Python 语 法 手 册

基础知识 A

Reference: datasciencefree.com xaero@msn.cn

Python 概况

- 字符区分大小写
- 索引（下标）从 0 开始
- 用空白符（4 个空格）缩进代替花括号来表示代码的开始、结束和从属关系

获取帮助

帮助首页	`help()`
函数帮助	`help(list.sort)`
模块帮助	`help(numpy)`

导入模块

模块就是一个“`.py`”文件

查看模块内容	`dir(module1)`
导入模块	`import module1` ※
调用模块函数	`module1.func()`

※ 还有一种方式：`from module1 import func`，则无需使用命名空间，直接使用函数func()

变量和基本类型

标识符

- 汉字等 Unicode 字符也可以，但不建议
- 非数字开头
- 建议用字母、数字或下划线 _
- 注意避讳、区分大小写，如：

```
☺ a    b1    y_max    BigOne
☒ 8a   and   for
```

运算符

例中 a=17, b=10, c=5；优先级数值小者优先

算术运算符

运算符	功能	优先级	举例
**	幂次	3	`c ** 3 = 255`
*	乘	5	`c * 3 = 15`
/	除	5	`a / b = 1.7`
//	除	5	`a // b = 1`
%	求余	5	`a % b = 7`
+	加	6	`a + b = 27`
-	减	6	`a - b = 7`

比较运算符

运算符	功能	优先级	举例
<	小于	11	`a < b False`
<=	小于等于	11	`c <= 5 True`
>	大于	11	`c > 5 False`
>=	大于等于	11	`c >= 5 True`
==	等于	12	`c == 5 True`
!=	不等于	12	`c != 6 True`

逻辑运算符

运算符	功能	优先级	描述
not	逻辑非	13	not x 当 x 为非 0 时返回 False，其他情况返回 True，如 not a 返回 False
and	逻辑与	14	x and y 当 x 为 False 时返回 False，其他情况返回 y 的值，如 a and b 返回 10
or	逻辑或	15	x or y 当 x 为非 0 时返回 x 的值，其他情况返回 y 的值，如 a or b 返回 17

位运算符

运算符	功能	优先级	举例
~	位非	4	`~c = -6`
&	位与	8	`b & 3 = 2`
^	位异或	9	`b ^ 3 = 9`
|	位或	10	`b |3 = 11`
<<	位左移	7	`c << 1 = 10`
>>	位右移	7	`c >> 1 = 2`

6 种基本数据类型

int/long 如：`7 0b010 0o67 0xF3`，大数自动转 long
float 如：`3.14 1.7e-6`，64 位，注意没有 double 类型
bool True 或者 False
str 可用单引号、双引号或三引号表示，值不可变
NoneType 只有一个数值 **None** 一般当成函数的默认参数，或者是否是空对象：

```
def func1(a, b, c = None)
if variable is None:
```

datetime 日期、时间类型

- 从字符串创建时间日期：

```
dt1 = datetime.strptime('20210817','%Y%m%d')
```

- 获取日期、时间：`d, t = date(), time()`

类型转换

```
type(x)          # 查看变量类型
int(7.79)        # 结果是 7
int('7.79')      # 出错
int('7')         # 结果是 7
int('3f', 16)    # 转成 16 进制，结果是 63
float('7.79')    # 结果是实数7.79
str(7.79)        # 字符串7.79，详见格式化字符
chr(65)          # 大写字母A
ord('A')         # A的ASCII码值65
bytes([65, 9, 64]) # 二进制值b'A\t@'
```

变量赋值

- 先计算表达式右侧的值
- 按顺序依次赋给左侧的变量（容器）

```
x = 1.2 + 8 + sin(y)
a = b = c = 0              # 赋予同一个值
y, z, r = 9.2, -7.6, 0     # 多个赋值
a, b = b, a                # 变量值交换
a, *b = seq                # 解压缩序列
*a, b = seq                # 解压缩序列
x += 3                     # 算术、位运算都支持混合
x = None                   # 未定义变量
del x                      # 删除变量
```

数据结构

Tuple 元组

元组是 Python 一维的、固定长度、不可变的、可以由任意数据组成的序列

```
tup1 = 4, 5, 6          # 创建元组
tup2 = (4, 5, 6)        # 同 tup1
tup3 = (4,5,6), (7,8) # 嵌套
tuple([1, -2, 7])       # 转成元组
tup1 + tup2             # 元组合并
a, b, c = tup1          # 元组解包
```

List 列表

列表是 Python 一维的、长度可变的、数据可变的、由任意数据组成的序列

```
lst1 = [-1, 'a', 7]          # 创建列表
lst2 = list(tup1)            # 由元组创建
lst3 = lst1 + lst2           # 列表合并，不建议用
lst4 = lst1.extend(lst2)     # 同 lst3，但效率高
lst1.append('b')             # 在表尾添加元素
lst1.insert(p, 'x')          # 在 p 号位置插入
lst1.pop(p)                  # 删除 p 号位置元素
lst1.remove('a')             # 删除值是'a' 的元素
7 in lst1                    # True，线性复杂度
lst1.sort()                  # lst1 原地排序（升序）
lst1.sort(reverse=True)      # lst1 变成降序
lst1.sort(key=len)           # 用 len() 函数排序
```

