Sutaz

[Project Euler (https://projecteuler.net/)](https://projecteuler.net/)

- Vyriesit co najviac uloh funkcionalne
- Najlepsi dostanu **plny pocet bodov** z Python casti zaverecnej skusky

Nemenne objekty a funkcie vyssej urovne

Nemenné (Immutable) objekty

Nemenný objekt sa po vytvorení už nemôže meniť

In [5]:
```
x = 'foo'
print(id(x))
print(id(x.upper()))
print(id(x + 'bar'))
```

```
140314250459208
140314102913768
140314102913320
```

Neznamena to, ze referencia na objekt sa nemoze menit

v cisto funkcionalnom jazyku by sa nemalo diat ani to

In [6]:
```
x = 'foo'
y = x
print(x, id(x))
x = 'bar'
print(x, id(x))# objekt foo sa nezmenil, to len x uz smeruje na iny objekt
print(y, id(y))
```

```
foo 140314250459208
bar 140314250459152
foo 140314250459208
```

Nie je to to iste ako klucove slovo *final* v Jave

Final premenna po vytvoreni nemoze smerovat na iny objekt

Objekt samtny ale moze byt zmeneny

In [ ]:
```
# -- JAVA --
final List<Integer> list = new ArrayList<Integer>();
list = new ArrayList<Integer>(); // toto sa neskompiluje
```

In [ ]:
```
# -- JAVA --
final List<Integer> list = new ArrayList<Integer>();
list.add(1); //toto prejde bez problemov
```

In [ ]:
```
# -- JAVA --
final List<Integer> list = Collections.unmodifiableList(new ArrayList<Integer>(...)); //toto je immutable
list
```

Imutable znamena, ze hociaka operacia nad objektom vytvori novy objekt

In [14]:
```
x = 'foo'
```

```
y = x
print(x) # foo
y += 'bar'
print(x) # foo
print(y)
```

```
foo
foo
foobar
```

In [15]:
```
x = [1, 2, 3]
y = x
print(x)
y += [3, 2, 1]
print(x)
```

```
[1, 2, 3]
[1, 2, 3, 3, 2, 1]
```

Pozor, v Pythone sa parametre funkcie predavaju referenciou

Pri mutable objektoch to moze sposobit necakane veci ak neviete, co sa vo funkcii deje

In [16]:
```
def func(val):
    val += 'bar'

x = 'foo'
print(x)
func(x)
print(x)
```

```
foo
foo
```

In [17]:
```
def func(val):
    val += [3, 2, 1]

x = [1, 2, 3]
print(x)
func(x)
print(x)
```

```
[1, 2, 3]
[1, 2, 3, 3, 2, 1]
```

Ak predate immutable objekt funkcii, tak vam ho funkcia urcite nezmeni

String je imutable

Podobne ako vsetky zakladne typy

In [18]:
```
a = 'text'
print(a)
print('Adresa je: {}'.format(id(a)))
```

```
text
Adresa je: 140314297470624
```

In [19]:
```
# Znamena to, ze neviem menit hodnotu
a[0] = 'T'
print(a)
print('Adresa je: {}'.format(id(a)))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-309bd94f7dc8> in <module>()
      1 # Znamena to, ze neviem menit hodnotu
----> 2 a[0] = 'T'
      3 print(a)
      4 print('Adresa je: {}'.format(id(a)))

TypeError: 'str' object does not support item assignment
```

List je mutable

```
In [20]: a = [1,2,3,4,5]
         print(a)
         print('Adresa je: {}'.format(id(a)))

         [1, 2, 3, 4, 5]
         Adresa je: 140314102958728
```

```
In [21]: # Znamena to, ze neviem menit hodnotu
         a[0] = 'T'
         print(a)
         print('Adresa je: {}'.format(id(a)))

         ['T', 2, 3, 4, 5]
         Adresa je: 140314102958728
```

