Python编程基础

Week3 Wanqi

布尔逻辑操作符
(Boolean Operators)

- Python支持：AND, OR, NOT
- True and not False or not True and False

- (True and (not False)) or ((not True) and False)
- NOT: 最高优先权
- AND
- OR: 最低优先权

比较运算符
(Comparison Operations)

- >, <, >=, <=, ==, !=
- 5 <= 5
- 4 != 6
- a = 4
  a == 6

控制结构
(Control Structures)

- 条件语句（Conditional Statements）
- if从句

-
  ```
  if <expression>:
    <suite>
  elif <expression>:
    <suite>
  else:
    <suite>
  ```

```
if a<4:
  print ‘a is less than 4.’
elif a==4:
  print ‘a is equal to 4.’
else
  print ‘a is bigger than 4.’
```

- 循环（Loop）
- while <expression>:

```
    <suite>
```

- for <expression>:

```
    <suite>
```

```
k = 0
while k<5:
    k = k+1
    return k
for k in range(0,5):
    k = k+1
    return k
```

Function & Higher Order Function

- **高阶函数** (Higher Order Function)：以函数作为参数

```
def square(x):
    return x*x
def sum_square(f,a,b):
    total = f(a) + f(b)
    print total
sum_square(square,2,3)
```

Function & Higher Order Function

- 练习：

- 写一个keep_odds函数，输入一个数字后，可以输出该数字范围内的所有正奇数。

- 要求：利用高阶函数

Function & Higher Order Function

- 参考答案：

```
def is_odd(x):
    return x%2 == 1
def keep_odds(f,n):
    i = 0
    while i <= n:
        if f(i):
            print i
        i += 1
```

递归（Recurssion）

- 定义：函数中包含了对自身的调用

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

递归（Recurssion）

- 练习：
- 1. 写一个函数ab_plus_c(a,b,c)，计算ab+c的值。
- 要求：不可以使用*号，使用递归完成。
- 2. 写一个函数is_prime(n)，用于判断n是否为质数。

递归（Recurssion）

- 参考答案1：

```
def ab_plus_c(a,b,c):
    if b == 0:
        return c
    return a + ab_plus_c(a,b-1,c)
```

