课程内容介绍和准备

Guangrui QIAN

课程安排

Python语言 + 机器学习（Sklearn） + 深度学习（TF+Pytorch）

课程目标

了解基本概念 + 简单动手实践 + 行业介绍

Python语言特性

- Python是一种跨平台的计算机程序设计语言。 是一个高层次的结合了解释性、编译性、互动性和面向对象的脚本语言。最初被设计用于编写自动化脚本(shell)，随着版本的不断更新和语言新功能的添加，越多被用于独立的、大型项目的开发。

- 机器学习领域的主要语言。“不会python，就不会人工智能。”

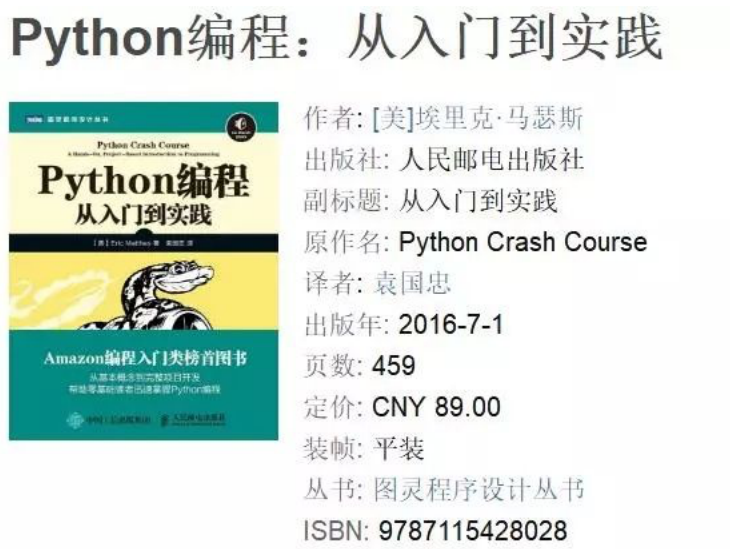
- Python拥有强大的工具包，且免费。

Python编程

Guangrui Qian

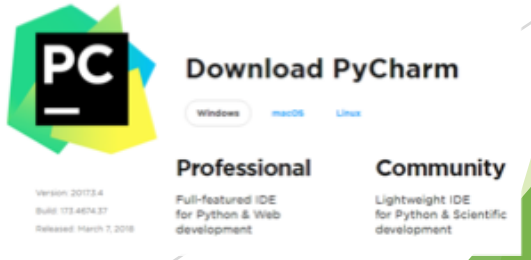
Python学习资料汇总

- Python书籍推荐：



- Python相关工具使用推荐：
  - ANACONDA.
  - Jupiter Notebook

  - Pycharm



- 学习视频：B站

强烈推荐：http://www.allitebooks.org

所有书籍免费下载，无版权问题

Python的安装和版本

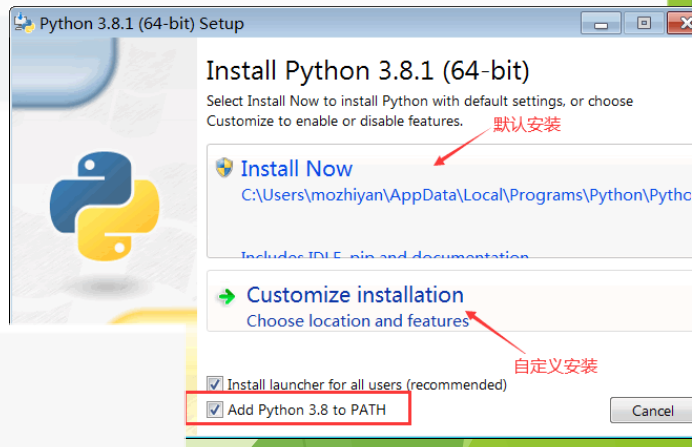
- Python安装：
- Python 安装包下载地址：https://www.python.org/downloads/

- 建议安装方式： Anaconda
- Anaconda安装包下载地址： https://www.anaconda.com

- 3.5以上版本，建议3.6和3.7版本

一定避免使用2.x版本

Release version	Release date
Python 3.5.10	Sept. 5, 2020
Python 3.7.9	Aug. 17, 2020
Python 3.6.12	Aug. 17, 2020
Python 3.8.5	July 20, 2020
Python 3.8.4	July 13, 2020
Python 3.7.8	June 27, 2020
Python 3.6.11	

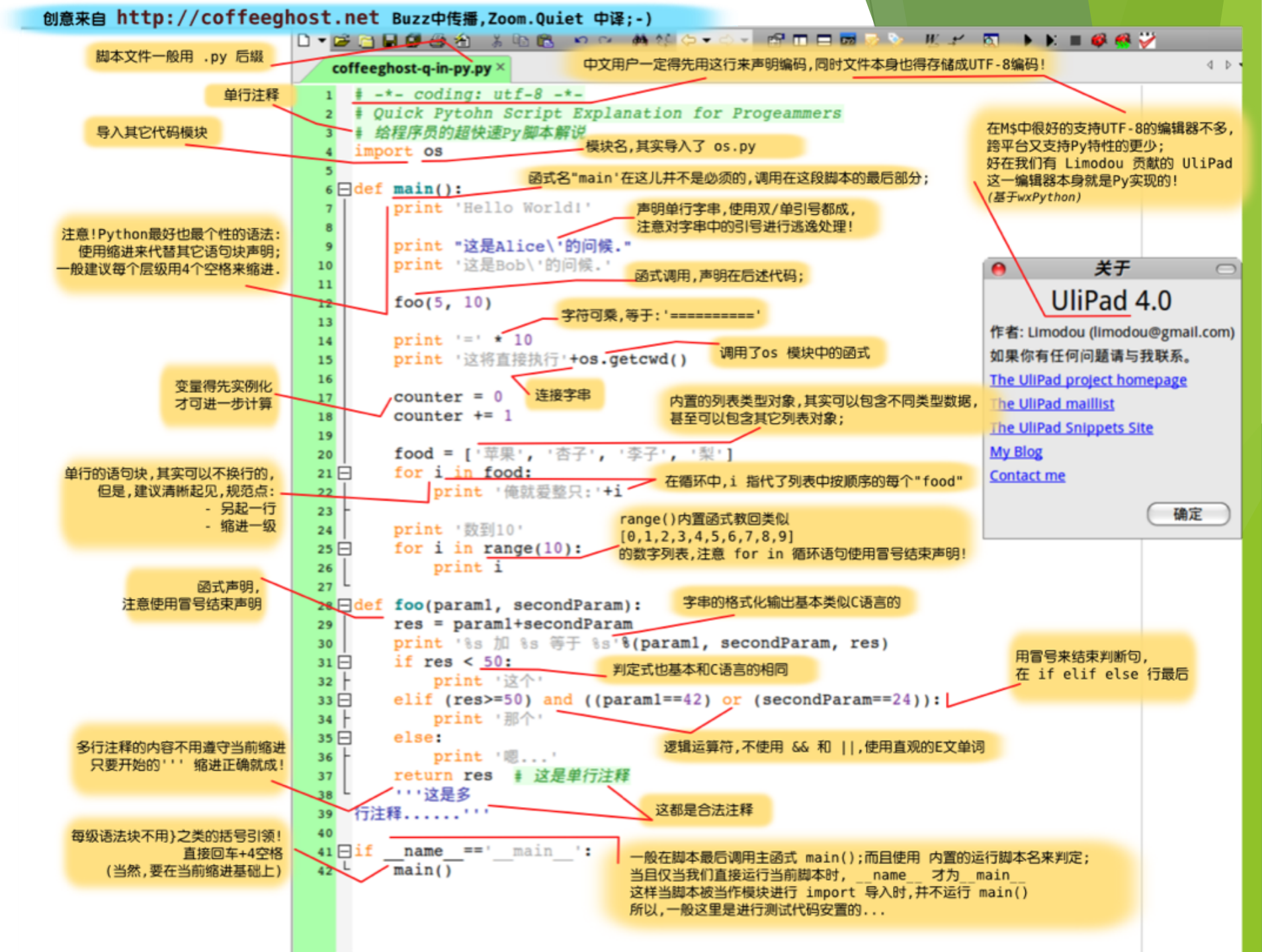



```
print "Hello World!"
print "Hello Again"
print "I like typing this."
print "This is fun."
print 'Yay! Printing.'
print "I'd much rather you 'not'."
print 'I "said" do not touch this.'
```

