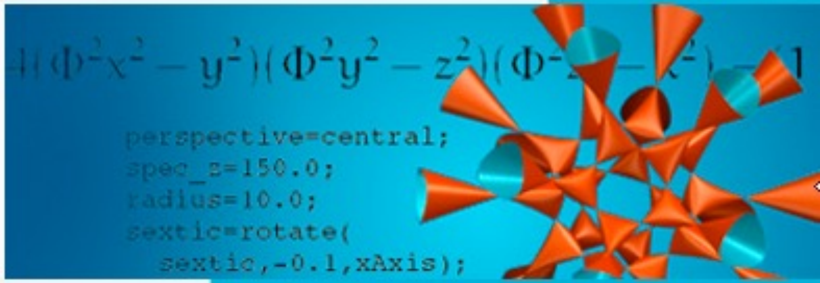
Philipps Universität Marburg

Declarative programming

Summer semester 2024

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

Imke Gürtler, Daniel Hinkelmann, Aaron Schafberg, Stefan Störmer



```
perspective=central;
spec_z=150.0;
radius=10.0;
sextic=rotate(
   sextic,-0.1,xAxis);
```

[Script 2.4 - 3.2]

Comments

- A semicolon introduces a comment
- Applies to the end of the line
- Is not processed by DrRacket and does not have to have a specific form
- Enter explanations for the code

```
Determines whether water freezes or not
(define (frost? temperature)
        (if (< temperature 0) true false))
```

Programmers design languages!

- Programming languages specify
  - Grammar
  - How the importance of a program is determined
- Programmers define the words of the language
  - Functions
  - Constants
- Programmers write "sentences" (expressions) in the language
- Goal: create a language so that your problem can be elegantly expressed in simple "sentences"

Don't Repeat Yourself (DRY)

- Avoid redundancies:
  - You don't want to keep repeating yourself in a story!
- Define functions for multiple calculation rules
- Define constants for values that occur more than once
- Use the definitions sensibly

Don't Repeat Yourself (DRY)

Case study:
See screen video or script

What does program design mean?

- Programming = writing down calculation rules
- Meaning of calculation rules either
  - Primitive (built-in function) or
  - User-defined (function defined in the program)
- User-defined definitions avoid redundancy
- Why should we also use our own definitions?

Recipe as programming

```
(cook
  (heat (fetch-from-cupboard oil))
  (fold-in
    (whisk (break (fetch-from-fridge eggs)))
    (mix
      (warm (fetch-from-fridge butter))
      (warm (fetch-from-tap water))
      (warm (fetch-from-cupboard milk))
      (sift salt)
      (fetch-from-cupboard flour))))
```

What is being done?

Hide details

- We can hide details using function definitions
- Replace "how" (implementation) with "what" (function name)
- Program becomes shorter, easier to understand

Example serial letter

```
(define (letter fst lst signature-name)
  (string-append
    (opening lst)
    "\n"
    (body fst lst)
    "\n"
    (closing signature-name)))
```

Salutation

Message

Greeting formula

```
> (letter "Tillmann" "Rendel" "Klaus Ostermann")
```

Example serial letter

```
(define (letter fst lst signature-name)
  (string-append
    (opening lst)
    "\n"
    (body fst l
    "\n"
    (closing sign
```

```
(define (opening lst)
  (string-append "Dear " lst ","))
```

```
(define (body fst lst)
  (string-append
    "After the last annual calculations of your GNB"
    "account activity we have determined that you, "
    fst lst
    ", are eligible to receive a tax refund of $479.30.\n"
    "Please submit the tax refund request (http://www...)"
    "and allow us 2-6 days in order to process it.")))
```

```
(define (closing signature-name)
  (string-append
    "With best regards,\n"
    signature-name))
```

Splitting the program

- One function per task
- Requirements change per task
- This minimizes the impact of changes
- Goal: Size of the program change proportional to the size of the request change

