CAS CS 320
Lec 1

Introduction to SML

1. SML
    a. Standard ML (Meta Language - Language for writing scripts for theory)
        i. Language is formally specified (unlike Python)
        ii. Not practical language
    b. Ocaml is an example that is becoming famous
    c. Jane Street uses Ocaml
    d. Compiler → SML/NJ (New Jersey)
        i. Printing is difficult (have to implement the function)
    e. Functional Programming Language
        i. Difficult to move around
        ii. Try to minimize the update of variables (can no longer change the binding)
        iii. X = 1 (X is name, does not have a memory location, and no way to find the memory address of X)
    f. Example 1 → variables
        i. val x = 1 (* declaration *) - ML has series of declarations
        ii. val y = x + x (* declaration *)
        iii. val z = y + z → z is unbounded name (using variable that is not introduced, declared)
        iv. val x = x + 1 → introducing the name twice (is allowed) → shadowing (in ML, they are different variable xs → two xs exist)
        v. Since those variables cannot coexist, compiler can use one memory location for x (compiler has a choice)
    g. Example 2 → functions
        i. fun int_double(x) = x + x (no return in functional programming)
        ii. On the right-hand side, there is big notation
        iii. Every function only takes one argument
        iv. If want to take arguments, we can submit a pair that contains two variables
        v. fun int_add(x,y) = x + y
        vi. This function takes one argument → the argument happens to be two variables
    h. Example 3 → functions (recursion)
        i. fun fact(x) =
            if x > 0 then x * fact(x-1) else 1
        ii. In the body of the function, we call the function being defined (recursion)

    iii. In ML, the two fact functions are always the same (unlike Python)
    iv. New functions can call previously defined functions
  i. Allocate frame for each recursive call (every time fact is called) in Python
  j. If such pattern continues, it will hit the bottom of the stack (if you call fact many times) → difficult bug to fix (not asked much to use recursion)
  k. ML is different (it handles recursion differently) → standard ML has more space to handle recursion
2. Lambda
  a. Convenient feature in Python (expression that represents a function)
  b. Example
    i. fun intdbl(x) = x + x (syntactic sugar of the following code below)
    ii. val intdbl = fn(x) => x + x (anonymous function)
    iii. In python, intdbl = lambda x: x + x
3. Git code
  a. val x = 1
  b. = + 2
  c. ; (finish one line)
  d. val x = 7 : int (automatically done by the compiler)
  e. x + 7;
  f. val it = 14 : int (automatically done by the compiler) ← it is a name of variable given by the compiler if not defined previously
  g. fun intdbl(x) = x + x;
  h. val intdbl = fn : int -> int
  i. intdbl(intdbl(x)); → can call function within function
  j. val it = 56: int
4. Factorial example
  a. fun fact(x) = if x > 0 then x * fact(x-1) else 1
  b. If x == 13 in this computer, 13 yields an exception Overflow (int value is too large to be stored) → doing this one by one to check what value of x yields an error is time consuming
  c. fun myloop(x: int): int = (fact(x); myloop(x+1))
  d. myloop(0) → myloop starts from 0 and continues until there is an error → it is essentially a loop
  e. Still inefficient since it prints 0 to 12 until 13 gives an error
  f. fun myloop(x: int): int = (fact(x); myloop(x+1)) handle Overflow => x
  g. Now, myloop(0) only gives val it = 13: int
5. Git
  a. Use "./../assign00-01.sml"; → gets all functions from the file assign00-01.sml