Основи на Програмирането

Итерация и Рекурсия. Сравнение и примери. Функции от по-висок ред. Специални функции.

Какво ще научите

- ✓ Какво е рекурсията
- ✓ Свойства на рекурсията
- ✓ Сравнение на рекурсията с итерация
- ✓ Различни примери за рекурсия
- ✓ Функции от по-висок ред
- ✓ Анонимна функция
- ✓ Функциите map, filter, reduce
- ✓ Функцията sorted

Рекурсия - дефиниране

- Алгоритмично: решаване на проблеми чрез метода **„разделяй и владей”:**
  - решаването на проблема се разделя на два по-малки и прости за решаване проблеми
- Семантично: програмна техника където **функция се обръща към (извиква) себе си**
  - Целта не е да имаме безкрайно изпълнение
  - Трябва да има **1 или повече гранични случаи** които са лесни за решаване
  - Рекурсивното обръщение трябва да бъде с **други входни аргументи** с които се опростява проблема

Методи за итерация

- Вградени оператори и изрази (**while** и **for** цикли) за изразяване на **итеративни алгоритми**
- При изпълнението на повтарящите се команди (итерации, цикли) управление чрез **променливи на състоянията** които променят стойността си за всяка итерация (цикъл)

Намиране на n! с рекурсия

n! = n×(n-1)×(n-2)×…×1

n! = n×[(n-1)!] “опростяване с индукция”

0! = 1 “граничен случай”

```
0! = 1
n! = n×[(n-1)!]
```

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

Друг пример за рекурсия с n!

n! = n*(n-1)*(n-2)*(n-3)* … * 1

- Граничен случай: за кое n знаем резултата?

```
n = 1        - >        if n == 1:   # граничен случай
                            return 1
```

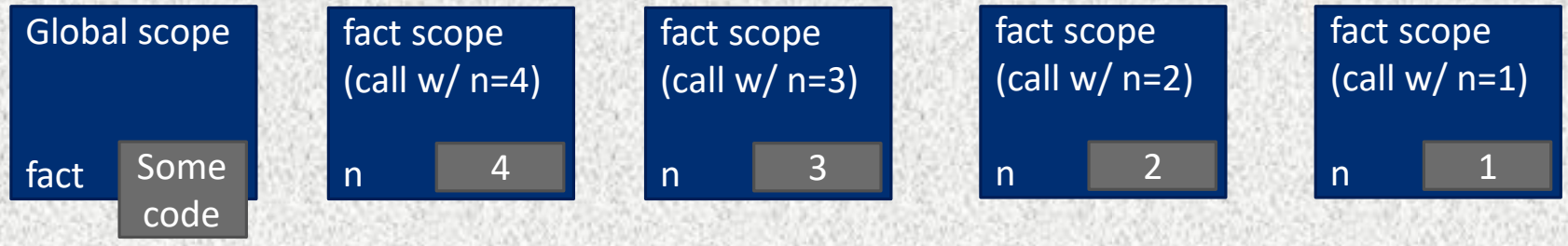
- Опростяване до случай с по-малки данни

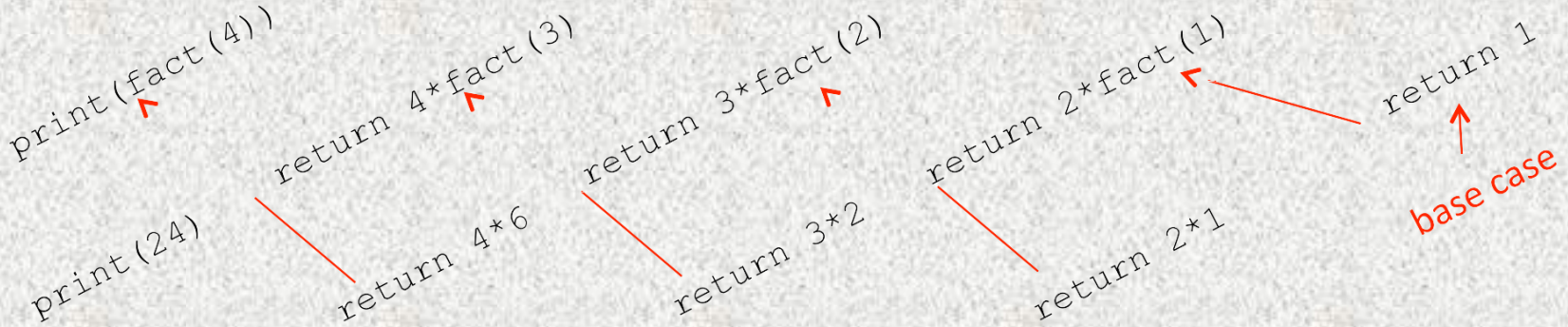
```
n*(n-1)!   - >       else:     # рекурсия
                          return n*factorial(n-1)
```

