Paradigms of computer programming

- Louv1.1x and Louv1.2x form a two-course sequence
  - Together they teach programming as a unified discipline that covers all programming languages
  - Second-year university level: requires some programming experience and mathematics (sets, lists, functions)
- The two courses cover four important themes:
  - *Functional programming* (and basic data structures)
  - *Formal semantics* (and computational complexity)

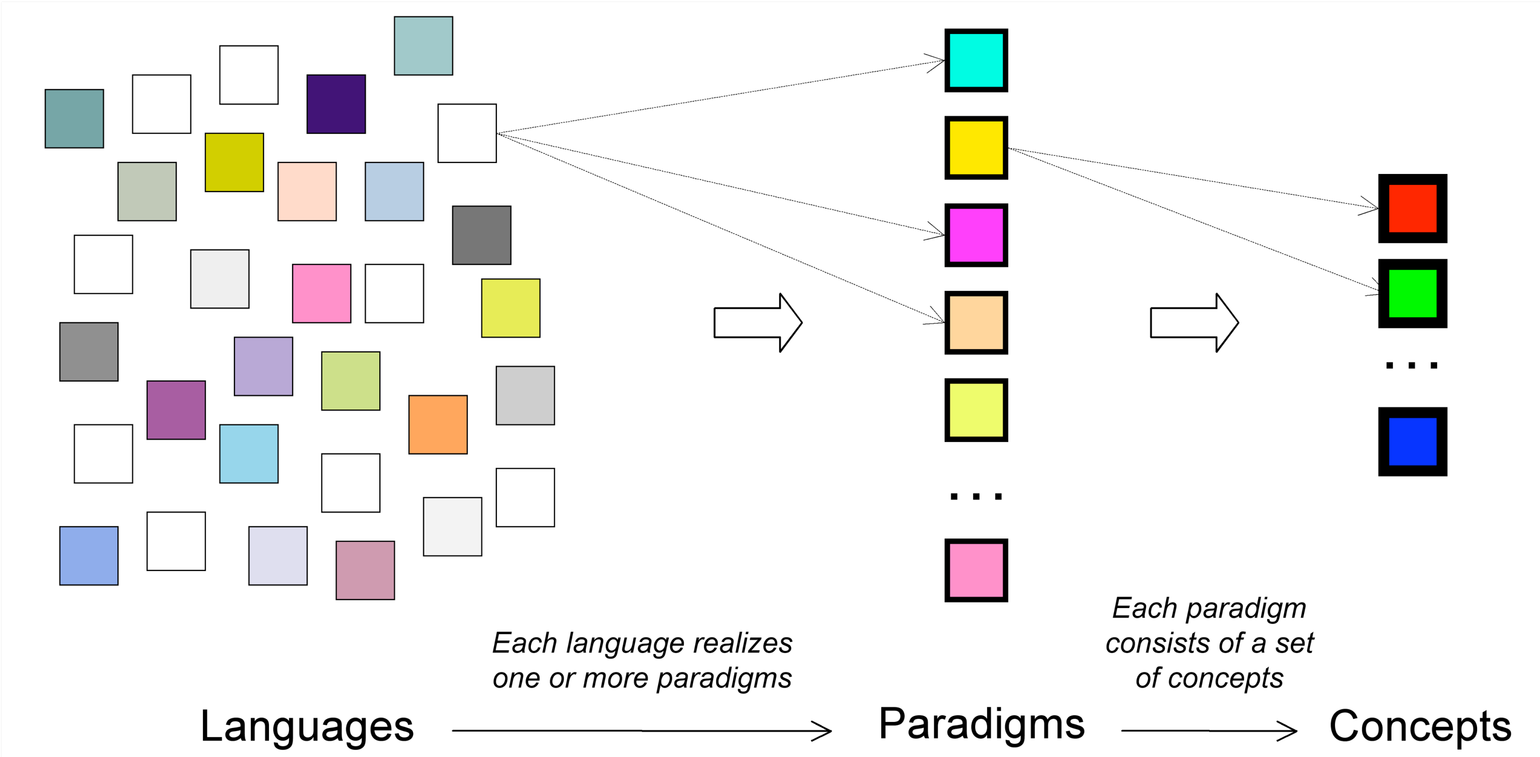
  } Louv1.1x

  - *Data abstraction* (and object-oriented programming)
  - *Concurrency* (and deterministic dataflow)

  } Louv1.2x
- Let's see how this works in practice

Hundreds of programming languages are in use...

