学习目标:

- 掌握开发环境的使用
- 了解简单的程序

学习重点:

- 开发环境的使用

介绍

- 为什么要学习 Python ?

VB.NET 之父

JAVA之父

Objective-C 之父

C++ 之父

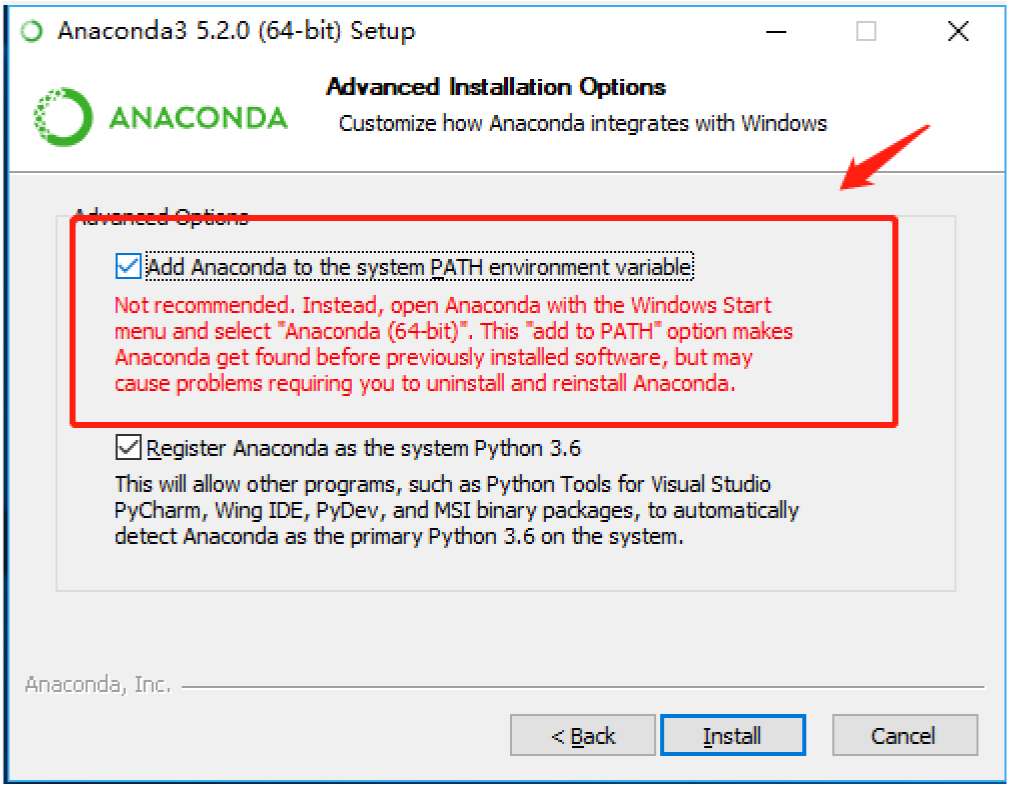
Golang 之父

PHP 之父

JavaScript 之父

Python 之父

安装 **Anaconda**


Anaconda3 5.2.0 (64-bit) Setup

Advanced Installation Options

Customize how Anaconda integrates with Windows

Advanced Options

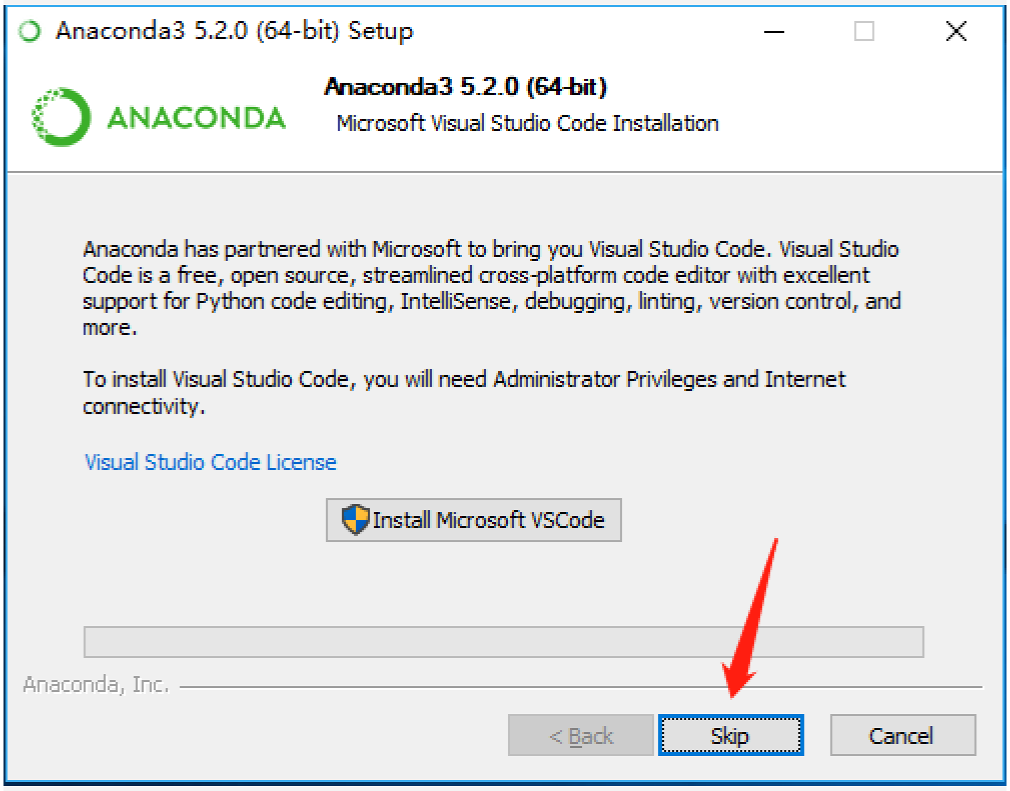
☑ Add Anaconda to the system PATH environment variable

Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.

☑ Register Anaconda as the system Python 3.6

This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.6 on the system.

Anaconda, Inc.


Anaconda3 5.2.0 (64-bit) Setup

Anaconda3 5.2.0 (64-bit)

Microsoft Visual Studio Code Installation

Anaconda has partnered with Microsoft to bring you Visual Studio Code. Visual Studio Code is a free, open source, streamlined cross-platform code editor with excellent support for Python code editing, IntelliSense, debugging, linting, version control, and more.

To install Visual Studio Code, you will need Administrator Privileges and Internet connectivity.

Visual Studio Code License

Anaconda, Inc.

为什么要学习 **Python?**

- https://www.jianshu.com/p/364c46ebfbbf

开发环境

- Anaconda
- Spyder 和 Jupyter Notebook 的使用

免费学习资源:

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

https://www.liaoxuefeng.com/wiki/1016959663602400

https://docs.python.org/3.7/

Hello World

In [1]:
```
print('Hello World')
```
```
Hello World
```

In [2]:
```
print("Hello World")
```
```
Hello World
```

在 Spyder 以及 Command Line 中运行以上程序

In [2]:
```
#引号嵌套
print("xxxx'Hello'xxxx")
```
```
xxxx'Hello'xxxx
```

In [2]:
```
#一次运行多行代码
print('Hello')
print('Python Programming')
```
```
Hello
Python Programming
```

函数

- 超越单行代码片段，并执行整个语句序列

In [3]:
```
def hello():
    print('Hello')
    print('Python Programming!')
```

- 第一行定义新函数，名称为 hello
- 接下来两行缩进，表明它们是 hello 函数的一部分

In [4]:
```
hello()
```
```
Hello
Computers ar fun!
```

- 函数定义（调用）中的括号是做什么的？ —— 参数

In [5]:
```
def greet(person):
    print('Hello', person)
    print('How are you?')
```

In [6]:
```python
greet('John')
```

```
Hello John
How are you?
```

In [7]:
```python
greet('Emily')
```

```
Hello Emily
How are you?
```

一个稍微复杂的示例程序

In [4]:
```python
# chaos
def main():
    print('This program illustrates a chaotic function')
    x = float(input('Enter a number between 0 and 1: '))  # 获取用户输入
    for i in range(10):   # 循环 10 次 —— 做 10 次相同的事情
        x = 3.9 * x * (1-x)    # 等号表示赋值
        print(x)
```

In [9]:
```python
main()
```

```
This program illustrates a chaotic function
Enter a number between 0 and 1: .25
0.73125
0.76644140625
0.6981350104385375
0.8218958187902304
0.5708940191969317
0.9553987483642099
0.166186721954413
0.5404179120617926
0.9686289302998042
0.11850901017563877
```

- Chaos 程序的有趣之处：初始值的非常小的差别可以导致结果的巨大差异

> 练习
>
> 1. 修改 chaos 程序，让它打印出 20 个值，而不是 10 个
> 2. 修改 chaos 程序，让打印值的数量由用户确定（注意：循环次数需要被转换成 int 而非 float）

学习目标:

- 掌握程序的要素
- 掌握赋值语句
- 掌握确定循环

学习重点:

- 程序的要素

1. 示例程序：汇率换算

In [3]:
```python
def main():
    usd = float(input('Please input the dollar amount: '))
    rmb = usd * 7.5
    print('The RMB amount is ', rmb)

main()
```

```
Please input the dollar amount: 9
The RMB amount is  67.5
```

2. 程序要素

2.1 名称——起名字

名称是编程的重要组成部分，所有名称都成为“标识符”

- 模块名称
- 函数名称(main)
- 变量名称(usd, rmb)

命名规则

- 以字母或下划线开头
- 后跟字母、数字或下划线的任意序列
- 不能包含空格

合法标识符示例

- x
- rmb
- spam
- spam2
- SpamAndEggs
- Spam_and_Eggs

标识符区分大小写

- spam, Spam, SPAM 是不同的标识符

好的程序员总是试图选择一些名字，它们能描述被命名的东西

关键字和内置函数名称不能被用来作为标识符

Python 关键字

- False
- True
- is
- return
- def
- for
- ...

Python 内置函数

- help()
- print()
- abs()
- float()
- int()
- ...

2.2 表达式

产生或计算新数据值的程序代码片段被称为表达式

- 最简单的表达式：字面量——汇率换算中的 7.5
- 字符串表达式：'Please input the dollar amount: '
- 计算表达式：usd * 7.5

将表达式转换为基础数据类型的过程称为“求值”

```
In [7]: 32
Out[7]: 32
```

"Hello"

"32"

- 注意 32 与 "32" 的区别

一个简单的标识符也可以是一个表达式。我们使用标识符作为变量来给名字赋值。

当标识符作为表达式出现时，它的值会被取出，作为表达式的结果。

In [8]:
```
x = 5
x
```
Out[8]:
```
5
```

In [9]:
```
print(x)
```
```
5
```

数学运算

1. +
2. -
3. *
4. /
5. **

遵循优先级和结合律，可以使用小括号改变求值的顺序

数学运算示例

- 3.9 *x* (1-x)
- usd * 7.5

空格在表达式中没有作用，只是为了方便阅读

字符串操作

In [11]:
```
"Bat" + "man"
```
Out[11]:
```
'Batman'
```

3. 输出语句

In [12]:
```
print(3+4)
```
```
7
```

In [13]:
```
print(3, 4, 3+4)
```
```
3 4 7
```

In [14]:
```
print()
```

In [15]:
```
print('The answer is', 3+4)
```
```
The answer is 7
```

4. 赋值语句

4.1 简单赋值

基本形式

- < variable > = < expr >

variable 是一个标识符, expr 是一个表达式。赋值的语义是，右侧的表达式被求值，然后产生的值与左侧命名的变量相关联。

示例

- x = 5
- x = 3.9 *x* (1-x)
- rmb = usd * 7.5

变量可以多次赋值。它总是保留最新赋的值。

```
In [16]: myVar = 0
         myVar
```

```
Out[16]: 0
```

```
In [17]: myVar = 7
         myVar
```

```
Out[17]: 7
```

```
In [18]: myVar = myVar + 1
         myVar
```

```
Out[18]: 8
```

4.2 赋值输入

输入语句的目的是从程序的用户那里获取一些信息，并存储到变量中

- < variable > = input(< prompt >)

- prompt 是一个字符串表达式，用于提示用户输入。
- 当 Python 遇到对 `input` 的调用时，它在屏幕上打印提示。
- 然后，Python 暂停并等待用户输入一些文本，输入完成后按 < Enter > 键。
- 用户输入的任何东西都会存储为字符串。

```
In [19]: name = input('Enter your name:')
         name
```

```
Enter your name:Emily
```

```
Out[19]: 'Emily'
```

如果需要输入的是数字，需要对输入结果进行转换

4.3 同时赋值

In [26]:
```
x = 5
y = 3
summ, diff = x + y, x - y
print('summ =',summ,'    diff =',diff)
```

```
summ = 8     diff = 2
```

如果我们交换 **x** 和 **y** 的值**...**

错误的做法:

- x=y
- y=x

正确的做法

In [27]:
```
temp = x
x = y
y = temp
print('x =',x,'    y =',y)
```

```
x = 3     y = 5
```

更加便捷的做法 (Python 特有)

In [31]:
```
x,y = y,x
print('x =',x,'    y =',y)
```

```
x = 3     y = 5
```

> 练习
>
> - 编写一个程序，将以千米为单位的距离转换为英里。1千米约为0.62英里

5. 确定循环

用循环连续多次执行一系列语句。最简单的循环称为确定循环。在循环开始时，Python 就知道循环次数。

In [1]:
```
#计数循环
x = 0.1
for i in range(10):
    x = 3.9 * x * (1-x)
    print(x)
```

```
0.35100000000000003
0.8884161
0.3866184397170808
0.9248640249724619
0.2710131851083772
0.7705036505625796
0.6896283226260395
```

```
0.8347602871063352
0.5379486456882877
0.9693836111326567
```

Python 的 **for** 循环具有以下一般形式：

for < var > in < sequence >:

< body >

- 关键字 `for` 后面的变量称为 循环索引。它依次取 sequence 中的每个值，并针对每个值都执行一次循环体中的语句。
- 通常，sequence 部分由“列表”构成。

In [2]:
```
for i in [0,1,2,3]:
    print(i)
```

```
0
1
2
3
```

In [3]:
```
for odd in [1,3,5,7,9]:
    print(odd * odd)
```

```
1
9
25
49
81
```

In [4]:
```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

使用 range 的效果等价于使用以下 list

In [5]:
```
for i in [0,1,2,3,4,5,6,7,8,9]:
    print(i)
```

```
0
1
2
3
4
5
6
7
```

```
8
9
```

6. 示例程序：终值

In [7]:

```
#给定本金（principal）和利率（apr），计算 10 年终值
def main():
    print('This program calculates the future value')
    print('of a 10-year investment')

    principal = float(input('Enter the initial principal: '))
    apr = float(input('Enter the anuual interest rate: '))

    for i in range(10):
        principal = principal * (1+apr)

    print('The value in 10 years is:', principal)

main()
```

```
This program calculates the future value
of a 10-year investment
Enter the initial principal: 100
Enter the anuual interest rate: 0.1
The value in 10 years is: 259.37424601000026
```

> 练习
>
> - 修改温度转换器，让它计算并打印一个摄氏温度和华氏温度的对应表，从0(C)到100(C)，每隔10(C)一个值。

> 练习
>
> - 账户所产生的利息通常通过名义利率和复利期数来描述。例如，如果利率为3%，利息按季度计算复利，则该账户实际上每3个月赚取0.75%的利息。请编写程序，用此方法输入利率。程序应提示用户每年的利率（rate）和利息每年复利的次数（periods）。程序计算10年的价值，程序将循环10 x periods次，并在每次迭代中累积 rate/periods 的利息。

学习目标:

- 掌握数值数据类型;
- 掌握数值数据的操作;
- 能够在程序中使用合适的数值类型;
- 掌握 Math 库的使用

学习重点:

- 数值数据类型及其使用

1. 数值数据类型

In [1]:
```
# 整数
x = 3

# 浮点数
pi = 3.14
```

In [2]:
```
# 使用 type 查看数据类型
type(3)
```

Out[2]:
```
int
```

In [3]:
```
type(3.14)
```

Out[3]:
```
float
```

In [4]:
```
type(x)
```

Out[4]:
```
int
```

In [5]:
```
type(pi)
```

Out[5]:
```
float
```

为什么需要 **`int`**?

- 告诉读者程序的值不能是一个分数
- `int` 具有更高运算速度
- `float` 只能表示对实数的近似, `int` 总是精确的

在程序中尽量使用 **int**

Python 内置的数值操作

操作符	操作
+	加
-	减

*	乘
/	浮点除
**	指数
abs()	绝对值
//	整数除
%	取余

```
In [7]: 4 ** 3
Out[7]: 64

In [8]: abs(-3.3)
Out[8]: 3.3

In [9]: 10/3
Out[9]: 3.3333333333333335

In [10]: 10//3
Out[10]: 3
```

2. 类型转换和舍入

```
In [13]: # 显示类型转换
         int(4.5)
Out[13]: 4

In [14]: float(4)
Out[14]: 4.0
```

- 将 float 转换为 int 是一个危险的步骤
- int 可以安全的转换为 float

```
In [16]: # 使用 round 进行舍入操作
         round(3.14)
Out[16]: 3

In [17]: round(3.5)
Out[17]: 4

In [18]: round(3.14159, 2)
Out[18]: 3.14

In [19]: # 将字符串转换为数字
         int('32')
Out[19]: 32
```

In [12]:
```
5.0 * 2
```

Out[12]:
```
10.0
```

将以上代码理解成 **2** 步：

1. 2 -> 2.0 (隐式类型转换)
2. 5.0 * 2.0

3. 使用 Math 库

常用的 Math 库函数

Python	描述
pi	π 的近似值
e	e 的近似值
sqrt(x)	x 的平方根
sin(x)	x 的正弦
cos(x)	x 的余弦
tan(x)	x 的正切
log(x)	x 的自然对数（以 e 为底）
log10(x)	x 的常用对数（以 10 为底）
exp(x)	e 的 x 次方
ceil(x)	最小的 >=x 的整数
floor(x)	最大的 <=x 的整数

二次方程的形式为 $ax^2+bx+c=0$。这样的方程有两个解：$$x=\frac{-b\pm \sqrt{b^2-4ac}}{2a}$$ 编写一个程序，找到二次方程的解

In [24]:
```
import math

def main():
    a = float(input('Enter coefficient a: '))
    b = float(input('Enter coefficient b: '))
    c = float(input('Enter coefficient c: '))

    tmp = math.sqrt(b*b-4*a*c)
    root1 = (-b+tmp)/ (2*a)
    root2 = (-b-tmp)/ (2*a)

    print()
    print('The solutions are:', root1, root2)

main()
```

```
Enter coefficient a: 2
Enter coefficient b: 4
Enter coefficient c: -2

The solutions are: 0.41421356237309515 -2.414213562373095
```

> 练习
>
> - 修改以上程序，不使用 Math 库来求根

4. 示例程序：积累相乘

In [25]:
```
def main():
    n = int(input('Please enter a whole number: '))
    fact = 1
    for factor in range(n,1,-1):
        fact = fact * factor
    print('The factorial of', n, 'is', fact)

main()
```

```
Please enter a whole number: 6
The factorial of 6 is 720
```

> 练习
>
> - 使用以下公式编写程序计算三角形的面积，其三边的长度为a、b 和 c:

$$s=\frac{a+b+c}{2}$$ $$A=\sqrt{s(s-a)(s-b)(s-c)}$$

> 练习
>
> - 编写程序，通过对这个级数的项求和来求近似的 π 的值: 4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + ... 程序应该提示用户输入 n, 要求和的项数，然后输出该级数的前 n 项和。让你的程序从 math.pi 的值中减去近似值，看看它的准确性

> 练习
>
> - 斐波那契序列是数字序列，其中每个连续数字是前两个数字的和。经典的斐波那契序列开始于 1，1，2，3，5，8，13，...。编写计算第 n 个斐波那契序列的程序，其中 n 是用户输入的值。例如，如果 n=6，则结果为 8。

> 练习
>
> - 计算债券价格：编写一个程序，让用户输入债券剩余年限、票面利率、债

> 券面值以及折现利率，计算债券价格（假设每年付息一次）。

学习目标:

- 掌握字符串数据类型
- 掌握字符串的处理
- 掌握文本文件的处理

学习重点:

- 字符串的处理

1 字符串数据类型

In [1]:
```
#使用单引号和双引号都可以定义字符串
str1 = 'Hello'
str2 = "Hello"
```

In [1]:
```
#对于初学者，要记住一个字符串是什么：一个字符序列
greet = 'Hello World'
print(greet[0],greet[2],greet[4])
print(greet[-1],greet[-3])
```

