Python Funcional

O como aprendí a dejar de preocuparme
y amar las lambdas

Presentador

- Ezequiel Alvarez @clrnd
- Cs. de la Computación en la FCEN
- Programador en GRUPO MSA

¿Qué es funcional?

- Inmutabilidad
- High-order/first-class
- Funciones puras
- Recursión
- “Composability” (reusabilidad)
- Abstracción

¿Python Funcional?

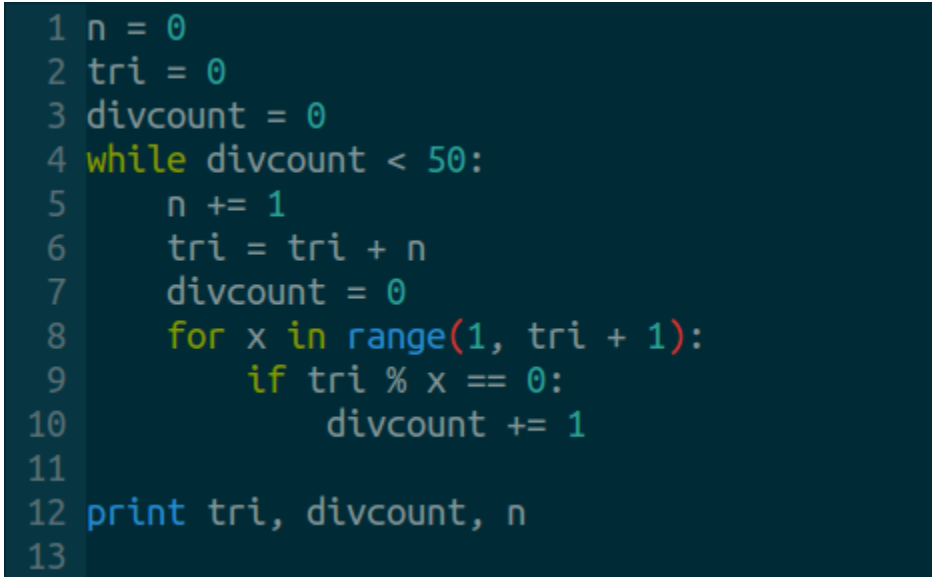
- A Guido no le importa
- operator
- itertools
- functools (al menos en py3)
- fn.py

Ejercicio: Project Euler - Problem 12

- ¿Cuál es el valor del primer número triangular en tener más de 500 divisores?
- Números triangulares: 1, 3, 6, 10, 15
- Cada uno es la suma de todos los anteriores
- Para llegar a 500 necesitamos hacer bardo matematico
- Así que lo hacemos hasta 50
- :D

Ejercicio: Project Euler - Problem 12

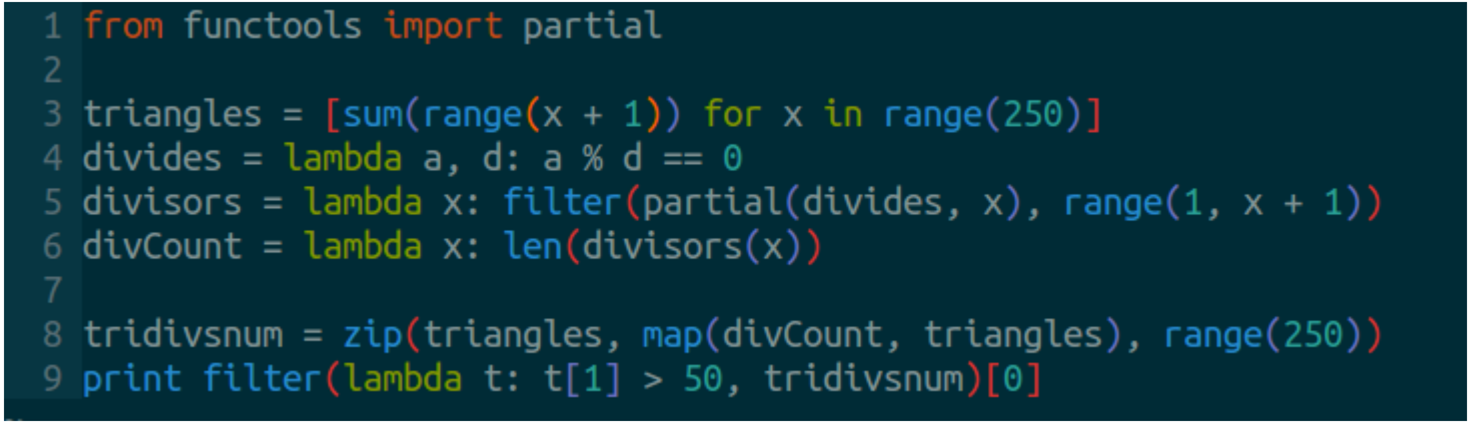
Solución imperativa:



```python
n = 0
tri = 0
divcount = 0
while divcount < 50:
    n += 1
    tri = tri + n
    divcount = 0
    for x in range(1, tri + 1):
        if tri % x == 0:
            divcount += 1

print tri, divcount, n

