Google Hash Code 2017

UniBG - 15 / 02 / 2017

Unibg Seclab

est. 2015

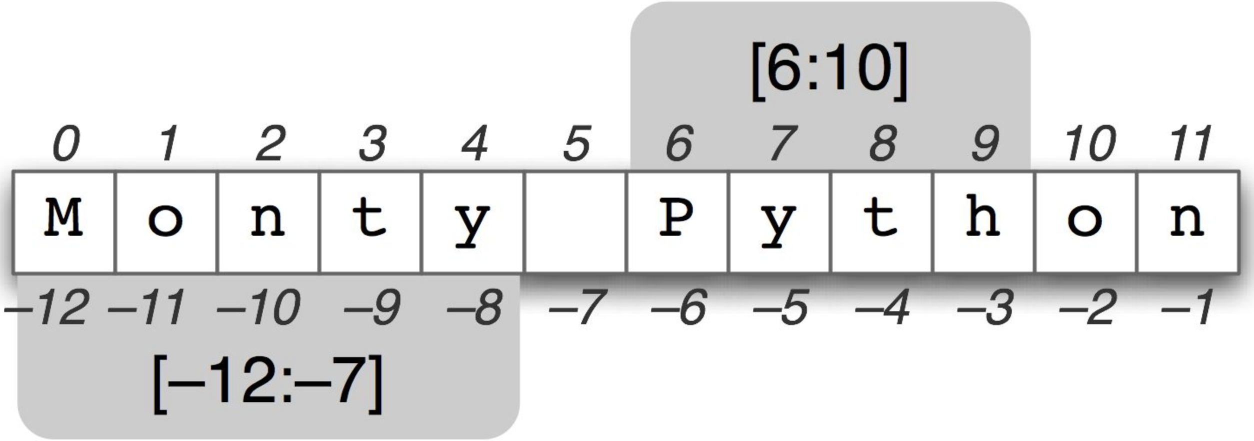
seclab.unibg.it

Python

- Dynamically typed and interpreted language
- **REPL** - Read Eval Print Loop
- Huge Standard Library
- Data structures:
  - Strings
  - Lists
  - Tuples
  - Dictionaries
  - Sets
- http://pythontutor.com/live.html
- https://learnxinyminutes.com/docs/python/
- https://learnxinyminutes.com/docs/python3/

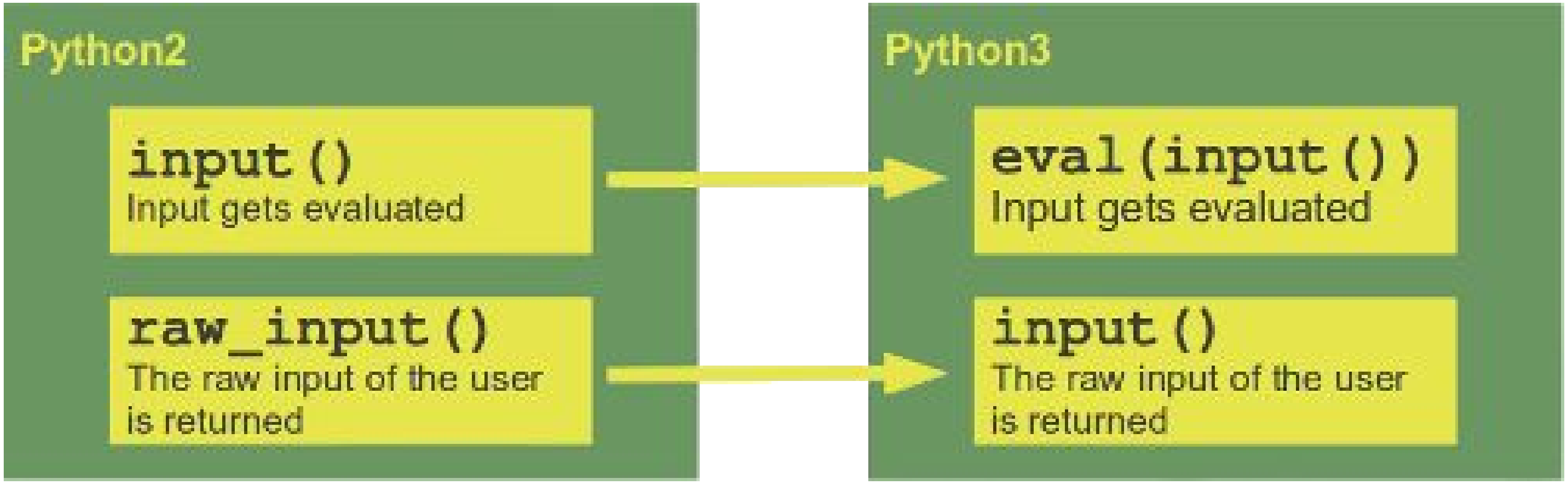
slicing

```
a[start:end]        # items start through end-1
a[start:]           # items start through the rest of the array
a[:end]             # items from the beginning through end-1
a[:]                # a copy of the whole array
a[start:end:step]   # start through not past end, by step
a[-1]               # last item in the array
a[-2:]              # last two items in the array
a[:-2]              # everything except the last two items
```



python 2 vs python 3

- Many small incompatibilities
- If you are new to python, you should use python 3



Python2

`input()`
Input gets evaluated

`raw_input()`
The raw input of the user is returned

Python3

`eval(input())`
Input gets evaluated

`input()`
The raw input of the user is returned

Dynamic Programming

- as seen last week, yet in Python -

The Fibonacci Sequence

1,1,2,3,5,8,13,21,34,55,89,144,233,377…

1+1=2

1+2=3

2+3=5

3+5=8

5+8=13

8+13=21

13+21=34

21+34=55

34+55=89

55+89=144

89+144=233

144+233=377

Problem:

Compute Fibonacci for
N OVER 9000!

