Principles
of
Programming Language

[BE SE-6th Semester]

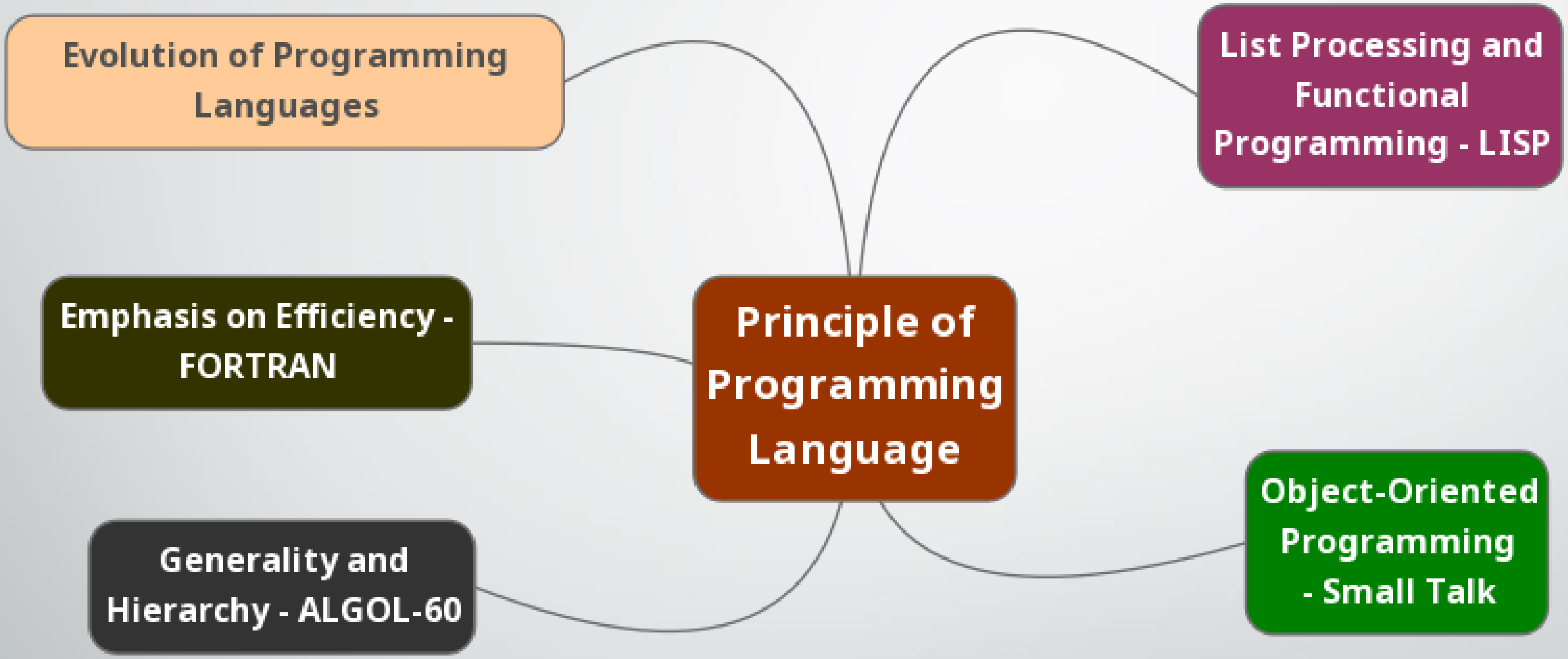
Rishi K. Marseni

Textbook:

Principles of programming languages: design, evaluation, and implementation.

