code whose purpose is to compute based on the parameter(s) and return a value
- However, the function, as we see them in programming languages, does not necessarily reflect a mathematical function

Programming paradigm



Functions vs Procedure

- Functions may act more like procedures
  - a procedure is another unit of modularity
  - the idea behind a procedure is to accomplish one or more related activities
  - the activities should make up some logical goal of the program but may not necessarily be based on producing a single result
  - procedures may return 0 items or multiple items unlike functions
- In languages like C, there are no procedures, so functions must take on multiple roles
  - mathematical types of functions
  - functions that act as procedures
- Such functions can produce side effects
  - mathematical functions do not produce side effects

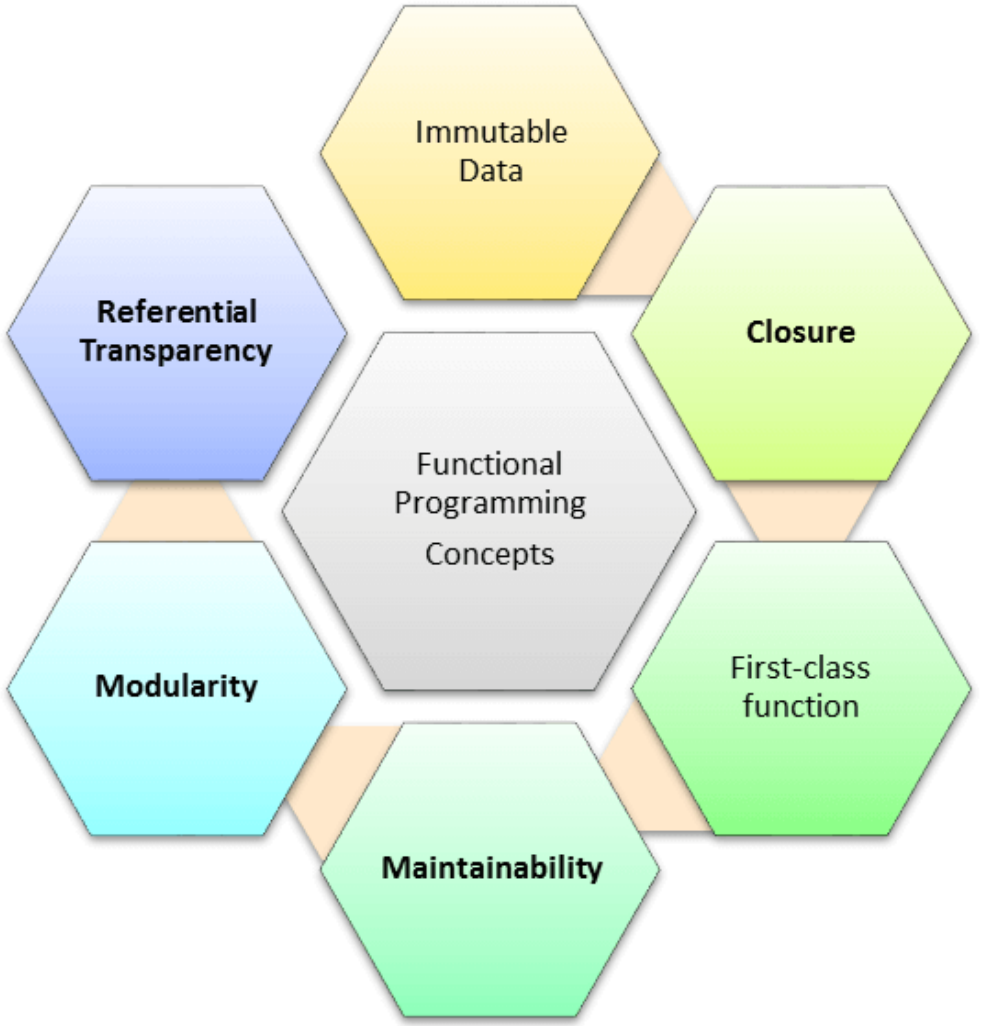
Example

```
// function
def square( n ):
        return n * n;
```

```
// procedure
def display(n):
        print( "The value is:" + n );
```

Why functional programming?

