Python数据分析
（2）

殷传涛 教授
chuantao.yin@buaa.edu.cn

程序控制结构

- 程序可以由三种控制结构组成
  - 顺序语句：按代码的先后顺序执行
    - 对应串行程序，按顺序执行语句，直到语句执行完毕
  - 选择语句：根据条件进行分支
    - 对应选择结构，典型是if-elif-else语句
  - 循环语句：在一定条件下重复执行代码
    - 对应迭代结构，典型是for和while语句

逻辑表达式

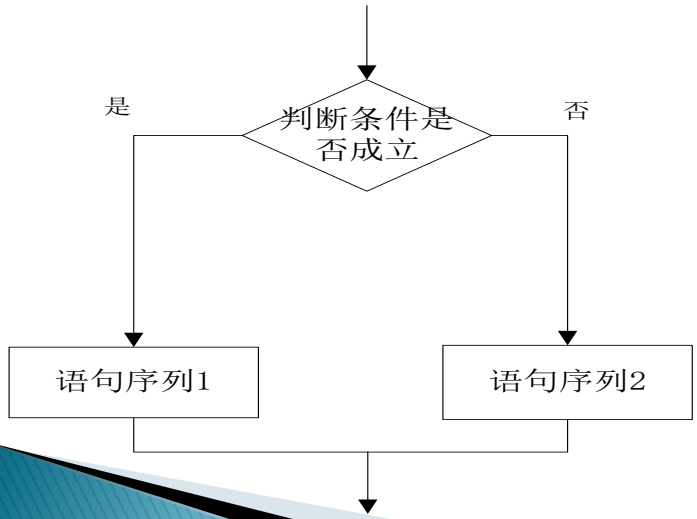
- 逻辑表达式
  - 结果为布尔型数据（True或False）的表达式
  - 选择或循环语句的判定条件
- 带比较运算符的逻辑表达式
  - ==, !=　等于，不等于
  - <, >　小于，大于
  - >=, <=　大于等于，小于等于
- 带逻辑运算符的逻辑表达式
  - and
  - or
  - not

```
>>> x=3
>>> y=4
>>> print(x==y, x!=y, x<y, x<=y, x>y, x>=y )
False True True True False False
```

```
>>> a = True
>>> b = False
>>> print(a and b, a or b, not a)
False True False
```

选择语句

- 由if, else引出的选择语句：
  - 条件部分，即一个逻辑表达式，值为True或者False
  - 如果条件值为True，则执行语句序列1，
  - 如果条件值为False，则执行语句序列2，




```
x = int(input('Enter an integer:'))
if x%2 == 0:
    print('Even')
else:
    print('odd')
print('done with conditional')
```

选择语句

```
if condition_1:
    statement_block_1
elif condition_2:
    statement_block_2
else:
    statement_block_3
```

1. 如果 "condition_1" 为 True 将执行 "statement_block_1" 块语句
2. 如果 "condition_1" 为False，将判断 "condition_2"
3. 如果"condition_2" 为 True 将执行 "statement_block_2" 块语句
4. 如果 "condition_2" 为False，将执行 "statement_block_3"块语句

- elif 即为else if
- elif和else均为可选，elif可以有多个
- if,elif,else后面都要有冒号：，表示后面有语句块
- 使用相同的缩进来表示同一个语句块

```
# 如果冒号输入正确
# IDLE会自动缩进
```

选择语句

- 示例：根据输入的分数，输出等级评语

2.py - C:/Users/chuan/Desktop/2.py (3.12.0)

File Edit Format Run Options Window Help

```
# 获取用户输入
score = int(input("请输入你的考试成绩: "))

# 根据成绩输出评价
if score >= 90:
    print("优秀！")
elif score >= 80:
    print("良好。")
elif score >= 70:
    print("中等。")
elif score >= 60:
    print("及格。")
else:
    print("不及格，继续努力。")
```

```
==========
请输入你的考试成绩: 95
优秀！

==========
请输入你的考试成绩: 45
不及格，继续努力。

==========
请输入你的考试成绩: 65
及格。

==========
请输入你的考试成绩: 80
良好。
```