第一个Python程序

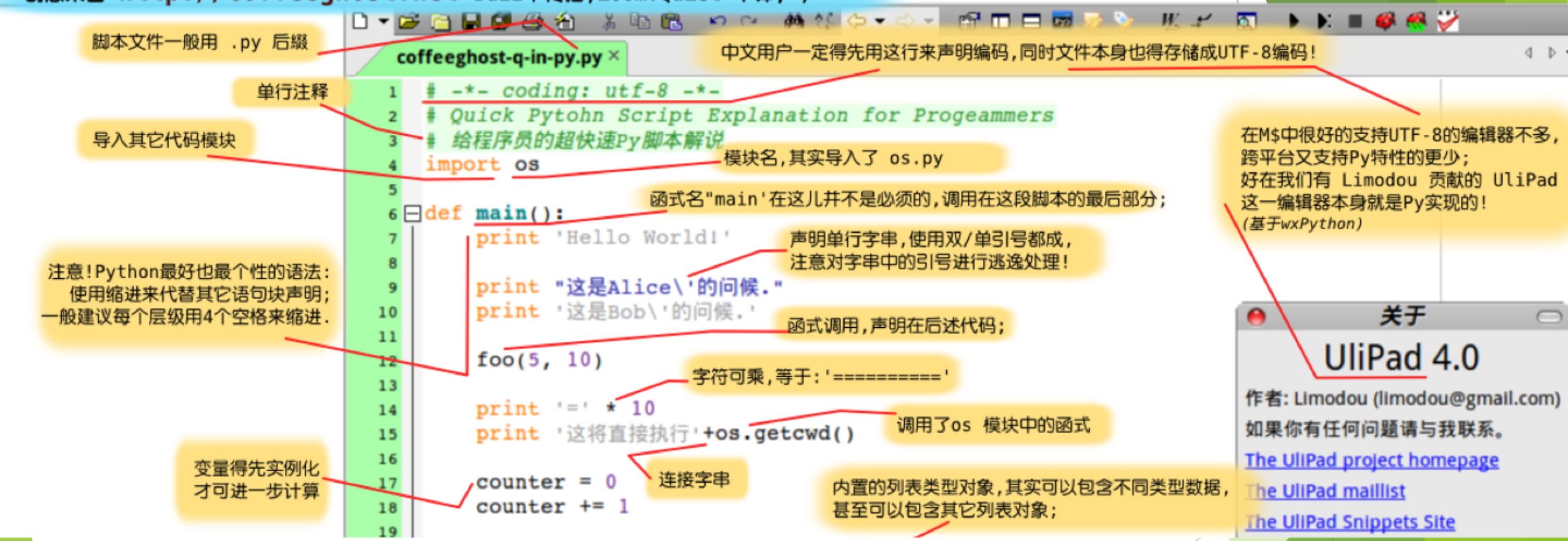
一定避免使用2.x版本

这是一个错误示范

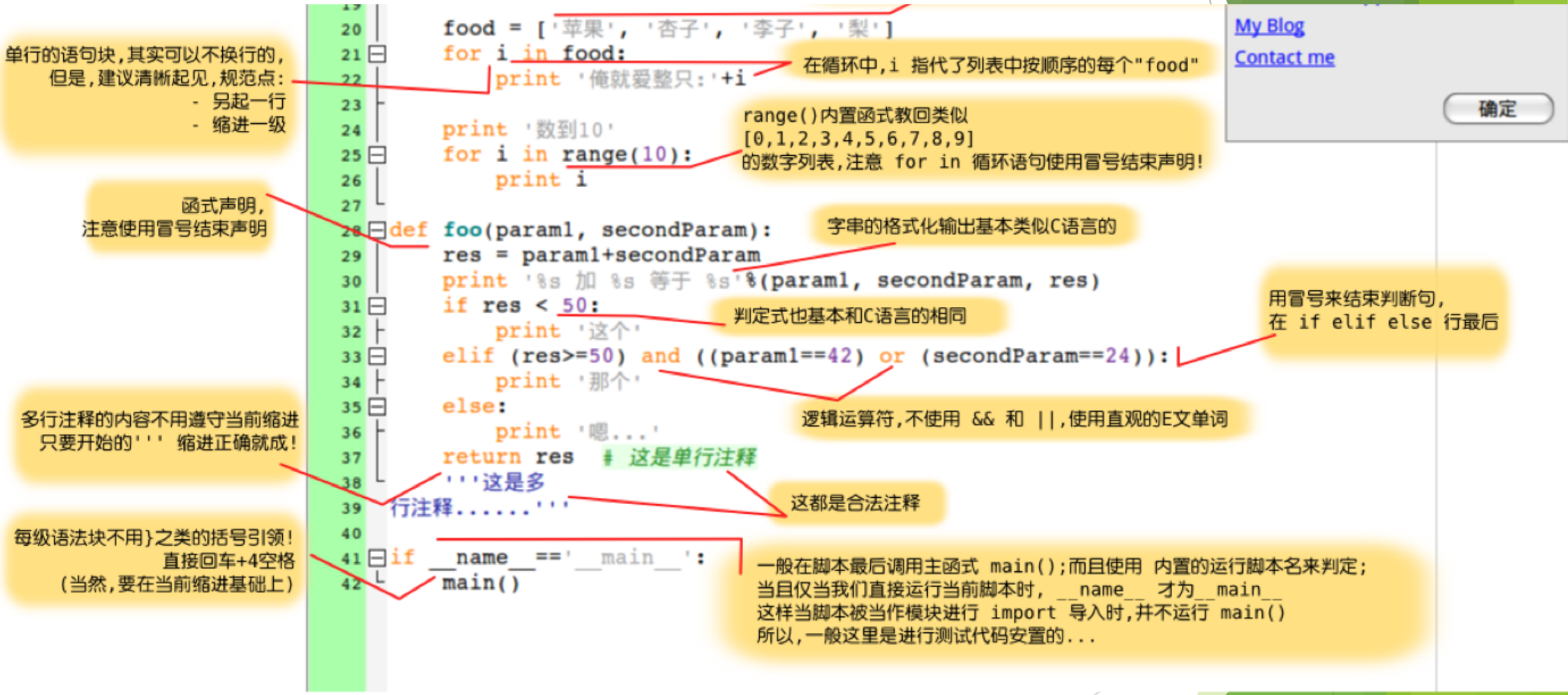



```
# -*- coding: utf-8 -*-
# Quick Pytohn Script Explanation for Progeammers
# 给程序员的超快速Py脚本解说
import os

def main():
    print 'Hello World!'

    print "这是Alice\'的问候."
    print '这是Bob\'的问候.'

    foo(5, 10)

    print '=' * 10
    print '这将直接执行'+os.getcwd()

    counter = 0
    counter += 1

    food = ['苹果', '杏子', '李子', '梨']
    for i in food:
        print '俺就爱整只:'+i

    print '数到10'
    for i in range(10):
        print i

def foo(param1, secondParam):
    res = param1+secondParam
    print '%s 加 %s 等于 %s'%(param1, secondParam, res)
    if res < 50:
        print '这个'
    elif (res>=50) and ((param1==42) or (secondParam==24)):
        print '那个'
    else:
        print '嗯...'
    return res  # 这是单行注释
    '''这是多
行注释.....'''

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

创意来自 http://coffeeghost.net Buzz中传播,Zoom.Quiet 中译;-)




