③ 再帰呼出し

アルゴリズムの学習の準備として、再帰呼出しを学習します。

再帰呼出しとは

- アルゴリズムの講義では「再帰呼出し」というものがしばしば出てきます
- 再帰
  - あるものの記述の中に、そのもの自身が含まれていること
- 再帰呼出し
  - プログラムにおいて、関数の中でその関数自身を呼び出すこと

階乗の計算

- 自然数を引数にとり、その階乗を返す関数 fact(n) を考えてみます
- for文を利用すると以下のようになりますね

```
def fact(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```

```
fact(10)
```

```
3628800
```

階乗の計算：再帰呼出しの例

- この関数は、再帰呼び出しを利用すると、以下のように記述することができます

```
def fact(n):
    if n <= 1:
        return 1
    return n * fact(n - 1)
```

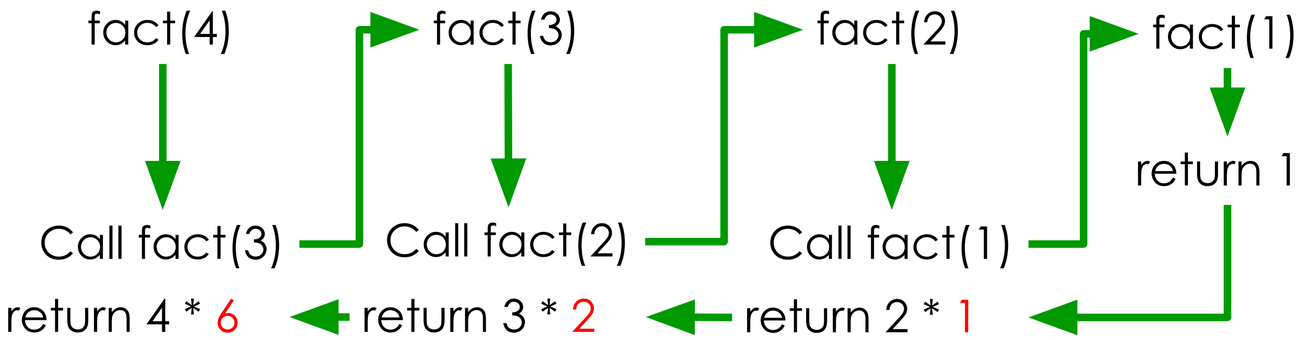
```
fact(10)
```

```
3628800
```

再帰呼出しの例

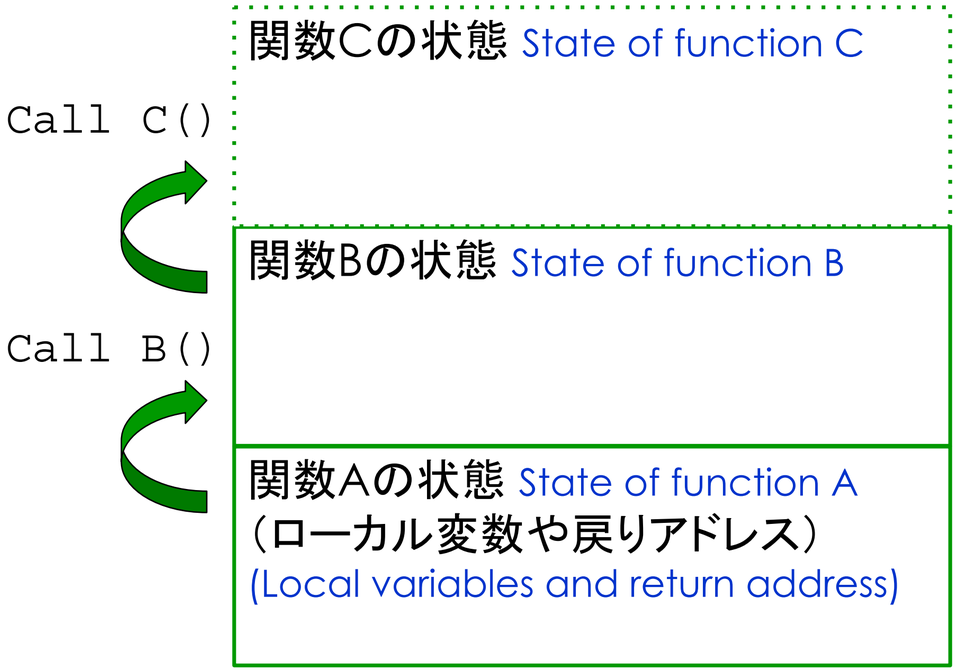
- 例えば、fact(4) を呼び出すと、以下のような手順で実行されて結果が求まります

