用Python做些事

04-高效做事的函数

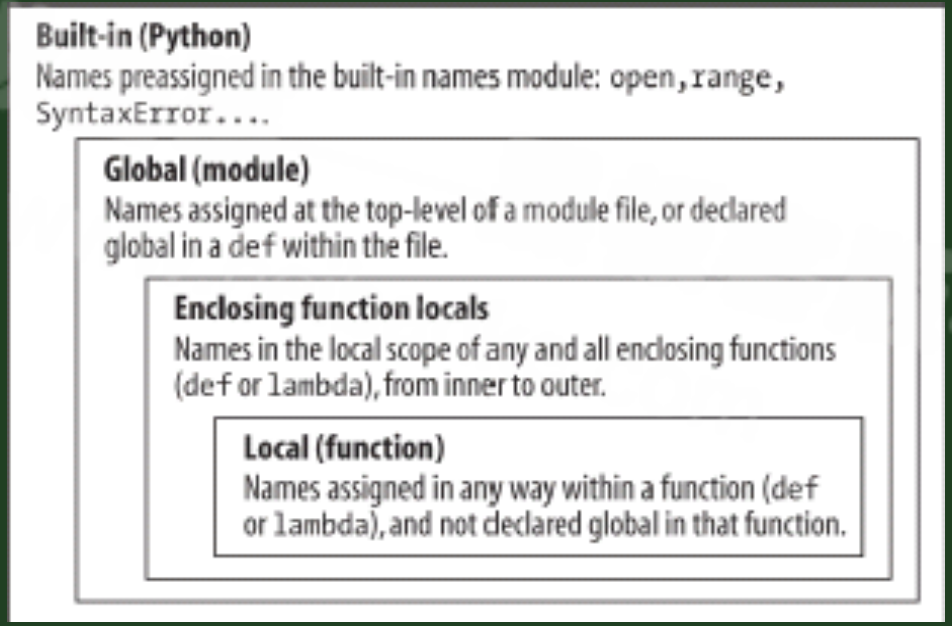
高效做事的函数

- 基本语法
- 函数式编程
- 回调
- 闭包和装饰器
- 递归
- 生成器和yield
- 井字棋-Tictactoe

基本语法

语法

作用域LEGB



基本语法—参数

arguments

```
def f1(a, b=2, c=3):
    print a, b, c
```

*arg和**kargs

```
def f2(s, *args):
    print args

def g(a, *args, **kargs):
    print a, args, kargs

f2(1,2,3,4)
g(1,2,3,4), g(1,2,3,b=5,c=6),
```

基本语法—二分法例子

math库

```
import math
math.sqrt(20.0)
```

公开课:
MIT--计算机科学与编程导论

二分法

基本语法—牛顿法例子

牛顿法

F(x)=x^2-20
F'(x)=2x
F'(x0)=2x0

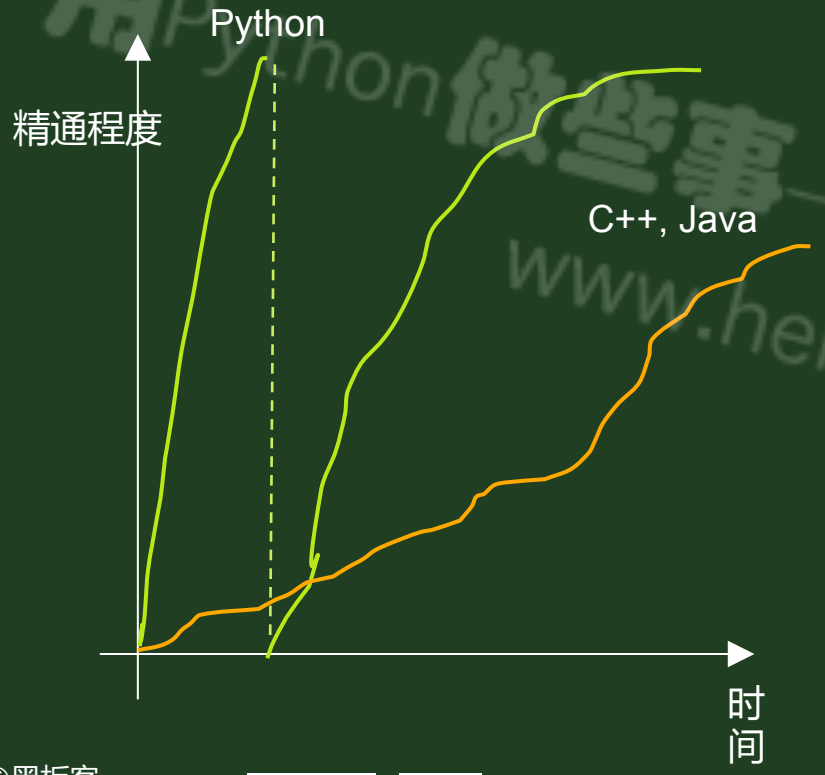
L(x) = F'(x0)x+b
b=F(x0)-F'(x0)x0
0=F'(x0)*x1+b
x1 = -b/F'(x0)
x1 = x0-F(x0)/F'(x0)