static int fibonacci_1(int n)

```java
static int fibonacci_1(int n) {
    System.out.println("computing " + n);
    if (n <= 2)
        return 1;
    else
        return fibonacci_1(n - 1) + fibonacci_1(n - 2);
}
```

fibonacci_1(n)

```python
def fibonacci_1(n):
    print "Computing %d" % n
    if n <= 2:
        return 1
    else:
        return fibonacci_1(n - 1) + fibonacci_1(n - 2)
```

main

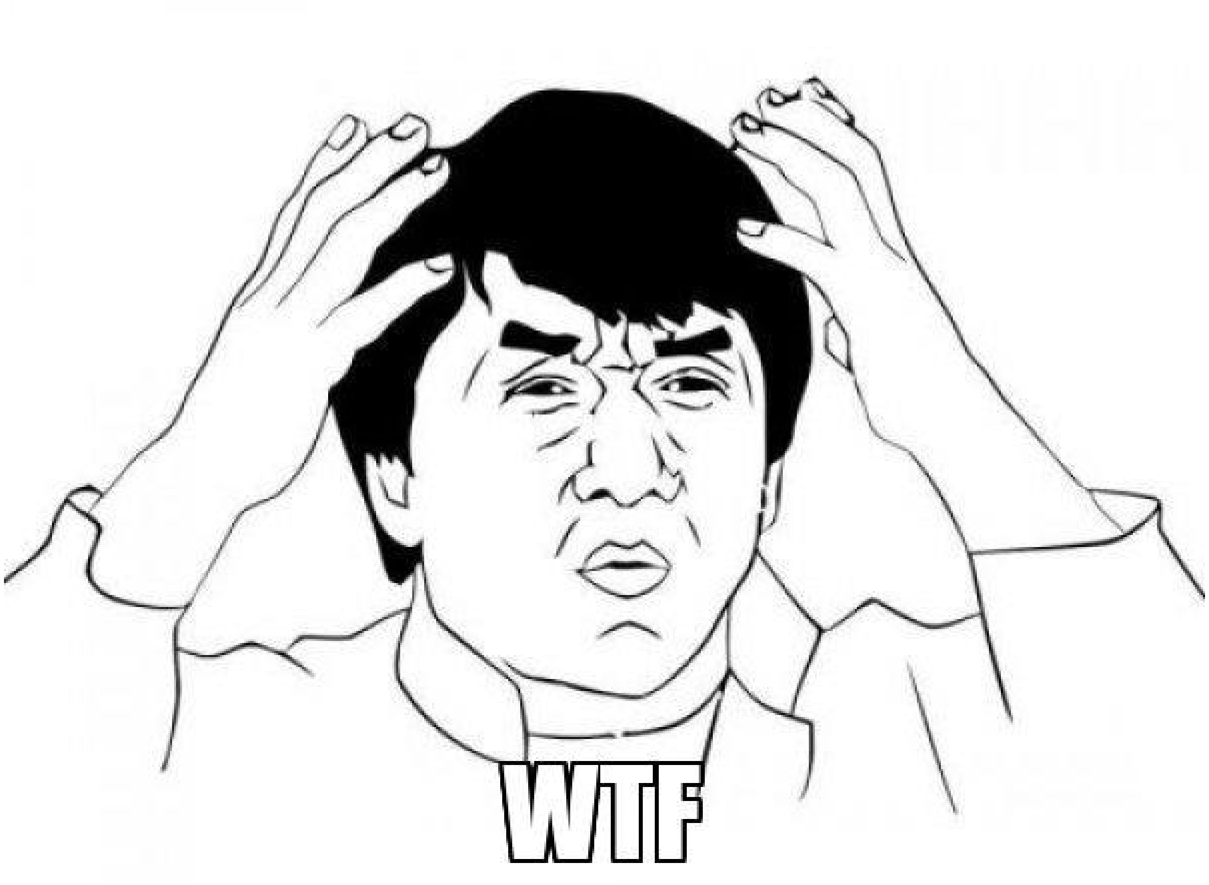
```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    System.out.println(fibonacci_1(n));
}
```

main

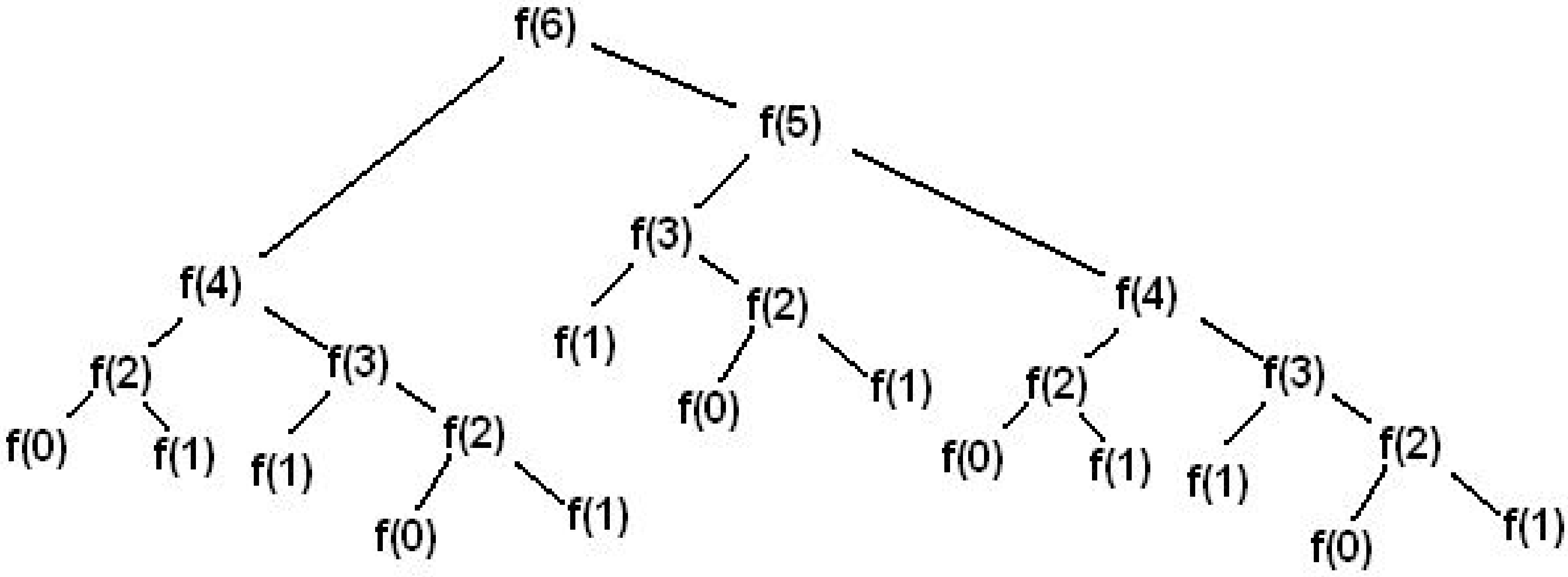
```python
def main():
    n = input("Insert number: ")
    print fibonacci_1(n)

if __name__ == '__main__':
    main()
```

