Programming Paradigms: logic, functional, object oriented, imperative

- paradigms: just an approximation of the rich diversity of real world of programming languages
- "50 shades of gray" : as they evolve, actual programming languages "learn form each other" and quickly adopt patterns and styles
- general concepts, history and refinements

Declarative vs. imperative programming

- declarative: execution details left to system, emphasis on specifying *WHAT* to do
- functional programming
- logic programming
- imperative:emphasis in specify *HOW* to do things
- procedural languages
- object orientation: mechanisms for inheritance (a logic concept), encapsulation of state

Functional programming

- theoretical model: lambda calculus, combinators
- intuition: working with functions as used in mathematics (e.g., Calculus)
- single assignment / referential transparency
- no side effects
- application and composition of functions

general concept and history

Logic Programming

- theoretical model: logic
- propositional and predicate calculus
  - truth tables, axioms, inference rules, variables, quantifiers
  - classical vs. intuitionistic
- Horn Clauses: a computationally efficient subset of predicate calculus $\Rightarrow$ Prolog
- Answer Set Programming: expressive extension based on propositional logic, SAT solvers

history, overview and subfields

Imperative Programming

- *procedural programming*
- procedures (also called functions or subroutines)
- hand-crafted data structures
- compilation to machine language
- structured programming, modularity, scoping constructs
- evolution to *object oriented programming*

general concept and subfields

Object Oriented Programming

- encapsulation of state
- code reuse via inheritance
- class vs. prototype based
- static vs. dynamic aspects
- design patterns

general concepts and refinements