

Permutations

- Given a *set* of numbers, generate all possible (unique) permutations.
 - each permutation is an ordering of the elements.
- Example: $\{1\ 2\ 3\}$:

(1 2 3)	(1 3 2)
(2 1 3)	(2 3 1)
(3 1 2)	(3 2 1)

Scheme Permutations

- Write a function that generates all permutations given a list that represents the set.
 - each permutation will also be a list
 - produces a list of lists.

```
(generate-permutations '(1 2 3)) =>
```

```
'( (1 2 3) (1 3 2) (2 1 3)  
   (2 3 1) (3 1 2) (3 2 1) )
```

Some Assumptions

- The elements of the list passed to generate-permutations are unique numbers (no duplicates).
- We can use append:

```
(append '(1 2 3) '(4 5)) => '(1 2 3 4 5)
```

```
(append '((1 2 3) (1 3 2)) '((2 1 3))) =>  
'( (1 2 3) (1 3 2) (2 1 3) )
```

Design Ideas/Issues

- Try to divide the problem in to some separate, independent problems (each of which is easier).
 - For each sub-problem, determine a strategy for solving the sub-problem.
 - list what you need to implement for each sub-problem.
 - implement each function identified.
- Glue everything together.

Permutation Generation Design

- One idea:
 - 1) generate all permutations that begin with the first element in the set.
 - 2) generate all permutations that begin with elements that are not the first element in the set.

– combine the results of 1 and 2.

Generate all permutations that start with the first element of `alist`.

- Idea:
 - 1) generate all permutations of the subset defined by `(rest alist)`.
 - 2) add `(first alist)` to the beginning of each of these.
- We need:
 - 1) the function `generate-permutations`.
 - 2) a function that will add an element to the beginning of a bunch of lists (a list of lists)

Add an element `x` to the beginning of a list `l`

- For one list this is easy:

```
(cons x l)
```

- We need to do this for a bunch of lists, for example:

```
(add-element-to-lists 1 '( (2 3) (3 2) ))  
=> '( (1 2 3) (1 3 2) )
```

add-element-to-lists

```
(define (add-element-to-lists elem lists)
  (cond
    [(empty? lists) empty]
    [else
     (cons
      (cons elem (first lists))
      (add-element-to-lists elem
                            (rest lists)))]))
```

Generate all permutations that start with the first element of `alist`.

```
(add-element-to-lists (first alist)
  (generate-permutations (rest alist)))
```

1) generate all permutations that begin with the first element in the set.

**This part is now done**

2) G

enerate all permutations that begin with elements that are not the first element in the set.

Exercise

Finish the *design* of generate-permutations:

- "Generate all permutations that begin with elements that are not the first element in the set."
- Treat this as "the problem".
 - separate into sub-problems.
 - work on each subproblem.
- end result is a list of functions you need to write (that you know how to write).

Exercise

- *Design* a program (strategy, algorithm) to determine whether or not an integer is prime.
- Document your design (write it down).
- Implement and test your design.

Work with someone on this!