第0章 课程介绍

李宇 特聘助理教授

东北大学-计算机学院-智慧系统实验室

liyu@cse.neu.edu.cn

个人简介

- 李宇
  - 计算机科学与工程学院-智慧系统实验室（南湖校区，易购大厦3楼）
    - 实验室研主要究方向：可更新CPS系统、智慧医疗、微视觉、声学
  - 个人研究方向：可更新CPS系统设计理论研究、EDA工具研发静态程序分析、网络安全
  - QQ：705120314
  - 邮箱：liyu@cse.neu.edu.cn

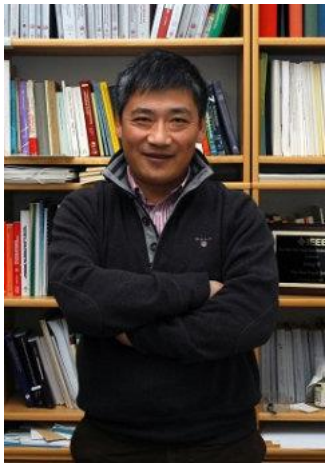
实验室介绍

东北大学智慧系统实验室隶属于计算机科学与工程学院，成立于2017年11月，依托于计算机应用技术国家重点学科，计算机科学与技术专业，创办于1958年，1981年在国内首批获得计算机应用博士授权点，目前具有国家一级博士学位授予权和博士后工作流动站。

实验室按照国际惯例管理和运行，力争打造东北大学科研特区，实验室依托于计算机软件国家工程研究中心、**医学影像智能计算教育部重点实验室**进行建设，坐落在南湖区，主要围绕**“信息物理系统”**、**“智慧医疗”**等方向开展原创性理论研究和特色鲜明的应用技术开发。