```
H l o
d r
```

In [2]:
```
#索引和切片
greet[0:3]
```

Out[2]:
```
'Hel'
```

In [3]:
```
greet[5:9]
```

Out[3]:
```
' Wor'
```

In [4]:
```
greet[:5]
```

Out[4]:
```
'Hello'
```

In [5]:
```
greet[5:]
```

Out[5]:
```
' World'
```

In [6]:
```
greet[:]
```

Out[6]:
```
'Hello World'
```

Python 字符串操作

操作符	含义
+	连接
*	重复

[]	索引
[:]	切片
len(str)	长度
for s in str	遍历字符串

In [9]:
```
'spam' + 'eggs'
```
Out[9]:
```
'spameggs'
```

In [10]:
```
'egg' * 3
```
Out[10]:
```
'eggeggegg'
```

In [11]:
```
len('eggs')
```
Out[11]:
```
4
```

In [12]:
```
for c in 'Eggs':
    print(c, end=' ')
```
```
E g g s
```

2 简单的字符串处理

根据实际姓名生成用户名的示例：取用户第一个字母，然后是用户姓氏的最多 7 个字母

- Zaphod Beeblebrox -> zbeebleb
- John Smith -> jsmith

In [16]:
```
def main():
    print('This program generates computer usernames.\n')   #注意这里有一个换行符

    first = input('First name (all lowercases): ')
    last = input('Last name (all lowercases): ')

    uname = first[0] + last[:7]

    print('Your username is:', uname)

main()
```
```
This program generates computer usernames.

First name (all lowercases): john
Last name (all lowercases): smith
Your username is: jsmith
```

编写一个程序：打印给定月分数的月份的英文缩写

- 3 -> Mar

In [7]:
```
# month.py
#  A program to print the abbreviation of a month, given its number
```

```
def main():
    # months is used as a lookup table
    months = "JanFebMarAprMayJunJulAugSepOctNovDec"

    n = int(input("Enter a month number (1-12): "))

    # compute starting position of month n in months
    pos = (n-1) * 3

    # Grab the appropriate slice from months
    monthAbbrev = months[pos:pos+3]

    # print the result
    print("The month abbreviation is", monthAbbrev + ".")

main()
```

```
Enter a month number (1-12): 2
The month abbreviation is Feb.
```

思考：以上程序有什么缺点？

> 练习：（要使用 `if` 语句）
>
> - 根据用户输入的股票代码，识别是上市交易所（沪/深），将交易所信息添加至代码后面
>   - 600000 -> 600000.SH
>   - 000001 -> 000001.SZ

3 列表作为序列

In [18]:
```
[1,2] + [3,4]
```
Out[18]:
```
[1, 2, 3, 4]
```

In [20]:
```
[1,2]*3
```
Out[20]:
```
[1, 2, 1, 2, 1, 2]
```

In [21]:
```
grades = ['A', 'B', 'C', 'D', 'F']
grades[0]
```
Out[21]:
```
'A'
```

In [22]:
```
grades[2:4]
```
Out[22]:
```
['C', 'D']
```

In [23]:
```
len(grades)
```
Out[23]:
```
5
```

列表的好处是它们比字符串更通用。字符串总是字符序列，而列表可以是任意对象的序列。你可以

创建数字列表或字符串列表。你甚至可以混合它们。

```
In [24]: myList = [1, "Spam ", 4, "U"]
```

使用列表可以大大简化月份缩写程序

```
In [8]: # 使用列表重写月份缩写程序
        def main():

            # months is a list used as a lookup table
            months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun",
                      "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]

            n = int(input("Enter a month number (1-12): "))

            print ("The month abbreviation is", months[n-1] + ".")

        main()
```

```
Enter a month number (1-12): 2
The month abbreviation is Feb.
```

虽然字符串和列表都是序列，但有重要区别：

- 列表可变
- 字符串不可变

```
In [26]: myList = [34, 26, 15, 10]
         myList[2] = 0
         myList
```

```
Out[26]: [34, 26, 0, 10]
```

```
In [27]: myString = "Hello World"
         myString[2] = 'a'
```

```
------------------------------------------------------------------------
---
TypeError                                 Traceback (most recent call la
st)
<ipython-input-27-afe475befeef> in <module>()
      1 myString = "Hello World"
----> 2 myString[2] = 'a'

TypeError: 'str' object does not support item assignment
```

4 字符串表示和消息编码

4.1 字符串表示

字符串 --> ASCII 编码

```
In [1]: #ASCII 编码
        ord('a')
```

Out[1]:
```
97
```

In [2]:
```
ord('A')
```

Out[2]:
```
65
```

ASCII 编码 --> 字符串

In [3]:
```
chr(97)
```

Out[3]:
```
'a'
```

In [4]:
```
chr(90)
```

Out[4]:
```
'Z'
```

4.2 示例程序：编码器

编写一个程序，将消息转换为数字序列

In [9]:
```
def main():
    print("This program converts a textual message into a sequence")
    print ("of numbers representing the Unicode encoding of the message.
\n")

    # Get the message to encode
    message = input("Please enter the message to encode: ")

    print("\nHere are the Unicode codes:")

    # Loop through the message and print out the Unicode values
    for ch in message:
        print(ord(ch),  end=" ")

    print()   # blank line before prompt
main()
```

```
This program converts a textual message into a sequence
of numbers representing the Unicode encoding of the message.

Please enter the message to encode: Hello World

Here are the Unicode codes:
72 101 108 108 111 32 87 111 114 108 100
```

5 字符串方法

5.1 示例程序：解码器

In [6]:
```
def main():
    print ("This program converts a sequence of Unicode numbers into")
    print ("the string of text that it represents.\n")
    # Get the message to encode
    inString = input("Please enter the Unicode-encoded message: ")

    # Loop through each substring and build Unicde message
    message = ""
    for numStr in inString.split():
        # convert the (sub)string to a number
        codeNum = int(numStr)
        # append character to message
        message = message + chr(codeNum)

    print("\nThe decoded message is:", message)
main()
```

```
This program converts a sequence of Unicode numbers into
the string of text that it represents.

Please enter the Unicode-encoded message: 83 116 32 114

The decoded message is: St r
```

5.2 更多字符串方法

https://docs.python.org/3.7/library/stdtypes.html#string-methods

In [11]:
```
s = 'the market rebound'
```

In [12]:
```
s.capitalize()
```

Out[12]:
```
'The market rebound'
```

In [13]:
```
s.title()
```

Out[13]:
```
'The Market Rebound'
```

In [14]:
```
s.lower()
```

Out[14]:
```
'the market rebound'
```

In [15]:
```
s.upper()
```

Out[15]:
```
'THE MARKET REBOUND'
```

In [16]:
```
s.replace("rebound","drop")
```

Out[16]:
```
'the market drop'
```

In [17]:
```
s.center(50)
```

Out[17]:
```
'                the market rebound                '
```

In [18]:
```
s.count('e')
```

Out[18]:
```
3
```

In [20]:
```
s.find('r')
```

Out[20]:
```
6
```

In [22]:
```
' '.join(['Show','me','the','money'])
```
Out[22]:
```
'Show me the money'
```

In [23]:
```
','.join(['Show','me','the','money'])
```
Out[23]:
```
'Show,me,the,money'
```

6 列表

In [24]:
```
# 创建一个列表，并向其中添加元素
squares = []
for x in range(1,101):
    squares.append(x*x)
```

7 输入/输出

7.1 示例应用程序：日期转换

用户输入一个日期，例如 "05/24/2020"，程序将显示日期为 "May 24, 2020"

In [24]:
```
# dateconvert.py
#   Converts a date in form "mm/dd/yyyy" to "month day, year"

def main():
    # get the date
    dateStr = input("Enter a date (mm/dd/yyyy): ")

    # split into components
    monthStr, dayStr, yearStr = dateStr.split("/")

    # convert monthStr to the month name
    months = ["January", "February", "March", "April",
              "May", "June", "July", "August",
              "September", "October", "November", "December"]
    monthStr = months[int(monthStr)-1]

    # output result in month day, year format
    print("The converted date is:", monthStr, dayStr+",", yearStr)

main()
```
```
Enter a date (mm/dd/yyyy): 05/24/2020
The converted date is: May 24, 2020
```

7.2 字符串格式化

In [25]:
```
print('The stock price is',15.1)
```
```
The stock price is 15.1
```

In [30]:
```
# 如果我们想将股票价格显示为 2 位小数
```

```
print('The stock price is {0:0.2f}'.format(15.1))
```

```
The stock price is 15.10
```

格式

{插槽: 宽度.精度 类型}

```
In [25]: # 插槽
         "Hello {0} {1}, there are ${2} in your account".format("Mr.", "Smith", 1
         0000)
Out[25]: 'Hello Mr. Smith, there are $10000 in your account'

In [26]: "Hello {0} {1}, there are ${2} in your account" .format("Mr.", "Smith",
         10000)
Out[26]: 'Hello Mr. Smith, there are $10000 in your account'

In [27]: # 宽度
         'This int, {:10}, was placed in a field of witdh 10'.format(10)
Out[27]: 'This int,         10, was placed in a field of witdh 10'

In [28]: # 精度 类型
         'This float, {:10.5}, has width 10 and precision 5.'.format(3.1415926)
Out[28]: 'This float,     3.1416, has width 10 and precision 5.'

In [29]: 'This float, {:10.5f},  is fixed at 5 decimal places.'.format(3.1415926)
Out[29]: 'This float,    3.14159,  is fixed at 5 decimal places.'

In [30]: # 对齐
         "left justification: {0:<5}".format("Hi!")
Out[30]: 'left justification: Hi!  '

In [31]: "right justification: {0:>5}".format("Hi!")
Out[31]: 'right justification:   Hi!'

In [32]: "centered: {0:^5}".format("Hi!")
Out[32]: 'centered:  Hi! '
```

8 文件处理

8.1 文件的本质

你可以将文本文件看成一个（可能很长的）字符串，恰好存储在磁盘上。当然，典型的文件通常包含多于一行的文本。使用换行符（\n）来换行。

- Hello
- World
-

- Goodbye 32

如果存储到文件，你会得到以下序列

Hello\nWorld\n\nGoodbye 32\n

In [48]:
```
print('Hello\nWorld\n\nGoodbye 32\n')
```

```
Hello
World

Goodbye 32
```

8.2 文件处理

In [53]:
```
# printfile.py
#     Prints a file to the screen.

def main():
    fname = input("Enter filename: ")
    infile = open(fname,'r')
    data = infile.read()    # 将文件的全部内容作为单个（可能很大的）字符串返回
    print(data)
main()
```

```
Enter filename: names.txt
John Zelle
Zaphod Beeblebrox
Guido VanRossum
```

In [55]:
```
# readline() 读取一行
# 以下程序读取文件前 5 行
infile = open('names.txt', "r")
for i in range(5):
    line = infile.readline()
    print(line[:-1])
```

```
John Zelle
Zaphod Beeblebrox
Guido VanRossum
```

In [33]:
```
# 使用以下方式读取文件每一行
infile = open('names.txt', "r")
for line in infile:
    # process the line here
    pass
infile.close()
```

8.3 示例程序：批处理用户名

In [58]:
```
# userfile.py
#    Program to create a file of usernames in batch mode.
```

```
def main():
    print("This program creates a file of usernames from a")
    print("file of names.")

    # get the file names
    infileName = input("What file are the names in? ")
    outfileName = input("What file should the usernames go in? ")

    # open the files
    infile = open(infileName, "r")
    outfile = open(outfileName, "w")

    # process each line of the input file
    for line in infile:
        # get the first and last names from line
        first, last = line.split()
        # create the username
        uname = (first[0]+last[:7]).lower()
        # write it to the output file
        print(uname, file=outfile)

    # close both files
    infile.close()
    outfile.close()

    print("Usernames have been written to", outfileName)

main()
```

```
This program creates a file of usernames from a
file of names.
What file are the names in? names.txt
What file should the usernames go in? out.txt
Usernames have been written to out.txt
```

8.4 文件对话框

In [59]:
```
from tkinter.filedialog import askopenfilename, asksaveasfilename
infilename = askopenfilename()
print(infilename)
```

```
E:/Tianhua/Python/names.txt
```

> 练习
>
> - 某个教授给出了 5 分测验，等级为 5-A、 4-B、 3-C、 2-D、 1-E、 0-F。编写一个程序，接受测验分数作为输入，并打印出相应的等级。

> 练习
>
> - 编写一个程序，接收用户输入的一个句子，输出:
>   - 该句子中单词首字母的大写（比如，输入 Initial public offering，输出 IPO）

> - 该句子中的单词个数
> - 该句子中单词的平均长度

> 练习
>
> - 统计《红楼梦》中分别提到了多少次“贾宝玉”和“林黛玉”。(使用 `for` 语句逐行读取 '红楼梦.txt' 文件)

学习目标:

- 掌握函数的概念和使用;
- 掌握实参和形参;
- 掌握传值和传引用的区别。

学习重点:

- 函数的定义和使用

1 函数的功能

- 避免重复
- 使程序易于理解
- 使程序易于维护

2 示例程序

- 你可以将函数想象成一个“子程序”：程序里面的一个小程序
- 函数的基本思想是写一个语句序列，并给这个序列取一个名字
- 然后可以通过引用函数名称，在程序中的任何位置执行这些指令。

In [1]:
```
def main():
    print("Happy birthday to you!" )
    print("Happy birthday to you!" )
    print("Happy birthday, dear Fred...")
    print("Happy birthday to you!")

main()
```

```
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Fred...
Happy birthday to you!
```

你可以为重复的部分定义函数

In [2]:
```
# 定义 happy 函数
def happy():
    print("Happy birthday to you!")
```

In [3]:
```
happy()
```

```
Happy birthday to you!
```

In [4]:
```
# 定义 singFred
def singFred():
    happy()
    happy()
    print("Happy birthday, dear Fred...")
    happy()
```

In [5]:
```
singFred()
```

```
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Fred...
Happy birthday to you!
```

In [6]:
```
# 假设今天也是 Lucy 生日，我们再定义 singLucy
def singLucy():
    happy()
    happy()
    print("Happy birthday, dear Lucy...")
    happy()
```

In [7]:
```
def main():
    singFred()
    print()
    singLucy()
main()
```

```
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Fred...
Happy birthday to you!

Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Lucy...
Happy birthday to you!
```

In [8]:
```
# 我们已经通过 happy 函数消除了一些重复，能否做得更好？
def sing(person):
    happy()
    happy()
    print("Happy birthday, dear", person + ".")
    happy()
```

In [9]:
```
def main():
    sing('Fred')
    print()
    sing('Lucy')
    print()
    sing('Elmer')
main()
```

```
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Fred.
Happy birthday to you!

Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Lucy.
Happy birthday to you!
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Elmer.
Happy birthday to you!
```

3 函数和参数

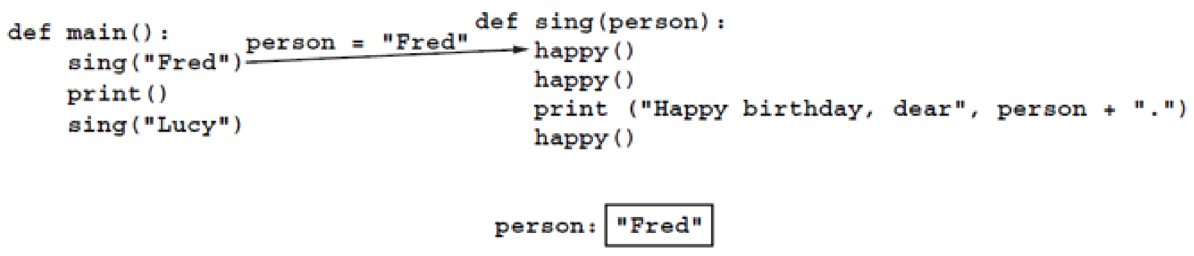
In [7]:
```
#函数的定义如下:
#
#def <name>(<formal-parameters>):
#   <body>
#
#函数的调用
#
#<name>(<actual-parameters>)
```

形参和实参

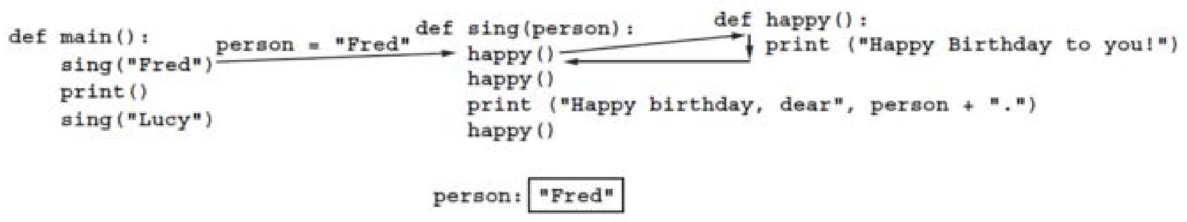
- 形参与函数中使用的所有变量一样，只能在函数体中访问
- 函数的形参获得实参提供的值
- 可以将形参理解为“占位”的人

Python 调用一个函数时，启动一个四步过程:

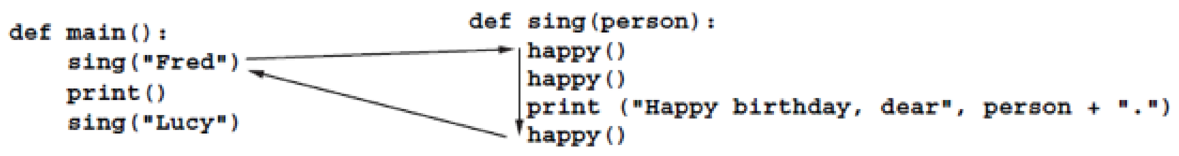
- 调用程序在调用点暂停运行（进入函数）
- 函数的形参获得调用中实参提供的值
- 执行函数体
- 控制返回到函数被调用之后的点



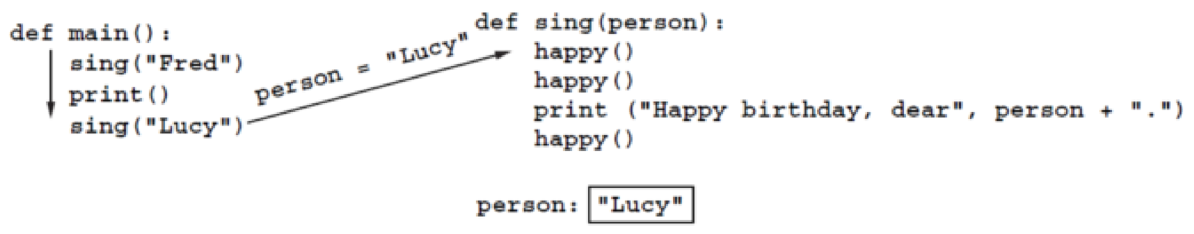
```
def main():                          def sing(person):
    sing("Fred")   person = "Fred"       happy()
    print()                              happy()
    sing("Lucy")                         print ("Happy birthday, dear", person + ".")
                                         happy()

                     person: "Fred"
```



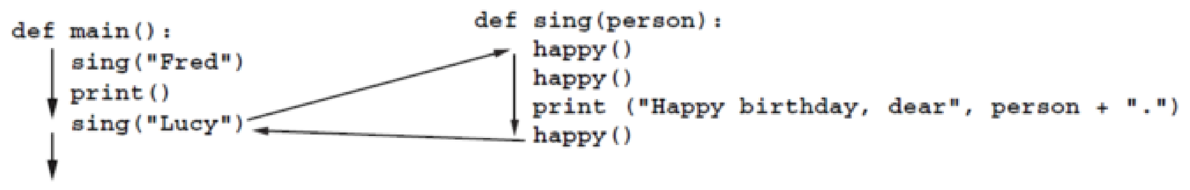
```
def main():                     def sing(person):        def happy():
    sing("Fred")  person = "Fred"   happy()                  print ("Happy Birthday to you!")
    print()                         happy()
    sing("Lucy")                    print ("Happy birthday, dear", person + ".")
                                    happy()

                  person: "Fred"
```



```
def main():                 def sing(person):
    sing("Fred")                happy()
    print()                     happy()
    sing("Lucy")                print ("Happy birthday, dear", person + ".")
                                happy()
```