选择语句的嵌套

- 在if-elif-else语句的程序块中可能出现另外的if-elif-else语句

```
if 表达式1:
    语句
    if 表达式2:
        语句
    elif 表达式3:
        语句
    else:
        语句
elif 表达式4:
    语句
else:
    语句
```

```
# !/usr/bin/python3

num=int(input("输入一个数字："))
if num%2==0:
    if num%3==0:
        print ("你输入的数字可以整除 2 和 3")
    else:
        print ("你输入的数字可以整除 2，但不能整除 3")
else:
    if num%3==0:
        print ("你输入的数字可以整除 3，但不能整除 2")
    else:
        print  ("你输入的数字不能整除 2 和 3")
```

```
$ python3 test.py
输入一个数字：6
你输入的数字可以整除 2 和 3
```

选择语句

- 示例：输入三个整数，求出其中的最小的数

```
x = int(input('Enter an integer for x:'))
y = int(input('Enter an integer for y:'))
z = int(input('Enter an integer for z:'))

if x < y and x < z:
    print('x is the least')
elif y < z:
    print('y is the least')
else:
    print('z is the least')
```

循环语句：while语句

- while语句
  - 在执行前先进行条件表达式判断，如果结果为True，则执行一次程序块；
  - 然后再回到条件表达式重新进行判断……直到结果为False，跳过程序块，执行后面的代码。

3.py - C:/Users/chuan/AppData/Local/Programs/Python/Python312/3.py

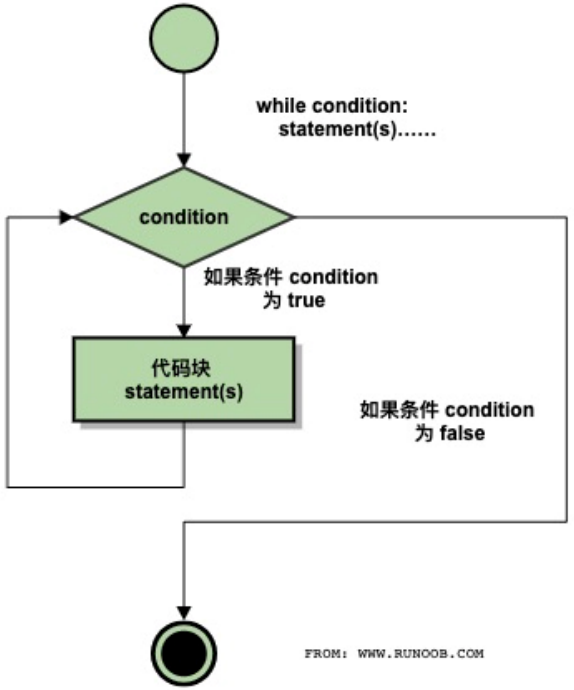
File Edit Format Run Options Window Help

```
# 初始化计数器
count = 1

# while循环，当count小于等于5时执行
while count <= 5:
    print(count)
    count = count + 1  # 更新计数器

print("循环结束")
```

```
==============
1
2
3
4
5
循环结束
```



- 注意冒号与语句块的格式

循环语句

- 示例：求1到100之间数字的和

4.py - C:/Users/chuan/Desktop/4.py (3.12.0)

File Edit Format Run Options Window Help

```
# 初始化变量
sum_result = 0
current_number = 1

# 使用while循环求和
while current_number <= 100:
    sum_result += current_number
    current_number += 1

# 输出结果
print("1到100的和是:", sum_result)
```

```
==========
1到100的和是: 5050
```

循环语句

- 示例：用户名和密码验证

5.py - C:/Users/chuan/Desktop/5.py (3.12.0)

File Edit Format Run Options Window Help

```
# 设定用户名和密码
correct_username = "user123"
correct_password = "pass456"

# 获取用户输入
username = input("请输入用户名: ")
password = input("请输入密码: ")

# 验证用户名和密码
while username != correct_username or password != correct_password:
    print("用户名或密码错误，请重新输入。")
    username = input("请输入用户名: ")
    password = input("请输入密码: ")

# 循环结束，输出登录成功消息
print("登录成功！")
```

