Programming Paradigms

What is a Programming Language?

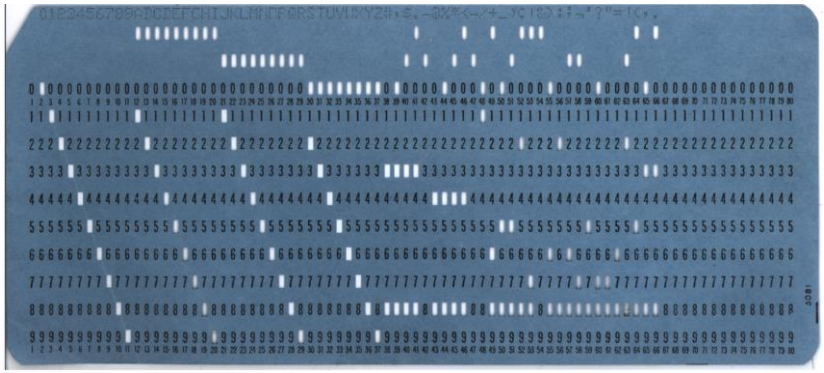
- Programming Languages 2nd edition Tucker and Noonan
- Online resources
  - https://www.learncomputerscienceonline.com/programming-paradigm/
  - https://www.baeldung.com/cs/imperative-vs-declarative-programming
  - https://web.cs.ucdavis.edu/~vemuri/classes/ecs170/lispintro.htm

What is a Programming Language?

- A PL is a system of notation for writing computer programs:
  - Most PLs are text-based formal languages, but they may also be graphical
- The description of a PL is usually split into the two components of syntax (form) and semantics (meaning), which are usually defined by a formal language.
  - Formal notation for specifying computations, independent of a specific machine
    - A factorial function takes a single non-negative integer argument and computes a positive integer result
    - Mathematically, written as fact: nat $\rightarrow$ nat

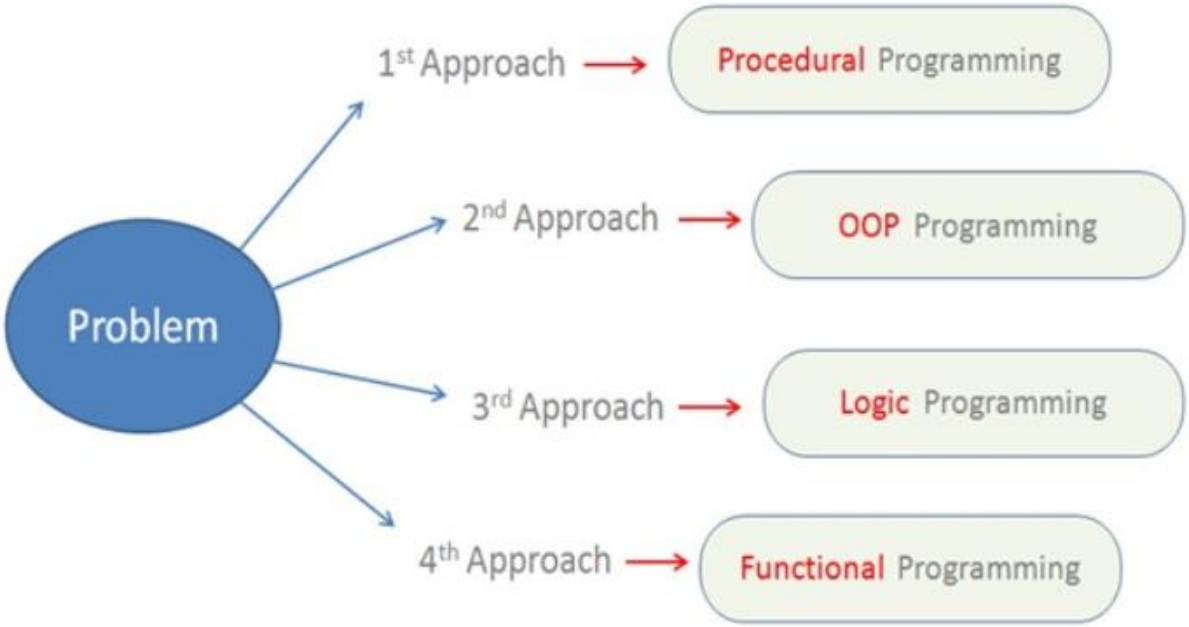
What is a Programming Language?

