length

```
def fn(l: List): Int =
  if l.isEmpty then 0
  else 1 + fn(l.tail)
```

product

```
def fn(l: List): Int =
  if l.isEmpty then 1
  else l.head * fn(l.tail)
```

increment

```
def fn(l: List): List =
  if l.isEmpty then Nil()
  else Cons(l.head + 1, fn(l.tail))
```

allEven

```
def fn(l: List): Boolean =
  if l.isEmpty then true
  else (l.head % 2 == 0) && fn(l.tail)
```

append

```
def fn(l1: List, l2: List): List =
  if l1.isEmpty then l2
  else Cons(l1.head, fn(l1.tail, l2))
```

countPositive

```
def fn(l: List): Int =
  if l.isEmpty then 0
  else (if l.head > 0 then 1 else 0)
         + fn(l.tail)
```

allPositiveOrZero

```
def fn(l: List): Boolean =
  if l.isEmpty then true
  else l.head ≥ 0 && fn(l.tail)
```

anyOdd

```
def fn(l: List): Boolean =
  if l.isEmpty then false
  else (l.head % 2 ≠ 0) || fn(l.tail)
```

multiplyBy2

```
def fn(l: List): List =
  if l.isEmpty then Nil()
  else Cons(2 * l.head, fn(l.tail))
```

horner

```
def fn(x: Int, l: List): Int =
  if l.isEmpty then 0
  else l.head + x * fn(x, l.tail)
```

contains

```
def fn(l: List, n: Int): Boolean =
  if l.isEmpty then false
  else (l.head == n) || fn(l.tail, n)
```

removeOdd

```
def fn(l: List): List =
  if l.isEmpty then Nil()
  else if l.head % 2 ≠ 0 then fn(l.tail)
  else Cons(l.head, fn(l.tail))
```

sum

```
def fn(l: List): Int =
  if l.isEmpty then 0
  else l.head + fn(l.tail)
```

decrement

```
def fn(l: List): List =
  if l.isEmpty then Nil()
  else Cons(l.head - 1, fn(l.tail))
```

anyNegative

```
def fn(l: List): Boolean =
  if l.isEmpty then false
  else l.head < 0 || fn(l.tail)
```

reverseAppend

```
def fn(l1: List, l2: List): List =
  if l1.isEmpty then l2
  else fn(l1.tail, Cons(l1.head, l2))
```

isSubset

```
def fn(l: List, l1: List): Boolean =
  if l.isEmpty then true
  else contains(l1, l.head) && fn(l.tail, l1)
```

countEven

```
def fn(l: List): Int =
  if l.isEmpty then 0
  else (if l.head % 2 == 0 then 1 else 0)
         + fn(l.tail)
```

capAtZero

```
def fn(l: List): List =
  if l.isEmpty then Nil()
  else Cons(if l.head > 0 then 0 else l.head,
            fn(l.tail))
```

subtract

```
def fn(l: List): Int =
  if l.isEmpty then
    throw Exception("Empty list!")
  else if l.tail.isEmpty then l.head
  else l.head - fn(l.tail)
```

last

```
def fn(l: List): Int =
  if l.isEmpty then
    throw Exception("Empty list!")
  else if l.tail.isEmpty then l.head
  else fn(l.tail)
```

difference

```
def fn(l: List, ll: List): List =
  if l.isEmpty then Nil()
  else if contains(ll, l.head) then
    fn(l.tail, ll)
  else Cons(l.head, fn(l.tail, ll))
```

removeZeroes

```
def fn(l: List): List =
  if l.isEmpty then Nil()
  else if l.head == 0 then fn(l.tail)
  else Cons(l.head, fn(l.tail))
```

takeWhilePositive

```
def fn(l: List): List =
  if l.isEmpty then Nil()
  else if l.head > 0 then
    Cons(l.head, fn(l.tail))
  else Nil()
```

init

```
def fn(l: List): List =
  if l.isEmpty then
    throw Exception("Empty list!")
  else if l.tail.isEmpty then Nil()
  else Cons(l.head, fn(l.tail))
```

multiplyOdd

```
def fn(l: List): Int =
  if l.isEmpty then 1
  else
    val m = if l.head % 2 != 0
            then l.head else 1
    m * fn(l.tail)
```

collectEven

```
def fn(l: List): List =
  if l.isEmpty then Nil()
  else if l.head % 2 == 0 then
    Cons(l.head, fn(l.tail))
  else fn(l.tail)
```

collectMultiples

```
def fn(d: Int, l: List): List =
  if l.isEmpty then Nil()
  else if l.head % d == 0 then
    Cons(l.head, fn(d, l.tail))
  else fn(d, l.tail)
```

intersection

```
def fn(l: List, ll: List): List =
  if l.isEmpty then Nil()
  else if contains(ll, l.head) then
    Cons(l.head, fn(l.tail, ll))
  else fn(l.tail, ll)
```

min

```
def fn(l: List): Int =
  if l.isEmpty then
    throw Exception("Empty list!")
  else if l.tail.isEmpty then l.head
  else
    val m = fn(l.tail)
    if l.head < m then l.head else m
```