Ok, let's try with 6

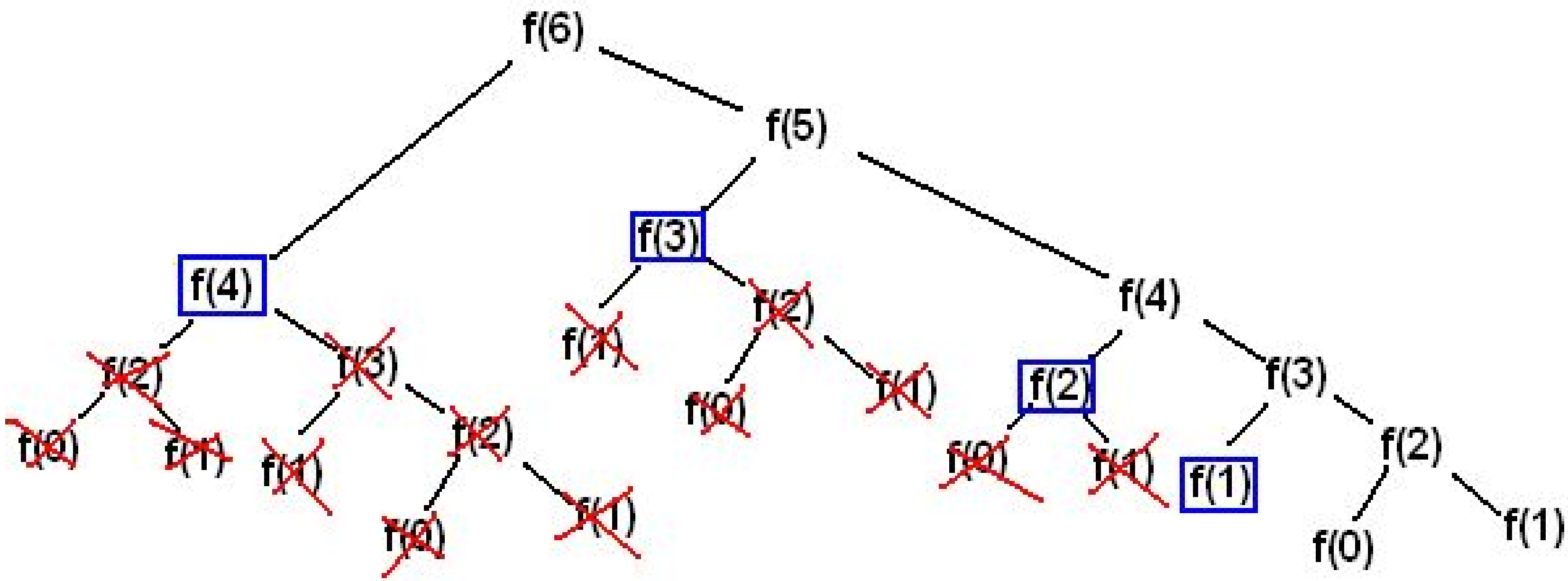
```
6
computing 6
computing 5
computing 4
computing 3
computing 2
computing 1
computing 2
computing 3
computing 2
computing 1
computing 4
computing 3
computing 2
computing 1
computing 2
8
```



Successione di Fibonacci - Stack delle chiamate



Successione di Fibonacci - Dynamic Programming



static int fibonacci_2(int n)

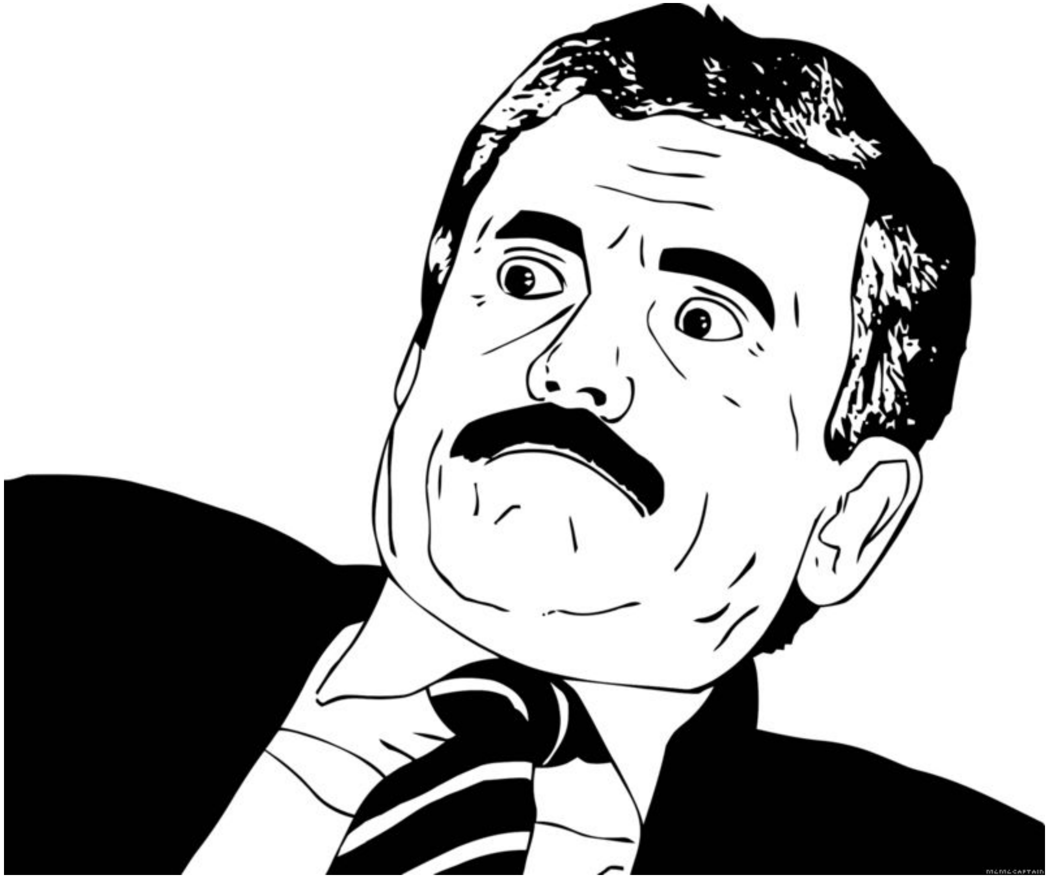
```java
static Map<Integer, Integer> cache_2 = new HashMap<Integer, Integer>();

static Integer fibonacci_2(int n) {
    if (cache_2.containsKey(n))
        return cache_2.get(n);

    System.out.println("computing " + n);
    int result;

    if (n <= 2)
        result = 1;
    else
        result = fibonacci_2(n - 1) + fibonacci_2(n - 2);

    cache_2.put(n, result);
    return result;
}
```

fibonacci_2(n)

```python
cache2 = {}

def fibonacci_2(n):
   if cache2.has_key(n):
       return cache2[n]

   print "Computing %d" % n
   if n <= 2:
       result = 1
   else:
       result = fibonacci_2(n - 1) + fibonacci_2(n - 2)

   cache2[n] = result
   return result
```

