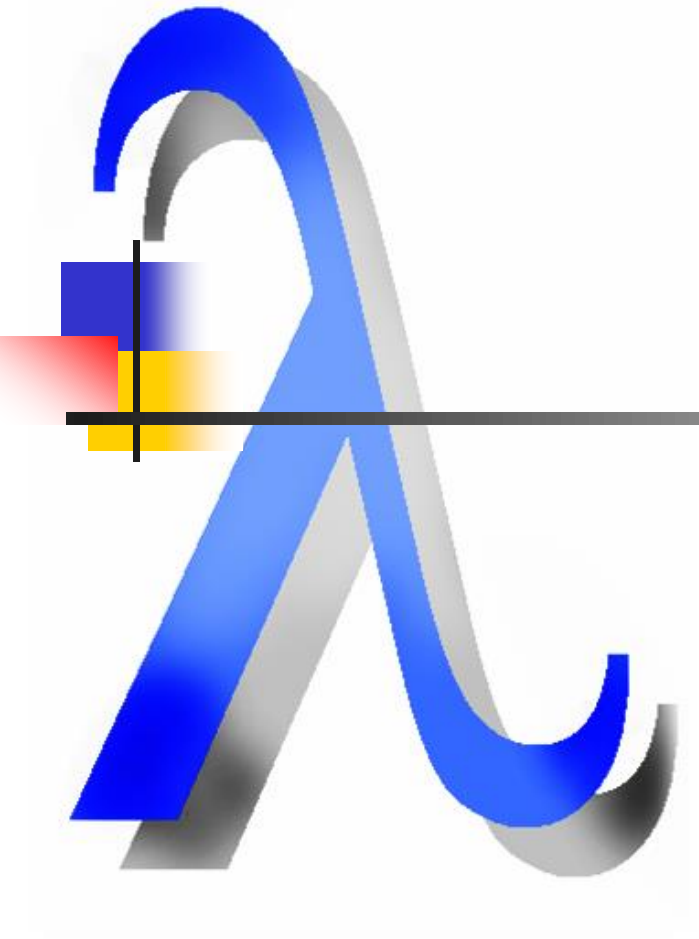
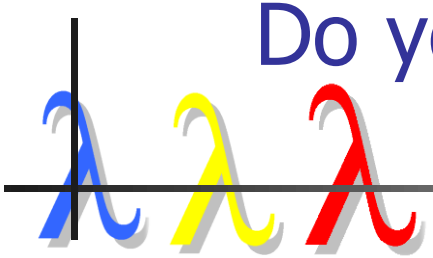
Vector Programming Using Structural Recursion

An Introduction to Vectors for Beginners



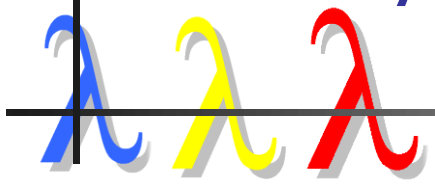
Marco T. Morazán
Seton Hall University

Do you remember summing a vector?



```
; sum-vector: (vectorof number) → number
; Purpose: Add  all vector numbers
(define (sum-vector V)
  ; natnum  Purpose: (add1 k) is the index of the next element to add
  (define k -1)
  ; number
  ; Purpose: The sum V[0] to V[k]
  (define sum 0)
  ; loop: → number
  ; Purpose: To add the numbers in V
  (define (loop)
    (cond [(>= k (vector-length V)) sum]
          [else
            (begin (set! k (add1 k))  (set! sum (+ sum (vector-ref V k)))  (loop))]))
  (begin  (loop)   sum))
```

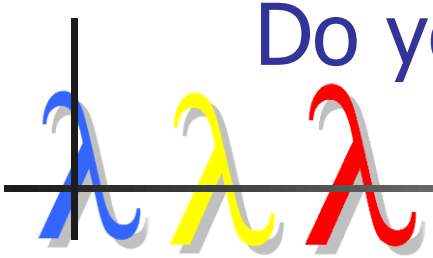
Do you remember summing a vector?



```
; sum-vector: (vectorof number) → number  Purpose: Add  all vector numbers
(define (sum-vector V)
  ; natnum  Purpose: (add1 k) is the index of the next element to add
  (define k -1)
  ; number
  ; Purpose: The sum V[0] to V[k]
  (define sum 0)
  ; loop: natnum => number
  ; Purpose: To add the numbers in V
  (define (loop)
    (cond [(>= k (vector-length V)) sum]
          [else
           (begin (set! k (add1 k))  (set! sum (+ sum (vector-ref V k)))  (loop))]))
  (begin (loop) sum))
```

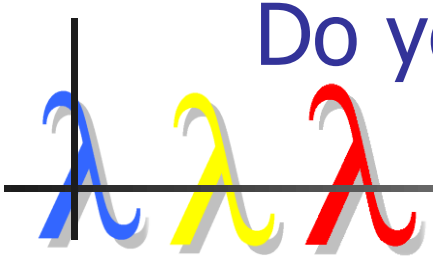
```
> (sum-vector (vector 1 2 3))
vector-ref: index is out of range
index: 3
valid range: [0, 2]
vector: '#(1 2 3)
```

Do you remember summing a vector?