切片

切片操作对所有序列容器都有效，包括列表、字符串、

元组、数组 (array) 等

```
负索引：   -5 |-4 |-3 |-2 |-1
正索引：    0 | 1 | 2 | 3 | 4
   lst = [10, 20, 30, 40, 50]
正切片：  0   1   2   3   4   5
负切片：-5  -4  -3  -2  -1
```

- 切片格式：**lst[start:end:step]**
- step 省略即为 1；start 省略表示从 0 开始，end 省略表示到末尾
- 用切片、下标即可以读取元素，也可以修改元素

```
lst[0]       # 索引，结果 10
lst[0:]      # 切片，结果 [10, 20, 30, 40, 50]
lst[-1]      # 索引，结果是元素 50
lst[-1:]     # 切片，结果是列表 [50]
lst[:-1]     # [10, 20, 30, 40]
lst[1:-1]    # [20, 30, 40]
lst[::2]     # [10, 30, 50]
lst[::-1]    # [50, 40, 30, 20, 10]
lst[::-2]    # [50, 30, 10]
lst[:]       # 原序列的影子拷贝
lst[-3:-1]   # [30, 40]
```

Dict 字典

- 列表可以看做用整数做下标（索引），而字典可以用任何类型做下标（索引）
- 字典每个元素包含两部分：键和值
- 字典键或者值都是无序的

```
dic1 = {'color': 'gray', 'age': 17, 7: {3, 2}}
dic2 = dict(zip(k, val))  # k、val 是两个列表
dic3 = dict([('abc',1),(1,'abc')])
                    # 结果 {'abc': 1, 1: 'abc'}
a = dic1['age']     # 获取某个键对应的值，注意若'age'
                      键不存在，则出错KeyError
a = get('age', 0) # 获取键值的建议用法，当键不存在时
                    返回给定的默认值；如果未给
                    定默认值，则返回None
dic1['gen'] = 19  # 插入键值时，当键不存在则相当于新
                    增；键存在时相当于修改
dic1.setdefault('gen', 19)  # 插入，键存在时不更新
dic1.update(dic2) # 用 dic2 更新 dic1，键存在的更
                  # 新，键不存在的插入
dic1.keys()       # 返回键列表,还有 dic1.values()
dic1.items()      # 返回键值元组列表
dic1.pop('age')   # 删除键和值，并返回对应的值
dic1.popitem()    # 删除并返回最后一对键和值
del dic1['age']   # 同 pop() 但没有返回值
dic1.clear()      # 删除整个字典
```

Set 集合

- 集合由无需、不重复的任意元素构成
- 可以把字典看作只有值（没有键）的字典
- 注意不能用下标引用元素

```
a = set([3,6,3])
a = {3, 3, 6}    # 结果都是 {3,6}
a.union(b)       # 合并，相当于 a=a|b，交集是 &
                   差集是 -，不重复差集是 ^
a.update(b)      # 同上，可以多个集合
a.add(5)         # 添加元素
a.discard(7)     # 删除元素
```

Python 语 法 手 册

基础知识 B

Reference: datasciencefree.com xaero@msn.cn

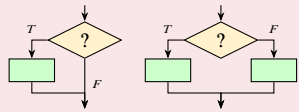
流程控制

条件判断语句 if

当且仅当 if 的条件为真时，执行程序块

```
if 条件:
    程序块

if 条件:
    程序块1
else:
    程序块2
```



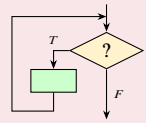

```
if age <= 18:
    state = "Kid"
elif age > 65:
    state = "Retired"
else:
    state = "Active"
```

条件循环语句 while

当且仅当条件为 True 时，执行循环程序块

```
while 条件:
    程序块
```

$s = \sum_{i=1}^{100} i^2$



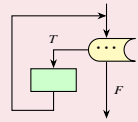

```
i = s = 0
while i < 100:
    i += 1
    s += i ** 2
```

枚举循环语句 for

针对容器、序列中的每一个条目，执行程序块

```
for var in 序列:
    程序块
```

假设下列例子中：
lst = [10, 30, 50, 70, 90]



- 循环指定次数（计数循环）：

```
for i in range(0, 5):
    print(i)        # 0 1 2 3 4
```

- 字符串的遍历：

```
s = "SomE text"
cnt = 0
for c in s:
    if c == 'e':
        cnt += 1
print("Found", cnt, "'e'")   # Found 1 'e'
```

- 列表（切片）的遍历：

```
for item in lst[3:]:
    print(item)     # 50 70 90
```

- 用下标遍历列表（切片）：

```
for i in range(len(lst)):
    print(lst[i])   # 10 30 50 70 90
```

- 用下标遍历列表（切片），方式二：

```
for i, val in enumerate(lst):
    print(i, val, lst[i])
# i 是下标，val 和 lst[i] 是同一个值
```

- 多个列表一起遍历：

```
name = ['Pete', 'John', 'Elizabeth']
age = [6, 23, 44, 13, 19]
for n, a in zip(name, age):
    print(n, a)
# 依次遍历，但 13、19 不会遍历到
```

- 字典的遍历（默认是键）：

```
myCat = {'size': 'fat', 'color': 'gray',
         'disposition': 'loud'}
for x in myCat:
    print(x)  # size color disposition
# 输出同上
for k in myCat.keys():
    print(k)
```