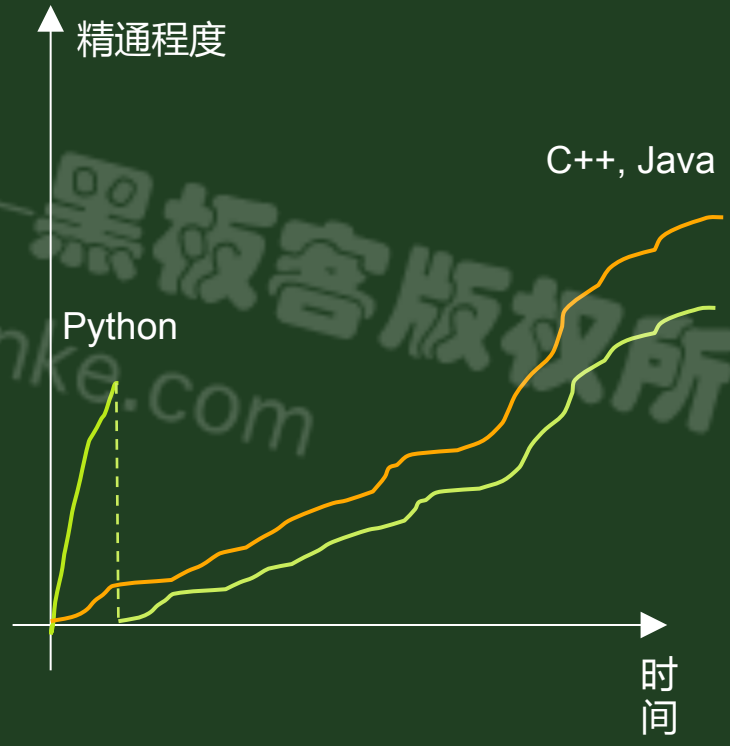
x0 = 10
x1 = 10-80/20 = 6
x2 = 6 – 16/12 = 4.66667
x3 = 4.66667 – 0.190476 = 4.4762

高效做事的函数

- ☑ ~~基本语法~~
- ☐ 函数式编程
- ☐ 回调
- ☐ 闭包和装饰器
- ☐ 递归
- ☐ 生成器和yield
- ☐ 井字棋-Tictactoe

函数式编程—编程方法





函数式编程—编程方法

不同语言的不同点

语法，运行环境和库的使用等

相同点

- ✓ 语言层面：编程方法和模式。
  哪些场景适合什么方法和模式？
- ✓ 底层：数据结构和算法
  怎么存储最高效，怎么折腾数据最快。概率论，统计学，数学。
- ✓ 高层：各种应用所需要的原理。
  比如，机器学习，网络，爬虫，信号处理等。

函数式编程—函数即对象

可直接赋给变量

My_sum = sum

有属性和方法

__call__
__name__

函数式编程—函数做参数

高阶函数

```
def f1(f, a, b):
    print f(a,b)
```

Forloop vs list comp

自带高阶函数

Filter
Map
Reduce

函数式编程—Map, Reduce

统计单词词频

很多文章，要统计他们十大最热门词汇

1. 文本处理，统计每个文章的词频
2. 合并不同文章的词频
3. 排序，输出

Github ----- wordsworth

函数式编程—lambda

用法

```
lower = (lambda x, y: x if x < y else y)
```

函数编程

```
lst = random.randint(-50,50)
lst2 = filter(lambda n: n > 0, lst)
lst3 = map(lambda x: x*2, lst)

c = sorted(lst, cmp=lambda x,y: x-y)
lst.sort(lambda x, y: 1 if x>y else -1))
```

函数式编程—函数返回

内部函数

```
def calc2(s):

    def f_add(a,b):  return a+b
    def f_mul(a,b):  return a*b
    def f_sub(a,b):  return a-b

    if s=='+':
        return  f_add
    elif s=='*':
        return f_mul
    elif s=='-':
        return f_sub
    else:
        assert False,   “error”
```

