

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)th 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:
 $((f(0) \oplus f(1)) \oplus f(2)) \oplus \dots f(n) \rightarrow n + 1$ times
 Matters for non-associative operations e.g. $((1 - 2) - 3) - 4$

fold_right

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

fold_left

```
fold(n)
=> fold(n-1) ⊕ f(n)
=> (fold(n-2) ⊕ f(n-1)) ⊕ f(n)
=> ((fold(n-3) ⊕ f(n-2)) ⊕ f(n-1)) ⊕ f(n)
:
=> (((f(0) ⊕ f(1)) ⊕ f(2)) ⊕ f(3)) ... ⊕ 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) ⊕ [term(next(a)) ⊕ [.. ⊕ 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
```