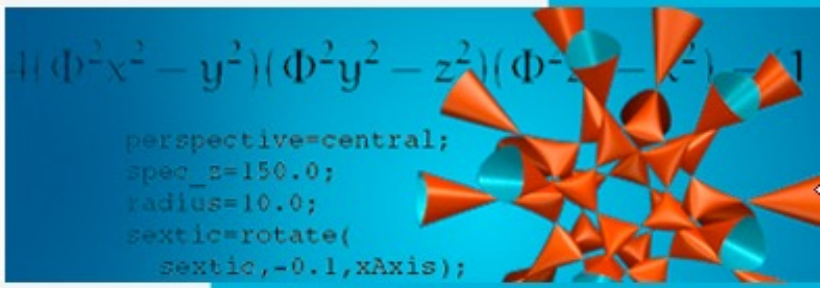
Philipps Universität Marburg

Declarative Programming

Summer semester 2024

Prof. Christoph Bockisch, Steffen Dick
(Programming Languages and Tools research group)

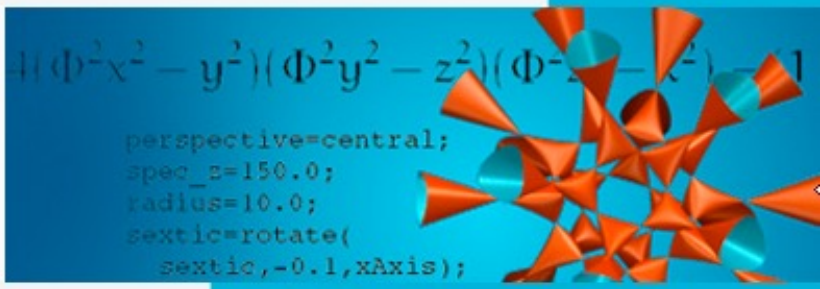
Imke Gürtler, Daniel Hinkelmann, Aaron Schafberg, Stefan Störmer



[Script 1 - 1.1]

Philipps Universität Marburg

Organization



Organization

- Lecture (participation in both dates per week)
  - Tuesdays 16:15 - 17:45, HS B
  - Wednesdays 16:15 - 17:45, HS B

- Tutorials (participation voluntary)
  - Thursdays 10:00 - 12:00 *(2 supervising tutors)*
  - Thursdays 14:00 - 16:00 *(2 supervising tutors)*
  - Registration for tutorials from Wednesday, 16.04. at 20:00h
    - If necessary, you can also change without consultation

Organization

- Exercise mode (for details see document in ILIAS)
  - **Self-study** *(voluntary)*
    - Self-exercise sheet
      - Will be published gradually in ILIAS
      - Sample solutions after completion of the topic in lecture
    - Partial programming tasks with automatic testing tool
    - Open tutorials
  - **Mandatory**
    - **Electronic test** in ILIAS (usually **weekly**), **automatically** assessed *(max. 2 with 0 points)*
    - **Submission** with correction by **tutors** (dates will be announced on ILIAS)
      - **Programming task** *(must not be 0 points)*
      - **Proof task** *(must not be 0 points)*
    - **Exam admission** *(at least 50%)*
    - **Bonus point** *(at least 80%)*

PubQuiz

- Instead of a lecture
- "Puzzles" on the subject matter of the lecture
  - Approx. 8 tasks
  - Processing time approx. 50 minutes
  - The tasks are then discussed
  - Editing in groups allowed
- You can win additional points here
- This means that you can still acquire the closed course admission or the bonus if you are only a little short of the 50% or 80% limit

Teacher training students

- Regular exercise operation as previously described
  - Changes
    - Elimination of an electronic test and the evidence task
    - Corresponding topic is also replaced by an interface task in the written exam
    - Adding the interface exercise
- Exercise mode - interface exercise (teaching degree)
  - Participation mandatory for teacher training students
  - Largely analog, more details in interface exercise

Master Students (Conditional Module)

- Join the group within this ILIAS course
  "Group for Master students with DP as conditional module"
  - When necessary we will communicate with this group of students via the ILIAS group
- English versions of slides and lecture notes are available in the regular ILIAS course
- If you follow this course as conditional module,
  - You are automatically admitted to the exam.
  - Participation in the exercises is not required ...
  - ... but still recommended as practice for the exam.