实验室目前有国家**“千人计划”**入选者2人、国家**“外专千人计划”**入选者2人、**“长江学者奖励计划”**特聘教授1人、青年骨干教师10余人。

实验室介绍-CPS方向



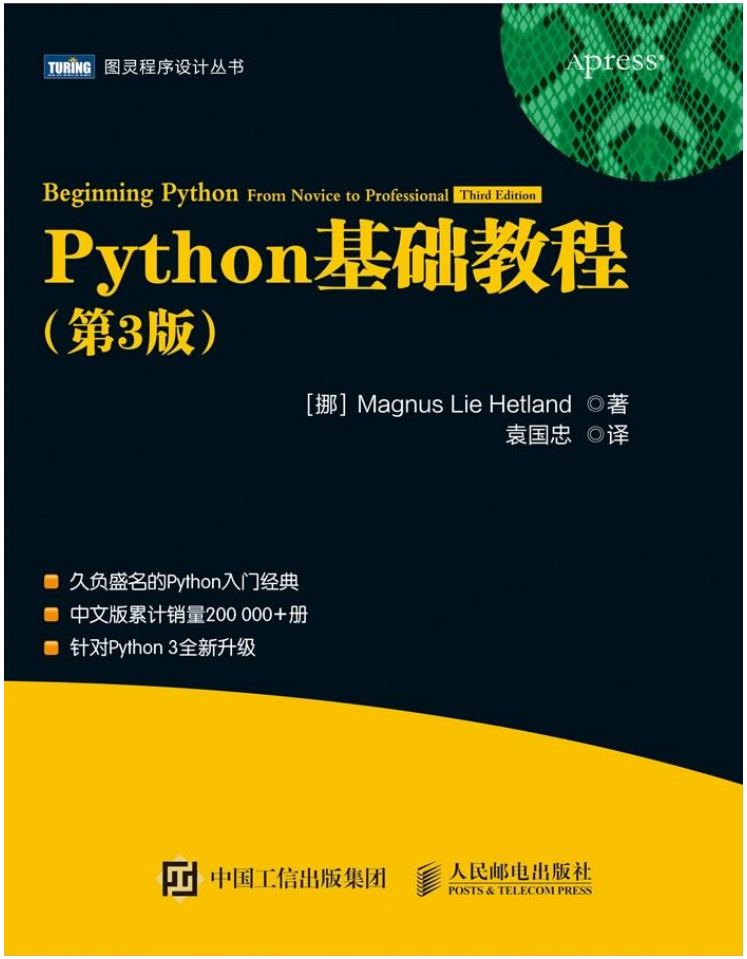
王义

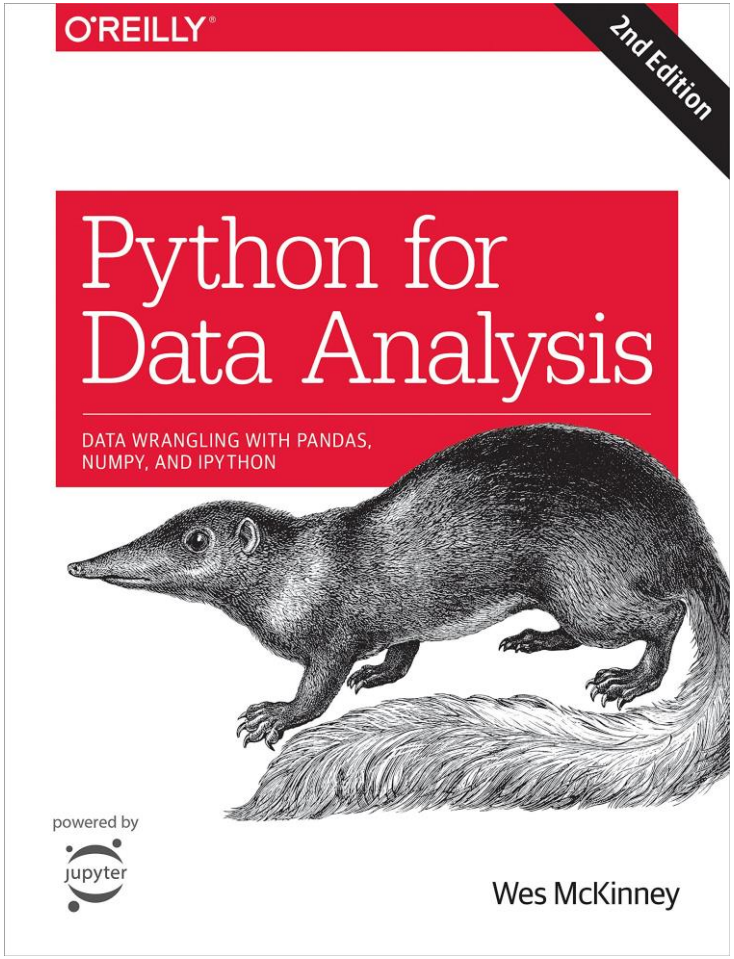
- 瑞典乌普萨拉大学讲席教授，欧洲科学院院士
- 东北大学计算机科学与工程学院院长，千人计划特聘教授，长江学者奖励计划特聘教授
- 著名实时系统验证工具UPPAAL系统的创始人；
- 实时系统调度理论权威专家；
- 欧盟顶级科研基金ERC项目获得者
- 2013年获得CAV奖
- 2019年获得IEEE TCRTS杰出贡献与领导奖

Kim G Larsen

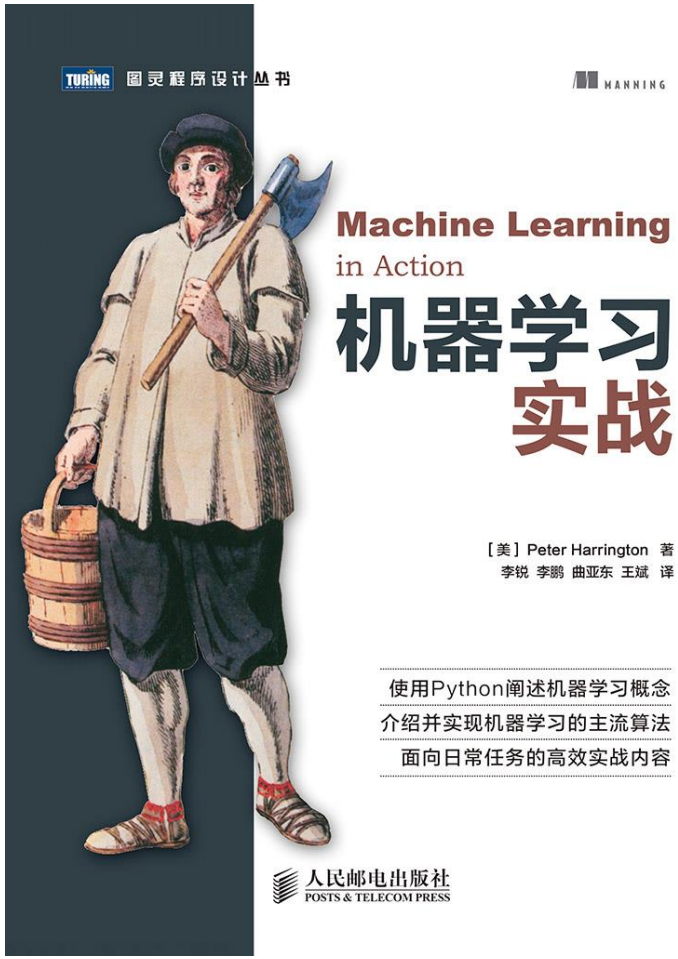
- 丹麦奥尔堡大学教授，欧洲科学院院士，丹麦皇家科学院与工程院院士
- 东北大学外专千人特聘教授
- 著名实时系统验证工具UPPAAL系统的创始人；
- 实时系统形式化验证理论权威专家；
- 欧盟顶级科研基金ERC项目获得者
- 2013年获得CAV奖

