

### sum(term, a next, b)

term: fn applied to each term  
 term(a): first term  
 next: applied b - a times  
 b: no. of terms (0 is returned for the (b+1)<sup>th</sup> term)

### sum implementation

```
def sum(term, a, next, b):
    sum(a, b) = t(a) + t(n(a)) + t(n2(a)) + ... + t(n(b-a)(a)) + 0
    if a > b: return 0
    else: return term(a) + sum(term, next(a), next, b)
```

### fold(op, f, n)

n: op applied for n times  
 fold\_right:  
 $(f(n) \oplus (f(n-1) \oplus (f(n-2) \dots \oplus (f(1) \oplus f(0))))$   
 $(4 - (3 - (2 - (1 - 0))))$   
 fold\_left:  
 $((((f(0) \oplus f(1)) \oplus f(2)) \oplus \dots f(n) \rightarrow n + 1 \text{ times}$   
 Matters for non-associative operations e.g.  $((1 - 2) - 3) - 4$

### fold\_right

```
fold(..) = op(f(n), fold(n-1))
# = f(n) \oplus fold(n-1)
# = f(n) \oplus [ f(n-1) \oplus fold(n-2) ]
# = f(n) \oplus f(n-1) \oplus fold(n-2) --- assumes op is assoc./commu.
# = f(n) \oplus f(n-1) \oplus f(n-2) \oplus \dots \oplus f(n-(n-1)) \oplus fold(n-n)
# = f(n) \oplus f(n-1) \oplus f(n-2) \oplus \dots \oplus f(n-(n-1)) \oplus f(0)
```

### fold\_left

```
fold(n)
=> fold(n-1) \oplus f(n)
=> (fold(n-2) \oplus f(n-1)) \oplus f(n)
=> ((fold(n-3) \oplus f(n-2)) \oplus f(n-1)) \oplus f(n)
:
=> (((f(0) \oplus f(1)) \oplus f(2)) \oplus f(3)) \dots \oplus f(n)
```

### accumulate(fn, initial, seq)

```
>>> accumulate(lambda x, y: x + y, 0, (1, 2, 3, 4, 5))
15
>>> accumulate(lambda x, y: (x, y), (), (1, 2, 3, 4, 5))
(1, (2, (3, (4, (5, ())))))
```

seq contains n elements  
 when seq[n] == () i.e. index out of range,  
 initial is returned

### enumerate\_interval(low, high)

```
enumerate_interval(2, 7)
| (2, 3, 4, 5, 6, 7)
```

```
enumerate_interval(1, 1)
| (1,)
```

```
enumerate_interval(4, -1)
| ()
```

### sum and product using fold

```
def sum_of_int(a, b):
    return fold(op = lambda x, y: x + y, f = lambda x: a + x, n = b - a)
# a + (a + 1) + (a + 2) + ... + b - 1, b
# b = a + (b - a)
def product_int(a, b):
    return fold(op = lambda x, y: x * y, f = lambda x: a + x, n = b - a)
```

### fold2(op, term, a, next, b, base)

```
fold2() = term(a) \oplus [term(next(a)) \oplus [.. \oplus base]
#op applied for (b-a) times
#base occurs when b > a
```

### iterative fold (left)

```
def iter_fold(op, f, n):
    res = f(0)
    for i in range(n):
        res = op(res, f(i + 1))
    return res
```