```
1  # -*- coding: utf-8 -*-
2  # Quick Pytohn Script Explanation for Progeammers
3  # 给程序员的超快速Py脚本解说
4  import os
5
6  def main():
7      print 'Hello World!'
8
9      print "这是Alice\'的问候."
10     print '这是Bob\'的问候.'
11
12     foo(5, 10)
13
14     print '=' * 10
15     print '这将直接执行'+os.getcwd()
16
17     counter = 0
18     counter += 1
19
```


单行的语句块,其实可以不换行的,
但是,建议清晰起见,规范点:
- 另起一行
- 缩进一级
函式声明,
注意使用冒号结束声明
多行注释的内容不用遵守当前缩进
只要开始的''' 缩进正确就成!
每级语法块不用}之类的括号引领!
直接回车+4空格
(当然,要在当前缩进基础上)
food = ['苹果', '杏子', '李子', '梨']
for i in food:
print '俺就爱整只:'+i
在循环中,i 指代了列表中按顺序的每个"food"
print '数到10'
for i in range(10):
print i
range()内置函式教回类似
[0,1,2,3,4,5,6,7,8,9]
的数字列表,注意 for in 循环语句使用冒号结束声明!
def foo(param1, secondParam):
res = param1+secondParam
print '%s 加 %s 等于 %s'%(param1, secondParam, res)
字串的格式化输出基本类似C语言的
if res < 50:
print '这个'
判定式也基本和C语言的相同
elif (res>=50) and ((param1==42) or (secondParam==24)):
print '那个'
用冒号来结束判断句,
在 if elif else 行最后
逻辑运算符,不使用 && 和 ||,使用直观的E文单词
else:
print '嗯...'
return res # 这是单行注释
'''这是多
行注释.....'''
这都是合法注释
if __name__=='__main__':
main()
一般在脚本最后调用主函式 main();而且使用 内置的运行脚本名来判定;
当且仅当我们直接运行当前脚本时, __name__ 才为__main__
这样当脚本被当作模块进行 import 导入时,并不运行 main()
所以,一般这里是进行测试代码安置的...
My Blog
Contact me
确定

Python保留字

and	用于表达式运算，逻辑与操作
as	用于类型转换
assert	断言，用于判断变量或条件表达式的值是否为真
break	中断循环语句的执行
class	用于定义类
continue	继续执行下一次循环
def	用于定义函数或方法
del	删除变量或序列的值
elif	条件语句，与if,else结合使用
else	条件语句，与if,elif结合使用，也可用于异常和循环语句
except	except包含捕获异常后的操作代码块，与try,finally结合使用
exec	用于执行python 语句
for	for循环语句
finally	用于异常语句，出现异常后，始终要执行finally，包含的代码块，与try，except结合使用
from	用于导入模块，与import结合使用
globe	定义全局变量
if	条件语句，与else，elif结合使用

import	用于导入模块，与from结合使用
in	判断变量是否在序列中
is	判断变量是否为某个类的实例
lambda	定义匿名变量
not	用于表达式运算，逻辑非操作
or	用于表达式运算，逻辑或操作
pass	空的类，方法，函数的占位符
print	打印语句
raise	异常抛出操作
return	用于从函数返回计算结果
try	try包含可能会出现异常的语句，与except，finally结合使用
while	while的循环语句
with	简化python的语句
yield	用于从函数依此返回值
nonlocal	
false	

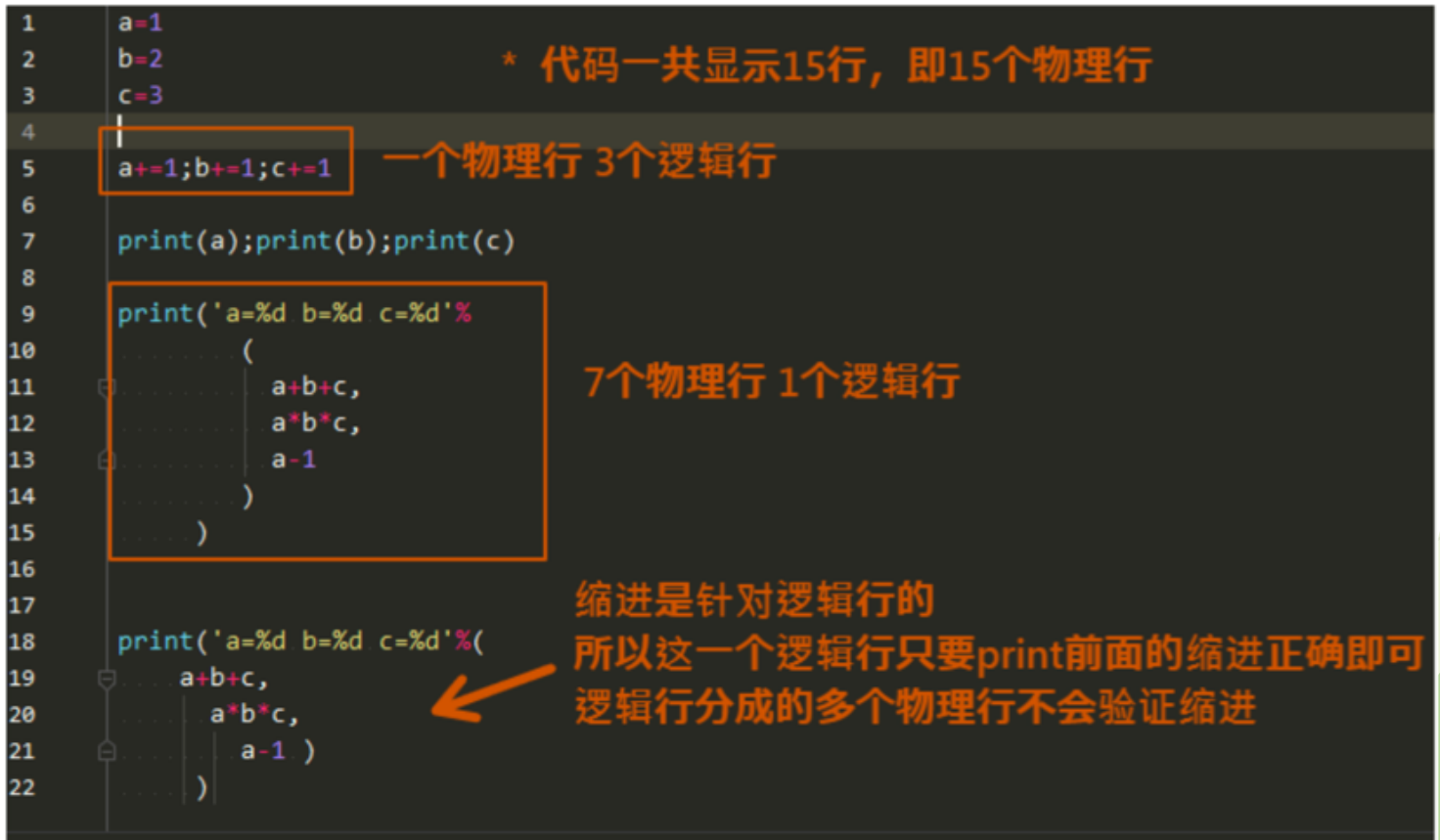
Python代码规范 – 缩进

- Python代码对**缩进**非常敏感
- 特殊情况下tab符和空格做为缩进不能混用
- 缩进规则
  - 逻辑行的 首行需要顶格，即0缩进
  - 相同逻辑层保持相同的缩进
  - ":"标记一个新的逻辑层， 增加缩进进入下一个代码层，减少缩进返回上一个代码层
  - ":"后面未换行，解释器视为一个逻辑行，那么下一行一定不能增加缩进

```
6   a=1
7   if a==1:
8       print(1)      #4个空格缩进
9   else:
10    print(2)        #2个空格缩进
