**History of programming languages**

First programming languages include the seeds for the programming language models

* Functional, imperative, object-oriented, modular

**Charles Babbage**

The origins of the digital programmable computer. In 1837 he has designed Analytical engine, the first mechanical computer that led to the design of more complex electronic computers.

**Analytical engine** - mechanical computation of arithmetic operations, branching, loops were the results of the previous computation are taken as an input to the following iterations.

**Turing-complete machine** - mathematical model of computation that defines an abstract machine that manipulates symbols on a strip of tape according to a table of rules. Despite the model's simplicity, given any computer algorithm, a Turing machine capable of implementing that algorithm's logic can be constructed.

**Konrad Zuse**

-The first electro-mechanical computer

-First programming language **Plankalkuel** (loops, no branching, sequence of arithmetical operations)

**ENIAC** (Electronic Numerical Integrator and Computer)

-First general-purpose electronic computer

-Programming

* 10 digit arithmetic, accumulators, general bus
* Complex functions, branching, loops, procedures

**Electronic Computers**

-ASCC (Automatic Sequence Controlled Calculator)

-Design, architecture

• Decimal, 60 sets of 24 switches for data input, 72 numbers

• ~2000 counters, 72 adding machines,

• Punched paper tape, program, data

• Instructions, execute current, read next

• Loops, paper tape attached to the beginning

• Branching, manual, later electronic (1946)

• 800km wire, 5 tons, 5KW power

-von Neuman architecture

-Principal abstraction, representation of the basic function of computers

• central processing unit

• main memory and

• input/output devices

**First compiler**

-A-0 programming language - sequence of subroutine calls together with the parameters

-The compiler was merely a linker and a loader

**FORTRAN** - Mathematical **FOR**mula **TRAN**slating System

-First complete compiler

-Constructs of Fortran – DO, GOTO, IF, SUBRUTINE, CALL

-Numeric processing

**FLOW-MATIC** – Business Language version 0 or B-0

-First language that made a clear distinction between the data definition section and the code section

-Strong influence on the development of the programming language COBOL.

**Information Processing Language**

-Assembly language - operations manipulate lists

-IPL computer

– Set of symbols, set of cells, set of primitive functions

– Functions defined on the cells and lists

– primitive PL run-time environment

-AI language

- Abstract and suitable for expressing AI programs

-Programs are not efficient

**Algol**

-Originally: Algebraic Language or IAL

-Formal semantics, two parameter passing methods: cbv and cbr, code blocks, nested functions, lexical scope

-– Imperative effect of cbr to lambda calculus studied

-Influence on: Pascal, C, Modula, Java, C++

**Lisp**

-Based on lambda calculus

-Lists used for the representation of programs and data

-New concepts

– Automatic storage management, dynamic typing, selfhosting compiler, and others

-AI language (Second-oldest programming language that is still in use)

**Cobol**

-Programming language for business applications

-Imperative and procedural, object-oriented (since 2002)

**Pascal**

-Structural programming language (Algol family)

-Data structures, pointers, arrays, variable records, …

-Delphi: a nice programming environment

**C Programming Language**

-Close to hardware, system programming

-Arrays and references are closely related

**Models of programming languages**

**Imperative programming languages**

-Rooted in assembly language

--Instructions, procedures, loops, branching

– Program is a sequence of instructions

– The outcome is built in the execution of instructions

– Instruction changes the contents of the main memory

-Abstract syntax added (Variables, IF, LOOP, FOR, PROCEDURE, FUNCTION)

-Pascal, C, Ada, Modula

**Functional languages**

-Rooted in logic (lambda calculus)

-LISP – implementation of lambda calculus

-Meta-Language

-Haskel, OCaml, Ruby, Scala

**Object oriented languages**

-Simula – first OO language

-Simulation programming environment

-It included everything recent OO languages have

-Smalltalk – one of the most famous

-Everything is object! Every object belongs to one class.