Organization

- Exam
  - First and second exam
  - Dates in MARVIN
    - As parallel groups of the lecture
    - **Will be announced later**

- Exam registration *(approx. middle of the semester)*
  Please refer to the website of the examination office!

ILIAS

- Materials via ILIAS
  - Script, slides, attendance and homework sheets
  - Submission and feedback
  - Auto-Testat tool

- Join the lecture:
  Magazine → MARVIN: Courses of all semesters → Summer semester 2024 → FB12 Mathematics and Computer Science → Declarative Programming

Scheduling

- Canceled lecture
  - 01.05.: Labor Day
- Failures exercises
  - 09.05.: Ascension Day, 30.05.: Corpus Christi
    - Tutorials are canceled
    - Electronic tests over two weeks

Learning objectives

- Learning a declarative programming language
- Recognizing and applying abstractions
- Understanding and recognizing cross-linguistic concepts and their areas of application
- Formal definition of the semantics of programming languages
- Scientific work and communication skills

Topics

- Programming with expressions
- Programming with function definitions
- Conditional expressions
- Design of function definitions
- Algebraic data types
- Proof of program properties
- Abstraction via functions and types
- Scoping
- Pattern Matching
- Generative recursion
- Accumulators
- Logic programming

Organization

- Script
  - Based on script by Prof. Klaus Ostermann
  - Based on "How to Design Programs" by M. Felleisen, R. B. Findler, M. Flatt and S. Krishnamurthi
- Programming language: DrRacket (Version 8.12)

DrRacket

- Lisp dialect
- Different language levels selectable
- Installation: http://racket-lang.org
- Settings
  - Language -> Select language: "How to Design Programs - Beginner"
  - Language -> Add Teachpack: "universe" and "2htdp/image"
- Use
  - Interaction area: lower half of the screen
  - Enter expression, confirm with Enter → Result is displayed in the next line

Installation

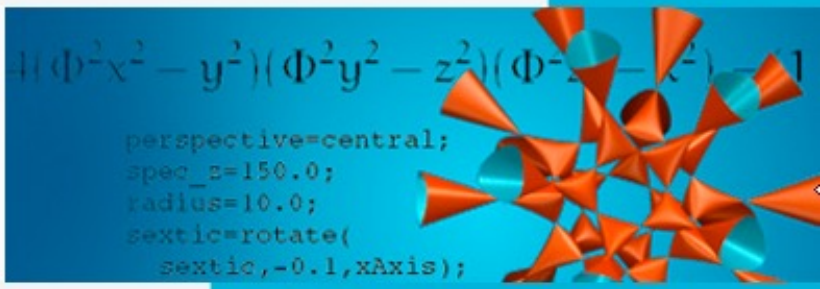
Install DrRacket.

Install the tool for automatic testing.

If you have problems with the installation, we can support you during consultation hours.

Philipps Universität Marburg

Introduction



Algorithms

- Precise definition of the steps to solve a problem
- The same problem can be solved by different algorithms
- Programming algorithms enables execution by computers
- For ease of understanding, we use abstractions

Arithmetic

- Programming languages already contain defined algorithms for recurring tasks
  - Evaluating expressions
  - Applying operations etc.

Simple evaluations

We want to evaluate this expression:

$$3 + 4 * 5$$

What can go wrong?

Simple evaluations

It is possible to erroneously evaluate "+" first and read the expression as "7 * 5".

$$3 + 4 * 5$$

How can we avoid such mistakes?

Simple evaluations

We can bracket the expression.

$$3 + (4 * 5)$$

Simple evaluations

We can also bracket the expression completely:

$$(3 + (4 * 5))$$

The meaning is still the same as at the beginning. But dependencies are now explicit.