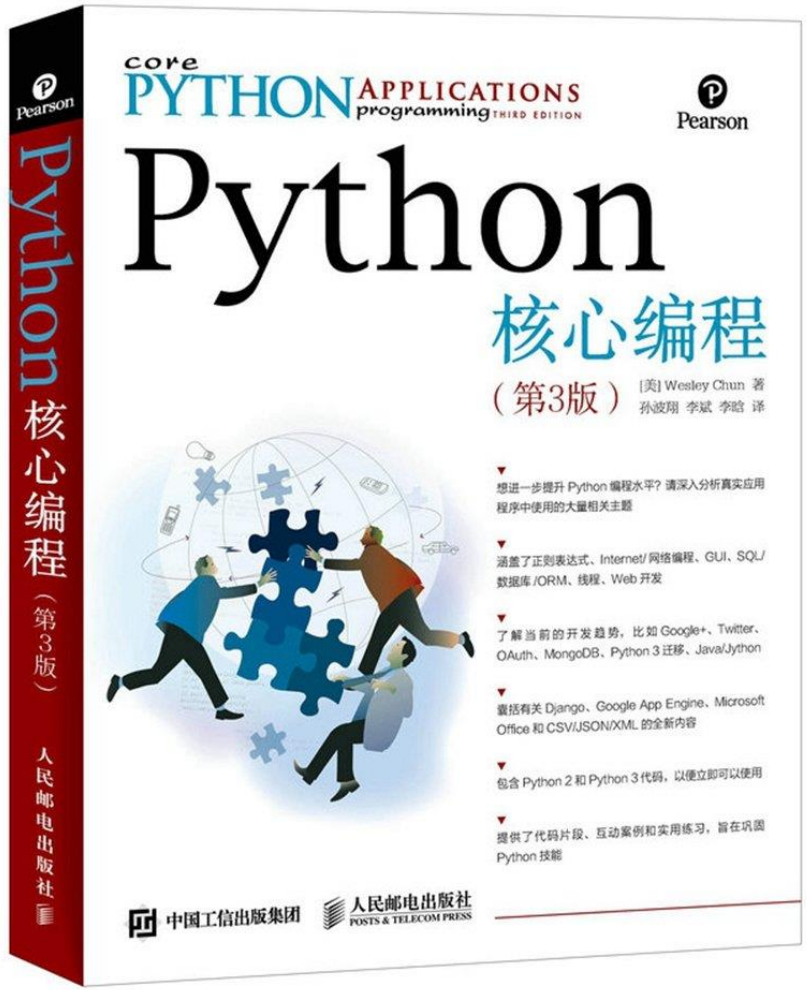
主要选用教材


TURING 图灵程序设计丛书
Apress
Beginning Python From Novice to Professional Third Edition
Python基础教程
（第3版）
[挪] Magnus Lie Hetland ◎著
袁国忠 ◎译
久负盛名的Python入门经典
中文版累计销量200 000+册
针对Python 3全新升级
中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS


O'REILLY
2nd Edition
Python for Data Analysis
DATA WRANGLING WITH PANDAS, NUMPY, AND IPYTHON
powered by
jupyter
Wes McKinney

参考教材


TURING 图灵程序设计丛书
MANNING
Machine Learning
in Action
机器学习
实战
[美] Peter Harrington 著
李锐 李鹏 曲亚东 王斌 译
使用Python阐述机器学习概念
介绍并实现机器学习的主流算法
面向日常任务的高效实战内容
人民邮电出版社
POSTS & TELECOM PRESS


Pearson
Python 核心编程（第3版）
人民邮电出版社
core PYTHON APPLICATIONS programming THIRD EDITION
Pearson
Python
核心编程
（第3版）
[美] Wesley Chun 著
孙波翔 李斌 李晗 译
想进一步提升 Python 编程水平？请深入分析真实应用程序中使用的大量相关主题
涵盖了正则表达式、Internet/网络编程、GUI、SQL/数据库/ORM、线程、Web 开发
了解当前的开发趋势，比如 Google+、Twitter、OAuth、MongoDB、Python 3 迁移、Java/Jython
囊括有关 Django、Google App Engine、Microsoft Office 和 CSV/JSON/XML 的全新内容
包含 Python 2 和 Python 3 代码，以便立即可以使用
提供了代码片段、互动案例和实用练习，旨在巩固 Python 技能
中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

课程结构

周次	章节	知识点	学时
10	编程基础-语言	介绍Python程序设计语言的基本语法、控制、异常、文件输入输出、代码风格及标准	2
11	编程基础-数据类型	介绍列表、字典、元组、集合	2
11	编程基础-函数	介绍Python中函数定义与基本内嵌函数	2
12	编程基础-面向对象	介绍Python中对象概念，面向对象设计与实现	2
12	编程基础-基础库	介绍数据持久化、正则表达式等标准库	2
13	编程基础-网络获取	介绍Python中数据网络爬虫基本功能、网络数据解析BS库	2
13	编程基础-GUI编程	介绍wxPython、PyQT、Tkinter等常用图形界面开发技术	2
14	数据分析-数据加载	介绍对各种格式数据读写技术，如JSON、XML、HTML、R-Database、二进制、Excel等	2
14	数据分析-数据表示	介绍基于Numpy的科学数据模型及矩阵、向量等数据类型操作	2
15	数据分析-数据预处理	介绍基于Pandas的矩阵数据降维、降噪、转换、合并、广播、结构化记录及多维预处理	2
15	数据分析-数值计算	介绍基于Scipy的常见数值计算模型	2
16	数据分析-数据可视化	介绍基于Matplotlib的数据交互可视化技术	2
16	数据分析-机器学习	介绍基于Scikit-learn的初级数据回归、分类、聚类基本理论	2
17	数据分析-文本语言处理	介绍基于Python的文本分词、文本分类、命名实体识别技术	2
17	数据分析-图像处理	介绍基于Python的图像数据基础处理、图像分割、特征提取与内容识别	2
18	数据分析-应用实践	介绍一个具体应用实践	2

