Lisp Tutorial

Jan Winkler

Institute for Artificial Intelligence
Universität Bremen

May 26, 2014
Summer Term

Today's Schedule

- Lisp Project plans
- Documentation galore
- Details on the Lisp ROS node template
- Code Style Guidelines
- Assembly and Programming

Lisp Project
Project Specification

Goals for this course:

- Specify a (robotics related) problem and solve it using Lisp
- Vessel for the solution: Self-assembled Lego Mindstorms Robots
- Solving of the actual problem in a structured manner
- Document your Project:
  - **Project Specification**: Problem to be solved, techniques used
  - **Code Documentation**: *Semantic* meaning of functions, parameters, return values
- Short demonstration of your solution

If anything is unclear, **ask early**!

Lisp Project

Individual Evaluation per Participant

Evaluation of course participants is done **individually**.

Clearly state your own contribution to the group project by

- properly documenting **your** part and
- differentiating **your** code to make clear what functionality you contributed to the group project.

The baseline grade is based on the overall group success (e.g. *"Did the project work?"*, *"Was the overall goal achieved?"*).

So make sure that you

- **Work together,**
- **but clearly show your own contribution!**

Be prepared for a short individual interview at the end of the project.

Project Documentation

How to write proper Documentation - A primer

How to write your documentation strongly depends on **what** you want to document.

The main variance axes in documentation are:

- The level of **technical detail**
- Mentioning of, and identifying with **related work**
- **Special focus** can lie on
  - *Usability* (make others able to use the documented entity)
  - *Proof* (show that the documented entity works as expected)
  - *Comprehension* (give an extensive overview about your work to external auditors)

Project Documentation

How to write proper Documentation - A primer

How to write your documentation strongly depends on **what** you want to document.

The main variance axes in documentation are:

- The level of **technical detail**
- **Special focus** can lie on
  - *Usability* (make others able to use the documented entity)
  - *Proof* (show that the documented entity works as expected)
  - *Comprehension* (give an extensive overview about your work to external auditors)

Project Documentation

How to write proper Documentation - A primer

Organizational matters:

- Language: German, English (your choice)
- Per Group
- 3-6 pages
- Make *meaningful separate sections* for each group member
- **Don't** include source code
- The amount of figures should not exceed 20% of the document. Images are figures.
- Include your e-mail address and name for reference
- Handin: July 14th, 2014 (during the course)

Project Documentation

How to write proper Documentation - A primer

Format matters:

- When possible, use LaTeX. Word/Office documents are okay, too.
- File format for handin independent of word processor: PDF
- Write a short introduction about what you did, and what problems you approached.
- Later on, show the solutions to the approached problems, as well as the techniques used.

From the documentation alone the significance of what you did should be clear (just building a robot that simple drives in a circle continuously, or comparable examples, is not significant).

Project Documentation

How to write proper Documentation - A primer

When writing technical documentation,

- either document Top-Down **or** Bottom-Up.
  Don't mix both styles in the same document.
- clearly get to the point.
  A documentation not understood does not satisfy the requirements.
- if necessary, explain details in-depth.
  If not, don't over-exaggerate on unimportant details.

Lisp ROS Node Template
The Template itself

Git repository for the NXT ROS Lisp Node Template:
`https://github.com/ai-seminar/nxt_lisp.git`

ASDF System name: `nxt-lisp`

TODO for your own project when using the template:

- Change the author and the description in
  - `nxt-lisp.asd`
  - `package.xml`
  - `src/package.lisp`
  - all BSD headers (present in almost every file)

Lisp ROS Node Template

How to get the Code

Best practice on GitHub (`https://www.github.com`):

- Register/Log in to GitHub (free)
- Go to the NXT Lisp Template repository
- Click on *"Fork"* (upper right corner)
- Check out your own repository (the fork) and work on that copy

These repositories are representative for your code throughout grading.

That means: **Code Quality matters. This IS a Lisp course, after all.**

Code Style

Code Documentation

Document functions you write in the code:

Function Documentation Example

```
(defun my-function (param-1 &optional param-2)
  "This function performs very important tasks. It has two
  parameters. ‘param-1’ is the first one and is very important
  for doodledy-do. ‘param-2’ is optional and is used for
  fiddely-doodle."
  (some-very-sophisticated-code))
```

When the first line in a function/method definition is a string, it is handled as the documentation string (*doc-string*).

Find the doc-string of any function using

```
(describe '<function-name>)
```

~~Also, document whatever code pieces might not be clear.~~

Code Style
Code Documentation

Also, document whatever code pieces might not be clear.
Add comments in your code using semicolons:

Comments in Lisp

```
;; From here on, quantum functionality is coded
;; TODO(winkler): Finish function five
(defun quant-function-1 (param-1 &optional param-2)
  "This function makes use of the Skasis Paradigm,
  and does more interesting things."
  (skasis-par :quantum t) ;; This function is super important.
                          ;; It calculates quantum irregularities
                          ;; using the Skasis Paradigm.
  (do-more-interesting-stuff))
```

Code Style

No newlines for closing parantheses

Putting closing parantheses on separate lines of commonly considered *bad programming style*. It makes code **difficult to read** and **longer than necessary**.

Always close parantheses **on the same level** as the **last form they include**!

Code Style

No newlines for closing parantheses

Example of code with newlines for closing parantheses

```
(defun function-1 (param-1)
  (let ((a 5)
        (b 6)
       )
    (unwind-protect
        (prog2
          (format t "b = ~a~%" b)
          (incf b)
          (socket-on *socket* :timeout a)
        )
      (let ((socket-status (socket-status *socket)
            )
           )
        (socket-off *socket*)
      )
    )
  )
)
```

Code Style

No newlines for closing parantheses

The same code, without newlines

```
(defun function-1 (param-1)
  (let ((a 5)
        (b 6))
    (unwind-protect
        (prog2
          (format t "b = ~a~%" b)
          (incf b)
          (socket-on *socket* :timeout a))
      (let ((socket-status (socket-status *socket*)))
        (socket-off *socket*)))))
```

Assembly And Programming

The rest of today's course will be dedicated for assembly and programming of your projects.

Be sure to think of project topics **soon**.
Don't hesitate to **ask** should any questions arise.