- 字典的遍历（遍历值）：

```
myCat = {'size': 'fat', 'color': 'gray',
         'disposition': 'loud'}
for v in myCat.values():
    print(v)  # fat gray loud
```

- 字典的遍历（同时遍历键和值）：

```
myCat = {'size': 'fat', 'color': 'gray',
         'disposition': 'loud'}
for k, val in myCat.items():
    print(k, val)
```

- 集合的遍历（同列表，但输出不一定按顺序）：

```
age = {6, 23, 44, 53, 59}
for a in age:
    print(a)  # 可能是 6 44 23 59 55
```

跳出循环

break 跳出当前循环
continue 结束当前循环，继续下一次循环
sys.exit() 结束整个程序，需要 import sys

捕获错误

```
try:
    正常处理程序块
except Exception as e:
    出错时执行的程序块
```

函数

自定义函数

```
def 函数名(x, y, z):
    """函数说明"""
    程序块
    return 返回值
```

- 函数的作用范围是模块内部
- x、y、z 是参数，多个参数逗号分隔，有顺序关系
- 参数和函数内部定义的变量，只在函数内部有效
- 返回值可以省略，省略后返回 None 值

匿名函数

```
sumf = lambda x, y : x * 2 + y
print(sumf(7, 3))            # 17 (7*2+3)
```

常用内置函数

len(x)	返回容器 x 中元素个数
min(x)	返回容器 x 中元素最小值
max(x)	返回容器 x 中元素最大值
sum(x)	返回容器 x 中元素之和
sorted(x)	返回 x 排好序的新结果
enumerate(x)	返回迭代器 (index, value)
zip(x1,x2..)	返回相同索引位 x_i 的元组
all(x)	x 中所有元素都 True，返回 True
any(x)	x 中任一元素为 True，返回 True
reversed(x)	返回 x 反向的迭代器
range(x)	返回 $[0, x)$ 区间内的连续整数
range(x,y)	返回 $[x, y)$ 区间内的连续整数
range(x,y,s)	$x, x+s, x+2s, x+3s, \cdots$

数学函数

```
import math
math.sin(math.pi / 2)        # 1.0
math.cos(math.pi / 2)        # 约为 0
math.sqrt(81)                # 9.0
math.log(e ** 2)             # 2.0
math.ceil(12.3)              # 13
math.floor(12.6)             # 12
```

字符串

字符串基础和转义符

```
a = "Hello"
b = "Python"
print(a+b)          # HelloPython
print(a * 2)        # HelloHello
print('\n')         # 输出换行符，“\” 是转义
print(r'\n')        # 输出\n，r 表示 Raw String
```

字符串处理函数

```
a = ['h', 'e', 'vt']
b = "Hello P ython"
```

```
''.join(a)          # 用空字符连接，结果 hevt
'-'.join(a)         # h-e-v-t
b.strip()           # 去除头尾空格（不包括中间）
b.split()           # 用指定字符分隔字符串，默认是空格
                    结果是['Hello', 'P', 'ython']
'7'.zfill(3)        # 007
s.capitalize()      # 首字母大写
s.upper()           # 全大写，类似的 lower()
s.isupper()         # 是否全大写，islower()
s.isalpha()         # 是否是字母
s.isalnum()         # 是否仅含字母或数字字符
s.isspace()         # 是否是空格、回车、Tab 等空白符
s.count(a)          # 返回 a 在 s 中出现的次数
s.find(a)           # a 在 s 中出现的下标，否则-1
s.replace(a, b)     # 用 b 替换 s 中的 a
```

字符串格式化

```
s = "我叫 %s 今年 %d 岁!" % ('Pete', 12)
字符：c，整数：d，十六进制：x，字符串：s
```

```
"{格式模型}".format(模型对应值)
"{} {}".format("ab", "cd")          # 按默认顺序
"{1} {0} {1}".format("ab", "cd")    # 设置指定位置
a = {"name": "XOJ", "url": "https://xoj.red"}
"网站：{name}，地址{url}".format(**a)  # 用字典
a = ['XOJ', 'https://xoj.red']
```

输入、输出和文件操作

```
s = input("提示符")    # 提示符可省略，返回 str
print(a, b, sep=' ', end='\n')
                       # sep 是分隔符，默认空格
                       # end 结束符，默认换行
```

文件读写

```
f = open("file.txt", "r", encoding="utf8")
                       # 只读 r，可写 w，附加 a
f.read(n)              # 读取下 n 个字符
f.readlines(n)         # 返回 n 行组成的列表
f.readline()           # 读下一行
f.write(s)             # 字符串 s 写入
f.writelines(s)        # 列表 s 写入，不换行
f.close()              # 保存并关闭文件
# 一般的读操作，用 with 程序块，它会自动关闭
with open(...) as f:
    for line in f:
        # 处理 line
```

输入输出重定向

```
import sys
f = open("file.txt", "w")
org = sys.stdout       # 可以先保存好
sys.stdout = f         # 重定向到标志输出
print(...)             # 会输出到文件
f.close()
sys.stdout = org       # 还原标准输出
```

pandas常用操作手册

https://pandas.pydata.org/pandas-docs/stable/reference/frame.html

xaero@msn.cn

数据结构

Series

类似 Python 的一维列表，形态结构类似 Excel 的一列，与列表相比，Series 多了一个索引列“index”，若没有指定 index，则默认是从 0 递增的整数

