Functional Programming (FP) -- Scheme with a Python-like syntax	John Backus: Can Programming Be Liberated from the von Neumann Style? (1977 Turing Award Lecture)	
A program is a sequence of "defs" followed by an expression.	def ...; ... def ...; expression	
simplest program	a single expression	

expressions:
(from eval.scm in /home/jlu/public_html/cs482/share)
	(define eval_expr
	  (lambda (exp)
	  (cond ((number? exp) exp)
	  ((sum? exp)
	  (+ (eval_expr (subexp1 exp))
	  (eval_expr (subexp2 exp))))
	  ((difference? exp)
	  (- (eval_expr (subexp1 exp))
	  (eval_expr (subexp2 exp))))
	  ((product? exp)
	  (* (eval_expr (subexp1 exp))
	  (eval_expr (subexp2 exp))))
	  ((quotient? exp)
	  (/ (eval_expr (subexp1 exp))
	  (eval_expr (subexp2 exp))))
	  (else (error eval_expr "invalid expression -- " exp))))))	

names: add an *environment* to stores name-value	;; a name evaluates to the value associated with the name in ;; the environment ;; associate list (i.e., dictionary)	

boolean expressions (comparison):
 	
  +x y <= 1
	(define eval_bool
	  (lambda (bexpr env) ...)	

conditional expression
 	
if (bexpr):
expr1
else
expr2
	;; if bexpr evaluates to true, return the evaluation of expr1
	;; otherwise return the valuation of expr2	

defines: (sequential or collateral)	(define eval_def (lambda (def env) ...)	
expression: def a: expr;	;; add to the current environment the association (a v) where ;; v is the evaluation of expr	
function: def f(x,y): expr;	;; add to the current environment the association (f c) where ;; c is the "closure" that when called, evaluates expr in the ;; current environment, modified by replacing the parameters ;; with the corresponding values of the actual parameter	((function? def) (create-closure (params-of def) (body-of def) env))
function call: f(e1, e2)	;; evaluate the actual parameters in the current environment, ;; pass the resulting values to the closure associated with f	(let ((cl (assoc (name-of expr) env)) (actuals (eval_args (params-of expr) env))) (apply-closure cl actuals))
		apply closure: - add or replace the values of the formals with the actuals in the env of the closure ==> evaluation environment - call eval_expr with the body of function in the evaluation environment
Issues: visibility/scope rules of names - collateral vs linear definition - dynamic vs static scope rule	work out examples on the board	