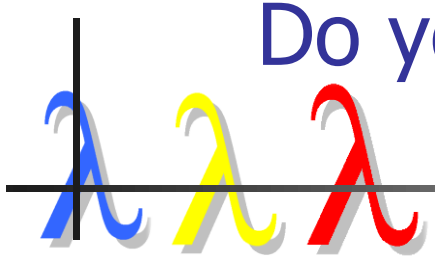
```
; sum-vector: (vectorof number) → number Purpose: Add  all vector numbers
(define (sum-vector V)
  ; natnum  Purpose: (add1 k) is the index of the next element to add
  (define k -1)
  ; number
  ; Purpose: The sum V[0] to V[k]
  (define sum 0)
  ; loop: natnum --> number
  ; Purpose: To add the numbers in V
  (define (loop)
    (cond [(>= k (vector-length V)) sum]
          [else          The bug is manifested here and here is where we must fix it!
           (begin (set! k (add1 k))  (set! sum (+ sum (vector-ref V k)))  (loop))]))
  (begin  (loop)   sum))
```

Do you remember summing a vector?



```
; sum-vector: (vectorof number) → number Purpose: Add  all vector numbers
(define (sum-vector V)
  ; natnum  Purpose: (add1 k) is the index of the next element to add
  (define k -1)
  ; number
  ; Purpose: The sum V[0] to V[k]
  (define sum 0)
  ; loop: natnum --> number
  ; Purpose: To add the numbers in V
  (define (loop)
    (cond [(>= k (vector-length V)) sum]
          [else          The bug is manifested here and here is where we must fix it!
            (begin (set! k (add1 k)) (set! sum (+ sum (vector-ref V (sub1 k)))) (loop))]))
  (begin  (loop)   sum))
```

Do you remember summing a vector?

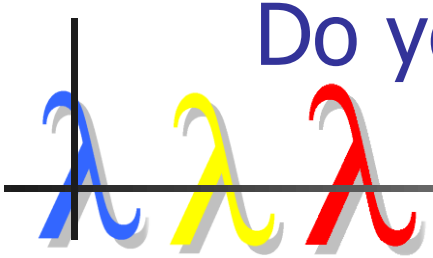


```
; sum-vector: (vectorof number) → number Purpose: Add all vector numbers
(define (sum-vector V)
  ; natnum  Purpose: (add1 k) is the index of the next element to add
  (define k -1)
  ; number
  ; Purpose: The sum V[0] to V[k]
  (define sum 0)
  ; loop: natnum --> number
  ; Purpose: To add the numbers in V
  (define (loop)
    (cond [(>= k (vector-length V)) sum]
          [else
           (begin (set! k (add1 k)) (set! sum (+ sum (vector-ref V (sub1 k)))) (loop))]))
  (begin (loop) sum))
```

> (sum-vector (vector 1 2 3))
> vector-ref: contract violation
> expected: exact-nonnegative-integer?
> given: -1
> argument position: 2nd
> other arguments...:

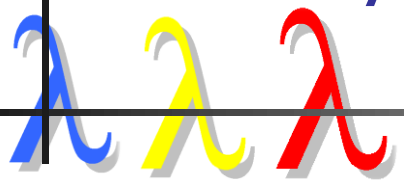
The bug is manifested here

Do you remember summing a vector?



```
; sum-vector: (vectorof number) → number Purpose: Add  all vector numbers
(define (sum-vector V)
  ; natnum  Purpose: (add1 k) is the index of the next element to add
  (define k -1)
  ; number
  ; Purpose: The sum V[0] to V[k]
  (define sum 0)
  ; loop: natnum --> number
  ; Purpose: To add the numbers in V
  (define (loop)
    (cond [(>= k (vector-length V)) sum]
          [else                 The bug is manifested here and here is where we must fix it!
           (begin (set! k (add1 k)) (set! sum (+ sum (vector-ref V (add1 k)))) (loop))]))
  (begin  (loop)   sum))
```

Do you remember summing a vector?

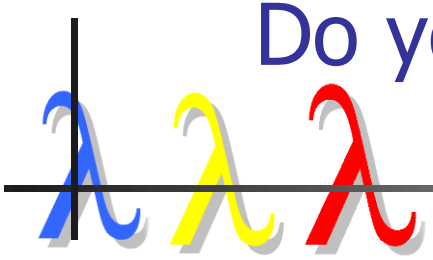


```
; sum-vector: (vectorof number) → number Purpose: Add  all vector numbers
(define (sum-vector V)
  ; natnum  Purpose: (add1 k) is the index of the next element to add
  (define k -1)
  ; number
  ; Purpose: The sum V[0] to V[k]
  (define sum 0)
  ; loop: natnum --> number
  ; Purpose: To add the numbers in V
  (define (loop)
    (cond [(>= k (vector-length V)) sum]
          [else
           (begin (set! k (add1 k)) (set! sum (+ sum (vector-ref V (add1 k)))) (loop))]))
  (begin  (loop)  sum))
```

> (sum-vector (vector 1 2 3))
> vector-ref: index is out of range
> index: 3
> valid range: [0, 2]
> vector: '#(1 2 3)

The bug is manifested here

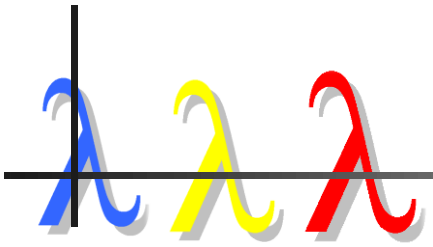
Do you remember summing a vector?