11  k=1
12  while k<3:
13      print(3)      #tab符缩进
14      k+=1
```

Python代码规范 – 缩进

- 物理行：编辑器中显示的代码，每一行内容是一个物理行。
- 逻辑行：Python解释器对代码进行解释，一个语句是一个逻辑行。



Python代码规范 – 缩进

```
In [ ]:  import os
```

```
In [ ]:  a=4
           b=5
```

```
In [ ]:  a=2
         if a==1:print(a); print("print me")
             print(1)
```

```
In [ ]:  a=2
         print("缩进1")

         if a==1:
             print(a)
             print("print me")

         print("缩进2")
         if a==2:
             print(a)
         print("print me")
```

Python Cheat Sheet

Beginner's Python Cheat Sheet

Variables and Strings

Variables are used to store values. A string is a series of characters, surrounded by single or double quotes.

Hello world

```
print("Hello world!")
```

Hello world with a variable

```
msg = "Hello world!"
print(msg)
```

Concatenation (combining strings)

```
first_name = 'albert'
last_name = 'einstein'
full_name = first_name + ' ' + last_name
print(full_name)
```

Lists

A list stores a series of items in a particular order. You access items using an index, or within a loop.

Make a list

```
bikes = ['trek', 'redline', 'giant']
```

Get the first item in a list

```
first_bike = bikes[0]
```

Get the last item in a list

```
last_bike = bikes[-1]
```

Looping through a list

```
for bike in bikes:
    print(bike)
```

Adding items to a list

```
bikes = []
bikes.append('trek')
bikes.append('redline')
bikes.append('giant')
```

Making numerical lists

```
squares = []
for x in range(1, 11):
    squares.append(x**2)
```

Lists (cont.)

List comprehensions

```
squares = [x**2 for x in range(1, 11)]
```

Slicing a list

```
finishers = ['sam', 'bob', 'ada', 'bea']
first_two = finishers[:2]
```

Copying a list

```
copy_of_bikes = bikes[:]
```

Tuples

Tuples are similar to lists, but the items in a tuple can't be modified.

Making a tuple

```
dimensions = (1920, 1080)
```

If statements

If statements are used to test for particular conditions and respond appropriately.

Conditional tests

```
equals            x == 42
not equal         x != 42
greater than      x > 42
  or equal to     x >= 42
less than         x < 42
  or equal to     x <= 42
```

Conditional test with lists