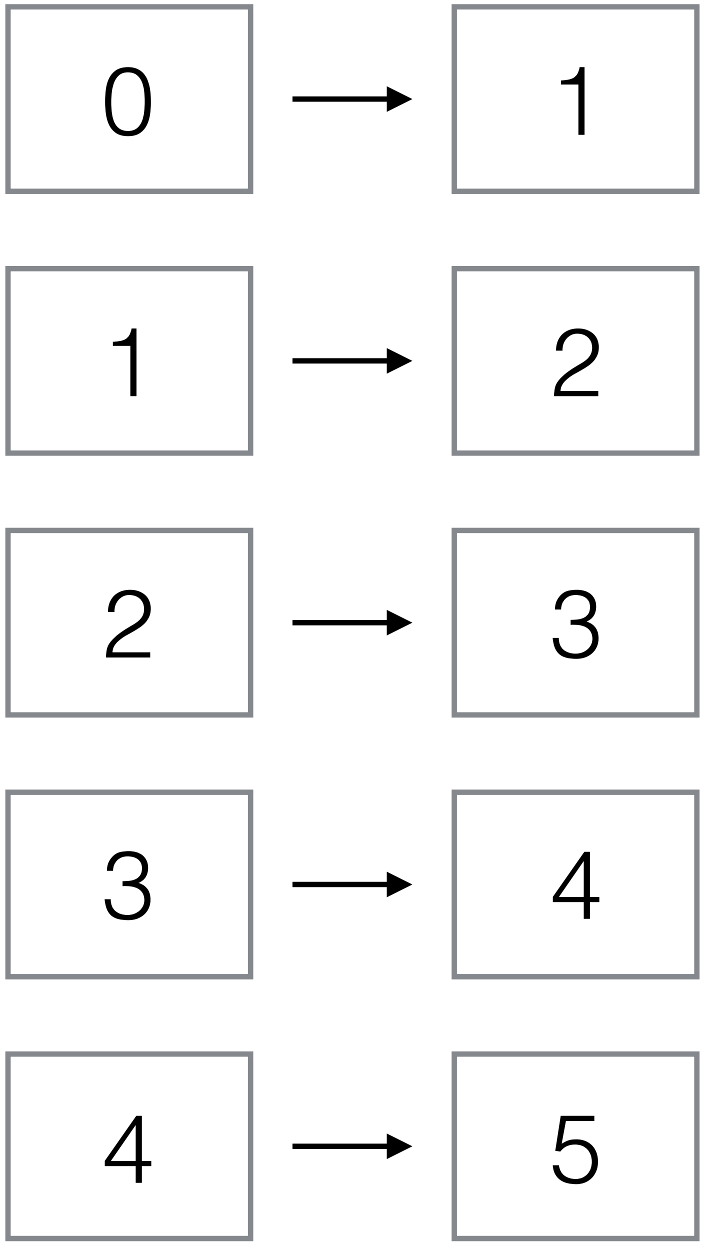
递归（Recurssion）

- 参考答案2：
- def is_prime(n):

```
def is_prime(n):
     def helper(i):
       if i>(n**0.5):
         return True
       elif n%i == 0:
         return False
       return helper(i+1)
   return helper(2)
```

数据类型-列表（list）

- lst = [1,2,3,4,5]
- **index**
- lst[0]
- let [-1]



0 → 1

1 → 2

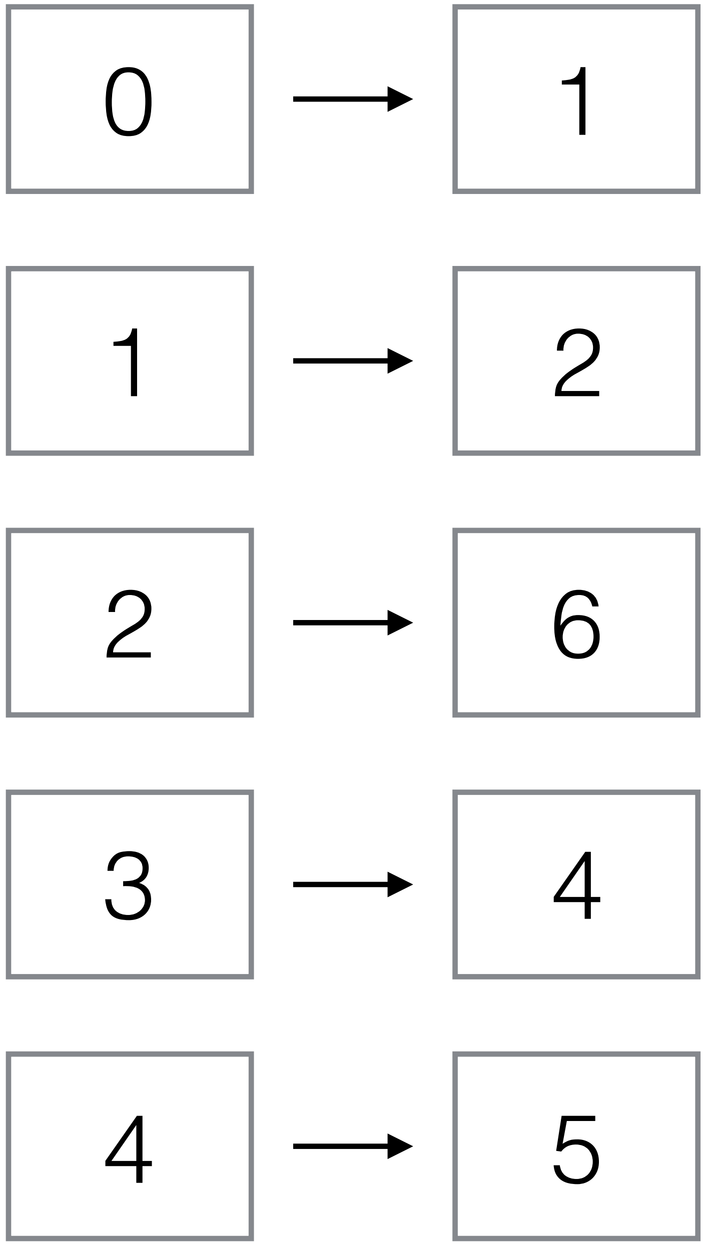
2 → 3

3 → 4

4 → 5

数据类型-列表（list）

- **list implementation**
- lst[2] = 6
- lst
- [1,2,6,4,5]