Example serial letter

```
(define (letter fst lst signature-name)
  (string-append
   (opening lst)
   "\n"
   (body fst l
   "\n"
   (closing sign
```

Change salutation → Only one small function needs to be changed.

```
(define (opening lst)
  (string-append "Dear Mr./Mrs. " lst ","))
```

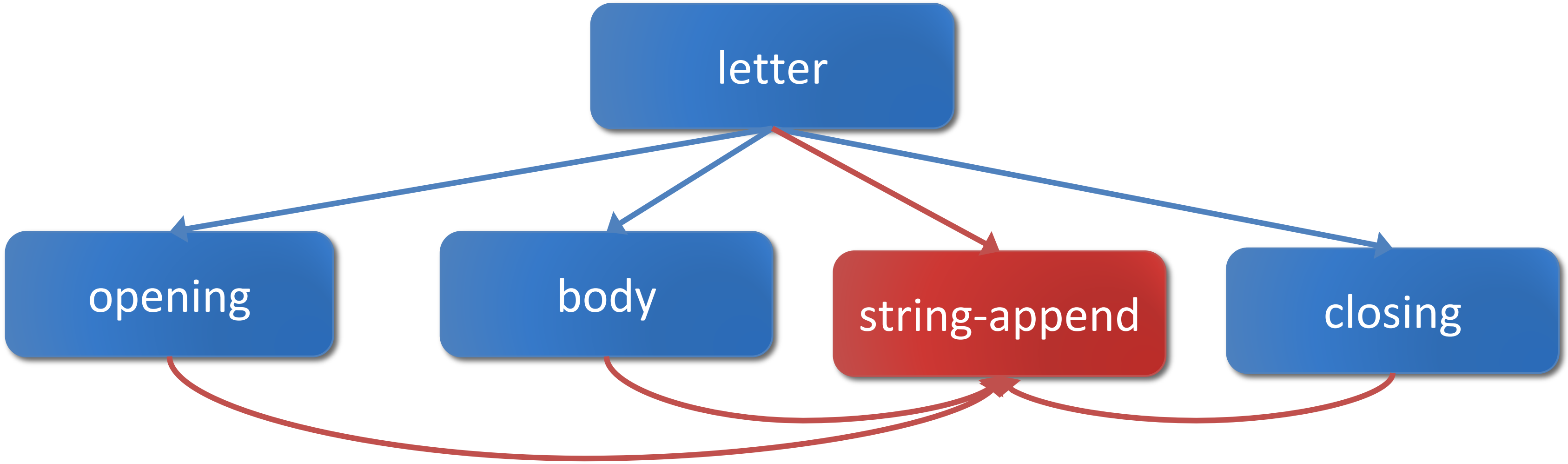
```
(define (body fst lst)
  (string-append
 "After the last annual calculations of your GNB"
    "account activity we have determined that you, "
    fst lst
    ", are eligible to receive a tax refund of $479.30.\n"
    "Please submit the tax refund request (http://www...)"
    "and allow us 2-6 days in order to process it.")))
```

```
(define (closing signature-name)
  (string-append
   "With best regards,\n"
   signature-name))
```

