⚠ Principles of Programming Languages

- Imperativ
  - What is it?
    - Explicit programming
      - Fully specified control flow
      - Fully specified data management
      - do this, then that
      - data x added to data y
  - Pro/Con
  - Data
    - is typed
    - named
    - somethimes anonymous (e.g. 4*5)
    - declaration
      - names a address ranges
      - imposes structure on address range
    - Variable is combination of name, type and value
  - Overlading
    - Binds name to multiple objects
      - Opposite to renaming
    - Context in which name is used tells which actual object to use
    - Math operators usually overloaded
  - Types
    - Enumeration
    - Compound
    - Data Structures
      - Sequence / Lists
        - Series of unkown number of elements, all of the same tpe
      - Array
      - Sets and Bags
      - Records and Pointers
      - Unions
    - Type Orthogonality
      - todo ⚠
    - Type Equivalence
    - Coercions / Contexts / Casts
      - Coercions are implicit type conversions
        - C/C++ has coercions, ADA not
        - 'Good/Bad?'
      - Conversions are function call that do the conversions
        - (ADA) A: Integer := Integer(3.14)
    - Abstract Data Types (ADT)
      - A named data-type defined solely by the operations on/for it
    - Generics
    - Typing
      - dynamic
        - types are inferred
      - static
  - Assignments: Lvalue = Rvalue
    - Lvalue indicates memory address
    - Completely evaluate the RHS (right-hand-side)?
    - Rvalue delivers a vlaue and is in form of an expression
  - Flow of Control
    - if then else
    - switch / case
    - Repetition
      - for
      - while
    - Signals and Exceptions
  - Procedures and Functions
    - Procedures have no return values
    - Functions have return values
  - Scope
    - Global
    - Local
    - Persistent
      - All global data in 'ABC' is persistent
      - Survives program exits
    - binding
      - is the process of attaching a use-name to a declaration
      - static
        - compile-time
        - C, ADA
      - dynamic
        - run-time
        - Python, Perl, shell-scripts
    - Scoping problems
      - declaration-before-use
      - forward-declarations
  - Parameter Passing mechanisms
    - Pass by value
    - Pass by reference
    - Pass by result
  - Modules and Packages
    - Group of related types, variables, routines, etc.
  - Program Composition
- Parallel/Distributed
  - Motivation
    - easier / more efficient implementation of some problems
    - some simulations require independent workers
  - characteristics
    - virtual cpus
    - shared memory / distributed memory
  - problems
    - deadlock
    - livelock
    - loadbalancing
    - data-race
      - inconsistent states
      - really hard to find
  - co-routines
    - shared memory
    - control-flow ping-pong
  - parallel statements
    - on top of functional / imperative lang.
    - parfor / par_begin + par_end
    - solves just the problem of allocating virtual processors
    - doesn't help with structure parallelism
    - data-races still possible
  - processes
    - child / parent
    - fork / join
    - 'shared' variables
      - multi processor: in shared memory
      - distributed: messages for reads/writes
  - synchronization
    - producer / consumer
    - mutex
      - no language statement is atomic
      - lock / unlock
      - semaphore
      - problem
        - locking granularity
        - cost of lock / unlock non-trivial
      - locking strategy
        - reads safe / writes require locking? → Copy-on-write
        - optimistic vs pessimistic
    - condition synchronization
    - monitor
      - special abstract-data-type
      - data access just over operators (mutual exclusive)
      - condition variables
        - wait blocks until signal → wakeup 1 waiter
  - message passing
    - distributed computers communicate over explicit messages over a network
    - =distributed assignment statement
    - language level
      - messages are typed
      - send and receive are blocking
  - naming sender / receiver
    - by process name
    - by I/O object
  - sync vs async
    - sync-send-recv
      - wait for msg recv (=delay=slow)
      - simple
    - async
      - send immediately → continue with next statement
      - wait for fixed set / singleton msg
      - recv: test if msg pending else error msg
    - most common: explicit receipt
      - alternative: create subprocess for each arriving msg within the receiver
  - one-way vs two-way
    - one-way: either input or output
      - occam
    - two-way: call and get return value
      - RPC
        - used by Java RPC, corba + others
        - wait for return value
        - looks like a lokal procedure call (remote references)
        - marshalling & unmarshalling of parameters
      - rendezvous
        - used by Ada
        - uses mailbox principle
  - linda tupell space
    - distributed database of tuples
    - in(tuple), out(tuple), read(tuple)
    - can be used to simulate complex data structures
      - trees
      - lists
      - arrays
  - actor languages & agents
    - autonomous objects = actors = active objects (own virt./mobile processor)
    - actors communicate with msgs
      - salsa!
  - serializing ressources
  - Transactions
    - set of statements whose effects are applied atomically
    - nested transactions?