Tuple je immutable

```
In [22]: t1 = (1, 2, 3, 4, 5)
         t1

Out[22]: (1, 2, 3, 4, 5)
```

```
In [23]: t1[1]

Out[23]: 2
```

```
In [24]: t1[1]=3

         ---------------------------------------------------------------------------
         TypeError                                 Traceback (most recent call last)
         <ipython-input-24-ab9dff0930da> in <module>()
         ----> 1 t1[1]=3

         TypeError: 'tuple' object does not support item assignment
```

Nemennost moze komplikovat pracu s objektami

```
In [25]: t1 = (1, 2, 3, 4, 5)
         # Ked chceme update, treba vyrobit novy objekt
         t2 = t1[:2] + (17, ) + t1[3:]
         t2

Out[25]: (1, 2, 17, 4, 5)
```

```
In [26]: # alebo
         l1 = list(t1)
         l1[2] = 17
         t2 = tuple(l1)
         t2

Out[26]: (1, 2, 17, 4, 5)
```

```
In [27]: # vs.
         a = [1,2,3,4,5]
         a[2] = 17
         a

Out[27]: [1, 2, 17, 4, 5]
```

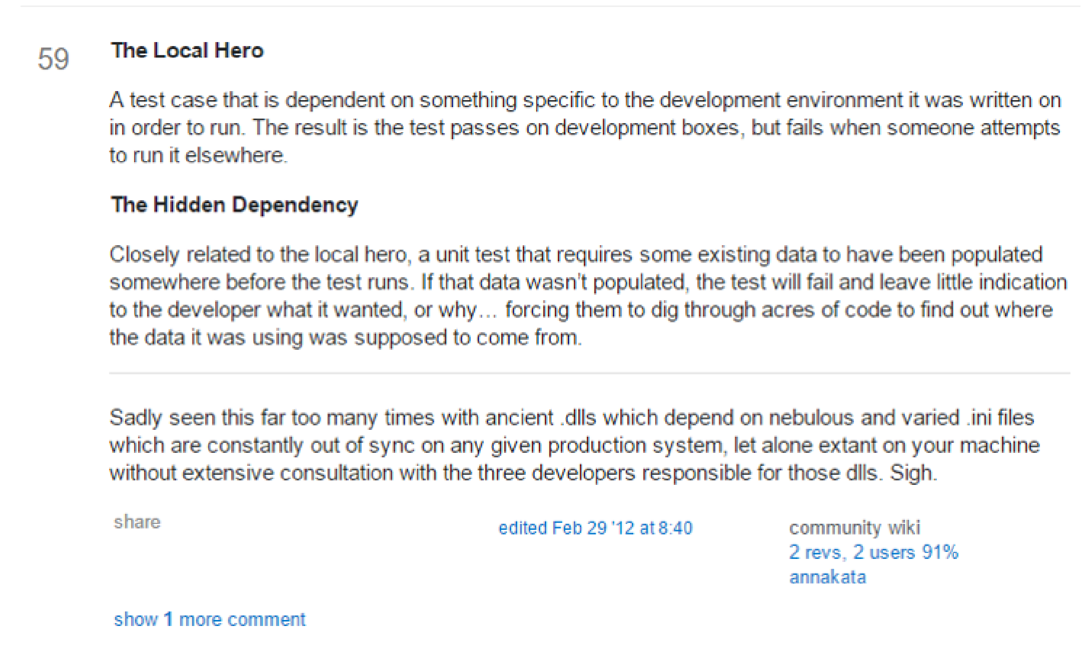
Preco je nemennost dobra

Netreba pocitat s tym, ze sa vam moze objekt zmenit

- Je to bezpecnejsie.
- vznika menej chyb
- Lahsie sa debuguje

Lahsie sa testuje

- staci test na jednu funkciu a nie celu skupinu objektov



59

The Local Hero

A test case that is dependent on something specific to the development environment it was written on in order to run. The result is the test passes on development boxes, but fails when someone attempts to run it elsewhere.