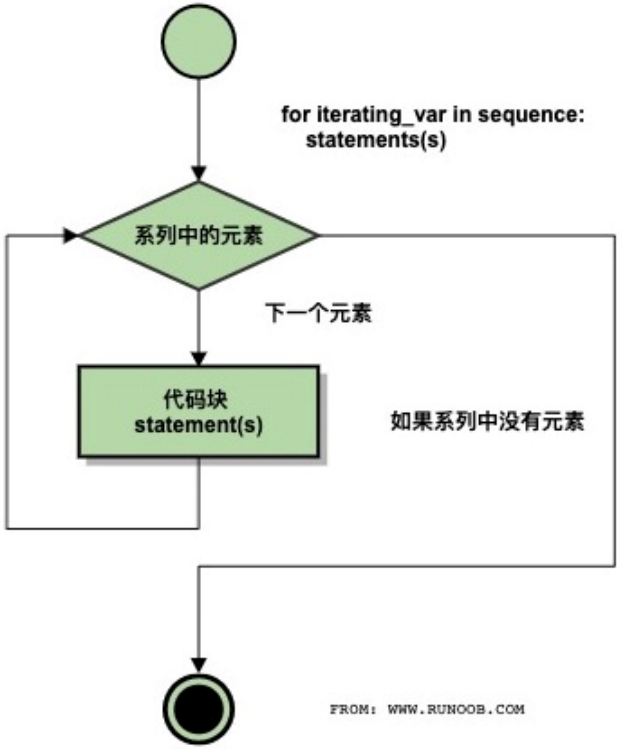
```
=======
请输入用户名: use111
请输入密码: passgood
用户名或密码错误，请重新输入。
请输入用户名: use222
请输入密码: pass333
用户名或密码错误，请重新输入。
请输入用户名: user123
请输入密码: pass456
登录成功！
```

循环语句：for语句

- for语句：针对一个序列，重复执行程序
  - 令<循环变量>取遍<序列>中每一个值，同时执行一次循环体的语句块。
  - <序列>可以是列表、字符串等，也可以使用range( )表示整数序列。

```
for <variable> in <sequence>:
    <statements>
```

```
>>> sites = ["Baidu", "Google","Runoob","Taobao"]
>>> for site in sites:
...     print(site)
...
Baidu
Google
Runoob
Taobao
```



循环语句

- range()函数表示数字序列

```
>>> help(range)
Help on class range in module builtins:

class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
```

```
>>>for i in range(5):
...     print(i)
...
0
1
2
3
4
```

```
>>>for i in range(5,9) :
    print(i)


5
6
7
8
>>>
```

```
>>>for i in range(0, 10, 3) :
    print(i)


0
3
6
9
>>>
```

循环语句

- for语句

- **计数器:序列只是用来控制循环的次数.**

```
for i in range(10):
    print ("烦")
```

  - **循环体不引用循环变量.**

- **数据:序列本身是循环体处理的数据.**

```
for i in range(10):
    print (i*i)
```

  - **循环体引用循环变量.**

```
>>> for i in range(10):
...     print("烦")
...

烦
烦
烦
烦
烦
烦
烦
烦
烦
烦
>>>
```

```
>>> for i in range(10):
...     print(i*i)
...

0
1
4
9
16
25
36
49
64
81
```

循环语句的嵌套

6.py - C:/Users/chuan/Desktop/6.py (3.12.0)

File Edit Format Run Options Window Help

```
# 两层嵌套的for循环输出六层三角形
for i in range(1, 7):  # 控制层数
    for j in range(i):  # 控制每层的*数量
        print("*", end=" ")
    print()  # 换行，开始新的一层


print("\n\n")
# 左右颠倒的六层三角形
for i in range(6, 0, -1):  # 控制层数，递减
    # 添加空格以控制左右位置
    print("  " * (6 - i), end="")

    for j in range(i):  # 控制每层*数量
        print("*", end=" ")

    print()  # 换行，开始新的一层
```

```
= RESTART: C:/Users/chuan/Desk
*
* *
* * *
* * * *
* * * * *
* * * * * *



* * * * * *
  * * * * *
    * * * *
      * * *
        * *
          *
```

关于循环的注意问题：

- while和for循环的区别
  - while的循环次数不确定，主要靠条件来判断
  - for循环的次数确定，由被遍历的集合来确定
- 关于while循环
  - 条件判断：若条件满足，则执行循环一次；若条件不满足，则不执行循环。
  - 循环体内的语句，要影响下一次循环的条件判断
    - 如：使用input()，i=i+1等改变条件判断中的变量
    - 否则将导致无穷循环 (用ctr+c中断，或关掉进程)

