COURSE SYLLABUS

YEAR COURSE OFFERED: 2020

SEMESTER COURSE OFFERED: Spring

DEPARTMENT: Computer Science

COURSE NUMBER: COSC4315

NAME OF COURSE: Programming Languages and Paradigms

NAME OF INSTRUCTOR: Carlos Ordonez

The information contained in this class syllabus is subject to change without notice. Students are expected to be aware of any additional course policies presented by the instructor during the course.

Learning Objectives

Students will gain a good understanding on the theory, principles and computer systems aspects to design and implement programming languages. This course will cover basic compiler and interpreter system issues. Students will learn why it is necessary to develop specialized programming languages and why one language cannot be a universal solution. The course will explain imperative, procedural, modular, functional and object-oriented approaches and how they can interoperate. The course will make emphasis on which design and theory principles are essential and which practical features have made some languages more successful than others. Students will learn concepts in mainstream programming languages like C++ and Python.

Major Assignments/Exams

This is a course that gives more weight to programming homeworks. Grading is as follows:

- 80%: 4 programming assignments (2 on recursion, functional programming and data types, 2 on parsing/evaluation)
- 20%: midterm exam (around 10th week)

Required Reading

There is no single textbook since the subject is very broad, but the following textbooks are used in the course:

COURSE SYLLABUS

Daniel P. Friedman, Essentials of Programming Languages, MIT Press, 3rd edition

John C. Mitchell, Concepts in Programming Languages, 2002

List of lecture topics

1. Taxonomy of languages
2. Data types (simple, data structures, inference, dynamic vs static, ADT, pattern matching)
3. Programming: OO vs functional decomposition
4. Recursion (types, functions, tail, fixed points, stack manipulation)
5. Evaluation (translation, macros, lazy, memoization, exceptions, garbage collection)
6. Functional aspects (lambda calc, closures, higher order functions, currying, no mutation)
7. Object-oriented aspects (classes, polymorphism, containers, multiple inheritance, subtyping, extensibility)
8. Concurrent vs parallel programming (shared memory, distributed memory)