```
(ns n-gram.misc.misc-functions)

(defn cumsum

   "With one arg x returns lazy cumulative sum sequence y with (= (nth y) (cumsum x))

    With two args t, x returns lazy cumulative sum sequence y with (= (nth y) (+ t
(cumsum x)))"

  ([x] (cumsum 0 x))

  ([t x] (if (empty? x) ()

           (let [y (+ t (first x))]

             (cons y ( lazy-seq (cumsum y (rest x))))))))

(defn get-count "Returns 0 if word not found, otherwise returns count"[func args]
  (if (nil? (func args)) 0 (func args)))

(def get-count-memo (memoize get-count))

(defn sum [x] (reduce + x))

(defn equal [num1 num2 accuracy] (letfn [(bignum [num] (.setScale (BigDecimal. num)
accuracy BigDecimal/ROUND_DOWN))]
                                   (= 0 (.compareTo (bignum num1) (bignum num2)))))

(defn cartesian-product
  "All the ways to take one item from each sequence"
  [& seqs]
  (let [v-original-seqs (vec seqs)
  step
  (fn step [v-seqs]
    (let [increment
        (fn [v-seqs]
          (loop [i (dec (count v-seqs)), v-seqs v-seqs]
            (if (= i -1) nil
              (if-let [rst (next (v-seqs i))]
                (assoc v-seqs i rst)
                (recur (dec i) (assoc v-seqs i (v-original-seqs i))))))))]
      (when v-seqs
         (cons (map first v-seqs)
             (lazy-seq (step (increment v-seqs)))))))]
    (when (every? first seqs)
      (lazy-seq (step v-original-seqs)))))
```

```clojure
(defn selections
  "All the ways of taking n (possibly the same) elements from the sequence of items"
  [items n]
  (apply cartesian-product (take n (repeat items))))

(defn vector_to_string [vector] (str (reduce str vector)))
```