What about 100 now?

```
100
computing 100
computing 99
computing 98
computing 97
computing 96
computing 95
…
computing 5
computing 4
computing 3
computing 2
computing 1
-980107325
```



What about 100 now?

```
100
computing 100
computing 99
computing 98
computing 97
computing 96
computing 95
…
computing 5
computing 4
computing 3
computing 2
computing 1
354224848179261915075L
```

GO BIG!

```
1000

computing 1000
computing 999
computing 998
computing 997
computing 996
computing 995
…
computing 5
computing 4
computing 3
computing 2
computing 1
4346655768693745643568852767504062580256466051737178040248172908953655541794905189040387984007925516929592259308032263477520968962323987332247116164299644090653318793829896964992851600370447613779516684922
8875L
```

GO BIIIIIIIIIIIIIIIIIIG!

```
10000
computing 10000
computing 9999
computing 9998
computing 9997
computing 9996
computing 9995
…
computing 4471

Exception in thread "main"
   java.lang.StackOverflowError
   …
   at com.company.Main.fibonacci_3
```

GO BIIIIIIIIIIIIIIIIIIG!

```
10000
computing 10000
computing 9999
computing 9998
computing 9997
computing 9996
computing 9995
…
computing 9002
```

```
File “...”, line xx, in fibonacci_2
   …
RuntimeError: maximum recursion depth exceeded
```

fibonacci_2(n)

```python
import sys
sys.setrecursionlimit(10000)

...

cache2 = {}

def fibonacci_2(n):
    ...
```

GO BIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIGGG!!111

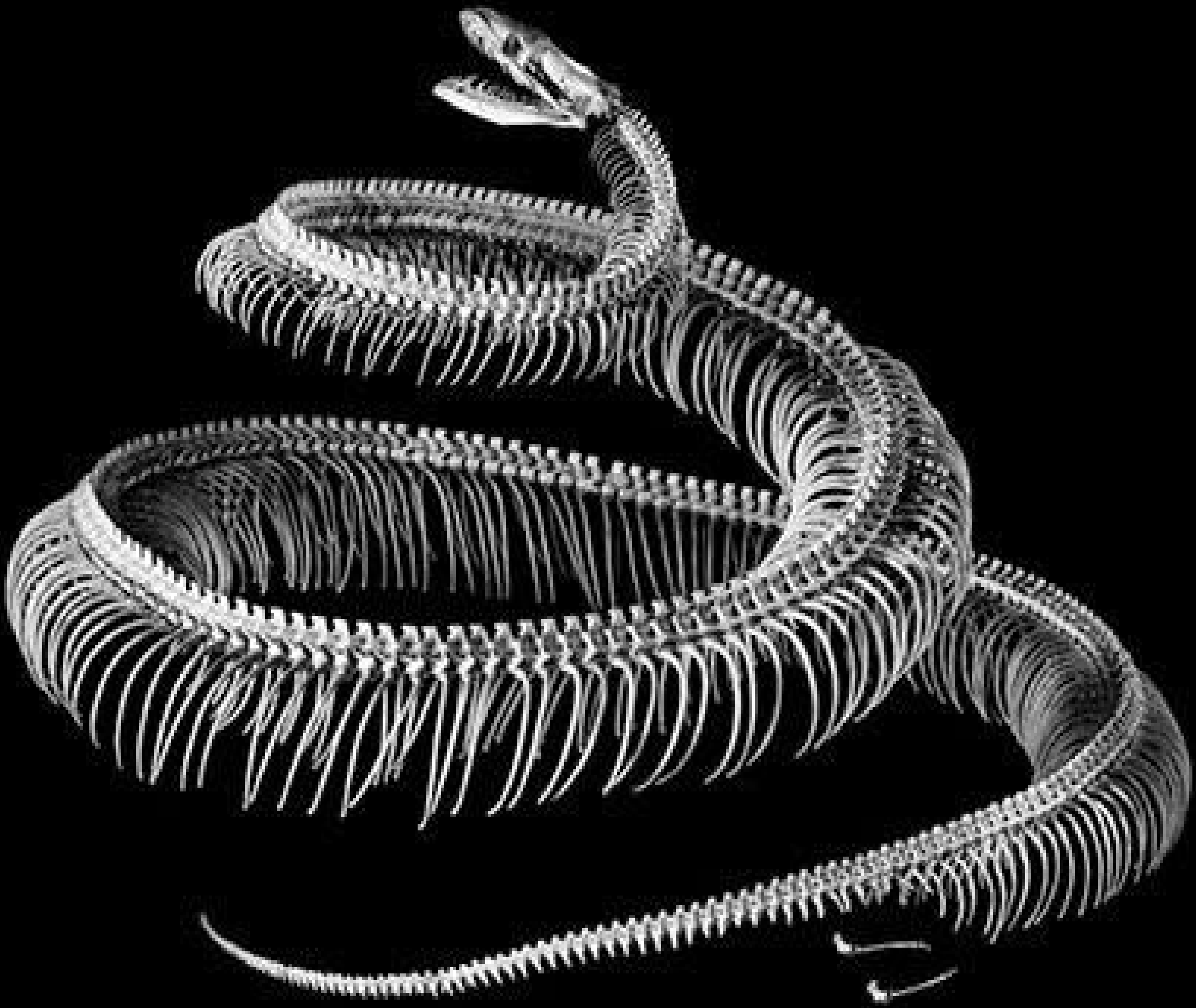
```
3364476487643178326662161200510754331030214846068006390656476997468008144216666236815559551
3633734025582065332680836159373734790483865268263040892463056431887354544369559827491606602
0998841839338646527313000888302692356736131351175792974378544137521305205043477016022647583
1890652789085515436615958298727968298751063120057542878345321551510387081829896979161312785
6265033195487140214287532698187962046936097879900350962302291026368131493195275630227837628
4415403605844025721143349611800230912082870460889239623288354615057765832712525460935911282
0392528539343462090424524892940390170623388899108584106518317336043747073790855263176432573
3993712871937587746897479926305837065742830161637408969178426378624212835258112820516370298
0893320999057079200643674262023897831114700540749984592503606335609338838319233867830561364
3535189213327973290813373264265263398976392272340788292817795358057099369104917547080893184
1056146322338217465637321248226383092103297701648054726243842374862411453093812206564914032
7510866433945175121615265453613331113140424368548051067658434935238369596534280717687753283
4823434555736671973139274627362910821067928078471803532913117677892465908993863545932789452
3777674406192240337638674004021330343297496902028328145933418826817683893072003634795623117
1031012919531697946076327375892535307725523759437884345040677155557790564504430166401194625
8097221672975861502696844314695203461493229110597067624326851599283470989128470674086200858
7135016260312071903172086094081298321581077282076353186624611278245537208532365305775956430
0725177443150515396009051686032203491632226408852488524331580515348496224348482993809050704
8348244932745373262456775587908918719080366205800959474315005240253270974699531877072437682
5907419939632265984147498193609285223945039707165443156421328157688908058783183404917434556
2705202235648464951961124602683139709750693826487066132645076650746115126775227486215986425
3071129844118262266105716351506926002986170494542504749137811515413994155067125627119713325
2763631939606902895650288268608362241082050562430701794976171121233066073310059947366875L
```