高效做事的函数

- ☑ ~~基本语法~~
- ☑ ~~函数式编程~~
- ☐ 回调
- ☐ 闭包和装饰器
- ☐ 递归
- ☐ 生成器和yield
- ☐ 井字棋-Tictactoe

回调Callback

函数作为参数

```
def test(callback):
    print 'test func begin'
    callback()
    print 'test func end'

def cb1():
    print 'callback 1'

def cb2():
    print 'callback 2'

test(cb1)
test(cb2)
```

例子：
不同顾客有不同的响应要求

有的顾客通知要发邮件
有的顾客通知要发QQ
有的要发微信
有的要发短信

需求：
文本文件，注册一个通知方式，一个地址，注册后通知消息就跟已注册的一起发送。

高效做事的函数

- ☑ ~~基本语法~~
- ☑ ~~函数式编程~~
- ☑ ~~回调~~
- ☐ 闭包和装饰器
- ☐ 递归
- ☐ 生成器和yield
- ☐ 井字棋-Tictactoe

闭包closure

绑定外部变量的函数

```
def pow_x(x):
    def echo(value):
        return value**x
    return echo

lst = [pow_x(2), pow_x(3), pow_x(4)]
for p in lst:
    print p(2)
```

1. 嵌套函数
2. 内部函数用到了外部变量
3. 外部函数返回内部函数

闭包closure

绑定外部变量的函数

```
def pow_y(x):
    def echo(value):
        #x[0]=x[0]*2
        #x=[2,2]
        return value**x[0],value**x[1]
    return echo


def largerx(x):
    def echo(value):
        return True if value>x[0] else False
    return echo
```

1. 内部函数不能”改变”外部变量
2. 内部函数用到了外部变量为list，则可以从外部或内部改变值，并且即使外部没有引用也不会回收

装饰器—无嵌套

函数作为返回值

1. @装饰器会提前执行
2. 目标函数无法带参数
3. 目标函数调用后无法插入代码

```
def decorator(f):
    print "before f() called."
    return f


def myfunc1():
    print " myfunc1() called."

@decorator
def myfunc2():
    print " myfunc2() called."

if __name__=="__main__":
    pass
    #decorator(myfunc1)
    #myfunc2()
```

装饰器—2层嵌套

函数带参数

```
def time_cost(f):
    def _f(*arg, **kwarg):
        start = time.clock()
        f(*arg,**kwarg)
        end = time.clock()
        print end-start
    return _f

@time_cost
def list_comp(length):
    return [(x,y) for x in range(length) for y in range(length) if x*y > 25]

list_comp(1000)
```

1. ~~@装饰器会提前执行~~
2. ~~目标函数无法带参数~~
3. ~~函数调用后无法插入代码~~
4. ~~装饰器无法带参数~~

装饰器—3层嵌套

装饰器带参数

```
def time_cost(f):
    def _f(*arg, **kwarg):
        start = time.clock()
        f(*arg,**kwarg)
        end = time.clock()
        print end-start

    return _f
```

```
def time_cost(timef):

    def decorator(f):
        def _f(*arg, **kwarg):
            start = timef()
            a=f(*arg,**kwarg)
            end = timef()
            print f.__name__,"run cost time is ",end-start
            return a
        return _f

    return decorator
```