- Programming languages usually contain abstractions for defining and manipulating data structures or controlling the flow of execution.
  - The practical necessity that a programming language supports adequate abstractions is expressed by the abstraction principle.
  - This principle is sometimes formulated as a recommendation to the programmer to make proper use of such abstractions.



What is a Programming Language?

- A programming language is essentially a problem-solving tool.
  - For each problem there can be many solutions.
  - Each solution can adopt a different approach in providing solution to the problem.



What is a Programming Language?

- A programming language is essentially a problem-solving tool.
  - For each problem there can be many solutions.
  - Each solution can adopt a different approach in providing solution to the problem.
- The computer programs are written using different programming languages.
  - Each programming language has unique programming style that implements a specific programming paradigm.

Principal Paradigms

- The programming paradigm (PP) is all about the writing style and organizing the program code and data in a specific way.
  - Each paradigm advocates a specific way to organize the program code and data.

- A PP can be understood as an abstraction of a computer system, for example, the von Neumann model used in traditional sequential computers.
  - The program and its data (memory) are what is abstracted in a programming language and translated into machine code by the compiler/interpreter.

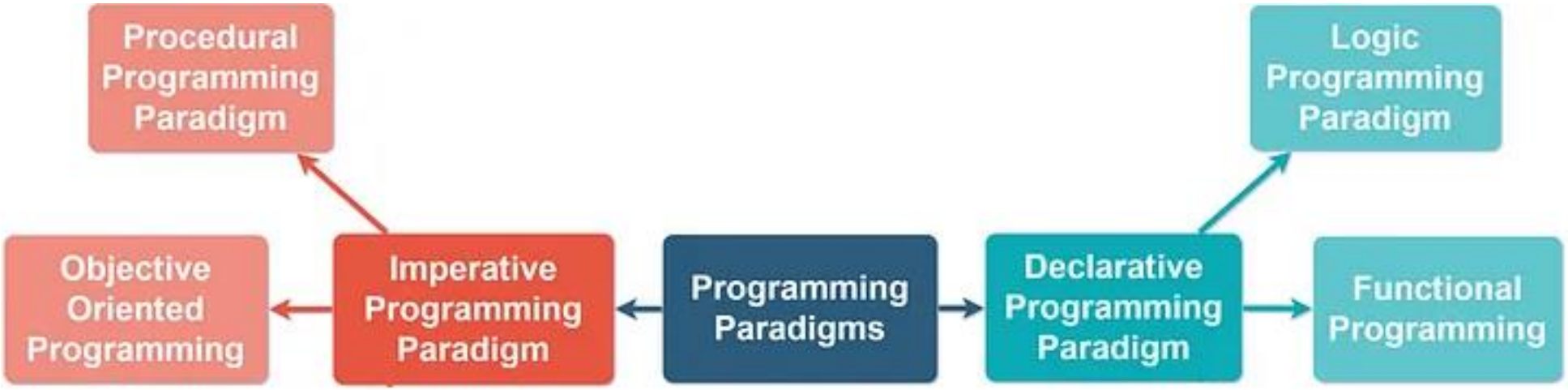
Principal Paradigms

- Programming paradigms are the result of people's ideas about how programs should be constructed
  - … and formal linguistic mechanisms for expressing them
  - … and software engineering principles and practices for using the resulting programming language to solve problems

- Compare with a software development methodology, which is a style of solving specific software engineering problems.

Principal Paradigms

- The programming paradigms can be classified into two main types. The paradigm type depends upon the programming language features and a particular style of organizing the program code (main paradigms):



Principal Paradigms

- Imperative Paradigm (IP)
  - Said to be command driven.
  - The program code in IP programming language directs the program execution as sequence of statements executed one by one.
    - Each statement directs the computer to perform a specific task.
    - The programmer has to elaborate each statement in details.
      - Each statement directs what is to be done and how it is to be done.

Principal Paradigms

- Imperative Paradigm (IP)
  - Said to be command driven.
  - The program code in IP programming language directs the program execution as sequence of statements executed one by one.
    - Each statement directs the computer to perform a specific task.
    - The programmer has to elaborate each statement in details.
      - Each statement directs what is to be done and how it is to be done.

Principal Paradigms

- Imperative Paradigm (IP)
  - The IP has several references in von Neumann's architecture.
  - This architecture takes into account the use of a processing unit, a control unit, and memory. In summary, the processing unit consists of an arithmetic module and processor registers, and the control unit has instruction registers and a program counter. The memory, in turn, keeps data and instructions.

Principal Paradigms

- The declarative paradigm (DP)
  - focused on the logic of the program and the end result.
    - Control flow is not the important element of the program.
    - The main focus of the declarative style of programming is achieving the end result.
  - DP approach to the programming is pretty much straight forward that leads to what is do be done.