课程时间地点

- 周二 10:40-11:30、11:40-12:30（三四节）

地点：1号楼A201

（9-16周）

- 周五 10:40-11:30、11:40-12:30 （三四节）

地点：1号楼A404

（10-17周）

成绩构成

Northeastern University

序号	类型	分值	内容	时间	说明
1	项目/实验	40	Project 1	00-08学时	1. 项目延迟提交扣分，提交代码文件和报告；
			Project 2	09-16学时	2. 两次实验课时间用于做项目（即：P2和P3项目），不另外设立实验题目；
			Project 3	17-24学时	3. 学时开始发布任务，结束当周周日提交；
			Project 4	25-32学时	
2	随堂测试	20	1-2次闭卷	随堂	
3	期末开始	40	1次闭卷	随堂/另外时间	

Project（40分）：4个projects,每个10分

第10周、第12周（实验课）、第14周（实验课）、第16周分别提交

晚交一天扣10%

随堂测试、随堂测试既点名（20分）

期末考试（40分）：闭卷

Project 1：

- 计划做一个关于**二手房数据分析项目**，共分4个小Project，每2周做一个，项目总名称为PyHouse，4个小project分别为v1/v2/v3/v4，每个小project需**提交报告+代码**，报告中对实验步骤做说明，列出并解释关键代码。**所有project在BB平台定时发布并设置提交日期，提交时间暂定在每个提交日的5：30pm，晚交按日扣分，提交打包文件命名规则：姓名_班级_学号**。项目需求为基本需求，没有特别的限定死，自由发挥的空间较大，使用的方法和库不设限。
- **Project1**：0-8学时，10月25日（周二）发布，11月8日（周二）提交
- **需求：**

1）做一个控制台程序，输入**省会**城市名称，获取对应汉语拼音首字母，例：北京->BJ;

2）支持输入的模糊匹配（搜索“京”可以提示“北京”或者把“南京”和“北京”都匹配出来），带参数解析（参数为要加载城市文件的路径、要输出的日志路径）；

3）写日志（将日志写到log文件里，记录内容自定）；

4）任意方法做一个加载城市文件的进度条（城市文件自己编辑）

实验课

- 班级：物联网2001-02；计算机2001-06
  - 12、14周周日下午
- 班级：人工智能（一）2001-02；人工智能2102-04，人工智能2101（未来）
  - 12、14周周三下午
- 班级：物联网2101-02 计算机2101-07
  - 12、14周周日晚
- 地点：实验中心

