

## Лекция 6

---

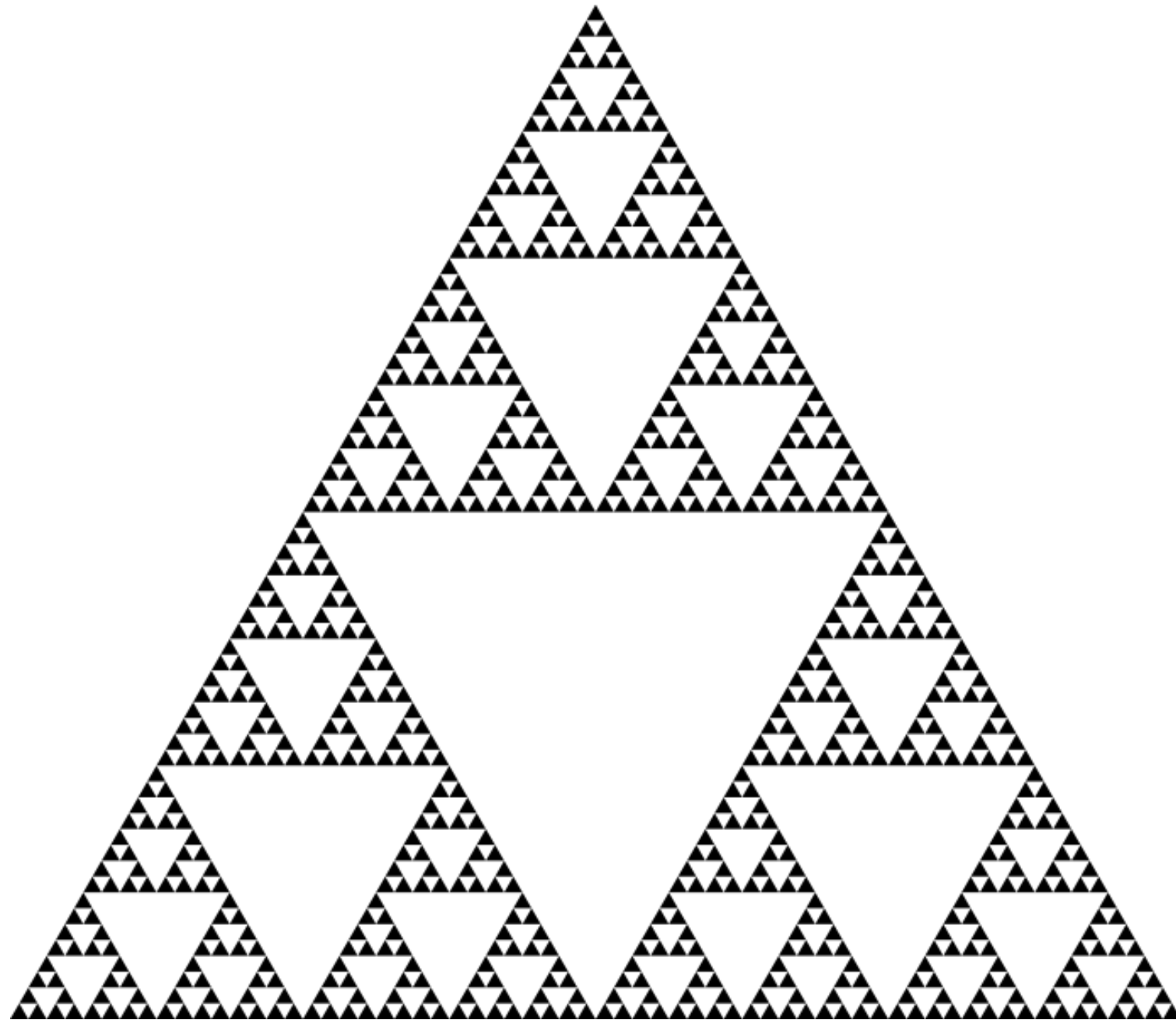
# Рекурсивные функции

# Рекурсивные функции

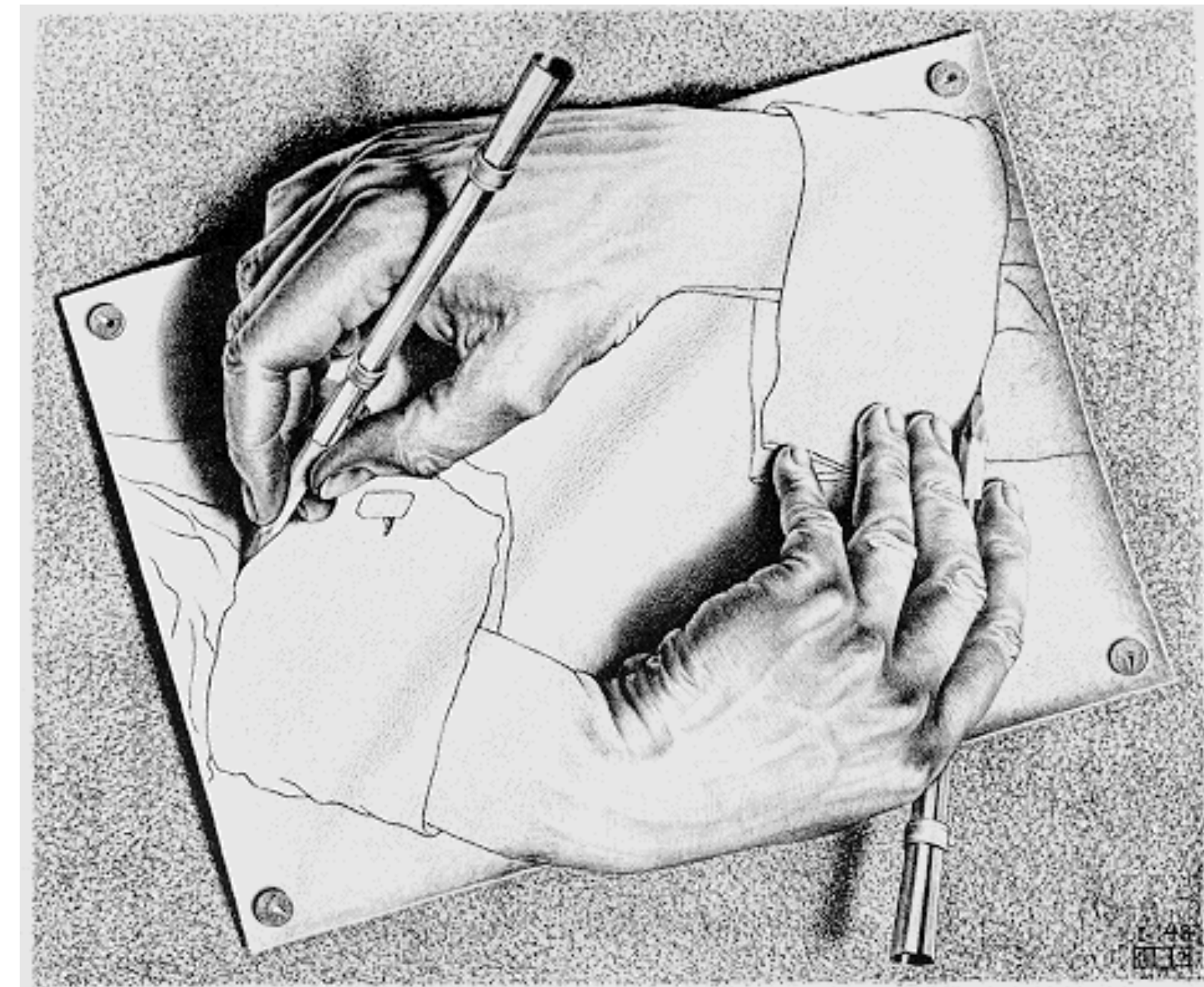
---

## Определение

Рекурсивной называется функция, тело которой прямо или косвенно вызывает само себя.



Треугольник Серпинского (1915)



Рисующие руки, Морис Корнелиус Эшер (литография, 1948)

## Суммирование цифр

---

$$1 + 3 + 3 + 7 = 14$$

- Если число  $a$  делится на 9, то сумма его цифр – `sum_digits(a)` – также делится на 9.
- Полезно для определения опечаток!

**The Bank of Python**  
1234 5678 9098 7658  
MONTY THE LORD

Цифра с контрольной суммой – функция от остальных цифр. Она может использоваться при определении опечаток.

- В номерах пластиковых карт используется алгоритм Луна, который будет рассмотрен чуть позже.

