CSE 341
Lecture 7

anonymous functions; composition of functions
Ullman 5.1.3, 5.6

slides created by Marty Stepp
http://www.cs.washington.edu/341/
• Define an operator \( \min -- \max \) that will produce a list of the integers in the range \([\min, \max]\) inclusive.
  - Example: \(2--7\) produces \([2, 3, 4, 5, 6, 7]\)
    
    *(We'll use -- as a helper for several later examples.)*

• Solution:
  
  ```plaintext
  infix --;
  fun min -- max =
    if min > max then []
    else min :: ((min+1) -- max);
  ```
Anonymous functions (5.1.3)

\[ \text{fn } \text{parameter(s)} \Rightarrow \text{expression} \]

- Example:
  - `map(fn x => x+1, [2, 0, 9, ~3]);`
  
    `val it = [3,1,10,~2] : int list`

- allows you to define a function without giving it a name
- useful with higher-order functions e.g. map/filter/reduce

- `fun name...` is the same as `val name = fn...`
Pascal's triangle exercise

• *Pascal's triangle* is a sequence of numbers of the form:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

• Define a function `triangle` that takes an integer `n` and produces a list of the first `n` levels of the triangle.

  - `triangle(6)` produces `[[1], [1,1], [1,2,1], [1,3,3,1], [1,4,6,4,1], [1,5,10,10,5,1]]`
Pattern of numbers

- The values at the two ends of a row are always 1.
- An interior number is the sum of the two values above it:
  - value at (row \( n \), col \( k \)) = value at \((n-1, k-1)\) + value at \((n-1, k)\)

<table>
<thead>
<tr>
<th>row</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

- Can we turn these observations into a helping function?
Binomial coefficients

• the numbers in Pascal's triangle also relate to binomial coefficients, or "\( n \) choose \( k \)" combinations:

\[
\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \text{for all integers } n, k > 0,
\]

\[
\binom{n}{0} = 1 \quad \text{for all } n \in \mathbb{N}, \quad \binom{0}{k} = 0 \quad \text{for all integers } k > 0.
\]

• Use the following function as a helper:

(* returns \( n \) choose \( k \) *)

```haskell
fun combin(n, k) =
    if k = 0 orelse k = n then 1
    else if k = 1 then n
    else combin(n - 1, k - 1) + combin(n - 1, k);
```
The triangle function

- The overall triangle consists of rows of the form:
  - $[r \text{ choose } 1, \ r \text{ choose } 2, \ ... , \ r \text{ choose } r]$

- To produce a triangle of $n$ levels:
  - for each number $r$ in the range 1 through $n$,
    - for each number $k$ in the range 1 through $r$,
      - compute $(r \text{ choose } k)$. put all such values together into a list.
triangle solution

(* Returns level r of Pascal's triangle (1-based). *)
fun makeRow(r) =
    let fun rChoose(k) => combin(r, k)
    in  map(rChoose, 1--r)
    end;

(* Returns the first n levels of Pascal's triangle. *)
fun triangle(n) = map(makeRow, 1--n);

(* Version that uses anonymous functions *)
fun triangle(n) =
    map(fn(r) => map(fn(k) => combin(r, k), 1--r), 1--n);
Exercise

• Write an ML expression that produces the square roots of the integers from 1-100, rounded to the nearest integer.
  - Write it as a one-line expression without let or fun.
    
    \[ [1,1,2,2,2,2,3,3,3,3,3,3,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,7,7,7,7,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,10,10,10,10,10,10,10,10,10,10,10,10,10,10] : \text{int list} \]

• Solution:
  
  \[
  \text{map}(\text{fn}(n) => \text{round}((\text{Math.sqrt})(\text{real}(n)))) , 1--100);
  \]
• The preceding code is really just a combination (composition) of other existing functions.
  - round(Math.sqrt(real(n)))

• Consider the following function. How could we use it?

  (* Produces a new function H that calls G and F. *)
  fun compose(F, G) =
    let fun H(x) = F(G(x))
    in  H
    end;
Composition operator, \( \circ \) (5.6.2)

\[ \text{function}_1 \circ \text{function}_2 \]

- the \( \circ \) operator is similar to our compose function
  - \textbf{val} \: H = F \circ G;
    produces a new function \( H \) such that \( H(x) = F(G(x)) \)

- function composition is so important that most functional languages include a convenient syntax for it
Composition exercise

• Write an ML expression that produces the square roots of the integers from 1-100, rounded to the nearest integer.
  ▪ Use function composition with the o operator.

• Solution:
  \[
  \text{map} \left( \text{round } o \text{ Math.sqrt } o \text{ real}, 1--100 \right);
  \]
• Define a function `squareWhole` that takes a list of reals and produces the squares of their integer portions.
  - (a one-liner using composition and higher-order functions)
  - Example:
    \[
    \text{squareWhole([3.4, 1.7, 5.8, 10.6]) produces [9.0,1.0,25.0,100.0]}
    \]

• Solution:
  ```
  fun squareWhole(lst) = 
    map(real o (fn(x) => x*x) o trunc, lst);
  ```