

Lab 2

Recursion & Iteration in LISP

Much processing in LISP utilizes recursion; the body of a procedure includes one or more calls to the very same procedure -- calls that deal with simpler and more readily computable values of the parameters.

1. Define a recursive procedure *countdown* that takes any non-negative integer as its argument and returns a list of all the positive integers less than or equal to the initial argument, in descending order:

```
(countdown 5) ==> (5 4 3 2 1)
(countdown 1) ==> (1)
(countdown 0) ==> ()
```

2. Write a recursive procedure *replicate* that takes two arguments, *size* and *item*, and returns a list of *size* elements, each of which is the name of the *item*:

```
(replicate 6 'foo) ==> (foo foo foo foo foo foo)
(replicate 2 NIL) ==> (NIL NIL)
(replicate 1 15) ==> (15)
(replicate 3 '(alpha beta)) ==> ((alpha beta) (alpha beta) (alpha beta))
(replicate 0 'help) ==> ()
```

3. Write a recursive procedure *double-each-element* that takes a list of numbers and returns a list of their doubles:

```
(double-each-element '(3 -62 41.4 17/4)) ==> (6 -124 82.8 17/2)
(double-each-element '(0)) ==> (0)
(double-each-element '()) ==> ()
```

4. Define a function *add-only-numbers* that adds the numbers on the top level in a list.

```
(add-only-numbers '(a 1 b (b 4) 2 3))
=> 6
```

5. Define a function *remove-vowels* that takes a list of letters (symbols) and returns a new

list where the vowels are removed.

```
(remove-vocals '(b i r g i t t a))
=> (b r g t)
```

In some cases, it may be helpful to define two procedures to solve a problem -- one to set up the initial call to the other, supplying additional arguments to assist the recursion. This is illustrated in the following problem:

On Thursday we will do these as Iterative functions ..

Finish up your assignment and submit to *lab2*