```

Ejercicio: Project Euler - Problem 12

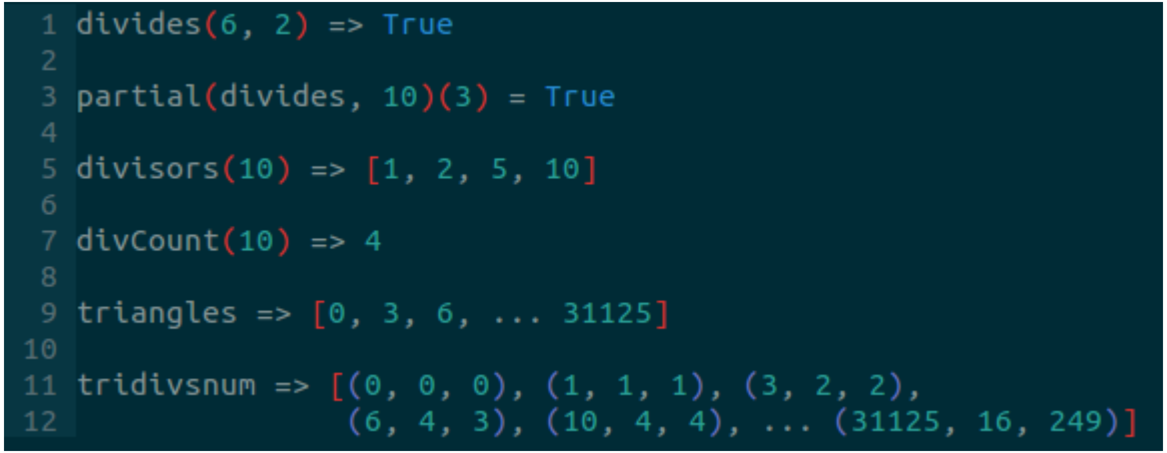
Solución funcional:



```python
from functools import partial

triangles = [sum(range(x + 1)) for x in range(250)]
divides = lambda a, d: a % d == 0
divisors = lambda x: filter(partial(divides, x), range(1, x + 1))
divCount = lambda x: len(divisors(x))

tridivsnum = zip(triangles, map(divCount, triangles), range(250))
print filter(lambda t: t[1] > 50, tridivsnum)[0]
```

Ejercicio: Project Euler - Problem 12

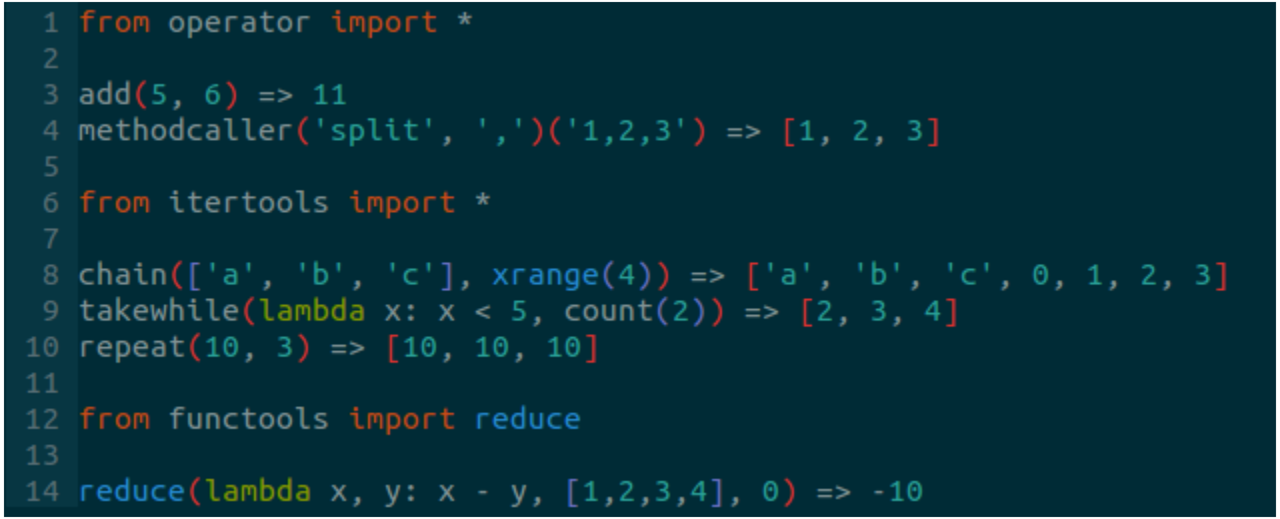
Modularidad:



```
divides(6, 2) => True

partial(divides, 10)(3) = True

divisors(10) => [1, 2, 5, 10]

divCount(10) => 4

triangles => [0, 3, 6, ... 31125]

tridivsnum => [(0, 0, 0), (1, 1, 1), (3, 2, 2),
               (6, 4, 3), (10, 4, 4), ... (31125, 16, 249)]
```