## Суммирование цифр без инструкции «while»

---

```
def split(n):  
    """Разделяет положительное n на последнюю цифру и все остальные."""  
    return n // 10, n % 10  
  
def sum_digits(n):  
    """Возвращает сумму цифр положительного целого n."""  
    if n < 10:  
        return n  
    else:  
        all_but_last, last = split(n)  
        return sum_digits(all_but_last) + last
```

# Структура рекурсивной функции

---

- Заголовок инструкции `def` как и в других функциях
- Инструкция ветвления отделяет **простые (базовые)** случаи
- Простые случаи выполняются **без рекурсивных вызовов**
- Рекурсивные случаи выполняются **с рекурсивными вызовами**

```
def sum_digits(n):
```

```
    """Возвращает сумму цифр положительного целого n."""
```

```
    if n < 10:
```

```
        return n
```

```
    else:
```

```
        all_but_last, last = split(n)
```

```
        return sum_digits(all_but_last) + last
```

(Пример)

# Рекурсия на диаграммах окружения

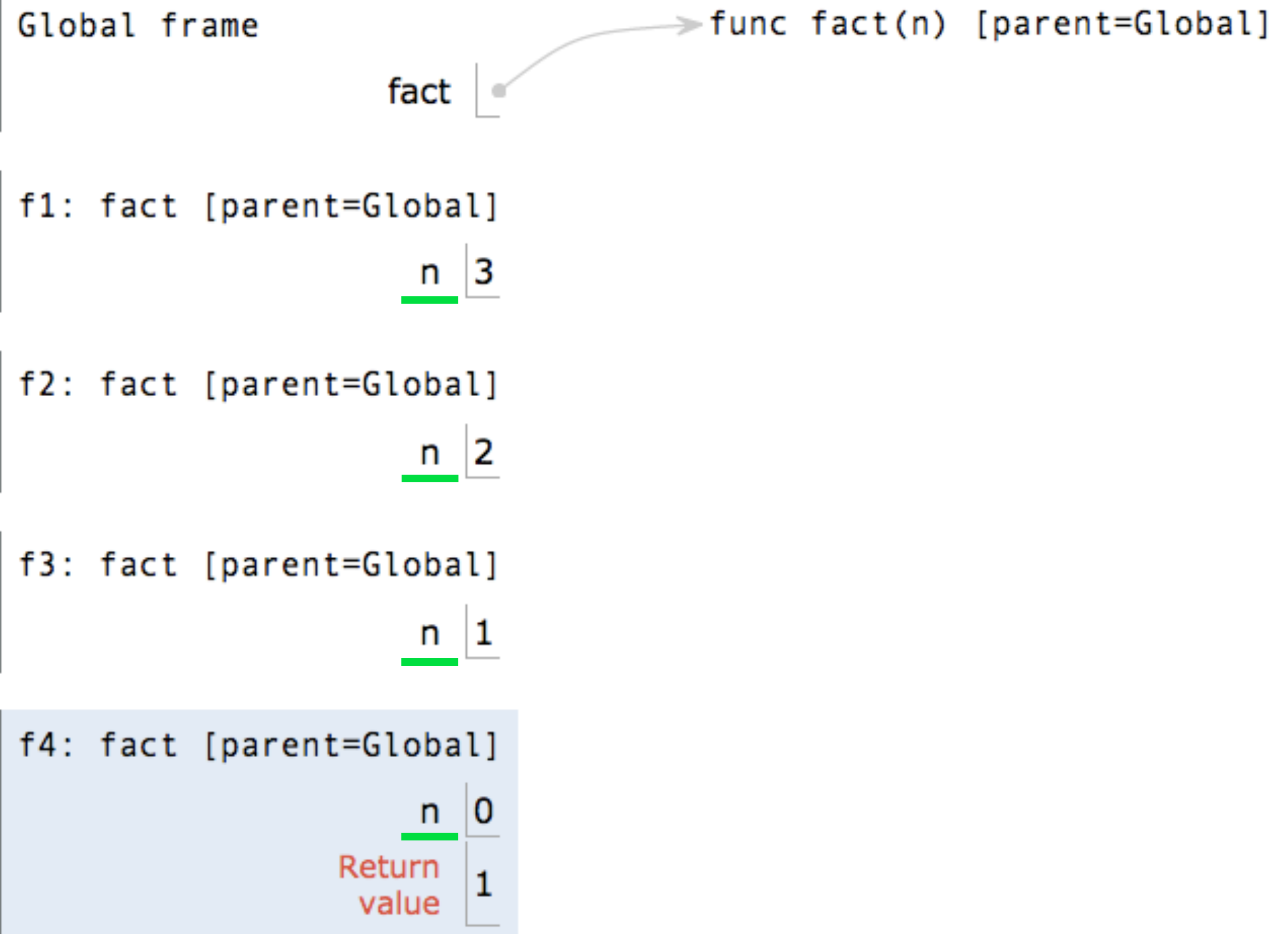


# Рекурсия на диаграммах окружения

```
1 def fact(n):  
2     if n == 0:  
3         return 1  
4     else:  
5         return n * fact(n-1)  
6  
7 fact(3)
```

- Одна и та же функция **fact** вызывается множество раз.
- Различные фреймы хранят разные аргументы для разных вызовов.
- Значение **n** зависит от текущего окружения.
- Каждый вызов **fact** решает более простую задачу, чем предыдущий: **n** уменьшается.

(Пример)





# Итерации vs рекурсия

---

Итерация – это частный случай рекурсии.

$$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$$

Используя while:

```
def fact_iter(n):  
    total, k = 1, 1  
    while k <= n:  
        total, k = total*k, k+1  
    return total
```

Математически:

$$n! = \prod_{k=1}^n k$$

Имена: n, total, k, fact\_iter

Используя рекурсию:

```
def fact(n):  
    if n == 0:  
        return 1  
    else:  
        return n * fact(n-1)
```

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n - 1)! & \text{otherwise} \end{cases}$$