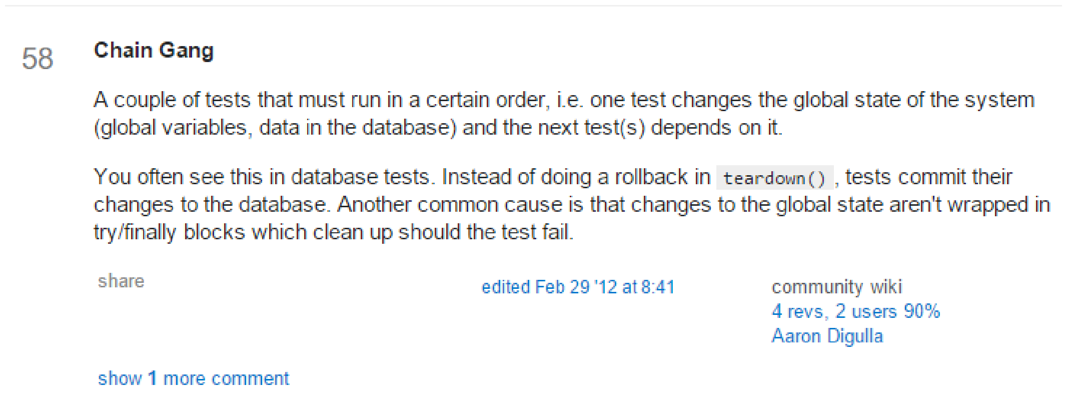
The Hidden Dependency

Closely related to the local hero, a unit test that requires some existing data to have been populated somewhere before the test runs. If that data wasn't populated, the test will fail and leave little indication to the developer what it wanted, or why… forcing them to dig through acres of code to find out where the data it was using was supposed to come from.

Sadly seen this far too many times with ancient .dlls which depend on nebulous and varied .ini files which are constantly out of sync on any given production system, let alone extant on your machine without extensive consultation with the three developers responsible for those dlls. Sigh.

share | edited Feb 29 '12 at 8:40 | community wiki
2 revs, 2 users 91%
annakata

show **1** more comment



58

Chain Gang

A couple of tests that must run in a certain order, i.e. one test changes the global state of the system (global variables, data in the database) and the next test(s) depends on it.

You often see this in database tests. Instead of doing a rollback in `teardown()`, tests commit their changes to the database. Another common cause is that changes to the global state aren't wrapped in try/finally blocks which clean up should the test fail.

share | edited Feb 29 '12 at 8:41 | community wiki
4 revs, 2 users 90%
Aaron Digulla

show **1** more comment

Toto je dovod, preco ma Test Driven Development (TDD) taky usepch

- Testy sa píšu ešte pred kódom
- Zamýšľate sa ako napísať kód tak aby bol testovateľný
- Bez toho aby ste o tom vedeli odstraňujete vedľajšie efekty
- Nazite sa o to, aby na sebe funkcie co najmenej zavyseli
- Pripravovanie objektov je pre vas zbytocnou komplikaciou
- Zmena stavu objktu sposobuje, ze musite pisat velmi vela tetsov aby ste osetrili mnozstvo hranicnych stavov

Da sa lahsie zdielat medzi vlaknami a procesmi

- netreba synchronizovat pristup k objektom

Da sa hashovat

- ak pouzijete objekt ako kluc, tak sa urcite nezmeni a mozete
- hashovacia funkcia nad nim vzdy vrati rovnaku hodnotu

Objekty mozu byt mensie. Zaberaju menej miesta v pamati a operacie nad nimi su rychlejsie.

Ale

- Je treba vytvarat velmi vela objektov.
- Garbage collector sa narobi.