Ejercicio: Project Euler - Problem 12

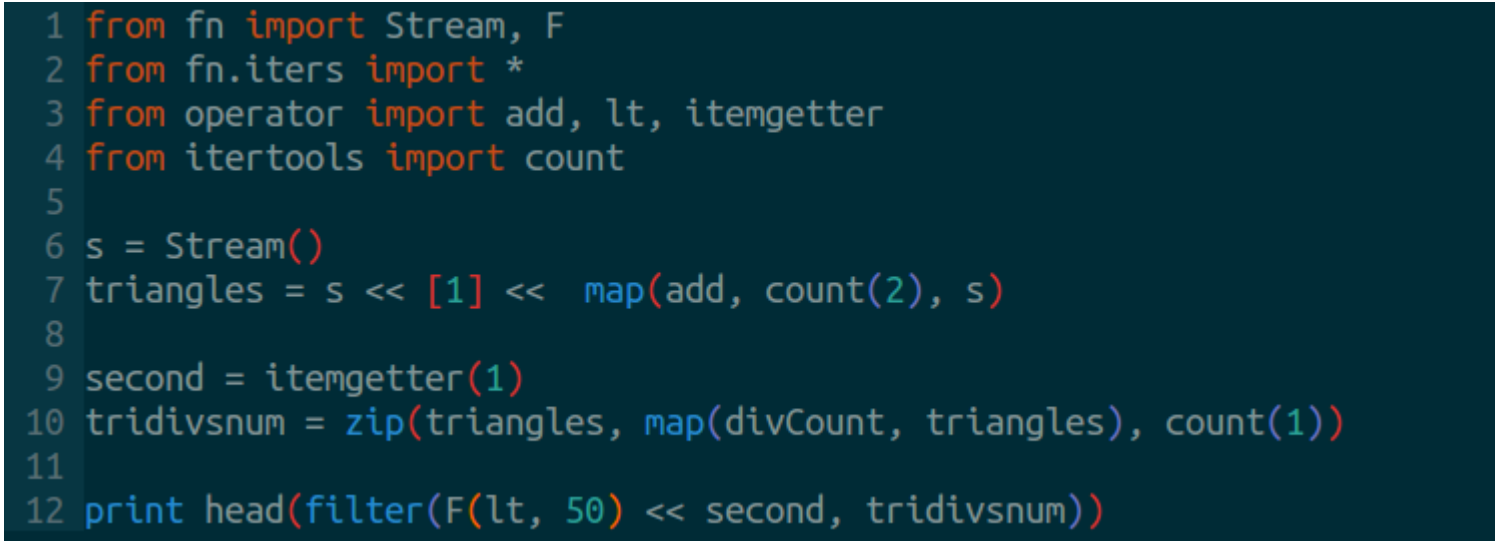
operator, itertools, functools



```
from operator import *

add(5, 6) => 11
methodcaller('split', ',')('1,2,3') => [1, 2, 3]

from itertools import *

chain(['a', 'b', 'c'], xrange(4)) => ['a', 'b', 'c', 0, 1, 2, 3]
takewhile(lambda x: x < 5, count(2)) => [2, 3, 4]
repeat(10, 3) => [10, 10, 10]

from functools import reduce

reduce(lambda x, y: x - y, [1,2,3,4], 0) => -10
```

Ejercicio: Project Euler - Problem 12

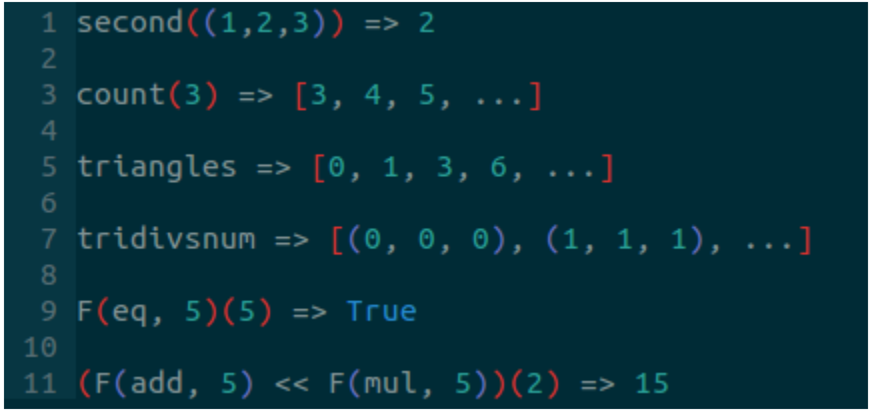
Solución funcional genérica:



```python
from fn import Stream, F
from fn.iters import *
from operator import add, lt, itemgetter
from itertools import count

s = Stream()
triangles = s << [1] <<  map(add, count(2), s)

second = itemgetter(1)
tridivsnum = zip(triangles, map(divCount, triangles), count(1))

print head(filter(F(lt, 50) << second, tridivsnum))
```

