PP2

Recursion and Higher-order functions

The goals of PP1, lecture 2

- Knowledge about
  - Recursion and iteration, including the handling of tail calls (tail recursion).
  - Currying
  - Higher-order functions, including mapping, filtering and reduction/accumulation/folding.
- Skills:
  - To be able to use higher-order functions as a programming technique in Scheme
  - To be able to use recursive programming techniques to express algorithms in Scheme

Please register your groups for submission

- Get the link in PP Moodle News forum

This Zoom session is not recorded

Preparation for the exercises in groups

- Please register for group exercises
- See link in News Forum

Topics today

- Tail recursion
  - Iteration by means of recursion
- Higher-order functions
  - map, filter, reduce
- Currying

Tail calls in Scheme

- Must be handled in a memory-efficient way

Higher-order functions

- Functions that accept functions as parameters
- Functions that return a function

- The real power of functional programming

- Abstract recurring recursive patterns
  - Mapping, filtering, folding, …

So currying is for us just just an academic curiosity?

So currying is for us just just an academic curiosity?

NO

Currying is a novel way to produce functions

(define (f a b) …)

(f x) gives an *error*

(f x) returns a function which accepts the second parameter

In principle...

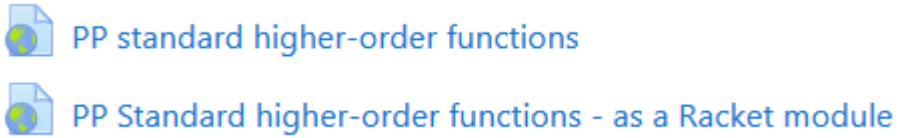
Currying

```
(define every-second-element
  (every-nth-element 2))

(define every-second-element
  (lambda (lst)
    (every-nth-element 2 lst)))

(define every-second-element
  ((curry2 every-nth-element) 2))
```

Exploration in REPL session:
currying-motivation-every-second-element.scm

Standard Higher-order functions in Scheme?

- Depends on which Scheme system you use
- R5RS only have few of the functions we have studied in today's lecture



- I have arranged a set of standard PP Scheme higher-order functions.
  - Available from Moodle
    - Also available as a Racket module
  - Can/should be used in the first PP miniproject

Now exercises

15 minutes

Numerical differentiation

- Given `f`: Real -> Real
- Write the *higher-order function* `derivative` such that `(derivative f)` and `f'` are close to each other.

$$\frac{f(x + delta) - f(x)}{delta}$$

- Comparare `f'` and `(derivative f)` by mapping each of them on a list of sample input values.

25 minutes

C-style compare functions

- Generate a C-style copare function from *less than* and *greater than*
  - And the other way around
- C_style_compare(x, y)
  - Negative if x is considered less than y (-1)
  - 0 if x is considered equal to y
  - Positive if x is considered larger than y (1)

20 minutes

replicate-to-length

```
(replicate-to-length '(a b c) 8) => (a b c a b c a b)

(replicate-to-length '(a b c) 2) => (a b)
```

25 minutes

for-all and there-exists

```
(for-all lst p)

(there-exists lst p)

(there-exists-1 lst p)
```

self-compose* and compose*

```
(self-compose* f n)

(compose* fn-lst)
```

Ad hoc currying in Scheme

Exploration in REPL session:
curry-generalized.scm

Coding Style

- Layout of special forms
  - define, lambda, if, cond, let, …
- Level of indentation
- Pending parentheses
- Breaking of function calls with many (and long) parameters

Coding Style

- REPL EXAMPLES: Coding-style.scm