- Other Paradigms ⚠
  - Domain Specific Languages
  - Constraint Programming
    - Ideally: Problem = a data type
      - data-type has a value domain/range
      - Value domain has constraints to only 'desirable' values
    - Example: 8 Queens problem
      - data type: checker-board
      - valid: data-type + 8 queens
      - desirable: 8 queens that do not attach
      - On Activation: system searches for solution -> constrain solver
    - Similar to prolog
  - Access-oriented Programming
    - Whenever a variable changes, call ossociated function
    - Used inernally in constraint based languages or in debuggers
    - Example Languages
      - 'Loops' from Xerox PARC
      - Spreadsheets
      - JavaFX
  - Single-datastructure languages
    - All the world is an array in 'JPL'
      - sucessor is named 'J'
    - Everything is a set in 'SETL'
  - Dataflow programming
  - Text processing: AWK
    - All the world is a string
    - Pattern matching
      - regular expressions
    - text replacement
    - pattern -> action
      - print has a monkey if the line contains a monkey
        - /monkey/ { print "monkey"; print" }
  - Shell programming
    - Unix shell
      - redirect i/o
      - global variables
      - loops, conditions etc
    - Windows shell: Powershell
      - OO Shell
  - Database languages
  - Music languages
  - Real-time languages
    - Correcteness criterion: What *and* When
  - Probabilistic / Fuzzy programming
  - Math languages
    - Matlab
    - Octave
  - GUI Programming
    - Separate graphics from implementation
    - Silverlight
    - JavaFX
  - Aspect Oriented Programming
  - WWW
    - Perl
    - Javascript
  - Literate Programming
    - Glue documentation and code together
- Future Paradigms
  - Coordination Languages
    - Handles when/where to start programs and how to connect them
    - Unix shell scripting can be seen as coordination language
      - not really good for parallel programming
      - Distributed programming involves external tools ala netpipe, ssh, etc
      - Perhaps we need an internet aware scripting language
    - Hermes
      - Resource: Software component offering some services, with well-defined interfaces
      - Taks use resources
        - compound task: multiple resources used
      - Service flows connect inputs to putputs upon event occurence
  - Component based programming
  - Interface Definition Languages
    - Corba
      - Objective: Multi-process, multi-language communication
      - Object lookup via strings, called "IOR"
      - Problem: Plymorphic remote calls
      - use a different language to interface to a (module|package|library)
  - Router Building Language: Click
    - Component types
      - DSL for building routers/packet-processors
      - pull, push something to/from queues to others can use it
      - Classifier: takes one packet and selects one of many outputs
      - Scheduler: tackes a packet from one of many inputs and puts it one output
      - Forward?Processor: Takes a packet from one input and puts it on one output
  - Agents
    - Are "autonomous fotare components"
    - Multi-agent system: Multiple intelligent agents interacting with each other
      - not monolithic
      - no global view of whole system
      - no global coordinator for system
  - XQuery
    - MAML: Multi-Agent Modeling Language
    - Language for querying XML formatted databases
    - Has: functions, conditional execution, looping, etc.
    - Return <E> statement add <E> to return set *but* does not immediately return control
  - R-Spec
    - Framework with a domain specific language
      - Based on ruby
      - Used for expressing tests in almost flued English
    - "Behavior Driven Development"
  - Fortress
    - OO Language with
      - parallelism
      - strucuring
        - trait (like java interface but may has implementations and inheritance)
        - object is set of traits with fields and mehtods
        - components / API
      - Extensibilty: A program can extend the base language
      - parametric types (like c++ template but not like java generics)
      - type inference where possible
      - top level functions (like static methods in java)
      - functions can have pre/post conditions
    - Almost anything allowed parallel
    - Juxtaposition. Context of symbols defines meaning. Example "n m"
      - Mul if n is variable
      - function call if n is a function
      - append if n a string
  - X10
    - Set of extensions for Java for easier/better parallel and distributed programming
      - lightweight taks
      - async expression + statement executions
      - immutable objects
      - multi-dim arrays
        - java has arrays of arrays only
    - A reference may not be null or the type must be nullable for this to happen
    - Partitioned global address space
    - atomic blocks
    - futures
- Object Oriented
  - Identify 'data' rather than 'functions'
  - Encapsulate data in Objects
  - Objects can be categorized and reasoned over their types
  - Problems
    - Functions change often, data changes little
    - Objects hide data from implementation
  - Binding Times:Objects decides which operation applies
  - Minimal language requirements
    - Data encapsulation
    - Inheritance
    - Dynamic Binding
  - Classes
    - Is Type of an Object
      - "a class is the category of an object"
    - Internal variables are members (fields, methods)
    - Class == type ?
      - Not every language looks at classes as types
    - Introspection
      - called 'meta classes'
      - Java: Reflection
      - Meta class contains information about
        - class constructors
        - class fields
  - Polymorphism
    - Polymorphic data type
      - Data type that can refer to different types at runtime
      - Provide interface to allow manipulation of multiple data types
    - Polymorphic function
      - Can operate on different argument types
  - Reference Semantics
  - Object Management
    - 'new' says nothing over 'where'
    - same language privde no 'delete'
    - If most/everything is a object, the per-boject memory overhead will kill us
      - Java JDK: 16 bytes per boject
      - C++: 8 bytes (64bit)
  - Inheritance
    - Linearization
    - Mixins
    - Inheritance as a Contract
      - Trait: A partially implemented contract
  - Generics/Templates
    - ⚠ generics instantiation
- Functional
- Logic
  - when is a set of rules a logic?
    - basic assumptions(axioms)
    - rules create facts from existing facts
    - a set of facts can only increase (once proven right, cannot prove false and retract the fact)
    - Predicate logic / (first,second) order logic / typed logic
  - programmer states problem. language/compiler figures out how to solve it
    - search for the answer
    - apply rules to problem to reduce it to a solution
    - problem specific knowledge can be applied
    - log.programming a step towards declarative programming
  - horn logic
    - condition is true if 0 or more other clauses are true
    - subset of predicate logic
    - predicates
      - a form of function: 0 or more arguments
      - argument variable or constant
    - system works back to prove query using facts and if-rules
      - search order is non-deterministic
      - clause can't be proven → try another clause: backtracking
      - all possible solutions are searched
        - DFS = expensive
    - closed world assumption: all that is needed must be in rules/facts
  - unification
    - head and goal match if equal / can be made equal by binding variables in head/goal
    - equality
      - same predicate
      - same number of arguments
      - each argument in head must be equal or made equal to goals by binding
      - if both are unbound, rename variable in goal
  - relations
    - horn clause specifies relation between argument variables
  - data structures
    - lists
    - facts as tuples