Author: Bruce J. MacLennan

Principle of Programming Language


Principle of Programming Language

Evolution of Programming Languages

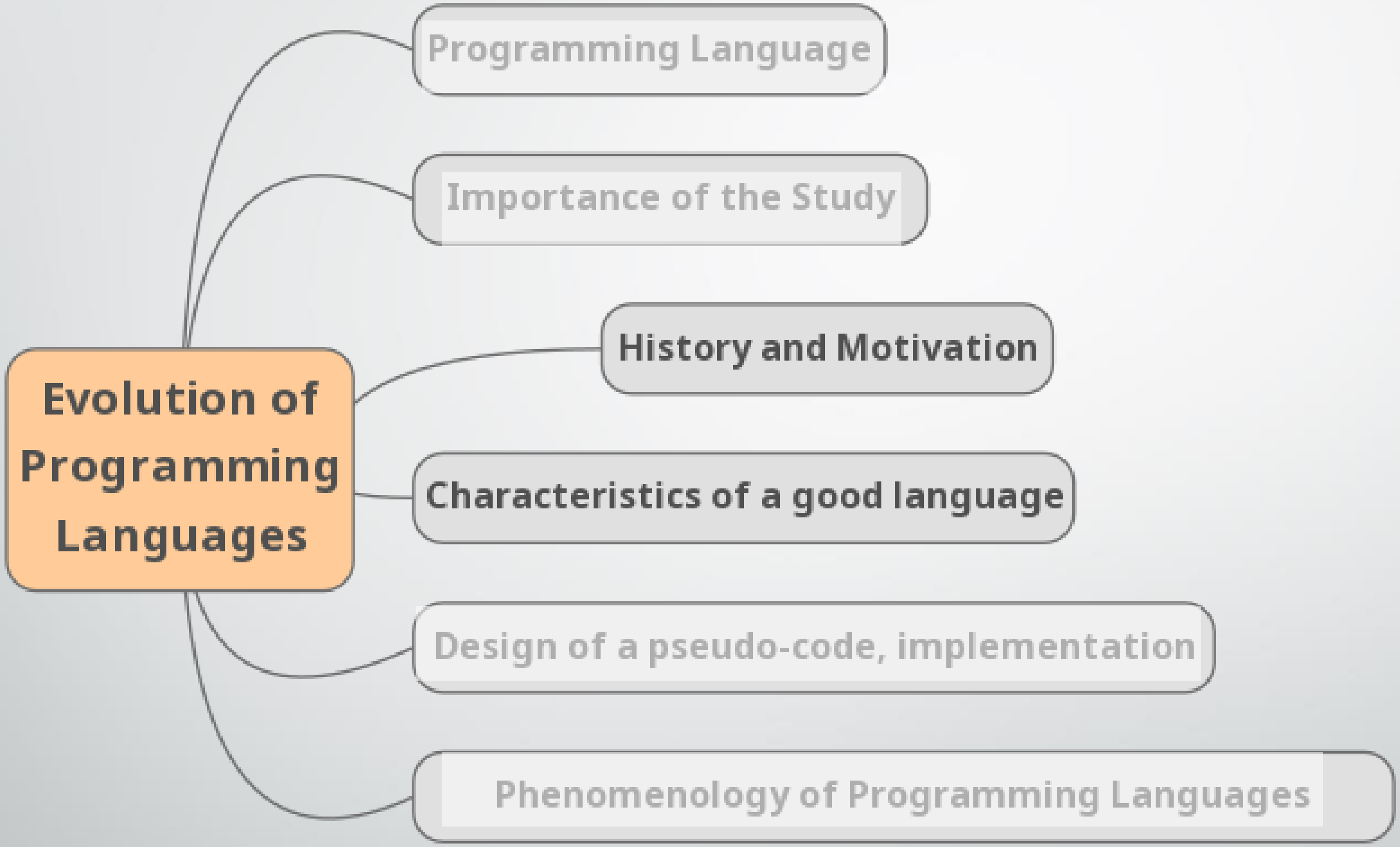
Emphasis on Efficiency - FORTRAN

Generality and Hierarchy - ALGOL-60

List Processing and Functional Programming - LISP

Object-Oriented Programming - Small Talk

Unit 1: Evolution of Programming Language


Evolution of Programming Languages
Programming Language
Importance of the Study
History and Motivation
Characteristics of a good language
Design of a pseudo-code, implementation
Phenomenology of Programming Languages

