Python基础知识（三）

陈赟

对外经济贸易大学金融学院
yunchen@uibe.edu.cn

2022年10月17日

目录

一、函数的基本使用

函数

- 函数（function）是带名字的代码块，用于完成具体的工作
- 执行函数定义的特定任务时，可调用该函数
- 程序中多次执行同一项任务时，无须反复编写完成该任务的代码，只需要调用执行该任务的函数即可
- 通过使用函数，程序编写、阅读、测试和修复起来都更加容易
- 内建函数（built-in functions）与自定义函数

函数的基本结构

- 示例

```
>>> def greet_user():
...     """ Show simple greeting """
...     print("Hello !")
...
>>> greet_user()
Hello !
```

- 关键字def告诉python后面是定义函数，并指出了函数名
- 括号里面可以为空，也可以指出函数为完成任务需要什么信息，比如输入变量X
- 后面的所有缩进构成了函数体，第一行注释称为文档字符串（docstring）的注释，描述了函数是做什么的

函数的返回值

- 函数除执行一些任务外，还可返回一些结果（数、字符串、字典、列表以及复杂的数据结构等），函数返回的值称为返回值。return语句将值返回到调用函数的代码行

```
>>> def add_num(x, y):
...     print("%d + %d = %d" % (x, y, x+y))
...     return x + y
...
>>> res = add_num(1, 2)
1 + 2 = 3
>>> print(res)
3
```

函数的返回值

- 函数可返回任何类型的值，包括列表和字典等较复杂的数据结构

```
>>> def uibe_transcript(course, score):
...     transcript = {"course": course, "score": score}
...     return transcript
...
>>> my_script = uibe_transcript("Python", 100)
>>> my_script
{'course': 'Python', 'score': 100}
```

- 函数也可以没有return语句，但此时函数仍然有返回值：None
- 函数可以返回多个对象（即多个返回值），返回值更像是元组

传递参数

- 示例：

```
>>> def greet_user(username):
...     print("Hello,", username.title())
...
>>> greet_user("python")
Hello, Python
```

- 变量username是一个形参(parameter)，即函数完成工作所需的信息
- greet_user("python")中的"python"是一个实参(argument)，即调用函数时传递给函数的信息

传递实参的方式

函数定义中可能包含多个形参，因此函数调用也可能包含多个实参，通常有以下方式来向函数传递实参：

- **位置实参**：将传递的实参按照位置顺序与形参绑定

```
>>> def describe_pet(animal_type, pet_name):
...     print(f"My {animal_type}'s name is {pet_name}")
...
>>> describe_pet("Cat", "DwenDwen")
My Cat's name is DwenDwen
>>> describe_pet("DwenDwen", "Cat")
My DwenDwen's name is Cat
```

传递实参的方式

- **关键字实参**：传递给函数名称值对。因为直接在实参中将名称和值关联起来，所以向函数传递实参时不会混淆

```
>>> describe_pet(pet_name="DwenDwen", animal_type="Cat")
My Cat's name is DwenDwen
```

- 注意：使用关键字实参时，务必准确指定函数定义中的形参名。关键字实参的顺序无关紧要
- 如果需要同时传递位置参数与关键字参数，位置参数的顺序应优先于关键字参数，否则会报错

```
>>> describe_pet(pet_name="DwenDwen", "Cat")
  File "<stdin>", line 1
SyntaxError: positional argument follows keyword argument
```

传递实参的方式

- **默认参数**：编写函数时，可给形参指定默认值。此时，当调用函数时给形参提供实参，Python将使用指定的实参值，否则将使用形参的默认值

```
>>> def describe_pet(pet_name, animal_type="Cat"):
...     print(f"My {animal_type}'s name is {pet_name}")
...
>>> describe_pet("DwenDwen")
My Cat's name is DwenDwen
>>> describe_pet("DwenDwen", "Dog")
My Dog's name is DwenDwen
```

- 注意：函数定义时，默认参数放在参数列表最后。

传递任意数量的实参

- 有时候预先不知道函数需要接受多少个实参，Python允许函数从调用语句中收集任意数量的实参

```
>>> def make_lunch(*materials):
...     print(materials)
...
>>> make_lunch("Rice")
('Rice',)
>>> make_lunch("Rice", "Beef", "Cabbage")
('Rice', 'Beef', 'Cabbage')
```