Пространство на имената при рекурсия

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

print(fact(4))
```





Факти за рекурсията

- Всяко рекурсивно обръщение към функция създава нейно **собствено пространство на имената (scope)**
- **Свързването на променливи със стойности** в дадено пространство не влияе на същите променливи в друго състояние
- Всяко рекурсивно обръщение завършва при първи срещнат return и управлението се връща в **предишния scope**

ИТЕРАЦИЯ сравнена с РЕКУРСИЯ

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

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

- Рекурсивното решение е по-кратко и ясно
- Рекурсивното решение е оптимално от гледна точка на програмиста
- Рекурсивното решение не е оптимално от гледна точка на компютърните ресурси

Още примери

```
def mysum(L):
    print(L) # Проследяване на рекурсивните нива
    if not L: # дали L вече е празен списък
        return 0
    else:
        return L[0] + mysum(L[1:])
>>> mysum([1, 2, 3, 4, 5])
[1, 2, 3, 4, 5]
[2, 3, 4, 5]
[3, 4, 5]
[4, 5]
[5]
[]
15
```

Още примери

```
def mysum(L):
    return 0 if not L else L[0] + mysum(L[1:])
# използване на if като израз

def mysum(L):
    return L[0] if len(L) == 1 else L[0] + mysum(L[1:])
# За всеки контейнер, с поне един елемент

def mysum(L):
    first, *rest = L
    return first if not rest else first + mysum(rest)
# използва разширено присвояване
```

Същата задача чрез итерация

```
>>> L = [1, 2, 3, 4, 5]
>>> sum = 0
>>> for x in L:
        sum += x
>>> sum
15
```

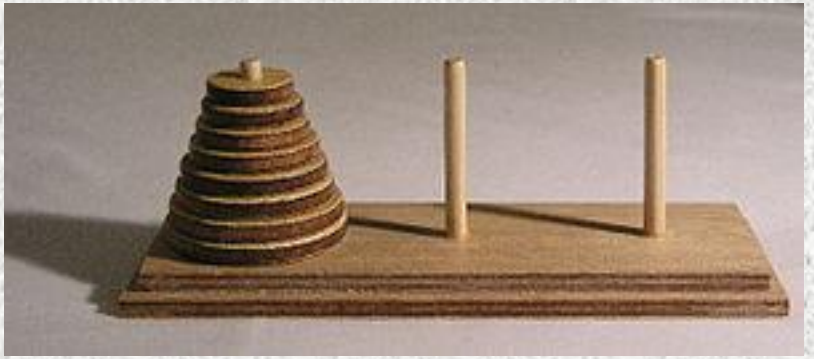
Този начин е много по-бърз и много по-икономичен (не се генерира ново пространство при всяка нова итерация от цикъла).

Задача, която не може с итерация

```
def sumtree(L):
    tot = 0
    for x in L: # за всеки елемент от това ниво
        if not isinstance(x, list):
            tot += x # числата се прибавят
        else:
            tot += sumtree(x) # рекурсивно към подсписък
    return tot