History of Programming Language

Art | Science | Engineering

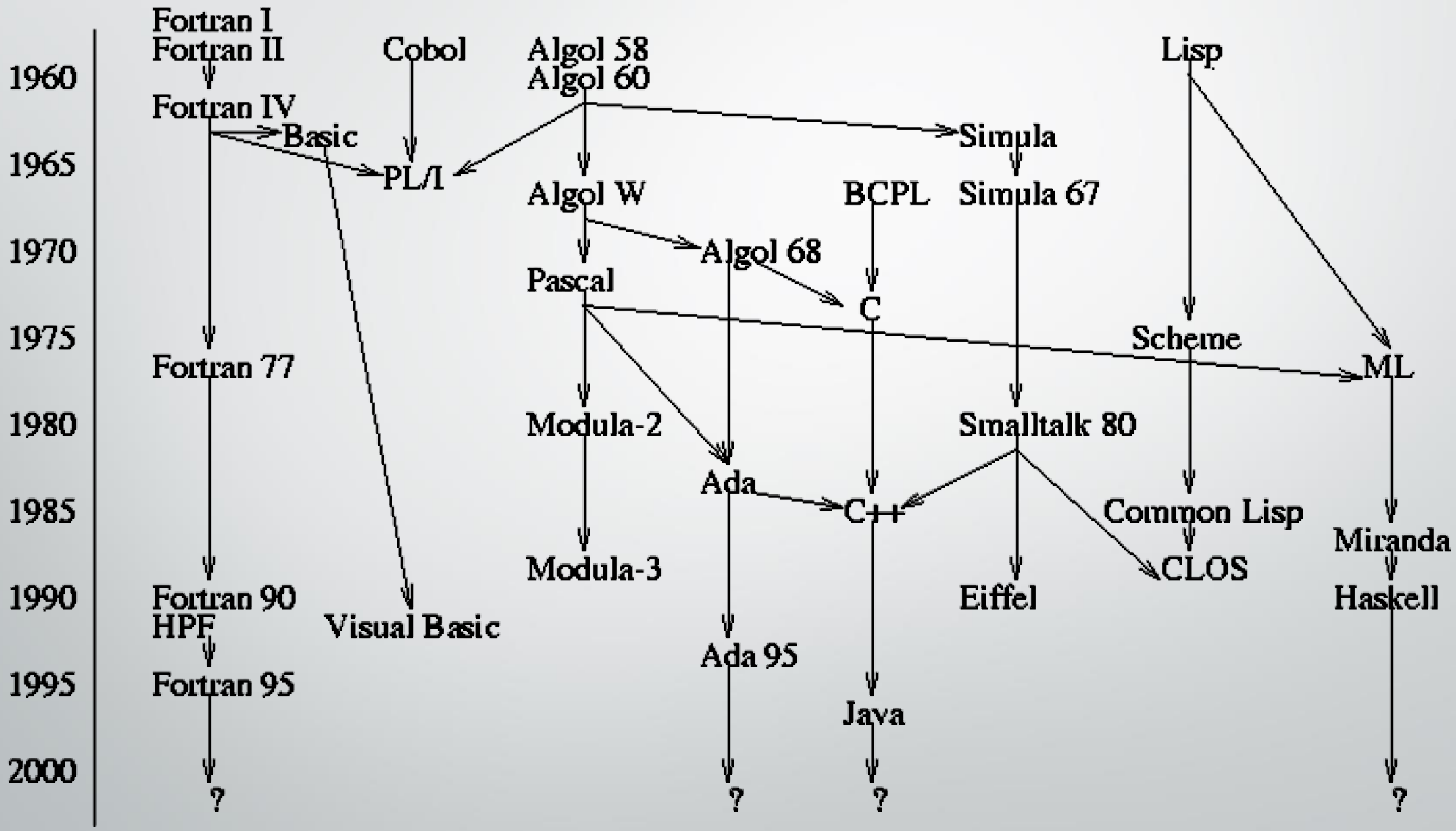
- Several ENIAC coding system
  { before 1950 }

- **Plankalkül** → The first high-level programming language
  { Konrad Zuse < 1942 – 1945 >}