```
def main():
    sing("Fred")
    print()
    sing("Lucy")

def sing(person):
    happy()
    happy()
    print ("Happy birthday, dear", person + ".")
    happy()
```



```
def main():
    sing("Fred")
    print()
    sing("Lucy")

def sing(person):
    happy()
    happy()
    print ("Happy birthday, dear", person + ".")
    happy()
```

当函数定义多个参数时，实参按照位置与形参匹配

如果要改变顺序**...**

In [28]:
```
def print_num(x,y):
    print('x=',x)
    print('y=',y)
```

In [29]:
```
print_num(y=10,x=20)
```

```
x= 20
y= 10
```

4 有返回值的函数

In [13]:
```
def square(x):
    return x ** 2
```

In [14]:
```
square(-3)
```

Out[14]:
```
9
```

In [15]:
```
# happy2.py
#    Happy Birthday using value returning functions

def happy():
    return "Happy birthday to you!\n"

def verseFor(person):
    lyrics = happy()*2 + "Happy birthday, dear " + person + ".\n" + happy()
    return lyrics

def main():
    for person in ["Fred", "Lucy", "Elmer"]:
        print(verseFor(person))

main()
```

```
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Fred.
Happy birthday to you!

Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Lucy.
Happy birthday to you!

Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Elmer.
Happy birthday to you!
```

所有的打印都在 main 中进行，happy 和 verseFor 只负责创建和返回适当的字符串

In [17]:
```
#可以方便地输出到文件
def main():
    outf = open('Happy_Birthday.txt','w')
    for person in ['Fred','Lucy','Elmer']:
        print(verseFor(person), file=outf)
    outf.close()
main()
```

返回多个值的函数

In [19]:
```
#返回多个值
def sumDiff(x, y):
    summ = x + y
    diff = x - y
    return summ, diff
```

In [20]:
```
sumDiff(10,3)
```

Out[20]:
```
(13, 7)
```

In [21]:
```
x,y = sumDiff(10,3)
```

In [22]:
```
x = sumDiff(10,3)
```

In [23]:
```
x
```

Out[23]:
```
(13, 7)
```

5 修改参数的函数

假设你正在编写一个管理银行账户的程序，必须执行的常见任务是在账户上累积利息。我们可以考虑编写一个函数，自动将利息添加到账户余额。

In [24]:
```
def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    balance = newBalance
```

In [25]:
```
# 测试 addInterest 函数
amount = 100
rate = 0.05
addInterest(amount, rate)
print(amount)
```
```
100
```

重要概念：

- 函数的形参只接收实参的“值”
- 函数不能访问保存实参的变量

In [8]:
```
# 使用返回值
def addInterest(balance, rate):
    newBalance = balance * (1 + rate)
    return newBalance
```

In [27]:
```
amount = 1000
rate = 0.05
amount = addInterest(amount, rate)
print(amount)
```
```
1050.0
```

再考虑以下程序……

In [29]:
```
# 处理很多账户
# addinterest3.py

def addInterest(balances, rate):
    for i in range(len(balances)):
        balances[i] = balances[i] * (1+rate)

def test():
    amounts = [1000, 2200, 800, 360]
    rate = 0.05
    addInterest(amounts, rate)
    print(amounts)

test()
```
```
[1050.0, 2310.0, 840.0, 378.0]
```

发生了什么？

列表没变，但列表中的内容变了

传值与传引用

- 传值：int float str... 等基础数值
- 传引用：list dict set array dataframe... 等集合数据

6 默认参数

调用函数时，如果没有传递参数，则会使用默认参数。

编写一个打印菱形的程序，使用默认参数

In [24]:
```python
def diamond(n=5):
    for i in range(n):
        print('{:^100}'.format('*'*(2*i+1)))
    for i in range(n-2,-1,-1):
        print('{:^100}'.format('*'*(2*i+1)))
diamond(5)
```

```
                                                  *                                                 

                                                 ***                                                

                                                *****                                               

                                               *******                                              

                                              *********                                             

                                               *******                                              

                                                *****                                               

                                                 ***                                                

                                                  *                                                 
```

注意:默认参数后不可有非默认参数

In [27]:
```python
# 默认参数前后不可有非默认参数
def add_two(x,y=0):
    return x + y
```

7 不定长参数

In [34]:
```python
def printit(arg1,*args):
    print('args:', args)
    print('type of args', type(args))
    print('Length of args:', len(args))
```

In [35]:
```python
printit(70, 60, 50)
```

```
args: (60, 50)
type of args <class 'tuple'>
Length of args: 2
```

In [36]:
```python
printit(70)
```

```
args: ()
type of args <class 'tuple'>
Length of args: 0
```

强制指定参数名称

In [37]:
```
def printit(arg1,**args):
    print('args:', args)
    print('type of args', type(args))
    print('Length of args:', len(args))
```

In [38]:
```
printit(70, x=60, y=50)
```

```
args: {'x': 60, 'y': 50}
type of args <class 'dict'>
Length of args: 2
```

In [39]:
```
printit(70)
```

```
args: {}
type of args <class 'dict'>
Length of args: 0
```

> 练习
>
> - 编写 sumN(n) 函数：返回前 n 个数的总和
> - 编写 sumNCubes(n) 函数：返回前 n 个数的立方的总和
> - 编写一个程序，提示用户输入 n， 然后显示以上两个函数的计算结果

> 练习
>
> - 以下函数允许计算两个数的乘积，请稍加改造，变成可接收一个或多个数并计算乘积：
>
>   ```
>   def product(x, y):
>     return x * y
>   ```

> 练习
>
> 求 S=a+aa+aaa+aaaa+aa...a （共 n 项）的值，其中 a 是一个数字。例如 2+22+222+2222+22222(此时共有5个数相加)。编写一个程序，让用户输入 a 和 n 的值，然后计算 S 的值。
>
> - 编写一个函数，以 a 和 n 为参数，输出 aa...a （n个a）
> - 编写一个函数，调用以上函数，计算 S 的值

学习目标:

- 掌握条件语句的编写;
- 掌握异常处理

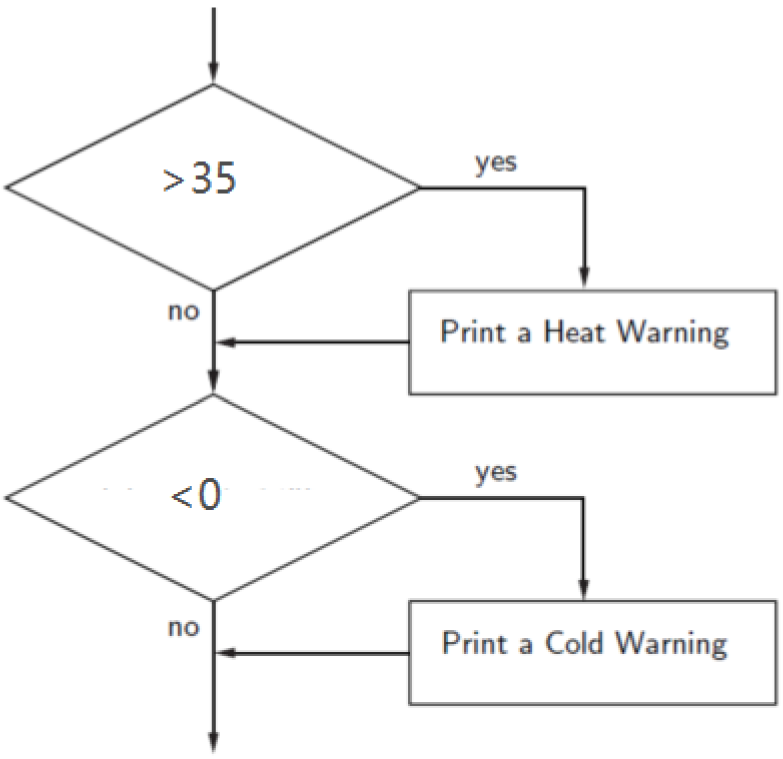
学习重点:

- 条件语句的编写

1. 示例：温度警告

In [1]:
```python
#如果温度高于35度，则发出高温警告，如果低于0度，则低温警告
temperature = float(input('Please input temperature: '))
if temperature > 35:
    print('Warning: Hot')
if temperature < 0:
    print('Warning: Cold')
```

```
Please input temperature: 90
Warning: Hot
```

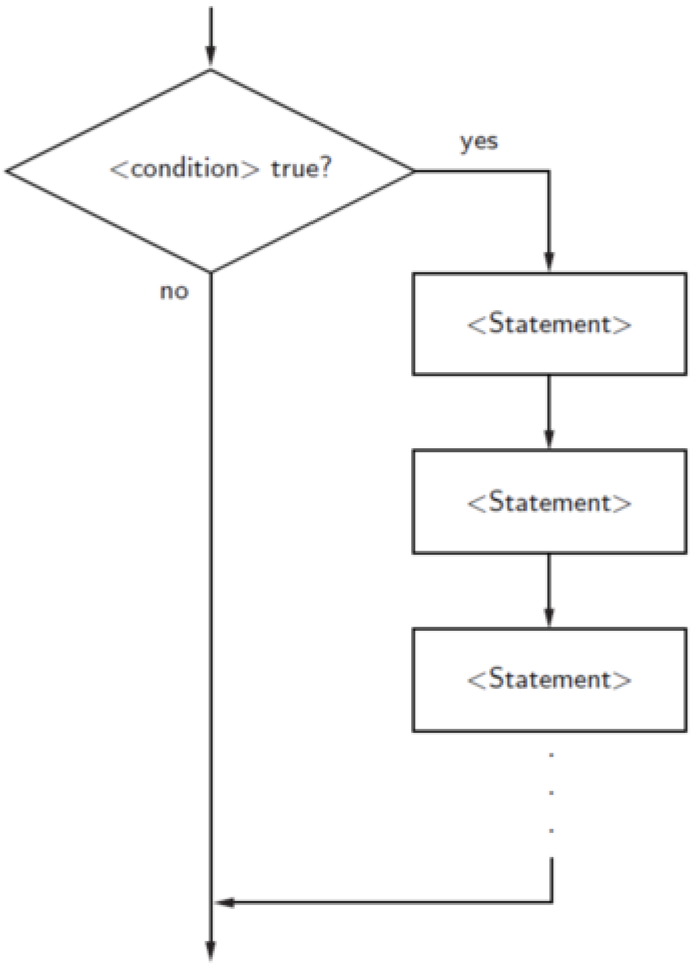


条件判断语句

- 条件表达式成为布尔表达式，产生布尔值 （True/False）
- 判断是否相等用 ==
- 比较字符串时，用“字典序”。所有大写字母位于小写字母前面。例如 "Bbbb" 在 "aaaa" 之前

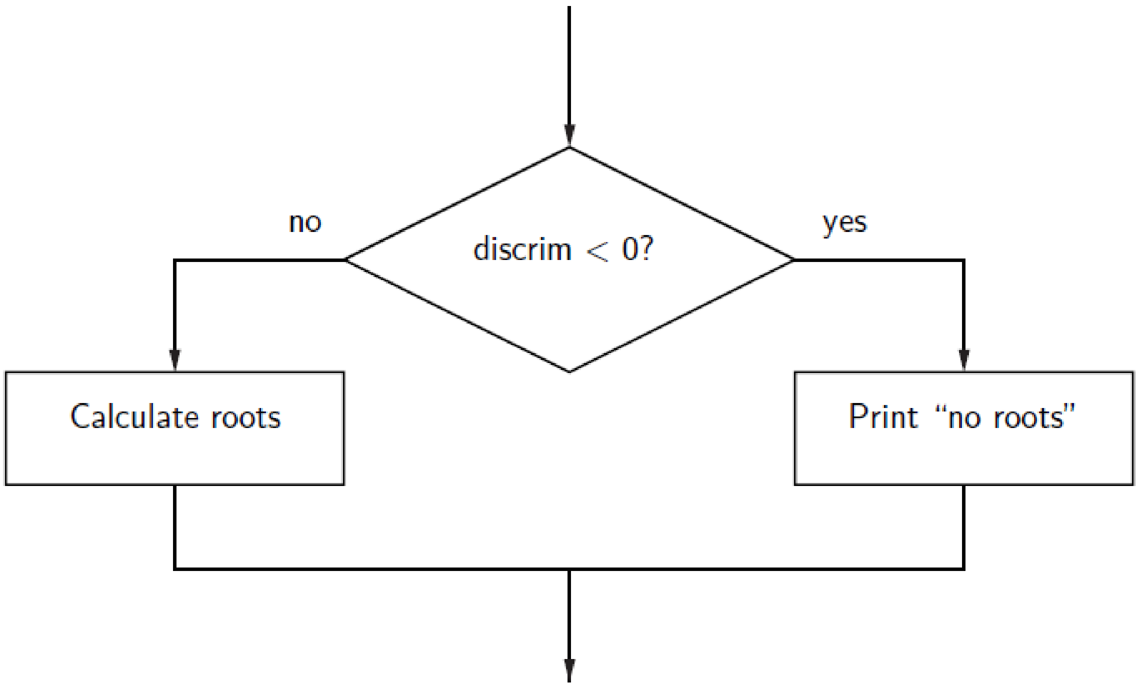
Python	Mathematics	Meaning
<	<	Less than
<=	≤	Less than or equal to
==	=	Equal to
>=	≥	Greater than or equal to
>	>	Greater than
!=	≠	Not equal to

2 单路判断



3 两路判断

修改求二次方程根的程序，使之能够对没有根的情况做出说明



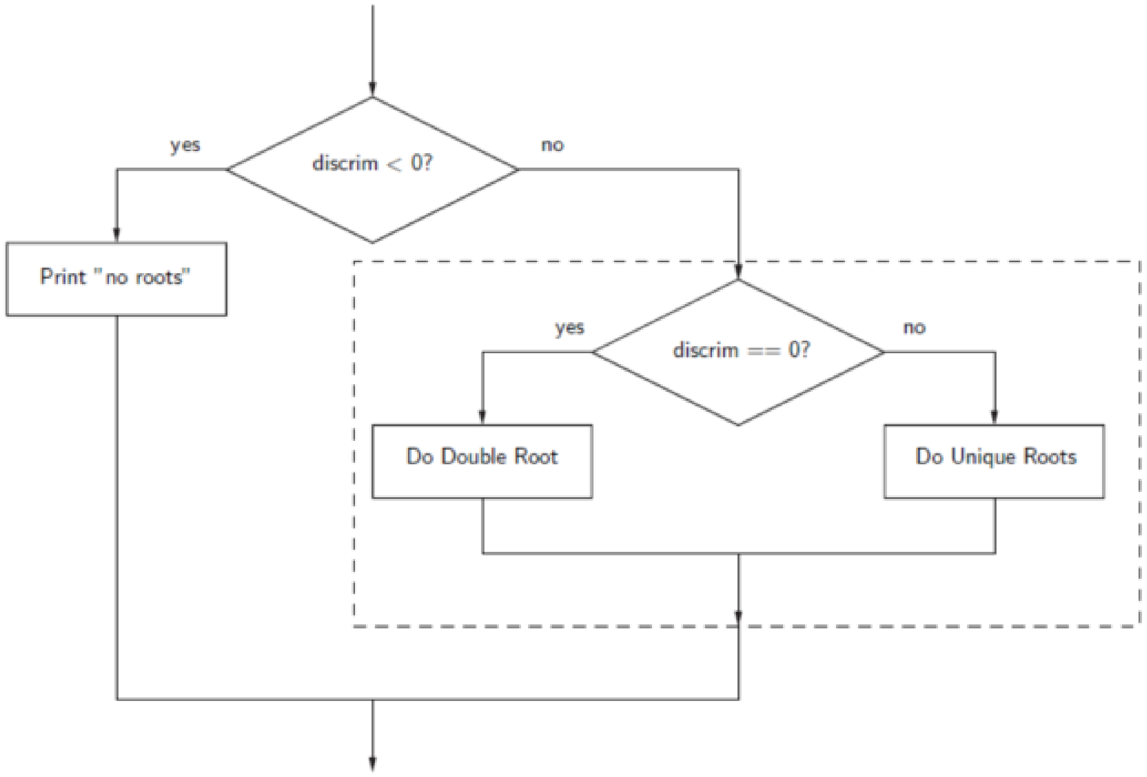
In [6]:

```python
# quadratic3.py
import math

def main():
    print("This program finds the real solutions to a quadratic\n")
    a = float(input("Enter coefficient a: "))
    b = float(input("Enter coefficient b: "))
    c = float(input("Enter coefficient c: "))

    discrim = b * b - 4 * a * c
    if discrim < 0:
        print("\nThe equation has no real roots!")
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2)
main()
```

