

# 第二章 Python快速介绍

## 2.1 Python介绍和基本语法

### 2.1.1 Python简介、安装、编辑器

### 2.1.2 基本语法

## 2.2 数据分析常用库的介绍

### 2.2.1 高性能计算NumPy

### 2.2.2 画图和可视化matplotlib

### 2.2.3 数据导入和预处理pandas

# Python简介

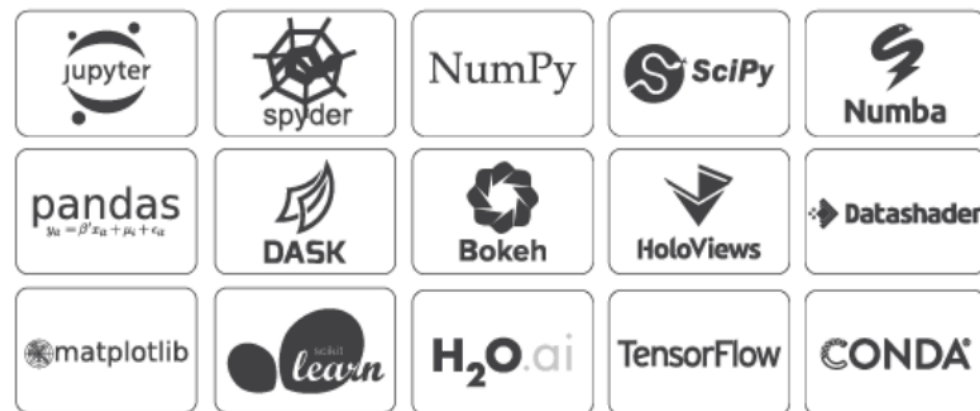
- 于90年代早期在荷兰由吉多·范罗苏姆（Guido van Rossum）发明；
- 是一种开源的脚本语言，但比普通脚本具有更多功能；
- Python不需要编译
  - C需要编译，通过编译得到二进制机器码后才能被执行
  - Python是解释器，逐行解释后立即执行
- Python语言具有简洁性、易读性以及可扩展性；
- 目前流行用Python做科学计算；
- 经典的科学计算扩展库：NumPy、SciPy和matplotlib，分别为Python提供了快速进行数组处理、数值运算以及绘图功能。

# Python 安装: (方法一) Anaconda

- 用于科学计算的开源发行版, 包括 python 环境和 IDE (spyder) ;
- 集成了上百个科学计算的第三方模块, 需要时可直接导入;
- 解决多版本并存, 如 Install with python 3.6 (and [install python 2.7 as secondary from anaconda prompt](#))

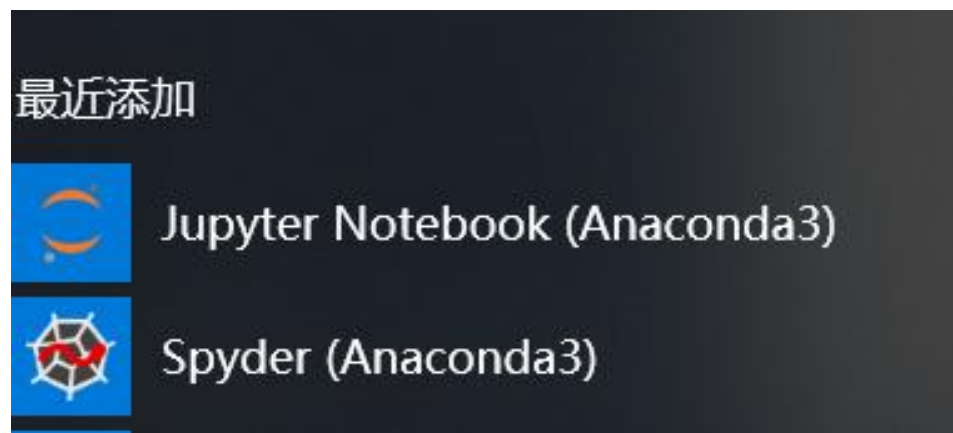
官方下载地址: <https://www.anaconda.com/download/>