第1章 Python编程基础

李宇

东北大学-计算机学院-智慧系统实验室

liyu@cse.neu.edu.cn

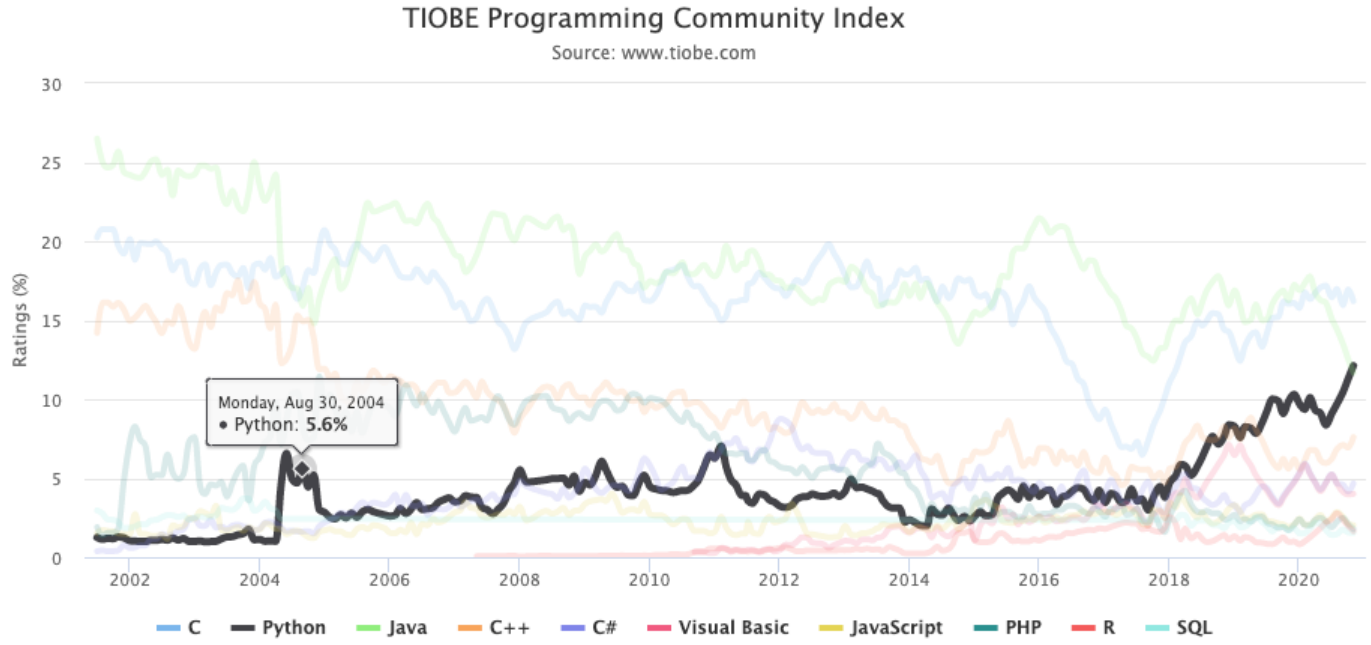
Python简介

- Python之父
  - 1989年 由Guido van Rossum(龟叔)创造

- 什么场景下用Python？哪些项目在使用Python?
  - 系统管理任务（多个Linux发行版的组成部分）
  - 适合向新手介绍编程
  - **NASA**的多个系统研发
  - 工业光魔公司在**影片**中用Python制作特效
  - Google用Python实现了**网络爬虫**和搜索引擎的组件
  - 游戏制作、生物信息分析、数据分析、金融量化交易、**人工智能**
  - 大型网站: 例如YouTube、Instagram，Facebook、国内的豆瓣、知乎…

编程语言的热度

- 约600种编程语言，20种流行
- 语言热度排行：https://www.tiobe.com/tiobe-index/



Python的优缺点

- 优点：优雅、明确、简单（来自龟叔对Python的定位）
  - Python简单易懂，适合初学者，不但入门容易，而且将来深入下去，可以编写非常复杂的程序。
  - 代码量少
  - 完善的基础代码库和第三方库

- 缺点：
  - Python：**慢，**和C比起来**非常慢**（解释型语言，需要翻译成CPU能理解的字节码，翻译过程非常耗时）
    - C语言是可以用来编写操作系统的贴近硬件的语言，所以，C语言适合开发那些追求运行速度、充分发挥硬件性能的程序
  - 不能加密，发布python程序就是发布源代码，而C语言可以把编译后的机器码（xx.exe文件）发布出去，无法反推出C代码

Python编程方式

- 交互式编程：
  - 不必要创建脚本文件，通过Python解释器的交互模式来编写代码
  - Linux，MacOS中的命令行窗口、Windows上安装交互式编程客户端
  - 常见解释器：CPython（官方）、IPython、PyPy、Jython、IronPython
- 脚本式编程：
  - 把Python程序保存在.py的文件中
  - 执行语句：python xx.py
  - 常用IDE：
    - Jupyter Notebook，PyCharm，Spyder，Eclipse（安装python开发插件），PyScripter等

Python 2.x和3.x版本

- Python3.x在设计的时候没有考虑向下兼容早期版本
  - 很多 Python 旧版本的程序无法在 Python 3.x 上正常执行
  - 许多语法不同
    - print：2.x时代是语句，3.x时代是一个函数
      - `print "fish" #Python 2.x`
      - `print ("fish") #Python 3.x`
    - 除法运算
      - `>>> 1 / 2 #Python 2.x 输出0`
      - `>>> 1 / 2 #Python 3.x 输出0.5`
    - 异常、不等运算符…都有改动
- 本课的示例代码使用Python3.x

表达式和语句

- **表达式（expression）**是值、变量和运算符的组合：
  - 5
  - x
  - x + 1
- **语句（statement）**是一个会产生影响的代码单元，例如新建一个变量或显示某个值。
  - n = 17.
  - n = “I am here”
  - print (n)
  - [a,b,c] = (1,2,3)
  - (a,b,c) = 'ABC’
  - a = b = c = 'spam'

Python 标识符

- 标识符(Identifier)由字母、数字、下划线(_)组成
- 区分大小写
- 保留字段，不能用作变量或其它标识符名称

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

赋值语句

- 序列解包：将一个序列解包，并将得到的值存储到一系列的变量中。
  - x, y, z = 1, 2, 3
  - values = 1, 2, 3
  - x, y, z = values

```
>>> dict = {'name': 'Robin', 'girlfriend': 'Marion'}
>>> key, value = dict.popitem()
>>> key
'girlfriend'
>>> value
'Marion'
```

返回并删除字典中的最后一对键和值

- 要解包的序列包含的元素个数必须和等号左边的目标个数相同，否则将引发异常

```
>>> x, y, z = 1, 2
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: need more than 2 values to unpack
```

赋值语句

- 可使用星号运算符（* ）来收集多余的值，这样无需确保值和变量的个数相同

```
>>> a, b, *rest = [1, 2, 3, 4]
>>> rest
[3, 4]
```

```
>>> name = "Albus Percival Wulfric Brian Dumbledore"
>>> first, *middle, last = name.split()
>>> middle
['Percival', 'Wulfric', 'Brian']
```