IT'S OVER 9000!!!

aaaand it’s done !

of course one should not use recursion for Fibonacci, but a loop… anyway…

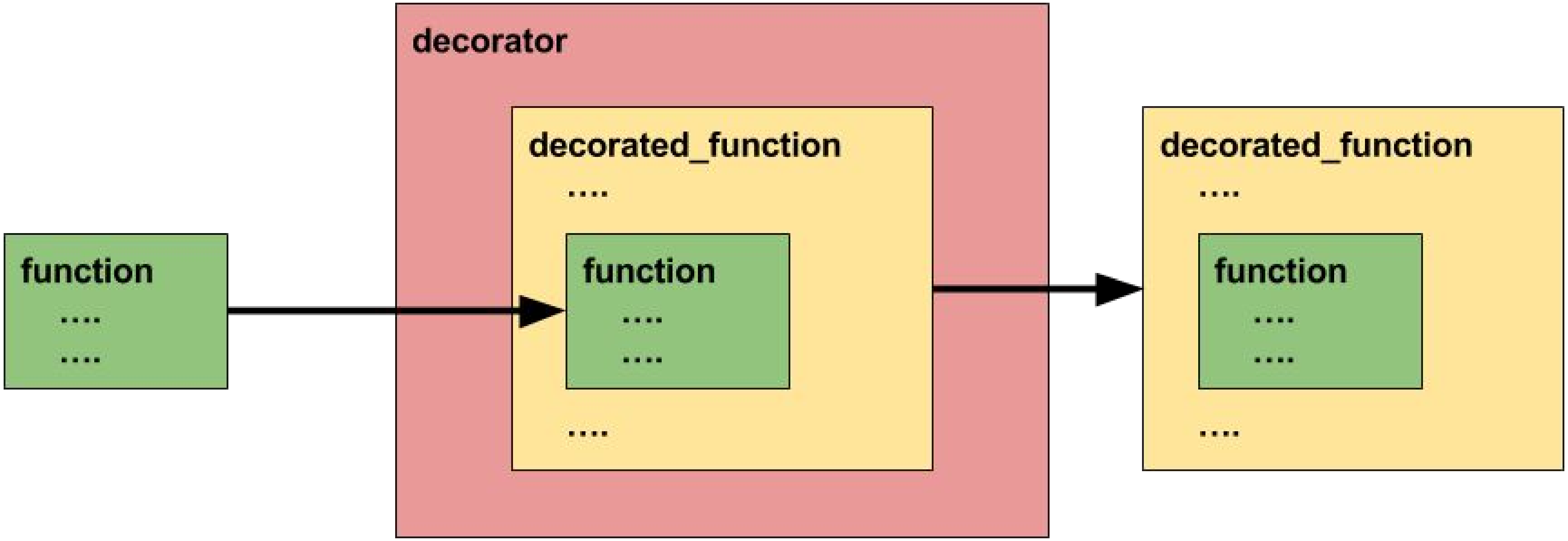
```python
def fibonacci_5(n):
    a = 1
    b = 1
    for i in range(2, n):
        next = a + b
        a = b
        b = next
    return b
```



© Éditions Xavier Barral / Patrick Gries

decorators

The decorator pattern is a pattern in which a function is wrapped by another function in order to add functionalities.



fibonacci_3(n)

```python
def memo(fn):
   cache = {}
   def _fn(n):
       if n not in cache:
           cache[n] = fn(n)
       return cache[n]
   return _fn

@memo
def fibonacci_3(n):
   if n <= 2:
       return 1
   return fibonacci_3(n - 1) + fibonacci_3(n - 2)
```

fibonacci_3(n)

```python
def memo(fn):                       # permits multiple argument functions
    cache = {}
    def _fn(*args):                 # args is a tuple of the arguments
        if args not in cache:
            cache[args] = fn(*args) # *args unpacks the arguments
        return cache[args]
    return _fn

@memo
def fibonacci_3(n):
    if n <= 2:
        return 1
    return fibonacci_3(n - 1) + fibonacci_3(n - 2)
```

fibonacci_3(n)

```python
import sys
import threading

def main():
    print fibonacci_3(10000)

threading.stack_size(128 * 2**20) # 128MB stack
sys.setrecursionlimit(2**20)  # something really big

# only new threads get the redefined stack size
thread = threading.Thread(target=main)
thread.start()
```

Easy parsing

```
def row(fn):
    return map(fn, raw_input().strip().split())

a, b, c = row(int)
```

https://goo.gl/Qs9Gg6

```
from collections import namedtuple

Item = namedtuple('Item', 'id value weight')

it = Item(1, 20, 7.5)
print it.value
print it.weight
```

https://goo.gl/MZqmJd

Generators - numbers

```python
def numbers(start=0):
    while True:
        yield start
        start += 1

for n in numbers():
    print n
```

tip: you can kill computations with Ctrl + C

Generators - fibonacci (again!)

```
def fibonacci_generator():
    a = b = 1
    while True:
        yield a
        a, b = b, a + b

for x in fibonacci_generator():
    print x
```

tip: you can kill computations with Ctrl + C

How to run programs

- Read from standard input (**input()**, **raw_input()**)
- Print to standard output (**print**)
- Use redirections

```
python solution.py < input.txt > output.txt
```

What about libraries?