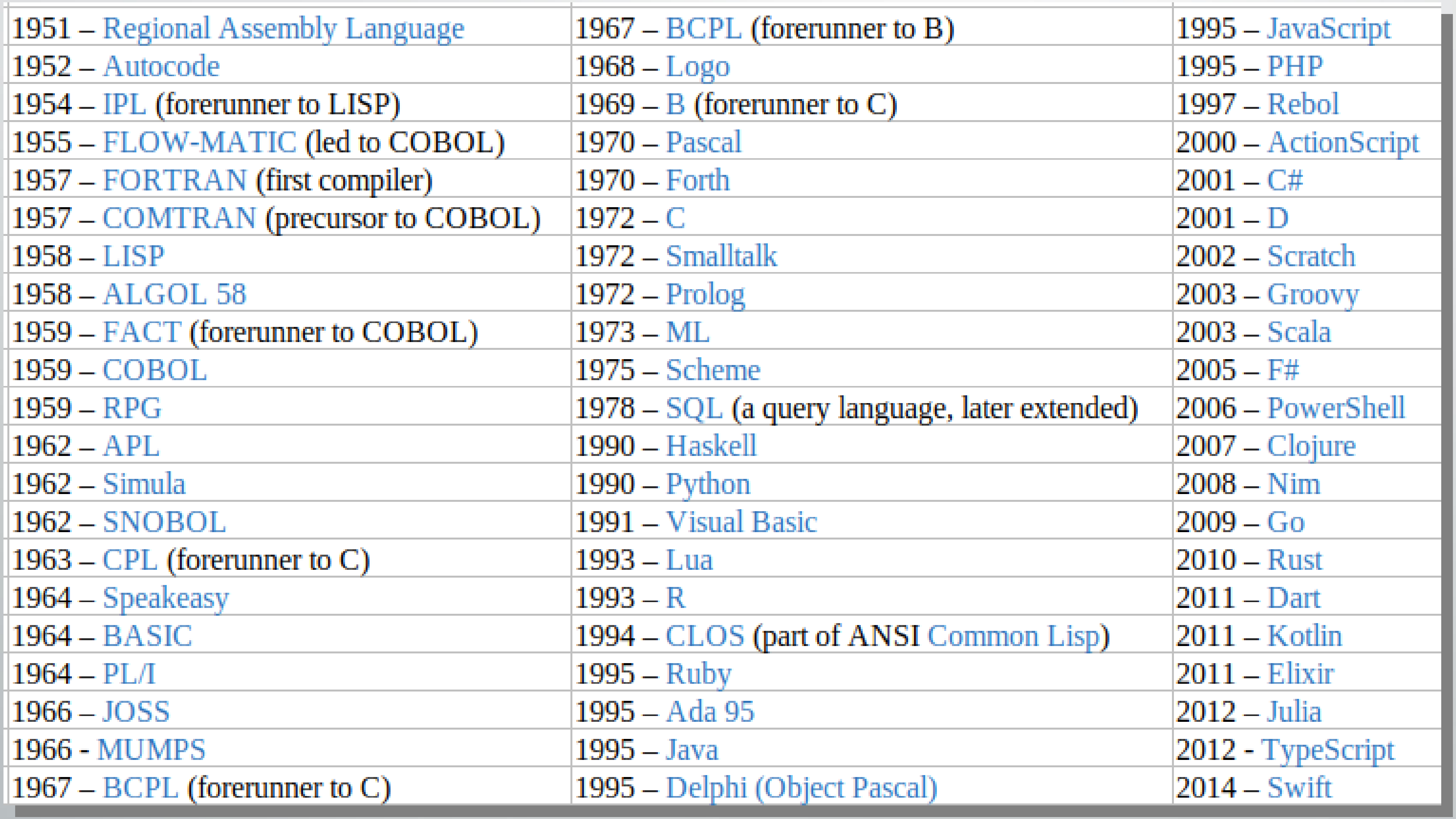
- The first high-level language with compiler
  { Corrado Böhm < 1951 >}

- **FORTRAN** → The first commercially available language
  { John Backus & his team @ IBM < 1954 - 1957 > }