```
This program finds the real solutions to a quadratic

Enter coefficient a: 1
Enter coefficient b: 2
Enter coefficient c: 3

The equation has no real roots!
```

3 多路判断

修改求根程序，分三种情况：有两个不同根、有两个相同的根、没有实数根



In [7]:
```
# quadratic4.py
import math

def main():
    print("This program finds the real solutions to a quadratic\n")
    a = float(input("Enter coefficient a: "))
    b = float(input("Enter coefficient b: "))
    c = float(input("Enter coefficient c: "))

    discrim = b * b - 4 * a * c
    if discrim < 0:
        print("\nThe equation has no real roots!")
    elif discrim == 0:
        root = -b / (2 * a)
        print("\nThere is a double root at", root)
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2 )
main()
```

```
This program finds the real solutions to a quadratic

Enter coefficient a: 1
Enter coefficient b: -2
Enter coefficient c: 1

There is a double root at 1.0
```

4 异常处理

使用专门的异常处理方法来应对程序中可能出现的错误。本质上它是说："做这些步骤，如果出现问题，以这种方式处理它"

In [8]:
```
# quadratic5.py
import math

def main():
    print ("This program finds the real solutions to a quadratic\n")

    try:
        a = float(input("Enter coefficient a: "))
        b = float(input("Enter coefficient b: "))
        c = float(input("Enter coefficient c: "))
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2)
    except ValueError:
        print("\nNo real roots")
main()
```

```
This program finds the real solutions to a quadratic

Enter coefficient a: 1
Enter coefficient b: 2
Enter coefficient c: 3

No real roots
```

In [9]:
```
#判断异常的类型
import math

def main():
    print("This program finds the real solutions to a quadratic\n")

    try:
        a = float(input("Enter coefficient a: "))
        b = float(input("Enter coefficient b: "))
        c = float(input("Enter coefficient c: "))
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2 )
    except ValueError as excObj:
        if str(excObj) == "math domain error":
            print("No Real Roots")
        else:
            print("Invalid coefficient given")
    except:
        print("\nSomething went wrong, sorry!")

main()
```

```
This program finds the real solutions to a quadratic

Enter coefficient a: x
Invalid coefficient given
```

在刚开始编程时，你可能不太担心错误的输入。但专业品质的软件应该采取所有可行的方法，防止用户得到意外的结果。

5 if...else...的精简表达

In [4]:
```
x = float(input('Enter a number: '))
if x > 0:
    y = 1
else:
    y= -1
print('y = ', y)
```

```
Enter a number: -9
y =  -1
```

In [6]:
```
x = float(input('Enter a number: '))
y = 1 if x > 0 else -1
print('y = ', y)
```

```
Enter a number: -9
y =  -1
```

> 练习
>
> - 许多公司对每周超出 40 小时以上的工作时间支付 150% 的工资。编写程序，输入工作小时数和小时工资，计算一周的总工资。

> 练习
>
> - 某个教授给出了 5 分测验，等级为 5-A、 4-B、 3-C、 2-D、 1-E、 0-F。编写一个程序，接受测验分数作为输入，并使用判断结构计算相应的等级。

> 练习
>
> 编写一个程序，用户输入股票价格以及净利润，输出市盈率的值。 思考：
>
> 1. 该程序有可能出现什么异常?
> 2. 如何捕获该异常。

学习目标：

- 掌握循环语句的编写
- 掌握循环的退出方式
- 理解布尔值的计算

学习重点：

- 循环语句的编写

1 确定循环

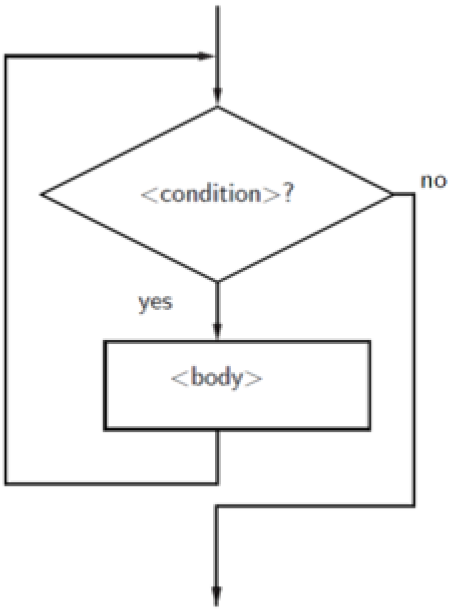
编写一个程序，计算用户输入的一系列数字的平均值

In [1]:
```
# average1.py

def main():
    n = int(input("How many numbers do you have? "))
    total = 0.0
    for i in range(n):
        x = float(input("Enter a number >> "))
        total = total + x
    print("\nThe average of the numbers is", total / n)

main()
```

```
How many numbers do you have? 3
Enter a number >> 11
Enter a number >> 22
Enter a number >> 33

The average of the numbers is 22.0
```

2 不定循环

In [3]:
```
# while <condition>:
#   <body>
```



In [4]:
```python
# 打印从 0 到 10 的数字
i = 0
while i <= 10:
    print(i)
    i = i + 1
```

```
0
1
2
3
4
5
6
7
8
9
10
```

In [5]:
```python
# 等价于以下 for 循环
for i in range(11):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
```

3 常见循环模式 （不定循环）

3.1 交互式循环

交互式循环背后的想法是，允许用户根据需要重复程序的某些部分。

In [6]:
```
# average2.py

def main():
    total = 0.0
    count = 0
    moredata = "yes"
    while moredata[0] == "y":
        x = float(input("Enter a number >> "))
        total = total + x
        count = count + 1
        moredata = input("Do you have more numbers (yes or no)? ")
    print("\nThe average of the numbers is", total / count)

main()
```

```
Enter a number >> 12
Do you have more numbers (yes or no)? y
Enter a number >> 12
Do you have more numbers (yes or no)? n

The average of the numbers is 12.0
```

3.2 哨兵循环

哨兵循环不断处理数据，直到达到一个特定的值，即“哨兵”

In [7]:
```
# 使用负数作为哨兵

def main():
    total = 0.0
    count = 0
    x = float(input("Enter a number (negative to quit) >> "))
    while x >= 0:
        total = total + x
        count = count + 1
        x = float(input("Enter a number (negative to quit) >> "))
    print("\nThe average of the numbers is", total / count)

main()
```

```
Enter a number (negative to quit) >> 123
Enter a number (negative to quit) >> 34
Enter a number (negative to quit) >> -1

The average of the numbers is 78.5
```

思考：使用负数作为哨兵好吗 **?**

- 让我们来设置一个 **“**真正的哨兵**”**

In [3]:
```
def main():
    total = 0.0
    count = 0
    xStr = input("Enter a number (<Enter> to quit) >> ")
    while xStr != "":
        x = float(xStr)
        total = total + x
        count = count + 1
        xStr = input("Enter a number (<Enter> to quit) >> ")
    print("\nThe average of the numbers is", total / count)

main()
```

```
Enter a number (<Enter> to quit) >> 123
Enter a number (<Enter> to quit) >> 20
Enter a number (<Enter> to quit) >> 2
Enter a number (<Enter> to quit) >>

The average of the numbers is 48.333333333333336
```

思考：以上程序还有什么问题？如何改进？

3.3 文件循环

In [1]:

```
infile = open('average.txt','r')
total = 0.0
count = 0
for line in infile:
    total = total + float(line)
    count = count + 1
print("\nThe average of the numbers is", total / count)
```

```
The average of the numbers is 187.14285714285714
```

3.4 嵌套循环

In [2]:

```
#允许数字用“，”分割
infile = open('average2.txt','r')
total = 0.0
count = 0
for line in infile:
    for number in line.split(','):
        total = total + float(number)
        count = count + 1
print("\nThe average of the numbers is", total / count)
```

```
The average of the numbers is 135.66666666666666
```

In [17]:

```
#打印9*9乘法表
for i in range(1,10):
    print('\n')
    for j in range(1,i+1):
        print('{} * {} = {:<2}'.format(i,j,i*j),end='    ')
```

```

1 * 1 = 1

2 * 1 = 2     2 * 2 = 4

3 * 1 = 3     3 * 2 = 6     3 * 3 = 9
```

```
4 * 1 = 4    4 * 2 = 8    4 * 3 = 12   4 * 4 = 16

5 * 1 = 5    5 * 2 = 10   5 * 3 = 15   5 * 4 = 20   5 * 5 = 25

6 * 1 = 6    6 * 2 = 12   6 * 3 = 18   6 * 4 = 24   6 * 5 = 30   6 * 6 =
 36

7 * 1 = 7    7 * 2 = 14   7 * 3 = 21   7 * 4 = 28   7 * 5 = 35   7 * 6 =
 42   7 * 7 = 49

8 * 1 = 8    8 * 2 = 16   8 * 3 = 24   8 * 4 = 32   8 * 5 = 40   8 * 6 =
 48   8 * 7 = 56   8 * 8 = 64

9 * 1 = 9    9 * 2 = 18   9 * 3 = 27   9 * 4 = 36   9 * 5 = 45   9 * 6 =
 54   9 * 7 = 63   9 * 8 = 72   9 * 9 = 81
```

4 布尔值计算

4.1 布尔运算符

- and
- or
- not

优先级从高到低：**not, and, or**

- a or not b and c 等于 (a or ((not b) and c))

不清楚时，使用小括号总是没错的

4.2 布尔代数

DeMorgan 第一定律

not(a or b) = (not a) and (not b)

5 Break / Continue

- break语句用来终止循环语句，即循环条件没有False条件或者序列还没被完全递归完，也会停止执行循环语句。
- break语句用在while和for循环中。
- 如果你使用嵌套循环，break语句将停止执行最深层的循环，并开始执行下一行代码。

In [20]:
```
for s in 'Hello World':
    if s == ' ':
        break
    print(s, end='')
```

```
Hello
```

continue 语句用来告诉Python跳过当前循环的剩余语句，然后继续进行下一轮循环。

- continue 语句跳过本次循环，而break跳出整个循环。

In [21]:
```python
for s in 'Hello World':
    if s == ' ':
        continue
    print(s, end='')
```

```
HelloWorld
```

> 练习
>
> - 斐波那契数列开始是 1，1，2，3，5，8，......序列中每个数字是前两个数字之和。编写一个程序，计算并输出低 n 个斐波那契数，其中 n 为用户输入值

> 练习
>
> - 国家气象局使用以下公式计算风寒指数： $$35.74 + 0.6215T - 35.75(V^{0.16}) + 0.4275T(V^{0.16})$$ 其中，T 是以华氏度为单位的温度，V 是以小时为单位的风速。编程打印以长格式漂亮的风寒指数表格。行代表风速为 0~50，以 5英里/小时为增量，列表示温度从 -20~60，以 10 度为增量。注意：该公式仅适用于每小时超过 3 英里的风速。

> 练习
>
> - 使用 while 循环，来确定投资在特定利率下翻倍需要多长时间。输入年利率，输出投资增加一倍的年数。

> 练习
>
> - Syraucuse 序列的生成从一个自然数来时，重复应用以下函数，直至达到 1：
>
> $$syr(x)=\begin{cases} x/2, &x 为偶数 \\ 3x+1, &x 为奇数 \end{cases}$$
>
>   数学中一个悬而未决的问题是：对于每个可能的起始值，该序列是否总会达到1.
>   编写一个程序，从用户获取起始值，然后打印该起始值对应的 Syraucuse 序列。

学习要求:

- 掌握随机数的生成方法;
- 掌握蒙特卡洛模拟方法;

学习重点:

- 蒙特卡洛模拟

1 计算积分

使用模拟方法计算积分 $$\int_0^1 x^2 dx = \lim_{n \to \infty} \frac{1}{n}\sum_{i=1}^{n}(\frac{i}{n})^2$$

```
In [128]: def summ(n = 10000):
              total = 0
              for i in range(1, n+1):
                  total += (i/n)**2
              return total / n
```

```
In [129]: summ()
```

```
Out[129]: 0.33338333349999983
```

2 随机数

随机生成一个实数，它在[0,1)范围内（均匀分布）

```
In [8]: from random import random
        x = random()
        print(x)
```

```
0.774470318097844
```

思考：如何验证 **random** 函数生成的数字确实是均匀分布的？

3 求 π 的值

```
In [14]: M = 10000
         count = 0
         for i in range(M):
             x = 2 * random() - 1
             y = 2 * random() - 1
             d = (x**2+y**2)**0.5
             if d < 1:
                 count += 1
         est_pi = 4 * count / M
```

```
print(est_pi)
```

```
3.1612
```

4 壁球游戏

Denny 与 Mike 打壁球，Denny 似乎总是被击败，输掉了绝大多数比赛，但在他看来， Mike 的水平仅比他高一点。从表面来看，人们会认为那些“稍好”一点的球员应该“稍微”多赢一点，但对 Denny 而言，Mike 似乎赢的太多了。

我们来考虑这样一种可能性：对于壁球比赛来说，能力上的小差异会导致结果的大差异（类似蝴蝶效应）。

比赛规则:

- 一名选手先发球，然后选手交替击球
- 如果发球选手输，则发球权交给另一名选手；如果发球选手赢，则得 1 分
- 选手只能在自己发球回合得分
- 第一个得到 15 分的选手赢得比赛

（有限状态机）对于每一场比赛，有两个状态 **A** 和 **B**，根据每一局的输赢，状态发生变化

In [91]:
```
from random import random

def sim_one_game(pA, pB, serving = 'A'):
    scoreA, scoreB = 0, 0
    while(True):
        if serving == 'A':
            if random() < pA:
                scoreA += 1
            else:
                serving = 'B'
        else:
            if random() < pB:
                scoreB += 1
            else:
                serving = 'A'
        if scoreA == 15 or scoreB == 15:
            return scoreA, scoreB
```

In [92]:
```
sim_one_game(0.65,0.6,'A')
```

Out[92]:
```
(15, 5)
```

In [97]:
```
def sim_games(pA, pB, serving = 'A', M = 50000):
    winsA, winsB = 0, 0
    for i in range(M):
        scoreA, scoreB = sim_one_game(pA, pB, serving = serving)
        if scoreA > scoreB:
            winsA += 1
        else:
            winsB += 1
    return winsA / M
```

In [127]:
```
sim_games(0.65, 0.6, 'B')
```

Out[127]:
```
0.60044
```

练习

- 编写一个模拟程序，估计一把掷五个六面骰子，点数都相同的概率

1 数组的概述

- 什么是数组:
    - 使用一个标量保存一系列数据
- 数据与列表的异同:
    - 相同点:
        - 都表示一系列数据
        - 都从 0 开始索引
    - 不同点:
        - 数据长度固定，列表长度不固定
        - 数组中元素具有相同类型，而列表可以包含不同类型的元素
        - 数据支持更多运算，并且运算速度更快
        - 列表是内置数据类型，而使用数据需要导入 numpy 库

In [2]:
```
#导入 numpy 库
import numpy as np
```

2 数组的创建

No.	函数	含义
1	np.array(object)	创建数组
2	np.zeros(shape)	根据 shape 指定的结构创建全零数组
3	np.ones(shape)	根据 shape 指定的结构创建全一数组
4	np.empty(shape)	根据 shape 指定的结构创建空数组
5	a.copy()	复制一个同 a 一样的数组
6	np.zeros_like(a)	创建与 a 同样结构的全零数组
7	np.ones_like(a)	创建与 a 同样结构的全零数组
8	np.empty_like(a)	创建与 a 同样结构的空数组
9	np.arange([start,] stop, [step])	创建等差数据
10	np.linspace(start, stop, num)	创建等差数据

2.1 通过列举创建数组

`np.array(object)`
`object` 可以为一个列表

In [3]:
```
# 创建一个数组，保存一个股票过去 5 个交易日的收盘价
a = np.array([5,5.1,5.4,5.2,4.9])
a
```

Out[3]:
```
array([5. , 5.1, 5.4, 5.2, 4.9])
```

常见错误：遗漏 `[]`

数组的属性

No.	函数	含义
1	len(a)	获得数组的长度
2	a.shape	数组的结构
3	a.size	数组中的元素个数
4	a.ndim	数组的维度数
5	a.dtype	数组中的数据类型

In [5]:
```
# 创建一个二维数组 price, 反应两只股票过去 5 个交易日的收盘价
price = np.array([[5,5.1,5.2,5.3,5.4],[2.2,2.1,2,1.9,1.8]])
```

In [6]:
```
price.ndim
```
Out[6]:
```
2
```

In [7]:
```
price.shape
```
Out[7]:
```
(2, 5)
```

In [8]:
```
price.size
```
Out[8]:
```
10
```

In [9]:
```
price.dtype
```
Out[9]:
```
dtype('float64')
```

In [10]:
```
len(price)
```
Out[10]:
```
2
```

In [4]:
```
# 查看 digits 数组的属性
from sklearn import datasets
digits = datasets.load_digits()
images = digits.images
```

回答以下问题：

- images 的是几维数组？
- images 中有多少张图片？
- 每张图片的大小是多少？

2.2 创建等差数组

`np.arange([start,], stop, [step])`

包含 `start`，但不包含 `stop`

`start` 默认为 0，`step` 默认为1

In [20]:
```
# 创建数组：0，1，2，3，4，5，6，7，8，9
a = np.arange(0, 10, 1)
a
```
Out[20]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [21]:
```
# 简写
a = np.arange(10)
a
```
Out[21]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

常见错误：以为所生成的数组包含 `stop`

> 练习： 创建以下数组
>
> (a) 1，3，5，7，9
> (b) 0.1, 0.2, 0.3, 0.4, 0.5
> (c) 9, 8, 7, 6, 5, 4, 3, 2, 1, 0

`np.linspace(start, stop, num)`

- 在 `start` 和 `stop` 之间创建长度为 `num` 的等差数组
- 所创建的数组包含 `stop`

In [28]:
```
# [0,1] 区间生成长度为 10 的等差数列
a = np.linspace(0,1,10)
a
```
Out[28]:
```
array([0.        , 0.11111111, 0.22222222, 0.33333333, 0.44444444,
       0.55555556, 0.66666667, 0.77777778, 0.88888889, 1.        ])
```

2.3 创建随机数组

In [15]:
```
#生成均匀分布随机数
np.random.uniform(0,1,5)
```
Out[15]:
```
array([0.45204227, 0.56321316, 0.58795597, 0.36783195, 0.18377031])
```

In [18]:
```
#生成随机整数
np.random.randint(0,10,5)
```
Out[18]:
```
array([0, 0, 2, 4, 1])
```

In [19]:
```
#生成正态分布随机数
np.random.normal(0,1,5)
```
Out[19]:
```
array([-0.34228196,  0.58611873,  0.76037952, -0.46219832, -1.28596482])
```

In [29]:
```
#随机采样
```