By the way ...

Writing the operator between the operands is an arbitrary choice.

$$(1 + 2)$$

The operator can also be placed in front.

$$(+ 1 2)$$

Or re-enacted

$$(1 2 +)$$

By the way ...

Writing the operator between the operands is an arbitrary choice.

$$(1 + 2)$$

The operator can also be placed in front.

$$(+ 1 2)$$

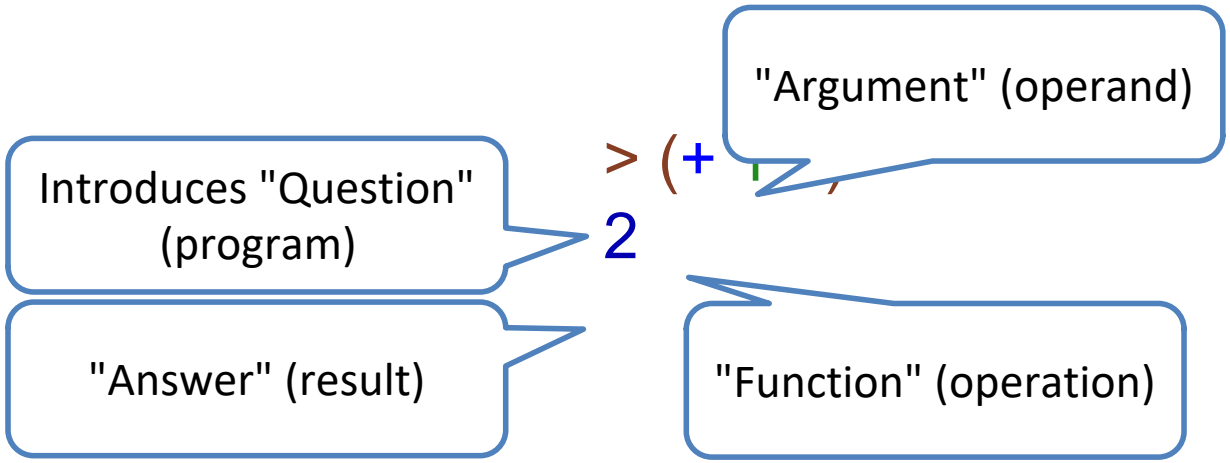
We already use "prefix notation" for some operations:
-5
sin(0)

Or recreated

$$(1\ 2\ +)$$

Arithmetic

- DrRacket programs are arithmetic expressions
- The result of the program is the evaluated expression
- DrRacket uses prefix notation throughout



Examples

```
> (+ 2 2)
4
> (* 3 3)
9
> (- 4 2)
2
> (/ 6 2)
3
> (sqr 3)
9
> (expt 2 3)
8
> (sin 0)
0
```

DEMO DrRacket

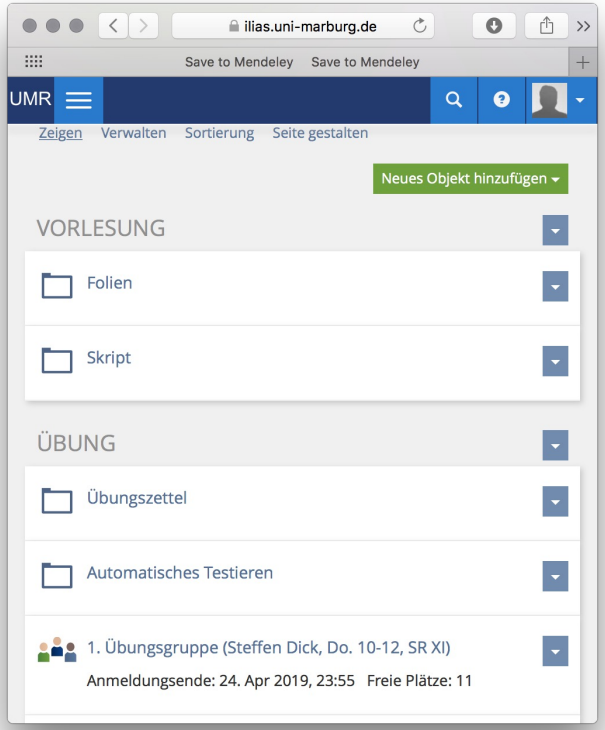
- Interaction area
- Definition range
- Save