Ejercicio: Project Euler - Problem 12

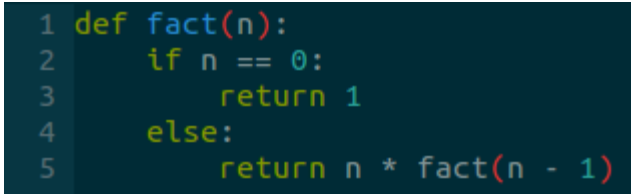
Listas infinitas y composición:



```
second((1,2,3)) => 2

count(3) => [3, 4, 5, ...]

triangles => [0, 1, 3, 6, ...]

tridivsnum => [(0, 0, 0), (1, 1, 1), ...]

F(eq, 5)(5) => True

(F(add, 5) << F(mul, 5))(2) => 15
```

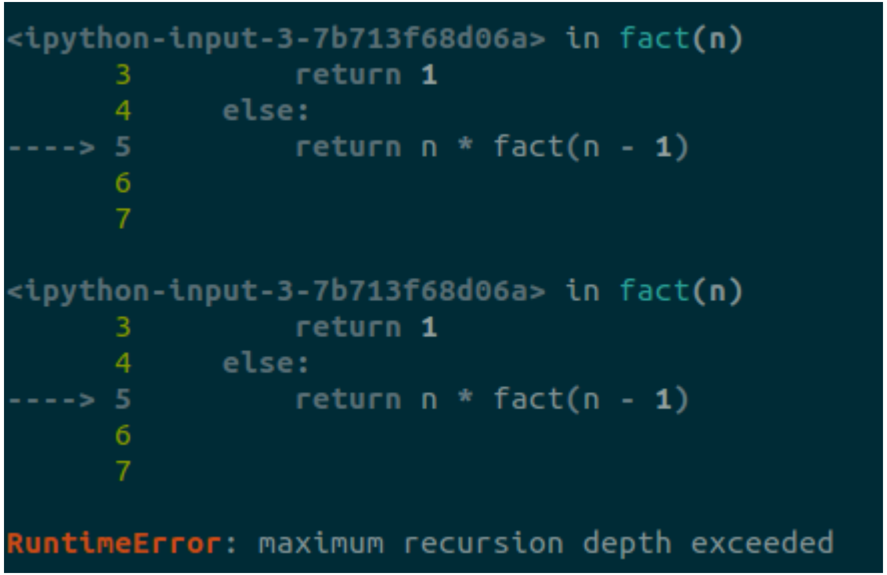
Recursión: Say Goodbye to Whiles?

Factorial



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

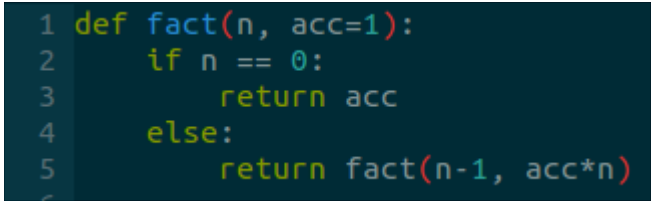
fact(1000)



```
<ipython-input-3-7b713f68d06a> in fact(n)
      3         return 1
      4     else:
----> 5         return n * fact(n - 1)
      6
      7

<ipython-input-3-7b713f68d06a> in fact(n)
      3         return 1
      4     else:
----> 5         return n * fact(n - 1)
      6
      7

RuntimeError: maximum recursion depth exceeded
```

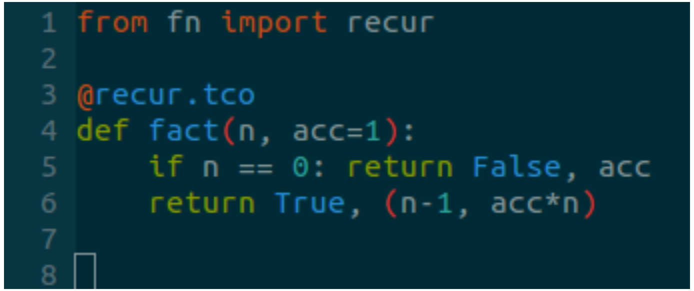
Recursión: TCO

Tail Call Optimization:



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

En python:



