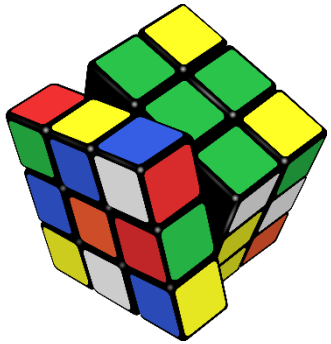


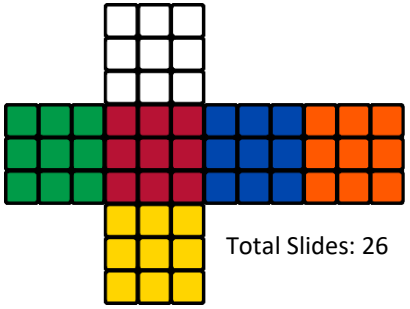
Big Data Engineering For Analytics

Functional Thinking

Suria R Asai

(suria@nus.edu.sg)

NUS-ISS



Total Slides: 26

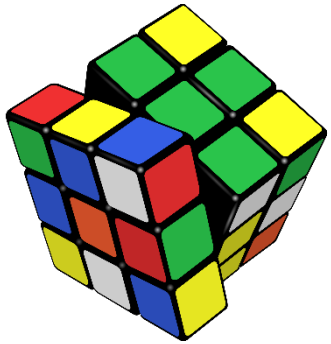
Learning Objectives & Agenda

Learning Objectives

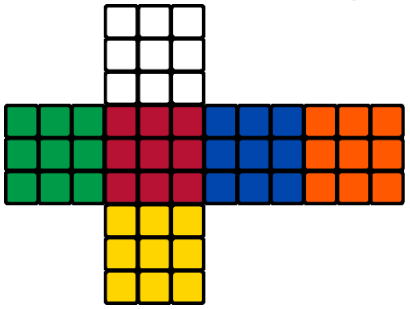
- Understand and apply principles of declarative programming
- Understand pure functions, the concept of immutability, and referential transparency
- Understand declarative vs imperative collections

Agenda

- Functional Programming – A quick Introduction
- Functional Paradigms
  - Pure Functions
  - Currying
  - Immutable Data Sets
  - Recursion