The problem is not knowing how to reason and process an

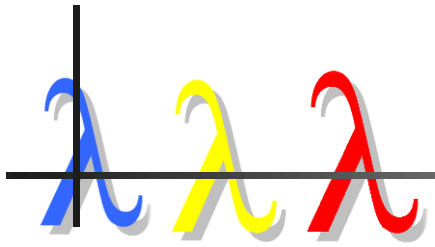
interval of indices

Vectors give me nothing, but sphilkes!



Still the same

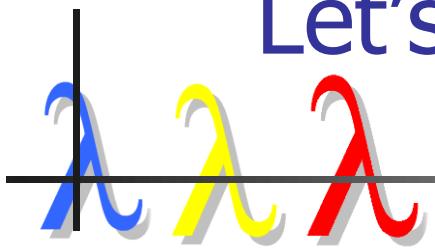
- Students today still find vector programming hard
  - index out of bounds errors
- Introduction to Vectors
  - Syntax
  - Examples with no design principles
  - Left to their devices to figure out indexing



Still the same

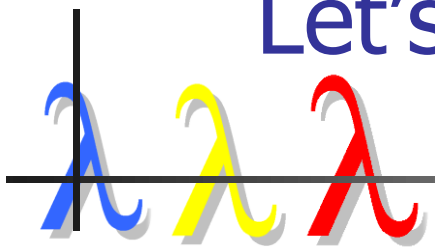
- a collection of variables of the same type with each element having an index
- a finite sequential list of elements of the same datatype identifying the first element, the second element, the third element, and so forth

Let's Build on what students learn!



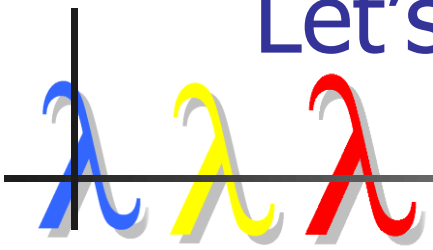
- At SHU
  - structural, generative, and accumulative recursion
  - recursive data definitions
    - lists, natural numbers, trees
  - function templates
  - The Design Recipe

Let's Build on what students learn!



- An interval is…I know what it is. I just can't explain it.
- An interval is [i..j], where i < j
- Inadequate
  - does not expose the structure of an interval
  - interval can be empty is well-hidden

Let's Build on what students learn!

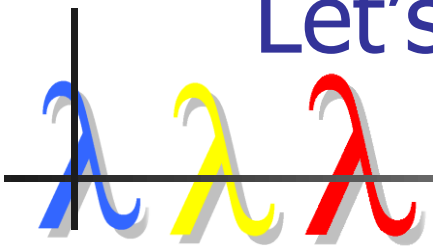


- An INTV is two integers, low & high, such that it is either:
  1. empty (low > high)
  2. [low..high], where n is an integer, high = n+1 & low ≤ high

[-1..1] = [[-1..0]..1]
= [[-1..-1]..0..1]
= [[-1..-2]..-1..0..1]
= [empty..0..-1..0..1]
= [0..-1..0..1]

- An INTV is built from a sub-INTV is clear!

Let's Build on what students learn!

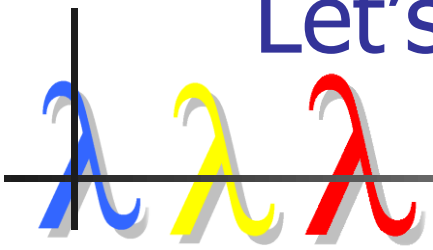


- An INTV is two integers, low & high, such that it is either:
  1. empty (low > high)
  2. [low..high], where n is an integer, high = n+1 & low ≤ high

- Template

```
; f-on-INTV: int int → …
; Purpose: For the given INTV, …
(define (f-on-interval low high)
  (cond [(empty-INTV? low high) …]
        [else high…(f-on-INTV low (sub1 high))]))
```

Let's Build on what students learn!

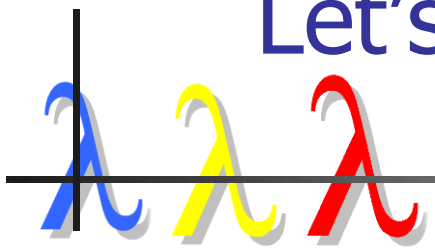


- An INTV is two integers, low & high, such that it is either:
  1. empty (low > high)
  2. [low..high], where n is an integer, high = n+1 & low ≤ high

- Template

```
; empty-INTV?: int int → Boolean
; Purpose: For the given INTV, determine if it is empty
(define (empty-INTV? low high) (< high low))
```

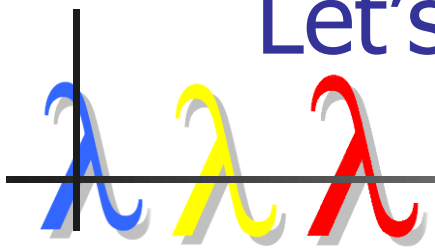
Let's Build on what students learn!



