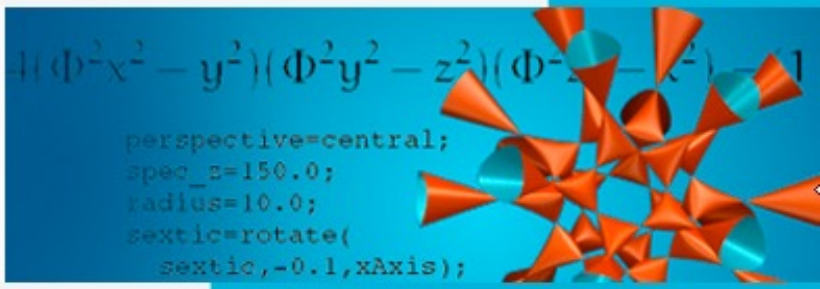
Philipps Universität Marburg

Declarative programming

Summer semester 2024

Prof. Christoph Bockisch, Steffen Dick
(Programming languages and tools)

Imke Gürtler, Daniel Hinkelmann, Aaron Schafberg, Stefan Störmer



[1.2 ,2.2-2.3]

Function definitions

- So far: predefined functions
- Functions can also be defined
  (use the definition area of DrRacket)
- Why define functions?

```
> (/ (+ 12 17) 2)
14.5
> (/ (+ 100 200) 2)
150
```

Function definitions

- So far: predefined functions
- Functions can also be defined
  (use the definition area of DrRacket)
- Why define functions?

```
> (/ (+ 12 17) 2)
14.5
> (/ (+ 100 200) 2)
150
```

Avoidance of redundancy

Better comprehensibility through abstract names

Function definitions

- Keyword "define" to differentiate between expressions

```
(define (average x y) (/ (+ x y) 2))
```

- Self-defined functions can be used in the same way as predefined functions

```
> (average 12 17)
14.5
> (average 100 200)
150
```

Function definitions

(define (FunctionName InputName1 InputName2 ...) BodyExpression)

- define
  - Keyword introduces function definition
- FunctionName
  - Name via which the function can be called
- InputName1, InputName2, ...
  - Names for the passed arguments
- BodyExpression
  - Expression that determines what the application of the function to the arguments means
  - Usually uses InputName1, InputName2, ...
  - Also "implementation" or "body" of the function

Function definitions

- Function definitions are not expressions
  - Cannot be evaluated
  - Cannot be passed as an argument

Calling user-defined functions

(FunctionName ArgumentExpression1 ArgumentExpression2 ...)

- FunctionName
  - Name of a defined function
- ArgumentExpression1, ArgumentExpression2, ...
  - Expressions to which the function is to be applied

- Calls to user-defined and predefined functions do not differ

Reduction with function definitions

- Reduction of an expression *e*

1. (PRIM) If *e* has the form (f v_1 ... v_n), f is a "**primitive**" function and the application of f to v_1 ... v_n has the value *v*, then (f e_1 ... e_n) → *v* applies.

2. (FUN) If *e* has the form (f v_1 ... v_n), f is **not a primitive** function and the "context" contains the function definition of f: (define (f x_1 ... x_n) e_{Body}), then (f v_1 ... v_n) → *eNewBody* applies, whereby *eNewBody is created* from e_{Body} by replacing all x_i with v_i (i = 1 ... n).

3. (KONG) If *e* has a sub-expression e_1 in an evaluation item with e_1 → e_1 ', then *e* → *e'* applies, whereby *e'* is generated from *e* by replacing e_1 with e_1 '.

Reduction with function definitions

- Reduction of an expression *e*

1. (PRIM) If *e* has the form (f v_1 ... v_n), f is a "**primitive**" function and the application of f to v_1 ... v_n has the value *v*, then (f e_1 ... e_n) → *v* applies

2. (FUN) If *e* has the form (f v_1 ... v_n), nction and the "context" contains the functi (define (f x_1 ... x_n) e_{Body}), then (f plies, whereby *eNewBody is created* from e_{Body} by i (*i* = 1 ... *n*).

3. (KONG) If *e* has in an evaluation item with e_1 then *e* → *e'* applies, whereby *e'* is generated from *e* by replacing e_1 with e_1 '.

This means "predefined" or "built-in".

Search space for all known definitions.

Context of function definitions

- Function definitions are part of the program
- Function definitions must be found when **evaluating expressions**
- Search in context
- Context = entire program up to the current printout

Context of function definitions

- Function definitions are part of the program
- Function definitions must be found when **evaluating expressions**
- Search in cont
- Context = entire program up to the current printout

Definition of functions is not an evaluation!
Therefore, function definitions may refer to functions that are defined later.