```
'trek' in bikes
'surly' not in bikes
```

Assigning boolean values

```
game_active = True
can_edit = False
```

A simple if test

```
if age >= 18:
    print("You can vote!")
```

If-elif-else statements

```
if age < 4:
    ticket_price = 0
elif age < 18:
    ticket_price = 10
else:
    ticket_price = 15
```

Dictionaries

Dictionaries store connections between pieces of information. Each item in a dictionary is a key-value pair.

A simple dictionary

```
alien = {'color': 'green', 'points': 5}
```

Accessing a value

```
print("The alien's color is " + alien['color'])
```

Adding a new key-value pair

```
alien['x_position'] = 0
```

Looping through all key-value pairs

```
fav_numbers = {'eric': 17, 'ever': 4}
for name, number in fav_numbers.items():
    print(name + ' loves ' + str(number))
```

Looping through all keys

```
fav_numbers = {'eric': 17, 'ever': 4}
for name in fav_numbers.keys():
    print(name + ' loves a number')
```

Looping through all the values

```
fav_numbers = {'eric': 17, 'ever': 4}
for number in fav_numbers.values():
    print(str(number) + ' is a favorite')
```

User input

Your programs can prompt the user for input. All input is stored as a string.

Prompting for a value

```
name = input("What's your name? ")
print("Hello, " + name + "!")
```

Prompting for numerical input

```
age = input("How old are you? ")
age = int(age)

pi = input("What's the value of pi? ")
pi = float(pi)
```

print打印函数

%s　　字符串采用str()的显示
%x　　十六进制整数
%r　　字符串(repr())的显示
%e　　指数（基底写e）
%c　　单个字符
%E　　指数（基底写E）
%b　　二进制整数
%f, %F　浮点数
%d　　十进制整数
%g　　指数(e)或浮点数(根据显示长度)
%i　　十进制整数
%G　　指数(E)或浮点数(根据显示长度)
%o　　八进制整数
%%　　字符%

Variables and Strings

Variables are used to store values. A string is a series of characters, surrounded by single or double quotes.

Hello world

```
print("Hello world!")
```

Hello world with a variable

```
msg = "Hello world!"
print(msg)
```

Concatenation (combining strings)

```
first_name = 'albert'
last_name = 'einstein'
full_name = first_name + ' ' + last_name
print(full_name)
```

print format() 格式化内置函数

- Python2.6 开始，新增了一种格式化字符串的函数 str.format()，它增强了字符串格式化的功能。
- 基本语法是通过 {} 和：来代替以前的 % 。
- format 函数可以接受不限个参数，位置可以不按顺序。

```
print('hello %s' % 'world')
# hello world
print('hello {}'.format('world'))
# hello world
```

print format() 格式化内置函数

- format()不用理会数据类型，%s，%f等记忆复杂；
- format()功能更丰富，填充方式，对齐方式灵活，打印效果更美观；
- format()是官方推荐的，%指不定就在未来版本中给废弃掉了。

```
print('{} {}'.format('hello', 'world'))  # 最基本的

print('{0} {1}'.format('hello', 'world'))  # 通过位置参数

print('{0} {1} {0}'.format('hello', 'world'))  # 单个参数多次输出
```

```
# 通过关键词参数
print('我的名字是{name},我今年{age}岁了。'.format(name='小明', age='12'))

# 与位置参数一样，单个参数也能多次输出
print('{name}说："我的名字是{name},我今年{age}岁了。"'.format(name='小明', age='12'))
```

print format() 格式化内置函数

数字	格式	输出	描述
3.141592	{:.2f}	3.14	保留小数点后两位
3.141592	{:+.2f}	+3.14	带符号保留小数点后两位
3.141592	{:.0f}	3	不带小数，四舍五入
3	{:0>2d}	03	数字补零 (填充左边, 宽度为2)
3	{x<4d}	3xxx	数字补x (填充右边, 宽度为4)
123456789	{:,}	123,456,789	千分位隔开
0.38	{:.2%}	38%	百分比格式
1000000000	{:.2e}	1.00e+09	指数记法
11	{:b}	1011	二进制转化

Python注释

- Python中的注释有单行注释和多行注释
- Python中单行注释以 # 开头
- 行注释用三个单引号 ''' 或者三个双引号 """ 将注释括起来

```
In [ ]: # 这是一个注释
        print("Hello, World!")
```

```
In [ ]: #!/usr/bin/python3
        '''
        这是多行注释，用三个单引号
        这是多行注释，用三个单引号
        这是多行注释，用三个单引号
        '''
        print("Hello, World!")
```

```
In [ ]: #!/usr/bin/python3
        """
        这是多行注释，用三个双引号
        这是多行注释，用三个双引号
        这是多行注释，用三个双引号
        """
        print("Hello, World!")
```

Python数据基本类型

- Number（数字）
- String（字符串）
- List（列表）
- Tuple（元组）
- Set（集合）
- Dictionary（字典）

Lists

A list stores a series of items in a particular order. You access items using an index, or within a loop.

Make a list

```
bikes = ['trek', 'redline', 'giant']
```

Get the first item in a list

```
first_bike = bikes[0]
```

Get the last item in a list

```
last_bike = bikes[-1]
```

Looping through a list

```
for bike in bikes:
    print(bike)
```

Adding items to a list

```
bikes = []
bikes.append('trek')
bikes.append('redline')
bikes.append('giant')
```

Making numerical lists

```
squares = []
for x in range(1, 11):
    squares.append(x**2)
```

Tuples

Tuples are similar to lists, but the items in a tuple can't be modified.

Making a tuple

```
dimensions = (1920, 1080)
```

Dictionaries

Dictionaries store connections between pieces of information. Each item in a dictionary is a key-value pair.

A simple dictionary

```
alien = {'color': 'green', 'points': 5}
```

Accessing a value

```
print("The alien's color is " + alien['color'])
```

Adding a new key-value pair

```
alien['x_position'] = 0
```

Looping through all key-value pairs

```
fav_numbers = {'eric': 17, 'ever': 4}
for name, number in fav_numbers.items():
    print(name + ' loves ' + str(number))
```

Looping through all keys

```
fav_numbers = {'eric': 17, 'ever': 4}
for name in fav_numbers.keys():
    print(name + ' loves a number')
```

Looping through all the values