- 创建 Series

```
# 用列表创建，index 可以指定
s1 = pd.Series([1,2], index = ['A','B'])
# 用字典创建，键名当做 index
s2 = pd.Series({'A':1, 'B': 2})
```

- 获取 Series 的值

```
s1.values
```

- 获取 Series 的索引

```
s1.index
```

DataFrame

二维表格形式的数据结构，可以看成多列共享一个 index 的 Series 集合

- 创建 DataFrame

```
# 用字典创建，index 值默认是 0 开始的整数
dic = {'标识':['Apple','Microsoft','Tencent'],
       '名称':['苹果','微软','腾讯'],
       '市值':[158100,136800,45100]}
df1 = pd.DataFrame(dic)
'''名称        标识       市值
0  苹果       Apple  158100
1  微软   Microsoft  136800
2  腾讯     Tencent   45100
'''
# 指定索引值
df2 = pd.DataFrame(dic, index=['row1',
                               'row2','row3'])
'''         标识   名称      市值
row1      Apple   苹果  158100
row2  Microsoft   微软  136800
row3    Tencent   腾讯   45100
'''
# 指定列的顺序
df3 = pd.DataFrame(dic, columns=['名称',
                                 '市值','标识'])
'''  名称      市值       标识
0  苹果   158100      Apple
1  微软   136800  Microsoft
2  腾讯    45100    Tencent
'''
```

- 获取各个列的名称

```
df1.columns
```

- 获取各个行的名称（索引值）

```
df1.index
```

- 获取所有值

```
df1.values
```

- 行列转置

```
df1.T
```

常用操作

获取指定数据

- 预览前（后）5 行数据

```
df1.head(x)  # x 表示行数，省略表示 5
df1.tail(x)  # 同上
```

- DataFrame 切片

```
df1[0:2]  # 第 0、1 两行记录
```

- 通过列标题获取数据

```
df1['名称']            # 一列数据
df1[['名称','标识']]   # 多列列数据
```

loc、iloc、at、iat

- df.loc[行标签, 列标签]，其中行列标签可以是:
  - 单一标签，如 5 或'a'
  - 标签列表，如 ['a', 'b', 'c']
  - 标签切片，如'a' : 'f'
- df.iloc[行号, 列号], 其中行号和列号必须都是整数:
  - 单一整数，如 5
  - 整数列表，如 [4, 3, 0]
  - 整数切片，如 1 : 7
- at = loc，只不过是单一单元格
- iat = iloc，也是单一单元格

```
df2.loc['row1']          # df2 的第 0 行
df1.loc[0,'名称']        # 第 0 行，第“名称”列
df1.loc[0:2,'名称']      # 第 0-2 行，第“名称”列
df2.iloc[0:2,'名称']     # 第 0-1 行，第“名称”列
# 取第 0、1 行，第 0、2 列
df1.loc[[True,True,False],[True,False,True]]
# 取第 row1 行到 row3 行，所有列
df1.loc['row1':'row3']
# 取一个单元格的值
df2.at['row1', '名称']
```

统计函数

```
df1.count()          # 每列的数据个数
df1.sum()            # 每列的总和
df1.mean()           # 每列的平均值
df1.max()            # 每列的最大值
df1.min()            # 每列的最小值
df1.describe()       # 每列的统计值
'''                市值
count       3.000000
mean   113333.333333
std     60043.845091
min     45100.000000
25%     90950.000000
50%    136800.000000
75%    147450.000000
max    158100.000000
'''
```

插入数据

```
# 在末尾插入行，ignore_index=True 可以忽略索引号
df1.append({'标识':'Alibaba','名称':'阿里巴巴'})
# 在第 0 列后插入新列，列标签是 newcol
df1.insert(1, "newcol", [5,10])
```

删除数据

```
# 删除两列，axis=1 表示这两个标签是列标签
df1.drop(['名称', '标识'], axis=1)
# 功能同上
df1.drop(columns=['名称', '标识'], axis=1)
# 删除第 0、5 两行数据；删除多行 drop(range(0,99))
df1.drop([0,5])
# 删除第 0 行数据
df1.drop(0)
```

修改数据

前面取数据的方式都可以完成修改数据，只要将符合规则的数据项赋值给对应的“取数”项即可

```
df1.at[0, '名称'] = '苹果梨'
# 注意教材上的 set_value() 现在已经淘汰
df1.set_value(0, '名称', '苹果葡萄')
```

修改列名或索引

```
# 用字典改名，键是老标签名称，值是新标签名称
df1.rename(columns={'名称': '新名称',
                    '标识': '新标识'})
# 修改索引
df2.rename(index={'row1': '第1行',
                  'row2': '某行'})
```

排序

```
# 按名称排列（默认升序）
df1.sort_values(by=['名称'])
# 按名称降序排序
df1.sort_values(by='名称', ascending=False)
# 先按名称升序，若相同再按标识升序排序
df1.sort_values(by=['名称','标识'])
```

