CS 314:
Principles of Programming Languages

He Zhu

Course Goals

- Understand why there are so many languages
- Describe and compare their main features
- Choose the right language for the job
- Write better code
  - Code that is shorter, more efficient, with fewer bugs

- In short:
  - Become a better programmer with a better understanding of your tools.

Course Activities

- Learn different **types of languages**
- Learn different **language features** and tradeoffs
  - Programming patterns repeat between languages
- Study how languages are **specified**
  - **Syntax**, **Semantics** — mathematical formalisms
- Study how languages are **implemented**
  - Mechanisms such as **closures**, **tail recursion**, **lazy evaluation**, **garbage collection**, …

Syllabus

- Functional programming (OCaml)
- Lambda Calculus (OCaml)
- Dynamic / Scripting languages (Python)
- Logical Programming (Prolog)
- Object-Oriented Programming (Python)
- Scoping, type systems, parameter passing, Comparing language styles; other topics (OCaml, Prolog, Python)
- Program Verification and Program Synthesis (if possible)

Workload

Grading (subject to change)

Item	Due	Weightage (%)
Assignment 1 (OCaml)		5.0
Assignment 2 (OCaml)		5.0
Assignment 3 (OCaml)		15.0
Assignment 4 (Prolog)		15.0
Assignment 5 (Python)		10.0
Assignment 6 (Python)		10.0
Written Assignment 1		5.0
Written Assignment 2		5.0
Final Project		30.0

Rules and Reminders

- Use lecture notes as your text
  - Supplement with readings, Internet
- Keep ahead of your work
  - Get help as soon as you need it
    - Office hours, Piazza (email as a last resort)
- Assignment late penalties
  - 1 day late – 5%
  - 2 days late – 15%
  - 3 days late – 30%
  - 4 days late – 50%
  - > 4 days late – No grade

Academic Integrity

- All written work (including projects) must be done on your own
  - Do not copy code from other students
  - Do not copy code from the web
  - Do not post your code on the web
- Auto-comparing code for every assignment
  - Receive 0 if two assignments are flagged by the tool.
- Work together on *high-level* project questions
  - Do not look at/describe another student's code
  - If unsure, ask an instructor!

Other information

Zoom: (https://rutgers.zoom.us/j/96220906368?pwd=aVM2aUQ1SytWcDc5d0hEYjhWcXd0dz09)

Sakai: (https://sakai.rutgers.edu/portal/site/8acbdc1d-e374-4a79-bafa-bab0e8d58811)

Piazza: (https://piazza.com/rutgers/spring2021/cs314)

Website: (https://ru-automated-reasoning-group.github.io/cs314_s21/)

Office Hours: Thursday 1:20p - 2:30p.

TA information will be posted on course website shortly.

CS 314:
Principles of Programming Languages

Overview

Plethora of programming languages

- LISP:

```
(defun double (x) (* x 2))
```

- Prolog:

```
size([],0).
size([H|T],N) :-
size(T,N1), N is N1+1.
```

- OCaml:

```
List.iter (fun x -> print_string x)
          [“hello, ”; s; "!\n”]
```

All Languages Are (kind of) Equivalent

- A language is Turing complete if it can compute any function computable by a Turing Machine

- Essentially all general-purpose programming languages are Turing complete
  - I.e., any program can be written in any programming language
- Therefore this course is useless?!
  - Learn one programming language, always use it

Studying Programming Languages

- Will make you a better programmer
  - Programming is a human activity
    - Features of a language make it easier or harder to program for a specific application
  - Ideas or features from one language translate to, or are later incorporated by, another
    - Many "design patterns" in Java are functional programming techniques
  - Using the right programming language or style for a problem may make programming
    - Easier, faster, less error-prone

Studying Programming Languages

- Become better at learning new languages
  - A language not only allows you to express an idea, it also shapes how you think when conceiving it
  - You may need to learn a new (or old) language
    - Paradigms change quickly in CS
    - Also, may need to support or extend legacy systems

Changing Language Goals

- 1950s-60s – Compile programs to execute efficiently
  - Language features based on hardware concepts
    - Integers, reals, goto statements
  - Programmers cheap; machines expensive
    - Computation was the primary constrained resource
    - Programs had to be efficient because machines weren't
      - Note: this still happens today, just not as pervasively

Changing Language Goals

- Today
  - Language features based on design concepts
    - Encapsulation, records, inheritance, functionality, assertions
  - Machines cheap; programmers expensive
    - Scripting languages are slow(er), but run on fast machines
    - They've become very popular because they ease the programming process
  - The constrained resource changes frequently
    - Communication, effort, power, privacy, …
    - Future systems and developers will have to be nimble