把字符串name按空格拆分成列表

条件和条件语句（一）

- 格式：

```
if 判断条件1:
        执行语句1……
elif 判断条件2:
        执行语句2……
elif 判断条件3:
        执行语句3……
else:
        执行语句4……
```

- 例子：

```
num = 5
if num == 3: # 判断num的值
        print("boss")
elif num == 2:
        print("user")
elif num == 1:
        print("worker")
elif num < 0: # 值小于零时
        print("error")
else: print("roadman") # 条件均不成立时
```

- Python 不支持switch case语句

条件和条件语句（二）

Northeastern University

- 条件判断的优先级
  - 当if有多个条件时可使用**括号**来区分判断的先后顺序，括号中的判断优先执行，此外 **and** 和 **or** 的优先级低于**>**（大于）、**<**（小于）等判断符号，即大于和小于在没有括号的情况下会比与或要优先判断。

```
num = 8 # 判断值是否在0~5或者10~15之间
if (num >= 0 and num <= 5) or (num >= 10 and num <= 15):          print('hello')
else:
        print('undefine')
```

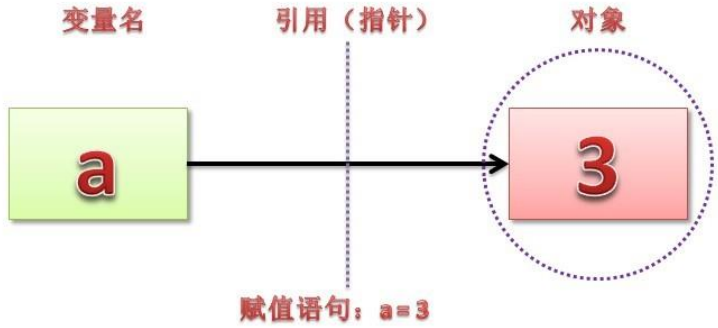
- 执行语句可以和条件判断语句在同一行

```
var = 100
if ( var == 100 ) : print("变量 var 的值为100")
print("Good bye! ")
```

条件和条件语句（三）

- is：相同运算符
  - 与 == 不同， == 比较两个对象的内容是否相等
  - is 检查两个对象是否相同（而非内容相等）

```
>>> x = y = [1, 2, 3]
>>> z = [1, 2, 3]
>>> x == y
True
>>> x == z
True
>>> x is y
True
>>> x is z
False
```



该赋值语句做了三件事：
1）开辟了一块内存（可用id(a)查看地址）
2）将该内存的值设为3
3）将变量名a指向该内存

深入赋值语句

变量名=对象

```
>>>a = 1
>>>b = 1
>>>id(a)
>>>id(b)
#输出是什么?
```

```
>>>a = 300
>>>b = 300
>>>id(a)
>>>id(b)
#输出是什么?
```

```
#验证:
for i in range(-1000,1000):
    a = i #直接将i赋给a
    b = int(i.__str__())
    if a is b: #判断a和b的地址是否相同
        print(i)
```

- 从-5开始到256结束，共**262个**整数会共享内存。python之所以这么干，是因为这些小的整数在编程时出现的频率非常高，可以节省内存开销
- 浮点型，字符串，列表，元组（空元组除外），字典不共享内存，每次开辟新内存

变量B=变量A

```
>>>a = 1
>>>b = a
>>>id(a)
>>>id(b)
```

无论a是何种数据类型，a和b都指向同一地址

其他判断用法

- in：成员资格运算符
  - 常用于条件表达式中

```
name = input('What is your name?') #等待键盘输入
if 's' in name:
        print('Your name contains the letter "s".')
else:
        print('Your name does not contain the letter "s".')
```

- assert： 断言
  - if语句的“亲戚，满足特定条件下，程序才能正常运行，否则崩溃
  - 充当程序的检查点

```
>>> age = 10
>>> assert 0 < age < 100
>>> age = -1
>>> assert 0 < age < 100
Traceback (most recent call last):
File "<stdin>", line 1, in ?
AssertionError
```

循环类型

- while循环

```
while 判断条件:
        执行语句.....
```

- for 循环

```
for iterating_var in sequence:
        执行语句.....
```

```
#例子
x = 1
while x <= 100:
        print(x)
        x += 1
```

```
#例子
for letter in 'Python':
        print ('当前字母 :', letter)
```

输出:

```
当前字母 : P
当前字母 : y
当前字母 : t
当前字母 : h
当前字母 : o
当前字母 : n
```

循环类型（二）

- 嵌套循环
  - **for 循环嵌套**
  - **while循环嵌套**
  - **while嵌入for， for嵌入while**

```
for iterating_var in sequence:
        for iterating_var in sequence:
                statements
                statements
```

```
while expression:
        while expression:
                statements
                statements
```

- 循环控制语句
  - break　#跳出整个循环。
  - continue #continue 语句跳出本次循环的剩余语句
  - pass #pass 不做任何事，一般用做占位语句

并行迭代

迭代两个序列

```
names = ['anne', 'beth', 'george', 'damon']
ages = [12, 45, 32, 102]
#打印名字和对应的年龄
for i in range(len(names)):
        print(names[i], 'is', ages[i], 'years old')
```