- Sum the elements of an INTV

```
; f-on-INTV: int int → int
; Purpose: For the given INTV, …
(define (f-on-INTV low high)
  (cond [(empty-INTV? low high) …]
        [else high…(f-on-INTV low (sub1 high))]))
```

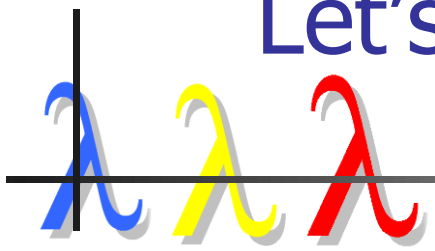
Let's Build on what students learn!



- Sum the elements of an INTV

```
; sum-INTV: int int → int
; Purpose: For the given INTV, sum its elements
(define (sum-INTV low high)
  (cond [(empty-INTV? low high) ...]
        [else high...(sum-INTV low (sub1 high))]))
```

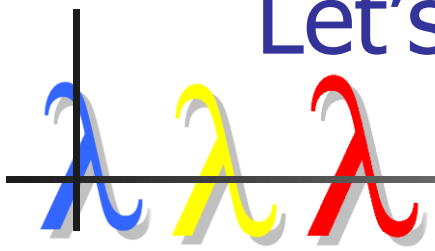
Let's Build on what students learn!



- Sum the elements of an INTV

```
; sum-INTV: int int → int
; Purpose: For the given INTV, sum its elements
(define (sum-INTV low high)
  (cond [(empty-INTV? low high) 0]
        [else high…(sum-INTV low (sub1 high))]))
```

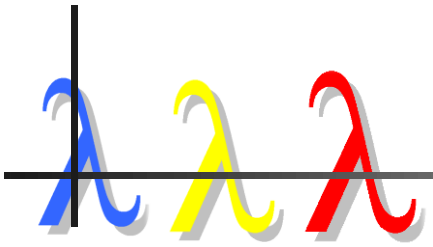
Let's Build on what students learn!



- Sum the elements of an INTV

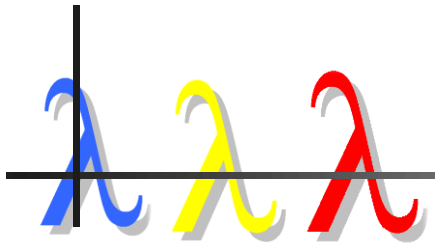
```
; sum-INTV: int int → int
; Purpose: For the given INTV, sum its elements
(define (sum-INTV low high)
  (cond [(empty-INTV? low high) 0]
        [else (+ high (sum-INTV low (sub1 high)))]))
```

But, Marco!



- This suggests always processing the INTV from right to left

```
; f-on-INTV: int int → int
; Purpose: For the given INTV, …
(define (f-on-INTV low high)
  (cond [(empty-INTV? low high) …]
        [else high…(f-on-INTV low (sub1 high))]))
```



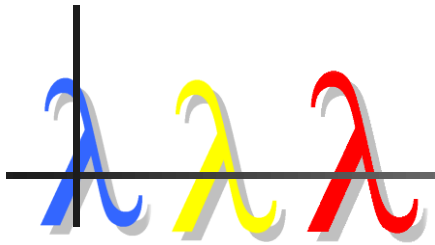
But, Marco!

- An INTV can be built from low to high

An INTV is two integers, low and high, such that either it is:

1. empty (i.e., low > high)
2. [low..high], where n is an integer, low = n-1 and low ≤ high

But, Marco!



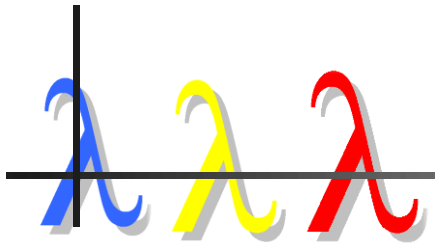
An INTV is two integers, low and high, such that it is either:

1. empty (i.e., low > high)
2. [low..high], where n is an integer, low = n-1 and low ≤ high

```
; f-on-interval2: natnum natnum → …
; Purpose: …
(define (f-on-interval2 low high)
      (cond [(empty-INTV? …]
            [else low…(f-on-interval2 (add1 low) high)]))
```

Process from left to right

But, Marco!

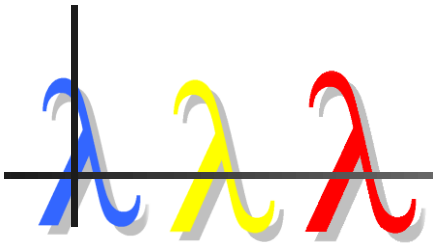


```
; sum-INTV: int int → int
; Purpose: For the given INTV, sum its elements
(define (sum-INTV low high)
  (cond [(empty-interval? low high) 0]
        [else (+ high (sum-INTV low (sub1 high)))]))
```

Process from right to left

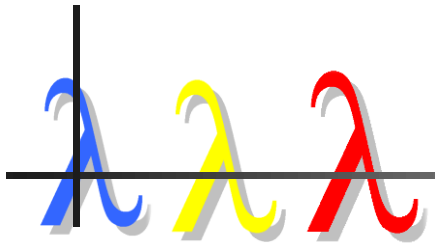
```
; sum-INTV2: natnum natnum --> natnum
; Purpose: Sum all the integers in the given interval
(define (sum-INTV2 low high)
  (cond [(empty-interval? low high) 0]
        [else (+ low (sum-INTV2 (add1 low) high))]))
```