L = [1, [2, [3, 4], 5], 6, [7, 8]] # произволно влагане
print(sumtree(L)) # извежда 36
print(sumtree([1, [2, [3, [4, [5]]]]])) # извежда 15
```

Ханойски кули



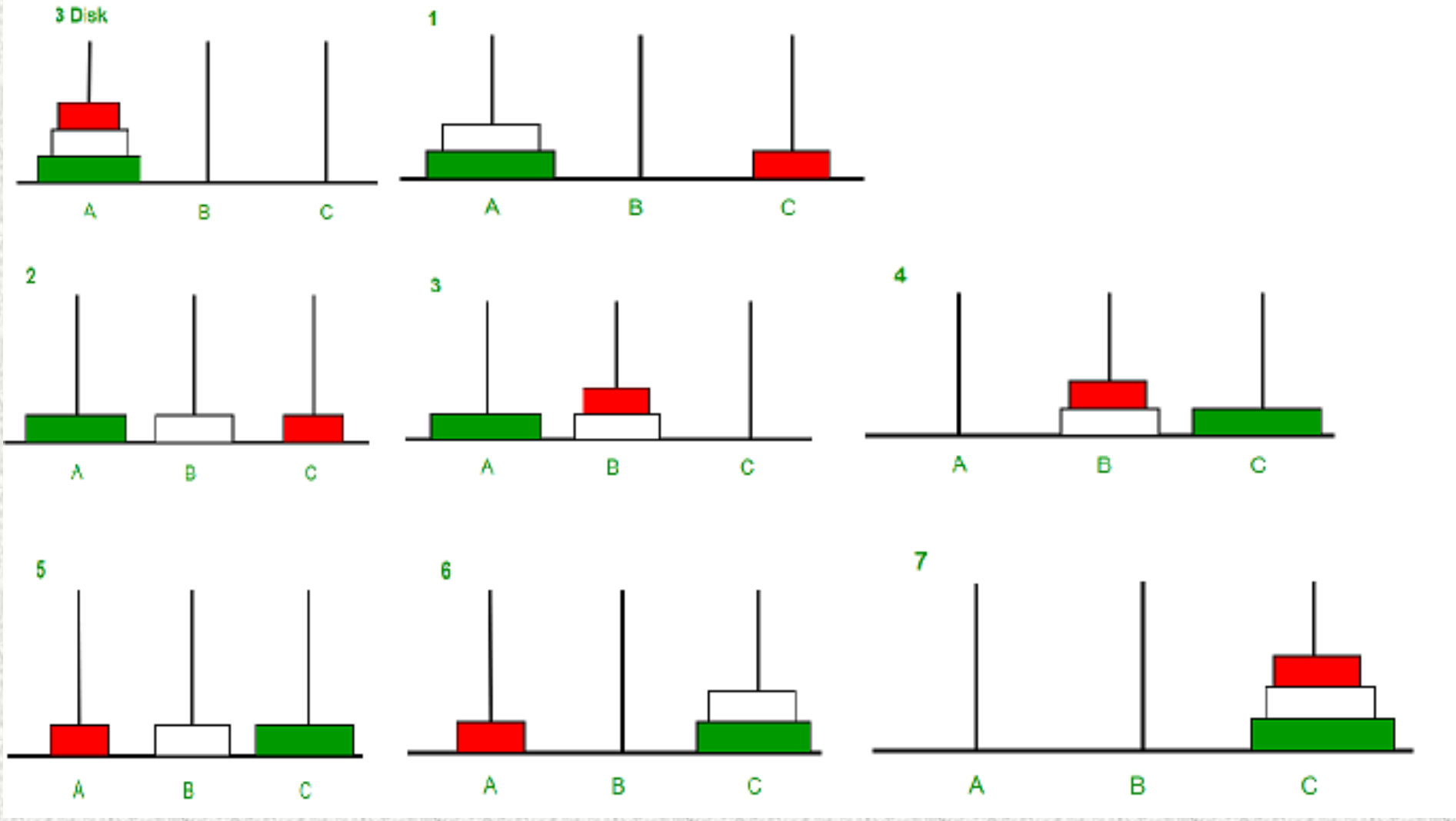
Задача:

- 3 високи колони
- Множество от 64 диска, всички с различни размери, наредени по-малък върху по-голям на първата колона
- Всички дискове трябва да се преместят върху третата колона и да са разположени в същия ред
- Може да се мести само един диск на ход, като всеки диск може да се премести само на празна колона или върху колона, където най-отгоре е по-голям от него диск

Задача за Ханойските кули

- Да се напише програма, която да извежда правилната поредица от ходове за преместването на дисковете
- Итеративното решение изглежда много сложно
- **Мисли рекурсивно!**
  - Намери прост случай (гранично решение)
  - Опитай се да намериш как сложното решение може да се разложи на по-прости
  - Опиши рекурсивното решение

Задача за Ханойските кули



Задача за Ханойските кули - рекурсия

- Гранично решение:
  имаме само един (най-малкия) диск на първия прът и всички останали дискове подредени правилно на втория прът.
  Ход: местим най-малкия диск от 1-я на 2-я прът
- Общ случай: свеждаме задачата за n диска до:
  - Местим n-1 диска от 1-я прът на 3-я прът
  - Местим един диск от 1-я прът на 2-я прът
  - Местим n-1 диска от 3-я прът на 2-я прът

Решение на задачата за Ханойските кули