Principal Paradigms

- IP vs DP
- Imagine that you have to invite your friend to your house.
  - In DP style, you simply give him your address and let your friend figure out how to reach your home.
  - In IP style, instead of giving him simply address, you would rather write down number of statements. Each statement would guide him step by step to reach your place.

Principal Paradigms

- IP vs DP

Paradigm	Imperative	Declarative
Operation	Defines HOW tasks should be accomplished	Defines WHAT tasks should be accomplished
Computation	Defines the control flow and states changes	Defines only the logic
Mutating Variables	Very usual	Unusual and not recommended
Advantages	Easy to learn notation; Machine architecture compliant	Easy to optimize codes; High abstraction level
Disadvantages	Hard debugging; Vulnerable to data race	Unfamiliar notation; Less customizable codes
Popular Derived Paradigms	Procedural; Object-oriented	Functional; Logic

Principal Paradigms

- There are four main programming paradigms:
  - Procedural
  - Object-oriented
  - Functional
  - Logic

- Very few languages are “pure”. Most combine features of different paradigms
  - The design goal of multi-paradigm languages is to allow programmers to use the best tool for a job, admitting that no one paradigm solves all problems in the easiest or most.

Procedural Paradigms

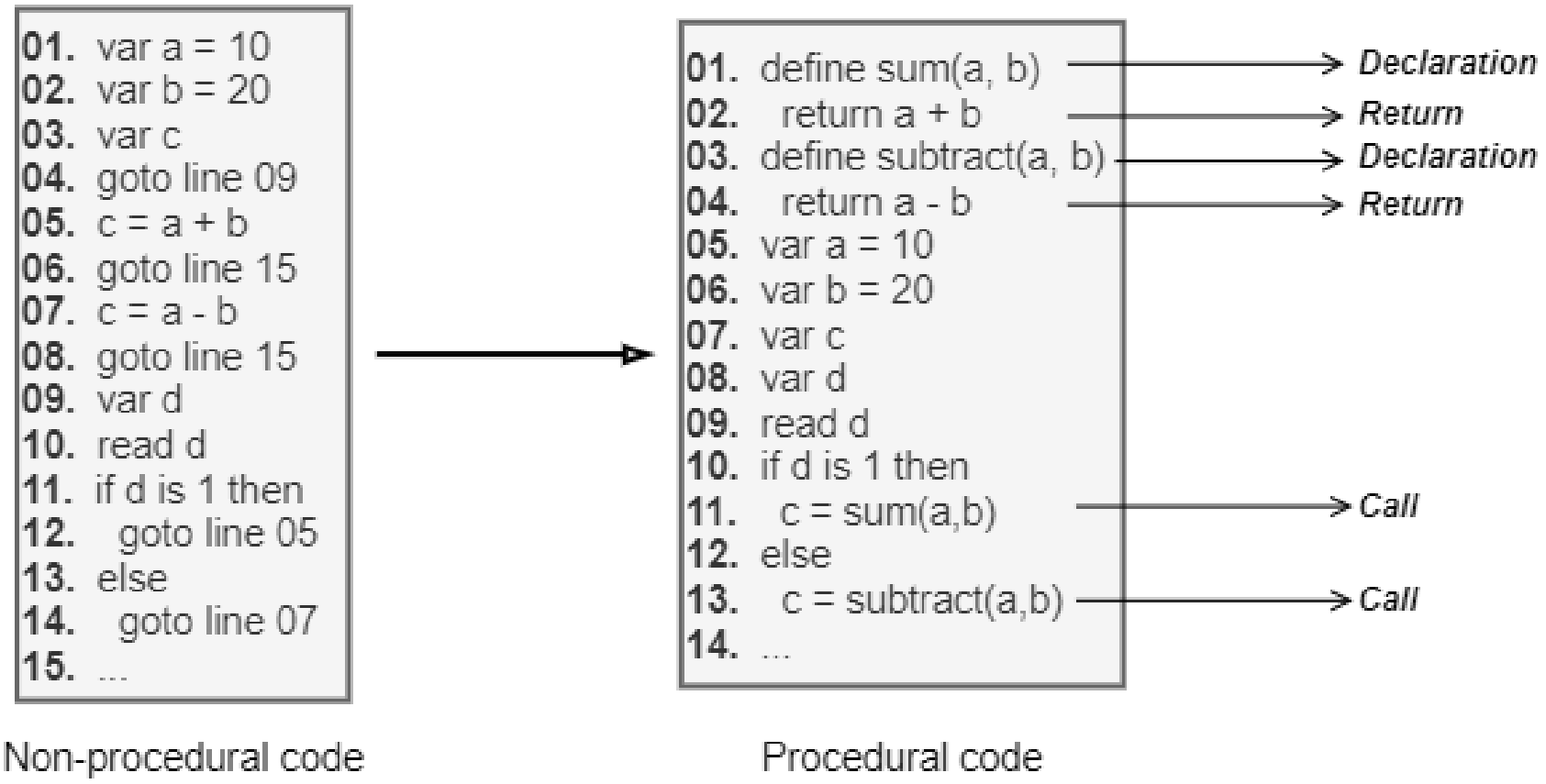
- Often thought as a synonym for imperative programming.
  - Specifying the steps, the program must take to reach the desired state.
  - Focused on subdividing a program from a simple sequence of instructions to a collection of procedure with particular instructions, structures, and variables.
    - Procedures, also known as routines, subroutines, methods, or functions that contain a series of computational steps to be carried out and tailored to accomplish a single well-defined task