Process from left to right



Tackling vectors

- Processing the whole vector: [0..(sub1 (vector-length V))]
- Processing part of a contiguous subset of a vector: [low..high]
- Clearly, an interval needs to be processed
  - index must be a natnum
  - out of bound errors



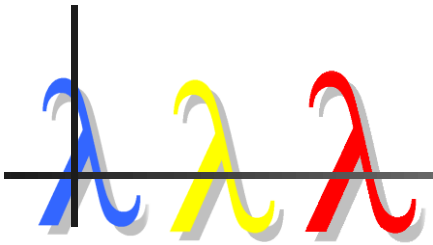
Tackling vectors

Given a vector of length N and a natural number n, a vector interval, VINTV, is two integers, low >= 0 and -1 <= high <= N-1, such that it is either:

1. empty (i.e., low > high)
2. [low..high], where high=n+1 and low ≤ high

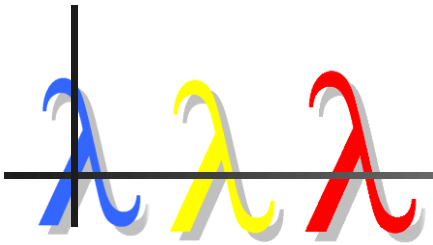
- When the VINTV is not empty, it is an INTV of natnums
- Similar definition to process a VINTV left to right

Tackling vectors



```
; f-on-vector: (vector X) → …
; Purpose: …
(define (f-on-vector V)
  (local [; f-on-VINTV: int int → …
          ; Purpose: For the given VINTV, …
          (define (f-on-VINTV low high)
            (cond [(empty-VINTV? low high) …]
                  [else (vector-ref V high)…(f-on-VINTV low (sub1 high))]))

          ; f-on-VINTV2: int int → …
          ; Purpose: For the given VINTV, …
          (define (f-on-VINTV2 low high)
            (cond [(empty-VINTV2? low high) …]
                  [else (vector-ref V low)…(f-on-VINTV2 (add1 low) high)]))]
    …))
```

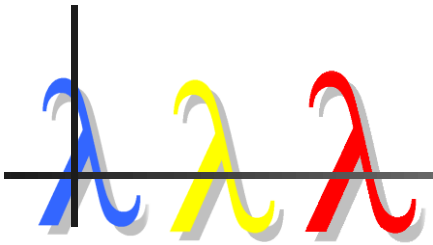
Tackling vectors



Consider computing the average of a vector of numbers

```
; f-on-vector: (vector X) → …
; Purpose: …
(define (f-on-vector V)
  (local [; f-on-VINTV: int int → …
          ; Purpose: For the given VINTV, …
          (define (f-on-VINTV low high)
            (cond [(empty-VINTV? low high) …]
                  [else (vector-ref V high)…(f-on-VINTV low (sub1 high))]))

          ; f-on-VINTV2: int int → …
          ; Purpose: For the given VINTV, …
          (define (f-on-VINTV2 low high)
            (cond [(empty-VINTV2? low high) …]
                  [else (vector-ref V low)…(f-on-VINTV2 (add1 low) hig)]))]
    …))
```

Tackling vectors

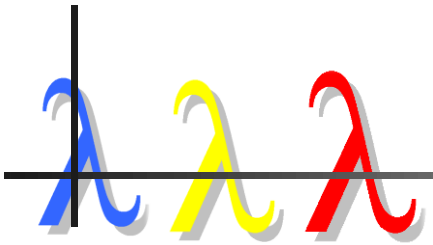


Consider computing the average of a vector of numbers

```
; avg-vector: (vector number) → number
; Purpose: To compute the average of the given vector
(define (avg-vector V)
  (local [; f-on-VINTV: int int → …
          ; Purpose: For the given VINTV, …
          (define (f-on-VINTV low high)
            (cond [(empty-VINTV? low high) …]
                  [else (vector-ref V high)…(f-on-VINTV low (sub1 high))]))

          ; f-on-VINTV2: int int → …
          ; Purpose: For the given VINTV, …
          (define (f-on-VINTV2 low high)
            (cond [(empty-VINTV2? low high) …]
                  [else (vector-ref V low)…(f-on-VINTV2 (add1 low) hig)]))]
    …))
```

What kind of expression do we need in the body of the local?

Tackling vectors

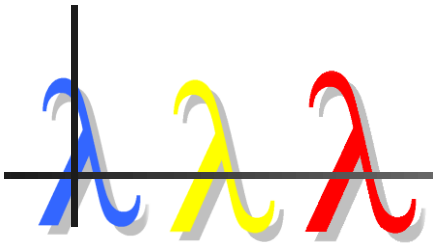


Consider computing the average of a vector of numbers

```
; avg-vector: (vector number) → number
; Purpose: To compute the average of the given vector
(define (avg-vector V)
  (local [; f-on-VINTV: int int → …
          ; Purpose: For the given VINTV, …
          (define (f-on-VINTV low high)
            (cond [(empty-VINTV? low high) …]
                  [else (vector-ref V high)…(f-on-VINTV low (sub1 high))]))
          ; f-on-VINTV2: int int → …
          ; Purpose: For the given VINTV, …
          (define (f-on-VINTV2 low high)
            (cond [(empty-VINTV2? low high) …]
                  [else (vector-ref V low)…(f-on-VINTV2 (add1 low) hig)]))]
    (/ (sum-elems ???  ???))
       (vector-length V))))
```

What VINTV do we want to process?