- The first rule of PyClub is: **use pip**
- The second rule of PyClub is: **USE PIP + VENV**

Useful Python Libraries:

Virtualenv	github.com/pypa/virtualenv
Numpy	github.com/numpy/numpy
Pool	docs.python.org/2/library/multiprocessing

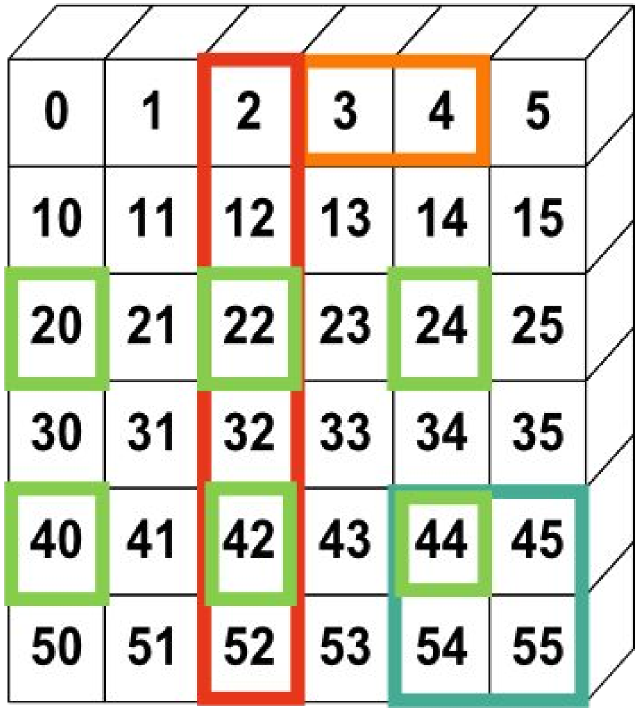
pip install bigstack python-memo

```python
from bigstack import *
from memo import *

@memo
def fibonacci(n):
    print 'computing %d' % n
    if n <= 2: return 1
    return fibonacci(n-1) + fibonacci(n-2)

@bigstack
def main():
    print fibonacci(10000)

main()
```

DECORATORS
DECORATORS EVERYWHERE
memegenerator.net

numpy slicing

```
>>> a[0,3:5]
array([3,4])


>>> a[4:,4:]
array([[44, 45],
       [54, 55]])


>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

- Numpy is implemented in C => **super fast**
- Numpy slicing does not duplicate data => **super fast**

LET'S DO MULTITHREADING
In Python
quickmeme.com

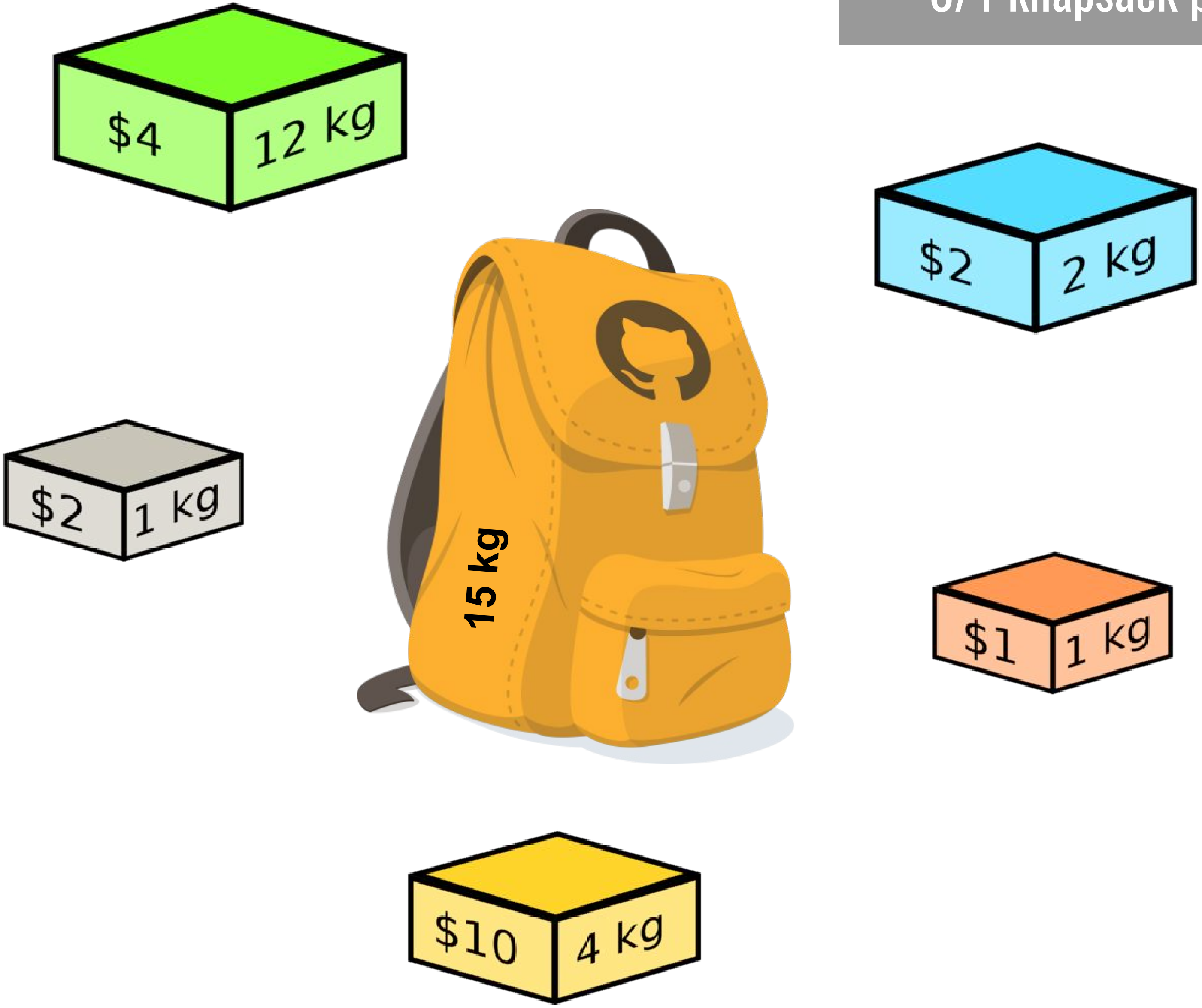
multiprocessing.Pool

```
from multiprocessing import Pool
from time import sleep

def slow_square(x):
    print 'computing square(%d)' % x
    sleep(2)
    return x * x


pool = Pool(processes=4)
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9]

print pool.map(slow_square, lst)
```

Greedy Algorithms

- altresì detti *golòssi* -

0/1 knapsack problem



tip: click the knapsack

0/1 Knapsack problem