```
np.random.choice([11,22,33,44,55],2 ,replace=False)
```

Out[29]:
```
array([22, 44])
```

In [30]:
```
np.random.choice([11,22,33,44,55],2, replace=True)
```

Out[30]:
```
array([33, 22])
```

2.4 全零数组、全一数组、空数组

In [24]:
```
# 创建五个 0 的数组
a = np.array([0,0,0,0,0])
a
```

Out[24]:
```
array([0, 0, 0, 0, 0])
```

更方便的方法: **`np.zeros(shape)`**

- `shape` 可以是一个数字，表示创建一个长度为 shape 的一维数组
- `shape` 也可以是一个元组，表示所要创建的数组的结构

In [2]:
```
a = np.zeros(5)
a
```

Out[2]:
```
array([0., 0., 0., 0., 0.])
```

In [6]:
```
a = np.zeros((3,4))
a
```

Out[6]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

常见错误： 遗漏 ()

创建全一数组： **`np.ones(shape)`**

In [17]:
```
a = np.ones(5)
a
```

Out[17]:
```
array([1., 1., 1., 1., 1.])
```

> 练习:
> 创建一个含有五个 6 的数组

创建空数组： **`np.empty(shape)`**

In [18]:
```
a = np.empty(5)
a
```

Out[18]:
```
array([1., 1., 1., 1., 1.])
```

注意: emtpy 并非真正的 empty

2.5 根据一个现有数组创建新数组

In [15]:
```
a = np.array([4,5,6])
#--------------------
print("数组a: ", a)
b = a.copy()
print("数组b: ", b)
```
```
数组a:  [4 5 6]
数组b:  [4 5 6]
```

In [16]:
```
a = np.array([4,5,6])
#--------------------
print("数组a: ", a)
b = np.zeros_like(a)
print("数组b: ", b)
```
```
数组a:  [4 5 6]
数组b:  [0 0 0]
```

In [17]:
```
a = np.array([4,5,6])
#--------------------
print("数组a: ", a)
b = np.ones_like(a)
print("数组b: ", b)
```
```
数组a:  [4 5 6]
数组b:  [1 1 1]
```

In [19]:
```
a = np.array([4,5,6])
#--------------------
print("数组a: ", a)
b = np.empty_like(a)
print("数组b: ", b)
```
```
数组a:  [4 5 6]
数组b:  [4 5 6]
```

2.6 数组的类型

在创建数组时指定数据类型

In [31]:
```
a = np.ones(3)
a.dtype
```
Out[31]:
```
dtype('float64')
```

In [33]:
```
a = np.ones(3, dtype=int)
a.dtype
```
Out[33]:
```
dtype('int32')
```

数组类型的转换

In [21]:
```
a = np.ones((3,3))
print('a:\n', a)
print('\ndtype of a:\n', a.dtype)
```

```
a:
 [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

dtype of a:
 float64
```

In [22]:
```
a = np.ones((3,3))
#------------------
a = a.astype(int)
print('a:\n', a)
print('\ndtype of a:\n', a.dtype)
```

```
a:
 [[1 1 1]
 [1 1 1]
 [1 1 1]]

dtype of a:
 int32
```

In [23]:
```
a = np.ones((3,3))
#------------------
a = a.astype(bool)
print('a:\n', a)
print('\ndtype of a:\n', a.dtype)
```

```
a:
 [[ True  True  True]
 [ True  True  True]
 [ True  True  True]]

dtype of a:
 bool
```

常见错误：使用标量类型转换方式对数组进行类型转换

3 索引和切片

3.1 简单索引

数组从 `0` 开始索引

a[0], a[1], a[2], ...

In [8]:
```
a = np.array([11,22,33])
print('a:\t', a)
print('a[0]:\t', a[0])
print('a[2]:\t', a[2])
```

```
a:	 [11 22 33]
```

```
a[0]: 11
a[2]: 33
```

> 练习：获取数组最后一个元素（使用 len）

- 数组索引值可以为负，表示从最后开始索引
- a[-1], a[-2], a[-3], ...

使用索引进行赋值

In [10]:
```python
a = np.array([11,22,33,44,55])
#--------------------------------------------------------------------------
print('原数组 a:\t', a)
a[3] = 888
print('赋值后数组 a:\t', a)
```

```
原数组 a: [11 22 33 44 55]
赋值后数组 a: [ 11  22  33 888  55]
```

3.2 切片

切片是获取多个元素的索引

`a[start:stop:step]`

`stop` 不包含

`step` 默认为 1

In [11]:
```python
a = np.array([11,22,33,44,55])
#--------------------------------------------------------------------------
print('a:\t\t', a)
print('a[1:4]\t\t', a[1:4])
print('a[1:5:2]\t', a[1:5:2])
```

```
a:          [11 22 33 44 55]
a[1:4]          [22 33 44]
a[1:5:2]    [22 44]
```

> 练习：
> `a = np.array([11,22,33,44,55])`
> 以下语句会得到什么结果
>
> - a[:3]
> - a[2:]
> - a[::3]

3.3 二维数组的索引

- 先行后列
- a[i][j] = a[i,j]

> 练习：
> `a = np.arange(64).reshape(8,8)`
> 下列表达式的值分别是什么？
>
> - a[0,0], a[0,2], a[1,5], a[3][4], a[-1,-2]
> - a[1:2,4:5],a[:,:],a[:,2]

注意: `a[:,:]` 与 a 的区别

> 练习：
> `a = np.arange(64).reshape(8,8)`
>
> - 获取数组的第一行
> - 获取数组的第一列
> - 获取数组的最后一列
> - 获取数组的前两列
> - 获取数组的最后三行

> 练习：
> `a = np.arange(64).reshape(8,8)`
> 求 a[3,3] 周围 8 个元素的和

3.4 布尔索引

适用 bool 数组作为索引值

a[b], 其中 b 为与 a 具有相同 `shape` 的布尔序列

In [13]:
```
a = np.arange(12)
b = a > 4
print(a)
print(b)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
[False False False False False  True  True  True  True  True  True  True]
```

In [14]:
```
print(a[b])
```

```
[ 5  6  7  8  9 10 11]
```

In [15]:
```
# 布尔索引结合赋值
a[b] = 0
print(a)
```

```
[0 1 2 3 4 0 0 0 0 0 0 0]
```

In [16]:
```
a = np.arange(12)
# 将数组 a 中大于 4 的元素赋值为 0
a[a>0]=0
print(a)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0]
```

In [17]:
```
a = np.arange(12).reshape(3,4)
b1 = np.array([False,True,True])          # first dim selection
b2 = np.array([True,False,True,False])      # second dim selection
print(a)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [18]:
```
a[b1]
```

Out[18]:
```
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [19]:
```
a[:, b2]
```

Out[19]:
```
array([[ 0,  2],
       [ 4,  6],
       [ 8, 10]])
```

3.5 智能索引

使用整数数组进行索引

In [22]:
```
a = np.array([11,22,33,44,55,66,77,88])
i = np.array( [1,3,5] )            # an array of indices
a[i]                               # the elements of a at the positions i
```

Out[22]:
```
array([22, 44, 66])
```

In [24]:
```
# 可以重复索引
j = np.array([0,0,1,1])
a[j]
```

Out[24]:
```
array([11, 11, 22, 22])
```

4 数组的形态操作

4.1 数组重排

In [31]:
```
a = np.arange(16)
a
```

Out[31]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

In [32]:
```
a.reshape(4,4)
```

Out[32]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [33]:
```
a.reshape(2,-1)
```

Out[33]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15]])
```

4.2 数组转置

In [36]:
```
aa = a.reshape(2,-1)
```

In [37]:
```
aa.T
```

Out[37]:
```
array([[ 0,  8],
       [ 1,  9],
       [ 2, 10],
       [ 3, 11],
       [ 4, 12],
       [ 5, 13],
       [ 6, 14],
       [ 7, 15]])
```

4.3 数组压平

In [38]:
```
aa.ravel()
```

Out[38]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

4.4 数组合并

In [40]:
```
a = np.array([[1,2],[3,4]])
b = np.array([[111,222],[333,444]])
print(a)
print(b)
```

```
[[1 2]
 [3 4]]
[[111 222]
 [333 444]]
```

In [42]:
```
np.vstack((a,b))
```

Out[42]:
```
array([[  1,   2],
       [  3,   4],
       [111, 222],
       [333, 444]])
```

In [43]:
```
np.hstack((a,b))
```

Out[43]:
```
array([[  1,   2, 111, 222],
       [  3,   4, 333, 444]])
```

5. 数组的运算

5.1 数组和标量之间的运算

将标量与数组中元素一一进行计算

c + [a1,a2,a3] = [c+a1, c+a2, c+a3]

In [81]:
```
a = np.array([1,2,3])
# --------------------
print('a:\t', a)
print('a+2:\t', a+2)
```
```
a:	[1 2 3]
a+2:	[3 4 5]
```

In [82]:
```
a = np.array([1,2,3])
# --------------------
print('a:\t', a)
print('a**2:\t', a**2)
```
```
a:	[1 2 3]
a**2:	[1 4 9]
```

In [83]:
```
a = np.array([1,2,3])
# --------------------
print('a:\t', a)
print('1/a:\t', 1/a)
```
```
a:	[1 2 3]
1/a:	[1.         0.5        0.33333333]
```

5.2 数组和数组之间的运算

将数组对应元素进行一一运算

数组必须具有相同的 shape

[a1, a2, a3] + [b1, b2, b3] = [a1+b1, a2+b2, a3+b3]

In [3]:
```
a = np.array([1,2,3])
b = np.array([4,5,6])
#----------------------
print('a:\t', a)
print('b:\t', b)
print('a+b:\t', a+b)
```
```
a:	[1 2 3]
b:	[4 5 6]
a+b:	[5 7 9]
```

In [4]:
```
a = np.array([1,2,3])
```

```
b = np.array([4,5,6])
#------------------------
print('a:\t', a)
print('b:\t', b)
print('a*b:\t', a*b)
```

```
a:	[1 2 3]
b:	[4 5 6]
a*b:	[ 4 10 18]
```

In [5]:

```
a = np.array([1,2,3])
b = np.array([4,5,6])
#------------------------
print('a:\t', a)
print('b:\t', b)
print('a**b:\t', a**b)
```

```
a:	[1 2 3]
b:	[4 5 6]
a**b:	[  1  32 729]
```

常见错误：两个数组的 shape 不一致

> 练习：
> 一个股票期初价格为 100， 未来 100 天每天张 1%，生成股票价格的数组

5.3 通用函数

对数组中每一个元素进行运算

f([a1, a2, a3]) = [f(a1), f(a2), f(a3)]

常用通用函数列表（一元）

No.	函数	含义
1	abs(a)	绝对值
2	sqrt(a)	开平方
3	square(a)	平方值
4	exp(a)	e 的指数
5	log(a)	自然对数
6	sin(a)、cos(a)	三角函数
7	sign(a)	正负号，1（正数）、0（零）、-1（负数）
8	isnan(a)	返回一个表示 a 中元素是否为 NaN 的布尔数组

常用通用函数列表（二元）

No.	函数	含义

1	power(a,b)	a 的 b次幂指数，power(a,2) = square(a)
2	maximum、fmax	相应位置上元素的最大值。fmax 将忽略 NaN
3	minimum、fmin	相应位置上元素的最小值。fmax 将忽略 NaN

所有通用函数
https://docs.scipy.org/doc/numpy/reference/ufuncs.html

In [44]:
```
#计算对数价格
price = np.power(1.01,np.arange(5)) * 100
#--------------------------------------------
logprice = np.log(price)
print('price:\t\t', price)
print('logprice:\t', logprice)
```

```
price:          [100.      101.      102.01     103.0301   104.060401]
logprice:   [4.60517019 4.61512052 4.62507085 4.63502118 4.64497151]
```

5.4 统计函数

No.	函数	含义
1	sum	求和
2	mean	平均值
3	std	标准差
4	var	方差
5	min, max	最小、最大值

In [18]:
```
price = np.array([5,5.2,5.1,4.9])
#--------------------------------
print('price:\t', price)
print('最高价:\t', price.max())
print('最低价:\t', price.min())
print('均价:\t', price.mean())
print('标准差:\t', price.std())
```

```
price:      [5.  5.2 5.1 4.9]
最高价:   5.2
最低价:   4.9
均价:      5.05
标准差:   0.11180339887498938
```

注意： maximum 与 max 的区别

6 数组的遍历

6.1 索引遍历

In [55]:
```
a = np.arange(10) * 11
for i in range(len(a)):
    print(i,a[i])
```

```
0 0
1 11
2 22
3 33
4 44
5 55
6 66
7 77
8 88
9 99
```

6.2 元素遍历

In [54]:
```
for x in a:
    print(x)
```

```
0
11
22
33
44
55
66
77
88
99
```

6.3 使用枚举器

In [53]:
```
for i,x in enumerate(a):
    print(i,x)
```

```
0 0
1 11
2 22
3 33
4 44
5 55
6 66
7 77
8 88
9 99
```

7 线性代数

In [56]:
```
a = np.array([[1.0, 2.0], [3.0, 4.0]])
print(a)
```

```
[[1. 2.]
 [3. 4.]]
```

In [59]:
```
# 求逆
a_inv = np.linalg.inv(a)
print(a_inv)
```

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

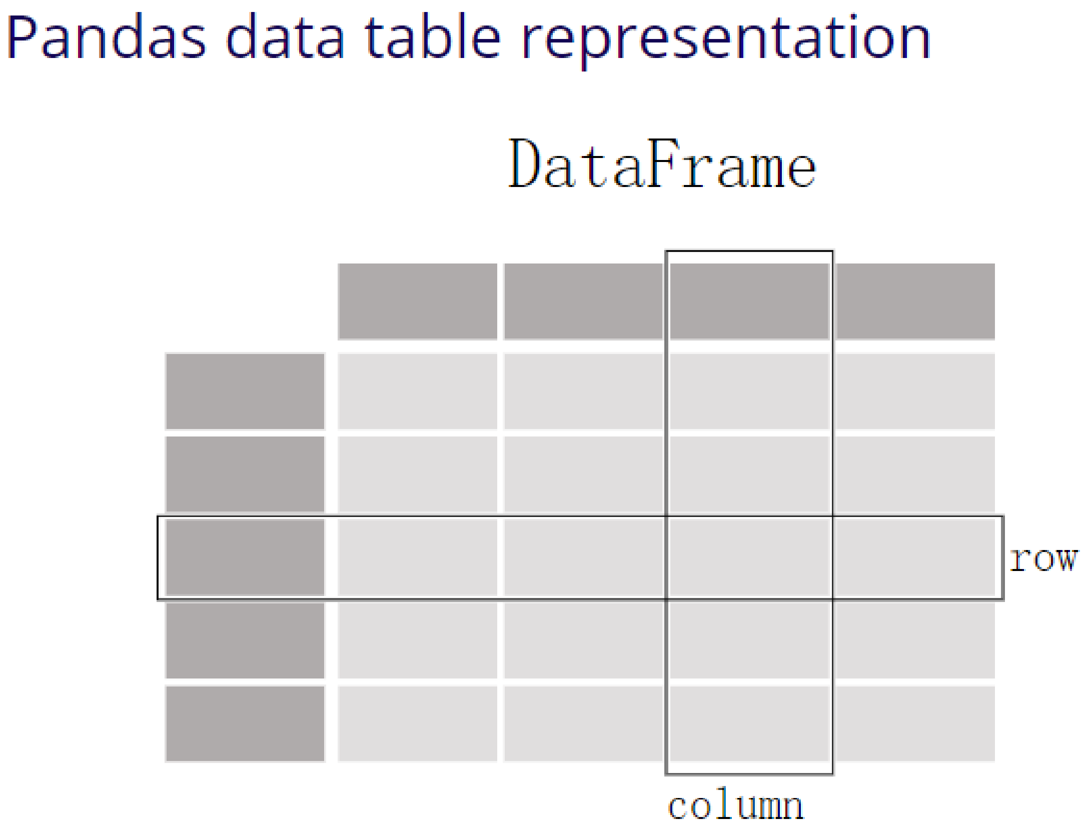
In [62]:
```
#矩阵乘法
a_inv.dot(a)
```

Out[62]:
```
array([[1.00000000e+00, 0.00000000e+00],
       [1.11022302e-16, 1.00000000e+00]])
```

In [63]:
```
np.dot(a_inv,a)
```

Out[63]:
```
array([[1.00000000e+00, 0.00000000e+00],
       [1.11022302e-16, 1.00000000e+00]])
```

1 Pandas 的数据结构



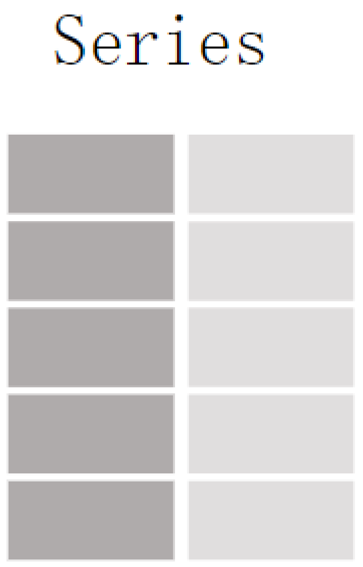
In [2]:
```
import pandas as pd
```

In [2]:
```
# 创建一个 DataFrame
import pandas as pd
df = pd.DataFrame({
    "Name": ["Braund, Mr. Owen Harris",
             "Allen, Mr. William Henry",
             "Bonnell, Miss. Elizabeth"],
    "Age": [22, 35, 58],
    "Sex": ["male", "male", "female"]})
df
```

Out[2]:

	Name	Age	Sex
0	Braund, Mr. Owen Harris	22	male
1	Allen, Mr. William Henry	35	male
2	Bonnell, Miss. Elizabeth	58	female

Each column in a DataFrame is a Series



```
In [3]:  # DataFrame 的每一列为一个 Series
         df['Age']

Out[3]:  0    22
         1    35
         2    58
         Name: Age, dtype: int64
```

```
In [14]: # 另外一种写法
         df.Age

Out[14]: 0    22
         1    35
         2    58
         Name: Age, dtype: int64
```

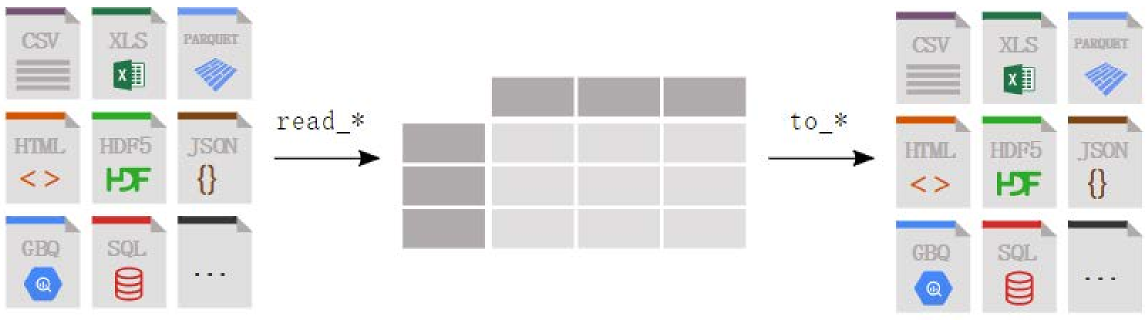
```
In [4]:  # 创建 Series
         prz = pd.Series([10,11,12], name='prz')
         prz

Out[4]:  0    10
         1    11
         2    12
         Name: prz, dtype: int64
```

```
In [5]:  # 统计
         df['Age'].max()

Out[5]:  58
```

```
In [6]:  df['Age'].describe()

Out[6]:  count     3.000000
         mean     38.333333
         std      18.230012
         min      22.000000
         25%      28.500000
         50%      35.000000
```

```
75%      46.500000
max      58.000000
Name: Age, dtype: float64
```