```
fav_numbers = {'eric': 17, 'ever': 4}
for number in fav_numbers.values():
    print(str(number) + ' is a favorite')
```

Python字符串

- 字符串是 Python 中最常用的数据类型。我们可以使用引号(' 或 ")来创建字符串。
- Python 不支持单字符类型，单字符在 Python 中也是作为一个字符串使用。
- Python 支持格式化字符串的输出 。

操作符	描述	实例
+	字符串连接	>>>a + b 'HelloPython'
*	重复输出字符串	>>>a * 2 'HelloHello'
[]	通过索引获取字符串中字符	>>>a[1] 'e'
[:]	截取字符串中的一部分	>>>a[1:4] 'ell'
in	成员运算符 - 如果字符串中包含给定的字符返回 True	>>>"H" in a True
not in	成员运算符 - 如果字符串中不包含给定的字符返回 True	>>>"M" not in a True

r/R	原始字符串 - 原始字符串：所有的字符串都是直接按照字面的意思来使用，没有转义特殊或不能打印的字符。 原始字符串除在字符串的第一个引号前加上字母"r"（可以大小写）以外，与普通字符串有着几乎完全相同的语法。	>>> print(r'\n')
		\n
		>>> print(R'\n')
		\n

Python 的字符串内建函数

序号	方法及描述
1	capitalize() 将字符串的第一个字符转换为大写
2	center(width, fillchar) 返回一个指定的宽度 width 居中的字符串，fillchar 为填充的字符，默认为空格。
3	count(str, beg= 0,end=len(string)) 返回 str 在 string 里面出现的次数，如果 beg 或者 end 指定则返回指定范围内 str 出现的次数
4	bytes.decode(encoding="utf-8", errors="strict") Python3 中没有 decode 方法，但我们可以使用 bytes 对象的 decode() 方法来解码给定的 bytes 对象，这个 bytes 对象可以由 str.encode() 来编码返回。
5	encode(encoding='UTF-8',errors='strict') 以 encoding 指定的编码格式编码字符串，如果出错默认报一个ValueError 的异常，除非 errors 指定的是'ignore'或者'replace'
6	endswith(suffix, beg=0, end=len(string)) 检查字符串是否以 obj 结束，如果beg 或者 end 指定则检查指定的范围内是否以 obj 结束，如果是，返回 True,否则返回 False.

https://www.runoob.com/python3/python3-string.html

Python的数学运算符

以下假设变量a为10，变量b为21：

运算符	描述	实例
+	加 - 两个对象相加	a + b 输出结果 31
-	减 - 得到负数或是一个数减去另一个数	a - b 输出结果 -11
*	乘 - 两个数相乘或是返回一个被重复若干次的字符串	a * b 输出结果 210
/	除 - x 除以 y	b / a 输出结果 2.1
%	取模 - 返回除法的余数	b % a 输出结果 1
**	幂 - 返回x的y次幂	a**b 为10的21次方

//	取整除 - 向下取接近商的整数	`>>> 9//2`
		`4`
		`>>> -9//2`
		`-5`

Python列表

- 序列是Python中最基本的数据结构。序列中的每个元素都分配一个数字 - 它的位置，或索引，第一个索引是0，第二个索引是1，依此类推。
- Python有6个序列的内置类型，但最常见的是列表和元组。
- 序列都可以进行的操作包括索引，切片，加，乘，检查成员。
- Python已经内置确定序列的长度以及确定最大和最小的元素的方法。
- 列表是最常用的Python数据类型，它可以作为一个方括号内的逗号分隔值出现。
- 列表的数据项不需要具有相同的类型
- 创建一个列表，只要把逗号分隔的不同的数据项使用方括号括起来即可。

创建列表

```
In [ ]: list1 = ['Google', 'Runoob', 1997, 2000]
        list2 = [1, 2, 3, 4, 5 ]
        list3 = ["a", "b", "c", "d"]
```

更新列表

```
In [ ]: list = ['Google', 'Runoob', 1997, 2000]
        print ("第三个元素为 : ", list[2])
        list[2] = 2001
        print ("更新后的第三个元素为 : ", list[2])
```

删除列表元素

```
In [ ]: list = ['Google', 'Runoob', 1997, 2000]
        print ("原始列表 : ", list)
        del list[2]
        print ("删除第三个元素 : ", list)
```

列表操作

Python 表达式	结果	描述
len([1, 2, 3])	3	长度
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	组合
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	重复
3 in [1, 2, 3]	True	元素是否存在于列表中
for x in [1, 2, 3]: print(x, end=" ")	1 2 3	迭代

Python元组和字典

- Python 的元组与列表类似，不同之处在于元组的元素不能修改。
- 元组使用小括号，列表使用方括号
- 元组中的元素值是不允许删除的，但我们可以使用del语句来删除整个元组

```
In [ ]: tup1 = ('Google', 'Runoob', 1997, 2000)
        tup2 = (1, 2, 3, 4, 5 )
        tup3 = "a", "b", "c", "d"   #  不需要括号也可以
        type(tup3)
```

tup1[0] = 100是非法的

- 字典是另一种可变容器模型，且可存储任意类型对象。
- 字典的每个键值 **key=>value** 对用冒号 **:** 分割，每个对之间用逗号(**,**)分割，整个字典包括在花括号 **{}** 中 。

```
In [ ]: dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
        dict['Alice']
```

Python 流程控制

- if 语句
  - 关键字 'elif' 是 'else if' 的缩写，这个可以有效地避免过深的缩进
- for 语句
- while语句
- range() 函数
  - 打印一个序列： print(range(10))
  - 从可迭代（对象）中创建列表:list(range(5))
- break 和 continue 语句
- pass 语句

code

```
1  a = 1
2  while a < 7 :
3      if(a % 2 == 0):
4          print(a, "is even")
5      else:
6          print(a, "is odd")
7      a += 1
```

output

variables

```
In [ ]: for i in range(10):
            print(i, end=',')
```

Python函数

- 函数代码块以 **def** 关键词开头，后接函数标识符名称和圆括号 **()**。
- 任何传入参数和自变量必须放在圆括号中间，圆括号之间可以用于定义参数。
- 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。
- 函数内容以冒号起始，并且缩进。
- **return [表达式]** 结束函数，选择性地返回一个值给调用方。不带表达式的return相当于返回 None。

```
def 函数名（参数列表）：
    函数体
```

```python
# 计算面积函数

def area(width=1, height=2):
    return width * height

def print_welcome(name):
    print("Welcome", name)

print_welcome("Runoob")

#case 1
print("width = None", " height = None",  " area =", area())

#case 2

w = 4
h = 5
print("width =", w, " height =", h, " area =", area(w, h))
```

Python面向对象- 类

- **类(Class):** 用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例。
- **方法**：类中定义的函数。
- **类变量**：类变量在整个实例化的对象中是公用的。类变量定义在类中且在函数体之外。类变量通常不作为实例变量使用。
- **数据成员**：类变量或者实例变量用于处理类及其实例对象的相关的数据。
- **局部变量**：定义在方法中的变量，只作用于当前实例的类。
- **实例变量**：在类的声明中，属性是用变量来表示的，这种变量就称为实例变量，实例变量就是一个用 self 修饰的变量。
- **实例化**：创建一个类的实例，类的具体对象。
- **对象**：通过类定义的数据结构实例。对象包括两个数据成员（类变量和实例变量）和方法。