0 → 1

1 → 2

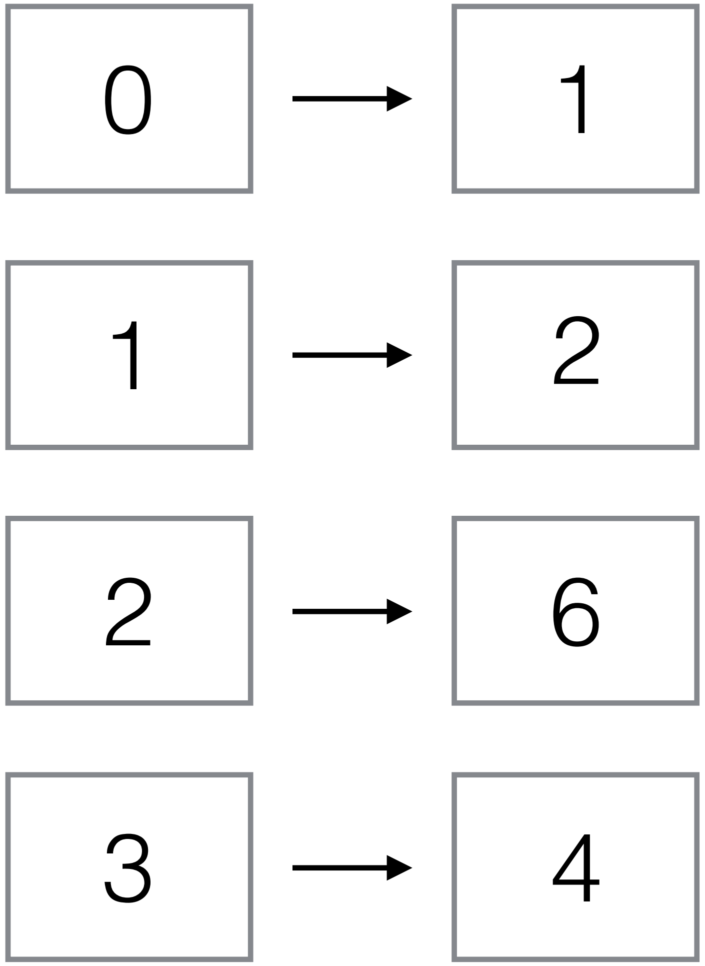
2 → 6

3 → 4

4 → 5

数据类型-列表（list）

- **list implementation**
- lst.pop()
- lst
- [1,2,6,4]



0 → 1

1 → 2

2 → 6

3 → 4

数据类型-列表（list）

- **list implementation**
- a = [i*i for i in range(0,5) if i%2==0]
- b = [i+5 for i in [n for n in range(1,4)]]
- a+b

数据类型-字典（dictionary）

- singers = {‘Adele’ : ‘[Hello’, ‘Skyfall’],

  ‘Taylor’: ‘Love Story’,

  ‘Beyonce’ : ‘Halo’}

- singers.keys()
- singers.values()
- singers[‘Adele’][0]

数据类型-字典（dictionary）

- singers = {‘Adele’ : ‘[Hello’, ‘Skyfall’],

  ‘Taylor’: ‘Love Story’,

  ‘Beyonce’ : ‘Halo’}
- **dictionary implementation**
- singers.keys()
- singers.values()