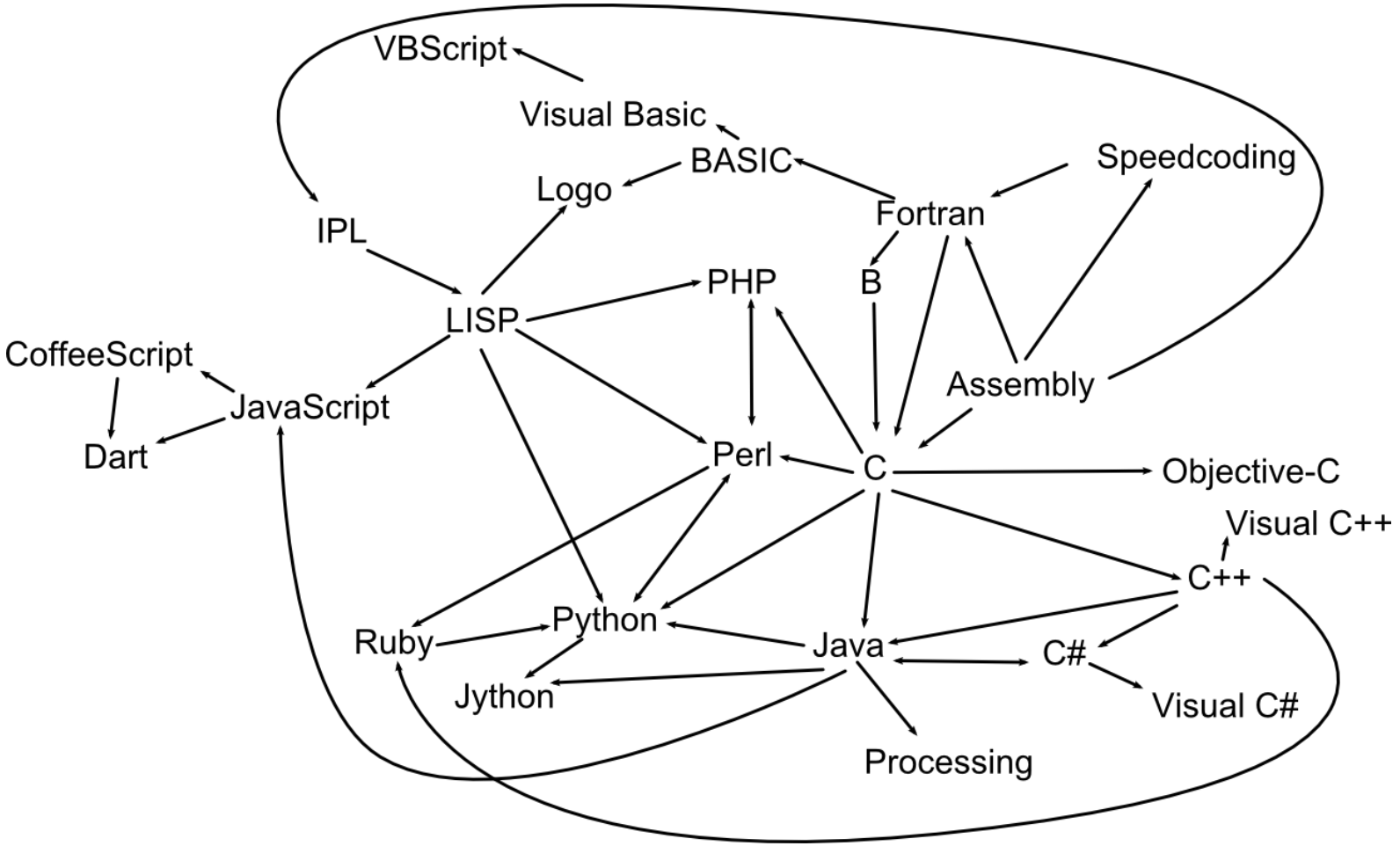
Big Data Engineering For Analytics



Introduction to Functional Thinking

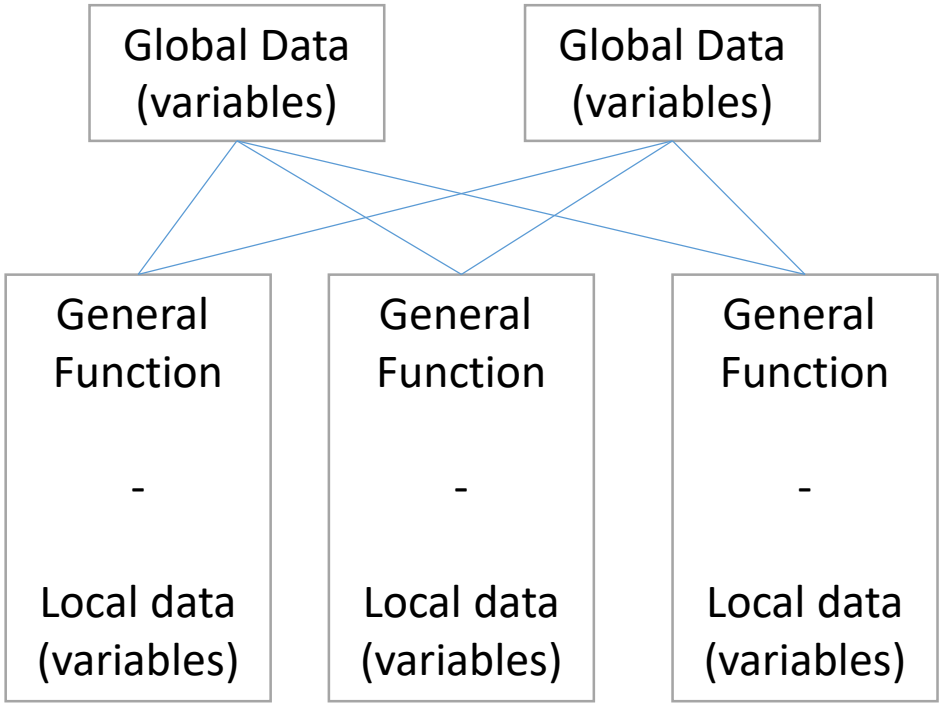
"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult." – C.A.R. Hoare (British computer scientist, winner of the 1980 Turing Award)