3.py - C:/Users/chuan/AppData/Local/Programs/Python/Python312/3.py

File Edit Format Run Options Window Help

```
# 初始化计数器
count = 1

# while循环，当count小于等于5时执行
while count <= 5:
    print(count)
    count = count + 1  # 更新计数器

print("循环结束")
```

3-1.py - C:\Users\chuan\Desktop\3-1.py (3.12.0)

File Edit Format Run Options Window Help

```
# 初始化计数器
count = 1

# while循环，当count小于等于5时执行
while count <= 5:
    print(count)

print("循环结束")
```

关于循环的注意问题：

- 循环非正常中断
  - 关键字break
    - 中止整个循环
  - 关键字continue
    - 中止本次循环

```
a=[1, 3, 4, 5, 7, 8, 10, 11, 14]
sum=0
for i in a:
    if i%2 ==0:
        break
    sum=sum+i;

print(sum)
```

```
==================
4
>>>
```

```
a=[1, 3, 4, 5, 7, 8, 10, 11, 14]
sum=0
for i in a:
    if i%2 ==0:
        continue
    sum=sum+i;

print(sum)
```

```
==================
27
>>>
```

求序列中的奇数之和

学习编程易犯的错误

- 语法错误
  - 严谨的编程语言均会进行语法错误检查，如果含有任何语法错误，用户都无法执行程序
- 静态语义错误
  - 在允许执行程序前需要进行很多静态语义检查。
- 程序的执行并不是创建者的意图
  - 它可能崩溃，也就是停止运行，给出示意
  - 它可能继续运行，持续运行而不停止
  - 它也可能完成运行，产生不正确的结果，甚至有可能是看似正确的结果

编程规范

- 现实：
  - 软件全生命周期的70%成本是维护
  - 软件很少由原编写人员进行维护

- 软件编程规范：与特定语言相关的描述如何编写高质量代码的相关规则集合

- 目的：
  - 提高编程质量，避免程序错误
  - 增强代码的可读性、可重用性、可维护性

编程规范

https://peps.python.org/pep-0008/
https://github.com/google/styleguide

Google Style Guides

Every major open-source project has its own style guide: a set of conventions (sometimes arbitrary) about how to write code for that project. It is much easier to understand a large codebase when all the code in it is in a consistent style.

"Style" covers a lot of ground, from "use camelCase for variable names" to "never use global variables" to "never use exceptions." This project holds the style guidelines we use for Google code. If you are modifying a project that originated at Google, you may be pointed to this page to see the style guides that apply to that project.

Our C++ Style Guide, Objective-C Style Guide, Java Style Guide, Python Style Guide, R Style Guide, Shell Style Guide, HTML/CSS Style Guide, JavaScript Style Guide, AngularJS Style Guide, Common Lisp Style Guide, and Vimscript Style Guide are now available. We have also released cpplint, a tool to assist with style guide compliance, and google-c-style.el, an Emacs settings file for Google style.

If your project requires that you create a new XML document format, our XML Document Format Style Guide may be helpful. In addition to actual style rules, it also contains advice on designing your own vs. adapting an existing format, on XML instance document formatting, and on elements vs. attributes.

编程规范

- 注释
  - 写注释是一个良好的编程习惯。
  - 注释用来向程序的读者说明编程思路及算法逻辑
  - 注释的语句不会被执行
  - 单行的注释：以#开头
  - 多行的注释：用”””或’’’括住

```
# 这是一个注释
print("Hello, World!")
```

```
#!/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!")
```

```
#!/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!")
```

```
# compute the circumference of circle using radius
pi = 3.14159
radius = 10.5
circumference = 2 * pi * radius

print(circumference)

"""
This is an example
This line can't be executed
"""
```

编程规范

- 单行注释与多行注释的区别

- 形式1：由 # 开头的“真正的”注释，说明选择当前实现的原因以及这种实现的原理和难点；

```
def gold_divide(n):
    # 黄金分割点比例为(√5-1)/2≈0.618034

    return n * 0.618 # 直接取0.618以加速
```

