Programming Language Concepts
Language Paradigms. Functional Languages

Janyl Jumadinova

10 April, 2023

Programming Paradigms

Imperative: how to solve

- Procedural
- Object-Oriented

Programming Paradigms

Imperative: how to solve

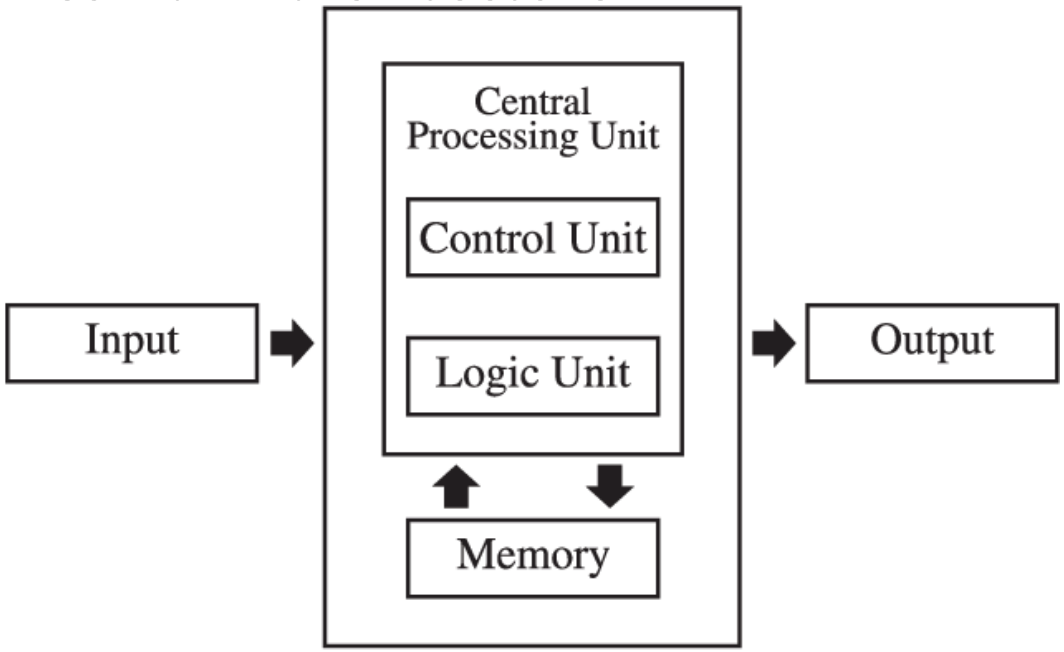
- Procedural
- Object-Oriented

Declarative: what to solve

- Functional
- Logical

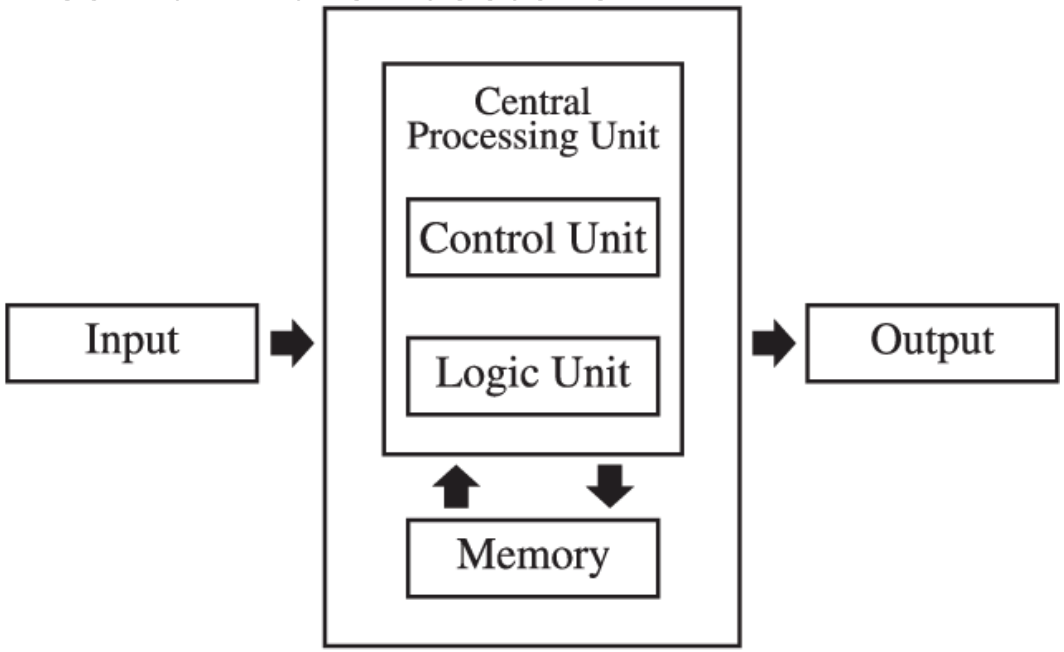
Imperative Languages

- The design of the imperative languages is based directly on the **von Neumann architecture**.