n, fact

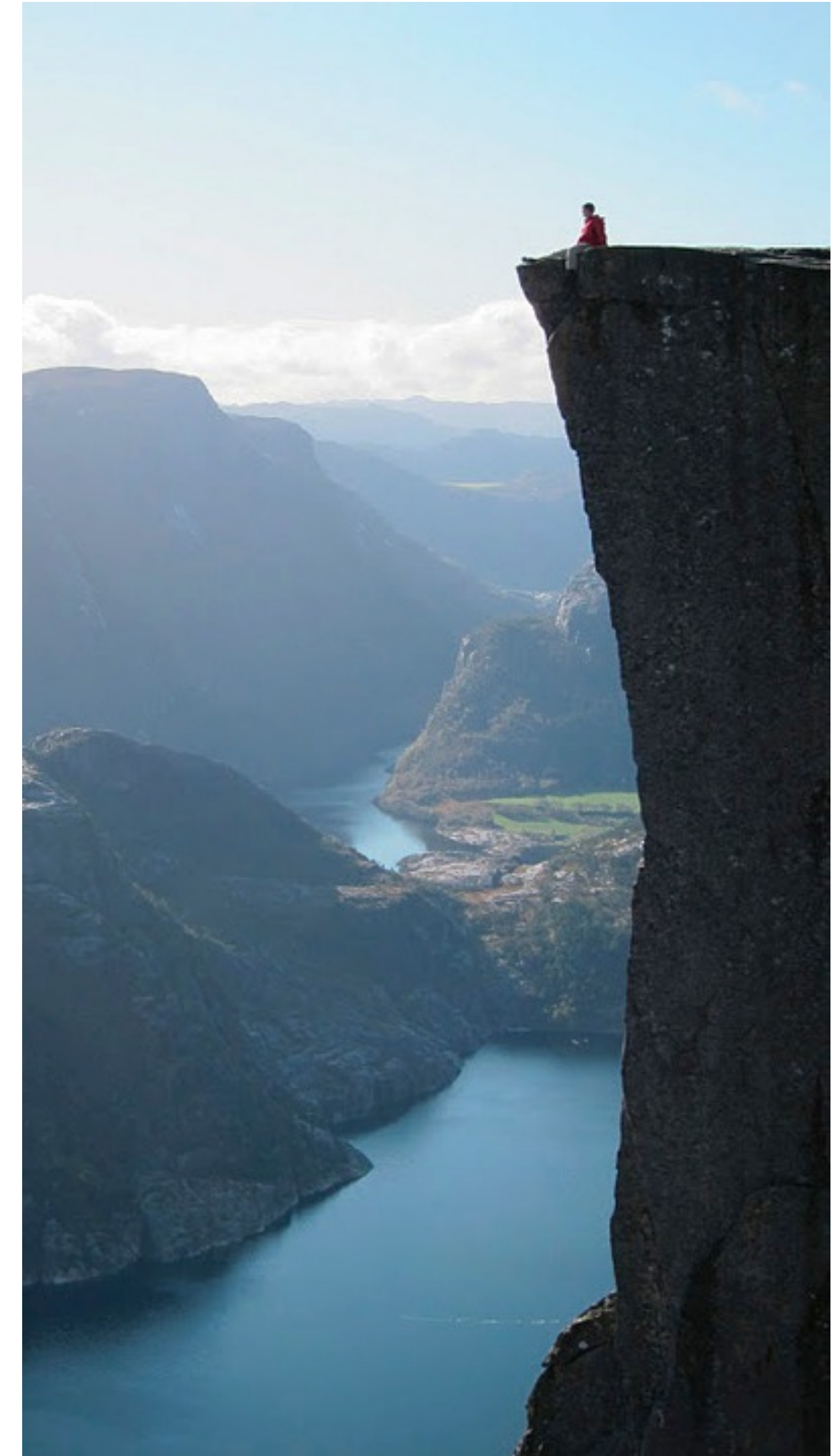
# Проверка рекурсивных функций

# Рекурсивный «прыжок веры»

```
def fact(n):  
    if n == 0:  
        return 1  
    else:  
        return n * fact(n-1)
```

Корректна ли функция **fact**?

1. Проверь простой случай.
2. Считай **fact** функциональной абстракцией!
3. Предположи, что **fact(n-1)** дает правильный результат.
4. Проверь, что результат **fact(n)** правильно задан в терминах **fact(n-1)**.



## Проверяем суммирование цифр

---

Функция `sum_digits` вычисляет сумму цифр положительного `n` правильно, поскольку:

Сумма цифр

(простой случай)

Предполагая, что

(абстракция)

для всех

(предположение)

`sum_digits(n)` будет равно

(вывод)

```
def sum_digits(n):  
    """Возвращает сумму цифр положительного целого n."""  
    if n < 10:  
        return n  
    else:  
        all_but_last, last = split(n)  
        return sum_digits(all_but_last) + last
```

# Взаимная рекурсия

# Алгоритм Луна

---

Используется для проверки номеров пластиковых карточек.

Из Википедии: [https://ru.wikipedia.org/wiki/Алгоритм\\_Луна](https://ru.wikipedia.org/wiki/Алгоритм_Луна)

- Цифры проверяемой последовательности нумеруются справа налево. Цифры, оказавшиеся на нечётных местах, остаются без изменений. Цифры, стоящие на чётных местах, умножаются на 2. Если в результате такого умножения возникает число больше 9 (например,  $7 * 2 = 14$ ), оно заменяется суммой цифр получившегося произведения – однозначным числом, т. е. цифрой (например,  $10: 1 + 0 = 1$ ,  $14: 1 + 4 = 5$ ).