So many, how can we understand them all?



- Key insight: languages are based on paradigms, and there are many fewer paradigms than languages
- We can understand *many* languages by learning *few* paradigms!

What is a paradigm?

- A programming paradigm is an approach to programming a computer based on a coherent set of principles or a mathematical theory

- A program is written to solve problems
  - Any realistic program needs to solve different kinds of problems
  - Each kind of problem needs its own paradigm
  - So we need multiple paradigms and we need to combine them in the same program

How can we study *multiple* paradigms?

- How can we study multiple paradigms without studying multiple languages (since most languages only support one, or sometimes two paradigms)?
- Each language has its own syntax, its own semantics, its own system, and its own quirks
  - We could pick three languages, like Java, Erlang, and Haskell, and structure our course around them
  - This would make the course complicated for no good reason
- Our pragmatic solution: we use one language, Oz, a research language designed for many paradigms
  - This lets us focus on the real issues
  - Our textbook, *Concepts, Techniques, and Models of Computer Programming*, uses Oz to cover many paradigms

How can we *combine* paradigms in a program?

- Each paradigm is a different way of thinking
  - How can we combine different ways of thinking in one program?
- We can do it using the concept of a kernel language
  - Each paradigm has a simple core language, its kernel language, that contains its essential concepts
    - Every practical language, even if it's complicated, can be translated easily into its kernel language
  - Even very different paradigms have kernel languages that have much in common; often there is only one concept difference
- We start with a simple kernel language that underlies our first paradigm, functional programming
  - We then add concepts one by one to give the other paradigms
  - Scientific method: understand a system in terms of its parts

Summary of the approach

- Hundreds of languages are used in practice: we cannot study them all in one course or in one lifetime
  - Solution: **focus on paradigms**, since each language is based on a paradigm and there there are many fewer paradigms than languages
- One language per paradigm is too much to study in a course, since each language is already complicated by itself
  - Solution: **use one research language**, Oz, that can express many paradigms
- Realistic programs need to combine paradigms, but how can we do it since each paradigm is a different way of thinking?
  - Solution: **define paradigms using kernel languages**, since different paradigms have kernel languages with much in common
  - Kernel languages allow us to define many paradigms by focusing on their differences, which is much more economical in time and effort

Let’s get started

- Probably you already know an object-oriented language
  - Object-oriented programming, with its coherent principles, is clearly an important paradigm
  - But what about the other paradigms?
- Isn’t object-oriented programming by far the most important and useful paradigm?
  - Actually, no, it’s not!
  - Many other paradigms are extremely useful, often more so than OOP! For example, to make robust and efficient distributed programs on the Internet, OOP just does not solve the right problems. Multi-agent dataflow programming is much better.
  - The two courses cover five paradigms that solve many problems

Five paradigms

- The two courses cover five paradigms:
  - Functional programming
  - Object-oriented programming
  - Deterministic dataflow programming
  - Multi-agent dataflow programming (*bonus lesson in Louv1.2x*)
  - Active objects

- These are probably the most important programming paradigms for general use
  - But there are many other paradigms, made for other problems: these two courses give you a good foundation for studying them later if you wish

Many
important ideas

Louv1.1x

- Identifiers and environments
- Functional programming
- Recursion
- Invariant programming
- Lists, trees, and records
- Symbolic programming
- Instantiation
- Genericity
- Higher-order programming
- Complexity and Big-O notation
- Moore's Law
- NP and NP-complete problems
- Kernel languages
- Abstract machines
- Mathematical semantics

Louv1.2x

- Explicit state
- Data abstraction
- Abstract data types and objects
- Polymorphism
- Inheritance
- Multiple inheritance
- Object-oriented programming
- Exception handling
- Concurrency
- Nondeterminism
- Scheduling and fairness
- Dataflow synchronization
- Deterministic dataflow
- Agents and streams
- Multi-agent programming

Next steps

- Practical organization of Louv1.1x
  - 6 lessons + 1 final exam
  - 1 lesson per week
  - Weekly exercises (highly recommended)
- Programming exercises
  - INGInious grader: gives feedback on errors
  - Mozart Programming System
- Our first paradigm: functional programming
  - Interactive examples and fundamental concepts