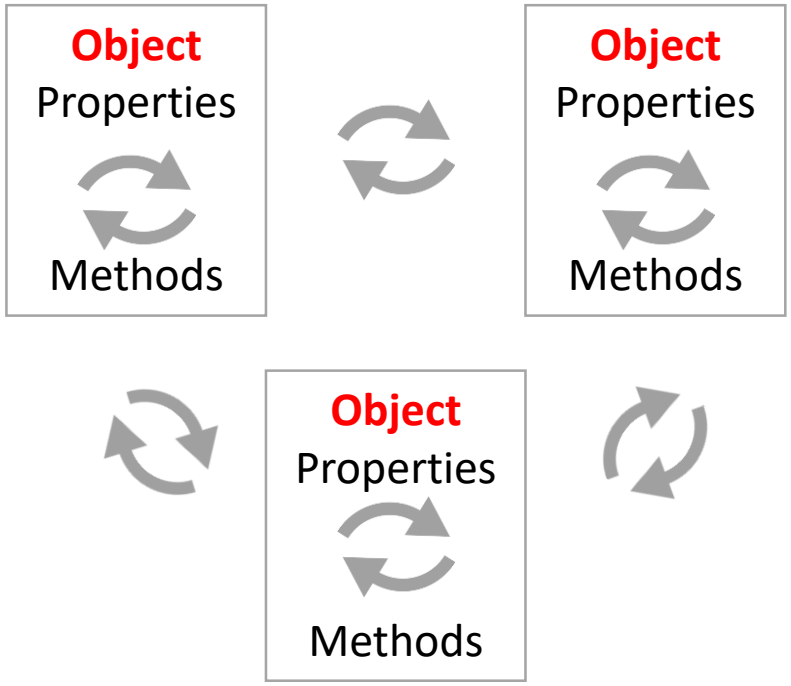
Programming Languages Through The Years



Procedural vs Object Oriented Programming

Data and Code

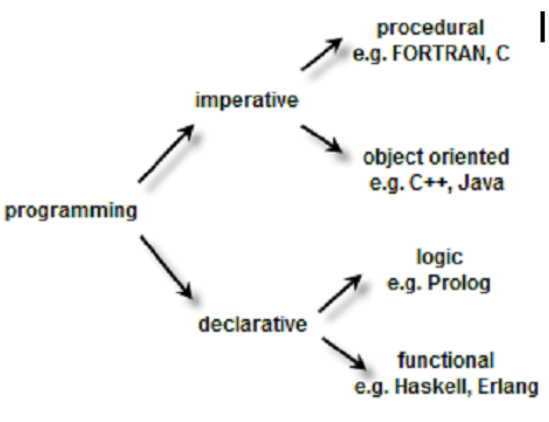


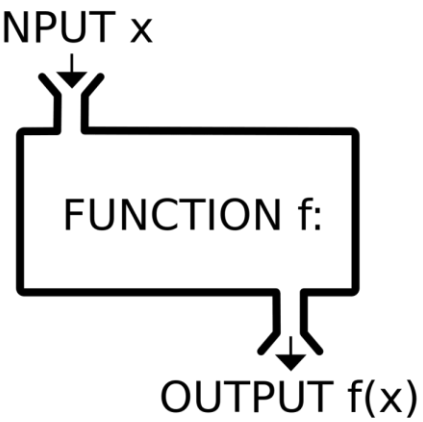


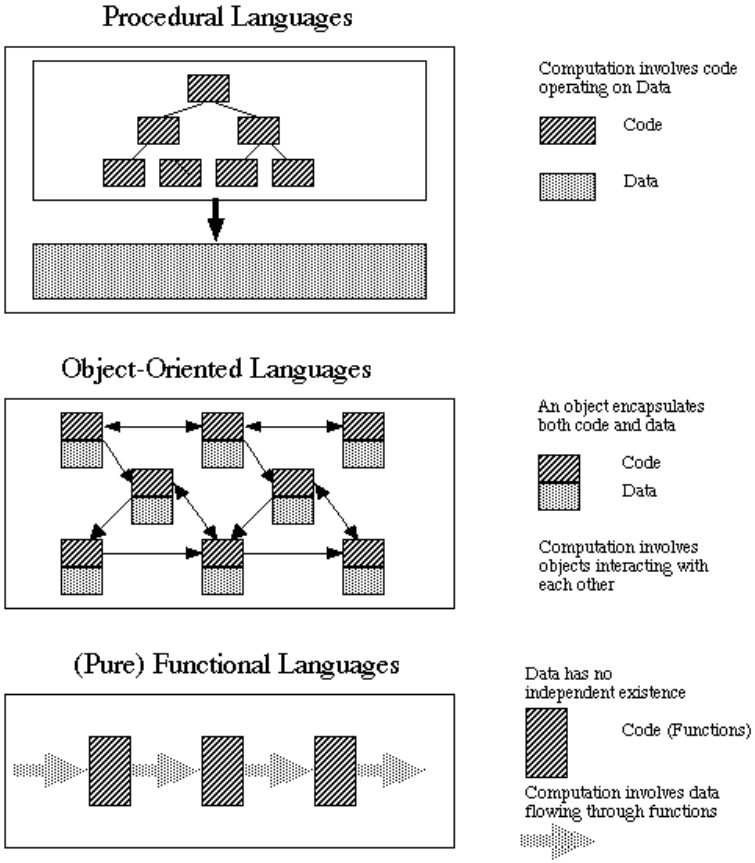
One **of** the most important characteristics **of procedural programming** is that it relies on procedures that operate on data - these are two separate concepts. In **object-oriented programming**, these two concepts are bundled into **objects**. This makes it possible to create more complicated behavior with less code.

Functional Programming

Data Data Data – Everywhere!!!!