建议对命令行不熟悉的同学采用该方法



# 运行和编辑

- 安装完Anaconda3后就已经装了jupyter nootbook和Spyder两个编辑器， 可以从开始菜单里面找到：



- 也可以用Python 自带的编辑器IDLE



# 开启jupyter notebook

- 启动 jupyter notebook

```
Jupyter Notebook (Anaconda3)
[I 16:37:28.798 NotebookApp] JupyterLab extension loaded from D:\Anaconda3\lib\site-packages\jupyterlab
[I 16:37:28.798 NotebookApp] JupyterLab application directory is D:\Anaconda3\share\jupyter\lab
[I 16:37:28.801 NotebookApp] Serving notebooks from local directory: C:\Users\梅建萍
[I 16:37:28.802 NotebookApp] The Jupyter Notebook is running at:
[I 16:37:28.802 NotebookApp] http://localhost:8888/?token=516ea08db27994ebb1aa35ebfa25b6bbd605316d4737cbcf
[I 16:37:28.802 NotebookApp] or http://127.0.0.1:8888/?token=516ea08db27994ebb1aa35ebfa25b6bbd605316d4737cbcf
[I 16:37:28.803 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 16:37:28.846 NotebookApp]

To access the notebook, open this file in a browser:
file:///C:/Users/%E6%A2%85%E5%BB%BA%E8%90%8D/AppData/Roaming/jupyter/runtime/nbserver-38172-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=516ea08db27994ebb1aa35ebfa25b6bbd605316d4737cbcf
or http://127.0.0.1:8888/?token=516ea08db27994ebb1aa35ebfa25b6bbd605316d4737cbcf
```

有时需要手动复制URL到浏览器:

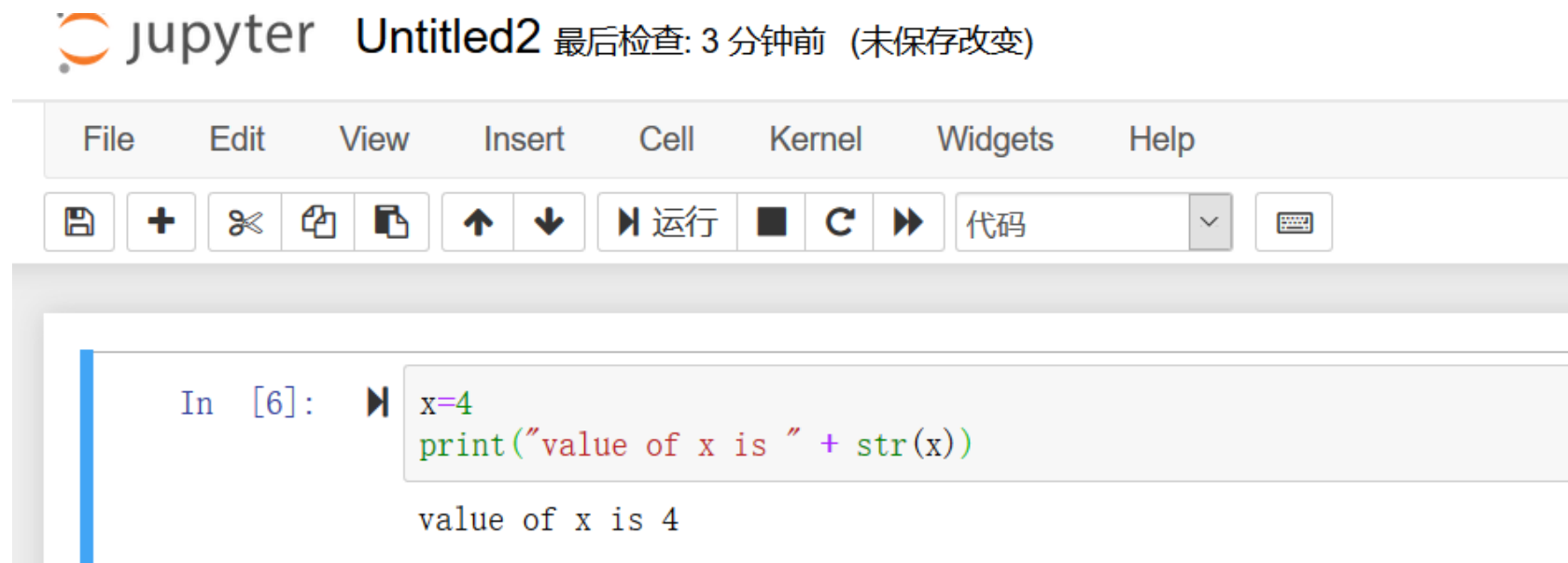
 Quit Logout

Files Running Clusters

Select items to perform actions on them. Upload New ▾ ↻

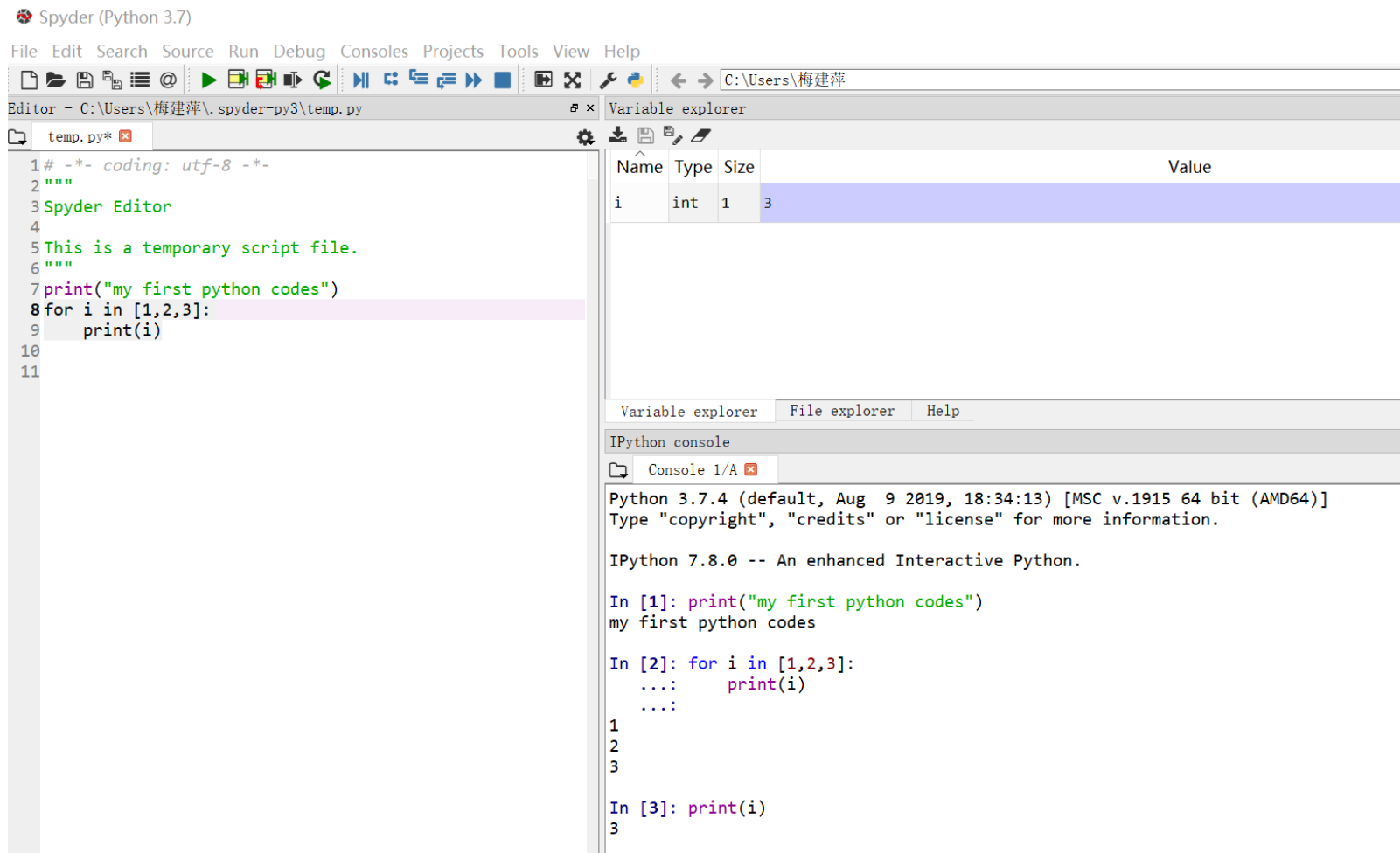
<input type="checkbox"/> 0 ▾	📁 /	Name ▾	Last Modified	File size
<input type="checkbox"/>	📄	Untitled.ipynb	4 个月前	8.13 kB
<input type="checkbox"/>	📄	Untitled1.ipynb	1 个月前	22.9 kB
<input type="checkbox"/>	📄	easy_install-3.7.exe	4 个月前	103 kB
<input type="checkbox"/>	📄	iptest.exe	4 个月前	103 kB