2 表格数据的读写



In [12]:
```
titanic = pd.read_csv("datasets/Titanic.csv")
```

In [8]:
```
# 展示前 5 行
titanic.head()
```

Out[8]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Far
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.283
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

In [9]:
```
# 展示最后 3 行
titanic.tail(3)
```

Out[9]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	C
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	N
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	N

In [11]:
```python
#查看每一列的数据类型
titanic.dtypes
```

Out[11]:
```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

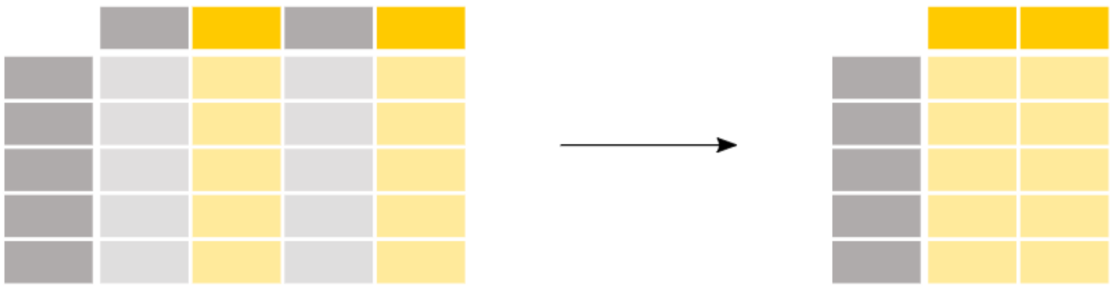
In [12]:
```python
#展示存储信息
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [13]:
```python
#保存为excel文件
titanic.to_excel('Titanic.xlsx', sheet_name='passengers')
```

3 提取数据表的某一部分

3.1 提取列



In [15]:
```python
# 选取 age 和 sex
age_sex = titanic[["Age", "Sex"]]
age_sex.head()
```

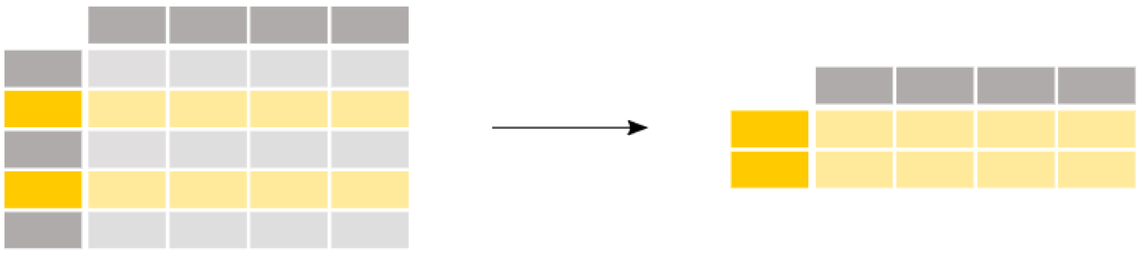
Out[15]:

	Age	Sex
0	22.0	male
1	38.0	female
2	26.0	female
3	35.0	female
4	35.0	male

注意： 结果是什么数据类型？

> 思考：以下两个语句有什么区别？
> `titanic['Age']`
> `titanic[['Age']]`

3.2 提取行



In [17]:
```python
# 提取所有年纪大于 35 岁的乘客信息
```

```
above_35 = titanic[titanic['Age'] > 35]
above_35.head()
```

Out[17]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500
13	14	0	3	Andersson, Mr. Anders Johan	male	39.0	1	5	347082	31.2750
15	16	1	2	Hewlett, Mrs. (Mary D Kingcome)	female	55.0	0	0	248706	16.0000

In [18]:

```
#二等舱和三等舱的乘客信息
class_23 = titanic[titanic["Pclass"].isin([2, 3])]
class_23.head()
```

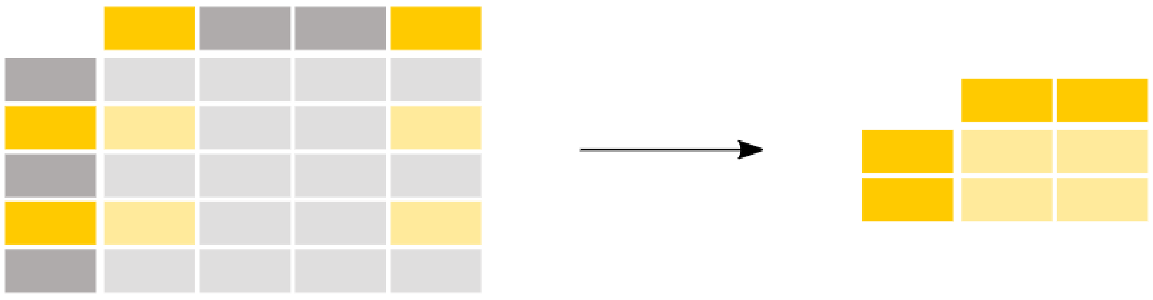
Out[18]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Far
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.075

In [19]:
```
# 以上等价于
class_23 = titanic[(titanic["Pclass"] == 2) | (titanic["Pclass"] == 3)]
```

In [20]:
```
# 提取有年龄信息的乘客
age_no_na = titanic[titanic["Age"].notna()]
```

3.3 提取某一区域



In [21]:
```
# 大于 35 岁的乘客名
adult_names = titanic.loc[titanic["Age"] > 35, "Name"]
adult_names.head()
```

Out[21]:
```
1     Cumings, Mrs. John Bradley (Florence Briggs Th...
6                              McCarthy, Mr. Timothy J
11                            Bonnell, Miss. Elizabeth
13                         Andersson, Mr. Anders Johan
15                    Hewlett, Mrs. (Mary D Kingcome)
Name: Name, dtype: object
```

注意：adult_names 是什么数据类型？

In [23]:
```
# iloc
titanic.iloc[9:13, 2:5]
```

Out[23]:

	Pclass	Name	Sex
9	2	Nasser, Mrs. Nicholas (Adele Achem)	female
10	3	Sandstrom, Miss. Marguerite Rut	female
11	1	Bonnell, Miss. Elizabeth	female
12	3	Saundercock, Mr. William Henry	male

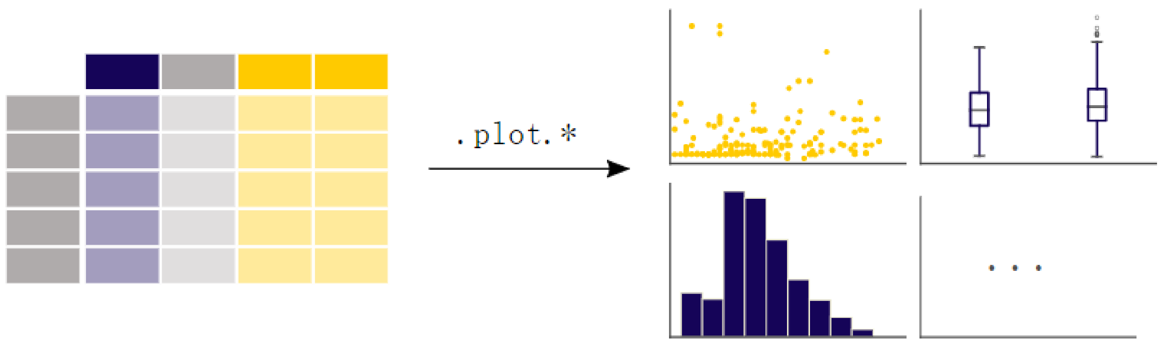
In [24]:
```
# loc
titanic.loc[9:13,['Name','Sex']]
```

Out[24]:

	Name	Sex
9	Nasser, Mrs. Nicholas (Adele Achem)	female
10	Sandstrom, Miss. Marguerite Rut	female

11	Bonnell, Miss. Elizabeth	female
12	Saundercock, Mr. William Henry	male
13	Andersson, Mr. Anders Johan	male

4 作图



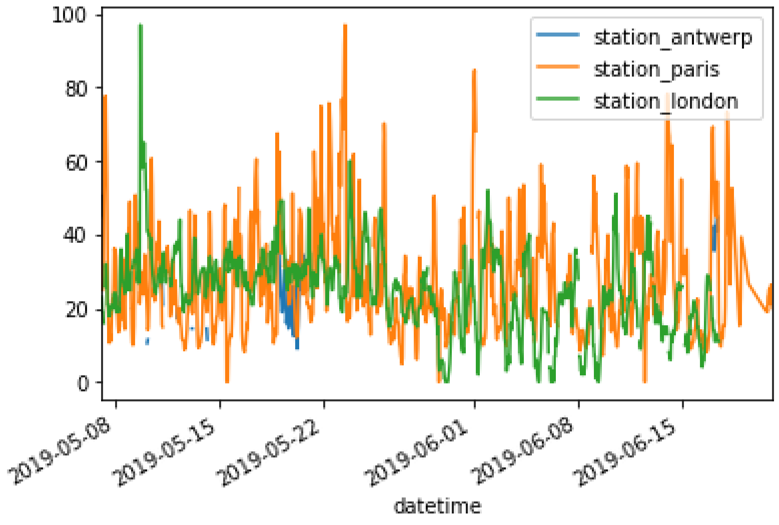
In [3]:
```
# 以下代码读取 csv 文件，并且将第一列设置为 index, 并其转换为日期类型
air_quality = pd.read_csv("datasets/air_quality_no2.csv",index_col=0, pa
rse_dates=True)
air_quality.head()
```

Out[3]:

	station_antwerp	station_paris	station_london
datetime			
2019-05-07 02:00:00	NaN	NaN	23.0
2019-05-07 03:00:00	50.5	25.0	19.0
2019-05-07 04:00:00	45.0	27.7	19.0
2019-05-07 05:00:00	NaN	50.4	16.0
2019-05-07 06:00:00	NaN	61.9	NaN

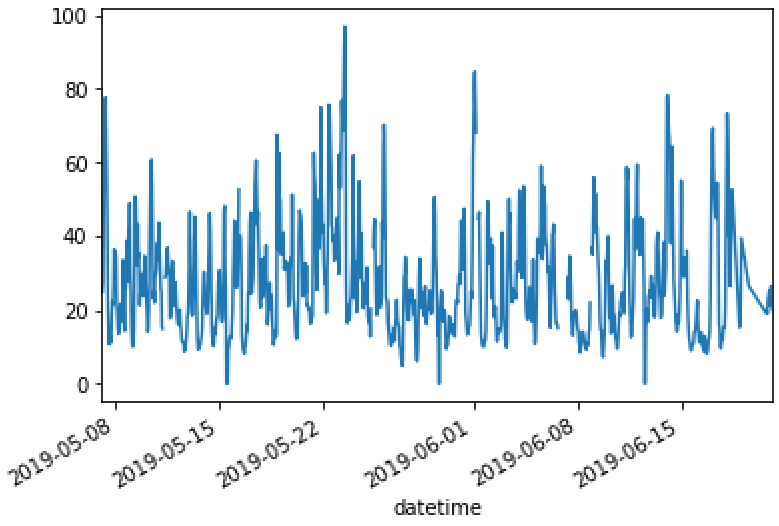
In [29]:
```
# df.plot() 将为 df 中的每一列做折线图
air_quality.plot()
```

Out[29]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21f25cb9508>
```



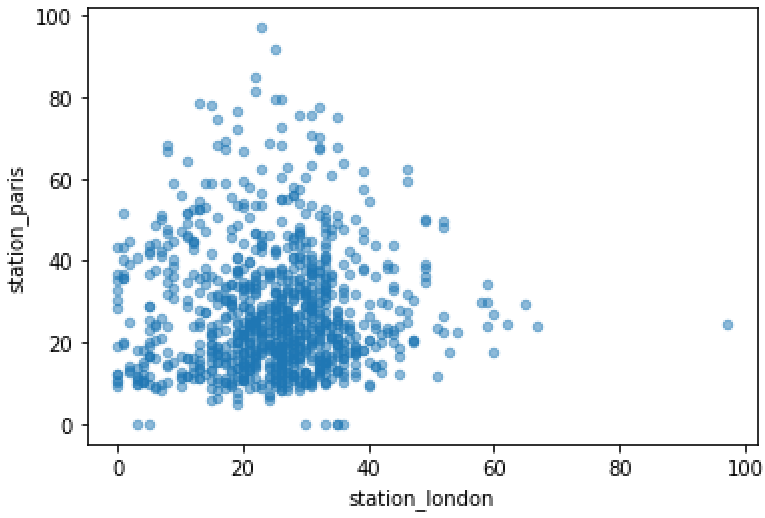
In [30]:
```
#仅仅展示巴黎的数据
air_quality["station_paris"].plot()
```

Out[30]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21f263a8c08>
```



In [31]:
```
#比较伦敦和巴黎的数据，散点图
air_quality.plot.scatter(x="station_london",y="station_paris",alpha=0.5)
```

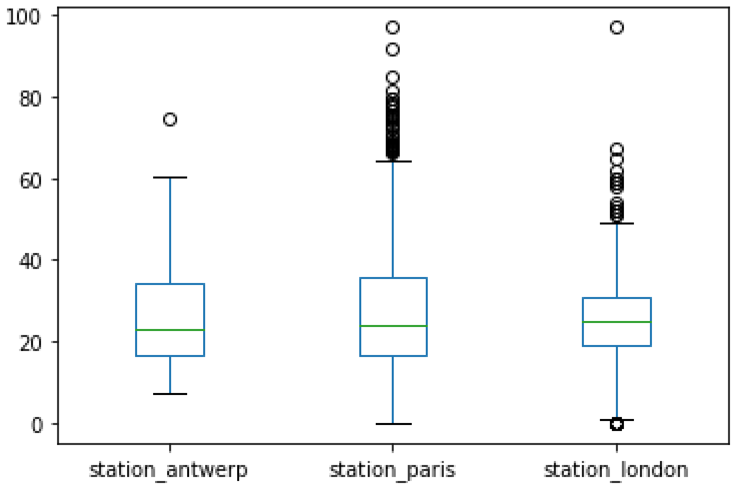
Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21f26463188>
```



In [32]:
```
# Box plot
```

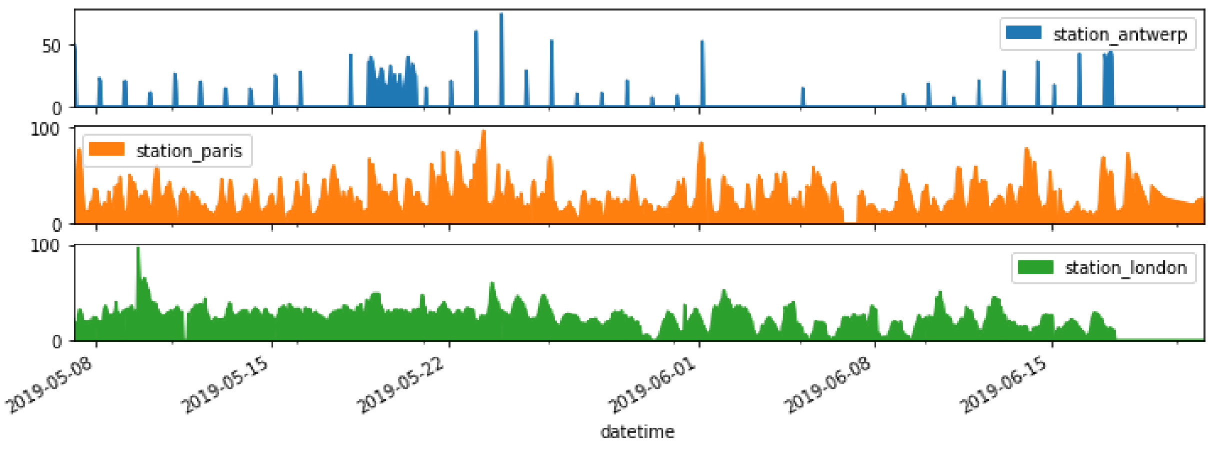
```
air_quality.plot.box()
```

Out[32]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21f264d6648>
```



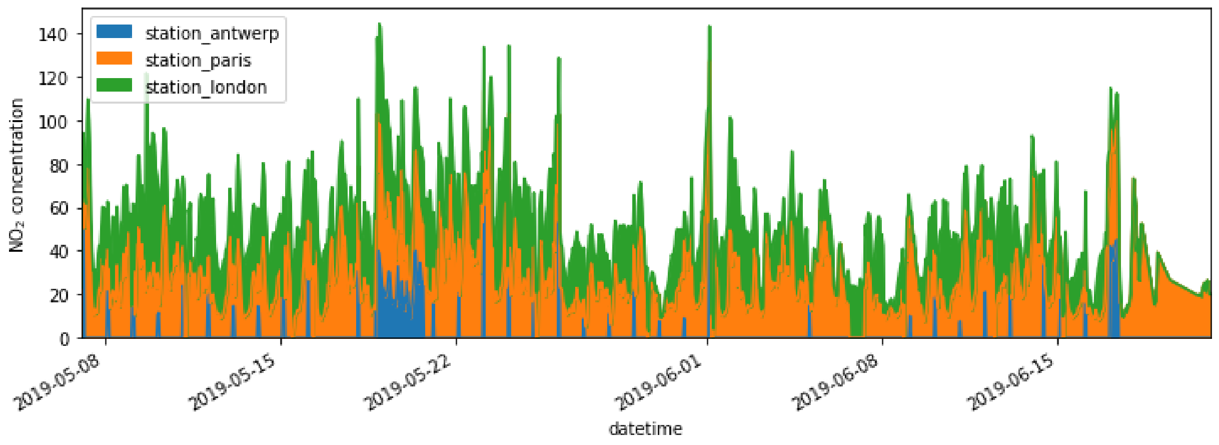
> 箱式图
> 是指—种描述数据分布的统计图，是表述最小值、第一四分位数、中位数、第三四分位数与最大值的一种图形方法。
> 它也可以粗略地看出数据是否具有对称性，分布的分散程度等信息。
> 在箱图中，最上方和最下方的线段分别表示数据的最大值和最小值，其中箱图的上方和下方的线段分别表示第三四分位数和第一四分位数，箱图中间的粗线段表示数据的中位数。 计算IQR(四分位数间距)即IQR=Q3-Q1,所有不在（Q1-1.5IQR，Q3+1.5IQR）的区间内的数为离群值，剩下的值最大的为最大值，最小的为最小值

In [36]:
```
# 显示在不同的图上
ax = air_quality.plot.area(figsize=(12, 4), subplots=True)
```