Splitting the program

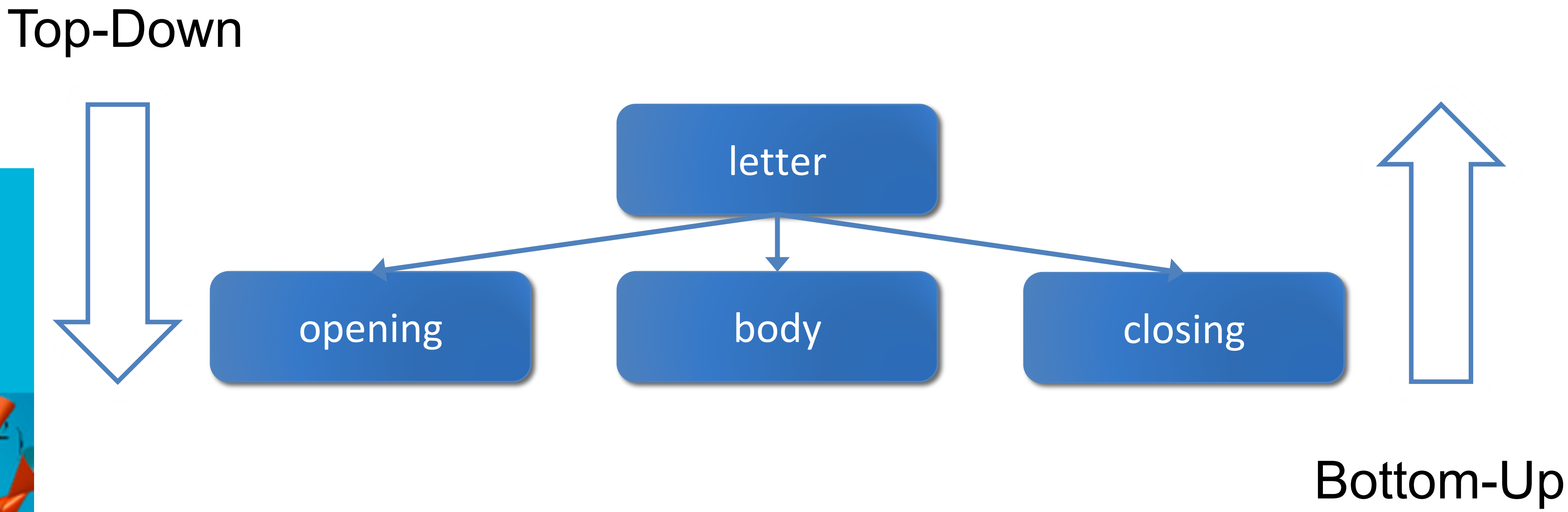
- Hierarchical
  - Top level: Describe the overall task by combining subtasks
  - Intermediate levels: Describe subtask by combining subtasks
  - Lowest level: Describe subtask by composing primitive functions