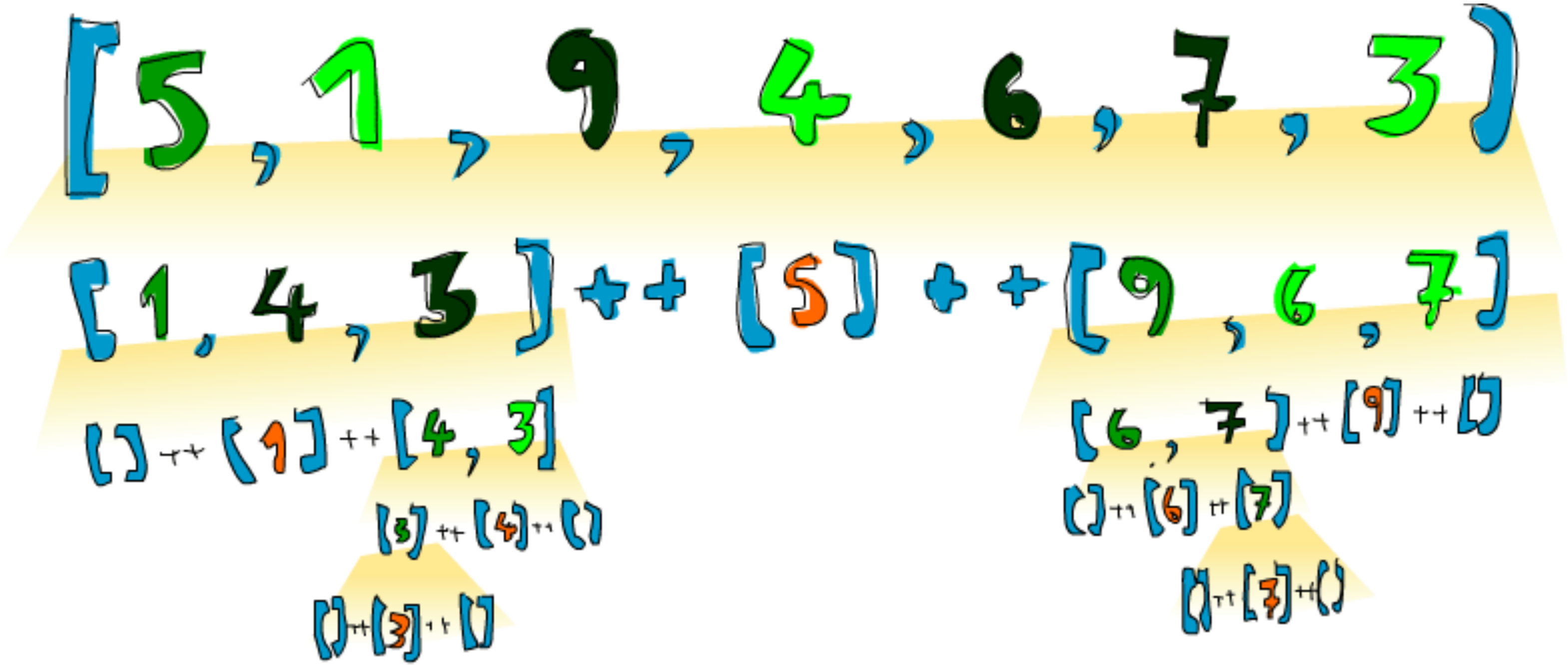
```python
from fn import recur

@recur.tco
def fact(n, acc=1):
    if n == 0: return False, acc
    return True, (n-1, acc*n)
```

Recursión: Quicksort

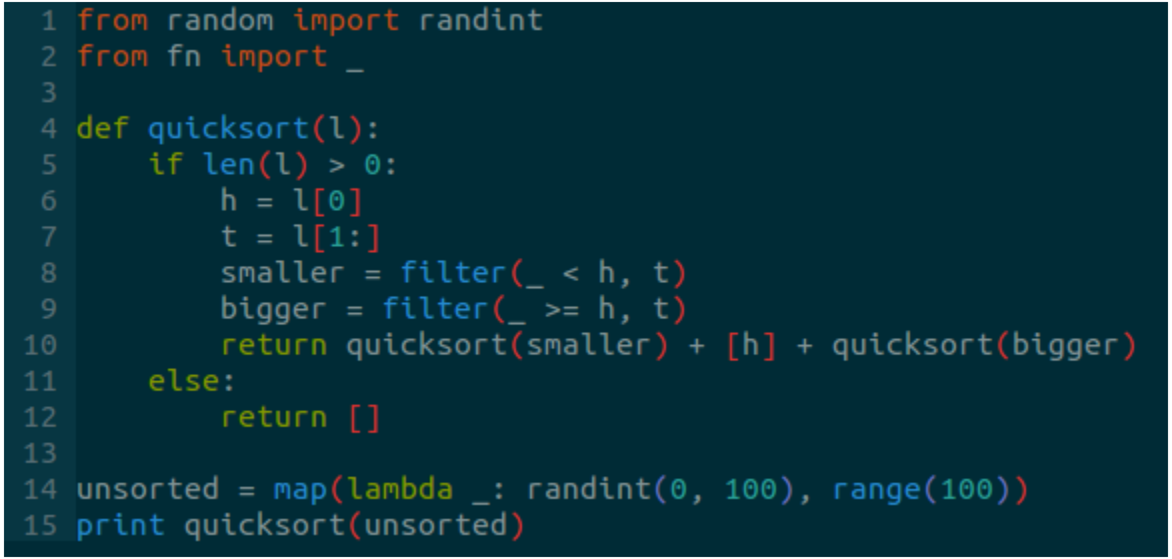
1. Tomo algún elemento E
2. Tomo todos los menores L y los ordeno
3. Tomo todo los mayores G y los ordeno
4. Devuelvo L + E + G

