
1\. 10%

Write an equivalent Lisp expression for

```
   exp(x)  -  cos(y)
---------------------   where x, y, sin, cos, log and exp have been
 sin(x - y * log(y))       defined somewhere.
```

2\. 10% + 5%

```
(let ((x (+ z 5))
      (y (+ x 2))
      (z (+ x y)))
  (* (+ x y) z))
```

a) For the above 'let expression', write its equivalent 'lambda form'.
b) With x=1 , y=2, and z=4, evaluate the above 'let expression'.

3\. 15%

Write a procedure, (make-mod-m-k m k), which accepts two parameters, m, k and will return a one-argument function. The returned function is a predicate which will determine if remainder of the argument by m equals to k. That is, if the parameter is named by n, it will return true if n % m == k, or return false.

```
> (define mod-3-1? (make-mod-m-k 3 1))
; no values returned
> (mod-3-1? 10)
#t
> (mod-3-1? 12)
#f
>
```

4\. 15% + 15%

A function f defined for non-negative integers is defined as:

```
f(n) = 3,                    if n=0
       1,                    if n=1
       f(n-1) + f(n-2),      if n>1 and n mod 3 == 1
       f(n-1) - f(n-2),      if n>1 and n mod 3 != 1
```

a) Write a procedure, (rec n), that computes f in linear recursive way.
b) Write a procedure, (iter n), that computes f in linear iterative way.

Note: You may assume (mod-3-1? n) defined in previous problem
 is available.

5\. 15%

The previous process can be generalized as follows:

```
f(n) = f0,                          if n=0
       f1,                          if n=1
       alpha(n, f(n-1), f(n-2))     if n>1
```

where alpha is a function.

Write a function, (make-fun f0 f1 alpha), which will return a designed function as specified and the returned function is run iteratively.

6\. 10% + 5%

a) Use previous (make-fun f0 f1 alpha) to redefine (iter n)
 (the function defined in problem 4).
b) Use previous (make-fun f0 f1 alpha) to redefine (fib n).

```
> (iter 1035)
-3
> (redefine-iter 1035)
-3
> (fib 102)
927372692193078999176
> (redefine-fib 102)
927372692193078999176
> ,exit
```

0! =1
1! = 1
-a+5b-