Theme: Software Security

- Security is a big issue today
- Features of the language can help (or hurt)
  - C/C++ lack of memory safety leaves them open for many vulnerabilities: **buffer overruns**, **use-after-free** errors, **data races**, etc.
  - Type safety is a big help, but so are abstraction and isolation, to help enforce security policies, and limit the damage of possible attacks
- Secure development requires vigilance
  - Do not trust inputs – unanticipated inputs can effect surprising results! Therefore: verify and sanitize

Zero-cost Abstractions in Rust

- A key motivator for writing code in C and C++ is the low cost of the abstractions use
  - Data is represented minimally; no metadata required
  - Stack-allocated memory can be freed quickly
  - Malloc/free maximizes control – no GC or mechanisms to support it are needed
- But no-cost abstractions in C/C++ are insecure
- Rust language has safe, zero-cost abstractions
  - Type system enforces use of ownership and lifetimes
  - Used to build real applications – web browsers, etc.

Language Attributes to Consider

- Syntax
  - What a program looks like
- Semantics
  - What a program means (mathematically)
- Paradigm and Pragmatics
  - How programs tend to be expressed in the language
- Implementation
  - How a program executes (on a real machine)

Syntax

- The keywords, formatting expectations, and “grammar” for the language
  - Differences between languages usually superficial
    - C / Java if (x == 1) { … } else { … }
    - Ruby if x == 1 … else … end
    - OCaml if (x = 1) then … else …

  - Differences initially annoying; overcome with experience

- Concepts such as regular expressions, context-free grammars, and parsing handle language syntax

Semantics

- What does a program *mean*? What does it *do*?
  - Same syntax may have different semantics in different languages!

	Physical Equality	Structural Equality
Java	a == b	a.equals(b)
C	a == b	*a == *b
Ruby	a.equal?(b)	a == b
OCaml	a == b	a = b

- Can specify semantics informally (in prose) or formally (in mathematics)

Why **Formal** Semantics?

- Textual language definitions are often incomplete and ambiguous
  - Leads to two different implementations running the same program and getting a different result!
- A formal semantics is basically a mathematical definition of what programs do
  - Benefits: concise, unambiguous, basis for proof
- We will consider operational semantics
  - Consists of rules that define program execution
  - Basis for implementation, and proofs that programs do what they are supposed to

Paradigm

- There are many ways to compute something
  - Some differences are superficial
    - For loop vs. while loop
  - Some are more fundamental
    - Recursion vs. looping
    - Mutation vs. functional update
    - Manual vs. automatic memory management
- Language's paradigm favors some computing methods over others. This class:
  - Imperative
  - Functional
  - Scripting/dynamic

Imperative Languages

- Also called procedural or von Neumann
- Building blocks are procedures and statements
  - Programs that write to memory are the norm

```
int x = 0;
while (x < y) x = x + 1;
```

  - FORTRAN (1954)
  - Pascal (1970)
  - C (1971)

Functional (Applicative) Languages

- Favors immutability
  - Variables are never re-defined
  - New variables a function of old ones (exploits recursion)
- Functions are higher-order
  - Passed as arguments, returned as results

  - LISP (1958)
  - ML (1973)
  - Scheme (1975)
  - Haskell (1987)
  - OCaml (1987)

OCaml

- A mostly-functional language.
  - Has objects, but won't discuss (much)
  - Developed in 1987 at INRIA in France
  - Dialect of ML (1973)
- Natural support for pattern matching.
  - Generalizes switch/if-then-else - very elegant
- Has full featured module system.
  - Much richer than interfaces in Java or headers in C
- Include type inference.
  - Ensures compile-time type safety, no annotations.

Dynamic (Scripting) Languages

- Rapid prototyping languages for common tasks
  - Traditionally: text processing and system interaction
- "Scripting" is a broad genre of languages
  - "Base" may be imperative, functional, OO…
- Increasing use due to higher-layer abstractions
  - Originally for text processing; now, much more

  - sh (1971)
  - perl (1987)
  - Python (1991)
  - Ruby (1993)

Other Language Paradigms

- Logic programming
  - Prolog, λ-prolog, CLP, Minikanren, Datalog
- Object-oriented programming
  - Simula, Smalltalk, C++, Java, Scala, Python
- Parallel/concurrent/distributed programming
  - Cilk, Fortress, Erlang, MPI, Hadoop

Concurrent / Parallel Languages

- Traditional languages had one thread of control
  - Processor executes one instruction at a time
- Newer languages support many threads
  - Thread execution conceptually independent
  - Means to create and communicate among threads
- Concurrency may help/harm
  - Readability, performance, expressiveness
- Won't cover in this class

Summary

- Programming languages vary in their
  - Syntax
  - Semantics
  - Style/paradigm and pragmatics
  - Implementation
- They are designed for different purposes
  - And goals change as the computing landscape changes, e.g., as programmer time becomes more valuable than machine time
- Ideas from one language appear in others