```python
#类定义
class people(object):
    #定义基本属性
    nationality = 'Chinese'

    __weight = 0
    #定义构造方法
    def __init__(self,n,a,w):
        self.name = n
        self.age = a
        self.__weight = w #定义私有属性,在类外部无法直接进行访问
    def speak(self):
        print("%s 说: 我 %d 岁。" %(self.name,self.age))
        print("my weight is {}".format(self.__weight))
        print("I am {}".format(self.nationality))

# 实例化类
p = people('runoob',10,30)
print(p.name)
p.speak()
#print(p.__weight)
```

PEP8 Python 编码规范 - 1

- **代码编排:**
  - **缩进：**使用 4 空格缩进，而非 TAB。4个空格的缩进（编辑器都可以完成此功能），不使用Tap，更不能混合使用Tap和空格。
  - **折行：**每行最大长度79，换行可以使用反斜杠，最好使用圆括号。换行点要在操作符的后边敲回车。
  - **空行：**使用空行分隔函数和类，以及函数中的大块代码。类和top-level函数定义之间空两行；类中的方法定义之间空一行；函数内逻辑无关段落之间空一行；其他地方尽量不要再空行。

PEP8 Python 编码规范 - 2

- **空格的使用:**
  - 各种右括号前不要加空格。
  - 逗号、冒号、分号前不要加空格。
  - 函数的左括号前不要加空格。如Func(1)。
  - 序列的左括号前不要加空格。如list[2]。
  - 操作符左右各加一个空格，不要为了对齐增加空格。
  - 函数默认参数使用的赋值符左右省略空格。
  - 不要将多句语句写在同一行，尽管使用‘;’。
  - if/for/while语句中，即使执行语句只有一句，也必须另起一行。

总体原则，避免不必要的空格。

PEP8 Python 编码规范 - 3

- **注释：**
  - 块注释：在一段代码前增加的注释。在‘#’后加一空格。段落之间以只有‘#’的行间隔。
  - 行注释：在一句代码后加注释。比如：**x = x + 1 # Increment x**（这种方式尽量少使用。）
  - 避免无谓的注释。

- 总体原则，错误的注释不如没有注释。所以当一段代码发生变化时，第一件事就是要修改注释！
- 注释可使用英文、中文，最好是完整的句子。如果是英文，首字母大写；句后要有结束符，结束符后跟两个空格，开始下一句。如果是短语，可以省略结束符。

PEP8 Python 编码规范 - 4

- 命名规范
  - 尽量单独使用小写字母‘l’，大写字母‘O’等容易混淆的字母。
  - 模块命名尽量短小，使用全部小写的方式，可以使用下划线。
  - 包命名尽量短小，使用全部小写的方式，不可以使用下划线。
  - 类的命名使用CapWords的方式，模块内部使用的类采用_CapWords的方式。
  - 异常命名使用CapWords+Error后缀的方式。
  - 全局变量尽量只在模块内有效，类似C语言中的static。实现方法有两种，一是__all__机制;二是前缀一个下划线。
  - 函数命名使用全部小写的方式，可以使用下划线。
  - 常量命名使用全部大写的方式，可以使用下划线。

总体原则，新编代码必须按下面命名风格进行，现有库的编码尽量保持风格。

PEP8 Python 编码规范 - 5

- **命名规范**
  - 类的属性（方法和变量）命名使用全部小写的方式，可以使用下划线。
  - 类的属性有3种作用域public、non-public和subclass API，可以理解成C++中的public、private、protected，non-public属性前，前缀一条下划线。
  - 类的属性若与关键字名字冲突，后缀一下划线，尽量不要使用缩略等其他方式。
  - 为避免与子类属性命名冲突，在类的一些属性前，前缀两条下划线。比如：类Foo中声明__a,访问时，只能通过Foo._Foo__a，避免歧义。如果子类也叫Foo，那就无能为力了。
  - 类的方法第一个参数必须是self，而静态方法第一个参数必须是cls。

总体原则，新编代码必须按下面命名风格进行，现有库的编码尽量保持风格。

写出优雅的代码

- 参考pyguide.pdf中chapter2
- 版本控制
- 单元测试

编码建议

- 尽可能使用‘is’, ‘is not’取代‘==’，比如if x is not None 要优于if x。
- 判断序列空或不空，有如下规则

  Yes: if not seq:　　No: if len(seq)

  　　if seq:　　　　if not len(seq)
- 字符串不要以空格收尾
- 异常中try的代码尽可能少

错误和异常 - 1

- **语法错误**，也被称作解析错误，是学习 Python 过程中最常见抱怨:

```
>>> while True print('Hello world')
  File "<stdin>", line 1, in ?
    while True print('Hello world')
                   ^
SyntaxError: invalid syntax
```

- 语法分析器指出错误行，并且在检测到错误的位置前面显示一个小“箭头”。错误是由箭头 *前面* 的标记引起的（或者至少是这么检测的）：这个例子中，函数 **print()** 被发现存在错误，因为它前面少了一个冒号（ **':'** ）。错误会输出文件名和行号，所以如果是从脚本输入就知道去哪里检查错误了。

Python 中（至少）有两种错误：语法错误和异常（ ***syntax errors*** 和 ***exceptions*** ）

错误和异常 - 2

- 即使一条语句或表达式在语法上是正确的，当试图执行它时也可能会引发错误。
- 运行期检测到的错误称为**异常**，并且程序不会无条件的崩溃。大多数异常都不会被程序处理，如下所示最终会产生一个错误信息：

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: int division or modulo by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: Can't convert 'int' object to str implicitly
```

示例中的异常分别为 零除错误（ ZeroDivisionError ），
命名错误（ NameError ）和 类型错误（ TypeError ）。

Python异常类

异常	描述
NameError	尝试访问一个没有申明的变量
ZeroDivisionError	除数为0
SyntaxError	语法错误
IndexError	索引超出序列范围
KeyError	请求一个不存在的字典关键字
IOError	输入输出错误（比如你要读的文件不存在）
AttributeError	尝试访问未知的对象属性
ValueError	传给函数的参数类型不正确，比如给int()函数传入字符串形

异常处理：示例

```
>>> while True:
...     try:
...         x = int(input("Please enter a number: "))
...         break
...     except ValueError:
...         print("Oops!  That was no valid number.  Try again...")
...
```