Hierarchical division



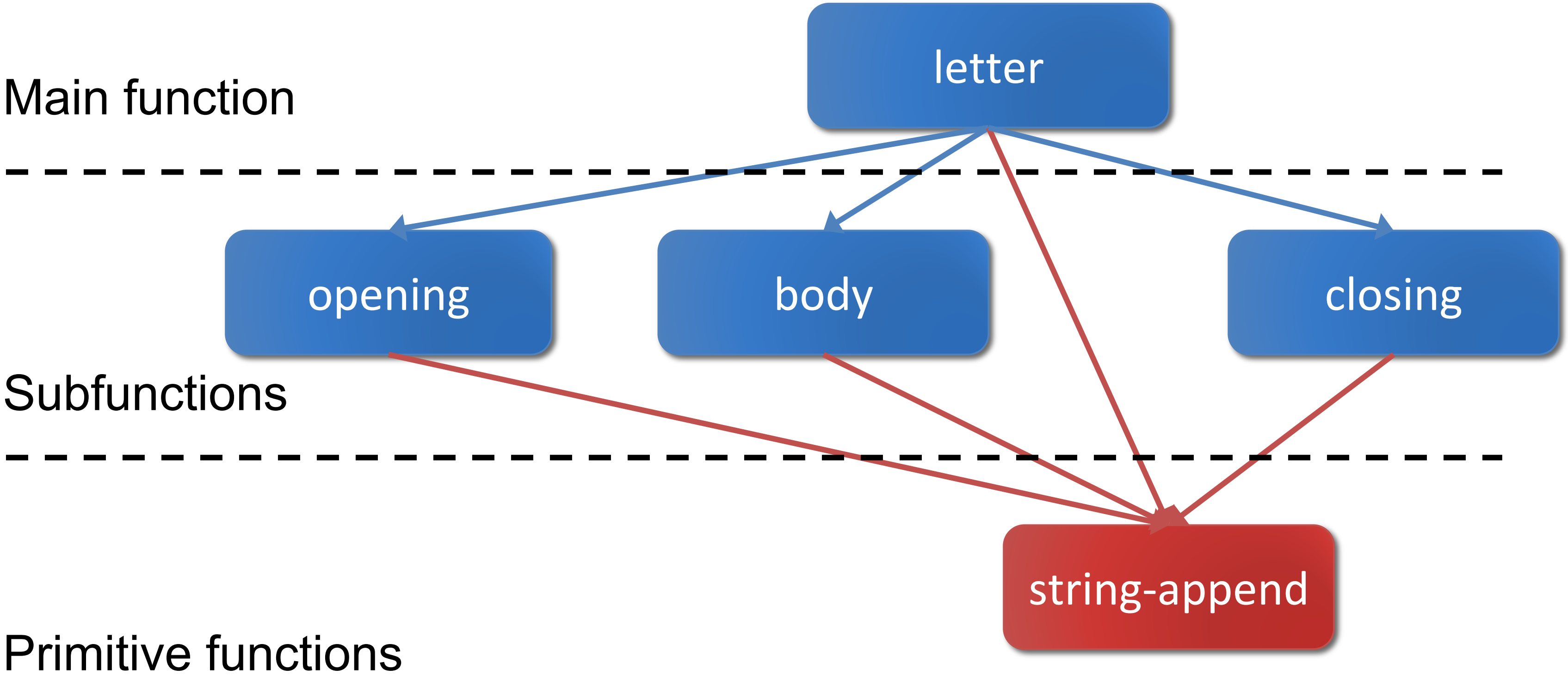
- Acyclic graph
  - Use of functions only lower in the hierarchy
- Excellent start node (root)
  - Main function
- Leaves
  - Use of primitive functions

Design approaches



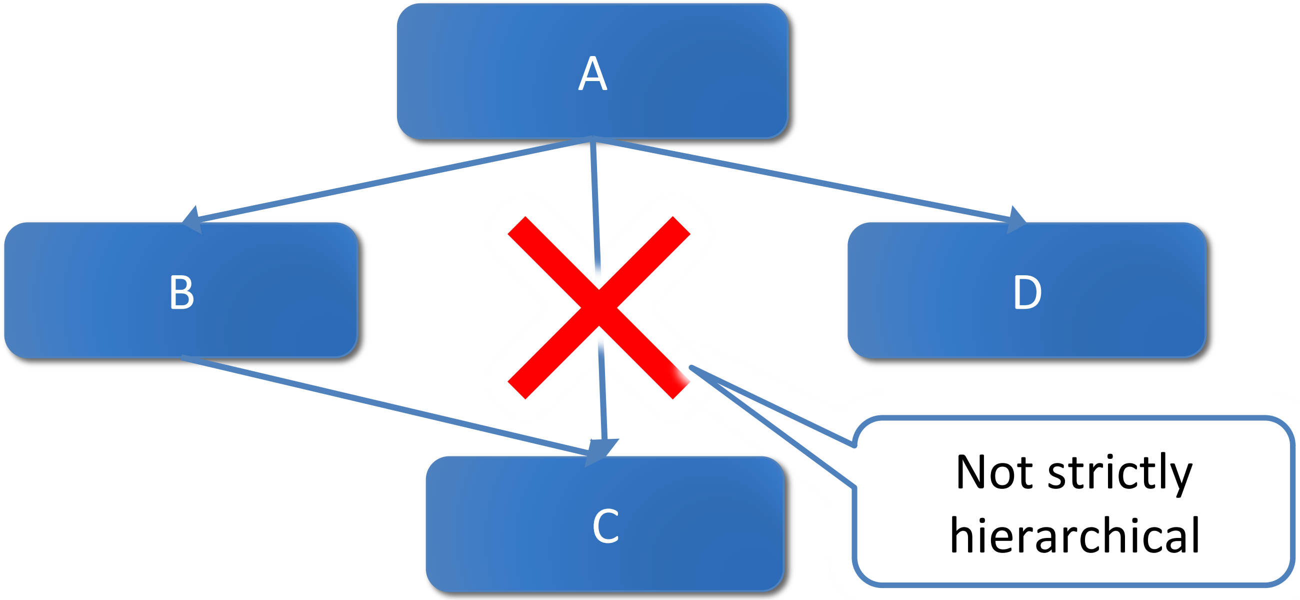
- Choice of approach Depending on the situation, e.g:
  - Is the "How?" important?
  - Are input/output for the sheet functions known?