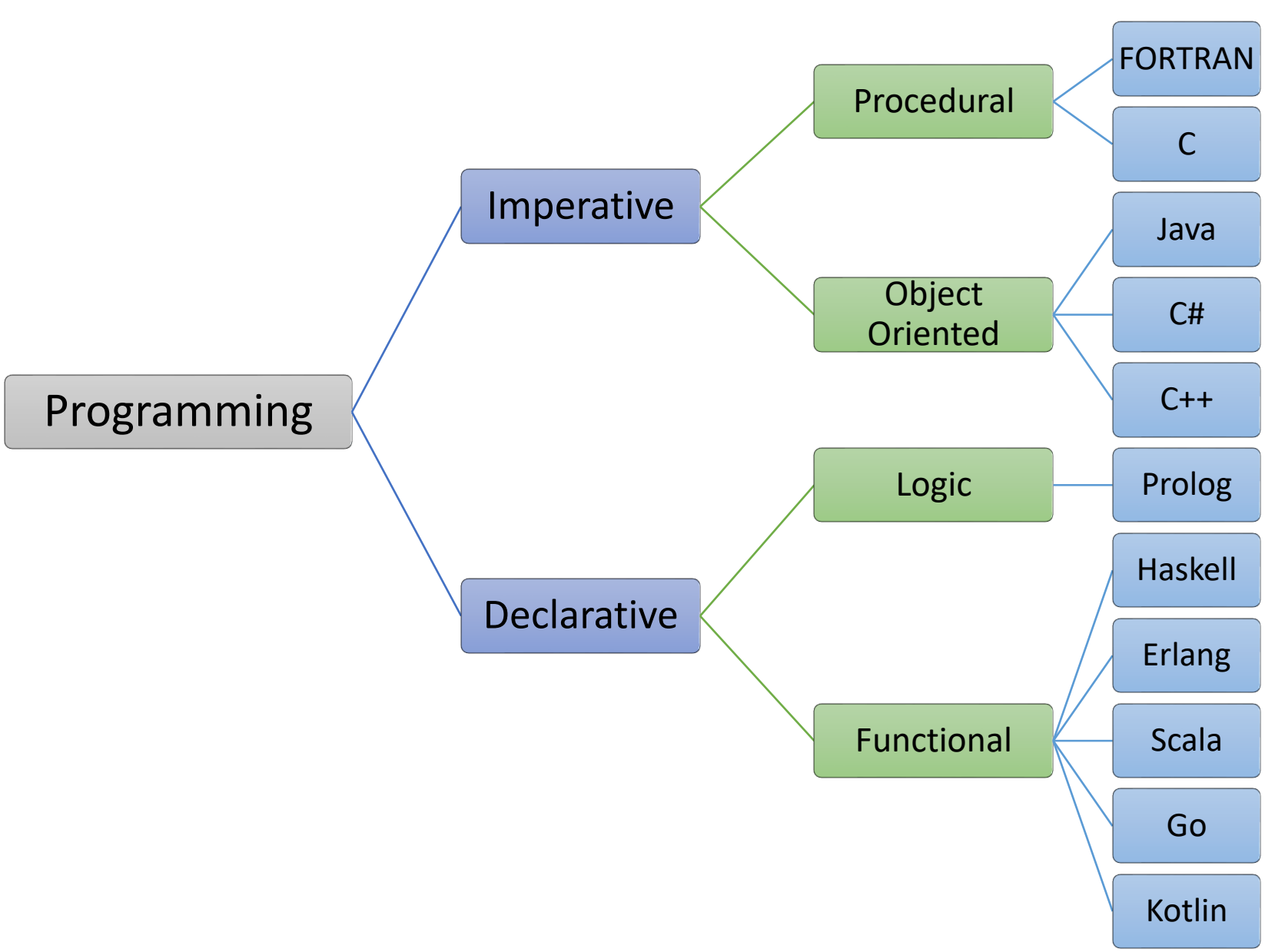






Functional programming is a **programming** paradigm is a style of building the structure and elements of computer programs that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.

https://medium.com/@richardeng/fp-is-for-nerds-6ed1ca43bb34



Principles of declarative programming

In declarative programming,

- We rely on **primitives** that are much closer to our domain.
- We are able to **create** primitives as we go.
- The language itself can be regarded as a domain-specific language (**DSL**)
- We can identify patterns and make them reusable. Don't repeat yourself. **(DRY Principle)** We can look for frequent operations and create patterns around the. Example: Filtering is important.
- We can pass functions to other functions inline, without first defining them. Treating functions as if they are ordinary variables allow us to achieve a new level of **abstraction**.
- We can create **anonymous functions**, functions as first-class citizens, and **custom operator** specifications.

Uniqueness of Functional Programming

- **Currying:** A single argument can translate multiple arguments in the series of functions.
- **Type Interference:** The FP languages are intelligent to reduces the efforts made during the programming.
  - We don't have to mention the return type of function and data types in a clear and detailed manner these kind of stuffs will be accomplished by the programming itself.
- **Immutability:** The already declared variables values can't be modified, this feature is known as Immutability.

Functions have no side effects

Functional languages such as **Standard ML, Scheme** and **Scala** do not restrict side effects, but it is customary for programmers to avoid them. The functional language **Haskell** expresses side effects such as I/O and other stateful computations using monadic actions.

What are side effects? A function has a side effect if it does something other than simply return a result, for example:

- Modifying a variable
- Modifying a data structure in place
- Setting a field on an object
- Throwing an exception or halting with an error
- Printing to the console or reading user input
- Reading from or writing to a file
- Drawing on the screen

Pure Functions

1. A pure function depends only on (a) its declared input parameters and (b) its algorithm to produce its result. A pure function has no "back doors," which means:
   - Its result can't depend on *reading* any hidden value outside of the function scope, such as another field in the same class or global variables.
   - It cannot *modify* any hidden fields outside of the function scope, such as other mutable fields in the same class or global variables.
   - It cannot depend on any external I/O. It can't rely on input from files, databases, web services, UIs, etc; it can't produce output, such as writing to a file, database, or web service, writing to a screen, etc.
2. A pure function does not modify its input parameters.