分类汇总

groupby() 对列或行中的数据进行分组，然后对每组数据进行操作

```
df = pd.DataFrame(
       {'Animal': ['鹰', '鹰', '禽', '禽'],
        'Name': ['秃鹰', '隼', '雁', '鸭'],
        'Max Speed': [200, 390, 60, 1]})
''' df的结果是：
     Animal Name  Max Speed
0       鹰    秃鹰       200
1       鹰     隼        390
2       禽     雁        60
3       禽     鸭        1
'''
df.groupby('Animal').mean()
'''Animal Max Speed
禽             30.5
鹰            295.0
'''
# 统计每种动物的个数
df.groupby('Animal').count()
'''Animal  Name  Max Speed
禽           2           2
鹰           2           2
'''
# 上例每列都统计了，没必要；统计单列可以：
df.groupby("Animal")['Name'].count()
'''Animal
禽    2
鹰    2
Name: Name, dtype: int64
'''
```

文件读写

Excel 文件读写

```
# 提供 Excel 文件名即可打开
# sheet_name 是指第几张工作表，默认是 0，即第 1 张
pd.read_excel("文件名.xlsx"[, sheet_name=0])

# 提供 Excel 文件名即可新建，并写入 df 中的数据
df.to_excel("文件名.xlsx")
```

csv 文件读写

“.csv”文件可以认为是文本文件，默认用逗号分隔 Excel 中的每列数据

```
# sep 表示列分隔符是什么，默认是逗号
pd.read_csv("文件名.csv"[, sep=","])

# 将 df 写入 csv 文件
df.to_csv("文件名.csv")
```

Python 数据科学 *速查表*
Pandas 基础

Pandas

Pandas 是基于 Numpy 创建的 Python 库，为 Python 提供了易于使用的**数据结构**和**数据分析**工具。

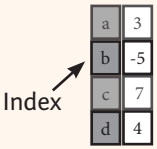
使用以下语句导入 Pandas 库：

```
>>> import pandas as pd
```

Pandas 数据结构

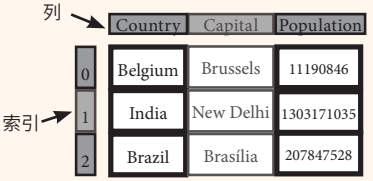
Series - 序列

存储任意类型数据的**一维**数组




```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame - 数据框



存储不同类型数据的**二维**数组

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

输入/输出

读取/写入CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

读取/写入Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

读取内含多个表的Excel

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

调用帮助

```
>>help(pd.Series.loc)
```

选择

参阅 NumPy Arrays

取值

```
>>> s['b']
  -5
```

取序列的值

```
>>> df[1:]
    Country     Capital  Population
  1   India  New Delhi  1303171035
  2  Brazil   Brasília   207847528
```

取数据框的子集

选取、布尔索引及设置值

按位置

```
>>> df.iloc[[0],[0]]
  'Belgium'
>>> df.iat([0],[0])
  'Belgium'
```

按行与列的位置选择某值

按标签

```
>>> df.loc[[0], ['Country']]
  'Belgium'
>>> df.at([0], ['Country'])
  'Belgium'
```

按行与列的名称选择某值

按标签/位置

```
>>> df.ix[2]
  Country      Brazil
  Capital    Brasília
  Population  207847528
```

选择某行

```
>>> df.ix[:,'Capital']
  0     Brussels
  1    New Delhi
  2     Brasília
```

选择某列

```
>>> df.ix[1,'Capital']
  'New Delhi'
```

布尔索引

```
>>> s[~(s > 1)]
```

序列 S 中没有大于1的值

```
>>> s[(s < -1)  |  (s > 2)]
```

序列 S 中小于-1或大于2的值

```
>>> df[df['Population']>1200000000]
```

使用筛选器调整数据框

设置值

```
>>> s['a'] = 6
```

将序列 S 中索引为 a 的值设为6

读取和写入 SQL 查询及数据库表

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

read_sql()是 read_sql_table() 与 read_sql_query()的便捷打包器

```
>>> pd.to_sql('myDf', engine)
```

删除数据

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```

按索引删除序列的值 (axis=0)
按列名删除数据框的列(axis=1)

排序和排名

```
>>> df.sort_index()
>>> df.sort_values(by='Country')
>>> df.rank()
```

按索引排序
按某列的值排序
数据框排名

查询序列与数据框的信息

基本信息

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```

(行,列))
获取索引
获取列名
获取数据框基本信息
非Na值的数量

汇总

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idxmin()/df.idxmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

合计
累计
最小值除以最大值
索引最小值除以索引最大值
基础统计数据
平均值
中位数

应用函数

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

应用匿名函数lambda
应用函数
对每个单元格应用函数

数据对齐

内部数据对齐

如有不一致的索引，则使用NA值：

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
  a     10.0
  b      NaN
  c      5.0
  d      7.0
```

使用 Fill 方法运算

还可以使用 Fill 方法进行内部对齐运算：

```
>>> s.add(s3, fill_value=0)
  a    10.0
  b    -5.0
  c    5.0
  d    7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

Python 数据科学 *速查表*

Pandas 进阶

数据重塑

透视

```
>>> df3= df2.pivot(index='Date',
                   columns='Type',
                   values='Value')
```

将行变为列

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

Type	a	b	c
Date			
2016-03-01	11.432	NaN	20.784
2016-03-02	1.303	13.031	NaN
2016-03-03	99.906	NaN	20.784

透视表

