Zadanie 1

;; Funkcja, która oblicza sumę kwadratów pierwszych n liczb naturalnych

(defn sum-of-squares [n]

  ;; Redukcja sumuje kwadraty liczb z zakresu od 1 do n włącznie

  (reduce + (map #(\* % %) (range 1 (inc n)))))

;; Funkcja, która oblicza kwadrat sumy pierwszych n liczb naturalnych

(defn square-of-sum [n]

  ;; Oblicza sumę liczb z zakresu od 1 do n włącznie

  (let [sum (reduce + (range 1 (inc n)))]

    ;; Zwraca kwadrat tej sumy

    (\* sum sum)))

;; Funkcja, która oblicza różnicę między kwadratem sumy a sumą kwadratów

(defn difference [n]

  (- (square-of-sum n) (sum-of-squares n)))

;; Funkcja główna programu

(defn -main []

  ;; Wynik różnicy dla pierwszych 100 liczb naturalnych

  (println "The difference between the sum of the squares and the square of the sum for the first 100 natural numbers is:"

           (difference 100)))

;; Funkcja główna

(-main)

Zadanie 2

;; a. Wzór na pierwiastek sześcienny

(defn cube-root [x]

  (Math/pow x (/ 1.0 3.0)))

;; b. Procedura Herona zależna od epsilon

(defn heron-method-epsilon [x epsilon]

  (let [next (fn [guess] (/ (+ (/ x (\* guess guess)) (\* 2 guess)) 3.0))]

    (loop [guess x]

      (let [new-guess (next guess)]

        (if (< (Math/abs (- new-guess guess)) epsilon)

          new-guess

          (recur new-guess))))))

;; c. Procedura Herona zależna od liczby kroków

(defn heron-method-steps [x steps]

  (let [next (fn [guess] (/ (+ (/ x (\* guess guess)) (\* 2 guess)) 3.0))]

    (loop [guess x step 0]

      (if (>= step steps)

        guess

        (recur (next guess) (inc step))))))

;; Funkcja główna

(defn -main []

  (println "Pierwiastek sześcienny z 27 to:" (cube-root 27))

  (println "Pierwiastek sześcienny z 27 z epsilon 0.0001 to:" (heron-method-epsilon 27 0.0001))

  (println "Pierwiastek sześcienny z 27 po 10 krokach to:" (heron-method-steps 27 10)))

(-main)

Zadanie 3

;; a. Wzór na pierwiastek sześcienny

(defn cube-root [x]

  (Math/pow x (/ 1.0 3.0)))

;; b. Procedura Herona zależna od epsilon

(defn heron-method-epsilon [x epsilon]

  (let [next (fn [guess] (/ (+ (/ x (\* guess guess)) (\* 2 guess)) 3.0))]

    (loop [guess x]

      (let [new-guess (next guess)]

        (if (< (Math/abs (- new-guess guess)) epsilon)

          new-guess

          (recur new-guess))))))

;; c. Procedura Herona zależna od liczby kroków

(defn heron-method-steps [x steps]

  (let [next (fn [guess] (/ (+ (/ x (\* guess guess)) (\* 2 guess)) 3.0))]

    (loop [guess x step 0]

      (if (>= step steps)

        guess

        (recur (next guess) (inc step))))))

;; Funkcja główna

(defn -main []

  (println "Pierwiastek sześcienny z 27 to:" (cube-root 27))

  (println "Pierwiastek sześcienny z 27 z epsilon 0.0001 to:" (heron-method-epsilon 27 0.0001))

  (println "Pierwiastek sześcienny z 27 po 10 krokach to:" (heron-method-steps 27 10)))

(-main)

Zadanie 4

(defn powerset [s]

  (if (empty? s)

    (list '())

    (let [rest (powerset (rest s))

          first-element (first s)]

      (concat rest (map #(cons first-element %) rest)))))

;; Funkcja główna

(defn -main []

  ;; Test funkcji

  (println "Powerset [1 2 3]:")

  (println (powerset [1 2 3])))

(-main)

Zadanie 5

;; a. Pierwiastek sześcienny z wykorzystaniem average-damp oraz FIXED-POINT

(defn average-damp [f]

  (fn [x] (/ (+ (f x) x) 2)))

(defn fixed-point [f guess tolerance]

  (let [close-enough? (fn [v1 v2] (< (Math/abs (- v1 v2)) tolerance))]

    (loop [guess guess]

      (let [next (f guess)]

        (if (close-enough? guess next)

          next

          (recur next))))))

(defn cube-root [x]

  (fixed-point (average-damp (fn [y] (/ x (\* y y)))) 1.0 0.0001))

;; b. Pierwiastek sześcienny z wykorzystaniem Newton's method

(defn newton-method [f df guess tolerance]

  (let [close-enough? (fn [v1 v2] (< (Math/abs (- v1 v2)) tolerance))]

    (loop [guess guess]

      (let [next (- guess (/ (f guess) (df guess)))]

        (if (close-enough? guess next)

          next

          (recur next))))))

(defn cube-root-newton [x]

  (newton-method (fn [y] (- (\* y y y) x))

                 (fn [y] (\* 3 y y))

                 1.0

                 0.0001))

;; c. Procedura implementująca złożenie funkcji

(defn zloz [f g]

  (fn [x] (f (g x))))

;; d. Procedura implementująca n-krotne złożenie funkcji

(defn n-krotne-zlozenie [f n]

  (if (= n 1)

    f

    (zloz f (n-krotne-zlozenie f (dec n)))))

;; Funkcja główna

(defn -main []

  ;; Testy funkcji

  (println "Pierwiastek sześcienny z 27 to:" (cube-root 27))

  (println "Pierwiastek sześcienny z 27 (Newton's method) to:" (cube-root-newton 27))

  (def kwadrat (fn [x] (\* x x)))

  (def f (zloz kwadrat inc))

  (println "Złożenie funkcji kwadrat i inc dla 6 to:" (f 6)) ;; powinno zwrócić 49

  (defn inc-f [x] (inc x))

  (def f2 (n-krotne-zlozenie inc-f 5))

  (println "5-krotne zastosowanie funkcji inc do 10 to:" (f2 10))) ;; powinno zwrócić 15

(-main)