- $O(n \log n)$ en general
- $O(n^2)$ con mala suerte

Recursión: Quicksort

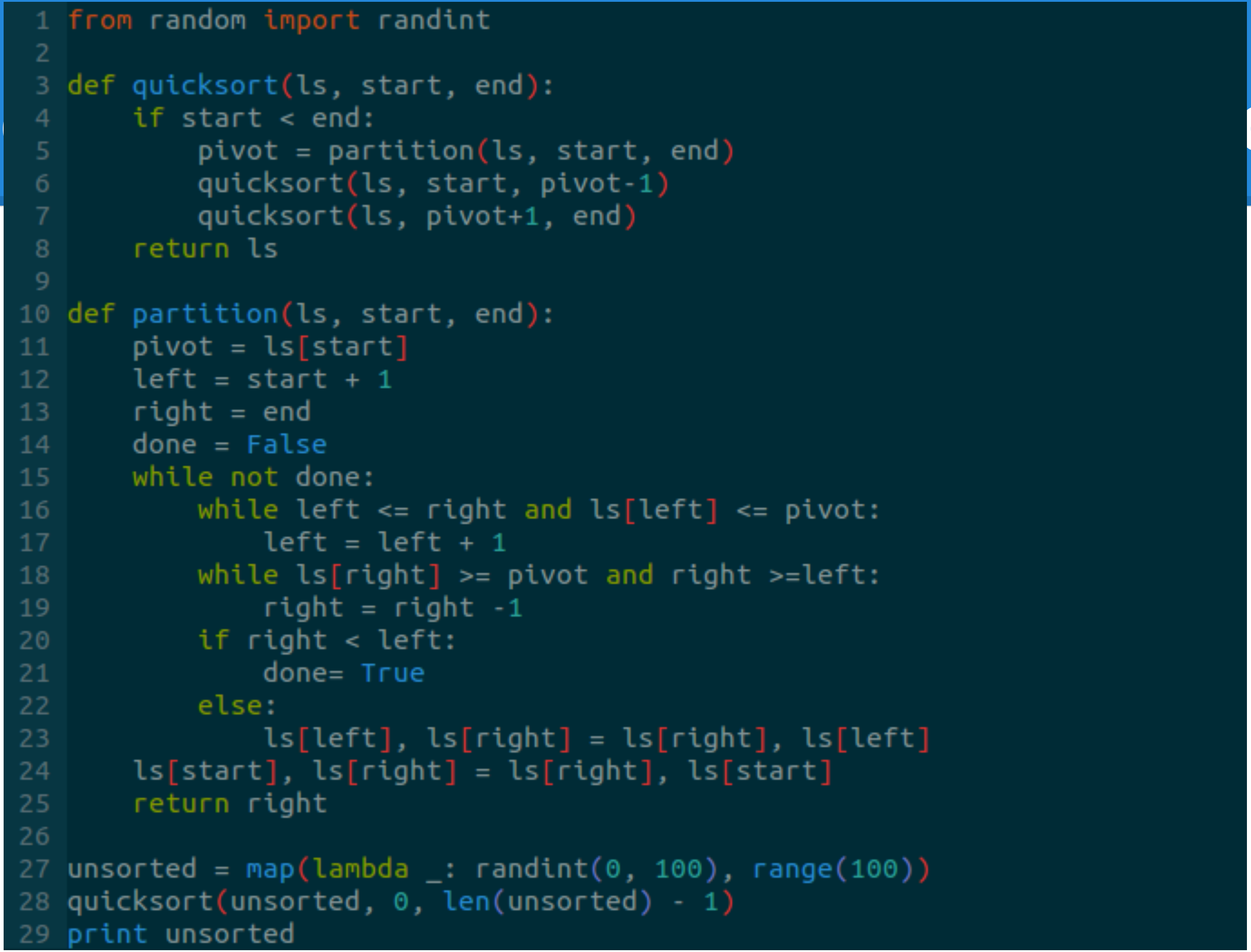


Recursión: Quicksort - Funcional



```python
from random import randint
from fn import _

def quicksort(l):
    if len(l) > 0:
        h = l[0]
        t = l[1:]
        smaller = filter(_ < h, t)
        bigger = filter(_ >= h, t)
        return quicksort(smaller) + [h] + quicksort(bigger)
    else:
        return []

unsorted = map(lambda _: randint(0, 100), range(100))
print quicksort(unsorted)
```

Re o



```python
from random import randint

def quicksort(ls, start, end):
    if start < end:
        pivot = partition(ls, start, end)
        quicksort(ls, start, pivot-1)
        quicksort(ls, pivot+1, end)
    return ls

def partition(ls, start, end):
    pivot = ls[start]
    left = start + 1
    right = end
    done = False
    while not done:
        while left <= right and ls[left] <= pivot:
            left = left + 1
        while ls[right] >= pivot and right >=left:
            right = right -1
        if right < left:
            done= True
        else:
            ls[left], ls[right] = ls[right], ls[left]
    ls[start], ls[right] = ls[right], ls[start]
    return right

unsorted = map(lambda _: randint(0, 100), range(100))
quicksort(unsorted, 0, len(unsorted) - 1)
print unsorted
```

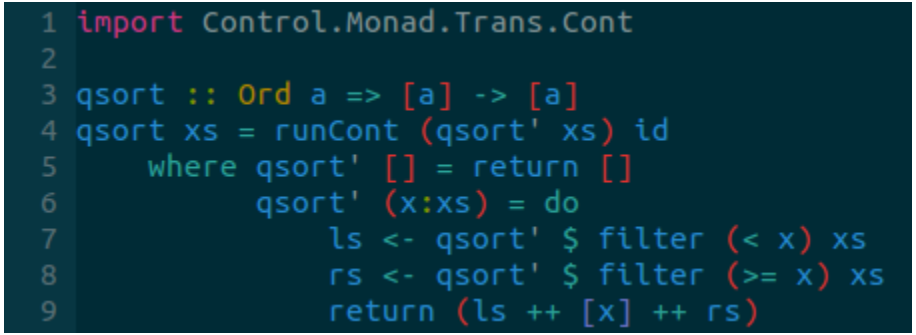
Python Funcional: Pros

- lambda! (y clausuras)
- generators para evaluación retardada
- decorators
- ...

