*Is this language regular?

a … z
must be an equal number of a's and z's
such as: az, aazz, azaz
-No way to count/keep track in a regular language!
-Pumping Lema for Regular Languages: (NOT ON EXAM)

> Let L be a regular language, there exists some number P >= 1, depending on L such that for every word in the language where the length of w >= P, W = xyz with the length of y has to be at least 1, the length of x &y has to be <= P, and for every i > 0, (x&y) repeated i times followed by z has to be within L

-So assume a language is regular until it violates the Lemma, so come up with a W that is big enough that if you break it apart you are in trouble!
-What might I pick for W? :

> W = a(a)^P (z)^P z = a billion a's followed by a billion z's
> = xyz
> x = aaa…aa y = (aaa…)Z
> Violates the Lemma!

-Understand difference between regular language and context free language
-Can this be a rule in a context free grammar aS —> Sa?
-NO! because of the left hand side
-Regular Grammar: N —> aN | empty | a | N
-CFG: N —> Anything Goes

-ML stuff up to patterns

Chs 1-(6 or 7)

-Lexing and parsing and stuff, show you one and tell me what it does

-Environments:
-Mapping of variables to values/types
-Interface:
-Create an empty environment w/o bindings (no pairs)—> empty-env
-Extend an environment (add a new binding to existing environment) —> ended-env
-Apply an environment (look up a value/type for a variable) —> apply-env
((x 5) (y 6) (a 10) empty-env)

-fair number of build a racket/ML function
-process a list, cons stuff
-No more than 10 lines of code
-Concept questions: ambiguity of a grammar, build me a parse tree, here is a language build me a grammar
-Whats a linker, lexer, parser

-Program:

-Text: Executable

Lexical analysis —> Parsing

(did you build program text that is syntactically correct?)

-Lexical Analysis outputs a list of tokens which is sent to the next step (Parsing)

-Are the individual words correct? Has nothing to do if they are put together in the right way —> about individual words

-Parsing: about "phrase structure", are the words in a correct syntax order? such as : did you say "if(bool expression) { }" correctly

-How does Cons work? :

-Cons as a function takes in two things (cons 'a '(b c)) —> '(a b c) —> List

-(cons 'a 'b) forms a pair —> a.b —> dotted pair/improper list (NOT A LIST!)

-The second thing that you gave it is not a list, so it cannot build a list from it!

-Cons Cell: In memory is two things paired together, a structure that has two elements.

-Removing ambiguities from Grammars:

<S> ::= <ifStatement> l S1 l S2 (S1 and S2 are Terminals)

<ifStatement> ::= if <expr> then <S> else <S> l if <expr> then <S>

<expr> ::= expr1l expr2

Is this ambiguous or not?

if e1 then l if e2 then S1 else S2

```
<S>
        \
        <ifStatement>    —\      —- <S>
        /  l        l   \  else       \
 if      <expr>   then <S>               <S2>
           l           /
        e1           if, <exp>, then , <S>
                         l               l
                         e2              S1
```

So then: if, e1, then, if, e2, then ,s1,else, s2

-Associativity of the Uniary Operator:

not (not b) = (not (not b)) —> right associativity

not b and c

can be: not (b and c)

or: (not b) and c

ML:

```
fun f n = n * n;
val f = fn : int -> int
```

```
f 5;
val it = 25 : int
```

```
fn n  => n * n;  (temporary function)
val it = fn : int -> int
```

-Map: applies a function to everything in a list

```
map (fn n  => n*n) [1,2,3,4];
          (maps the function to everyone of the items in the list)
val it = [1,4,9,16] : int list
```

```
map (fn n => if n > 4 then true else false) [1,4,9,16];
val it = [false, false, true, true] : bool list
```

```
map (fn n => n +4) [1,4,9,16];
val it = [5,9,13,20] : int list
```

-Fold: goes through a list and applies an operation to the list as you go
give it a starting value and an operation and it will apply the operation as it goes

foldl - start at the left and work right
foldr - start at the right and work left

```
foldl (op +) 0  [1,2,3,4];
val it = 10 : int
  (0+1) + (0+2) + (0+3) + (0+4) = 10
```

```
foldr (op +) 0 [1,2,3,4,5,6];
val it = 21 : int
(0+6) + (0+5) + (0+4)….
```

-Map is a “curried” function, it takes multiple arguments one at a time. It takes in a function and gives you back a function, and then applies the function. You take two parameters but one at a time.

-Fold takes in an operator, returns a function that knows how to use a function. Takes in the base case, returns a function that knows how to apply the base case. Then finally returns a function that knows how to apply them to the list

-Given a list of ints, convert the list of ints to a list of real values: 1 —> 1.0

```
fun convert aList  = map real aList;
```

```
convert[1,2,3,4] ——> [1.0, 2.0, 3.0, 4.0]
```

-Square a List:

```
fun sqList aList = map (fn n => n*n) aList;
```

-Increment a List:

```
fun incList inc aList = map (fn n => inc +n) aList;
```

-Calculate Length: (answer will be an int, not a list. For this reason we use a fold instead of a map)

```
fun len aList = foldl (fn (a, b) => a + 1) 0 aList;
```

```
len [1,2,3] = 4
```