Tackling vectors

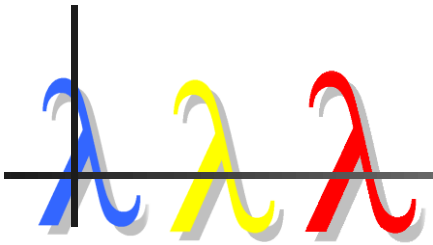


Consider computing the average of a vector of numbers

```
; avg-vector: (vector number) → number
; Purpose: To compute the average of the given vector
(define (avg-vector V)
  (local [; f-on-VINTV: int int → …
          ; Purpose: For the given VINTV, …
          (define (f-on-VINTV low high)
            (cond [(empty-VINTV? low high) …]
                  [else (vector-ref V high)…(f-on-VINTV low (sub1 high))]))
          ; f-on-VINTV2: int int → …
          ; Purpose: For the given VINTV, …
          (define (f-on-VINTV2 low high)
            (cond [(empty-VINTV2? low high) …]
                  [else (vector-ref V low)…(f-on-VINTV2 (add1 low) hig)]))]
    (/ (sum-elems 0 (sub1 (vector-length V)))
       (vector-length V))))
```

How should we process the VINTV?

Tackling vectors

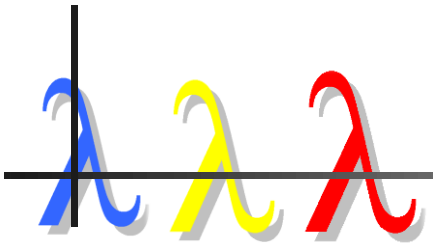


Consider computing the average of a vector of numbers

```
; avg-vector: (vector number) → number
; Purpose: To compute the average of the given vector
(define (avg-vector V)
  (local [; sum-VINTV: int int → number
          ; Purpose: For the given VINTV, sum the elements of V
          (define (sum-VINTV low high)
            (cond [(empty-VINTV? low high) …]
                  [else (vector-ref V high)…(sum-VINTV low (sub1 high))]))]
    (/ (sum-elems 0 (sub1 (vector-length V)))
       (vector-length V))))
```

What is the answer if VINTV is empty?

Tackling vectors

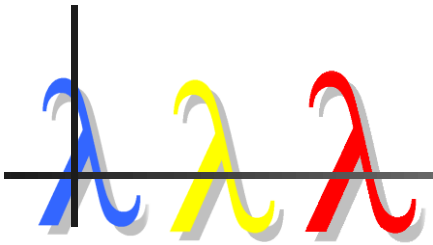


Consider computing the average of a vector of numbers

```
; avg-vector: (vector number) → number
; Purpose: To compute the average of the given vector
(define (avg-vector V)
  (local [; sum-VINTV: int int → number
          ; Purpose: For the given VINTV, sum the elements of V
          (define (sum-VINTV low high)
            (cond [(empty-VINTV? low high) 0]
                  [else (vector-ref V high)…(sum-VINTV low (sub1 high))]))]
    (/ (sum-elems 0 (sub1 (vector-length V)))
       (vector-length V))))
```

What is the answer if VINTV is not empty?

Tackling vectors

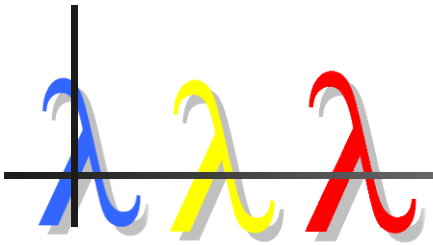


Consider computing the average of a vector of numbers

```
; avg-vector: (vector number) → number
; Purpose: To compute the average of the given vector
(define (avg-vector V)
  (local [; sum-VINTV: int int → number
          ; Purpose: For the given VINTV, sum the elements of V
          (define (sum-VINTV low high)
            (cond [(empty-VINTV? low high) 0]
                  [else (+ (vector-ref V high)  (sum-VINTV low (sub1 high)))]))]
    (/ (sum-elems 0 (sub1 (vector-length V)))
       (vector-length V))))
```

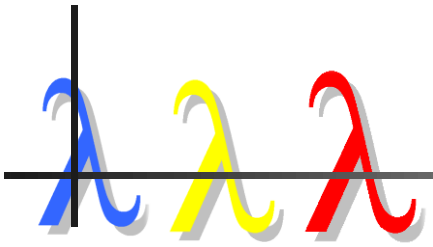
A valid VINTV for V → No indexing errors are possible!

