Examen de Diseño de Algoritmos

Se valorará tanto la organización del código, como la claridad de exposición.

Como se vio en el SICP, existe el método de Newton para hallar raíces de una función $f(x)$ que lo que hace es buscar un punto fijo de la siguiente función:

$$g(x) = x - \frac{f(x)}{f'(x)}.$$

La función *inversa* f^{-1} de una función f es aquella función que cumple que para cada x se tiene que $f^{-1}(f(x)) = x$. Una forma para calcular el valor de la función inversa en y, $f^{-1}(y)$, es calcular una raiz de $f(x) - y$.

En el intérprete de Scheme están definidas las funciones *cos* para el coseno y *sin* para el seno.

1) *(1.5 puntos) Programe la función arco-coseno y arco-seno, como las inversas de la funciones seno y coseno utilizando el método de Newton.*

La forma más tradicional de definir funciones recursivas es utilizando el procedimiento define:

```
(define (fib n) (if (< n 2) 1 (+ (fib (- n 1)) (fib (- n 2)))))
```

2) *(2.5 puntos) Programe el procedimiento fib utilizando solo funciones sin nombre, es decir, tiene que se una única expresión sin utilizar el método define. Además está expresión, al evaluarla, debe generar un procedimiento iterativo. Se permite que el método pueda tener argumentos adicionales.*

El llamado método *currying* es un procedimiento para transformar funciones con varios argumentos en funciones de un solo argumento. Ilustremos esta idea con un ejemplo, supongamos que tenemos la función $h(x, y) = x + y^2$. Para verla como la evaluación de funciones de un solo argumento podemos pensar que h_x es una función que, a un valor x, devuelve una función que toma un solo argumento y que al evaluarla devuelve $x + y^2$.

En código Python, tendríamos algo así:

```
def h(x):
    def temp(y):
        print(x)
        return x + y**2
    return temp
print(h(1)(2))
```

3) *(3 puntos) Haga un método llamado currying que aplique* currying *utilizando la «dotted-tail notation». El método tiene que devolver un método. ¿Qué ocurre si se aplica currying a si mismo? Razónelo.*

En la sección del libro SICP 3.2.3 se ve como se describe el comportamiento del modelo con las variables modificadas funciona. Cogemos el ejemplo descrito en la sección 3.1.1 con el siguiente código:

```
(define (make-account balance)
  (define (withdraw amount)
    (if (>= balance amount)
        (begin (set! balance
                     (- balance
                        amount))
               balance)
        "Insufficient funds"))
  (define (deposit amount)
    (set! balance (+ balance amount))
    balance)
  (define (dispatch m)
    (cond ((eq? m 'withdraw) withdraw)
          ((eq? m 'deposit) deposit)
          (else (error "Unknown request:
                       MAKE-ACCOUNT"
                      m))))
  dispatch)
```

y aquí tenemos un código de prueba:

```
(define acc (make-account 50))

((acc 'deposit) 40)
; -> 90

((acc 'withdraw) 60)
; -> 30

(define acc2 (make-account 100))
```

4) *(3 puntos) Explique como se comporta el ámbito y donde se guardan las variables locales de acc. Explique también como se mantiene distintas acc y acc2 y que partes son comunes.*