

- Adapted from [Revision Guide \(revised 2017\)](#)

I. Introduction and motivation

- design, methods, paradigms; Foundations; Standardisation.
- [y2006p6q7 \(a\)](#)
 - motivating application domains, abstract machines, theoretical understanding
- [y2012p3q6 \(a\)](#)
 - Execution models (abstract machines)
 - Storage allocation and deallocation
- [y2015p3q5 \(a, b\)](#)
 - Programming-language concepts, innovations, influences
- [y2007p6q7 \(b,c\)](#)
 - Parameter passing, value, reference, value/result, name
 - Aliasing

II. FORTRAN: Sequential procedural language

- FORTRAN 77; Data types; Control structures; Syntax; Storage; Aliasing; Parameters.
- [y2010p3q5 \(a\)](#)
 - FORTRAN vs Pascal
- [y2009p3q2 \(a\)](#)
 - LISP vs FORTRAN
- [y2006p6q7 \(b\)](#)
 - Types, advantages and disadvantage
- [y2007p5q7 \(a\)](#)
 - Execution model (or abstract machine)
 - Compilation

III. LISP: Declarative, Functions, recursion, and lists

- Programming-Language phrases; S-expressions; quote; Abstract machine; Recursion; Programs as data; Reflection
- [y2009p3q2 \(a\)](#)
 - LISP vs FORTRAN
- [y2011p3q6 \(a.ii\)](#)
 - LISP vs Smalltalk
- [y2006p6q7 \(c\)](#)
- [y2014p3q6 \(a\)](#)
 - Static and Dynamic scope

- [y2007p6q7 \(a\)](#)
 - Execution model (or abstract machine)
 - Compilation
- [y2007p5q7 \(b\)](#)
 - Garbage collection
- [y2008p6q7 \(a\)](#)
 - Static (renaming principle) and Dynamic scope

IV. Algol, Pascal: Block-structured procedural languages

- Block structure; Algol 60; Recursion; Stack; Type system; Algol 68; BNF syntax; Heap; Garbage collection; Quasi-strong typing;
- [y2011p3q6 \(a.i\)](#)
 - Algol and SIMULA
- [y2013p3q6 \(a.i\)](#)
 - Algol and Pascal
- [y2010p3q5 \(a\)](#)
 - FORTRAN vs Pascal
- [y2006p6q7 \(b\)](#)
 - Types, advantages and disadvantage
- [y2008p5q7 \(a\)](#), [y2013p3q6 \(b\)](#)
 - Parameter-passing: pass-by-reference, pass-by-value/result
- [y2009p3q2 \(c\)](#)
 - call-by-value vs call-by-reference
- [y2012p3q6 \(c\)](#)
- [y2007p5q7 \(c\)](#)
 - Algol 60 primitive static type system, Parameter-passing
- [y2015p3q5 \(c\)](#), [y2008p6q7 \(b\)](#)
 - Pascal variant records vs ML vs subclass

V. SIMULA, Smalltalk: Object-oriented languages

- Subtyping vs. inheritance; SIMULA; Classes, objects and activation records; Subclasses and inheritance; Smalltalk; Dynabook; Syntax; Abstraction; Messages; Methods; Instance variables; Interfaces as types; Subtyping.
- [y2010p3q5 \(b\)](#), [y2013p3q6 \(a.ii\)](#)
 - SIMULA vs Smalltalk
- [y2011p3q6 \(a.i\)](#)
 - Algol and SIMULA, LISP and Smalltalk
- [y2008p5q7 \(c\)](#)
 - Objects in SML (see SML module)
- [y2006p6q7 \(d\)](#)

- Dynamic lookup; Abstraction; Subtyping; Inheritance
- [y2007p6q7 \(d\)](#)
 - SIMULA, Type checking and subtyping
- [y2012p3q6 \(f\)](#)
 - Abstraction, `private` weakened by pointer / reflection

VI. Types

- Type checking in SML; Type equality; Type declarations; let-polymorphism; Polymorphic exceptions.
- [y2009p3q2 \(b\)](#)
 - weakness of type system in any languages
- [y2010p3q5 \(c\)](#)
 - Type checking vs Type inference
- [y2012p3q6 \(b\)](#)
- [y2015p3q5 \(d,e\)](#)
- [y2008p5q7 \(b\)](#)
 - static vs dynamic scoping, early LISP
 - static vs dynamic type checking
 - type-safe and counterexample
- [y2011p3q6 \(b\)](#), [y2013p3q6 \(c\)](#)
 - Type inference in SML, constraint
- [y2012p3q6 \(e\)](#)
 - Type safety and counterexample
 - Polymorphism in ML
- [y2014p3q6 \(c\)](#)
 - Exception
- [y2016p3q5 \(d, e\)](#)
 - Java covariant arrays, invariant Generics
- [y2015p3q5 \(e, f\)](#)
 - downcast

VII. Scripting Languages – JavaScript

- Scripting vs. dynamic typing; JavaScript; Prototypal inheritance; Browser integration.
- e.g.,
 - “Scripting languages and dynamically typed languages are identical; discuss”
 - “Discuss the notion of ‘class’ in relation to JavaScript”

VIII. Data abstraction and modularity – SML Modules

- Signature inclusion; Subtyping; Information hiding;
- [y2007p5q7 \(d\)](#)
 - SML module system, Signatures; Structures; ADT of stacks

- Functional / Imperative
- [y2010p3q5 \(d\)](#)
 - SML Signature matching
- [y2011p3q6 \(c\)](#), [y2009p3q2 \(d\)](#)
 - SML Signature, Functors
- [y2013p3q6 \(d\)](#)
- [y2014p3q6 \(d\)](#)
 - concrete signatures `sig type t = int` and opaque signature `sig type t`
 - constraint `;`, `:`, `>`

IX. Concurrency, parallelism

- Theoretical models ; Programming-language support for parallelism and distribution. Internal and external iteration.
- [y2014p3q6 \(b\)](#)
 - Threads, shared memory, message passing; Distributed memory, multi-core, cloud computing

X. Functional-style meets OOP

- Scala and Java 8; Procedural programming; Declarative programming; Mutable state; Blocks; Functions; Classes and objects; abstract classes; traits; Pattern matching;
 - [No longer explicitly lectured:] Type parameter bounds; View bounds; Implicit parameters; Implicit conversions; Mixin-class composition.
- [y2012p3q6 \(d\)](#)
 - Generic types and methods; Variance annotations
- [y2011p3q6 \(d\)](#)
 - Scala, parameter-passing
- [y2013p3q6 \(e\)](#)
 - Scala innovations
- [y2008p6q7 \(c\)](#)
 - Scala, function types ; Functions as objects
- [y2009p3q2 \(d\)](#)
 - Scala, variance
- [y2010p3q5 \(e\)](#)
 - Scala, Case classes

XI. Miscellaneous concepts

- Monads, GADTs, Reified continuations, Dependent typing.

- [y2016p3q5](#) (a, b, c)
 - unit/return, >=> / bind