Tackling vectors



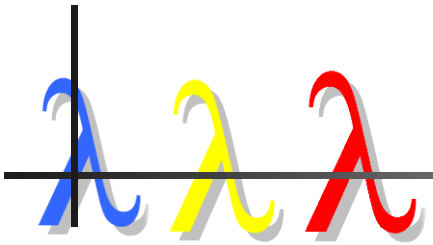
- Consider insertion-sorting in place
- Problem analysis
  - Sort the entire vector: vector interval [0..(sub1 (vector-length V))]
- To sort
  - empty vector interval → stop
  - Insert first element in the sorted rest of the vector interval
  - Process from low to high

Tackling vectors



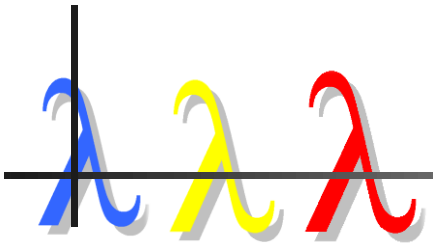
```
; f-on-vector: (vector X) →
; Purpose:
; Effect:  ← for vector mutator template
(define (insort-in-place! V)
 (local [  ; f-on-VINTV: int int →
           ; Purpose: For the given VINTV, ...
           (define (f-on-VINTV low high)
             (cond [(empty-VINTV? low high) ...]
                   [else (vector-ref V high)...(f-on-VINTV low (sub1 high))]))
           ; f-on-VINTV2: VINT: int int →
           ; Purpose: For the given VINTV2, ...
           (define (f-on-VINTV2 low high)
             (cond [(empty-VINTV2? high low) ...]
                   [else (vector-ref V low)...(f-on-VINTV2 (add1 low) high)]))]
   ...))
```

Tackling vectors



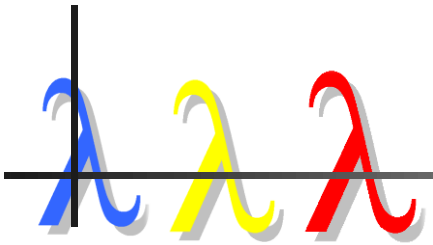
```
; insort-in-place!: (vector number) → (void)
; Purpose: To sort the given vector in non-decreasing order
; Effect: To rearrange the elements of the given vector in non-decreasing order
(define (insort-in-place! V)
 (local [  ; f-on-VINTV: int int →
           ; Purpose: For the given VINTV, ...
           (define (f-on-VINTV low high)
             (cond [(empty-VINTV? low high) ...]
                   [else (vector-ref V high)...(f-on-VINTV low (sub1 high))]))
           ; f-on-VINTV2: VINT: int int →
           ; Purpose: For the given VINTV2, ...
           (define (f-on-VINTV2 low high)
             (cond [(empty-VINTV2? high low) ...]
                   [else (vector-ref V low)...(f-on-VINTV2 (add1 low) high)]))]
   (sort! 0   (sub1 (vector-length V)))))
```

Tackling vectors



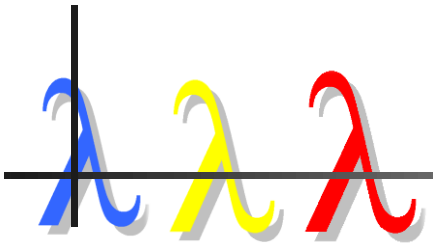
```
; insort-in-place!: (vector number) → (void)
; Purpose: To sort the given vector in non-decreasing order
; Effect: To rearrange the elements of the given vector in non-decreasing order
(define (insort-in-place! V)
 (local [

          …

          ; sort!: int int →  (void)
          ; Purpose: For the given VINTV2, sort V using insertion sort
          ; Effect: Rearrange V elements in the given VINTV2 in non-decreasing order
          (define (sort! low high)
              (cond [(empty-VINTV2? high low)  (void)]
                        [else (begin   (sort! (add1 low) high)   (insert!   ???   ???))]
   (sort! 0   (sub1 (vector-length V)))))
```

Tackling vectors



```
; insort-in-place!: (vector number) → (void)
; Purpose: To sort the given vector in non-decreasing order
; Effect: To rearrange the elements of the given vector in non-decreasing order
(define (insort-in-place! V)
 (local [

          …

          ; sort!: int int →  (void)
          ; Purpose: For the given VINTV2, sort V using insertion sort
          ; Effect: Rearrange V elements in the given VINTV2 in non-decreasing order
          (define (sort! low high)
              (cond [(empty-VINTV2? high low)  (void)]
                        [else (begin   (sort! (add1 low) high)   (insert!   low high))]
   (sort! 0   (sub1 (vector-length V)))))
```

Tackling vectors



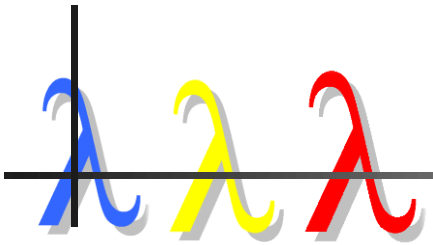
- Consider the problem of inserting

- To insert
  - Start at the low element
  - Stop if interval is empty or adjacent elements are in order
  - Otherwise,

    swap low and (add1 low)