装饰器—装饰模式

设计模式

给小明穿衣服，

工作时穿工作服，西装，皮鞋，裤子
运动时穿运动服，T恤，运动鞋，裤子，帽子

把这种搭配做成套装可以直接给另一个人小红穿上

这种套装可以根据日期随意更换，
比如周1-4穿工作服，但周2的工作服不穿西装，穿体恤。
周5-7穿运动装，但周5不带帽子。

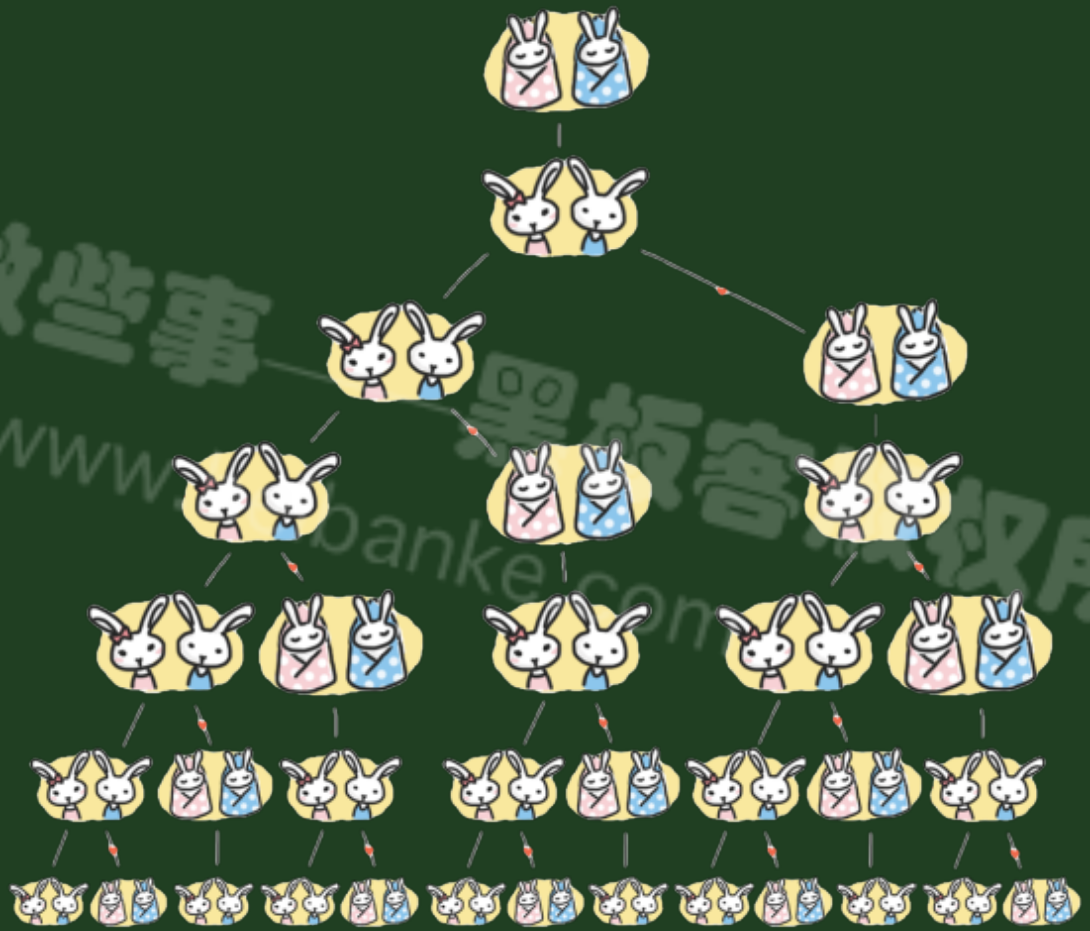
高效做事的函数

- ☑ ~~基本语法和作用域~~
- ☑ ~~函数式编程~~
- ☑ ~~回调~~
- ☑ ~~闭包和装饰器~~
- ☐ 递归
- ☐ 生成器和yield
- ☐ 井字棋-Tictactoe