```
>>> df4 = pd.pivot_table(df2,
                         values='Value',
                         index='Date',
                         columns='Type'])
```

将行变为列

堆栈 / 反堆栈

```
>>> stacked = df5.stack()
>>> stacked.unstack()
```

透视列标签
透视索引标签

		0	1
1	5	0.233482	0.390959
2	4	0.184713	0.237102
3	3	0.433522	0.429401

反堆栈

1	5	0	0.233482
		1	0.390959
2	4	0	0.184713
		1	0.237102
3	3	0	0.433522
		1	0.429401

堆栈

融合

```
>>> pd.melt(df2,
            id_vars=["Date"],
            value_vars=["Type", "Value"],
            value_name="Observations")
```

将列转为行

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

	Date	Variable	Observations
0	2016-03-01	Type	a
1	2016-03-02	Type	b
2	2016-03-01	Type	c
3	2016-03-03	Type	a
4	2016-03-02	Type	a
5	2016-03-03	Type	c
6	2016-03-01	Value	11.432
7	2016-03-02	Value	13.031
8	2016-03-01	Value	20.784
9	2016-03-03	Value	99.906
10	2016-03-02	Value	1.303
11	2016-03-03	Value	20.784

迭代

```
>>> df.iteritems()
>>> df.iterrows()
```

（列索引，序列）键值对
（行索引，序列）键值对

高级索引

参阅 NumPy Arrays

基础选择

```
>>> df3.loc[:,(df3>1).any()]
>>> df3.loc[:,(df3>1).all()]
>>> df3.loc[:,df3.isnull().any()]
>>> df3.loc[:,df3.notnull().all()]
```

选择任一值大于1的列
选择所有值大于1的列
选择含 NaN值的列
选择不含NaN值的列

通过isin选择

```
>>> df[(df.Country.isin(df2.Type))]
>>> df3.filter(items="a","b"])
>>> df.select(lambda x: not x%5)
```

选择为某一类型的数值
选择特定值
选择指定元素

通过Where选择

```
>>> s.where(s > 0)
```

选择子集

通过Query选择

```
>>> df6.query('second > first')
```

查询DataFrame

设置/取消索引

```
>>> df.set_index('Country')
>>> df4 = df.reset_index()
>>> df = df.rename(index=str,
                   columns={"Country":"cntry",
                            "Capital":"cptl",
                            "Population":"ppltn"})
```

设置索引
取消索引
重命名DataFrame列名

重置索引

```
>>> s2 = s.reindex(['a','c','d','e','b'])
```

前向填充

```
>>> df.reindex(range(4),
               method='ffill')
      Country     Capital   Population
0   Belgium    Brussels    11190846
1     India   New Delhi    1303171035
2    Brazil    Brasília     207847528
3    Brazil    Brasília     207847528
```

后向填充

```
>>> s3 = s.reindex(range(5),
                   method='bfill')
0    3
1    3
2    3
3    3
4    3
```

多重索引

```
>>> arrays = [np.array([1,2,3]),
              np.array([5,4,3])]
>>> df5 = pd.DataFrame(np.random.rand(3, 2), index=arrays)
>>> tuples = list(zip(*arrays))
>>> index = pd.MultiIndex.from_tuples(tuples,
                                      names=['first', 'second'])
>>> df6 = pd.DataFrame(np.random.rand(3, 2), index=index)
>>> df2.set_index(["Date", "Type"])
```

重复数据

```
>>> s3.unique()
>>> df2.duplicated('Type')
>>> df2.drop_duplicates('Type', keep='last')
>>> df.index.duplicated()
```

返回唯一值
查找重复值
去除重复值
查找重复索引

数据分组

聚合

```
>>> df2.groupby(by=['Date','Type']).mean()
>>> df4.groupby(level=0).sum()
>>> df4.groupby(level=0).agg({'a':lambda x:sum(x)/len(x),
                              'b': np.sum})
```

转换

```
>>> customSum = lambda x: (x+x%2)
>>> df4.groupby(level=0).transform(customSum)
```

缺失值

```
>>> df.dropna()
>>> df3.fillna(df3.mean())
>>> df2.replace("a", "f")
```

去除缺失值NaN
用预设值填充缺失值NaN
用一个值替换另一个值

合并数据

数据1

X1	X2
a	11.432
b	1.303
c	99.906

数据2

X1	X3
a	20.784
b	NaN
d	20.784

合并-Merge

```
>>> pd.merge(data1,
             data2,
             how='left',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN

```
>>> pd.merge(data1,
             data2,
             how='right',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
d	NaN	20.784

```
>>> pd.merge(data1,
             data2,
             how='inner',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN

```
>>> pd.merge(data1,
             data2,
             how='outer',
             on='X1')
```

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN
d	NaN	20.784

连接-Join

```
>>> data1.join(data2, how='right')
```

拼接-Concatenate

纵向

```
>>> s.append(s2)
```

横向/纵向

```
>>> pd.concat([s,s2],axis=1, keys=['One','Two'])
>>> pd.concat([data1, data2], axis=1, join='inner')
```