进入主页后点击右上角new 选择python3创建.ipynb文件



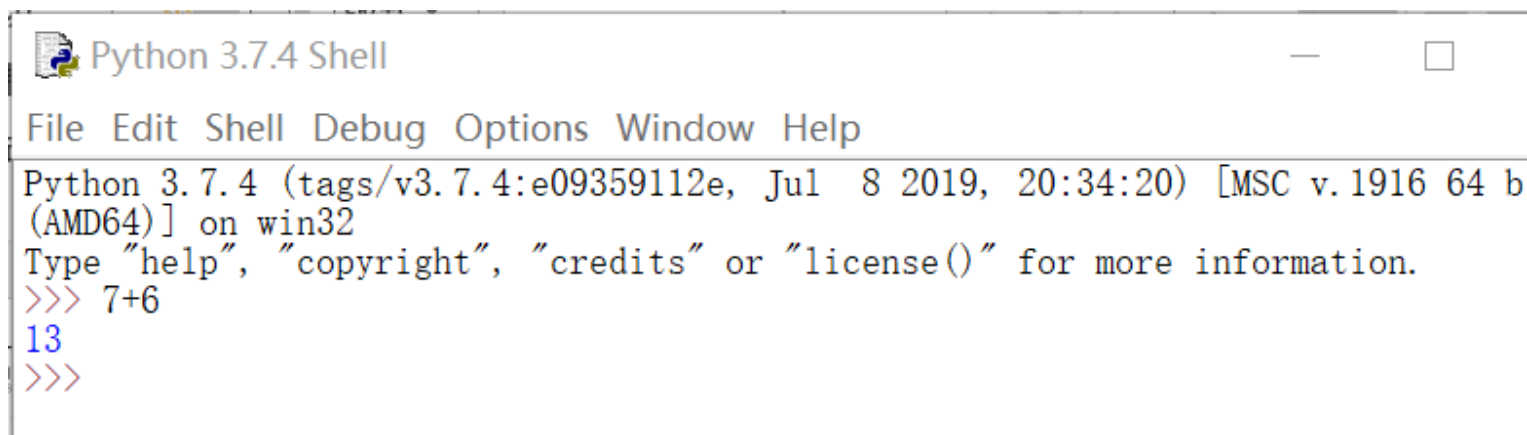
# spyder

- 与Matlab类似的python编辑器



如果你习惯  
Eclipse, 也可以用  
PyCharm。

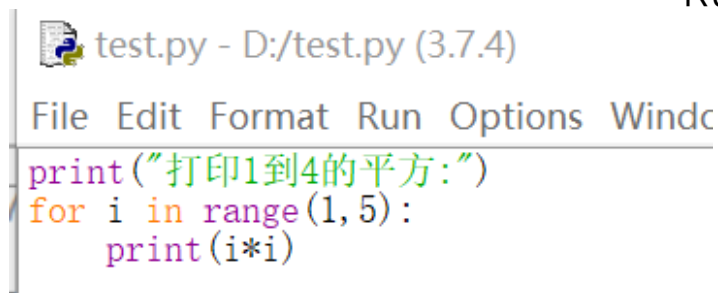
# IDLE (python自带编辑器)



```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 20:34:20) [MSC v.1916 64 b
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 7+6
13
>>>
```