- 形式2：文档字符串，说明如何使用包、模块、类、函数（方法），甚至包括使用示例和单元测试。

```
def gold_divide(n):
    """ Get gold divide value of n.

    Args:
        n: input number

    Returns:
        A float value
    """
    return n * 0.618
```

软件编程规范

- 注释

学会只编写够用的注释，过犹不及，重视质量而不是数量。

- 好的注释解释为什么，而不是怎么样
- 不要在注释中重复描述代码
- 当自己在编写密密麻麻的注释来解释代码时，需要停下来看是否存在更大的问题
- 想一想在注释中写什么，不要不动脑筋就输入
- 写完注释之后要在代码的上下文中回顾一下，它们是否包含正确的信息？
- 当修改代码时，维护代码周围的所有注释

软件编程规范

- 注释

```
def main(argv):
    """
    主函数，返回0表示成功
  """
  try:
        with open('cat.pic', 'rb') as fin:
            if fin.seek(0, 2) != 400 *400 *3: # 判断文件长度是否符合格式要求
          print("输入文件 cat.pic 不符合格式要求")
                return -1
            fin.seek(0)
            try:
                with open('cat2.pic', 'wb') as fout:
           # 下面是图像转换的算法实现。彩色图像到灰度图像的转换主要利用
          # RGB色彩空间到YUV色彩空间的变换公式来取得灰度值Y，公式是
          # Y = 0.299R + 0.587G + 0.114B
          for i in range(400):
                        for j in range(400):
                            b = ord(fin.read(1))
                            g = ord(fin.read(1))
                            r = ord(fin.read(1))
                            y = (299 * r + 587 * g + 114 * b) / 1000
                            fout.write(chr(y))
```

软件编程规范

- 命名

好的名字一目了然，不需要读者去猜，甚至不需要注释。

- 命名规范

module_name, package_name, ClassName, method_name, ExceptionName, function_name, GLOBAL_CONSTANT_NAME, global_var_name, instance_var_name, function_parameter_name, local_var_name.

  - 注：单个字母名称，仅用在计数器或者迭代器

编程规范

- Python的命名规则
  - 变量命名：除了函数中的临时变量，一般情况下，变量命名要求有意义，多用小写字母，全局变量一般用大写。
  - 函数命名：必须以字母或下划线开头，可包含任意字母、数字或下划线的组合；区分大小写；不能是系统保留字
  - 类和对象命名：类名首字母要大学，其它字母小写；类私有属性和方法以两个下划线作前缀；对象命名用小写字母
  - 包和模块命名：均用小写字母不要含下划线

软件编程规范

- 命名

```
def fval(i):
    ret = 2
    n1 = 1
    n2 = 1
    i2 = i - 3
    while i2 >= 0:
        n1 = n2
        n2 = ret
        ret = n2 + n1
        i2 -= 1
    return 1 if i < 2 else ret
```

```
def fibonacci(position):
    if position < 2:
        return 1

    previous_but_one = 1
    previous = 1
    result = 2
    for n in range(2, position):
        previous_but_one = previous
        previous = result
        result = previous + previous_but_one
    return result
```

编程规范

- Python编码规则
  - 缩进：不同于类似 C 族语言的 { } 符号或 Pascal 的 begin/end关键字标记块；Python语句块的组织依赖于缩进。在一行代码前缩进若干个单位表示行与行之间的层次关系，且不能忽略。比较常用是缩进4个空格或使用Tab键。
  - 冒号：Python的语法格式，在if，while，for语句，def 函数名后面要紧跟冒号
  - 空行：空行不是Python的语法格式，但是函数间或类间用空行分隔，表示新代码的开始，能增加代码的可读性。

编程实践

- 交互模式下
  - help()可进入帮助状态，然后：
    - help>modules列出所有模块
    - help>keywords列出所有关键字
    - help>topics列出所有主题
  - 也可使用help(函数或模块名)
    - 如help(print)

```
>>> help()

Welcome to Python 3.6's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/3.6/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> print
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

- 多看，多练，多问，多搜
  - 看：阅读优秀的代码，学习别人的代码
  - 练：亲自动手编写代码，实践、实践、再实践
  - 问：问高手，能够较快看出自己看不出的问题
  - 搜：百度，google，github等

谢谢！