Procedural Paradigms

- Often thought as a synonym for imperative programming.
  - Based upon the concept of the procedure call.
    - Any given procedure might be called at any point during a program's execution, including by other procedures or itself.
  - Using a procedural language, the programmer specifies language statements to perform a sequence of algorithmic

Procedural Paradigms

- Example of code



Procedural Paradigms

- Follows the classic von Neumann-Eckert model:
  - Program and data are indistinguishable in memory
  - Program = a sequence of commands
  - State = values of all variables when the program runs
  - Large programs use procedural abstraction

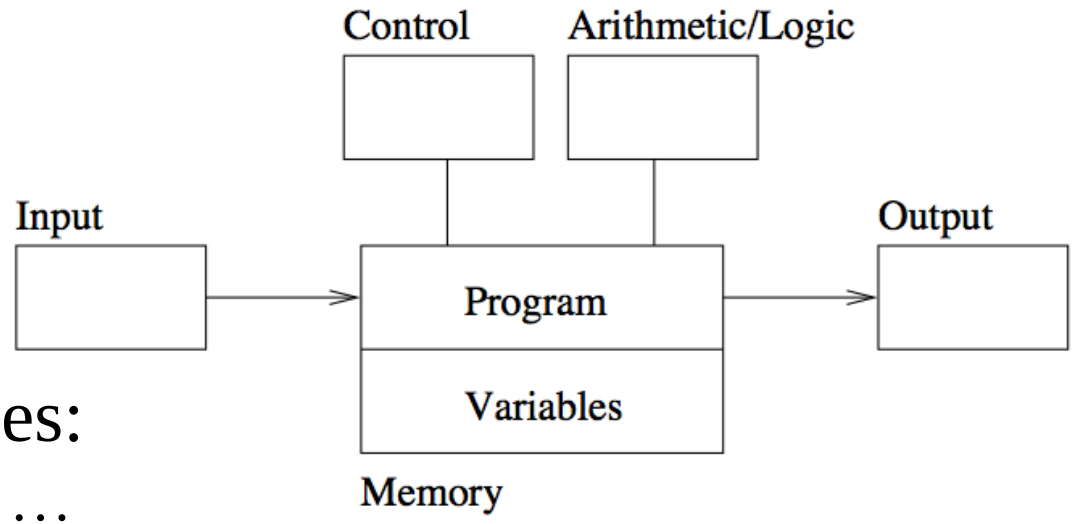


Figure 1.1: The von Neumann-Eckert Computer Model

- Example imperative languages:
  - Cobol, Fortran, C, Ada, Perl, …

Procedural Paradigms

- Possible benefits:
  - Often a better choice than simple sequential or unstructured programming in many situations which involve moderate complexity or require significant ease of maintainability.
  - The ability to reuse the same code at different places in the program without copying it.
  - An easier way to keep track of program flow than a collection of "GOTO" or "JUMP" statements (which can turn a large, complicated program into spaghetti code).
  - The ability to be strongly modular or structured.

Object-oriented programming paradigm

- Object-oriented paradigm (OOP) aims to represent the real world, modeling it similarly to our brains.
  - OOP use "objects" – data structures encapsulating data fields and procedures together with their interactions – to design applications and computer programs.
- The most important distinction is whereas procedural programming uses procedures to operate on data structures, object-oriented programming bundles the two together, so an "object" operates on its "own" data structure.
- Associated programming techniques may include features such as data abstraction, encapsulation, modularity, polymorphism, and inheritance.