In [28]:
```
# inspirovane https://www.youtube.com/watch?v=5qQQ3yzbKp8
employees = ['Jozo', 'Eva', 'Fero', 'Miro', 'Anna', 'Kristina']

output = '<ul>\n'

for employee in employees:
    output += '\t<li>{}</li>\n'.format(employee)
#     print('Adresa outputu je: {}'.format(id(output)))

output += '</ul>'

print(output)
```

```
<ul>
        <li>Jozo</li>
        <li>Eva</li>
        <li>Fero</li>
        <li>Miro</li>
        <li>Anna</li>
        <li>Kristina</li>
</ul>
```

Ako zabezpecit nemennost objektov?

- konvencia
- vynutit si ju

S vela vecami si mozeme pomoct kniznicou Pyrsistent

In [30]:
```
import pyrsistent as ps
```

List / Vektor

In [31]:
```
v1 = ps.pvector([1, 2, 3, 4])
v1 == ps.v(1, 2, 3, 4)
```

Out[31]:
```
True
```

In [32]:
```
v1[1]
```

Out[32]:
```
2
```

In [33]:
```
v1[1:3]
```

Out[33]:
```
pvector([2, 3])
```

In [34]:
```
v1[1] = 3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-34-00087ab71557> in <module>()
----> 1 v1[1] = 3

TypeError: 'pvectorc.PVector' object does not support item assignment
```

In [35]:
```
v3 = v1.set(1, 5)
print(v3)
print(v1)
```

```
pvector([1, 5, 3, 4])
pvector([1, 2, 3, 4])
```

Map / dict

```
In [36]: m1 = ps.pmap({'a':1, 'b':2})
         m1 == ps.m(a=1, b=2)

Out[36]: True
```

```
In [37]: m1['a']

Out[37]: 1
```

```
In [38]: m1.b # toto s dict nejde

Out[38]: 2
```

```
In [39]: print(m1.set('a', 3))
         print(m1)

         pmap({'b': 2, 'a': 3})
         pmap({'b': 2, 'a': 1})
```

```
In [40]: print(id(m1), id(m1.set('a', 3)))

         140314100233632 140314100234424
```

Transformacia mutable <=> immutable

```
In [41]: ps.freeze([1, {'a': 3}])

Out[41]: pvector([1, pmap({'a': 3})])
```

```
In [42]: ps.thaw(ps.v(1, ps.m(a=3)))

Out[42]: [1, {'a': 3}]
```

... a dalsie immutable struktury

https://github.com/tobgu/pyrsistent (https://github.com/tobgu/pyrsistent)

- PVector, similar to a python list
- PMap, similar to dict
- PSet, similar to set
- PRecord, a PMap on steroids with fixed fields, optional type and invariant checking and much more
- PClass, a Python class fixed fields, optional type and invariant checking and much more
- Checked collections, PVector, PMap and PSet with optional type and invariance checks and more
- PBag, similar to collections.Counter
- PList, a classic singly linked list
- PDeque, similar to collections.deque
- Immutable object type (immutable) built on the named tuple
- freeze and thaw functions to convert between pythons standard collections and pyrsistent collections.
- Flexible transformations of arbitrarily complex structures built from PMaps and PVectors.

Da sa nieco spravit s tou spotrebou pamati?