-Class is prototype + class is object

-Complex inheritance hierarchies

-C++, Java, Objective C, Eiffel, Ruby, Scala, …

**Script languages**

-Web programming language

-Can use resources of Webserver

-JSP, PhP, JavaScript, ASP, …

-System programming languages

-Strong ties to OS

-C, Awk, Perl, Python (general-purpose programming language)

-System integration language

**High-level languages**

-Information systems

-UML – designing and specification of information systems

-PL/SQL, SQL3 – abstract languages, working with tables

-Electronics

-Designed and specification of electronic circuits

**Logic programming languages**

-Predicate calculus

-Horn clauses, unification, resolution

-Strong, abstract and simple language

-Back-tracking

-Database programming languages – Datalog

-Sictus, SB Prolog, SWI Prolog

**Concepts of programming languages**

-Programming language is a tool, defined in the form of a language, that is used for design and implementation of computer programs.

-Concepts of programming languages are abstractions used for the representation of the structure and behavior of modeled systems.

**Abstraction** - A specific operation of the intellect consisting in detaching some properties and retaining some other properties of a thing

-Integrated into the language

Values, variables, branch, iteration, function, procedure, parameter passing, function composition, recursion, higher-order function, polymorphism, object, method, class, abstract class, specialization, aggregation, classification, module, functor, etc.

-Define language model

-- Imperative languages: variables, loops, procedures

– Functional languages: functions, composition, recursion

– Object languages: objects, methods, classes

Algorithmic and data abstractions

– Algorithmic abstractions

• Imperative: sequence, loop, branch, blocks, etc.

• Functional: function, recursion, polymorphism, etc.

– Data abstractions

• Simple: integer, bool, real

• Structured: n-tuple, list, array, dictionary, set, recursive data structures, etc.

– Algorithmic and data abstractions

• Objects & classes: encapsulation, classification, inheritance, etc.

• Modules: abstract data types, functions, abstract values

Implementation of the language concepts

– Function

• Function call, parameter passing, activation records

– Recursion

• Stack of activation records

– Variables

• Symbol tables, value/reference models, namespaces

– Memory representation of data structures

• Lists, n-tuples, records, unions, objects, classes, modules

– Objects and classes

• Static/dynamic binding, inheritance

– Memory management

• Stack, heap

Properties of abstractions in PL

Closer to hardware, simpler abstractions

– Basic types: integer, real, string, etc.

– Imperative language: loops, functions, procedures, etc.

– Usually very efficient languages

Higher level abstractions

-Objects, classes, modules, functors

General high-level programming languages

-Include multiple levels of abstractions

-C++, C#, F#, OCaml, Java

-General applicability

• Financial analyses, numerical calculations (Fortran, Ocaml)

• System programming (C++, Rust, Swift, Python, Ocaml)

• Information systems (C#, F#, Java)

**Meta-Language ML**

**Lambda calculus** - fundamental formalism for studying programming languages.

Theory of programming languages

– Using rule-based reasoning for:

• deriving expression types,

• evaluation of expressions,

• Proving the properties of language, etc.

– Static and dynamic semantics of PL

– Proving the properties of languages (determinism, strict typing, termination, etc.).

**Meta-Language**

-For LCF (Logic for computable functions) system

-Automatization of logic reasoning

– Type systems

– Notation for programs

– Higher-order functions

– Programs that search for proof

-Success story from functional branch of programming languages

– Standard ML, SML/NJ, Alice ML, Ocaml, Moscow ML, Mlton, MLKit, SML.NET, etc.

**Objective Caml**

-Objective Caml is reference language

– Caml core is pure lambda calculus.

– Theoretically well-studied language.

– Caml has strong typing.

– Imperative constructs.

– Parametric polymorphism

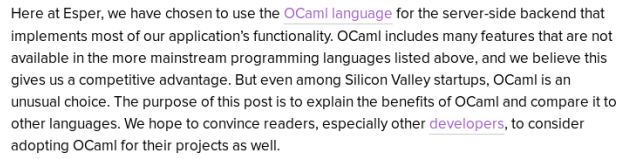
– Nice object model

– Modules and functors

- Ocaml offers more than one programming models

– Imperative + functional + object-oriented + modular programming model

– Also other PLs: C#, Rust, C++, Java, Erlang



**F#**

-F# is like C#

– .NET language

– F# is based on OCaml

– C# is C-like language, similar to Java and C++ …

• Features

– Lightweight syntax

– Immutable by default

– Type inference and automatic generalization

– First-class functions

– Powerful data types

– Pattern matching

– Async programming