日期

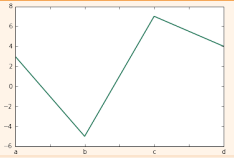
```
>>> df2['Date']= pd.to_datetime(df2['Date'])
>>> df2['Date']= pd.date_range('2000-1-1',
                               periods=6,
                               freq='M')
>>> dates = [datetime(2012,5,1), datetime(2012,5,2)]
>>> index = pd.DatetimeIndex(dates)
>>> index = pd.date_range(datetime(2012,2,1), end, freq='BM')
```

可视化

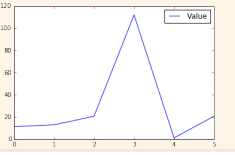
参阅 Matplotlib

```
>>> import matplotlib.pyplot as plt
```

```
>>> s.plot()
>>> plt.show()
```



```
>>> df2.plot()
>>> plt.show()
```



Python 数据科学 *速查表*
Numpy 基础

NumPy

Numpy 是 Python 数据科学计算的核心库，提供了高性能的多维数组对象及处理数组的工具。

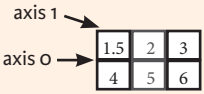
使用以下语句导入 Numpy 库：

```
>>> import numpy as np
```

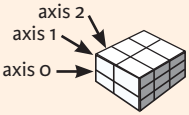
NumPy 数组

1维数组

2维数组



3维数组



创建数组

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

初始化占位符

`>>> np.zeros((3,4))`	创建值为0数组
`>>> np.ones((2,3,4),dtype=np.int16)`	创建值为1数组
`>>> d = np.arange(10,25,5)`	创建均匀间隔的数组（步进值）
`>>> np.linspace(0,2,9)`	创建均匀间隔的数组（样本数）
`>>> e = np.full((2,2),7)`	创建常数数组
`>>> f = np.eye(2)`	创建2x2单位矩阵
`>>> np.random.random((2,2))`	创建随机值的数组
`>>> np.empty((3,2))`	创建空数组

输入/输出

保存与载入磁盘上的文件

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

保存与载入文本文件

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

数据类型

`>>> np.int64`	带符号的64位整数
`>>> np.float32`	标准双精度浮点数
`>>> np.complex`	显示为128位浮点数的复数
`>>> np.bool`	布尔值：True值和False值
`>>> np.object`	Python对象
`>>> np.string_`	固定长度字符串
`>>> np.unicode_`	固定长度Unicode

数组信息

```
>>> a.shape
>>> len(a)
>>> b.ndim
>>> e.size
>>> b.dtype
>>> b.dtype.name
>>> b.astype(int)
```

数组形状，几行几列
数组长度
几维数组
数组有多少元素
数据类型
数据类型的名字
数据类型转换

调用帮助

```
>>> np.info(np.ndarray.dtype)
```

数组计算

算数运算

```
>>> g = a - b
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)
>>> b + a
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)
>>> a / b
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)
>>> a * b
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)
>>> np.exp(b)
>>> np.sqrt(b)
>>> np.sin(a)
>>> np.cos(b)
>>> np.log(a)
>>> e.dot(f)
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

减法

减法
加法

加法
除法

除法
乘法

乘法
幂
平方根
正弦
余弦
自然对数
点积

比较

```
>>> a == b
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)
```

对比值

对比值

对比数组

聚合函数

```
>>> a.sum()
>>> a.min()
>>> b.max(axis=0)
>>> b.cumsum(axis=1)
>>> a.mean()
>>> b.median()
>>> a.corrcoef()
>>> np.std(b)
```

数组汇总
数组最小值
数组最大值，按行
数组元素的累加值
平均数
中位数
相关系数
标准差

数组复制

```
>>> h = a.view()
>>> np.copy(a)
>>> h = a.copy()
```

使用同一数据创建数组视图
创建数组的副本
创建数组的深度拷贝

数组排序

```
>>> a.sort()
>>> c.sort(axis=0)
```

数组排序
以轴为依据对数组排序

子集、切片、 索引

参阅 列表

子集

```
>>> a[2]
  3
```

选择索引2对应的值

```
>>> b[1,2]
  6.0
```

选择行1列2对应的值（等同于b[1][2]

切片

```
>>> a[0:2]
  array([1, 2])
```

选择索引为0与1对应的值

```
>>> b[0:2,1]
  array([ 2.,  5.])
```

选择第1列中第0行、第1行的值

```
>>> b[:1]
  array([[1.5, 2., 3.]])
```

选择第0行的所有值（等同于b[0:1,:1]

```
>>> c[1,...]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
```

等同于 [1,:,:]

```
>>> a[ : :-1]
  array([3, 2, 1])
```

反转数组a

条件索引

```
>>> a[a<2]
  array([1])
```

选择数组a中所有小于2的值

花式索引

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]
  array([ 4. , 2. , 6. , 1.5])
```

选择(1,0),(0,1),(1,2) 和(0,0)所对应的值

```
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]
  array([[ 4. ,  5. ,  6. ,  4. ],
         [ 1.5,  2. ,  3. ,  1.5],
         [ 4. ,  5. ,  6. ,  4. ],
         [ 1.5,  2. ,  3. ,  1.5]])
```

选择矩阵的行列子集

数组操作

转置数组

```
>>> i = np.transpose(b)
>>> i.T
```

转置数组
转置数组

改变数组形状

```
>>> b.ravel()
```

拉平数组

```
>>> g.reshape(3,-2)
```

改变数组形状，但不改变数据

添加或删除值

```
>>> h.resize((2,6))
>>> np.append(h,g)
>>> np.insert(a, 1, 5)
>>> np.delete(a,[1])
```