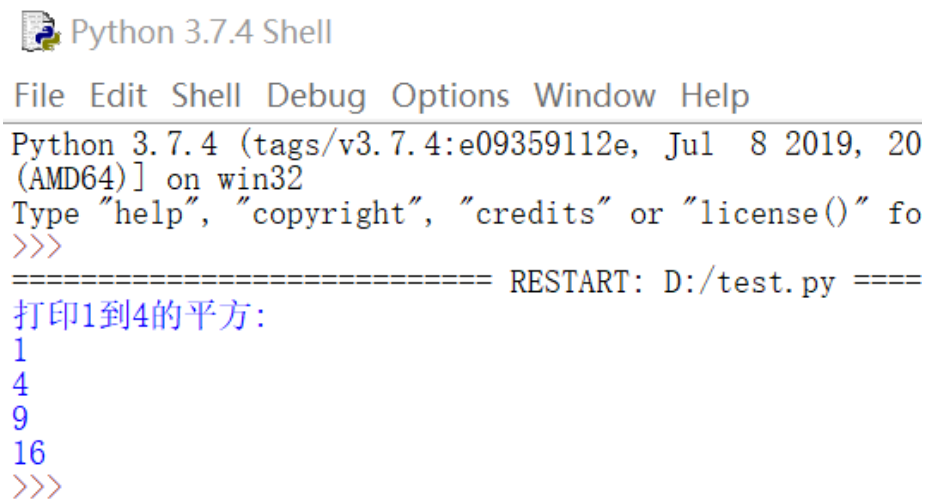
File->New File

编写程序



```
test.py - D:/test.py (3.7.4)
File Edit Format Run Options Windc
print("打印1到4的平方:")
for i in range(1,5):
    print(i*i)
```

Run->Run Module(F5)



```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 20
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" fo
>>>
===== RESTART: D:/test.py =====
打印1到4的平方:
1
4
9
16
>>>
```



## Python 安装: (方法二) 通过pip自定义安装

1. 安装 python 3.7.4 官方[下载](#)

2. 安装完python后用pip安装其他packages

在python 安装目录下, 命令行输入:

```
python -m pip install --upgrade pip
```

```
python -m pip install --user ipython jupyter numpy scipy matplotlib pandas
```

ipython、jupyter为基于网页的交互式应用

**NumPy, SciPy, matplotlib, pandas**最常用的 Python科学计算和数据分析工具包/库, 用户说明和例子

<https://www.scipy.org/docs.html>

# 第二章 Python简单介绍

## 2.1 python介绍和基本语法

2.1.1 python简介、安装、运行、编辑器

2.1.2 基本语法、标准数据类型

## 2.2 数据分析常用库的介绍

# Python 基本语法

1. 与许多其他编程语言用花括号来表示代码段不同，Python 用缩进来表示。不对齐的缩进会导致错误。
2. 用 #进行注释；
3. 换行表示一个语句

```
for i in [1, 2, 3, 4, 5]:  
    # first line in "for i" block  
    print (i)  
    for j in [1, 2, 3, 4, 5]:  
        # first line in "for j" block  
        print (j)  
        # last line in "for j" block  
        print (i + j)  
    # last line in "for i" block print "done looping"  
    print (i)  
print ("done looping")    #在 python 3, print()是函数, 必须有括号
```