- 形参*materials中的星号让Python创建一个名为materials的空元组，并将收集到的所有值都封装到这个元组中
- 函数外如果想调用一个可变参数，也可以用*变量名的形式进行调用，其中变量为list或tuple：

```
>>> lst = ["Rice", "Beef", "Cabbage"]
>>> make_lunch(*lst)
('Rice', 'Beef', 'Cabbage')
>>> make_lunch(lst)
(['Rice', 'Beef', 'Cabbage'],)
```

传递任意数量的实参

- 如果要让函数接受不同类型的实参，必须在函数定义中将接纳任意数量实参的形参放在最后
- Python先匹配位置实参和关键字实参，再将余下的实参都收集到最后一个形参中

```
>>> def make_lunch(cost, *materials):
...     print("Total cost: ", cost)
...     print(materials)
...
>>> make_lunch(30, "Rice", "Beef", "Cabbage")
Total cost: 30
('Rice', 'Beef', 'Cabbage')
```

- 实际应用中，会看到通用形参名*args，它也收集任意数量的位置实参

传递任意数量的关键字实参

- 有时候需要接受任意数量的实参，但预先不知道传递给函数的会是什么样的信息，可将函数编写成能够接受任意数量的键值对，

```
>>> def build_profile(name, age, **user_info):
...     user_info['name'] = name
...     user_info['age'] = age
...     return user_info
...
>>> user_profile = build_profile('Wang', 20, location='UIBE',
     field='finance')
>>> print(user_profile)
{'location': 'UIBE', 'field': 'finance', 'name': 'Wang', 'age': 20}
```

- 形参**user_info中的两个星号让Python创建一个名为user_info的空字典，并将收到的所有名称值对都放到这个字典中
- 实际应用中，经常能看到形参名**kwargs，用于收集任意数量的关键字实参。类似地，函数外可以用**变量名来调用可变关键字参数，其中变量为字典
- 编写函数时，能以各种方式混合使用位置实参、关键字实参和任意数量的实参

传递列表

- 列表传递给函数时，函数可对其进行修改。在函数中对这个列表所做的任何修改都是永久性的

```
>>> course_list = ["Python", "Econ", "Quant"]
>>> def print_script(course_list):
...     while course_list:
...         print(course_list.pop())
...
>>> print_script(course_list)
Quant
Econ
Python
>>> course_list
[]
```

传递列表

- 禁止函数修改列表：可向函数传递列表的副本而非原件，这样函数所做的任何修改都只影响副本

```
>>> course_list = ["Python", "Econ", "Quant"]
>>> print_script(course_list[:])
Quant
Econ
Python
>>> course_list
['Python', 'Econ', 'Quant']
```

变量作用域

- Python中变量的访问权限取决于其赋值的位置，这个位置被称为变量的作用域（scope）。作用域也叫命名空间，如：全局命名空间，局部命名空间。
- Python中的作用域分4种情况：
  - 局部作用域（Local）：当前函数内的作用域
  - 嵌套作用域（Enclosing）：外部嵌套函数的作用域
  - 全局作用域（Global）：函数外部所在的命名空间
  - 内建作用域（Built-in）：Python内置模块的命名空间
- 变量在作用域中查找的顺序是$LEGB$，即当在局部找不到时会去局部外的局部找（例如闭包），再找不到会在全局范围内找，最后去内置函数所在模块的范围中找。

变量作用域

- 全局变量：如果一个变量名在代码文件内部(一般指Module模块内部)、函数外部被赋值和创建，则该变量的作用域是全局的(Global)，它在该文件内部的任何地方都可见。

```
>>> x = 10
>>> def fun_global(value):
...     return x + value
...
>>> fun_global(5)
15
```

- 本地变量：若变量名在函数内部被赋值与创建，则该变量的作用域是本地的(Local),只能在该函数内部被访问，在该函数外部不可见。

```
>>> def fun_local(x, y):
...     y = 10
...
>>> x + y
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'y' is not defined
```

变量作用域

- 如果本地变量与全局变量名称相同，那么存取到的会是本地变量(即，本地变量会覆盖全局变量)。

```
>>> x = 10
>>> def fun_test1(value):
...     x = 60
...     return x + value
...
>>> fun_test1(5)
65
```