```
def printMove(fr, to):
    print('move from ' + str(fr) + ' to ' + str(to))

def Towers(n, fr, to, spare):
    if n == 1:
        printMove(fr, to)
    else:
        Towers(n-1, fr, spare, to)
        Towers(1, fr, to, spare)
        Towers(n-1, spare, to, fr)
```

Задача за степени

Математика

$$\texttt{tower(3)} = 2^{2^2} = 2^4 = 16$$

$$\texttt{tower(4)} = 2^{2^{2^2}} = 2^{16}$$

$$\texttt{tower(5)} = 2^{2^{2^{2^2}}} = 2^{(2^{16})}$$

Индуктивна дефиниция:

Python - рекурсия

```
# рекурсивно степенуване
def tower(n):
```

Задача за степени

Математика

$$power(3) = 2^{2^2} = 2^4 = 16$$

$$power(4) = 2^{2^{2^2}} = 2^{16}$$

$$power(5) = 2^{2^{2^{2^2}}} = 2^{(2^{16})}$$

Индуктивна дефиниция:

power(1) = 2
power(n) = 2 ** power(n-1)

Python - рекурсия

```
# рекурсивно степенуване
def power(n):
  if n==1:
    return 2
  else:
    return 2 ** power(n-1)
```

Рекурсия с повече гранични условия

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

Пресмята n-тото число на Фибоначи

Рекурсия за нечислови обекти

Проблем: как да проверим дали низ от символи е **палиндром (palindrome)**, т.е. чете се по един и същи начин отпред назад и отзад напред.
Примери:

- “Able was I, ere I saw Elba” – Наполеон
- “Are we not drawn onward, we few, drawn onward to new era?” – поет и писател Anne Michaels

Рекурсивно решение

Преобразуваме низа до последователност от символи, премахвайки специалните знаци и преобразувайки всички символи в малки

Решение:

- Граничен случай: низ с 0 или 1 символа е палиндром
- Рекурсивен случай:
  - Ако първия и последен символ са еднакви, проверяваме дали средната част е палиндром

Пример

‘Able was I, ere I saw Elba’ à ‘ablewasiereisawleba’

isPalindrome(‘a b l e w a s i e r e i s a w l e b a’) е палиндром ако:

- ‘a’ == ‘a’ и

  isPalindrome(‘blewasiereisawleb’)

```
def isPalindrome(s):
    def toChars(s):
        s = s.lower()
        ans = ''
        for c in s:
            if c in 'abcdefghijklmnopqrstuvwxyz':
                ans = ans + c
        return ans

    def isPal(s):
        if len(s) <= 1:
            return True
        else:
            return s[0] == s[-1] and isPal(s[1:-1])