Po niektorych operaciach sa objekty dost podobaju

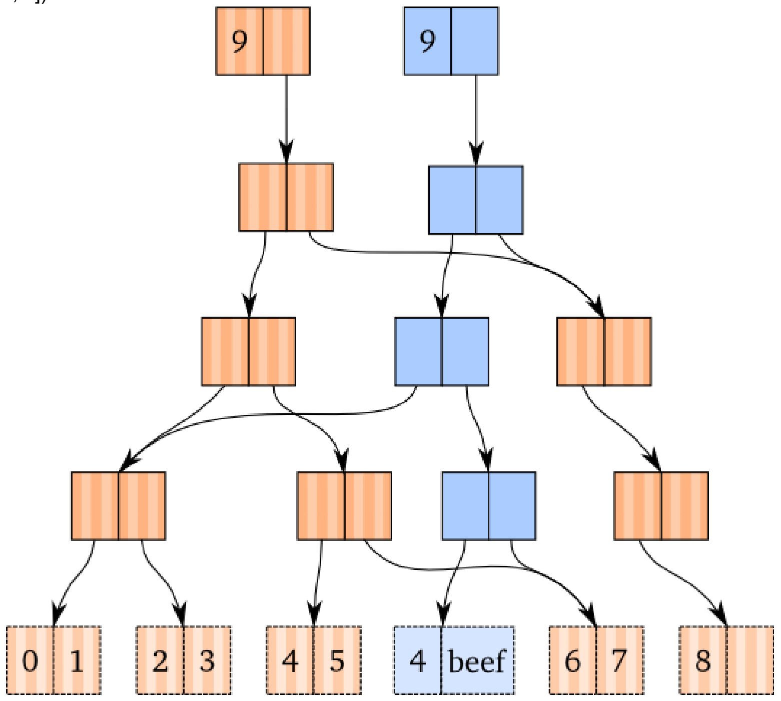
```
In [43]: v1 = ps.v(0, 1, 2, 3, 4, 5, 6, 7, 8)
         print(v1)
         v2 = v1.set(5, 'beef')
         print(v2)

         pvector([0, 1, 2, 3, 4, 5, 6, 7, 8])
         pvector([0, 1, 2, 3, 4, 'beef', 6, 7, 8])
```

Zdielanie casti datovej struktury

pvector([0, 1, 2, 3, 4, 5, 6, 7, 8])

pvector([0, 1, 2, 3, 4, 'beef', 6, 7, 8])



http://hypirion.com/musings/understanding-persistent-vector-pt-1 (http://hypirion.com/musings/understanding-persistent-vector-pt-1)

Higher order functions

Funkcional v LISPe je funkcia, ktora ma ako argument funkciu alebo funkciu vracia

- FUNCALL - vykonanie funkcie s argumentami
- MAPCAR - zobrazenie
- REMOVE-IF/REMOVE-IF-NOT - filter
- REDUCE - redukcia
- ...

V Pythone a inych jazykoch

- **Funkcia vyssej urovne** (Higher order function) - je funkcia, ktora dostava funkciu ako parameter
- **Generator** - je funkcia, ktora vracia funkciu

Funkcie vyssej urovne sa daju velmi dobre pouzit na spracovanie zoznamu

Najcastejsie operacie so zoznamom:

- zobrazenie
- filter
- redukcia

Zobrazenie

Aplikovanie funkcie na vsetky prvky zoznamu a vytvorenie noveho zoznamu z transformovanych prvkov

In [44]:
```
def process_item(x):
    return x*x
item_list = [1,2,3,4,5,6]
```

In [45]:
```
# impertivny zapis
```

```
collection = []
for item in item_list:
    partial_result = process_item(item)
    collection.append(partial_result)
collection
```

```
Out[45]: [1, 4, 9, 16, 25, 36]
```

```
In [46]: #  C-like zapis
collection = []
index = 0
while index < len(item_list):
    partial_result = process_item(item_list[index])
    collection.append(partial_result)
    index += 1
collection
```

```
Out[46]: [1, 4, 9, 16, 25, 36]
```

Zobrazenie pomocou funkcie vyssej urovne je prehladnejsie

```
In [47]: def process_item(x):
    return x*x
item_list = [1,2,3,4,5,6]
```

```
In [48]: # funkcionalny zapis
collection = map(process_item, item_list)
collection
```

```
Out[48]: <map at 0x7f9d6c1aa320>
```

Dalsi priklad pouzitia funkcie map

```
In [49]: def fahrenheit(T):
    return ((float(9)/5)*T + 32)

def celsius(T):
    return (float(5)/9)*(T-32)

temperatures = (36.5, 37, 37.5, 38, 39)
F = list(map(fahrenheit, temperatures))
C = list(map(celsius, F))
print(F)
print(C)
```

```
[97.7, 98.60000000000001, 99.5, 100.4, 102.2]
[36.5, 37.00000000000001, 37.5, 38.00000000000001, 39.0]
```