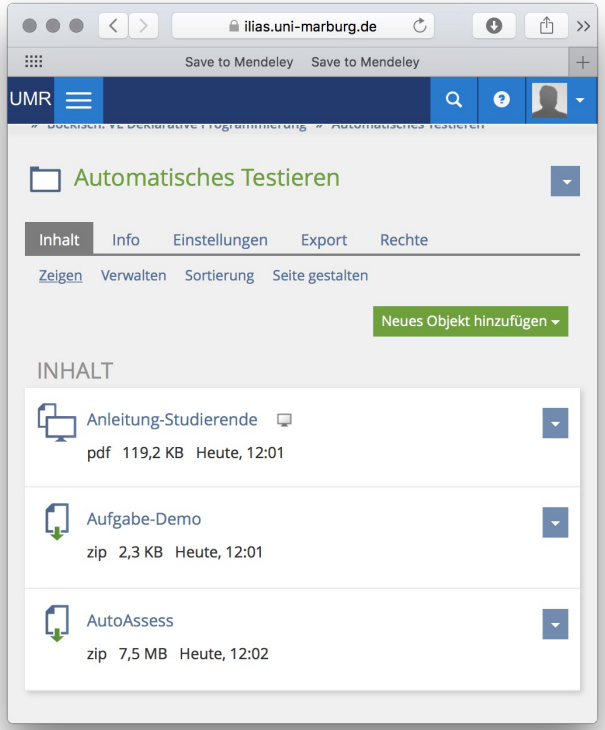
DEMO Automatic testing

- Task:

- Write a program consisting of
  - The function "double", which calculates the double of a number without using multiplication
  - And a call of the function for the argument 21

DEMO Automatic testing





Numbers are expressions

- Until now: arguments were numbers
- Numbers are expressions that are evaluated to themselves

  > 5

  5
- Numbers are "atomic" expressions

Compound expressions

- Expressions can be used as arguments

```
> (+ 2 (+ 3 4))
9
```

Compound expressions

- Expressions can be used as arguments

```
> (+ 2 (+ 3 4))
9
```

- Nesting of expressions can be arbitrarily deep

```
> (+ (* 5 5) (+ (* 3 (/ 12 4)) 4))
?
```

What is the result?

Compound expressions

- Expressions can be used as arguments

```
> (+ 2 (+ 3 4))
9
```

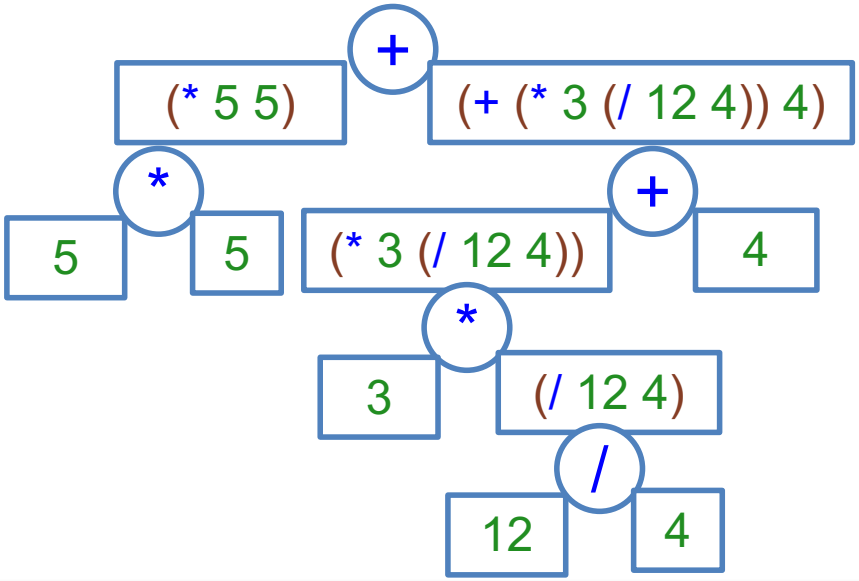
- Nesting of expressions can be arbitrarily deep

```
> (+ (* 5 5) (+ (* 3 (/ 12 4)) 4))
38
```