Python Funcional: Cons

- estructuras mutables (y nada más)
- TCO = NOPE
- pocas funciones de alto orden
- no pattern matching (a la Clojure)
- no let bindings

Haskell



```haskell
import Control.Monad.Trans.Cont

qsort :: Ord a => [a] -> [a]
qsort xs = runCont (qsort' xs) id
    where qsort' [] = return []
          qsort' (x:xs) = do
              ls <- qsort' $ filter (< x) xs
              rs <- qsort' $ filter (>= x) xs
              return (ls ++ [x] ++ rs)
```

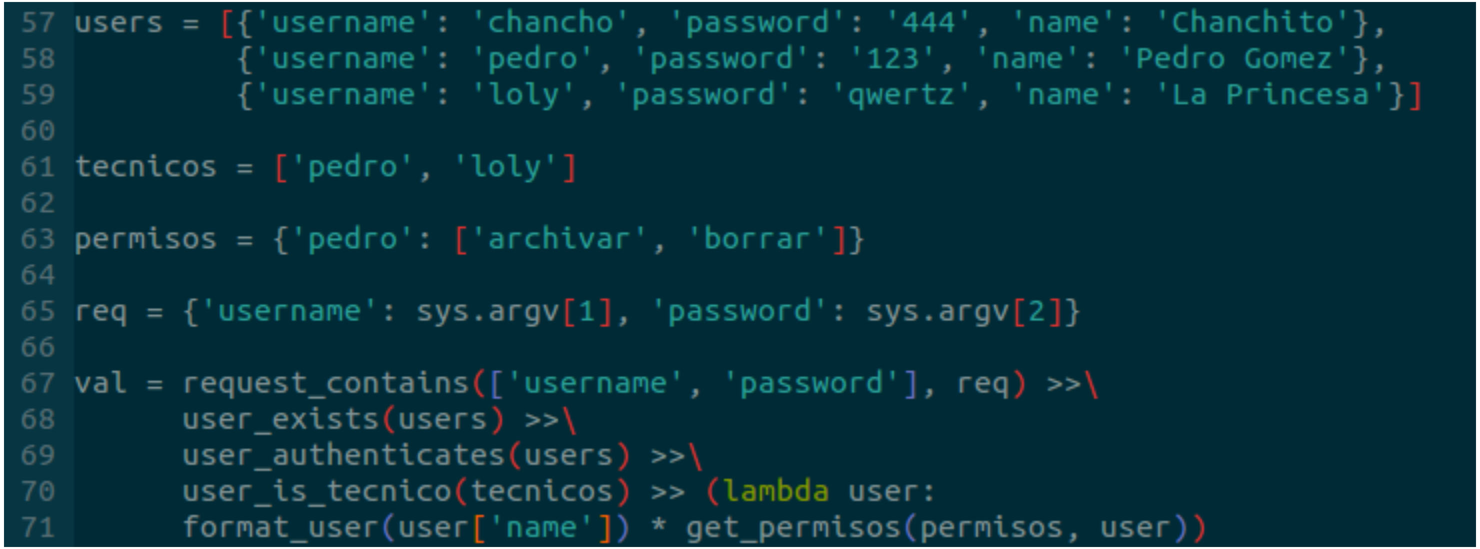
Links

- fn.py
    - https://github.com/kachayev/fn.py
- Structure and Interpretation of Computer Programs
    - http://mitpress.mit.edu/sicp/
- Okasaki: Purely Functional Data Structures
    - www.cs.cmu.edu/~rwh/theses/okasaki.pdf
- Learn You A Haskell For Great Good
    - http://learnyouahaskell.com