Programming Language Genealogy



List of Programming Languages



1951 – Regional Assembly Language	1967 – BCPL (forerunner to B)	1995 – JavaScript
1952 – Autocode	1968 – Logo	1995 – PHP
1954 – IPL (forerunner to LISP)	1969 – B (forerunner to C)	1997 – Rebol
1955 – FLOW-MATIC (led to COBOL)	1970 – Pascal	2000 – ActionScript
1957 – FORTRAN (first compiler)	1970 – Forth	2001 – C#
1957 – COMTRAN (precursor to COBOL)	1972 – C	2001 – D
1958 – LISP	1972 – Smalltalk	2002 – Scratch
1958 – ALGOL 58	1972 – Prolog	2003 – Groovy
1959 – FACT (forerunner to COBOL)	1973 – ML	2003 – Scala
1959 – COBOL	1975 – Scheme	2005 – F#
1959 – RPG	1978 – SQL (a query language, later extended)	2006 – PowerShell
1962 – APL	1990 – Haskell	2007 – Clojure
1962 – Simula	1990 – Python	2008 – Nim
1962 – SNOBOL	1991 – Visual Basic	2009 – Go
1963 – CPL (forerunner to C)	1993 – Lua	2010 – Rust
1964 – Speakeasy	1993 – R	2011 – Dart
1964 – BASIC	1994 – CLOS (part of ANSI Common Lisp)	2011 – Kotlin
1964 – PL/I	1995 – Ruby	2011 – Elixir
1966 – JOSS	1995 – Ada 95	2012 – Julia
1966 - MUMPS	1995 – Java	2012 - TypeScript
1967 – BCPL (forerunner to C)	1995 – Delphi (Object Pascal)	2014 – Swift

“A computer without FORTRAN and COBOL is like a chocolate cake without mustard and ketchup - Internet wisdom

History : Plankalkül

- Between 1942-45, Concept proposed by Konrad Zuse
- Used to program his Z4 computer
- Introduced:
    - the assignment operation
    - if’s (but no else’s)
    - loops

History : FORTRAN

- 1954-57, Developed by John Backus for, Numeric computing
- Parameter pass by value and , Static allocation
- Separate compilation (because hardware failures were very frequent, length of a program could not exceed 300/400 lines) (FORTRAN II)
- Modularity (separately developed subprograms)
- Sharing of data among modules via a global environment
- Still in existence today, mostly in science/academia

History : Algol60

- 1958-60, Numeric computing, Descended from Fortran
- Stack allocation (Algol58), Stack dynamic variables
- Compound statements (group statements into one) (Algol58)
- BNF (Backus-Naur Form) was used to describe Algol60’s syntax
- Block structure, Block nesting with scope and Recursive procedures
- Spawned numerous other languages

History : LISP

- 1962, J. McCarthy
- Symbolic computing and AI (mostly in the US)
- Garbage collection, Heap allocation
- Father and mother of all functional languages
- Free from Von Neumanean notions of variables, assignments, goto's etc.
- One data structure available (and needed) – a list
- LISP interpreters are simple to write and difficult to execute

History : Simula67

- 1967, Developed by Ole-Johan Dahl and Kristen Nygaard
- Simulation problems
- Descended from ALGOL 60
- Object-oriented Programming
- Abstract data types
- Classes
- Inheritance

History : Smalltalk

- First appeared in 1972 and Smalltalk80 was first publicly available (published in 1980)
- Designed by Alan Kay, Dan Ingalls, Adele Goldberg
- Personal computing
- Descended from SIMULA 67 and LISP
- First full implementation of an object-oriented language
- First design and use of window-based graphical user interfaces (GUIs)

Motivation(1)

- Programming in early computers was especially difficult
- Programming is being easier these days
- Computer programming making the hardware as the Right tool for the several different kinds of job
- Programming Languages continue to grow with new features
- A programming language with sufficiently good features may improve the programmer productivity

Motivation(2)

- Area all languages equal? How small can a programming language be?
- Can we prove that a program is correct before we run it?
- What is the difference between a program and its data? Can programs write programs?
- How does the structure of a machine affect the computations it can perform?
- Are there functions beyond mechanical computation?

General principles of good programming

- **Correctness** → a *program should do what it is supposed to do*
- **Efficiency** → *should not waste computational resources(time| space)*
- **Transparency** → *should not be more complicated than necessary*
- **Modifiability** → *should be easy extend*
- **Robustness** → *should not crash even with wrong input*
- **Documentation** → *at least program with comments*