## 4. ( ) 和 [ ] 内的空格不起作用

```
long_winded_computation = (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 +  
                           9 + 10 + 11 + 12 + 13 + 14 +  
                           15 + 16 + 17 + 18 + 19 + 20)
```

```
list_of_lists = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
easier_to_read_list_of_lists =  
[ [1, 2, 3],  
  [4, 5, 6],  
  [7, 8, 9] ]
```

或者

```
long_winded_computation = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + \  
                           9 + 10 + 11 + 12 + 13 + 14 + \  
                           15 + 16 + 17 + 18 + 19 + 20
```

# 模块加载

- 有些功能所在的模块默认情况下没有被装载；
- 如果要适用这些功能，你需import它们所在的模块：

例如

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

载入用于画图的matplotlib下面的pyplot  
载入用于科学计算的 numpy模块

想知道某个模块、函数的具体用法，输入模块/函数名后加？就能得到更多信息和例子：

如np? 或 np.arrange?

# 变量和对象

- 变量在第一次赋值的时候被创建,不需要申明

- `x = 5`
- `x = [1, 3, 5]`
- `x = 'python'`

In [18]: ▶

```
import numpy as np
```

```
x=5  
print(x)  
x=[1, 2, 3]  
print(x)  
x="pyhton"  
print(x)
```

```
5  
[1, 2, 3]  
pyhton
```

- 对于列表，赋值建立的是引用,不是内容复制;

In [20]: ▶

```
import numpy as np  
x=[1, 2, 3]  
y=x;  
x[1]=5  
print(y)
```

```
[1, 5, 3]
```

下标从0开始

# 赋值

- 可以同时给多个变量赋值

$x, y = 2, 3$

- 交换值

$x, y = y, x$

- 可以连等

$x = y = z = 3$

# 算术运算

- $a = 5 + 2$  # a 为 7
- $b = 9 - 3.$  # b 为6.0
- $c = 5 * 2$  # c 为 10
- $d = 5 ** 2$  # d 为25
- $e = 5 \% 2$  # e 为 1
- $f = 7 / 2$  # f为 3.5
- $f = 7 // 2$  # f 为 3
- $f = \text{int}(7 / 2)$  # f为 3

内置数值类型: int, float, complex

注意: python 3 "/"表示除, 而python2里面"/"表示整除;



# 字符串 - 1

- 字符串可以用单引号或双引号来框定

```
single_quoted_string = 'data science'
double_quoted_string = "data science"
escaped_string = 'Isn\'t this fun'
another_string = "Isn't this fun"
```

```
real_long_string = 'this is a really long string. \
It has multiple parts, \
but all in one line.'
```

#长句子用\

- 用三个双引号表示跨行字符串

```
multi_line_string = """This is the first line.
and this is the second line
and this is the third line"""
```

# 字符串 - 2

- 字符串可以通过 + 操作连起来, 用 \* 进行重复

```
s = 3 * 'un' + 'ium' # s is 'ununium'
```

- 在引号内的相邻的字符串会自动连接

```
s1 = 'Py' 'thon'
```

```
s2 = s1 + '2.7'
```

```
real_long_string = ('this is a really long string. '  
'It has multiple parts, '  
'but all in one line.')
```

# “真”和“假”

## 所有关键字区分大小写

- True
- False
- None
- and
- or
- not
- any 有不为0则返回True
- all 所有不为0则返回True

0, 0.0, [], (), "", 都为 False.

```
a = [0, 0, 0, 1]
```

```
any(a)
```

```
Out[135]: True
```

```
all(a)
```

```
Out[136]: False
```

# 关系运算

<

<=

>

>=

==

!=

is

is not

```
a = [0, 1, 2, 3, 4]
b = a      #列表赋值时引用
c = a[:]   #列表值的复制
```

```
a == b
Out[129]: True
```

```
a is b
Out[130]: True
```

```
a == c
Out[132]: True
```

```
a is c
Out[133]: False
```

is 比较两个对象是否指向同一个内存地址， 而 == 比较两个对象的值是否相等。

# if语句

```
p = "偶数" if x % 2 == 0 else "奇数"
```

```
if 1 > 2:
```

```
    message = "如果1 > 2"
```

```
elif 1 > 3:
```

```
    message = "elif 代表'else if'"
```

```
else:
```

```
    message = "其他情况"
```

```
print (message)
```