递归

函数调用自己

1对小兔子，一个月后成年，成年后每月下一对小兔子。1年后多少对？



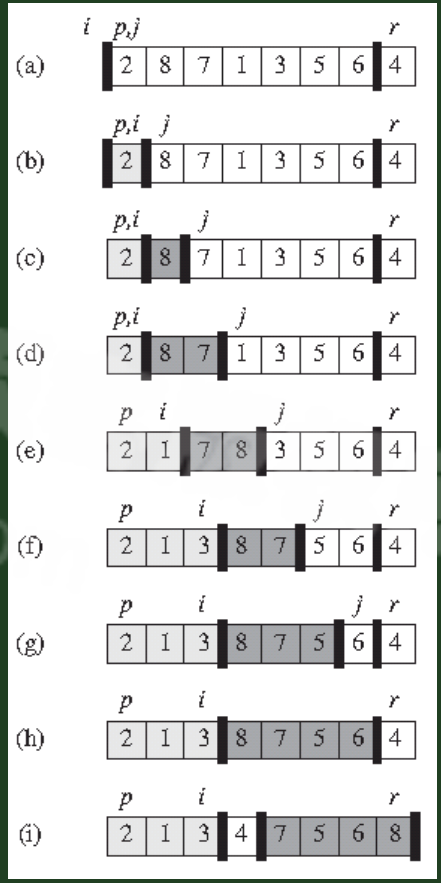
递归—编程方法

快速排序

Introduction to algorithm，Tomas H.Cormen
算法导论

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```



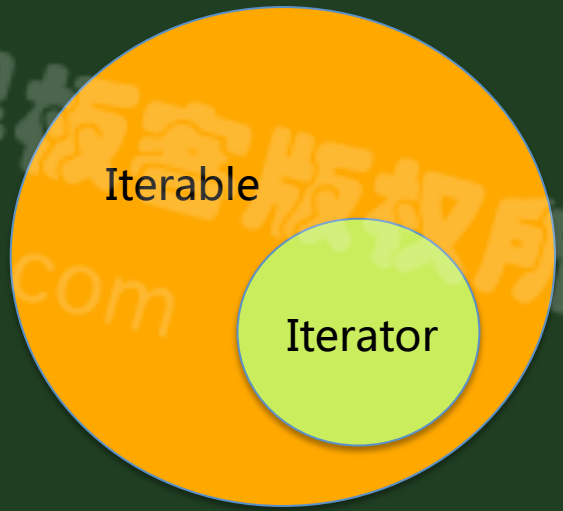
高效做事的函数

- ☑ ~~基本语法~~
- ☑ ~~函数式编程~~
- ☑ ~~回调~~
- ☑ ~~闭包和装饰器~~
- ☑ ~~递归~~
- ☐ 生成器和yield
- ☐ 井字棋-Tictactoe

生成器和yield

Iterable, Iterator, Generator

```
def fib():
    a, b = 0, 1
    while  True:
        yield b
        a, b = b, a + b

A=fib()
A.next()
```



生成器和yield

send用法

```
def func():
    input =[]
    while True:
        a = (yield)
        Your statement
        input.append(a)
```

Send, itertools

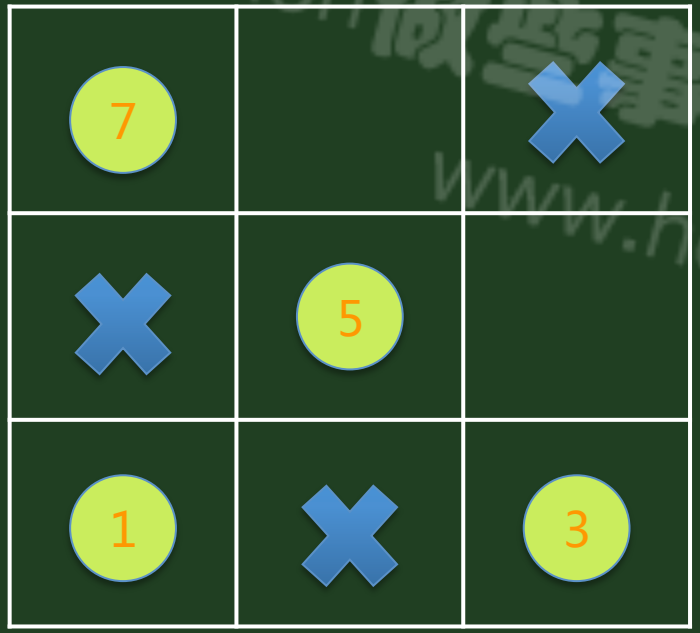
```
horses=[1,2,3,4]
races = itertools.permutations(horses)

A=itertools.product([1,2],[3,4])
B=itertools.repeat([1,2],4)

C=itertools.chain(races, a, b)
```

高效做事的函数

- ☑ ~~基本语法~~
- ☑ ~~函数式编程~~
- ☑ ~~回调~~
- ☑ ~~闭包和装饰器~~
- ☑ ~~递归~~
- ☑ ~~生成器和yield~~
- ☐ 井字棋-Tictactoe

井字棋Tictactoe

井字棋



1. 选择先走的符号　inputPlayerLetter
2. 打印棋盘　drawBoard
3. 轮流输入要走的位置　playerMove
4. 连3个棋子就胜利　isWinner
5. 有效判断，平局判断
   isSpaceFree　isBoardFull

THANKS

?

作业

- 4-1 实现一个fibnacci函数，能够高效返回随机数n的fibnacci数列。如
nlist=[randint(1,40) for I in xrange(100)]
[fib(n) for n in nlist]
- 4-2 公交系统的读文件和查找分别改为函数形式。输入起点和终点，返回最少换乘的方案。
- 4-3 tictactoe的改进，增加人机对弈。

思考题

- 4-4 *用wxpython，实现井字棋的图形界面
- 4-5 *用minimax算法提升井字棋的AI。