Object-oriented programming paradigm

- An OO Program is a collection of objects that interact by passing messages that transform the state.
- When studying OO, we learn about:
  - Sending Messages
  - Inheritance
  - Polymorphism
- Example OO languages:
  - Smalltalk, Java, C++, C#, and Python

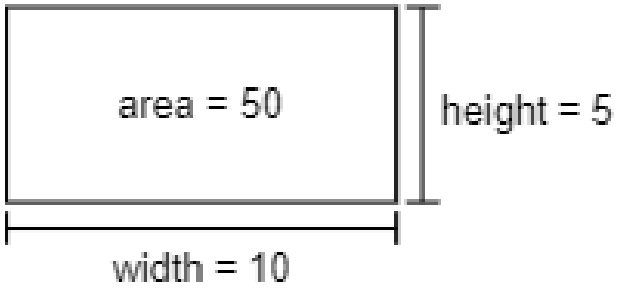
Object-oriented programming paradigm

- Example of code

```
01. class RECTANGLE
02.   var width = 0
03.   var height = 0
04.   define attributes(w, h)
05.     width = w
06.     height = h
07.   define area()
08.     return width * height
09. var rect
10. rect = new RECTANGLE
11. rect.attributes(10,5)
12. print rect.area()
13. ...
```

Class
Attribute
Attribute
Method
Method
Object

Object-oriented code



Functional Paradigm

- Functional programming models a computation as a collection of mathematical functions.
  - Input = domain
  - Output = range
- The desired result is declared as the value of a series of function applications
- Functional languages are characterized by:
  - Functional composition
  - Recursion
- Example functional languages:
  - Lisp, Scheme, ML, Haskell, …

Functional Paradigm

- Example of code
- In Imperative Programming Languages:

```
fact(int n) {
  if  (n <= 1)    return 1;
  else
      return n * fact(n-1); }
```

- In Functional Programming Languages: Scheme

```
(define (fact n)
  (if (< n 1) 1 (* n (fact (- n 1))))
))
```

Functional Paradigm

- We can employ functions as any data type.
  - Functions can have identifiers and work as arguments or results of other functions.
- They are different categories of functions, each one considering particular characteristics:
  - First-class functions: can be assigned to variables, passed as arguments to other functions, and returned as other functions result,
  - Pure functions: follow the properties of transparency and lack of side effects.
  - High-order functions: receive other functions as arguments or return a function as a result (or even both).

Functional Paradigm

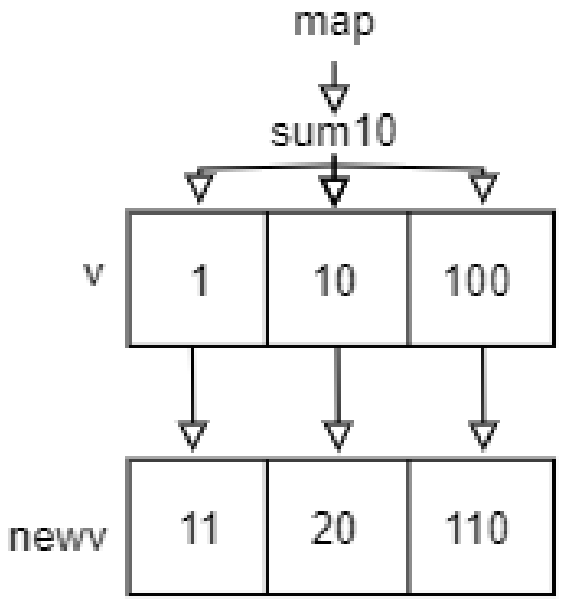
- Example of code

```
01. define sum10(a)
02.   return a + 10
03. def v
04. v = [1, 10, 100]
05. def newv
06. newv = map(sum10, v)
07. ...
```