# 循环

```
x = 0
while x < 10:
    print (x, "is less than 10")
    x += 1
```



如果忘记缩进会怎样?

```
for x in range(10):
    if x == 3:
        continue # 进入下一个迭代
    if x == 5:
        break # 退出循环
    print (x)
```

# 函数functions - 1

- 用 *def* 定义函数

```
def double(x):  
    return x * 2
```

- 定义好以后就可以调用

```
z = double(10) # z is 20
```

- 定义的时候可以设置参数的默认值

```
def my_print(message="my default message"):  
    print (message)
```

```
my_print("hello") # 打印 'hello'
```

```
my_print() # 没有输入时打印 'my default message'
```

# 函数functions - 2

- 有时候调用时直接用输入变量名更方便：只需要对某些输入变量赋值，其他用默认值

```
def subtract(a=0, b=0):  
    return a - b
```

```
subtract(10, 5) # a=10, b=5
```

```
subtract(0, 5) # a=0, b=5
```

只需要对某些输入变量赋值，其他用默认值

```
subtract(b = 5) # b=5, a用默认值0
```

不需要按定义时输入变量顺序

```
subtract(b = 5, a = 0) # b=5, a=0
```



# Python标准数据类型和函数

- 列表(list)
- 元组(tuple)
- 字典(dictionary)
- range()函数

# 列表 - 1

- 创建列表用[]

与数组有点像，但元素类型可以不同

```
integer_list = [1, 2, 3]
heterogeneous_list = ["string", 0.1, True]
list_of_lists = [integer_list, heterogeneous_list, []]
list_length = len(integer_list) # 等于 3
list_sum = sum(integer_list) # 等于 6
```

- 引用列表元素

```
x = [i for i in range(10)] # 结果是列表 [0, 1, ..., 9]
zero = x[0] # 等于 0, 列表的下标从0开始
one = x[1] # 等于 1
nine = x[-1] # 等于 9, 最后一个元素
eight = x[-2] # 等于 8, 倒数第二个元素
```

# 列表 - 2

- 获得列表中的一个切片（起始元素包含，终止元素不包含）

```
one_to_four = x[1:5] # [1, 2, 3, 4] 从下标1到下标4的元素  
first_three = x[:3] # [0, 1, 2] 前面3个  
last_three = x[-3:] # [7, 8, 9] 后面3个  
four_to_end = x[3:] # [3, 4, ..., 9] 下标3到最后  
without_first_and_last = x[1:-1] # [1, 2, ..., 8]
```

- 检查一个元素是否在列表

```
1 in [1, 2, 3] # True  
0 in [1, 2, 3] # False
```

# 列表 - 3

- 列表连接/合并

```
x = [1, 2, 3]
```

```
y = [4, 5, 6]
```

```
x.extend(y) # x 为 [1,2,3,4,5,6]
```

```
x = [1, 2, 3]
```

```
y = [4, 5, 6]
```

```
z = x + y # z 为 [1,2,3,4,5,6]; x 不变.
```

如果想把x和y对应元素的值相加，需要用循环。

- 列表拆分 (多个变量赋值)

```
x, y = [1, 2] # x等于1, y 等于 2
```

```
[x, y] = 1, 2 # 同上
```

```
x, y = [1, 2] # 同上
```

```
_, y = [1, 2] # y等于2, 不需要第一个元素
```

# 列表 - 4

- 改变列表内容

```
x = [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

```
x[2] = x[2] * 2    # x 为 [0, 1, 4, 3, 4, 5, 6, 7, 8]
```

```
x[-1] = 0         # x 为 [0, 1, 4, 3, 4, 5, 6, 7, 0]
```

```
x[3:5] = x[3:5] * 3    # x 为 [0, 1, 4, 3, 4, 3, 4, 3, 4, 5, 6, 7, 0]
```

上面把切片重复3次插入