    return isPal(toChars(s))
```

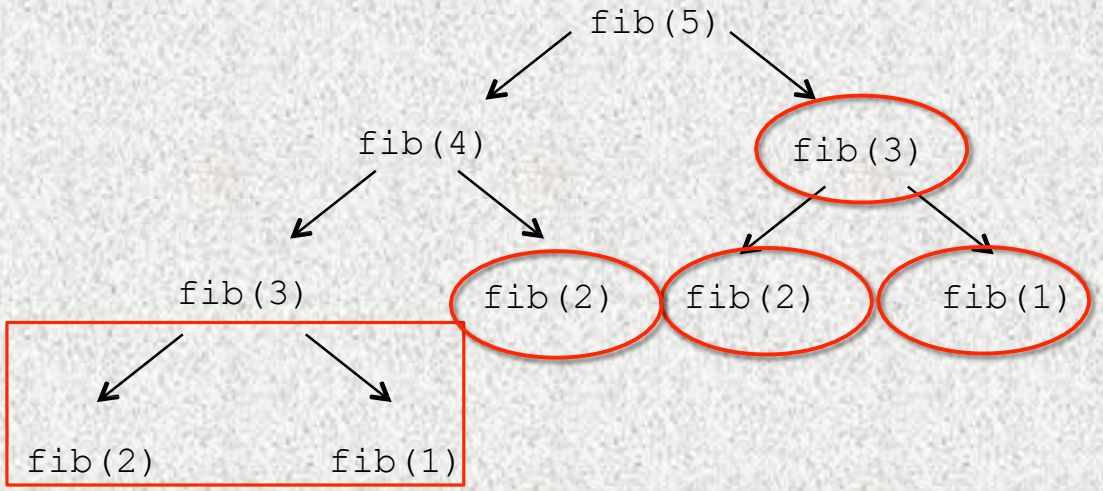
Примерът с Фибоначи

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

- два гранични случая
- две рекурсивни обръщения
- неефективна програма – много повторения

Неефективна рекурсия

```
fib(n) = fib(n-1) + fib(n-2)
```



- **пресмятане** на еднакви случаи многократно!
- Може да **пази** резултати от пресметнати случаи

Фибоначи с речници

```
def fib_efficient(n, d):
        if n in d:
            return d[n]
        else:
            ans = fib_efficient(n-1,d) + fib_efficient(n-2,d)
            d[n] = ans
        return ans
d = {1:1, 2:2} # т.е. задават се граничните случаи
print(fib_efficient(6, d))
```

- **Първо се търси** пресметната стойност в речника
- **Ако няма: добавя новата в речника** и продължава

Постигната ефективност

- При fib(34) имаме 11,405,773 рекурсивни обръщения
- При fib_efficient(34) имаме 65 рекурсивни обръщения
- Използването на речник за съхраняване на междинни резултати може да бъде много ефективно
- Това работи само за функции без страничен ефект (функции, които дават един и същ резултат за дадени аргументи, независимо от обръщенията)

Функциите в Python: като функции от по-висок ред

- Могат да се предават като аргументи на други функции
- Могат да се връщат като резултат от изпълнението на други функции.
- Да има поддръжка и за анонимни функции

Пример за функция от по-висок ред

```
def func(x):
    if x == 0:
        return 0
    else:
        return func(func(x-1))+1
```

Анонимни функции

- Наподобява едноименния механизъм в езика Лисп, наречен lambda функции
- Задават се с израз наподобяващ командата def, но не се дава име на функцията, а само се връща като резултат от израза (затова и се наричат анонимни – без име)
- Изразът се предхожда от ключовата дума lambda

```
>>> f = lambda x, y, z : x + y + z
>>> f(2, 3, 4)
9
```

Анонимни функции

С ключовата дума lambda се дефинира анонимна функция

```
def square(x): return x**2
```

Е еквивалентно на

```
square = lambda x: x**2
```

Освен че нямат име, lambda функциите в езика Python не могат да бъдат дефинирани с блок от команди, а само с един израз на един ред от програмата

Пример за анонимна функция

```
>>> def knights():
    title = 'Sir'
    action = (lambda x: title + ' ' + x)
# title в lambda е от тялото на външната дефиниция
    return action # връща обект функция
>>> act = knights()
>>> msg = act('robin')
# 'robin' се подава на анонимната функция
>>> msg
'Sir robin'
>>> act # act: обект тип функция
<function knights.<locals>.<lambda> at 0x0000000029CA488>
```

Примери за анонимна функция

```
L = [lambda x: x ** 2, lambda x: x ** 3,
lambda x: x ** 4] # Списък с 3 анонимни функции
for f in L:
    print(f(2))         # Извежда 4, 8, 16
print(L[0](3))   # Извежда 9

>>> key = 'got'
>>> {'already': (lambda: 2 + 2), 'got': (lambda: 2*4),
'one': (lambda: 2 ** 6)}[key]()
8
```

Вложени анонимни функции