```
def fact(n):
    if n <= 1:
        return 1
    return n * fact(n - 1)
```



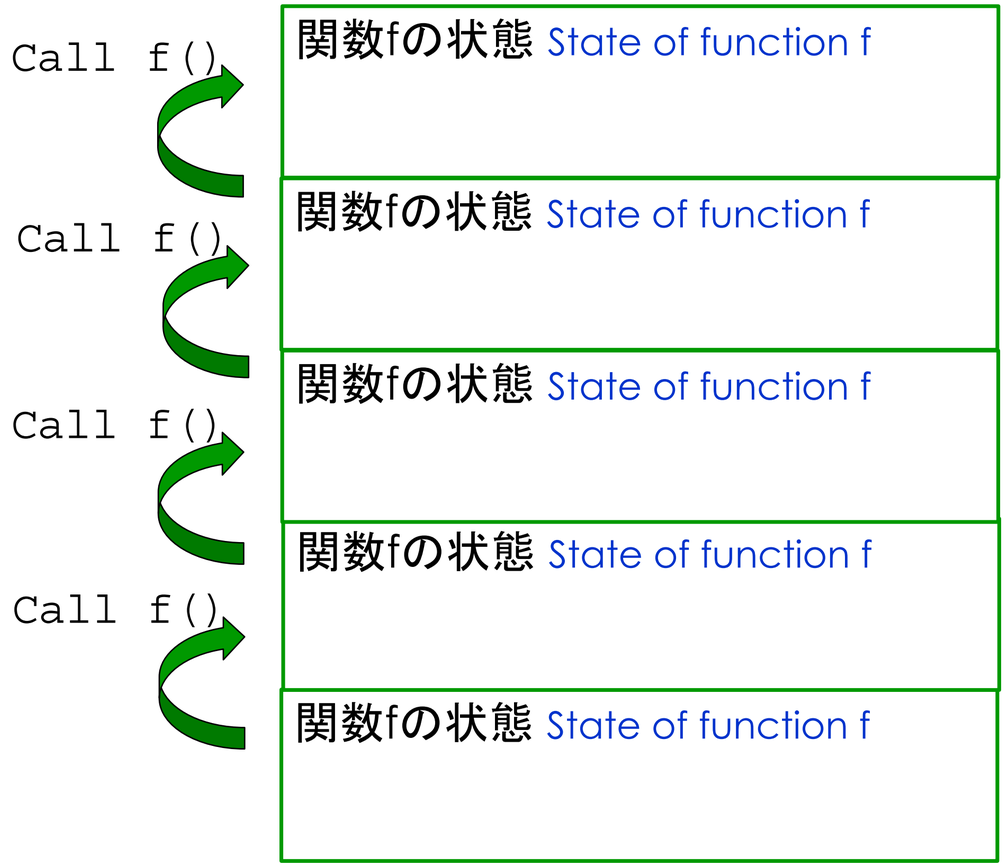
注意：再帰呼出しとメモリ消費

- 関数呼出しを行う際には、メモリを消費します
  - 関数Aの中で関数Bを呼び出す時には、関数Aの状態（変数の値など）や戻り先のアドレスを、覚えておかなければなりません
  - このことを実現するために、関数呼び出しの度にメモリ上に、必要なデータ（変数の値や戻り先のアドレス）を保存します
  - 関数呼出しが繰り返されると、メモリ上にデータが毎回積み上がっていきます



注意：再帰呼出しとスタック

- 再帰呼出しを行うと、呼出しの都度、メモリ上にデータが積み重なっていきます
- そのため、あまりに再帰呼出しを繰り返すと、最終的にメモリがあふれてしまいます
  - 言語によって挙動は違いますが、Pythonの場合は、再帰呼出しの上限が決まっています
- 再帰呼出しを利用してはいけない、ということではありませんが、メモリを消費することは覚えておきましょう



試してみましょう

- ためしに、正の整数 n を引数にとり、1からnまでの総和を返す関数 sigma(n) を再帰呼出しを使って定義します
- sigma(10000) を呼び出してみると…

```
def sigma(n):
    if n <= 1:
        return 1
    return n + sigma(n - 1)
```

```
sigma(10000)
```

むやみやたらに再帰呼出しを使わない方がよさそうですね？

なぜ「再帰呼出し」を使うのか？

- アルゴリズムの中には、大きな問題を解く時に、小問題に分割して解いていく、という方法をとるものが多数あります
  - 問題を分割していくと、最終的には自明な問題に帰着し、最後にそれをまとめるというイメージです
  - 一般に「分割統治法」と呼ばれるアプローチです
- このようなアルゴリズムでは、一般に分割した小問題に「再帰呼出し」を適用することを繰り返すことで、分かりやすくプログラムを記述できます

二分探索：再帰版

- 先ほど学習した、二分探索も、再帰呼出しを利用すると、以下のように記述することができます
  - ここではリストのスライスを用いてリストを分割し、再帰的に探索を行っています

```
def binary_search(lst, x):
    if len(lst) == 0:
        return False
    center = len(lst) // 2
    if lst[center] == x:
        return True
    elif lst[center] > x:
        return binary_search(lst[:center], x)
    else:
        return binary_search(lst[center + 1:], x)
```

80を探す例だと…