Design in layers



Hierarchical abstraction

- Exclusively calling functions in the layer directly below



Not strictly hierarchical

Choose a clear function name! Documentation!

- No knowledge of the entire program necessary for understanding
- Only "abstract" knowledge of the functions used required

Documentation of functions

- Designation as documentation
  - Descriptive
  - Expressive power
  - Unambiguous
- Comment before function
  - Short description
  - Expected input (restrictions?)
  - Produced output

Software is in the maintenance phase for the longest time. Understanding the program is very important then!

From problem to program

The owner of a movie theater has complete freedom in setting ticket prices. The more he charges, the fewer the people who can afford tickets. In a recent experiment the owner determined a precise relationship between the price of a ticket and average attendance. At a price of $5.00 per ticket, 120 people attend a performance. Decreasing the price by a dime ($0.10) increases attendance by 15. Unfortunately, the increased attendance also comes at an increased cost. Each performance costs the owner $180. Each attendee costs another four cents ($0.04). The owner would like to know the exact relationship between profit and ticket price so that he can determine the price at which he can make the highest profit.

From problem to program

The owner of a movie theater has complete freedom in setting ticke arges, the fewer the people who cent experiment the owner determined a precise re nship between the price of a ticket and average attendance. At a price of $5.00 per ticket, 120 people attend a performance. Decreasing the price by a dime ($0.10) increases attendance by 15. U ed attendance also comes at an ir rmance costs the owner $180. E ner four cents ($0.04). The owner would like to know the exa rofit and ticket price so that he can ich he can make the highest prof

Sub-tasks clearly defined.

Sub-tasks clearly defined.

Solution approach still unclear.

From problem to program

- Overall solution approach still unclear
- Sub-tasks clearly defined

- → Bottom-up design

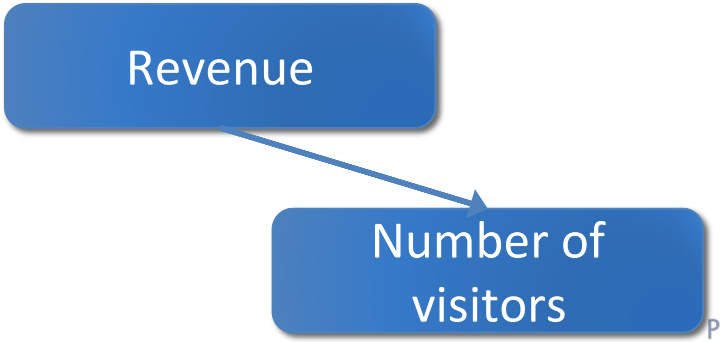
From problem to program

```
Number -> Number
; compute the number of attendees that
; pay the given ticket-price
(define (attendees ticket-price)
         (+ 120 (* (/ 15 0.1) (- 5.0 ticket-price))))
```