- Mathematical approach for problem solving
- Simple and less abstract
- Characteristics
  - Reuse
  - Test
  - Handle concurrency


Immutable Data
Referential Transparency
Closure
Functional Programming Concepts
Modularity
First-class function
Maintainability

Immutable Data

- Immutable Data implies that you ought to effectively ready to create data structures as opposed to altering ones which is now exist.

Example:

```
employee={
    'name': 'Ahmed',
    'salary': 6000,
}

updated_ employee={
    'name': 'Ahmed',
    'salary': 8000,
}
```

Referential transparency

- Referential transparency is the ability to replace a function call with its return value, the behavior of the program would be as predictable as before.

Example:

```
def two():
    return 2;
four = two() + two(); // 4
// or
four = two() + 2; // 4
// or
four = 2 + two(); // 4
```

First-class function

- ‘First-class function’ is a definition, attributed to programming language elements that have no limitation on their use. Hence, first-class functions can show up anyplace in the program.

Example:

```
def say_hello(name):
    return(f'Hello {name}')

Say_hello_2 = say_hello

print(Say_hello_2('Ahmed’))  // output: Hello Ahmed
```

Closure

- The closure is an inner function which can access variables of parent function's, even after the parent function has executed.

Example:

```
def create_counter():
    count=0
        def get_count():
            return count
        def increment():
            nonlocal count
            count+=1
    return (get_count, increment)
```

```
get_count, increment= creater_printer()
print(get_count())
Increment()
Increment()
print(get_count())
Increment()
Increment()
Increment()
print(get_count())
```

Ourput:

0

2

5

FP Concepts

- Modularity design builds usefulness. small modules can be coded rapidly and have a greater chance of re-use which without a doubt leads quicker development of projects. Aside from it, the modules can be tried separately which assists you with reduce the time spent on unit testing and debugging.
- Maintainability is a simple term which means FP programming is easier to maintain as you don’t need to worry about accidentally changing anything outside the given function.

Principles of Functional Programming

- Purity
  - Acts on their parameters only, produce same result, no side effect
- High order
  - A concept of first-class-citizen of the language, can take values or functions as parameters or return another functions
- Composition
  - One function become a result of another functions, e.g., func = f(g(x))
- Currying
  - Takes only one/single parameter at a time e.g., f(x)

Pure functions

- A pure function is a function which:
  1. Given the same inputs, always returns the same output
  2. Has no side-effects

Example:

```
//pure function
def sum(a, b):
      return a + b
```

```
//non-pure function
b =0
def sum(a):
      return a + b
```

Recursion in FP

- Break down a problem into smaller pieces.
- Benefit: helps us eliminate the side effects, which is typical of any imperative style looping.

- Example:

```
def count_down(x):
    if x<0:
        print('Done!')
        return
    print(x)
    count_down(x-1)
```

Advantages of FP over other programming paradigm

- High level – describing the result rather than step to get there. This single statement is preferable
- Transparent – behavior of FP depends only on the input, eliminating side effects
- Parallelizable – functions that have no side effects runs easily in parallel

Limitations of FP

- Not easy for beginner
- Hard to maintain as many objects evolve during coding
- Reuse is complicated
- Objects may not represent the problem correctly