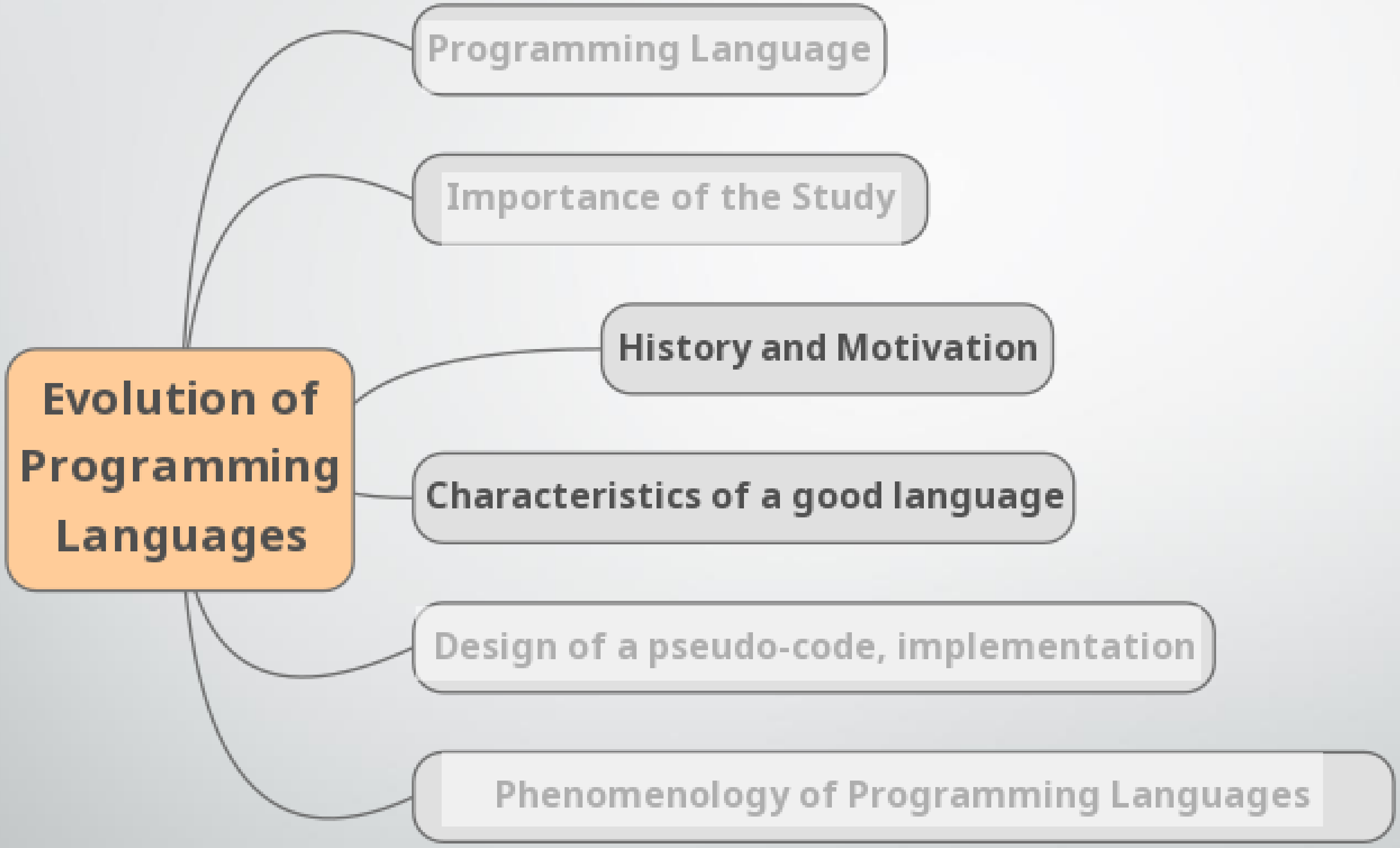
Characteristics of a good language(1)

- Clarity Simplicity and Unity
- Orthogonality
- Naturalness for the application
- Support for Abstraction
- Ease of Program Verification
- Programming Environment
- Portability of Programs

Characteristics of a good language(2)

- Cost of Use:
  - Cost of Program Execution
  - Cost of Program Translation
  - Cost of Program Creation, Testing and Use
  - Cost Of Program Maintenance

Unit 1: Evolution of Programming Language



Evolution of Programming Languages

- Programming Language
- Importance of the Study
- History and Motivation
- Characteristics of a good language
- Design of a pseudo-code, implementation
- Phenomenology of Programming Languages