

CSC 324 2006W
ASSIGNMENT 1

- (1) Suppose we want to say
 `(sqrt (+ (squares (list 3 4))))`
to get the size of a vector represented by a list.
- (a) Define `pluralize` that:
- Takes a single argument: a procedure `p` taking one argument.
 - Returns a procedure of one argument that given a list returns a list of the elements transformed by `p`.
- (b) Define `squares` using `pluralize`.
- (c) Define `auto-apply` that:
- Takes a single argument: a procedure `p`.
 - Returns a procedure that:
 - Given a list, returns (the result of) `p` called with the elements of the list as its arguments.
 - Given other arguments, returns `p` called with those arguments.
- Note: to create a procedure that takes a variable number of arguments use the following form:
 `(lambda args ; args gets the arguments as a list
 ...)`
- (d) Use `auto-apply` to overload `+` so that it can be used as shown above.
- (2) (a) Define `fix` that:
- Takes three arguments: a procedure `p`, natural number `n` and arbitrary value `a`.
 - Returns a procedure that given `m` arguments, where $n \leq m+1$, calls `p` with `m+1` arguments, where `a` is used as an extra argument inserted as the `n`th argument.
- (b) Use `fix` to define `square` using Scheme's `expt` procedure.
- (3) (a) Define `tester` that:
- Takes a procedure of one argument and a list of pairs of test input and expected output.
 - Returns the results of calling the procedure on the test inputs, comparing the results with the expected outputs, as a list of the following form:
 `(passed <test-input>)`
 `(failed <test-input> expected <expected-output> got <actual-output>)`
- Use `equal?` to compare the actual and expected outputs.
For example:
 `(tester - '((1 2) (3 -3)))`
 `=> ((failed 1 expected 2 got -1) (passed 3))`
- (b) Define `regression-tester`, useful for comparing whether a new procedure behaves the same as an existing one. It:
- Takes a procedure `p` of one argument and a list of test inputs.
 - Returns a procedure that: given a procedure `q` tests it using `tester`, comparing `q` and `p` on the list of test inputs.
- For example:
 `(define sqrt-tester (regression-tester sqrt '(0 1 2.25)))`
 ; + works surprisingly well if we want the square root of 0 or 1, but not so well on 2.25
 `(sqrt-tester +)`
 `=> ((passed 0) (passed 1) (failed 2.25 expected 1.5 got 2.25))`
- The expected output (of `sqrt` in our example) should be calculated when `regression-tester` is called, not every time its result (`sqrt-tester` in our example) is called.
- (c) Use `tester` to test `sqrt-tester` (!) on two well-chosen test cases (`sqrt-tester` should of course pass your test).
- (4) We can create things in Scheme that behave like objects.
- (a) Define `make-2d-vector` to create 2-dimensional vector 'objects' that behave as follows:
- ```
(define v (make-2d-vector 3 4)) ; v represents vector (3,4)
(v 'x) => 3
(v 'y) => 4
(v 'display) => display (3,4)
(v 'length) => 5
(define w (v 'scale 2)) ; w represents vector (6,8)
(v 'add (make-2d-vector -5 7)) => vector representing (-2,11)
```
- (b) Extend the `add` method to accept any number of vectors. For example:
- ```
((v 'add v w) 'display) ; display (12,16)
((v 'add) 'display) ; display (3,4)
```