Functional code

First-class and pure function

First-class, pure, and high-order function

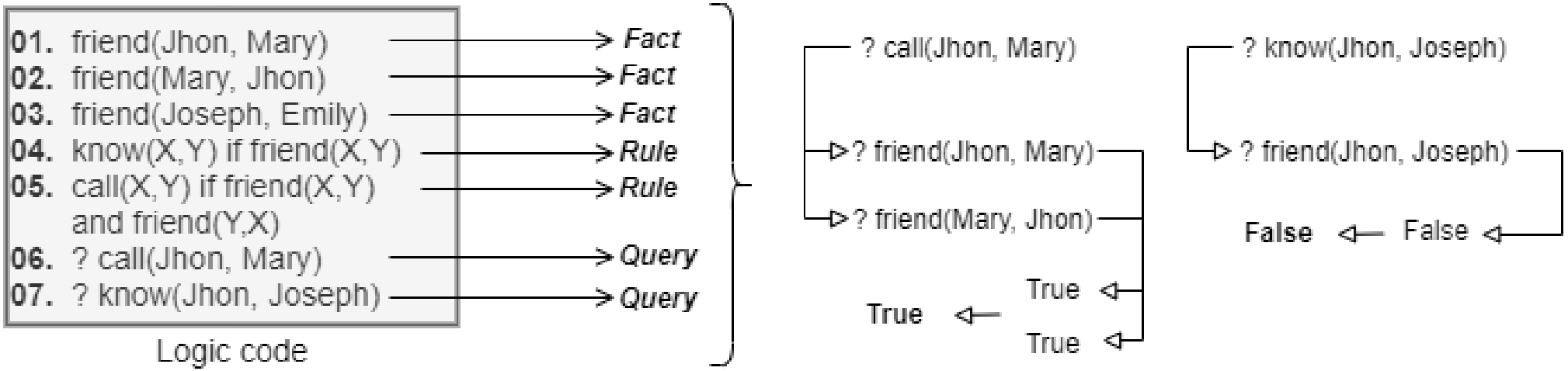


Logic Paradigm

- The logic programming paradigm uses formal logic to solve a myriad of problems. This paradigm relies on a knowledge base with several facts and rules to answer queries:
  - Facts: represent specific relations among objects. Concretely, facts are assertions about objects
  - Rules: they are predicates that describe a logical relation among facts. In practice, rules have head and body, where generally head is an assertion, and the body is a condition
  - Queries: a query consists of one or more logic expressions interleaved with logic operators.