- 虽然在函数内部可以访问到全局变量，但不能对其进行修改；若要在函数内部修改全局变量，可以在变量前面加上global声明语句。

```
>>> a = 10
>>> def fun_test2():
...     global a
...     a = 20
...
>>> fun_test2()
>>> print(a)
20
```

lambda函数

- 匿名函数lambda：是指一类无需定义标识符（函数名）的函数或子程序，常用来简化函数定义方式
- 语法形式：

```
lambda [arg1 [, arg2, ..., argN]]: expression
```

  - lambda函数可以接收任意多个参数（包括可选参数），并且返回单个表达式的值
  - lambda函数冒号后面是表达式，只能有一个

```
>>> S = lambda x, y: x + y
>>> S(2, 3)
5
```

课堂练习- 编写函数

- 定义一个函数calSum(n)，它接收参数n，返回1到n里面所有整数的和。
- 定义一个函数findMax，它接收一个列表，其功能为默认返回列表中的最小的一个元素，也可以返回列表中最小的n个元素。
- 定义一个lambda函数，计算三个数的和。

二、模块

模块(module)

- 函数的优点是可将代码块与主程序分离，更进一步地，可以将函数存储在称为模块的独立文件中，再将模块导入到主程序中
- 将函数存储在独立的文件中，可隐藏程序代码的细节，将重点放在程序的高层逻辑上
- 可与其他程序员共享这些文件而不是整个程序
- 导入整个模块，模块是扩展名为.py的文件，包含要导入到程序中的代码
- import语句导入名为module_name.py的整个模块，并采用下列方式来调用该模块中的函数function_name

```
import module_name
module_name.function_name()
```

导入模块中的函数

- 导入模块中的特定函数

```
from module_name import function_0, function_1, function_2
```

- 使用as给函数指定别名

```
from module_name import function_name as fn
```

- 使用as给模块指定别名

```
import module_name as mn
mn.function_name()
```

- 使用星号(*)运算符可让Python导入模块中的所有函数，从而可以直接通过名称来调用每个函数。（注意：可能存在自己编写的函数与模块中的函数名称相同，会发生覆盖现象）

```
from module_name import *
```

三、Numpy

Numpy简介

- Numpy(Numerical Python的简称)是Python的一个很重要的第三方库，很多其他科学计算的第三方库都是以Numpy为基础建立的。
- Numpy的部分功能：
  - ndarray (n-dimensional array object)，一个具有矢量算术运算和复杂广播能力的快速且节省空间的多维数组
  - 用于对整组数据进行快速运算的标准数学函数（无需编写循环）
  - 用于集成由C、C++、Fortran等语言编写的代码等工具
  - 线性代数、随机数生成以及傅立叶变换功能

导入Numpy

- 在使用Numpy之前，我们需要导入numpy包。导入的方法有以下几种：

```
import numpy
import numpy as np
from numpy import *
from numpy import array, sin
```

- 这里常用的是import numpy as np

创建Numpy数组

- ndarray (N-dimensional array object)，N维数组对象是NumPy中基本数据结构
  - 一种由相同类型的元素组成的多维数组，由两部分构成:
    - 实际的数据
    - 描述这些数据的元数据(数据维度、数据类型等)
  - 元素的数据类型由dtype(data-type)对象来指定，每个ndarray只有一种dtype类型
  - 一般情况下是大小固定的，创建好数组时一旦指定好大小，就不会再发生改变。
- 使用array函数创建一维、二维数组

```
>>> arr1 = np.array([6, 7.5, 8, 0, 1])
>>> arr1
array([6. , 7.5, 8. , 0. , 1. ])
>>> arr2 = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
>>> arr2
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

Numpy数组

- 为什么使用Numpy数组？思考：假如数据X、Y存储在两个列表里面，如何来进行加法运算？可能存在哪些问题？

```
X = np.array([1, 2, 3, 4])
Y = np.array([5, 6, 7, 8])
Z = X * 2 + Y * 2
```

- 数组对象可以去掉元素间运算所需的循环，使一维向量更像单个数据
- 可以提升这类应用的运算速度
- 数组对象采用相同的数据类型，有助于提升运算速度和节省存储空间

查看Numpy数组的属性

- 查看类型：type(arr1)
- 查看数组中的数据类型：arr1.dtype
- 查看形状，会返回一个元组，每个元素代表这一维的元素数目：arr1.shape
- 查看元素数目：arr1.size
- 查看数组维数：arr1.ndim

查看Numpy数组的属性