```
>>> action = (lambda x: (lambda y: x + y))
>>> act = action(99)
>>> act(3)
102

>>> ((lambda x: (lambda y: x + y))(99))(4)
103
```

Функции от по-висок ред

- map(function, container) -> обект итератор, който връща резултат от прилагането на function към поредния елемент от контейнера
- filter(function, container) -> обект итератор, който връща поредната стойност от контейнера, за която прилагането на function е истина
- reduce(function, sequence[, initial]) -> прилага function последователно и кумулативно към елементите от sequence, докато ги редуцира до една стойност

Функции от по-висок ред - примери

Прилагане върху итератор:

```
list(map(lambda x: x**2, range(1,5)))
-> [1, 4, 9, 16]
```

Избор за прилагане с предикат

```
list(filter(lambda x: x%2==0, range(10)))
-> [0, 2, 4, 6, 8]
```

Постепенно намаляване стойността до 1 елемент

```
reduce(lambda x,y: x+y, [7, 3, 12])
-> 22
```

Функция map

Функцията map замества цикъл for по следния начин:

```
for loc in it: # цикъл с оператор for
    func(loc)
```

Еквивалентно описание с функцията map, без да е необходима локална променлива loc:

```
map(func, it) # цикъл с функцията map()
```

Функция map

- Синтаксис: map(function, container)
- Действие – връща обект от тип итератор, който на всяка итерация връща стойност, която е резултат от прилагането на функцията към поредния елемент от контейнера
- Свойства:
  - ако резултатът се подаде на функцията list, се получава списък от всички стойности
  - Ако се подаде на функцията next, връща следващата стойност

Примери с map

```
>>> s={3, 5, 6, 2, 8, 23}
>>> map((lambda x : x**2 -4), s)
<map object at 0x00502BF0>
>>> list(map((lambda x : x**2 -4), s))
[0, 5, 21, 32, 60, 525]
>>> mm = map((lambda x : x**2 -4), s)
>>> mm
<map object at 0x00502BF0>
>>> next(mm)
0
```

Примери с map

```
>>> pow(3, 4) # 3**4
81
>>> list(map(pow, [1, 2, 3], [2, 3, 4])) # 1**2, 2**3,
3**4
[1, 8, 81]
>>> l1,l2,l3=[1,2,3],[3,4,5],[6,7,9]
>>> oo = list(map((lambda x,y,z : x+y+z), l1, l2, l3))
>>> oo
[10, 13, 17]
```

Примери с map 2

```
>>> res = []
>>> for x in 'spam':
    res.append(ord(x))
>>> res
[115, 112, 97, 109]
>>> res = list(map(ord, 'spam'))
>>> res
[115, 112, 97, 109]
>>> res = [ord(x) for x in 'spam']
>>> res
[115, 112, 97, 109]
```

Примери с map 3

```
>>> [x ** 2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> list(map((lambda x: x ** 2), range(10)))
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Комбиниране на map за сумиране

```
import math
def fsum(f):
    def apply(a, b):
        return sum(map(f, range(a,b+1)))
    return apply
simple_sum = fsum(int)
square_sum = fsum(lambda x: x ** 2)
sqrt_sum = fsum(math.sqrt)
print(simple_sum(1,10))  #  55
print(square_sum(1,10))  #  385
print(sqrt_sum(1,10))      # 22.4682781862041
```

Още няколко примера

```
>>> [line.rstrip() for line in open('myfile')]
['aaa', 'bbb', 'ccc']

>>> list(map((lambda line: line.rstrip()), open('myfile')))
['aaa', 'bbb', 'ccc']

>>> list_tuple = [('bob', 35, 'mgr'), ('sue', 40, 'dev')]

>>> [age for (name, age, job) in list_tuple]
[35, 40]

>>> list(map((lambda row: row[1]), list_tuple))
[35, 40]
```

Функция filter

- Синтаксис: filter(function, container)
- Действие – връща обект от тип итератор, който на всяка итерация връща поредната стойност от контейнера, за която резултатът от прилагането на функцията към този елемент е истина
- Свойства:
  - ако се подаде на функцията list, се получава списък от всички стойности
  - Ако се подаде на функцията next, връща следващата стойност