Context of function definition

- Convention: all definitions are at the beginning
- Then: Context = whole program
- This means that there is only one "global context"
- This simplifies the formal definition of the reduction rules

Example

```
(define (g y z) (+ (f y) y (f z)))
(define (f x) (* x 2))
```

(g (+ 2 3) 4)	
→ (g 5 4)	Rule KONG
→ (+ (f 5) 5 (f 4))	Rule FUN
→ (+ (* 5 2) 5 (f 4))	Rule KONG
→ (+ 10 5 (f 4))	Rule KONG
→ (+ 10 5 8)	Rule KONG
→ 23§	PRIM rule

Example

```
(define (g y z) (+ (f y) y (f z)))
(define (f x) (* x 2))
```

(g (+ 2 3) 4) — Rule KONG

→ (g 5 4) — Rule FUN

→ (+ (f 5) 5 (f 4)) — Rule KONG

→ (+ (* 5 2) 5 (f 4)) — Rule KONG

→ (+ 10 5 (f 4)) — Rule KONG

→ (+ 10 5 8) — PRIM rule

→ 23

(+ 2 3) → 5 Rule PRIM

Example

(define (g y z) (+ (f y) y (f z)))
(define (f x) (* x 2))

Replace y with 5 and z with 4.

(g (+ 2 3) 4)
Rule KONG
→ (g 5 4)
(g 5 4) □ Rule FUN
→ (+ (f 5) 5 (f 4))
Rule KONG
→ (+ (* 5 2) 5 (f 4))
Rule KONG
→ (+ 10 5 (f 4))
Rule KONG
→ (+ 10 5 8)
PRIM rule
→ 23

Example

```
(define (g y z) (+ (f y) y (f z)))
(define (f x) (* x 2))
```

Replace x with 5.

(g (+ 2 3) 4)

Rule KONG

→ (g 5 4)

Rule FUN

(f 5) □ Rule FUN

→ (+ (f 5) 5 (f 4))

Rule KONG

→ (+ (* 5 2) 5 (f 4))

Rule KONG

→ (+ 10 5 (f 4))

Rule KONG

→ (+ 10 5 8)

PRIM rule

→ 23

Example

```
(define (g y z) (+ (f y) y (f z)))
(define (f x) (* x 2))
```

(g (+ 2 3) 4) — Rule KONG

→ (g 5 4) — Rule FUN

→ (+ (f 5) 5 (f 4)) — Rule KONG

→ (+ (* 5 2) 5 (f 4)) — Rule KONG

(* 5 2) → 10 Rule PRIM

→ (+ 10 5 (f 4)) — Rule KONG

→ (+ 10 5 8) — PRIM rule

→ 23

Example

```
(define (g y z) (+ (f y) y (f z)))
(define (f x) (* x 2))
```

Replace x with 4.

(g (+ 2 3) 4) — Rule KONG

→ (g 5 4) — Rule FUN

→ (+ (f 5) 5 (f 4)) — Rule KONG

→ (+ (* 5 2) 5 (f 4)) — Rule KO

(f 4) □ Rule FUN

→ (+ 10 5 (f 4)) — Rule KONG

→ (+ 10 5 8) — PRIM rule

→ 23

Example

```
(define (g y z) (+ (f y) y (f z)))
(define (f x) (* x 2))
```

(g (+ 2 3) 4)	Rule KONG
→ (g 5 4)	Rule FUN
→ (+ (f 5) 5 (f 4))	Rule KONG
→ (+ (* 5 2) 5 (f 4))	Rule KONG
→ (+ 10 5 (f 4))	Rule KONG
→ (+ 10 5 8)	(+ 10 5 8) → 23 Rule PRIM
→ 23	

Definition sequence

```
(define (g y z) (+ (f y) y (f z)))
(define (f x) (* x 2))
(g (+ 2 3) 4)
```

Function definition may refer to function defined later.

Definition sequence

```
(define (g y z) (+ (f y) y (f z)))
(f 4)
(define (f x) (* x
```

Expression must **not access a** function defined later.

Definition sequence

```
(define (g y z) (+ (f y) y (f z)))
(g (+ 2 3) 4)
(define (f x) (
```

What happens?

A function that refers to another function may not be used in an expression if the other function is not yet defined.

Context contains g but not f.

Tasks

What is the result of the following programs?

