

# Рекурсия. Динамическое программирование. Строковые алгоритмы.

@pvavilin

26 марта 2023 г.

# Outline

# Что такое рекурсия?

Приём в программировании, когда задача может быть разделена на несколько таких же, но проще, задач.

```
def pow(x, n):  
    # возведение числа в степень это  
    # умножение числа на число  
    # в степени n-1  
    if n == 0:  
        return 1  
    return x * pow(x, n-1)
```

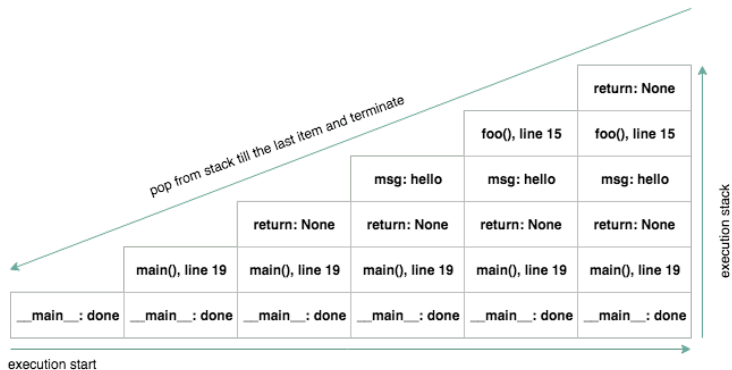
# Правильная рекурсия

```
def pow(x, n):  
    # хорошо бы проверить,  
    # что база достижима  
    assert n >= 0  
    # base case / база рекурсии  
    if n == 0:  
        return 1  
    # recursive case / шаг рекурсии  
    return x * pow(x, n-1)
```

## Что такое стек вызовов?

```
def foo(msg):  
    print '{} foo'.format(msg)  
  
def main():  
    msg = 'hello'  
    foo(msg)  
  
if __name__ == '__main__':  
    main()
```

# Что такое стек вызовов?



# Почему рекурсия это плохо

- стек вызовов растёт вместе с ростом глубины рекурсии
- можно попасть в бесконечную рекурсию и истратить всю память на стек вызовов

## Recursion depth

```
def inf_counter(x):  
    print(x)  
    return inf_counter(x+1)  
inf_counter(0)
```



# Глубина рекурсии

```
import sys

print(sys.getrecursionlimit())
sys.setrecursionlimit(
    sys.getrecursionlimit() + 234
)
print(sys.getrecursionlimit())
1000
1234
```

# Почему рекурсия это хорошо

Помогает описать решение задачи понятным языком

```
#  $n! = n * (n-1)$ 
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n-1)

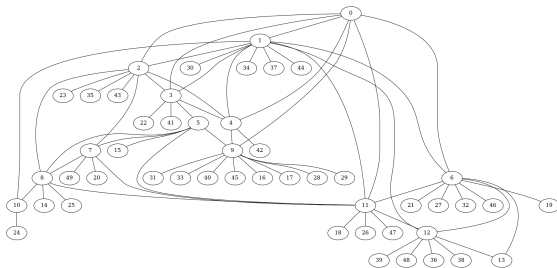
print(factorial(5))
120
```

## Варианты задач для рекурсии. Сортировка

```
def sort_alg(L: List[int]) -> List:
    if L:
        return (
            sort_alg(
                [e for e in L[1:] if e < L[0]]
            ) +
            L[0:1] +
            sort_alg(
                [e for e in L[1:] if e >= L[0]]
            )
        )
    return []
```

# Варианты задач для рекурсии. Графы

```
fname = "graph.png"
G = (
    nx.internet_as_graphs
    .random_internet_as_graph(
        n=50, seed=42
    )
)
```



## Варианты задач для рекурсии. Графы

```
def DFS(G, V, p=None):  
    if p is None:  
        p = []  
    p += [V]  
    for n in G[V]:  
        if n not in p:  
            p = DFS(G, n, p)  
    return p
```