```python
from collections import namedtuple
from random import randint

Item = namedtuple("Item", "id weight value")

def build_items(n):
    return [Item(i, randint(1,9), randint(1,9)) for i in range(n)]

n = 20
max_weight = 15
items = build_items(n)
```

Bruteforce solution

```python
from itertools import combinations

def powerset(lst):
    for length in range(len(lst) + 1):
        for combination in combinations(lst, r=length):
            yield combination

def knapsack_bruteforce(items, max_weight):
    best_set = []
    best_value = 0
    for item_set in powerset(items):
        value = sum(item.value for item in item_set)
        weight = sum(item.weight for item in item_set)
        if weight <= max_weight and value > best_value:
            best_set = item_set
            best_value = value
    return best_set, best_value

print 'bruteforce...'
k, v = knapsack_bruteforce(items, max_weight)
print 'value: %d\nknapsack: %s\n' % (v, k)
```

TOO SLOW
quickmeme.com

Greedy solution - 1

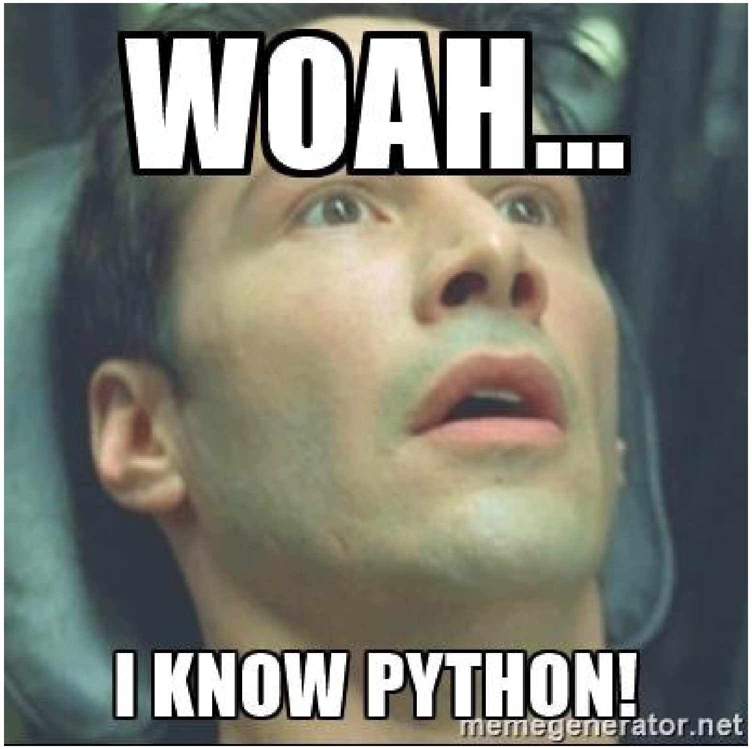
```python
def value(item): return item.value

def weight(item): return item.weight

def density(item): return float(item.value) / item.weight

def knapsack_greedy(items, max_weight, keyFunc):
    knapsack = []
    knapsack_value = 0
    remaining_weight = max_weight
    items = sorted(items, key=keyFunc, reverse=True)

    for item in items:
        if item.weight <= remaining_weight:
            remaining_weight -= item.weight
            knapsack_value += item.value
            knapsack.append(item)

    return knapsack, knapsack_value
```

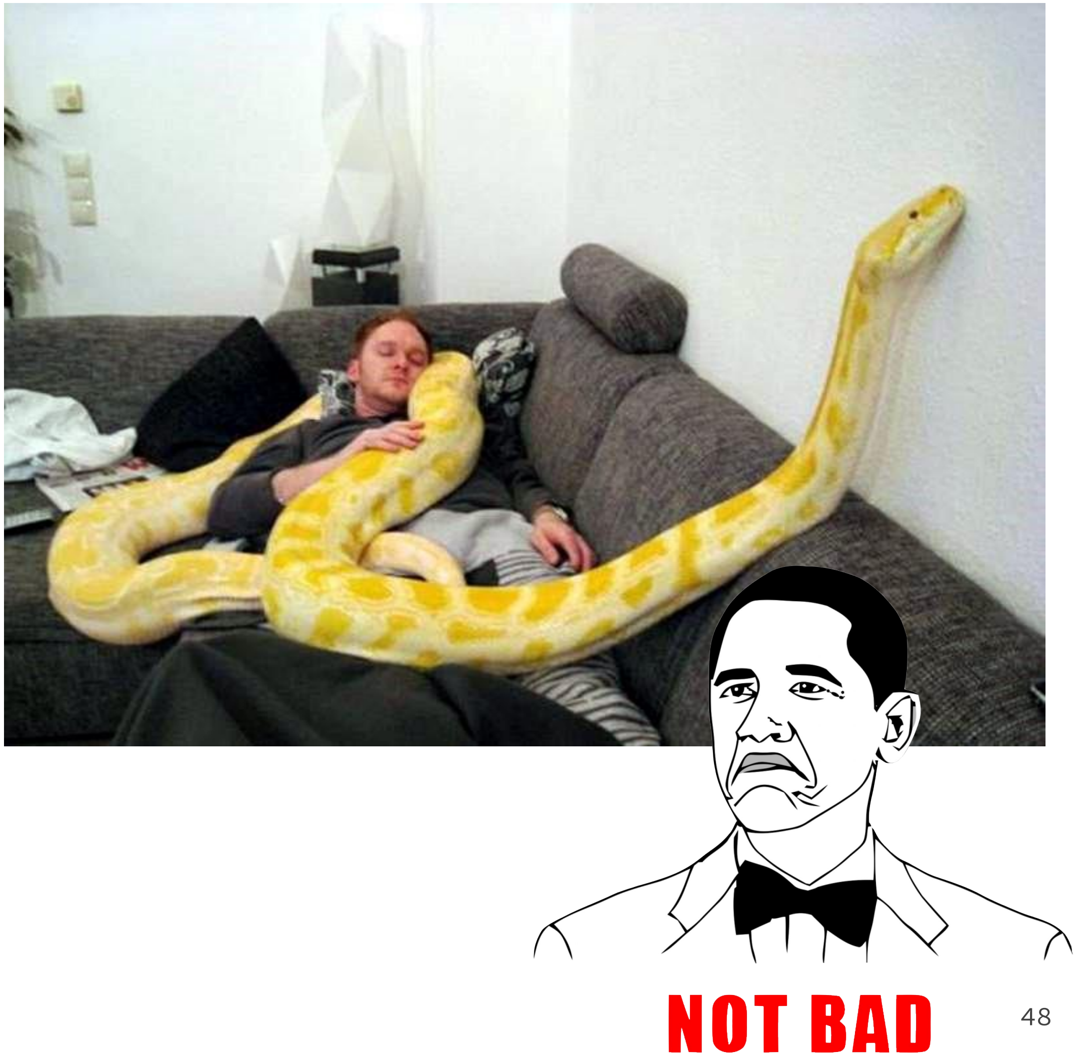
Greedy solution - 2