返回形状为(2,6)的新数组
追加数据
插入数据
删除数据

合并数组

```
>>> np.concatenate((a,d),axis=0)
  array([ 1,  2,  3, 10, 15, 20])
```

拼接数组

```
>>> np.vstack((a,b))
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
```

纵向以行的维度堆叠数组

```
>>> np.r_[e,f]
```

纵向以行的维度堆叠数组

```
>>> np.hstack((e,f))
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
```

横向以列的维度堆叠数组

```
>>> np.column_stack((a,d))
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
```

以列的维度创建堆叠数组

```
>>> np.c_[a,d]
```

以列的维度创建堆叠数组

分割数组

```
>>> np.hsplit(a,3)
  [array([1]),array([2]),array([3])]
```

纵向分割数组为3等份

```
>>> np.vsplit(c,2)
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

横向分割数组为2等份

Python数据科学 *速查表*

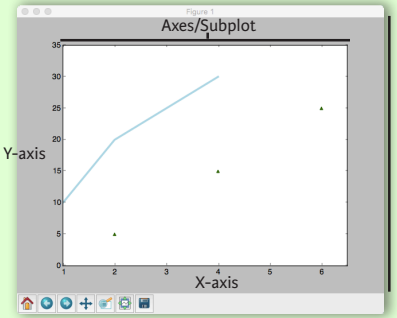
Matplotlib

Matplotlib

Matplotlib 是 Python 的二维绘图库，用于生成符合出版质量或跨平台交互环境的各类图形。

图形解析与工作流

图形解析



工作流

Matplotlib 绘图的基本步骤：

1 准备数据　2 创建图形　3 绘图　4 自定义设置　5 保存图形　6 显示图形

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                                  Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                             Step 2
>>> ax = fig.add_subplot(111)                      Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)  Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                                     Step 6
```

1 准备数据

参阅 列表与 NumPy

一维数据

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

二维数据或图片

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 绘制图形

```
>>> import matplotlib.pyplot as plt
```

画布

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

坐标轴

图形是以坐标轴为核心绘制的，大多数情况下，子图就可以满足需求。子图是栅格系统的坐标轴。

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

3 绘图例程

一维数据

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)
>>> ax.scatter(x,y)
>>> axes[0,0].bar([1,2,3],[3,4,5])
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])
>>> axes[1,1].axhline(0.45)
>>> axes[0,1].axvline(0.65)
>>> ax.fill(x,y,color='blue')
>>> ax.fill_between(x,y,color='yellow')
```

用线或标记连接点
缩放或着色未连接的点
绘制等宽纵向矩形
绘制等高横向矩形
绘制与轴平行的横线
绘制与轴垂直的竖线
绘制填充多边形
填充y值和0之间

二维数据或图片

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

色彩表或RGB数组

向量场

```
>>> axes[0,1].arrow(0,0,0.5,0.5)
>>> axes[1,1].quiver(y,z)
>>> axes[0,1].streamplot(X,Y,U,V)
```

为坐标轴添加箭头
二维箭头
二维箭头

数据分布

```
>>> ax1.hist(y)
>>> ax3.boxplot(y)
>>> ax3.violinplot(z)
```

直方图
箱形图
小提琴图

```
>>> axes2[0].pcolor(data2)
>>> axes2[0].pcolormesh(data)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2]= ax.clabel(CS)
```

二维数组伪彩色图
二维数组等高线伪彩色图
等高线图
等高线图标签

4 自定义图形

颜色、色条与色彩表

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

标记

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

线型

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

文本与标注

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

数学符号

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

尺寸限制、图例和布局

尺寸限制与自动调整

```
>>> ax.margins(x=0.0,y=0.1)
```
添加内边距

```
>>> ax.axis('equal')
```
将图形纵横比设置为1

```
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])
```
设置x轴与y轴的限制

```
>>> ax.set_xlim(0,10.5)
```
设置x轴的限制

图例

```
>>> ax.set(title='An Example Axes',
           ylabel='Y-Axis',
           xlabel='X-Axis')
```
设置标题与x、y轴的标签

```
>>> ax.legend(loc='best')
```
自动选择最佳的图例位置

标记

```
>>> ax.xaxis.set(ticks=range(1,5),
                 ticklabels=[3,100,-12,"foo"])
```
手动设置X轴刻度

```
>>> ax.tick_params(axis='y',
                   direction='inout',
                   length=10)
```
设置Y轴长度与方向

子图间距

```
>>> fig3.subplots_adjust(wspace=0.5,
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
```
调整子图间距

```
>>> fig.tight_layout()
```
设置画布的子图布局

坐标轴边线

```
>>> ax1.spines['top'].set_visible(False)
```
隐藏顶部坐标轴线

```
>>> ax1.spines['bottom'].set_position(('outward',10))
```
设置底部边线的位置为outward

5 保存

保存画布

```
>>> plt.savefig('foo.png')
```

保存透明画布

```
>>> plt.savefig('foo.png', transparent=True)
```

6 显示图形

```
>>> plt.show()
```

关闭与清除

`>>> plt.cla()`	清除坐标轴
`>>> plt.clf()`	清除画布
`>>> plt.close()`	关闭窗口