6. Functional Programming

6.1. Recursion

Defining solution of a problem in terms of the same problem, typically of smaller size, is called recursion. Recursion makes it possible to express solution of a problem very concisely and elegantly.

A function is called recursive if it makes call to itself. Typically, a recursive function will have a terminating condition and one or more recursive calls to itself.

6.1.1. Example: Computing Exponent

Mathematically we can define exponent of a number in terms of its smaller power.

```
def exp(x, n):
    """
    Computes the result of x raised to the power of n.

        >>> exp(2, 3)
        8
        >>> exp(3, 2)
        9
    """
    if n == 0:
        return 1
    else:
        return x * exp(x, n-1)
```

Lets look at the execution pattern.

```
exp(2, 4)
+-- 2 * exp(2, 3)
|       +-- 2 * exp(2, 2)
|       |       +-- 2 * exp(2, 1)
|       |       |       +-- 2 * exp(2, 0)
|       |       |       |       +-- 1
|       |       |       +-- 2 * 1
|       |       |       +-- 2
|       |       +-- 2 * 2
|       |       +-- 4
|       +-- 2 * 4
|       +-- 8
+-- 2 * 8
+-- 16
```

Number of calls to the above `exp` function is proportional to size of the problem, which is `n` here.

We can compute exponent in fewer steps if we use successive squaring.

```python
def fast_exp(x, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return fast_exp(x*x, n/2))
    else:
        return x * fast_exp(x, n-1)
```

Lets look at the execution pattern now.

```
fast_exp(2, 10)
+-- fast_exp(4, 5) # 2 * 2
|   +-- 4 * fast_exp(4, 4)
|   |       +-- fast_exp(16, 2) # 4 * 4
|   |       |   +-- fast_exp(256, 1) # 16 * 16
|   |       |   |   +-- 256 * fast_exp(256, 0)
|   |       |   |             +-- 1
|   |       |   |   +-- 256 * 1
|   |       |   |   +-- 256
|   |       |   +-- 256
|   |       +-- 256
|   +-- 4 * 256
|   +-- 1024
+-- 1024
1024
```

Problem 1: Implement a function `product` to multiply 2 numbers recursively using `+` and `-` operators only.