```
print 'greedy by value...'
k, v = knapsack_greedy(items, max_weight, value)
print 'value: %d\nknapsack: %s\n' % (v, k)

print 'greedy by weight...'
k, v = knapsack_greedy(items, max_weight, weight)
print 'value: %d\nknapsack: %s\n' % (v, k)

print 'greedy by density...'
k, v = knapsack_greedy(items, max_weight, density)
print 'value: %d\nknapsack: %s\n' % (v, k)
```

See the code running at:

https://repl.it/Floh/1


WOAH...
I KNOW PYTHON!
memegenerator.net

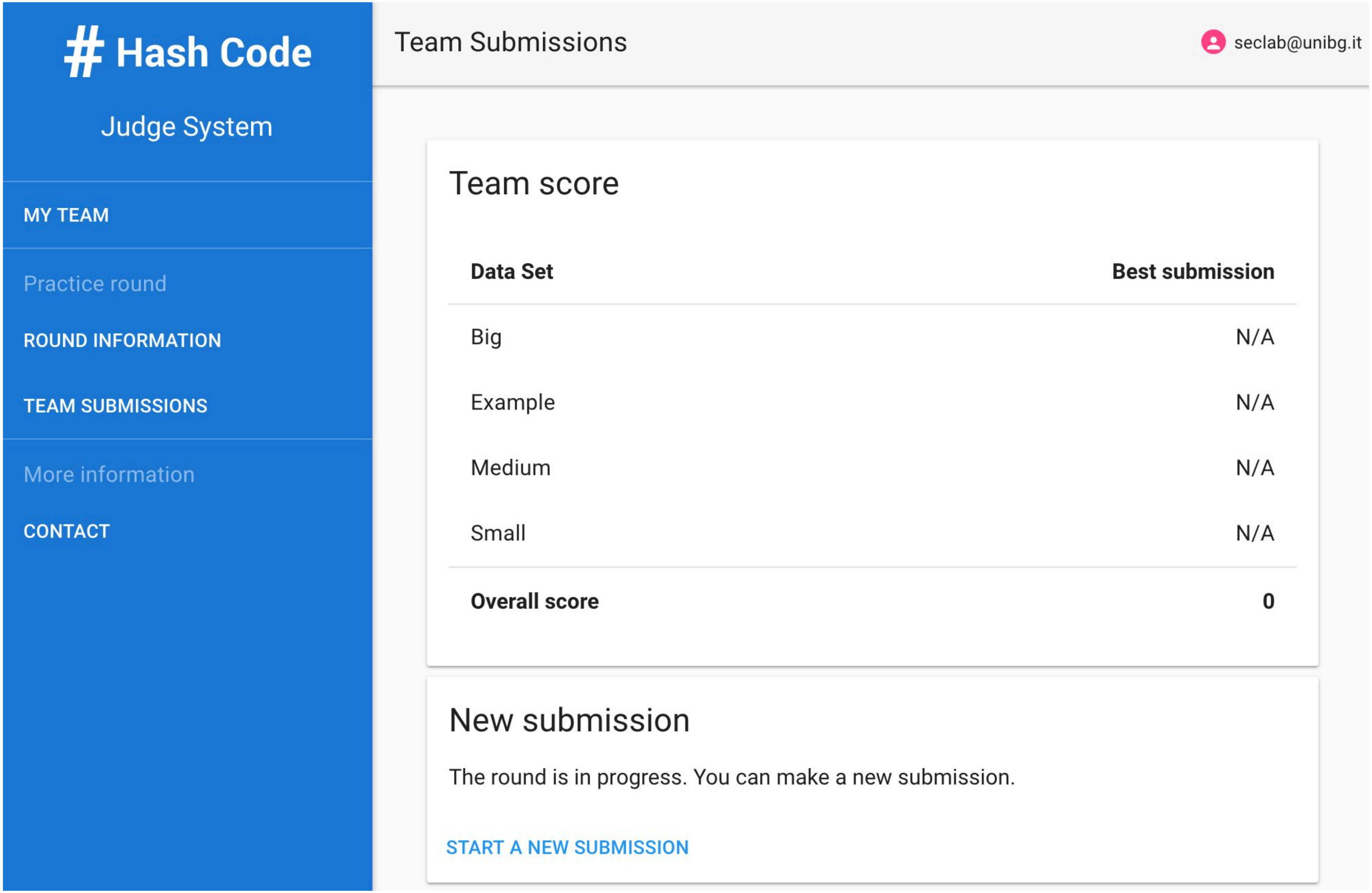

NOT BAD

a kind reminder...

Hash Code

Pizza

Practice Problem for Hash Code 2017



Hash Code

Judge System

MY TEAM

Practice round

ROUND INFORMATION

TEAM SUBMISSIONS

More information

CONTACT

Team Submissions

seclab@unibg.it

Team score

Data Set	Best submission
Big	N/A
Example	N/A
Medium	N/A
Small	N/A
Overall score	**0**

New submission

The round is in progress. You can make a new submission.

START A NEW SUBMISSION

https://hashcodejudge.withgoogle.com

Unibg Seclab - Practice problem internal competition

The team that submits the highest scores for the practice problem gets free pizza during the competition.

=>

Unibg Seclab - Practice problem internal competition

Wednesday February 22 - 16:30 - B004

Solutions for “Pizza”

How it works

1. Send an e-mail at seclab@unibg.it (even if you didn’t solve the problem, even just to say you love us)
2. Prepare a couple of slides to explain your solution
3. Eat pizza!
4. GOTO 1

(we will also show you our solution)

Feedback

Ti chiediamo di dedicarci 2 minuti a compilare il form qui sotto. Nessuna risposta è obbligatoria, ma più informazioni ci darai, più ci aiuterai a fare meglio le prossime volte!

https://goo.gl/forms/IA0HrHyGMlfbXB7F3

Unibg Seclab

est. 2015

seclab.unibg.it