## Вариант задачи для рекурсии

Попробуйте реализовать решение этой задачи без использования рекурсии 😊

# решение на LISP

## count-change.lisp

```
(defun count-change (amount)
  (cc amount 5))
(defun cc (amount kinds-of-coins)
  (cond ((= amount 0) 1)
        ((or (< amount 0) (= kinds-of-coins 0)) 0)
        (t (+ (cc amount
                    (- kinds-of-coins 1))
                (cc (- amount
                        (first-denomination kinds-of-coins))
                    kinds-of-coins)))))
(defun first-denomination (kinds-of-coins)
  (cond ((= kinds-of-coins 1) 1)
        ((= kinds-of-coins 2) 5)
        ((= kinds-of-coins 3) 10)
        ((= kinds-of-coins 4) 25)
        ((= kinds-of-coins 5) 50)))
(count-change 100)
```

## трассировка

```
def trace(f):  
    indent = 0  
    def g(*args, **kwargs):  
        nonlocal indent  
        print('|' * indent + '|--',  
              f.__name__, *args, **kwargs)  
        indent += 1  
        value = f(*args, **kwargs)  
        print('|' * indent + '|--',  
              'return', repr(value))  
        indent -= 1  
        return value  
    return g  
cc = trace(cc)  
count_change(10)
```



# Хвостовая рекурсия

Рекурсия, не требующая действий с возвращённым результатом из шага рекурсии.

```
def factorial(n, collected=1):  
    if n == 0:  
        return collected  
    return factorial(n-1, collected*n)  
  
print(factorial(5))  
120
```

# Оптимизация хвостовой рекурсии и почему её нет в Python

- Интерпретаторы/компиляторы могут оптимизировать хвостовую рекурсию (Tail Call Optimization) и не делать записей в стек вызовов, а подменять переменные в стеке вызовов, таким образом код получится равнозначным обычному циклу
- Почему TCO нет и не будет в Python

## Пример когда рекурсия помогает

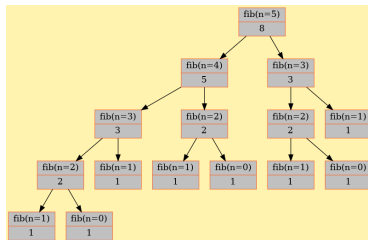
**Задача** У вас есть вложенная структура данных и вы хотите просуммировать значения поля X во всех объектах этой структуры.

**Решение задачи**

```
https://github.com/pimiento/  
recursion\_webinar/blob/  
master/recursion\_example.py
```

# Динамическое программирование

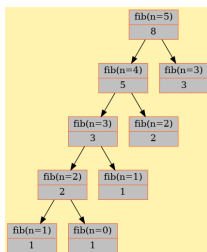
```
def fib(n):  
    if n == 0:  
        return 1  
    if n == 1:  
        return 1  
    return fib(n=n-1) + fib(n=n-2)
```



# Динамическое программирование. Кэширование

```
cache = {0: 1, 1: 1}
```

```
def fib(n):  
    if n not in cache:  
        cache[n] = \  
            fib(n=n-1) + fib(n=n-2)  
    return cache[n]
```



## Поиск приблизительно совпадающих строк

Возможные действия над строками, каждое действие будет иметь стоимость 1

*замена* заменить один символ в строку A1 на символ из строки A2. ("мама" → "рама")

*вставка* вставить один символ в строку A1 так чтобы она совпала с подстрокой A2. ("роза" → "проза")

*удаление* удалить один символ в строке A1 так чтобы она совпала с подстрокой A2. ("гроза" → "роза")

## Рекурсивное решение

```
def lev(a: str, b: str) -> int:
    if not a: return len(b)
    if not b: return len(a)
    return min([
        lev(a[1:], b[1:]) + (a[0] != b[0]),
        lev(a, b[1:]) + 1,
        lev(a[1:], b) + 1
    ])

print(lev("salt", "foobar"))
print(lev("halt", "salt"))
- 6
- 1
```

# Динамическое программирование в действии

```
def levenshtein(  
    a: str, b: str, m: List[List[int]]  
) -> int:  
    for i in range(1, len(a)):  
        for j in range(1, len(b)):  
            m[i][j] = min(  
                m[i-1][j-1] + (a[i] != b[j]),  
                m[i][j-1] + 1,  
                m[i-1][j] + 1  
            )  
    return m[len(a)-1][len(b)-1]
```



## Дополнительная литература

- Графы для самых маленьких: DFS
- SICP

# Вопросы-ответы