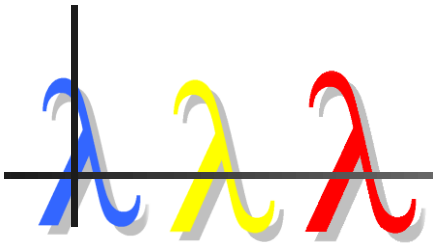
    insert in the rest of the interval

Tackling vectors



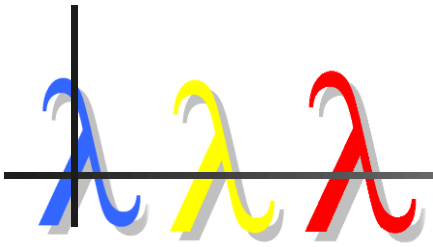
```
; f-on-VINTV2: int int →
; Purpose: For the given VINTV2, ...
(define (f-on-VINTV2 low high)
    (cond
        [(empty-VINTV2? low high) ...]
        [else (vector-ref V low)...(f-on-VINTV2 (add1 low)  high)]))
```

Tackling vectors



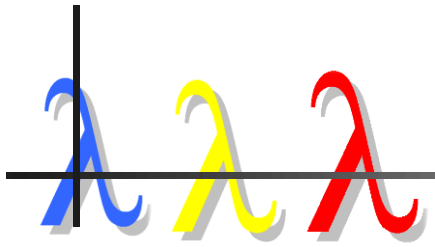
```
; insert!: int int → (void)
; Purpose: For the given VINTV2, insert V[low] in V[low+1..high]
;                such that V[low..high] is in non-decreasing order
; Effect: V elements are swapped until one is >= V[low] or
;           the given VINTV2 is empty
(define (insert! low high)
    (cond
          [(empty-VINTV2? low high)  (void)]
          [else (cond [(<= (vector-ref V low)  (vector-ref V (add1 low)))
                       (void)]]
                            ))
```

Tackling vectors



```
; insert!: int int → (void)
; Purpose: For the given VINTV2, insert V[low] in V[low+1..high]
;                such that V[low..high] is in non-decreasing order
; Effect: V elements are swapped until one is >= V[low] or
;           the given VINTV2 is empty
(define (insert! low high)
    (cond
          [(empty-VINTV2? low high)  (void)]
          [else (cond [(<= (vector-ref V low)  (vector-ref V (add1 low)))
                       (void)]]
                      [else  (begin   (swap low (add1 low))
                                      (insert! (add1 low) high))])]))
```

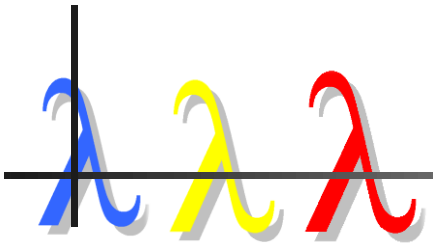
Tackling vectors



```
; insert!: int int → (void)
; Purpose: For the given VINTV2, insert V[low] in V[low+1..high]
;                such that V[low..high] is in non-decreasing order
; Effect: V elements are swapped until one is >= V[low] or
;         the given VINTV2 is empty
(define (insert! low high)
    (cond
        [(empty-VINTV2? low high)  (void)]
        [else (cond [(<= (vector-ref V low)  (vector-ref V (add1 low)))
                     (void)]]
                    [else  (begin   (swap low (add1 low))
                                    (insert! (add1 low) high))])]))
```

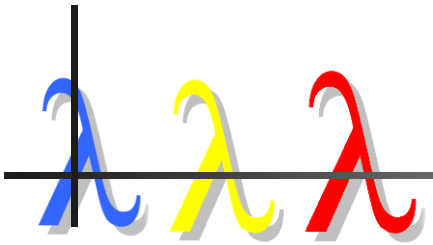
vector-ref: index is out of range
index: 5
valid range: [0, 4]
vector: (vector 10 3 7 17 11)

Tackling vectors



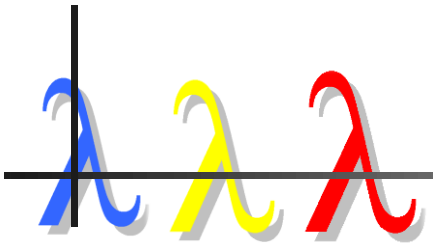
```
; insort-in-place!: (vector number) → (void)
; Purpose: To sort the given vector in non-decreasing order
; Effect: To rearrange the elements of the given vector in non-decreasing order
(define (insort-in-place! V)
 (local [

         …

         ; sort!: int int →  (void)
         ; Purpose: For the given VINTV2, sort V using insertion sort
         ; Effect: Rearrange V elements in the given VINTV2 in non-decreasing order
         (define (sort! low high)
            (cond [(empty-VINTV2? high low)  (void)]                    low (sub1 high)
                  [else (begin   (sort! (add1 low) high)   (insert!   low high))]
   (sort! 0   (sub1 (vector-length V))))
```

Extending the power



- Article contains other examples
  - Dot product: Process multiple VINTVs in step
  - Merge: Process multiple VINTVs not in step

Concluding Remarks



- We ought to exploit *vector intervals* to design vector processing functions in CS1-2
  - Perhaps beyond!
- Reasoning about vector intervals provides beginners with a framework for properly indexing a vector
- Future work
  - Extend the application of vector intervals to generative and accumulative recursion (e.g. quick and heap sort)
  - Multidimensional vectors
  - Object-oriented design

ANY QUESTIONS?

TFPIE now on FB! TFPIE wiki