Logic Paradigm

- Example of code



A Brief History

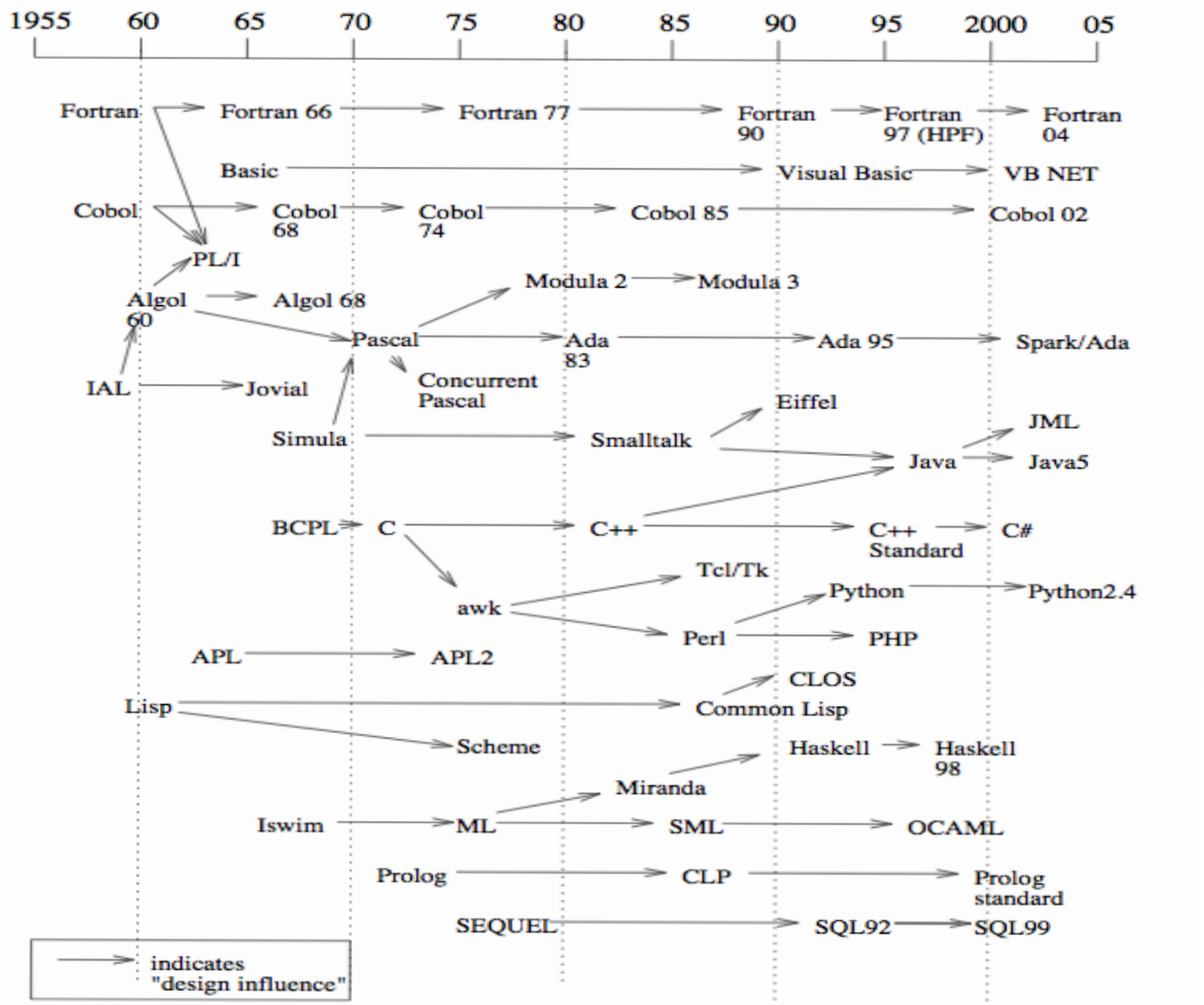


Figure 1.2: A Snapshot of Programming Language History

A Brief History

- What makes a successful language?
- Key characteristics:
  - Simplicity and readability
  - Clarity about binding
  - Reliability
  - Support
  - Abstraction
  - Orthogonality
  - Efficient implementation

A Brief History

- Simplicity and Readability
- Small instruction set
  - Is a smaller number of instructions better?
  - Simple syntax
- More than one way to accomplish a particular operation,
  - A single operation has more than one meaning
- Benefits:
  - Ease of learning
  - Ease of programming

A Brief History

- Clarity about binding
- A language element is bound to a property at the time that property is defined for it.
  - A binding is the association between an object and a property of that object
    - a variable and its type
    - a variable and its value
- Early binding takes place at compile-time
- Late binding takes place at run time

A Brief History

- Reliability

A language is reliable if:

- Program behavior is the same on different platforms
  - E.g., early versions of Fortran
- Type errors are detected
  - E.g., C vs Haskell
- Semantic errors are properly trapped
  - E.g., C vs C++
- Memory leaks are prevented
  - E.g., C vs Java

A Brief History

- Language support
- Accessible (public domain) compilers/interpreters
- Good texts and tutorials
- Wide community of users
- Integrated with development environments (IDEs)

A Brief History

- Abstraction in Programming
- Data
  - Programmer-defined types/classes
  - Class libraries
- Procedural
  - Programmer-defined functions
  - Standard function libraries

A Brief History

- Compilers and Virtual Machines
- Compiler – produces machine code
- Interpreter – executes instructions on a virtual machine
- Example compiled languages:
  - Fortran, Cobol, C, C++
- Example interpreted languages:
  - Scheme, Haskell, Python
- Hybrid compilation/interpretation
  - The Java Virtual Machine (JVM)