内置函数zip()：缝合多个序列

```
>>>list(zip(names, ages))
# [('anne', 12), ('beth', 45), ('george', 32), ('damon', 102)]
```

循环解包

```
for name, age in zip(names, ages):
        print(name, 'is', age, 'years old')
```

```
anne is 12 years old
beth is 45 years old
george is 32 years old
damon is 102 years old
```

pass、del

- pass
  - 什么都不做，用作占位

```
if name == 'Ralph Auldus Melish’:
        print('Welcome!')
elif name == 'Enid’:
        # 还未完成……
```

代码块为空，不能运行

```
if name == 'Ralph Auldus Melish':
        print('Welcome!')
elif name == 'Enid':
        # 还未完成……
        pass
```

- del
  - 删除对象的**名称**

```
name = {'first name': 'Robin', 'last name': 'James'}
robin = name
name = None
robin
```

无名称指向字典，被回收

输出 ???

```
#{'first name': 'Robin', 'last name': 'James'}
robin = None
```

```
>>> x = ["Hello", "world"]
>>> y = x
>>> y[1] = "Python"
>>> x
['Hello', 'Python’]
>>> del x
>>> y
['Hello', 'Python']
```

exec()、eval()

- exec()
  - 内置函数（Python2中是一种语句）
  - 将传入的字符串作为代码执行，**可以执行复杂**的代码逻辑
  - 无返回结果

```
exec("print('Hello, world!')")
```

- eval()
  - eval是一个类似于exec的内置函数
  - 只能传入单个运算表达式，**不支持复杂**代码逻辑
  - 返回结果

```
eval(input("Enter an arithmetic expression: "))
```

代码块：缩进

- Python特色：使用**缩进**来区分代码块的边界，而非{ }
- 缩进数量可变，但所有代码块必须包含**相同量的缩进**

四个空格

```
if True:
    print ("True")
else:
    print ("False")
```

```
if True:
    print ("Answer")
    print ("True")
else:
    print ("Answer")
    # 没有严格缩进，在执行时会报错
  print ("False")
```

- 缩进错误提示：
  - **IndentationError: unindent does not match any outer indentation level (缩进方式不一致)**
  - **IndentationError: unexpected indent (没对齐)**

Python代码风格规范

- **勿用分号**做结尾：
  - 不要在行尾加分号
  - 不要用分号将两条命令放在同一行.

✗ x=a+b;
✗ x=a+b; y=c+d

- 行长度：每行不超过**80个**字符
  - 使用**括号行连接**
  - 例外：
    - 长的导入模块语句
    - 注释里的URL

```
foo_bar(self, width, height, color='black', design=None,
        x='foo', emphasis=None, highlight=0)

if (width == 0 and height == 0 and color == 'red' and
        emphasis == 'strong'):
```

- 尽量**不要**在**返回语句**或**条件语句**中使用**括号**

✗
```
if (x):
    bar()
if not (x):
    bar()
return (foo)
```

✓
```
if foo:
    bar()
while x:
    x = bar()
if x and y:
    bar()
if not x: bar()
return foo
```

Python代码风格规范

- 空格
  - 括号内**尽量不要**有空格
    - ✓ spam(ham[1], {eggs: 2}, [])
    - ✗ spam(ham[1], { eggs: 2 }, [ ])
  - **尽量不要**在逗号, 分号, 冒号前面加空格, 但应该在它们后面加(除了在行尾).

✓
```
if x == 4:
    print(x, y)
x, y = y, x
```

✗
```
if x == 4 :
    print(x , y)
x , y = y , x
```

  - 在二元操作符两边都**尽量加上**一个空格（使用‘=’设置参数时除外）
    - ✓ x == 1
    - ✗ x<1

```
def complex(real, imag=0.0):
    return magic(r=real, i=imag)
```

Python注释

- 单行注释： #
  - # 第一个注释
    ```
    print ("Hello, Python!")  # 第二个注释
    ```

- 多行注释
  - 多行注释使用三个单引号(''')或三个双引号(""")。
    - '''
      这是多行注释，使用单引号。
      这是多行注释，使用单引号。
      这是多行注释，使用单引号。
      '''

      """
      这是多行注释，使用双引号。
      这是多行注释，使用双引号。
      这是多行注释，使用双引号。
      """

文件输入输出

- open函数
  - f = open(‘somefile.txt’)
  - 如果只指定文件名，则文件只可读
  - f = open(‘somefile.txt’,’打开模式参数’)

文件打开模式

模式	描述
t	文本模式 (默认)。
x	写模式，新建一个文件，如果该文件已存在则会报错。
b	二进制模式。
+	打开一个文件进行更新(可读可写)。
U	通用换行模式（不推荐）。
r	以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。
rb	以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式。一般用于非文本文件如图片等。
r+	打开一个文件用于读写。文件指针将会放在文件的开头。
rb+	以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。一般用于非文本文件如图片等。
w	打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。
wb	以二进制格式打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。一般用于非文本文件如图片等。
w+	打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。
wb+	以二进制格式打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。一般用于非文本文件如图片等。
a	打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
ab	以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
a+	打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。
ab+	以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。