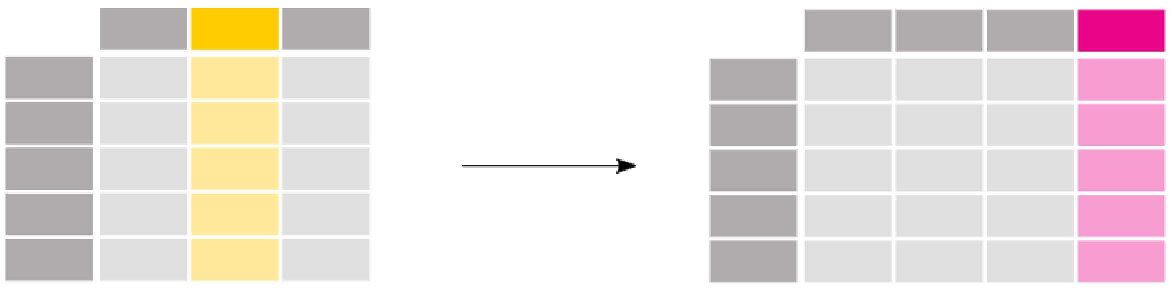
In [40]:
```
# 与 plt 结合使用，实现更多定制化信息
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(12,4))
air_quality.plot.area(ax=ax)
ax.set_ylabel("NO$_2$ concentration")
```

Out[40]:
```
Text(0, 0.5, 'NO$_2$ concentration')
```



More plot: https://pandas.pydata.org/docs/user_guide/visualization.html#visualization-other
https://pandas.pydata.org/docs/user_guide/visualization.html#visualization-formatting

5 新增列



In [4]:
```
# NO2 concentration --> mg/m3, conversion factor = 1.882
air_quality["london_mg_per_cubic"] = air_quality["station_london"] * 1.8
82
air_quality.head(3)
```

Out[4]:

	station_antwerp	station_paris	station_london	london_mg_per_cubic
datetime				
2019-05-07 02:00:00	NaN	NaN	23.0	43.286
2019-05-07 03:00:00	50.5	25.0	19.0	35.758
2019-05-07 04:00:00	45.0	27.7	19.0	35.758

In [5]:
```
# 两列的比例
air_quality["ratio_paris_antwerp"] = air_quality["station_paris"] / air_
quality["station_antwerp"]
air_quality.head(3)
```

Out[5]:

	station_antwerp	station_paris	station_london	london_mg_per_cubic	ratio_pari
datetime					
2019-05-07 02:00:00	NaN	NaN	23.0	43.286	NaN
2019-05-07 03:00:00	50.5	25.0	19.0	35.758	0.495050
2019-05-07 04:00:00	45.0	27.7	19.0	35.758	0.615556

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

6 重命名

In [2]:
```
air_quality = pd.read_csv("datasets/air_quality_no2.csv",index_col=0, pa
rse_dates=True)
air_quality.head()
```

Out[2]:

	station_antwerp	station_paris	station_london
datetime			
2019-05-07 02:00:00	NaN	NaN	23.0
2019-05-07 03:00:00	50.5	25.0	19.0
2019-05-07 04:00:00	45.0	27.7	19.0
2019-05-07 05:00:00	NaN	50.4	16.0
2019-05-07 06:00:00	NaN	61.9	NaN

In [4]:
```
air_quality_renamed = air_quality.rename(
    columns={"station_antwerp": "BETR801",
             "station_paris": "FR04014",
             "station_london": "London Westminster"})
air_quality_renamed.head()
```

Out[4]:

	BETR801	FR04014	London Westminster
datetime			
2019-05-07 02:00:00	NaN	NaN	23.0
2019-05-07 03:00:00	50.5	25.0	19.0
2019-05-07 04:00:00	45.0	27.7	19.0
2019-05-07 05:00:00	NaN	50.4	16.0
2019-05-07 06:00:00	NaN	61.9	NaN

In [5]:
```
air_quality_renamed = air_quality_renamed.rename(columns=str.lower)
air_quality_renamed.head()
```

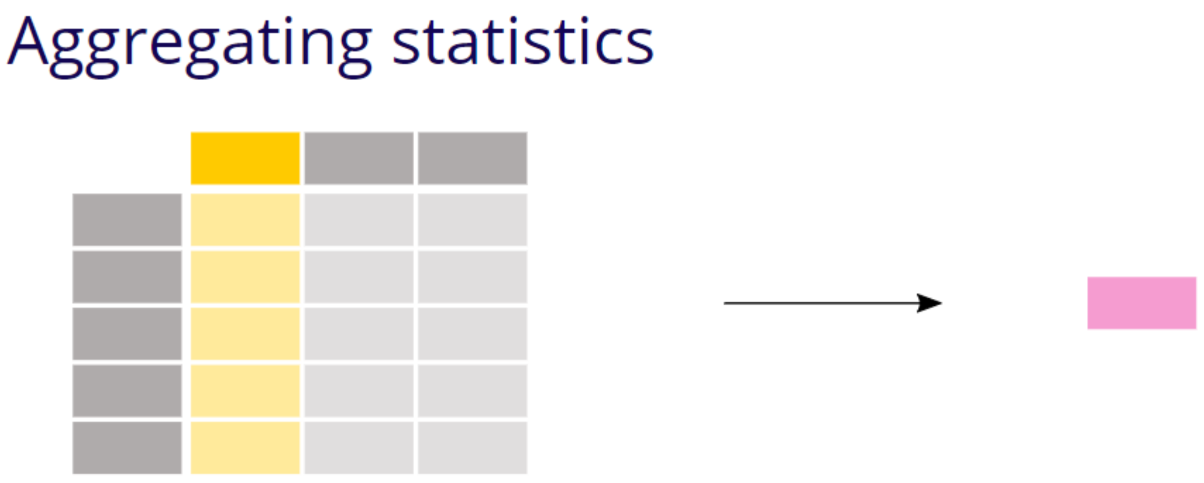
Out[5]:

	betr801	fr04014	london westminster
datetime			
2019-05-07 02:00:00	NaN	NaN	23.0
2019-05-07 03:00:00	50.5	25.0	19.0
2019-05-07 04:00:00	45.0	27.7	19.0
2019-05-07 05:00:00	NaN	50.4	16.0

2019-05-07 06:00:00	NaN	61.9	NaN

7 统计

7.1 整体统计



```
In [5]:  # titanic 乘客的平均年龄
         titanic = pd.read_csv("datasets/Titanic.csv")
         titanic["Age"].mean()

Out[5]:  29.69911764705882
```

```
In [14]:  # 年龄和票价的平均值
          titanic[["Age", "Fare"]].median()

Out[14]:  Age      28.0000
          Fare     14.4542
          dtype: float64
```

注意：返回值是什么类型？

```
In [15]:  titanic[["Age", "Fare"]].describe()
```

Out[15]:

	Age	Fare
count	714.000000	891.000000
mean	29.699118	32.204208
std	14.526497	49.693429
min	0.420000	0.000000
25%	20.125000	7.910400
50%	28.000000	14.454200
75%	38.000000	31.000000
max	80.000000	512.329200

In [17]:
```
# 自定义统计指标
titanic.agg({'Age': ['min', 'max', 'median', 'skew'],
             'Fare': ['min', 'max', 'median', 'mean']})
```

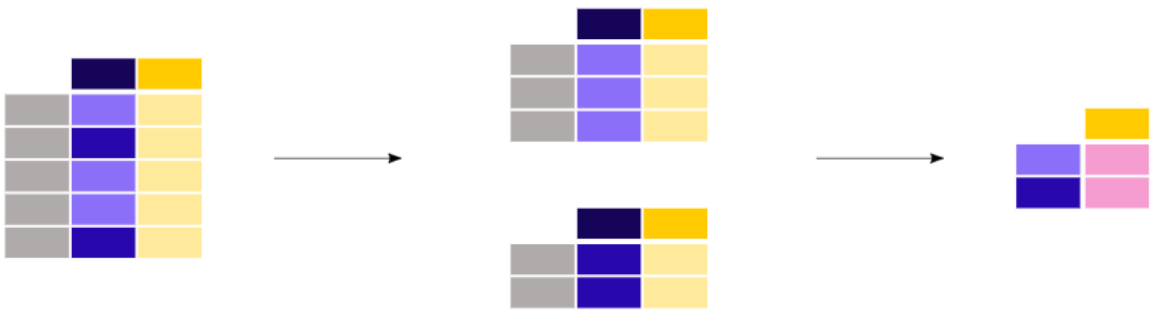
Out[17]:

	Age	Fare
max	80.000000	512.329200
mean	NaN	32.204208
median	28.000000	14.454200
min	0.420000	0.000000
skew	0.389108	NaN

更多统计指标：https://pandas.pydata.org/docs/getting_started/basics.html#basics-stats

7.2 分组统计

Aggregating statistics grouped by category



In [18]:
```
# 男女乘客的平均年龄
titanic[["Sex", "Age"]].groupby("Sex").mean()
```

Out[18]:

	Age
Sex	
female	27.915709
male	30.726645

以上代码执行了三步操作：`split-apply-combine`

In [20]:
```
# 将 groupby 直接作用于整个数据集
titanic.groupby("Sex").mean()
```

Out[20]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare

Sex							
female	431.028662	0.742038	2.159236	27.915709	0.694268	0.649682	44.479818
male	454.147314	0.188908	2.389948	30.726645	0.429809	0.235702	25.523893

In [22]:
```
#分多层统计
titanic.groupby(["Sex", "Pclass"])["Fare"].mean()
```

Out[22]:
```
Sex     Pclass
female  1         106.125798
        2          21.970121
        3          16.118810
male    1          67.226127
        2          19.741782
        3          12.661633
Name: Fare, dtype: float64
```

In [23]:
```
#分组计数
titanic["Pclass"].value_counts()
```

Out[23]:
```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

8 调整表的形态结构

8.1 排序

In [24]:
```
#按照年龄排序
titanic.sort_values(by="Age").head()
```

Out[24]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Ca
803	804	1	3	Thomas, Master. Assad Alexander	male	0.42	0	1	2625	8.5167	N
755	756	1	2	Hamalainen, Master. Viljo	male	0.67	1	1	250649	14.5000	N
644	645	1	3	Baclini, Miss. Eugenie	female	0.75	2	1	2666	19.2583	N
469	470	1	3	Baclini, Miss. Helene Barbara	female	0.75	2	1	2666	19.2583	N
78	79	1	2	Caldwell, Master. Alden Gates	male	0.83	0	2	248738	29.0000	N

In [26]:
```python
# 按照 Pclass 和 年龄 降序排序： 先按 Pclass 排，Pclass 相同的按照 Age 排序排
titanic.sort_values(by=['Pclass', 'Age'], ascending=False).head()
```

Out[26]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabir
851	852	0	3	Svensson, Mr. Johan	male	74.0	0	0	347060	7.7750	Na
116	117	0	3	Connors, Mr. Patrick	male	70.5	0	0	370369	7.7500	Na
280	281	0	3	Duane, Mr. Frank	male	65.0	0	0	336439	7.7500	Na
483	484	1	3	Turkula, Mrs. (Hedwig)	female	63.0	0	0	4134	9.5875	Na
326	327	0	3	Nysveen, Mr. Johan Hansen	male	61.0	0	0	345364	6.2375	Na

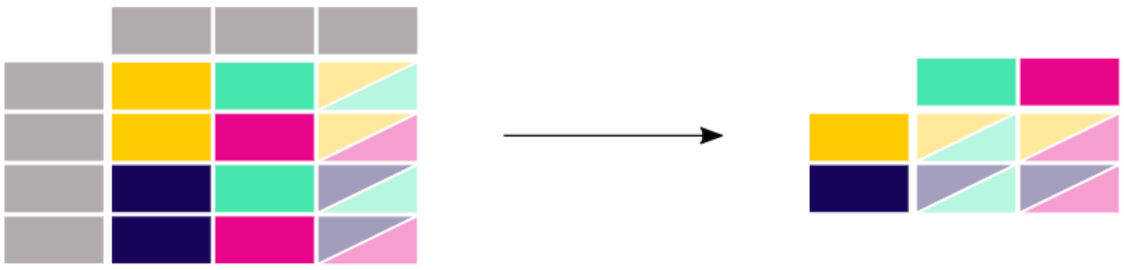
In [28]:
```python
# sort index
titanic.sort_index().head()
```

Out[28]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cab
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	Na
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	Na
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	Na

注意： `sort_index` 和 `sort_values` 都不改变原数据表

8.2 Long to Wide



In [29]:
```python
air_quality = pd.read_csv("datasets/air_quality_long.csv",index_col="date.utc", parse_dates=True)
air_quality.head()
```

Out[29]:

date.utc	city	country	location	parameter	value	unit
2019-06-18 06:00:00+00:00	Antwerpen	BE	BETR801	pm25	18.0	µg/m³
2019-06-17 08:00:00+00:00	Antwerpen	BE	BETR801	pm25	6.5	µg/m³
2019-06-17 07:00:00+00:00	Antwerpen	BE	BETR801	pm25	18.5	µg/m³
2019-06-17 06:00:00+00:00	Antwerpen	BE	BETR801	pm25	16.0	µg/m³
2019-06-17 05:00:00+00:00	Antwerpen	BE	BETR801	pm25	7.5	µg/m³

In [34]:
```python
# filter for no2 data only
no2 = air_quality[air_quality["parameter"] == "no2"]
no2.head()
```

Out[34]:

date.utc	city	country	location	parameter	value	unit
2019-06-21 00:00:00+00:00	Paris	FR	FR04014	no2	20.0	µg/m³
2019-06-20 23:00:00+00:00	Paris	FR	FR04014	no2	21.8	µg/m³
2019-06-20 22:00:00+00:00	Paris	FR	FR04014	no2	26.5	µg/m³
2019-06-20 21:00:00+00:00	Paris	FR	FR04014	no2	24.9	µg/m³
2019-06-20 20:00:00+00:00	Paris	FR	FR04014	no2	21.4	µg/m³

In [40]:
```python
no2.pivot(columns="location", values="value")
```

Out[40]:

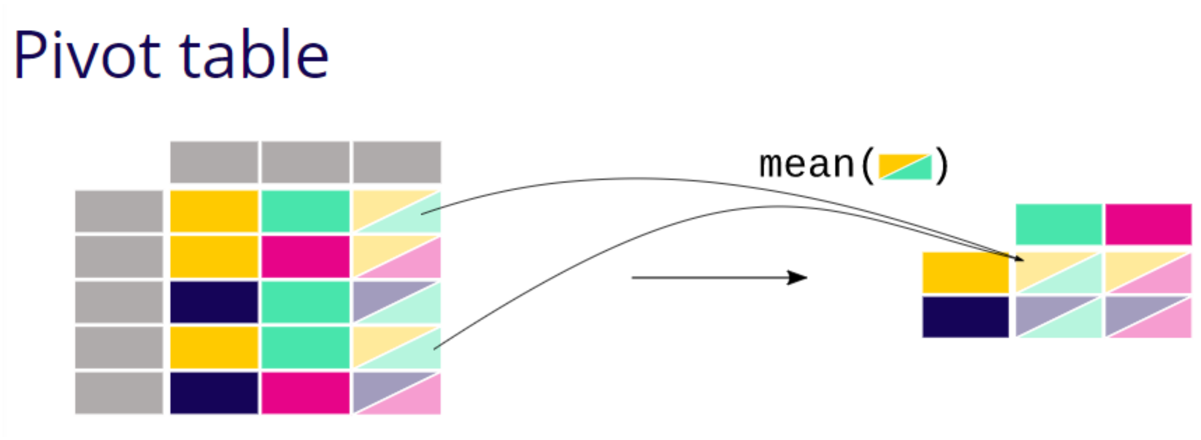
location	BETR801	FR04014	London Westminster
date.utc			
2019-04-09 01:00:00+00:00	22.5	24.4	NaN
2019-04-09 02:00:00+00:00	53.5	27.4	67.0
2019-04-09 03:00:00+00:00	54.5	34.2	67.0
2019-04-09 04:00:00+00:00	34.5	48.5	41.0
2019-04-09 05:00:00+00:00	46.5	59.5	41.0

...	...	...	...
2019-06-20 20:00:00+00:00	NaN	21.4	NaN
2019-06-20 21:00:00+00:00	NaN	24.9	NaN
2019-06-20 22:00:00+00:00	NaN	26.5	NaN
2019-06-20 23:00:00+00:00	NaN	21.8	NaN
2019-06-21 00:00:00+00:00	NaN	20.0	NaN

1705 rows × 3 columns

思考：要实现以上变化，有什么先决条件？

8.3 数据透视表



In [41]:
```python
# 每个 station 的 NO2 、 PM2.5 的均值
air_quality = pd.read_csv('datasets/air_quality_long.csv',index_col='dat
e.utc',parse_dates=True)
air_quality.pivot_table(values="value", index="location",columns="parame
ter", aggfunc="mean")
```

Out[41]:

parameter	no2	pm25
location		
BETR801	26.950920	23.169492
FR04014	29.374284	NaN
London Westminster	29.740050	13.443568

思考： `pivot` 和 `pivot_table` 有什么区别？

In [42]:
```python
# margin = True
air_quality.pivot_table(values="value", index="location",
                        columns="parameter", aggfunc="mean",
                        margins=True)
```

Out[42]:

parameter	no2	pm25	All
location			

BETR801	26.950920	23.169492	24.982353
FR04014	29.374284	NaN	29.374284
London Westminster	29.740050	13.443568	21.491708
All	29.430316	14.386849	24.222743

In [44]:
```
# 以上统计也可以通过 groupby 获得
air_quality.groupby(["parameter", "location"]).mean()
```

Out[44]:

		value
parameter	location	
no2	BETR801	26.950920
	FR04014	29.374284
	London Westminster	29.740050
pm25	BETR801	23.169492
	London Westminster	13.443568

8.4 Wide to long

In [45]:
```
no2_pivoted = no2.pivot(columns="location", values="value").reset_index(
)
no2_pivoted.head()
```

Out[45]:

location	date.utc	BETR801	FR04014	London Westminster
0	2019-04-09 01:00:00+00:00	22.5	24.4	NaN
1	2019-04-09 02:00:00+00:00	53.5	27.4	67.0
2	2019-04-09 03:00:00+00:00	54.5	34.2	67.0
3	2019-04-09 04:00:00+00:00	34.5	48.5	41.0
4	2019-04-09 05:00:00+00:00	46.5	59.5	41.0

In [47]:
```
# 在一列中显示 NO2
no_2 = no2_pivoted.melt(id_vars="date.utc")
no_2.head()
```

Out[47]:

	date.utc	location	value
0	2019-04-09 01:00:00+00:00	BETR801	22.5
1	2019-04-09 02:00:00+00:00	BETR801	53.5
2	2019-04-09 03:00:00+00:00	BETR801	54.5
3	2019-04-09 04:00:00+00:00	BETR801	34.5
4	2019-04-09 05:00:00+00:00	BETR801	46.5

9 合并

In [49]:
```
air_quality_no2 = pd.read_csv("datasets/air_quality_no2_long.csv",parse_
dates=True)
air_quality_no2 = air_quality_no2[["date.utc", "location","parameter", "
value"]]
air_quality_no2.head()
```

Out[49]:

	date.utc	location	parameter	value
0	2019-06-21 00:00:00+00:00	FR04014	no2	20.0
1	2019-06-20 23:00:00+00:00	FR04014	no2	21.8
2	2019-06-20 22:00:00+00:00	FR04014	no2	26.5
3	2019-06-20 21:00:00+00:00	FR04014	no2	24.9
4	2019-06-20 20:00:00+00:00	FR04014	no2	21.4

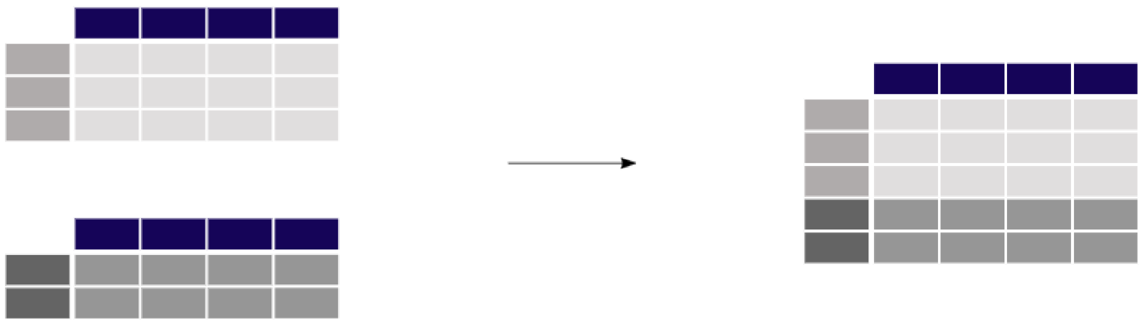
In [51]:
```
air_quality_pm25 = pd.read_csv("datasets/air_quality_pm25_long.csv",pars
e_dates=True)
air_quality_pm25 = air_quality_pm25[["date.utc", "location","parameter",
 "value"]]
air_quality_pm25.head()
```

Out[51]:

	date.utc	location	parameter	value
0	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0
1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5
2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5
3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0
4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5

9.1 Concat

Concatenating objects



In [53]:
```
air_quality = pd.concat([air_quality_pm25, air_quality_no2], axis=0) #纵
向连接
air_quality.head()
```

Out[53]:

	date.utc	location	parameter	value
0	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0
1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5
2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5
3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0
4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5

In [58]:

