

Vectors in Racket:

```
(define nums (vector 4 8 15 16 23 42))  
(check-expect (vector-ref 0) 4)  
(check-expect (vector-ref 1) 8)  
(check-expect (vector-length num) 6)
```

Arrays in Java:

```
int[] nums = {4, 8, 15, 16, 23, 42}  
t.checkExpect(nums[0], 4);  
t.checkExpect(nums[1], 8);  
t.checkExpect(nums.length, 6);
```

Vectors in Racket:

```
(define nums (make-vector 10))  
(check-expect (vector-ref nums 1) 0)  
(vector-set! nums 1 17)  
(check-expect (vector-ref nums 1) 17)
```

Arrays in Java:

```
int[] nums = new int[10];  
t.checkExpect(nums[1], 0);  
nums[1] = 17;  
t.checkExpect(nums[1], 17);
```

Functions on Vectors

How do you sum the numbers in a vector?

```
; sum : vector-of-num -> num
(define (sum nums)
  (sum-from nums 0))
```

```
; sum-from : vector-of-num nat -> num
(define (sum-from nums pos)
  (cond
    [(= pos (vector-length nums)) 0]
    [else (+ (vector-ref nums pos)
              (sum-from nums (add1 pos)))])))
```

Summing a Vector

```
(define (sum nums)
  (sum-from nums 0))
```

```
(define (sum-from nums pos)
  (cond
    [(= pos (vector-length nums)) 0]
    [else (+ (vector-ref nums pos)
              (sum-from nums (add1 pos)))]))
```

```
(sum (vector 7 5 9))
```

→

```
(sum-from (vector 7 5 9) 0)
```

Summing a Vector

```
(define (sum nums)
  (sum-from nums 0))
```

```
(define (sum-from nums pos)
  (cond
    [(= pos (vector-length nums)) 0]
    [else (+ (vector-ref nums pos)
              (sum-from nums (add1 pos)))]))
```

```
(sum-from (vector 7 5 9) 0)
```

→

```
(+ 7 (sum-from (vector 7 5 9) 1))
```

Summing a Vector

```
(define (sum nums)
  (sum-from nums 0))
```

```
(define (sum-from nums pos)
  (cond
    [(= pos (vector-length nums)) 0]
    [else (+ (vector-ref nums pos)
              (sum-from nums (add1 pos)))]))
```

```
(+ 7 (sum-from (vector 7 5 9) 1))
```

→

```
(+ 7 (+ 5 (sum-from (vector 7 5 9) 2)))
```

Summing a Vector

```
(define (sum nums)
  (sum-from nums 0))
```

```
(define (sum-from nums pos)
  (cond
    [(= pos (vector-length nums)) 0]
    [else (+ (vector-ref nums pos)
              (sum-from nums (add1 pos)))]))
```

```
(+ 7 (+ 5 (sum-from (vector 7 5 9) 2)))
```

→

```
(+ 7 (+ 5 (+ 9 (sum-from (vector 7 5 9) 3))))
```

Summing a Vector

```
(define (sum nums)
  (sum-from nums 0))
```

```
(define (sum-from nums pos)
  (cond
    [(= pos (vector-length nums)) 0]
    [else (+ (vector-ref nums pos)
              (sum-from nums (add1 pos)))]))
```

```
(+ 7 (+ 5 (+ 9 (sum-from (vector 7 5 9) 3))))
```

→

```
(+ 7 (+ 5 (+ 9 0)))
```

Methods on Arrays

How do you sum the numbers in an array?

```
static int sum(int nums[]) {  
    return sumFrom(nums, 0);  
}
```

```
static int sumFrom(int nums[], int pos) {  
    if (pos == nums.length)  
        return 0;  
    else  
        return nums[pos] + sumFrom(nums, pos+1);  
}
```

... but this probably fails on `new int[10000]`

Running out of Stack

`sum(new int[10000])` leads to

`nums[0] + nums[1] + nums[2] + ...`

- Waiting additions are the **continuation**
- The continuation might be represented as a **stack**
- The size of the stack might be limited

Accumulating to Keep the Continuation Small

```
; sum : vector-of-num -> num
(define (sum nums)
  (sum-from-onto nums 0 0))

; sum-from : vector-of-num nat -> num
(define (sum-from-onto nums pos result)
  (cond
    [(= pos (vector-length nums)) result]
    [else
     (sum-from-onto nums (add1 pos)
                     (+ result
                        (vector-ref nums pos))))]))
```

Summing a Vector with an Accumulator

```
(define (sum-from-onto nums pos result)
  (cond
    [(= pos (vector-length nums)) result]
    [else
     (sum-from-onto nums (add1 pos)
                     (+ result
                        (vector-ref nums pos))))]))
```

```
(sum (vector 7 5 9))
```

