More on Logistics (I)

- **Final grade**
  
  $$\text{final grade} = \left( \frac{\text{attendance} + \text{hw-points} + \text{quiz-points} + \text{extra-credit-points} + \text{exam-points}}{\text{max-attendance} + \text{max-hw-points} + \text{max-quiz-points} + \text{max-exam-points}} \right)$$
  
  - Using this formula you can always check where you are standing. 50% corresponds to F, 50-62.5 is D range, 62.5-75 is C range, 75-87.5 is B range, and 87.5-100 is A range.

- **Homeworks and Late submissions**
  
  - To get full credit submit homework before or on the due date!
  - 20% of the grade is lost each late day

- **Quizzes**
  
  - One lowest score will be dropped
More on Logistics (II)

- Extra credit
  - Be active in forums, answer questions, report errors and problems
  - Take part in extra credit studies
- Catch up early:
  - Get books, ask questions, seek help
  - Run examples, experiment, write your code
- Integrity

Outline

- Karel program syntax
- Programming errors
- Edit-Compile-Run-Test loop
- Karel built-in commands
- Defining new commands for Karel
- Naming Karel commands
Karel Program Syntax

- Karel programs have the following structure

```
beginning-of-program
beginning-of-execution
<commands>
  turnoff;
end-of-execution
end-of-program
```

- Where `<commands>` is a sequence of Karel commands separated by semicolons ;
- Note that it is a bit different from C language: in C a semicolon ends a command
- "One command in each line" is a good style, not a syntax rule!

Syntax Errors

- What happens if the syntax rules are broken?

```
beginning-of-program
beginning-of-execution
  move;
  move;
  turnleft
  move;
  turnoff;
end-of-execution
end-of-program
```
No “;”
Semantic Errors (bugs)

- If there are no syntax errors, does it mean that the program is correct?

```plaintext
beginning-of-program
beginning-of-execution
move;
move;
move;
turnleft;
turnoff;
end-of-execution
end-of-program
```

Where is the error?

```plaintext
beginning-of-program
beginning-of-execution
move;
move;
turnoff;
move;
turnleft;
end-of-execution
end-of-program
```
The edit-compile-run loop

1. Edit program
2. Compile program
3. If there are errors, fix and go back to 1
   • you have got syntax error
   • fix and go back to 1
4. Run it
5. If it produce wrong results
   • you have got semantic error
   • find the source of the error (debug)
   • fix and go back to 1

The iterative nature of programming

The “programming in small” loop
The Full set of Karel commands

- **move** - move one corner in the current direction
- **turnleft** - turn left, change direction
- **pickbeeper** - pick 1 beeper from the current corner, put into the beeper bag
- **putbeeper** - place 1 beeper from the beeper bag on the current corner
- **turnoff** - turns itself off

Foolproof Karel: Error shutoff

- Can your errors hurt Karel?
  - **move** - shutoff if facing a wall
  - **pickbeeper** - shutoff if no beepers on the corner
  - **putbeeper** - shutoff if no beepers in the beeper bag
  - **turnleft and turnoff** - always possible
Problem: Move beeper

- Move a beeper from 1:4 to 3:5

Example: Move beeper

```plaintext
beginning-of-program
    beginning-of-execution
    move;
    move;
    pickbeeper;
    move;
    turnleft;
    move;
    move;
    putbeeper;
    move;
    turnoff;
    end-of-execution
end-of-program
```
We can define new instructions

- How to extend Karel’s set of instructions?

```python
define-new-instruction <name> as <instruction>
```

- Example:
  ```python
define-new-instruction go as move
```

Why? Case 1: Square Dance

```python
beginning-of-program
  beginning-of-execution
    move;
    turnleft;
    move;
    turnleft;
    move;
    turnleft;
    move;
    turnleft;
    move;
    turnleft;
    move;
    turnleft;
    turnoff;
  end-of-execution
end-of-program
```
Block

A syntactically correct way to make a sequence of instruction looking as one instruction. A block can be used whenever single instruction can be used

\[
\text{begin} \\
\text{<instruction>;} \\
\text{<instruction>;} \\
\text{...} \\
\text{<instruction>}\;
\text{end}
\]

Create a new instruction with the block construct

Blocks can be used to define new instructions from several elementary ones

\[
\text{define-new-instruction} \ <\text{name}> \ \text{as} \\
\text{begin} \\
\text{<instruction>;} \\
\text{<instruction>;} \\
\text{...} \\
\text{<instruction>}\;
\text{end;}
\]
Solution 1: The Missing turnright

- Now we can define turnright

```plaintext
define-new-instruction turnright as begin
  turnleft;
  turnleft;
  turnleft;
end;
```

Square Dancing Clockwise

```plaintext
beginning-of-program
  define-new-instruction turnright as begin
    turnleft;
    turnleft;
    turnleft;
  end;
  beginning-of-execution
    move;
    turnright;
end-of-execution
end-of-program
```

The place for defining new instructions is between beginning-of-program and beginning-of-execution.
The Flow of Execution: The Glossary Model

- When Karel encounters the new name in the process of program execution, it looks for its “definition” in the glossary of commands.
- If the definition of the new command is found, Karel executes the body of the command definition.
- After that, Karel returns to the next instruction.

Name does not matter (for execution)

- Names are just names. What the new command will do is defined by its body, not by its name.

```plaintext
define-new-instruction turnright as begin
  move;
  move;
  move;
  move;
end;
```
Name does matter (for understanding)

- From syntactic prospect, name could be any combination of letters, numbers and hyphens that starts with a letter
- From the understanding prospect, the name should express the function of the new command

```plaintext
define-new-instruction i543 as begin
    turnleft;
    turnleft;
    turnleft;
end;
```

Before next lecture:

- **Reading assignment**
  - Pattis:
    - Chapter 2
    - Chapter 3, Sections 3.1 - 3.7
  - Tutorial: Lesson 4
- **Follow Chapter 2 by writing and running code**
- **Check yourself by doing exercises from Chapter 2**