```
>>> arr1 = np.array([1, 2, 3, 4])
>>> arr2 = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
>>> arr3 = np.array([1., 2, 3, 4])
>>> arr4 = np.array([[1, 2, 3, 4]])
>>> type(arr1)
<class 'numpy.ndarray'>
>>> print(arr1.dtype, arr3.dtype)
int64 float64
>>> print(arr1.shape, arr2.shape, arr4.shape)
(4,) (2, 4) (1, 4)
>>> print(arr1.size, arr2.size)
4 8
>>> print(arr1.ndim, arr2.ndim, arr4.ndim)
1 2 2
```

快速创建数组的一些函数

- np.arange(start, end, step, dtype)

```
>>> np.arange(2, 10, 3)
array([2, 5, 8])
>>> np.arange(2, 10, 3, dtype='float64')
array([2., 5., 8.])
```

- np.linspace(start, stop, num=50)：创建一个等差数列

```
>>> np.linspace(2, 10, 4)
array([ 2.        ,  4.66666667, 7.33333333, 10.        ])
```

- np.logspace(start, stop, num=50, base=10.0)：创建一个等比数列，其中起始值为base ** start，终止值为base ** end

```
>>> np.logspace(2, 5, 4, base=4)
array([ 16.,  64., 256., 1024.])
```

快速创建数组的一些函数

- 创建指定形状全0的数组：np.zeros(shape, dtype=float)

```
>>> np.zeros(3)
array([0., 0., 0.])
>>> np.zeros((1,3))
array([[0., 0., 0.]])
>>> np.zeros((2, 4))
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

- 创建指定形状全1的数组：np.ones(shape, dtype=float)

```
>>> np.ones((2,2))
array([[1., 1.],
       [1., 1.]])
```

- 创建指定形状的单位数组：np.eye(N,M=None, dtype=float)

```
>>> np.eye(2)
array([[1., 0.],
       [0., 1.]])
>>> np.eye(2, 3)
array([[1., 0., 0.],
       [0., 1., 0.]])
```

数组的索引和切片

- 一维数组的索引与切片规则与列表相似

```
>>> arr = np.arange(10)
>>> arr
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> arr[5]
5
>>> arr[5:8]
array([5, 6, 7])
```

- 二维数组的索引

```
>>> arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> arr2d
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> arr2d[2]
array([7, 8, 9])
>>> arr2d[0][2]
3
>>> arr2d[0, 2]
3
```

数组的索引和切片

- 二维数组的切片

```
>>> arr2d
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> arr2d[:2]
array([[1, 2, 3],
       [4, 5, 6]])
>>> arr2d[:2, 1:]
array([[2, 3],
       [5, 6]])
>>> arr2d[1, :2]
array([4, 5])
>>> arr2d[:, :1]
array([[1],
       [4],
       [7]])
```

数组的数学和统计方法

- sum、mean、median以及标准差std等聚合计算，可以当作数组的实例方法调用，也可以当作Numpy函数使用：

```
>>> arr = np.arange(20).reshape((4, 5))
>>> arr
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
>>> arr.mean()
9.5
>>> np.mean(arr)
9.5
>>> arr.sum()
190
>>> arr.mean(axis=1)
array([ 2.,  7., 12., 17.])
>>> arr.sum(0)
array([30, 34, 38, 42, 46])
```

课堂练习- 使用Numpy

- 创建一个长度为10的一维数组，初始元素值为0，将最后一个值修改为1
- 创建一个元素为30到90，间隔为3的一维数组，将所有元素反转
- 创建一个1到2之间的长度为10的等差数列数组
- 创建一个1到512之间的长度为10的等比数列数组，基为2
- 创建一个5*4的二维数组，值为1到20，并分析它的行、列的最小值、最大值、均值、和

四、Pandas

Pandas简介

- Pandas是面板数据(Panel Data)的简写。它是Python最强大的数据分析和探索工具，因金融数据分析工具而开发，支持时间序列分析，灵活处理缺失数据。
- Pandas和NumPy的区别：
  - NumPy是数值计算的扩展包：N维数组容器，可用来存储和处理大型矩阵Matrix，比Python自身的嵌套列表list结构要高效。Numpy扩展包提供了数组支持，同时很多高级扩展包依赖它。例如Scipy、Matplotlib、Pandas。
  - Pandas是做数据处理：表格容器，是基于NumPy 的一种工具，该工具是为了解决数据分析任务而创建的。Pandas 提供了大量快速便捷地处理数据的函数和方法。
- 导入方式：

```
import pandas as pd
```

