CSCE 4430: Programming Languages - Fall 2020

Instructor: Paul Tarau, Professor - see my [home page](#) for contact info and office hours.

course directory
code seen in class at github

Grader(s):

contact via [Canvas](#)

Midterm exam: Wednesday, Oct. 28, *open net*

Final exam: TBD, *cumulative, open net*

Description and Objectives:

A comprehensive programming language course, with emphasis on programming paradigms and language processors - and some of their formal models like Predicate Logic and Lambda Calculus. Hands-on work with implementations of key concepts (recursion, inheritance, unification, backtracking, type inference, infinite and lazy data objects, threads. The course also provides a glimpse at salient features of modern object oriented languages and an overview of language implementation techniques, run-time systems and memory management in an efficient procedural language.

Syllabus

Basics

- From Turing machines to today's programming languages
- Programming Paradigms: logic, functional, object oriented, imperative
- Language Processors: Interpreters and Virtual Machines, Compilers

Functional Programming (with Python and Haskell)
Intro to Untyped Functional Programming with Python
- Finite functions: dictionaries, list, tuples, sets
- Finite and Infinite Generators, Coroutining with *yield*
- List, set and dict comprehensions
- Programming with Recursive Functions and Lists
- Higher Order Functions: lambdas, map, reduce, itertools
- Intro to Types and Functional Programming with Haskell
- Lazy Evaluation, working with infinite lists, fold, map, zip
- Intro to Lambda Calculus, Simply Typed Lambda Terms
- Type inference, Polymorphism
- Recursive Data Types and Pattern Matching in Haskell

**Logic Programming (with Prolog and Python)**
- Intro to Prolog: Facts and Rules, Recursion, Lists and Compound Terms, Arithmetic
- Propositional and Predicate logic
- Unification and Horn Clause Resolution in Prolog
- Non-determinism and Backtracking
- Higher order Predicates and Meta-interpreters
- Modeling Prolog's execution mechanism in Python
- Definite Clause Grammars, Parsing and Generation
- Problem solving with Prolog
- Generating Lambda Terms in Prolog
- Type inference for simply-typed Lambda Terms in Prolog

**Object Oriented Programming (with Python, Java, Swift, Julia)**
- Classes and Objects in Python 3
- Overview of key features of Java, Swift and Julia
- Inheritance mechanisms
- Static vs. Dynamic Aspects
- Classes, Instances, Objects, Interfaces
- Multi-threading and multi-processing
- Collections, Streams, Lambdas, Iterators and IO operations
- Reflection and Serialization

**Low Level Imperative Programming (C)**
- Basics: assignment, function calls, lexical scoping, memory representations, stack and heap
- Implementing dynamic memory management and garbage collection
- Implementing high-level programming languages in C

**Concurrent Programming**

- Event driven programming, Coroutining, Futures, Multi-threading
- Shared Memory vs. Message Passing, Coordination, Distributed Programming

**Future Trends in Programming Language Design**

**Prerequisites: Data Structures, Algorithms, Solid Coding Experience**

**Recommended books and online materials:**

- Tucker & Noonan: Programming Languages, Principles and Paradigms, McGraw Hill
- The Art of Prolog by Sterling and Shapiro, MIT Press
- Doets & van Eijck: The Haskell Road to Logic, Math and Programming
- Harper’s draft PL book
- Java for Students: Douglas Bell & Mike Parr, Prentice Hall

**Recommended books and online materials:**

- Advanced Programming in Python
- The Art of Prolog by Sterling and Shapiro, MIT Press
- Doets & van Eijck: The Haskell Road to Logic, Math and Programming
- Java for Students: Douglas Bell & Mike Parr, Prentice Hall
- Tucker & Noonan: Programming Languages, Principles and Paradigms, McGraw Hill

**Evaluation:**

- 2 Individual Exams: 30%+30%
- Assignments (teams of up to 3): 40%

**Software, tutorials and related links:**
Outcomes:

1. Understand foundational concepts of programming languages, with emphasis on programming paradigms and language processors.
2. Have a practical understanding of commonalities and differences between major programming paradigms.
3. Understand key object oriented, logic and functional programming concepts.
4. Understand key concepts of event driven and concurrent programming.
5. Understand the use and implementation of modern programming language concepts like recursion, inheritance, reflection, unification, backtracking, type inference, infinite data objects and threads.