Example program

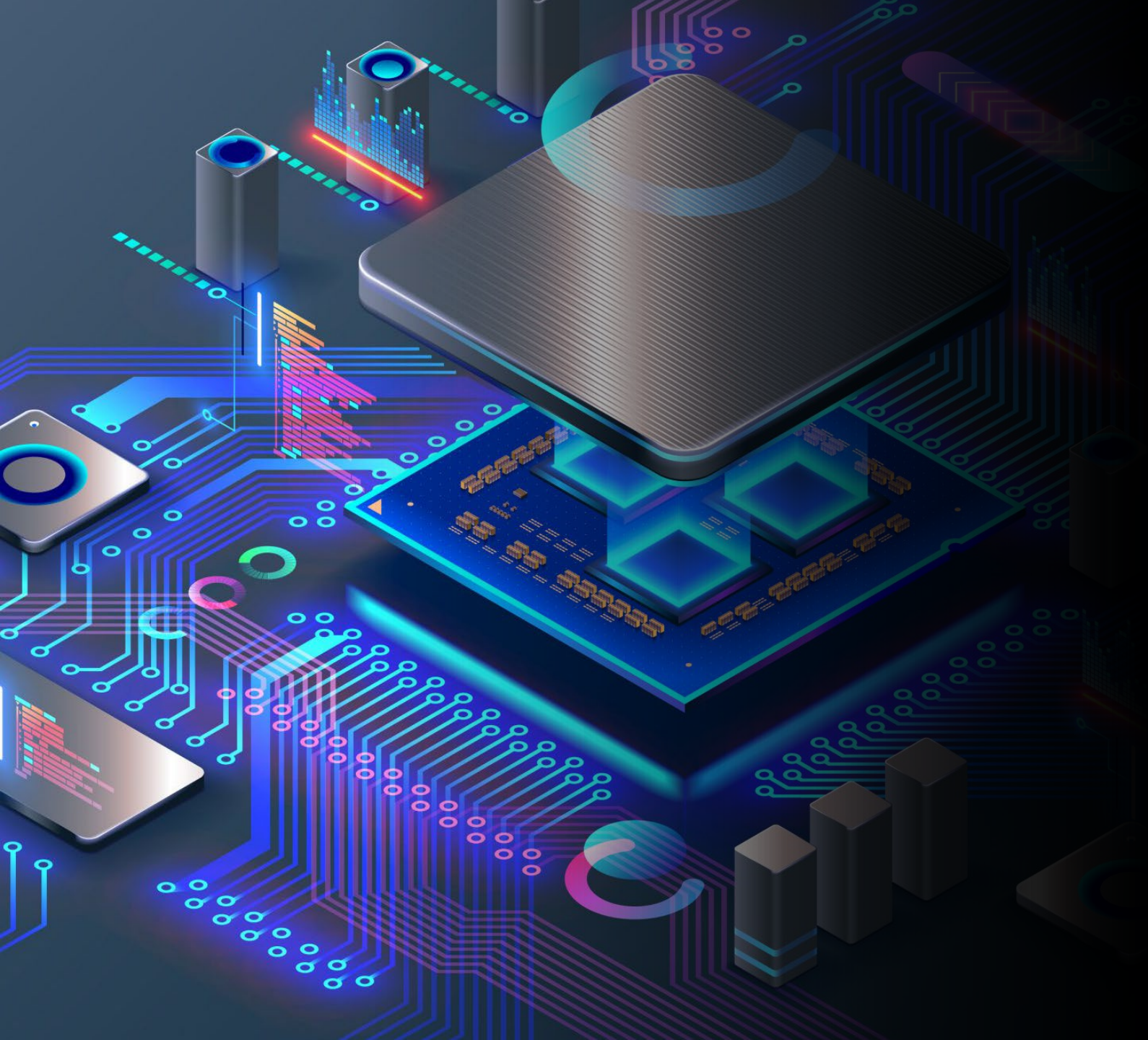
- Imagine that you want to find the sum of a sequence of numbers. In an imperative language like Java:

```
public int sum(int[] numbers) {
        int accumulator = 0;
        for (int n: numbers)
                accumulator += n;
        return accumulator;
}
```

- Is the program functional programming type?
- Discuss according to mutable state.

Summary

- FP is way of thinking on software development based on functional perspective
- FP focus on results rather than process
- FP is mimicking the mathematical function but not necessarily construct mathematical representation in the program
- Prominent programming languages for FP: Haskell, Clojure, Scala, Erlang
- Pure function depends on input and output
- FP uses immutable data, OOP uses mutable data



Thank You

Next: Functional Parallelism and Concurrency