它会一直要求用户输入，直到输入一个合法的整数为止，但允许用户中断这个程序（使用 Control-C 或系统支持的任何方法）。

异常处理：Try语句

- **try** 语句按如下方式工作
  - 首先，执行 ***try*** 子句（在 **try** 和 **except** 关键字之间的部分）。
  - 如果没有异常发生， ***except*** 子句 在 **try** 语句执行完毕后就被忽略了。
  - 如果在 **try** 子句执行过程中发生了异常，那么该子句其余的部分就会被忽略。
  - 如果异常匹配于 **except** 关键字后面指定的异常类型，就执行对应的 **except**子句。然后继续执行 **try** 语句之后的代码。
  - 如果发生了一个异常，在 **except** 子句中没有与之匹配的分支，它就会传递到上一级 **try** 语句中。
  - 如果最终仍找不到对应的处理语句，它就成为一个 *未处理异常*，终止程序运行，显示提示信息。

异常处理注意事项(异常捕获)

- 最后一个 except 子句可以省略异常名称，以作为通配符使用。你需要慎用此法，因为它会轻易隐藏一个实际的程序错误！
- try … except 语句可以带有一个 *else 子句*，该子句只能出现在所有 except 子句之后。当 try 语句没有抛出异常时，需要执行一些代码，可以使用这个子句。 使用 else 子句比在 try 子句中附加代码要好，因为这样可以避免 try … except 意外的截获本来不属于它们保护的那些代码抛出的异常。

```
for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
    except IOError:
        print('cannot open', arg)
    else:
        print(arg, 'has', len(f.readlines()), 'lines')
        f.close()
```

就是当没有检测到异常的时候，则执行**else**语句。

异常处理注意事项(Finally)

- finally子句是无论是否检测到异常，都会执行的一段代码。我们可以丢掉except子句和else子句，单独使用try...finally，也可以配合except等使用。

```python
>>> import syslog
>>> try:
...     f = open("/root/test.py")
... except IOError,e:
...     syslog.syslog(syslog.LOG_ERR,"%s"%e)
... else:
...     syslog.syslog(syslog.LOG_INFO,"no exception caught\n")
... finally:
>>>     f.close()
```

抛出异常

- raise 语句允许程序员强制抛出一个指定的异常。
- 要抛出的异常由 raise 的唯一参数标识。它必需是一个异常实例或异常类（继承自 Exception 的类）。

```
>>> raise NameError('HiThere')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: HiThere
```

课后作业

- 使用python3构造函数: 矩阵乘、转置、矩阵求逆
  - 不使用numpy等库，只使用python自带标准库
  - 需要使用类
  - 需要使用单元测试
  - 需要使用错误和异常
- 使用python3构造最小二乘法代码

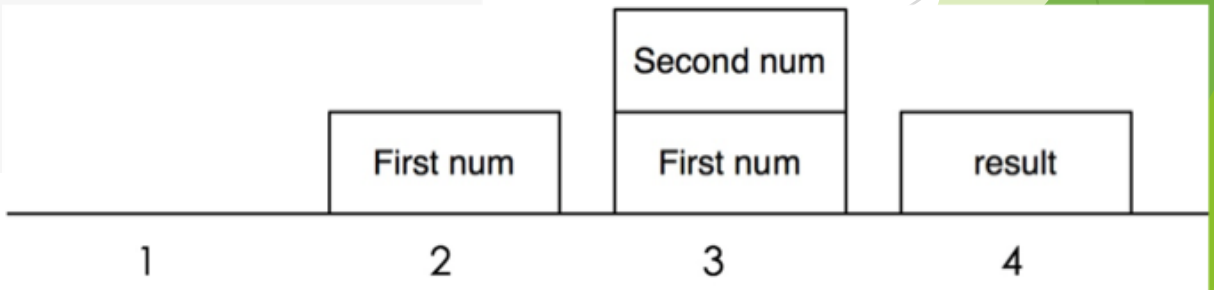
谢谢

Python 解释器

- 有时候我们会把Python的REPL(命令行下Python的交互环境)当作解释器
- 有时候Python解释器这一说法可以指代整个Python，它会将源代码编译为字节码并执行

- Python解释器是一个模拟堆栈机器的虚拟机，仅使用多个栈来完成操作。解释器所处理的字节码来自于对源代码进行词法分析、语法分析和编译后所生成的code object中的指令集合。它相当于Python代码的一个中间层表示，好比汇编代码之于C代码。

```
what_to_execute = {
    "instructions": [("LOAD_VALUE", 0),  # 第一个数
                     ("LOAD_VALUE", 1),  # 第二个数
                     ("ADD_TWO_VALUES", None),
                     ("PRINT_ANSWER", None)],
    "numbers": [7, 5] }
```



Python源程序编码

- 默认情况下，Python 源文件是 UTF-8 编码。在此编码下，全世界大多数语言的字符可以同时用在字符串、标识符和注释中 – 尽管 Python 标准库仅使用 ASCII 字符做为标识符，这只是任何可移植代码应该遵守的约定。如果要正确的显示所有的字符，编辑器必须能识别出文件是 UTF-8 编码，并且它使用的字体能支持文件中所有的字符。

- 也可以为源文件指定不同的字符编码。为此，在 #! 行（首行）后插入至少一行特殊的注释行来定义源文件的编码:

  # -- coding: encoding -*-*

Python语法常用注意点- 字符串

- 字符串
  - Python 可以用单引号 (‘...’) 或双引号 (“...”) 标识的不同方式表示的字符串。 与其它语言不同，特殊字符例如 \n 在单引号('...')和双引号("...")中具有相同的含义。两者唯一的区别是在单引号中，不需要转义 " （但必须转义 \' ），反之亦然。
  - 如果前面带有 \ 的字符被当作特殊字符，可以使用 *原始字符串*，方法是在第一个引号前面加上一个 r。
  - 字符串文本能够分成多行。一种方法是使用三引号："""..."""或者 '''...'''。行尾换行符会被自动包含到字符串中，但是可以在行尾加上 \ 来避免这个行为。下面的示例： 可以使用反斜杠为行结尾的连续字符串，它表示下一行在逻辑上是本行的后续内容:
  - Python 能够优雅地处理那些没有意义的切片索引：一个过大的索引值(即下标值大于字符串实际长度)将被字符串实际长度所代替，当上边界比下边界大时(即切片左值大于右值)就返回空字符串。
  - Python字符串不可以被更改 — 它们是 **不可变的** 。因此，赋值给字符串索引的位置会导致错误。