Alebo este iny

```
In [50]: list(map(len, open('data/morho.txt')))
```

```
Out[50]: [8, 1, 42, 39, 40, 39]
```

```
In [51]: list(map(print, open('data/morho.txt')))
```

```
Mor ho!



Zleteli orly z Tatry, tiahnu na podolia,

ponad vysoké hory, ponad rovné polia;

preleteli cez Dunaj, cez tú šíru vodu,

sadli tam za pomedzím slovenského rodu.
Out[51]: [None, None, None, None, None, None]
```

Funkcia *map* odstranuje potrebu udrzovat si stav

- nepotrebujem ziadnu kolekciu, ktora je v nejakom case ciastocne naplnena
- nepotrebujem ziadny index, ktory sa inkrementuje
- nestaram sa o to, ako map funguje
    - iterativne, rekurziou, paralelne, distribuovane, pomocou indexu?
- nestaram sa o vnutornu strukturu kolekcie
    - staci aby sa cez nu dalo iterovat
    - o tomto si povieme viac nabuduce

Funkcia map by mohla byt implementovana napriklad takto

In [52]:
```python
def my_map(f, seq): # Takto by to mohlo byt v pythone 2 a nie 3. Tam map vracia iterator.
    result = []
    for x in seq:
        result.append(f(x))
    return result
```

Filter

Zo zoznamu sa vytvara novy zoznam s tymi prvkami, ktore splnaju podmienku

In [53]:
```python
item_list = [1,2,3,4,5,6]
def condition(x):
    return(x % 2 == 0)
```

In [54]:
```python
collection = []
for item in item_list:
    if condition(item):
        collection.append(item)
collection
```

Out[54]:
```
[2, 4, 6]
```

Filter pomocou funkcie vyssej urovne

In [55]:
```python
item_list = [1,2,3,4,5,6]
def condition(x):
    return(x % 2 == 0)
```

In [56]:
```python
collection = filter(condition, item_list)
list(collection)
```

Out[56]:
```
[2, 4, 6]
```

Dalsi priklad pouzitia funkcie *Filter*

In [57]:
```python
fibonacci = [0,1,1,2,3,5,8,13,21,34,55]
def is_even(x):
    return x % 2 == 0

list(filter(is_even, fibonacci))
```

Out[57]:
```
[0, 2, 8, 34]
```

Redukcia

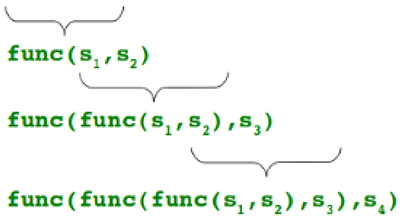
reduce(func, seq, init)

func(a, b)

Opakovane aplikuje funkciu na sekvenciu.

func prijma dva argumenty: hodnotu akumulatora a jeden prvok mnoziny

Atributom *func* moze byt prvok sekvencie alebo navratova hodnota inej *func*



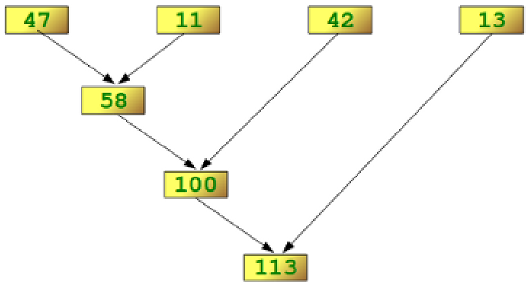
Typicky priklad je suma prvkov zoznamu

In [58]:
```
item_list = [47,11,42,13]
def add(a,b):
    return(a+b)
```

In [59]:
```
from functools import reduce

reduce(add, item_list)
```

Out[59]:
```
113
```



In [60]:
```
total = 0 # Takto by to bolo imperativne
for item in item_list:
    total = add(total, item)
total
```