→

```
(sum-from-onto (vector 7 5 9) 0 0)
```

Summing a Vector with an Accumulator

```
(define (sum-from-onto nums pos result)
  (cond
    [(= pos (vector-length nums)) result]
    [else
     (sum-from-onto nums (add1 pos)
                     (+ result
                        (vector-ref nums pos)))]))
```

```
(sum-from-onto (vector 7 5 9) 0 0)
```

→

```
(sum-from-onto (vector 7 5 9) 1 7)
```

Summing a Vector with an Accumulator

```
(define (sum-from-onto nums pos result)
  (cond
    [(= pos (vector-length nums)) result]
    [else
     (sum-from-onto nums (add1 pos)
                    (+ result
                       (vector-ref nums pos))))]))
```

```
(sum-from-onto (vector 7 5 9) 1 7)
```

→

```
(sum-from-onto (vector 7 5 9) 2 12)
```

Summing a Vector with an Accumulator

```
(define (sum-from-onto nums pos result)
  (cond
    [(= pos (vector-length nums)) result]
    [else
     (sum-from-onto nums (add1 pos)
                    (+ result
                       (vector-ref nums pos))))]))
```

```
(sum-from-onto (vector 7 5 9) 2 12)
```

→

```
(sum-from-onto (vector 7 5 9) 3 21)
```

Summing a Vector with an Accumulator

```
(define (sum-from-onto nums pos result)
  (cond
    [(= pos (vector-length nums)) result]
    [else
     (sum-from-onto nums (add1 pos)
                     (+ result
                        (vector-ref nums pos))))]))

(sum-from-onto (vector 7 5 9) 3 21)
```

→

21

An Accumulator in Java

```
static int sum(int nums[]) {  
    return sumFromOnto(nums, 0, 0);  
}
```

```
static int sumFromOnto(int nums[], int pos, int result) {  
    if (pos == nums.length)  
        return result;  
    else  
        return sumFromOnto(nums, pos+1, result+nums[pos]);  
}
```

```
int[] nums = {5, 7, 9};
```

```
sum(nums)
```

→

```
return sumFromOnto(nums, 0, 0)
```

An Accumulator in Java

```
static int sum(int nums[]) {  
    return sumFromOnto(nums, 0, 0);  
}
```

```
static int sumFromOnto(int nums[], int pos, int result) {  
    if (pos == nums.length)  
        return result;  
    else  
        return sumFromOnto(nums, pos+1, result+nums[pos]);  
}
```

```
int[] nums = {5, 7, 9};
```

```
return sumFromOnto(nums, 0, 0)
```

→

```
return return sumFromOnto(nums, 1, 7)
```

An Accumulator in Java

```
static int sum(int nums[]) {  
    return sumFromOnto(nums, 0, 0);  
}
```

```
static int sumFromOnto(int nums[], int pos, int result) {  
    if (pos == nums.length)  
        return result;  
    else  
        return sumFromOnto(nums, pos+1, result+nums[pos]);  
}
```

```
int[] nums = {5, 7, 9};
```

```
return return sumFromOnto(nums, 1, 7)
```

→

```
return return return sumFromOnto(nums, 2, 12)
```

When in Rome...

Java needs a hint:

- Convert accumulators to variables
- Stay in the same function instead of making a new call
- **for** is the key to staying in the same call

Using for

```
static int sum(int nums[]) {  
    int result = 0;  
    for (int n : nums) {  
        result = result + n;  
    }  
    return result;  
}
```

- An `int[]` is an `Iterable<int>`
- `for` works on any `Iterable<E>`
- `Collection<E>` extends `Iterable<E>`

Using for, Long Version

```
static int sum(int nums[]) {  
    int result = 0;  
    for (int pos = 0;  
         pos < nums.length;  
         pos = pos + 1) {  
        result = result + nums[pos];  
    }  
    return result;  
}
```

- This variant of **for** covers anything that preserves the continuation

for in Full Racket

```
(define (sum nums)
  (for/fold ([result 0]) ([n nums])
    (+ result n)))
```

```
(define (sum nums)
  (define len (vector-length nums))
  (for/fold ([result 0]) ([pos (in-range 0 len)])
    (+ result (vector-ref nums pos))))
```

while and do

```
while (test) { ... }
```

is a shorthand for

```
for (; test; ) { ... }
```

```
do { ... } while (test);
```

is a shorthand for

```
for (boolean ok=true; ok; ) { ... ok = test; }
```

Terminology

- **recursion** — when a function calls itself
- **loop** — recursion without building up work
 - Arguments in a loop can be converted to variables
 - **for** supports only loops