写入和读取

- f.write()函数
  - 向文件写入指定的字符串
- f.read()函数
  - 读取指定的字节数
  - f.readline()函数
    - 读取整行
- f.close()函数
  - 常在finally语句块中使用
  - 写入内容可能只在缓冲区

```
f = open('somefile.txt', 'w')
f.write('Hello\n')
f.write('World')
f.close()

with open('somefile.txt', 'r') as f:
    content = f.read(5)
    content1 = f.read()
    print(content,content1)
    f.close()

with open('somefile.txt', 'r') as f:
    while True:
        line = f.readline()
        if not line : break
        print(line)
    f.close()
```

从文件开头读5个字节

从文件指针位置读到结尾

逐行读取整个文件

异常的引发，创建与处理

- Python在遇到错误时引发异常，异常对象未被处理时，程序终止并显示错误消息(traceback)
- >>> 1 / 0

```
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
```

- raise语句
  - 手动**引发/抛出**异常
  - 一旦执行raise语句，**程序终止，显示错误消息**
  - 将一个类（Exception的子类）或实例做为参数
  - 内置异常类：Exception
    - >>> raise Exception #引发的是通用异常，没有指出出现了什么错误
    - >>> raise Exception('Drive Overload') #添加了错误消息hyperdrive overload
    - >>> raise OSError #内置异常类，从Exception类派生而来

处理异常

try/except语句

```
try:
        <语句> #运行代码
except <名字1>:
        <语句> #如果在try部份引发了<名字1>异常
except <名字2>，<异常参数>:
        <语句> #如果引发了<名字2>异常，获得附加的数据
else:
        <语句> #如果没有异常发生
```

```
try:
        f = open("testfile", "w")
        f.write("这是一个测试文件，用于测试异常!!")
except IOError:
        print("Error: 没有找到文件或读取文件失败")
else:
        print("内容写入文件成功")
            f.close()
```

捕捉多个异常
通常做法：except Exception as e

```
while True:
    try:
        x = int(input('Enter the 1st number: '))
        y = int(input('Enter the 2nd number: '))
        value = x / y
        print('x / y is', value)
    except Exception as e:
        print('Invalid input:', e)
        print('Please try again')
    else:
            break
```

常见的内置异常类

类　　名	描　　述
Exception	几乎所有的异常类都是从它派生而来的
AttributeError	引用属性或给它赋值失败时引发
OSError	操作系统不能执行指定的任务（如打开文件）时引发，有多个子类
IndexError	使用序列中不存在的索引时引发，为LookupError的子类
KeyError	使用映射中不存在的键时引发，为LookupError的子类
NameError	找不到名称（变量）时引发
SyntaxError	代码不正确时引发
TypeError	将内置操作或函数用于类型不正确的对象时引发
ValueError	将内置操作或函数用于这样的对象时引发：其类型正确但包含的值不合适
ZeroDivisionError	在除法或求模运算的第二个参数为零时引发

创建自定义异常类

- 内置异常类不够用时
- 专门的错误，用专门的异常类去处理
- **务必直接或间接继承Exception类**
- 创建实例

```
class SomeCustomException(Exception):
    pass
```

try…finally语句

- 无论try语句块中是否触发异常，都会执行finally子句中的语句块

```
try:
        fh = open("testfile", "w")
        fh.write("这是一个测试文件，用于测试异常!!")
finally:
        print("Error: 没有找到文件或读取文件失败")
```

- 什么语句放在finally中？
  - 关闭文件或关闭因系统错误而无法正常释放的资源。
    - 文件关闭
    - 释放锁
    - 把数据库连接返还给连接池

模块（Module）

- 模块是一个Python文件，以 .py结尾，包含了Python对象定义和Python语句。
  - 例如，在support.py模块中：

```
def print_func( var ):
    print("Hello : ", var)
    return
```

- 导入模块：**import** 语句

```
# 导入模块
import support
# 现在可以调用模块里包含的函数了
support.print_func("Bob")
```

```
# 使用from语句导入模块中的某个函数,
# 不会把整个模块导入到当前命名空间中
from support import print_func
# 导入所有内容，使用*号
from math import *
math.floor(32.9)
```

向下取整

模块内有函数依赖

- 假设下列代码存储在test.py里
  - quarter()中调用了average()
  - 使用from test import quarter能行吗?

```
#test.py内容
def average(x, y):
        return((x + y) /2)

def quarter(x,y):
        print(average(x, y)/2)
```

```
>>> from test import quarter    #导入成功
>>> quarter(3,5)      #2
>>> average(x, y):    # NameError: name 'average' is not defined
```

导入时重命名

- 导入时重命名：如果导入两个模块， module1和module2，都含有函数open，怎么办?
  - 方法一:
    - **module1**.open(...)
    - **module2**.open(...)
  - 方法二： 在import语句末添加as子句，给**模块**起**别名**。
    - import module1 as mo
    - mo.open(…)
  - 方法三： 给两个**open函数**分别起**别名**
    - from module1 import open as open1
    - from module2 import open as open2

感谢