Out[60]:
```
113
```

Dalsi priklad - nasobenie prvkov zoznamu

In [61]:
```
from functools import reduce
def mul(a,b):
    return a * b

reduce(mul, [1,2,3,4,5])
```

Out[61]:
```
120
```

Vela funkcii uz je predpripravenych

In [62]:
```
from operator import add
```

In [63]:
```
from operator import mul
```

Da sa spracovavat aj nieco ine ako cisla

In [64]:
```
from functools import reduce
from operator import add

print(reduce(add, open('data/morho.txt')))
```

```
Mor ho!

Zleteli orly z Tatry, tiahnu na podolia,
ponad vysoké hory, ponad rovné polia;
```

```
preleteli cez Dunaj, cez tú šíru vodu,
sadli tam za pomedzím slovenského rodu.
```

Da sa napriklad pracovat s mnozinami

In [65]:
```python
from operator import or_
reduce(or_, ({1}, {1, 2}, {1, 3}))  # union
```

Out[65]:
```
{1, 2, 3}
```

In [66]:
```python
from operator import and_
reduce(and_, ({1}, {1, 2}, {1, 3}))
```

Out[66]:
```
{1}
```

Lambda funkcia

anonymna funkcia

In [67]:
```python
my_sum = lambda x, y: x + y
my_sum(1,2)
```

Out[67]:
```
3
```

- obemdzenie na jediny riadok
- nepotrebuje return

Lambda je celkom prakticka ako parameter funkcie

In [68]:
```python
item_list = [1,2,3,4,5]
print(list(map(lambda x: x**2, item_list)))
```

```
[1, 4, 9, 16, 25]
```

In [69]:
```python
item_list = ["auto", "macka", "traktor"]
list(map(lambda x: x.upper(), item_list))
```

Out[69]:
```
['AUTO', 'MACKA', 'TRAKTOR']
```

Spracovanie zoznamu (list comprehension)

In [70]:
```python
print(list(map(lambda x: x**2, [1,2,3,4,5])))
print([x**2 for x in [1,2,3,4,5]])
```

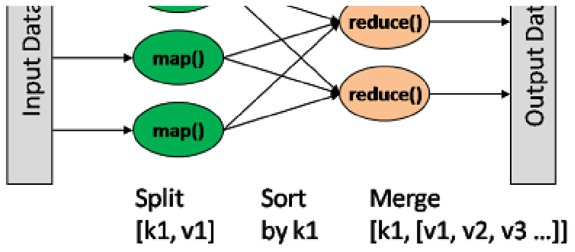
```
[1, 4, 9, 16, 25]
[1, 4, 9, 16, 25]
```

In [71]:
```python
print(list(filter(lambda x: x % 2 == 0, [1,2,3,4,5])))
print([x for x in [1,2,3,4,5] if x % 2 == 0])
```

```
[2, 4]
[2, 4]
```

Na co je to cele dobre - MapReduce

- je programovací model (framework) vyvinutý a patentovaný spoločnosťou Google, Inc. v roku 2004
- hlavným cieľom jeho vývoja bolo uľahčiť programátorom vytváranie paralelných aplikácií, ktoré spracovávajú veľké objemy dát
- zložité výpočty nad veľkým objemom dát musia byť vykonávané paralelne, niekedy až na stovkách alebo tisíckach počítačov súčasne
- pri takomto spracovaní sa treba okrem samotného výpočtu sústrediť napríklad aj na
  - rovnomerné rozdelenie záťaže všetkým dostupným počítačom
  - kontrolovanie výpadkov a porúch spolu s ich následným riešením
- MapReduce prináša ďalšiu vrstvu abstrakcie medzi výpočet, ktorý sa má realizovať paralelne a jeho vykonanie na konkrétnom hardvéri
- Keď napíšem program správne, tak sa nemusím starať na koľkých počítačoch bude bežať



GOTO priklad z netu

Celkom pekny priklad na jednoduchu MapReduce ulohu v Pythone.

Klasicky Word count priklad

http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/ (http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/)