Evaluation of expressions

- Applying the function to the values of the arguments
- Argument is expression: first evaluate this expression ("recursive application" of the algorithm)
- The evaluation order of the sub-expressions is irrelevant.

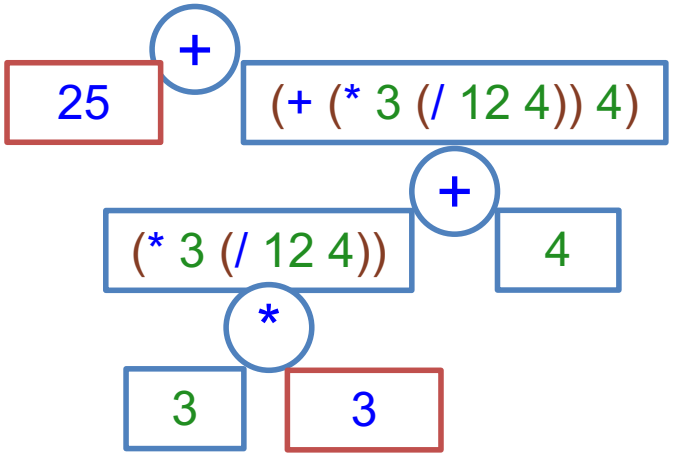
(+ (* 5 5) (+ (* 3 (/ 12 4)) 4))



Evaluation of expressions

- Applying the function to the values of the arguments
- Argument is expression: first evaluate this expression ("recursive application" of the algorithm)
- The evaluation order of the sub-expressions is irrelevant.

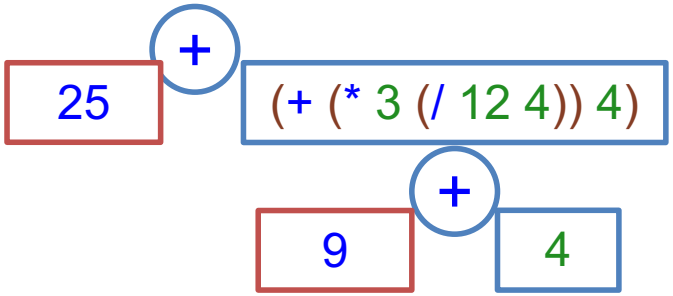
(+ (* 5 5) (+ (* 3 (/ 12 4)) 4))



Evaluation of expressions

- Applying the function to the values of the arguments
- Argument is expression: first evaluate this expression ("recursive application" of the algorithm)
- The evaluation order of the sub-expressions is irrelevant.

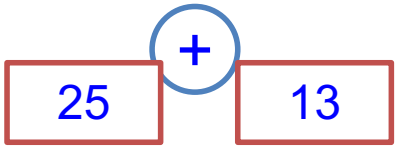
(+ (* 5 5) (+ (* 3 (/ 12 4)) 4))



Evaluation of expressions

- Applying the function to the values of the arguments
- Argument is expression: first evaluate this expression ("recursive application" of the algorithm)
- The evaluation order of the sub-expressions is irrelevant.

(+ (* 5 5) (+ (* 3 (/ 12 4)) 4))



Evaluation of expressions

- Applying the function to the values of the arguments
- Argument is expression: first evaluate this expression ("recursive application" of the algorithm)
- The evaluation order of the sub-expressions is irrelevant.

```
(+ (* 5 5) (+ (* 3 (/ 12 4)) 4))
```

38