Higher-Order Function (HOF) basically means that
(a) you can treat a function as a value (val) — just like you can treat a String as a value
(b) you can pass that value into other functions.

"**Recursion** is a requirement of functional programming."

PF	=	ODI	+	NSE
Pure Function		Output Depends on Input		No Side Effects

Functional Programming

- FP prefers **pure functions**.
- FP **avoids side effects**.
- Functions are **first class objects**.
- FP prefers **immutable** objects.
- FP prefers **iterators** over lists.
- FP favours **lazy evaluation**.
- FP **avoids** traditional control constructs such as **loops** and **if** statements.
- FP often uses **recursion** to avoid loops .
- FP uses **higher order functions** to define new functions.

Functional Constructs

- Writing ***pure functions*** means that we focus on functions that take immutable data and produce new immutable data as output.
- *An **anonymous function** (function literal, lambda abstraction, or lambda expression) is a function definition that is not bound to an identifier.*
- ***Type inference*** means that we have the option to decide whether to specify a type or not.
- A ***closure*** is a programming abstraction that is able to capture all the *free variables* defined in its context.
- A ***partial application*** consists in calling a function with fewer parameters than the ones declared in its signature.
  - The return value is simply a function that takes as input the remaining parameters.
- ***Tail recursion*** is a particular kind of recursion that can reuse the same stack frame for all the recursive function calls.
- ***Pattern matching*** is a powerful facility that allows to construct and deconstruct data types to control execution flows.

Benefits of Functional Programming

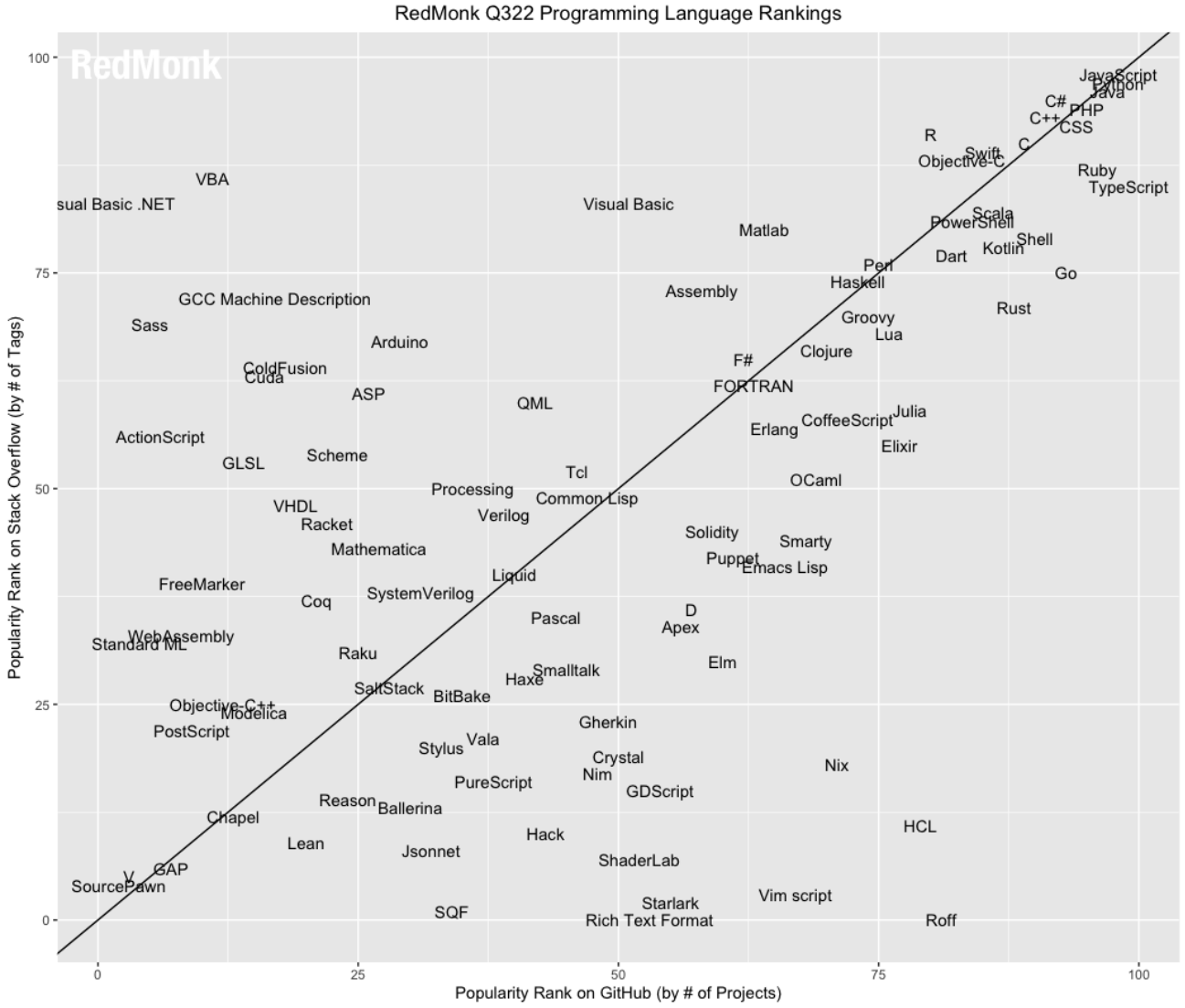
- Pure functions are easier to reason about.
- Testing is easier, and pure functions lend themselves well to techniques like property-based testing.
- Debugging is easier.
- Programs are more bulletproof.
- Programs are written at a higher level, and are therefore easier to comprehend.
- Function signatures are more meaningful.
- Parallel/ concurrent programming is easier.
- Additionally Scala is:
  - Concise, readable and also collection classes have functional APIs.
  - Scala runs on JVM using the wealth of libraries and tools alongside.