- Pandas的两种数据结构：Series和DataFrame

一维数据结构：Series

- Series是一维带标记的数组结构，可以存储任意类型的数据（整数，浮点数，字符串，Python 对象等等）。
- 作为一维结构，它的索引叫做index，生成Series的方法：

```
s = pd.Series(data, index=index)
```

- data可以是以下结构：字典、ndarray、标量
- index 是一维坐标轴的索引列表。

一维数据结构：Series

- 如果data是个ndarray，那么index的长度必须跟data一致：

```
>>> s = pd.Series(np.random.randn(3), index=["a", "b", "c"])
>>> s
a    0.897045
b   -2.146609
c   -0.248715
dtype: float64
>>> s.index
Index(['a', 'b', 'c'], dtype='object')
>>> s.values
array([ 0.8970447 , -2.14660878, -0.24871533])
>>> s.name = "RandomValue"
>>> s
a     0.897045
b    -2.146609
c    -0.248715
e    12.000000
Name: RandomValue, dtype: float64
s.index.name = "index_name"
s.index = list('cdef')
```

使用Series

- 像ndarray一样使用Series：

```
>>> s = pd.Series(np.random.randn(3), index=["a", "b", "c"])
>>> s
a   -0.755471
b    0.431814
c   -0.639069
dtype: float64
>>> s[0]
-0.7554711618837914
>>> s[:1]
a   -0.755471
dtype: float64
>>> s[s>0]
b    0.431814
dtype: float64
>>> np.exp(s)
a    0.469789
b    1.540048
c    0.527783
dtype: float64
```

使用Series

- 像字典一样使用Series：

```
>>> s["a"]
0.8970446971697016
>>> s["e"] = 12
>>> s
a     0.897045
b    -2.146609
c    -0.248715
e    12.000000
dtype: float64
>>> "e" in s
True
>>> "f" in s
False
>>> s.get("f", np.nan)
nan
>>> s.get("e")
12.0
```

使用Series

- 向量化操作

```
>>> s * 2
a     1.794089
b    -4.293218
c    -0.497431
e    24.000000
dtype: float64
>>> s + 2
a     2.897045
b    -0.146609
c     1.751285
e    14.000000
dtype: float64
```

使用Series

- Series和ndarray不同的地方在于，Series的操作默认是使用index的值进行对齐的，而不是相对位置：

```
>>> s[1:] + s[:-1]
a         NaN
b   -4.293218
c   -0.497431
e         NaN
dtype: float64
```

对于上面两个不能完全对齐的Series，结果的index是两者index的并集，同时不能对齐的部分当作缺失值处理。

DataFrame

- DataFrame是pandas中的二维数据结构，可以看成一个Excel中的工作表，或者一个存储Series对象的字典。

```
pd.DataFrame(data, index, columns)
>>> d = {'one' : pd.Series([1., 2., 3.], index=['a', 'b', 'c']),
...      'two' : pd.Series([1., 2., 3., 4.], index=list('abcd'))}
>>> pd.DataFrame(d)
   one  two
a  1.0  1.0
b  2.0  2.0
c  3.0  3.0
d  NaN  4.0
>>> pd.DataFrame(d, index=['d', 'b', 'a'])
   one  two
d  NaN  4.0
b  2.0  2.0
a  1.0  1.0
>>> pd.DataFrame(d, index=['d', 'b', 'a'], columns=['two', 'three'])
   two three
d  4.0   NaN
b  2.0   NaN
a  1.0   NaN
```

DataFrame

- 查看DataFrame的属性

```
>>> df = pd.DataFrame(d)
>>> df.index
Index(['a', 'b', 'c', 'd'], dtype='object')
>>> df.columns
Index(['one', 'two'], dtype='object')
>>> df.values
array([[ 1., 1.],
       [ 2., 2.],
       [ 3., 3.],
       [nan, 4.]])
>>> df.index.name
>>> df.index.name = "label"
>>> df
       one  two
label
a      1.0  1.0
b      2.0  2.0
c      3.0  3.0
d      NaN  4.0
```

DataFrame

- DataFrame的列操作

```
>>> df["one"]
label
a    1.0
b    2.0
c    3.0
d    NaN
Name: one, dtype: float64
>>> df['three'] = df['one'] * df['two']
>>> df['flag'] = df['one'] > 2
>>> df
       one  two  three   flag
label
a      1.0  1.0    1.0  False
b      2.0  2.0    4.0  False
c      3.0  3.0    9.0   True
d      NaN  4.0    NaN  False

del df["two"]
three = df.pop("three")
df["test"] = 3
```