```
# 可以为新的表分配新的 index
air_quality_ = pd.concat([air_quality_pm25, air_quality_no2],axis=0,
                         keys=["PM25", "NO2"])
air_quality_
```

Out[58]:

		date.utc	location	parameter	value
PM25	0	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0
	1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5
	2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5
	3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0
	4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5
...	...	...	...	...	...
NO2	2063	2019-05-07 06:00:00+00:00	London Westminster	no2	26.0
	2064	2019-05-07 04:00:00+00:00	London Westminster	no2	16.0
	2065	2019-05-07 03:00:00+00:00	London Westminster	no2	19.0
	2066	2019-05-07 02:00:00+00:00	London Westminster	no2	19.0
	2067	2019-05-07 01:00:00+00:00	London Westminster	no2	23.0

3178 rows × 4 columns

In [63]:

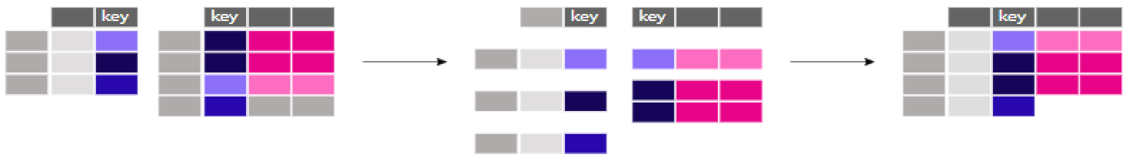
```
# air_quaility_ 具有 MultiIndex，使用 reset_index 可将其转换为普通的表
air_quality_.reset_index().head()
```

Out[63]:

	level_0	level_1	date.utc	location	parameter	value
0	PM25	0	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0
1	PM25	1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5
2	PM25	2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5
3	PM25	3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0
4	PM25	4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5

9.2 Join

Join tables using a common identifier



In [64]:
```
stations_coord = pd.read_csv("datasets/air_quality_stations.csv")
stations_coord.head()
```

Out[64]:

	location	coordinates.latitude	coordinates.longitude
0	BELAL01	51.23619	4.38522
1	BELHB23	51.17030	4.34100
2	BELLD01	51.10998	5.00486
3	BELLD02	51.12038	5.02155
4	BELR833	51.32766	4.36226

In [65]:
```
air_quality.head()
```

Out[65]:

	date.utc	location	parameter	value
0	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0
1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5
2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5
3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0
4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5

In [66]:
```
# 在 air_quality 表中，增加 location 对应的经度和维度数据
air_quality = pd.merge(air_quality, stations_coord,how='left',on='locati
on')
```

In [68]:
```
air_quality.head()
```

Out[68]:

	date.utc	location	parameter	value	coordinates.latitude	coordinates.longitude
0	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0	51.20966	4.43182
1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5	51.20966	4.43182
2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5	51.20966	4.43182
3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0	51.20966	4.43182
4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5	51.20966	4.43182

`how = 'left'` 表示：对 `air_quality` 中的每一条数据进行链接，即 `air_quality` 中的每一条数据都出现在最终的数据表中，其他的选项有 `right` , `outer` , `inner`

`on='location'` 表示；链接关系的 key

如果 `key` 在两张表中名称不一样，应该怎么办?

In [69]:
```
air_quality_parameters = pd.read_csv("datasets/air_quality_parameters.cs
v")
air_quality_parameters.head()
```

Out[69]:

	id	description	name
0	bc	Black Carbon	BC
1	co	Carbon Monoxide	CO
2	no2	Nitrogen Dioxide	NO2
3	o3	Ozone	O3
4	pm10	Particulate matter less than 10 micrometers in...	PM10

In [70]:
```
air_quality.head()
```

Out[70]:

	date.utc	location	parameter	value	coordinates.latitude	coordinates.longitude
0	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0	51.20966	4.43182
1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5	51.20966	4.43182
2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5	51.20966	4.43182
3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0	51.20966	4.43182
4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5	51.20966	4.43182

In [71]:
```
# air_quanlity 的 parameter 相当于 air_quality_parameters 中的 id
air_quality = pd.merge(air_quality, air_quality_parameters,
                       how='left', left_on='parameter', right_on='id')
air_quality.head()
```

Out[71]:

	date.utc	location	parameter	value	coordinates.latitude	coordinates.longitude	id	d
0	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0	51.20966	4.43182	pm25	n
1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5	51.20966	4.43182	pm25	n

2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5	51.20966	4.43182	pm25	n
3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0	51.20966	4.43182	pm25	n
4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5	51.20966	4.43182	pm25	n

10 时间序列

In [3]:
```
air_quality = pd.read_csv("datasets/air_quality_no2_long.csv")
air_quality = air_quality.rename(columns={"date.utc": "datetime"})
air_quality.head()
```

Out[3]:

	city	country	datetime	location	parameter	value	unit
0	Paris	FR	2019-06-21 00:00:00+00:00	FR04014	no2	20.0	µg/m³
1	Paris	FR	2019-06-20 23:00:00+00:00	FR04014	no2	21.8	µg/m³
2	Paris	FR	2019-06-20 22:00:00+00:00	FR04014	no2	26.5	µg/m³
3	Paris	FR	2019-06-20 21:00:00+00:00	FR04014	no2	24.9	µg/m³
4	Paris	FR	2019-06-20 20:00:00+00:00	FR04014	no2	21.4	µg/m³

10.1 处理日期

In [6]:
```
# 将 datetime 列转换为日期
air_quality["datetime"] = pd.to_datetime(air_quality["datetime"])
```

In [7]:
```
air_quality["datetime"].min(), air_quality["datetime"].max()
```

Out[7]:
```
(Timestamp('2019-05-07 01:00:00+0000', tz='UTC'),
 Timestamp('2019-06-21 00:00:00+0000', tz='UTC'))
```

In [8]:
```
air_quality["datetime"].max() - air_quality["datetime"].min()
```

Out[8]:
```
Timedelta('44 days 23:00:00')
```

In [9]:
```
air_quality["month"] = air_quality["datetime"].dt.month
air_quality.head()
```

Out[9]:

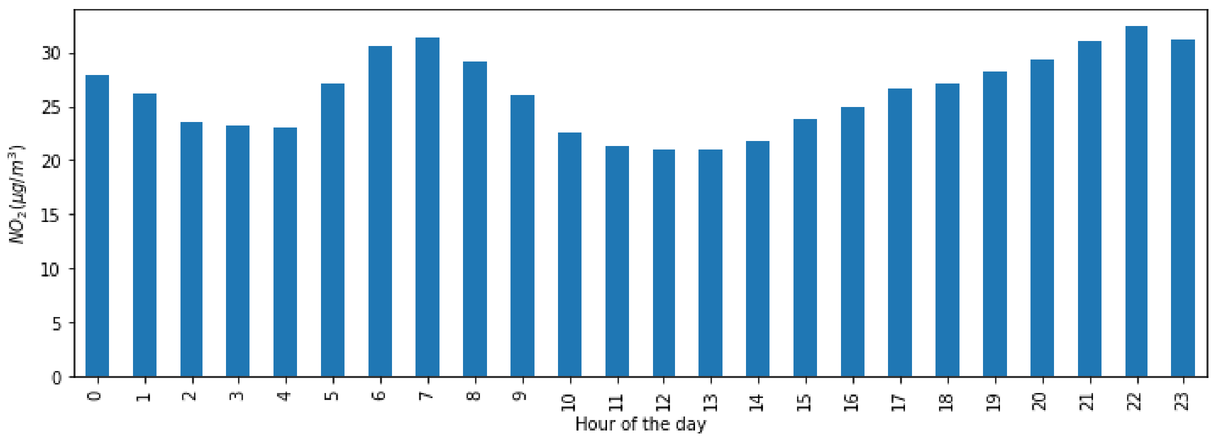
	city	country	datetime	location	parameter	value	unit	month

0	Paris	FR	2019-06-21 00:00:00+00:00	FR04014	no2	20.0	µg/m³	6
1	Paris	FR	2019-06-20 23:00:00+00:00	FR04014	no2	21.8	µg/m³	6
2	Paris	FR	2019-06-20 22:00:00+00:00	FR04014	no2	26.5	µg/m³	6
3	Paris	FR	2019-06-20 21:00:00+00:00	FR04014	no2	24.9	µg/m³	6
4	Paris	FR	2019-06-20 20:00:00+00:00	FR04014	no2	21.4	µg/m³	6

In [10]:
```
# 统计一周中每一天、每个 location 的 NO2 的集中度
air_quality.groupby([air_quality["datetime"].dt.weekday, "location"])["v
alue"].mean()
```

Out[10]:
```
datetime  location          
0         BETR801               27.875000
          FR04014               24.856250
          London Westminster    23.969697
1         BETR801               22.214286
          FR04014               30.999359
          London Westminster    24.885714
2         BETR801               21.125000
          FR04014               29.165753
          London Westminster    23.460432
3         BETR801               27.500000
          FR04014               28.600690
          London Westminster    24.780142
4         BETR801               28.400000
          FR04014               31.617986
          London Westminster    26.446809
5         BETR801               33.500000
          FR04014               25.266154
          London Westminster    24.977612
6         BETR801               21.896552
          FR04014               23.274306
          London Westminster    24.859155
Name: value, dtype: float64
```

In [12]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(12, 4))

air_quality.groupby(air_quality["datetime"].dt.hour)["value"].\
    mean().plot(kind='bar',ax=ax)

plt.xlabel("Hour of the day");  # custom x label using matplotlib
plt.ylabel("$NO_2 (µg/m^3)$");
```



In [14]:
```
# 将 datetime 设置为 index
no_2 = air_quality.pivot(index="datetime", columns="location", values="v
alue")
no_2.head()
```

Out[14]:

location	BETR801	FR04014	London Westminster
datetime			
2019-05-07 01:00:00+00:00	50.5	25.0	23.0
2019-05-07 02:00:00+00:00	45.0	27.7	19.0
2019-05-07 03:00:00+00:00	NaN	50.4	19.0
2019-05-07 04:00:00+00:00	NaN	61.9	16.0
2019-05-07 05:00:00+00:00	NaN	72.4	NaN

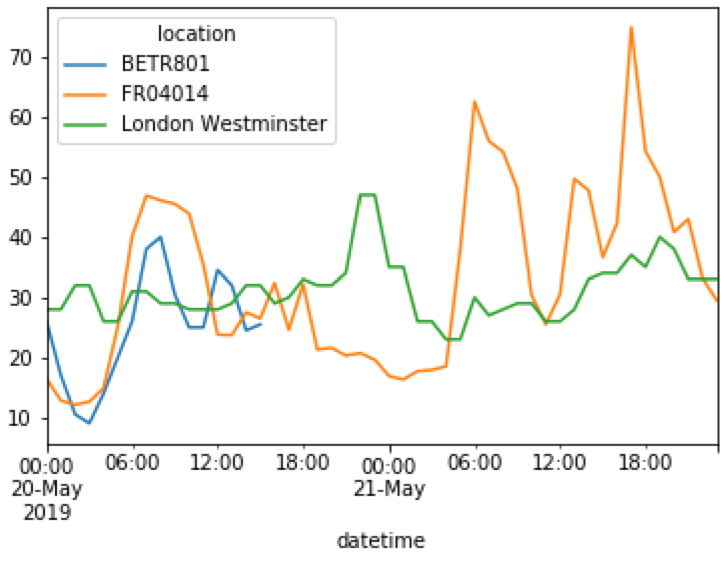
In [16]:
```
no_2.index.year, no_2.index.weekday
```

Out[16]:
```
(Int64Index([2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019,
             ...
             2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019]
,
            dtype='int64', name='datetime', length=1033),
 Int64Index([1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
             ...
             3, 3, 3, 3, 3, 3, 3, 3, 3, 4],
            dtype='int64', name='datetime', length=1033))
```

In [17]:
```
# 5 月 20 日 到 5 月 21 日的数据
no_2["2019-05-20":"2019-05-21"].plot()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\arrays\datetimes.
py:1269: UserWarning: Converting to PeriodArray/Index representation wil
l drop timezone information.
  UserWarning,
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2426295e188>
```



10.2 重采样

In [18]:
```
# 月度 resample
monthly_max = no_2.resample("M").max()
monthly_max
```

Out[18]:

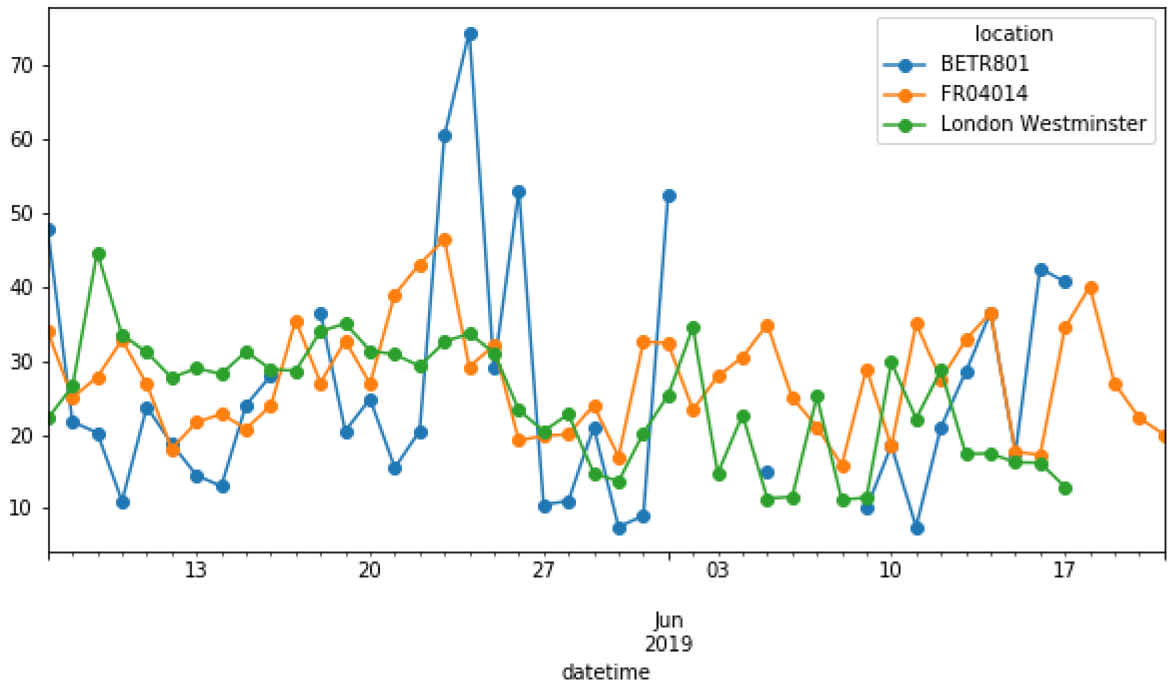
location	BETR801	FR04014	London Westminster
datetime			
2019-05-31 00:00:00+00:00	74.5	97.0	97.0
2019-06-30 00:00:00+00:00	52.5	84.7	52.0

In [19]:
```
monthly_max.index.freq
```

Out[19]:
```
<MonthEnd>
```

In [20]:
```
# 每个 station 的每日均值
no_2.resample("D").mean().plot(style="-o", figsize=(10, 5))
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2426293ec88>
```



11 处理文本数据

In [22]:
```
titanic = pd.read_csv("datasets/Titanic.csv")
titanic.head()
```

Out[22]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cab
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	Na
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs	female	38.0	1	0	PC 17599	71.2833	C

				Th...							
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	Na
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	Na

In [23]:
```
# 名字变小写
titanic["Name"].str.lower()
```

Out[23]:
```
0                              braund, mr. owen harris
1      cumings, mrs. john bradley (florence briggs th...
2                               heikkinen, miss. laina
3         futrelle, mrs. jacques heath (lily may peel)
4                             allen, mr. william henry
                             ...
886                              montvila, rev. juozas
887                       graham, miss. margaret edith
888           johnston, miss. catherine helen "carrie"
889                              behr, mr. karl howell
890                                dooley, mr. patrick
Name: Name, Length: 891, dtype: object
```

In [25]:
```
# Surname
titanic["Surname"] = titanic["Name"].str.split(",").str.get(0)
titanic.head()
```

Out[25]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cab
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	Na
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	Na
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C1
				Allen, Mr.							

4	5	0	3	William Henry	male	35.0	0	0	373450	8.0500	Na

In [28]:
```
# 找出 Countess 伯爵夫人
titanic[titanic["Name"].str.contains("Countess")]
```

Out[28]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
759	760	1	1	Rothes, the Countess. of (Lucy Noel Martha Dye...	female	33.0	0	0	110152	86.5	B77

In [29]:
```
# 那个人的名字最长
titanic["Name"].str.len().idxmax()
```

Out[29]:
```
307
```

In [30]:
```
titanic.loc[titanic["Name"].str.len().idxmax(), "Name"]
```

Out[30]:
```
'Penasco y Castellana, Mrs. Victor de Satode (Maria Josefa Perez de Soto
 y Vallejo)'
```

In [32]:
```
# 在 sex 列，将 male female 分别替换为 M F
titanic["Sex_short"] = titanic["Sex"].replace({"male": "M",
                                               "female": "F"})
titanic["Sex_short"]
```

Out[32]:
```
0      M
1      F
2      F
3      F
4      M
      ..
886    M
887    F
888    F
889    M
890    M
Name: Sex_short, Length: 891, dtype: object
```