Disadvantages of Functional Programming

- Writing pure functions is easy, but combining them into a complete application is where things get hard.
- The advanced math terminology (monad, monoid, functor, etc.) makes FP intimidating.
- For many people, recursion doesn't feel natural.
- Because you can't mutate existing data, you instead use a pattern that I call, "Update as you copy."
- Pure functions and I/ O don't really mix.
- Using only immutable values and recursion can potentially lead to performance problems, including RAM use and speed.
- GUIs and Pure FP are not a good fit

Note: "ReactiveX is a combination of the best ideas from the Observer pattern, the Iterator pattern, and Functional programming. It is a library for composing asynchronous and event-based programs by using observable sequences"

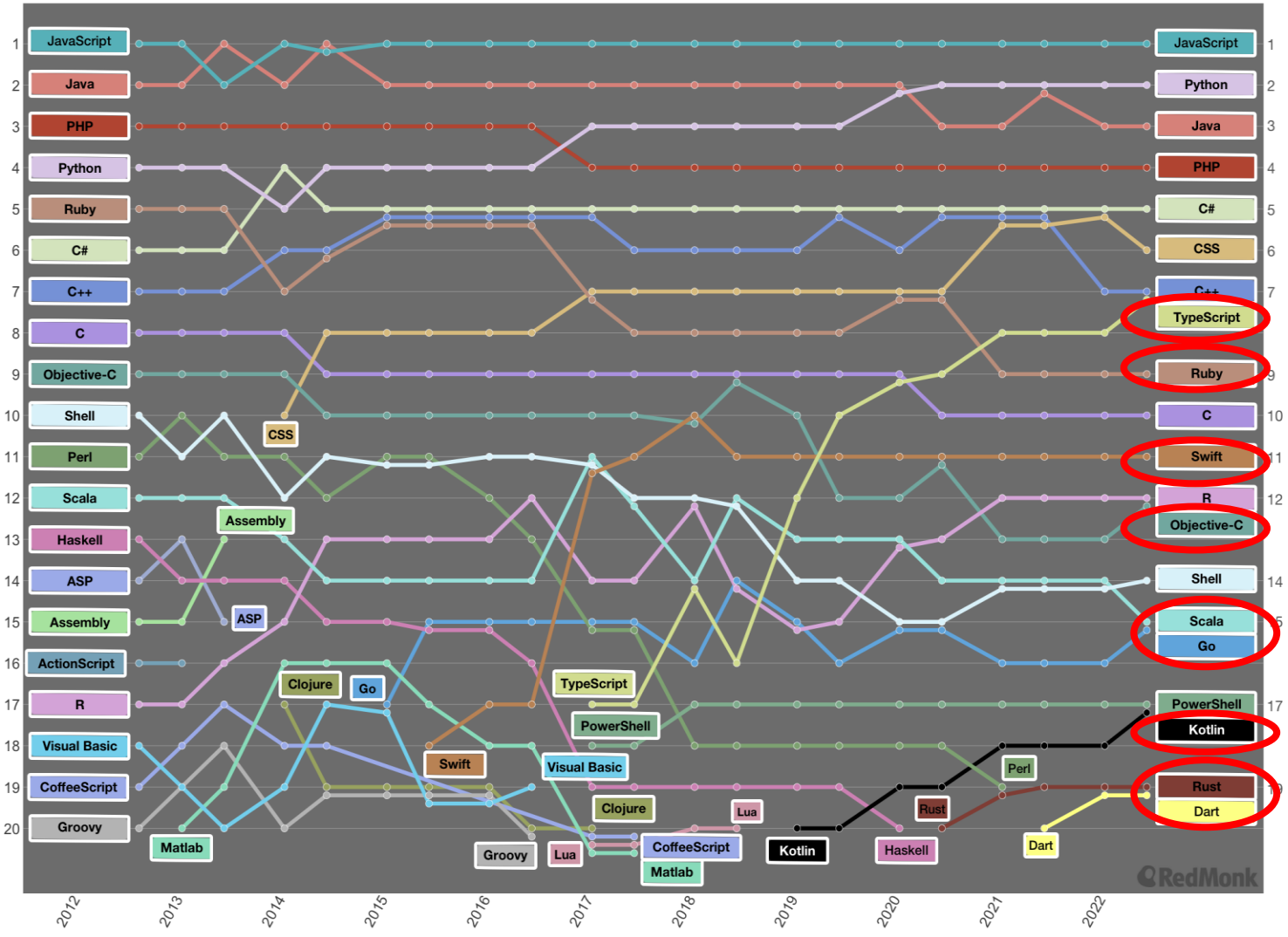
GitHub and Stack Overflow are used here because of their size and second because of their public exposure of the data necessary for the analysis.