数据类型-字典（dictionary）

- singers = {‘Adele’ : ‘[Hello’, ‘Skyfall’],

  ‘Taylor’: ‘Love Story’,

  ‘Beyonce’ : ‘Halo’}

- **dictionary implementation**

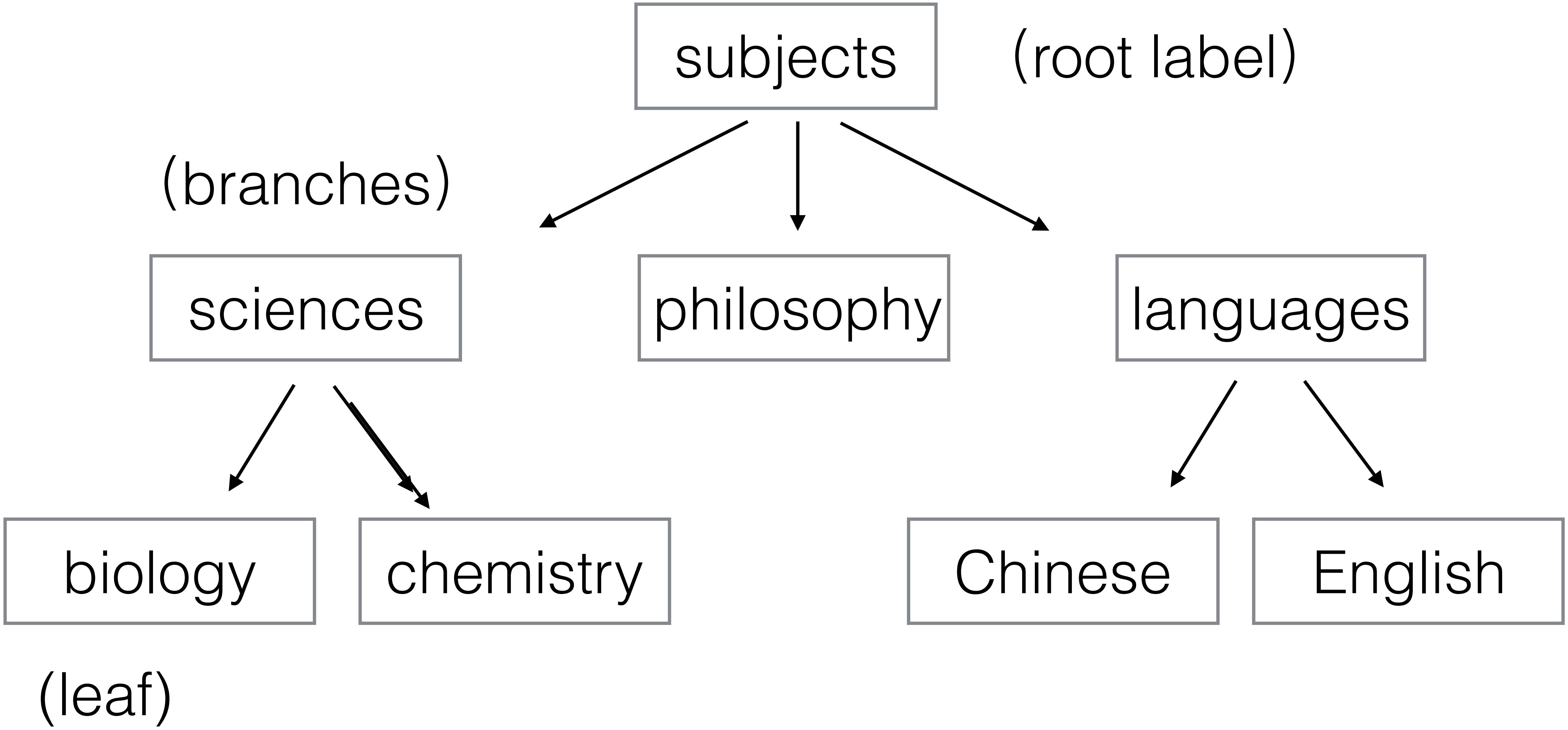
- singers[‘Adele’][0]

- singers[‘Beyonce’] = [‘Halo’, ‘Formation’]

数据类型-字典（dictionary）

- singers = {‘Adele’ : ‘[Hello’, ‘Skyfall’],
  ‘Taylor’: ‘Love Story’,
  ‘Beyonce’ : ‘Halo’}
- **dictionary implementation**
- Adele' in singers
- 'Love Story' in singers

数据类型-树（tree)


subjects
(root label)
(branches)
sciences
philosophy
languages
biology
chemistry
Chinese
English
(leaf)

数据类型-树（tree）

- 嵌套列表
- t = ['subjects',

  ['sciences', ['biology','chemistry']],

  [‘philosophy'],

  ['languages',['Chineses','English']]]

数据类型-树（tree）

- t = ['subjects',

  ```
  ['sciences', ['biology','chemistry']],

  [‘philosophy'],

   ['languages',['Chineses','English']]]
  ```

- t[0]
- t[1][1][0]

作业

- 现在，我们构造一系列关于tree的函数
- # Constructor
- def tree(label, branches=[]):

      return [label] + list(branches)

- # Selectors
- def label(tree):

      return tree[0]

- def branches(tree):

      return tree[1:]

作业

- 因此，我们前面subjects的树就可以改写为：

-
  ```
  t = tree(‘subjects',
                 [tree(‘sciences’,
                 [tree(‘biology’), tree(‘chemistry’)]),
                 tree(‘philosophy’),
          tree(‘languages’),
                  [tree(‘Chineses’), tree(’English’)]])
  ```

作业

- **Q1:** def is_leaf(t):

  *return t is a leaf or not. * （判断t是否为leaf）

- **Q2:** def square_tree(t):

  return a tree with the square of every element in t.

  （t中的每一个数都为原来的平方）

- **Q3:** def height(t):

  return the height of a tree. （计算t的高度）

- **Q4:** def tree_max(t):

  return the max of a tree. （找出t的最大值）