Imperative Languages

- The design of the imperative languages is based directly on the **von Neumann architecture**.



  - Efficiency is the primary concern, rather than the suitability of the language for software development.

Functional Languages

- The design of the functional languages is based on **mathematical functions**.
  - A solid theoretical basis that is also closer to the user, but relatively unconcerned with the architecture of the machines on which programs will run.

Mathematical Functions

A **mathematical function** is

a **mapping** of members of one set, called the **domain set**, to another set, called the **range set**.

Mathematical Functions

A **mathematical function** is

a **mapping** of members of one set, called the **domain set**, to another set, called the **range set**.

- A **lambda expression** specifies the parameter(s) and the mapping of a function in the following form:
  $\lambda(x)\ x * x * x$ for the function $cube(x) = x * x * x$

Lambda Expressions

- Lambda expressions describe nameless functions.
- Lambda expressions are applied to parameter(s) by placing the parameter(s) after the expression.
  e.g., $(\lambda(x)\ x * x * x)(2)$ which evaluates to 8

Functional Forms

A higher-order function, or **functional form**, is one that either takes functions as parameters or yields a function as its result, or both.

Functional Programming Concepts

- Functional languages such as Lisp, Scheme, FP, ML, Miranda, and Haskell are an attempt to realize Church's lambda calculus in practical form as a programming language

The key idea:

do everything by composing functions

- no mutable state
- no side effects

Function Composition

- A functional form that takes two functions as parameters and yields a function whose value is the first actual parameter function applied to the application of the second.

Function Composition

- A functional form that takes two functions as parameters and yields a function whose value is the first actual parameter function applied to the application of the second.

Form: $h \equiv f \circ g$

Function Composition

- A functional form that takes two functions as parameters and yields a function whose value is the first actual parameter function applied to the application of the second.

Form: $h \equiv f \circ g$ which means $h(x) \equiv f(g(x))$
For $f(x) \equiv x + 2$ and $g(x) \equiv 3 * x$,
$h \equiv f \circ g$ yields $(3 * x) + 2$

Apply-to-all

- A functional form that takes a single function as a parameter and yields a list of values obtained by applying the given function to each element of a list of parameters.

Apply-to-all

- A functional form that takes a single function as a parameter and yields a list of values obtained by applying the given function to each element of a list of parameters.

Form: α

Apply-to-all

- A functional form that takes a single function as a parameter and yields a list of values obtained by applying the given function to each element of a list of parameters.

Form: α
For $h(x) \equiv x * x$
$\alpha(h, (2, 3, 4))$ yields $(4, 9, 16)$

Fundamentals of Functional Programming Languages

- The objective of the design of a FPL is to mimic mathematical functions to the greatest extent possible.

Fundamentals of Functional Programming Languages

- The objective of the design of a FPL is to mimic mathematical functions to the greatest extent possible.
- The basic process of computation is fundamentally different in a FPL than in an imperative language.
  - In an imperative language, operations are done and the results are stored in variables for later use.
  - Management of variables is a constant concern and source of complexity for imperative programming.

Fundamentals of Functional Programming Languages

- The objective of the design of a FPL is to mimic mathematical functions to the greatest extent possible.
- The basic process of computation is fundamentally different in a FPL than in an imperative language.
  - In an imperative language, operations are done and the results are stored in variables for later use.
  - Management of variables is a constant concern and source of complexity for imperative programming.
- In an FPL, variables are not necessary, as is the case in mathematics.

Fundamentals of Functional Programming Languages

Referential Transparency

In an FPL, the evaluation of a function always produces the same result given the same parameters.

Fundamentals of Functional Programming Languages

Referential Transparency

In an FPL, the evaluation of a function always produces the same result given the same parameters.

Tail Recursion

Writing recursive functions that can be automatically converted to iteration.

λ in Python

Python adopted some functional concepts
(*map*(), *filter*(), *reduce*(), *lambda*)

lambda $x : x$

- `lambda`: keyword
- x: bound variable
- x: body