https://redmonk.com/sogrady/2022/10/20/language-rankings-6-22/

RedMonk Language Rankings

September 2012 - June 2022

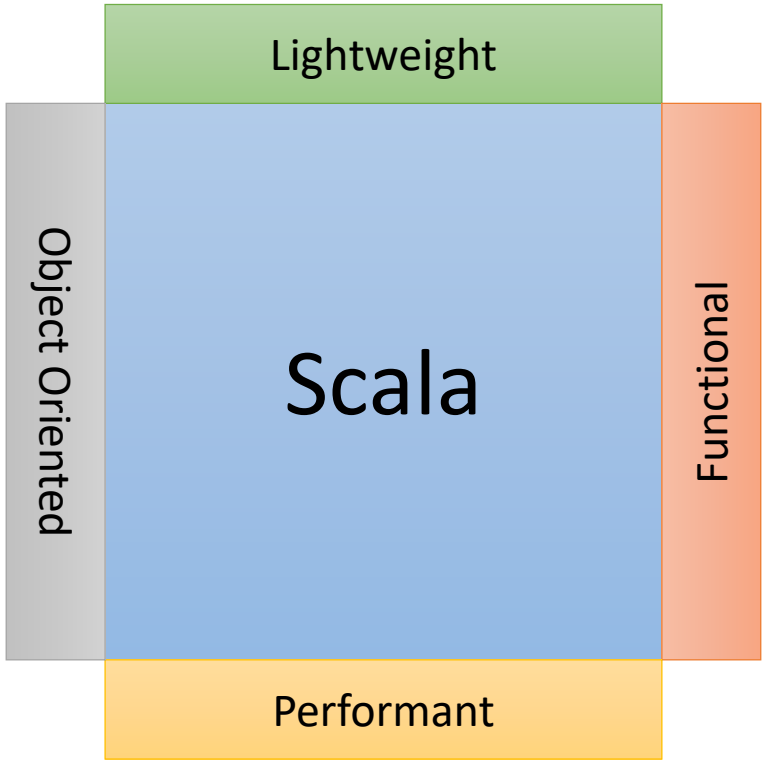


Ranking

1 JavaScript
2 Python
3 Java
4 PHP
5 C#
6 CSS
7 C++
7 TypeScript
9 Ruby
10 C
11 Swift
12 R
12 Objective-C
14 Shell
15 Scala
15 Go
17 PowerShell
17 Kotlin
19 Rust
19 Dart

Big Data Engineering For Analytics

Why Scala?



- Scala is a type-safe language that relies on the JVM runtime.
  - Scala is being used by numerous tech companies, including Netflix, Twitter, LinkedIn, AirBnB, AT&T, eBay, and even Apple. It's also used in finance, by the likes of Bloomberg and UBS.
- Scala supports lazy evaluation.
  - Scala is not backward compatible.
- Scala variables are by default immutable type.
  - In Scala, all the operations on entities are done by using method calls (there is no static).

Concise.. For example a class

... in Java:

```
public class Person {
  public final String name;
  public final int age;
  Person(String name, int age) {
      this.name = name;
      this.age = age;
  }
}
```

Demo Time

... in Scala:

```
class Person(val name: String,
             val age: Int) {}
```

Pattern Matched … For example

… in Java:

```
import java.util.ArrayList;
...
Person[] people;
Person[] minors;
Person[] adults;
{  ArrayList<Person> minorsList = new ArrayList<Person>();
   ArrayList<Person> adultsList = new ArrayList<Person>();
   for (int i = 0; i < people.length; i++)
       (people[i].age < 18 ? minorsList : adultsList)
         .add(people[i]);
   minors = minorsList.toArray(people);
   adults = adultsList.toArray(people);
}
```

A function value

An infix method call

… in Scala:

```
val people: Array[Person]
val (minors, adults) = people partition (_.age < 18)
```

A simple pattern match

What People Say of Scala?

If I were to pick a language to use today other than Java, it would be Scala."

- James Gosling, creator of Java

"Scala, it must be stated, is the current heir apparent to the Java throne. No other language on the JVM seems as capable of being a "replacement for Java" as Scala, and the momentum behind Scala is now unquestionable. While Scala is not a dynamic language, it has many of the characteristics of popular dynamic languages, through its rich and flexible type system, its sparse and clean syntax, and its marriage of functional and object paradigms."

- Charles Nutter, creator of JRuby

"I can honestly say if someone had shown me the Programming in Scala book by Martin Odersky, Lex Spoon & Bill Venners back in 2003 I'd probably have never created Groovy."

- James Strachan, creator of Groovy.

