csci54 – discrete math & functional programming

tuples and lists
Recap

- Write a function \( \text{cap}' \) that not only caps the upper limit at 100, but additionally evaluates to 0 if \( n \) is less than or equal to 0.

- Write a function \( \text{pow} \) that takes two parameters \( n \) and \( k \) and returns \( n \) to the \( k \)th power. (assume that \( k \) is guaranteed to be a non-negative integer. do not use the ** operator)

\[
\text{cap}' \ n = \\
\quad \text{if } n > 100 \\
\quad \quad \text{then } 100 \\
\quad \text{else} \\
\quad \quad \text{if } n < 0 \\
\quad \quad \quad \text{then } 0 \\
\quad \quad \text{else } n 
\]

\[
\text{cap } n = \\
\quad \text{if } n > 100 \\
\quad \quad \text{then } 100 \\
\quad \text{else } n 
\]

\[
\text{pow } n \ k = \\
\quad \text{if } k == 0 \\
\quad \quad \text{then } 1 \\
\quad \text{else } n \times (\text{pow } n \ (k-1)) 
\]
maxInt

- write a function `maxInt` that takes a list of integers and returns the value of the largest element. You may assume the list is not empty.

\[
\begin{align*}
\text{maxInt} \ [x] &= x \\
\text{maxInt} \ (x:xs) &= \text{max} \ x \ (\text{maxInt} \ xs)
\end{align*}
\]
Lists in Haskell

- Homogenous (all same type)
- square brackets with element separated by commas
- building lists
  - square brackets with values separated by commas

```ghci
ghci> aList = [1, 10, -3, 5]
```

- cons

```ghci
ghci> aList2 = 2 : [1, 10, -3, 5]
```

- concatenation

```ghci
ghci> aList3 = aList ++ aList2
```
Lists in Haskell continued

- functions on lists
  - head, tail
  - init, last
  - take, drop
  - length, null
  - reverse
  - ...

- `'elem` vs elem
  - infix vs. prefix
  - same with arithmetic functions: div, mod
    - div: round down
    - mod: integer mod (goes with div)

```
aList = [2, 1, 10, -3, 5]
elem 1 [2, 1, 10, -3, 5]
elem 1 [2, 1, 10, -3, 5]
1 `elem` [2, 1, 10, -3, 5]

(Haskell also has quot, rem, which behave differently than div/mod with negative numbers)
```
Practice problems

- what does this function do?

```haskell
numList n = 
    if n <= 0 
    then [] 
    else 
        n : (numList (n-1))
```
Practice problems

- (on week01-ps) numList n evaluates to a list of integers from n down to 1

```
numList n = 
  if n <= 0 
  then [] 
  else 
    n : (numList (n-1))
```

- Write a function oddList where oddList n evaluates to a list of odd integers from n down to 1. If n < 1 the function should return an empty list.

- Write a function oddList' where oddList' evaluates to a list of odd integers from 1 up to, but possibly not including, n. If n < 1 the function should return an empty list. Do not use the reverse function.
what does the following do?

```python
aList = [2, 1, 10, -3, 5]
aList = 2:aList
```
List comprehensions (and ranges)

- A way to build up lists:

\[ \{ x^2 | x \in [1..3] \} \]

- Note use of ranges in Haskell

\([ 1, 3..10 ]\)
\([ 10, 9..1 ]\)
\([ 1, 4..]\)
\([ 47.. ]\)

- Can add more to list comprehensions:

\([ x \cdot y | x \in [1..3], y \in [6, 4, 2] ]\)
\([ x \cdot y | y \in [6, 4, 2], x \in [1..3] ]\)
More on list comprehensions

- Can add predicates:

  \[ [ x*y | x <- [1..3], y <- [1..3], x > y] \]

- Can use any expression:

  \[ [ if x*y > 3 then "BIG" else "SMALL" | x <- [1..3], y <- [1..3]] \]

  \[ [ (x,y) | x <- ['a'..'c'], y <- ["rat","ox","tiger"] ] \]

- A tuple does not need to be homogenous. Cannot append or concatenate, so must know number of elements from start.
Practice problems

- Write a function `oddList` where `oddList n` evaluates to a list of odd integers from `n` down to 1. If `n < 1` the function should return an empty list.

- Write a function `oddList'` where `oddList' n` evaluates to a list of odd integers from 1 up to, but possibly not including, `n`. If `n < 1` the function should return an empty list.

- Rewrite `oddList` and `oddList'` using list comprehensions.

- What do these evaluate to?

```plaintext
[ if x*y > 3 then [1] else [2] | x <- [1..3], y <- [1..3] ]
[ (x,y,z) | x <- [1..3], y <- [1..3], z <- [1..3], x < y, y < z ]
[ (x,y,z) | z <- [1..3], y <- [1..3], x <- [1..3], x < y, y < z ]
```