Number of visitors

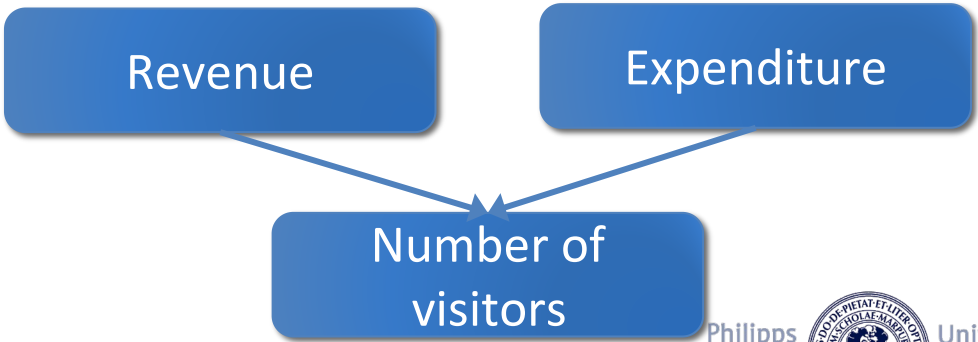
From problem to program

```
Number -> Number
; compute the revenue based on the ticket-price and the
; number of attendees which depends on the ticket-price
(define (revenue ticket-price)
        (* (attendees ticket-price) ticket-price))
```



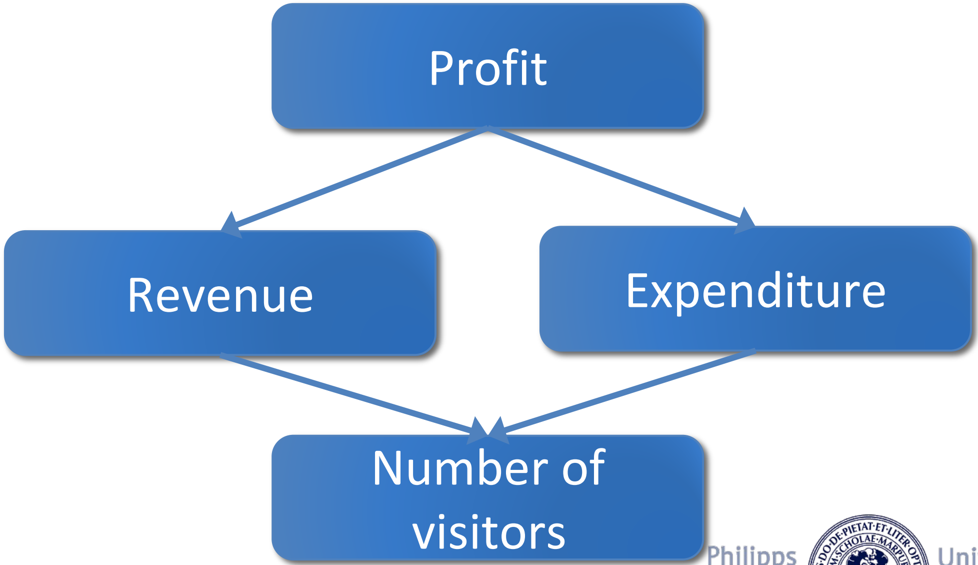
From problem to program

```
Number -> Number
; compute the total costs for the show including the variable
; costs depending on the number of attendees determined
; by the ticket-price
(define (cost ticket-price)
        (+ 180 (* 0.04 (attendees ticket-price))))
```



From problem to program

```
Number -> Number
; compute the profit depending on the specified ticket-price
(define (profit ticket-price)
         (- (revenue ticket-price)
                  (cost ticket-price)))
```



Profit

Revenue

Expenditure

Number of visitors

From problem to program

```
(define (profit price)
  (- (* (+ 120
          (* (/ 15 0.1)
             (- 5.0 price)))
        price)
     (+ 180
        (* 0.04
           (+ 120
              (* (/ 15 0.1)
                 (- 5.0 price)))))))
```

Without abstraction:
General meaning
obscured;
Redundancies