Understanding Python

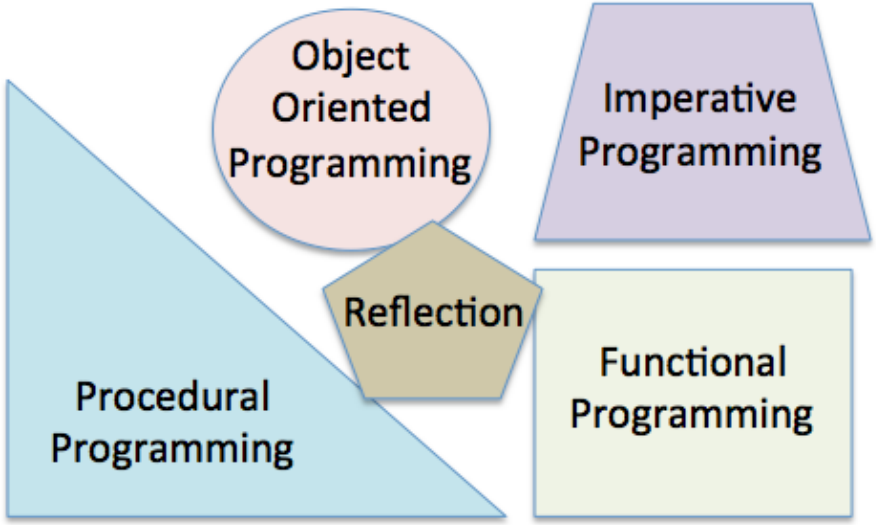
Python is not a purely functional language, and a strict definition isn't helpful. Instead, we'll identify some common features that are indisputably important in functional programming.

- Functional programming language compilers can optimize these kinds of simple recursive functions. **Python can't do the same optimizations.**
  - Functional programming defines a computation using expressions and evaluation; often these are encapsulated in function definitions.
  - It de-emphasizes or avoids the complexity of state change and mutable objects.
  - **Python is a hybrid** of imperative and functional programming; ***some parts of Python don't allow purely functional programming***; however we can attempt to create hybridized functional programming in Python.

(Avoid) Flow Control

- **Regular imperative constructs to be avoided**
  - Example a block of code generally consists of some outside loops (for or while)
  - Example assignment of state variables within those loops, modification of data structures like dicts, lists, and sets (or various other structures, either from the standard library or from third-party packages)
  - Example some branch statements (if/elif/else or try/except/finally)

Programming paradigms supported by Python

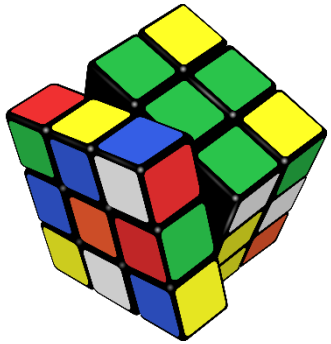


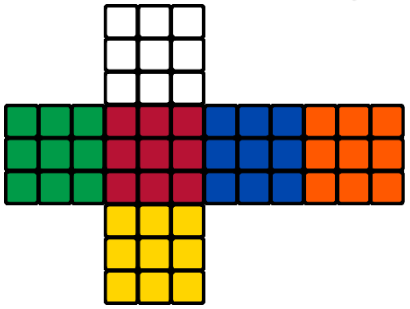
In Summary . . .

> *"Functional code is characterised by one thing: the absence of side effects."*

- Language ***features*** that aid functional programming are *immutable data, first class functions and tail call optimizations*.
- Programming ***techniques*** used to write functional code are *mapping, reducing, recursing, currying, and use of higher order functions*.
- Advantageous ***properties*** of functional programming are *parallelization, lazy evaluation and determinism*.
- A pure function has ***no side effects*** – meaning it doesn't rely on data outside the current function, and it doesn't change data that exists outside the current function.
  - Every other 'functional' thing can be derived from this property.

Reference: A Practical Introduction to Functional Programming by Mary Rose Cook.





References

"People think that computer science is the art of geniuses but the actual reality is the opposite, just many people doing things that build on each other, like a wall of mini stones."

– Donald Knuth (computer scientist)

Books You May Enjoy. . .

- Mastering Functional Programming by Anatolii Kmetiuk, Packt Publishing, August 2018, ISBN: 9781788620796
- Programming in Scala, 3rd ed, Updated for Scala 2.12, by **Martin Odersky,** Lex Spoon, Bill Benners
- Functional Programming in Scala, by Paul Chiusano, Rúnar Bjarnason, Manning
  - Scala for the Impatient, by Cay S. Horstmann, Addison-Wesley
  - Programming Scala, Updated for Scala 2.11, by Alex Payne, Dean Wampler, O'Reilly
- Scala in Depth, by Joshua D. Suereth, Manning
- Functional Python Programming - Second Edition, by Steven F. Lott, Published by Packt Publishing, 2018
- Functional Programming in Python by Martin McBride Published by Packt Publishing, 2019
- Functional Programming in Scala By Paul Chiusano and Rúnar Bjarnason, 2014