ID2005 Dynamic Programming Languages 7.5 credits

Dynamiska programmeringsspråk

This is a translation of the Swedish, legally binding, course syllabus.

If the course is discontinued, students may request to be examined during the following two academic years

Establishment

Course syllabus for ID2005 valid from Spring 2009

Grading scale

A, B, C, D, E, FX, F

Education cycle

Second cycle

Main field of study

Specific prerequisites

Moderate programming skills (B-level) in a systems programming language
(e.g. Pascal, C, C++, Java, Haskell, OCaml, Mozart, OZ).

För fristående studerande gäller följande behörighetskrav:

- Grundläggande högskolebehörighet, dvs avslutad gymnasieutbildning inkl svenska och engelska el motsv och

- Kandidatexamen/180 hp (120 gamla poäng) i informationsteknik, informationssystem, datateknik eller data- och systemvetenskap.

Language of instruction

The language of instruction is specified in the course offering information in the course catalogue.

Intended learning outcomes

Dynamic languages, sometimes called 'scripting languages', are high-level, dynamically typed programming languages, often developed in the open-source community and prone to change more rapidly than proprietary-owned languages such as Java and C#. Dynamic languages are flexible and powerful; recent studies suggest that programs written in dynamic languages are about five times shorter than their static equivalents. This means shorter development time and thus cost, and produces programs that are easier to read, change and maintain.
Dynamic languages have been called the 'unsung hero of the world-wide web'. Many well-known systems, like the Amazon shopping-site or Google are built on top of dynamic languages. Dynamic languages have also found widespread use as a tool for intergrating different systems and components. Dynamic languages are also used to script larger systems or as a high-level interface to a lower-level language. The nature of dynamic languages also facilitate metaprogramming, where the language is used to manipulate the language itself. This is superb for the implementation of domain-specific languages. Domain-specific languages raise the level of abstraction even further than objects and allows the programmer to express the program in terms of the domain. An excellent example of the use of metaprogramming is in the implementation of Ruby on Rails, an MVC framework for building internet applications. Rails allows automatic generation of interfaces straight from the code and provides high-level variable declarations such as "belongs_to" and "has_many", all coded in Ruby, and accessible from within the programming language itself.
The popularity of dynamic languages is increasing. The old maxim that static compile-time checks are necessary to build secure software is questioned time and time again by acknowledged people, programmers and academics alike and the dynamic philosophy maps very well with the popular agile methods like XP and test-first development.
In this course, we will examine the philosophy behind dynamic languages. We do so in the context of Python and Ruby, two proven dynamic languages with different strengths and weaknesses. As well as using dynamic languages as the full-blown programming languages they are, we will examine the use of dynamic languages for integration and as tools for metaprogramming, especially for implementation of domain-specific languages. We will also look at prototype-based languages like IO and high-level languages like Groovy, for interacting with e.g., Java programs and the Java API.
This course is part of the competence track in Software Construction (Software Engineering): http://dsv.su.se/~tobias/inriktningen/

Course contents

Having successfully completed the course with grade E or higher, a student should be able to:
1. use (at least) Ruby and Python in program development, both as application languages, as embedded languages and scripting tools.

2. quickly pick up other dynamic programming languages with imperative core.
3 use prototype-based programming languages.
4. reason about the effects of choosing a dynamic programming language over a systems programming language (or vice versa), for a project with respect to design, programming, testing, static cheking, maintenance, memory, speed and safety.
5. reason about the philosophy of the languages used in the course, how this philosophy is affected by language change and additions, enough to interact with communities around dynamic programming languages.
6. use metapogramming and reason about the effects of using metaprogramming instead of "traditional programming" in a project.
7. analyse her own beliefs and preconceptions about dynamic programming languages, their underlying rationale and criticise them.
8. discuss current trends in the programming language developing communities and research community such as gradual typing and relate these to more mature concepts such as duck typing and structural typing.

Disposition

Lectures and discussions. Several (approx. 3) small assignments in pairs or one larger project in a 3-4 person group. Also literature seminars for those aiming for a high grade.

Course literature

There are a lot of suitable books for this course, and we encourage the student to choose a set of books herself. For a list of suitable books and a discussion of their merits, see this list:
http://people.dsv.su.se/~tobias/dypl.cgi?Literature
Login name: dypl
Password: guido
Since there are so many different books, we don't ask the bookshops here to buy them. They are generally much, much cheaper through e.g., AdLibris or Amazon.

Examination

- INL1 - Exercise, 1.5 credits, grading scale: A, B, C, D, E, FX, F
- INL2 - Exercise, 1.5 credits, grading scale: A, B, C, D, E, FX, F
- INL3 - Exercise, 1.5 credits, grading scale: A, B, C, D, E, FX, F
- INLA - Assignment, 3.0 credits, grading scale: A, B, C, D, E, FX, F

Based on recommendation from KTH's coordinator for disabilities, the examiner will decide how to adapt an examination for students with documented disability.

The examiner may apply another examination format when re-examining individual students.

Other requirements for final grade

To pass DYPL, you must:
a) complete three programming exercises in Ruby, Python and a systems programming language of your choice (Java, C/C++, Haskell, etc.); and b) pass the take-home exam.
a) is examined as three assignments with separate deadlines. Each assignment is to be implemented by two student working as a pair. The pairs are rotated. The grade for each assignment of the assignment is pass or fail.
b) is examined individually in the form of a take-home exam. Possible grades are U, 3, 4, 5.
A student that passes a) and b) receives a course grade that is the same as the grade on the take-home exam.

Ethical approach

- All members of a group are responsible for the group's work.
- In any assessment, every student shall honestly disclose any help received and sources used.
- In an oral assessment, every student shall be able to present and answer questions about the entire assignment and solution.