Примери за използване

```
>>> list(filter((lambda x: x > 0), range(−5, 5)))
[1, 2, 3, 4]
```

функцията е сходна с израза:

```
>>> [x for x in range(−5, 5) if x > 0]
[1, 2, 3, 4]
```

Примери за използване

```
>>> def f(x): return x % 2 != 0 and x % 3 != 0

...

>>> list(filter(f, range(2, 25)))

[5, 7, 11, 13, 17, 19, 23]
```

Комбиниране на map с filter

```
>>> [x ** 2 for x in range(10) if x % 2 == 0]

[0, 4, 16, 36, 64]

>>> list( map((lambda x: x**2),
filter((lambda x: x % 2 == 0), range(10))) )

[0, 4, 16, 36, 64]
```

Функция reduce

- Синтаксис: reduce(function, sequence[, initial])
- Действие –прилага функцията на два елемента последователно и кумулативно към елементите от редицата, докато ги редуцира до една стойност
- Ако третия аргумент е зададен, той се използва като първи в редицата
- from functools import reduce
- Пример: reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]) е еквивалентно на: ((((1+2)+3)+4)+5)

Примери за функция reduce

```
def myreduce(function, sequence):
    tally = sequence[0]
    for next in sequence[1:]:
        tally = function(tally, next)
    return tally
```

```
>>> myreduce((lambda x, y: x + y), [1, 2, 3, 4, 5])
15
```

```
>>> myreduce((lambda x, y: x * y), [1, 2, 3, 4, 5])
120
```

Примери за функция reduce

```
from functools import reduce
from operator import mult

def factorial(n):
  return reduce(mult,range(1,n),1)

fact10 = factorial(10)
```

Примери за функция reduce

```
from functools import reduce

def factorial(n):
    return reduce(lambda res,next:res*next,range(1,n),1)

def factorial1(n):
    def mult(a, b): return a*b
    return reduce(mult, range(1, n), 1)
```

Предимства на функциите от висок ред

- Дават възможност за съкратено и опростено записване на сложни цикли
- Могат да се комбинират заедно
- Могат да се прилагат за по-съкратено и ефективно описание на различни сложни изчислителни задачи
- Могат лесно да се използват и прилагат (продукт MapReduce на Google)

Предимства на функциите от висок ред

```
def  is_prime(n):
    return len(filter(lambda k: n%k==0, range(2,n)))  ==  0

def  primes(m):
    return filter(is_prime,  range(1,m))
```

Функция sorted

sorted(*iterable*, key=*key*, reverse=*reverse*)

iterable: Задължителен. Контейнер за сортиране (списък, редица, речник и др.)

key: Незадължителен. Функция задаваща редът на подредба. По подразбиране е None

reverse: Незадължителен. Логически тип. False подрежда в нарастващ ред, True в намаляващ. По подразбиране е False

Примери за sorted

```
names = ['Yang', 'Robert', 'Tom', 'Gates']
names = sorted(names, key=len)
print(names)
['Tom', 'Yang', 'Gates', 'Robert’]

sorted({1: 'D', 2: 'B', 3: 'B', 4: 'E', 5: 'A'})
[1, 2, 3, 4, 5]

sorted("This is a test string".split(), key=str.lower)
['a', 'is', 'string', 'test', 'This']
```

Примери за sorted

```
student = [
    ('john', 'A', 15),
    ('jane', 'B', 12),
    ('dave', 'B', 10),
]

sorted(student, key=lambda st: st [2])   # по възраст

[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

Примери за list.sort

names.sort(key=len)

```
names = ['Yang', 'Robert', 'Tom', 'Gates']
names.sort(names, key=len)
print(names)
['Tom', 'Yang', 'Gates', 'Robert']
```

Заключение

- ✓ Какво е рекурсията
- ✓ Свойства на рекурсията
- ✓ Сравнение на рекурсията с итерация
- ✓ Различни примери за рекурсия
- ✓ Функции от по-висок ред
- ✓ Анонимна функция
- ✓ Функциите map, filter, reduce
- ✓ Функцията sorted