```
x[5:6] = []        # x 为 [0, 1, 4, 3, 4, 4, 3, 4, 5, 6, 7, 0]
```

```
del x[:2]          # x 为 [4, 3, 4, 4, 3, 4, 5, 6, 7, 0]
```

```
del x[:]           # x 为 [], 内容删除, 变量依然存在
```

```
del x              # 变量删除, 此后引用x会报错
```

- 字符串也能切片 (提取子串) . 但是不能被改变 (是常量)

```
s = 'abcdefg'
```

```
a = s[0]          # 'a'
```

```
x = s[:2]          # 'ab'
```

```
y = s[-3:]         # 'efg'
```

```
s[:2] = 'AB'       # 报错
```

```
s = 'AB' + s[2:]    # s变成 ABcdefg
```

# range() 函数

```
for i in range(5):
    print (i) # 打印出 0, 1, 2, 3, 4 (竖着打印)
for i in range(2, 5):
    print (i) # 打印出 2, 3, 4
for i in range(0, 10, 2):
    print (i) # 打印出 0, 2, 4, 6, 8
for i in range(10, 2, -2):
    print (i) # 打印10, 8, 6, 4

>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print(i, a[i])
...
0 Mary
1 had
2 a
3 little
4 lamb
```

# range()

在python **3**, range() 是一个可以被遍历的对象, 可以通过下标引用, 但不能被赋值

```
print (range(3)) # 显示结果: range(0, 3)
print (list(range(3))) # 显示结果: [0, 1, 2]
```

```
x = range(5)
print (x[2])          # 打印出 "2"
x[2] = 5              # python 3, 报错, 不支持赋值.
```

```
a = list(x) #转成列表
a[2] = 4
print(a) # [0, 1, 4, 3, 4]
```

# 元组(tuples)-1

- 与列表相似, 但是常量, 不能被赋值
- `a_tuple = (0, 1, 2, 3, 4)`
- `Other_tuple = 3, 4`
- `Another_tuple = tuple([0, 1, 2, 3, 4])` 列表转tuple
- `Hetergeneous_tuple = ('john', 1.1, [1, 2])`

- 可以被切片, 连接, 或重复

```
In [39]: a_tuple[2:4]  
Out[39]: (2, 3)
```

- 不能赋值

`a_tuple[2] = 5` 报错:不支持赋值



# 元组(tuples)- 2

- 返回多个函数值

```
def sum_and_product(x, y):  
    return (x + y), (x * y)  
sp = sum_and_product(2, 3)      # 等于 (5, 6)  
s, p = sum_and_product(5, 10)  # s 等于 15, p 等于 50
```

# 字典(dictionaries)-1

- 字典把值和关键字相关联

```
empty_dict = {}          # 空字典
grades = { "Joel" : 80, "Tim" : 95 }
```

- 通过关键字引用/改变对应的值

```
joels_grade = grades["Joel"]          # 等价于 80
grades["Tim"] = 99                     # 改变原来的值
grades["David"] = 100                  # 加入一个新的成员
num_students = len(grades)            # 等于 3

try:
    kates_grade = grades["Kate"]
except KeyError:
    print ("no grade for Kate!")
```

# 字典(dictionaries)- 2

- 检查是否存在关键字

```
joel_has_grade = "Joel" in grades      # True
kate_has_grade = "Kate" in grades      # False
```

- 用 “get” 避免 keyError，没有则返回默认值

```
joels_grade = grades.get("Joel", 0) # 等于 80
kates_grade = grades.get("Kate", 0) # 等于 0 (可以设定成其他值)
```

- 得到所有的项

```
all_keys = grades.keys() #dict_keys(['Joel', 'Tim', 'David'])
all_values = grades.values() # dict_values([80, 99, 100])
all_pairs = grades.items() # dict_items([('Joel', 90), ('Tim', 99), ('David', 100)])
```

返回可遍历的对象：

不能用下标引用，如 all\_keys[2] #错误

用循环 `for key in all_keys: print (key)` #ok