DataFrame的简单操作

- 查看数据头尾：df.head(), df.tail()
- 转置：df.T
- 索引排序：df.sort_index(axis=0, ascending=True) 按照index大小进行排序，axis=0表示按第0维进行排序
- 值排序：df.sort_values(by, axis=0, ascending=True) 方法按照by 的值的大小进行排序

DataFrame的索引

```
>>> dates = pd.date_range('20220101', periods=4)
>>> df = pd.DataFrame(np.random.randn(4,4), index=dates,
     columns=list('ABCD'))
>>> df
                   A         B         C         D
2022-01-01 0.906539 -1.000682 -1.293240 0.898528
2022-01-02 -0.242724 -0.124767 -0.618350 -0.320222
2022-01-03 1.417078 0.536002 0.958161 -0.359186
2022-01-04 -0.702457 -0.693509 0.115511 1.633843
```

- 选择单列数据：df["A"] 或df.A
- 使用label索引(loc)：

```
df.loc["20220104"], df.loc['20130102':'20130104', ['A','B']]
df.loc['20130102', ['A','B']], df.loc[dates[0], 'B']
```

- 切片读取多行：df[0:3]，df["20220101":"20220103"]
- 使用位置索引：df.iloc[3]
- 布尔型索引：df[df.A $>$ 0]，df[df $>$ 0]

DataFrame的缺失值处理

- 检查缺失值：pd.isnull()，pd.notnull()。既可以作为函数，也可以作为方法

```
>>> df1 = pd.DataFrame([[1, 2, 3, np.nan], [2, 3, np.nan, 4], [7, 8,
    9, 10], [np.nan, np.nan, np.nan, np.nan]])
>>> df1
     0    1    2     3
0  1.0  2.0  3.0   NaN
1  2.0  3.0  NaN   4.0
2  7.0  8.0  9.0  10.0
3  NaN  NaN  NaN   NaN
>>> pd.isnull(df1)
       0      1      2      3
0  False  False  False   True
1  False  False   True  False
2  False  False  False  False
3   True   True   True   True
>>> df1.isnull()
       0      1      2      3
0  False  False  False   True
1  False  False   True  False
2  False  False  False  False
3   True   True   True   True
```

DataFrame的缺失值处理

- 删除缺失值：dropna()

```
>>> df1.dropna(how='any')
     0    1    2     3
2  7.0  8.0  9.0  10.0
>>> df1.dropna(how='all')
     0    1    2     3
0  1.0  2.0  3.0   NaN
1  2.0  3.0  NaN   4.0
2  7.0  8.0  9.0  10.0
```

- 填充缺失值：fillna()

```
>>> df1.fillna(value=5)
     0    1    2     3
0  1.0  2.0  3.0   5.0
1  2.0  3.0  5.0   4.0
2  7.0  8.0  9.0  10.0
3  5.0  5.0  5.0   5.0
df1.fillna(method="ffill")
df1.fillna(method="bfill")
```

DataFrame的基本统计运算

- 均值：df.mean()，df.mean(axis=1)
- 求和：df.sum()，df.sum(axis=1)
- 描述性统计：df.describe()
- 相关系数：df.corr()
- 协方差：df.cov()

数据读写

- dataframe.to_csv(), pandas.read_csv()

```
import pandas as pd
import numpy as np

dates = pd.date_range('20130101', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates,
     columns=list('ABCD'))
df.index.name = 'time'

df.to_csv('data1.csv', index=False)
df_new = pd.read_csv('data1.csv', index_col='time')
```

- dataframe.to_excel(), pandas.read_excel()

```
df.to_excel('data1.xlsx', sheet_name='Sheet1')
df2 = pd.read_excel('data1.xlsx', 'Sheet1')
```

数据合并

- pandas.merge() 根据一个或多个键将不同DataFrame中的行连接起来

```
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'], 'data1':
     range(6)})
df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'], 'data2':
     range(5)})
pd.merge(df1, df2, on='key', how='inner')

df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
     'data1': range(7)})
df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'], 'data2': range(3)})
pd.merge(df3, df4, left_on='lkey', right_on='rkey')

df5 = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'],'value':
     range(6)})
df6 = pd.DataFrame({'group_val': [3.5, 7]}, index=['a', 'b'])
pd.merge(df5, df6, left_on='key', right_index=True, how='outer')
```