1.
```
(define (g x y) (* (f x) (f y)))
(define (f x) (+ x 1))
(g 2 3)
```
   a) 12
   b) 5
   c) A mistake

2.
```
(define (g x y) (* (f x) (f y)))
(define (f x) (g x 1))
(define (h x) (/ x 2))
(h 6)
```
   a) 12
   b) 3
   c) A mistake

Live Vote

https://ilias.uni-marburg.de/vote/2O53

Data type: Images

- In DrRacket images are values
- Insert via copy/paste or insert → Insert image
- Images are literals and evaluate themselves

- Further functions for images

DEMO

Data type: Images

- In DrRacket images are values
- Insert via copy/paste or insert → Insert image
- Images are literals and evaluate themselves

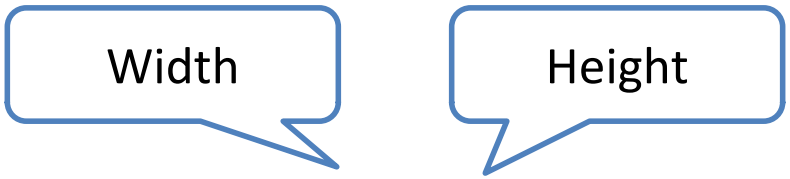
- Further functions for images

> (* (image-width  age-height)) 
600
> (circle 10 "solid" "red")

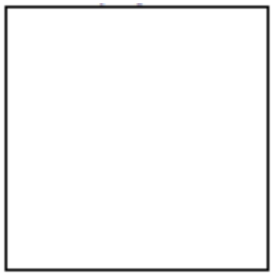
- Etc.

Useful functions for images

- A "Scene" allows images to be combined




```
>(empty-scene 100 100)
```



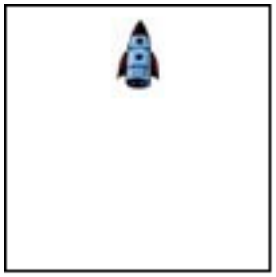
Useful functions for images

- Insert an image into a scene

From the left

From above

```
>(place-image  20 (empty-scene 100 100))
```



Useful functions for images

- Determine the size of an image

```
>(image-width  )
20

>(image-height  )
30
```

Useful functions for images

- Insert an image into a scene

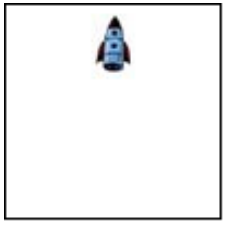
>(place-image  20 (empty-scene 100 100))

Reads like (almost) an English sentence:

place-the-imag  t x-position 50 and y-position 20 into an empty-scene with width 100 and height 100.

Program with pictures

```
>(place-image 50 20 (empty-scene 100 100))
```



```
>(place-image 50 40 (empty-scene 100 100))
```

```
>(place-image 50 60 (empty-scene 100 100))
```

Program with pictures

```
(place-image  20 (empty-scene 100 100))
(place-image  40 (empty-scene 100 100))
(place-image  60 (empty-scene 100 100))
```

The only difference

- Redundancy
  - Complex
  - Changes difficult

Function definitions avoid redundancy

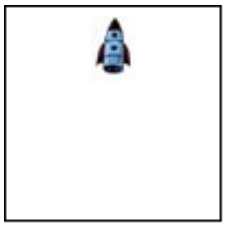
- Function definitions make commonality reusable

```
(define (create-rocket-scene height)
        (place-image 50 height (empty-scene 100 100)))
```

- Function definitions make differences to parameters

Program with pictures

```
> (create-rocket-scene 20)
```



```
> (create-rocket-scene 40)
```

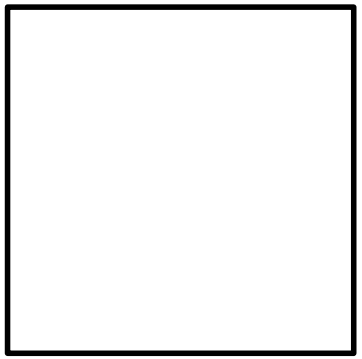
```
> (create-rocket-scene 60)
```

Easier to read:

```
create-rocket-in-scene at height 20.
create-rocket-in-scene at height 40.
create-rocket-in-scene at height 60.
```

Useful functions for images

- Animate a scene



```
>(animate create-rocket-scence)
138
```

Number of frames until abort.

Called 28 times per second with counter passed as argument.

Useful functions for images

- Animate a scene

```
>(animate create-rocket-scence)
138
```

Passing a function name as an argument. Formal meaning comes much later!