A Brief History

- Compilation process

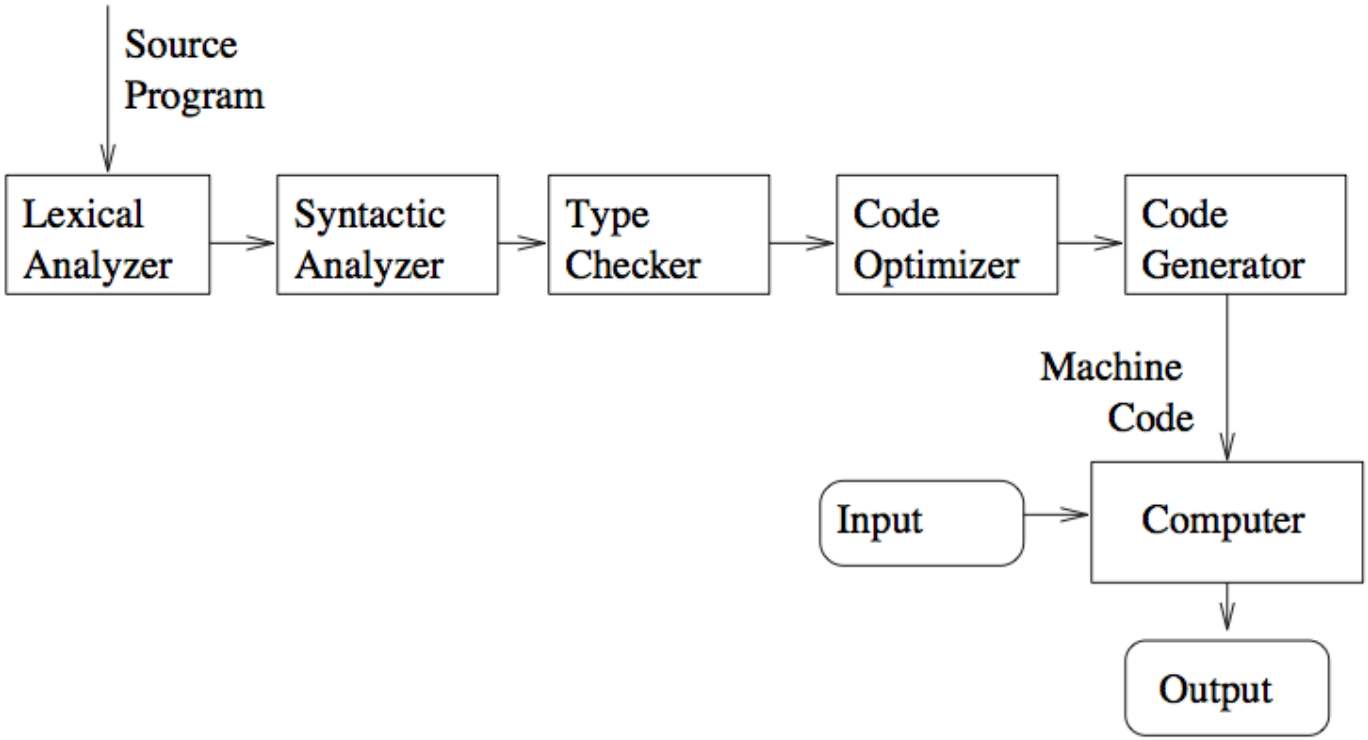


Figure 1.4: The Compile-and-Run Process

A Brief History

- Interpretation process

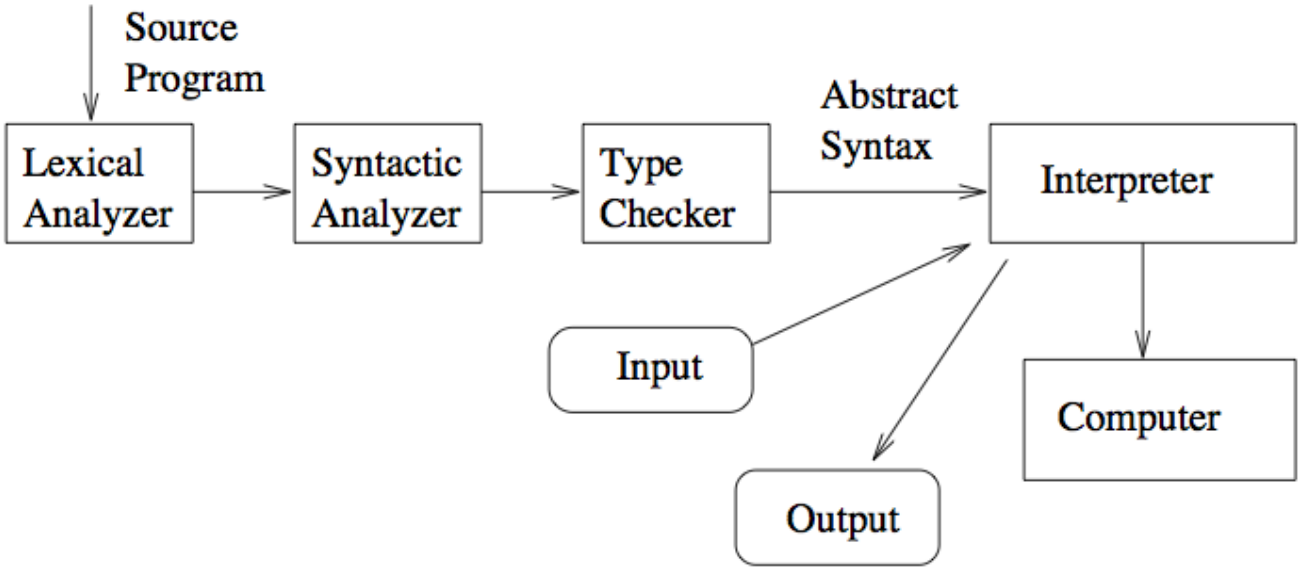


Figure 1.5: Virtual Machines and Interpreters