- Все полученные в результате преобразования цифры складываются.

1	3	8	7	4	3
2	3	1+6=7	7	8	3

= 30

Если сумма кратна 10, то исходные данные верны.

(Пример)



# Рекурсия и итерация

## Перевод рекурсии в итерацию

---

Может быть непростым: итерация – частный случай рекурсии.

Идея: определить, какое состояние должно переходить от итерации к итерации.

```
def sum_digits(n):  
    """Возвращает сумму цифр положительного целого n."""  
    if n < 10:  
        return n  
    else:  
        all_but_last, last = split(n)  
        return sum_digits(all_but_last) + last
```

То, что осталось  
СЛОЖИТЬ

Частичная сумма

(Пример)

## Перевод итерации в рекурсию

---

Более формален: итерация – частный случай рекурсии.

Идея: состояние от итерации к итерации может передаваться через аргументы.

```
def sum_digits_iter(n):  
    digit_sum = 0  
    while n > 0:  
        n, last = split(n)  
        digit_sum = digit_sum + last  
    return digit_sum
```

Обновления через присвоения  
становятся...

```
def sum_digits_rec(n, digit_sum):  
    if n == 0:  
        return digit_sum  
    else:  
        n, last = split(n)  
        return sum_digits_rec(n, digit_sum + last)
```

...аргументами рекурсивного вызова