More Links

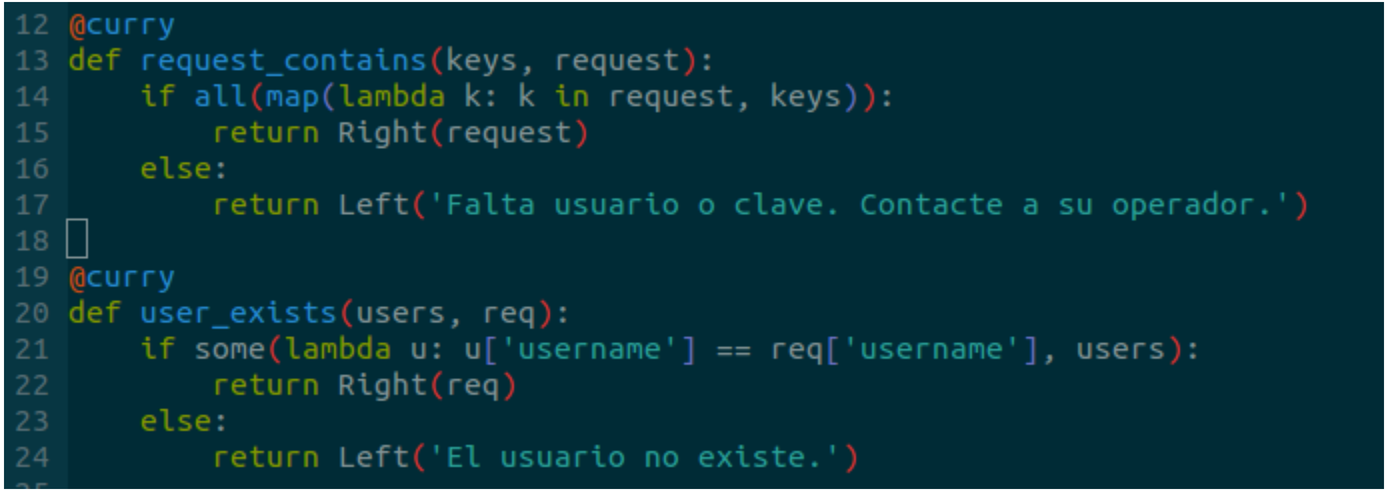
- Backus: Can programming be liberated from the von Neumann style?:
  - http://dl.acm.org/citation.cfm?id=359579
- Hutton - Meijer: Monadic Parsing in Haskell
  - http://www.cs.nott.ac.uk/~gmh/pearl.pdf
- Hickey: Are We There Yet?
  - http://www.infoq.com/presentations/Are-We-There-Yet-Rich-Hickey

Extra: PyMonad



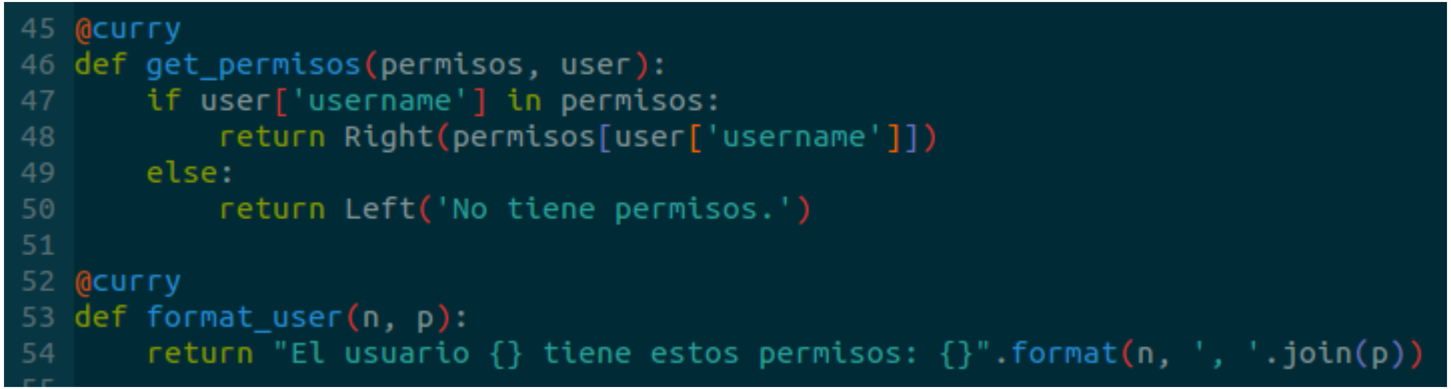
```
users = [{'username': 'chancho', 'password': '444', 'name': 'Chanchito'},
         {'username': 'pedro', 'password': '123', 'name': 'Pedro Gomez'},
         {'username': 'loly', 'password': 'qwertz', 'name': 'La Princesa'}]

tecnicos = ['pedro', 'loly']

permisos = {'pedro': ['archivar', 'borrar']}

req = {'username': sys.argv[1], 'password': sys.argv[2]}

val = request_contains(['username', 'password'], req) >>\
      user_exists(users) >>\
      user_authenticates(users) >>\
      user_is_tecnico(tecnicos) >> (lambda user:
      format_user(user['name']) * get_permisos(permisos, user))
```

Extra: PyMonad



```python
@curry
def request_contains(keys, request):
    if all(map(lambda k: k in request, keys)):
        return Right(request)
    else:
        return Left('Falta usuario o clave. Contacte a su operador.')

@curry
def user_exists(users, req):
    if some(lambda u: u['username'] == req['username'], users):
        return Right(req)
    else:
        return Left('El usuario no existe.')
```

Extra: PyMonad



```python
@curry
def get_permisos(permisos, user):
    if user['username'] in permisos:
        return Right(permisos[user['username']])
    else:
        return Left('No tiene permisos.')

@curry
def format_user(n, p):
    return "El usuario {} tiene estos permisos: {}".format(n, ', '.join(p))
```