数据合并

- pandas.concat() 可以沿着一条轴将多个对象堆叠到一起

```
df1 = pd.DataFrame(np.random.randn(3, 4), columns=['a', 'b', 'c',
    'd'])
df2 = pd.DataFrame(np.random.randn(2, 3), columns=['b', 'd', 'a'])
pd.concat([df1, df2, df2], join='outer', sort=False,
    ignore_index=True)
pd.concat([df1, df2], axis=1)
```

课堂练习- DataFrame数据处理

- 读取数据CME_Gdp1.xlsx，删除前两行。【或者读取数据CME_Gdp1.csv，将年份作为index，重命名列名】
- 针对GDP和GNP进行描述性分析，计算各年度三大产业产值占GDP的比重
- 尝试用索引的方式，查看2008年的数据
- 对第三产业占比进行排序
- 将结果写入到Excel中
- 将2019年的GDP的值替换为空值。检查缺失值，并采用上一年的值进行替换
- 使用课件上的其他命令，进行练习

五、Matplotlib

Matplotlib简介

- matplotlib是一个Python的2D图形包。在线文档：http://matplotlib.org
- matplotlib.pyplot
  - 包含一系列类似MATLAB中绘图函数的相关函数。
  - 其中的函数对当前的图像进行一些修改，例如：产生新的图像，在图像中产生新的绘图区域，在绘图区域中画线，给绘图加上标记，等等。
  - 它会自动记住当前的图像和绘图区域，因此这些函数会直接作用在当前的图像上。
- 导入相关的包：

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

Matplotlib画图函数

- 画折线图：plt.plot(x, y, format_str)，x,y为数据（列表、numpy数组、series数据），format_str控制点和线点形状及颜色
  - 颜色参数：b - 蓝色、g - 绿色、r - 红色、k - 黑色
  - 类型参数：- 实线、– 虚线、-. 虚点线；. 点、o 圆点、+加号点

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro-.')
plt.show()

plt.plot([1, 4, 9, 16])

t = np.arange(0., 5., 0.2)
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
```

Matplotlib画图函数

- 如果要在一副图中生成多个子图，可以使用plt.subplot()，其参数为：

```
plt.subplot(numrows, numcols, fignum)
```

当numrows * numcols < 10 时，中间的逗号可以省略，因此plt.subplot(211)就相当于plt.subplot(2, 1, 1)

```
def f(t):
    return np.exp(-t) * np.cos(2 * np.pi * t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

这里figure()函数会产生一个指定编号为num的图，figure(1) 其实是可以省略的，因为默认情况下plt 会自动产生一幅图像。

一个画图的例子

```
import matplotlib.pyplot as plt
import numpy as np

t = np.arange(0., 5., 0.2)
fontsize = 10
fig_dir = '/inputYourPath/'

plt.close('all')
plt.figure(figsize=(8, 6))
plt.tick_params(labelsize=fontsize)
plt.plot(t, t, 'r-', t, t ** 2, 'b--', linewidth=3)
plt.xlabel('Time', fontsize=fontsize)
plt.ylabel('Value', fontsize=fontsize)
plt.title('Here is the title name', fontsize=fontsize)
plt.legend(['Legend1', 'Legend2'], fontsize=fontsize, loc=(.02,.90),
     frameon=False)
plt.xlim(0, 5)
plt.ylim(0, 30)
# plt.show()
plt.savefig(fig_dir + 'Test1' + '.png', dpi=100, bbox_inches='tight')
```

其他图形

- 直方图：plt.hist()

```
df = pd.read_excel('CSI300Daily.xlsx')
temp = plt.hist(df['volume'], 50, density=True, facecolor='g',
     alpha=0.75)
```

- 散点图：plt.scatter()

```
plt.scatter(df['amt'], df['pct_chg'], marker="*", c="r")
```

- 柱状图：plt.bar()

```
plt.bar([0, 1, 2, 3, 4], [20, 30, 40, 50, 40], color='w',
     edgecolor='k')
```

课堂练习- 画图

- 使用数据CME_Gdp1.csv，画出GDP的时间变化趋势，添加相关的标签和图例，将结果存储为PDF
- 画出GNP和GDP的散点图